

Knowledge Discovery

Ľubomír Benko
Ján Skalka
Jaroslav Reichel
Lívia Kelebercová
Michal Munk
Júlia Tomanová
Vladimiras Dolgopolas
Małgorzata Przybyła-Kasperek

www.fitped.eu

2024

Knowledge Discovery

Published on

November 2024

Authors

Ľubomír Benko | Constantine the Philosopher University in Nitra, Slovakia

Ján Skalka | Constantine the Philosopher University in Nitra, Slovakia

Jaroslav Reichel | Constantine the Philosopher University in Nitra, Slovakia

Lívia Kelebercová | Constantine the Philosopher University in Nitra, Slovakia

Michal Munk | Constantine the Philosopher University in Nitra, Slovakia

Júlia Tomanová | Constantine the Philosopher University in Nitra, Slovakia

Vladimiras Dolgopolovas | Vilnius University, Lithuania

Małgorzata Przybyła-Kasperek | University of Silesia in Katowice, Poland

Reviewers

Piet Kommers | Helix5, Netherland

Peter Švec | Teacher.sk, Slovakia

Vaida Masiulionytė-Dagienė | Vilnius University, Lithuania

Cyril Klimeš | Mendel University in Brno, Czech Republic

Erasmus+ FITPED-AI

Future IT Professionals Education in Artificial Intelligence

Project 2021-1-SK01-KA220-HED-000032095



**Funded by
the European Union**

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Slovak Academic Association for International Cooperation. Neither the European Union nor the granting authority can be held responsible for them.



Licence (licence type: Attribution-Non-commercial-No Derivative Works) and may be used by third parties as long as licensing conditions are observed. Any materials published under the terms of a CC Licence are clearly identified as such.

All trademarks and brand names mentioned in this publication and all trademarks and brand names mentioned that may be the intellectual property of third parties are unconditionally subject to the provisions contained within the relevant law governing trademarks and other related signs. The mere mention of a trademark or brand name does not imply that such a trademark or brand name is not protected by the rights of third parties.

© 2024 Constantine the Philosopher University in Nitra

ISBN 978-80-558-2227-3

TABLE OF CONTENTS

1 Basic Features	5
1.1 Introduction	6
1.2 Data description.....	9
2 Exploratory Analysis	21
2.1 Descriptive statistics	22
2.2 Data visualisation	34
2.3 Data summarization	48
3 Data Analysis	58
3.1 Univariate analysis.....	59
3.2 Bivariance analysis	73
3.3 Multivariate analysis.....	83
4 Project - Data Analysis	101
4.1 Project – Exploration data analysis	102
5 Analysis of Titanic Data	119
5.1 Analysis of Titanic data.....	120
Summarisation.....	137
6 Introduction.....	138
6.1 What is summarisation	139
6.2 Quality and challenges	141
7 Extraction and Abstraction	143
7.1 Extraction	144
7.2 Abstraction I.....	148
7.3 Abstraction II.....	151
8 Keyword Extraction.....	156
8.1 Introduction	157
8.2 Statistical approaches.....	159
8.3 Graph based approaches	177
8.4 Machine learning based approaches.....	186
8.5 Deep learning based approaches.....	196
8.6 Evaluation	198

Basic Features

Chapter **1**

1.1 Introduction

1.1.1

The Knowledge Discovery - Introduction course focuses on the process of transforming data into information and knowledge. We will introduce the field of knowledge discovery and practically demonstrate how to extract relevant information from data. The course will consist of a theoretical and a practical part that complements each other. We will work in the Python programming language and will use mainly the **Pandas** library.

1.1.2

As more and more data accumulates in today's world, whether on the web or other physical storage, the concept of **Knowledge Discovery** has emerged. By knowledge we mean information that is of value to us. Knowledge discovery can be understood as a process that consists of the following tasks:

- data selection,
- data preprocessing,
- data transformation,
- data analysis,
- results interpretation.

We can discover knowledge from a variety of sources, whether from databases, texts, or the web.

1.1.3

The **CRISP-DM methodology** is one of the most widely used and versatile techniques for solving various knowledge discovery tasks. The methodology consists of the following steps:

- business understanding,
- data understanding,
- data preparation,
- modeling,
- evaluation,
- deployment.

The order of the phases is not fixed and the process is cyclical. It was primarily developed for project management in the area of knowledge discovery from databases, but is applicable to other areas as well.

1.1.4

First, let's recall the work with data files. In our course, we will mainly use the **pandas** library for working with data. Pandas contains a function for importing data from different data files and writing back the output in different formats. Most often we will encounter files saved in CSV format. Reading a CSV file and then transforming it into a tabular structure (**DataFrame**) is built into the pandas library using the **read_csv()** function. The first parameter of the function is the path to the file and the second parameter is **sep**, which we can use to define a separator. The default value in the case of the separator is a comma but we will often encounter a semicolon.

```
import pandas as pd

df = pd.read_csv('dataset.csv', sep=';')
```

1.1.5

Another option is to use datasets provided by other libraries such as **Sklearn**. This library is designed to work with machine learning and provides multiple datasets for different tasks. Using the **import** function, we can import different data files. Then we just need to create an instance of that data file and load it into the pandas DataFrame structure. In the final result, the result is similar to if we loaded a CSV file from disk.

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print(df)
```

Program output:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium
total_phenols \					
0	14.23	1.71	2.43	15.6	127.0
	2.80				
1	13.20	1.78	2.14	11.2	100.0
	2.65				
2	13.16	2.36	2.67	18.6	101.0
	2.80				
3	14.37	1.95	2.50	16.8	113.0
	3.85				
4	13.24	2.59	2.87	21.0	118.0
	2.80				

```

..      ...      ...      ...      ...      ...
...
173     13.71     5.65  2.45     20.5     95.0
1.68
174     13.40     3.91  2.48     23.0     102.0
1.80
175     13.27     4.28  2.26     20.0     120.0
1.59
176     13.17     2.59  2.37     20.0     120.0
1.65
177     14.13     4.10  2.74     24.5     96.0
2.05

      flavanoids  nonflavanoid_phenols  proanthocyanins
color_intensity  hue \
0                3.06                0.28                2.29
5.64  1.04
1                2.76                0.26                1.28
4.38  1.05
2                3.24                0.30                2.81
5.68  1.03
3                3.49                0.24                2.18
7.80  0.86
4                2.69                0.39                1.82
4.32  1.04
..      ...
...     ...
173     0.61                0.52                1.06
7.70  0.64
174     0.75                0.43                1.41
7.30  0.70
175     0.69                0.43                1.35
10.20  0.59
176     0.68                0.53                1.46
9.30  0.60
177     0.76                0.56                1.35
9.20  0.61

      od280/od315_of_diluted_wines  proline
0                3.92  1065.0
1                3.40  1050.0
2                3.17  1185.0
3                3.45  1480.0
4                2.93  735.0

```



```

..          ...      ...
173         1.74     740.0
174         1.56     750.0
175         1.56     835.0
176         1.62     840.0
177         1.60     560.0

[178 rows x 13 columns]

```

1.1.6

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. List the names of the columns that the dataset contains, separated by commas.

```

import pandas as pd
from sklearn.datasets import fetch_california_housing

```

1.2 Data description

1.2.1

In the first part, we focus on the fact that it needs to understand what data we've actually retrieved. However, we don't go in-depth yet because we are trying to first understand the problem we want to solve in the context of the whole dataset and the meaning of the variables. So let's look first at how much and what type of data is in the data set. This is what the `shape()` and `info()` functions that describe the data set are there to do. Shape returns information about the number of rows and columns. Info also provides more detailed information about the individual variables and especially their data type.

```

import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
print(df.info())

```

Program output:

```

RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   alcohol                                178 non-null    float64
1   malic_acid                             178 non-null    float64
2   ash                                     178 non-null    float64
3   alcalinity_of_ash                      178 non-null    float64
4   magnesium                              178 non-null    float64
5   total_phenols                          178 non-null    float64
6   flavanoids                             178 non-null    float64
7   nonflavanoid_phenols                   178 non-null    float64
8   proanthocyanins                        178 non-null    float64
9   color_intensity                        178 non-null    float64
10  hue                                     178 non-null    float64
11  od280/od315_of_diluted_wines          178 non-null    float64
12  proline                                 178 non-null    float64
dtypes: float64 (13)
memory usage: 18.2 KB
None

```

The dataset contains 178 rows and 13 columns. All variables are in decimal format. We can also see that the dataset does not contain any missing values.

1.2.2

Load from the *sklearn* library the dataset `california_housing` that contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. Examine the dataset and select the correct assertions about the retrieved data.

```

import pandas as pd
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(df.info())

```

Program output:

```

RangeIndex: 20640 entries, 0 to 20639

```

```
Data columns (total 8 columns):
#      Column          Non-Null Count  Dtype
---  -
0      MedInc             20640 non-null  float64
1      HouseAge           20640 non-null  float64
2      AveRooms            20640 non-null  float64
3      AveBedrms           20640 non-null  float64
4      Population          20640 non-null  float64
5      AveOccup            20640 non-null  float64
6      Latitude            20640 non-null  float64
7      Longitude           20640 non-null  float64
dtypes: float64(8)
memory usage: 1.3 MB
None
```

- the dataset consists of 20640 rows and 8 columns
- all variables are in decimal format
- the dataset consists of 8 rows and 20640 columns
- all variables are in integer format
- the dataset also contains missing values
- the dataset does not contain missing values

1.2.3

Most often, the first functions used when loading a data file are the pandas **head()** and **tail()** library functions. These functions display the first and last 5 records of the dataset. In this way, we are able to quickly explore a small portion of the data file.

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print('Head:')
print(df.head())
print('Tail:')
print(df.tail())
```

Program output:

```
Head:
  alcohol  malic_acid  ash  alcalinity_of_ash  magnesium
total_phenols  \
```

```

0    14.23      1.71  2.43      15.6      127.0
2.80
1    13.20      1.78  2.14      11.2      100.0
2.65
2    13.16      2.36  2.67      18.6      101.0
2.80
3    14.37      1.95  2.50      16.8      113.0
3.85
4    13.24      2.59  2.87      21.0      118.0
2.80

```

```

      flavanoids  nonflavanoid_phenols  proanthocyanins
color_intensity  hue  \
0          3.06          0.28          2.29
5.64  1.04
1          2.76          0.26          1.28
4.38  1.05
2          3.24          0.30          2.81
5.68  1.03
3          3.49          0.24          2.18
7.80  0.86
4          2.69          0.39          1.82
4.32  1.04

```

```

      od280/od315_of_diluted_wines  proline
0          3.92      1065.0
1          3.40      1050.0
2          3.17      1185.0
3          3.45      1480.0
4          2.93      735.0

```

Tail:

```

      alcohol  malic_acid  ash  alcalinity_of_ash  magnesium
total_phenols  \
173    13.71      5.65  2.45      20.5      95.0
1.68
174    13.40      3.91  2.48      23.0      102.0
1.80
175    13.27      4.28  2.26      20.0      120.0
1.59
176    13.17      2.59  2.37      20.0      120.0
1.65
177    14.13      4.10  2.74      24.5      96.0
2.05

```

	flavanoids	nonflavanoid_phenols	proanthocyanins
color_intensity	hue \		
173	0.61	0.52	1.06
7.7	0.64		
174	0.75	0.43	1.41
7.3	0.70		
175	0.69	0.43	1.35
10.2	0.59		
176	0.68	0.53	1.46
9.3	0.60		
177	0.76	0.56	1.35
9.2	0.61		
	od280/od315_of_diluted_wines	proline	
173	1.74	740.0	
174	1.56	750.0	
175	1.56	835.0	
176	1.62	840.0	
177	1.60	560.0	

1.2.4

Load from the *sklearn* library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. The dataset consists of the following variables:

- `MedInc` - the median income of homes in the block
- `HouseAge` - the median age of houses in the block
- `AveRooms` - the average number of rooms per household
- `AveBedrms` - the average number of bedrooms per household
- `Population` - population
- `AveOccup` - the average number of household members
- `Latitude` - latitude of the block
- `Longitude` - longitude of the block

Examine the dataset and list the median age of the houses of the first block. Round the result to a whole number.

```
import pandas as pd
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
```

```
print(df.head())
```

Program output:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467

	Longitude
0	-122.23
1	-122.22
2	-122.24
3	-122.25
4	-122.25

1.2.5

The **describe()** function provides purely descriptive information about the dataset. This information includes statistics that summarize the variables, their variance, the presence of missing values, and their shape. The basic statistics displayed by the function are as follows:

- count - number of elements,
- mean - average value,
- std - standard deviation of observations
- min - minimum value
- 25% - lower quartile
- 50% - median
- 75% - upper quartile
- max - maximum value

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
```

```
print(df.describe())
```

Program output:

```

      alcohol  malic_acid      ash  alcalinity_of_ash
magnesium \
count  178.000000  178.000000  178.000000      178.000000
178.000000
mean    13.000618    2.336348    2.366517      19.494944
99.741573
std     0.811827    1.117146    0.274344      3.339564
14.282484
min     11.030000    0.740000    1.360000      10.600000
70.000000
25%     12.362500    1.602500    2.210000      17.200000
88.000000
50%     13.050000    1.865000    2.360000      19.500000
98.000000
75%     13.677500    3.082500    2.557500      21.500000
107.000000
max     14.830000    5.800000    3.230000      30.000000
162.000000

      total_phenols  flavanoids  nonflavanoid_phenols
proanthocyanins \
count    178.000000  178.000000      178.000000
178.000000
mean     2.295112    2.029270      0.361854
1.590899
std     0.625851    0.998859      0.124453
0.572359
min     0.980000    0.340000      0.130000
0.410000
25%     1.742500    1.205000      0.270000
1.250000
50%     2.355000    2.135000      0.340000
1.555000
75%     2.800000    2.875000      0.437500
1.950000
max     3.880000    5.080000      0.660000
3.580000

      color_intensity      hue
od280/od315_of_diluted_wines  proline
count    178.000000  178.000000
178.000000  178.000000

```

```

mean          5.058090    0.957449
2.611685     746.893258
std           2.318286    0.228572
0.709990     314.907474
min           1.280000    0.480000
1.270000     278.000000
25%           3.220000    0.782500
1.937500     500.500000
50%           4.690000    0.965000
2.780000     673.500000
75%           6.200000    1.120000
3.170000     985.000000
max           13.000000    1.710000
4.000000     1680.000000

```

1.2.6

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. What is the average value of the average population per block?

```

import pandas as pd
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(df.describe())

```

Program output:

```

           MedInc      HouseAge      AveRooms      AveBedrms
Population \
count  20640.000000  20640.000000  20640.000000  20640.000000
20640.000000
mean       3.870671    28.639486     5.429000     1.096675
1425.476744
std        1.899822    12.585558     2.474173     0.473911
1132.462122
min         0.499900     1.000000     0.846154     0.333333
3.000000
25%        2.563400    18.000000     4.440716     1.006079
787.000000

```



```

50%          3.534800    29.000000    5.229129    1.048780
1166.000000
75%          4.743250    37.000000    6.052381    1.099526
1725.000000
max          15.000100    52.000000    141.909091   34.066667
35682.000000

           AveOccup    Latitude    Longitude
count  20640.000000  20640.000000  20640.000000
mean     3.070655    35.631861   -119.569704
std     10.386050     2.135952     2.003532
min     0.692308    32.540000   -124.350000
25%     2.429741    33.930000   -121.800000
50%     2.818116    34.260000   -118.490000
75%     3.282261    37.710000   -118.010000
max    1243.333333    41.950000   -114.310000

```

1.2.7

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. What is the median age of the houses in the block? Print the result as an integer.

```

import pandas as pd
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(df.describe())

```

Program output:

```

           MedInc    HouseAge    AveRooms    AveBedrms
Population \
count  20640.000000  20640.000000  20640.000000  20640.000000
20640.000000
mean     3.870671    28.639486     5.429000     1.096675
1425.476744

```

```

std      1.899822      12.585558      2.474173      0.473911
1132.462122
min      0.499900      1.000000      0.846154      0.333333
3.000000
25%     2.563400      18.000000      4.440716      1.006079
787.000000
50%     3.534800      29.000000      5.229129      1.048780
1166.000000
75%     4.743250      37.000000      6.052381      1.099526
1725.000000
max     15.000100      52.000000     141.909091     34.066667
35682.000000

          AveOccup      Latitude      Longitude
count  20640.000000  20640.000000  20640.000000
mean    3.070655     35.631861    -119.569704
std     10.386050     2.135952     2.003532
min     0.692308     32.540000    -124.350000
25%     2.429741     33.930000    -121.800000
50%     2.818116     34.260000    -118.490000
75%     3.282261     37.710000    -118.010000
max     1243.333333    41.950000    -114.310000

```

1.2.8

Another way to get to know a data file is to use the **info()** function. This function gives us more concise information than **describe()** but we get information about the data type of the variables. We can also use the **info()** function to find out if the data file contains missing values.

```

import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print(df.info())

```

Program output:

```

RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   alcohol               178 non-null    float64

```

```

1  malic_acid          178 non-null    float64
2  ash                 178 non-null    float64
3  alkalinity_of_ash  178 non-null    float64
4  magnesium          178 non-null    float64
5  total_phenols      178 non-null    float64
6  flavanoids         178 non-null    float64
7  nonflavanoid_phenols 178 non-null    float64
8  proanthocyanins    178 non-null    float64
9  color_intensity    178 non-null    float64
10 hue                178 non-null    float64
11 od280/od315_of_diluted_wines 178 non-null    float64
12 proline            178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
None

```

1.2.9

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. What data type are most of the variables in the dataset?

```

import pandas as pd
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(df.info())

```

Program output:

```

RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   MedInc          20640 non-null   float64
1   HouseAge        20640 non-null   float64
2   AveRooms        20640 non-null   float64
3   AveBedrms       20640 non-null   float64
4   Population      20640 non-null   float64
5   AveOccup        20640 non-null   float64

```

```
6  Latitude    20640 non-null  float64
7  Longitude   20640 non-null  float64
dtypes: float64(8)
memory usage: 1.3 MB
None
```

Exploratory Analysis

Chapter **2**

2.1 Descriptive statistics

2.1.1

Exploratory analysis methods are used to discover patterns, generate hypotheses, recognize specificities, and illustrate phenomena. The starting point of any data analysis is the data itself. The data do not have to satisfy certain conditions (e.g. the data must have been obtained by random sampling). The main point is to represent the data in different ways and to recognise regularities and irregularities, structures, patterns and peculiarities. In the exploratory process, we look for interesting configurations and relationships in the data. If we want to compare two or more variables, we need appropriate quantities that will numerically characterize the basic properties of the frequency distribution. Such amounts are called **numerical characteristics** and can be divided into three categories:

- *position characteristics* - represent a certain level or position of the character around which the residuals are concentrated. This position is measured by different kinds of mean values such as **arithmetic, harmonic and geometric mean, modus, median and quantiles**.
- *variability characteristics* - they express the differences (variability, dispersion) of the values and are an important factor when comparing variables in which the position characteristics are identical. The best known are **quantile, quartile and variation range, quartile deviation, mean deviation, proportional mean deviation, variance, standard deviation and coefficient of variation**.
- *characteristics of skewness and peakedness measures* - moment characteristics are required for their calculation. The best known are **the skewness coefficient, the kurtosis coefficient and the Pearson skewness measure**.

2.1.2

Most descriptive statistics include Python functions. However, in order to understand what is behind the called function, we need to understand at least the mathematical notation of the statistics. Let's first introduce different averages.

Arithmetic mean - is the sum of all given values divided by their number. In Python, we can use the **mean()** function of the *statistics* library to calculate it.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Harmonic mean - is the inverse of the arithmetic mean of the inverted values. In Python, we can use the `harmonic_mean()` function of the `statistics` library to calculate it.

$$x_H = \frac{n}{\sum_{i=1}^n \frac{1}{x_i}}$$

Geometric mean - is the product of the positive numbers is the product of the values squared to the number of values. The similarity to the arithmetic mean is in the substitution of the sum of the operation by product and division by the n-th root. In Python, we can use the `geometric_mean()` function of the `statistics` library to do the calculation.

$$x_G = \sqrt[n]{\prod_{i=1}^n x_i}$$

```
import pandas as pd
import statistics as stat
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print('Arithmetic mean:', stat.mean(df['magnesium']))
print('Harmonic mean:', stat.harmonic_mean(df['magnesium']))
print('Geometric mean:', stat.geometric_mean(df['magnesium']))
```

Program output:

```
Arithmetic mean: 99.74157303370787
Harmonic mean: 97.9056614747819
Geometric mean: 98.79450755406194
```

2.1.3

Load from the `sklearn` library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the

`fetch_california_housing()` function. What is the value of the harmonic mean of the age of the houses in the block? Round the result to two decimal places.

```
import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(round(stats.hmean(df['HouseAge']), 2))
```

Program output:

20.38

2.1.4

Other statistics used include the modus and median.

Modus - represents the most frequent value occurring in the variable under study. In Python, we can use the `mode()` function of the *statistics* library to calculate it.

$$\hat{x} = a_{Mo} + h \frac{d_1}{d_1 + d_2}$$

Median - this is the mean value of the variable under study, with the requirement that the values must be arranged in a non-decreasing sequence. We have defined n as the number of values and x_i as the value at the i -th position. Then for an even number of elements we calculate the median as follows:

$$\tilde{x} = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$$

For an odd number of elements, we proceed as follows:

$$\tilde{x} = x_{\frac{n+1}{2}}$$

In Python, we can use the `median()` function of the *statistics* library to calculate.

We distinguish three cases depending on what is the relative position of the *modus*, *median* and *arithmetic mean* of the examined variable. If , then we speak about **symmetric frequency distribution**. If , then we speak about **negative skewness**. In the case of , we speak about **positive skewness**.

```
import pandas as pd
import statistics as stat
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print('Modus: ',stat.mode(df['magnesium']))
print('Median: ',stat.median(df['magnesium']))
```

Program output:

```
Modus: 88.0
Median: 98.0
```

2.1.5

Use of individual position characteristics:

- We use the mean mainly for metric variables in the case of symmetric distributions and the use of parametric tests.
- We use the median for intensive variables in the case we want to know the centre of the data distribution, in the case of outliers and skewed distribution.
- We use the modus for variables when the distribution has multiple peaks.
- In the case of a symmetric distribution, all these characteristics are approximately the same.

2.1.6

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. What is the most common value for the age of the houses in the block? Print the result as an integer.

```
import pandas as pd
import statistics as stat
from sklearn.datasets import fetch_california_housing
```

```
cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(round(stat.mode(df['HouseAge']), 2))
```

Program output:

```
52.0
```

 2.1.7

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. Examine the variable age of the houses in the block and identify the frequency distribution of the variable being examined. List the values of the mean, median, and mode rounded to two decimal places in the following form:

```
positive skewness, mean: 42.53, median: 22.36, modus: 30.00
```

```
import pandas as pd
import statistics as stat
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(round(stat.mode(df['HouseAge']), 2))
print(round(stat.mean(df['HouseAge']), 2))
print(round(stat.median(df['HouseAge']), 2))
```

Program output:

```
52.0
28.64
29.0
```

 2.1.8

Quantiles are numerical values that divide the sorted values of the variable under study in non-decreasing order into k equal parts. The best-known are the median ($k=2$), quartiles ($k=4$), deciles ($k=10$) and percentiles ($k=100$).

Quartiles represent percentiles with levels of 25%, 50% and 75%. Quartiles divide the set into 4 parts.

- Q_I is the first/lower quartile and the 25th percentile or $x_{0.25}$.
- Q_{II} is the second quartile or 50th percentile or median $x_{0.5}$.
- Q_{III} is the third/upper quartile or 75th percentile or $x_{0.75}$.

$$Q_1^{(4)} = a_1^{(4)} + h \frac{\frac{n}{4} + 0.5 - N_{j-1}}{n_j}$$

$$Q_3^{(4)} = a_3^{(4)} + h \frac{\frac{3n}{4} + 0.5 - N_{j-1}}{n_j}$$

In Python, we have two options to get the upper and lower quartile. The first option is the **describe()** function of the *pandas* library. The second option is to use the *numpy* library, which contains a **quantile()** function whose second parameter is the percentile. So if we specify 0.25 as a parameter the function will result in a lower quartile and 0.75 will result in an upper quartile.

Using the upper and lower quartiles, we can calculate the quartile range which represents the region of the middle 50% of the values of the variable. This measure of variability is not affected by extreme values of the variable. In Python, we can use the **iqr()** function of the *scipy* library to calculate this or substitute the upper and lower quartiles into the formula:

$$R_Q^4 = Q_3^{(4)} - Q_1^{(4)}$$

```
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
print(df['magnesium'].describe())
print('Upper quartile:', np.quantile(df['magnesium'], 0.75))
print('Lower quartile:', np.quantile(df['magnesium'], 0.25))
print('Quartile range:', stats.iqr(df['magnesium']))
```

Program output:

```

count      178.000000
mean       99.741573
std        14.282484
min        70.000000
25%        88.000000
50%        98.000000
75%       107.000000
max       162.000000
Name: magnesium, dtype: float64
Upper quartile: 107.0
Lower quartile: 88.0
Quartile range: 19.0

```

 2.1.9

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. Examine the variable `age` of houses in the block and calculate the quartile range of the variable being examined. Round the result to integers.

```

import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print('Quartile range:', stats.iqr(df['HouseAge']))

```

Program output:

```
Quartile range: 19.0
```

 2.1.10

Data with the same mean can have different scatter. The amount of variability in the data can be determined by a suitably chosen variability characteristic or measure of dispersion. One of these is the **quartile range** introduced earlier. Others are:

The variance - the most commonly used characteristic of variability, referred to as s^2 , which is the root mean square deviation of the measurement from the arithmetic

mean. The larger the variance the more the data deviate from the mean. In Python, we can use the `var()` function of the `numpy` library or the `pvariance()` function of the `statistics` library to calculate this.

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

Standard deviation - this is the positive square root of the variance, denoted as **s**. The greater the difference in the values of the examined variable the greater the value of the standard deviation. In Python, we can use the `std()` function of the `numpy` library or the `pstdev()` function of the `statistics` library to do the calculation.

$$s = \sqrt{s^2}$$

Coefficient of variation - used for comparing variability and represents a relative measure of variability. It does not depend on the units in which the values of the variable are expressed, unlike the variance and standard deviation. If the value of the coefficient of variation is greater than 50%, the arithmetic mean loses its meaning because the statistical population is heterogeneous and the arithmetic mean cannot represent it. In this case, we use the median instead of the arithmetic mean as mean. In Python, we have to calculate the given coefficient using the following formula:

$$v = \frac{s}{\bar{x}} * 100$$

```
import pandas as pd
import statistics as stat
import numpy as np
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print('The variance
Statistics:', stat.pvariance(df['magnesium']))
print('The variance Numpy:', np.var(df['magnesium']))
print('Standard deviation
Statistics:', stat.pstdev(df['magnesium']))
print('Standard deviation Numpy:', np.std(df['magnesium']))
```

```
print('Coefficient of variation
Statistics:', stat.pstdev(df['magnesium'])/stat.mean(df['magnesium'])*100)
print('Coefficient of variation
Numpy:', np.std(df['magnesium'])/np.mean(df['magnesium'])*100)
```

Program output:

```
The variance Statistics: 202.8433278626436
The variance Numpy: 202.8433278626436
Standard deviation Statistics: 14.242307673359806
Standard deviation Numpy: 14.242307673359806
Coefficient of variation Statistics: 14.27920899998899
Coefficient of variation Numpy: 14.27920899998899
```

2.1.11

Use of individual variability characteristics:

- Standard deviation and variance measure the dispersion around the mean and are used when the mean is appropriate as a measure of the mean.
- Standard deviation and dispersion are strongly affected by outliers, so in this case, we prefer the quartile range, median absolute deviation, and mean absolute deviation from the median, respectively.
- In the case of a strongly skewed distribution, the standard deviation and variance do not provide good information about the dispersion of the data.
- In case we want to assess the relative magnitude of the dispersion of the data from the mean we use the coefficient of variation.

2.1.12

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. Examine the variable `age` of the houses in the block to see if the coefficient of variation is greater than 50%. List the yes/no values and write the result as a percentage rounded to two decimal places. For example:

```
yes, 58.56%
```

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
```

```
print('Coefficient of
variation: ', round(np.std(df['HouseAge'])/np.mean(df['HouseAge'])
)*100,2))
```

Program output:

```
Coefficient of variation: 43.94
```

 2.1.13

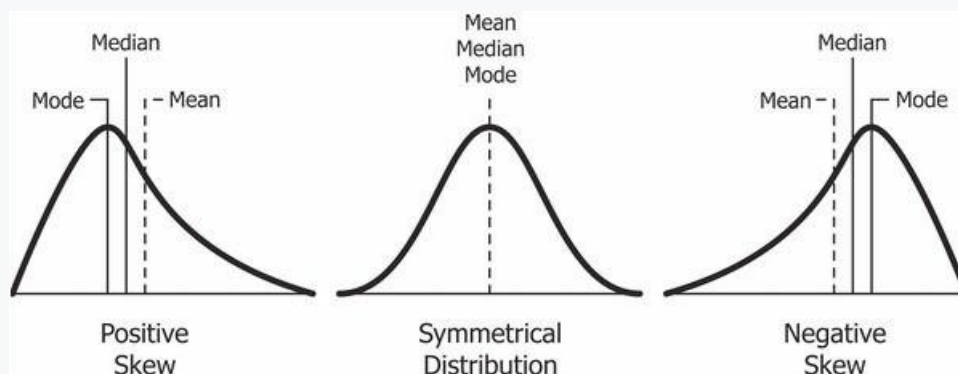
A final option in descriptive statistics is to look at the shape of the data distribution using skewness and kurtosis.

The skewness a_3 measures the degree of asymmetry in the distribution of a variable. A positive value means that the mean is greater than the median, so most of the values are less than the mean. In this case, the distribution is skewed to the left. A negative value means that the median is greater than the mean and hence most values are greater than the mean. In this case, the distribution is skewed to the right. Values close to 0 indicate a symmetric distribution, which means that the mean and median are equal. In Python, we can use the **skew()** function of the *scipy* library to calculate this. It is calculated as follows:

$$a_3 = \frac{m_3}{s^3}$$

where

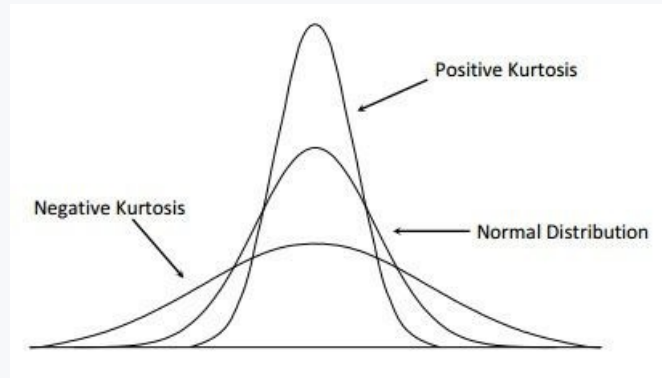
$$m_k = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$



Kurtosis a_4 measures the degree of steepness of the distribution of a variable. A positive value means that the distribution is more skewed. A negative value means

that the distribution is flatter. In Python, we can use the `kurtosis()` function of the `scipy` library to calculate this. It is given by the relation

$$a_4 = \frac{m_4}{s^4} - 3$$



```
import pandas as pd
from scipy import stats
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
print('Skewness:', stats.skew(df['magnesium']))
print('Kurtosis:', stats.kurtosis(df['magnesium'],
fisher=True))
```

Program output:

```
Skewness: 1.088914887210701
Kurtosis: 2.0128060084773907
```

2.1.14

If we have non-zero values for the result of skewness and skewness, then it is obvious that the data under study do not have a normal distribution. However, it may be that the values are close enough, but not quite equal to 0. We can use the **Shapiro-Wilk test** to estimate the probability that the data under study have a normal distribution. The null hypothesis of the Shapiro-Wilk test is that the data have a normal distribution. If the resulting *p-value* is less than or equal to 0.05, we reject the null hypothesis and assume that the data under study do not have a normal distribution. In Python, we can use the `shapiro()` function of the `scipy` library to perform the calculation.

Using individual shape characteristics:

- We use skewness if we want to see if lower values are more frequent than higher values or vice versa.
- We use kurtosis if we want to see how the values of a variable actually cluster around the mean.

```
import pandas as pd
from scipy import stats
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
p_value = round(stats.shapiro(df['magnesium'])[1],4)
if p_value<=0.05:
    print('p =',p_value, 'the null hypothesis is rejected')
else:
    print('p =',p_value, 'the null hypothesis is not rejected')
```

Program output:

```
p = 0.0 the null hypothesis is rejected
```

 **2.1.15**

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. Examine the variable `age` of the houses in the block to see if the variable has a normal distribution. Print if it does/does not have a normal distribution and also list the associated skewness, and kurtosis statistics and verify the p-value. Round the results to two decimal places. Notation:

```
does not have a normal distribution, p = 0.02, skew = 0.12,
kurtosis = -0.25
```

```
import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print('Skewness:',round(stats.skew(df['HouseAge']),2))
print('Kurtosis:',round(stats.kurtosis(df['HouseAge'],
fisher=True),2))
```

```
p_value = round(stats.shapiro(df['HouseAge'])[1],4)
if p_value<=0.05:
    print('p =',p_value, 'the null hypothesis is rejected')
else:
    print('p =',p_value, 'the null hypothesis is not rejected')
```

Program output:

```
Skewness: 0.06
Kurtosis: -0.8
p = 0.0 the null hypothesis is rejected
```

2.2 Data visualisation

2.2.1

Data visualization can tell us much more about the data than just the numbers. With visualization, we can more easily uncover configurations and data structures. We use graphical methods to look for outliers, recognize clusters in data, check data distributions and assumptions, explore relationships between variables, compare measures of mean and variance, or examine time-dependent data. Graphical methods are useful for showing broader properties of data. If we want to present the selected data in a precise form it is better to show it in tables. When analyzing a graph we evaluate densities, clusters, gaps, outliers, and the shape of the distribution.

Graphs can be grouped according to different criteria. In our case, we will divide them by usage. However, we will by no means cover all possibilities but we will try to present the most important ones. Some graphs are so specific that they are only part of specific analyses. An example of such a graph is the dendrogram that is part of cluster analysis and is used to visualize clusters in the data space.

2.2.2

We can examine the abundance of the data in each variable in different ways. One possibility is by using the **value_counts()** function of the *pandas* library. The result is a listing of the unique values and the number of repetitions in the data set. If we set the **normalize** parameter in the function to *True*, the resulting counts are output in percentage notation. The last option is to visualize the frequencies using the **plot()** function, where we can choose a bar chart type by setting the **kind** parameter to *bar*.

We have also added a *target* column to our data file. This column is used for the classification task, where based on the other variables we can classify the wine into the given three categories. In our case, for the moment, it will mainly serve us to better understand the data.

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
df["target"] = wine.target
print('Frequencies:',df['target'].value_counts(),sep='\n')
print('Percentages:',df['target'].value_counts(normalize=True)
,sep='\n')
df['target'].value_counts().plot(kind='bar')
```

Program output:

Frequencies:

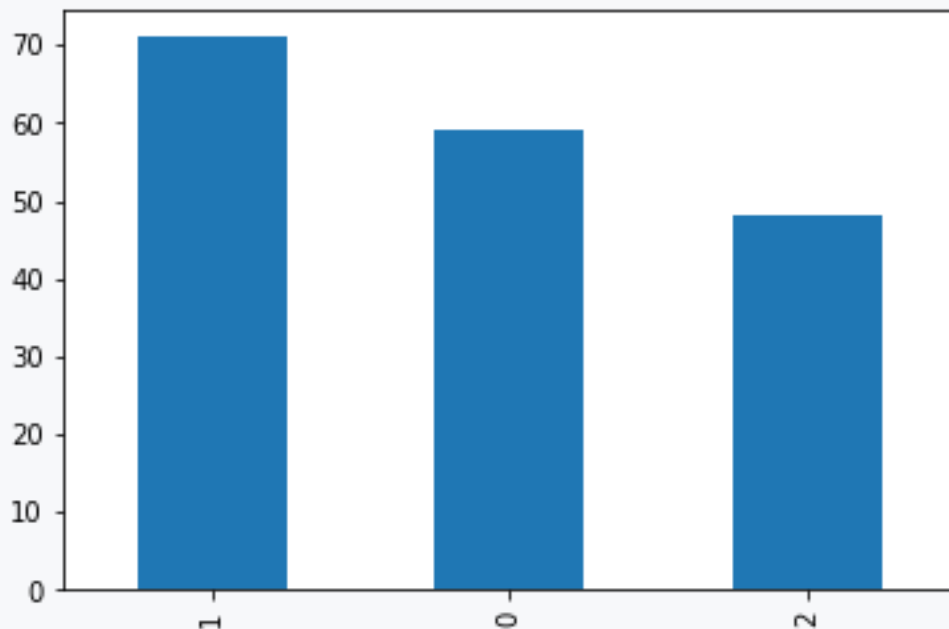
```
1    71
0    59
2    48
```

Name: target, dtype: int64

Percentages:

```
1    0.398876
0    0.331461
2    0.269663
```

Name: target, dtype: float64



2.2.3

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the

`fetch_california_housing()` function. What is the number of oldest houses by the average age of the houses in the block? List the average age and the number of records for it.

24: 875

```
import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
df['HouseAge'].value_counts().plot(kind='bar')
print(df.HouseAge.value_counts())
```

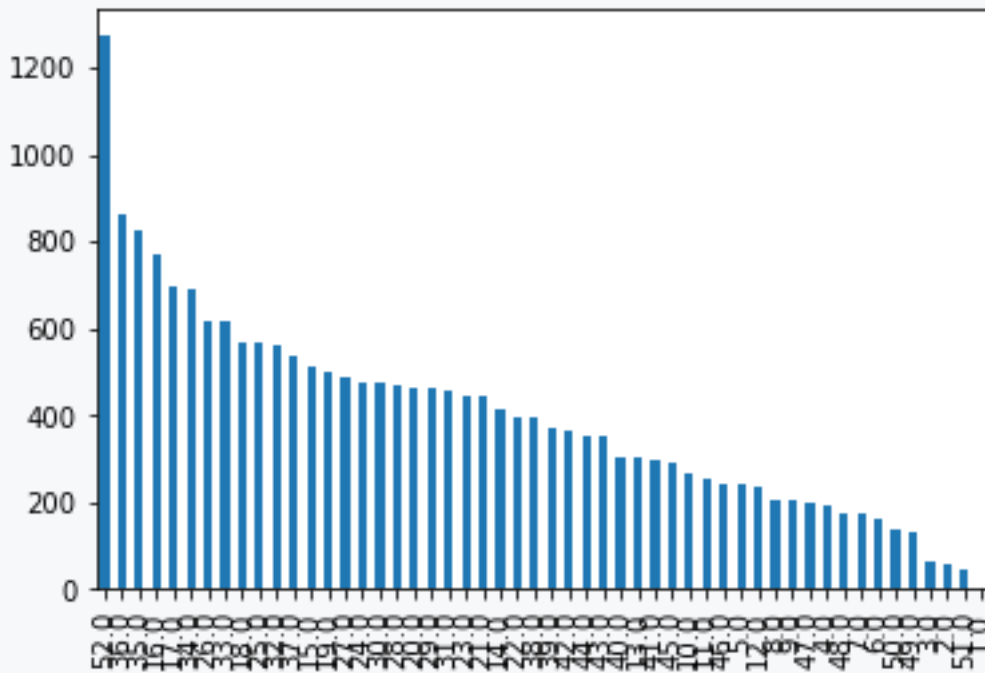
Program output:

```
52.0    1273
36.0     862
35.0     824
16.0     771
17.0     698
34.0     689
26.0     619
33.0     615
18.0     570
25.0     566
32.0     565
37.0     537
15.0     512
19.0     502
27.0     488
24.0     478
30.0     476
28.0     471
20.0     465
29.0     461
31.0     458
23.0     448
21.0     446
14.0     412
22.0     399
38.0     394
39.0     369
42.0     368
```

```

44.0    356
43.0    353
40.0    304
13.0    302
41.0    296
45.0    294
10.0    264
11.0    254
46.0    245
5.0     244
12.0    238
8.0     206
9.0     205
47.0    198
4.0     191
48.0    177
7.0     175
6.0     160
50.0    136
49.0    134
3.0     62
2.0     58
51.0    48
1.0     4
    
```

Name: HouseAge, dtype: int64



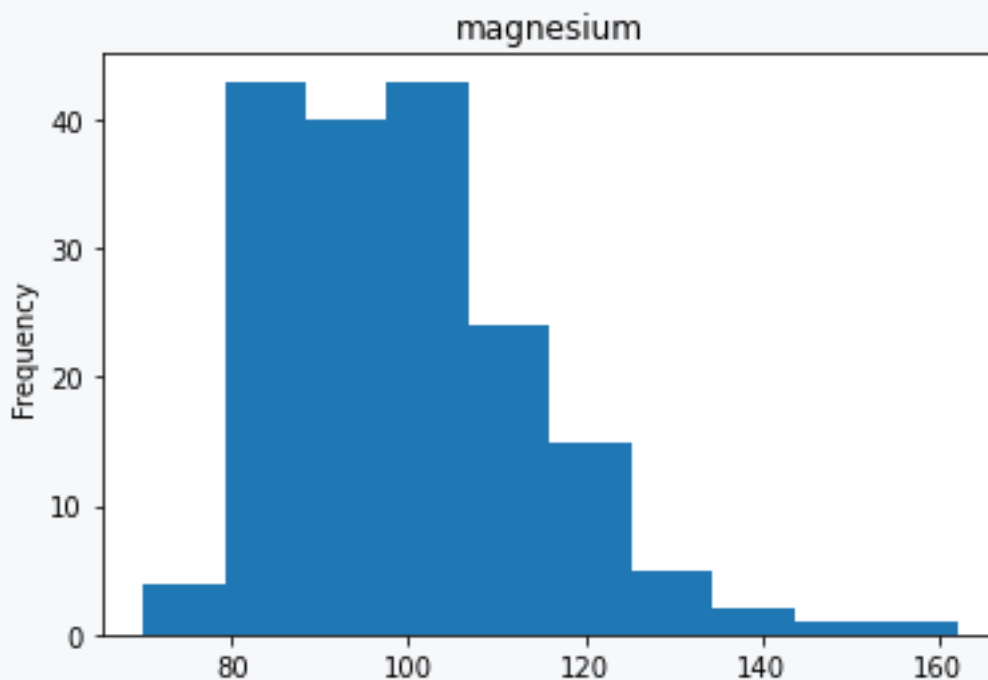
2.2.4

If we want to look at the distribution of the data or the distribution of the data, we can use a **histogram**. The histogram works with intervals where the intervals are represented by the width of the bar (x-axis) and the number of cases that fall within the interval is represented by the height of the bar (y-axis). Visualization of the histogram is possible using the **plot()** function, where we can choose the type of the plot by setting the *kind* parameter to **hist**.

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
df['magnesium'].plot(kind='hist', title='magnesium')
```

Program output:



2.2.5

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the **fetch_california_housing()** function. Visualize a histogram of each variable in the dataset. Which of the histograms visualize information about the rooms in the houses?

```

import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(df.info())
#df['MedInc'].plot(kind='hist')
#df['HouseAge'].plot(kind='hist')
#df['AveRooms'].plot(kind='hist')
#df['AveBedrms'].plot(kind='hist')
#df['Population'].plot(kind='hist')
#df['AveOccup'].plot(kind='hist')
#df['Latitude'].plot(kind='hist')
#df['Longitude'].plot(kind='hist')

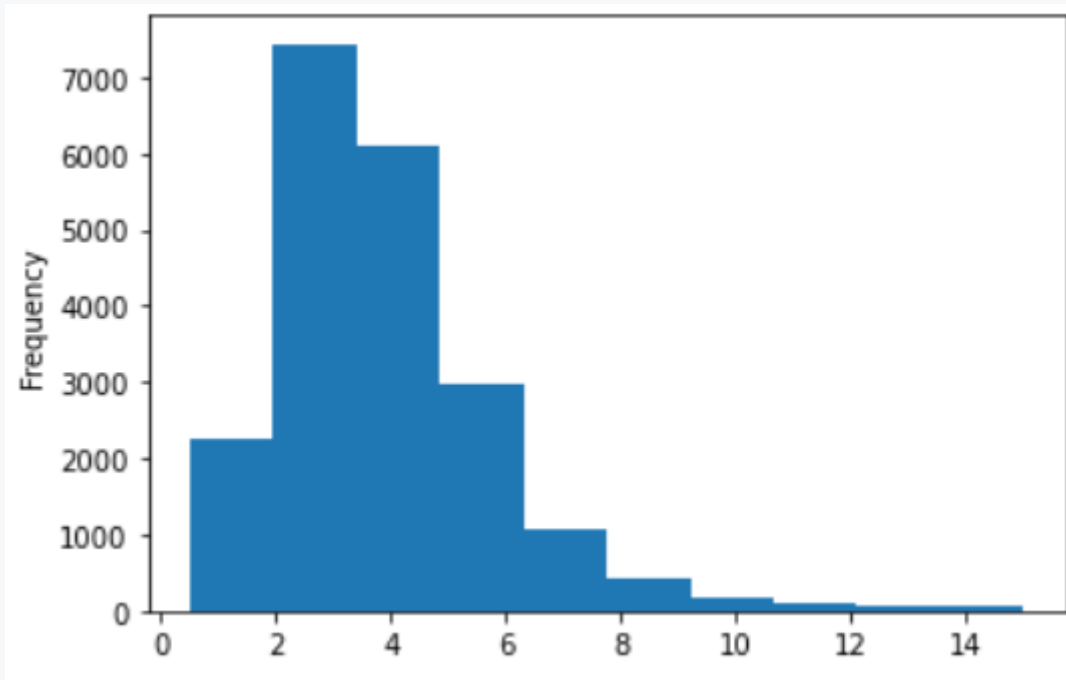
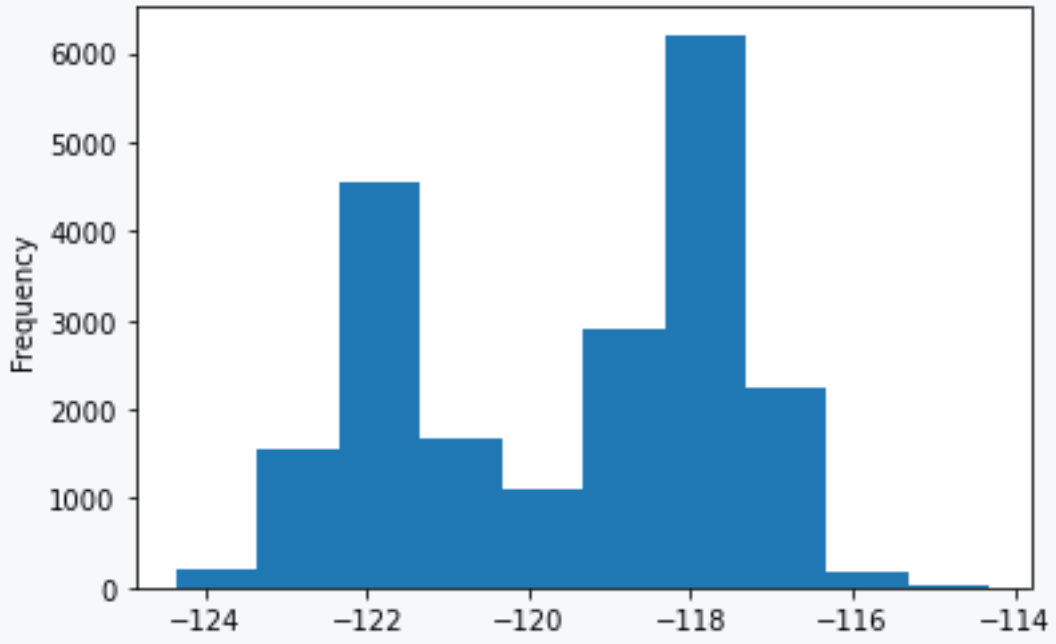
```

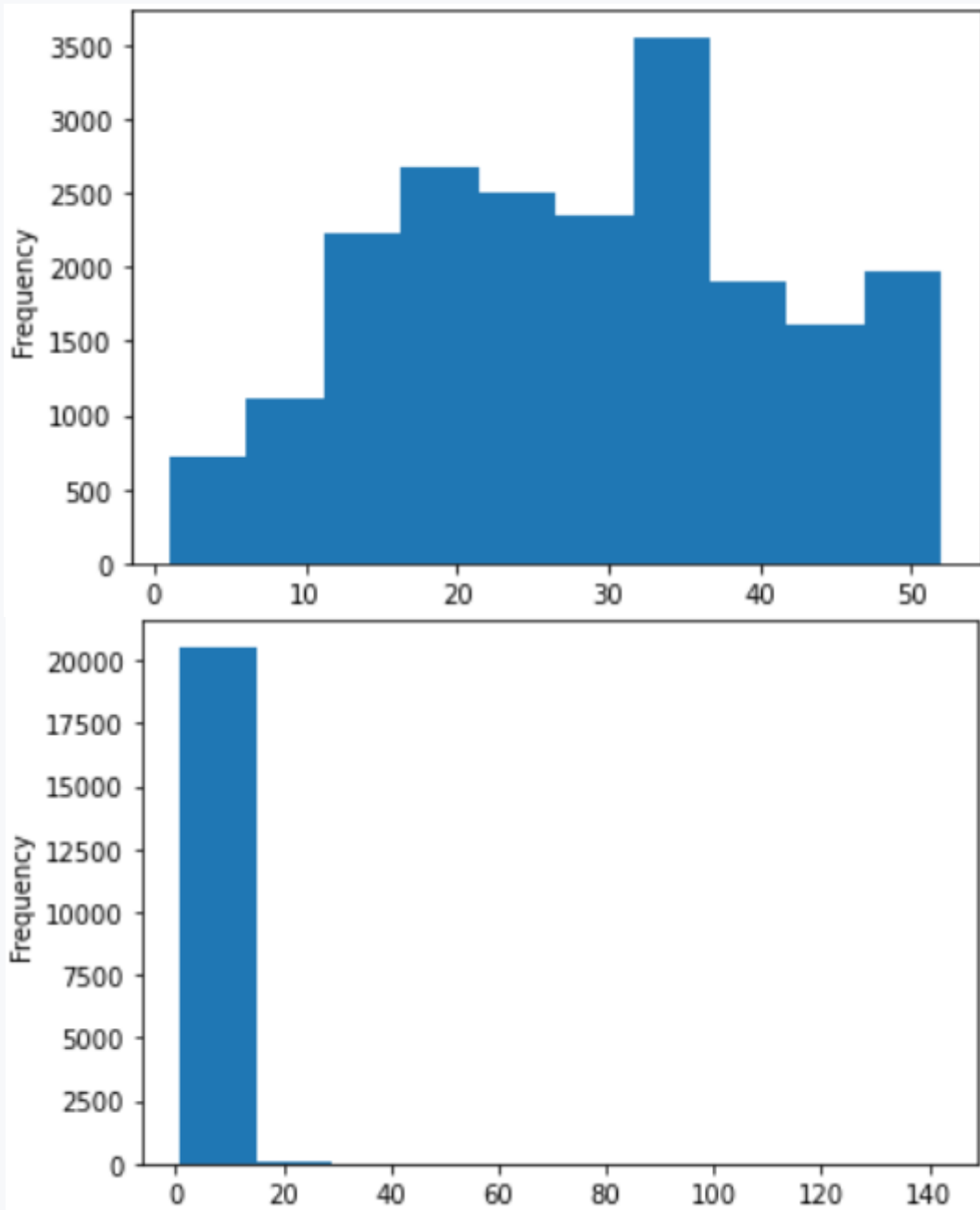
Program output:

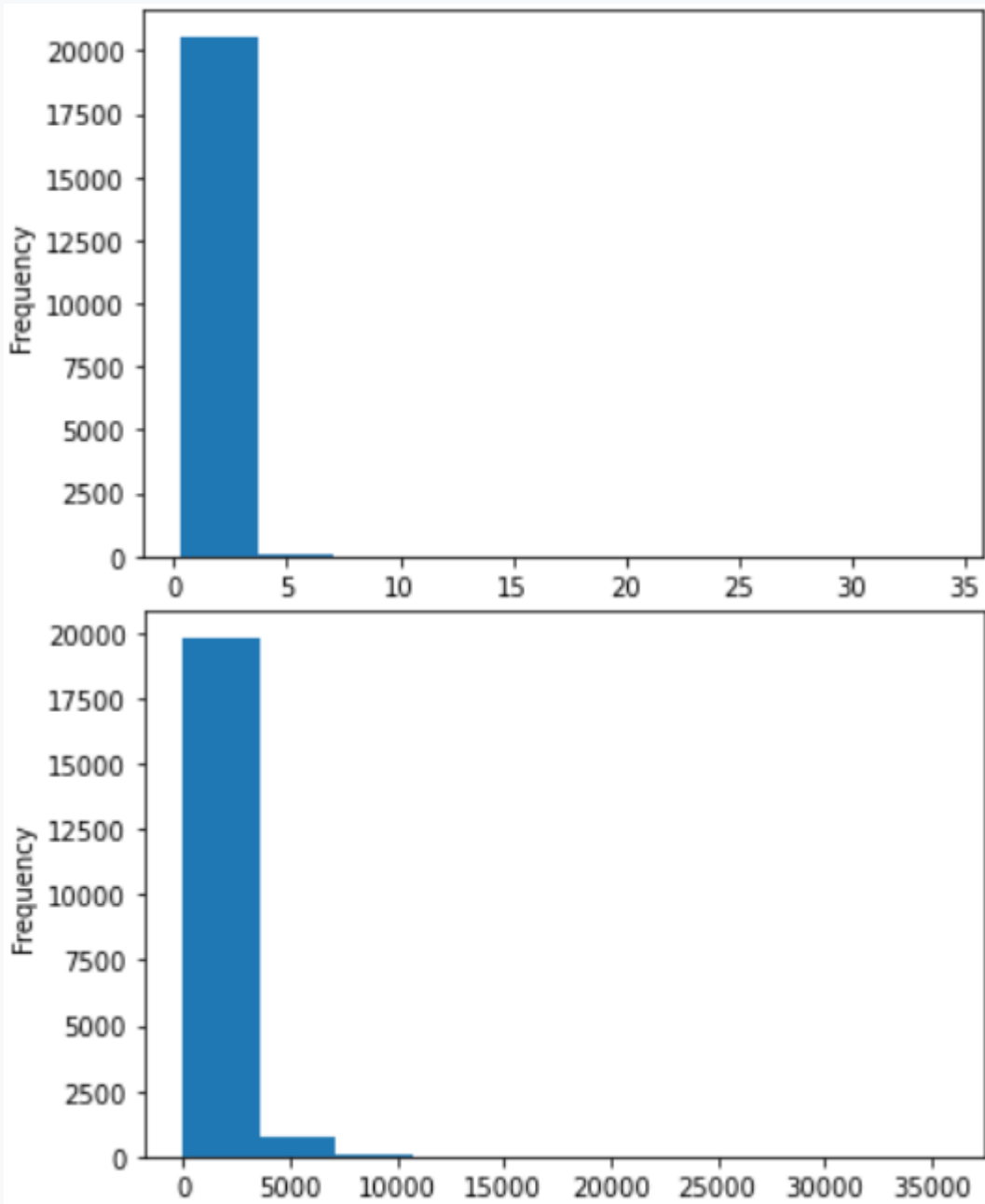
```

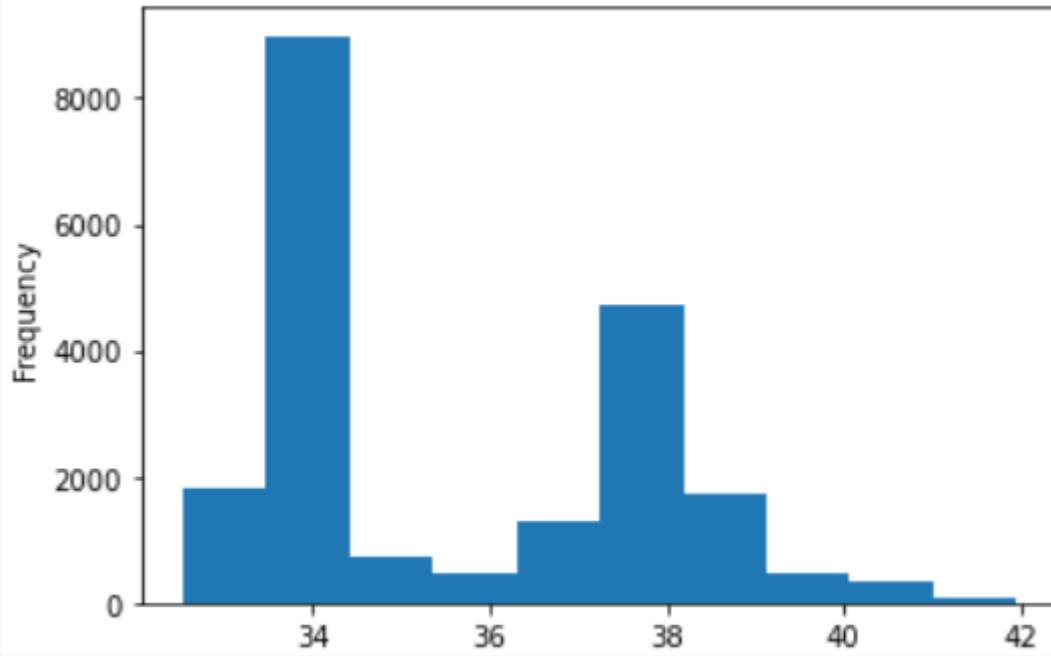
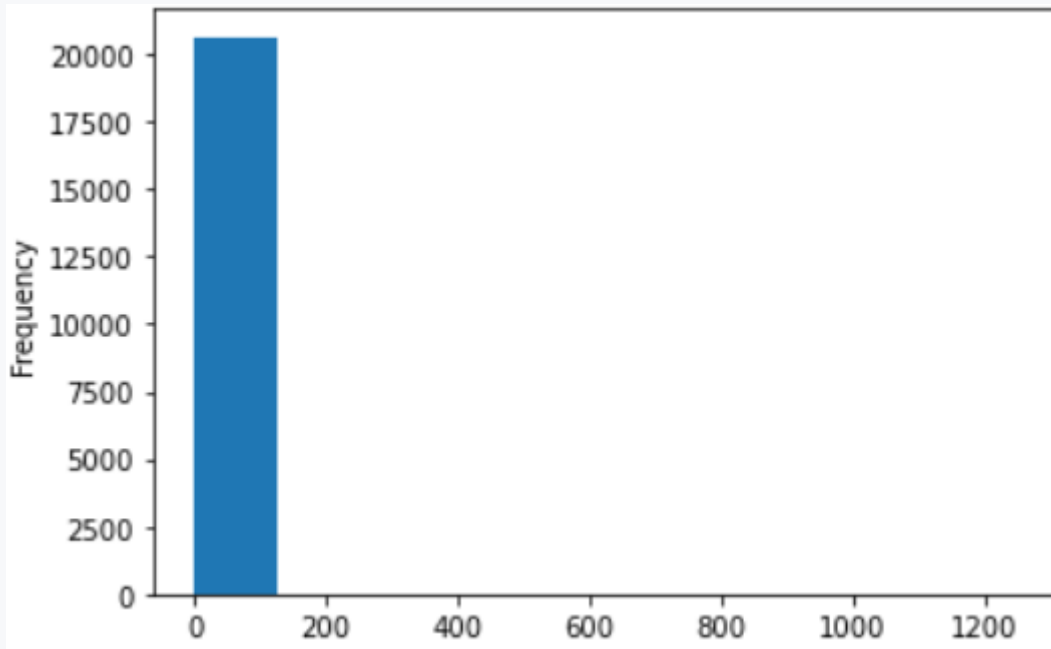
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   MedInc          20640 non-null   float64
1   HouseAge        20640 non-null   float64
2   AveRooms        20640 non-null   float64
3   AveBedrms       20640 non-null   float64
4   Population      20640 non-null   float64
5   AveOccup        20640 non-null   float64
6   Latitude         20640 non-null   float64
7   Longitude       20640 non-null   float64
dtypes: float64(8)
memory usage: 1.3 MB
None

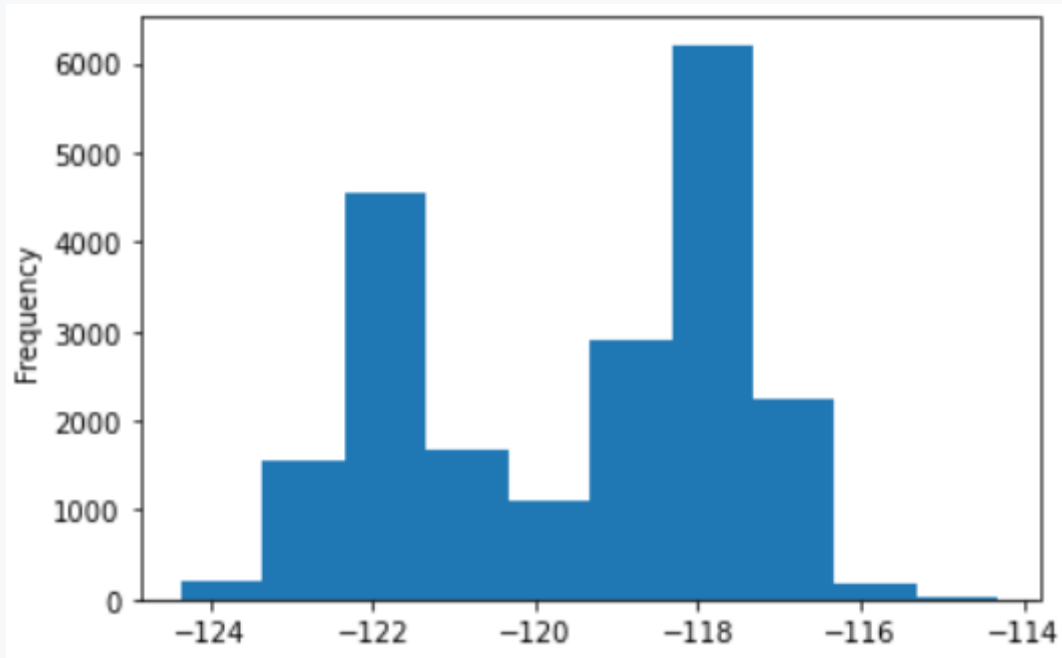
```











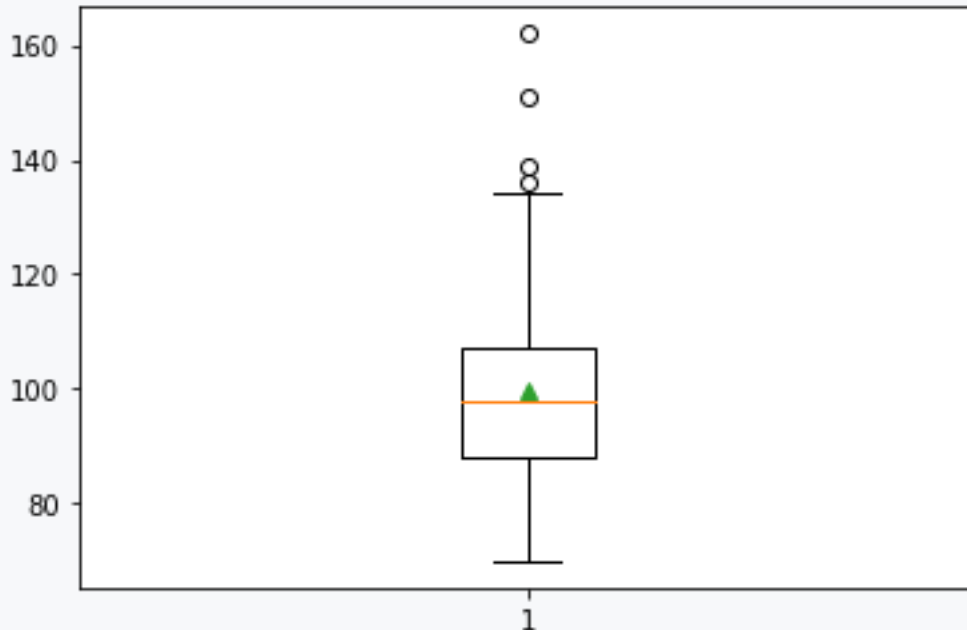
2.2.6

We covered descriptive statistics in the previous lesson. In addition to numerical characteristics, we can also visualize descriptive statistics using a **box plot**. Thus, we can assess and compare measures of the location and dispersion of values in their neighbourhood. Visualization of the histogram is possible using the **boxplot()** function, which is found in the **matplotlib** library. As the first parameter, we specify the variable we want to visualize. The *showmeans* parameter adds visual information about the mean value to our graph, which is represented by the green triangle. The red line tells us the mean value. The rectangle, in turn, gives us the upper-to-lower quartile boundary. The maximum and minimum are bounded by lines from the rectangle upwards and downwards.

```
import pandas as pd
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
plt.boxplot(df['magnesium'], showmeans=True)
```

Program output:



2.2.7

Using the *matplotlib* library, we can also visualize multiple box plots at the same time. As a first parameter, we send not a single variable but a list of variables to be examined. We can then color-code the variables using various settings, which you can see in the following code. In our case, we have combined variables whose range of values is approximately similar. However, it is more transparent to observe the individual variables separately so that we are not affected by the different scales of values.

```
import pandas as pd
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt

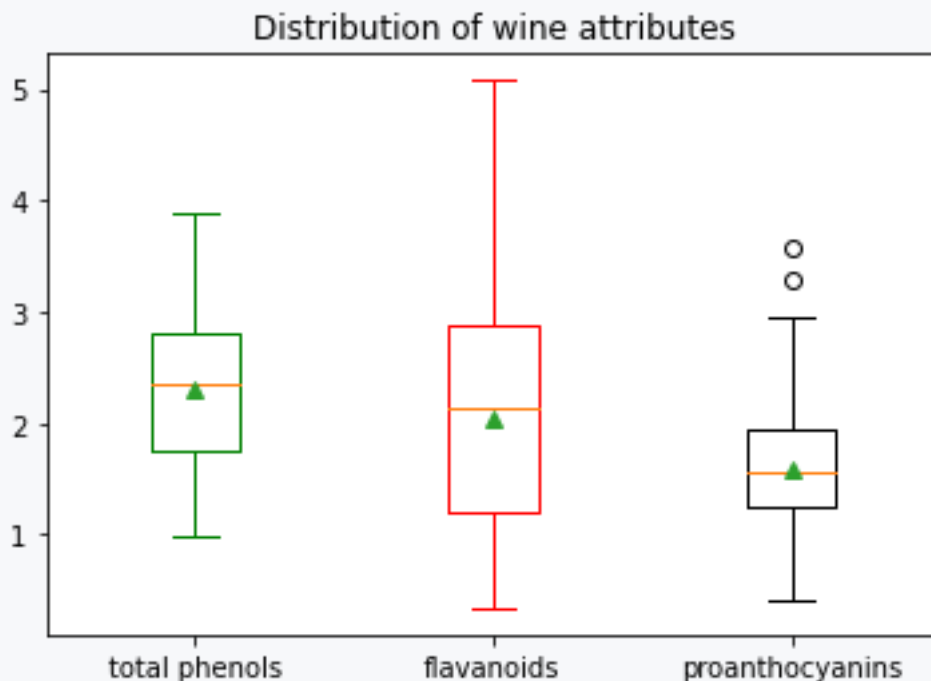
wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
box =
plt.boxplot([df['total_phenols'],df['flavanoids'],df['proantho
cyanins']], showmeans=True)
#boxes customization
plt.setp(box['boxes'][0], color='green')
plt.setp(box['caps'][0], color='green')
plt.setp(box['caps'][1], color='green')
plt.setp(box['whiskers'][0], color='green')
plt.setp(box['whiskers'][1], color='green')
```

```
plt.setp(box['boxes'][1], color='red')
plt.setp(box['caps'][2], color='red')
plt.setp(box['caps'][3], color='red')
plt.setp(box['whiskers'][2], color='red')
plt.setp(box['whiskers'][3], color='red')

plt.title('Distribution of wine attributes')
plt.xticks([1,2,3], ['total
phenols', 'flavanoids', 'proanthocyanins'])

plt.show()
```

Program output:



2.2.8

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function. Use the box plot to examine each attribute of the dataset and select the correct assertions.

We will add one more column to our data file, **target**. This column is used for the classification task where based on the other variables we can classify the median California home price value, expressed in hundreds of thousands of dollars. In our case, it will mainly serve us to better understand the data.

```

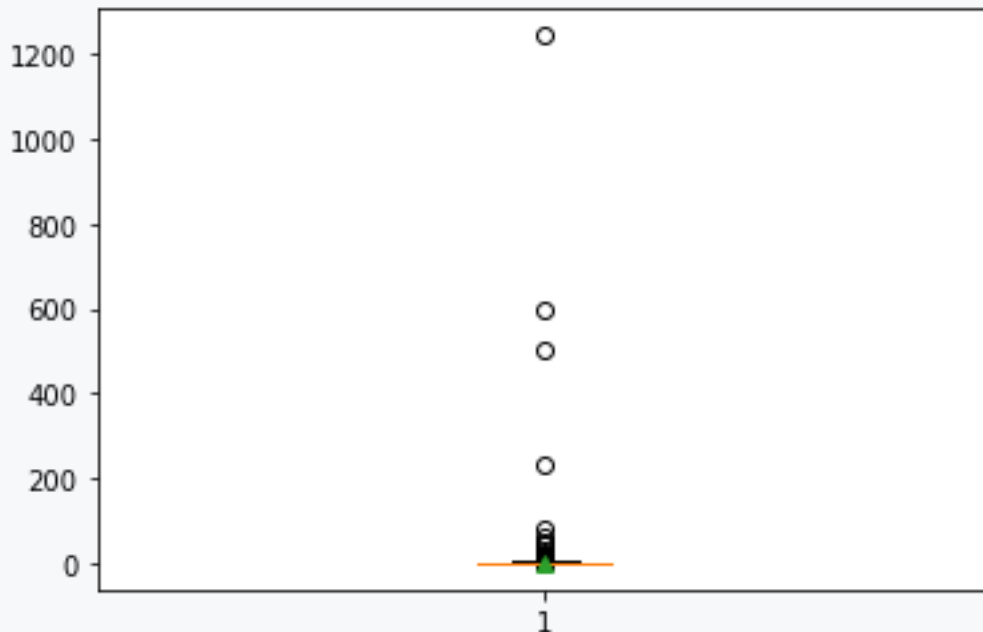
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
#print(df.info())
#plt.boxplot(df['HouseAge'], showmeans=True)
#plt.boxplot(df['AveRooms'], showmeans=True)
#plt.boxplot(df['AveBedrms'], showmeans=True)
plt.boxplot(df['AveOccup'], showmeans=True)
#plt.boxplot(df['Population'], showmeans=True)

```

Program output:



- the average age of the houses is close to the median age of the houses in the block
- descriptive statistics of the average number of rooms and bedrooms are similar
- the age of houses has a normal distribution
- the average age of the houses is similar to the average number of rooms

2.2.9

There is no standard that specifies which chart we should use to visualize the data. However, there are a few guidelines that can help us choose:

- It is important to understand what type of data we are examining. If you have continuous variables, then a histogram would be a good choice. Similarly, if we want to display a ranking, an ordered bar chart would be a good choice.
- Let's choose a graph that effectively conveys the correct and relevant meaning of the data without actually misrepresenting the facts.
- Simplicity is best. It is considered better to draw a simple graph that is easy to understand than to draw complex graphs that require several reports and texts to understand.
- Let's choose a diagram that does not overwhelm the audience with information. Our goal should be to illustrate abstract information clearly.

2.3 Data summarization

2.3.1

During data analysis, it is often necessary to group data based on certain criteria. The concepts of clustering occur in several parts of data analysis. The pandas library contains a **groupby()** function that groups our dataset into different classes over which we can perform aggregation. The groupby() function performs two basic functions: it divides the data into groups based on certain criteria and applies the function to each group separately. The result of groupby() is a structure that provides us with several aggregation functions such as **sum()**, **mean()**, **median()**, **min()**, **max()**, and so on.

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
df["target"] = wine.target
print(df.groupby('target').mean())
```

Program output:

	alcohol	malic_acid	ash	alcalinity_of_ash
0	13.744746	2.010678	2.455593	17.037288
1	12.278732	1.932676	2.244789	20.238028


```

2          13.153750      3.333750  2.437083          21.416667
99.312500

          total_phenols  flavanoids  nonflavanoid_phenols
proanthocyanins \
target
0          2.840169      2.982373          0.290000
1.899322
1          2.258873      2.080845          0.363662
1.630282
2          1.678750      0.781458          0.447500
1.153542

          color_intensity      hue
od280/od315_of_diluted_wines      proline
target
0          5.528305  1.062034
3.157797  1115.711864
1          3.086620  1.056282
2.785352  519.507042
2          7.396250  0.682708
1.683542  629.895833

```

2.3.2

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function.

We'll also add a *target* column to our dataset. This column is used for the classification task, where based on the other variables we can classify the median price value of California homes, expressed in hundreds of thousands of dollars. In our case, for the moment, it will mainly serve us to better understand the data.

Using clustering based on the target variable, find the median value of the age of homes in the block for a target value of 5. Round the result to a whole number.

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)

```

```
df['target'] = cali.target
print(df.groupby('target').mean())
```

Program output:

```

      AveOccup  \
target
0.14999  2.122475  30.750000  6.575951  2.016259  305.25000
2.566440
0.17500  2.366700  39.000000  3.572464  1.217391  259.00000
1.876812
0.22500  1.818075  36.250000  3.975628  1.265805  2112.00000
3.652335
0.25000  0.857100  21.000000  1.629630  1.222222   64.00000
2.370370
0.26600  2.301300  34.000000  4.897959  1.051020  808.00000
2.748299
...
...
4.98800  8.248000  29.000000  7.072727  0.978182  826.00000
3.003636
4.99000  8.148900  18.000000  6.600817  1.001362  1634.00000
2.226158
4.99100  6.786100  28.000000  7.386861  1.083942  617.00000
2.251825
5.00000  3.899581  38.000000  4.773400  1.094456  1036.00000
2.097639
5.00001  7.825123  33.802073  6.817436  1.097833  1112.80829
2.570442

      Latitude  Longitude
target
0.14999  37.665000 -120.197500
0.17500  34.150000 -118.330000
0.22500  36.005000 -119.335000
0.25000  32.790000 -114.650000
0.26600  35.130000 -119.450000
...
...
4.98800  37.330000 -122.060000
4.99000  37.890000 -122.180000
4.99100  33.550000 -117.770000
5.00000  35.584444 -120.155556
5.00001  35.225751 -119.702477

[3842 rows x 8 columns]
```

2.3.3

Aggregation is the process of performing any mathematical operation on a set of data or a subset of it. Aggregation is one of the many techniques in the pandas library that is used to manipulate data in data analysis.

The **aggregate()** function is used to apply aggregation to one or more columns. Some of the most commonly used aggregations are as follows:

- *sum*: returns the sum of the values
- *min*: returns the minimum of the values
- *max*: returns the maximum of the values

It is important to note that we can only perform aggregations over numeric values.

```
import pandas as pd
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
df["target"] = wine.target
print(df.aggregate('max'))
```

Program output:

```
alcohol          14.83
malic_acid       5.80
ash              3.23
alcalinity_of_ash 30.00
magnesium       162.00
total_phenols   3.88
flavanoids      5.08
nonflavanoid_phenols 0.66
proanthocyanins 3.58
color_intensity 13.00
hue              1.71
od280/od315_of_diluted_wines 4.00
proline         1680.00
target           2.00
dtype: float64
```

2.3.4

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function.

Use aggregation to find the lowest value in the `MedInc` column. Round the result to two decimal places.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
print(round(df.aggregate(min), 2))
```

Program output:

```
MedInc          0.50
HouseAge        1.00
AveRooms        0.85
AveBedrms       0.33
Population      3.00
AveOccup        0.69
Latitude        32.54
Longitude       -124.35
dtype: float64
```

2.3.5

The most important operations implemented by `groupby()` are aggregation, filter, transform, and apply. An efficient way to implement aggregation functions in a data file is to execute them after grouping the required columns. The aggregation function returns one aggregated value for each group. After creating these groups, we can apply several aggregation operations to the data grouped in this way.

The advantage of aggregation is that we can also work with functions from other libraries, such as `numpy`, in the call to get the value of standard deviation and so on. The following notation will allow us to create different views of the variables we are examining, with the addition that we can also create their naming and thus make the table in question clearer.

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
df["target"] = wine.target

df_group = df.groupby('target').aggregate(
    mean_alcohol=('alcohol', np.mean),
    max_ash=('ash', np.max),
    std_magnesium=('magnesium', np.std)
)
print(df_group)
```

Program output:

	mean_alcohol	max_ash	std_magnesium
target			
0	13.744746	3.22	10.498949
1	12.278732	3.23	16.753497
2	13.153750	2.86	10.890473

2.3.6

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function.

Combine different aggregation methods for different variables. Aggregate the data based on the variable `target`. Then output a value of 5 for the `target`:

- the minimum of the `AveRooms` variable
- the median of the variable `AveOccup`
- the maximum of the variable `AveBedrms`

Round the result to two decimal places and output in the following format:

```
AveRooms: 3.52 AveOccup: 2.98 AveBedrms: 1.25
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()
```

```
df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
df['target']=cali.target
df_group = df.groupby('target').aggregate(
    min_rooms=('AveRooms', np.min),
    med_occup=('AveOccup', np.median),
    max_bedrms=('AveBedrms', np.max)
).round(2)
print(df_group)
```

Program output:

	min_rooms	med_occup	max_bedrms
target			
0.14999	3.57	2.52	3.50
0.17500	3.57	1.88	1.22
0.22500	2.02	3.35	1.49
0.25000	1.63	2.37	1.22
0.26600	4.90	2.75	1.05
...
4.98800	7.07	3.00	0.98
4.99000	6.60	2.23	1.00
4.99100	7.39	2.25	1.08
5.00000	2.83	1.90	1.36
5.00001	1.82	2.52	25.64

[3842 rows x 3 columns]

 2.3.7

An essential part of data summarization is the use of a **contingency table**. A contingency table is a table that is used to clearly summarize the relationship between two (or more) variables. The rows of the contingency table correspond to the possible values of the first variable, and the columns to the possible values of the second. The corresponding cell of the contingency table usually contains the number of cases where at the same time the first variable had a value corresponding to the corresponding row and the second variable had a value corresponding to the corresponding column. The *pandas* library provides two options for creating a contingency table, the **pivot_table()** and **crosstab()** functions. Since both functions generate the same output but the **pivot_table()** function offers more options, we will only work with it. Using the *aggfunc* parameter, we can again specify the aggregation function. If we don't specify this parameter, the contingency table generates average values by default. The parameter *margins=True* allows us to turn on aggregation for all rows in the table.

```

import pandas as pd
import numpy as np
from sklearn.datasets import load_wine

wine = load_wine()
df = pd.DataFrame(data=wine.data, columns= wine.feature_names)
df["target"] = wine.target

table = pd.pivot_table(df, index=["target"], aggfunc=np.mean,
margins=True)

print(table)

```

Program output:

```

          alcalinity_of_ash    alcohol    ash
color_intensity  flavanoids  \
target
0                17.037288   13.744746   2.455593
5.528305         2.982373
1                20.238028   12.278732   2.244789
3.086620         2.080845
2                21.416667   13.153750   2.437083
7.396250         0.781458
All              19.494944   13.000618   2.366517
5.058090         2.029270

          hue    magnesium    malic_acid    nonflavanoid_phenols
\
target
0          1.062034   106.338983     2.010678             0.290000
1          1.056282    94.549296     1.932676             0.363662
2          0.682708    99.312500     3.333750             0.447500
All        0.957449    99.741573     2.336348             0.361854

          od280/od315_of_diluted_wines    proanthocyanins
proline  \
target
0                3.157797             1.899322
1115.711864
1                2.785352             1.630282
519.507042
2                1.683542             1.153542
629.895833
All              2.611685             1.590899
746.893258

```

```

                total_phenols
target
0                2.840169
1                2.258873
2                1.678750
All              2.295112

```

2.3.8

Load from the sklearn library the dataset `california_housing`, which contains records of homes in California. You fetch the dataset into an object using the `fetch_california_housing()` function.

Group the data based on the `target` variable. Use the contingency table to find the standard deviation value for the entire table for the Population column. Round the result to two decimal places.

```

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing()

df = pd.DataFrame(data=cali.data, columns=cali.feature_names)
df['target']=cali.target

table = pd.pivot_table(df, index=["target"], aggfunc=np.std,
margins=True).round(2)
print(table)

```

Program output:

```

                AveBedrms  AveOccup  AveRooms  HouseAge  Latitude
Longitude  MedInc  \
target
0.14999          1.03    0.38    4.05    16.68    2.86
3.21    1.53
0.225           0.15    0.74    2.31    20.85    2.50
2.89    1.02
0.3            1.01    0.68    2.27    15.56    1.22
2.63    1.01

```



```
0.325      0.67      0.32      3.49      16.58     2.71
2.98      1.13
0.375      0.47      3.22      1.56      13.95     2.73
1.79      0.56
...
...
4.956      0.01      0.21      1.43      1.41      0.01
0.06      3.13
4.964      0.11      0.13      0.64      8.49      3.20
3.34      0.93
5.0        0.09      0.58      1.54      12.73     1.98
2.21      1.31
5.00001    0.80      1.49      4.67      13.03     1.78
1.95      3.25
All        0.47      10.39     2.47      12.59     2.14
2.00      1.90

      Population
target
0.14999    299.62
0.225      3186.56
0.3         114.55
0.325      415.47
0.375      2745.95
...
4.956      272.94
4.964      160.51
5.0        671.25
5.00001    813.32
All        1132.43

[3117 rows x 8 columns]
```

Data Analysis

Chapter **3**

3.1 Univariate analysis

3.1.1

Each data set we want to analyze will have different fields (i.e., columns) of multiple observations (i.e., variables) that represent different facts. The columns of the dataset are most likely related to each other because they are collected from the same event. One field of a record may or may not affect the value of another field. To examine the type of relationships that these columns have, and to analyze the cause and effect between them, we need to work our way to identifying the dependencies that exist between the variables. The strength of such a relationship between two fields of a data set is called correlation, which is represented by a numerical value between -1 and 1.

For example, height and weight are correlated, so it can be assumed that taller people are usually heavier than shorter ones. If we have a new person who is taller than the average height we observed before, then they are more likely to weigh more than the average weight we observed.

Correlation tells us how variables change together, in the same or opposite direction, and in the strength of the relationship. We calculate the Pearson correlation coefficient to find the correlation. If the correlation is +1, then it can be said to be a perfect positive/linear correlation (variable A is directly proportional to variable B), while a correlation of -1 is a perfect negative correlation (variable A is inversely proportional to variable B). Values closer to 0 are not correlated. If the correlation coefficients are close to 1 in absolute value, the variables are said to have a strong correlation; in comparison, those close to 0.5 have a weak correlation.

3.1.2

In the previous chapter, we focused on descriptive statistics. We had a variable that contained numerical values and we calculated the mean, median, and mode and analyzed the distribution of the values. We then grouped the data based on the *target* variable and then calculated the mean, median, modus, and standard deviation for each option. Analysis of one type of data is called **univariate analysis**.

Univariate analysis is the simplest form of data analysis. It means that our data has only one type of variable and that we perform the analysis over it. The main goal of the univariate analysis is to take the data, summarize it, and find patterns among the values. It does not deal with causes or relationships between values. A few techniques that describe ways found in univariate data include central tendency (i.e., mean, mode, and median) and dispersion (i.e., range, variance, maximum and minimum quartiles (including interquartile range), and standard deviation).

Let us recap the whole process over the new data matrix. The data matrix contains information on the sales of games in recent years. Using the **info()** function, we can find out what variables are in the dataset and possibly how much missing data each

variable contains. Then, using the **describe()** function we can find the mean, median, maximum, minimum and standard deviation.

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')

print(df.info())

print(df.describe())
```

Program output:

```
RangeIndex: 55792 entries, 0 to 55791
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Rank                   55792 non-null  int64
1   Name                   55792 non-null  object
2   Genre                  55792 non-null  object
3   ESRB_Rating            23623 non-null  object
4   Platform               55792 non-null  object
5   Publisher              55792 non-null  object
6   Developer              55775 non-null  object
7   Critic_Score           6536 non-null   float64
8   User_Score             335 non-null    float64
9   Total_Shipped          1827 non-null   float64
10  Global_Sales           19415 non-null  float64
11  NA_Sales               12964 non-null  float64
12  PAL_Sales              13189 non-null  float64
13  JP_Sales               7043 non-null   float64
14  Other_Sales            15522 non-null  float64
15  Year                   54813 non-null  float64
dtypes: float64(9), int64(1), object(6)
memory usage: 6.8+ MB
None
```

	Rank	Critic_Score	User_Score	Total_Shipped
Global_Sales \				
count	55792.000000	6536.000000	335.000000	1827.000000
19415.000000				
mean	27896.500000	7.213709	8.253433	1.887258
0.365503				

std	16105.907446	1.454079	1.401489	4.195693
0.833022				
min	1.000000	1.000000	2.000000	0.030000
0.000000				
25%	13948.750000	6.400000	7.800000	0.200000
0.030000				
50%	27896.500000	7.500000	8.500000	0.590000
0.120000				
75%	41844.250000	8.300000	9.100000	1.800000
0.360000				
max	55792.000000	10.000000	10.000000	82.860000
20.320000				
	NA_Sales	PAL_Sales	JP_Sales	Other_Sales
Year				
count	12964.000000	13189.000000	7043.000000	15522.000000
54813.000000				
mean	0.275541	0.155263	0.110402	0.044719
2005.659095				
std	0.512809	0.399257	0.184673	0.129554
8.355585				
min	0.000000	0.000000	0.000000	0.000000
1970.000000				
25%	0.050000	0.010000	0.020000	0.000000
2000.000000				
50%	0.120000	0.040000	0.050000	0.010000
2008.000000				
75%	0.290000	0.140000	0.120000	0.040000
2011.000000				
max	9.760000	9.850000	2.690000	3.120000
2020.000000				

3.1.3

Read data from the banking.csv file, which contains information about the bank's customers. There are several variables in the file, which can be clearly divided into 3 categories:

Customer demographic information:

- customer_id - customer identifier
- vintage - how long the customer has been with the bank in the number of days
- age - age of the customer

- gender - gender of the customer
- occupation - occupation of the customer
- city - city of the customer (anonymised)

Information related to the bank for customers:

- customer_nw_category - customer value (3:low 2:medium 1:high)
- branch_code - branch code for the customer's account
- days_since_last_transaction - number of days since the last payment in the last 1 year

Transaction information:

- current_balance - balance as of the current day
- previous_month_end_balance - month-end balance in the previous month
- average_monthly_balance_prevQ - average monthly balances in the previous quarter
- average_monthly_balance_prevQ2 - average monthly balances two quarters back
- percent_change_credits - percentage change in credits between the last two quarters
- current_month_credit - the total amount of credits in the current month
- previous_month_credit - the total amount of credit in the previous month
- current_month_debit - the total amount of debt in the current month
- previous_month_debit - the total amount of debt in the previous month
- current_month_balance - average balance in the current month
- previous_month_balance - average balance in the previous month
- churn - client at risk - client's average balance falls below the minimum balance in the following quarter (1/0)

After loading the data file, examine the variables and print the average value of the current balance across all accounts (current_balance).

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')
pd.set_option('display.float_format', lambda x: f'{x:.3f}')
print(df.info())

print(df.describe())
```

Program output:

```
RangeIndex: 28382 entries, 0 to 28381
```

```
Data columns (total 21 columns):
```

#	Column	Non-Null Count	Dtype
0	customer_id	28382 non-null	int64
1	vintage	28382 non-null	int64
2	age	28382 non-null	int64
3	gender	27857 non-null	object
4	dependents	25919 non-null	float64
5	occupation	28302 non-null	object
6	city	27579 non-null	float64
7	customer_nw_category	28382 non-null	int64
8	branch_code	28382 non-null	int64
9	current_balance	28382 non-null	float64
10	previous_month_end_balance	28382 non-null	float64
11	average_monthly_balance_prevQ	28382 non-null	float64
12	average_monthly_balance_prevQ2	28382 non-null	float64
13	current_month_credit	28382 non-null	float64
14	previous_month_credit	28382 non-null	float64
15	current_month_debit	28382 non-null	float64
16	previous_month_debit	28382 non-null	float64
17	current_month_balance	28382 non-null	float64
18	previous_month_balance	28382 non-null	float64
19	churn	28382 non-null	int64
20	last_transaction	28382 non-null	object

```
dtypes: float64(12), int64(6), object(3)
```

```
memory usage: 4.5+ MB
```

```
None
```

	customer_id	vintage	age	dependents	city
count	28382.000	28382.000	28382.000	25919.000	27579.000
mean	15143.509	2091.144	48.208	0.347	796.110
std	8746.454	272.677	17.807	0.998	432.872
min	1.000	73.000	1.000	0.000	0.000
25%	7557.250	1958.000	36.000	0.000	409.000
50%	15150.500	2154.000	46.000	0.000	834.000
75%	22706.750	2292.000	60.000	0.000	1096.000
max	30301.000	2476.000	90.000	52.000	1649.000

	customer_nw_category	branch_code	current_balance
count	28382.000	28382.000	28382.000
mean	2.226	925.975	7380.552

std	0.660	937.799	42598.712
min	1.000	1.000	-5503.960
25%	2.000	176.000	1784.470
50%	2.000	572.000	3281.255
75%	3.000	1440.000	6635.820
max	3.000	4782.000	5905904.030

previous_month_end_balance	
average_monthly_balance_prevQ \	
count	28382.000
28382.000	
mean	7495.771
7496.780	
std	42529.345
41726.219	
min	-3149.570
1428.690	
25%	1906.000
2180.945	
50%	3379.915
3542.865	
75%	6656.535
6666.887	
max	5740438.630
5700289.570	

average_monthly_balance_prevQ2		current_month_credit \
count	28382.000	28382.000
mean	7124.209	3433.252
std	44575.810	77071.452
min	-16506.100	0.010
25%	1832.507	0.310
50%	3359.600	0.610
75%	6517.960	707.272
max	5010170.100	12269845.390

previous_month_credit		current_month_debit
previous_month_debit \		
count	28382.000	28382.000
28382.000		
mean	3261.694	3658.745
3339.761		
std	29688.889	51985.424
24301.112		

min	0.010	0.010	
0.010			
25%	0.330	0.410	
0.410			
50%	0.630	91.930	
109.960			
75%	749.235	1360.435	
1357.553			
max	2361808.290	7637857.360	
1414168.060			
	current_month_balance	previous_month_balance	churn
count	28382.000	28382.000	28382.000
mean	7451.133	7495.177	0.185
std	42033.939	42431.979	0.389
min	-3374.180	-5171.920	0.000
25%	1996.765	2074.407	0.000
50%	3447.995	3465.235	0.000
75%	6667.958	6654.693	0.000
max	5778184.770	5720144.500	1.000

3.1.4

The next step is to use visualization to examine the distribution of the selected variables. Let's look at the distribution of the **Year** variable that we can examine using a histogram. Before we visualize the histogram, we can see how many years are actually in our dataset. We can get the number of unique years by using the **unique()** function, which returns the unique elements of the variable under study. We can then use this value to partition the histogram into exactly a unique number of years, giving us an accurate representation of the counts for those years. From the graph, we can observe that from around 2008 onwards, the production of games started to decline.

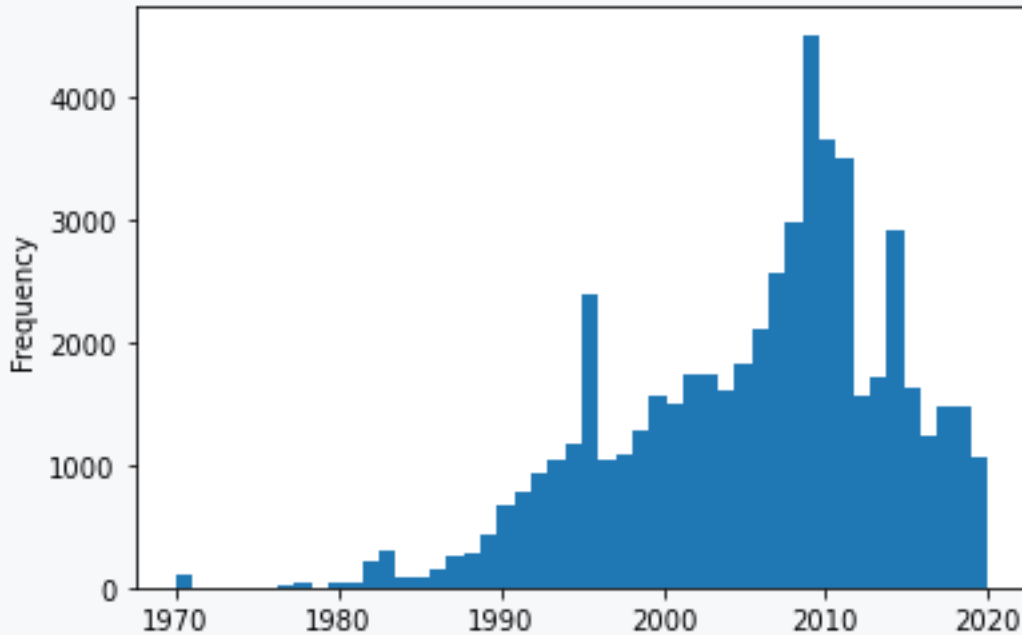
```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')

y_bins = len(df['Year'].unique())

df['Year'].plot(kind='hist', bins=y_bins)
```

Program output:



3.1.5

Next, we can take a look at the ratings of games by critics and users. On closer inspection of the records, we find that the **User_Score** variable contains a significant number of missing values. While we are left with few records after removing them we can observe through visualization that users tend to rate games more positively, as a higher value means a better score. This can also be seen by comparing the average values, which have a difference of about 1 point.

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
print(df['Critic_Score'].describe())

df['Critic_Score'].dropna().plot(kind='hist')

print(df['User_Score'].describe())

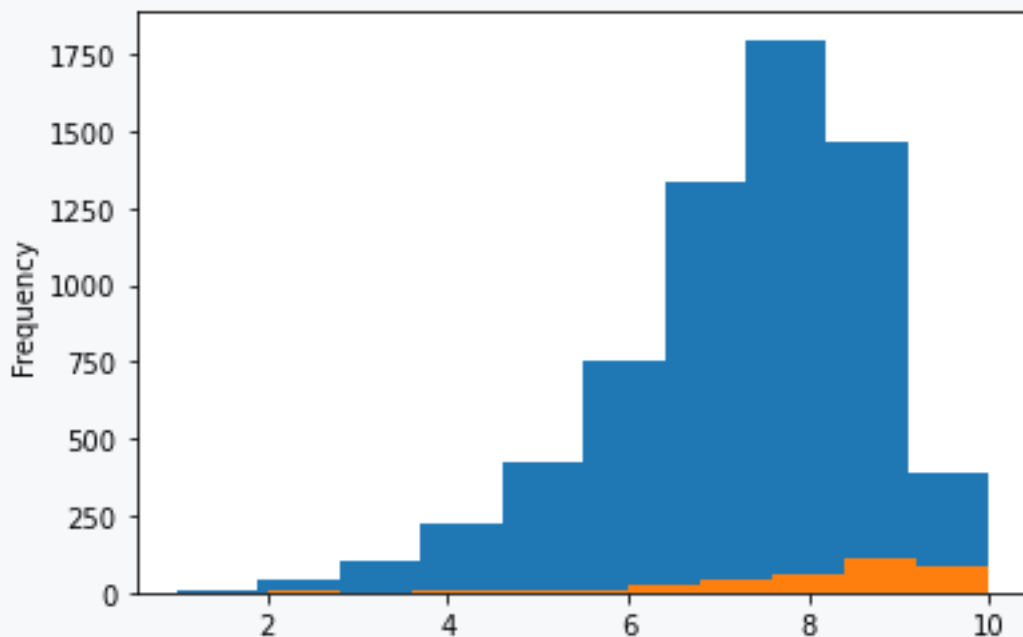
df['User_Score'].dropna().plot(kind='hist')
```

Program output:

```

count      6536.000000
mean       7.213709
std        1.454079
min        1.000000
25%        6.400000
50%        7.500000
75%        8.300000
max        10.000000
Name: Critic_Score, dtype: float64
count      335.000000
mean       8.253433
std        1.401489
min        2.000000
25%        7.800000
50%        8.500000
75%        9.100000
max        10.000000
Name: User_Score, dtype: float64

```



3.1.6

The next step is to examine the categorical variables. We start by looking at which platform most games have been produced for. However, since the frequency graph is rather opaque, we will only select the top 30 most numerous platforms. The **describe()** function doesn't give us information about the basic statistics in the case

of a categorical variable but we can find out the number of elements, the number of categories, and the most numerous category in this way.

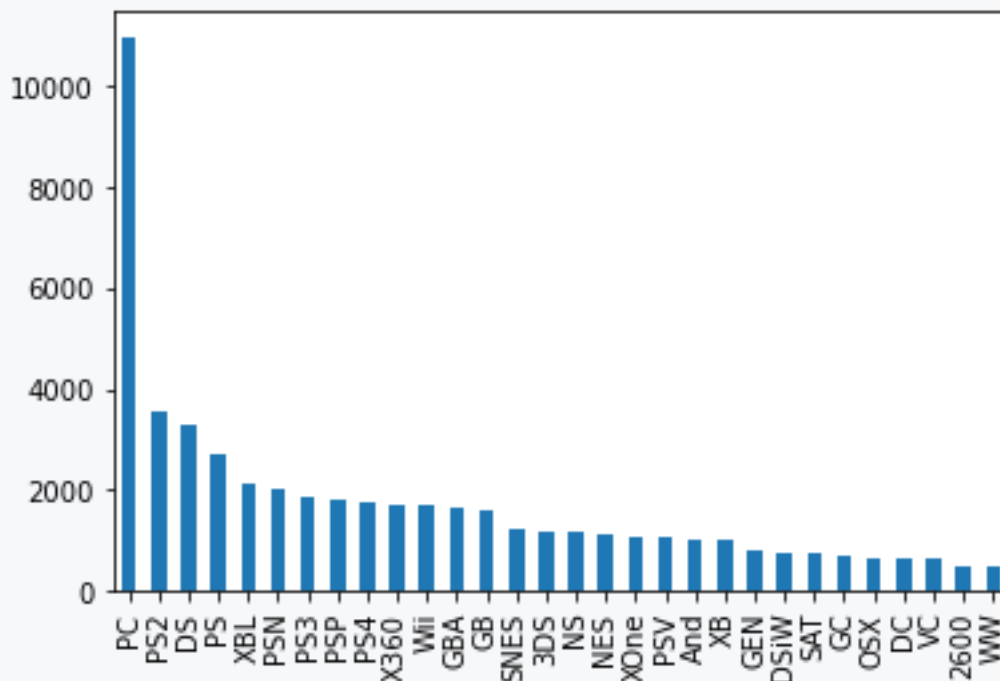
```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
print(df['Platform'].describe())

df['Platform'].dropna().value_counts().iloc[:30].plot(kind='bar')
```

Program output:

```
count      55792
unique       74
top         PC
freq       10978
Name: Platform, dtype: object
```



3.1.7

The genre of games gave us interesting results, where the most numerous games were from the miscellaneous genre, which can probably mean an increase in Indie games. The second most numerous games were action games, followed by

adventure and sports games. On the other hand, strategy games were not as abundant despite often being a popular game type.

We can follow a similar approach when examining other categorical variables such as publisher (*Developer*).

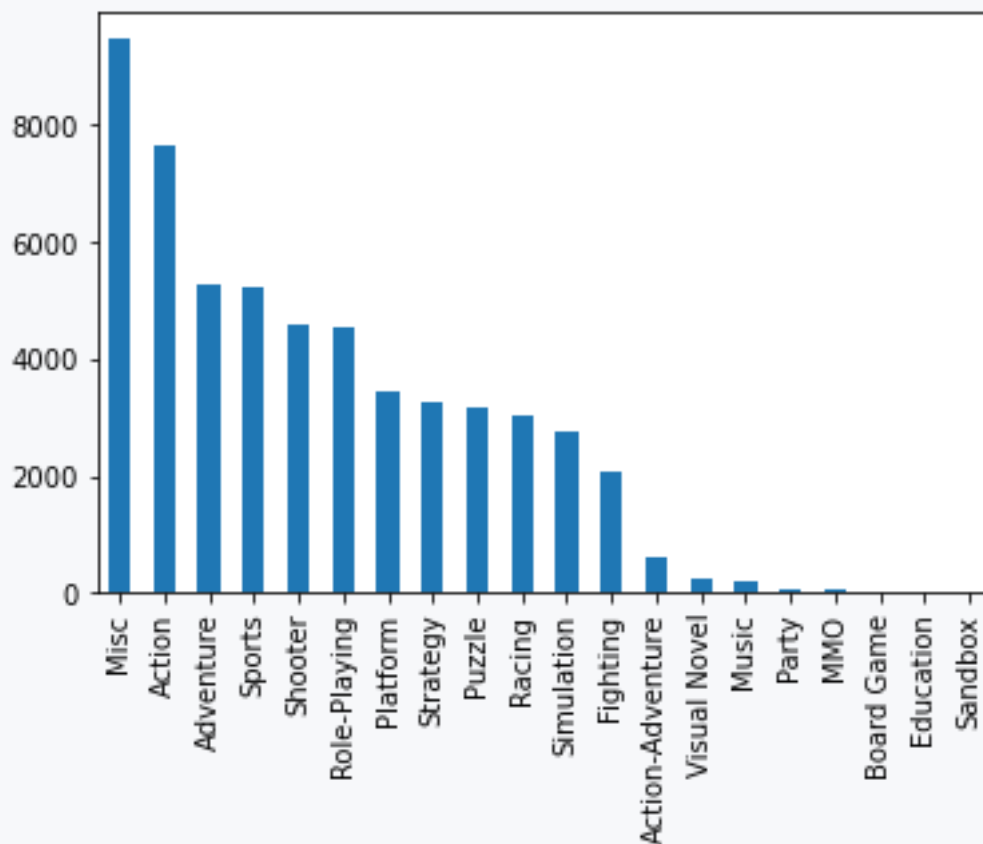
```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
print(df['Genre'].describe())

df['Genre'].dropna().value_counts().plot(kind='bar')
```

Program output:

```
count      55792
unique         20
top         Misc
freq        9476
Name: Genre, dtype: object
```



3.1.8

Load the data from the `banking.csv` file, which contains information about the bank's customers. After loading the data file, find out what is the ratio of males and females among the bank's customers (*gender*). We recommend using the visualization and writing out both genders and the percentages rounded to two decimal places in the result.

```
male: 54.25% female: 45.75%
```

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.c
sv', sep=',', decimal='.')

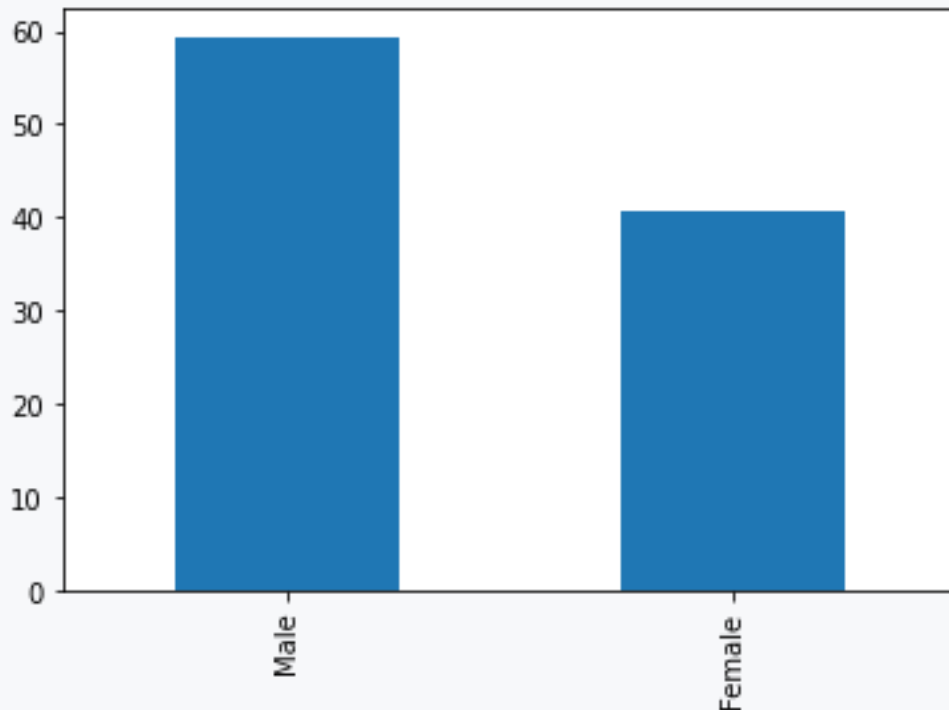
df['gender'] = df['gender'].astype('category') # set
occupation as categorical variable

df['gender'].value_counts(normalize=True).mul(100).plot(kind='
bar')

print(df['gender'].value_counts(normalize=True).mul(100).round
(2))
```

Program output:

```
Male      59.4
Female    40.6
Name: gender, dtype: float64
```



3.1.9

Load data from the `banking.csv` file, which contains information about the bank's customers. After loading the data file find out what is the most common occupation of the bank's customers (*occupation*). We recommend using the visualization and printing the occupation and the percentage rounded to two decimal places in the result.

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.c
sv', sep=',', decimal='.')

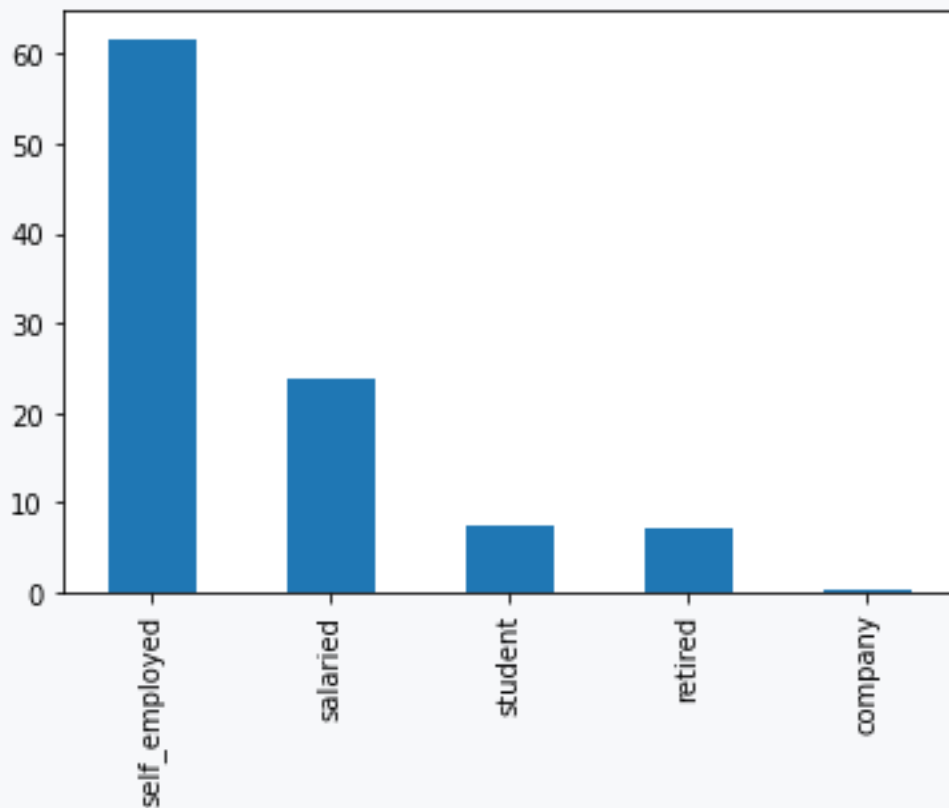
df['occupation'] = df['occupation'].astype('category') # set
occupation as categorical variable

df['occupation'].value_counts(normalize=True).mul(100).plot(ki
nd='bar')

print(df['occupation'].value_counts(normalize=True).mul(100).r
ound(2))
```

Program output:

```
self_employed    61.750
salaried         23.690
student          7.270
retired          7.150
company          0.140
Name: occupation, dtype: float64
```

 **3.1.10**

Load the data from the `banking.csv` file, which contains information about the bank's customers. After loading the data file find out what is the most common rating of the bank's customers (`customer_nw_category`). We recommend using the visualization and writing out the rating number and percentage rounded to two decimal places in the result.

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')
```



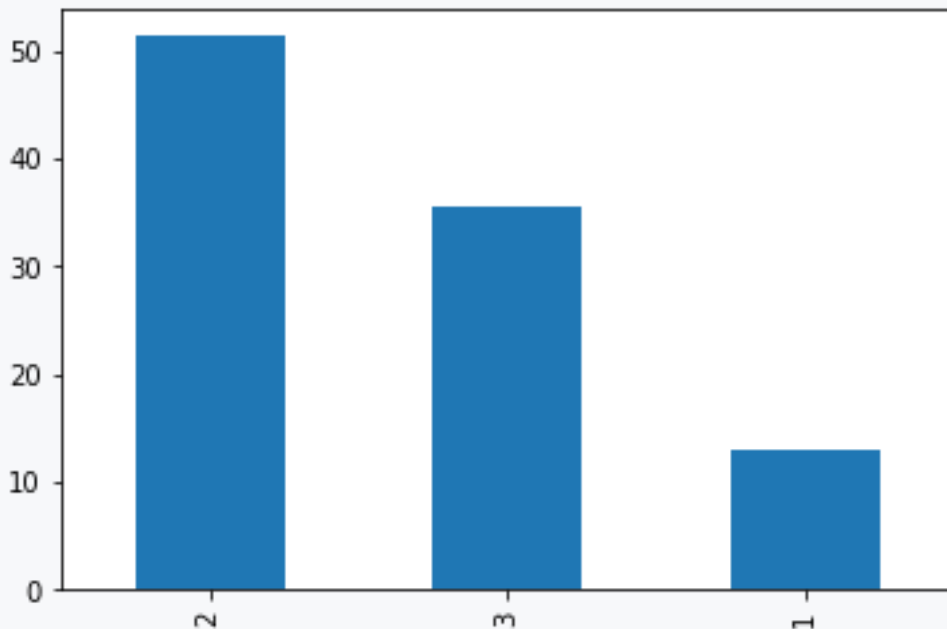
```
df['customer_nw_category'] =
df['customer_nw_category'].astype('category') # set occupation
as categorical variable

df['customer_nw_category'].value_counts(normalize=True).mul(100)
.plot(kind='bar')

print(df['customer_nw_category'].value_counts(normalize=True)
.mul(100).round(2))
```

Program output:

```
2    51.30
3    35.63
1    13.08
Name: customer_nw_category, dtype: float64
```



3.2 Bivariate analysis

3.2.1

This is an analysis of more than one (exactly two) type of variables. Bivariate analysis is used to see if there is a relationship between two different variables. When we create a scatter plot by plotting one variable against the other in the Cartesian plane (think of the x and y axes), we get a picture of what the data is trying to tell us. If the data points appear to correspond to a straight line or curve, then there is a relationship or correlation between the two variables. In general, bivariate analysis helps us predict the value of one variable (i.e., the dependent variable) if we know the value of the independent variable.

Let's look at our dataset of games. Using a scatter plot we can compare and see if critics' ratings have an impact on the worldwide sales of the games in question. From the graph, we can observe that sales increase as critics' ratings increase, so we can assume that ratings have an effect on the marketability of games. We can use either the `plot()` function of the pandas library. Or we can use the more advanced seaborn library, which offers a much larger number of functions when creating plots. The `lmpplot()` function adds a regression line to the scatter plot, which tells us whether two variables are dependent on each other. If the values are close to the line, then we can say that there is a dependency between the two variables.

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
print(df.info())
#df.plot(x='Critic_Score',y='Global_Sales',kind='scatter') #
using pandas
sns.lmpplot(x='Critic_Score',y='Global_Sales',data=df) # using
seaborn with line
```

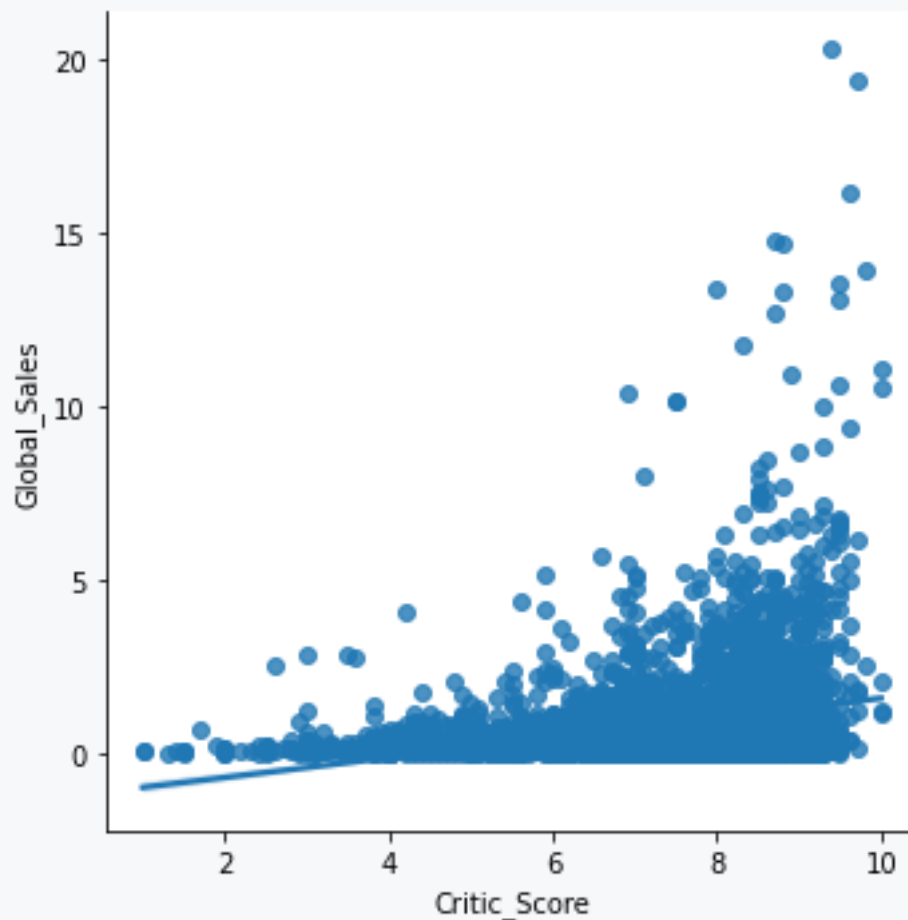
Program output:

```
RangeIndex: 55792 entries, 0 to 55791
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Rank                   55792 non-null  int64
1   Name                   55792 non-null  object
2   Genre                  55792 non-null  object
3   ESRB_Rating           23623 non-null  object
4   Platform               55792 non-null  object
5   Publisher              55792 non-null  object
6   Developer              55775 non-null  object
7   Critic_Score          6536 non-null   float64
8   User_Score            335 non-null    float64
9   Total_Shipped         1827 non-null   float64
10  Global_Sales           19415 non-null  float64
11  NA_Sales               12964 non-null  float64
12  PAL_Sales              13189 non-null  float64
13  JP_Sales               7043 non-null   float64
14  Other_Sales            15522 non-null  float64
```

```

15 Year          54813 non-null float64
dtypes: float64(9), int64(1), object(6)
memory usage: 6.8+ MB
None

```



3.2.2

Another way to find out the dependency between two variables is to use **boxplot()**. Again, we have the option to use both the *pandas* and *seaborn* libraries and the notation is similar. This time we look at the effect of game genre on the marketability of games. Since worldwide sales contain too much data, let's focus on just one market, e.g. Japan. As we can see from the graph, the number of genres can overwhelm the x-axis, so we need to rotate the labels 90 degrees to increase the clarity of the graph.

We can observe that the yield from the Role-playing and Sports genres is higher than that from the Racing and Shooter genres. Most genres contain outliers that represent high returns.

```

import pandas as pd
import seaborn as sns

```

```

from matplotlib import pyplot as plt

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
print(df.info())
#df.boxplot(by='Genre', column='JP_Sales')
gr = sns.boxplot(x='Genre', y='JP_Sales', data=df)
gr.set_xticklabels(gr.get_xticklabels(), rotation=90)

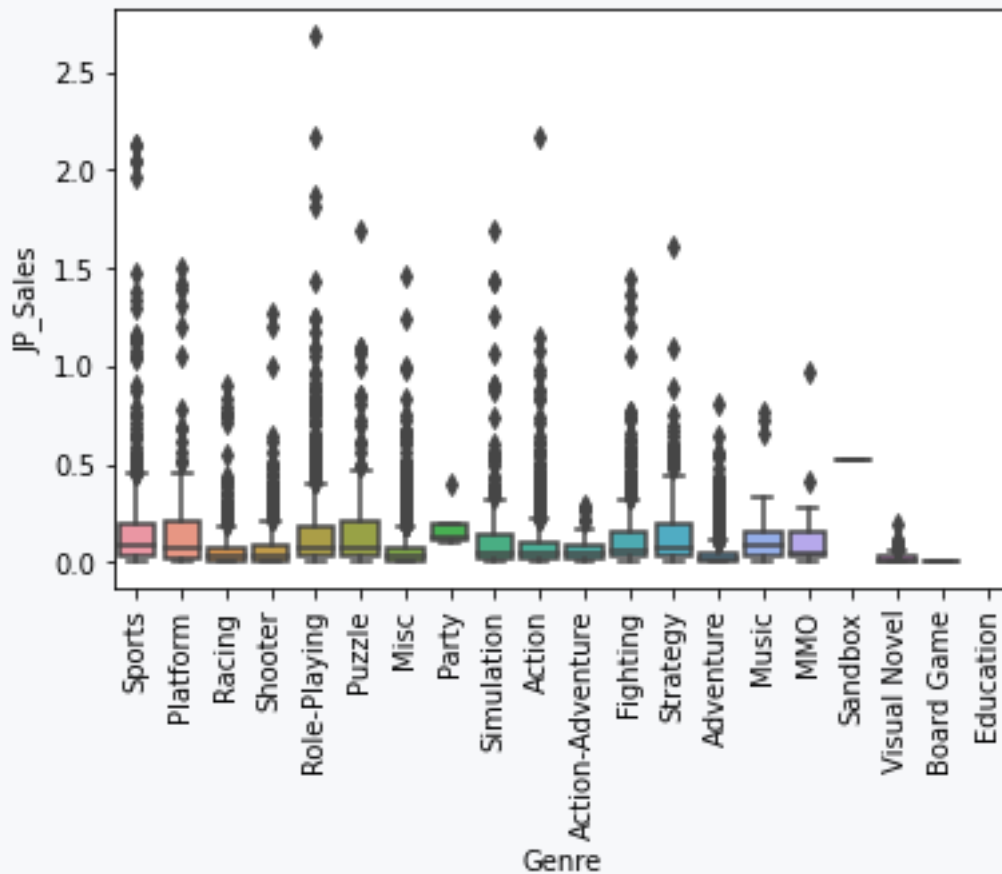
```

Program output:

```

RangeIndex: 55792 entries, 0 to 55791
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Rank                  55792 non-null  int64
1   Name                  55792 non-null  object
2   Genre                 55792 non-null  object
3   ESRB_Rating          23623 non-null  object
4   Platform              55792 non-null  object
5   Publisher             55792 non-null  object
6   Developer             55775 non-null  object
7   Critic_Score         6536 non-null   float64
8   User_Score           335 non-null    float64
9   Total_Shipped        1827 non-null   float64
10  Global_Sales          19415 non-null  float64
11  NA_Sales              12964 non-null  float64
12  PAL_Sales             13189 non-null  float64
13  JP_Sales              7043 non-null   float64
14  Other_Sales          15522 non-null  float64
15  Year                  54813 non-null  float64
dtypes: float64(9), int64(1), object(6)
memory usage: 6.8+ MB
None

```



3.2.3

In the next section, we can look at the impact of the game platform on marketability. However, we have too many platforms in the dataset to make sense of the visualization. Therefore, we will only choose the TOP10 most numerous platforms and visualize only their profit using **boxplot()**.

A surprising result from the graph is that the revenue of the most used platform (PC) is lower than for example the different PlayStation types.

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

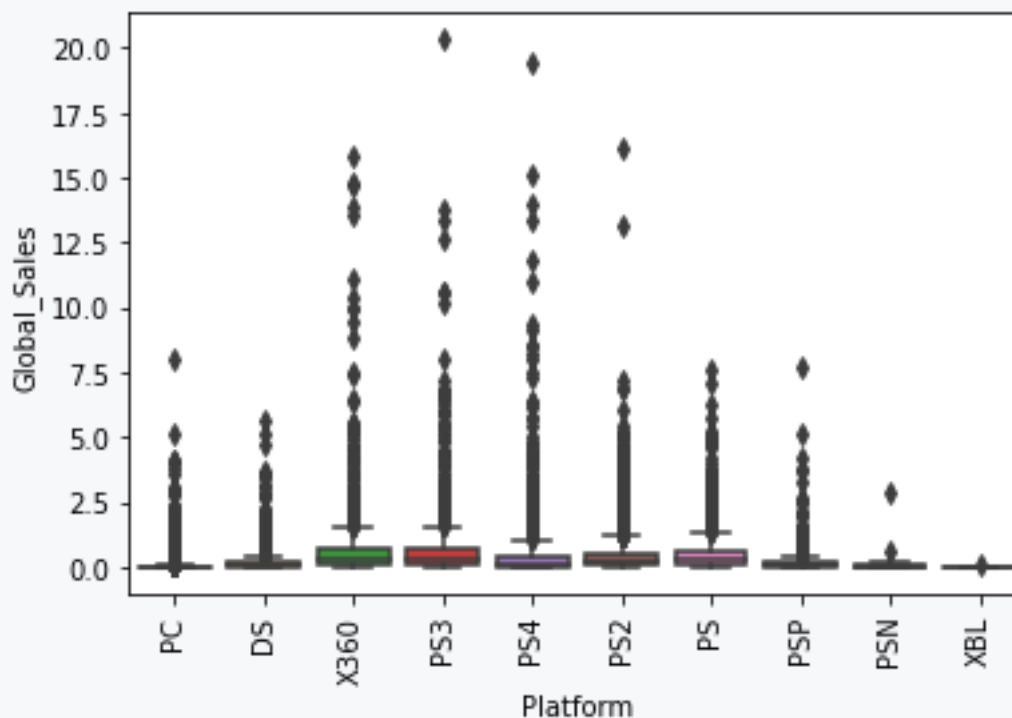
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
#print(df.info())
print(df['Platform'].dropna().value_counts().iloc[:10])
platforms =
['PC', 'PS2', 'DS', 'PS', 'XBL', 'PSN', 'PS3', 'PSP', 'PS4', 'X360']
```

```
df_plat = df[df['Platform'].isin(platforms)]
#df_plat.boxplot(by='Genre',column='Global_Sales')
gr = sns.boxplot(x='Platform',y='Global_Sales',data=df_plat)
gr.set_xticklabels(gr.get_xticklabels(), rotation=90)
```

Program output:

```
PC      10978
PS2     3564
DS      3292
PS      2703
XBL     2115
PSN     2004
PS3     1870
PSP     1804
PS4     1755
X360    1701
```

```
Name: Platform, dtype: int64
```



3.2.4

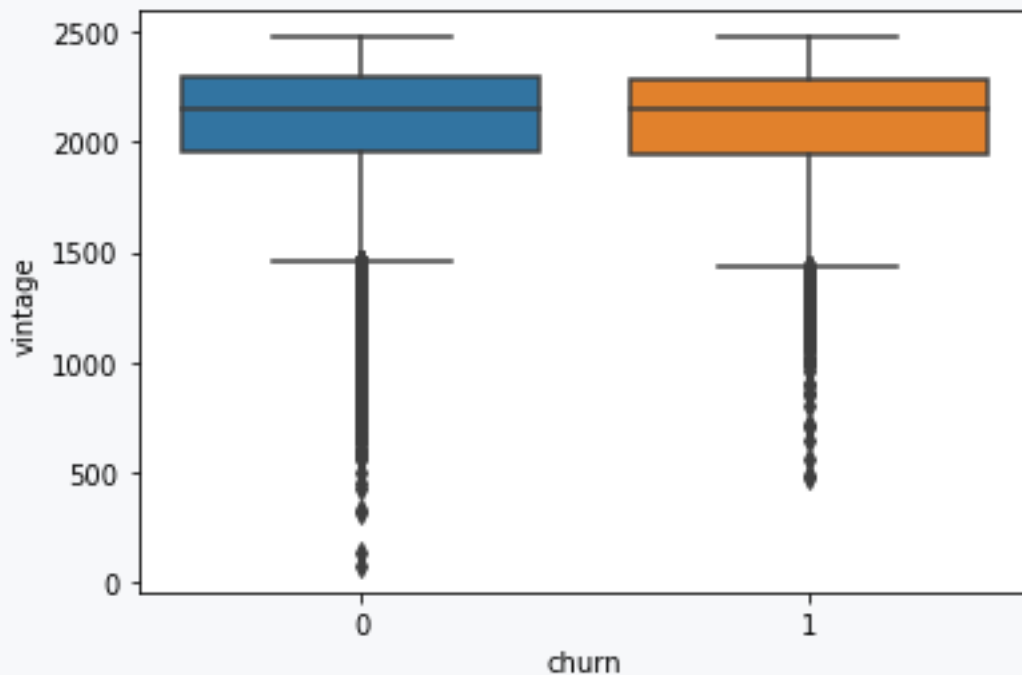
Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file, determine does the length of the customer's relationship with the bank have an impact on customer exposure (churn and vintage). We recommend using visualization in the form of a boxplot.

```
import pandas as pd
import seaborn as sns

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')

gr = sns.boxplot(x='churn', y='vintage', data=df)
```

Program output:



- the length of the contract has no effect
- the length of the contract has an impact
- the distribution of the variable is similar
- the distribution of the variable is significantly different

3.2.5

Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file, find out what is the ratio of male and female customers at risk among the bank's customers (churn and gender). We recommend using a visualization, listing both genders and the percentage rounded to two decimal places in the result. We recommend the use of a bar chart.

```
male churn: 54.25% female churn: 45.75%
```

```
import pandas as pd
import seaborn as sns
```

```

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')

df['gender'] = df['gender'].astype('category') # set as
categorical variable
dfd = df[['gender', 'churn']]
sns.countplot(x='gender', hue='churn', data=dfd)

print(dfd['churn'].loc[dfd['gender']=='Male'].value_counts(normalize=True).mul(100).round(2))
print(dfd['churn'].loc[dfd['gender']=='Female'].value_counts(normalize=True).mul(100).round(2))

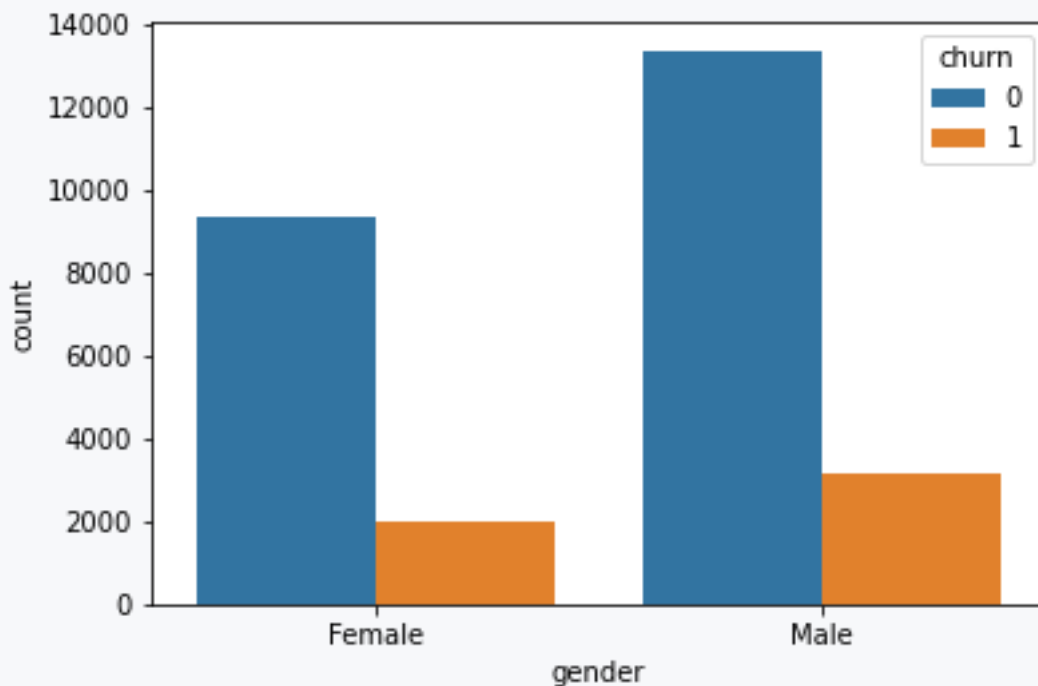
```

Program output:

```

0    80.85
1    19.15
Name: churn, dtype: float64
0    82.45
1    17.55
Name: churn, dtype: float64

```



3.2.6

Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file, find out what is the ratio of customers at risk

based on age among the bank's customers (churn and age). Create a new categorical variable to classify the following age categories:

- young - age<18
- adult - 18<=age<60
- senior - age>=60

We recommend using visualization and printing all age categories and percentages rounded to two decimal places in the result. We recommend the use of a bar chart.

```
young: 50.24% adult: 27.75% senior: 22.01%
```

```
import pandas as pd
import seaborn as sns

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')

dfd = df[['churn', 'age']]
dfd['age_group'] = 'str'
dfd['age_group'][dfd['age']>=60] = 'senior'
dfd['age_group'][(dfd['age']<60) & (dfd['age']>=18)] = 'adult'
dfd['age_group'][dfd['age']<18] = 'young'
sns.countplot(x='age_group', hue='churn', data=dfd)

print(dfd['churn'].loc[dfd['age_group']=='senior'].value_counts(normalize=True).mul(100).round(2))
print(dfd['churn'].loc[dfd['age_group']=='adult'].value_counts(normalize=True).mul(100).round(2))
print(dfd['churn'].loc[dfd['age_group']=='young'].value_counts(normalize=True).mul(100).round(2))
```

Program output:

```
:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy
    dfd['age_group'][dfd['age']>=60] = 'senior'
:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame
```

See the caveats in the documentation:

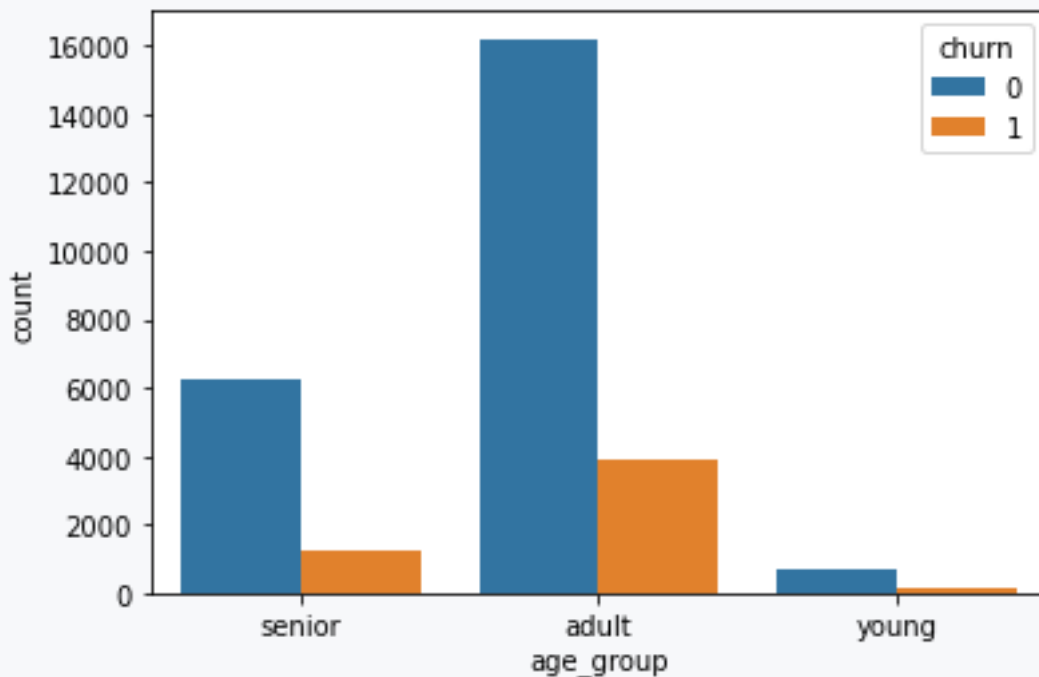
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
dfd['age_group'][(dfd['age']<60) & (dfd['age']>=18)] =
'adult'
:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame
```

See the caveats in the documentation:

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
dfd['age_group'][dfd['age']<18] = 'young'
0    83.17
1    16.83
Name: churn, dtype: float64
0    80.61
1    19.39
Name: churn, dtype: float64
0    87.1
1    12.9
Name: churn, dtype: float64
```



3.3 Multivariate analysis

3.3.1

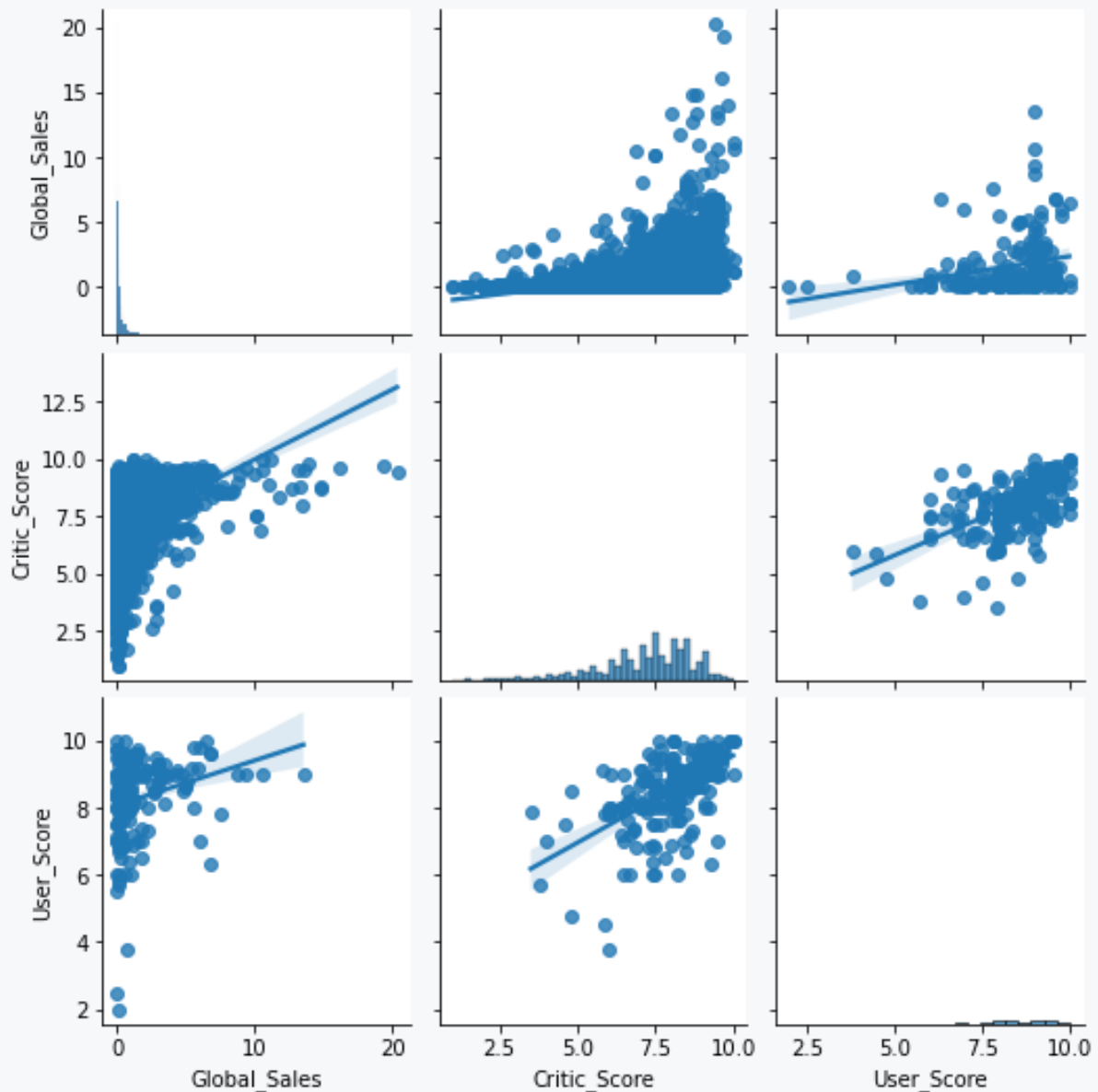
Multivariate analysis is the analysis of three or more variables. This allows us to examine correlations (i.e. how one variable changes relative to another) and attempt to make more accurate predictions of future behaviour than a bivariate analysis. Initially, we explored the visualization of univariate analysis and bivariate analysis; we will follow a similar approach for multivariate analysis.

One common way to visualize multivariate data is to create a matrix scatter plot, also known as a **pairwise plot**. A pairwise plot shows each pair of variables in contrast to each other. The pairwise plot allows us to see both the distribution of each variable and the relationships between the two variables.

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
#print(df.info())
sns.pairplot(data=df,
vars=['Global_Sales', 'Critic_Score', 'User_Score'], kind='reg')
```

Program output:



We obtained a 3x3 matrix graph for the *Global_Sales*, *Critics_Score* and *User_Score* columns. The histogram on the diagonal allows us to show the distribution of one variable. The regression plots on the upper and lower triangles show the relationship between the two variables. The left graph in the third row shows a regression plot representing that there is no correlation between global sales and user score. In comparison, the middle regression plot in the bottom row shows that there is a correlation between critic scores and user scores.

3.3.2

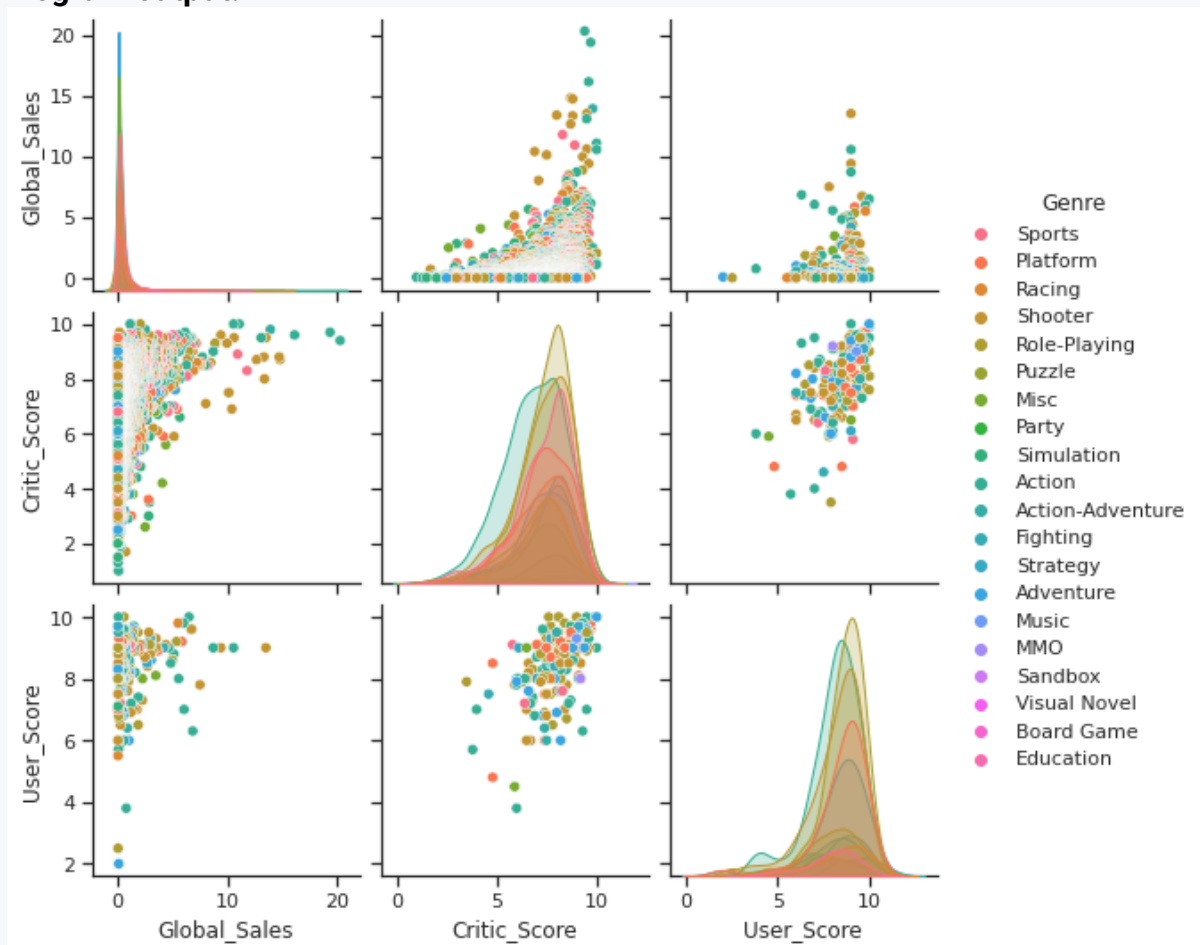
We can augment the pairwise graph with additional information by inserting a color into the graph based on a categorical variable. Therefore, let's insert information about different genres into the graph. Density plots on the diagonal allow us to see

the distribution of one variable, while scatter plots on the upper and lower triangles show the relationship (or correlation) between two variables. The hue parameter is the name of the variable that is used to label the data points, which in our case is the thesis genre. The downside of our view is that we have too many different genres and therefore the visualization is a bit messy.

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
#print(df.info())
sns.set(style='ticks', color_codes=True)
sns.pairplot(data=df,
vars=['Global_Sales', 'Critic_Score', 'User_Score'],
hue='Genre')
```

Program output:



3.3.3

Correlation analysis is an effective technique for determining whether there is a correlation or dependence (relationship) between variables. The calculation of the linear (Pearson) correlation coefficient for a pair of variables can be done using the `corr()` function of the *pandas* library or the `pearsonr()` function of the *scipy* library for a particular pair of variables. In this case, we can observe that there is a small dependence between critics' ratings and worldwide sales but it is statistically significant since the p-value is less than 0.05.

```
import pandas as pd
from scipy import stats

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')

dfd = df[['Global_Sales', 'Critic_Score']].dropna()

corr = stats.pearsonr(dfd['Global_Sales'],
dfd['Critic_Score'])
print("p-value:\t", corr[1])
print("cor:\t\t", corr[0])
```

Program output:

```
p-value:      3.7086715030237096e-87
cor:         0.2959412674530926
```

3.3.4

Load the data from the `banking.csv` file, which contains information about the bank's customers. After loading the data file, see if there is a correlation between the variables `age` and `current_balance`. In this way, we want to see if there is a correlation between the age of the customers and their current account balance. Print whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
import pandas as pd
from scipy import stats
```

```
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')

dfd =
df[['age', 'current_balance']].dropna() #df[['churn', 'gender']][: ]

corr = stats.pearsonr(dfd['age'], dfd['current_balance'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:      0.0
cor:         0.05
```

 3.3.5

Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file, see if there is a correlation between the **previous_month_end_balance** and **current_balance** variables. In this way, we want to see if there is a correlation between the previous month's account balance and the current account balance. List whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
import pandas as pd
from scipy import stats

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')

dfd =
df[['previous_month_end_balance', 'current_balance']].dropna()

corr = stats.pearsonr(dfd['previous_month_end_balance'],
dfd['current_balance'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:    0.0
cor:       0.95
```

 3.3.6

Using the `corr()` function of the *pandas* library, we can generate a table of correlations of all variables in the dataset. A correlation coefficient approaching 1 indicates a very strong positive correlation between two variables. We can observe this on the diagonal, which actually compares a given variable to itself, so it will be 1.

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')

dfd = df[['Global_Sales', 'Critic_Score']].dropna()

correlation = df.corr(method='pearson')
print(correlation)
```

Program output:

```

          Rank  Critic_Score  User_Score
Total_Shipped \
Rank          1.000000    -0.137650    -0.293034    -
0.441132
Critic_Score  -0.137650     1.000000     0.582673
0.203425
User_Score    -0.293034     0.582673     1.000000    -
0.025732
Total_Shipped -0.441132     0.203425    -0.025732
1.000000
Global_Sales  -0.554659     0.295941     0.241650
NaN
NA_Sales      -0.550922     0.314285     0.234039
NaN
PAL_Sales     -0.438841     0.253431     0.190490
NaN
JP_Sales      -0.443212     0.174933     0.108721
NaN
```



```

Other_Sales    -0.427737      0.254755      0.224679
NaN
Year           -0.097345      0.015670     -0.116728      -
0.169701

                Global_Sales  NA_Sales  PAL_Sales  JP_Sales
Other_Sales \
Rank           -0.554659 -0.550922 -0.438841 -0.443212
-0.427737
Critic_Score   0.295941  0.314285  0.253431  0.174933
0.254755
User_Score     0.241650  0.234039  0.190490  0.108721
0.224679
Total_Shipped      NaN      NaN      NaN      NaN
NaN
Global_Sales     1.000000  0.914964  0.904582  0.228782
0.856798
NA_Sales         0.914964  1.000000  0.683959  0.075239
0.687831
PAL_Sales        0.904582  0.683959  1.000000  0.123954
0.814068
JP_Sales         0.228782  0.075239  0.123954  1.000000
0.082254
Other_Sales      0.856798  0.687831  0.814068  0.082254
1.000000
Year            -0.041354 -0.059352  0.082548 -0.351626
0.089282

                Year
Rank           -0.097345
Critic_Score   0.015670
User_Score     -0.116728
Total_Shipped -0.169701
Global_Sales   -0.041354
NA_Sales       -0.059352
PAL_Sales      0.082548
JP_Sales       -0.351626
Other_Sales    0.089282
Year           1.000000

```

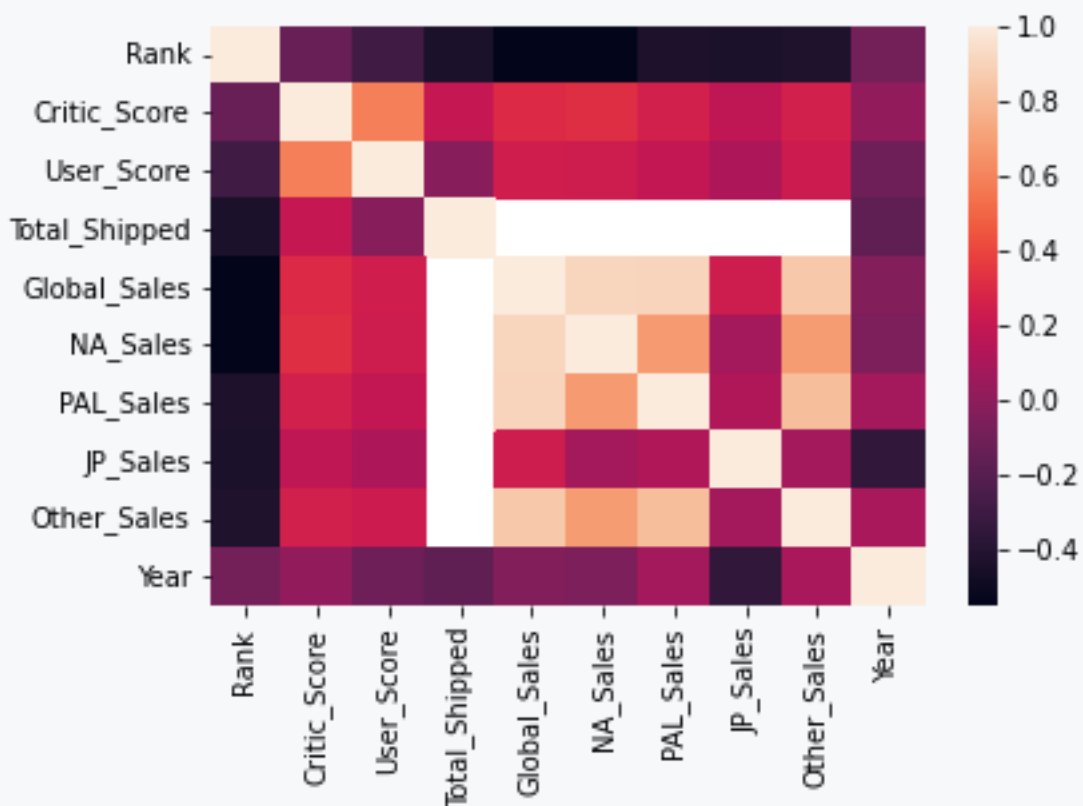
3.3.7

We can also visualize the correlation between variables using a heatmap. This way we can immediately see which variables have a high correlation and vice versa. We will use the `heatmap()` function of the `seaborn` library.

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/games_sales.csv', sep=',')
correlation = df.corr(method='pearson')
sns.heatmap(correlation, xticklabels=correlation.columns,
yticklabels=correlation.columns)
```

Program output:



3.3.8

Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file, find out the correlation between all the variables. We recommend using a **heatmap()** type chart. Based on the visualization, select the true statements.

```
import pandas as pd
from scipy import stats
```

```

import seaborn as sns

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')

correlation = df.corr(method='pearson')
sns.heatmap(correlation, xticklabels=correlation.columns,
yticklabels=correlation.columns)
print(correlation)

```

Program output:

	customer_id	vintage	
age			
dependents			\
customer_id	1.000000	-0.007750	-
0.000442	-0.008616		
vintage	-0.007750	1.000000	
0.006220	0.005192		
age	-0.000442	0.006220	
1.000000	-0.000612		
dependents	-0.008616	0.005192	-
0.000612	1.000000		
city	0.000743	0.007616	
0.015439	0.001892		
customer_nw_category	0.009618	-0.001154	-
0.076532	0.013134		
branch_code	-0.000286	0.003512	-
0.058990	0.020141		
current_balance	0.006589	0.000031	
0.054346	-0.003070		
previous_month_end_balance	0.005819	-0.000669	
0.058342	0.000216		
average_monthly_balance_prevQ	0.004485	-0.002054	
0.061708	0.001213		
average_monthly_balance_prevQ2	-0.002532	-0.001759	
0.059607	0.002949		
current_month_credit	0.002494	-0.004617	
0.023840	0.003260		
previous_month_credit	-0.006414	-0.000169	
0.029961	0.025054		
current_month_debit	0.002603	-0.004978	
0.027702	0.008207		
previous_month_debit	-0.008760	-0.006760	
0.033296	0.032021		

current_month_balance	0.005140	-0.000550
0.057662	-0.000652	
previous_month_balance	0.004553	-0.002208
0.060297	0.001239	
churn	-0.002723	-0.004769
0.020012	0.033487	
	city	customer_nw_category
branch_code \		
customer_id	0.000743	0.009618
-0.000286		
vintage	0.007616	-0.001154
0.003512		
age	0.015439	-0.076532
-0.058990		
dependents	0.001892	0.013134
0.020141		
city	1.000000	0.006613
-0.061234		
customer_nw_category	0.006613	1.000000
0.235059		
branch_code	-0.061234	0.235059
1.000000		
current_balance	-0.005654	-0.058314
0.000181		
previous_month_end_balance	-0.004089	-0.059854
0.000214		
average_monthly_balance_prevQ	-0.006298	-0.059535
0.001955		
average_monthly_balance_prevQ2	-0.007891	-0.047010
0.001310		
current_month_credit	0.004118	-0.025254
-0.013988		
previous_month_credit	0.008087	-0.072374
-0.023849		
current_month_debit	0.001465	-0.035917
-0.016944		
previous_month_debit	0.005995	-0.071721
-0.017584		
current_month_balance	-0.005796	-0.058648
0.001031		
previous_month_balance	-0.005839	-0.059113
0.002080		

churn	-0.001585	0.006551
0.035469		
	current_balance	
previous_month_end_balance \		
customer_id	0.006589	
0.005819		
vintage	0.000031	
-0.000669		
age	0.054346	
0.058342		
dependents	-0.003070	
0.000216		
city	-0.005654	
-0.004089		
customer_nw_category	-0.058314	
-0.059854		
branch_code	0.000181	
0.000214		
current_balance	1.000000	
0.947276		
previous_month_end_balance	0.947276	
1.000000		
average_monthly_balance_prevQ	0.958307	
0.970530		
average_monthly_balance_prevQ2	0.714600	
0.722998		
current_month_credit	0.030371	
0.032493		
previous_month_credit	0.061754	
0.114222		
current_month_debit	0.044412	
0.066329		
previous_month_debit	0.081247	
0.109606		
current_month_balance	0.983412	
0.974714		
previous_month_balance	0.942207	
0.969605		
churn	-0.024181	
0.006886		
	average_monthly_balance_prevQ	
\		

customer_id	0.004485
vintage	-0.002054
age	0.061708
dependents	0.001213
city	-0.006298
customer_nw_category	-0.059535
branch_code	0.001955
current_balance	0.958307
previous_month_end_balance	0.970530
average_monthly_balance_prevQ	1.000000
average_monthly_balance_prevQ2	0.763495
current_month_credit	0.033639
previous_month_credit	0.085699
current_month_debit	0.060579
previous_month_debit	0.121272
current_month_balance	0.976290
previous_month_balance	0.994038
churn	0.011960
	average_monthly_balance_prevQ2
\	
customer_id	-0.002532
vintage	-0.001759
age	0.059607
dependents	0.002949
city	-0.007891
customer_nw_category	-0.047010
branch_code	0.001310
current_balance	0.714600
previous_month_end_balance	0.722998
average_monthly_balance_prevQ	0.763495
average_monthly_balance_prevQ2	1.000000
current_month_credit	0.036271
previous_month_credit	0.062264
current_month_debit	0.045239
previous_month_debit	0.102519
current_month_balance	0.725826
previous_month_balance	0.736635
churn	0.018376
	current_month_credit
previous_month_credit \	
customer_id	0.002494
-0.006414	

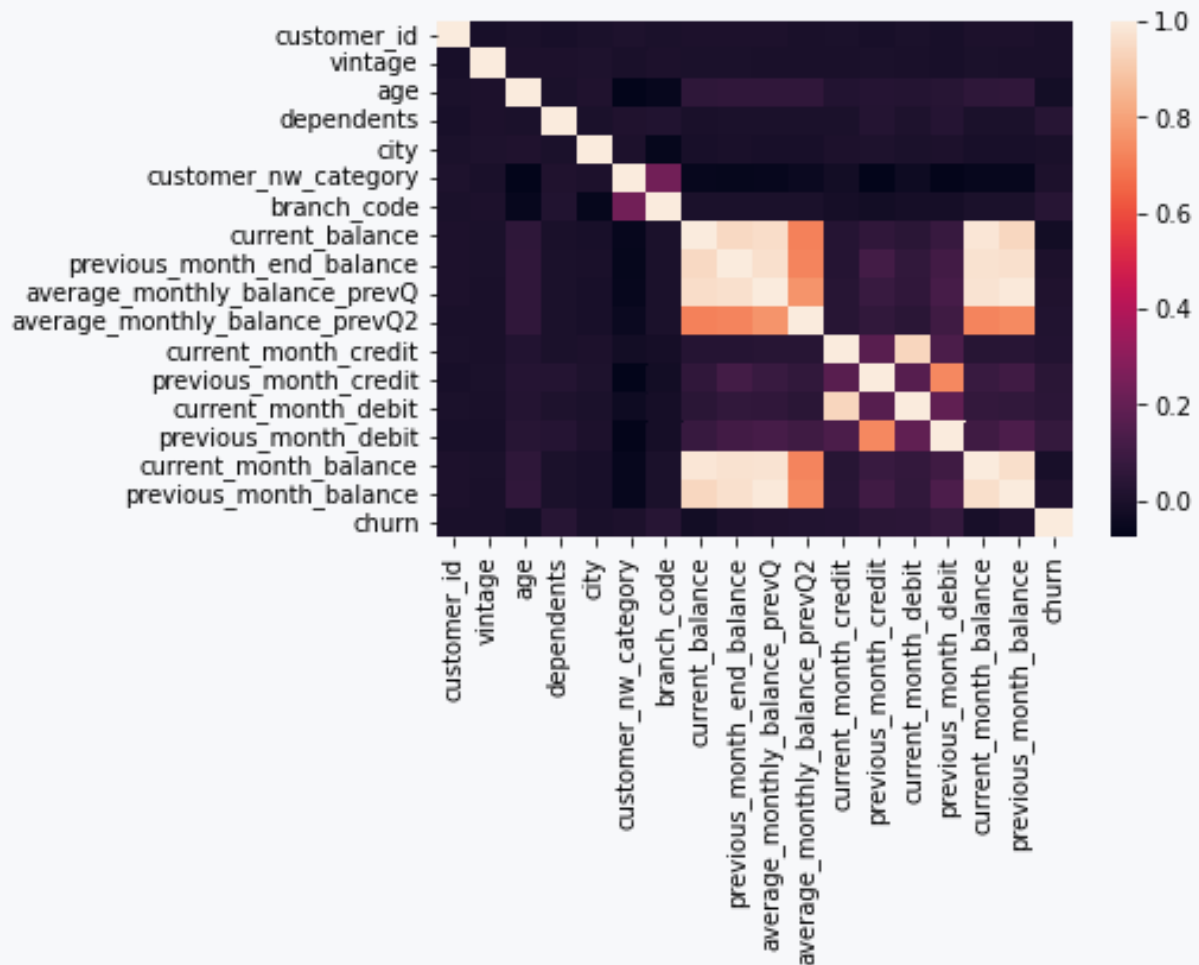
vintage	-0.004617
-0.000169	
age	0.023840
0.029961	
dependents	0.003260
0.025054	
city	0.004118
0.008087	
customer_nw_category	-0.025254
-0.072374	
branch_code	-0.013988
-0.023849	
current_balance	0.030371
0.061754	
previous_month_end_balance	0.032493
0.114222	
average_monthly_balance_prevQ	0.033639
0.085699	
average_monthly_balance_prevQ2	0.036271
0.062264	
current_month_credit	1.000000
0.168561	
previous_month_credit	0.168561
1.000000	
current_month_debit	0.937021
0.165092	
previous_month_debit	0.135729
0.733953	
current_month_balance	0.034182
0.085320	
previous_month_balance	0.038254
0.108496	
churn	0.020755
0.042179	
	current_month_debit
previous_month_debit \	
customer_id	0.002603
-0.008760	
vintage	-0.004978
-0.006760	
age	0.027702
0.033296	

dependents	0.008207
0.032021	
city	0.001465
0.005995	
customer_nw_category	-0.035917
-0.071721	
branch_code	-0.016944
-0.017584	
current_balance	0.044412
0.081247	
previous_month_end_balance	0.066329
0.109606	
average_monthly_balance_prevQ	0.060579
0.121272	
average_monthly_balance_prevQ2	0.045239
0.102519	
current_month_credit	0.937021
0.135729	
previous_month_credit	0.165092
0.733953	
current_month_debit	1.000000
0.191755	
previous_month_debit	0.191755
1.000000	
current_month_balance	0.069720
0.102010	
previous_month_balance	0.063375
0.139723	
churn	0.048041
0.073058	
	current_month_balance
previous_month_balance \	
customer_id	0.005140
0.004553	
vintage	-0.000550
-0.002208	
age	0.057662
0.060297	
dependents	-0.000652
0.001239	
city	-0.005796
-0.005839	

customer_nw_category	-0.058648
-0.059113	
branch_code	0.001031
0.002080	
current_balance	0.983412
0.942207	
previous_month_end_balance	0.974714
0.969605	
average_monthly_balance_prevQ	0.976290
0.994038	
average_monthly_balance_prevQ2	0.725826
0.736635	
current_month_credit	0.034182
0.038254	
previous_month_credit	0.085320
0.108496	
current_month_debit	0.069720
0.063375	
previous_month_debit	0.102010
0.139723	
current_month_balance	1.000000
0.963276	
previous_month_balance	0.963276
1.000000	
churn	-0.006391
0.014593	

	churn
customer_id	-0.002723
vintage	-0.004769
age	-0.020012
dependents	0.033487
city	-0.001585
customer_nw_category	0.006551
branch_code	0.035469
current_balance	-0.024181
previous_month_end_balance	0.006886
average_monthly_balance_prevQ	0.011960
average_monthly_balance_prevQ2	0.018376
current_month_credit	0.020755
previous_month_credit	0.042179
current_month_debit	0.048041
previous_month_debit	0.073058
current_month_balance	-0.006391

```
previous_month_balance    0.014593
churn                     1.000000
```



- there is no relationship between demographic variables
- there is a relationship between demographic variables
- there is a relationship between customer variables
- there is no relationship between customer variables
- there is a relationship between variables on transactions
- there is no relationship between transaction variables

3.3.9

Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file, find the correlation between the variables from the category of transaction information. We recommend using a **heatmap()** type chart. Based on the visualization, select the true statements.

```
import pandas as pd
from scipy import stats
import seaborn as sns
```

```

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.csv', sep=',', decimal='.')
print(df.info())
dfd =
df[['current_balance', 'previous_month_end_balance', 'average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2', 'current_month_credit', 'previous_month_credit', 'current_month_debit', 'previous_month_debit', 'current_month_balance', 'previous_month_balance', 'churn']][::]

correlation = dfd.corr(method='pearson')
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.columns)
#print(correlation)

```

Program output:

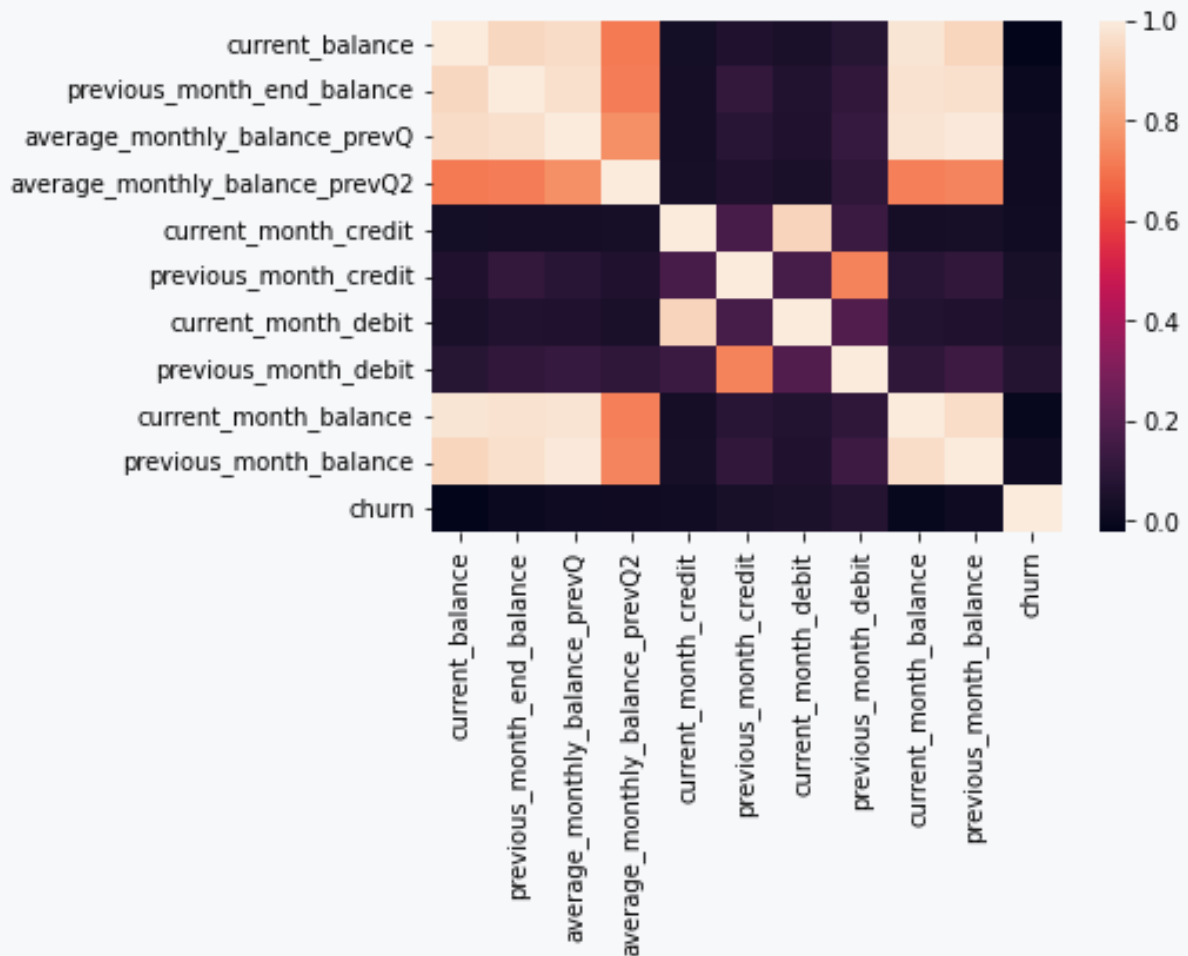
```

RangeIndex: 28382 entries, 0 to 28381
Data columns (total 21 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   customer_id                           28382 non-null  int64
1   vintage                                28382 non-null  int64
2   age                                     28382 non-null  int64
3   gender                                  27857 non-null  object
4   dependents                             25919 non-null  float64
5   occupation                              28302 non-null  object
6   city                                    27579 non-null  float64
7   customer_nw_category                  28382 non-null  int64
8   branch_code                            28382 non-null  int64
9   current_balance                        28382 non-null  float64
10  previous_month_end_balance              28382 non-null  float64
11  average_monthly_balance_prevQ           28382 non-null  float64
12  average_monthly_balance_prevQ2         28382 non-null  float64
13  current_month_credit                    28382 non-null  float64
14  previous_month_credit                   28382 non-null  float64
15  current_month_debit                     28382 non-null  float64
16  previous_month_debit                    28382 non-null  float64
17  current_month_balance                   28382 non-null  float64
18  previous_month_balance                  28382 non-null  float64
19  churn                                   28382 non-null  int64
20  last_transaction                        28382 non-null  object
dtypes: float64(12), int64(6), object(3)

```

memory usage: 4.5+ MB

None



- there is a relationship between the current balance and balances from previous months
- there is no relationship between the current balance and balances from previous months
- the transaction variables debit/credit are mainly correlated with each other
- the transaction variables debit/credit are correlated with all variables
- the transaction variables debit/credit do not correlate with the balance variables
- the transaction variables debit/credit are correlated with the balance variables

Project - Data Analysis

Chapter **4**

4.1 Project – Exploration data analysis

4.1.1

The project focuses on the analysis of the company's employees. The dataset contains information about employees. The most important data and variables used in the analysis are:

- **Age** - age of the employee
- **Department** - department
- **DistanceFromHome** - the distance of the employee's home from the place of work
- **Education** - level of education
- **EducationField** - the area in which the employee has studied
- **MonthlyIncome** - monthly income
- **JobLevel** - job position level (values from 1 to 5)
- **YearsAtCompany** - the number of years he has worked in the company
- **TotalWorkingYears** - total number of years of employment

```
# import library
import pandas as pd
# read csv https://priscilla.fitped.eu/data/nlp/employees.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# explore dataset
print(df.info())
```

Program output:

```
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Age                                    1470 non-null   int64
1   Attrition                             1470 non-null   object
2   BusinessTravel                         1470 non-null   object
3   DailyRate                              1470 non-null   int64
4   Department                             1470 non-null   object
5   DistanceFromHome                       1470 non-null   int64
6   Education                              1470 non-null   int64
7   EducationField                         1470 non-null   object
8   EmployeeCount                           1470 non-null   int64
9   EmployeeNumber                         1470 non-null   int64
10  EnvironmentSatisfaction                 1470 non-null   int64
```

```

11 Gender 1470 non-null object
12 HourlyRate 1470 non-null int64
13 JobInvolvement 1470 non-null int64
14 JobLevel 1470 non-null int64
15 JobRole 1470 non-null object
16 JobSatisfaction 1470 non-null int64
17 MaritalStatus 1470 non-null object
18 MonthlyIncome 1470 non-null int64
19 MonthlyRate 1470 non-null int64
20 NumCompaniesWorked 1470 non-null int64
21 Over18 1470 non-null object
22 OverTime 1470 non-null object
23 PercentSalaryHike 1470 non-null int64
24 PerformanceRating 1470 non-null int64
25 RelationshipSatisfaction 1470 non-null int64
26 StandardHours 1470 non-null int64
27 StockOptionLevel 1470 non-null int64
28 TotalWorkingYears 1470 non-null int64
29 TrainingTimesLastYear 1470 non-null int64
30 WorkLifeBalance 1470 non-null int64
31 YearsAtCompany 1470 non-null int64
32 YearsInCurrentRole 1470 non-null int64
33 YearsSinceLastPromotion 1470 non-null int64
34 YearsWithCurrManager 1470 non-null int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
None

```

4.1.2

Calculate the **absolute frequencies** of employees for all departments (**Department**). How many employees does the **Sales** Department have?

```

# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# calculate counts of employees in departments

```

4.1.3

You can already calculate the number of employees in each department. Complete the code **in one line** to calculate the average of these numbers. The result should be 490.

```
# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# calculate mean of counts of employess in departments
```

4.1.4

What command do we use to plot the **histogram** for sorting the **DailyRate** variable?

```
# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://raw.githubusercontent.com/sasu4/pris_data/main/employees.csv', sep=',')
df["DailyRate"].plot.hist()
df["DailyRate"].plot.bar()
```

-
- df["DailyRate"].value_counts().plot.bar()
- df["DailyRate"].value_counts().plot.hist()

4.1.5

Calculate the **frequencies** of employees according to the **level of education** they have attained. However, calculate these numbers only for employees from the **Sales** Department.

How many employees in the sales department have a level of education **higher than 3**?

```
# import library
import pandas as pd
# read csv
```



```
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# filter only the sales department and list the numbers for
education
```

4.1.6

How do we calculate the **variation range** of the **DailyRate** variable?

```
# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
df["DailyRate"].max()-df["DailyRate"].min()
df["DailyRate"].max()+df["DailyRate"].min()
df["DailyRate"].sum()-df["DailyRate"].count()
df["DailyRate"].min()-df["DailyRate"].max()
df["DailyRate"].sum()-df["DailyRate"].avg()
```

4.1.7

What does it mean if the **standard deviation is high**?

- The values are more scattered within the variation range.
- Most of the values are around the average.
- Most values are around the median.
- Values are scattered well outside the range of variation too.

4.1.8

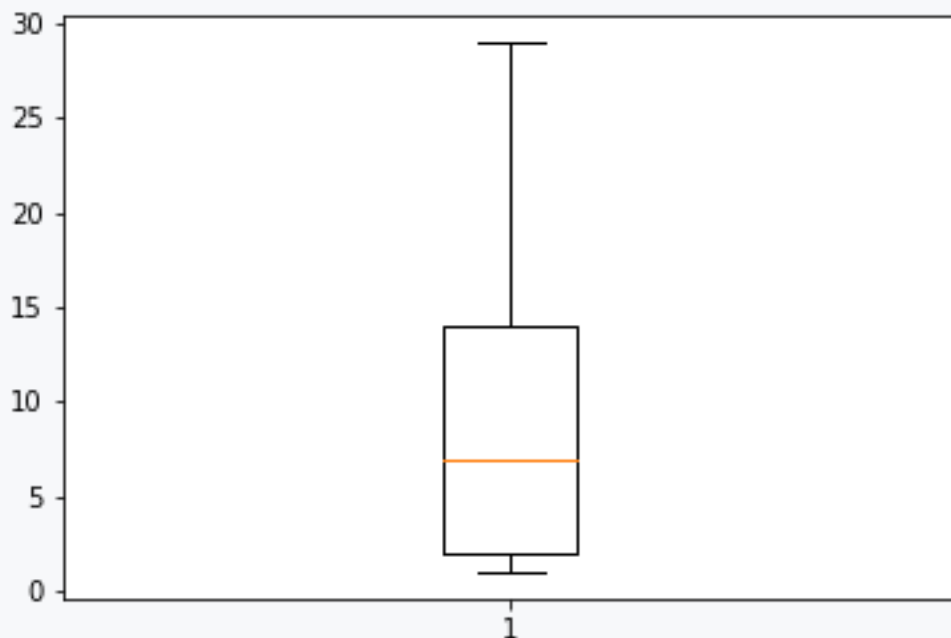
What is the standard deviation of the **age** of employees? (round the result to 2 decimal places)

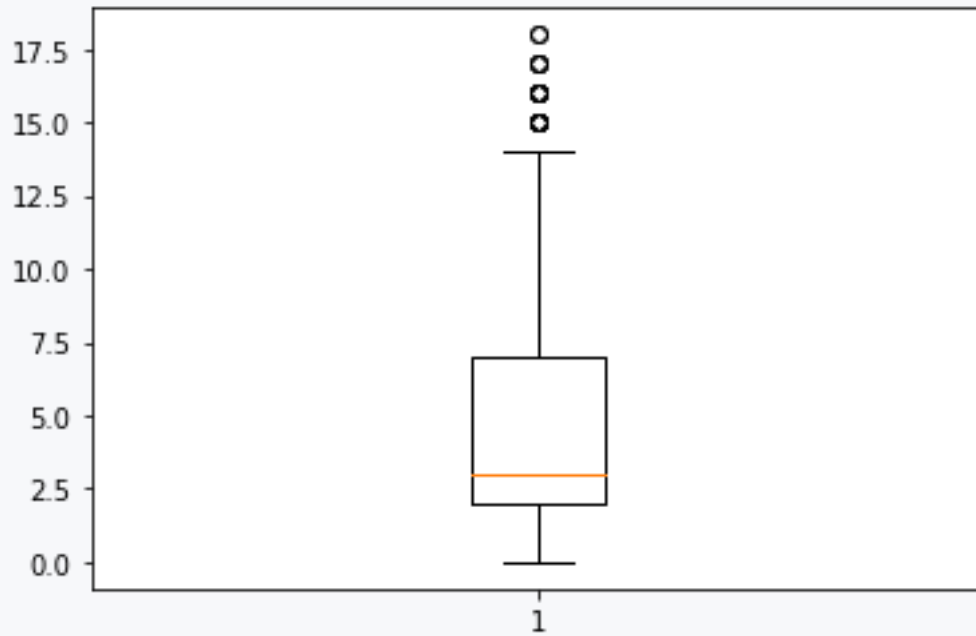
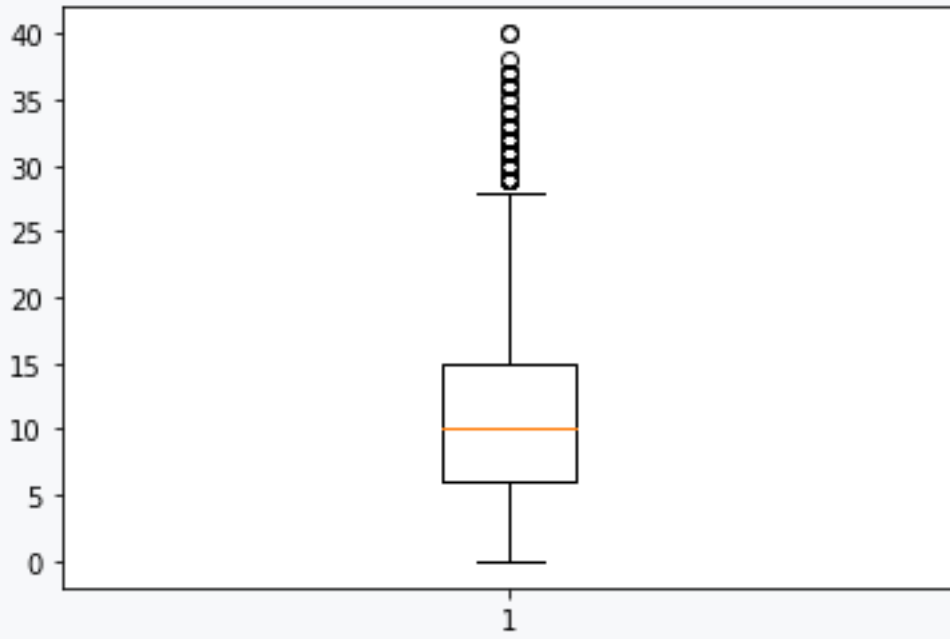
```
# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# calculate the standard deviation of the variable Age using
the pandas library
```

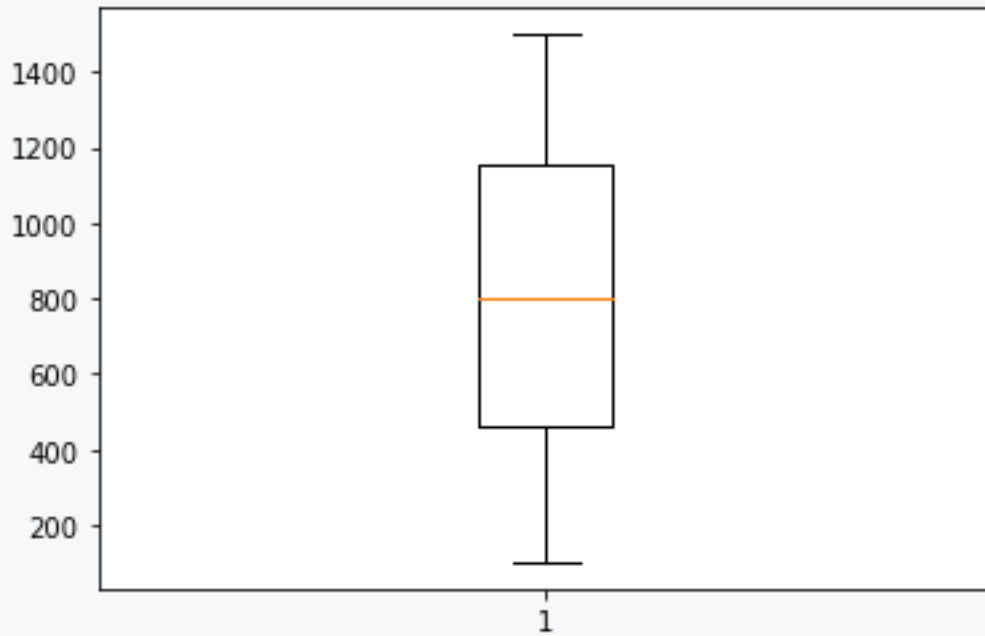
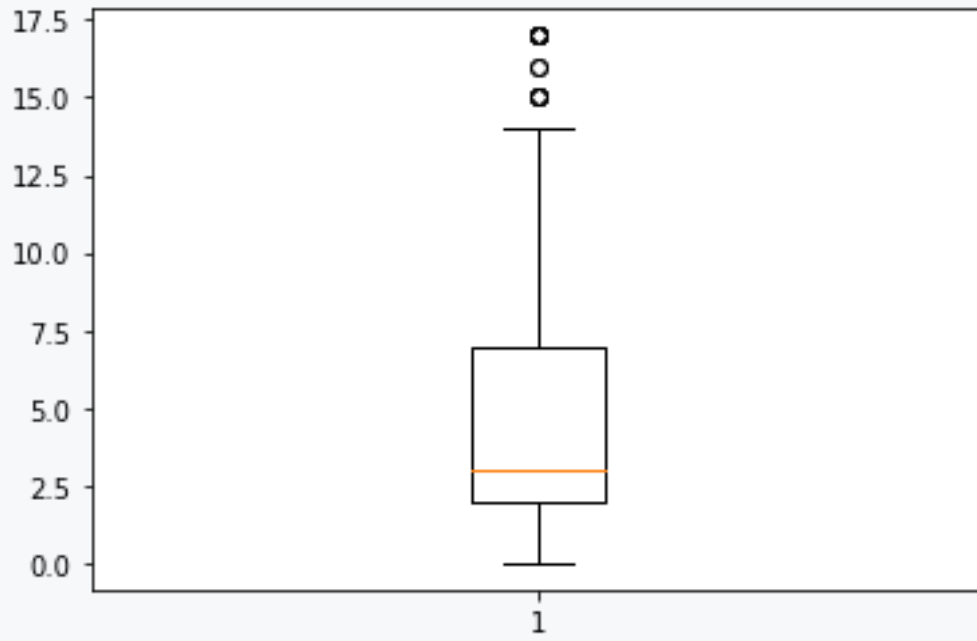
 4.1.9

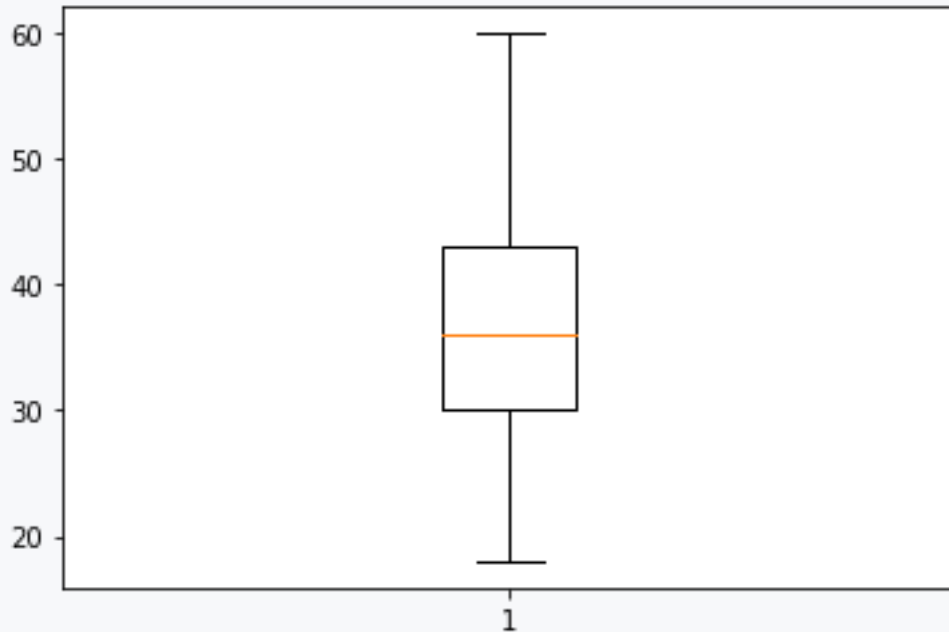
Use the *Matplotlib* library to plot a **box plot** for the **distance of the employee's home from the work location**. Which of the following box plots visualizes the distribution of this variable?

```
# import library
import pandas as pd
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# display a boxplot for distance from home using the
matplotlib library
```







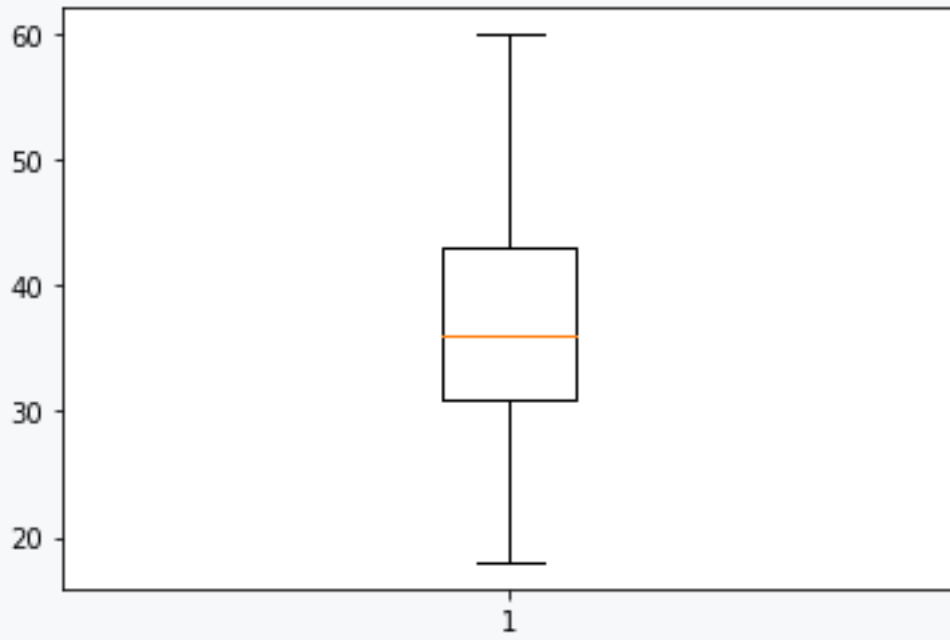
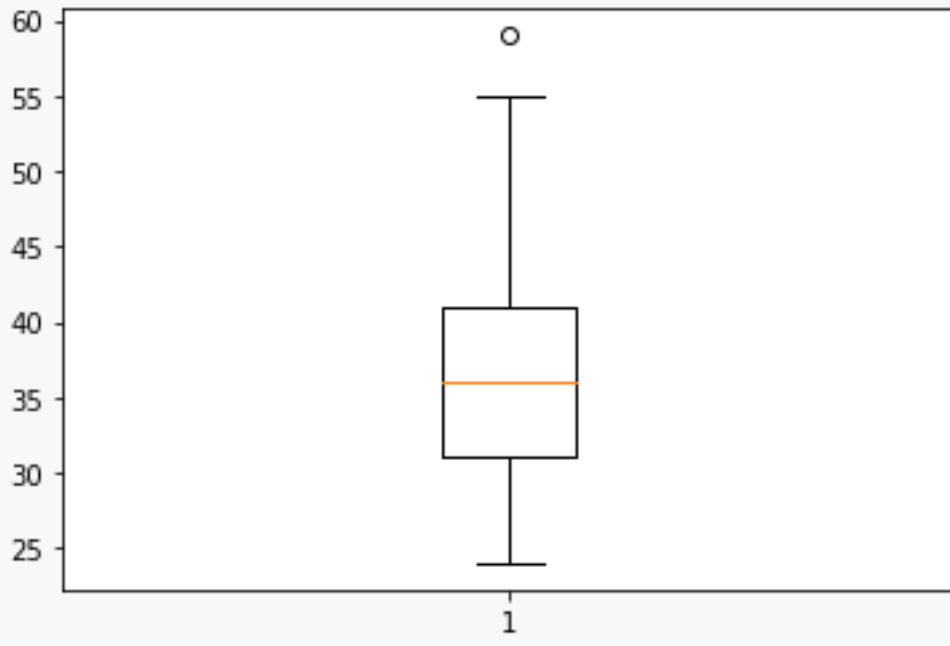


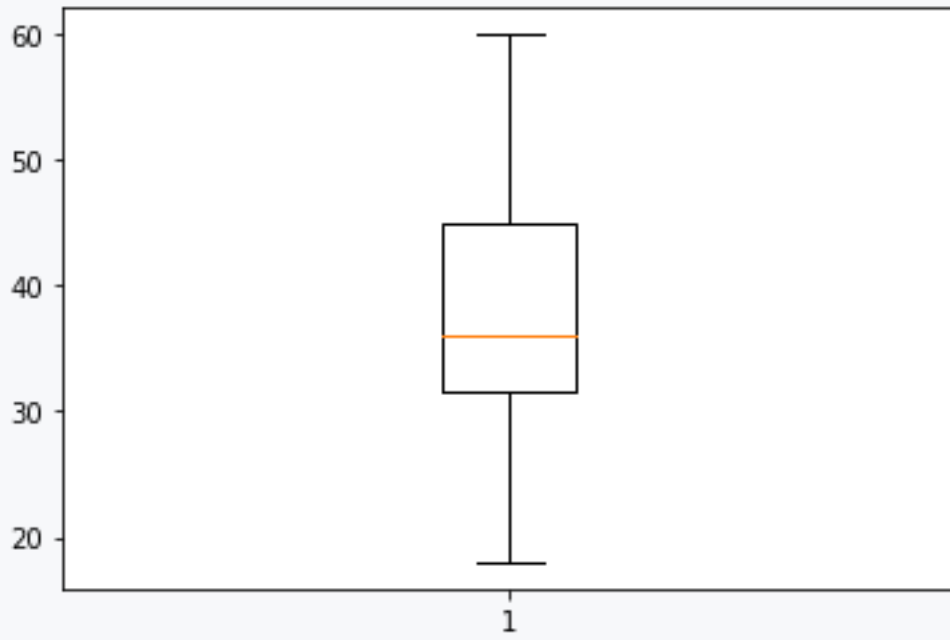
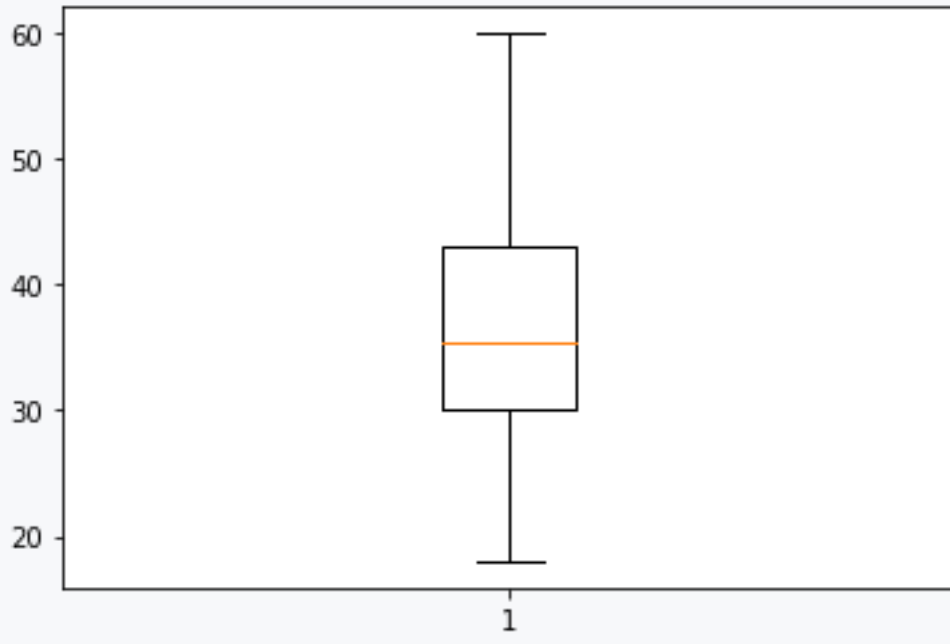
4.1.10

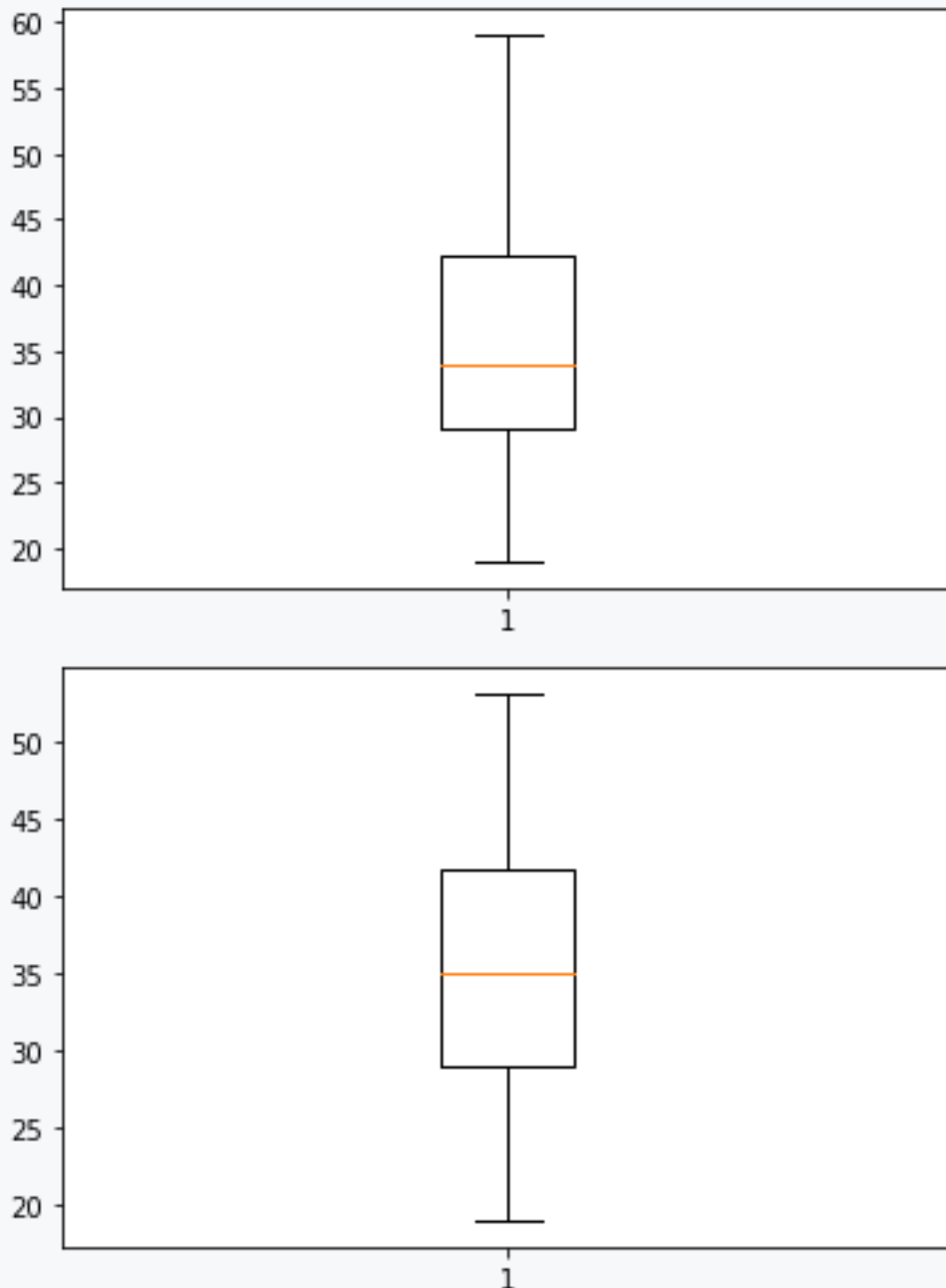
Plot a **box plot** of the distribution of the **age** of employees who have graduated with a degree in **human resources**.

Which of the following plots shows this?

```
# import library
import pandas as pd
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# display a box plot for the age of employees who have a
degree in human resources
```







4.1.11

If a variable has a **positive skewness**, it means that:

- Most values are close to the measure of central tendency
- The values are relatively homogeneously distributed over the variation range
- Most values are greater than average
- Most values are less than the average

4.1.12

Plot a **histogram** that describes the distribution of a variable that represents the total **number of years of employment** of an employee. Use 8 intervals.

Which of the following statements can be **read from the plot**?

```
# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# draw a histogram of the variable total number of years the
employee has worked
df["TotalWorkingYears"].plot.hist(bins = 8)
```

- The kurtosis is probably positive
- The kurtosis is probably negative
- The kurtosis is probably close to zero
- The skewness is probably positive
- The skewness is probably negative
- The skewness is probably close to zero
- Probably does not have a normal distribution
- Probably has a normal distribution
- The mode is 7.5
- The median is less than 15
- The mode is in the interval of 5 to 10
- The median is greater than 15

4.1.13

Show the **pivot table** to find the frequencies for the combinations of what **department** the employee works in and what **level of education** they have attained.

Select from the options, combining which will give the resulting number of such employees **128**.

```
# import library
import pandas as pd
import numpy as np
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
```

```
# draw a pivot table for department and level of education
```

- Sales
- Research & Development
- Human Resources
- 4
- 1
- 2
- 3
- 5

4.1.14

Use the Seaborn library to show **box plots** for monthly employee income (**MonthlyIncome**). Plot a box plot for each group by education (**Education**).

After the plots are drawn, identify the group (level of education attained) that has the **highest income**. What **color** is the box plot for this group with the default Seaborn setting?

```
# import libraries
import pandas as pd
import seaborn as sns
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
```

- purple
- blue
- yellow
- orange
- red

4.1.15

Draw **box plots** for the variable **age** using the Seaborn library. However, the output should contain two box plots, one for the group with **JobLevel equal to 1** and the other with **JobLevel equal to 5**.

What can be **clearly** deduced from this visualization?

```
# import libraries
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# dark design setting
plot = sns.set(style="darkgrid")
# draw boxplots for the age variable for the group with
JobLevel equal to 1 and the other with JobLevel equal to 2.
plot =
# show chart
plt.show()
```

- No employee at level 5 is less than 35 years of age.
- Every employee of the company is less than 60 years old.
- The youngest employee at Level 5 is older than 75% of all employees at Level 1.
- That a Level 1 employee would be over 53 years old is exceptional.
- The range of variation in the age of employees at level 1 is approximately 18 to 52 years.
- All employees at level 5 are between 39 and 60 years of age.
- The majority of Level 1 employees are between the ages of 27 and 37.
- The average age of employees at Level 1 is 32.

4.1.16

Which of the following tests are used to test the **normality** of a variable?

- Lilliefors' test
- Kolmogorov-Smirnov test
- Shapiro-Wilk W test
- T-test
- Cochran-Cox test
- Mann-Whitney U test

4.1.17

Use the Shapiro-Wilk test to check the normality of the variable **age**. Show the result. Copy the entire output of the test into the answer sheet.

```
# import library
import pandas as pd
from scipy import stats
# read csv
```

```
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# use the Shapiro-Wilk test to verify the normality of the age variable
```

4.1.18

Verify that the variable **age** has a **normal distribution**.

```
# import library
import pandas as pd
from scipy import stats
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
```

- It does not have a normal distribution.
- It has a normal distribution.

4.1.19

Draw a **jointplot** from the Seaborn library for the variable **monthly income** and **total number of years of employment** (not just at this company).

Which of the following statements can be **read from the plot**?

```
# import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')

# draw a jointplot from the Seaborn library for the variable
monthly income and total years worked
```

- Monthly income depends significantly on the number of years of employment.
- Monthly income does not depend significantly on the number of years of employment.
- The variable MonthlyIncome does not have a normal distribution.

- The variable MonthlyIncome has a normal distribution.
- The TotalWorkingYears variable does not have a normal distribution.
- The TotalWorkingYears variable has a normal distribution.
- If an employee has a higher income, he or she also has more years of employment.
- If an employee has less income, he or she has less years of employment.

4.1.20

Using the **Scipy** library, calculate **Pearson's R** with the corresponding p-value. Evaluate the correlation between the variable **monthly income** and the **number of years worked in the company**.

Copy the entire output into your answer.

```
# import library
import pandas as pd
from scipy import stats
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# evaluate the correlation between the variable monthly income
and the number of years worked in the company
```

4.1.21

Calculate the **correlation coefficients** between the variables **Age**, **DailyRate**, **JobLevel**, **MonthlyIncome**, **TotalWorkingYears**, **YearsAtCompany**.

On which variable does the employee's **monthly income** depend most?

```
# import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
```

```
# calculate the correlation coefficients between the variables  
Age, DailyRate, JobLevel, MonthlyIncome, TotalWorkingYears,  
YearsAtCompany
```

- JobLevel
- Age
- DailyRate
- TotalWorkingYears
- YearsAtCompany

Analysis of Titanic Data

Chapter **5**

5.1 Analysis of Titanic data

5.1.1

The data analysis project focuses on a very popular dataset related to the sinking of the Titanic. In this tragedy, 1502 of the 2224 passengers and crew died. The dataset contains information on 887 actual Titanic passengers. Each line represents one passenger. The columns contain the following information about the passengers:

- PassengerID - unique passenger identifier
- Survived - information on whether the passenger survived (1) or not (0)
- Pclass - passenger class (1,2,3)
- Name - name of the passenger
- Sex - passenger's gender
- Age - age of the passenger
- SibSp - number of siblings or spouses on board
- Parch - number of parents or children on board
- Ticket - ticket number
- Fare - fare of the ticket
- Cabin - cabin number
- Embarked - the city where the passenger boarded (C - Cherbourg, S - Southampton, Q - Queenstown)

In the following micro-lectures, we will look at which characteristics had the highest correlation with passengers' chances of survival.

```
# import library
import pandas as pd
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.csv', sep=',')
# explore dataset
print(df.info())
```

Program output:

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     891 non-null    int64
1   Survived        891 non-null    int64
2   Pclass          891 non-null    int64
```



```

3   Name          891 non-null   object
4   Sex           891 non-null   object
5   Age          714 non-null   float64
6   SibSp        891 non-null   int64
7   Parch        891 non-null   int64
8   Ticket       891 non-null   object
9   Fare         891 non-null   float64
10  Cabin         204 non-null   object
11  Embarked     889 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

```

5.1.2

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and see if the dataset contains any missing data. If so, list the variable with the largest number and its count. Print the result in the following form:

```
PassengerID: 235
```

```

# import library
import pandas as pd
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
# explore dataset
total = df.isnull().sum().sort_values(ascending=False)
print(total)

```

Program output:

```

Cabin          687
Age            177
Embarked        2
PassengerId     0
Survived        0
Pclass         0
Name           0
Sex            0
SibSp          0
Parch          0

```

```
Ticket      0
Fare        0
dtype: int64
```

5.1.3

After reviewing the missing data, decide which statements are true.

- except for the variables Cabin, Age and Cabin, the other variables are fine
- the Cabin variable contains too many missing values
- we need to delete all rows that contain missing values
- we need to complete all rows of the Cabin variable that contain missing values
- we will not consider the Cabin variable because it contains too many missing values
- the Age variable will not be considered because it contains too many missing values

5.1.4

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset to determine the ratio of male to female survivors. Write out the result as a percentage rounded to two decimal places and in the following format:

```
Male: 23.50%, Female: 33.42%
```

```
# import library
import pandas as pd
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
# explore dataset
#percentage of women survived
women = df.loc[df.Sex == 'female']["Survived"]
rate_women = round(sum(women)/len(women)*100,2)

#percentage of men survived
men = df.loc[df.Sex == 'male']["Survived"]
rate_men = round(sum(men)/len(men)*100,2)

print(str(rate_women) + " % of women who survived." )
print(str(rate_men) + " % of men who survived." )
```

Program output:

```
74.2 % of women who survived.
18.89 % of men who survived.
```

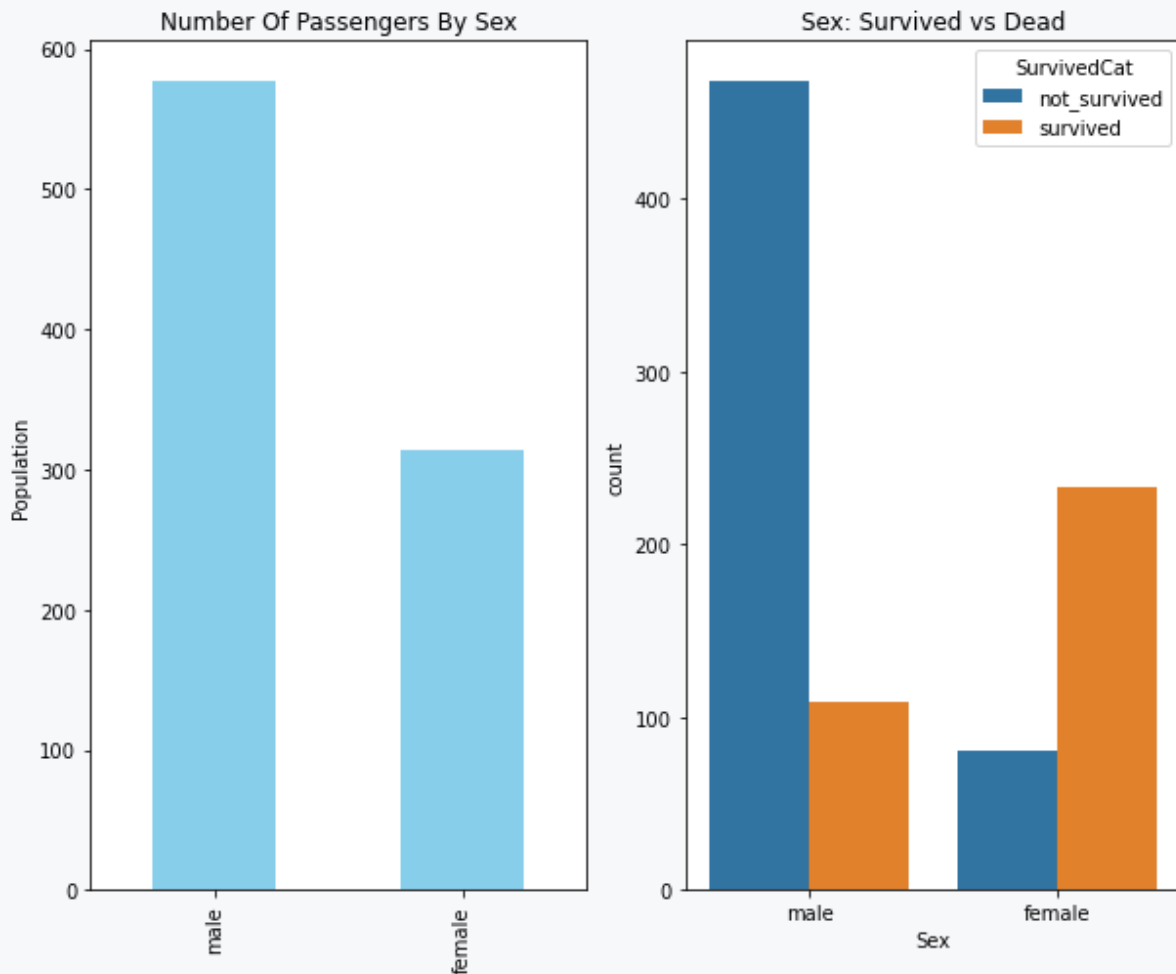
 **5.1.5**

Based on an examination of the ratio of male to female survivors of the disaster decide which statements are true. You can help by visualizing using a bar graph. Also, visualise the proportion of men and women on the boat.

```
# import library
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
# explore dataset
df['SurvivedCat'] = df['Survived'].map({0:"not_survived",
1:"survived"})

fig, ax = plt.subplots(1, 2, figsize = (10, 8))
df["Sex"].value_counts().plot.bar(color = "skyblue", ax =
ax[0])
ax[0].set_title("Number Of Passengers By Sex")
ax[0].set_ylabel("Population")
sns.countplot(x="Sex", hue = "SurvivedCat", data = df)
ax[1].set_title("Sex: Survived vs Dead")
plt.show()
```

Program output:



- the percentage of female survivors is high
- the percentage of male survivors is high
- the percentage of male survivors is low
- the percentage of female survivors is low
- gender can affect the chance of survival
- gender does not affect the chance of survival
- there were more men than women on the ship
- there were more women than men on the ship
- there were approximately the same number of men as women on the ship

5.1.6

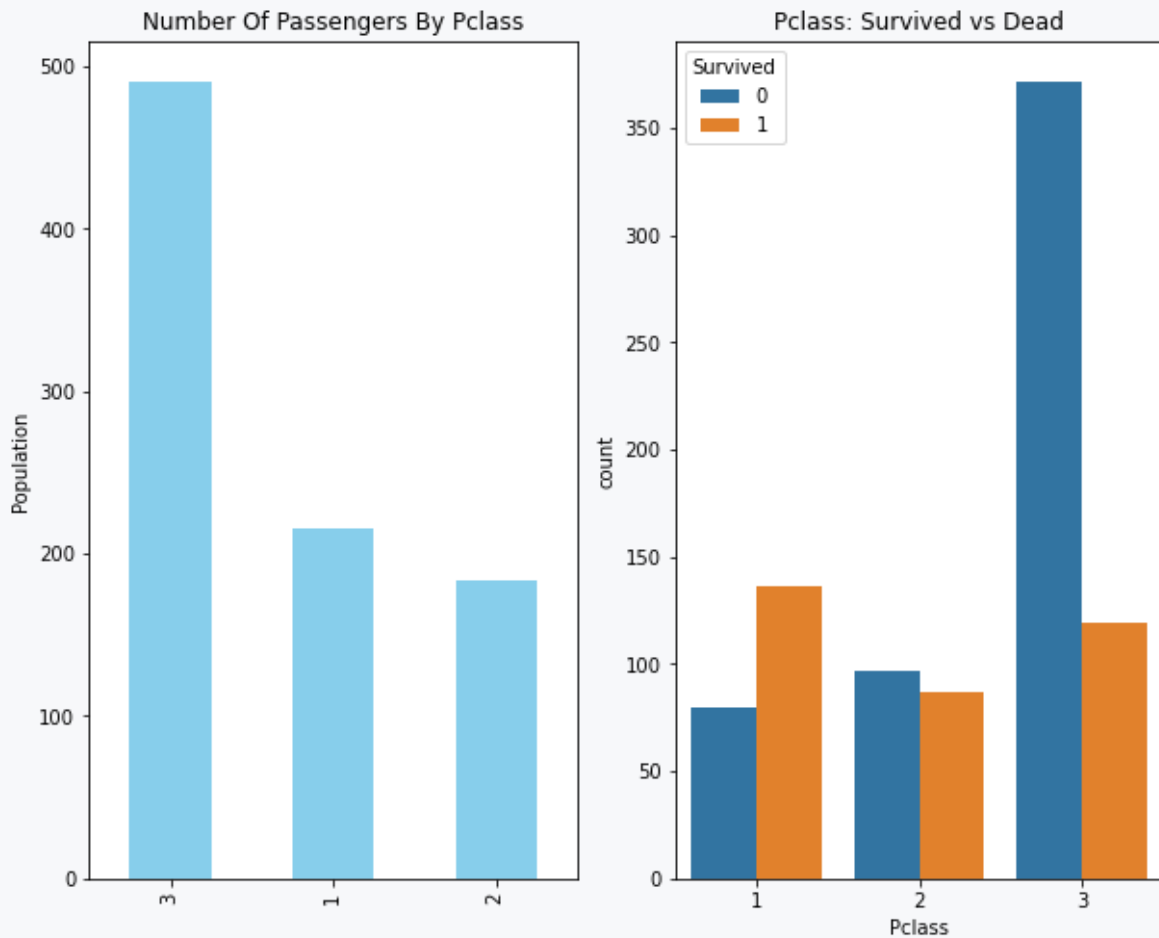
Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and find out the distribution of the number of passengers in each class. Write the result in numbers and in the following format:

```
Class 1: 459, Class 2: 232, Class 3: 120
```

```
# import library
import pandas as pd
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
# explore dataset
fig, ax = plt.subplots(1, 2, figsize = (10, 8))
df["Pclass"].value_counts().plot.bar(color = "skyblue", ax =
ax[0])
ax[0].set_title("Number Of Passengers By Pclass")
ax[0].set_ylabel("Population")
sns.countplot(x="Pclass", hue = "Survived", data = df, ax =
ax[1])
ax[1].set_title("Pclass: Survived vs Dead")
plt.show()

print(df['Pclass'].value_counts())
```

Program output:



```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

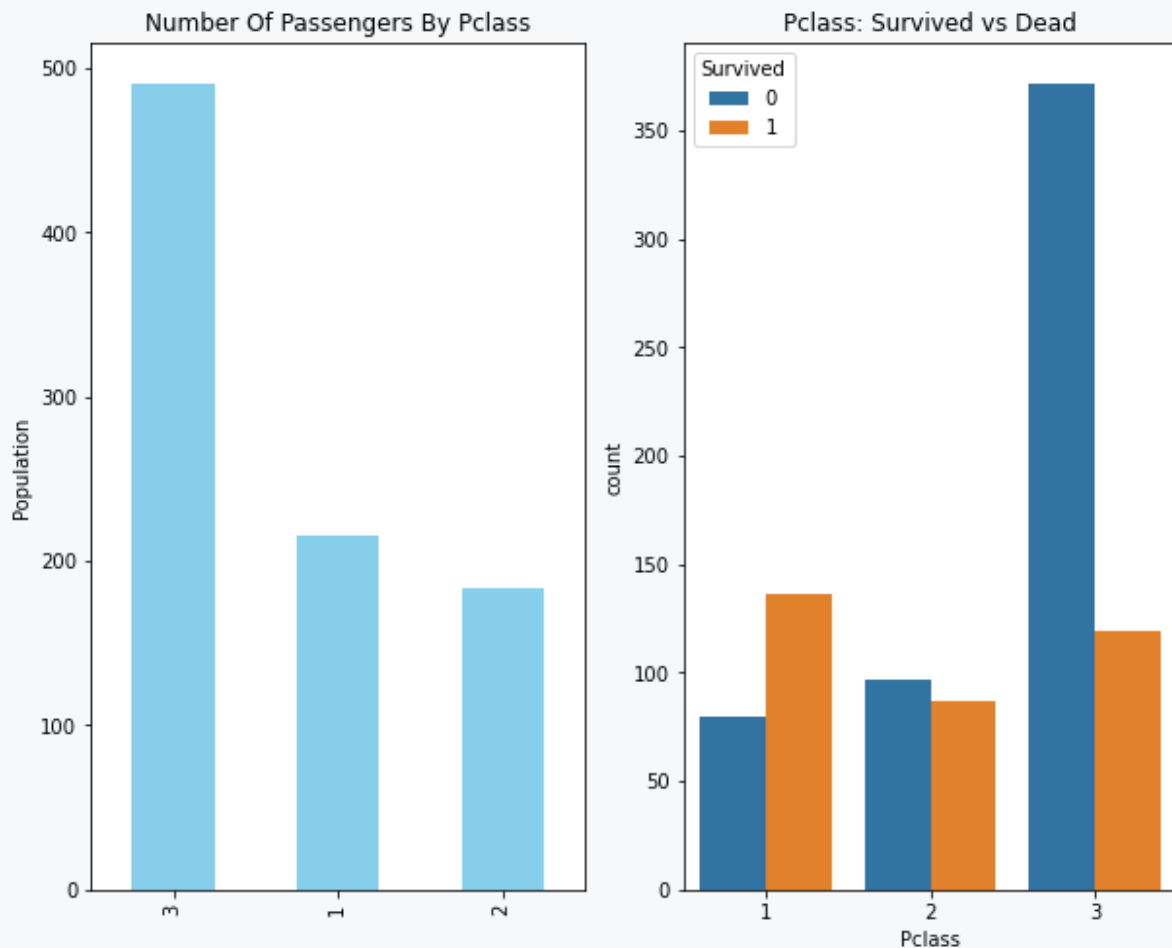
5.1.7

Based on a review of the distribution of passengers by class, review the distribution of passengers who survived the disaster by class. Decide which statements are true. You can help by visualizing using a bar graph.

```
# import library
import pandas as pd
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.csv', sep=',')
```

```
# explore dataset
fig, ax = plt.subplots(1, 2, figsize = (10, 8))
df["Pclass"].value_counts().plot.bar(color = "skyblue", ax =
ax[0])
ax[0].set_title("Number Of Passengers By Pclass")
ax[0].set_ylabel("Population")
sns.countplot(x="Pclass", hue = "Survived", data = df, ax =
ax[1])
ax[1].set_title("Pclass: Survived vs Dead")
plt.show()

print(df['Pclass'].value_counts())
```

Program output:

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

- most passengers were in 3rd class
- most passengers were in 2nd class

- most passengers were in 1st class
- fewest passengers were in 2nd class
- fewest passengers were in 1st class
- fewest passengers were in 3rd class
- most of the 3rd class passengers did not survive the crash
- most of the 3rd class passengers survived the crash
- most of the 1st class passengers did not survive the disaster

5.1.8

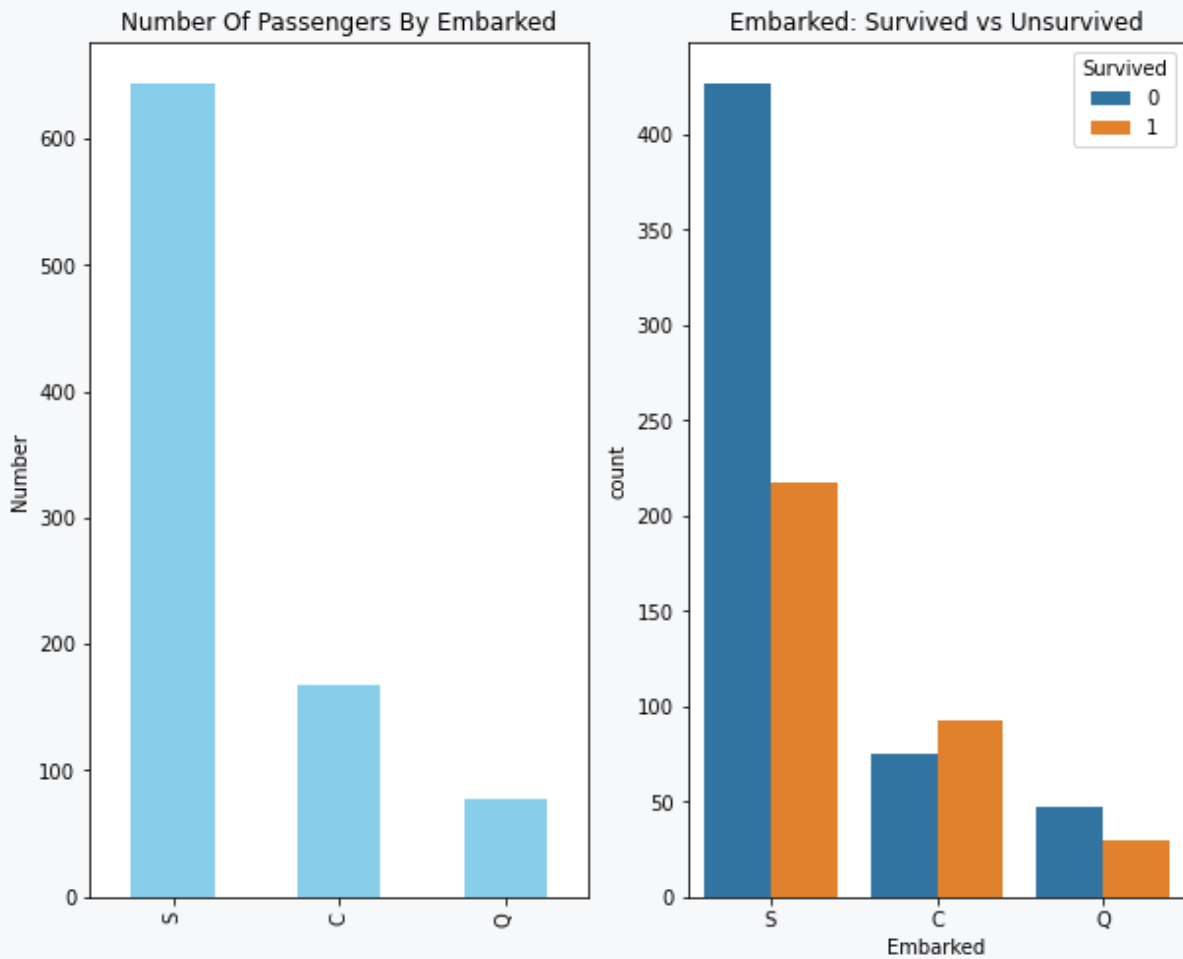
Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and find the distribution of the number of passengers by embarkation point. Write the result in numbers and in the following format:

```
S: 459, C: 232, Q: 120
```

```
# import library
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
# explore dataset
fig, ax = plt.subplots(1, 2, figsize = (10, 8))
df["Embarked"].value_counts().plot.bar(color = "skyblue", ax =
ax[0])
ax[0].set_title("Number Of Passengers By Embarked")
ax[0].set_ylabel("Number")
sns.countplot(x="Embarked", hue = "Survived", data = df, ax =
ax[1])
ax[1].set_title("Embarked: Survived vs Unsurvived")
plt.show()

print(df['Embarked'].value_counts())
```


Program output:



```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

5.1.9

Based on a review of passenger class distribution, examine the distribution of survivors by embarkation location. Decide which statements are true. You can help by visualizing using a bar graph.

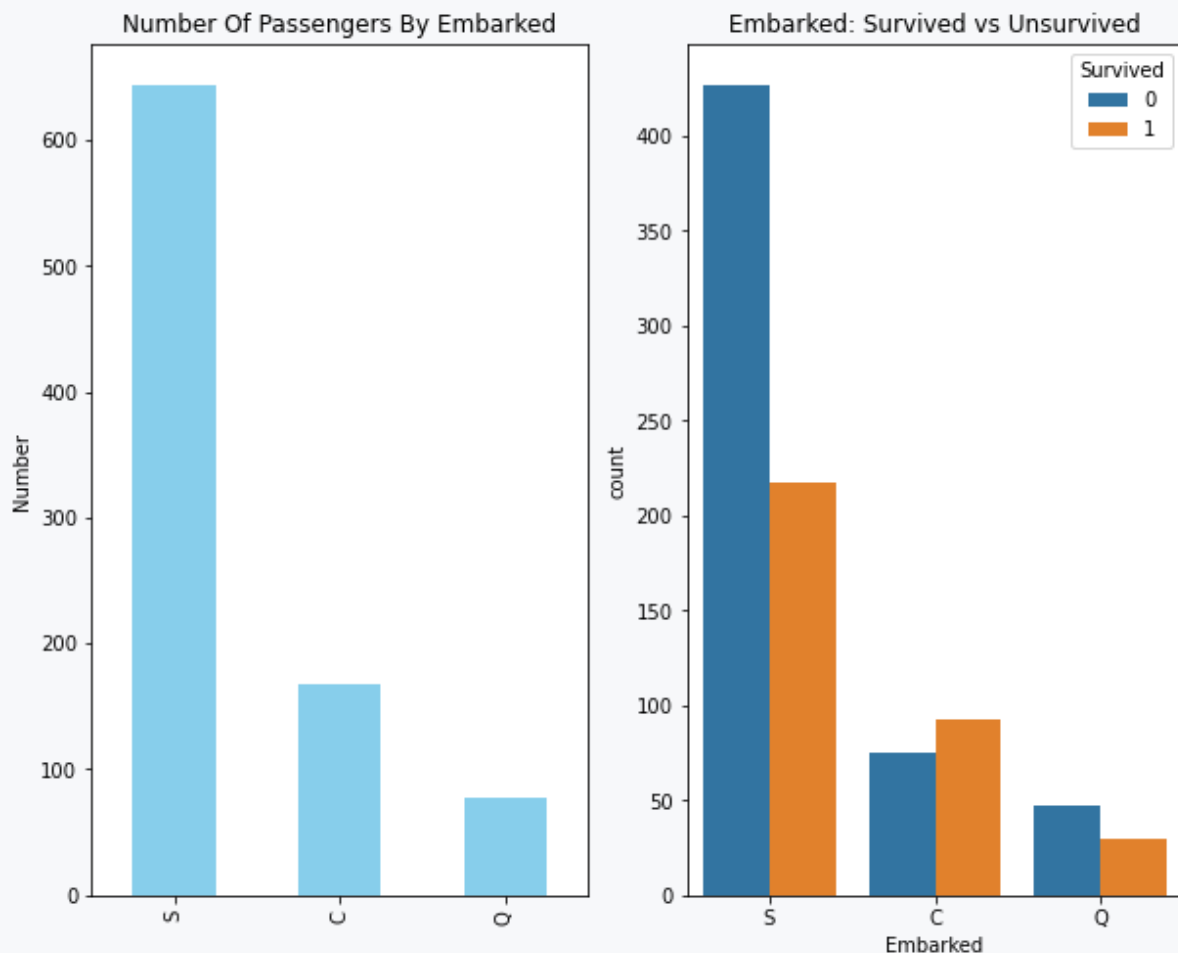
```
# import library
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
```

```

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.csv', sep=',')
# explore dataset
fig, ax = plt.subplots(1, 2, figsize = (10, 8))
df["Embarked"].value_counts().plot.bar(color = "skyblue", ax =
ax[0])
ax[0].set_title("Number Of Passengers By Embarked")
ax[0].set_ylabel("Number")
sns.countplot(x="Embarked", hue = "Survived", data = df, ax =
ax[1])
ax[1].set_title("Embarked: Survived vs Unsurvived")
plt.show()

```

Program output:



- most passengers boarded at Southampton
- more than half of the passengers boarded at Southampton did not survive the crash
- only the passengers who embarked at Cherbourg survived more than died
- fewest passengers boarded in Queenstown

- most passengers boarded in Queenstown
- fewest passengers boarded in Cherbourg
- most passengers embarked in Cherbourg
- more than half of the passengers embarked at Cherbourg did not survive the disaster

5.1.10

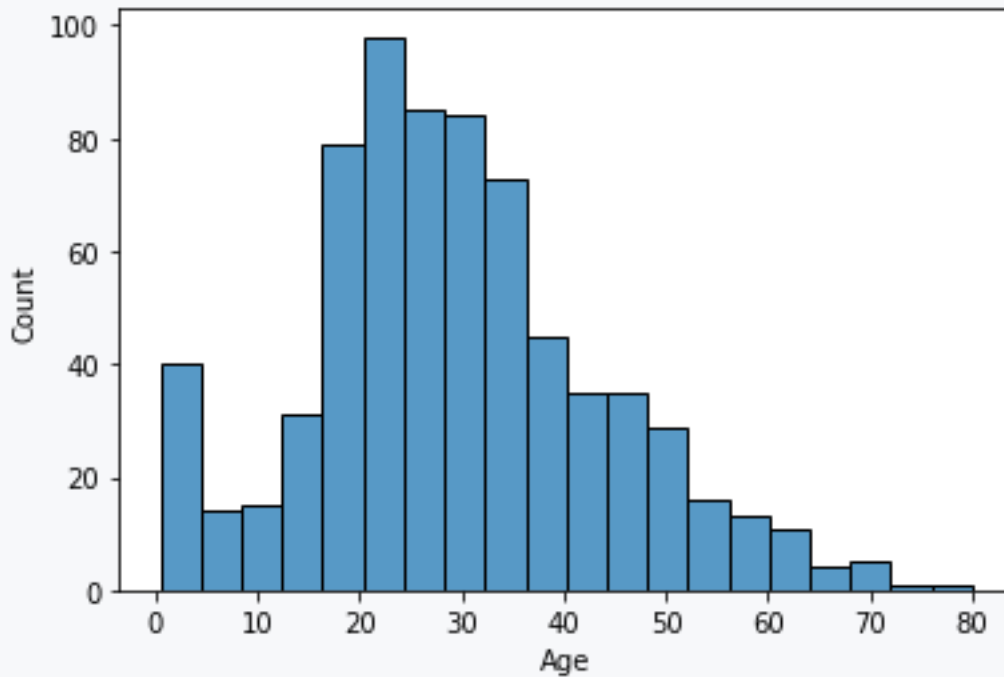
Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and find out the age distribution of the passengers. Write the most numerous age category in the following format (we recommend visualizing it as a histogram):

40-45

```
# import library
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.csv', sep=',')
# explore dataset
sns.histplot(df['Age'].dropna())
```

Program output:

```
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
      ..
36.50     1
55.50     1
 0.92     1
23.50     1
74.00     1
Name: Age, Length: 88, dtype: int64
```



5.1.11

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and see if there is a correlation between age and whether or not the passenger survived the crash. Write whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
# import library
import pandas as pd
from scipy import stats
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
dfd = df[['Age', 'Survived']].dropna()
# explore dataset
corr = stats.pearsonr(dfd['Age'], dfd['Survived'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:    0.04
cor:       -0.08
```

 5.1.12

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset to see if there is a correlation between `class` and whether or not the passenger survived the crash. Write whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
# import library
import pandas as pd
from scipy import stats
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
dfd = df[['Pclass', 'Survived']].dropna()
# explore dataset
corr = stats.pearsonr(dfd['Pclass'], dfd['Survived'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:    0.0
cor:       -0.34
```

 5.1.13

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and see if there is a correlation between the number of siblings (`Sibsp`) and whether or not the passenger survived the crash. Write whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
# import library
import pandas as pd
from scipy import stats
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
dfd = df[['SibSp', 'Survived']].dropna()
# explore dataset
corr = stats.pearsonr(dfd['SibSp'], dfd['Survived'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:      0.29
cor:         -0.04
```

 5.1.14

Load the data from the dataset `titanic.csv` (the file is located at `https://priscilla.fitped.eu/data/pandas/titanic.csv`). Examine the data in the dataset and see if there is a correlation between the number of children (`Parch`) and whether or not the passenger survived the crash. Write whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
# import library
import pandas as pd
from scipy import stats
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
dfd = df[['Parch', 'Survived']].dropna()
# explore dataset
corr = stats.pearsonr(dfd['Parch'], dfd['Survived'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:    0.01
cor:       0.08
```

 5.1.15

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and see if there is a correlation between the ticket price and whether or not the passenger survived the disaster. Write whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
# import library
import pandas as pd
from scipy import stats
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.c
sv', sep=',')
dfd = df[['Fare', 'Survived']].dropna()
# explore dataset
corr = stats.pearsonr(dfd['Fare'], dfd['Survived'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:    0.0
cor:       0.26
```

 5.1.16

Load the data from the dataset `titanic.csv` (the file is located at <https://priscilla.fitped.eu/data/pandas/titanic.csv>). Examine the data in the dataset and see if there is a correlation between the embarkation point and whether or not the passenger survived the disaster. The embarkation variable must be transformed into numerical values before analysis. Write whether there is a statistically significant relationship between the variables (yes/no) and the correlation value rounded to 2 decimal places and the p-value.

```
no, p-value: 0.12, cor: 0.45
```

```
# import library
import pandas as pd
from scipy import stats
# read csv from
https://priscilla.fitped.eu/data/pandas/titanic.csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/titanic.csv', sep=',')
dfd = df[['Embarked', 'Survived']].dropna()
dfd['Embarked'] = dfd['Embarked'].map({"S":1,
"C":2, "Q":2, "NaN":0})
# explore dataset
corr = stats.pearsonr(dfd['Embarked'], dfd['Survived'])
print("p-value:\t", round(corr[1],2))
print("cor:\t\t", round(corr[0],2))
```

Program output:

```
p-value:      0.0
cor:         0.15
```

5.1.17

Based on the results obtained from the data analysis, select the passenger characteristics that have an impact on disaster survival.

- Age
- Pclass
- Sibsp
- Parch
- Fare
- Embarked

Summarisation

Introduction

Chapter 6

6.1 What is summarisation

6.1.1

The explosion of electronic documents on the internet has made information more accessible than ever. However, the sheer length of many of these documents can make them difficult to digest quickly. This is where the field of **Natural Language Processing** (NLP) steps in, particularly with algorithms designed for text summarization. NLP, a branch of artificial intelligence, focuses on enabling computers to understand and process human language. One important task in NLP is **summarization**, which condenses lengthy text into shorter, coherent summaries without losing essential information. Summarization has become essential, allowing users to extract key information from documents swiftly.

NLP encompasses a variety of tasks that make human language processing easier for computers. Some of these tasks include analyzing the grammatical properties of texts, translating languages, and text autocompletion (as seen in search engines). Summarization is one of the more complex tasks, as it involves distilling a large body of information into a brief but complete version. This function is especially helpful in handling research articles, reports, and other long documents. Summarization can range from condensing a text to key sentences or phrases, down to a coherent paragraph that captures the main points.

A subfield of NLP, **automatic summarization**, allows for rapid and reliable extraction of the most relevant information from texts. This capability is commonly employed on news websites to provide quick article previews, helping readers grasp essential points without reading the entire text. However, automatic summarization is challenging. Unlike humans, computers do not "read" text in the same way, making it a difficult process to achieve both coherence and informativeness. Yet, with NLP advancements, automatic summarization continues to evolve, providing practical solutions to information overload.

6.1.2

Select tasks typical of natural language processing

- determining the parts of speech in the text
- machine translation of documents
- summarization of the text
- image processing

6.1.3

NLP allows computers to process human language, enabling tasks like translation, _____, and text _____. Summarization is one of the tasks that requires complex algorithms to convert long documents into _____ summaries.

- generation
- analysis
- coherent

6.1.4

Text summarization can significantly simplify the process of understanding long materials, such as research papers and reports, by condensing them into essential points. With **automatic text summarization**, it is possible to reduce extensive texts to a shorter form, be it a few sentences, paragraphs, or key phrases, without losing core information. This approach not only saves time but also helps readers to focus on the most important parts of a text. Text summarization has become an essential tool in NLP, where the goal is to produce a summary that contains all the critical details of the original document while taking up less space.

The essence of text summarization is in creating a **summary**, which is a shortened version of the text that preserves the original meaning and key points. An effective summary captures important information from one or more documents and usually should not exceed half the original document's length. The process can be applied in many domains, from academic research to media, where article snippets are generated automatically to give readers a quick overview. Summarization can be either **extractive** (taking parts of the original text directly) or **abstractive** (creating new sentences based on understanding the text), each with its benefits and challenges.

Automatic summarization is challenging because it requires algorithms to select, arrange, and present information coherently. While humans can read and interpret meaning, computers need structured approaches to ensure they capture essential ideas accurately. Nevertheless, advancements in NLP techniques continue to improve the quality of automatic summaries, making it a valuable asset in many fields, especially where quick comprehension of complex information is necessary.

6.1.5

What are benefits of text summarization?

- Reduces the length of texts without losing essential information
- Saves time for readers by providing condensed information
- Increases the original text length for better comprehension
- Adds redundant details to the summary

6.1.6

Summarization condenses long texts, helping readers grasp essential information. An effective summary retains the ____ meaning of the document, making sure all ____ details are kept, while removing less ____ parts.

- important

- core
- critical

6.2 Quality and challenges

6.2.1

Quality criteria

In text summarization, maintaining quality is essential to ensure that the summary captures the most relevant points of the original document. The quality of a summary can be judged by specific criteria: **information coverage**, **coherence**, **redundancy minimization**, and **brevity**. Each of these criteria plays a role in creating summaries that are not only concise but also meaningful. For example, **information coverage** is a measure of how much important information the summary retains from the original text.

Coherence is another critical factor, as it determines how logically and smoothly the sentences in the summary flow. A coherent summary helps readers to understand the relationships between ideas and follow the main points without confusion. **Minimizing redundancy** involves avoiding repetitive information, which can make summaries lengthy and less effective. Finally, **brevity** measures the length of the summary relative to the original text; an ideal summary is short yet packed with meaningful information, ensuring no unnecessary details are included.

By following these quality criteria, automated summarization systems aim to produce summaries that are both informative and compact. Each criterion contributes to the overall effectiveness of the summary, whether it is being used in research, news articles, or general information processing. Striking a balance between these criteria ensures that users receive a high-quality, easily understandable summary.

6.2.2

What is meant by "information coverage" in summarization?

- The summary contains all essential information from the original text
- The summary is short and lacks details
- The summary repeats key points multiple times
- The summary only covers minor points

6.2.3

Summaries should be _____ to keep essential points clear. They also need _____ to connect ideas smoothly and should avoid _____, which adds unnecessary length.

- redundancy

- brevity
- coherence

6.2.4

Challenges

Automatic summarization is a highly challenging task within NLP due to the difficulty of emulating human comprehension. When a person summarizes text, they rely on an in-depth understanding of context and meaning, which allows them to capture essential points accurately. However, computers cannot interpret language as intuitively as humans, making it difficult for algorithms to achieve a summary with similar coherence and depth. Summarization algorithms must therefore be carefully designed to handle complex tasks, such as selecting relevant information and arranging it meaningfully.

One primary challenge in automatic summarization is **maintaining coherence**, as computers may struggle to establish logical flow when sentences are reordered or rephrased. Furthermore, it can be difficult for algorithms to decide which details are essential without human-like understanding. Algorithms can end up with summaries that are either too brief and miss key information or too detailed and redundant.

Despite these challenges, advancements in NLP have led to improved methods for automatic summarization. With more sophisticated algorithms, summarization tools can now provide summaries that are more coherent, concise, and useful. However, the task remains complex, as each document presents unique content and context that requires careful processing for effective summarization.

6.2.5

What are challenges in automatic summarization?

- Achieving coherence in sentence structure
- Emulating human comprehension
- Repeating information to maintain context
- Increasing summary length for more detail

Extraction and Abstraction

Chapter **7**

7.1 Extraction

7.1.1

Approaches to summarization – extraction vs. abstraction

Summarization is an essential technique for simplifying lengthy documents and extracting key insights without losing critical information. In the context of Natural Language Processing (NLP), two main approaches to summarization are widely used: extraction and abstraction. Extractive summarization focuses on identifying and reordering the most relevant sentences directly from the source text to form a shorter version. Unlike abstraction, extractive summarization does not create any new sentences; instead, it uses pre-existing content. In contrast, abstraction takes a different approach by creating new sentences that capture the essential meaning of the original document.

Extractive summarization is typically more straightforward since it relies on identifying and selecting the most informative sentences. This is achieved through algorithms that rank sentences based on relevance, often using statistical or machine learning techniques to assess which parts of the text contain the most important information. Abstractive summarization, however, is more complex as it requires the generation of new text, demanding a higher level of linguistic and semantic understanding, often involving deep learning models.

Each method has distinct advantages and limitations. Extractive summaries maintain the original text structure, which may sometimes lack coherence in the summary. Abstractive summarization, while potentially more coherent, can be challenging due to the computational requirements and need for deep semantic comprehension. Both approaches are vital for various applications, from news aggregation to academic research.

7.1.2

Choose the correct statements about extractive summarization

- decides which sentences from the text are significant and need to be included in the summary
- the summarized text consists only of the sentences that were in the original text
- uses the concept of sentence scoring
- the summarized text also consists of new sentences that were not in the original text
- does not use the concept of sentence punctuation

 7.1.3

Which summarization approach involves generating new sentences?

- Abstraction
- Extraction
- Compression
- Rephrasing

 7.1.4

TF-IDF Method

Extractive summarization aims to rank sentences by relevance and extract only the most informative ones. One common technique is TF-IDF (Term Frequency-Inverse Document Frequency), a statistical measure used to identify significant words within a document. The “term frequency” component of TF-IDF calculates how frequently a word appears in the document. However, commonly used words, like “the” and “and,” often appear frequently across texts and add little to the summary. The “inverse document frequency” component of TF-IDF addresses this by reducing the importance of commonly used words and emphasizing terms unique to the document.

To calculate TF-IDF, each word’s frequency in the document is compared against its frequency in a larger set of documents, adjusting its significance accordingly. Words with high TF-IDF scores tend to be unique and relevant to the document, which helps algorithms identify which sentences should be extracted into a summary.

For extractive summarization, TF-IDF helps isolate sentences rich in high-score words. These sentences are then reassembled into a summary. This approach, although simple, is effective in creating summaries that retain important terms and concepts while discarding redundant information.

 7.1.5

Which of the following are components of the TF-IDF method?

- Term frequency
- Inverse document frequency
- Sentence length
- Sentence position

7.1.6

Graph-based methods

Graph-based methods for extractive summarization utilize a visual structure to represent relationships among sentences in a document. Here, sentences are represented as nodes within a graph, while edges connect sentences that share common words or phrases. The more connections a sentence node has, the more relevant it is considered, making it more likely to be included in the summary.

The algorithm typically starts by calculating a similarity score for each sentence, connecting sentences with overlapping terms. Sentences with the highest degree of connectivity (those with the most edges) are chosen for the summary. This approach is especially effective in maintaining coherence, as connected sentences tend to flow better in the final summary.

Graph-based methods are beneficial for extracting key content in documents where sentences are highly interconnected. An example of this approach is the TextRank algorithm, which is widely used in NLP for summarization and other tasks like keyword extraction.

7.1.7

In a graph-based summarization method, what do the nodes represent?

- Sentences
- Keywords
- Paragraphs
- Documents

7.1.8

Machine learning-based summarization

Machine learning methods treat summarization as a classification problem, where sentences are labeled as “summary” or “non-summary” sentences. During training, a model learns to recognize characteristics that typically belong to sentences in a summary. Features such as sentence length, position, and relevance to the document’s title are often used to train the classifier.

In practice, machine learning-based summarization requires a training dataset with examples of summary and non-summary sentences. Once trained, the classifier can identify and select sentences that are likely to be relevant, making the summarization process more accurate and adaptable to different types of content.

These models can be trained on labeled datasets and refined over time. As more data is used for training, the classifier becomes better at distinguishing between significant and insignificant information.

 7.1.9

Which features are typically used in machine learning-based summarization?

- Sentence position
- Relevance to title
- Language complexity
- Paragraph length

 7.1.10

What is the main goal of machine learning-based summarization?

- To classify sentences as summary or non-summary sentences
- To predict document length
- To extract keywords only
- To identify all repeated phrases

 7.1.11

Fuzzy logic in summarization

Fuzzy logic-based summarization methods evaluate sentences based on various characteristics, such as length, similarity to the title, and presence of keywords. Unlike strict binary logic, fuzzy logic assigns sentences a score between 0 and 1, indicating their degree of importance. This flexibility allows sentences with varying levels of relevance to be included in the summary based on defined rules.

For instance, sentences containing keywords relevant to the document's title may score higher, while sentences with excessive detail score lower. Fuzzy logic's adaptability makes it particularly useful for dynamic summaries, where multiple factors influence sentence selection. The scoring mechanism also helps in minimizing redundancy, ensuring that only the most unique and relevant sentences are included.

This approach is suitable for complex documents, where simple extraction methods might miss nuances. Fuzzy logic systems can be fine-tuned to prioritize sentences that best represent the document's overall theme.

 7.1.12

Which method assigns importance scores to sentences between 0 and 1?

- Fuzzy logic
- Graph-based methods
- Neural networks
- TF-IDF

7.2 Abstraction I.

7.2.1

Abstract summarization is a sophisticated technique in natural language processing (NLP) that aims to generate a coherent summary by creating new sentences. Unlike extractive summarization, which pulls directly from the original text, abstract summarization involves interpreting and paraphrasing the content, similar to how a human might summarize. This approach begins with generating a transient representation of the text, identifying primary topics and "indicators," such as sentence length and the presence of specific key terms, to determine the text's most important sections. The sentences with the highest scores are selected to build the summary.

This summarization approach is more complex than extractive summarization, as it requires a deep understanding of the text's semantics—the meaning and relationships between ideas, concepts, and topics within the document. To achieve this, advanced NLP techniques are employed to "read" the document in context and synthesize information into new, concise sentences, effectively rephrasing the source material without losing key information.

Various methods can be used in abstract summarization, each offering unique advantages in achieving coherence and informativeness. By creating summaries that are not limited to direct quotes from the text, abstract summarization is particularly useful for applications like news generation, academic research, and digital assistants, where a fluent and original summary is required.

7.2.2

Choose the correct statements about abstract summarization

- the created summary contains only the sentences that were in the original text
- it relies entirely on the concept of sentence scoring
- uses the indicator representation to express the importance of individual parts of the text
- the summary contains sentences that were not in the original text

7.2.3

Which elements are considered in creating a transient representation for abstract summarization?

- Main topics
- Indicator words
- Sentence punctuation
- Page length

7.2.4

Tree-based methods

Tree-based methods for abstract summarization use a dependency tree to analyze and structure the content of the document. In this approach, sentences are broken down into their grammatical components to reveal relationships between words. A notable technique is "sentence fusion," where sentences across multiple documents are combined to create a cohesive summary. This method is advantageous for summarizing extensive documents or combining information from various sources.

The dependency tree enables the summarizer to organize and fuse similar sentences, ensuring that the final summary contains only unique, relevant information. By examining sentence structures, tree-based methods can identify the core ideas across documents, simplifying complex information and removing redundancies.

Tree-based summarization is highly effective in environments requiring precision and clarity, such as legal documents, research papers, or medical reports. The dependency tree helps ensure that the summary retains its logical structure, making the information easier to understand.

7.2.5

What is the purpose of using a dependency tree in tree-based methods?

- To analyze relationships between words
- To select random sentences
- To count words
- To calculate term frequency

7.2.6

What are the benefits of tree-based methods in summarization?

- Combining information from multiple sources
- Organizing sentences for clarity
- Increasing document length
- Selecting text based on page layout

7.2.7

Template-based methods

Template-based methods approach summarization by mapping specific patterns and extraction rules within a document to a template structure. The system uses linguistic patterns to identify relevant text snippets, which are then matched to pre-

designed template "slots" or placeholders, resulting in a concise database-driven summary. For instance, in a business report, templates might include categories such as "Introduction," "Key Findings," and "Conclusion," with content from the document inserted into each relevant section.

This method is especially useful for summarizing repetitive or standardized content, where specific types of information consistently appear. Template-based methods can improve the efficiency of summarization processes, especially when dealing with highly structured data.

These methods are ideal in fields like finance or healthcare, where standardized reporting is essential, as they ensure critical details are included while reducing the need for manual input. However, template-based summaries may lack flexibility for unstructured or highly varied content, as they rely on predefined patterns.

7.2.8

What does a template-based method in summarization primarily rely on?

- Predefined patterns and rules
- Sentence length
- Word frequency
- Sentence complexity

7.2.9

What are advantages of template-based methods?

- Standardization of summaries
- Efficiency in structured data summarization
- Flexibility in unstructured content
- Emphasis on informal language

7.2.10

Rule-based methods

Rule-based methods in abstract summarization operate by defining rules that classify sentences into categories, such as "summary" or "non-summary." These rules focus on the use of verbs, nouns, and phrases with similar meanings, which help the system recognize important content. A set of extraction rules is created, enabling the summarizer to select sentences that best capture the essence of the document.

This approach is well-suited for specific domains where certain linguistic patterns regularly appear, making it easier to create reliable rules. For instance, in scientific articles, rules might prioritize sentences with terms like "results" or "conclusions" to capture essential findings.

While rule-based methods offer precise summaries in controlled environments, their effectiveness may be limited when applied to general text with unpredictable sentence structures or vocabulary. However, they remain a valuable tool for generating domain-specific summaries that require accuracy and consistency.

7.2.11

Rule-based methods in summarization work by:

- Applying predefined linguistic rules
- Randomly selecting sentences
- Only selecting the first sentences
- Ignoring all verbs and nouns

7.2.12

What advantages do rule-based summarization methods provide?

- Precision in domain-specific summaries
- Consistency across similar documents
- Adaptability to all types of documents
- Random selection of sentences

7.3 Abstraction II.

7.3.1

Ontology-based methods

Ontology-based methods leverage domain-specific knowledge to create summaries, making them particularly useful when summarizing content within a specific field. An ontology is a structured representation of knowledge, where concepts are organized and linked based on relationships relevant to the domain. In ontology-based summarization, sentences are processed to ensure they align with the defined concepts, and relationships in the ontology. For example, in a medical ontology, terms like "symptoms," "diagnosis," and "treatment" may guide the summarization process.

This method compresses and reformulates sentences based on domain-relevant criteria, using both linguistic and NLP techniques. By following a structured knowledge base, ontology-based methods ensure that the summary focuses on the most critical and relevant details, maintaining coherence and context.

Ontology-based summarization excels in areas like scientific literature and technical fields where precise terminology and concept relationships are essential. However, it may require extensive resources to develop domain-specific ontologies, making it best suited for specialized applications.

 7.3.2

In which situation is an ontology-based summarization method most useful?

- Summarizing domain-specific content
- Summarizing fiction books
- Summarizing random internet articles
- Selecting keywords for general essays

 7.3.3

What are key aspects of ontology-based summarization?

- Use of domain knowledge
- Structured relationships between concepts
- Emphasis on unrelated sentences
- Lack of specialized terminology

 7.3.4

Combined approaches

Combined summarization is an advanced method that integrates both extractive and abstractive techniques to create concise, coherent summaries. In this approach, an extractive summarizer is first used to strip the original text of redundant or irrelevant information, isolating the most critical sentences. This “cleaned” text is then passed to an abstract generator, which rephrases and reorganizes the selected content into a more fluent and cohesive summary. By using extraction as a preliminary step, the abstract generator can focus solely on rephrasing important details, resulting in a more accurate and meaningful summary.

One of the key advantages of combined summarization is its ability to work efficiently with complex documents, as it minimizes processing time by dealing only with relevant information. This approach also reduces potential errors in abstraction, as it prevents the generator from paraphrasing unnecessary details, which could lead to inaccuracies. Combined summarization is particularly valuable in applications requiring high precision and readability, such as news summarization, where both relevance and fluency are critical.

Modern algorithms, such as the BART (Bidirectional and Auto-Regressive Transformers) model, employ combined summarization techniques to deliver high-quality summaries. These algorithms are trained to process and refine extracted content through sophisticated language modeling, making combined summarization one of the most effective approaches in text summarization today.

 7.3.5

Which aspects characterize combined summarization?

- Integration of extractive and abstractive methods
- Use of an abstract generator on pre-selected text
- Ignoring redundant information
- Repeating important sentences

 7.3.6

Multimodal semantic model

The Multimodal Semantic Model is a sophisticated summarization technique that utilizes an object-based knowledge representation to create structured summaries. In this approach, text is broken down into nodes and links. Nodes represent individual concepts, while links show the relationships between these concepts, forming a network of interconnected ideas. This structure allows the summarizer to capture the core ideas within the text in an organized manner, enhancing the readability and cohesion of the resulting summary.

An information density metric is used to evaluate and score the significance of concepts within the text. This metric considers several factors, such as the completeness of ideas, the relationships between nodes, and the frequency of term occurrences. Important concepts are prioritized based on these scores, which ensures that the summary includes the most relevant and essential information without redundancy.

Once key concepts are identified, they are transformed into coherent sentences that form the final summary. By focusing on concept relationships and term significance, the Multimodal Semantic Model excels at producing summaries that are not only concise but also maintain the intended meaning of the original text, making it highly useful for applications that require precise and detailed information retention.

 7.3.7

In the Multimodal Semantic Model, what do the links represent?

- Relationships between concepts
- Individual terms
- Sentence scores
- Irrelevant information

 7.3.8

What factors does the information density metric consider in the Multimodal Semantic Model?

- Completeness of ideas
- Frequency of term occurrences
- Sentence length
- Grammar and syntax

 7.3.9

Semantic text representation model

The Semantic Text Representation Model is a summarization approach that focuses on the semantic content of words rather than their syntactic arrangement. Unlike traditional methods that rely on word positioning or structure, this model delves into the meaning behind words and phrases, analyzing the inherent concepts within the text. By interpreting semantic relationships, this model generates summaries that reflect the intended message of the original material more effectively.


In practice, the Semantic Text Representation Model works by identifying key concepts and ideas and examining the connections between them. These connections help the summarizer understand the underlying themes of the text, making it possible to condense content while preserving its core meaning. This method is especially beneficial when summarizing texts that contain nuanced language or multiple layers of meaning.

The Semantic Text Representation Model is particularly valuable in fields that require a deep understanding of context, such as legal or academic texts. It prioritizes semantic understanding over mere word frequency, which enables it to produce summaries that are accurate and contextually relevant, capturing both the explicit and implicit information within a document.

 7.3.10

What is the main focus of the Semantic Text Representation Model?

- Understanding the meaning behind words
- Counting sentence length
- Using only sentence structure
- Relying on word frequency alone

 7.3.11

Which features are central to the Semantic Text Representation Model?

- Focus on semantics of words
- Preservation of core meaning
- Only syntax and structure
- Surface-level word frequency

Keyword Extraction

Chapter **8**

8.1 Introduction

8.1.1

Keyword extraction is a summarization technique designed to capture essential words or phrases from a document, making it easier to understand the document's core themes without reading it in full. This process identifies a small but representative set of terms, known as key phrases, which succinctly convey the document's main ideas. Keyword extraction is widely applied in content management fields, including search engine optimization, advertising, and recommendation systems, to ensure relevant information is highlighted. For instance, when a user encounters an ad or webpage with well-selected keywords, they are more likely to find the content useful and engaging.

The primary goal of keyword extraction is to save time by efficiently conveying the main ideas within extensive text. Key phrases streamline the information for the end user, which is particularly valuable in our fast-paced, data-saturated environment. As a result, keyword extraction has become a fundamental tool for professionals across industries, especially in contexts where quickly grasping content is critical.

There are two main approaches to keyword extraction: simple statistical methods and machine learning-based techniques. Statistical methods use frequency and distribution patterns to select keywords, while machine learning methods apply trained models to recognize important phrases. By leveraging both approaches, systems can deliver accurate and contextually relevant keyword sets, improving content management and search efficiency.

8.1.2

Which field frequently uses keyword extraction for better content management?

- Search engine optimization
- Financial reporting
- Legal drafting
- Physics simulations

8.1.3

What are some common applications of keyword extraction?

- Content management
- Recommendation systems
- Real estate pricing
- Network security analysis

8.1.4

Types of keyword extraction approaches

Keyword extraction techniques are generally divided into two primary categories: simple statistical approaches and machine learning-based approaches. Statistical approaches focus on identifying keywords based on word frequency, co-occurrence, and other straightforward metrics. This means that frequently appearing or highly associated words are more likely to be selected as keywords. These methods are computationally efficient, making them popular in scenarios where quick keyword extraction is needed without extensive processing.

Machine learning-based approaches, on the other hand, involve training algorithms to recognize keywords based on patterns within a training dataset. These models are typically more accurate than simple statistical methods because they are tailored to specific contexts and learn from annotated data to make refined predictions. Machine learning approaches can identify keywords even in complex language structures by using labeled examples that guide the model to discern important terms.

In addition to these two categories, several hybrid approaches combine both statistical and machine learning techniques, resulting in more precise keyword extraction. For instance, hybrid methods may use statistical filters to pre-select potential keywords, then apply machine learning to fine-tune the list. By combining strengths from both approaches, hybrid methods offer a balanced solution that provides both speed and contextual accuracy.

8.1.5

Which of the following are main types of keyword extraction approaches?

- Simple statistical approaches
- Machine learning-based approaches
- Mathematical modeling
- Heuristic sorting

8.1.6

What do machine learning-based approaches in keyword extraction rely on?

- Pattern recognition in training data
- Frequency of stop words
- Random word selection
- Simple word counting

8.1.7

Preprocessing

Preprocessing text is an essential step in preparing documents for keyword extraction, as it cleans and standardizes the text for optimal results. One of the first steps in preprocessing is the removal of stop words—common words like “and,” “the,” and “in” that do not carry specific meaning. Stop words, although frequently occurring, do not contribute to the core message of a text and can clutter keyword extraction if left unfiltered. Many libraries, such as the Natural Language Toolkit (NLTK) in Python, provide extensive lists of stop words for different languages, simplifying this step.

Another crucial preprocessing step is converting all text to lowercase. This step prevents keywords from being duplicated due to case sensitivity, where “Keyword” and “keyword” might both be considered unique keywords. By converting the entire text to lowercase, we ensure a consistent format, enabling algorithms to identify keywords more accurately.

Finally, the removal of punctuation and special characters is necessary, especially when processing informal text such as social media content. Characters like hashtags, emojis, or unusual symbols often carry little to no semantic meaning and could distort keyword extraction if included. These preprocessing steps work together to enhance keyword extraction by ensuring only meaningful words are considered, resulting in cleaner, more relevant keywords.

8.1.8

What are some standard preprocessing steps for keyword extraction?

- Removing stop words
- Lowercasing text
- Increasing punctuation frequency
- Doubling key phrases

8.2 Statistical approaches

8.2.1

Simple statistical approaches in keyword extraction focus on analyzing word frequency and distribution within a text. These methods operate under the assumption that frequently occurring words or terms that appear together are likely to be essential to the text’s meaning. One common statistical method is Term Frequency-Inverse Document Frequency (TF-IDF), which evaluates a word’s importance based on how often it appears in a document relative to its occurrence across a set of documents. This way, words that are significant within the context of the document are highlighted as potential keywords.

Statistical approaches are particularly effective for quick, general keyword extraction since they rely on straightforward calculations rather than complex algorithms. For example, TF-IDF can be applied to identify terms that are unique to a particular article within a large set of documents, helping distinguish specialized topics or unique themes.

While statistical methods are easy to implement and efficient in processing time, they may not fully capture the context or nuanced meaning of a text. This is where machine learning-based methods often outperform simple statistical methods by providing richer and more context-aware keyword selections.

8.2.2

Which are characteristics of statistical keyword extraction methods?

- Relies on frequency and co-occurrence
- Offers computational efficiency
- Requires complex training data
- Depends on image recognition

8.2.3

Statistical approaches extract keywords by using statistical functions such as TF-IDF (Term Frequency-Inverse Document Frequency), n-gram statistics, word co-occurrences, and other statistics. Most statistical approaches are language-independent, meaning that they can be used for texts in a language if a large enough corpus is available. In addition to applicability to active language, speed is an indisputable advantage of statistical approaches. algorithms are rather faster in contrast to approaches that are based on machine learning.

TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF is one of the most well-known possible approaches to find important words from a document. TF-IDF talks about the importance of the words in the document in relation to the entire corpus. It is already clear from the name of the approach that this approach is composed of two components, namely the TF component and the IDF component.

The TF (Term Frequency) component expresses how often (frequency) a given word occurs in a document from the corpus. it is usually normalized by dividing the document's word count to avoid overestimating long documents, where the search term may appear more often than shorter ones, without making the document more relevant. Therefore, we obtain the TF component according to the following, where the number of occurrences of the word t_i in the document is not d_j . The denominator expresses the sum of the number of occurrences of all words in the document.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

IDF (Inverse Document Frequency) talks about specific words. In principle, it can be said that the more often a word occurs in documents, the less important it is (a word that occurs in all documents, such as the English article "the" or the Slovak conjunction "a", is mostly unusable in searches). We calculate the IDF for the word i using the formula below, where $|D|$ represents the number of documents in which we search and $|\{j : t_i \in d_j\}|$ is the number of documents that contain the word i .

$$IDF_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

8.2.4

TF-IDF talks about the importance of the words in the document in relation to the entire corpus

- True
- False

8.2.5

RAKE (Rapid Automatic Keyword Extraction)

RAKE (Rapid Automatic Keyword Extraction) is a statistical approach that has gained significant popularity in keyword extraction due to its efficiency and simplicity. The core idea behind RAKE is that important keywords often consist of multiple words, such as "good camera" or "quality sound," rather than isolated single words. This approach is particularly useful when dealing with multi-word phrases, which are frequently seen in customer feedback, product reviews, and other domain-specific texts. Unlike many other methods, RAKE does not rely on word frequency alone but on the co-occurrence of words within the same context, which makes it particularly adept at extracting meaningful phrases.

The RAKE algorithm works by first removing stop words (e.g., "and," "the," "is") and punctuation from the text, ensuring that only content-carrying words remain. Once this preprocessing is complete, the algorithm builds a co-occurrence matrix, which calculates how often words appear together in the same context. For example, in a product review, phrases like "good camera" might frequently appear together, making them a strong candidate for keyword extraction. The co-occurrence matrix helps identify these significant word pairs (or n-grams), highlighting those combinations that occur most frequently within the text.

One of the key advantages of RAKE is its ability to efficiently process large texts without the need for complex training data or heavy computational resources. It

focuses on word co-occurrence patterns, making it highly effective in domains where context and relationships between terms are crucial. However, while RAKE is relatively simple, its performance can vary depending on the quality of the input text and the relevance of the co-occurring words. Despite this, RAKE remains a popular choice for extracting meaningful keywords in many real-world applications, such as customer feedback analysis, document summarization, and more.

The input to the algorithm is the text cleaned of trace words and punctuation. The algorithm then calculates the co-occurrence matrix.

	feature	extraction	complex	algorithm	available	help	rapid	automatic	keyword
feature	2	2	0	0	0	0	0	0	0
extraction	2	3	0	0	0	0	0	1	1
complex	0	0	1	0	0	0	0	0	0
algorithm	0	0	0	1	1	0	0	0	0
available	0	0	0	1	1	0	0	0	0
help	0	0	0	0	0	1	0	0	0
rapid	0	1	0	0	0	0	1	1	1
automatic	0	1	0	0	0	0	1	1	1
keyword	0	1	0	0	0	0	1	1	1
TOTAL SUM	4	8	1	2	2	1	3	4	4

2 candidate key phrases for the word feature

Each word is then assigned a score. The degree of the word in the matrix is calculated - the sum of the number of common occurrences divided by the frequency of their occurrence. Frequency of occurrence means how many times a word occurs in the corpus.

Word	Degree Of Word	Word Frequency	Degree Score
feature	4	2	2
extraction	8	3	2.66
complex	1	1	1
algorithm	2	1	2
available	2	1	2
help	1	1	1
rapid	3	1	3
automatic	4	1	4
keyword	4	1	4

The final score for the identified key phrases will be the sum of the scores of the individual words that the key phrase contains. So for the keyword phrase "feature extraction" the value will be equal to 4.66.

8.2.6

What is the basis of the RAKE algorithm?

- co-occurrences of words
- cosine similarity
- frequency of words in text

8.2.7

Project: Implementation of the RAKE algorithm

To implement the RAKE algorithm, we will first start the nltk library, rake-nltk. After installation, we can import the libraries. Lists of stop words are available on various websites. We could download any of them and implement it in our code as a letter. However, we can also use the list of stop words offered by the nltk library. In our case, we will show the extraction of keywords from simple text, which will be stored in a string variable.

We will have to tokenize this text into sentences, for which we will use the Punkt Sentence Tokenizer, which divides the text into a list of sentences. We have the following text: "Text summarization is a method which belongs to the area of Natural Language Processing. Keyword extraction is a process of obtaining the most important keywords in a document. Keyword extraction is usefull text summarization technique." Let's save this text as a string variable. Let's just convert this text to lowercase letters. Let's save a list of our stop words in the stop_words variable.

```
import nltk
from rake_nltk import Rake
nltk.download('stopwords')
from nltk.corpus import stopwords
nltk.download('punkt')

text = "Text summarization is a method which belongs to the
area of Natural Language Processing. Keyword extraction is a
process of obtaining the most important words in document.
Keyword extraction is usefull text summarization technique."
text = text.lower()

stop_words = nltk.corpus.stopwords.words('english')
```

Program output:

```
[nltk_data] Downloading package stopwords to
/home/johny/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package punkt to
/home/johny/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In the `rake_extractor` variable, we initialize the `Rake` class that will perform the extraction. The `stopwords` parameter specifies a list of words to be removed from the text. The range of n-grams, i.e. the number of words we want our keywords to contain, is determined by the `min_length` parameter, which defines the minimum number of words that phrases must contain, and the `max_length` parameter, which defines the maximum number of words. words that the extracted key phrases may contain. In our case, we want phrases that have exactly two words. The `include_repeated_phrases` parameter specifies whether we want the extracted keywords to be repeated in the result. We then call the function `extract_keywords_from_text` which will accept our variable named `text` as a parameter.

```
# Initialize the RAKE extractor
rake_extractor = Rake(stopwords = stop_words, min_length=2,
max_length=2, include_repeated_phrases=False)

# Extract keywords from the input text
rake_extractor.extract_keywords_from_text(text)

# Get the ranked phrases with their scores
ranked_phrases = rake_extractor.get_ranked_phrases()

# Output the ranked keyword phrases with their relevance
scores
print(ranked_phrases)
```

Program output:

```
['text summarization', 'keyword extraction', 'important
words']
```

and to get the keywords we will use the `get_ranked_phrases` or `get_ranked_phrases_with_scores` method depending on whether we want to see the rank scores for our keywords as well.

```
print(rake_extractor.get_ranked_phrases_with_scores())
```

Program output:

```
[(4.0, 'text summarization'), (4.0, 'keyword extraction'),
(4.0, 'important words')]
```

📖 8.2.8

YAKE

YAKE (Yet Another Keyword Extractor) is another advanced keyword extraction method that uses the **TF-IDF** technique. However, YAKE introduces a more nuanced approach to extracting keywords by incorporating several additional features that enhance the accuracy of keyword selection. YAKE stands out by using a combination of both **TF-IDF** scores and new statistical features to better capture the context and importance of candidate keywords in a document.

YAKE works by analyzing the **location** and **frequency** of candidate words within a document. The method uses **five key features** to calculate the importance of each word:

1. **WC (Word Case)**: This feature reflects the case of the candidate word. Words that appear in **uppercase** or as part of titles may be considered more important because they often represent key concepts or proper nouns.
2. **WP (Word Position)**: This factor emphasizes the position of the word in the document. Words that appear near the **beginning** of the document are given more importance because they often introduce main topics.
3. **WF (Word Frequency)**: This feature reflects how frequently a word appears within the document. Words that are mentioned **more often** are assumed to be more relevant to the overall context of the document.
4. **WRC (Word-Related Context)**: This feature measures the **relatedness** of a candidate word to other words in its context. If a word is surrounded by different or important words, it is considered to be more significant.
5. **WD (Word Distribution in Sentences)**: This feature looks at how often the candidate word appears across **different sentences**. Words that appear across multiple sentences are deemed to represent broader concepts and are therefore given higher scores.

By considering these five features in addition to the traditional TF-IDF score, YAKE can provide a more comprehensive and context-sensitive evaluation of keywords. This makes YAKE particularly useful in extracting keywords from documents with complex structures or varied content, where the position, frequency, and context of words play an important role in determining their significance.

These five values are combined to calculate $S(w)$ as shown in the formula below.

$$S(w) = \frac{WR+WP}{WC+\frac{WF+WD}{WRC+WR}}$$

Finally, the final $S(kw)$ of each candidate word is calculated using the 3-gram model as shown in the following equation, where kw represents the candidate word and TF represents the frequency of the key phrase. The smaller the value of $S(kw)$, the more likely it is that kw will be a key phrase.

$$S(kw) = \frac{\prod_{w \in kw} S(w)}{TF(kw) * (1 + \sum_{w \in kw} S(w))}$$

8.2.9

Project: Implementation of the YAKE algorithm

Implement the YAKE (Yet Another Keyword Extractor) algorithm for automatic keyword extraction from a given text document. The YAKE method calculates keyword scores based on several features like word frequency, position, case, and context in the document.

```
import string
import nltk
from collections import Counter
import math

# Download NLTK stopwords if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
```

Program output:

```
[nltk_data] Downloading package stopwords to
/home/johny/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
/home/johny/nltk_data...
[nltk_data] Package punkt is already up-to-date!
```

1. Preprocessing

- Converts text to lowercase to avoid case sensitivity issues.
- Removes punctuation to ensure that symbols do not affect the keyword extraction.
- Removes stopwords (common words like "the", "is", etc.) that don't add much value to the meaning of the text.

```
# Step 1: Preprocess the Text
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
```

```

# Remove punctuation
text = text.translate(str.maketrans('', '',
string.punctuation))

# Tokenize the text into words
words = word_tokenize(text)

# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word not in
stop_words]

return filtered_words, text

```

2. Calculate Term Frequency (TF)

- Counts the frequency of each word and normalizes it by dividing by the total number of words in the text

```

# Step 2: Calculate Term Frequency (TF)
def compute_tf(words):
    tf = Counter(words)
    total_words = len(words)
    tf_scores = {word: count / total_words for word, count in
tf.items()}
    return tf_scores

```

3. Word Position (WP):

- Calculates the average position of each word in the sentences of the document. Words that appear earlier in the document are given higher importance.

```

# Step 3: Calculate Word Position (WP)
def compute_wp(text, words):
    sentences = sent_tokenize(text)
    wp_scores = {}

    for word in words:
        positions = []
        for i, sentence in enumerate(sentences):
            if word in sentence.lower():
                positions.append(i)

```

```

        wp_scores[word] = sum(positions) / len(positions) if
positions else 0
    return wp_scores

```

4. Word Frequency (WF):

- Counts how many times each word appears in the document, higher frequency means higher importance.

```

# Step 4: Calculate Word Frequency (WF)
def compute_wf(words):
    word_count = Counter(words)
    return word_count

```

5. Word Related Context (WRC):

- Analyzes the context in which a word appears by looking at its co-occurrence with other words in the same sentence. More diverse context usually implies higher importance.

```

# Step 5: Calculate Word Related Context (WRC)
def compute_wrc(text, words):
    sentences = sent_tokenize(text)
    wrc_scores = {}

    for word in words:
        related_words = set()
        for sentence in sentences:
            if word in sentence.lower():
                sentence_words = set(word_tokenize(sentence))
                related_words.update(sentence_words)
        wrc_scores[word] = len(related_words)
    return wrc_scores

```

6. Word Distribution (WD):

- Measures how spread out a word is across different sentences. A word that appears in multiple sentences tends to be more relevant to the overall document.

```

# Step 6: Calculate Word Distribution (WD)
def compute_wd(text, words):
    sentences = sent_tokenize(text)

```



```

wd_scores = {}

for word in words:
    sentence_count = 0
    for sentence in sentences:
        if word in sentence.lower():
            sentence_count += 1
    wd_scores[word] = sentence_count / len(sentences)
return wd_scores

```

7. Rank Keywords:

- Combines all the computed scores and ranks the keywords based on their final scores.

```

# Step 7: Rank Keywords with YAKE Features
def yake_ranking(tf_scores, wp_scores, wf_scores, wrc_scores,
wd_scores):
    keyword_scores = {}

    for word in tf_scores:
        # Calculate a combined score based on all features
        (you can add weights here)
        score = (tf_scores[word] + wp_scores.get(word, 0) +
                wf_scores[word] + wrc_scores.get(word, 0) +
                wd_scores.get(word, 0))
        keyword_scores[word] = score

    # Rank keywords by score (higher is better)
    ranked_keywords = sorted(keyword_scores.items(),
key=lambda x: x[1], reverse=True)

    return ranked_keywords

```

- and run:

```

# Example Text
text = """
Keyword extraction is an important task in Natural Language
Processing.
The goal is to extract the most relevant words or phrases that
represent the content of the document.

```

```

YAKE is a keyword extraction technique that uses several
features to evaluate the importance of a word.
"""

# Preprocess text
filtered_words, processed_text = preprocess_text(text)

# Compute TF, WP, WF, WRC, and WD features
tf_scores = compute_tf(filtered_words)
wp_scores = compute_wp(processed_text, filtered_words)
wf_scores = compute_wf(filtered_words)
wrc_scores = compute_wrc(processed_text, filtered_words)
wd_scores = compute_wd(processed_text, filtered_words)

# Rank keywords using YAKE
ranked_keywords = yake_ranking(tf_scores, wp_scores,
wf_scores, wrc_scores, wd_scores)

# Display the ranked keywords
print("Ranked Keywords:")
for word, score in ranked_keywords:
    print(f"{word}: {score:.4f}")

```

Program output:

```

Ranked Keywords:
keyword: 36.0800
extraction: 36.0800
important: 35.0400
task: 35.0400
natural: 35.0400
language: 35.0400
processing: 35.0400
goal: 35.0400
extract: 35.0400
relevant: 35.0400
words: 35.0400
phrases: 35.0400
represent: 35.0400
content: 35.0400
document: 35.0400
yake: 35.0400
technique: 35.0400
uses: 35.0400
several: 35.0400
features: 35.0400

```

```
evaluate: 35.0400
importance: 35.0400
word: 35.0400
```

8.2.10

KP-Miner

The KP-Miner method is a more advanced technique for keyword extraction that builds upon basic statistical methods like TF-IDF (Term Frequency-Inverse Document Frequency). Unlike simpler methods that rely solely on frequency, KP-Miner introduces additional steps and statistical functions to enhance keyword extraction, making it especially useful for handling more complex documents. This method consists of three key steps: candidate word selection, score calculation, and final selection of keywords based on a combination of these scores.

1. In the **first step**, candidate words are selected from the document. These are words that could potentially be important keywords. KP-Miner introduces two key factors at this stage: the **Least Allowable Seen Frequency (LASF)** factor, which ensures that only words that appear more than a specified number of times (denoted as "n") in the document are considered candidates. This prevents overly rare or irrelevant words from being considered as key phrases. Additionally, the method introduces the **CutOff** factor, which filters out words that appear after a certain position in the document. If a word appears beyond this threshold, it is deemed less likely to be important and is removed from the pool of candidates.
2. The **second step** involves calculating a score for each candidate word. This score is typically based on both the frequency of the word in the document and its importance across a larger corpus. The **TF-IDF** score is commonly used to determine how important a word is within the context of the document compared to other documents in the corpus. The higher the TF-IDF score, the more relevant the word is considered.
3. Finally, in the **third step**, the candidate words are ranked by their calculated scores. The words with the highest scores are selected as the final keywords. This combination of frequency-based filters and statistical scoring ensures that the keywords extracted are both relevant and representative of the document's main topics.

The KP-Miner method is particularly useful for long and complex documents because it filters out less relevant terms early in the process and uses statistical scoring to ensure that the most meaningful terms are selected. By combining multiple statistical techniques, it achieves more accurate and contextually relevant keyword extraction compared to simpler methods.

8.2.11

List the steps of the KP-miner algorithm

- TF calculation
- Calculation of candidate words
- Calculation of TF-IDF
- Calculation of IDF
- Calculation Factor of the lowest permissible frequency of vision and CutOff

8.2.12

The difference between YAKE and KP-miner is that KP-miner uses candidate word locations or TF-IDF information and introduces a new set of five features

- True
- False

8.2.13

Project: Implementation of KP miner method

Implement the KP-Miner algorithm for keyword extraction from a text document. KP-Miner is a more advanced method that uses Term Frequency-Inverse Document Frequency (TF-IDF) to assess the importance of candidate phrases. The algorithm includes three main steps: candidate selection, scoring of candidates, and final keyword selection. It introduces two unique statistical functions to improve keyword extraction: **Least Allowable Seen Frequency** (LASF) and **CutOff**.

```
import string
import nltk
from collections import Counter
import math
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

# Download necessary NLTK resources if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')
```

Program output:

```
[nltk_data] Downloading package stopwords to
/home/johny/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package punkt to
/home/johny/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

1. Preprocessing:

- Converts text to lowercase and removes punctuation and stopwords to avoid irrelevant words affecting the keyword extraction.

```
# Step 1: Preprocess the Text
def preprocess_text(text):
    text = text.lower() # Convert text to lowercase
    text = text.translate(str.maketrans('', '',
string.punctuation)) # Remove punctuation
    words = word_tokenize(text) # Tokenize the text into
words
    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in words if word not in
stop_words] # Remove stopwords
    return filtered_words
```

2. Compute Term Frequency (TF)

- Computes how frequently each word appears in the document. Higher frequency indicates that a word is more important.

```
# Step 2: Compute Term Frequency (TF)
def compute_tf(words):
    tf = Counter(words)
    total_words = len(words)
    tf_scores = {word: count / total_words for word, count in
tf.items()}
    return tf_scores
```

3. Compute Inverse Document Frequency (IDF)

- Calculates the IDF score for each word based on how often it appears in the entire corpus. Words that are common across documents get lower IDF scores, while rare words get higher scores.

```
# Step 3: Compute Inverse Document Frequency (IDF)
def compute_idf(corpus, word):
```

```

    num_docs_containing_word = sum(1 for doc in corpus if word
in doc)
    return math.log(len(corpus) / (num_docs_containing_word +
1))

```

4. Compute TF-IDF Score

- Combines the TF and IDF scores to give a weighted importance score for each word.

```

# Step 4: Compute TF-IDF Score
def compute_tfidf(tf_scores, idf_scores):
    tfidf_scores = {}
    for word, tf_score in tf_scores.items():
        tfidf_scores[word] = tf_score * idf_scores.get(word,
0)
    return tfidf_scores

```

5. Select Candidate Words

- **LASF (Least Allowable Seen Frequency)**: Ensures that only words that appear more than a certain number of times (threshold) in the document are considered as candidates.
- **CutOff**: Discards words that appear after a specified position in the document to filter out less relevant terms.

```

# Step 5: Select Candidate Words Using LASF and CutOff
def select_candidates(filtered_words, tfidf_scores,
original_text, cutoff_position=2, lasf_threshold=1):
    candidate_words = {}

    # Apply Least Allowable Seen Frequency (LASF)
    word_frequency = Counter(filtered_words)
    for word, freq in word_frequency.items():
        if freq >= lasf_threshold:
            candidate_words[word] = tfidf_scores.get(word, 0)

    # Apply CutOff - Discard words that appear after the
cutoff position in the document
    sentences = sent_tokenize(original_text)
    cutoff_words = set()
    for i, sentence in enumerate(sentences[:cutoff_position]):
        for word in word_tokenize(sentence):
            cutoff_words.add(word.lower())

```

```

    candidate_words = {word: score for word, score in
candidate_words.items() if word in cutoff_words}
    return candidate_words

```

6. Rank Keywords:

- Rank the candidate words based on their final score. The words with the highest score are considered the most important keywords.

```

# Step 6: Rank Keywords Based on Their Scores
def rank_keywords(candidate_words):
    ranked_keywords = sorted(candidate_words.items(),
key=lambda x: x[1], reverse=True)
    return ranked_keywords

```

- and run:

```

# Example corpus for testing the function
corpus = [
    "Keyword extraction is an essential process in natural
language processing.",
    "KP-Miner is a keyword extraction algorithm that uses TF-
IDF and additional statistical methods.",
    "In the KP-Miner algorithm, candidate words are selected
based on frequency and position within the document."
]

# Preprocess the corpus and extract words from all documents
all_filtered_words = [preprocess_text(doc) for doc in corpus]

# Compute the IDF score for each word in the entire corpus
idf_scores = {}
all_words = set(word for doc in all_filtered_words for word in
doc)
for word in all_words:
    idf_scores[word] = compute_idf(all_filtered_words, word)

# Initialize a list to store tfidf_scores for each document
separately
all_tfidf_scores = []

# For each document, compute the TF and TF-IDF scores

```

```

for i, doc in enumerate(all_filtered_words):
    tf_scores = compute_tf(doc)
    tfidf_scores = compute_tfidf(tf_scores, idf_scores)
    all_tfidf_scores.append(tfidf_scores)

# Apply LASF and CutOff to select candidate words for each
document in the corpus
for i, (doc, tfidf_scores, original_text) in
enumerate(zip(all_filtered_words, all_tfidf_scores, corpus)):
    candidate_words = select_candidates(doc, tfidf_scores,
original_text)
    ranked_keywords = rank_keywords(candidate_words)

    # Display the ranked keywords with their scores
    print(f"\nRanked Keywords for Document {i + 1}:")
    for word, score in ranked_keywords:
        print(f"{word}: {score:.4f}")

```

Program output:

Ranked Keywords for Document 1:

```

essential: 0.0579
process: 0.0579
natural: 0.0579
language: 0.0579
processing: 0.0579
keyword: 0.0000
extraction: 0.0000

```

Ranked Keywords for Document 2:

```

uses: 0.0451
additional: 0.0451
statistical: 0.0451
methods: 0.0451
keyword: 0.0000
extraction: 0.0000
algorithm: 0.0000

```

Ranked Keywords for Document 3:

```

candidate: 0.0405
words: 0.0405
selected: 0.0405
based: 0.0405
frequency: 0.0405
position: 0.0405
within: 0.0405

```



```
document: 0.0405  
algorithm: 0.0000
```

8.3 Graph based approaches

8.3.1

In keyword extraction, graph-based approaches leverage the structure of a graph to identify important terms or phrases in a document. These methods treat the words or phrases as nodes in a graph, where edges represent relationships between them based on co-occurrence, proximity, or semantic similarity. By incorporating global information from the entire document, graph-based algorithms can capture more complex relationships than local methods. Unlike simpler methods, which rely solely on individual words or statistical counts, graph-based methods consider both local and global information to rank words or phrases based on their overall importance in the context of the entire document.

8.3.2

Which of the following are characteristics of graph-based approaches in keyword extraction?

- They treat words or phrases as nodes in a graph, connected by edges.
- They incorporate global information from the entire document.
- They rely on statistical counts of individual words only.
- They only consider the co-occurrence of words within a single sentence.

8.3.3

PageRank

Many graph-based algorithms rely on the **PageRank** algorithm as their foundation. Originally developed to rank web pages based on the quantity and quality of links pointing to them, PageRank has been adapted for keyword extraction by ranking words or phrases in a document according to their relationships with other terms. In the context of keyword extraction, the goal is to identify which words or phrases are most important by considering not just their frequency but also their contextual connections with other terms in the text. These connections are often modeled as a graph, where the weight of the edges represents the strength of the relationship between two terms.

The **PageRank** algorithm operates recursively to rank nodes based on the importance of the nodes they are connected to. The basic PageRank formula for keyword extraction is similar to its original use in ranking web pages:

$$PR(A) = \frac{1-d}{N} + d \left(\sum_{i=1}^n \frac{PR(T_i)}{C(T_i)} \right)$$

Where:

- **PR(A)** is the PageRank of page **A**,
- **PR(T_i)** is the PageRank of page **T_i** (the pages linking to **A**),
- **C(T_i)** is the number of links on page **T_i**,
- **N** is the total number of pages,
- **d** is the damping factor, which accounts for the probability that a user will stop clicking on links (typically set around 0.85),
- **1-d / N** represents a baseline probability that a page may be randomly selected.

The damping factor **d** is a critical component, as it helps to mitigate the impact of pages that have a large number of outgoing links, which could otherwise unfairly skew the results. Additionally, it handles the situation where a page may not have any incoming links, ensuring that every page still receives a minimum rank.

In summary, graph-based approaches like PageRank allow for the calculation of node importance by incorporating both local and global structural information in the graph, making them effective for tasks like web page ranking, recommendation systems, and network analysis. The recursive nature of these algorithms ensures that they can dynamically adjust based on the entire graph, leading to more accurate and comprehensive results.

8.3.4

TextRank

TextRank applies the PageRank algorithm to a graph where vertices represent words in a document. The edges between the words are weighted based on their co-occurrence, proximity, and syntactic relationships within the text.

A distinctive feature of TextRank is its ability to handle different domains, languages, and genres by incorporating deep linguistic knowledge or using annotated corpora. Unlike the original PageRank algorithm, which assumes an unweighted graph, TextRank works with a weighted graph to reflect the strength of relationships between words. This is particularly important as natural language texts contain a rich variety of relationships that must be captured effectively. TextRank is flexible and can be adapted to various domains, making it highly transferable and useful in multiple applications.

The first step in applying TextRank is tokenizing the input text and then annotating the tokens with Part-of-Speech (POS) tags. Research has shown that TextRank

performs best when only nouns and adjectives are used as vertices. A syntactic filter is applied to exclude other parts of speech, ensuring that only meaningful and relevant words contribute to the graph's structure. This approach combines graph-based methods with linguistic features, making TextRank a powerful tool for keyword extraction across different types of text.

Here's how the TextRank algorithm works step-by-step:

1. Text Preprocessing - before applying TextRank, the input text needs to be preprocessed. This typically involves: **Tokenization**: Breaking the text into individual words or phrases; **POS Tagging**: Annotating each word with its part of speech (noun, verb, adjective, etc.). The TextRank algorithm typically focuses on nouns and adjectives because they carry the most semantic meaning relevant to keyword extraction; **Syntactic Filtering**: Filtering out parts of speech that are not useful for keyword extraction, such as verbs or stop words, leaving nouns and adjectives to form the vertices of the graph.

2. Constructing the Graph - In TextRank, the text is represented as a graph where **Vertices (nodes)**: The nodes of the graph represent words (or phrases) from the document that are considered important for the task, typically nouns and adjectives; **Edges**: Edges represent the relationships between words. These are created based on co-occurrence within a certain window (i.e., words that appear in close proximity to each other). This edge is weighted, with the strength of the relationship determined by the frequency of co-occurrence.

3. Building the Weighted Graph - After the preprocessing steps, the graph is built with the following details: Each word that remains after POS tagging and syntactic filtering is a vertex; The edges between these vertices are weighted based on how often the words co-occur or their semantic relationships. In some implementations, semantic similarity measures (such as cosine similarity) can also be used to adjust the weight of edges. In TextRank, the weight of the edge between two words is typically computed as: **Weight of an edge**: Based on the co-occurrence frequency between the two words in the document (or context window).

4. Applying the PageRank Algorithm - Once the graph is constructed, TextRank applies the **PageRank algorithm** to this graph to rank the vertices (words or phrases) based on their importance in the entire document. PageRank works iteratively by updating the rank of each vertex based on the ranks of neighboring vertices: Each vertex's rank is computed by accumulating the rank of its neighbors, weighted by the edges between them.

The formula for updating the rank of vertex V_i is:

$$R(V_i) = (1 - d) + d \times \sum \frac{R(V_j)}{C(V_j)}$$

where:

- $R(V_i)$ is the rank of vertex V_i ,
- d is the damping factor (usually set between 0.1 and 0.3),
- V_j represents neighboring vertices,
- $C(V_j)$ is the number of outgoing edges from V_j ,
- The sum represents the cumulative rank from neighboring vertices.

This process is repeated iteratively until the rank values converge (i.e., the ranks stop changing significantly between iterations).

5. Selecting the Key Phrases - After applying the PageRank algorithm, the vertices (words or phrases) are ranked based on their importance scores. The words/phrases with the highest ranks are considered the most important keywords or key phrases in the document. Typically, the top-ranked words are extracted as key terms or phrases that summarize the content of the document.

8.3.5

Which of the following are true about the TextRank algorithm?

- TextRank applies a syntactic filter to exclude parts of speech other than nouns and adjectives.
- The algorithm uses a weighted graph based on co-occurrence and syntactic relationships between words.
- TextRank uses an unweighted graph, like the original PageRank algorithm.
- TextRank is not applicable to different domains or languages.

8.3.6

The difference between the TextRank and PageRank algorithms is the weight assigned based on the two words in the predefined window and the number of words the algorithms keep as potential keywords.

- yes
- no

8.3.7

SingleRank

SingleRank is an extension of the TextRank algorithm, which itself is based on the PageRank algorithm. While SingleRank shares many similarities with TextRank, there are two key differences that set it apart and affect its performance in keyword extraction tasks.

1. Edge Weighting Based on Word Distance

Like TextRank, SingleRank constructs a graph where vertices represent words, and edges represent relationships between words based on their co-occurrence in the document. However, unlike TextRank, which considers the co-occurrence of words in a predefined window without accounting for the distance between them, SingleRank assigns weights to edges based on the distance between two co-occurring words. This means that words that appear close to each other in the document will have stronger connections (higher weights), reflecting a more significant relationship between them.

For example, in the sentence "The dog chased the cat," the words "dog" and "chased" would be connected with a higher weight because they are closer together, indicating a stronger semantic connection. Conversely, words that are farther apart in the text may be assigned a lower edge weight.

2. Retention of All Keywords

Another key difference between SingleRank and TextRank is how they handle the number of extracted keywords. In TextRank, the top-ranked 30% of words or phrases are typically selected as the final key terms, based on their PageRank scores. This means that only the most important terms are kept.

In contrast, SingleRank keeps **all the words** after running the PageRank algorithm, rather than just the top-ranked ones. This approach leads to a broader set of keywords, ensuring that more terms are retained, which can be useful in contexts where a larger pool of keywords is desired or where all terms are relevant for further analysis.

Steps in the SingleRank Algorithm

1. Text Preprocessing: Like TextRank, the first step in SingleRank is text preprocessing, which includes tokenization and POS tagging. Words are filtered based on their syntactic roles (e.g., nouns and adjectives are kept as vertices).

2. Graph Construction: A graph is built where:

- **Vertices** represent the filtered words (nouns and adjectives).
- **Edges** are formed based on word co-occurrence, with additional weight given to words that appear closer together in the text (based on a predefined window size).

3. PageRank Algorithm: The PageRank algorithm is applied to the graph to rank the importance of each word. Each word's rank is influenced by its connections with other words in the document, considering both local co-occurrence and the distance between words.

4. Selection of Keywords: Unlike TextRank, where only the top 30% of ranked words are selected, SingleRank retains **all** the words as potential keywords after running the PageRank algorithm. This ensures that a larger set of keywords is considered.

Key features of SingleRank

- **Distance-Aware Edge Weights:** By factoring in the distance between co-occurring words, SingleRank refines the relationship between words and improves the accuracy of keyword extraction, especially for longer documents.
- **Broader Keyword Set:** By keeping all the words rather than just the top-ranked 30%, SingleRank offers a more exhaustive list of keywords, which can be useful for tasks that require a wide range of terms.
- **Simplicity and Efficiency:** Like TextRank, SingleRank is unsupervised, meaning it does not require labeled data or extensive training, making it simple to implement and use.

8.3.8

What is the main difference between SingleRank and TextRank?

- SingleRank assigns edge weights based on word distance, while TextRank does not.
- SingleRank retains only the top 30% of ranked words, while TextRank retains all words.
- SingleRank uses only adjectives for keyword extraction, while TextRank uses all parts of speech.
- SingleRank does not use the PageRank algorithm, while TextRank does.

8.3.9

TopicRank

The TopicRank algorithm is another graph-based method used for keyword extraction. Unlike TextRank and SingleRank, which focus on individual words or word relationships, TopicRank targets extracting key phrases that represent topics present throughout the document. The algorithm assumes that key phrases related to the same topic are important for summarizing the document. Therefore, it ranks these key phrases based on their relevance and importance within the document.

The TopicRank algorithm involves three main steps: identifying topics, chart-based assessment, and keyword selection.

1. The first step is to identify topics within the document, which are groups of related key phrases that represent a central theme.
2. Once the topics are identified, a chart-based assessment step assigns relevance scores to different topics and their associated key phrases.

3. Finally, in the keyword selection step, the algorithm chooses the most relevant key phrase for each identified topic.

This process is achieved using different strategies, such as selecting the most frequent key phrase, choosing the key phrase that appears first in the document, or selecting the key phrase that represents the centroid of the cluster.

Topic identification strategies

There are three strategies used in TopicRank to select the most representative key phrase for each topic. The first strategy involves converting all key phrases back to their generic form and selecting the first one that appears in the document. The second strategy selects the most frequent key phrase, assuming that frequent phrases are more relevant to the overall topic. The third strategy is based on selecting the key phrase closest to the centroid of the cluster, where the centroid represents the central point or the most significant phrase in the group of related phrases. This approach is useful when the document contains multiple overlapping themes and helps in finding the most representative term for each topic.

8.3.10

Sort the steps of the TopicRank algorithm

- keyword selection
- chart-based assessment
- topic identification

8.3.11

Which of the following are strategies used by the TopicRank algorithm to select key phrases?

- Choosing the key phrase that appears first in the document.
- Selecting the most frequent key phrase.
- Selecting the key phrase that has the highest TF-IDF score.
- Selecting the key phrase closest to the centroid of the cluster.

8.3.12

Which of the following are strategies used by the TopicRank algorithm to select key phrases?

- Choosing the key phrase that appears first in the document.
- Selecting the most frequent key phrase.
- Selecting the key phrase that has the highest TF-IDF score.
- Selecting the key phrase closest to the centroid of the cluster.

 8.3.13

Project: Implementing the TextRank algorithm

The aim is to identify and extract important keywords from a given text document by leveraging the relationships between words in the document.

```
# Importing necessary libraries
import spacy
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Load the spaCy model for tokenization and POS tagging
nlp = spacy.load("en_core_web_sm")
```

1. Data Collection

- Collect or create a text document that you would like to extract keywords from. For this example, we will use a sample text document.

```
# Sample text to extract keywords from
text = """TextRank is a graph-based algorithm used for keyword
extraction. It ranks words based on their importance within a
text. TextRank uses graph-based methods and operates on
weighted graphs."""

# Step 1: Preprocessing the text (tokenization, POS tagging,
stop word removal, lemmatization)
doc = nlp(text)
tokens = [token.lemma_ for token in doc if not token.is_stop
and not token.is_punct and token.pos_ in ["NOUN", "ADJ"]]
```

2. Preprocessing the Text

- **Tokenization:** Split the text into tokens (words).
- **POS Tagging:** Use Part-of-Speech (POS) tagging to filter out non-nouns and adjectives, as they are considered key for extracting meaningful keywords.
- **Remove Stop Words:** Remove common stop words (e.g., "is", "and", "the") from the token list to avoid unnecessary words.
- **Lemmatization:** Convert words to their base form.

```
# Step 2: Constructing the graph (using co-occurrence of
words)
```



```
# Create a co-occurrence matrix (size: n x n, where n is the
number of tokens)
word_count = len(tokens)
co_occurrence_matrix = np.zeros((word_count, word_count))

# Calculate co-occurrence (based on window of 2 words)
window_size = 2
for i in range(word_count):
    for j in range(max(0, i - window_size), min(word_count, i
+ window_size + 1)):
        if i != j:
            co_occurrence_matrix[i][j] = 1
```

3. TextRank Algorithm Implementation

- **Graph Construction:** Build a graph where each word is a node, and edges represent the relationships between words based on co-occurrence (i.e., the words that appear near each other in the text).
- **Weighted Graph:** Assign weights to the edges based on the distance between words in the text.
- **PageRank Algorithm:** Apply the PageRank algorithm to rank the nodes (words) in terms of importance. Words with higher ranks will be considered more important.

```
# Step 3: Applying PageRank (using cosine similarity to weight
the edges)
cosine_sim = cosine_similarity(co_occurrence_matrix)

# Initialize PageRank scores (using a simple uniform
distribution)
pagerank_scores = np.ones(word_count) / word_count

# Damping factor for PageRank
damping_factor = 0.85
iterations = 100

# PageRank algorithm: iterative process to calculate the rank
of each word
for _ in range(iterations):
    new_scores = (1 - damping_factor) / word_count +
damping_factor * np.dot(cosine_sim, pagerank_scores)
    pagerank_scores = new_scores
```

4. Extracting Keywords

- After running the PageRank algorithm, sort the words based on their rank and select the top-ranking words as the extracted keywords.

```
# Step 4: Extracting the top keywords based on their rank
keyword_ranks = [(tokens[i], pagerank_scores[i]) for i in
range(word_count)]
keyword_ranks_sorted = sorted(keyword_ranks, key=lambda x:
x[1], reverse=True)
```

- and run:

```
# Top 5 keywords
top_keywords = keyword_ranks_sorted[:5]

# Displaying the results
print("Top Keywords and their scores:")
for keyword, score in top_keywords:
    print(f"{keyword}: {score}")
```

Program output:

```
Top Keywords and their scores:
importance: 1.6474711062025287e+47
word: 1.5133000924445996e+47
text: 1.5133000924445996e+47
extraction: 1.28080457127524e+47
graph: 1.28080457127524e+47
```

8.4 Machine learning based approaches

8.4.1

Machine learning-based approaches to keyword extraction transform the task into a classification or prediction problem. This process relies on supervised learning, where a model trained on a labeled dataset determines if a candidate word in the text qualifies as a keyword or not. By utilizing this data, machine learning methods provide a more refined extraction, often capturing words with high semantic relevance. These methods require less preprocessing of text and can yield keywords that align closely with the document's meaning, which is a key advantage over simpler, rule-based methods.

Despite their effectiveness, machine learning-based approaches also have certain limitations. Models are often specific to the language and context of the training data. This means that if the dataset changes or if keywords are required for a new domain, the model may need to be retrained, which can be time-consuming and resource-intensive. Furthermore, machine learning methods involve a higher computational load than statistical approaches, leading to slower extraction times.

For tasks that demand both accuracy and semantic richness, machine learning-based keyword extraction methods are highly effective. However, practitioners must weigh the need for context-specific training and higher processing demands. When used appropriately, machine learning approaches significantly enhance keyword extraction by focusing on semantic alignment with the document's content.

8.4.2

Which of the following is an advantage of machine learning-based keyword extraction methods?

- High semantic relevance
- Reduced computational load
- General domain adaptability
- Simple rule-based processing

8.4.3

What is a limitation of machine learning-based keyword extraction methods?

- Language and context dependency
- Higher computational load
- Requires extensive text preprocessing
- Minimal computational requirements

8.4.4

KeyBERT

KeyBERT is a widely used keyword extraction method that utilizes a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model. This technique begins by converting the document into a set of fixed-size vectors that capture its semantic meaning. KeyBERT then breaks down the document into smaller phrases and identifies the candidate phrases that best represent its content. These vectors are crucial because they provide a representation of the document's semantics, enabling the extraction of keywords that align with the document's core topics.

Next, candidate keywords are generated using simple statistical techniques, such as term frequency and TF-IDF. These phrases are processed through the BERT model to generate a phrase-level representation. This step is essential because it allows

KeyBERT to match the meaning of phrases closely with the content of the document, rather than relying solely on frequency or position in the text.

Finally, KeyBERT computes the cosine similarity between the document-level and phrase-level representations to identify the most relevant keywords. These keywords, ranked by similarity scores, are chosen as the most representative of the document's content. The use of a semantic model like BERT makes KeyBERT particularly effective in identifying keywords that are contextually accurate.

8.4.5

What is the main purpose of the BERT model in KeyBERT?

- To create a semantic representation of the document
- To count word frequencies
- To calculate TF-IDF scores
- To simplify keyword generation

8.4.6

List the steps of the KeyBERT algorithm

- Creating a phrase-level representation
- Creating a document-level representation
- Getting key phrases
- Selection of candidate phrases
- The document is sent to the BERT model
- Calculation of cosine similarity

8.4.7

KEA

KEA, one of the earliest machine learning-based keyword extraction methods, employs a Naive Bayes classifier to evaluate the importance of candidate keywords. In this approach, each candidate word is analyzed by calculating its TF-IDF score and its first occurrence within the text. These values serve as inputs to the Naive Bayes classifier, which then predicts the likelihood that a candidate word is a keyword. This classification enables KEA to capture words that are not only frequently used but also contextually significant within the document.

The TF-IDF calculation in KEA prioritizes words that appear often in a specific document but are less common across other documents. This makes it an effective measure for distinguishing words that are unique to the content. Additionally, the model considers where the word first appears, based on the assumption that words introduced earlier in the text may be more important.

While KEA offers a foundational method for keyword extraction, it has limitations in handling more complex language patterns. However, it remains a useful model for straightforward documents and demonstrates how early machine learning models can aid in efficient keyword extraction.

8.4.8

What feature does KEA use to classify a candidate word as a keyword?

- TF-IDF score
- Cosine similarity
- POS tagging
- Word embeddings

8.4.9

Which classifier does KEA use in its keyword extraction method?

- Naive Bayes
- SVM
- Decision Tree
- Random Forest

8.4.10

Project: KEA-based keyword extraction

Implement a KEA-based keyword extraction system that calculates TF-IDF values and the first occurrence position of candidate phrases, then uses a Naive Bayes classifier to determine whether a candidate phrase is a keyword. This system will classify keywords based on these features using labeled training data.

```
import string
import nltk
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
```

Program output:

```
[nltk_data] Downloading package punkt to
/home/johny/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
/home/johny/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

1. Preprocessing

- The text is tokenized, cleaned, and stopwords are removed.

```
# Preprocess text: tokenize, remove punctuation and stopwords
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans("", "",
string.punctuation))
    words = word_tokenize(text)
    stop_words = set(stopwords.words("english"))
    filtered_words = [word for word in words if word not in
stop_words]
    return filtered_words
```

2. TF-IDF Calculation

- Using TfidfVectorizer, we calculate the TF-IDF scores of candidate phrases in the corpus.

```
# Calculate TF-IDF scores for the corpus
def calculate_tfidf(corpus):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(corpus)
    feature_names = vectorizer.get_feature_names_out()
    return tfidf_matrix, feature_names

# Find the first occurrence position of a word in text
def get_first_occurrence_position(text, word):
    words = np.array(text.split())
    indices = np.where(words == word)[0]
    return indices[0] + 1 if indices.size > 0 else len(words)
```

3. Feature Extraction

- For each word, two features are generated: TF-IDF score and the first occurrence position.

```
# Extract features (TF-IDF score and first occurrence) for
each word
def extract_features(text, feature_names, tfidf_vector):
    features = []
    for word in feature_names:
        # Find the index of the word in feature_names using
np.where
        word_index = np.where(feature_names == word)[0][0]
        tfidf_score = tfidf_vector[0, word_index] # TF-IDF
value for the word
        first_occurrence = get_first_occurrence_position(text,
word) # First position
        features.append([tfidf_score, first_occurrence])
    return np.array(features)
```

4. Training the Classifier

- Using a Naive Bayes classifier, the model is trained on labeled data. Each word in the corpus is labeled as either a keyword (1) or a non-keyword (0).

```
# Sample corpus for testing the function
corpus = [
    "Machine learning-based approaches use supervised learning
for keyword extraction.",
    "KeyBERT is a keyword extraction model based on BERT
embeddings.",
    "KEA algorithm uses Naive Bayes classifier for keyword
extraction."
]

# Preprocess the corpus and extract features
tfidf_matrix, feature_names = calculate_tfidf(corpus)

# Labels (example labels) - tou need to correst it
labels = [
    [1, 0, 1, 0, 1, 0, 1, 0], # Adjusted to match document's
feature count after preprocessing
    [1, 1, 0, 1, 0, 1, 0, 1], # Adjusted
    [1, 0, 1, 1, 0, 0, 1, 0] # Adjusted
```

```

]

# Check and print feature extraction for each document
feature_data = []
all_labels = []

for i, doc in enumerate(corpus):
    tfidf_vector = tfidf_matrix[i]
    preprocessed_text = " ".join(preprocess_text(doc))
    features = extract_features(preprocessed_text,
feature_names, tfidf_vector)
    feature_data.append(features)
    all_labels.extend(labels[i]) # Flattened labels for each
document

# Convert to numpy arrays
X = np.vstack(feature_data) # Feature matrix
y = np.array(all_labels)    # Flattened labels

# Debug print to check shapes
print("Shape of X (features):", X.shape)
print("Shape of y (labels):", y.shape)

```

Program output:

```

Shape of X (features): (63, 2)
Shape of y (labels): (24,)

```

5. Keyword Prediction

- The classifier is used to predict keywords in a new document by applying it to the extracted features.

```

# Train-test split (proceed only if shapes are consistent)
if X.shape[0] == y.shape[0]:
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Evaluate the model
y_pred = classifier.predict(X_test)

```



```

print("Classification Report:\n",
classification_report(y_test, y_pred))

# New document for keyword extraction
new_document = "KEA algorithm uses Naive Bayes to classify
keywords based on TF-IDF and position."
preprocessed_text = "
".join(preprocess_text(new_document))
tfidf_vector, _ = calculate_tfidf([new_document])
new_features = extract_features(preprocessed_text,
feature_names, tfidf_vector)

# Predict keywords
predictions = classifier.predict(new_features)
predicted_keywords = [feature_names[i] for i in
range(len(predictions)) if predictions[i] == 1]
print("Predicted Keywords:", predicted_keywords)
else:
print("Error: The shapes of X and y are inconsistent.
Please check label alignment.")

```

Program output:

```

Error: The shapes of X and y are inconsistent. Please check
label alignment.

```

Correct the mistake and show that you understand the algorithm.

 **8.4.11**
Project: KeyBERT algorithm implementation

For running following project use another infrastructure (local or server).

Implement a KeyBERT-based keyword extraction system that identifies the most relevant keywords from a document based on their semantic similarity to the document's content. This system will utilize BERT embeddings to generate both document-level and phrase-level representations and identify keywords based on cosine similarity.

```

import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from transformers import AutoTokenizer, AutoModel
from keybert import KeyBERT

```

```
# Load the pre-trained BERT model and tokenizer
model_name = "distilbert-base-nli-mean-tokens" # Can use
other BERT variants
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

# Download stopwords if not already downloaded
import nltk
nltk.download('stopwords')
nltk.download('punkt')
```

1. Preprocessing

- The text is cleaned by removing punctuation and stopwords and converting it to lowercase.

```
def preprocess_text(text):
    # Lowercase text
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans("", "",
string.punctuation))
    # Tokenize text
    words = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words("english"))
    filtered_words = [word for word in words if word not in
stop_words]
    return " ".join(filtered_words)
```

2. BERT Embeddings

- Using a pre-trained BERT model, embeddings for both the document and each candidate phrase are generated.

```
def get_embedding(text):
    # Tokenize and create input tensors
    inputs = tokenizer(text, return_tensors="pt",
padding=True, truncation=True)
    outputs = model(**inputs)
```

```

    # Get the embeddings from the BERT model and compute mean
    pooling
    embeddings = outputs.last_hidden_state.mean(dim=1)
    return embeddings.detach().numpy()

# Example Document
document = "Machine learning-based approaches use supervised
learning to extract keywords. KeyBERT is a popular method for
this purpose."
preprocessed_document = preprocess_text(document)
doc_embedding = get_embedding(preprocessed_document)

```

3. Candidate Generation

- KeyBERT uses simple statistical methods to generate candidate keywords. These candidates are sent back into the model for further embedding.

```

# Initialize KeyBERT with the BERT model
kw_model = KeyBERT(model=model_name)

# Generate candidate keywords
candidate_keywords =
kw_model.extract_keywords(preprocessed_document, top_n=10)
print("Candidate Keywords:", candidate_keywords)

```

4. Cosine Similarity Calculation

- The similarity between the document embedding and each candidate phrase embedding is calculated using cosine similarity.

```

def rank_keywords(document, candidates):
    doc_embedding = get_embedding(document)
    candidate_embeddings = [get_embedding(candidate[0]) for
candidate in candidates]
    similarity_scores = [cosine_similarity(doc_embedding,
candidate_emb)[0][0] for candidate_emb in
candidate_embeddings]
    return [(candidates[i][0], similarity_scores[i]) for i in
range(len(candidates))]

# Rank candidates by similarity to the document
ranked_keywords = rank_keywords(preprocessed_document,
candidate_keywords)

```

```
ranked_keywords = sorted(ranked_keywords, key=lambda x: x[1],
reverse=True)
print("Ranked Keywords:", ranked_keywords)
```

5. Keyword Selection

- Keywords are ranked by their similarity scores, and the top results are chosen as the keywords that best represent the document.

```
# Choose top N keywords
top_n = 5
top_keywords = ranked_keywords[:top_n]
print("Top Keywords:", top_keywords)
```

8.5 Deep learning based approaches

8.5.1

Sequence labeling

Sequence labeling models, like Conditional Random Fields (CRF) and Hidden Markov Models (HMM), are another approach to keyword extraction. These models treat text as a sequence and analyze the relationship between words to identify likely keywords. Unlike simple frequency-based methods, sequence labeling considers the context of each word within the document, making it particularly useful for structured text with clear syntactic patterns.

In keyword extraction, sequence labeling models are trained on annotated datasets, where words are labeled as keywords or non-keywords. The model then learns patterns that are typical of keywords, such as certain positional features or surrounding words. Once trained, the model can analyze new texts to predict which words serve as keywords.

This approach is beneficial for documents where context matters, as it allows the model to identify keywords that are relevant based on word sequences and relationships. Sequence labeling models can adapt to more complex documents and offer a more nuanced analysis than basic statistical methods.

8.5.2

Which keyword extraction model treats text as a sequence?

- Sequence labeling models
- KeyBERT
- Term frequency-based models
- KEA

8.5.3

Which types of models are commonly used in sequence labeling for keyword extraction?

- Conditional Random Fields
- Hidden Markov Models
- Naive Bayes
- SVM

8.5.4

Challenges

Machine learning-based approaches to keyword extraction offer great benefits but also present certain challenges. One significant issue is the need for a labeled dataset, which requires time and resources to prepare. Unlike rule-based approaches, these methods depend on annotated data for training, which can make initial setup costly.

Furthermore, machine learning-based keyword extraction models are often language-specific. If a model is trained on English text, it may not perform well on text in other languages without additional training. This language dependency limits the scalability of these models and requires practitioners to consider multilingual datasets if they want to expand the model's application.

Finally, machine learning models typically demand substantial computational resources, which can be a barrier in cases where keyword extraction must be performed in real-time or on a large dataset. Despite these challenges, machine learning-based keyword extraction methods remain essential for capturing keywords with high semantic relevance in complex documents.

8.5.5

What is a common challenge of machine learning-based keyword extraction methods?

- Need for labeled datasets
- Language dependency
- High scalability across languages
- Low computational requirements

8.5.6

Which of the following is NOT a benefit of machine learning-based keyword extraction?

- Minimal preprocessing requirements
- High semantic relevance
- Capturing complex document relationships
- Use in specialized domains

8.6 Evaluation

8.6.1

Statistics-based metrics focus on evaluating an algorithm's keyword extraction performance by examining the proportions of true positives, false positives, and false negatives. These metrics are valuable for assessing the algorithm's general accuracy and ability to avoid false extractions. Precision, recall, and the F1-score are prominent statistics-based metrics used to analyze the extracted keywords' quality and relevance compared to a reference set of manually assigned keywords.

The precision metric calculates the proportion of true positives (correctly identified keywords) relative to all extracted keywords (both true positives and false positives). The formula for precision is:

$$\textit{Precision} = \frac{TP}{TP + FP}$$

Where:

- TP = True Positives (correctly identified keywords)
- FP = False Positives (incorrectly identified keywords)

In this way, precision assesses the accuracy of the keywords the algorithm identifies, showing how well it avoids extracting irrelevant keywords. Recall, on the other hand, measures the algorithm's coverage by calculating the proportion of true positives out of all relevant keywords, including those that were missed (false negatives). The formula for recall is:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Where:

- FN = False Negatives (missed relevant keywords)

The F1-score, or harmonic mean, combines precision and recall, reflecting an algorithm's ability to balance accuracy and coverage. It provides a single measure that accounts for both metrics, making it a valuable tool when an algorithm needs high precision and recall. The formula for the F1-score is:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Together, these metrics enable a statistical assessment that highlights an algorithm's strengths and areas that might need improvement.

8.6.2

What are two primary metrics used to evaluate keyword extraction algorithms?

- Precision
- Recall
- Random Sampling
- Harmonic Mean

8.6.3

Linguistics-based metrics

While statistical metrics assess the quantity and accuracy of extracted keywords, linguistics-based metrics focus more on the quality of keyword ranking and their semantic relevance within the document. These metrics are designed to evaluate how well the algorithm ranks key phrases and whether the most important phrases appear higher in the results. Three important ranking metrics are Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), and Precision at K (P@K).

Mean Reciprocal Rank (MRR) is a metric used to evaluate models that return an ordered list of key phrases. MRR gives the average rank of the first correct keyword in the list, penalizing the model if the correct keyword is ranked too low. The formula for MRR is:

$$MRR = \frac{1}{d} \cdot \sum_{i=1}^d \frac{1}{rank}$$

Where:

- **d** is the number of documents,
- **rank** is the rank position of the first relevant keyword in document *i*.

MRR only cares about the highest-ranked relevant key phrase, making it a suitable measure for evaluating the overall quality of keyword ranking.

Mean Average Precision (MAP), on the other hand, takes into account the order of the returned list of key phrases. It calculates the average precision over all relevant phrases, considering both the precision of the correct phrases and their positions. The formula for MAP is:

$$MAP = \frac{1}{|N|} \cdot \sum_{i=1}^{|N|} AP_i$$

Where:

- **AP_i** is the average precision for document *i*,
- **|N|** is the total number of documents.

MAP provides a more nuanced view of the ranking quality by considering the entire list of key phrases, not just the first relevant one.

8.6.4

Which metric measures how well an algorithm captures all relevant keywords?

- Recall
- Precision
- Mean Reciprocal Rank
- Mean Average Precision

8.6.5

Which metrics help determine the relevance and completeness of extracted keywords?

- F1-score
- Recall
- Mean Average Precision
- Keyword Density

8.6.6

Recall is another critical metric used in keyword extraction, and it evaluates how well the algorithm retrieves all the relevant keywords from a document. Unlike precision, which focuses on the relevance of the retrieved keywords, recall is concerned with the completeness of the extraction process. High recall indicates that most of the relevant keywords in a document have been successfully identified, while low recall suggests that the algorithm is missing many important keywords.

Recall is calculated by dividing the number of true positives (relevant keywords correctly identified) by the sum of true positives and false negatives (relevant keywords that were missed by the algorithm). The formula for recall is:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Where:

- TP = True Positives (relevant keywords identified correctly),
- FN = False Negatives (relevant keywords missed).

For many applications, recall is just as important as precision. In situations where it is crucial to extract all possible relevant keywords, such as in medical or legal text analysis, optimizing for recall can ensure that no key information is missed. However, recall optimization often results in lower precision, as more irrelevant keywords may be retrieved in an attempt to capture all relevant ones.

 8.6.7

What does the Mean Reciprocal Rank (MRR) metric measure?

- The rank of the first relevant keyword
- The total number of keywords
- The precision of all keywords
- The number of relevant keywords

 8.6.8

What metrics focus on the ranking order of key phrases?

- Mean Reciprocal Rank (MRR)
- Mean Average Precision (MAP)
- Precision
- Recall

 8.6.9

Precision is one of the most fundamental metrics for keyword extraction, assessing the accuracy of the keywords that are retrieved by the algorithm. Specifically, it measures how many of the extracted key phrases are actually relevant to the document or text. A high precision score means that the algorithm is very accurate in selecting the relevant keywords, while a low precision score indicates that the algorithm is pulling in a lot of irrelevant keywords.

Mathematically, precision is defined as the number of true positives (relevant keywords correctly identified) divided by the sum of true positives and false positives (irrelevant keywords incorrectly identified). The formula for precision is:

$$Precision = \frac{TP}{TP + FP}$$

Where:

- TP = True Positives (relevant keywords identified correctly),
- FP = False Positives (irrelevant keywords incorrectly identified).

Precision is particularly important in scenarios where extracting irrelevant keywords could significantly impact the quality of results. For example, in a search engine, displaying irrelevant results can detract from the user experience. By optimizing for precision, algorithms can ensure that only the most pertinent keywords are extracted.

 8.6.10

What does the precision metric specifically assess in keyword extraction?

- The accuracy of the keywords retrieved
- The number of relevant documents
- The total number of keywords
- The semantic relevance of keywords

 8.6.11

Which components are involved in calculating recall?

- True Positives (TP)
- False Negatives (FN)
- False Positives (FP)
- Total Number of Extracted Keywords



PRISCILLA



priscilla.fitped.eu