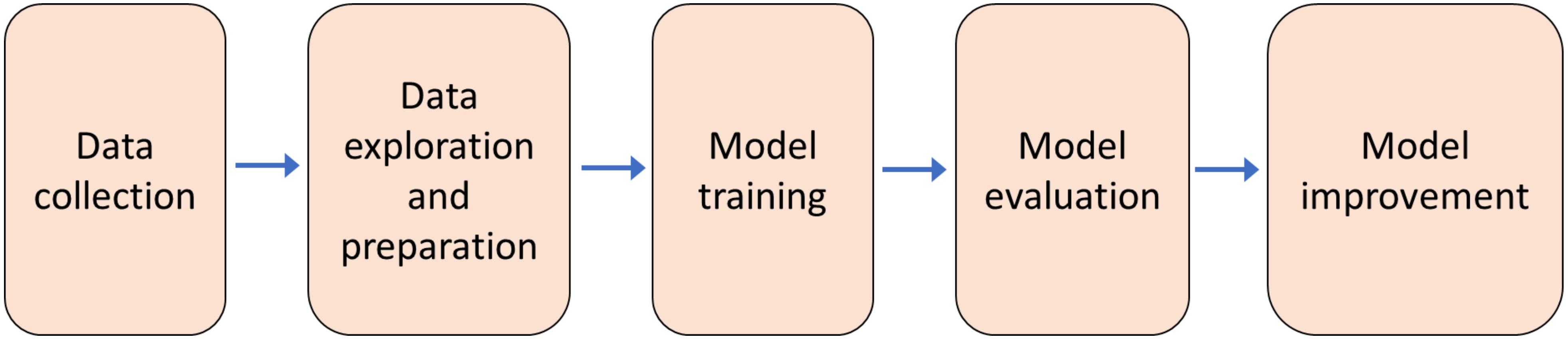
Machine Learning

Machine learning (ML) is the study of computer algorithms to learn how to react in a certain situation or recognize patterns.

According to Arthur Samuel (1959): *"Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed"*.

The field of machine learning provides a set of algorithms that transform data into actionable knowledge.

Machine Learning Process



Machine Learning Process Steps

1. Acquisition of the source database
2. Data pre-processing: filtering, cleaning and conversion to the required format
3. Data selection: selection of attributes and values
4. Model training
5. Model evaluation and improvement
6. Model deployment

Machine Learnig in Python

scikit-learn library

A core library for machine learning in Python.

Library website: https://scikit-learn.org/stable/

Documentation: https://scikit-learn.org/stable/modules/classes.html

Installation: `pip install -U scikit-learn`

Acquisition of the source database

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/employee.csv', sep=str(";")))
```

Data exploration

List of records

In [5]:
```
data.head()
```

Out[5]:

	name	age	income	gender	department	grade	performance_score
0	Allen Smith	45.0	NaN	NaN	Operations	G3	723
1	S Kumar	NaN	16000.0	F	Finance	G0	520
2	Jack Morgan	32.0	35000.0	M	Finance	G2	674
3	Ying Chin	45.0	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711

```
In [6]: data.tail()
```

Out[6]:

	name	age	income	gender	department	grade	performance_score
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711
5	Satyam Sharma	NaN	62000.0	NaN	Sales	G3	649
6	James Authur	54.0	NaN	F	Operations	G3	53
7	Josh Wills	54.0	52000.0	F	Finance	G3	901
8	Leo Duck	23.0	98000.0	M	Sales	G4	709

Information on data structure

```
In [7]: print(f"Number of records: {data.shape[0]}\nNumber of columns: {data.shape[1]}")
```

```
Number of records: 9
Number of columns: 7
```

```
In [8]: print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   name               9 non-null      object 
 1   age                7 non-null      float64
 2   income             7 non-null      float64
 3   gender             7 non-null      object 
 4   department         9 non-null      object 
 5   grade              9 non-null      object 
 6   performance_score  9 non-null      int64  
dtypes: float64(2), int64(1), object(4)
memory usage: 632.0+ bytes
None
```

Descriptive statistics

```
In [9]: data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
age	7.0	40.428571	12.204605	23.0	31.0	45.0	49.5	54.0
income	7.0	52857.142857	26028.372797	16000.0	38500.0	52000.0	63500.0	98000.0
performance_score	9.0	610.666667	235.671912	53.0	556.0	674.0	711.0	901.0

In [10]:
```
description = data.describe().T
description[(description['count'] < data.shape[0])]
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
age	7.0	40.428571	12.204605	23.0	31.0	45.0	49.5	54.0
income	7.0	52857.142857	26028.372797	16000.0	38500.0	52000.0	63500.0	98000.0

In [11]:
```
data.describe(include=['object']).T
```

Out[11]:

	count	unique	top	freq
name	9	9	Allen Smith	1
gender	7	2	F	5
department	9	3	Operations	3
grade	9	4	G3	5

Data pre-processing

Handling missing values

- Deletion of records with missing values

In [12]:
```
data_1 = data.dropna()
data_1
```

Out[12]:

	name	age	income	gender	department	grade	performance_score
2	Jack Morgan	32.0	35000.0	M	Finance	G2	674
3	Ying Chin	45.0	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711
7	Josh Wills	54.0	52000.0	F	Finance	G3	901
8	Leo Duck	23.0	98000.0	M	Sales	G4	709

- Filling in a value such as mean, median or mode

In [13]:
```
data.fillna({'age':data.age.mean()}, inplace=True)
data
```

Out[13]:

	name	age	income	gender	department	grade	performance_score
0	Allen Smith	45.000000	NaN	NaN	Operations	G3	723
1	S Kumar	40.428571	16000.0	F	Finance	G0	520
2	Jack Morgan	32.000000	35000.0	M	Finance	G2	674
3	Ying Chin	45.000000	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.000000	42000.0	F	Operations	G2	711
5	Satyam Sharma	40.428571	62000.0	NaN	Sales	G3	649
6	James Authur	54.000000	NaN	F	Operations	G3	53
7	Josh Wills	54.000000	52000.0	F	Finance	G3	901
8	Leo Duck	23.000000	98000.0	M	Sales	G4	709

In [14]:
```
data.fillna({'income':data.income.median()}, inplace=True)
data
```

Out[14]:

	name	age	income	gender	department	grade	performance_score
0	Allen Smith	45.000000	52000.0	NaN	Operations	G3	723
1	S Kumar	40.428571	16000.0	F	Finance	G0	520
2	Jack Morgan	32.000000	35000.0	M	Finance	G2	674
3	Ying Chin	45.000000	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.000000	42000.0	F	Operations	G2	711
5	Satyam Sharma	40.428571	62000.0	NaN	Sales	G3	649
6	James Authur	54.000000	52000.0	F	Operations	G3	53
7	Josh Wills	54.000000	52000.0	F	Finance	G3	901
8	Leo Duck	23.000000	98000.0	M	Sales	G4	709

In [15]:

```
data.fillna({'gender':data.gender.mode()[0]}, inplace=True)
data
```

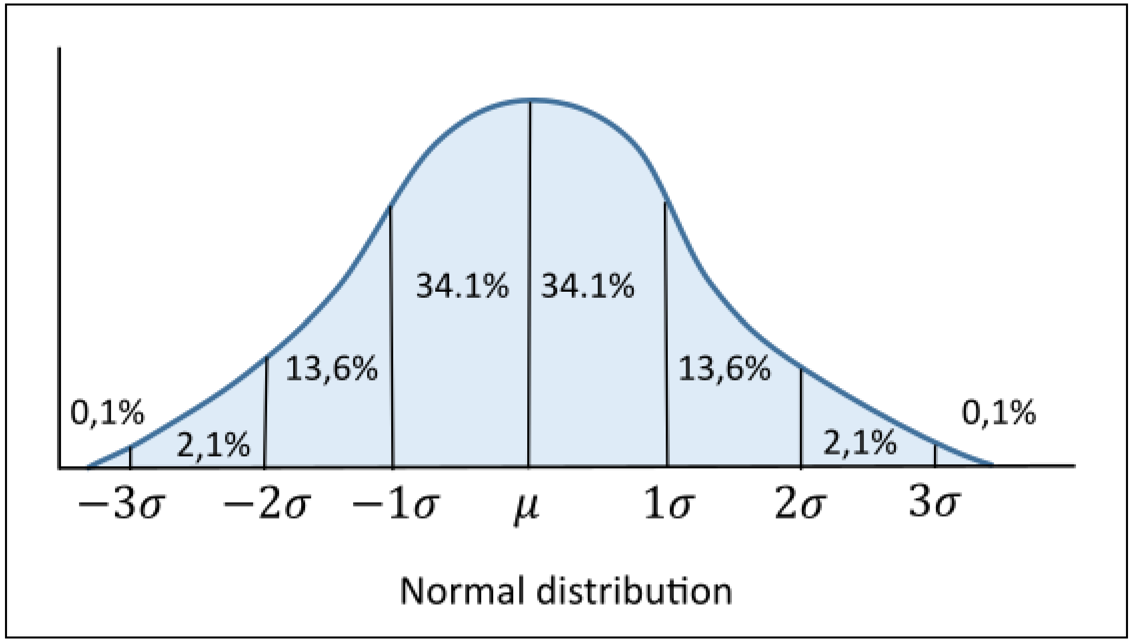
Out[15]:

	name	age	income	gender	department	grade	performance_score
0	Allen Smith	45.000000	52000.0	F	Operations	G3	723
1	S Kumar	40.428571	16000.0	F	Finance	G0	520
2	Jack Morgan	32.000000	35000.0	M	Finance	G2	674
3	Ying Chin	45.000000	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.000000	42000.0	F	Operations	G2	711
5	Satyam Sharma	40.428571	62000.0	F	Sales	G3	649
6	James Authur	54.000000	52000.0	F	Operations	G3	53
7	Josh Wills	54.000000	52000.0	F	Finance	G3	901
8	Leo Duck	23.000000	98000.0	M	Sales	G4	709

Detection of outliers

Rule 68-95-99.7

The 68-95-99.7 rule states that, for a normal distribution, 68 per cent of the values are at most one standard deviation away from the mean, while 95 per cent and 99.7 per cent of the values are no more than two or three standard deviations away from the mean, respectively.



```
In [16]: upper_limit= data['performance_score'].mean() + 3 * data['performance_score'].std()
         lower_limit= data['performance_score'].mean() - 3 * data['performance_score'].std()
         print(upper_limit)
         print(lower_limit)
```

```
1317.6824019936817
-96.34906866034851
```

```
In [17]: outliers = data[(data['performance_score'] < upper_limit) & (data['performance_score'] > lower_limit)]
         outliers
```

Out[17]:

	name	age	income	gender	department	grade	performance_score
0	Allen Smith	45.000000	52000.0	F	Operations	G3	723
1	S Kumar	40.428571	16000.0	F	Finance	G0	520
2	Jack Morgan	32.000000	35000.0	M	Finance	G2	674
3	Ying Chin	45.000000	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.000000	42000.0	F	Operations	G2	711
5	Satyam Sharma	40.428571	62000.0	F	Sales	G3	649
6	James Authur	54.000000	52000.0	F	Operations	G3	53
7	Josh Wills	54.000000	52000.0	F	Finance	G3	901
8	Leo Duck	23.000000	98000.0	M	Sales	G4	709

In [18]:
```
outliers = data[(data['performance_score'] > upper_limit) | (data['performance_score'] < lower_limit)]
outliers
```

Out[18]:

name	age	income	gender	department	grade	performance_score

Interquartile Range (IQR)

IQR is a statistical measure of the dispersion of data, known as the central dispersion or H-dispersion. It is calculated as the difference between the third and first quartiles.

$$IQR = Q3 - Q1$$

It is understood that:

$Q1 - 1,5 \cdot IQR$ - is the lower limit

$Q3 + 1,5 \cdot IQR$ - is the upper limit

and values outside the interval are outliers.

In [19]:
```
# 1. Q1,Q3
Q1 = data['performance_score'].quantile(0.25)
Q3 = data['performance_score'].quantile(0.75)
```

```
# 2. Q1,Q3
Q1,Q3 = np.percentile(data['performance_score'], [25,75])

# IQR
IQR = Q3 - Q1

# upper limit
upper_limit = Q3 + 1.5*IQR
print(upper_limit)

# lower limit
lower_limit = Q1 - 1.5*IQR
print(lower_limit)

# Find outliers
outliers = data[(data['performance_score'] > upper_limit) | (data['performance_score'] < lower_limit)]
outliers
```

```
943.5
323.5
```

Out[19]:

	name	age	income	gender	department	grade	performance_score
6	James Authur	54.0	52000.0	F	Operations	G3	53

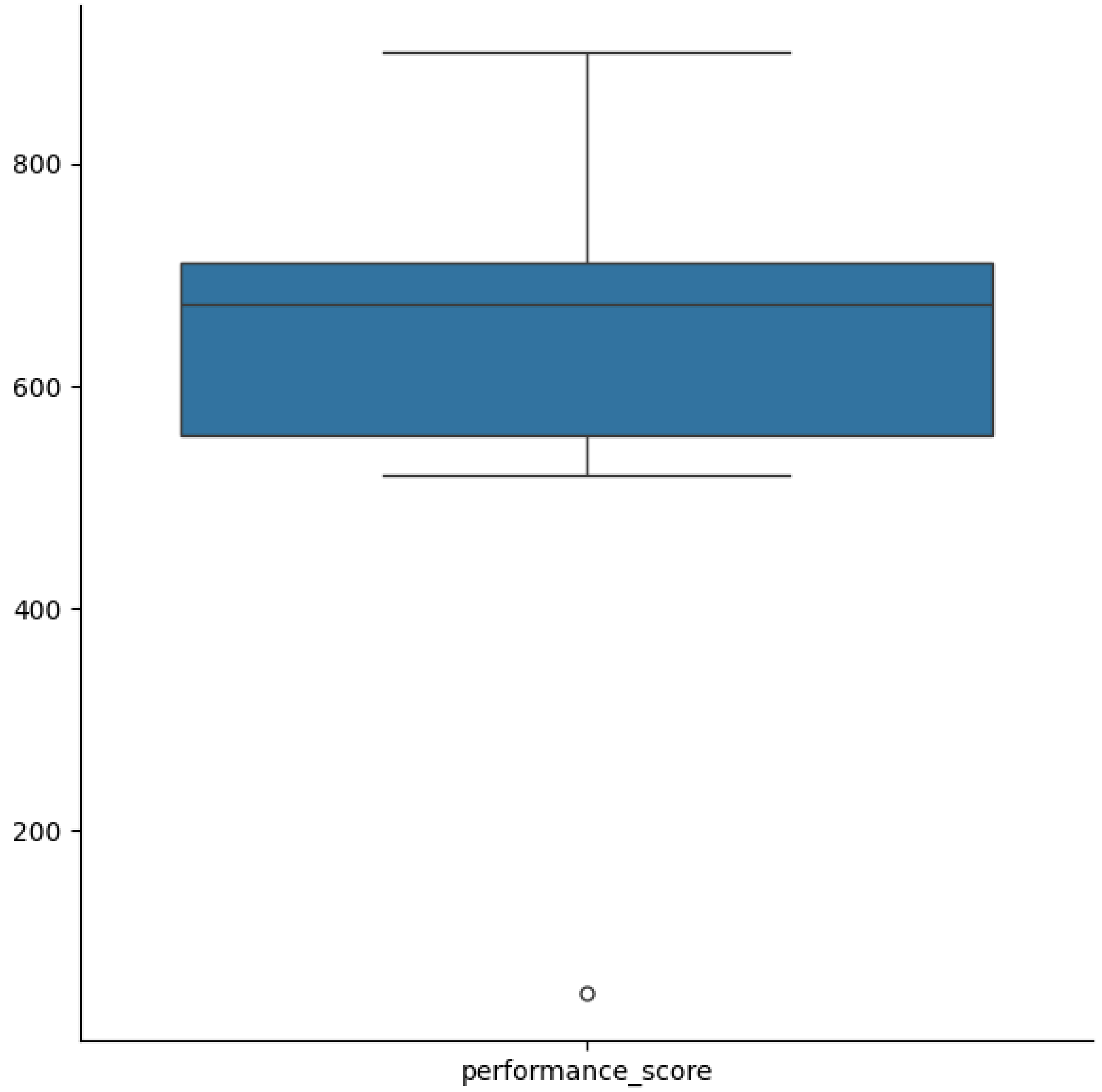
In [20]:
```
outliers = data[(data['performance_score'] < upper_limit) & (data['performance_score'] > lower_limit)]
outliers
```

Out[20]:

	name	age	income	gender	department	grade	performance_score
0	Allen Smith	45.000000	52000.0	F	Operations	G3	723
1	S Kumar	40.428571	16000.0	F	Finance	G0	520
2	Jack Morgan	32.000000	35000.0	M	Finance	G2	674
3	Ying Chin	45.000000	65000.0	F	Sales	G3	556
4	Dheeraj Patel	30.000000	42000.0	F	Operations	G2	711
5	Satyam Sharma	40.428571	62000.0	F	Sales	G3	649
7	Josh Wills	54.000000	52000.0	F	Finance	G3	901
8	Leo Duck	23.000000	98000.0	M	Sales	G4	709

Visualization using a box plot

```
In [21]: _ = sns.catplot(data=data[['performance_score']], kind='box', height=6)
```



Percentyle

A percentile is a statistical measure that divides data into 100 equal-sized groups. Its value indicates what percentage of the population falls below a given value. For example, the 95th percentile means that 95% of people fall into that category.

```
In [22]: # Percentiles 1, 99
         lower_limit, upper_limit = np.percentile(data['performance_score'] , [1,99])
```

```
In [23]: print(upper_limit)
         print(lower_limit)

         data[(data['performance_score'] > upper_limit) | (data['performance_score'] < lower_limit)]
```

```
886.76
90.36
```

Out[23]:

	name	age	income	gender	department	grade	performance_score
6	James Authur	54.0	52000.0	F	Operations	G3	53
7	Josh Wills	54.0	52000.0	F	Finance	G3	901

Feature coding techniques

Machine learning models are mathematical models that often require numerical and integer values for calculation. Such models cannot work on categorical features. Therefore, it is often necessary to convert categorical features to numerical ones. The performance of a machine learning model is affected by the encoding technique we use.

One-Hot Encoding

One-Hot Encoding converts a categorical column into a set of columns with labels that are category values. Then, a value of 1 is inserted in the column corresponding to the given label value and a value of 0 in the others.

sklearn library - `OneHotEncoder`

```
In [24]: from sklearn.preprocessing import OneHotEncoder

         data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/employee.csv', sep=str(";")))

         onehotencoder = OneHotEncoder()
         data['gender'] = data['gender'].fillna(data['gender'].mode()[0])
         encoded_data = onehotencoder.fit_transform(data[['gender']]).toarray()
         encoded_data = pd.DataFrame(encoded_data, columns=onehotencoder.categories_[0])
         data = data.join(encoded_data)
         data.head()
```

Out[24]:

```
           name   age   income gender  department grade  performance_score    F    M
0   Allen Smith  45.0      NaN      F  Operations    G3                723  1.0  0.0
1       S Kumar   NaN  16000.0      F     Finance    G0                520  1.0  0.0
2   Jack Morgan  32.0  35000.0      M     Finance    G2                674  0.0  1.0
3     Ying Chin  45.0  65000.0      F       Sales    G3                556  1.0  0.0
4  Dheeraj Patel 30.0  42000.0      F  Operations    G2                711  1.0  0.0
```

Pandas library - `get_dummies()`

In [25]:

```python
data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/employee.csv', sep=str(";")))
data['gender'] = data['gender'].fillna(data['gender'].mode()[0])
encoded_data = pd.get_dummies(data['gender'])
data = data.join(encoded_data)
data.head()
```

Out[25]:

```
           name   age   income gender  department grade  performance_score      F      M
0   Allen Smith  45.0      NaN      F  Operations    G3                723   True  False
1       S Kumar   NaN  16000.0      F     Finance    G0                520   True  False
2   Jack Morgan  32.0  35000.0      M     Finance    G2                674  False   True
3     Ying Chin  45.0  65000.0      F       Sales    G3                556   True  False
4  Dheeraj Patel 30.0  42000.0      F  Operations    G2                711   True  False
```

Label Encoding

Label Encoding is also known as integer encoding. Integer encoding replaces categorical values with numeric values - unique category values are replaced by a sequence of integer values.

sklearn library - `LabelEncoder`

In [26]:

```python
from sklearn.preprocessing import LabelEncoder

data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/employee.csv', sep=str(";")))

label_encoder = LabelEncoder()
```

```
encoded_data = label_encoder.fit_transform(data['department'])
encoded_data = pd.DataFrame(encoded_data, columns=['department_encoded'])
data = data.join(encoded_data)
data
```

Out[26]:

	name	age	income	gender	department	grade	performance_score	department_encoded
0	Allen Smith	45.0	NaN	NaN	Operations	G3	723	1
1	S Kumar	NaN	16000.0	F	Finance	G0	520	0
2	Jack Morgan	32.0	35000.0	M	Finance	G2	674	0
3	Ying Chin	45.0	65000.0	F	Sales	G3	556	2
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711	1
5	Satyam Sharma	NaN	62000.0	NaN	Sales	G3	649	2
6	James Authur	54.0	NaN	F	Operations	G3	53	1
7	Josh Wills	54.0	52000.0	F	Finance	G3	901	0
8	Leo Duck	23.0	98000.0	M	Sales	G4	709	2

In [27]:
```
departments_names = label_encoder.inverse_transform(data['department_encoded'])
print(departments_names)
```

```
['Operations' 'Finance' 'Finance' 'Sales' 'Operations' 'Sales'
 'Operations' 'Finance' 'Sales']
```

Ordinal Encoding

Ordinal Encoding is similar to label encoding, except that there is an order to the encoding. The output encoding starts at 0 and ends at one value less than the category size.

sklearn library - `OrdinalEncoder`

In [28]:
```
from sklearn.preprocessing import OrdinalEncoder

data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/employee.csv', sep=str(";")))

order_encoder = OrdinalEncoder(categories=['G0','G1','G2','G3','G4'])
data['grade_encoded'] = label_encoder.fit_transform(data['grade'])
data
```

Out[28]:

	name	age	income	gender	department	grade	performance_score	grade_encoded
0	Allen Smith	45.0	NaN	NaN	Operations	G3	723	2
1	S Kumar	NaN	16000.0	F	Finance	G0	520	0
2	Jack Morgan	32.0	35000.0	M	Finance	G2	674	1
3	Ying Chin	45.0	65000.0	F	Sales	G3	556	2
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711	1
5	Satyam Sharma	NaN	62000.0	NaN	Sales	G3	649	2
6	James Authur	54.0	NaN	F	Operations	G3	53	2
7	Josh Wills	54.0	52000.0	F	Finance	G3	901	2
8	Leo Duck	23.0	98000.0	M	Sales	G4	709	3

Data normalisation

In data, most features have different ranges, magnitudes, and units. From a data analysis perspective, large-scale features will be more important to machine learning models than smaller-scale features. Normalizing data brings all features to the same level of magnitude.

This is not mandatory for all types of algorithms; but some of them clearly need scaled data, such as those that rely on Euclidean distance measures: K-Nearest Neighbors or Kmeans clustering algorithm.

Z-score standardization

Z-score standardization measures the difference between a variable and the mean with respect to the standard deviation.

$$z_i = \frac{x_i - \mu}{\sigma}$$

x – non-standardised variable

μ - average

σ - standard deviation

sklearn library - `StandardScaler`

In [29]:
```
from sklearn.preprocessing import StandardScaler

data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/employee.csv', sep=str(";")))

scaler = StandardScaler()
scaler.fit(data['performance_score'].values.reshape(-1,1))
data['performance_std_scaler'] = scaler.transform(data['performance_score'].values.reshape(-1,1))
data
```

Out[29]:

	name	age	income	gender	department	grade	performance_score	performance_std_scaler
0	Allen Smith	45.0	NaN	NaN	Operations	G3	723	0.505565
1	S Kumar	NaN	16000.0	F	Finance	G0	520	-0.408053
2	Jack Morgan	32.0	35000.0	M	Finance	G2	674	0.285037
3	Ying Chin	45.0	65000.0	F	Sales	G3	556	-0.246032
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711	0.451558
5	Satyam Sharma	NaN	62000.0	NaN	Sales	G3	649	0.172522
6	James Authur	54.0	NaN	F	Operations	G3	53	-2.509823
7	Josh Wills	54.0	52000.0	F	Finance	G3	901	1.306668
8	Leo Duck	23.0	98000.0	M	Sales	G4	709	0.442557

Min-max normalisation

Min-max normalization determines how much a given value is greater than the minimum and scales this difference by the range.

$$x_i' = \frac{x_i - min_x}{max_x - min_x}(range_{max} - range_{min}) + range_{min}$$

, when the range <0,1>

$$x_i' = \frac{x_i - min_x}{max_x - min_x}$$

sklearn library - `MinMaxScaler`

In [30]:
```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
scaler.fit(data['performance_score'].values.reshape(-1,1))
data['performance_minmax_scaler'] = scaler.transform(data['performance_score'].values.reshape(-1,1))
data.head()
```

Out[30]:

	name	age	income	gender	department	grade	performance_score	performance_std_scaler	performance_minmax_scaler
0	Allen Smith	45.0	NaN	NaN	Operations	G3	723	0.505565	0.790094
1	S Kumar	NaN	16000.0	F	Finance	G0	520	-0.408053	0.550708
2	Jack Morgan	32.0	35000.0	M	Finance	G2	674	0.285037	0.732311
3	Ying Chin	45.0	65000.0	F	Sales	G3	556	-0.246032	0.593160
4	Dheeraj Patel	30.0	42000.0	F	Operations	G2	711	0.451558	0.775943

Regression model

Regression model is one of the most popular algorithms in statistics and machine learning. In the field of machine learning and data science, regression analysis is a part of the supervised learning domain that helps predict continuous variables such as stock prices, house prices, sales, rainfall, temperature, etc.

Linear regression

Data preparation

In [31]:
```
data = pd.DataFrame(pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/advertising.csv'))
data.head()
```

Out[31]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [32]:
```
data.drop(['Unnamed: 0'], axis=1, inplace=True)
data.head()
```

Out[32]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

Exploratory data analysis

Basic data information

In [33]:
```
data.describe().T
```

Out[33]:

	count	mean	std	min	25%	50%	75%	max
TV	200.0	147.0425	85.854236	0.7	74.375	149.75	218.825	296.4
Radio	200.0	23.2640	14.846809	0.0	9.975	22.90	36.525	49.6
Newspaper	200.0	30.5540	21.778621	0.3	12.750	25.75	45.100	114.0
Sales	200.0	14.0225	5.217457	1.6	10.375	12.90	17.400	27.0

Checking for null values

In [34]:
```
data.isnull().sum()
```

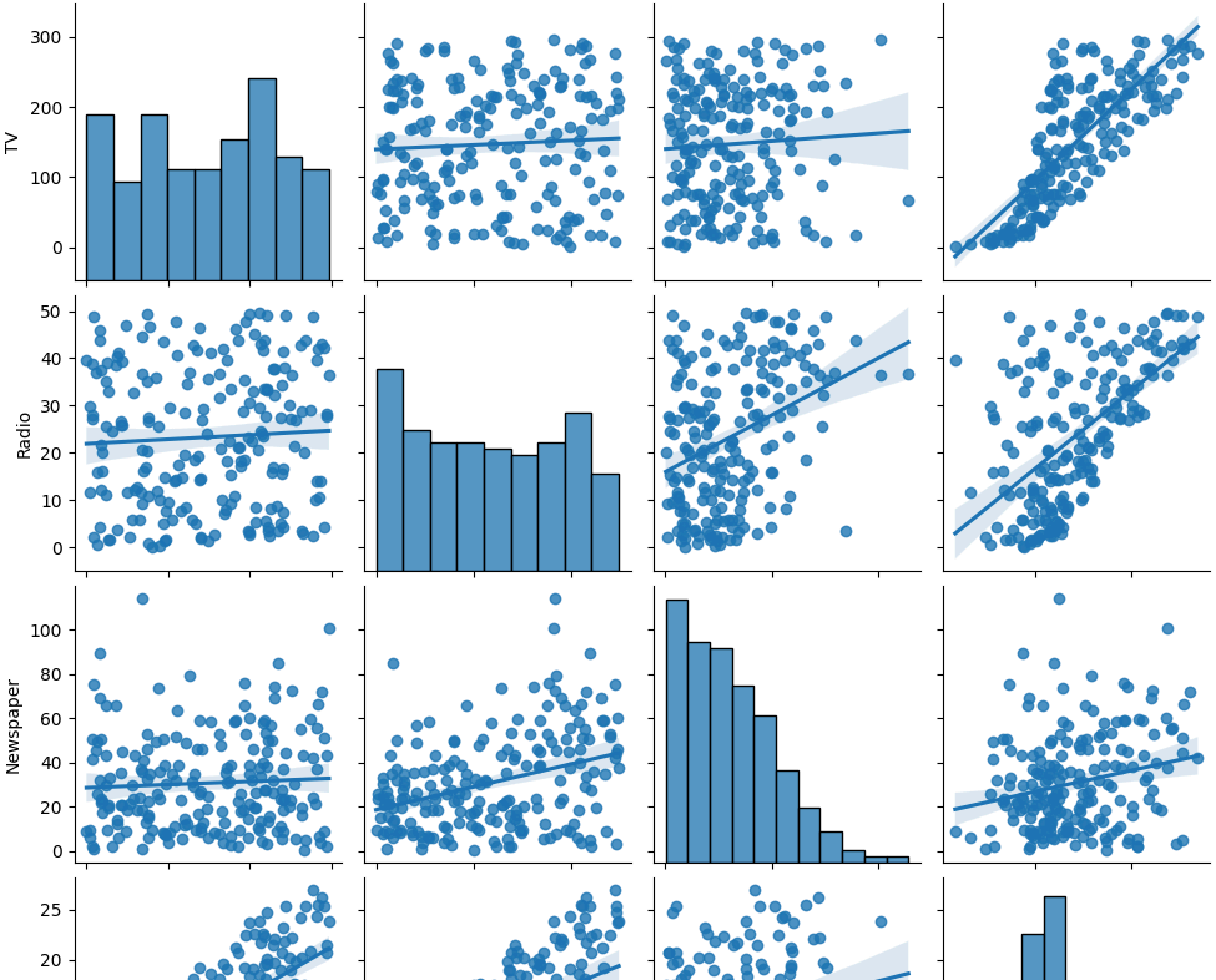
Out[34]:

	0
TV	0
Radio	0
Newspaper	0
Sales	0

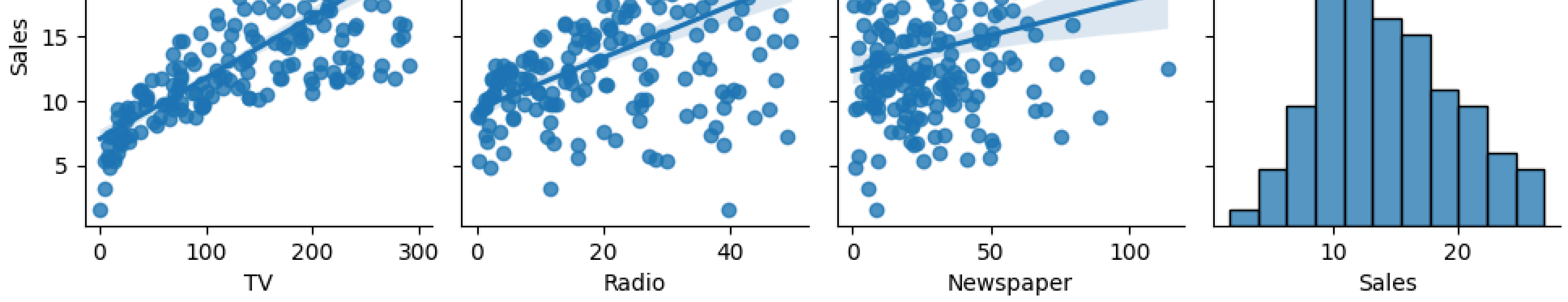
dtype: int64

Graph of dependencies between variables

```
In [35]: _ = sns.pairplot(data, kind='reg')
```

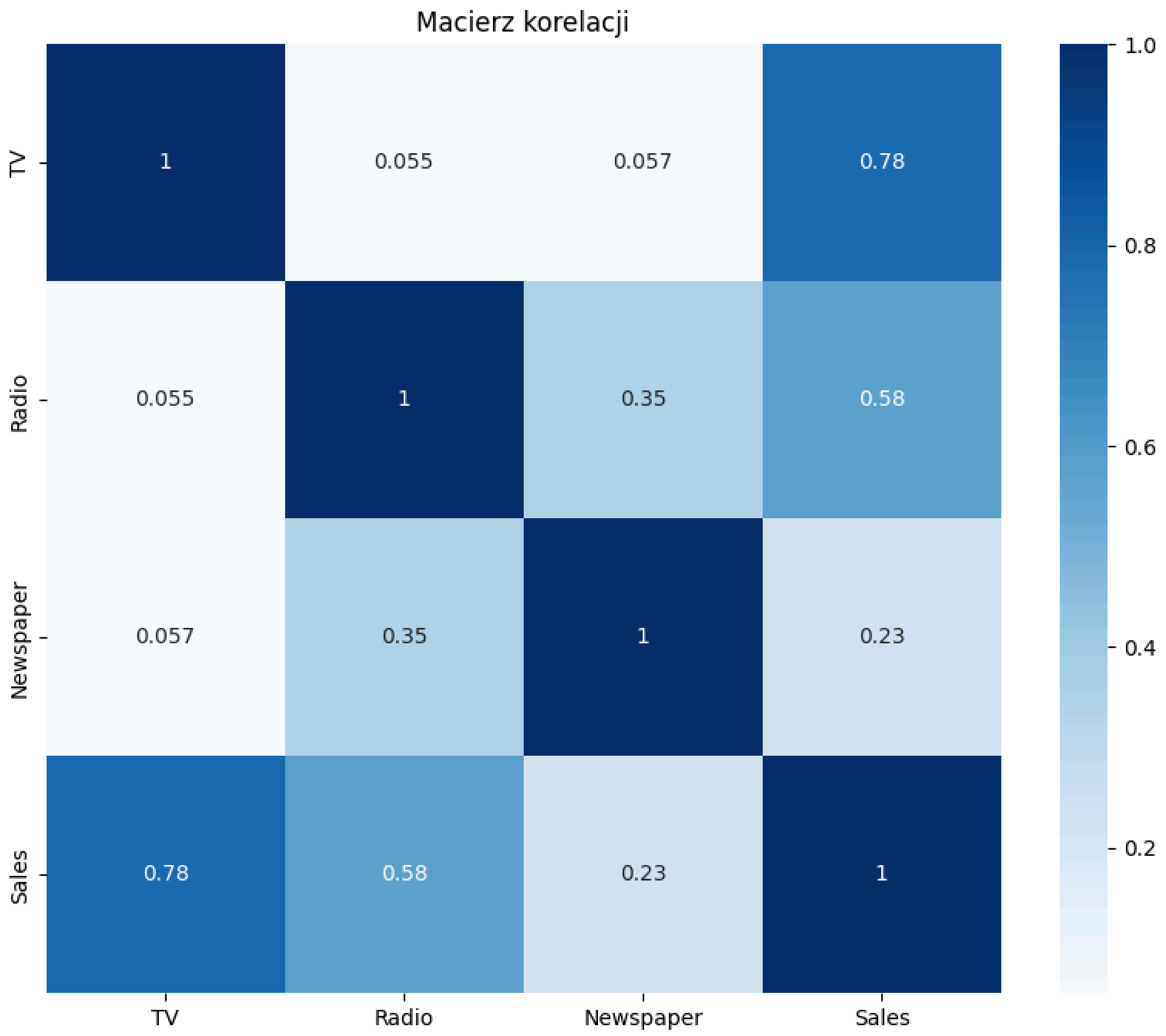


TV
Radio
Newspaper
300
200
100
0
50
40
30
20
10
0
100
80
60
40
20
0
25
20



Checking correlations between variables

```
In [36]: data_correlation = data.corr()
plt.figure(figsize = (10,8))
_ = sns.heatmap(data_correlation, annot=True, cmap='Blues').set(title='Macierz korelacji')
```



Building the model

Independent variables

```
In [37]: X = data[['TV', 'Radio', 'Newspaper']]
         X
```

Out[37]:

	TV	Radio	Newspaper
0	230.1	37.8	69.2
1	44.5	39.3	45.1
2	17.2	45.9	69.3
3	151.5	41.3	58.5
4	180.8	10.8	58.4
...	...	...	...
195	38.2	3.7	13.8
196	94.2	4.9	8.1
197	177.0	9.3	6.4
198	283.6	42.0	66.2
199	232.1	8.6	8.7

200 rows × 3 columns

Dependent variable (target)

```
In [38]: y = data.Sales
         y
```

```
Out[38]:
```

	Sales
0	22.1
1	10.4
2	9.3
3	18.5
4	12.9
...	...
195	7.6
196	9.7
197	12.8
198	25.5
199	13.4

200 rows × 1 columns

dtype: float64

Split into training and test data

```
In [39]: 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)
```

Linear regression model

```
In [40]: from sklearn.linear_model import LinearRegression

         lin_reg = LinearRegression()
         lin_reg.fit(X_train, y_train)
         print("Intercept:",lin_reg.intercept_)
         print("Coefficients:",lin_reg.coef_)
```

```
Intercept: 2.89257005115114833
Coefficients: [0.04416235 0.19900368 0.00116268]
```

Model prediction

```
In [41]: predictions = lin_reg.predict(X_test)
         print(predictions)
```

```
[10.0494569   7.43052335  6.97152143 24.16378667 12.00215643  6.54334645
 13.09526331 14.95879164 11.00528358 16.27234553 22.99324688  9.12188347
 10.33545333 15.39628185 11.60589932 12.11484332 18.60251172 10.81414474
 16.07541355 17.22753644 24.2342995   9.47711838 15.13960412 12.41064749
  5.67814427 15.22889947 12.21635459 20.94370559 13.28068231  9.16578351
 13.30285718 21.5770033  18.098111   21.15572322  6.69734039  6.15355714
  7.96280151 13.09426248 14.81032968  6.22020075 12.2799744   9.1817324
 15.04882696 16.26091437 17.16859664 13.32831849  3.69143664 12.43931798
 15.87909695  8.68626862]
```

Regression model evaluation

```
In [42]: import numpy as np
         from sklearn.metrics import mean_absolute_error
         from sklearn.metrics import mean_squared_error
         from sklearn.metrics import r2_score

         print(f"Mean absolute error (MAE): {mean_absolute_error(y_test,predictions)}")

         print(f"Mean square error (MSE): {mean_squared_error(y_test, predictions)}")

         print(f"Root mean square error (RMSE): {np.sqrt(mean_squared_error(y_test,predictions))}")

         print(f"R-square: {r2_score(y_test, predictions)}")
```

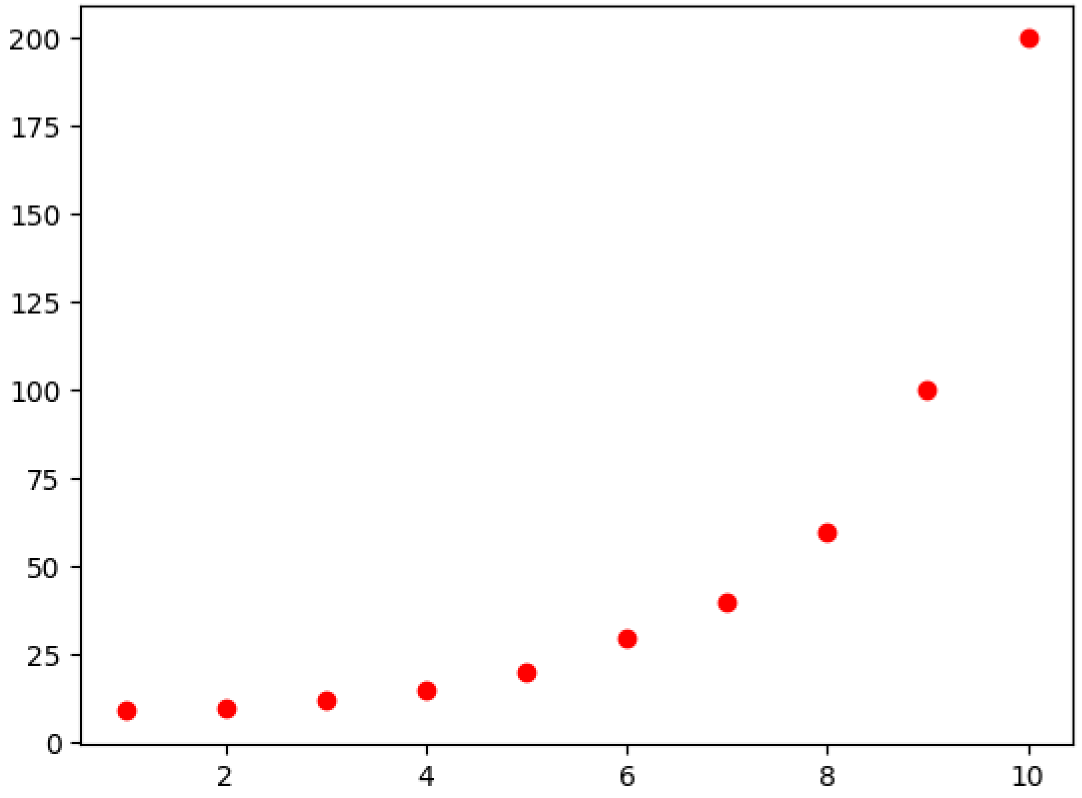
```
Mean absolute error (MAE): 1.3000320919235449
Mean square error (MSE): 4.012497522917099
Root mean square error (RMSE): 2.0031219440955406
R-square: 0.8576396745320893
```

Polynomial regression

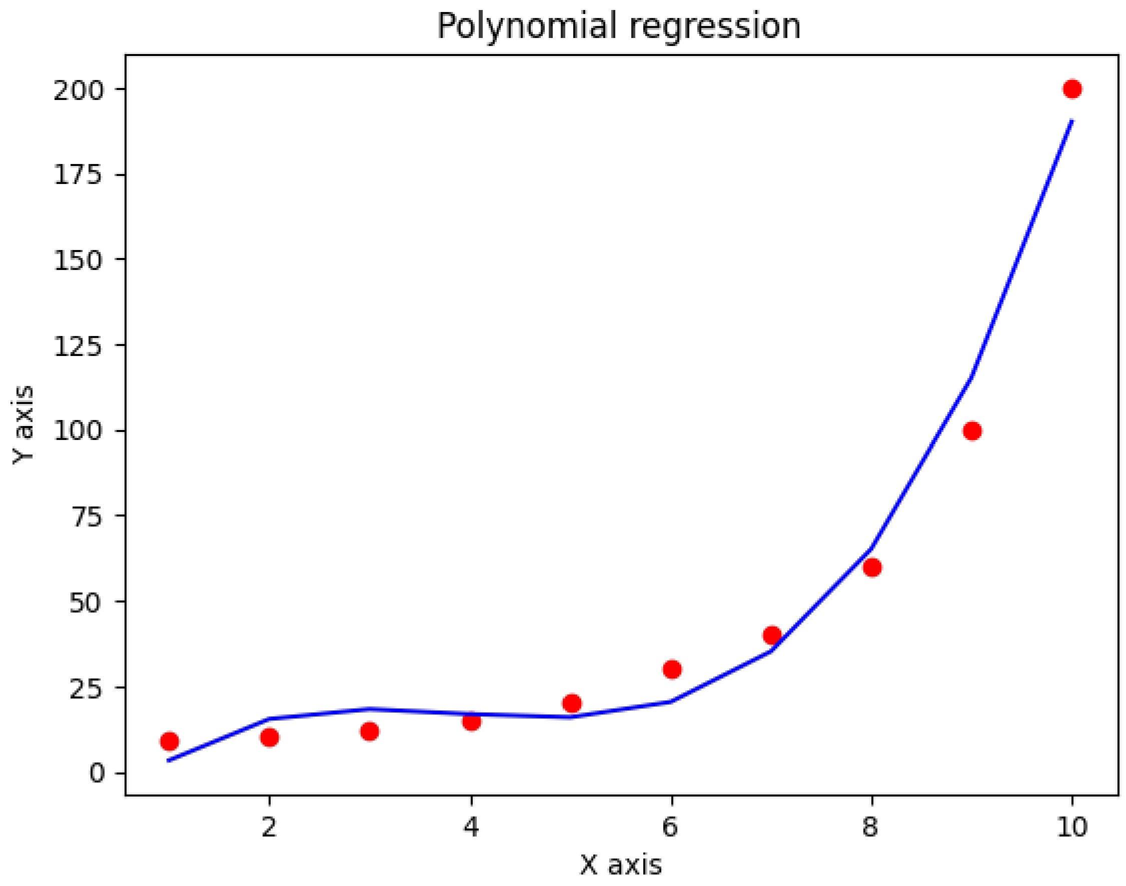
Polynomial regression is a type of regression analysis that is used to adapt nonlinear relationships between dependent and independent variables. In this type of regression, variables are modeled as an nth degree polynomial. It is used to understand the growth rate of various phenomena, such as epidemic outbreaks and sales growth.

```
In [43]: import pandas as pd
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression
```

```python
# Examples of data
data_pr = pd.DataFrame({
    "X":np.arange(1,11),
    "y":[9,10,12,15,20,30,40,60,100,200]
    })

X = data_pr[['X']]
y = data_pr[['y']]
_ = plt.scatter(X,y, color = 'red')
```



In [44]:
```python
# Application of the polynomial function
polynomial_function = PolynomialFeatures(degree = 3)
X_poly = polynomial_function.fit_transform(X)

# Application of the linear regression model
linear_regression = LinearRegression()
linear_regression.fit(X_poly, y)
prediction = linear_regression.predict(X_poly)

plt.title('Polynomial regression')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.scatter(X,y, color = 'red')
_  = plt.plot(X, prediction, color = 'blue')
```



Classification model

Classification is the most commonly used technique in machine learning and statistical learning. Most machine learning problems are classification problems, such as spam detection, financial risk analysis, customer churn analysis, and lead discovery.

Classification can be of two types:

- binary

> Binary classification target variables have only two values: 0 or 1 or 'yes' or 'no'. Examples of binary classification are whether or not a customer will buy an item, whether or not a customer will switch brands, spam detection, disease prediction and whether or not a loan applicant will be

> late in repayment, etc.

- multi-class

> A multi-class classification has more than two classes, for example, for a news article category, the classes can be sports, politics, business and many others.

Classification model building process

The process of building a classification model consists of three steps:

1. splitting the data into training and test data,
2. model generation
3. model evaluation.

In the splitting stage, the source dataset is split into a training dataset and a test dataset.

The training dataset is used to generate the model.

The test dataset is used in the model performance evaluation phase.

Data preparation

In [45]:
```python
# Diabetes dataset

import pandas as pd

data_diabetes = pd.read_csv('http://bartoszj.prz-rzeszow.pl/dane/diabetes.csv')
data_diabetes.head()
```

Out[45]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [46]:
```
data_diabetes.columns = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
data_diabetes.head()
```

Out[46]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Basic data information

In [47]:
```
data_diabetes.describe().T
```

Out[47]:

	count	mean	std	min	25%	50%	75%	max
pregnant	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
bp	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
skin	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
bmi	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
pedigree	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
label	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [48]:
```
data_diabetes.isnull().sum()
```

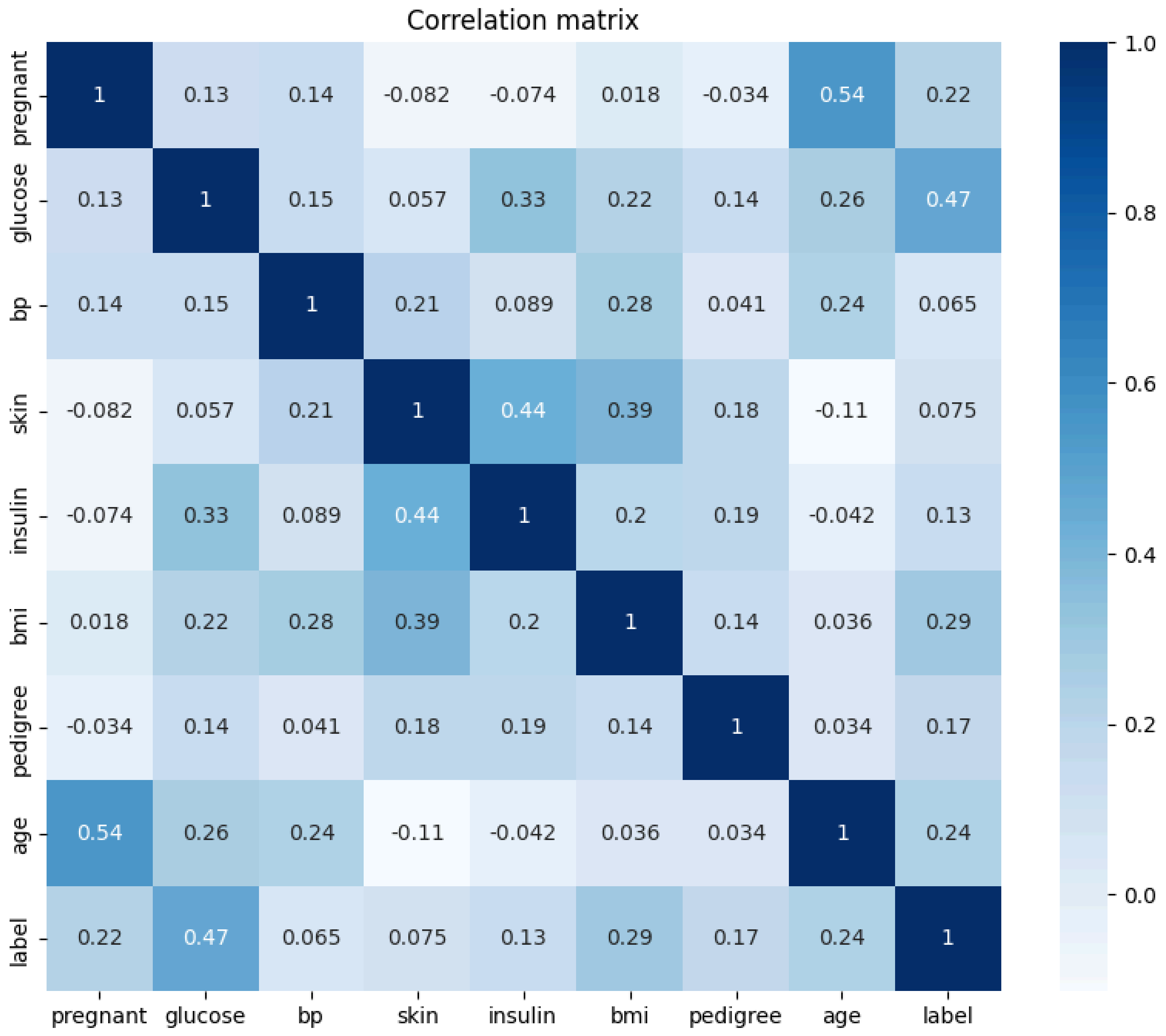
Out[48]:

	0
pregnant	0
glucose	0
bp	0
skin	0
insulin	0
bmi	0
pedigree	0
age	0
label	0

dtype: int64

In [49]:
```
data_correlation = data_diabetes.corr()
plt.figure(figsize = (10,8))
```

```python
_ = sns.heatmap(data_correlation, annot=True, cmap='Blues').set(title='Correlation matrix')
```



	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
pregnant	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
bp	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
skin	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
bmi	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
pedigree	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
label	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

Splitting the data into training and test data

Simple splitting

In [50]:
```
from sklearn.model_selection import train_test_split

X = data_diabetes.iloc[:,:-1]
X
```

Out[50]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

In [51]:
```
y = data_diabetes.label
y
```

Out[51]:

	label
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

768 rows × 1 columns

dtype: int64

The `train_test_split` function from the `sklearn.model_selection` library is used to split the test and training data. The `test_size` parameter determines how much of the data will be test data (usually takes the value 0.2-0.3).

In [52]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# Number of values of the dependent variable in the training set
print(y_train[y_train == 0].count())
print(y_train[y_train == 1].count())
```

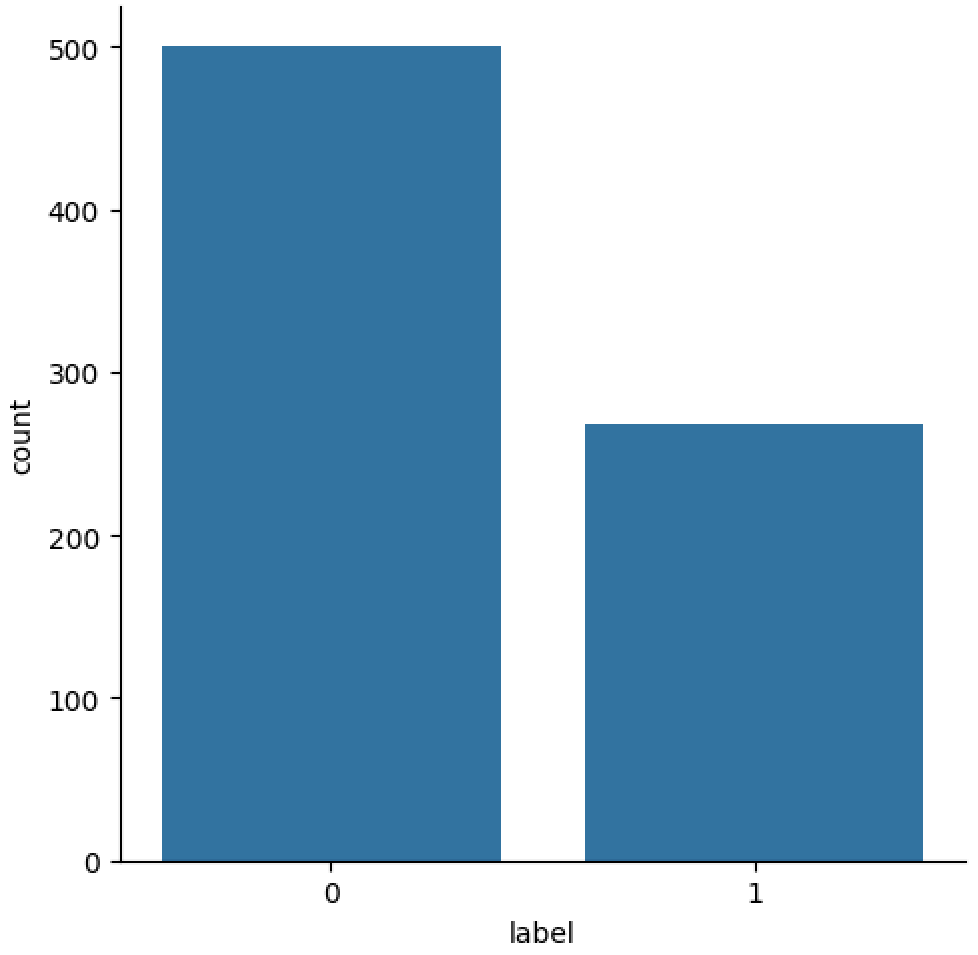
```
354
183
```

Balanced split

Where there is a large discrepancy between the number of values for each label of the dependent variable, a balanced split of the data can be used to ensure a proportionate number of records in the training and test sets.

Checking the distribution of the dependent variable

```
In [53]: _ = sns.catplot(data=data_diabetes, x='label', kind='count')
```



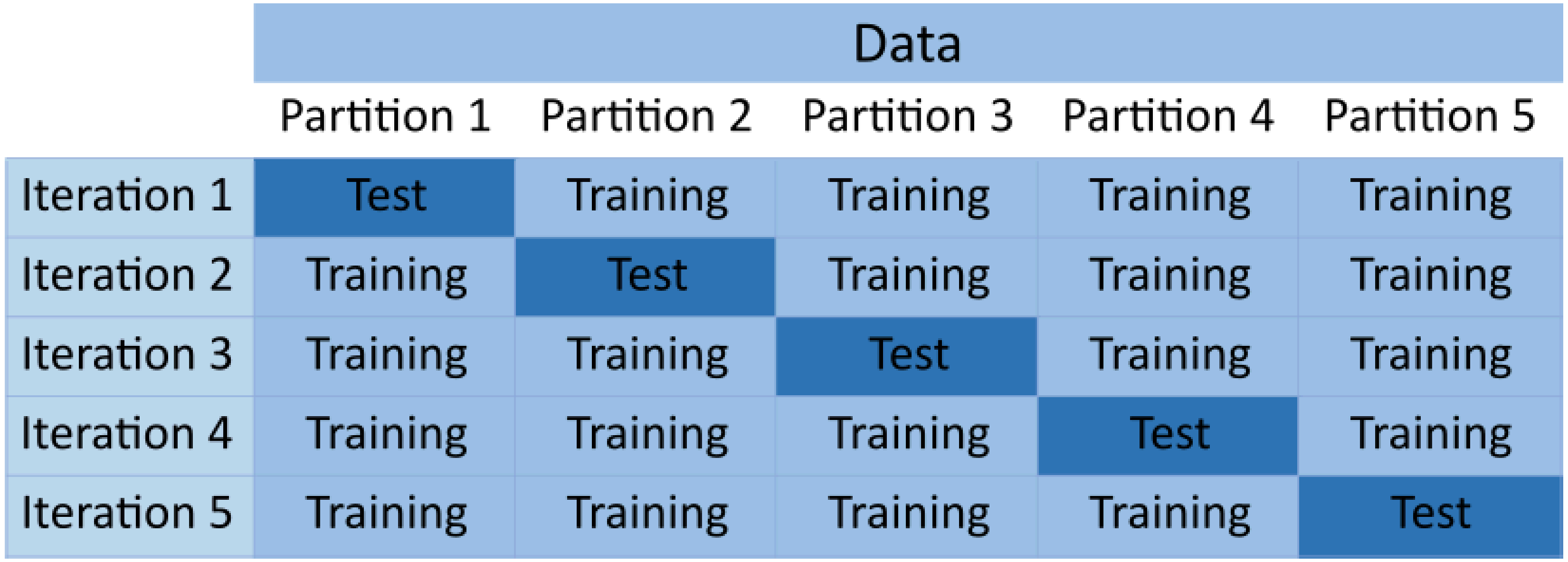
```
In [54]: X_train_bal, X_test_bal, y_train_bal, y_test_bal = train_test_split(X, y, test_size=0.3, random_state=1, stratify=data_diabetes.label)
         # Number of values of the dependent variable in the training set
         print(y_train_bal[y_train_bal == 0].count())
         print(y_train_bal[y_train_bal == 1].count())
```

```
350
187
```

K-fold cross-validation

In this approach, the data is divided into k partitions of approximately equal size. Then k models are trained, where in each iteration one partition is dedicated to testing and the remaining k-1 partitions are jointly used for the training purpose.

Example of 5-fold validation



	Data				
	Partition 1	Partition 2	Partition 3	Partition 4	Partition 5
Iteration 1	Test	Training	Training	Training	Training
Iteration 2	Training	Test	Training	Training	Training
Iteration 3	Training	Training	Test	Training	Training
Iteration 4	Training	Training	Training	Test	Training
Iteration 5	Training	Training	Training	Training	Test

Function `KFold`

In [55]:
```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5)
#  kf = KFold(n_splits=5, shuffle=True, random_state=100)

kf_generator = kf.split(X)

X_train_list = []
X_test_list = []
y_train_list = []
y_test_list = []
for train_index, test_index in kf_generator:
    X_train_list.append(X.iloc[train_index])
    X_test_list.append(X.iloc[test_index])
    y_train_list.append(y.iloc[train_index])
    y_test_list.append(y.iloc[test_index])

X_train_list
```

```
Out[55]: [     pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          154         8      188  78     0        0  47.9     0.137   43
          155         7      152  88    44        0  50.0     0.337   36
          156         2       99  52    15       94  24.6     0.637   21
          157         1      109  56    21      135  25.2     0.833   23
          158         2       88  74    19       53  29.0     0.229   22
          ..        ...      ...  ..   ...      ...   ...       ...  ...
          763        10      101  76    48      180  32.9     0.171   63
          764         2      122  70    27        0  36.8     0.340   27
          765         5      121  72    23      112  26.2     0.245   30
          766         1      126  60     0        0  30.1     0.349   47
          767         1       93  70    31        0  30.4     0.315   23

          [614 rows x 8 columns],
               pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          0           6      148  72    35        0  33.6     0.627   50
          1           1       85  66    29        0  26.6     0.351   31
          2           8      183  64     0        0  23.3     0.672   32
          3           1       89  66    23       94  28.1     0.167   21
          4           0      137  40    35      168  43.1     2.288   33
          ..        ...      ...  ..   ...      ...   ...       ...  ...
          763        10      101  76    48      180  32.9     0.171   63
          764         2      122  70    27        0  36.8     0.340   27
          765         5      121  72    23      112  26.2     0.245   30
          766         1      126  60     0        0  30.1     0.349   47
          767         1       93  70    31        0  30.4     0.315   23

          [614 rows x 8 columns],
               pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          0           6      148  72    35        0  33.6     0.627   50
          1           1       85  66    29        0  26.6     0.351   31
          2           8      183  64     0        0  23.3     0.672   32
          3           1       89  66    23       94  28.1     0.167   21
          4           0      137  40    35      168  43.1     2.288   33
          ..        ...      ...  ..   ...      ...   ...       ...  ...
          763        10      101  76    48      180  32.9     0.171   63
          764         2      122  70    27        0  36.8     0.340   27
          765         5      121  72    23      112  26.2     0.245   30
          766         1      126  60     0        0  30.1     0.349   47
          767         1       93  70    31        0  30.4     0.315   23

          [614 rows x 8 columns],
               pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          0           6      148  72    35        0  33.6     0.627   50
          1           1       85  66    29        0  26.6     0.351   31
          2           8      183  64     0        0  23.3     0.672   32
3           1       89  66    23       94  28.1     0.167   21
4           0      137  40    35      168  43.1     2.288   33
..        ...      ...  ..   ...      ...   ...       ...  ...
763        10      101  76    48      180  32.9     0.171   63
764         2      122  70    27        0  36.8     0.340   27
765         5      121  72    23      112  26.2     0.245   30
766         1      126  60     0        0  30.1     0.349   47
767         1       93  70    31        0  30.4     0.315   23

[615 rows x 8 columns],
     pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
0           6      148  72    35        0  33.6     0.627   50
1           1       85  66    29        0  26.6     0.351   31
2           8      183  64     0        0  23.3     0.672   32
3           1       89  66    23       94  28.1     0.167   21
4           0      137  40    35      168  43.1     2.288   33
..        ...      ...  ..   ...      ...   ...       ...  ...
610         3      106  54    21      158  30.9     0.292   24
611         3      174  58    22      194  32.9     0.593   36
612         7      168  88    42      321  38.2     0.787   40
613         6      105  80    28        0  32.5     0.878   26
614        11      138  74    26      144  36.1     0.557   50

[615 rows x 8 columns]]
```

Stratified K-fold cross-validation

Function `StratifiedKFold` - with checking the distribution of the dependent variable

In [56]:
```python
from sklearn.model_selection import StratifiedKFold

kf = StratifiedKFold(n_splits=5)
#  kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)

kf_generator = kf.split(X, y)

X_train_list_balanced = []
X_test_list_balanced = []
y_train_list_balanced = []
y_test_list_balanced = []
for train_index, test_index in kf_generator:
    X_train_list_balanced.append(X.iloc[train_index])
    X_test_list_balanced.append(X.iloc[test_index])
    y_train_list_balanced.append(y.iloc[train_index])
    y_test_list_balanced.append(y.iloc[test_index])
```

```
X_train_list_balanced
```

```
Out[56]: [     pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          154         8      188  78     0        0  47.9     0.137   43
          155         7      152  88    44        0  50.0     0.337   36
          156         2       99  52    15       94  24.6     0.637   21
          157         1      109  56    21      135  25.2     0.833   23
          158         2       88  74    19       53  29.0     0.229   22
          ..        ...      ...  ..   ...      ...   ...       ...  ...
          763        10      101  76    48      180  32.9     0.171   63
          764         2      122  70    27        0  36.8     0.340   27
          765         5      121  72    23      112  26.2     0.245   30
          766         1      126  60     0        0  30.1     0.349   47
          767         1       93  70    31        0  30.4     0.315   23

          [614 rows x 8 columns],
               pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          0           6      148  72    35        0  33.6     0.627   50
          1           1       85  66    29        0  26.6     0.351   31
          2           8      183  64     0        0  23.3     0.672   32
          3           1       89  66    23       94  28.1     0.167   21
          4           0      137  40    35      168  43.1     2.288   33
          ..        ...      ...  ..   ...      ...   ...       ...  ...
          763        10      101  76    48      180  32.9     0.171   63
          764         2      122  70    27        0  36.8     0.340   27
          765         5      121  72    23      112  26.2     0.245   30
          766         1      126  60     0        0  30.1     0.349   47
          767         1       93  70    31        0  30.4     0.315   23

          [614 rows x 8 columns],
               pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          0           6      148  72    35        0  33.6     0.627   50
          1           1       85  66    29        0  26.6     0.351   31
          2           8      183  64     0        0  23.3     0.672   32
          3           1       89  66    23       94  28.1     0.167   21
          4           0      137  40    35      168  43.1     2.288   33
          ..        ...      ...  ..   ...      ...   ...       ...  ...
          763        10      101  76    48      180  32.9     0.171   63
          764         2      122  70    27        0  36.8     0.340   27
          765         5      121  72    23      112  26.2     0.245   30
          766         1      126  60     0        0  30.1     0.349   47
          767         1       93  70    31        0  30.4     0.315   23

          [614 rows x 8 columns],
               pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
          0           6      148  72    35        0  33.6     0.627   50
          1           1       85  66    29        0  26.6     0.351   31
          2           8      183  64     0        0  23.3     0.672   32
3            1      89  66    23       94  28.1      0.167   21
4            0     137  40    35      168  43.1      2.288   33
..         ...     ...  ..   ...      ...   ...        ...  ...
763         10     101  76    48      180  32.9      0.171   63
764          2     122  70    27        0  36.8      0.340   27
765          5     121  72    23      112  26.2      0.245   30
766          1     126  60     0        0  30.1      0.349   47
767          1      93  70    31        0  30.4      0.315   23

[615 rows x 8 columns],
     pregnant  glucose  bp  skin  insulin   bmi  pedigree  age
0           6      148  72    35        0  33.6     0.627   50
1           1       85  66    29        0  26.6     0.351   31
2           8      183  64     0        0  23.3     0.672   32
3           1       89  66    23       94  28.1     0.167   21
4           0      137  40    35      168  43.1     2.288   33
..        ...      ...  ..   ...      ...   ...       ...  ...
610         3      106  54    21      158  30.9     0.292   24
611         3      174  58    22      194  32.9     0.593   36
612         7      168  88    42      321  38.2     0.787   40
614        11      138  74    26      144  36.1     0.557   50
618         9      112  82    24        0  28.2     1.282   50

[615 rows x 8 columns]]
```

Model building process

The process of building a model using the `scikit-learn` library consists of the following steps:

- importing the class of the selected classification algorithm,
- creation of an object of a given class with parameters,
- start the process of learning the model with the `fit()` function for training data,
- Making a prediction for the test data with the `predict()` function.

Types of classifiers

Among the many classification methods available in the `sklearn` library are:

- Logistic regression

  `from sklearn.linear_model import LogisticRegression`

- Naive Bayes classifier

`from sklearn.naive_bayes import GaussianNB`

- Decision tree

  `from sklearn.tree import DecisionTreeClassifier`

- K nearest neighbors (KNN)

  `from sklearn.neighbors import KNeighborsClassifier`

- Support vector machine (SVM)

  `from sklearn import svm`

Logistic regression

Logistic regression is a type of supervised machine learning algorithm that is used to predict a binary outcome and classify observations. Its dependent variable is a binary variable with two classes: 0 or 1.

In [57]:
```
from sklearn.linear_model import LogisticRegression

logreg_model = LogisticRegression(solver='lbfgs',max_iter=150)
logreg_model.fit(X_train, y_train)
logreg_prediction = logreg_model.predict(X_test)
logreg_prediction
```

Out[57]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0])
```

Naive Bayes classifier

Naive Bayes is a classification method based on Bayes' theorem that assumes conditional class independence. Conditional class independence means that each column of features is independent of the others.

Naive Bayes has been used effectively in text mining applications such as document classification, predicting sentiment in customer reviews, and spam filtering.

In [58]:
```python
from sklearn.naive_bayes import GaussianNB

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_prediction = nb_model.predict(X_test)
nb_prediction
```

Out[58]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0])
```

Decision tree

Decision tree is one of the most well-known classification techniques. It can be used for both types of supervised learning problems (classification and regression). When generating a model, a tree structure is created that resembles a flow chart and mimics human thinking, making it easier to understand and interpret. It also allows you to see the logic behind the prediction, unlike black-box algorithms such as SVM or neural networks.

In [59]:
```python
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_prediction = dt_model.predict(X_test)
dt_prediction
```

```
Out[59]: array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
                1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
                0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
                1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
                1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0])
```

K nearest neighbors (KNN)

The **K nearest neighbor method** (KNN) is a simple, easy to understand and implement classification algorithm that can also be used for regression problems.

Among the main applications of the KNN method are recommendation systems, e.g. suggesting videos or products on websites.

```
In [60]: from sklearn.neighbors import KNeighborsClassifier

         knn_model = KNeighborsClassifier(n_neighbors=3)
         knn_model.fit(X_train, y_train)
         knn_prediction = knn_model.predict(X_test)
         knn_prediction
```

```
Out[60]: array([1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0,
                1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
                0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
                1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
                1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0])
```

Support vector machine (SVM)

The **Support vector machine** (SVM) is quite popular due to its accuracy with low computational power. It is applicable to both classification and regression problems. Thanks to the use of a kernel parameter, it can also be used to model nonlinear relationships. SVM has many use cases, such as intrusion detection, text classification, face detection, and handwriting recognition.

In [61]:
```
from sklearn import svm

svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train, y_train)
svm_prediction = svm_model.predict(X_test)
svm_prediction
```

Out[61]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0])
```

Model evaluation

Among the many methods and tools for evaluating a model (classifier), the following are the most commonly used:

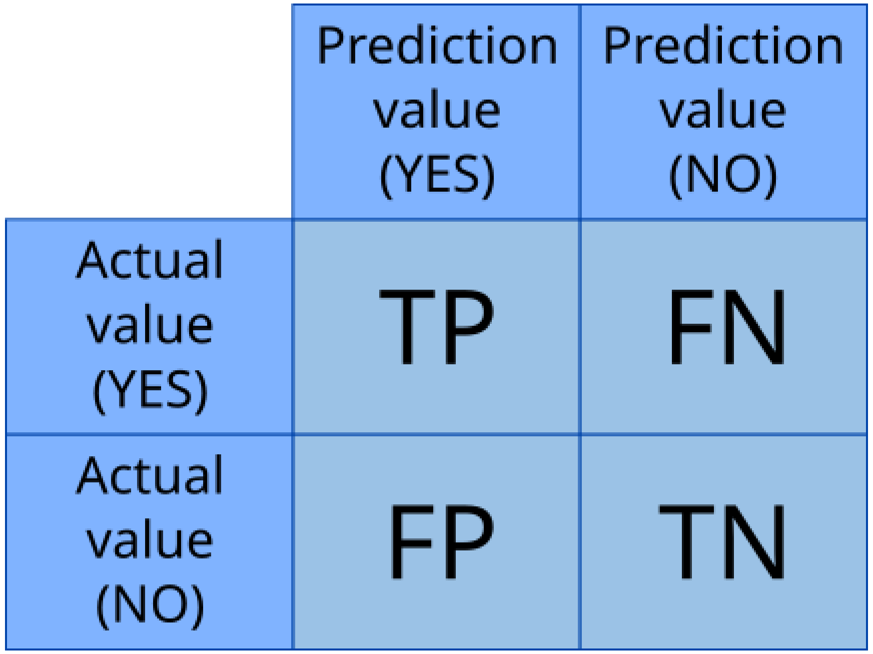
- Confusion matrix
- Accuracy
- Precision
- Recall
- F1-score
- ROC curve and AUC

Confusion matrix

Confusion matrix is an approach that provides a summary of prediction results for binary and multiclass classification problems. The concept of a confusion matrix is to find the number of correct and incorrect predictions, which are then summarized and distributed across classes.

The following concepts are defined:

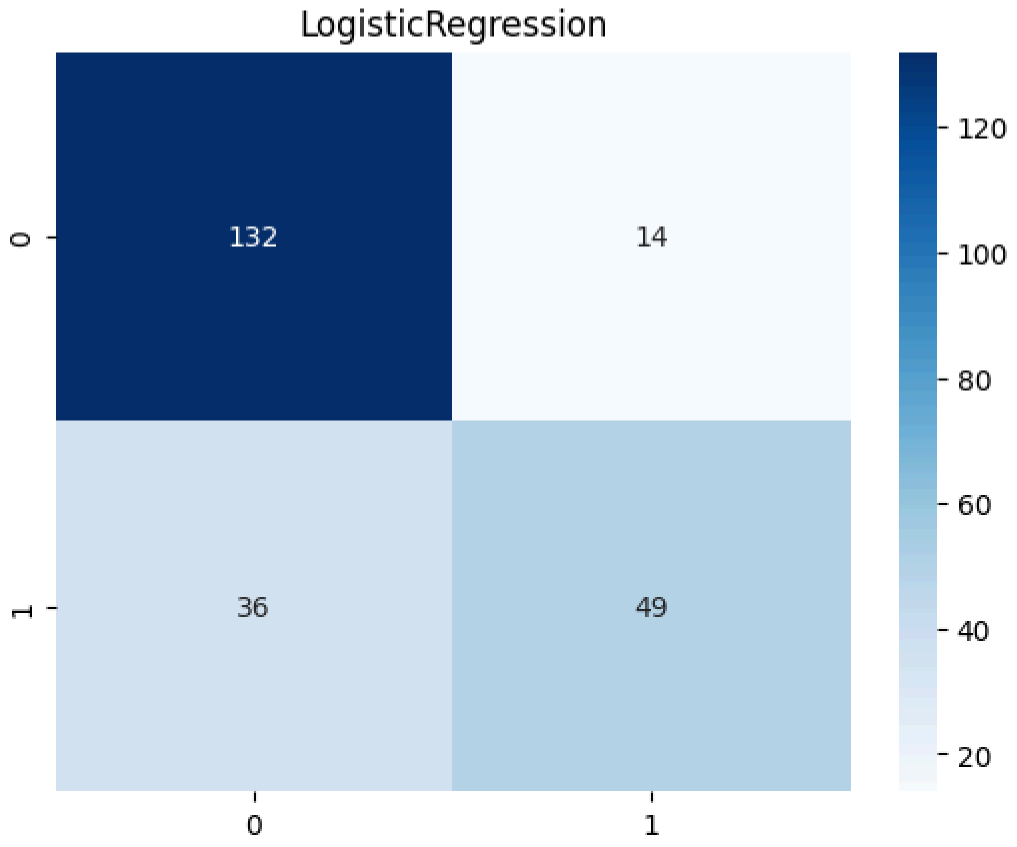
- **True-Positive** (**TP**): prediction value positive, actual value positive
- **True-Negative** (**TN**): prediction value negative, actual value negative
- **False-Positive** (**FP**): prediction value positive, actual value negative
- **False-Negative** (**FN**): prediction value negative, actual value positive

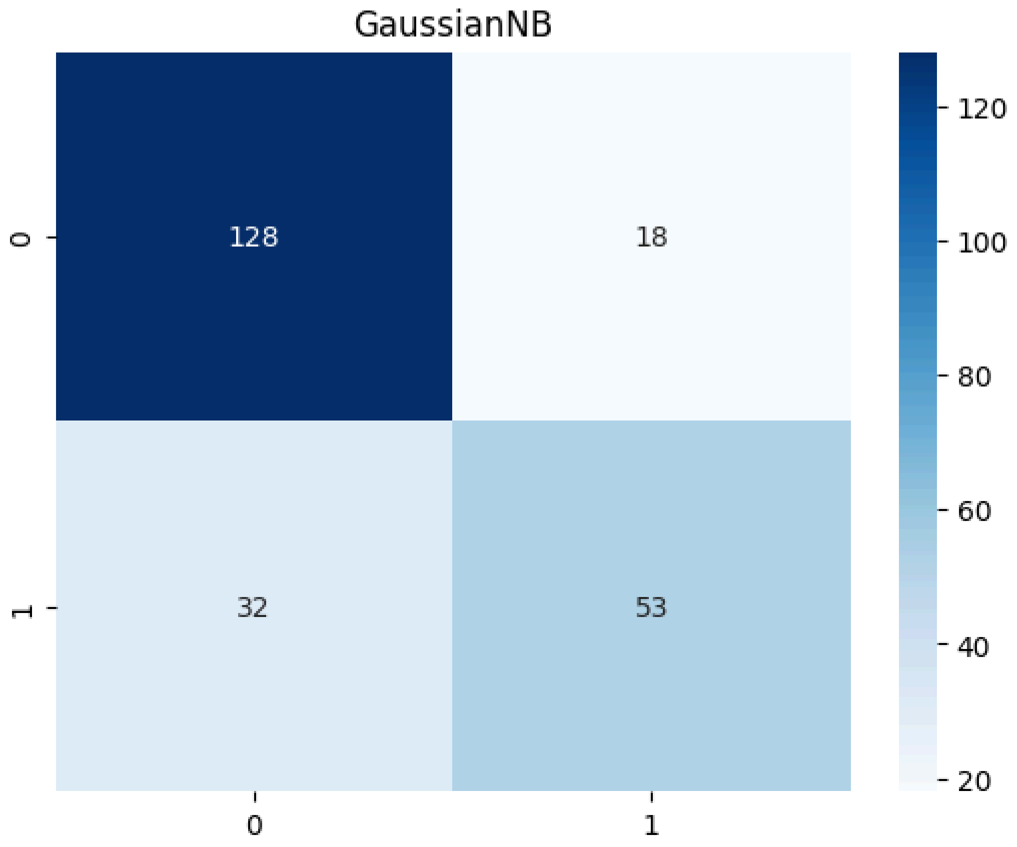


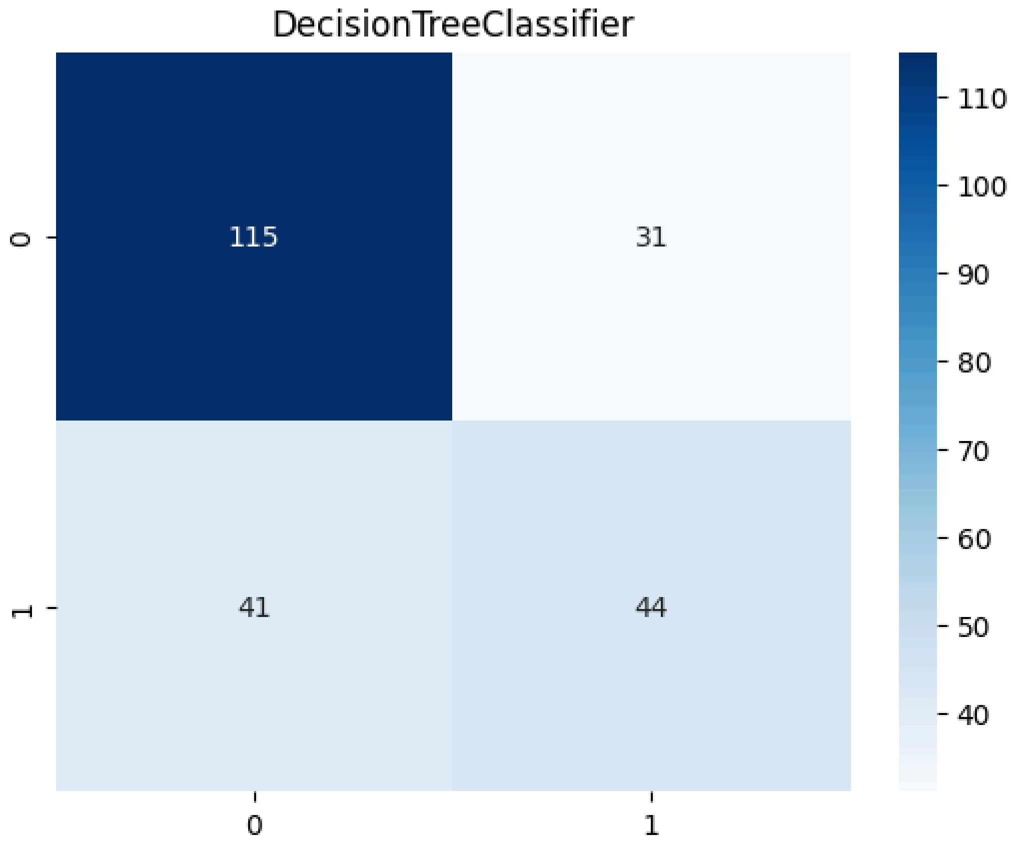
	Prediction value (YES)	Prediction value (NO)
Actual value (YES)	TP	FN
Actual value (NO)	FP	TN

In [62]:
```
models_names = ['LogisticRegression', 'GaussianNB', 'DecisionTreeClassifier', 'KNeighborsClassifier', 'SVM']
predictions = [logreg_prediction, nb_prediction, dt_prediction, knn_prediction, svm_prediction]
```

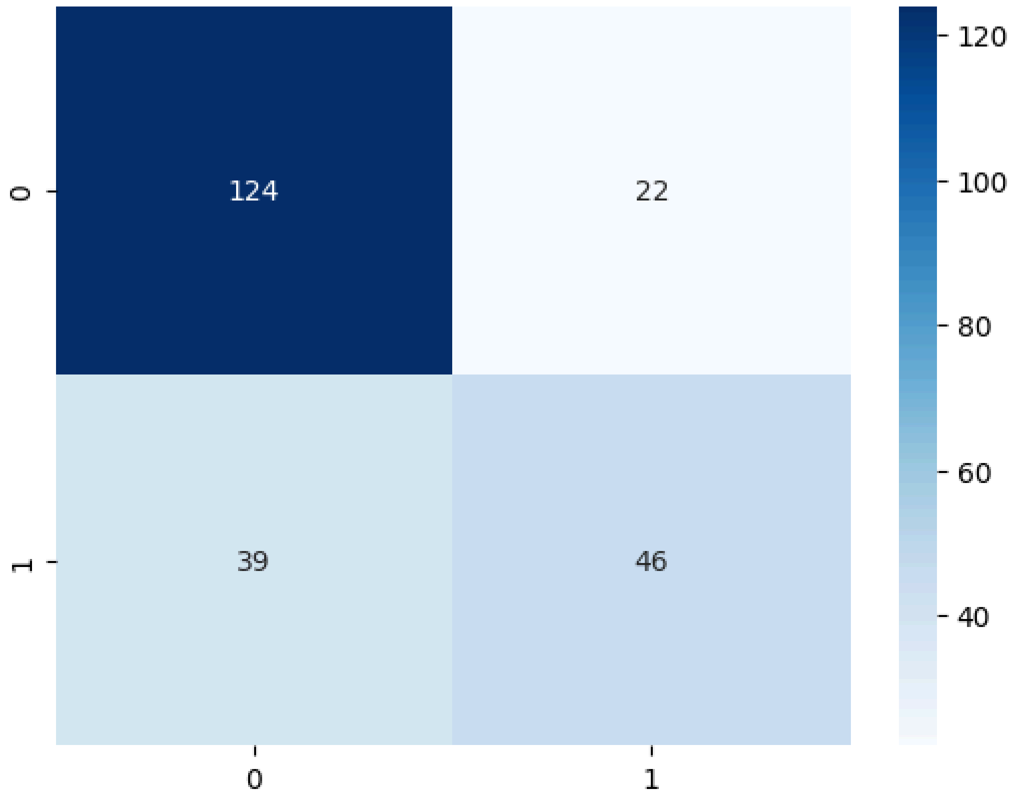
In [63]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

for prediction, name in zip(predictions, models_names):
  plt.figure()
  cm = confusion_matrix(y_test, prediction)
  _ = sns.heatmap(cm, annot=True, cmap='Blues', fmt='g').set(title=name)
```

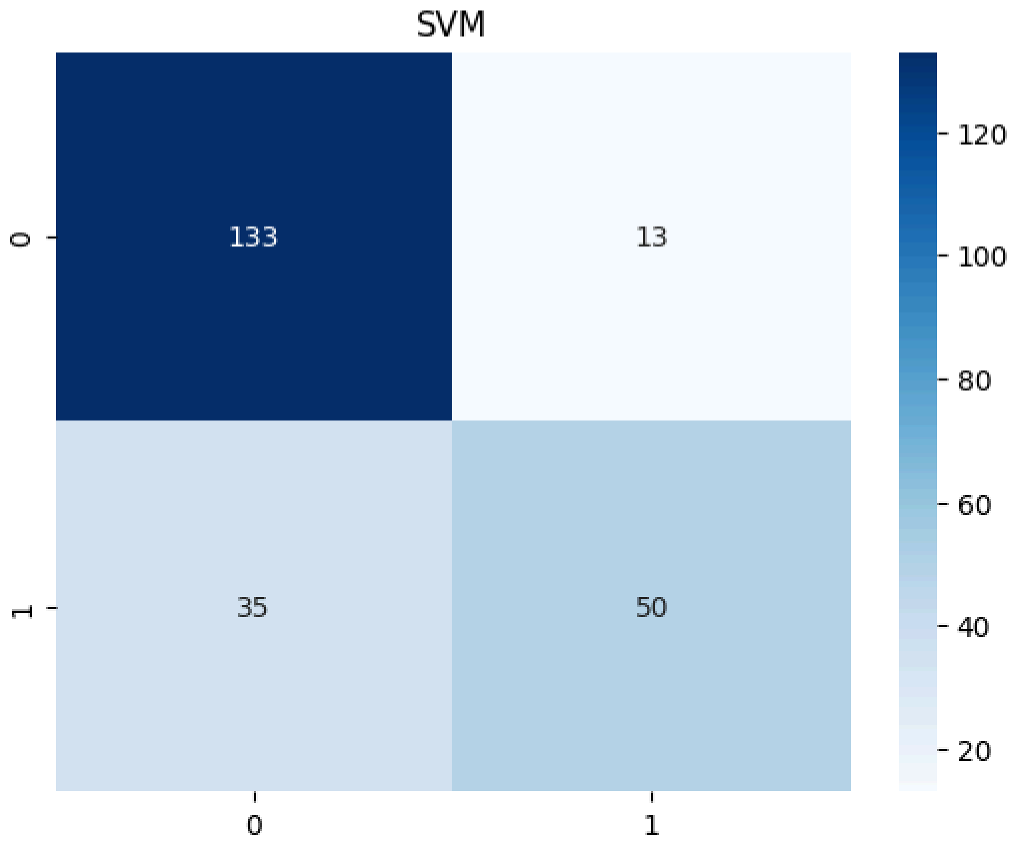

LogisticRegression
132
14
36
49
0
1
0
1
120
100
80
60
40
20


GaussianNB
128
18
32
53
0
1
0
1
120
100
80
60
40
20


DecisionTreeClassifier
0
1
115
31
41
44
0
1
110
100
90
80
70
60
50
40

KNeighborsClassifier





Accuracy

Accuracy indicates how many values from the test set were correctly classified relative to the number of all cases.

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

In [64]:
```
from sklearn.metrics import accuracy_score

for prediction, name in zip(predictions, models_names):
  print(name)
  print(f"{accuracy_score(prediction, y_test):.6f}")
```

```
LogisticRegression
0.783550
GaussianNB
0.783550
DecisionTreeClassifier
0.688312
KNeighborsClassifier
0.735931
SVM
0.792208
```

Precision

Precision determines what part of the results indicated by the classifier as positive is actually positive.

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

In [65]:
```python
from sklearn.metrics import precision_score

for prediction, name in zip(predictions, models_names):
  print(name)
  print(f"{precision_score(prediction, y_test):.6f}")
```

```
LogisticRegression
0.576471
GaussianNB
0.623529
DecisionTreeClassifier
0.517647
KNeighborsClassifier
0.541176
SVM
0.588235
```

Recall

Recall determines what fraction of positives the classifier has detected.

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

In [66]:
```python
from sklearn.metrics import recall_score

for prediction, name in zip(predictions, models_names):
  print(name)
  print(f"{recall_score(prediction, y_test):.6f}")
```

```
LogisticRegression
0.777778
GaussianNB
0.746479
DecisionTreeClassifier
0.586667
KNeighborsClassifier
0.676471
SVM
0.793651
```

F1-score

F1-score is the harmonic mean between precision (Precision) and sensitivity (Recall).

$$F1\text{-}score = \frac{2TP}{2TP + FP + FN} = \frac{2 Precision Recall}{Precision + Recall}$$

In [67]:
```
from sklearn.metrics import f1_score

for prediction, name in zip(predictions, models_names):
  print(name)
  print(f"{f1_score(prediction, y_test):.6f}")
```

```
LogisticRegression
0.662162
GaussianNB
0.679487
DecisionTreeClassifier
0.550000
KNeighborsClassifier
0.601307
SVM
0.675676
```

ROC curve and AUC

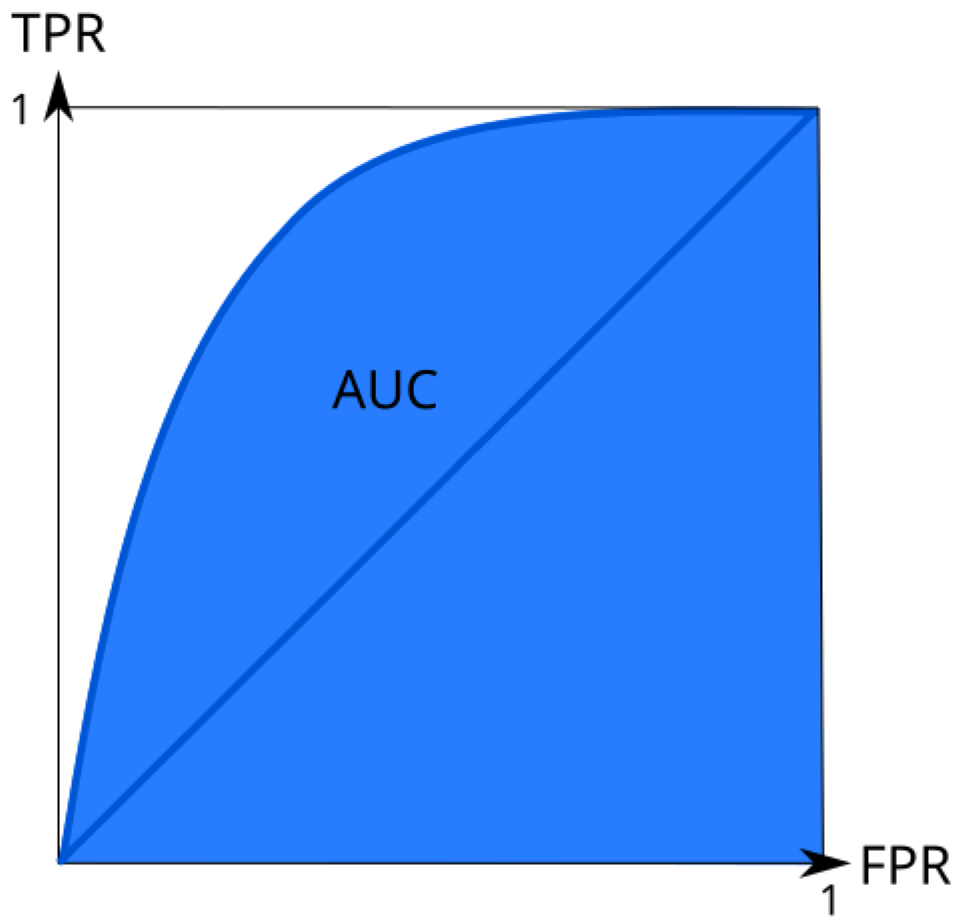
The AUC-ROC curve is a tool for measuring and evaluating the performance of classification models. ROC (Receiver Operating Characteristics) is one way to visualize the quality of classification, showing the relationship of TPR (True Positive Rate) and FPR (False Positive Rate) indicators.

TPR (True Positive Rate)

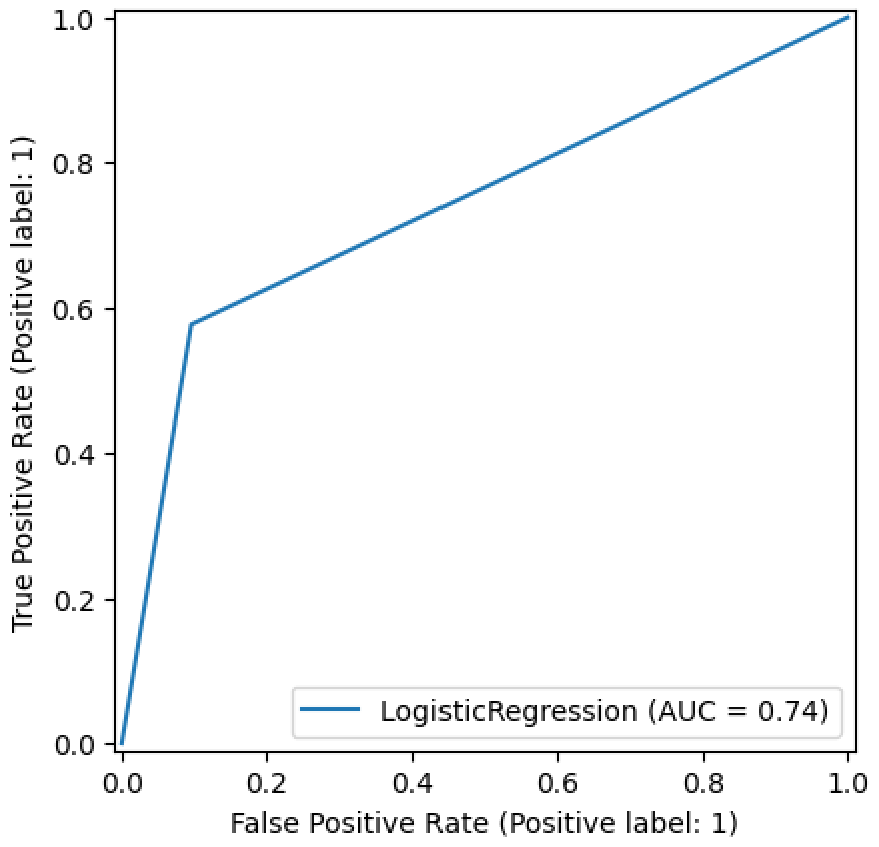
$$TPR = Recall = \frac{TP}{TP + FN}$$

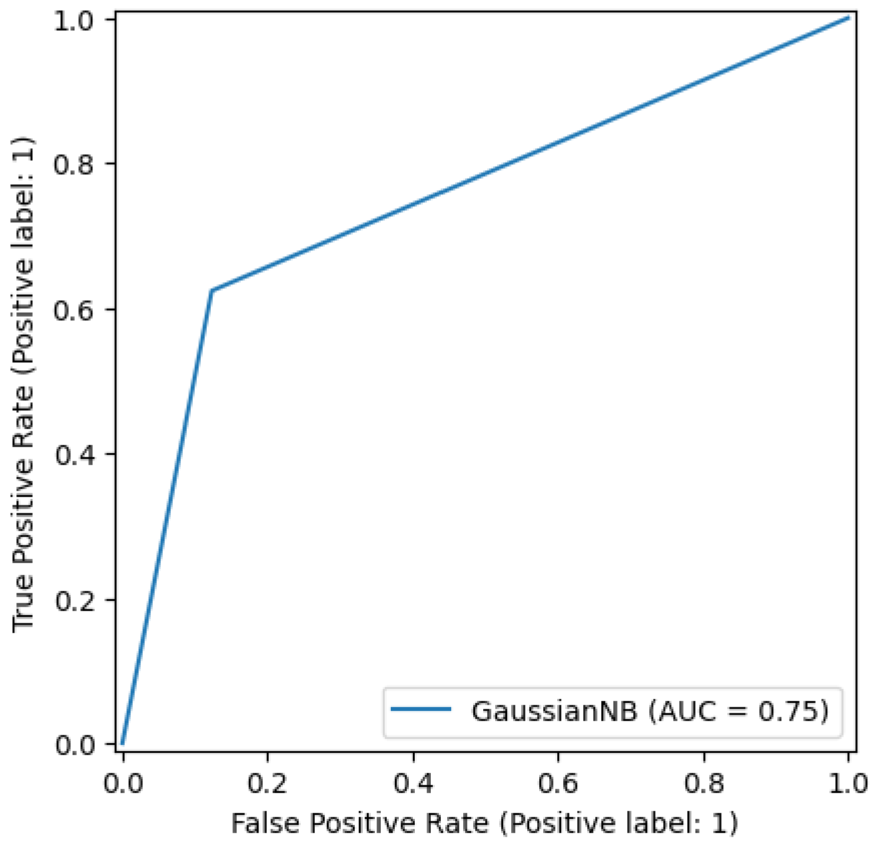
FPR (False Positive Rate)

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

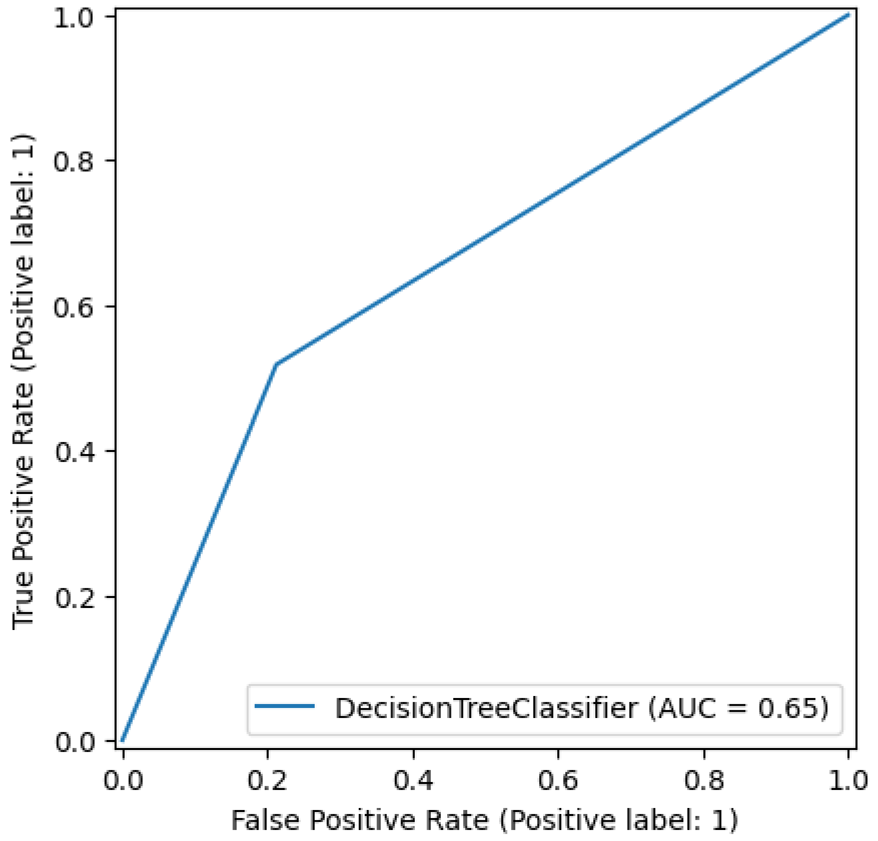


```
In [68]: import matplotlib.pyplot as plt
         from sklearn.metrics import RocCurveDisplay

         for prediction, name in zip(predictions, models_names):
           RocCurveDisplay.from_predictions(y_test, prediction, name=name)
           plt.show()
```

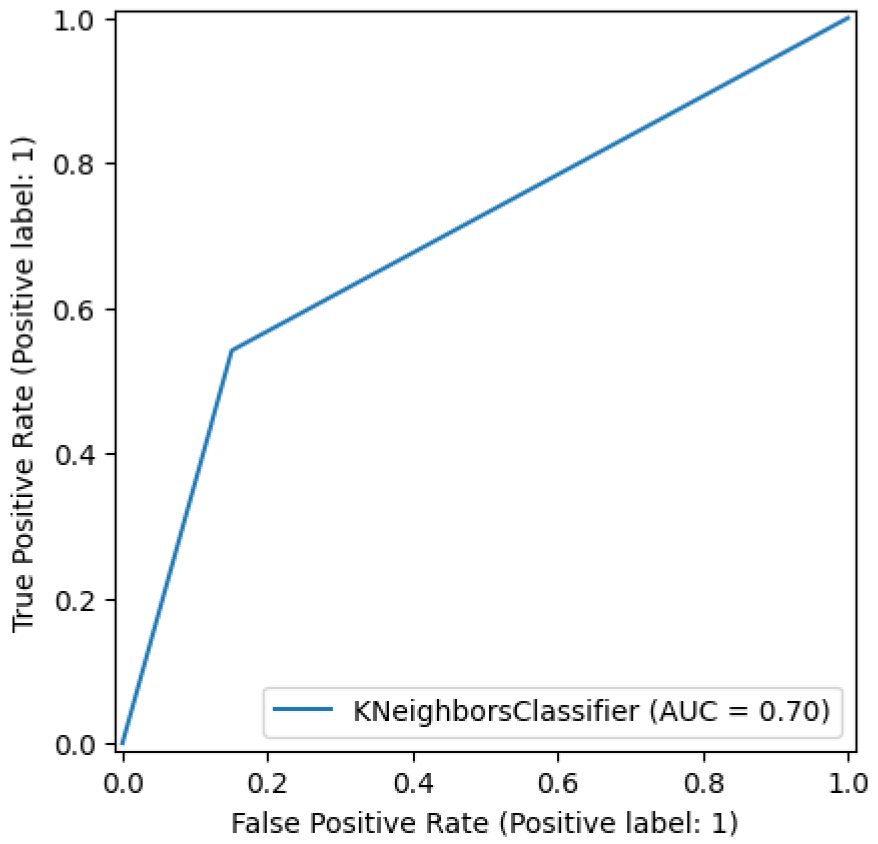

1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
LogisticRegression (AUC = 0.74)



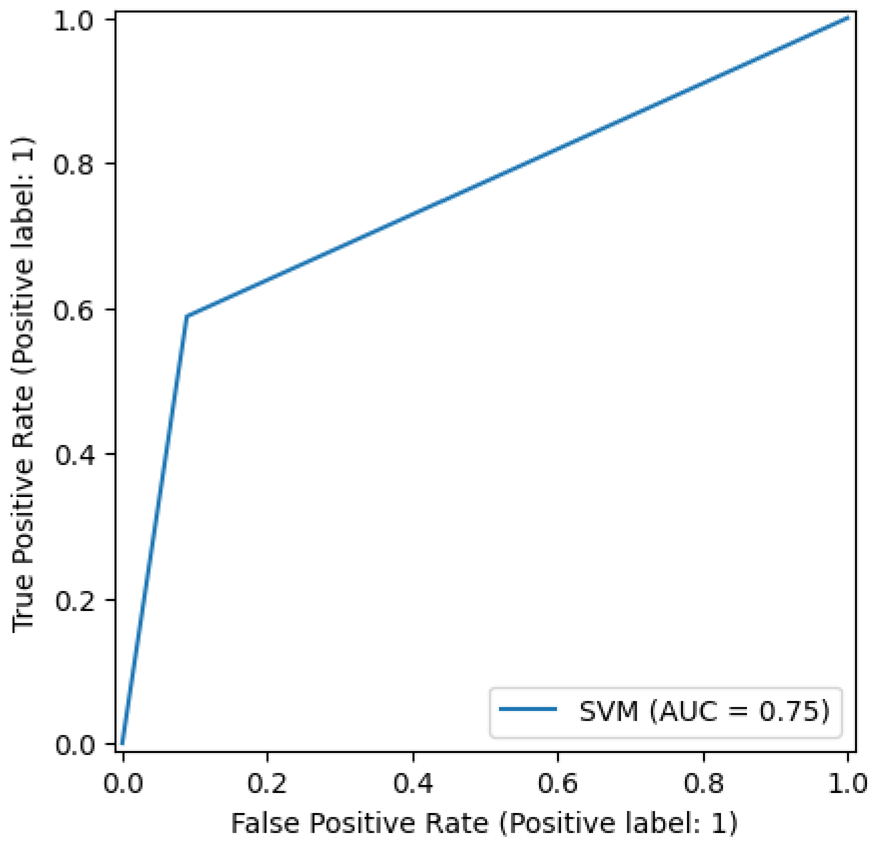
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
GaussianNB (AUC = 0.75)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)



DecisionTreeClassifier (AUC = 0.65)
True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0



1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
KNeighborsClassifier (AUC = 0.70)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)



AUC (Area Under ROC Curve)

The quality of classification using the ROC curve can be assessed by calculating **AUC** - the area under the ROC curve. The value of AUC indicates the ability of the model to distinguish between classes - a value closer to 1 is better.

In [69]:
```
from sklearn.metrics import roc_auc_score

for prediction, name in zip(predictions, models_names):
  auc = roc_auc_score(y_test, prediction)
  print(f"AUC dla {name}: {auc:.4f}")
```

```
AUC dla LogisticRegression: 0.7403
AUC dla GaussianNB: 0.7501
AUC dla DecisionTreeClassifier: 0.6527
AUC dla KNeighborsClassifier: 0.6952
AUC dla SVM: 0.7496
```