FITPED | AI

Knowledge Discovery

Published on

Work in progress version

Erasmus+ FITPED-AI

Future IT Professionals Education in Artificial Intelligence

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

TABLE OF CONTENTS

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
Latitude  \
0  8.3252       41.0  6.984127   1.023810       322.0  2.555556
37.88
1  8.3014       21.0  6.238137   0.971880      2401.0  2.109842
37.86
2  7.2574       52.0  8.288136   1.073446       496.0  2.802260
37.85
3  5.6431       52.0  5.817352   1.073059       558.0  2.547945
37.85
4  3.8462       52.0  6.281853   1.081081       565.0  2.181467
37.85

   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   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
```

 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.

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

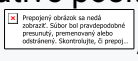
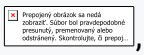
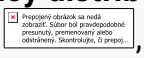
Median - this is the mean value of the variable under study, with the requirement that the values must be arranged in a non-decreasing sequence. We have defined *n* as the number of values and x_i as the value at the i-th position. Then for an even number of elements we calculate the median as follows:

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

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:

```
Kvartilove rozpatie: 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['magnes
ium'])*100)
print('Coefficient of variation
Numpy:',np.std(df['magnesium'])/np.mean(df['magnesium'])*100)
```

Program output:

```
Rozptyl Statistics: 202.8433278626436
Rozptyl Numpy: 202.8433278626436
Smerodajna odchylka Statistics: 14.242307673359806
Smerodajna odchylka Numpy: 14.242307673359806
Variacny koeficient Statistics: 14.27920899998899
Variacny koeficient 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('Variacny
koeficient:',round(np.std(df['HouseAge'])/np.mean(df['HouseAge
'])*100,2))
```

Program output:

```
Variacny koeficient: 43.94
```

2.1.13

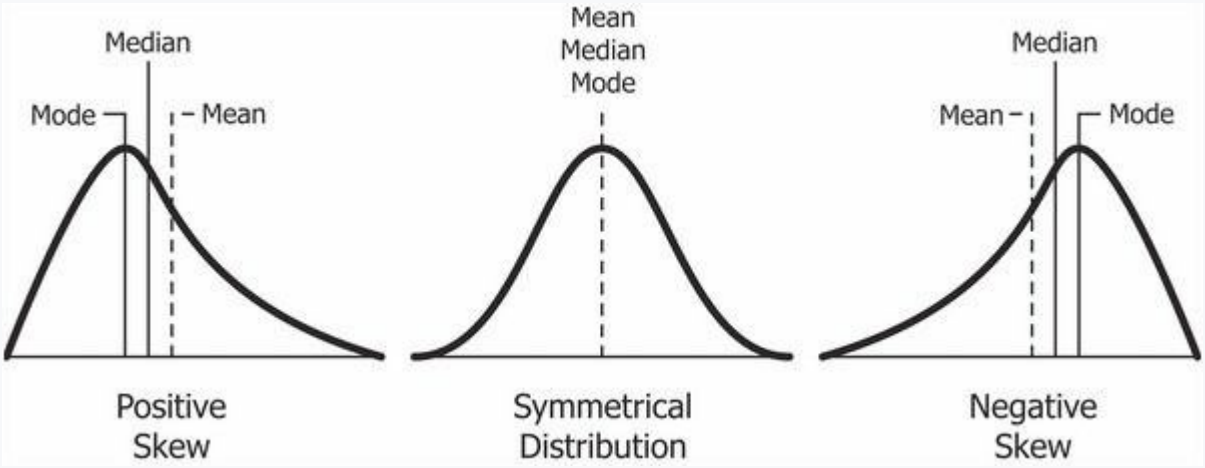
A final option in descriptive statistics is to look at the shape of the data distribution using skewness and kurtosis.

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:

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

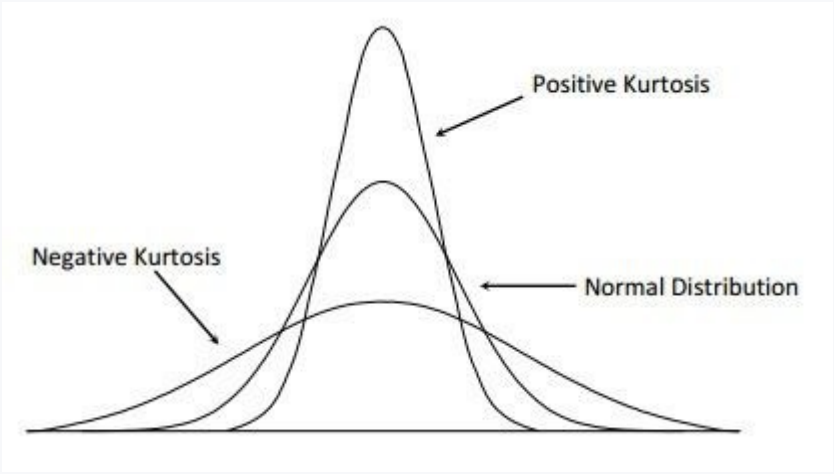
where

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



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('Sikmost:',round(stats.skew(df['HouseAge']),2))
print('Spicatost:',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, 'nulová hypotéza sa zamieta')
else:
  print('p =',p_value, 'nulová hypotéza sa nezamieta')
```

Program output:

```
Sikmost: 0.06
Spicatost: -0.8
p = 0.0 nulová hypotéza sa zamieta
/home/johny/.local/lib/python3.9/site-
packages/scipy/stats/_morestats.py:1800: UserWarning: p-value
may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

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.

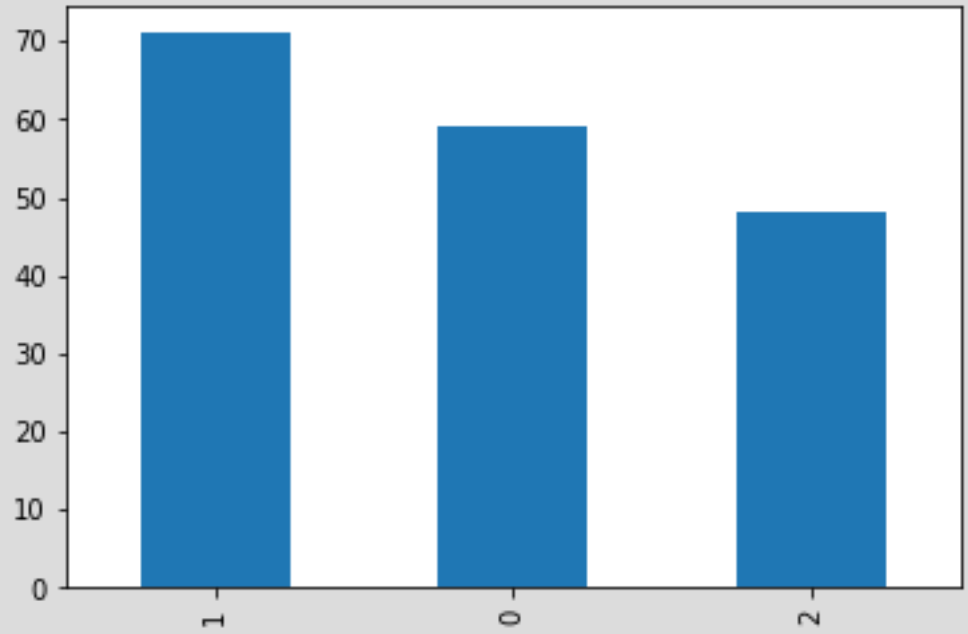
We have also added a *target* column to our data file. This column is used for the classification task, where based on the other variables we can classify the wine into the given three categories. In our case, for the moment, it will mainly serve us to better understand the data.

```
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:

```
Pocetnosti:
1    71
0    59
2    48
Name: target, dtype: int64
Pocetnosti percentualne:
```

```
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
```

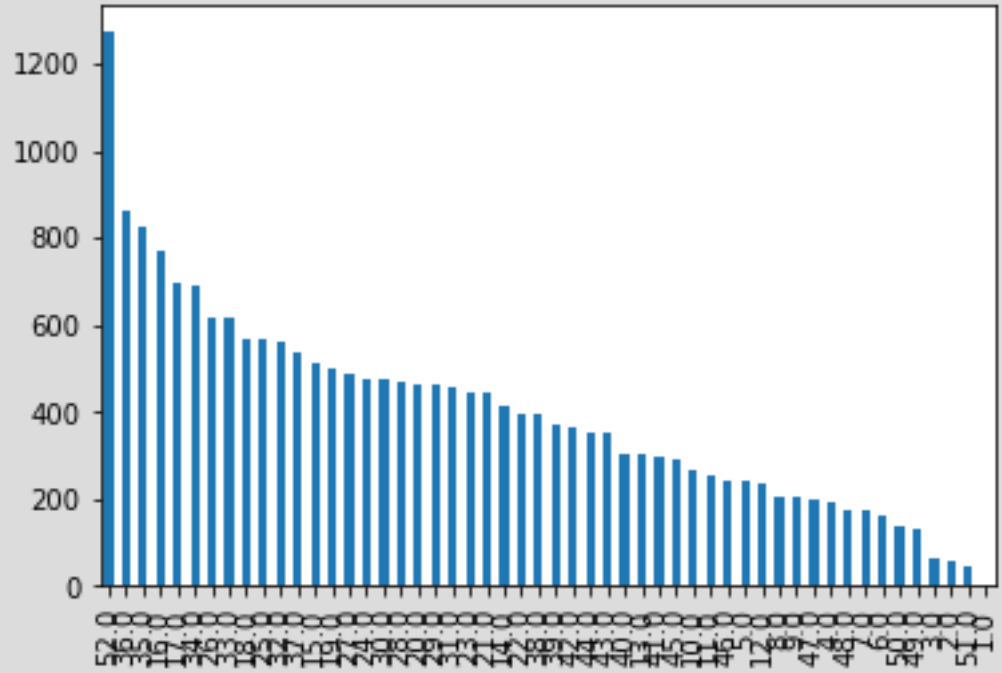
```
import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

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
```

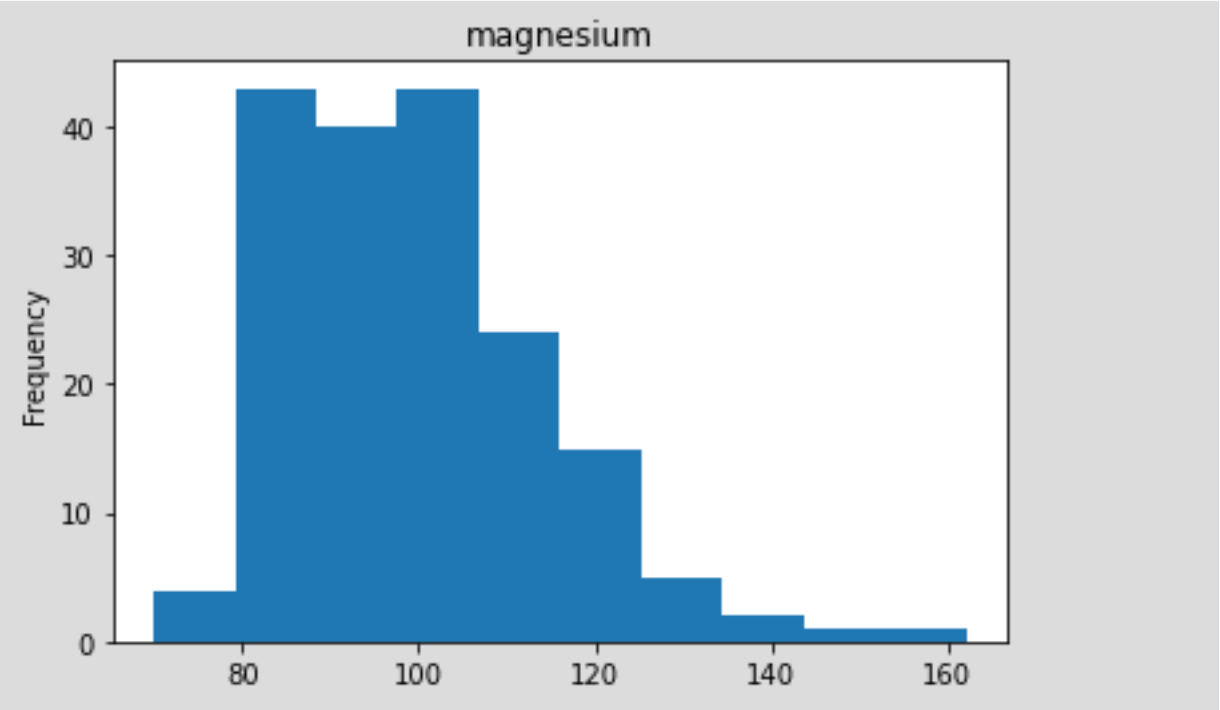


2.2.4

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:



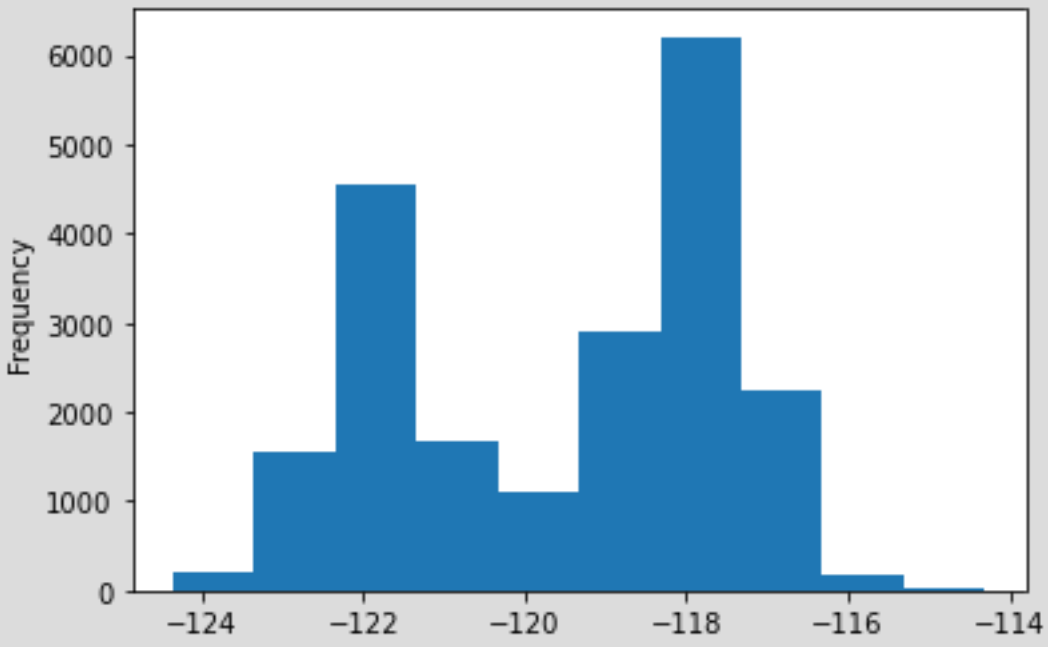
2.2.5

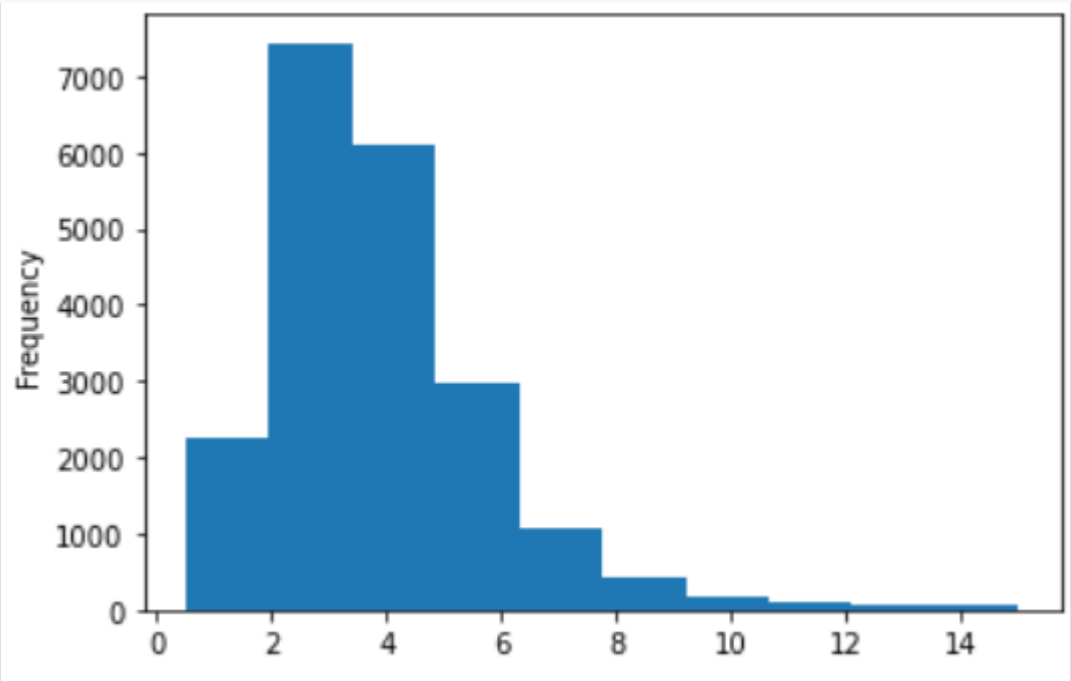
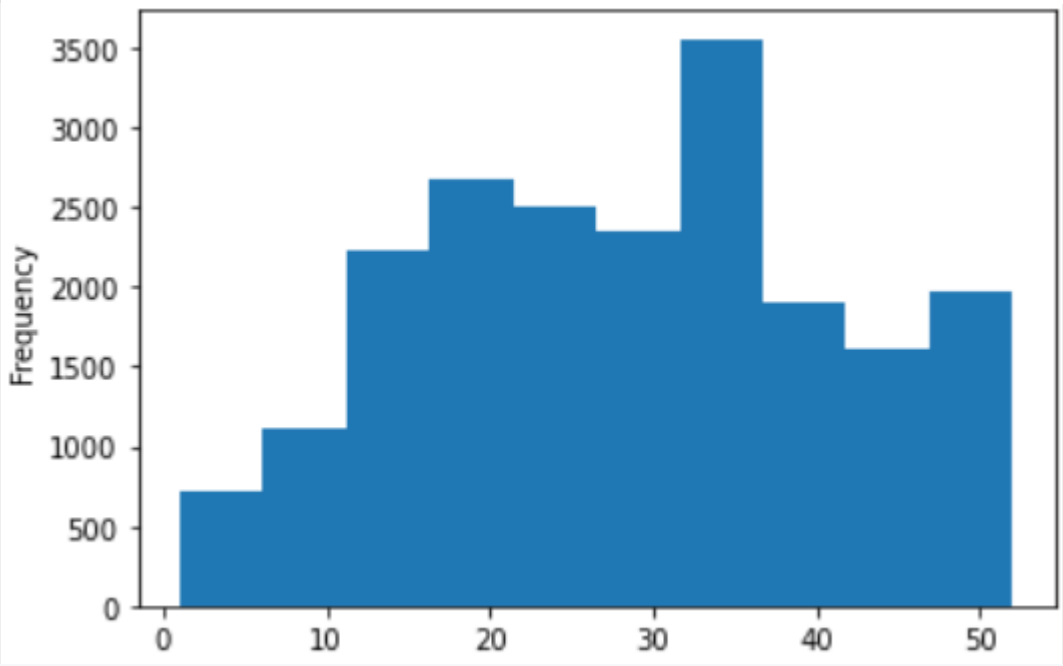
Load from the sklearn library the dataset california_housing, which contains records of homes in California. You fetch the dataset into an object using the **fetch_california_housing()** function. Visualize a histogram of each variable in the dataset. Which of the histograms visualize information about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

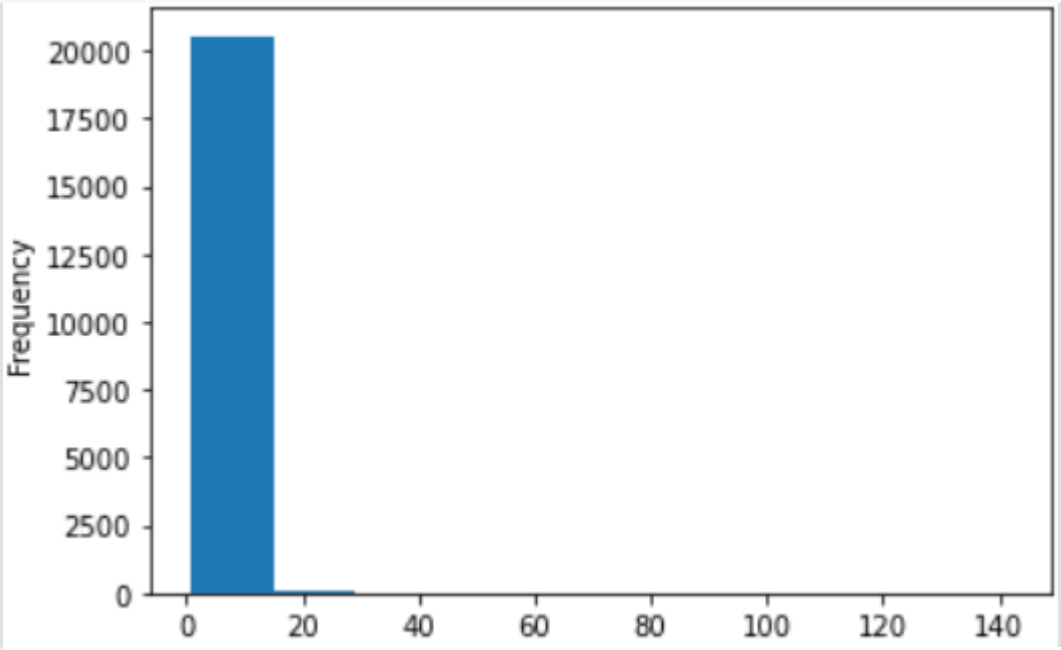
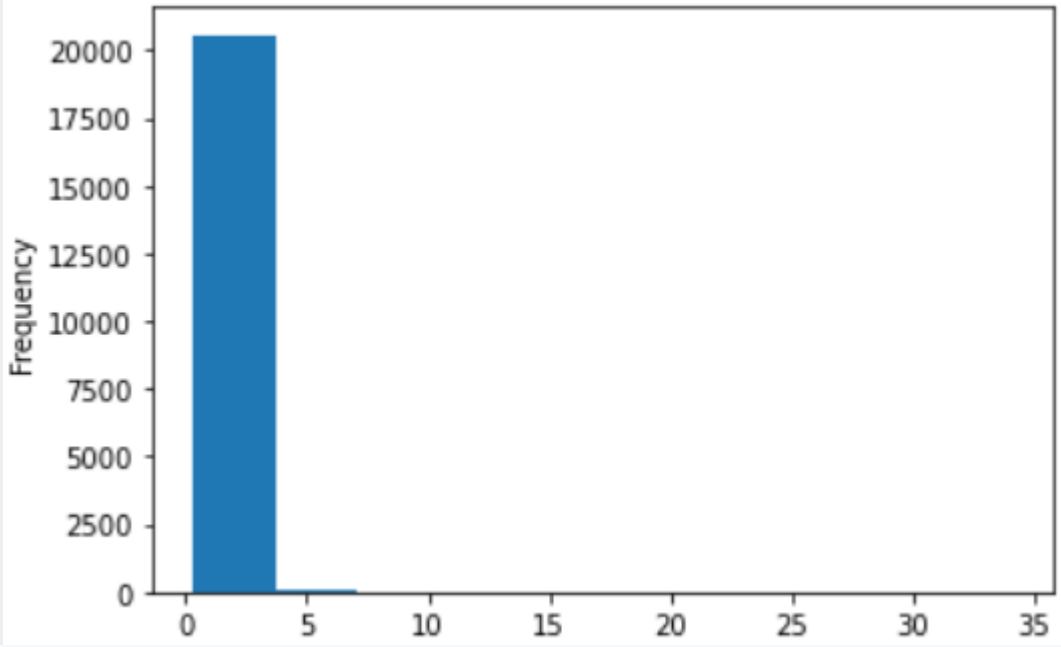
```
import pandas as pd
from scipy import stats
from sklearn.datasets import fetch_california_housing

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')
```

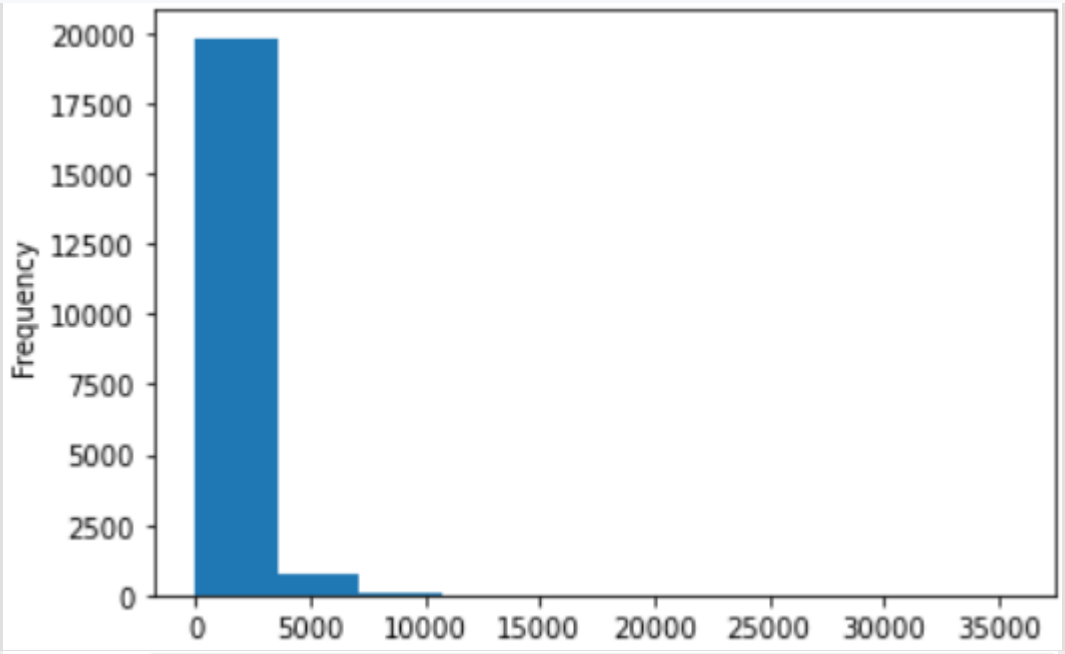
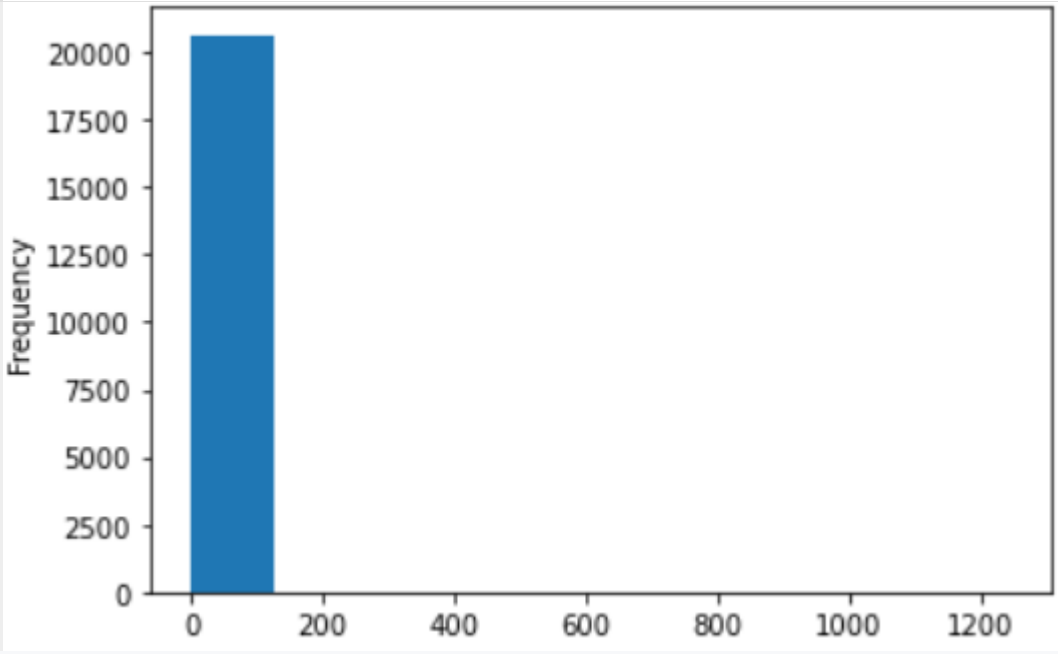
Program output:

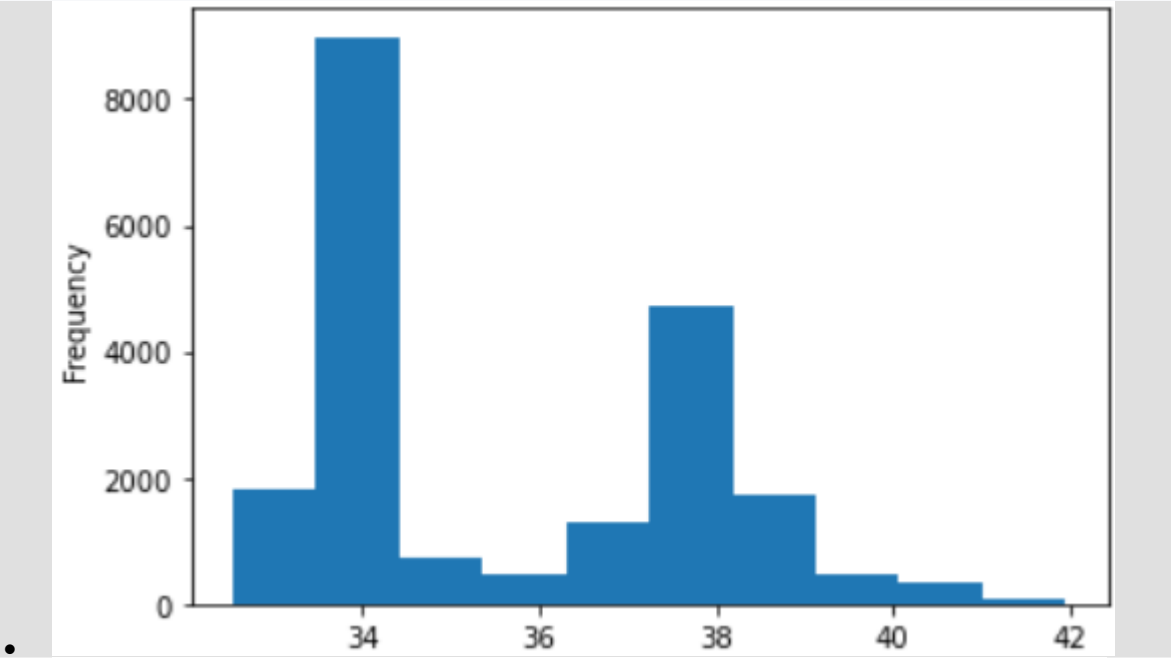
```
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null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

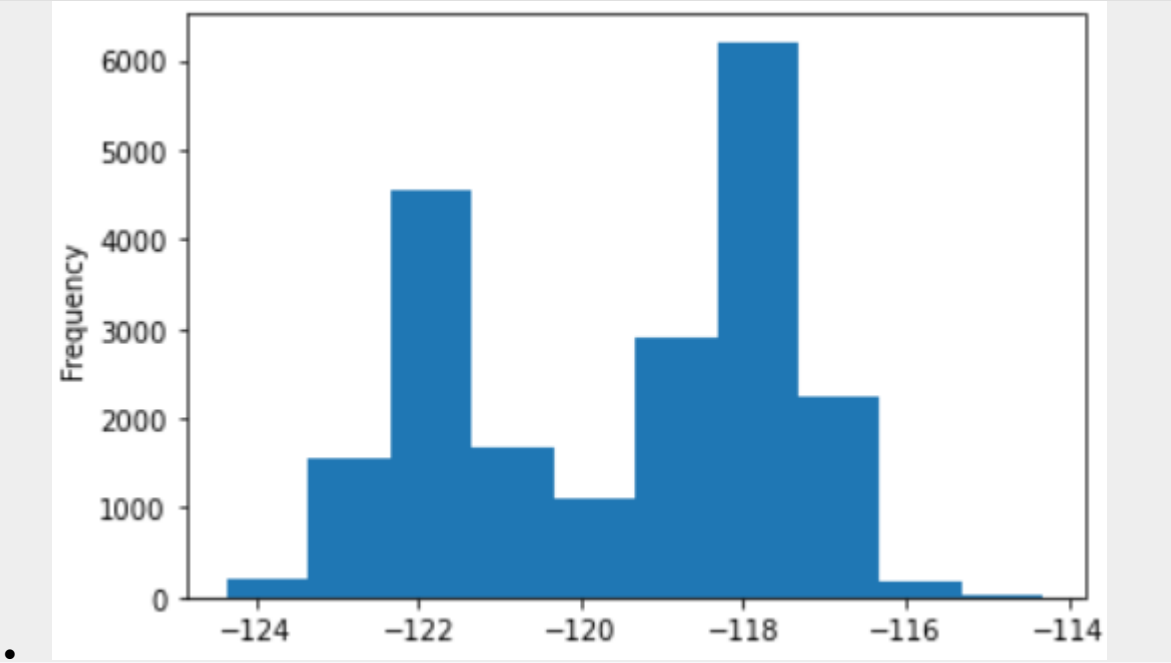


- 
Frequency
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
10
12
14
- 
Frequency
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50

- 
Frequency
20000
17500
15000
12500
10000
7500
5000
2500
0
0
20
40
60
80
100
120
140
- 
Frequency
20000
17500
15000
12500
10000
7500
5000
2500
0
0
5
10
15
20
25
30
35

- 
20000
17500
15000
12500
10000
7500
5000
2500
0
Frequency
0
5000
10000
15000
20000
25000
30000
35000
- 
20000
17500
15000
12500
10000
7500
5000
2500
0
Frequency
0
200
400
600
800
1000
1200



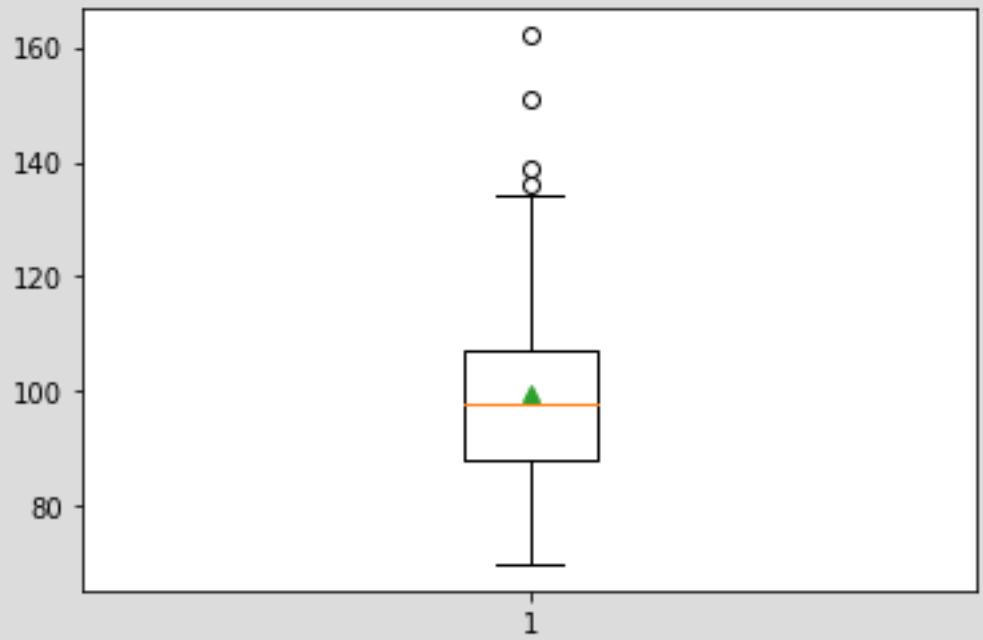


2.2.6

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:



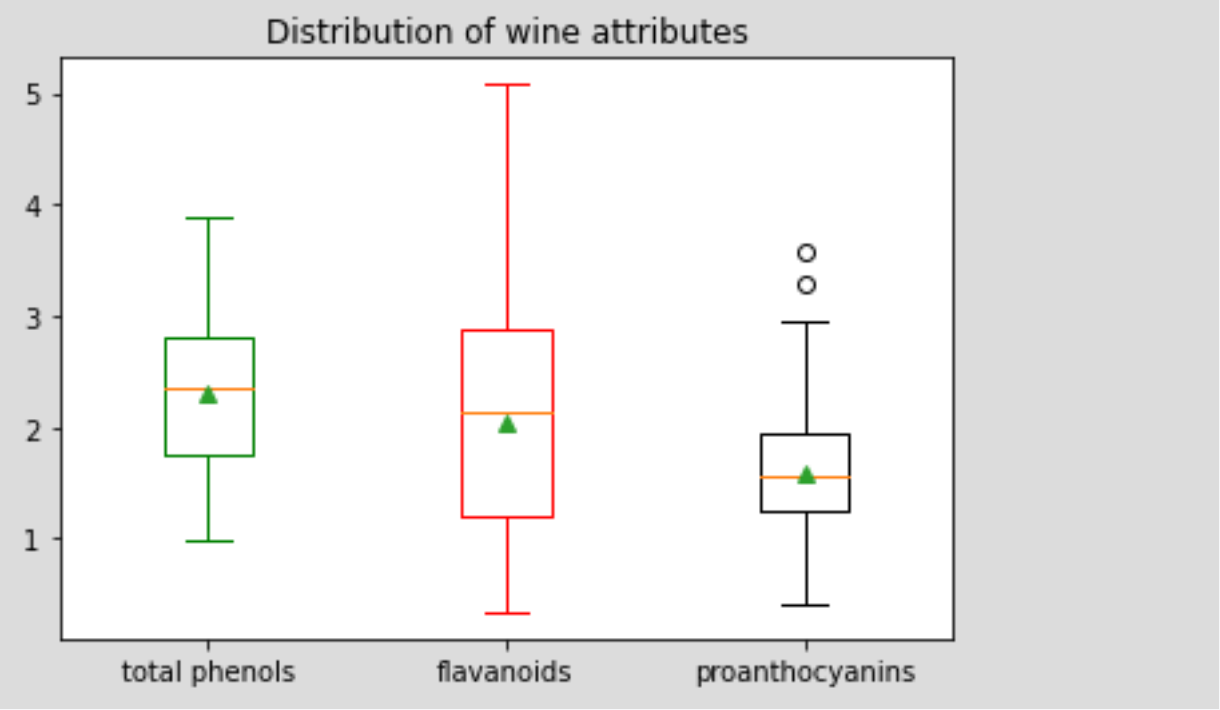
2.2.7

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:



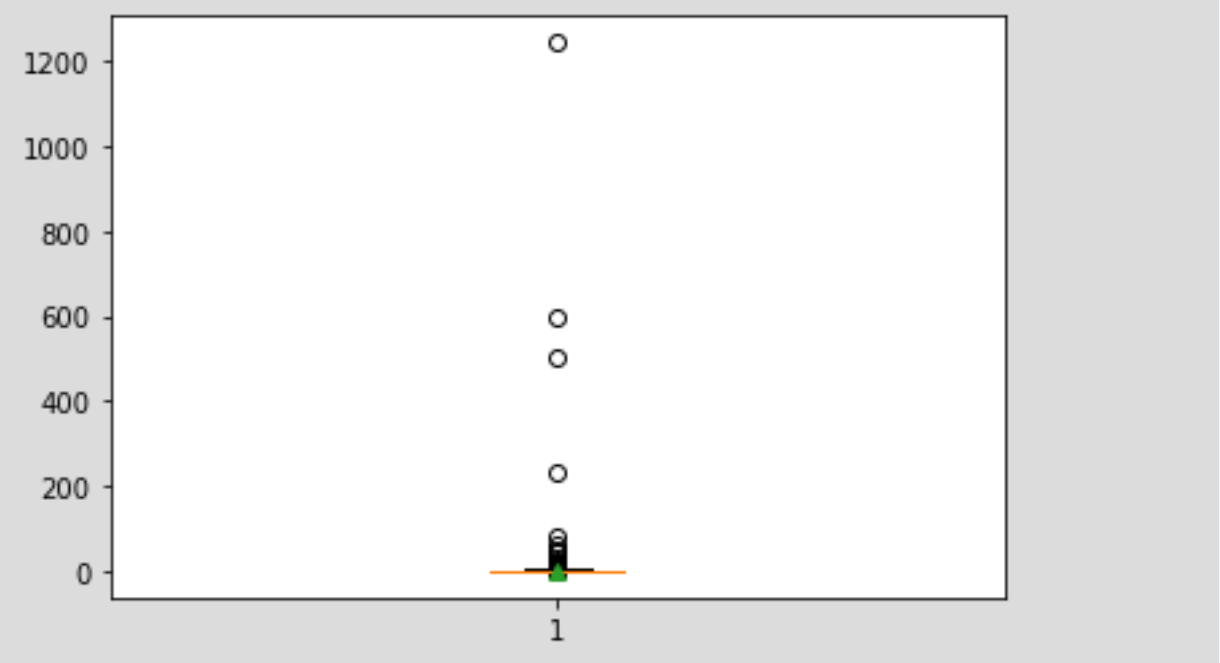
2.2.8

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
magnesium  \
target
0       13.744746    2.010678  2.455593          17.037288
106.338983
1       12.278732    1.932676  2.244789          20.238028
94.549296
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:

```
           MedInc   HouseAge  AveRooms  AveBedrms  Population
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_sal
es.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.c
sv', 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
```

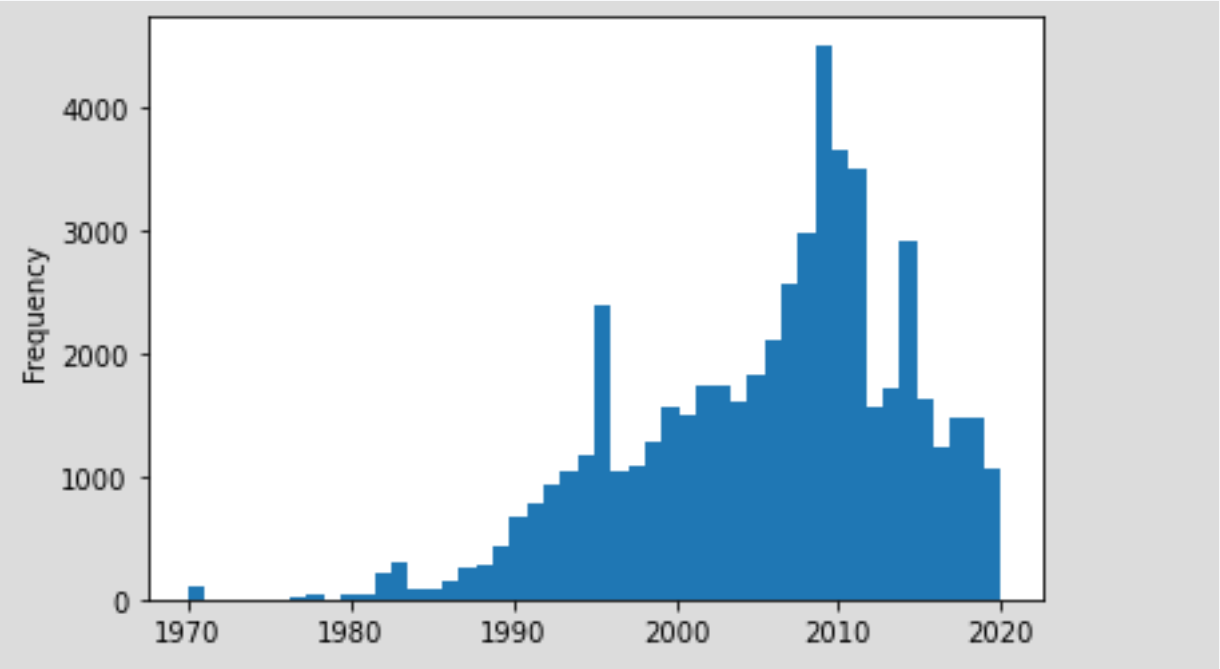
```
min                     0.010                 0.010
0.010
25%                     0.330                 0.410
0.410
50%                     0.630                91.930
109.960
75%                   749.235              1360.435
1357.553
max                2361808.290           7637857.360
1414168.060

       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_sal
es.csv', sep=',')

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

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

Program output:



3.1.5

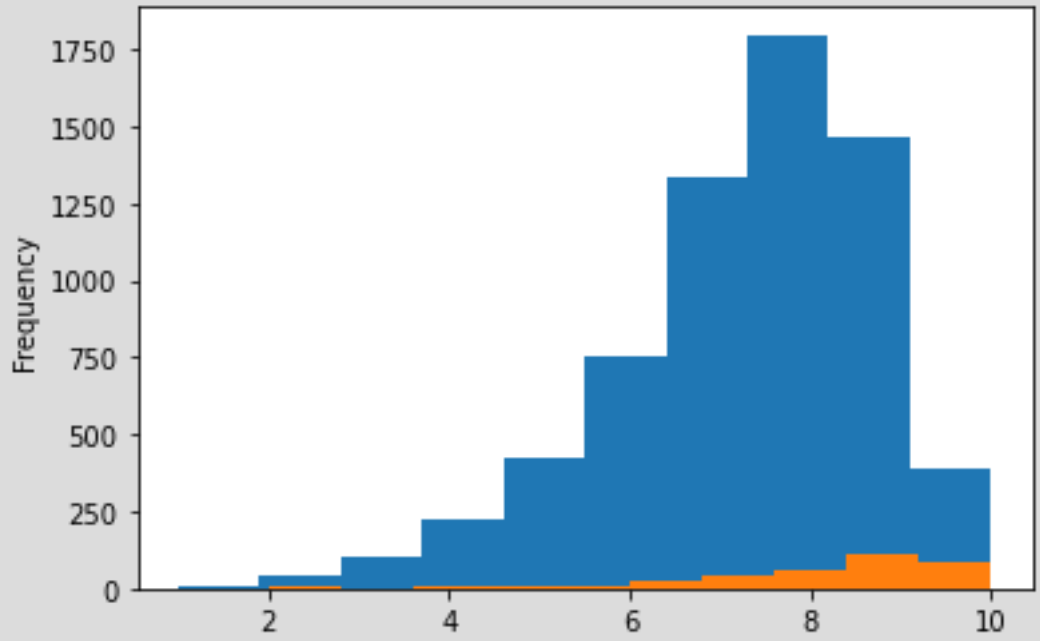
Next, we can take a look at the ratings of games by critics and users. On closer inspection of the records, we find that the **User_Score** variable contains a significant number of missing values. While we are left with few records after removing them we can observe through visualization that users 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_sal
es.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
```



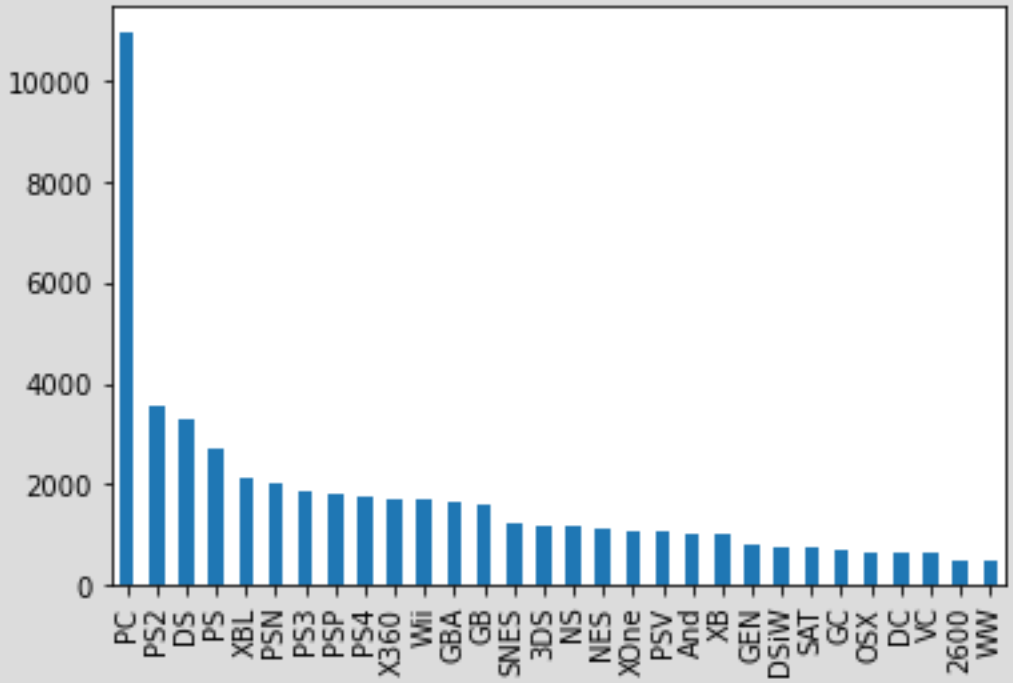
3.1.6

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_sal
es.csv', sep=',')
print(df['Platform'].describe())

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

Program output:

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



3.1.7

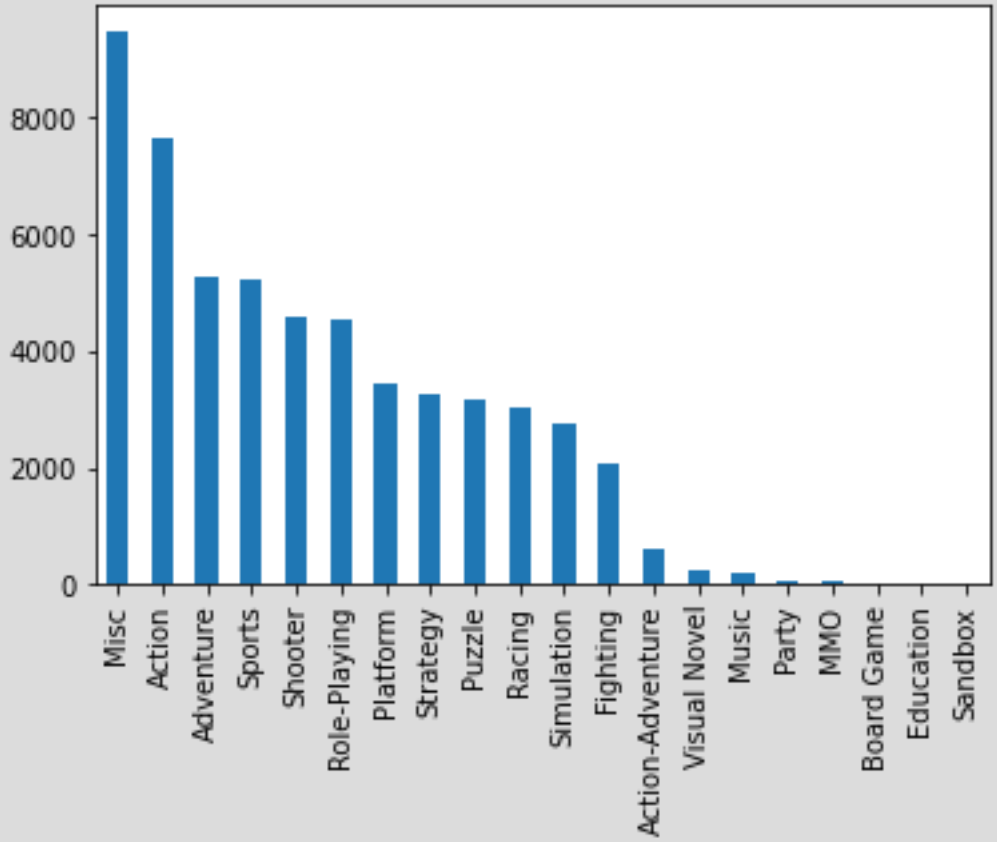
The genre of games gave us interesting results, where the most numerous games were from the miscellaneous genre, which can probably mean an increase in Indie games. The second most numerous games were action games, followed by adventure and sports games. On the other hand, strategy games were not as 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_sal
es.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

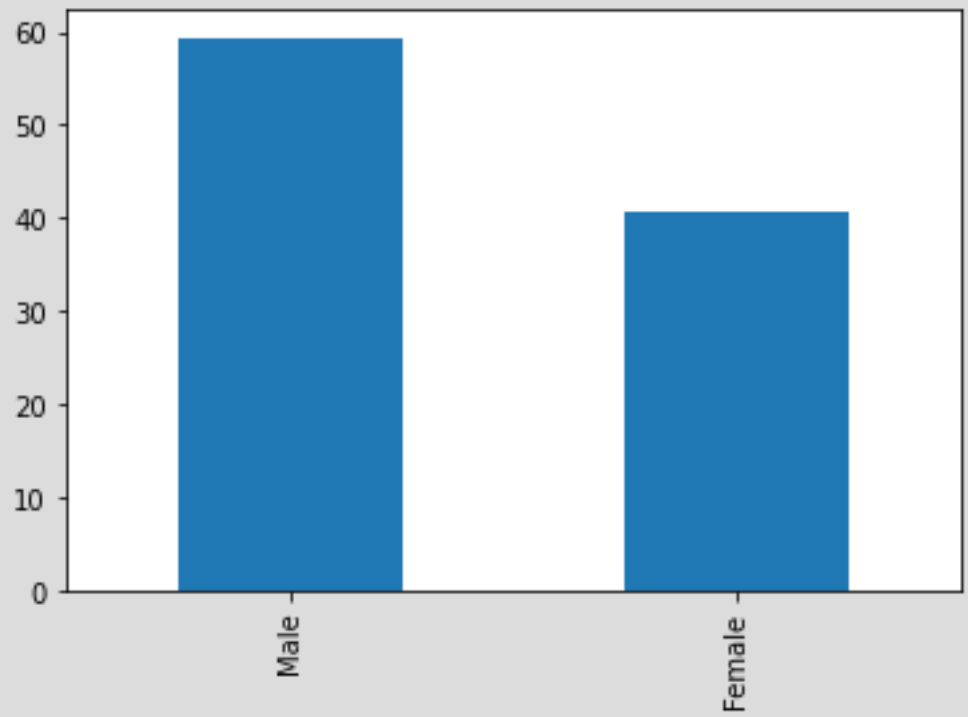
visualization and writing out both genders and the percentages rounded to two decimal places in the result.

```
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
```



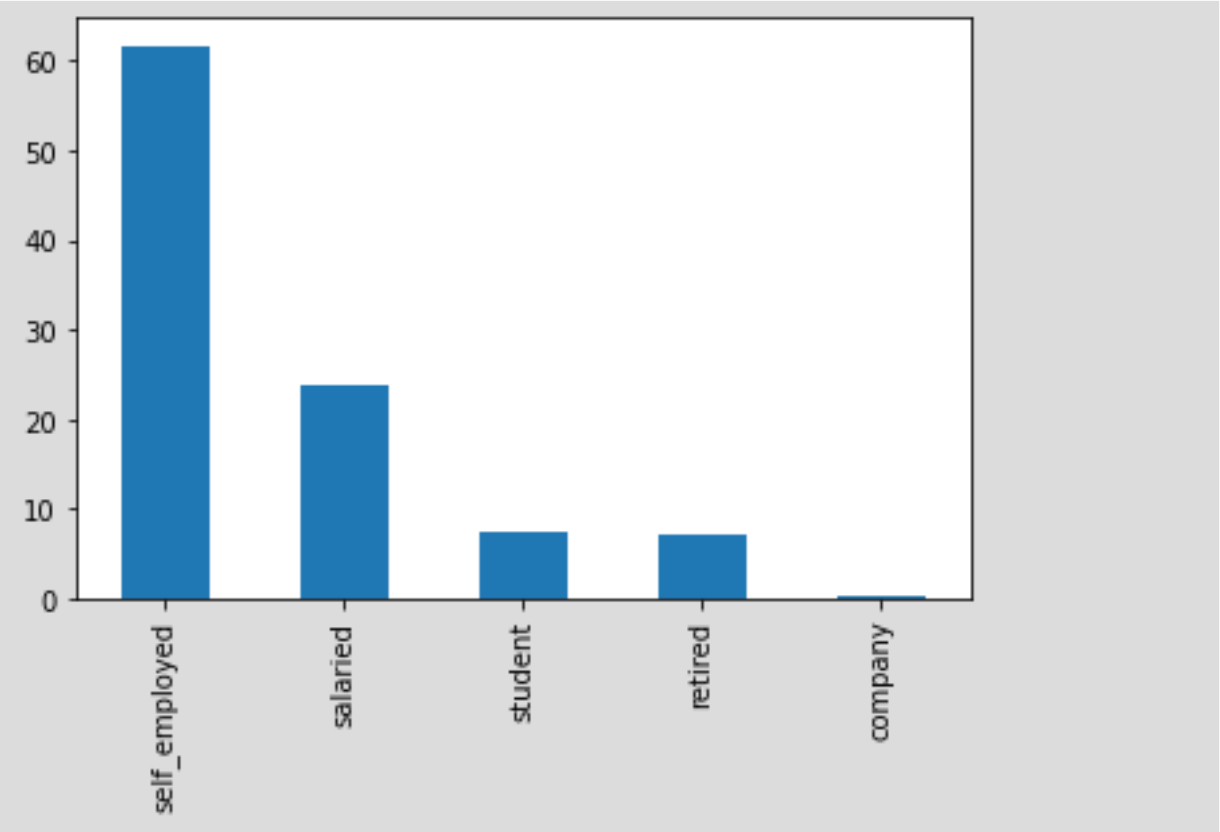
3.1.9

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

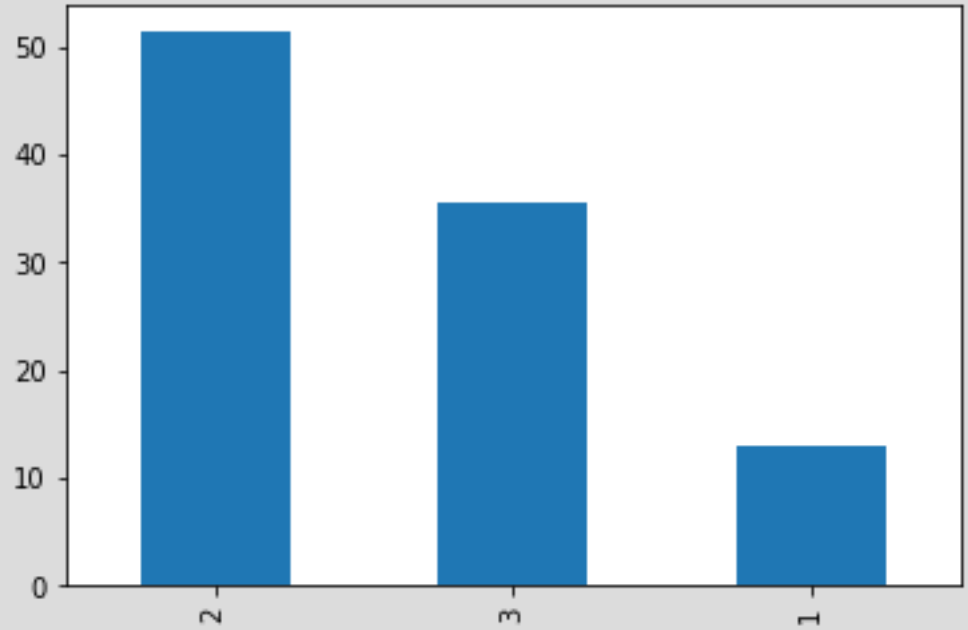
Load the data from the banking.csv file, which contains information about the bank's customers. After loading the data file find out what is the most common rating of the bank's customers (*customer_nw_category*). We recommend using the visualization and writing out the rating number and percentage rounded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

```
import pandas as pd

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.c
sv', 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(10
0).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 Bivariance 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.

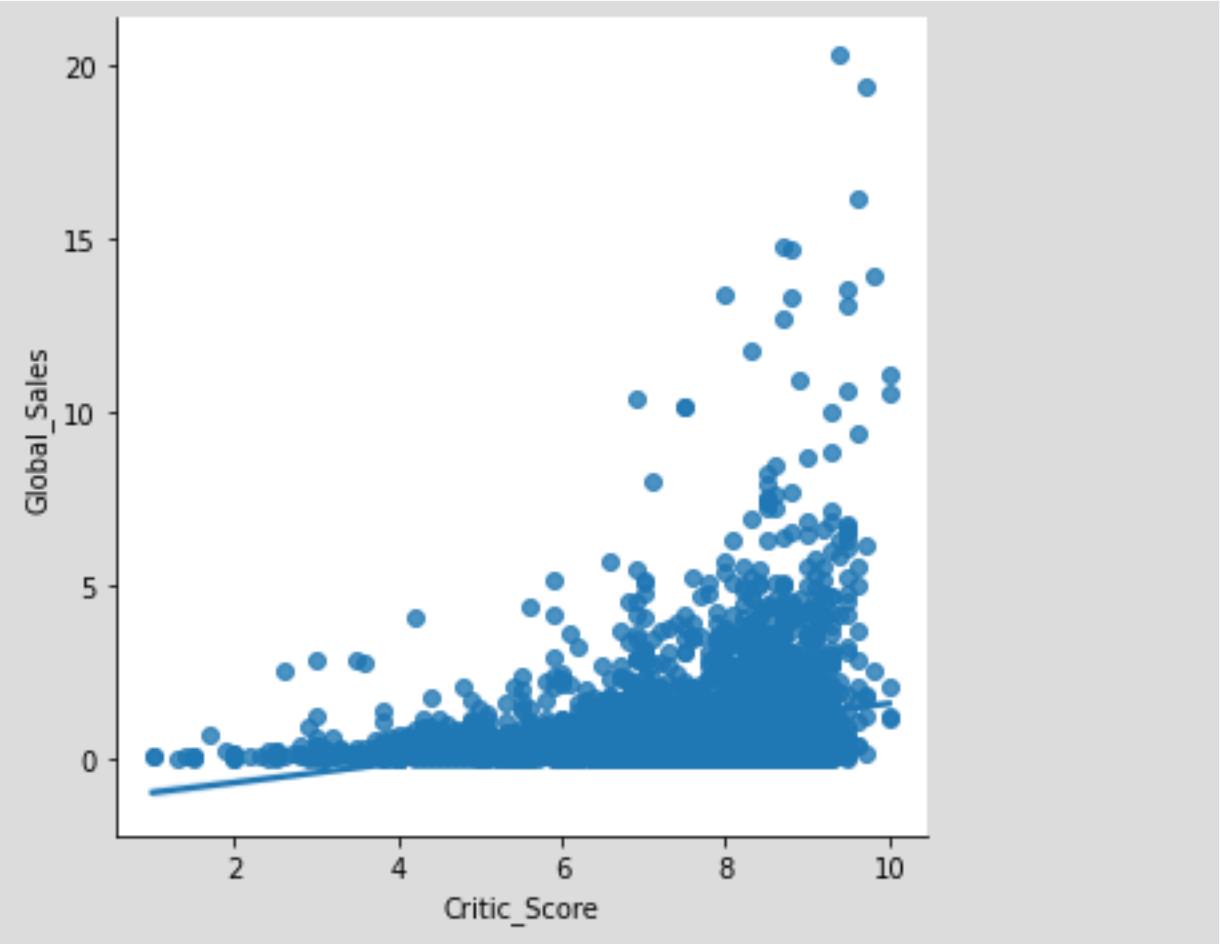
Let's look at our dataset of games. Using a scatter plot we can compare and see if critics' ratings have an impact on the worldwide sales of the games in question. From the graph, we can observe that sales increase as critics' ratings increase, so we can assume that ratings have an effect on the marketability 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 **lmplot()** 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_sal
es.csv', sep=',')
print(df.info())
#df.plot(x='Critic_Score',y='Global_Sales',kind='scatter') #
using pandas
sns.lmplot(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.

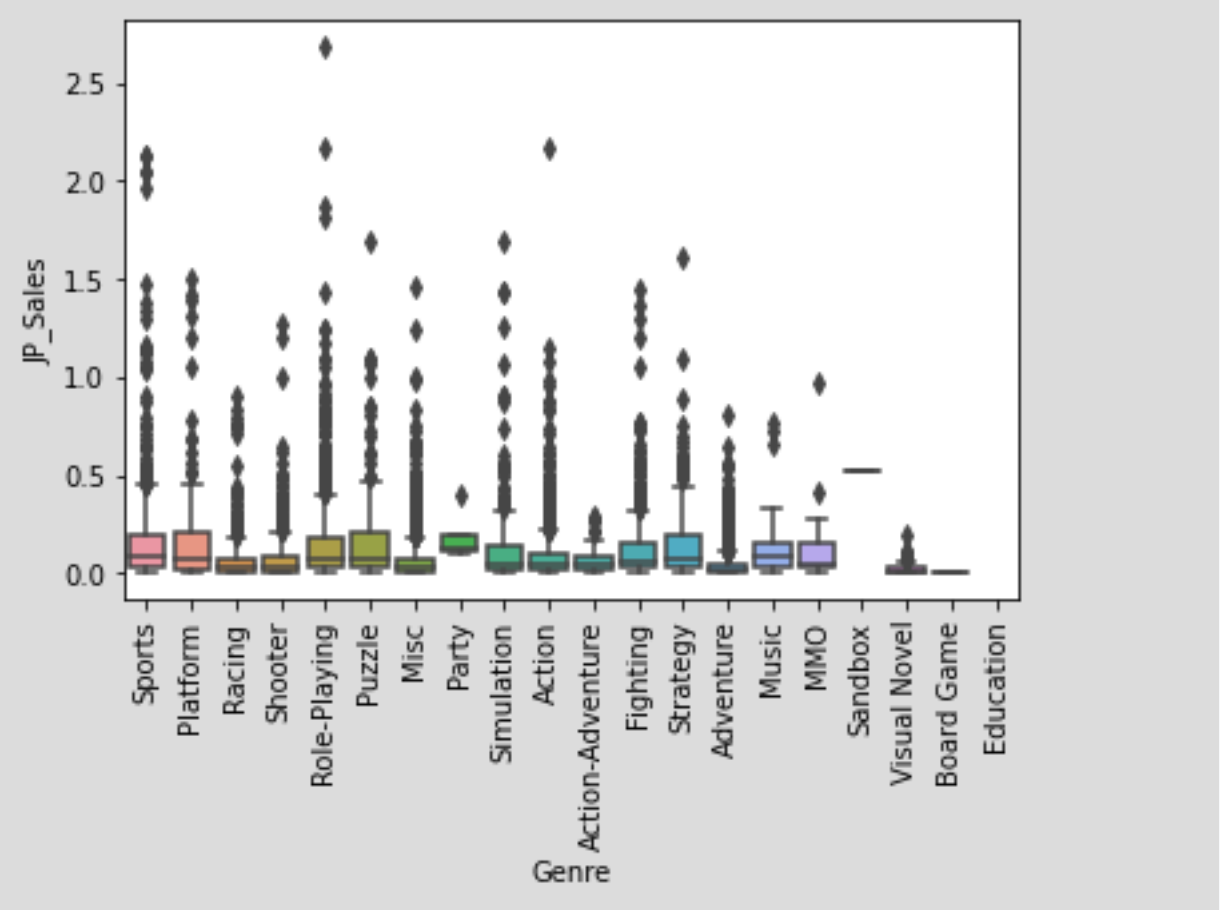
We can observe that the yield from the Role-playing and Sports genres is higher than that from the Racing and Shooter genres. Most genres contain outliers that represent high returns.

```
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_sal
es.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()**.

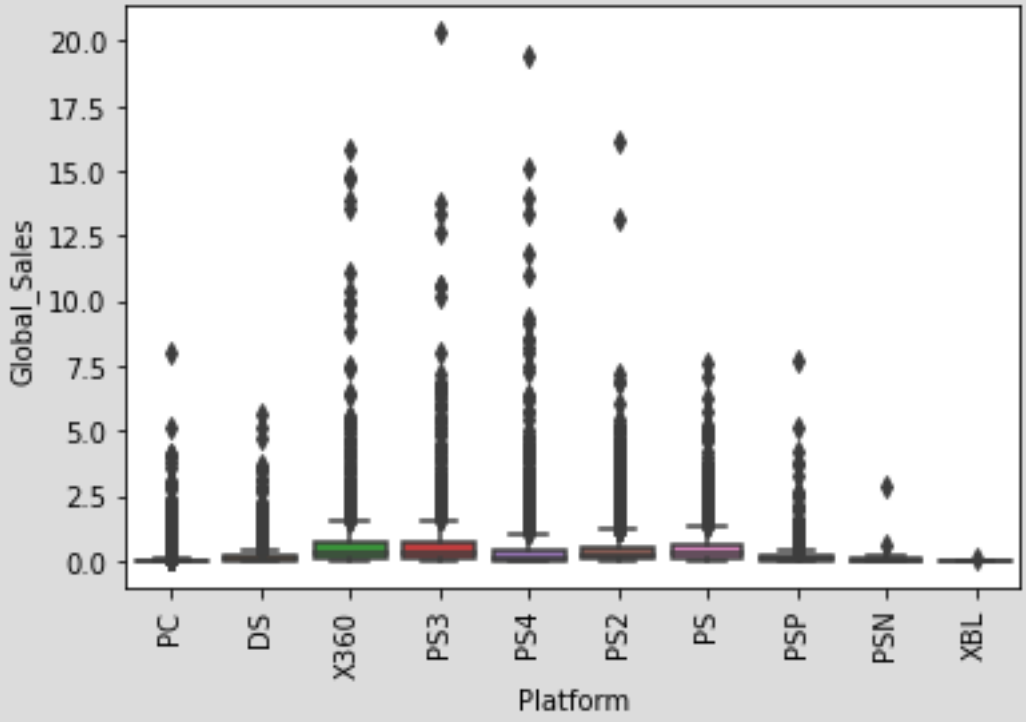
A surprising result from the graph is that the revenue of the most used platform (PC) is lower than for example the different PlayStation types.

```
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_sal
es.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
```

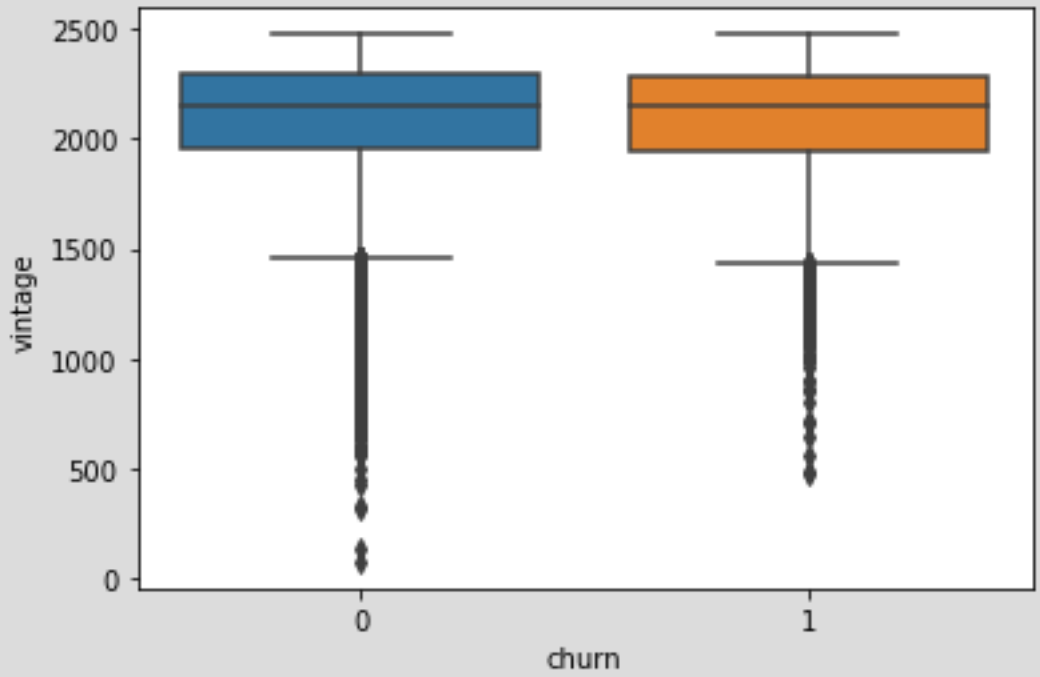


3.2.4

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.c
sv', 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.

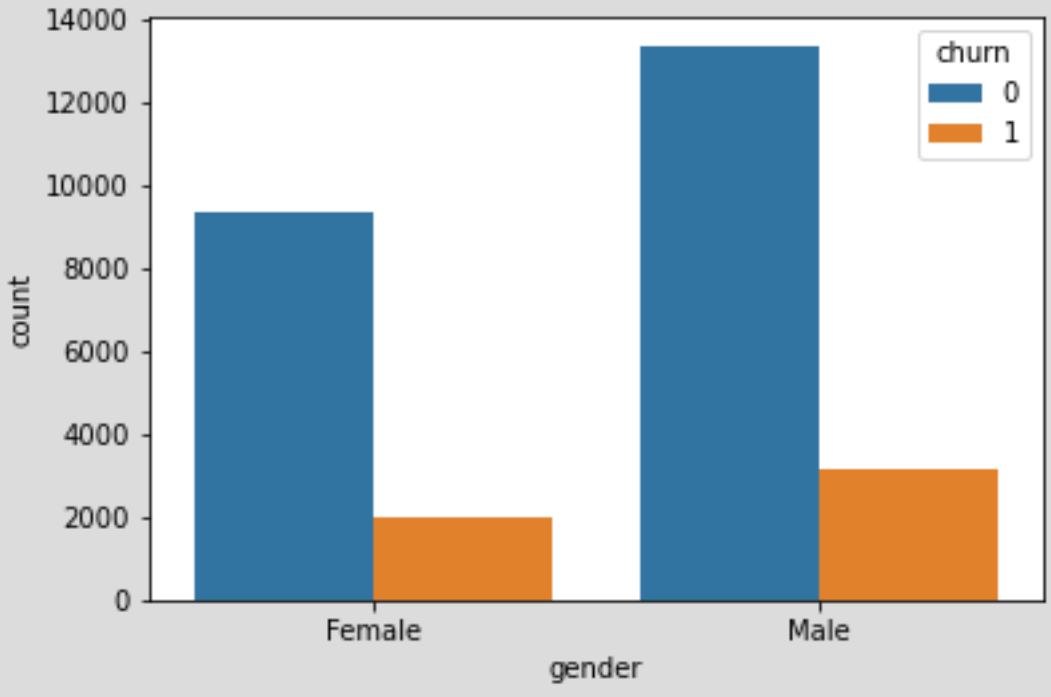
```
male churn: 54.25% female churn: 45.75%
```

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

```
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.c
sv', 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(nor
malize=True).mul(100).round(2))
print(dfd['churn'].loc[dfd['gender']=='Female'].value_counts(n
ormalize=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.

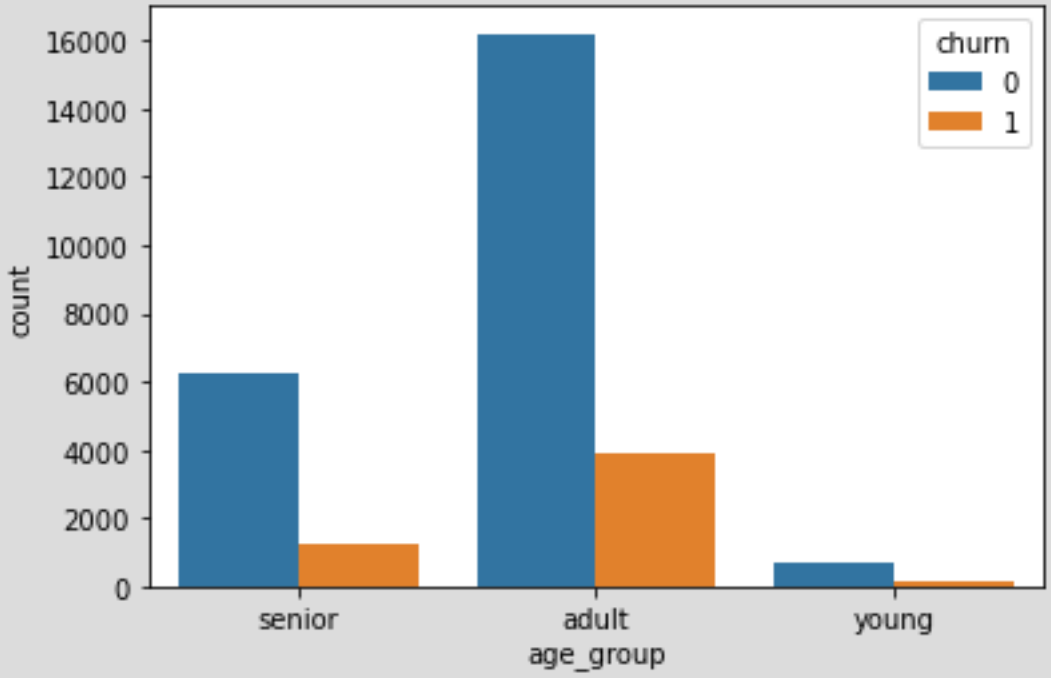
```
young: 50.24% adult: 27.75% senior: 22.01%
```

```
import pandas as pd
import seaborn as sns

df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.c
sv', 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_count
s(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

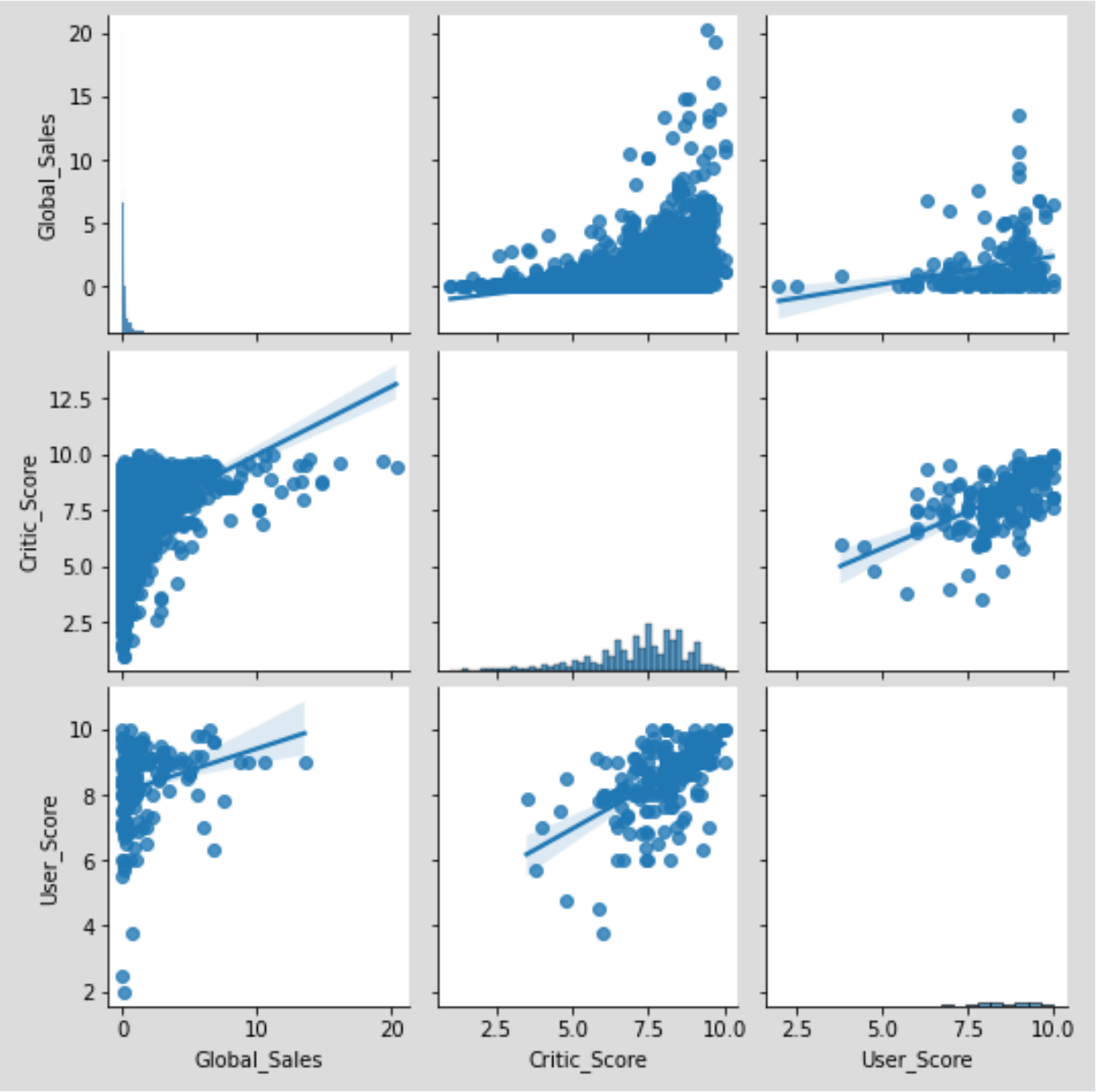
3.3.1

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_sal
es.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.

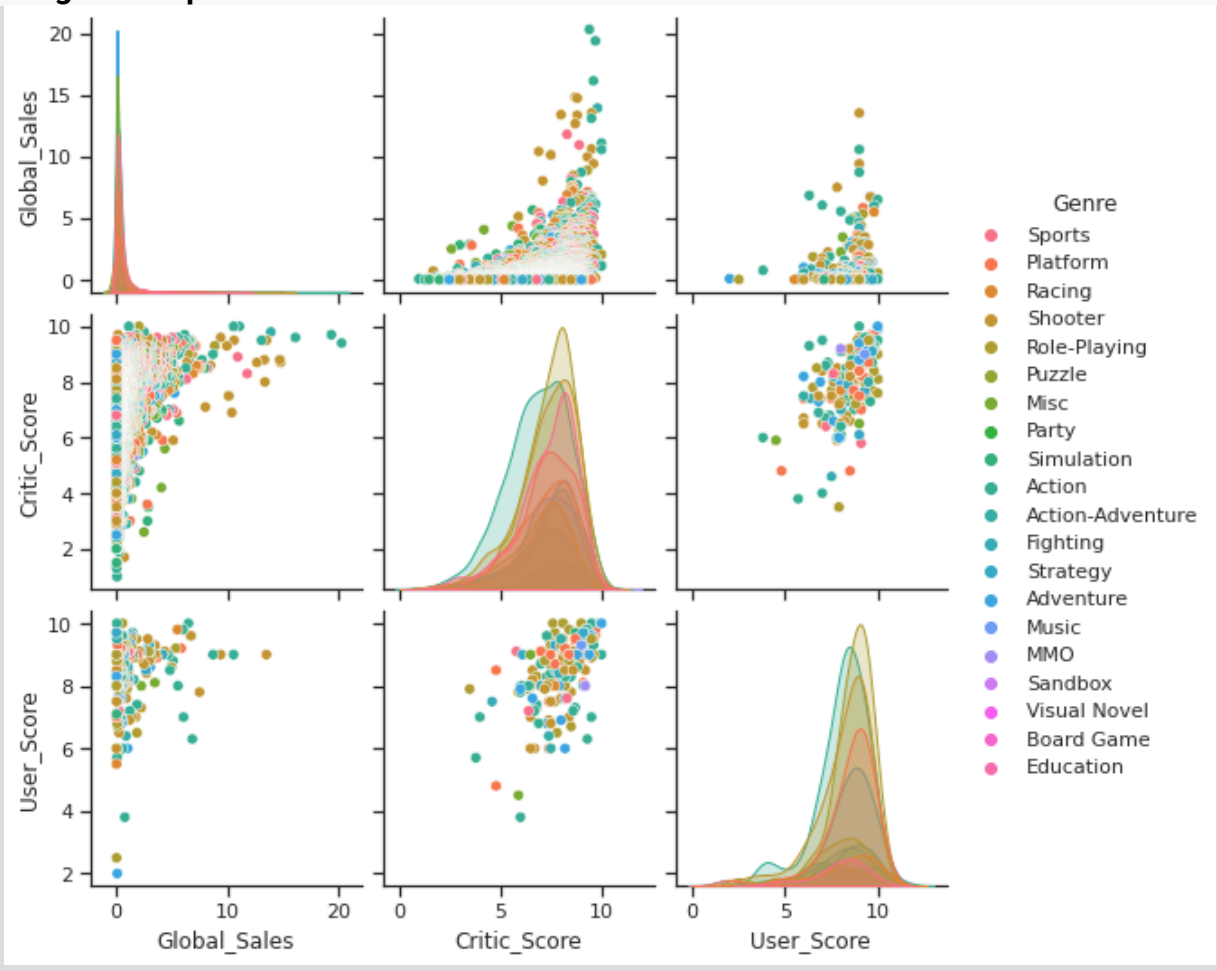
3.3.2

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_sal
es.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_sal
es.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.c
sv', 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.c
sv', 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_sal
es.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
```

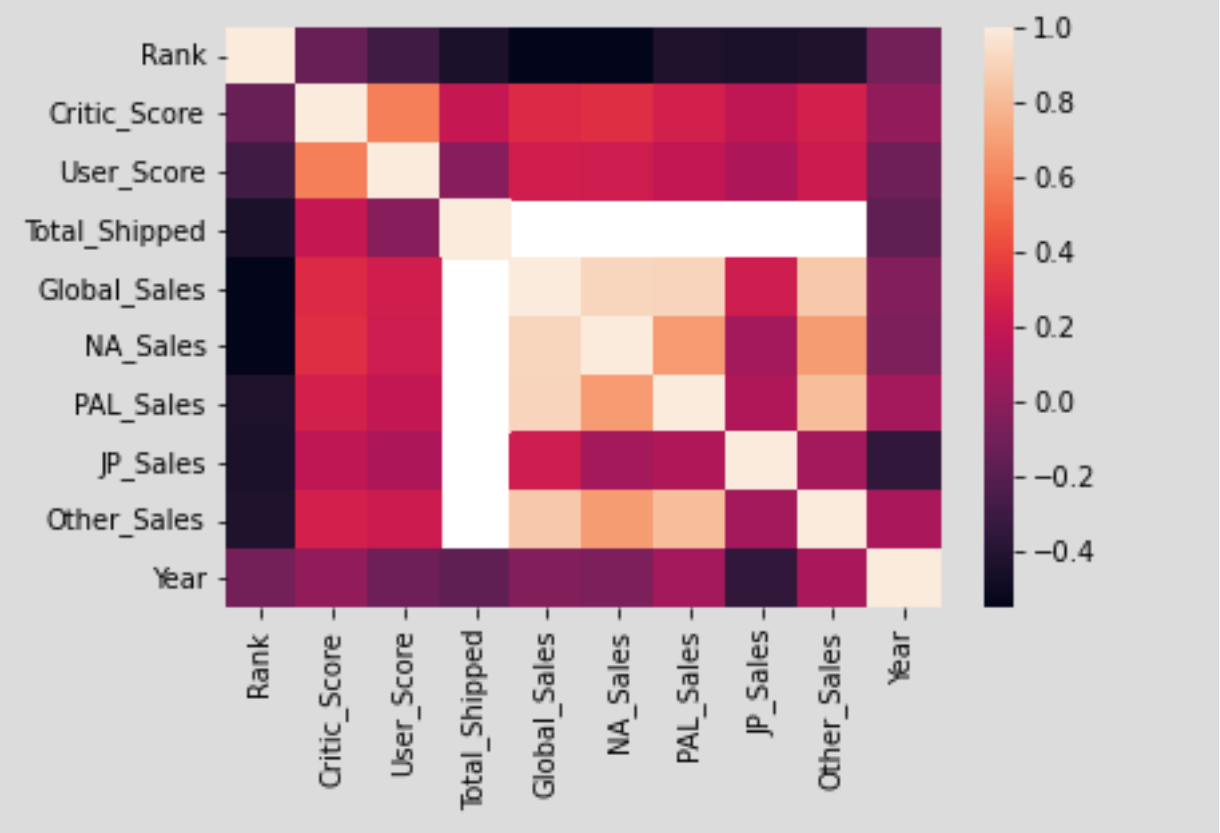
```
Year           -0.097345      0.015670   -0.116728      -
0.169701

               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_sal
es.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.c
sv', 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
```

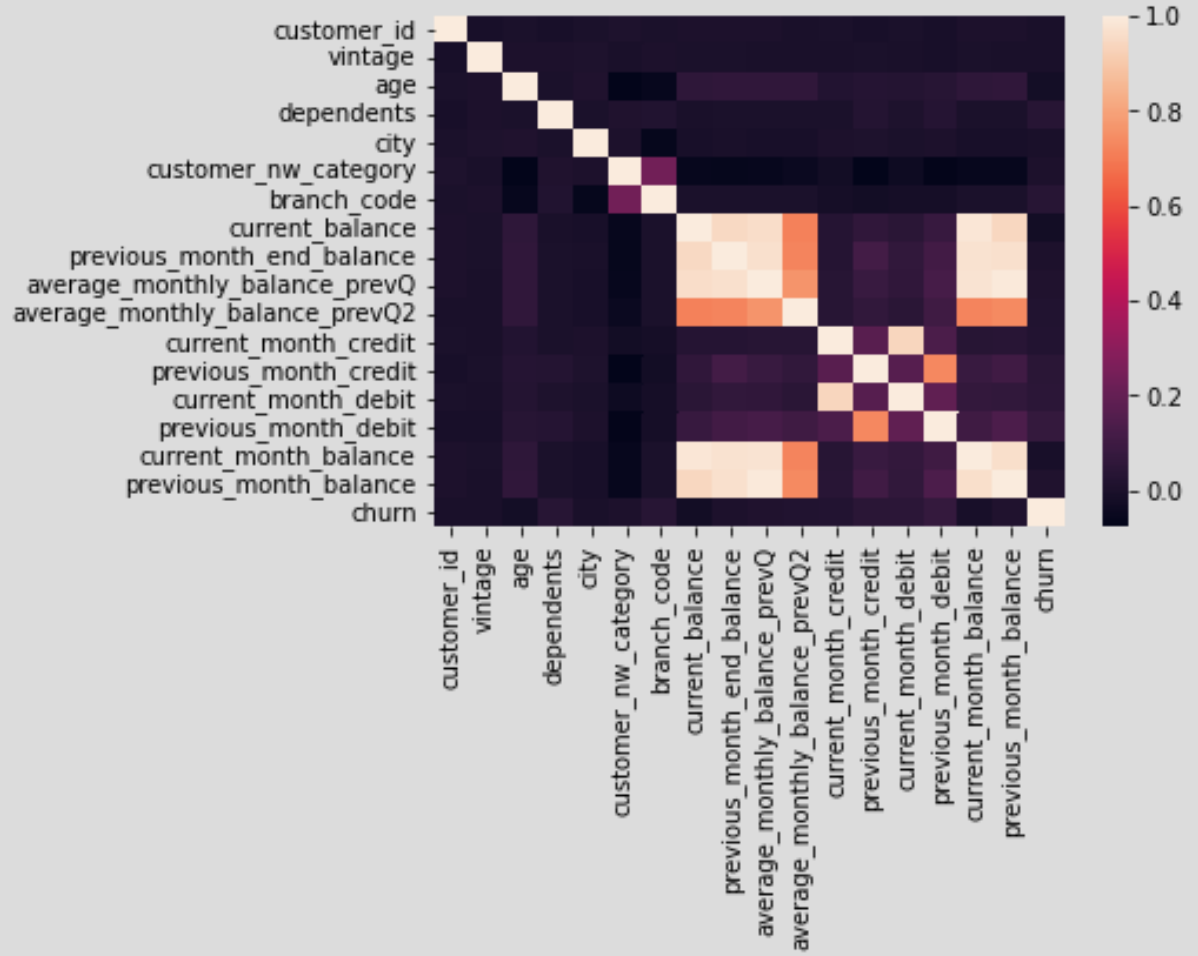
```
vintage                                    -0.004617
-0.000169
age                                         0.023840
0.029961
dependents                                  0.003260
0.025054
city                                        0.004118
0.008087
customer_nw_category                       -0.025254
-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.

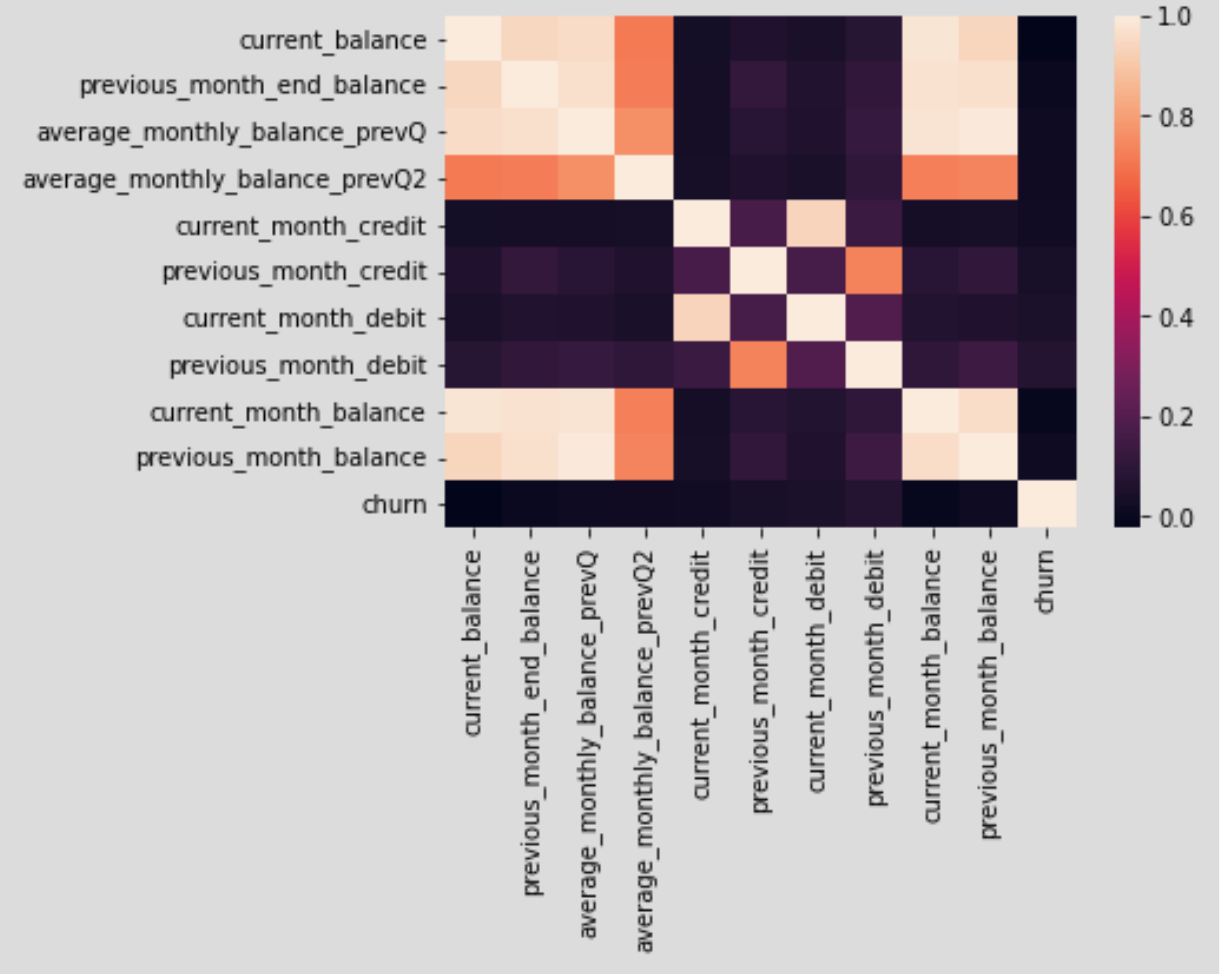
```
import pandas as pd
from scipy import stats
import seaborn as sns
```

```
df =
pd.read_csv('https://priscilla.fitped.eu/data/pandas/banking.c
sv', sep=',', decimal='.')
print(df.info())
dfd =
df[['current_balance','previous_month_end_balance','average_mo
nthly_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 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.cs
v', 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.cs
v', sep=',')
# calculate counts of employess 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.cs
v', 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.cs
v', 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

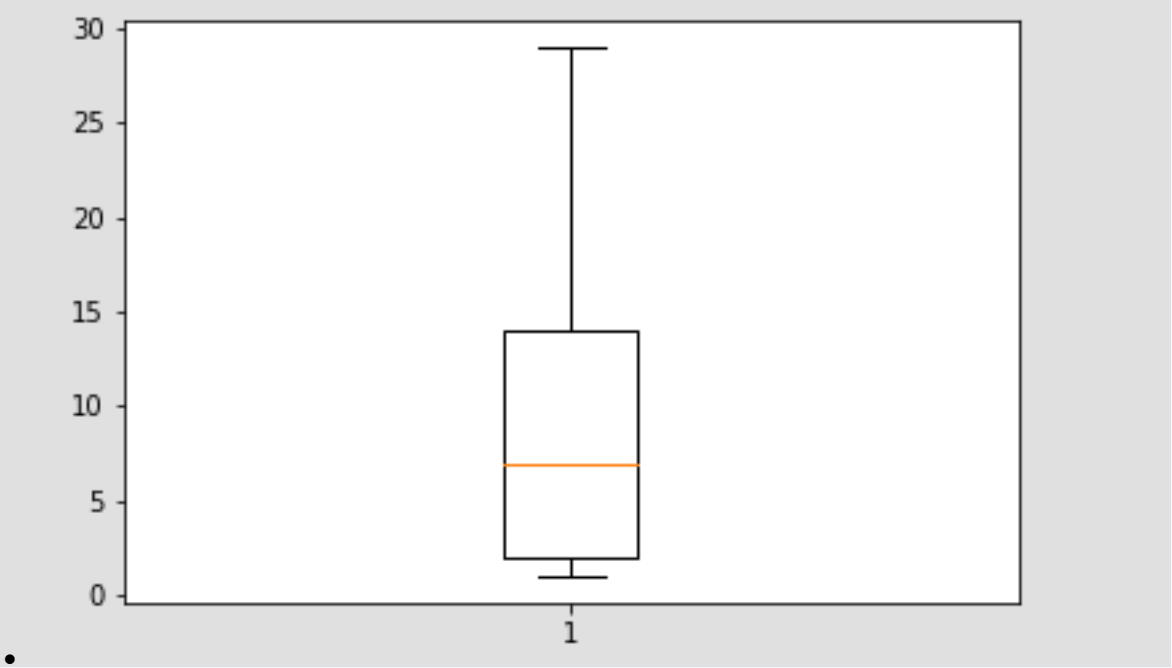
What is the standard deviation of the **age** of employees? (round the result to 2 decimal places)

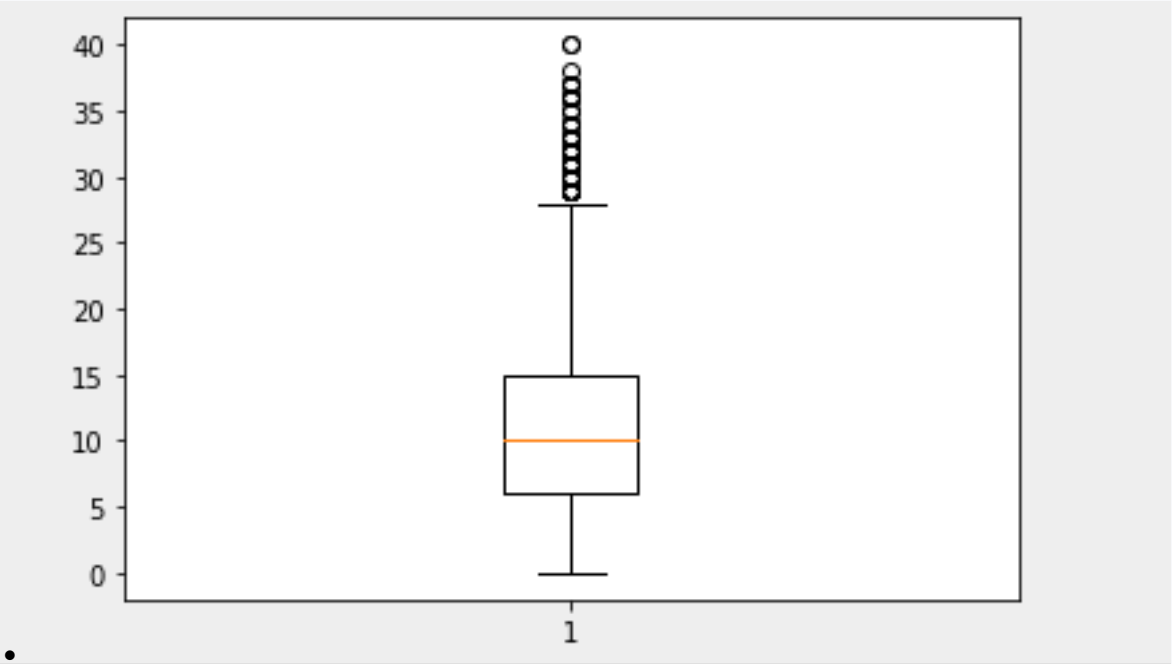
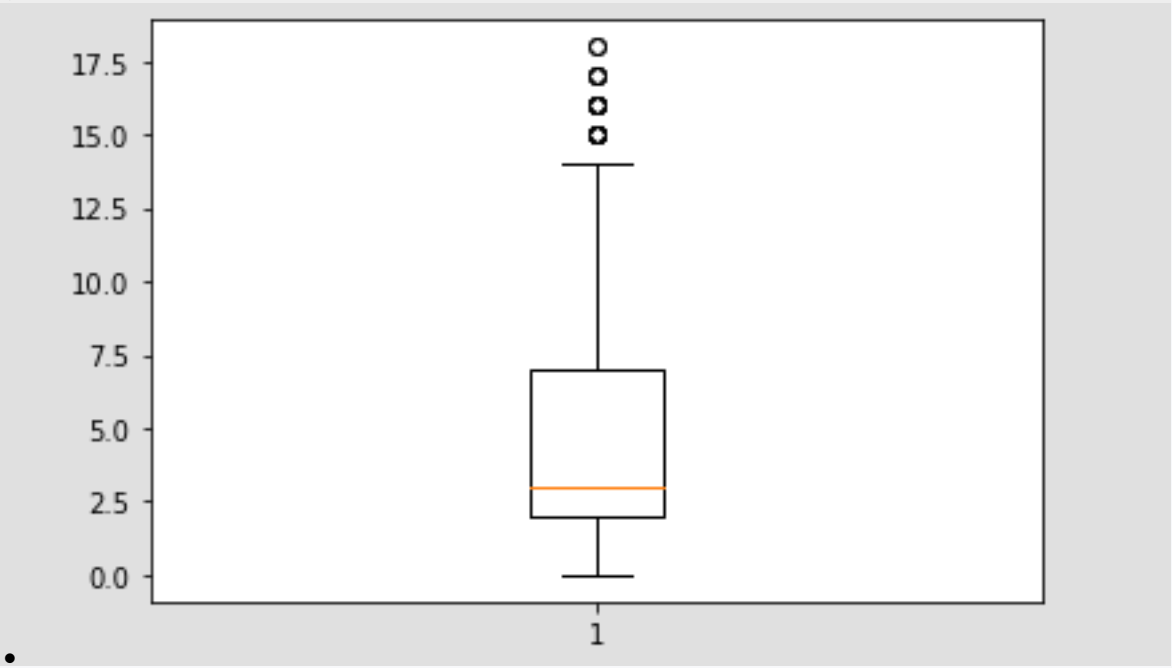
```
# import library
import pandas as pd
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.cs
v', sep=',')
# calculate the standard deviation of the variable Age using
the pandas library
```

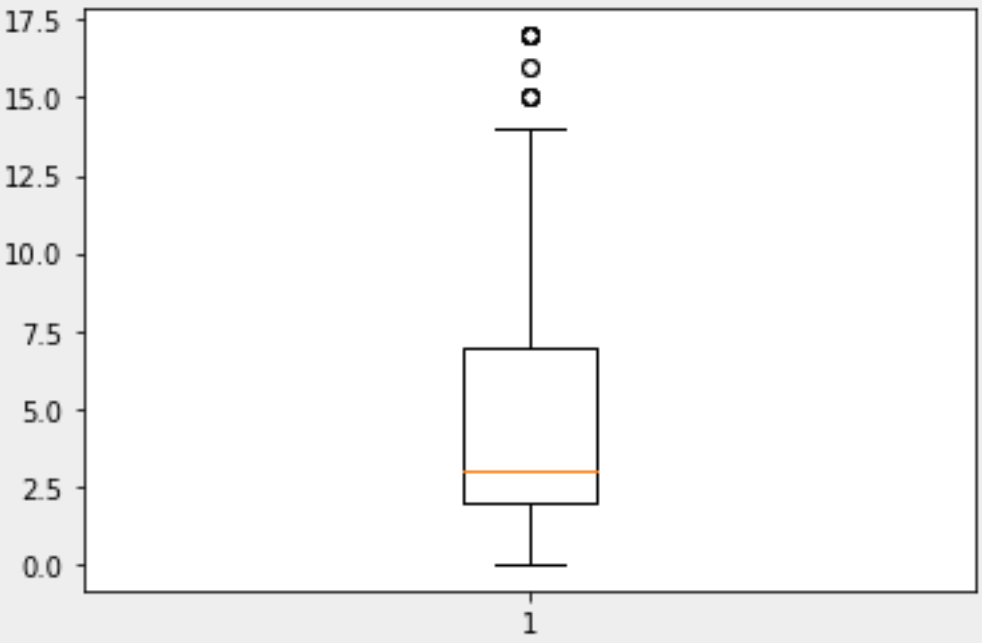
4.1.9

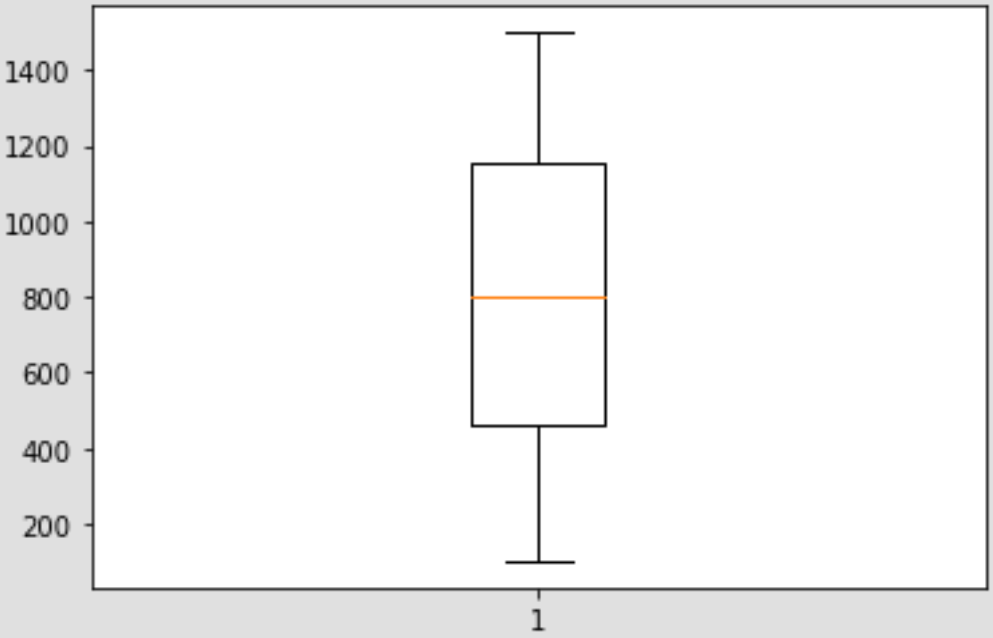
Use the ***Matplotlib*** library to plot a **box plot** for the **distance of the employee's home from the work location**. Which of the following box plots visualizes the distribution of this variable?

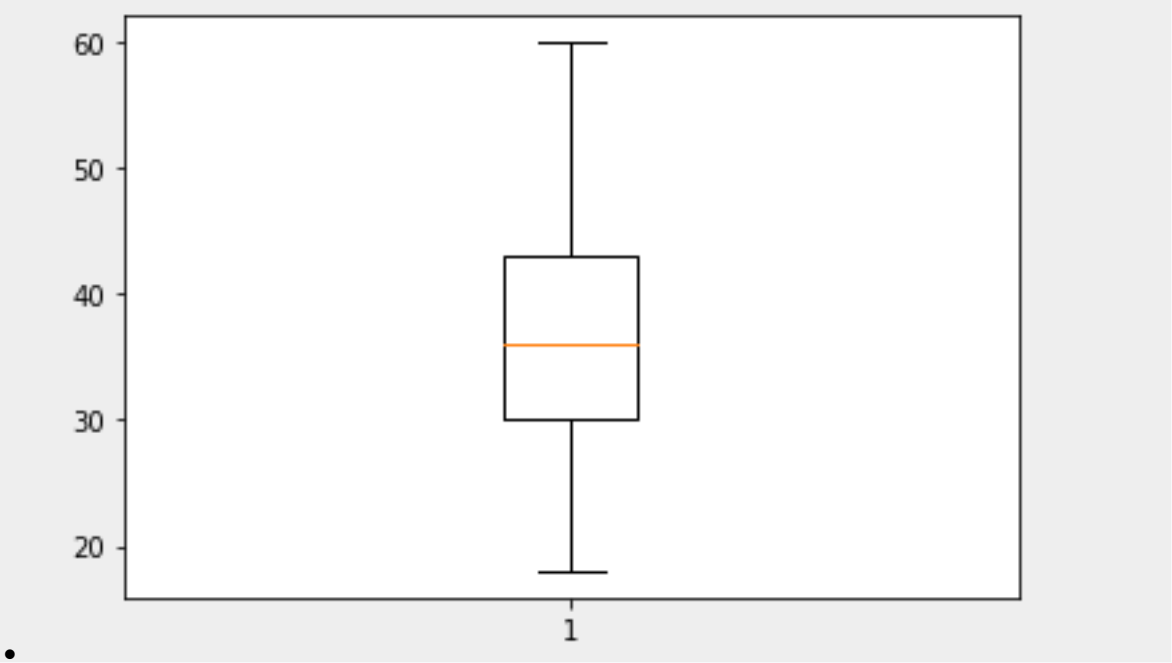
```
# import library
import pandas as pd
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# display a boxplot for distance from home using the matplotlib library
```



- 
40
35
30
25
20
15
10
5
0
1
- 
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1


1400
1200
1000
800
600
400
200
1



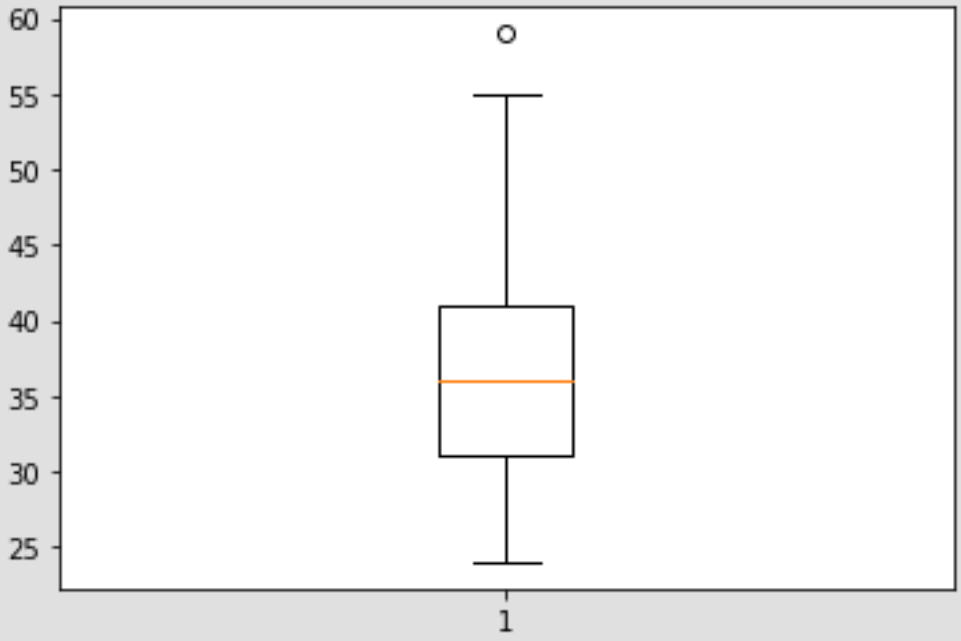
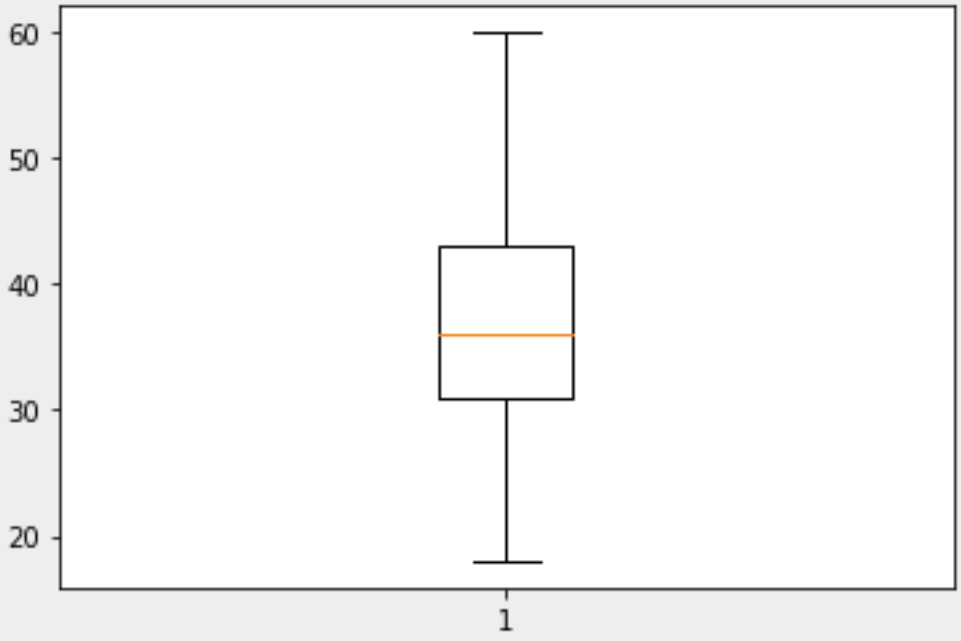
-

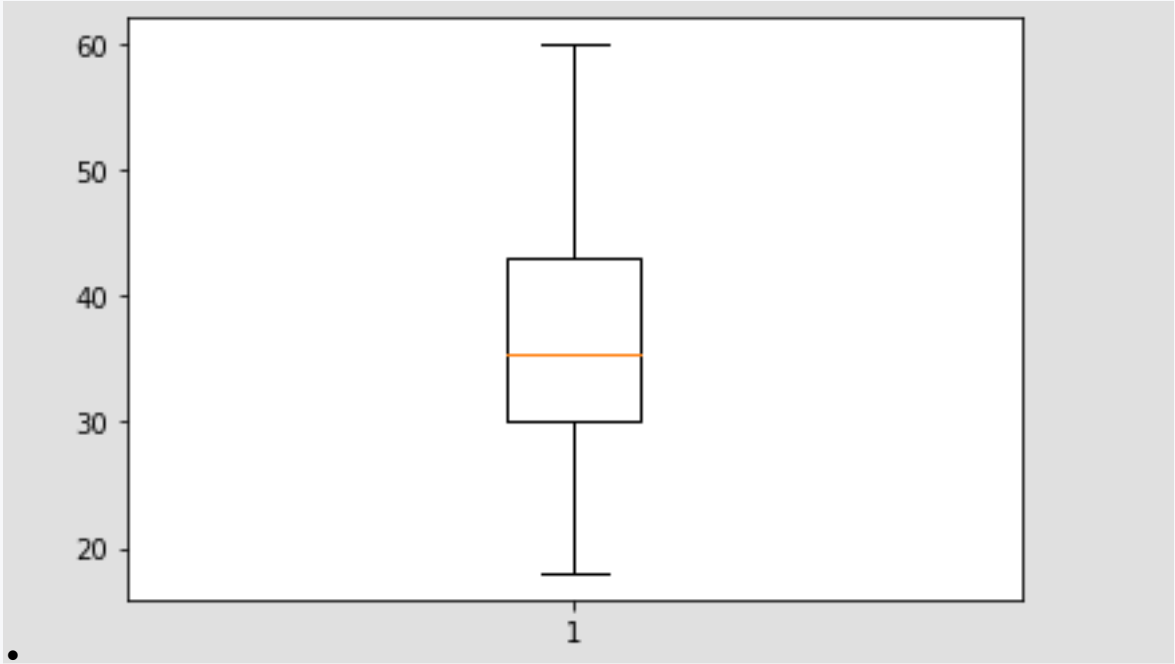
4.1.10

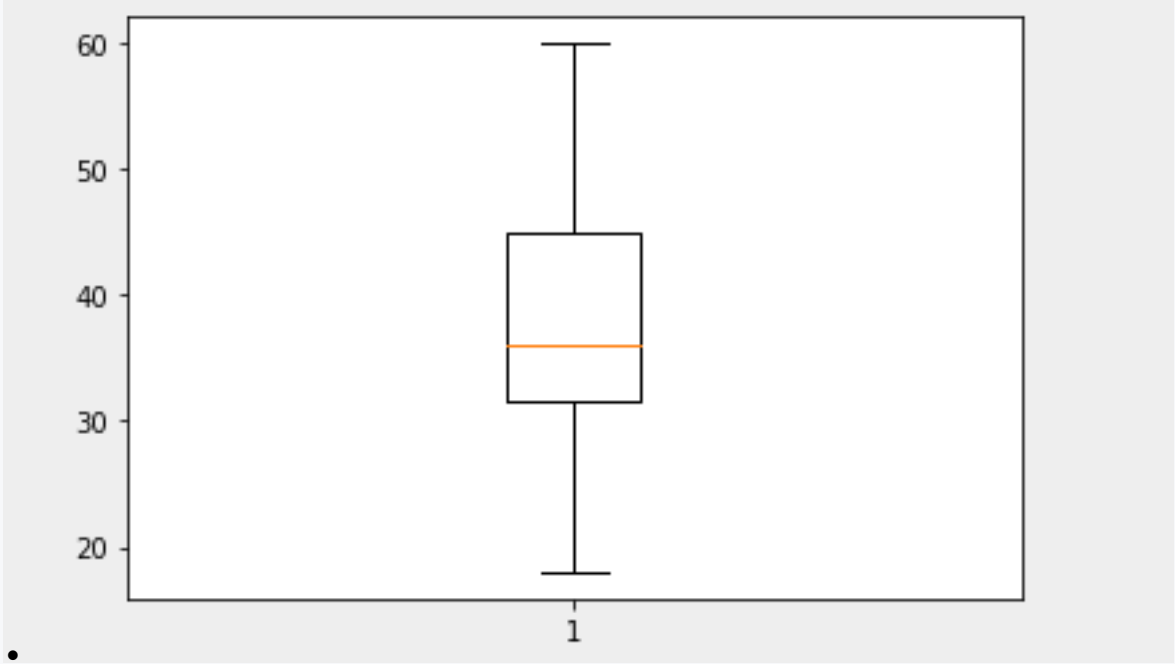
Plot a **box plot** of the distribution of the **age** of employees who have graduated with a degree in **human resources**.

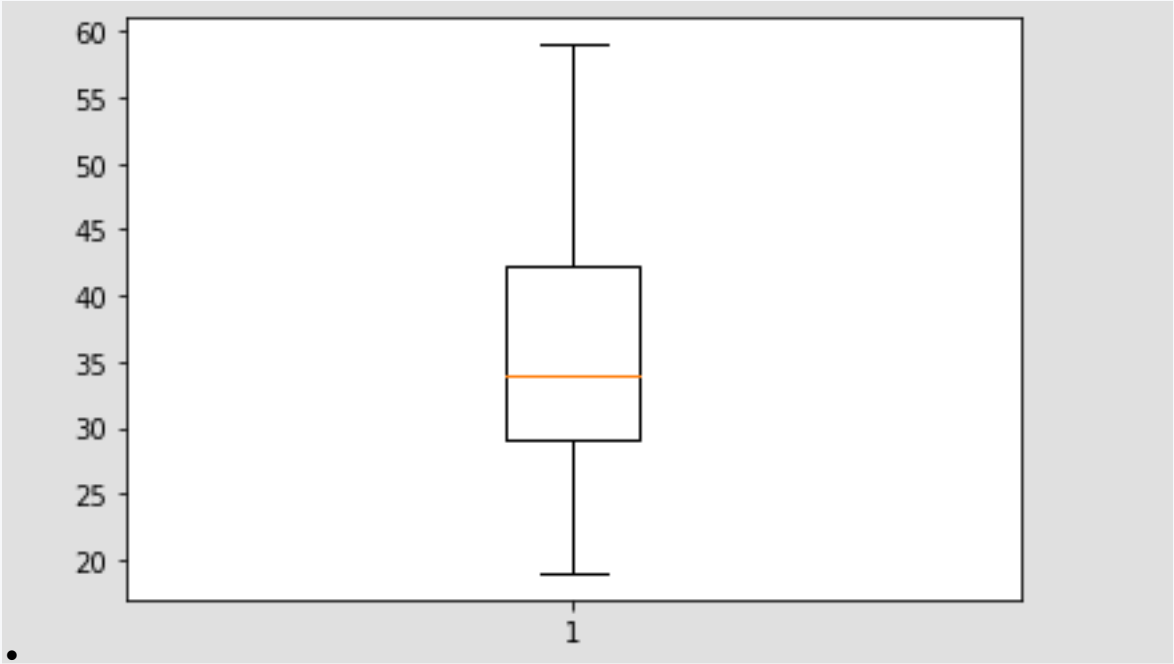
Which of the following plots shows this?

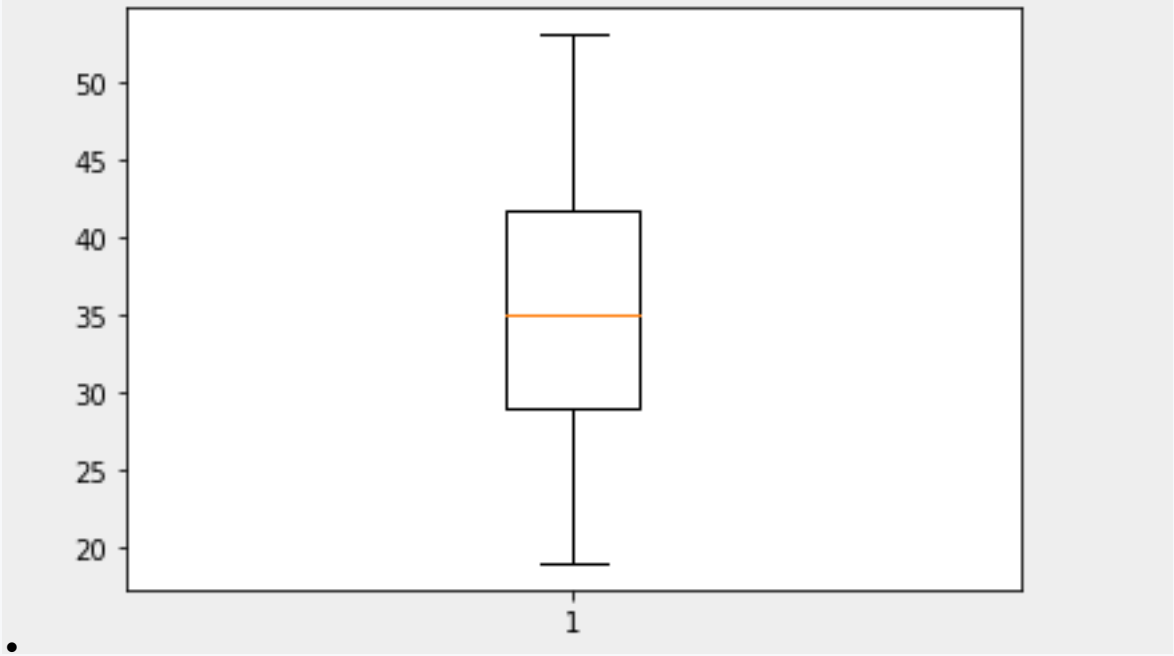
```
# import library
import pandas as pd
import matplotlib.pyplot as plt
# read csv
df =
pd.read_csv('https://priscilla.fitped.eu/data/nlp/employees.csv', sep=',')
# display a box plot for the age of employees who have a degree in human resources
```

- 
60
55
50
45
40
35
30
25
1
- 
60
50
40
30
20
1


60
50
40
30
20
1


60
50
40
30
20
1

- 

- 

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 homogenously 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.cs
v', 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.cs
v', 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.cs
v', 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.cs
v', 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.cs
v', 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.cs
v', 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:

- PassenderID - 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.c
sv', 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:

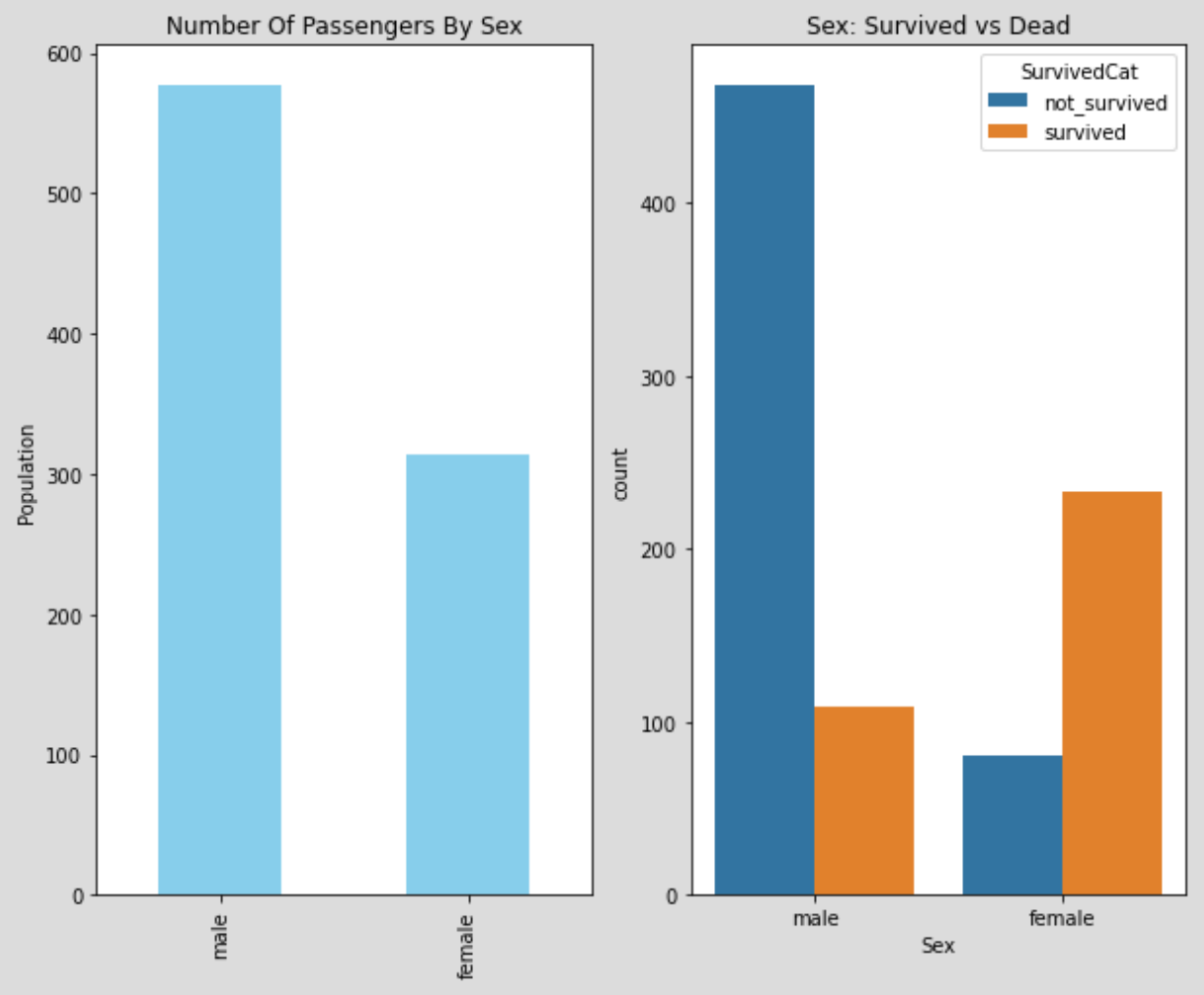
```
74.2 % of women who survived.
18.89 % of men who survived.
```

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

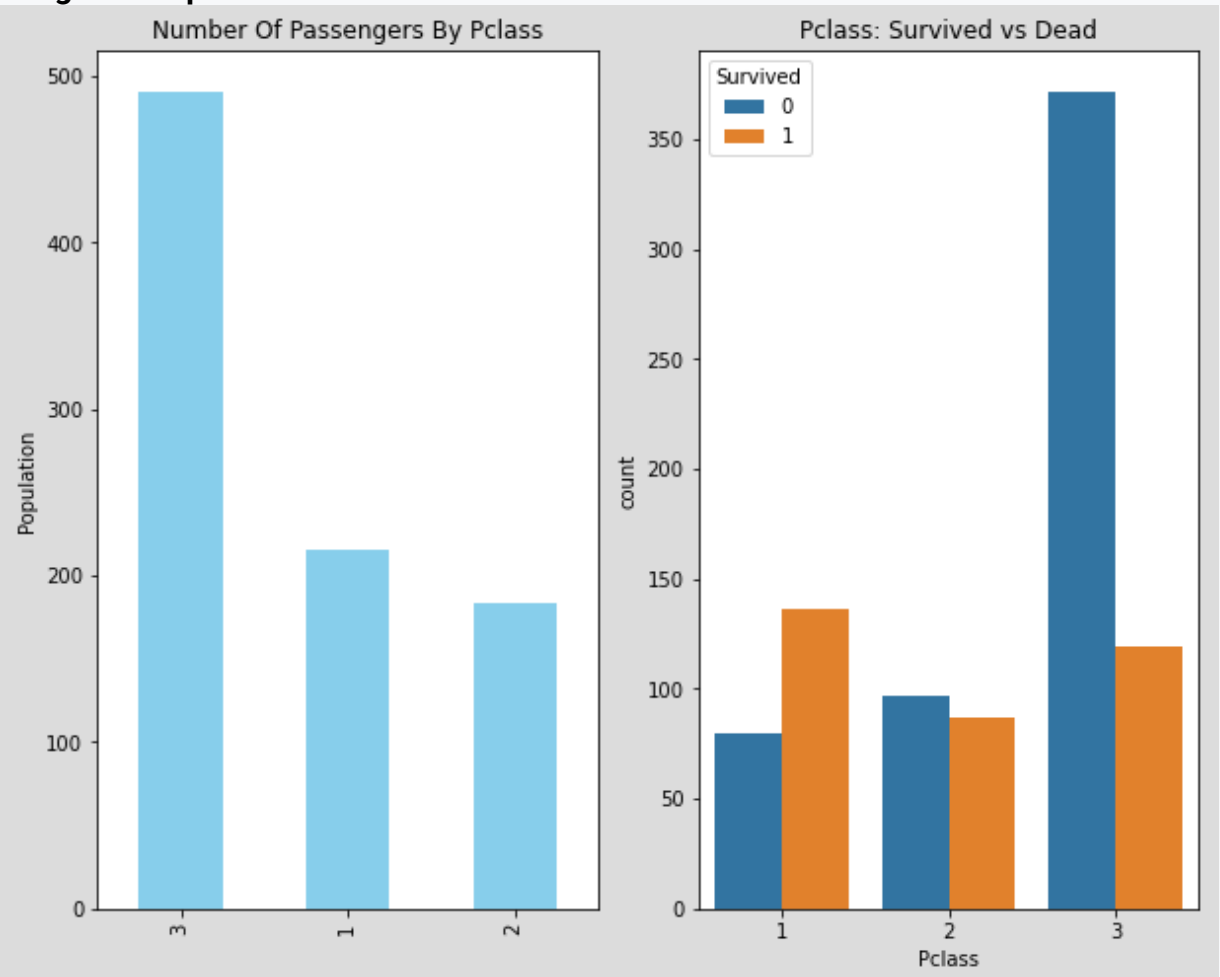
Load the data from the dataset titanic.csv (the file is located at https://priscilla.fitped.eu/data/pandas/titanic.csv). Examine the data in the dataset and find out the distribution of the number of passengers in each class. Write the result in numbers and in the following format:

```
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
```

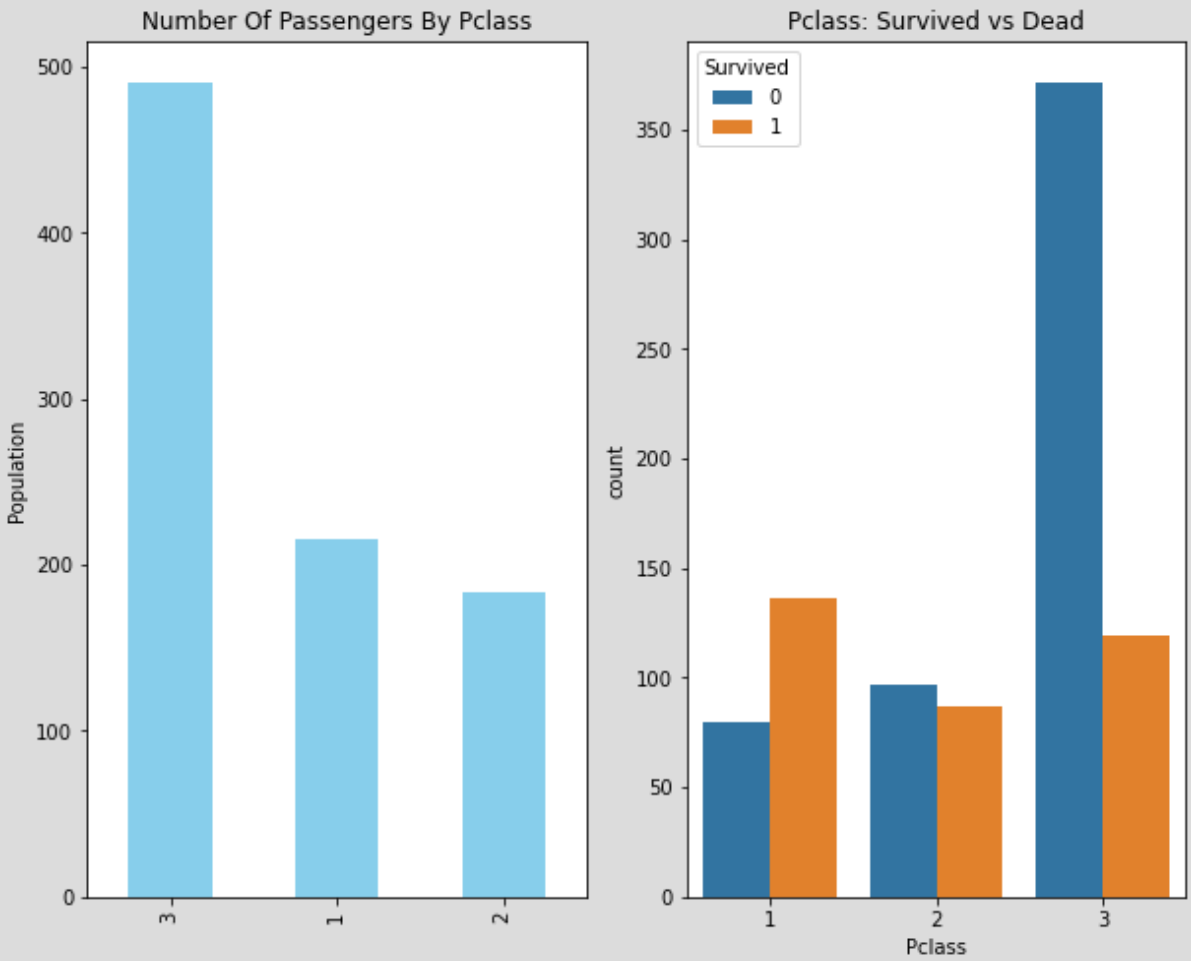
```
1    216
2    184
Name: Pclass, dtype: int64
```

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.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
```

- 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

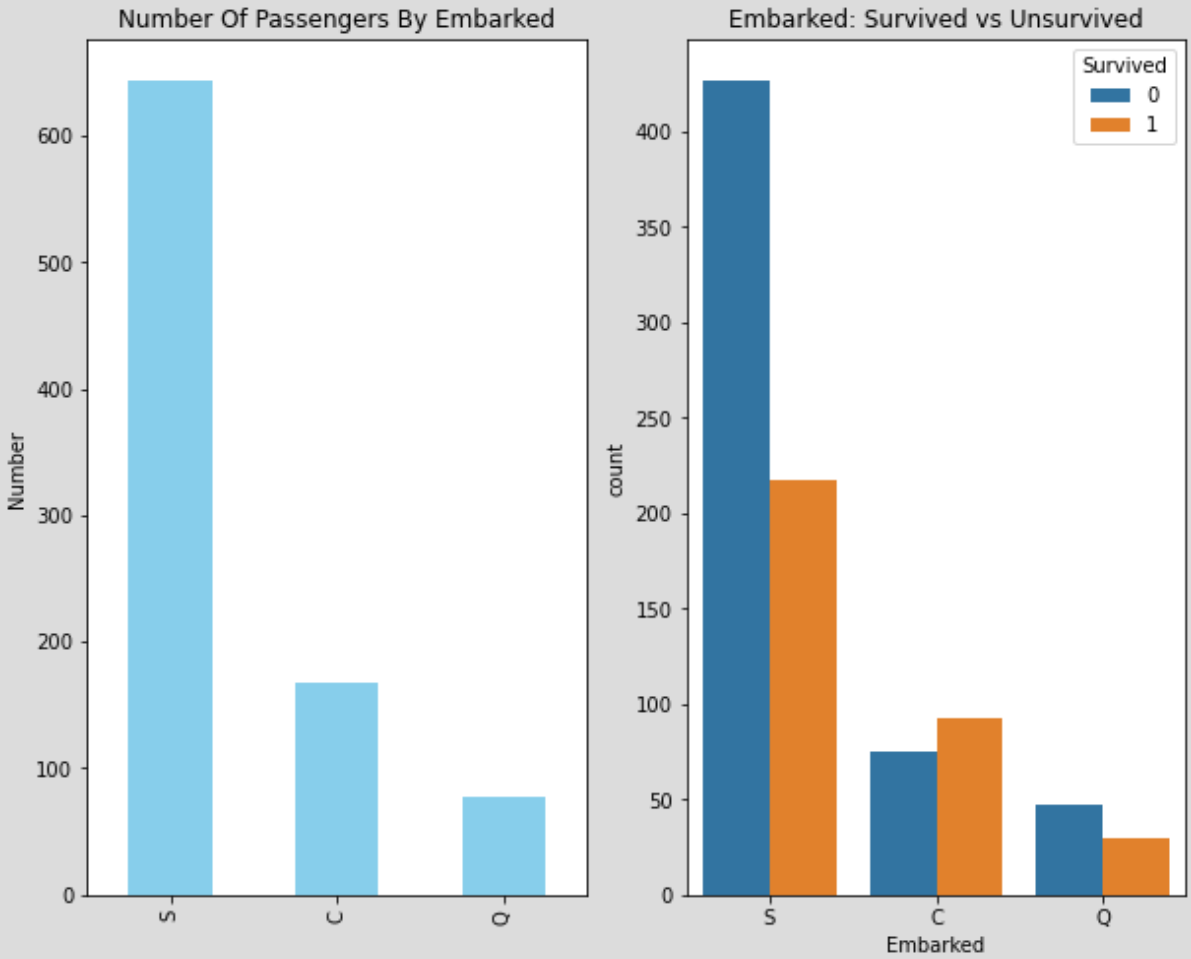
5.1.8

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
```

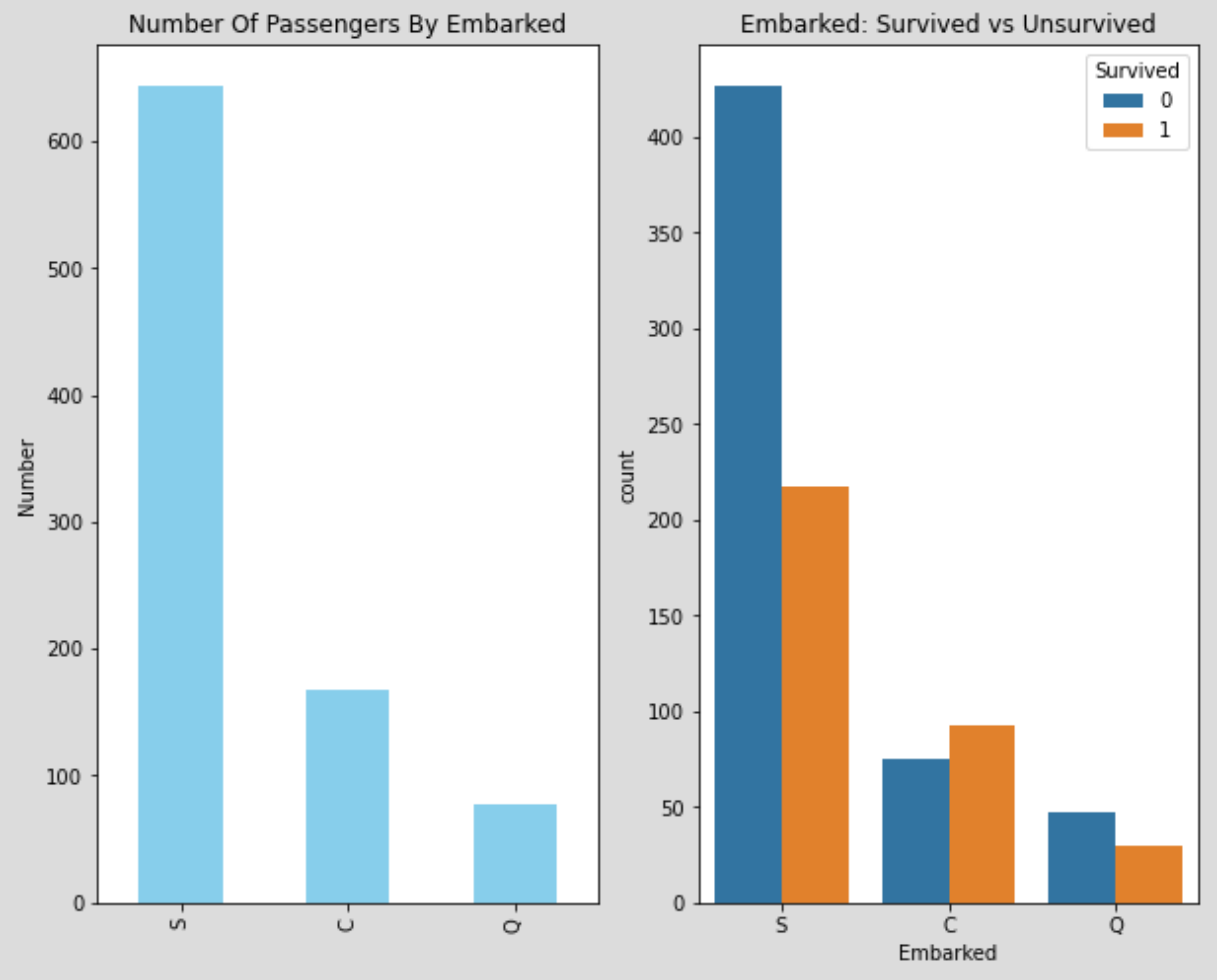
5.1.9

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.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()
```

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

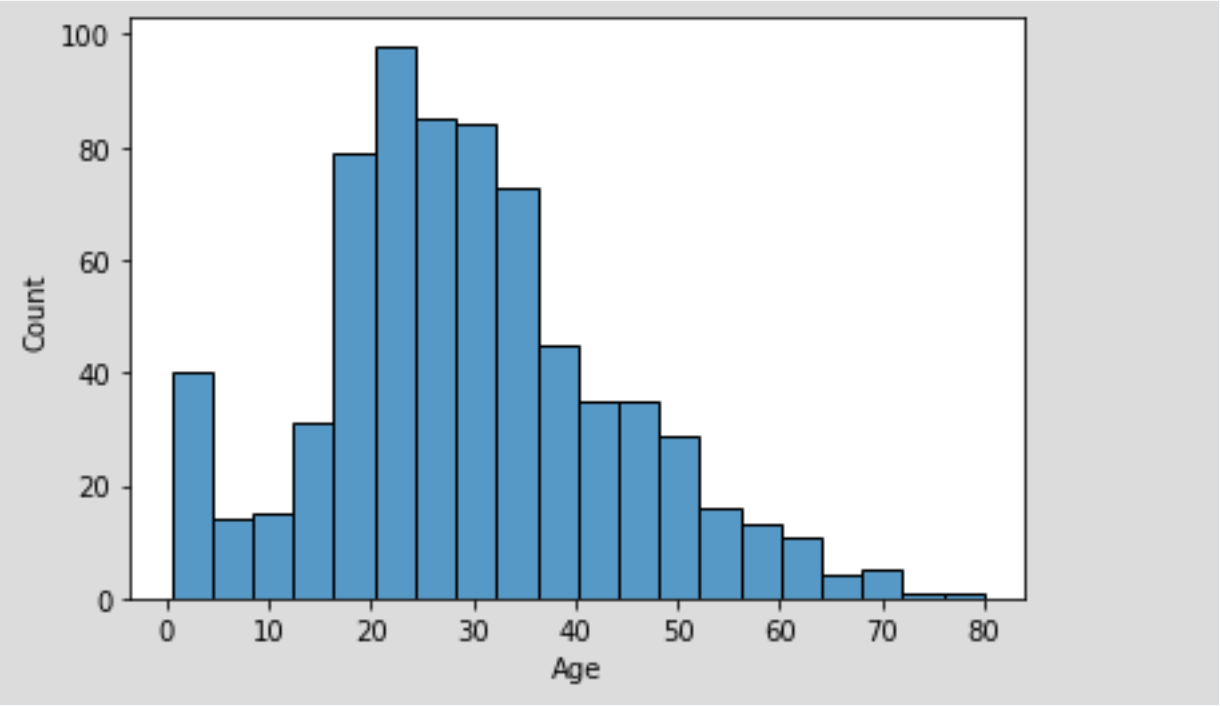
Load the data from the dataset titanic.csv (the file is located at https://priscilla.fitped.eu/data/pandas/titanic.csv). Examine the data in the dataset and find out the age distribution of the passengers. Write the most numerous age category in the following format (we recommend visualizing it as a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

Summarisation

Chapter 1

1.1 The introduction into summarization

📖 1.1.1

INTRODUCTION TO SUMMARIZATION

There are more and more electronic documents on the Internet, and the text contained in them can be too long and difficult to understand. With the development of the field of artificial intelligence, which is called natural language processing, algorithms were developed with the help of which the text can be shortened, i.e. summarized. Natural Language Processing is characterized by algorithms that can understand human language. Typical tasks with which natural language is processed are, for example, determining the morphological properties of texts (e.g. determining the parts of speech), translating documents, supplementing texts (e.g. whisperer in Google search) but also algorithms that can simplify a lot of long text into a coherent and fluent summary . The subfield of language processing that is simplified by text is called summarization.

📝 1.1.2

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

📖 1.1.3

With the help of text summarization, a long text can be simplified into paragraphs, sentences or capture key phrases. In this way, we can reduce the time needed to understand long materials such as research papers without missing important information. The basis of text summarization is the creation of a summary, which can be defined as a text created from one or more documents that provides important information in the original documents and does not exceed the length of half of the original document. The task of automatic text summarization is to create a concise and flowing summary that will preserve the content of key information and overall meaning. Automatic summarization is used, for example, in the generation of excerpts of articles, for example, on news websites. Automatic text summarization is very difficult. When a person tries to summarize a text, they usually read it in its entirety and can write a summary based on their understanding. However, computers cannot think about text in the same way as a human, so summarizing is a difficult and non-trivial task.

Text summarization is an important task in data science that allows us to remove excess information noise, thanks to which we can work only with essential information from the original texts. There are four criteria by which the quality of summarization can be judged:

• Information coverage - Information coverage tells how much of the important information in the text the summarization was able to contain.

• Coherence of summarization - Coherence of summarization is a measure that expresses the relationship and continuity between sentences.

• Minimizing redundancy - Minimizing redundancy means minimizing duplicate information in summaries.

• Brevity - Brevity is a metric that expresses how many words a summary needs to contain important information.

1.1.4

What is the name of the metric that expresses how many words the summary needs to contain important information?

1.2 The approaches to summarization

1.2.1

APPROACHES TO SUMMARIZATION

There are two main approaches to the task of summarization - extraction and abstraction. Extraction is based on extracting from an existing document the most important content that exists in the document, while abstraction approaches can generate new sentences.

EXTRACTIVE SUMMARY

Extractive summarization works on the principle that it decides which sentences from the text are significant and need to be included in the summary. For this purpose, the so-called sentence scoring, which means that each sentence is assigned a score and then the sentences are ranked. The sentences that have the highest score must be included in the summary.

Simply put, if we use an extractive approach, we try to find the most relevant informative sentences in a document, then "extract" them from the text and combine them again to create a new, shorter version of the original text. With this approach, no new sentences are created that did not previously exist in the text. The extracted sentences from the original document are just recombined. Extractive summarization can use several methods:

• **TF-IDF (Term Frequency - Inverse Document Frequency)** - This statistical method is used to assess the importance of words. Term frequency is used to determine how many times a term occurs in a document. The frequency of expressions like "the" can be very high. The inverse document frequency is calculated as the logarithm of the total number of documents divided by the total number of documents in which the term occurs. The inverse frequency of a term document can be low even though its term frequency is very high.

• **Graph-based methods** - With this technique, a graph is created. Graph nodes represent sentences. The edges of the graph symbolize connections between sentences that share the same words. Nodes that have more edges contain important sentences and have higher priority in summarization.

• **Principles using machine learning** - Using machine learning, we can view text summarization as a two-class classification problem. Sentences are grouped into summary and non-summary sentences. The summarizer is trainable, the training data set and their extraction summaries are used as a reference.

• **LSA (Latent Semantic Analysis)** - LSA is a robust algebraic-statistical method that extracts the hidden semantic structures of words and sentences, that is, it extracts properties that cannot be mentioned directly. These features are essential to the data, but are not native features of the dataset. It is an unsupervised approach along with the use of Natural Language Processing (NLP).

• **Methods using neural networks** – Neural networks try to imitate the activity of the human brain when learning. Like the human brain, they contain neurons that process data. So the neural network tries to think like a human to judge which sentences are important and should be included in the summary. Using training data, it tries to learn the types of sentences that should be included in the summary. After the network learns the features that must exist in the summary

sentences, we need to determine the trends and relationships between the features that are inherent in most of the sentences.

• **Methods based on Fuzzy logic** - This method considers each characteristic of the text, such as sentence length, title similarity, keyword similarity, etc. for the fuzzy system input. All rules needed for summarization are also input to the knowledge base. Each sentence receives a score ranging from 0 to 1. The obtained value determines the importance of the sentences for generating the summary.

1.2.2

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

1.2.3

ABSTRACT SUMMARY

Abstract summarization is a smarter form of summarization compared to extractive summarization because it can generate new sentences. In this approach, we must first create a transient representation of the input text. This is usually created by representing the topic, i.e. by transforming the text in order to identify the main topics of the text and by representing the indicator, where a set of "indicators" - e.g. the length of sentences or sentences containing certain "indicator" words expresses the importance of a part of the text. The individual sentences are again ranked according to their score, and the sentences with the highest scores are used to compile the summary.

Abstract summarization attempts to understand text using advanced natural language processing techniques and create new sentences by paraphrasing as a human would. This approach is significantly more complex, as it requires a semantic understanding of the text and the connection between concepts, context

and topics. Similar to extractive summarization, abstract summarization can be achieved by several methods:

• **Tree-based methods -** The main idea of this group of methods is to use a dependency tree that represents the text or content of a document. An example of such an algorithm is sentence fusion, which can process multiple documents.

• **Template-based methods -** In these methods, the entire document is represented using a certain wizard. Linguistic patterns or extraction rules are mapped to point text snippets that can be mapped to guide slots (to create a database).

• **Rule-based methods -** Rule-based methods display input documents in terms of classes and facet lists. Verbs and nouns with similar meanings are identified to create extraction rules. A number of candidate rules are selected and transferred to the summary.

• **Graph-based methods -** Similar to extractive summarization, abstract summarization can be achieved using graphs.

• **Ontology-based methods -** Ontologies are extremely popular in NLP, including both extractive and abstract summaries where appropriate, as they are usually limited to the same topic or domain. In addition, each domain has its own knowledge structure, which can be better represented using an ontology. Although they differ in their specific approaches, all ontology-based summarization methods involve sentence reduction by compression and reformulation using both linguistic and NLP techniques. Fuzzy ontology is a typical representative.

• **Multimodal Semantic Model -** A semantic model is initially created using an object-based knowledge representation. Nodes represent concepts and links between these concepts represent the relationship between them. Important ideas are scored using an information density metric that checks for completeness, relationship with others, and number of term occurrences. The selected terms are finally transformed into sentences to form a summary.

• **Semantic Text Representation Model -** The goal of this technique is to analyze the input text using the semantics of the words, rather than using the syntax or structure of the text.

1.2.4

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

1.2.5

COMBINED SUMMARY

There are combined approaches that use an abstract generator. The abstract generator takes as input the text that comes from the extractive summarizer. Combined summarization is more effective because it works with text that is already stripped of all redundant and irrelevant information. Many current algorithms, including the BART algorithm, are based on this approach.

1.3 Koncepts used in text summarization

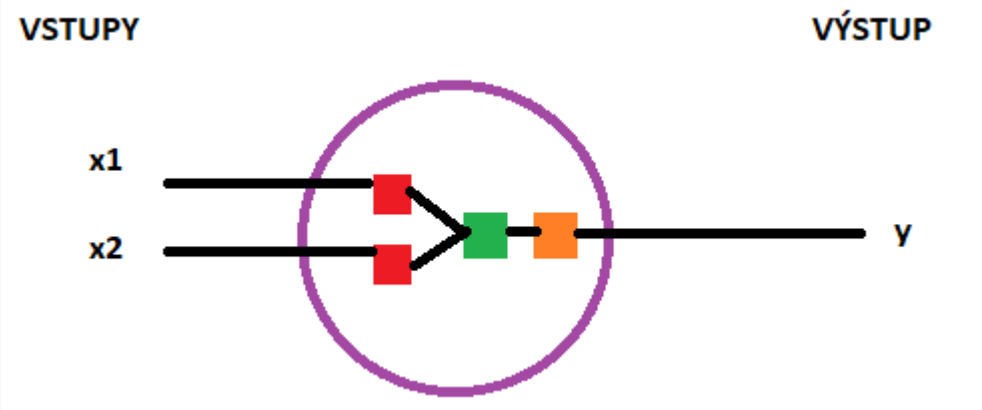
1.3.1

CONCEPTS USED IN TEXT SUMMARY

In order to understand the principle of operation of the individual algorithms that serve to summarize the text, we must first familiarize ourselves with neural networks. The human brain contains tens of thousands of brain cells called neurons. These neurons are interconnected, can communicate with each other and create complex structures. In artificial intelligence, there are methods that teach computers to process data similarly to the human brain. These methods are called neural networks.

NEURON

Before we explain the architecture of neural networks, it is necessary to familiarize ourselves with how a neuron works. The simplest possible implementation is a neuron with two inputs, similar to the figure below, where x1 x2 represent the inputs and y represents the output.



There are three things going on here:

- First, each input is multiplied by some weight w.
- The weighted inputs are counted and the so-called bias (some constant)
- The result from the second step is fed into the activation function, which is used to turn the result of the input into an output that has a nice, readable form. A commonly used activation function is a sigmoid function that converts a number to a value between 0 and 1.

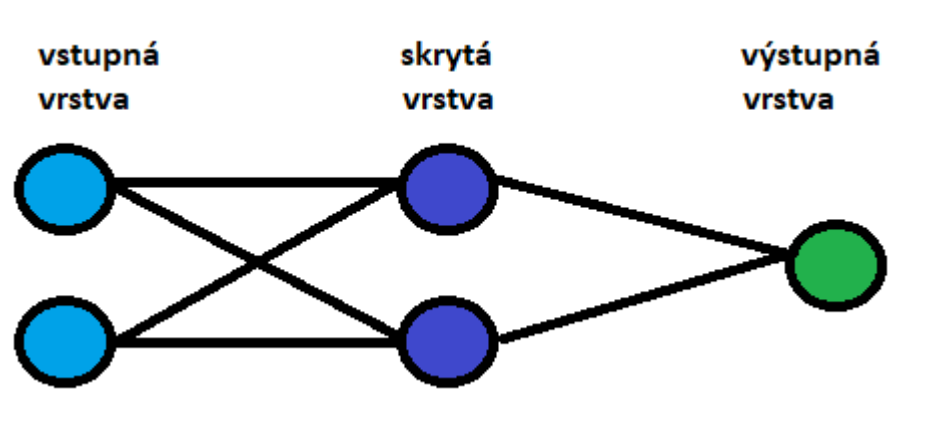
1.3.2

What is the basic unit of a neural network called?

1.3.3

NEURON NETWORKS

If we connect several neurons together, we talk about the so-called neural networks. A basic neural network has interconnected artificial neurons in three layers:

- **Input Layer** - These are our original inputs, similar to x1 and x2 in the previous image.
- **Hidden layer** - The hidden layer analyzes the output from the input layer. If the neural layer has several hidden layers, we are talking about deep neural networks.
- **Output layer** - The output layer gives the final result.
- 

1.3.4

The layer of the neural network that processes the output from the input layer is called

- hidden layer
- output layer
- input layer

1.3.5

TYPES OF NEURONAL NETWORKS

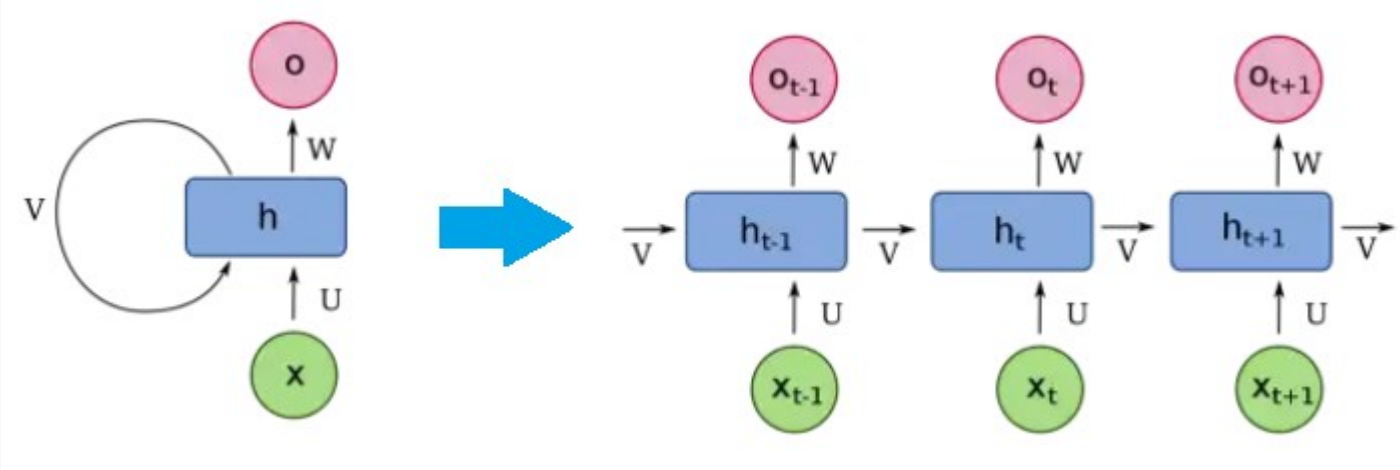
There are several types of neural networks. We will describe the most famous of them.

Feedforward neural networks

Feedforward neural networks process data in one direction, from the input layer to the output layer. Every node in one layer is connected to every other node in the next layer.

Recurrent Neural Networks (RNNs)

Recurrent neural networks are special architectures that take temporal information into account. The hidden state of the neural network at time t takes information from the input at time t and the activations from the hidden units at time t-1 to compute the outputs for time t. This can be seen in the image below. In this way, a recurrent neural network can remember previous inputs and their outputs.



Remembering previous input is particularly important in natural language processing tasks because the input words are not of equal size and the next word is highly dependent on the previous words. Thus, recurrent neural networks can remember context.

Long Short Term Memory (LSTM) Networks

The memory of RNNs is short, which can be a problem. The optimization of this problem was the creation of networks with a longer memory, which are called LSTMs and use the so-called cell state. This cell state is the state at any time and is updated with relevant information at each time step. The output at each time step is derived from the input, the previous output, and the updated state of the cell.

Transformer networks

With the introduction of LSTMs and their ability to remember the state of cells, the memory of neural networks improved but was still limited. To solve memory problems, transformer networks have been developed that introduce the concept of attention blocks. Attention blocks calculated how each word in the input was related to other words in the input. The higher the value, the more attention is paid to these words and the more dependent the set of words is. Attention increases the number of contextual connections a network can make, and the network can learn relationships and context from large data sets.

📖 1.3.6

Algorithms used in text summarization

Nowadays, since this is a field that produces really useful results, many different algorithms are used and constantly improved.

GPT-3

GPT-3 is an automatic regressive artificial intelligence algorithm developed by OpenAI, an AI-powered research lab based in San Francisco, California. It is a massive artificial neural network that uses deep learning to generate human text and is trained on huge text files with thousands of billions of words. It is the third-generation AI language prediction model in the GPT-n series and the successor to GPT-2.

This artificial intelligence algorithm is a program that can calculate a word or even a character that must appear in a text in relation to the words around it. This is called the conditional probability of words. It is a generative neural network that allows for a numerical score or a yes or no response. It also generates long sequences of original text as output. The total number of weights that OpenAI GPT-3 dynamically stores in its memory and uses to process each query is 175 billion.

BERT

BERT (Bidirectional Encoder Representation for Transformers) uses a fully bidirectional unsupervised approach and is pre-prepared for pure text corpus only (Wikipedia). The two-way approach means that the words in the sentence are evaluated not only from left to right and top to bottom, as a person would do, but also in the opposite direction. The interesting thing about this algorithm is that it learns using a masking function ("masked language modeling": some words are masked into a sentence) and then BERT has to predict which one is the missing word or if the sentence follows another sentence. BERT uses an attention mechanism that is able to learn the contextual relationships between words in a text. Below we briefly describe the BART algorithm, which is currently the most modern in the field of summarization and is derived from BERT.

BART

BART (Bidirectional Autoencoder Representation for Transformers). The BART algorithm generalizes both the GPT and BERT approaches, taking the best of the two models. BART is trained to corrupt the text with a noise function (which adds "noise" to the text, not just masks) and then trains the model to recover the original text. It is based on a transformer-based neural machine translation architecture with a bidirectional encoder (like BERT) and a left-to-right decoder (like GPT). The BART algorithm maps document corruptions to an input document and can be applied to any type of document corruption (token masking, token erasure, text padding, sentence permutation, document rotation, etc.). The BART algorithm achieves new, state-of-the-art results in abstract dialogue, text generation, question answering, and summarization tasks.

Keyword Extraction

Chapter 2

2.1 The introduction into keyword extraction

📖 2.1.1

Introduction to keyword extraction

Keyword extraction is one of the summarization techniques used to capture the most important words or phrases from a document. Using this technique, a small set of units consisting of one or more phrases can be extracted. Key phrases play an important role in quickly getting the idea of a textual data without having to read the whole text. This text summarization technique finds application in the field of content management, such as search engine optimization, advertising and user recommendation systems. For example, when visiting an ad or website, end users are attracted if the keywords are relevant to their needs.

Approaches to keyword extraction can be most simply divided into two basic groups, namely simple statistical approaches and approaches based on machine learning. If we wanted to take a closer look at keyword extraction, we could divide the extraction principles into five categories, namely simple statistical approaches, graph-based approaches, linguistic approaches, machine learning-based approaches, and hybrid approaches.

📝 2.1.2

Which of the following terms do not belong to approaches to keyword extraction

- absctractive
- extractive
- statistical
- graph based
- hybrid
- machine learning based

📖 2.1.3

Preprocessing of texts

Different approaches to keyword extraction may require different levels of pre-processing of the text from which we are going to extract keywords. Text preprocessing refers to techniques such as:

- **Removing stop words** - The most important step in the revision process is to remove words that have no meaning in the text. Such words include, for example, conjunctions, prepositions, or other words that occur frequently in the text but do not make sense by themselves. A list of such words for different languages can be obtained using the NLTK python library.
- **Text to lower case** - Keyword extraction algorithms can be case sensitive. For example, if we have the sentence "Keyword extraction is usefull. Rake is the best technique for keyword extraction", we would not want to get both "Keyword extraction" and "keyword extraction" in the list of keywords.
- **Removal of punctuation and special characters** - Texts may contain various special characters and unwanted punctuation. Let's say we're extracting keywords from social media statuses. Such statuses can often contain emojis that do not carry any semantic meaning and we do not want them to be extracted as keywords.

2.1.4

Conjunctions, prepositions, or other words that appear in the text are often marked as

2.2 Statistical Approaches

2.2.1

Statistical approaches

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 ti in the document is not dj. The denominator expresses the sum of the number of occurrences of all words in the document.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_{k}^{n} n_{k,j}}$$

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 : ti ϵ dj}| is the number of documents that contain the word i.

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

2.2.2

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

-
-

2.2.3

RAKE (Rapid Automatic Keyword Extraction)

RAKE enjoys the most popularity among statistics-based keyword extraction algorithms. The idea behind this algorithm is that keywords often contain multiple words, but rarely contain punctuation, stop words, or other words with minimal lexical meaning. The algorithm is primarily based on the co-occurrence of words,

for example, when extracting keywords from customer feedback on a specific phone, a key phrase could be represented by a bigram such as "good camera", "quality sound." These words in the feedback domain of a specific product often appeared together . It's a collocation. 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 kandidátne kľúčové frázy pre slovo 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.

2.2.4

What is the basis of the RAKE algorithm?

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

2.2.5

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.

```
pip install nltk
pip install rake-nltk
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')
```

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.

```
rake_extractor = Rake(stopwords = stop_words, min_length=2,
max_length=2, include_repeated_phrases=False)
rake_extractor.extract_keywords_from_text(text)

# To get keyword phrases ranked highest to lowest with scores.
rake_extractor.get_ranked_phrases_with_scores()
```

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.

```
rake_extractor.get_ranked_phrases_with_scores()
```

📖 2.2.6

KP-miner

There are more complex keyword extraction methods that use TF-IDF only as a statistical method to calculate the importance of key phrases. This includes, for example, the KP-miner method, which is divided into three steps. The first step is to select the candidate words from the documents, the second step is to calculate the score of the candidate words, and the third step is to select the candidate word with the highest score as the final keyword phrase. KP-miner introduced two new statistical functions in the candidate word selection phase. The Least Allowable Seen Frequency factor means that only words that appear more than n times in the document can be considered as candidate words. The second statistical function introduced by KP-miner is called CutOff and is based on the fact that if a word appears after a given threshold position in a long document, it will not be a key phrase, meaning that the word that appears after the CutOff will be filtered out.

Finally, the final key phrases are selected by combining the candidate word positions and TF-IDF scores.

2.2.7

List the steps of the KP-miner algorithm

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

2.2.8

YAKE

YAKE is a typical keyword phrase extraction method using TF-IDF. The difference between YAKE and KP-miner is that YAKE uses the candidate word location or TF-IDF information and introduces a new set of five features. The case of WC reflects the case of the candidate words. The WP variable in the formula reflects the position of the word, meaning that the more often the word is at the front of the document, the greater its value. Word frequency is WF expresses that the higher the frequency of a word in a document, the greater its value. WRC context word relatedness refers to the number of different words occurring on either side of a candidate word. Word DifSentence WD indicates the frequency of the candidate word in different sentences. These five values are combined to calculate S(w) as shown in the formula below.

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

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.

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

2.2.9

Implementation of the YAKE algorithm

To implement the algorithm, we need to download and import the appropriate library. The basic implementation of the algorithm is simple, it is enough to define the language in which our text is located and the maximum number of n-grams.

```
pip install yake
import yake
yake_extractor = yake.KeywordExtractor(lan="en", n=2)
keywords = yake_extractor.extract_keywords(text)
for kw in keywords:
    print(kw)
```

2.2.10

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

- yes
- no

2.3 Graph based approaches

2.3.1

Graph-based approaches

All graph-based approaches compute to a vertex in the graph, relying not only on information that is specific to a local vertex, but also taking into account global information that is recursively computed from the entire graph. The basis of many graph-based algorithms is the PageRank algorithm

PageRank

PageRank is an algorithm that was developed to rank web pages according to a system of quantity and quality of links that point to it. PageRank is calculated according to the formula below, where A represents the subpage for which PageRank is calculated and T1 to Tn are the subpages that link to subpage A. PR stands for PageRank of subpages, C stands for new links from subpages, and is a damping factor that takes care of reducing the excessive influence of some subpages, which can cause the use of fake bots. 1-d is the factor that takes care of eachnovú stránku, na ktorú neukazuje žiadny odkaz.

$$PR(A) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \cdots + \frac{PR(T_n)}{C(T_n)}\right)$$

TextRank

The most famous graph-based keyword extraction algorithm is TextRank, which is based on the PageRank algorithm. TextRank uses the PageRank algorithm on a graph where vertices correspond to words. An important aspect is a text that contains deep linguistic knowledge, or domain- or language-specific annotated corpora, making it highly transferable to other domains, genres, or languages.

The original PageRank algorithm assumes an unweighted graph. But the graphs for TextRank are built from natural language text and therefore would include many links between tokens extracted from the text. Therefore, useful pages of relationships between graph vertices would be important. For this reason, TextRank is applied to a weighted chart.

An interesting feature of the TextRank algorithm is that it also includes features of linguistic approaches. The first step of the algorithm is to tokenize the text and then annotate it with POS tags. The authors of TextRank conducted experiments and observed the best results when considering only nouns and adjectives. Vertices that pass a syntactic filter in which POS tags are only noun or adjective are graphed as vertices.

2.3.2

TextRank uses the PageRank algorithm on a graph where vertices correspond to words

- yes
- no

📖 2.3.3

SingleRank

Another graph-based keyword extraction algorithm is SingleRank, which extends TextRank with two main differences. As in the TextRank algorithm, with SingleRank, vertices are passed through a syntactic filter and edges are also assigned based on the co-occurrence of words in the window. The first major difference is that these edges are assigned a weight based on the distance between two words that are in a predefined window. The second difference counts the number of vertices it keeps as potential keywords after running the PageRank algorithm. SingleRank keeps all words, while with TextRank it's usually the top 30%.

📝 2.3.4

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.

📖 2.3.5

TopicRank

TopicRank uses a slightly different method from the TextRank and SingleRank algorithms. Its task is to extract key phrases from those equally present in the document. This algorithm considers the topic as similar candidates for key phrases. These topics are then ranked according to their importance in the document, and the most important key phrase for each topic.

The TopicRank algorithm consists of the following steps:

- identifying topics,
- chart-based assessment,
- keyword selection.

There are three strategies used to find the best keyword phrase for a given topic. One strategy converts all key phrases back to their generic form and selects the key phrase that appeared first in the document. The second strategy selects the most frequent key phrase, while the third selects based on the centroid of the cluster. The centroid is an imaginary or real location representing the center of the cluster.

Each data point is assigned to each of the clusters by the reduced sum of squares within the cluster.

2.3.6

Sort the steps of the TopicRank algorithm

- topic identification
- chart-based assessment
- <|br>
- keyword selection
- <|br>

2.4 Machine learning based approaches

2.4.1

Approaches based on machine learning

Keyword extraction approaches that are based on machine learning use supervised (supervised) learning and transform the keyword extraction task into a classification or prediction problem. A model trained on the labeled set is used to determine whether a candidate word in the text is a key phrase or not. The advantage of machine learning-based approaches is that they require less or no text pre-processing and extract key phrases with high semantic relevance. The disadvantage is that the models are language- and sometimes context-dependent, and changing the corpus may require training the model anew, or choosing a different model. Another disadvantage is that machine learning brings with it a higher computational effort, which makes extraction using machine learning-based approaches slower than approaches that are solely based on statistics and do not require training data.

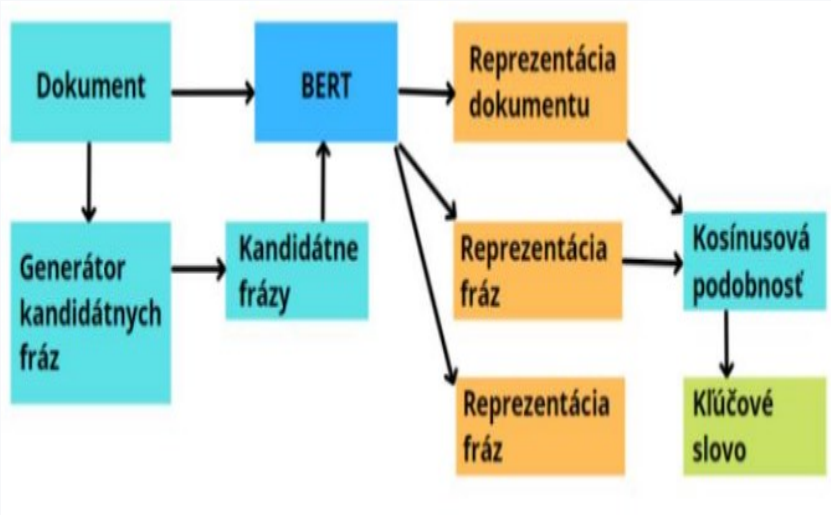
KEA

One of the first methods of keyword extraction that uses machine learning is KEA, which consists of determining whether a candidate word is a keyword phrase by calculating the TF-IDF of each candidate word and the place where it first appears in the text, and putting these value values into Naive Bayes.

📖 2.4.2

KeyBERT

The most widely used technique that can extract keywords with high semantic relevance is KeyBERT. This technique uses a pre-trained BERT (Bidirectional Encoder Representations from Transformes) model. The KeyBERT algorithm itself begins by sending a document to the BERT model, which creates a representation of the document by dividing the text into fixed-size vectors representing the semantics of the document.



In the second step, the candidate phrase generator extracts candidate phrases from the document using simple techniques such as occurrence count, TF-IDF, and so on. In the next step, this data is again sent to the BERT model and a phrase-level representation is obtained. Subsequently, the cosine similarity is calculated between the document-level representation and the phrase-level representation, to obtain the most similar words to the document representation, which are the resulting keywords. The calculation takes place according to the formula below.

$$podobnosť\ (A,B) = \frac{A*B}{|A|*|B|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

The results are then sorted in descending order and the top n items are selected.

📝 2.4.3

KeyBERT algorithm implementation

The first step that needs to be done is to install the library that implements KeyBERT and then import it.

Subsequently, we can implement the algorithm using two lines of code. The keyphrase_ngram_range parameter defines the range of desired n-grams. We insert the list of desired stop words into the stop_words parameter. The default model used for extractions is the "all-MiniLM-L6-v2" model. This model works with the English language and belongs

```
pip install keybert
from keybert import KeyBERT
keybert_extractor = KeyBERT()
keywords = keybert_extractor.extract_keywords(text,
keyphrase_ngram_range=(2, 2), stop_words=stop_words)
```

📝 2.4.4

List the steps of the KeyBERT algorithm

- Getting key phrases
- <|br>
- <|br>
- Creating a document-level representation
- Calculation of cosine similarity
- Creating a phrase-level representation
- The document is sent to the BERT model
- Selection of candidate phrases
- <|br>
- <|br>
- <|br>

2.5 Hybrid Approaches

🕮 2.5.1

Hybrid approaches

Hybrid approaches combine the previous methods. They use heuristic knowledge such as position, word length, HTML tags around words and other methods.

2.6 Evaluation

🕮 2.6.1

Evaluation of algorithms for keyword extraction

It is not easy to design an evaluation metric that could reflect the advantages and disadvantages of an algorithm. Since an evaluation metric can only evaluate one aspect of an algorithm, multiple metrics can more accurately and comprehensively evaluate an algorithm. For example, researchers usually use precision, coverage (recall), and F1-score (harmonic mean) to evaluate a method from multiple perspectives.

🕮 2.6.2

Metrics based on statistics

Statistics-based evaluation metrics analyze the performance of the method by calculating the proportion of the number of different key phrases, such as the number of extracted key phrases, correct key phrases, incorrect key phrases, and manually assigned key phrases. Standard statistics-based metrics include precision, coverage, and harmonic mean.

Precision

Mathematically, the precision metric is defined as the number of true positives tp divided by the sum of the true positives tp and the number of false positives fp. It can be calculated according to the formula below.

$$presnosť = \frac{tp}{tp+fp} = \frac{korektné\ frázy}{extrahované\ frázy}$$

Recall

This metric is defined by the formula below as the number of true positives tp divided by the sum of true positives tp and false negatives fn.

$$pokrytie = \frac{tp}{tp+fn}$$

Harmonic mean (F1 score)

The previous two metrics influence each other. In an ideal situation, both are high, but in general, when the precision metric is high, the coverage is low, and vice versa. The harmonic mean is a combination of both.

$$harmonický\ priemer = 2 * \frac{precission * recall}{precission + recall}$$

2.6.3

Precision is defined as the number of true positives tp divided by the sum of true positives tp and the number of false positives fp

- yes
- no

2.6.4

Recall is defined as the number of true positives tp divided by the sum of true positives tp and the number of false positives fp

- yes
- no

2.6.5

Metrics based on linguistics

The ranking metrics listed so far are based on the assumption that key phrases are independent of each other, but based on human language habits, we hope that more important key phrases should be placed higher. The following three ranking metrics can reflect the ranking functions among the key phrase outputs by the algorithm.

Mean Reciprocal Rank (MRR)

Mean reciprocal rank is a measure for evaluating models that return a document-ordered list of key phrases. MRR only cares about one highest rated relevant item. If the model returns a relevant keyword phrase in the third highest position, then MRR takes care of that. It doesn't matter if the other relevant key phrases (assuming there are any) rank #1 or #10.

$$MRR = \frac{1}{|d|} \sum_{i=1}^{d} \frac{1}{rank_i}$$

MRR gives the average ranking of the first correct prediction, where d is the number of documents and ranki is the rank in which the first correct key phrase of document i was found.

Mean Average Precision (MAP)

MAP takes into account the order of the particular returned list of key phrases. The average accuracy of AP is defined by the equation,

$$AP = \frac{\sum_{n=1}^{|N|} P(n)gd(n)}{|LN|}$$

where |N| the length of the list, |LN| represents the number of relevant items, P(n) is the precision, and gd(n) is equal to one if the nth item is a golden keyphrase and 0 otherwise. By averaging the AP over a set of n documents, the mean average accuracy (MAP) is defined as:

$$Bpref = \frac{1}{C}\sum_{c \in C} 1 - \frac{|I|}{M}$$

where C represents the number of correct key phrases, M represents the number of all extracted key phrases, and I represents the number of correct key phrases before incorrect phrases.

Classification

Introduction to classification

Chapter 1

1.1 Introduction

📖 1.1.1

Introduction into classification

So, in order to classify, we need two things. A classifier, that is, an algorithm that can provide a classification on some dataset. Suppose we go to solve one of the typical classification tasks, namely whether the report is fraud or not. The data file in our case could be a .csv file, in which we would have a message in one column and a binary label (0/1) in the other column. If the message is a hoax, it would have a label of 1, and if the message is not a hoax, it would have a label of 0.

At the moment we need to find (train) our model. By default, this is done in such a way that some part of the dataset (for example 75%) is taken, which will be used to learn the model. This part of the dataset is also called the training set. The model will learn similarly to how a human would. It will look at a message, read whether it is a hoax or not and will look for patterns among the data to learn which messages look like a hoax. Once the training process is complete, the second phase, testing. In this phase, the model will get the remaining 25% of the data, which they also refer to as the test set, and will try to classify the messages into that to look at the output. given the probability with which the model will assume that a given message is a hoax.

📝 1.1.2

The algorithm that implements the classification is called

📖 1.1.3

Types of classification tasks

Classification tasks are divided into:

- binary classification,
- multi-class classification,
- multi-label classification.

Binary classification

We already talked about binary classification in the previous examples, when we remembered the classification tasks like whether the picture is a dog or a cat or whether the message is a hoax. So it is a type of classification where we can classify our case only among the two tried ones. For example, whether the email is spam. The input variables would represent the characteristics/properties of the

email. This variable could be represented by one of two values (classes). If the email was spam, it would be assigned a value of 1, otherwise it would be assigned a value of 0.

Multiclass classification

Multiclass classification is used in cases where our input variable can take more than two values. A typical task of multi-class classification is the categorization of face or plant species.

Let's imagine that we wanted to create a model that could determine which of the flowers it was. In such a case, we would therefore have three possible values of the output variable, which is why we say it is a multi-class classification.

Multi-label classification

Let's imagine that we want to classify objects in a photo. One photo can contain a person, a table, a dog, etc. Compared to binary or multi-class classification, where we assumed a single class designation, a specific photo can therefore have several objects in the scene.

Symbolical classification models

Chapter 2

2.1 Decision Tree

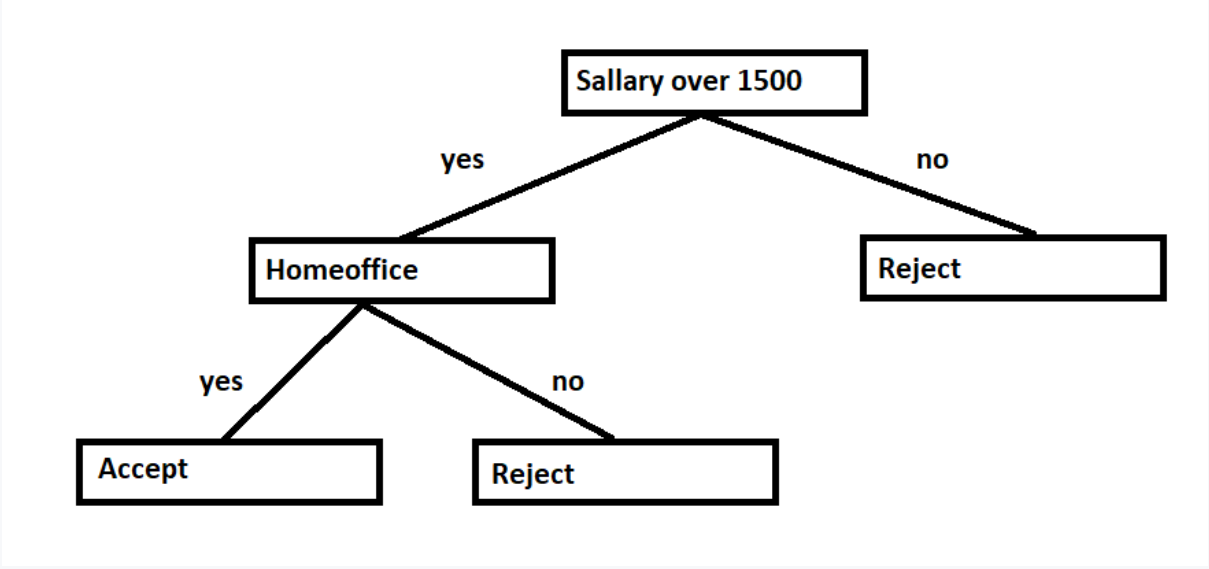
📖 2.1.1

Decision tree

A decision tree is a classifier with a tree structure. For the decision tree, we introduce the following concepts:

- Root - the place where the tree begins. From this point (vertex / node), the tree further branches into two or more parts.
- Leaf - the final output that does not branch further.
- Branch - a subtree that was created by branching.

The decision tree is very easy to understand because it makes decisions similar to a human. At the top of the tree is the question or main criterion from which the tree branches. Let's say that we are looking for a job with a salary above €1,500 and a home office option.



📝 2.1.2

What is the name of the place where the tree starts

📖 2.1.3

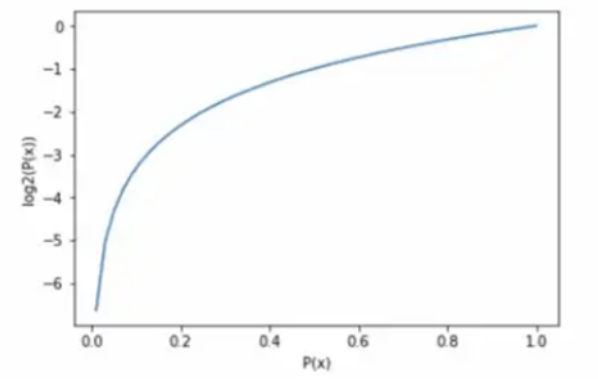
Measuring the quality of tree splitting

Two metrics are typically used to measure the quality of a distribution, namely the Gini index and entropy.

The Gini index is a metric for classification tasks whose value ranges from 0 to 1, where 0 indicates that all elements are associated with a certain class or that there is only one class. If the Gini index is 1, the elements are randomly assigned to different classes. If the Gini index reaches a value of 0.5, it means that the elements are evenly distributed in some classes. An attribute with a low Gini index should be preferred over a high Gini index. The Gini index is calculated according to the formula below, where pi is the probability that the object will be classified in a certain class.

$$Gini = 1 - \sum_{i=1}^{n}(pi)^2$$

Let's imagine that we choose a white chess piece from a box containing 100 white pieces. Then we can say that the box has zero entropy. Now let's imagine that 50 of them are replaced by black. Probability of drawing white dropped from 1.0 to 0.5 and entropy increased. Shannon's entropy model uses a base-2 logarithmic function (log2(P(x)) to measure entropy, because as the probability P(x) of randomly drawing a white piece increases, the result gets closer to the base-2 logarithm value of 1, as shown in the picture.



📝 2.1.4

An attribute with a low Gini index should be preferred over a high Gini index

- yes
- no

📖 2.1.5

Entropy is a metric for measuring the uncertainty with which a case is classified into a class, and the task of the algorithm is to minimize this uncertainty. Similar to the Gini index, the optimal distribution is chosen by the property with the least entropy. Entropy can be calculated according to the formula, where P(x=k) expresses the probability that the target character will have the value k.

$$E = -\sum_j (P(x = k) * log_2(P(x = k))$$

The logarithm of fractions gives a negative value, so the entropy formula uses a minus sign to negate these negative values. The maximum value of entropy depends on the number of classes. To find the best feature that serves as the root node in terms of information retrieval, we first use each descriptive feature and partition the dataset according to the values of these descriptive features, and then calculate the entropy of the dataset. This gives us the remaining entropy when we divide the data set by feature values. We then subtract this value from the originally calculated entropy of the data set to see how much this element distribution reduces the original entropy, which provides the information gain of the element and is calculated according to the formula, according to which the feature with the highest information gain will be used as the root node.

$$Informačný\ zisk(x) = E(dataset) - E(x)$$

📖 2.1.6

Algorithms for creating a tree

Algorithm ID3

The ID3 (Iterative DiChaudomiser 3) algorithm creates a tree structure from the training data set, which is used to classify the yet unclassified data. It tries to find the categorical feature that will yield the greatest informational gain for the categorical targets. Information gain is calculated using entropy. The algorithm searches each branch and stops when each subspace contains only elements of one class. Trees created by this algorithm are prone to overtraining.

Algorithm C4.5

A better version is the C4.5 algorithm, which eliminated the problem of classifying datasets that contain attributes with a large number of values. The entropy of attributes that take on a large number of values is very low. Therefore, the so-called normalized information gain. Another difference, compared to the previous algorithm, is that it can also work with attributes that have empty values. After building the model, it examines the tree once more and removes nodes that do not have a significant impact on the classification.

Algorithm C5.0

Another modification of the algorithm was named C5.0. This algorithm was faster, more efficient and it was possible to parallelize the algorithm using threads.

CART algorithm

CART is a decision tree where each branch is divided into a predictor variable and each node has a prediction for the target variable at the end. In the decision tree, nodes are divided into subnodes based on an attribute threshold. The root node is taken as the training set and is divided into two parts by considering the best attribute and the threshold value. Further, the subsets are also partitioned using the same logic. This continues until the last pure subset of the tree or the maximum possible number of leaves in this growing tree is found. The algorithm works according to the following procedure:

the best split point of each input is obtained,

based on these split points, a new best split point is identified,

divides the selected input according to the best dividing point,

splitting continues until the stopping rule is met or no further requested splitting is available.

A tree that is too large increases the risk of overlearning, and a small tree may not capture all the important features of a file. Therefore, a technique called pruning is used, which reduces the size of the tree without reducing accuracy.

There are two ways to reduce the size of the tree. The first method is precutting, which consists in tuning hyperparameters before training. It includes a heuristic known as "stopping early" that stops the growth of the decision tree - preventing it from reaching its full depth. Stops the tree building process to avoid creating leaves

with small patterns. Cross-validation error will be monitored during each phase of tree splitting. If the error value no longer decreases - we stop the growth of the decision tree. The hyperparameters that can be tuned to stop early and avoid congestion are the maximum depth, the minimum number of samples required to split an internal node, and the minimum number of samples required to be in a leaf.

The second method is post-pruning, which does not prevent the tree from growing but prunes the tree after the tree has grown to its full depth. For each non-leaf node in the tree, the algorithm calculates the expected error rate that may occur if the subtree at that node is truncated. Next, the expected error rate that would appear if the node had not been pruned is calculated using the error rate for each branch, combined by weighting according to the dimension of the observations along each branch. If pruning a node leads to a higher expected error rate, then the subtree is preserved.

2.1.7

The root node is the one with the lowest information gain

- yes
- no

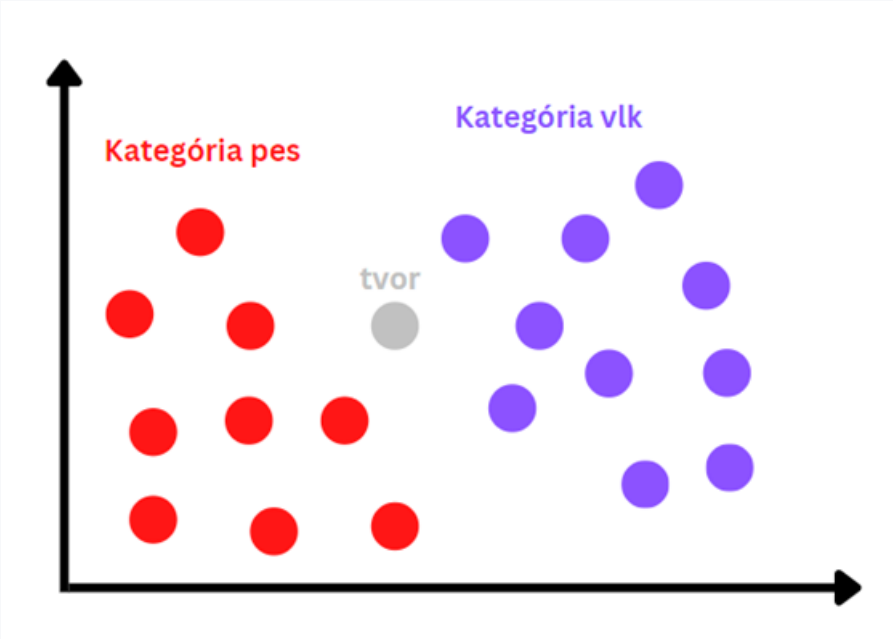
2.2 K-nearest neighbors classifier

2.2.1

K-nearest neighbors classifier

The K-Nearest Neighbors algorithm is among the lazy learning algorithms because it does not learn from the training set immediately, instead it stores the data set and performs an action on the data set at the time of classification.

Suppose we have a picture of a creature that looks similar, like a dog and a wolf, but we want to know if it is a dog or a wolf.



The first step of the algorithm is to find a suitable number K, which will indicate the number of nearest neighbors. Choosing the right K is an important task. Very low values of K could lead to unstable decision boundaries, and high values of K could be computationally demanding. After selecting K, the algorithm finds the K-nearest neighbors to our creature according to the Euclidean distance.

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

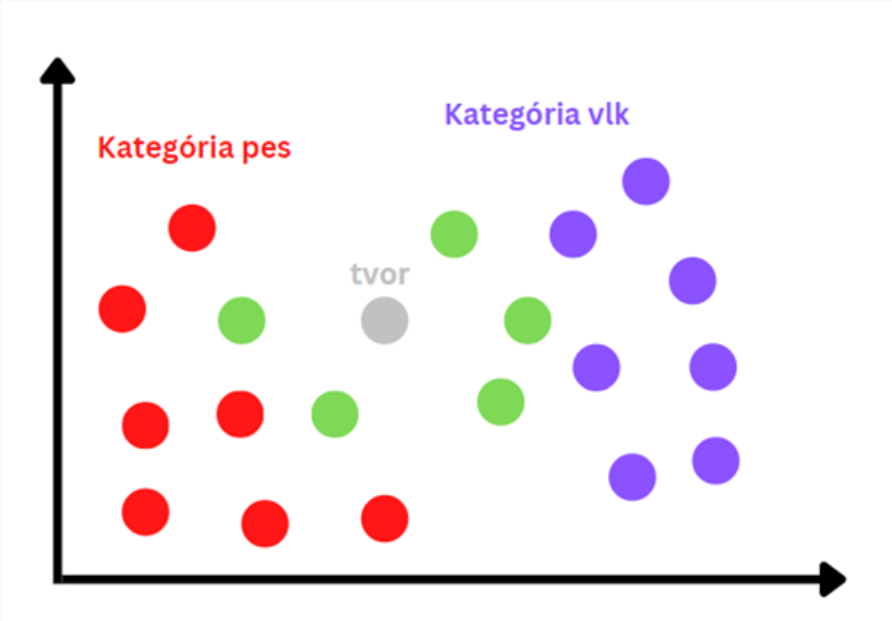
2.2.2

The number K is used to determine the number of nearest neighbors by which to determine the resulting class for the case

- yes
- no

2.2.3

Subsequently, the algorithm counts how many of the nearest neighbours belong to the dog category and how many of them belong to the wolf category. It assigns the creature the category that occurred most often in K-neighbours. Consider that our K=5. The creature will be assigned the wolf category.


Kategória pes
Kategória vlk
tvor

Logistic regression

Chapter 3

3.1 Logistic regression

📖 3.1.1

Logistic regression

Logistic regression is a classification algorithm that can be divided into three types:

- Binomial: In binomial logistic regression, there can be only two possible values for the dependent variable, such as 0 or 1, pass or fail, etc.
- Multinomial: In multinomial logistic regression, there can be 3 or more possible unordered values/categories of the dependent variable. The dependent variable takes on three values/categories: cat, dog, sheep, so it is a polytomous variable, in the previous case it is a dichotomous variable, what they have in common is that they are nominal, that means there is only discrimination, while in the following case I can also take order into account.
- Ordinal: In ordinal logistic regression, there can be 3 or more possible ordered values/levels of the dependent variable, such as "low", "medium" or "high".

📝 3.1.2

What types of logistic regression do we know?

- Multinominal
- Binary
- Cardinal

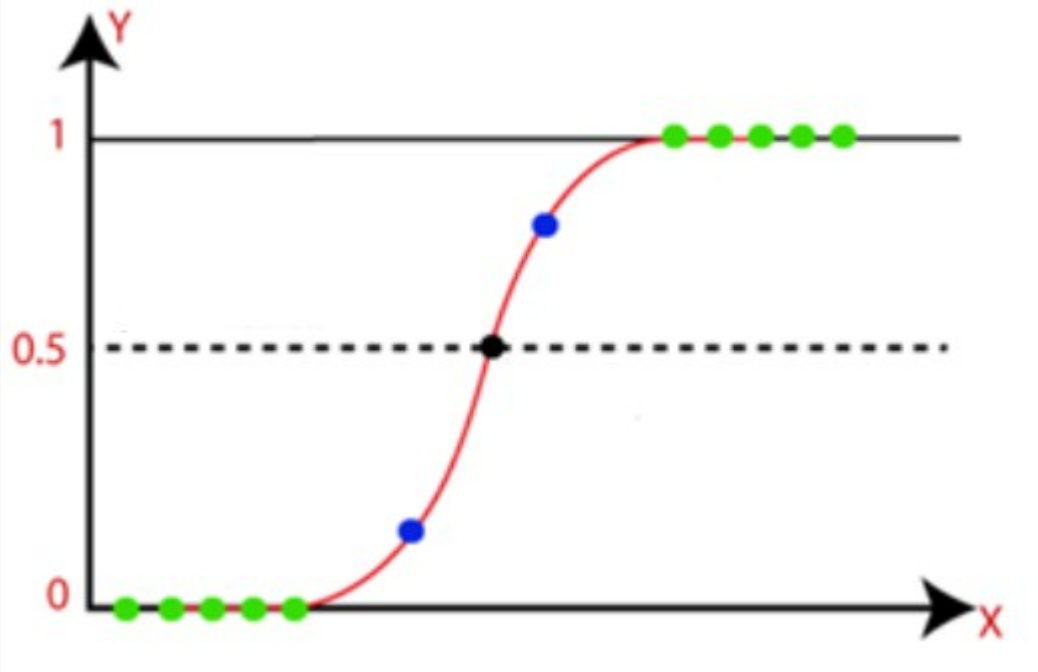
📖 3.1.3

Using logistic regression, we could train a model that would learn to estimate whether a person of a given gender with a certain age and weight might suffer from a heart attack in the future. The result of the model would be a value between 0 and 1 indicating the probability. In order to distribute the probability between the values 0 and 1, the logistic regression uses a sigmoid function (sigmoid) according to the formula below.

$$\sigma = \frac{1}{1+e^{-y}}$$

3.1.4

In the figure, we can then see that such a curve will not be a straight line, as in linear regression, but will have the shape of the letter S. In logistic regression, we use the concept of a threshold value, which defines the probability of either 0 or 1. For example, values above the threshold tend to 1 and the value tends to 0 below the thresholds.



3.1.5

The concept that logistic regression uses to make decisions is called

- sigmoid value
- threshold value
- distribution value

3.1.6

The logistic regression equation can be most easily understood from the linear regression equation. We know that the equation of a straight line can be calculated using the formula below.

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

In logistic regression, y must be equal to a value between 0 and 1. We work on this by adjusting the left side of the equation according to the following formula.

$$\sigma = \frac{1}{1+e^{-(b_0+b_1x_1+b_2x_2+\cdots+b_nx_n)}}$$

3.2 Naive Bayes Classifier

📖 3.2.1

Naive Bayes classifier

The classifier is called naive because it assumes that each input variable is independent. Naive Bayes classifier is used for both binomial and multinomial data. It is based on Bayes' theorem, also known as Bayes' rule or Bayes' law. The idea behind Bayes' theorem is to determine the probability of an outcome occurring. We call this probability conditional probability and it depends on previous results. In practice, Bayes' theorem could be used, for example, in determining Alzheimer's based on age. If Alzheimer's corresponds to a person's age, then we would be able to more accurately determine the probability of Alzheimer's. We could write the Bayes theorem according to the formula below, where A, B represent phenomena, P(A), P(B) their probability of occurrence and P(A|B) is the conditional probability of phenomenon A assuming that phenomenon B and P have occurred (B|A) is the conditional probability of event B given that event A has occurred.

$$P(A|B) = P(B|A)P(A)/P(B)$$

📖 3.2.2

According to the data distribution, there are three types of Naive Bayes models:

- **Gaussian** – The data is continuous and has a normal (Gaussian) distribution.
- **Multinomial** – In a multinomial model, variables are represented by frequencies of occurrence. In the case of classifying articles into categories, it could be about the frequency with which words occur in individual categories.

- **Bernoulli** – The Bernoulli classifier works similarly to the multinomial classifier, but the variables are Boolean. For example, whether a particular word is present in the document. This model is also known for document classification tasks.

The Bayesian classifier also comes with some problems, such as that all data must be discrete. Another problem with the algorithm is that a small test set can bias the calculation of relative probabilities. If a certain value is not found in the test set at all, the probability of its occurrence was 0.

3.2.3

Choose the correct statements about the Naive Bayes classifier

- The probability depends on the previous results
- It assumes that the input variables are independent
- It can only be used for binary data

Subsymbolical classification models

Chapter 4

4.1 SVM classifier

📖 4.1.1

Support Vector Machine (SVM)

The goal of support vector machines is to find a hyperplane, that is, a decision line or decision boundary that will help classify data points into classes. The dimensions of this surface depend on the elements present. The data points that support the hyperplane (are closest to the hyperplane) and influence its position are called support vectors.

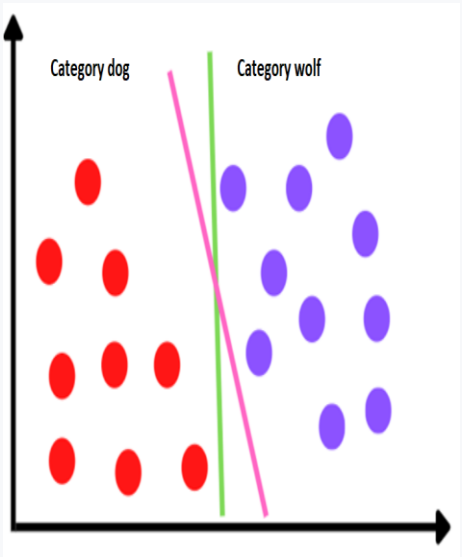
We know two types of support vector machines:

- **Linear** - if possible, split the data file with a straight line.
- **Non-linear** – if the data set cannot be divided by a straight line.

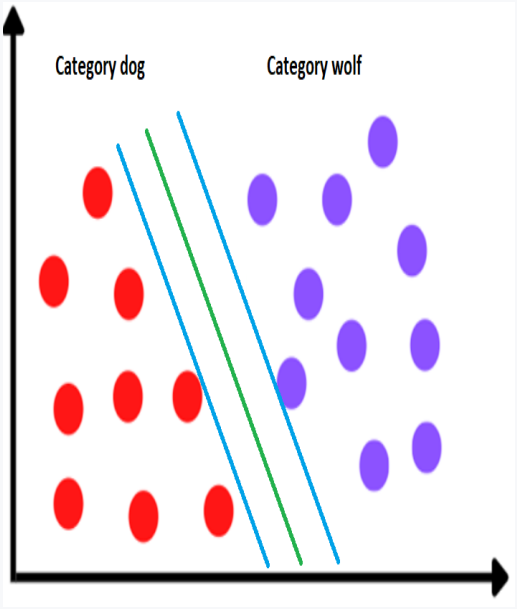
📖 4.1.2

Linear SVM

Let's consider the previous example, with the help of which we explained KNN. There are many possible ways in which these classes could be distinguished by a straight line.



The SVM algorithm will search for a line whose distance from the nearest points from both classes is maximal. Such a straight line (superplane) will be considered optimal.



In case we needed to classify a hitherto unknown data point, we would classify it according to which side of the hyperplane it is located on.

4.1.3

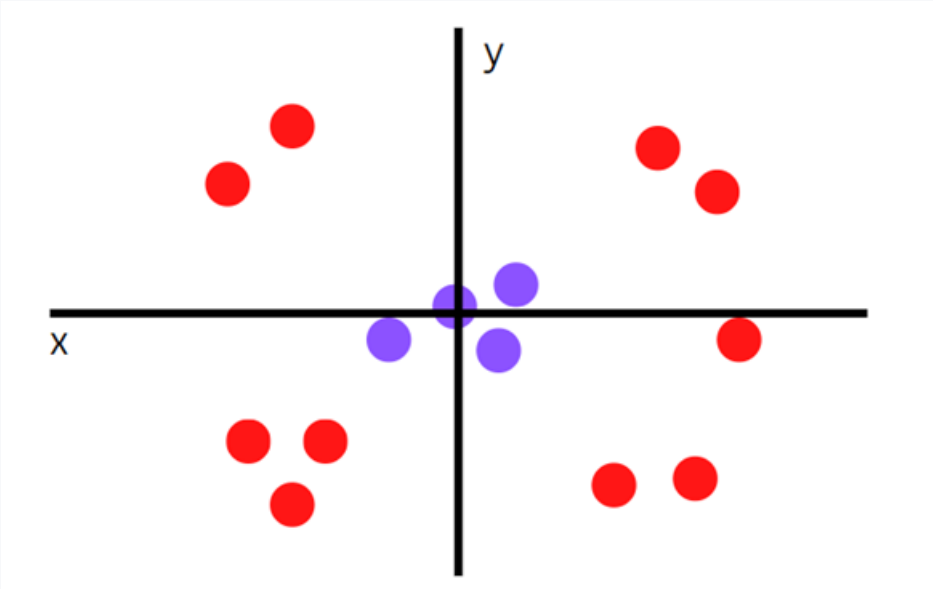
With linear SVM, the data set can be divided by a straight line

- yes
- no

4.1.4

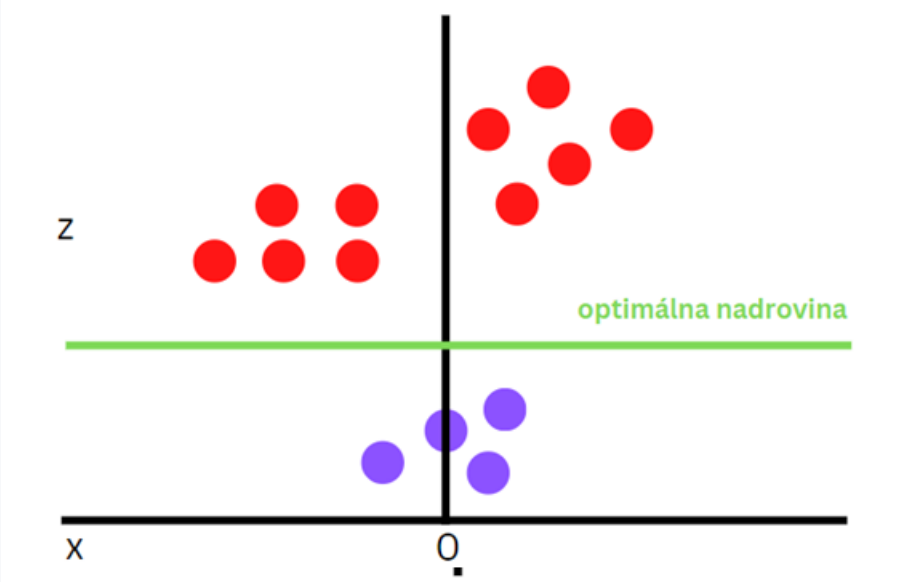
Nonlinear SVM

There are cases where the data cannot be divided using a straight line.

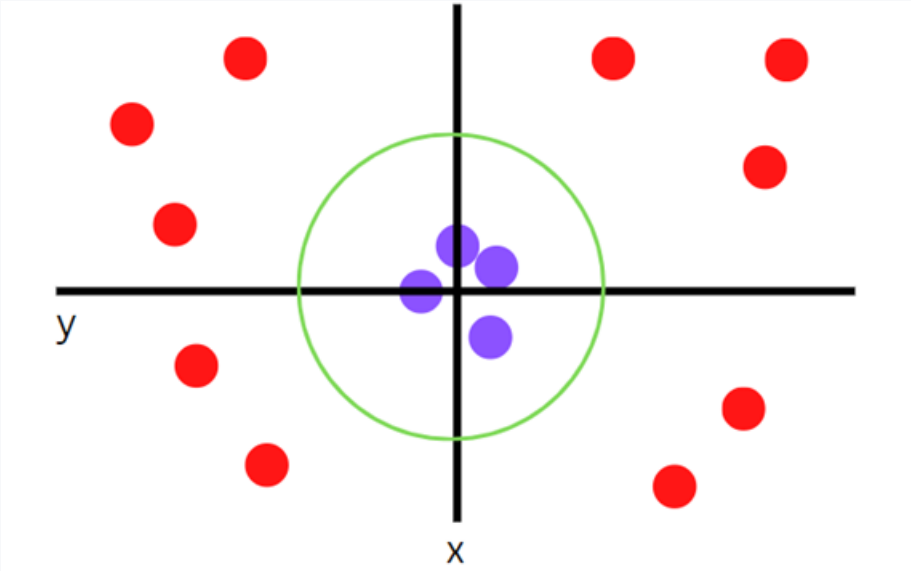


In this case, we can add the third dimension *z* and we can calculate it according to the formula.

z = x2+y2



Note that since we are now in three dimensions, the hyperplane is the plane parallel to the x-axis at some z (say z = 1). What remains is to map back to two dimensions.



4.1.5

With nonlinear SVM, another dimension is added

- yes
- no

Evaluation of classification models

Chapter 5

5.1 Accuracy

📖 5.1.1

Accuracy

Accuracy simply measures how often the classifier predicts correctly. Accuracy can be defined as the ratio of the number of correct predictions to the total number of predictions. Accuracy can be calculated according to the formula where TP represents true positive cases, TN true negative cases, FP false positive cases and FN false negative cases.

$$ACCURACY = \frac{TP+TN}{TP+TN+FP+FN}$$

When accuracy reaches high values, it does not necessarily mean that our model classifies correctly. Let's imagine that we have a classifier that determines whether there is a dog or a wolf in the image. We have a set of test images, along with labels, and we put the first dog image into the model. We assume that our model predicts that it is a dog, and then compare the prediction with the correct label. If the model predicts that it is a wolf and we compare it to the correct label, the model is wrong.

We repeat this process for all images in the test set. Finally, we will have the numbers of TP, TN, FP, FN. However, in reality, it is very rare that all wrong or right matches will be balanced.

📝 5.1.2

Accuracy môžeme definovať ako pomer počtu správnych predpovedí a celkového počtu predpovedí

- yes
- no

5.2 Accuracy, coverage and their harmonic mean

📖 5.2.1

Precision

This metric explains how many of the correctly predicted cases actually turned out to be positive and is defined as the number of actual positives divided by the number of predicted positives.

$$PRECISION = \frac{TP}{TP+FP}$$

5.2.2

Precision expresses how many of the correctly predicted cases actually turned out to be positive

- yes
- no

5.2.3

Recall

Recall explains how many true positives we were able to correctly predict using our model and is defined as the number of true positives divided by the total number of true positives.

$$RECALL = \frac{TP}{TP+FN}$$

5.2.4

Coverage (recall) refers to how many of the correctly predicted cases actually turned out to be positive

- yes
- no

5.2.5

F1 score

It provides a combined idea of precision and recall metrics. It is at its maximum when Accuracy equals Recall. Its advantage is that it penalizes extreme values.

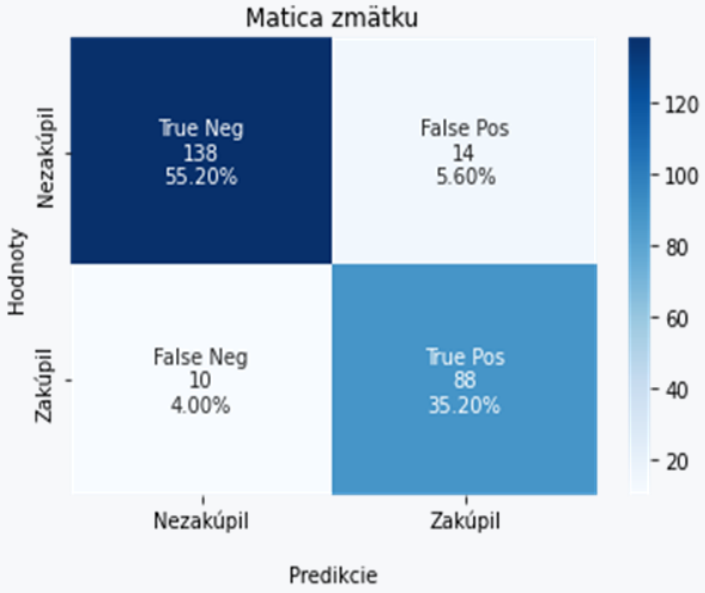
$$F1 = 2 * \frac{PRECISION\ x\ RECALL}{PRECISISION+RECALL}$$

5.3 Confusion matrix

📖 5.3.1

Confusion matrix

The confusion matrix is a table that contains the values of TP, TN, FP and FN. We read it so that the row represents the class and the columns represent the number of true and false cases in the class.



5.4 AUC-ROC

📖 5.4.1

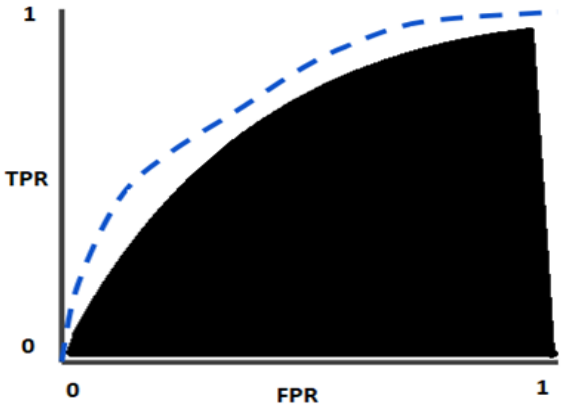
AUC-ROC

The ROC curve (receiver operating characteristic curve) shows the performance of the model at different thresholds. It contains two parameters namely the true positive rate of TPR and the false positive rate of FPR.

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

The area under the curve (AUC) is a measure of the classifier's ability to discriminate between classes. The larger the AUC, the better the performance of the model, and thus when the AUC is equal to 1, the classifier is able to perfectly distinguish between all positive and negative points of the class. When AUC equals 0, the classifier would predict all negative points as positive and vice versa. When the AUC is 0.5, the classifier is unable to distinguish between positive and negative classes.



5.4.2

When AUC is equal to 0, the classifier is able to perfectly distinguish between all positive and negative points of the class.

- yes
- no

5.5 Log loss

5.5.1

Log Loss, Cross Entropy Loss

Logarithm Loss or Cross Entropy Loss is one of the main metrics to assess the performance of a classification problem. For one sample with a true label y ∈ {0,1} and a probability estimate p = Pr(y = 1), the log loss is calculated according to the formula.

$$logloss_{(N=1)} = y \log(p) + (1 - y) \log(1 - p)$$

Cross-entropy loss, or logarithmic loss, measures the performance of a classification model that outputs a probability value between 0 and 1. Cross-entropy loss increases when the predicted probability differs from the actual label. A perfect model would have a log loss of 0.

5.5.2

Cross-entropy loss or logarithmic loss measures the performance of a classification model whose output is a probability value between 0 and -1

- no
- yes

Implementation of classification models in Python

Chapter 6

6.1 Classification models

6.1.1

Implementation in Python

In this section, we will show how classification can be easily implemented in Python. It is enough if we create one python file or jupyter notebook. Before we start writing the code, we need to install the libraries numpy pandas, matplotlib, seaborn and sklrearn.

Dataset

The data file with which we will work is the database of potential customers of a company that deals with the sale of fireplaces. It contains a thousand records, and using classifiers we will try to model how many people from our dataset bought a fireplace in the last year. The dataset contains the customer's age, monthly salary and information on whether the customer purchased the given product.

We import the numpy library, for working with fields, pandas, for working with our data file. The seaborn and matplotlib libraries will later be used to plot the data, and sklearn will be used to train our classification models.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
```

We load our dataset into the dataset variable using the pandas library and the read_csv method. This variable will be of type Dataframe. Using the print() method, we print the first and last 5 records. From the listing, we can further see that our dataset has a thousand rows and three columns.

```
dataset =
pd.read_csv("https://raw.githubusercontent.com/livi83/customer
s-dataset/main/dataset.csv", sep=";")
print(dataset)
```

```
print(dataset.describe())
```

We check if we don't have empty values in the dataset using the isnull method. We add their numbers using the sum() method.

```
print(dataset.isnull().sum())
```

Using the matplotlib library, we can visualize how many customers have purchased the product and how many customers have not purchased the product.

```
dataset.Zakúpil.value_counts().plot(kind='bar')
plt.xlabel('Zakúpil')
plt.ylabel('počet zákazníkov')
plt.title('Počet zákazníkov vzhľadom k tomu, či produkt
zakúpili')
plt.show()
```

We will access individual rows and columns using the iloc method. It accepts a list of rows and columns as parameters. If we put a colon in any of the positions, we say that we want all values (in our case, all rows). The list x represents our independent variables and contains all the rows of the zero and first column (we number from zero), and y represents our dependent variable, which contains all the rows of the last column.

```
x = dataset.iloc[:,[0,1]].values
y = dataset.iloc[:,2].values
print(x)
print(y)
```

Splitting the dataset into a training and testing set

We have loaded the data file, we need to divide it into a training and a test part. For this, we will use the sklearn library and its train_test_split method, with which we will split the data in a ratio of 75:25.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.25, random_state=0)
```

In this step, we created the variables x_train, x_test, y_train, y_test. As for y_train, y_test, the values are in the range 0-1 and therefore it is not necessary to modify them, but x_train, x_test must undergo standardization before training. For this purpose, we can use the StandardScaler class from the sklearn library, which subtracts the mean from their values and divides them by the standard deviation to adjust the data to unit variance.

```
#normalizacia
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

Import of evaluation metrics

Before we start with the implementation of the classifiers, we can import the evaluation metrics. We import the confusion matrix, accuracy, precision, recall and harmonic mean F1.

```
# Evalvácia
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score
```

Next, we create a con_matrix function that accepts the predicted values as a parameter and renders the confusion matrix

```
def conf_matrix(y_pred):
  cm = confusion_matrix(y_test, y_pred)

  names = ['True Neg','False Pos','False Neg','True Pos']
  counts = ["{0:0.0f}".format(value) for value in
                  cm.flatten()]
  percentages = ["{0:.2%}".format(value) for value in
                       cm.flatten()/np.sum(cm)]


  labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
            zip(names,counts,percentages)]

  labels = np.asarray(labels).reshape(2,2)

  ax = sns.heatmap(cm, annot=labels, fmt='', cmap='Blues')

  ax.set_title('Matica zmätku');
  ax.set_xlabel('\nPredikcie')
  ax.set_ylabel('Hodnoty');
  ax.xaxis.set_ticklabels(['Nezakúpil','Zakúpil'])
  ax.yaxis.set_ticklabels(['Nezakúpil','Zakúpil'])
```

```
plt.show()
```

Implementation of Logistic Regression

In the basic version, logistic regression can be implemented by calling the LogisticRegression class. Subsequently, we can use the imported metrics to determine the performance of our classifier. The ratio of the number of correct predictions to the total number of predictions is 85.6%, the rate of correctly identified positive cases out of all predicted positive cases is 82.2%, the rate of correctly identified positive cases out of all actual positive cases is 80.6%, and the harmonic mean rate is 81.4%. By calling the conf_matrix function, we can output the confusion matrix. Our model correctly classified 135 cases and incorrectly 17 for the class it did not purchase. The rest of the test data therefore belongs to the purchased class, while the model estimated 79 of them correctly and 19 incorrectly.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression().fit(x_train,y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')
conf_matrix(y_pred)
```

Implementation of the Naïve Bayes classifier

The ratio of the number of correct predictions to the total number of predictions is 88.8%, the rate of correctly identified positive cases out of all predicted positive cases is 85.7%, the rate of correctly identified positive cases out of all actual positive cases is 85.7%, and the harmonic mean rate is 85.7%.

By calling the conf_matrix function, we can output the confusion matrix. Our model correctly classified 138 cases for the class did not buy and incorrectly 14. Therefore, the rest of the test data belongs to the class purchased, while the model estimated 84 of them correctly and 14 incorrectly.

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB().fit(x_train,y_train)
y_pred = model.predict(x_test)
```

```
#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

Implementation of KNN classifier

To implement the KNN classifier, it is important to find the optimal value of K. Let's try values of K from 1 to 40 and try to find out at which value the classifier achieves the highest accuracy. As we can see from the graph, the optimal number of nearest neighbors will be 2.

```
from sklearn.neighbors import KNeighborsClassifier
test_error_rates = []

for k in range(1,40):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(x_train,y_train)

    y_pred = model.predict(x_test)

    test_error = 1 - accuracy_score(y_test,y_pred)
    test_error_rates.append(test_error)

min_value = min(test_error_rates)
print(test_error_rates.index(min_value))

plt.figure(figsize=(6,4),dpi=100)
plt.plot(range(1,40),test_error_rates,label='Test Error')
plt.legend()
plt.ylabel('Error Rate')
plt.xlabel("K Value")
```

The ratio of the number of correct predictions to the total number of predictions is 90%, the rate of correctly identified positive cases out of all predicted positive cases is 88.5%, the rate of correctly identified positive cases out of all actual positive cases is 86.7%, and the harmonic mean of the rates is 87. 6%.

By calling the conf_matrix function, we can print out the confusion matrix. Our model correctly classified 141 cases for the class did not buy and incorrectly 11.

The rest of the test data therefore belongs to the class bought, while the model estimated 85 of them correctly and 13 incorrectly.

```
model = KNeighborsClassifier(n_neighbors=4)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

Implementation of SVM

Our data is linear, so we will use the linear SVM implementation.

The ratio of the number of correct predictions to the total number of predictions is 90.4%, the rate of correctly identified positive cases from all predicted positive cases is 88.5%, the rate of correctly identified positive cases from all actual positive cases is 86.7%, and their harmonic mean is 87.6%. By calling the conf_matrix function, we can output the confusion matrix. Our model correctly classified 141 cases for the class did not buy and incorrectly 11. The rest of the test data therefore belongs to the class bought, while the model estimated 85 of them correctly and 13 incorrectly.

```
from sklearn.svm import SVC
model =SVC(kernel='linear').fit(x_train,y_train)
svm_y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

Implementation of the CART decision tree

To implement a decision tree using the sklearn library, we call the DecisionTreeClassifier class. The default metric for measuring the quality of

distribution is the Gini index. If we wanted to change the metric to entropy, it would be enough to insert the criterion parameter with the entropy value into the classifier.

We already used the random_state parameter when dividing the dataset into training and testing sets. This parameter controls how the data will be divided and can take the value none, which will give us a random distribution of data at each start, we can insert an integer into it. A value from 0 to 42 is usually chosen to ensure a random but always equal distribution of data. In other words, if we give the random_state parameter an integer value, we specify randomness in the selection. However, the data selection will be the same after each run. Likewise, if we want to control the randomness of the split in decision trees or random forests, the random_state parameter is useful.

The ratio of the number of correct predictions to the total number of predictions is 87.6%, the rate of correctly identified positive cases from all predicted positive cases is 86%, the rate of correctly identified positive cases from all actual positive cases is 81.6%, and their harmonic mean is 83. 7%. By calling the conf_matrix function, we can output the confusion matrix. Our model correctly classified 139 cases for the class did not buy and incorrectly 13. Therefore, the rest of the test data belongs to the class bought, while the model estimated 80 of them correctly and 18 incorrectly.

```
from sklearn.tree import DecisionTreeClassifier
model= DecisionTreeClassifier(random_state=0).fit(x_train,
y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

Ensemble learning

Chapter 7

7.1 The introduction into ensemble learning

🕮 7.1.1

Ensemble learning

So far we have shown how to train one specific model. In practice, the concept of ensemble learning (Ensemble Learning) is used more, the idea of which is to train several models using the same algorithm. The combination of several models makes it possible to achieve a better classification, compared to the use of a single classifier.

According to the method of work, ensemble learning methods can be divided into sequential and parallel.

Sequential methods

- They build the model sequentially, step by step.
- They support dependency between base classifiers
- For example Adaboost - multiple weak classifiers are trained sequentially and each model tries to correct the errors of the previous model using weighting. This process is iteratively repeated until a stopping condition is met, which can be, for example, the maximum number of iterations or a certain accuracy limit. The resulting prediction is a combination of all individual models with weights proportional to accuracy.

Parallel methods

- Classifiers work in parallel.
- They support independence between classifiers.
- The resulting prediction is given by averaging or voting.
- Random Forest, Gradient Boosting.
- Built on random selections.

According to the type of classifiers used, ensemble learning methods can be divided into homogeneous and heterogeneous.

Homogeneous methods

- All models are of the same type (eg only decision trees).
- Bagging, Boosting.
- The resulting prediction is given by majority voting or averaging.

Heterogeneous methods

- Models of different types.
- Stacking.
- The resulting prediction is given by weighted voting or linear combination.

7.1.2

Sequential methods support dependency between underlying classifiers

- yes
- no

7.1.3

Parallel methods support dependency between underlying classifiers

- yes
- no

7.1.4

The resulting prediction of parallel models is a combination of all individual models with weights proportional to accuracy.

- no
- yes

7.1.5

The resulting prediction of sequential models is given by the majority vote

- no
- yes

📖 7.1.6

Bagging

The term Bagging originated from the words Bootstrap Aggregating. Bootstrap is based on random sampling of small parts of the dataset, and these sets can be replaced. Such random sampling can help to better interpret the standard deviation in a dataset.

Bagging is a simple method of training a number of different models on different randomly selected subsets of the training set and then combining their predictions using voting. Thus, the data is first divided into several training and test sets, on which the models are subsequently trained and the prediction of the majority is the result.

Advantages of bagging:

- increases the accuracy score of the model,
- can handle overfitting,
- reduces distortion and dispersion errors,
- simple implementation.

📖 7.1.7

BOOSTING

To understand Boosting, it is important to realize that Boosting is a generic algorithm rather than a specific model. Boosting is a learning method that combines a group of weak classifiers into a strong classifier to minimize training errors. In contrast to bagging, we can talk about "teamwork" as the models run in parallel. In boosting, a random sample of data is taken, fitted with a model, and then trained sequentially—that is, each model tries to compensate for the weaknesses of its predecessor. At each iteration, the weak rules from each individual classifier are combined into a single, strong prediction rule.

Adaboost (Adaptive Boosting)

Adaboost is a boosting-based technique that is based on combining multiple weak classifiers into one strong classifier. A single split decision tree can be a weak classifier in Adaboost. At the moment the first distribution is created, all observations are given equal weight. Thus, for error correction, misclassified cases receive a higher weight.

Gradient Boosting

Like Adaboost, Gradient Boosting also tries to improve on its predecessor but uses a slightly different method. It does not try to change the weights for misclassified cases but tries to correct its passer in order to reduce the error rate. A modification of Gradient Boosting is the so-called XGBoost, which consists of boosted decision trees for higher performance.

📖 7.1.8

Stacking

Stacking is another method of ensemble learning. It is based on the composition of classification models and consists of two-layer estimates. The first layer is composed of all the underlying models that are used to predict the outputs on the test data sets. The second layer consists of a meta classifier that takes all the predictions of the underlying models as input and generates new predictions. The advantage of stacking is that it can take advantage of a set of well-performing models for a classification or regression task and produce predictions that perform better than any single model in the ensemble.