Predictive study of the Titanic incident

November 1, 2023

1 Predictive analysis

Based on certain characteristics of the passengers of the Titanic, we seek to build a classification algorithm that can predict with excellent efficiency the survival case of a passenger of the Titanic.

To do this, we use the different Scikit-Learn pipelines to preprocess the data, find the best hyperparameters and find the best classification algorithm among those tested.

Data Source

1.0.1 Importing the basics libraries

[163]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
pd.set_option("display.notebook_repr_html", False)
```

1.0.2 Importing dataset

[178]:

```
df = pd.read_csv("Titanic-Dataset.csv")

print("Data shape :", df.shape)

print("\nTwo first row of the dataset :\n")

df.head()
```

```
Data shape : (891, 12)

Two first row of the dataset :
```

[178]:

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
```

```
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

1.0.3 Data info

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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
```

1.1 Data preprocessing

SibSp:

This feature is based on the assumption that for a passenger whose loved ones have survived, the chances of survival are higher, and vice versa. Given the database provided, it cannot be said that there is a relationship between two passengers. We therefore prefer to remove this variable, but if you want more information, we advise you to take a look at this publication via this link below:

https://www.kaggle.com/code/ailuropus/extracting-family-relationships-on-titanic-sibsp

Name :

We prefer to remove this variable to simplify our work, but if you want to use it, you can use sklearn's CountVectorizer module to convert this variable to numbers. This will help determine if the name had an impact on a deceased person's case.

Ticket :

We also remove this variable, it deserves special treatment and the base dataset does not provide much information about it.

Passenger ID:

This feature is not important for predictive modeling. We delete it too.

1.1.1 Missing values

[4]:

```
df.isnull().sum()
```

[4]:

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

1.1.2 Let's put the features in a list according to their context to facilitate their pretreatment

[5]:

```
cat_missing_values = ['Embarked', 'Cabin']

cat_without_missing = ['Sex']

num_missing_values = ['Age']

drop_columns = ['PassengerId', 'Ticket', "Name"]

num_columns = ['Pclass', 'SibSp', 'Parch', 'Fare']
```

1.1.3 Importing librairies

[6]:

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
```

1.1.4 Pipline to replace missing categorical values and convert categories into numbers

[7]:

```
cat_preprocessor = make_pipeline(
                               ␣
 ↪SimpleImputer(strategy='constant',missing_values=np.nan,␣
 ↪fill_value='missing'),

                                OneHotEncoder(handle_unknown='ignore',␣
 ↪sparse_output=False)
                                )
```

1.1.5 Pipeline for scaling numeric values

[8]:

```
num_preprocessor = make_pipeline(SimpleImputer(strategy="median"),␣
 ↪MinMaxScaler())
```

1.1.6 Final data preprocessing

[9]:

```
data_preprocessing = make_column_transformer(

    (OneHotEncoder(sparse_output=False), cat_without_missing),

    (cat_preprocessor , cat_missing_values),

    (num_preprocessor, num_missing_values),

    ('drop' , drop_columns),

    (MinMaxScaler() , num_columns)
)
```

1.1.7 Split the dataset into x and y

[10]:

```
X = df.drop("Survived", axis = 1)
y = df["Survived"]
```

[11]:

```
X.shape
```

```
[11]: (891, 11)
```

```
[12]: y.shape
```

```
[12]: (891,)
```

1.1.8 Fit data_preprocessing

```
[14]: data_array = data_preprocessing.fit_transform(X)
```

```
[179]: df_encoded = pd.DataFrame(data_array,columns=data_preprocessing.
        ↪get_feature_names_out())
       df_encoded.head(2)
```

```
[179]:    onehotencoder__Sex_female  onehotencoder__Sex_male  pipeline-1__Embarked_C  \
       0                        0.0                      1.0                     0.0
       1                        1.0                      0.0                     1.0

          pipeline-1__Embarked_Q  pipeline-1__Embarked_S  \
       0                     0.0                     1.0
       1                     0.0                     0.0

          pipeline-1__Embarked_missing  pipeline-1__Cabin_A10  pipeline-1__Cabin_A14  \
       0                           0.0                    0.0                    0.0
       1                           0.0                    0.0                    0.0

          pipeline-1__Cabin_A16  pipeline-1__Cabin_A19  ...  pipeline-1__Cabin_F38  \
       0                    0.0                    0.0  ...                    0.0
       1                    0.0                    0.0  ...                    0.0

          pipeline-1__Cabin_F4  pipeline-1__Cabin_G6  pipeline-1__Cabin_T  \
       0                   0.0                   0.0                  0.0
       1                   0.0                   0.0                  0.0

          pipeline-1__Cabin_missing  pipeline-2__Age  minmaxscaler__Pclass  \
       0                        1.0         0.271174                   1.0
       1                        0.0         0.472229                   0.0

          minmaxscaler__SibSp  minmaxscaler__Parch  minmaxscaler__Fare
       0                0.125                  0.0            0.014151
       1                0.125                  0.0            0.139136

       [2 rows x 159 columns]
```

1.1.9 Split dataset with train_test_split method

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                                    X, y, test_size=0.25,␣
 ↪random_state=42)
```

[17]:
```
X_train.shape
```

[17]:
```
(668, 11)
```

[18]:
```
X_test.shape
```

[18]:
```
(223, 11)
```

1.2 Choose the classification algorithm using GridSearCV method of Sklearn

Selection of the most efficient classifier with the technical GridSearchCV

1.2.1 Random Forest Classifier

[19]:
```
from sklearn.ensemble import RandomForestClassifier as rfc
```

[47]:
```
pipe_rfc = Pipeline(steps=[('preprocessor', data_preprocessing),
                       ('rf_classifier', rfc(random_state = 42))])
```

[48]:
```
param_dict = {
    'rf_classifier__n_estimators' : [5, 10, 15, 20, 30, 60, 80, 100],
    'rf_classifier__max_features': ['sqrt', 'log2',None, .1, .25, .3, .35, .4],
    'rf_classifier__max_depth' : [None, 4, 7, 10, 15, 20, 25, 30, 35],
    'rf_classifier__criterion': ['gini', 'entropy', 'log_loss']
}
param_dict
```

[48]:
```
{'rf_classifier__n_estimators': [5, 10, 15, 20, 30, 60, 80, 100],
 'rf_classifier__max_features': ['sqrt',
  'log2',
  None,
  0.1,
  0.25,
  0.3,
  0.35,
  0.4],
 'rf_classifier__max_depth': [None, 4, 7, 10, 15, 20, 25, 30, 35],
 'rf_classifier__criterion': ['gini', 'entropy', 'log_loss']}
```

1.2.2 Cross validation

[49]:
```
from sklearn.model_selection import KFold

cross_validation= KFold(n_splits=5,
                                shuffle=True,
                                random_state=42)

cross_validation
```

[49]:
```
KFold(n_splits=5, random_state=42, shuffle=True)
```

1.2.3 GridSearch

[50]:
```
from sklearn.model_selection import GridSearchCV
```

[51]:
```
GridSear_rfc = GridSearchCV(pipe_rfc, param_dict, cv = cross_validation)
```

[52]:
```
GridSear_rfc
```

[52]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(sparse_output=False),
['Sex']),
('pipeline-1',
Pipeline(steps=[('simpleimputer',
          SimpleImputer(fill_value='missing',
                        strategy='constant')),
         ('onehotencoder',
          OneHotEncoder(handle_unknown='i…
['Pclass',
'SibSp',
'Parch',
'Fare'])])),
                                       ('rf_classifier',
RandomForestClassifier(random_state=42))]),
             param_grid={'rf_classifier__criterion': ['gini', 'entropy',
                                                      'log_loss'],
                         'rf_classifier__max_depth': [None, 4, 7, 10, 15, 20,
                                                      25, 30, 35],
                         'rf_classifier__max_features': ['sqrt', 'log2', None,
                                                         0.1, 0.25, 0.3, 0.35,
                                                         0.4],
                         'rf_classifier__n_estimators': [5, 10, 15, 20, 30, 60,
                                                         80, 100]})
```

[53]:
```
GridSear_rfc.fit(X_train, y_train)
```

[53]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(sparse_output=False),
['Sex']),
('pipeline-1',
Pipeline(steps=[('simpleimputer',
          SimpleImputer(fill_value='missing',
                        strategy='constant')),
         ('onehotencoder',
          OneHotEncoder(handle_unknown='i…
['Pclass',
'SibSp',
'Parch',
'Fare'])])),
                                        ('rf_classifier',
RandomForestClassifier(random_state=42))]),
             param_grid={'rf_classifier__criterion': ['gini', 'entropy',
                                                       'log_loss'],
                         'rf_classifier__max_depth': [None, 4, 7, 10, 15, 20,
                                                       25, 30, 35],
                         'rf_classifier__max_features': ['sqrt', 'log2', None,
                                                          0.1, 0.25, 0.3, 0.35,
                                                          0.4],
                         'rf_classifier__n_estimators': [5, 10, 15, 20, 30, 60,
                                                          80, 100]})
```

[54]:
```
GridSear_rfc.best_params_
```

[54]:
```
{'rf_classifier__criterion': 'entropy',
 'rf_classifier__max_depth': 10,
 'rf_classifier__max_features': 0.35,
 'rf_classifier__n_estimators': 15}
```

[196]:
```
round(GridSear_rfc.best_score_*100, 4)
```

[196]:
```
84.279
```

1.2.4 K-Nearest Neighbors Classifier

[56]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[57]:
```
pipe_knn = Pipeline(steps=[('preprocessor', data_preprocessing),
                        ('KN_classifier', KNeighborsClassifier())])
```

[58]:

```
params_knn = {'KN_classifier__n_neighbors': [1, 3, 5, 7, 9, 11, 12, 13, 14, 15,␣
 ↪16, 18, 20] ,
                "KN_classifier__weights": ['uniform', 'distance'],
                "KN_classifier__algorithm": ['auto', 'ball_tree', 'kd_tree',␣
 ↪'brute'],
              "KN_classifier__leaf_size" : [10, 20, 30, 40, 50, 60, 70, 80, 90,␣
 ↪100]
                }

params_knn
```

```
[58]: {'KN_classifier__n_neighbors': [1, 3, 5, 7, 9, 11, 12, 13, 14, 15, 16, 18, 20],
       'KN_classifier__weights': ['uniform', 'distance'],
       'KN_classifier__algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
       'KN_classifier__leaf_size': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
```

```
[59]: GridSear_knn = GridSearchCV(pipe_knn, params_knn  , cv = cross_validation)
```

```
[60]: GridSear_knn.fit(X_train, y_train)
```

```
[60]: GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
                   estimator=Pipeline(steps=[('preprocessor',
      ColumnTransformer(transformers=[('onehotencoder',
      OneHotEncoder(sparse_output=False),
      ['Sex']),
      ('pipeline-1',
      Pipeline(steps=[('simpleimputer',
                SimpleImputer(fill_value='missing',
                              strategy='constant')),
               ('onehotencoder',
                OneHotEncoder(handle_unknown='i…
      ('minmaxscaler',
      MinMaxScaler(),
      ['Pclass',
      'SibSp',
      'Parch',
      'Fare'])])),
                                             ('KN_classifier',
                                              KNeighborsClassifier())]),
                   param_grid={'KN_classifier__algorithm': ['auto', 'ball_tree',
                                                            'kd_tree', 'brute'],
                               'KN_classifier__leaf_size': [10, 20, 30, 40, 50, 60,
                                                            70, 80, 90, 100],
                               'KN_classifier__n_neighbors': [1, 3, 5, 7, 9, 11, 12,
                                                              13, 14, 15, 16, 18,
                                                              20],
                               'KN_classifier__weights': ['uniform', 'distance']})
```

```
[61]: GridSear_knn.best_params_
```

```
[61]: {'KN_classifier__algorithm': 'auto',
       'KN_classifier__leaf_size': 10,
       'KN_classifier__n_neighbors': 18,
       'KN_classifier__weights': 'uniform'}
```

```
[194]: round(GridSear_knn.best_score_*100, 4)
```

```
[194]: 80.6834
```

1.2.5 Adaboost classifier

```
[63]: # https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
       ↪AdaBoostClassifier.html

      from sklearn.ensemble import AdaBoostClassifier
```

```
[64]: pipe_adb = Pipeline(steps=[('preprocessor', data_preprocessing),
                             ('adb_classifier', AdaBoostClassifier(random_state =␣
       ↪42))])
```

```
[65]: params_adb = {'adb_classifier__n_estimators': [100, 120, 150, 160, 180, 200,␣
       ↪250, 300, 350] ,
                    "adb_classifier__learning_rate": [0.1, 0.2, 0.25, 0.3, 0.35, 0.
       ↪4, 0.5, 0.7, 0.8, 0.9, 1],
                    "adb_classifier__algorithm": ['SAMME', 'SAMME.R']
                    }

      params_adb
```

```
[65]: {'adb_classifier__n_estimators': [100, 120, 150, 160, 180, 200, 250, 300, 350],
       'adb_classifier__learning_rate': [0.1,
        0.2,
        0.25,
        0.3,
        0.35,
        0.4,
        0.5,
        0.7,
        0.8,
        0.9,
        1],
       'adb_classifier__algorithm': ['SAMME', 'SAMME.R']}
```

```
[66]: GridSear_adb = GridSearchCV(pipe_adb, params_adb  , cv = cross_validation)
```

[67]:

```
GridSear_adb.fit(X_train, y_train)
```

[67]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(sparse_output=False),
['Sex']),
('pipeline-1',
Pipeline(steps=[('simpleimputer',
          SimpleImputer(fill_value='missing',
                        strategy='constant')),
         ('onehotencoder',
          OneHotEncoder(handle_unknown='i…
'Ticket',
'Name']),
('minmaxscaler',
MinMaxScaler(),
['Pclass',
'SibSp',
'Parch',
'Fare'])])),
                                       ('adb_classifier',
                                        AdaBoostClassifier(random_state=42))]),
             param_grid={'adb_classifier__algorithm': ['SAMME', 'SAMME.R'],
                         'adb_classifier__learning_rate': [0.1, 0.2, 0.25, 0.3,
                                                           0.35, 0.4, 0.5, 0.7,
                                                           0.8, 0.9, 1],
                         'adb_classifier__n_estimators': [100, 120, 150, 160,
                                                          180, 200, 250, 300,
                                                          350]})
```

[68]:

```
GridSear_adb.best_params_
```

[68]:

```
{'adb_classifier__algorithm': 'SAMME',
 'adb_classifier__learning_rate': 1,
 'adb_classifier__n_estimators': 300}
```

[195]:

```
round(GridSear_adb.best_score_*100, 4)
```

[195]:

```
81.4252
```

1.2.6 Hist Gradient Boosting Classifier

[80]:

```
from sklearn.ensemble import HistGradientBoostingClassifier as HGBClassifier
```

[81]:

```
pipe_hgb = Pipeline(steps=[('preprocessor', data_preprocessing),
                       ('hgb_classifier', HGBClassifier(random_state = 42))])
```

[105]:
```
params_hgb = {'hgb_classifier__learning_rate': [0.03, 00.4, 0.05, 0.05, 0.1, 0.
 ↪2, 0.25],
                "hgb_classifier__l2_regularization" : [0, 4, 8, 10, 16, 20, 25],
                 "hgb_classifier__max_depth" : [6, 7, 8, 9, 12, 14, None],
                }

params_hgb
```

[105]:
```
{'hgb_classifier__learning_rate': [0.02, 0.03, 0.4, 0.05, 0.1, 0.2, 0.25],
 'hgb_classifier__l2_regularization': [0, 4, 8, 10, 16, 20, 25],
 'hgb_classifier__max_depth': [6, 7, 8, 9, 12, 14, None]}
```

[106]:
```
GridSear_hgb = GridSearchCV(pipe_hgb, params_hgb  , cv = cross_validation)
```

[107]:
```
GridSear_hgb.fit(X_train, y_train)
```

[107]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(sparse_output=False),
['Sex']),
('pipeline-1',
Pipeline(steps=[('simpleimputer',
          SimpleImputer(fill_value='missing',
                        strategy='constant')),
         ('onehotencoder',
          OneHotEncoder(handle_unknown='i…
['PassengerId',
'Ticket',
'Name']),
('minmaxscaler',
MinMaxScaler(),
['Pclass',
'SibSp',
'Parch',
'Fare'])])),
                                       ('hgb_classifier',
HistGradientBoostingClassifier(random_state=42))]),
             param_grid={'hgb_classifier__l2_regularization': [0, 4, 8, 10, 16,
                                                               20, 25],
                         'hgb_classifier__learning_rate': [0.02, 0.03, 0.4,
                                                           0.05, 0.1, 0.2,
                                                           0.25],
                         'hgb_classifier__max_depth': [6, 7, 8, 9, 12, 14,
                                                       None]})
```

[108]:
```
GridSear_hgb.best_params_
```

```
[108]: {'hgb_classifier__l2_regularization': 10,
        'hgb_classifier__learning_rate': 0.05,
        'hgb_classifier__max_depth': 7}
```

```
[197]: round(GridSear_hgb.best_score_*100, 4)
```

```
[197]: 83.9737
```

1.3 Selection of the best model

```
[129]: best_model = pd.DataFrame({'RF Classifier' : round(GridSear_rfc.
        ↪best_score_*100, 2)  , 'KNN Classifier' : round(GridSear_knn.
        ↪best_score_*100, 2),
                                'ADB Classifier' : round(GridSear_adb.
        ↪best_score_*100, 2) ,  'HGB Classifier' : round(GridSear_hgb.
        ↪best_score_*100, 2) }, index = ['GridSearch_best_score'])

       best_model
```

```
[129]:                        RF Classifier  KNN Classifier  ADB Classifier  \
       GridSearch_best_score          84.28           80.68           81.43

                              HGB Classifier
       GridSearch_best_score           83.97
```

The Random Forest Classifier achieved an efficiency of over 83.825% using the GridSearch method. We select this model and train it on the data set with the best hyperparameters

1.3.1 We train the best pipeline on all our data

```
[131]: final_model = Pipeline([('data_preprocessing', data_preprocessing),
                               ('rfc', rfc(random_state=42,
                                           max_depth=7,
                                           max_features='sqrt',
                                           n_estimators=100))
                              ])


       final_model.fit(X, y)
```

```
[131]: Pipeline(steps=[('data_preprocessing',
                        ColumnTransformer(transformers=[('onehotencoder',
       OneHotEncoder(sparse_output=False),
                                                         ['Sex']),
                                                        ('pipeline-1',
       Pipeline(steps=[('simpleimputer',
       SimpleImputer(fill_value='missing',
        strategy='constant')),
       ('onehotencoder',
       OneHotEncoder(handle_unknown='ignore',
        sparse_output=False))]),
                                                         ['Embarked', 'Cabin']),
                                                        ('pipeline-2',

       Pipeline(steps=[('simpleimputer',
       SimpleImputer(strategy='median')),
       ('minmaxscaler',
       MinMaxScaler())]),
                                                         ['Age']),
                                                        ('drop', 'drop',
                                                         ['PassengerId', 'Ticket',
                                                          'Name']),
                                                        ('minmaxscaler',
                                                         MinMaxScaler(),
                                                         ['Pclass', 'SibSp', 'Parch',
                                                          'Fare'])])),
                       ('rfc', RandomForestClassifier(max_depth=7, random_state=42))])
```

```
[198]: round(final_model.score(X, y)*100, 4)
```

```
[198]: 86.0831
```

Our model is ready to be saved for future predictions or for an internet deployment.

1.4 Save model

```
[141]: import joblib
```

```
[142]: # save the model to a file
       joblib.dump(final_model, 'randomforest_classifier.joblib')
```

```
[142]: ['randomforest_classifier.joblib']
```

1.5 Prediction

Predict a small random sample from the dataset.

```
[145]: random_set = df.sample(n=4, random_state=42)

       random_set_without_target = random_set.drop("Survived", axis = 1)

       random_set_without_target
```

```
[145]:      PassengerId  Pclass                                               Name  \
       709          710       3  Moubarek, Master. Halim Gonios ("William George")
       439          440       2             Kvillner, Mr. Johan Henrik Johannesson
       840          841       3                        Alhomaki, Mr. Ilmari Rudolf
```

```
720          721       2                  Harper, Miss. Annie Jessie "Nina"

        Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
709    male   NaN      1      1              2661  15.2458   NaN        C
439    male  31.0      0      0        C.A. 18723  10.5000   NaN        S
840    male  20.0      0      0  SOTON/O2 3101287   7.9250   NaN        S
720  female   6.0      0      1            248727  33.0000   NaN        S
```

[146]:
```python
model_save = joblib.load('randomforest_classifier.joblib')
print(model_save)
```

```
Pipeline(steps=[('data_preprocessing',
                 ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(sparse_output=False),
                                                  ['Sex']),
                                                 ('pipeline-1',
Pipeline(steps=[('simpleimputer',
SimpleImputer(fill_value='missing',
 strategy='constant')),
('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Embarked', 'Cabin']),
                                                 ('pipeline-2',
Pipeline(steps=[('simpleimputer',
SimpleImputer(strategy='median')),
('minmaxscaler',
MinMaxScaler())]),
                                                  ['Age']),
                                                 ('drop', 'drop',
                                                  ['PassengerId', 'Ticket',
                                                   'Name']),
                                                 ('minmaxscaler',
                                                  MinMaxScaler(),
                                                  ['Pclass', 'SibSp', 'Parch',
                                                   'Fare'])])),
                ('rfc', RandomForestClassifier(max_depth=7, random_state=42))])
```

[147]:
```python
model_save.predict(random_set_without_target)
```

[147]:
```
array([0, 0, 0, 1], dtype=int64)
```

1.5.1 The true values that were predicted

[148]:
```python
np.array(random_set["Survived"])
```

[148]:
```
array([1, 0, 0, 1], dtype=int64)
```