Titanic dataset-lesson3

May 19, 2022

1 1. Let's import the data

[1]:
```
import pandas as pd
```

[2]:
```
df = pd.read_csv("train.csv")
```

2 2. EDA (Exploratory data analysis)

Now, that we have the necessary data, the first step is to understand our data.

[3]:
```
# columns information
list(df.columns)
```

[3]:
```
['PassengerId',
 'Survived',
 'Pclass',
 'Name',
 'Sex',
 'Age',
 'SibSp',
 'Parch',
 'Ticket',
 'Fare',
 'Cabin',
 'Embarked']
```

[4]:
```
print(f"Total number of columns are {len(list(df.columns))}")
```

```
Total number of columns are 12
```

All of this columns are not the same, and it is important to make two differences. We need to separate the feature columns from our target column. In this case, the target column is **Survived**. Let's take a look on that data. Also the **PassengerId** is not important for the model because I assume that the id is total uncorrelated with the probability of survival

[5]:
```
# Let's seperate the dataset
y = df[["Survived"]]
```

[6]:
```
X = df.copy(deep=True) # Let's do a copy in order to preseve the df dataset in␣
↪memory and saved from inplace operations
```

[7]:
```
# Let's remove passenger id, and consider be moved -1
X = X.drop("PassengerId", axis=1)
```

[8]:
```
X
```

[8]:
```
     Survived  Pclass                                               Name  \
0           0       3                            Braund, Mr. Owen Harris
1           1       1  Cumings, Mrs. John Bradley (Florence Briggs Th…
2           1       3                             Heikkinen, Miss. Laina
3           1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4           0       3                           Allen, Mr. William Henry
..        …     …                                                …
886         0       2                              Montvila, Rev. Juozas
887         1       1                       Graham, Miss. Margaret Edith
888         0       3           Johnston, Miss. Catherine Helen "Carrie"
889         1       1                              Behr, Mr. Karl Howell
890         0       3                                Dooley, Mr. Patrick

        Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0      male  22.0      1      0         A/5 21171   7.2500   NaN        S
1    female  38.0      1      0          PC 17599  71.2833   C85        C
2    female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3    female  35.0      1      0            113803  53.1000  C123        S
4      male  35.0      0      0            373450   8.0500   NaN        S
..      …   …    …    …               …      …   …      …
886    male  27.0      0      0            211536  13.0000   NaN        S
887  female  19.0      0      0            112053  30.0000   B42        S
888  female   NaN      1      2        W./C. 6607  23.4500   NaN        S
889    male  26.0      0      0            111369  30.0000  C148        C
890    male  32.0      0      0            370376   7.7500   NaN        Q

[891 rows x 11 columns]
```

[9]:
```
# For our first version of the model let's remove also Name and ticket from the␣
↪equation
col_to_remove = ["Name", "Ticket", "Cabin"]
for col in col_to_remove:
    try:
        X = X.drop(col, axis=1) # can use inplace=True
    except KeyError:
        print(f"{col} not in data")
```

[10]:
```
# Let's take a look in the dataset
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Sex       891 non-null    object 
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64  
 5   Parch     891 non-null    int64  
 6   Fare      891 non-null    float64
 7   Embarked  889 non-null    object 
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

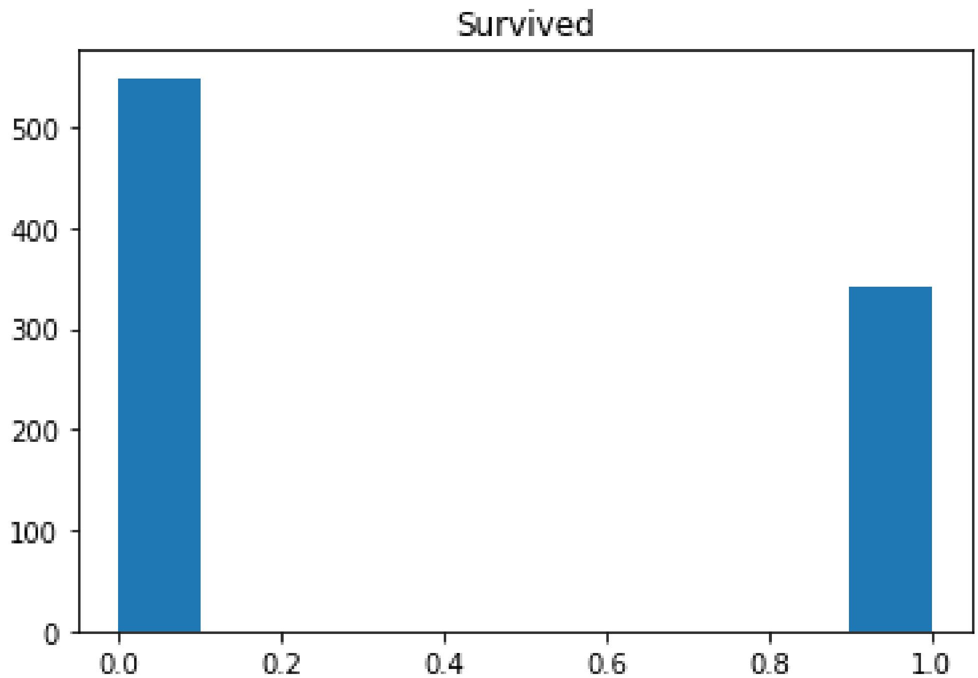
As we can see from the dataset there are some empty and some different types of objects that we have to take a look.

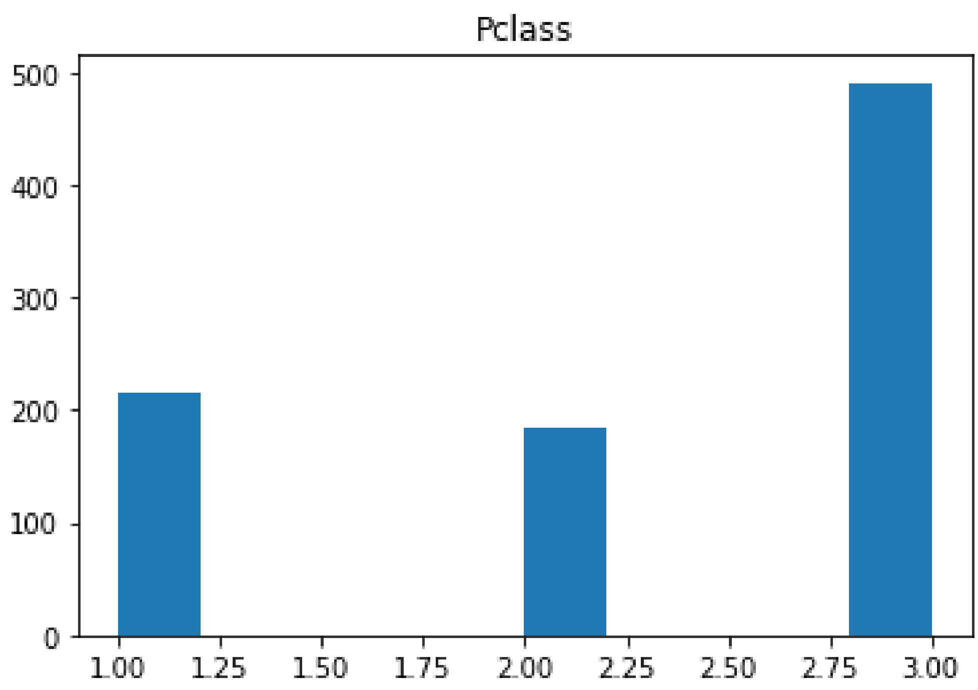
Let's compute some plots to see the distribution of the data.

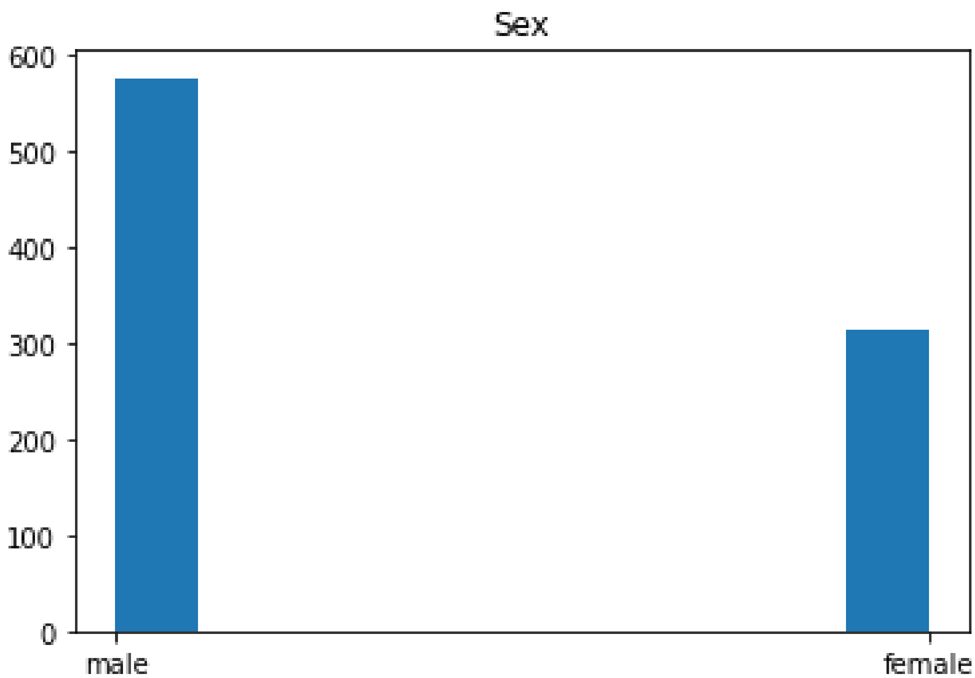
[11]:
```
import matplotlib.pyplot as plt
```

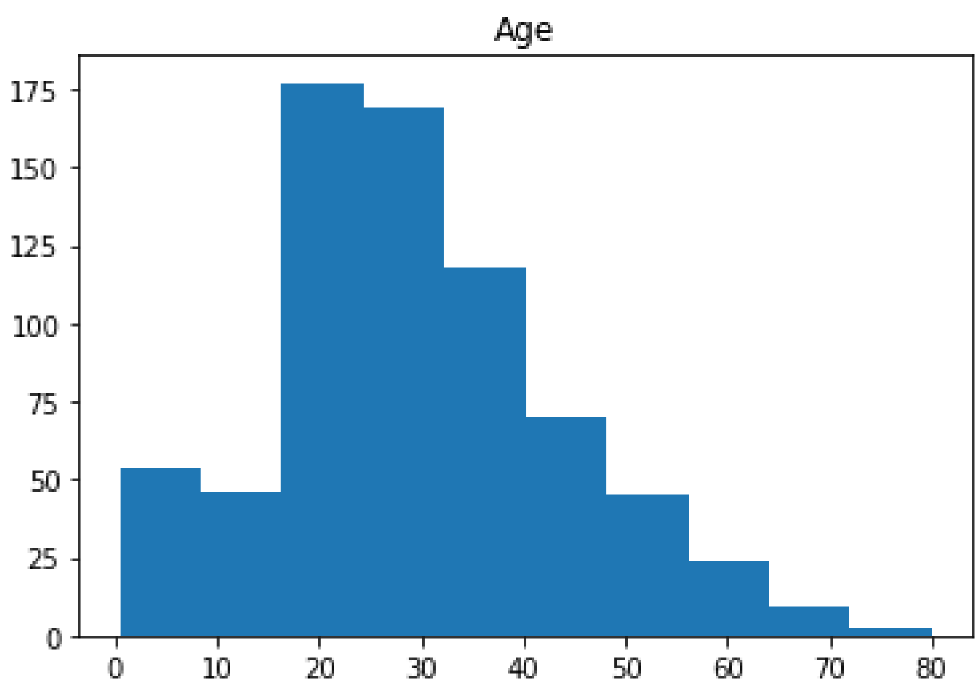
[12]:
```
columns = X.columns[:-1] # Embarked cannot be plotted just yet because is not␣
↪either categorical or converted to numbers
```

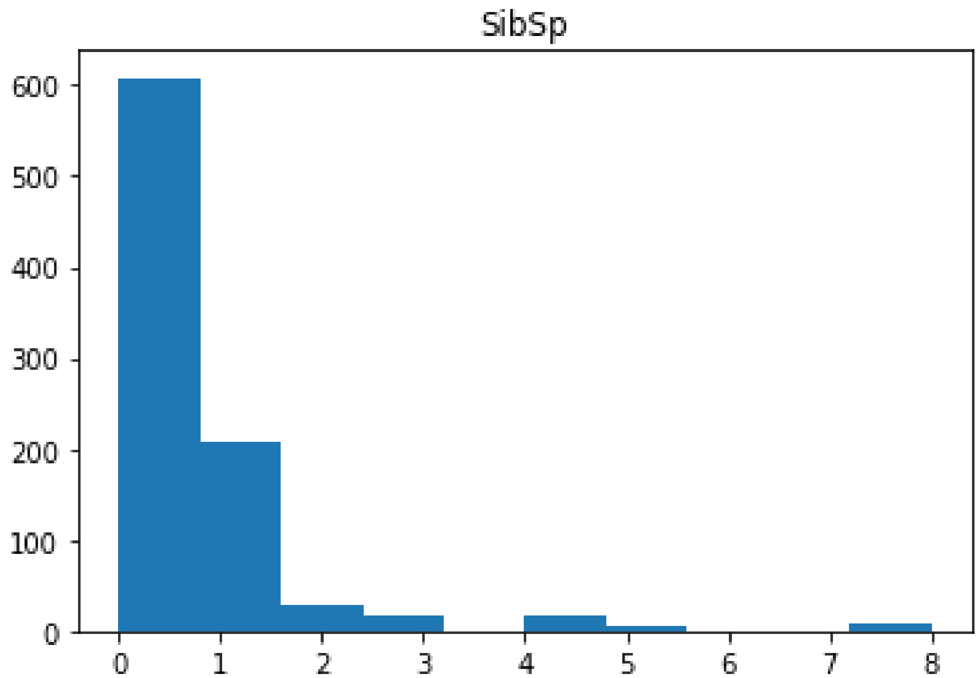
[13]:
```
for col in columns:
    plt.hist(x=df[col])
    plt.title(col)
    plt.show()
```

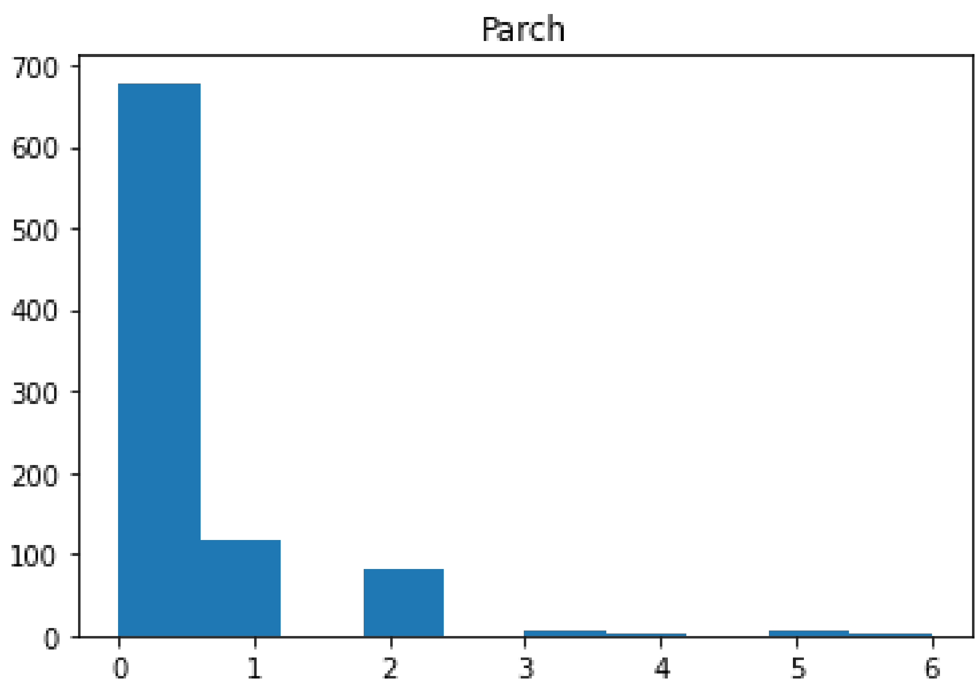

Survived
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

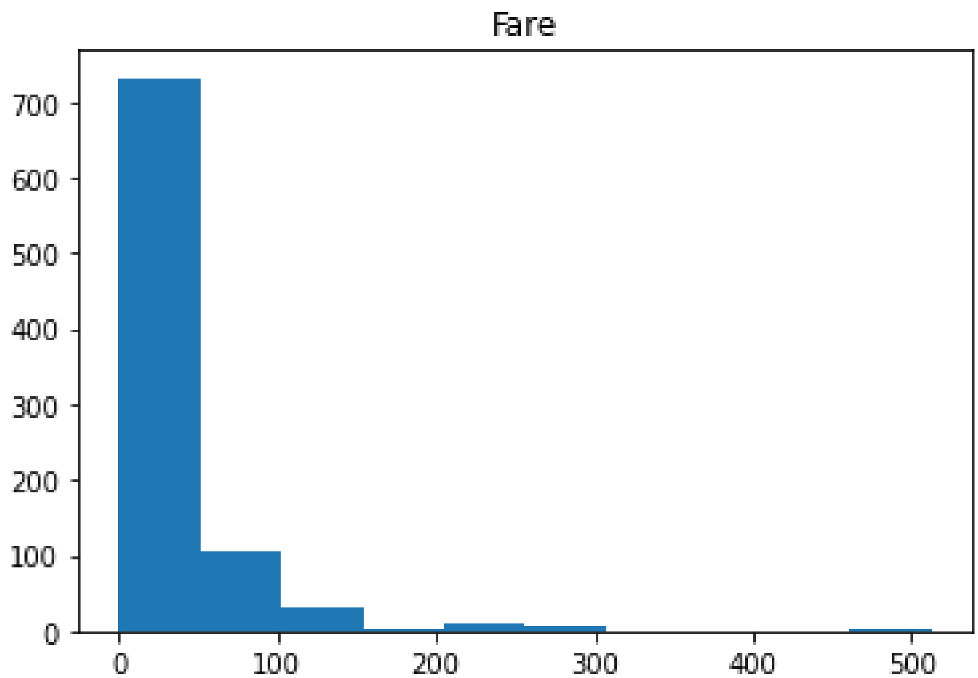

Pclass
500
400
300
200
100
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Sex
600
500
400
300
200
100
0
male
female


Age
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70
80


SibSp
0
100
200
300
400
500
600
0
1
2
3
4
5
6
7
8


Parch
0
100
200
300
400
500
600
700
0
1
2
3
4
5
6



Given this data, we have a better understanding of the data. And this helps to understand better how how should fill the values for the age. Let's start by filling with the average age

[14]:
```
# If we analyse some of the features with the predict label
X[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().
↪sort_values(by='Survived', ascending=False)
```

[14]:
```
   Pclass  Survived
0       1  0.629630
1       2  0.472826
2       3  0.242363
```

We can see that people on the best classes had better chances of survival

[15]:
```
X[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().
↪sort_values(by='Survived', ascending=False)
```

[15]:
```
      Sex  Survived
0  female  0.742038
1    male  0.188908
```

People from the female sex, as expected, had better chances of survival

[16]:
```
X[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().
↪sort_values(by='Survived', ascending=False)
```

[16]:
```
   SibSp  Survived
1      1  0.535885
2      2  0.464286
0      0  0.345395
3      3  0.250000
4      4  0.166667
5      5  0.000000
6      8  0.000000
```

[17]:
```
X[["Parch", "Survived"]].groupby(['Parch'], as_index=False).mean().
 ↪sort_values(by='Survived', ascending=False)
```

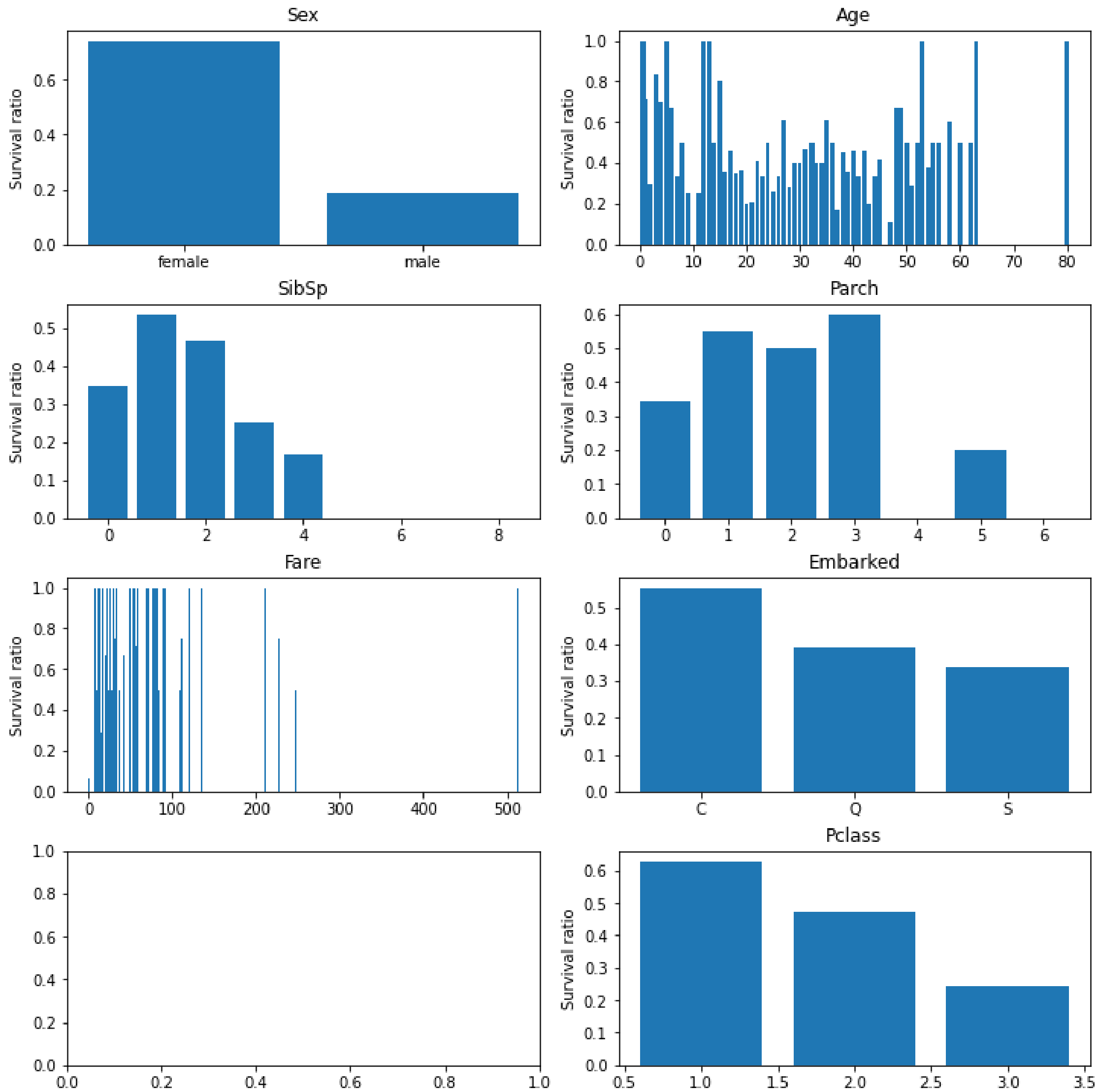
[17]:
```
   Parch  Survived
3      3  0.600000
1      1  0.550847
2      2  0.500000
0      0  0.343658
5      5  0.200000
4      4  0.000000
6      6  0.000000
```

[18]:
```
X.columns
```

[18]:
```
Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
       'Embarked'],
      dtype='object')
```

[ ]:
```

```

[19]:
```
# Lets compare the feature values with predict label
fig, axs = plt.subplots(4, 2, figsize=(10, 10), constrained_layout=True)

for i, col in enumerate(X.columns[1:]):
    sub_cols = [col, "Survived"]

    data = X[sub_cols].groupby([col], as_index=False).mean()

    axs.flat[i-1].bar(data[col], data["Survived"])
    axs.flat[i-1].set_title(col)
    axs.flat[i-1].set_ylabel("Survival ratio")
```



3 3. Data filling

[20]:
```
import numpy as np
age_mean = np.mean(X.Age)
```

[21]:
```
X.loc[:, ["Age"]] = X.Age.fillna(value=age_mean)
```

[22]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
```

```
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    object
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

Only **Embarked** missing. This, like gender are a very speacial type of variable, that we discussed. So, what we can do is to convert this, into categorical variable first. There are several techniques, that I we are going to test later, but for now we are going to do label encoding by mapping the categorical variable into a integer. We need to be careful with this because this might lead to trouble in linear models.

[23]:
```
# The first thing we do is to identify the unique variables
X.loc[:, "Sex"].unique()
```

[23]:
```
array(['male', 'female'], dtype=object)
```

4 4. Encode Categorical variables

[24]:
```
# as expected so now we need to map it to 0 and 1
sex_map = {"male": 0, "female": 1}
X.loc[:, "Sex"] = X.loc[:, "Sex"].apply(lambda x: sex_map[x] )
# This could also be done using LabelEncoder
```

[25]:
```
# now for embarked
# convert to categorical
X['Embarked'] = pd.Categorical(X.Embarked)
```

[26]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
```

```
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  889 non-null    category
dtypes: category(1), float64(2), int64(5)
memory usage: 49.9 KB
```

[27]:
```python
# The first thing we do is to identify the unique variables
X.loc[:, "Embarked"].unique()
```

[27]:
```
['S', 'C', 'Q', NaN]
Categories (3, object): ['C', 'Q', 'S']
```

[28]:
```python
# Let's count the most frequent embark location
X["Embarked"].value_counts() # Let's replace by S
```

[28]:
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

[29]:
```python
X["Embarked"] = X["Embarked"].fillna(value="S")
```

[30]:
```python
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  891 non-null    category
dtypes: category(1), float64(2), int64(5)
memory usage: 49.9 KB
```

[31]:
```python
Embarked_map = {"S":0, "C":1, "Q":2}
X.loc[:, "Embarked"] = X.loc[:, "Embarked"].apply(lambda x: Embarked_map[x] )
```

[32]:
```python
# conver Embarked to integer
X.Embarked = X.Embarked.astype("int")
```

5 5.Split data into train and test

[33]:
```
from sklearn.model_selection import train_test_split
```

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(X.drop("Survived", axis=1),
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 1)
```

[35]:
```
X_train
```

[35]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
114	3	1	17.000000	0	0	14.4583	1
874	2	1	28.000000	1	0	24.0000	1
76	3	0	29.699118	0	0	7.8958	0
876	3	0	20.000000	0	0	9.8458	0
674	2	0	29.699118	0	0	0.0000	0
..	…	…	…	…	…	…	…
715	3	0	19.000000	0	0	7.6500	0
767	3	1	30.500000	0	0	7.7500	2
72	2	0	21.000000	0	0	73.5000	0
235	3	1	29.699118	0	0	7.5500	0
37	3	0	21.000000	0	0	8.0500	0

[623 rows x 7 columns]

[36]:
```
y_train
```

[36]:

	Survived
114	0
874	1
76	0
876	0
674	0
..	…
715	0
767	0
72	0
235	0
37	0

[623 rows x 1 columns]

6 6. Train Decision Tree

[37]:
```
from sklearn import tree
```

[38]:
```
DTclf = tree.DecisionTreeClassifier()
```

[39]:
```
DTclf = DTclf.fit(X_train,y_train)
```

[40]:
```
results = DTclf.predict(X_test)
```

7 7.Let's analyse model performance

[41]:
```
from sklearn.metrics import confusion_matrix
```

[42]:
```
confusion_matrix(y_true=y_test, y_pred=results)
```

[42]:
```
array([[126,  27],
       [ 42,  73]])
```

[43]:
```
tn, fp, fn, tp =confusion_matrix(y_true=y_test, y_pred=results).ravel()
```

[44]:
```
# accuracy = (tp + tn)  / (tn + fp + fn + tp)
(tp + tn)  / (tn + fp + fn + tp)
```

[44]:
```
0.7425373134328358
```

[45]:
```
from sklearn.metrics import classification_report
```

[46]:
```
print(classification_report(y_test, results))
```

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.79       153
           1       0.73      0.63      0.68       115

    accuracy                           0.74       268
   macro avg       0.74      0.73      0.73       268
weighted avg       0.74      0.74      0.74       268
```

8 8. Let's generate predicts for the all test dataset

[47]:
```
df_test = pd.read_csv("test.csv")
```

[48]:
```
df_test
```

[48]:
```
     PassengerId  Pclass                                          Name  \
0            892       3                              Kelly, Mr. James
1            893       3              Wilkes, Mrs. James (Ellen Needs)
2            894       2                     Myles, Mr. Thomas Francis
3            895       3                              Wirz, Mr. Albert
4            896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
..           ...     ...                                           ...
413         1305       3                            Spector, Mr. Woolf
414         1306       1                  Oliva y Ocana, Dona. Fermina
415         1307       3                  Saether, Mr. Simon Sivertsen
416         1308       3                           Ware, Mr. Frederick
417         1309       3                      Peter, Master. Michael J

        Sex   Age  SibSp  Parch              Ticket      Fare Cabin Embarked
0      male  34.5      0      0              330911    7.8292   NaN        Q
1    female  47.0      1      0              363272    7.0000   NaN        S
2      male  62.0      0      0              240276    9.6875   NaN        Q
3      male  27.0      0      0              315154    8.6625   NaN        S
4    female  22.0      1      1             3101298   12.2875   NaN        S
..      ...   ...    ...    ...                 ...       ...   ...      ...
413    male   NaN      0      0           A.5. 3236    8.0500   NaN        S
414  female  39.0      0      0            PC 17758  108.9000  C105        C
415    male  38.5      0      0  SOTON/O.Q. 3101262    7.2500   NaN        S
416    male   NaN      0      0              359309    8.0500   NaN        S
417    male   NaN      1      1                2668   22.3583   NaN        C

[418 rows x 11 columns]
```

As can be seen, the predict dataset as features that we do not use on our dataset. So, before computing the predictions we need to prepare the dataset in the exact same way, we did for training. Let's start by removing the columns we do not want.

We start seeing some duplication in code. Good time to improve our code to be more production friendly.

[49]:
```python
# For our first version of the model let's remove also Name and ticket from the equation
col_to_remove = ["Name", "Ticket", "Cabin"]
for col in col_to_remove:
    try:
        df_test = df_test.drop(col, axis=1) # can use inplace=True
    except KeyError:
        print(f"{col} not in data")
```

[50]:
```python
df_test
```

[50]:
```
     PassengerId  Pclass     Sex   Age  SibSp  Parch      Fare Embarked
0            892       3    male  34.5      0      0    7.8292        Q
```

```
1              893       3  female  47.0       1       0    7.0000        S
2              894       2    male  62.0       0       0    9.6875        Q
3              895       3    male  27.0       0       0    8.6625        S
4              896       3  female  22.0       1       1   12.2875        S
..             …      …      …     …       …      …         …        …
413           1305       3    male   NaN       0       0    8.0500        S
414           1306       1  female  39.0       0       0  108.9000        C
415           1307       3    male  38.5       0       0    7.2500        S
416           1308       3    male   NaN       0       0    8.0500        S
417           1309       3    male   NaN       1       1   22.3583        C

[418 rows x 8 columns]
```

Now, in this case we cannot remove PassengerId, because we need to identify the predictions, in order to submit to kaggle.

[51]:
```
# let's move the columns to index
df_test.set_index("PassengerId", inplace=True)
```

[52]:
```
df_test
```

[52]:
```
             Pclass     Sex   Age  SibSp  Parch      Fare Embarked
PassengerId
892               3    male  34.5      0      0    7.8292        Q
893               3  female  47.0      1      0    7.0000        S
894               2    male  62.0      0      0    9.6875        Q
895               3    male  27.0      0      0    8.6625        S
896               3  female  22.0      1      1   12.2875        S
…             …      …     …     …      …       …       …
1305              3    male   NaN      0      0    8.0500        S
1306              1  female  39.0      0      0  108.9000        C
1307              3    male  38.5      0      0    7.2500        S
1308              3    male   NaN      0      0    8.0500        S
1309              3    male   NaN      1      1   22.3583        C

[418 rows x 7 columns]
```

[53]:
```
# Let's take a look at the data
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 892 to 1309
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    418 non-null    int64
 1   Sex       418 non-null    object
 2   Age       332 non-null    float64
```

```
 3   SibSp     418 non-null    int64
 4   Parch     418 non-null    int64
 5   Fare      417 non-null    float64
 6   Embarked  418 non-null    object
dtypes: float64(2), int64(3), object(2)
memory usage: 26.1+ KB
```

[54]:
```
# as expected we have some missing data.
# to fill the Age, we need to use the average that we computed in the train␣
 ↪dataset
df_test.loc[:, ["Age"]] = df_test.Age.fillna(value=age_mean)
```

[55]:
```
# Let's take a look at the data
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 892 to 1309
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    418 non-null    int64
 1   Sex       418 non-null    object
 2   Age       418 non-null    float64
 3   SibSp     418 non-null    int64
 4   Parch     418 non-null    int64
 5   Fare      417 non-null    float64
 6   Embarked  418 non-null    object
dtypes: float64(2), int64(3), object(2)
memory usage: 26.1+ KB
```

[56]:
```
# For the Fare, we can do the same, but we need to use the training avg. Since␣
 ↪is the first time, ww need  to compute values
fare_median = df.Fare.median()
```

[57]:
```
df_test.loc[:, ["Fare"]] = df_test.Fare.fillna(value=fare_median)
```

[58]:
```
# Let's take a look at the data
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 892 to 1309
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    418 non-null    int64
 1   Sex       418 non-null    object
 2   Age       418 non-null    float64
 3   SibSp     418 non-null    int64
```

```
 4   Parch     418 non-null    int64
 5   Fare      418 non-null    float64
 6   Embarked  418 non-null    object
dtypes: float64(2), int64(3), object(2)
memory usage: 26.1+ KB
```

[59]:
```
df_test
```

[59]:
```
             Pclass     Sex        Age  SibSp  Parch      Fare Embarked
PassengerId
892               3    male  34.500000      0      0    7.8292        Q
893               3  female  47.000000      1      0    7.0000        S
894               2    male  62.000000      0      0    9.6875        Q
895               3    male  27.000000      0      0    8.6625        S
896               3  female  22.000000      1      1   12.2875        S
…              …      …        …     …     …       …        …
1305              3    male  29.699118      0      0    8.0500        S
1306              1  female  39.000000      0      0  108.9000        C
1307              3    male  38.500000      0      0    7.2500        S
1308              3    male  29.699118      0      0    8.0500        S
1309              3    male  29.699118      1      1   22.3583        C

[418 rows x 7 columns]
```

[60]:
```
# we need also to apply the mappings
df_test.loc[:, "Sex"] = df_test.loc[:, "Sex"].apply(lambda x: sex_map[x] )
df_test.loc[:, "Embarked"] = df_test.loc[:, "Embarked"].apply(lambda x:␣
 ↪Embarked_map[x] )
```

[61]:
```
# lets make predictions
DTclf.predict(df_test)
```

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

```
        1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
        1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
        0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0])
```

[62]:
```
kaggle_data = df_test.copy(deep=True)
```

9 9. Generate results to kaggle

[63]:
```
kaggle_data["Survived"] = DTclf.predict(kaggle_data)
```

[64]:
```
kaggle_data
```

[64]:
```
             Pclass  Sex        Age  SibSp  Parch      Fare  Embarked  \
PassengerId
892               3    0  34.500000      0      0    7.8292         2
893               3    1  47.000000      1      0    7.0000         0
894               2    0  62.000000      0      0    9.6875         2
895               3    0  27.000000      0      0    8.6625         0
896               3    1  22.000000      1      1   12.2875         0
…               …  …        …      …    …        …       …
1305              3    0  29.699118      0      0    8.0500         0
1306              1    1  39.000000      0      0  108.9000         1
1307              3    0  38.500000      0      0    7.2500         0
1308              3    0  29.699118      0      0    8.0500         0
1309              3    0  29.699118      1      1   22.3583         1

             Survived
PassengerId
892                 0
893                 1
894                 0
895                 0
896                 1
…                 …
1305                0
1306                1
1307                0
1308                0
1309                0

[418 rows x 8 columns]
```

[65]:
```
# kaggle data format
kaggle_data = kaggle_data[["Survived"]]
```

[66]:
```
kaggle_data.reset_index(inplace=True)
```

[67]:
```
kaggle_data
```

[67]:
```
     PassengerId  Survived
0            892         0
1            893         1
2            894         0
3            895         0
4            896         1
..           ...       ...
413         1305         0
414         1306         1
415         1307         0
416         1308         0
417         1309         0

[418 rows x 2 columns]
```

[68]:
```
kaggle_data.to_csv("submission_v1.csv", index=False)
```

10 11. K-Fold validation

[69]:
```
# Let's see if by doing a k-fold validation we achieve different accuracy␣
 ↪metrics
from sklearn.model_selection import KFold
```

[70]:
```
k = 5
kf = KFold(n_splits=k, random_state=None)
```

[71]:
```
y
```

[71]:
```
     Survived
0           0
1           1
2           1
3           1
4           0
..        ...
886         0
887         1
888         0
889         1
890         0

[891 rows x 1 columns]
```

[72]:
```
# let's compute accury using scikit learn
from sklearn.metrics import accuracy_score
```

[73]:
```
acc_score = []

for train_index , test_index in kf.split(X.drop("Survived", axis=1)):

    X_train_fold , X_test_fold = X.drop("Survived", axis=1).iloc[train_index,:
↪],X.drop("Survived", axis=1).iloc[test_index,:]
    y_train_fold , y_test_fold = y.iloc[train_index] , y.iloc[test_index]

    DTclf.fit(X_train_fold,y_train_fold)
    pred_values = DTclf.predict(X_test_fold)

    acc = accuracy_score(pred_values , y_test_fold)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/k
```

[74]:
```
avg_acc_score
```

[74]: 0.7733412842884941

[75]:
```
# Let's also compute the ROC curve for the current model we have
from sklearn.metrics import roc_curve, auc
```

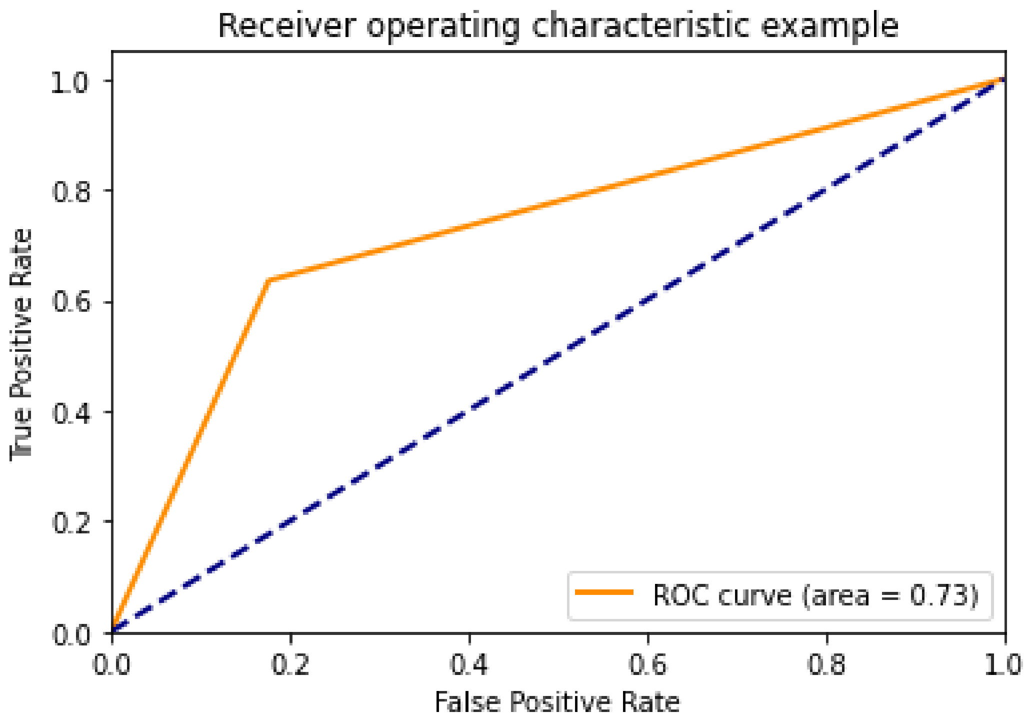
[76]:
```
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=results, pos_label=1)
```

[77]:
```
AUC = auc(fpr, tpr)
```

[ ]:
```

```

[78]:
```
plt.figure()
lw = 2
plt.plot(
    fpr,
    tpr,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % AUC,
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
```

```
plt.legend(loc="lower right")
plt.show()
```



11 12.Let's try to close the end to end process and try to improve our model

[79]:
```
# Most common parameters
MAX_DEPTH = [2, 5, 10, 25, 50]
MIN_SAMPLE_SPLIT = [2, 10, 20, 50]
MAX_FEATURES = [ 1,2,3,4,5,6,7, 8]
```

[80]:
```
from sklearn.model_selection import GridSearchCV
```

[81]:
```
parameters = {'max_depth':MAX_DEPTH, 'min_samples_split':MIN_SAMPLE_SPLIT,␣
 ↪"max_features":MAX_FEATURES}
```

[82]:
```
clf = GridSearchCV(DTclf, parameters, cv=5, scoring="accuracy")
```

[83]:
```
clf.fit(X.drop("Survived", axis=1), y)
```

```
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
```

```
100 fits failed out of a total of 800.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
100 fits failed with the following error:
Traceback (most recent call last):
  File
"/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/model_selection/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/tree/_classes.py", line 937, in fit
    super().fit(
  File
"/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/tree/_classes.py", line 308, in fit
    raise ValueError("max_features must be in (0, n_features]")
ValueError: max_features must be in (0, n_features]

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/model_selection/_search.py:969: UserWarning: One or more of the
test scores are non-finite: [0.6454146  0.66899127 0.69933463 0.70832339
0.73392756 0.69260561
 0.76202373 0.79123093 0.71279267 0.77668696 0.7507815  0.7307702
 0.77331618 0.78448936 0.744291   0.7418555  0.78673655 0.75309146
 0.77331618 0.78337204 0.77893415 0.77331618 0.78448936 0.78673655
 0.77331618 0.77331618 0.77331618 0.77331618        nan        nan
        nan        nan 0.7464817  0.71387232 0.73749922 0.73748038
 0.78340343 0.7733915  0.74973322 0.78113113 0.81372795 0.78111857
 0.8081037  0.80807231 0.80692989 0.81595631 0.80360304 0.79354717
 0.81483272 0.81817839 0.81707363 0.80920218 0.82041931 0.81817839
 0.79801017 0.79797251 0.81257297 0.81369657 0.81706108 0.8047078
        nan        nan        nan        nan 0.78676166 0.73291695
 0.78114996 0.76556399 0.78455213 0.790145   0.79128115 0.78009541
 0.79463311 0.80925868 0.80699893 0.79131881 0.79919654 0.81370912
 0.80922102 0.7868056  0.79237964 0.80026991 0.80251083 0.80138723
 0.79577553 0.81151215 0.81487038 0.7968803  0.80138723 0.81261063
 0.82271044 0.80920846        nan        nan        nan        nan
 0.75425271 0.79245496 0.79911493 0.78678049 0.75986442 0.798029
 0.79691168 0.7969054  0.77557592 0.80696127 0.79910866 0.79464566
 0.78118134 0.79463938 0.82041303 0.80585651 0.78344109 0.793566
 0.80146256 0.80139351 0.76437763 0.78683698 0.81375934 0.80251711
```

```
 0.76999561 0.79694307 0.8126483  0.80920846        nan        nan
        nan        nan 0.75983303 0.76091269 0.78563806 0.78901513
 0.76214299 0.79801017 0.77669952 0.7722428  0.77335384 0.81258553
 0.80356538 0.81595631 0.76548867 0.80255477 0.81153098 0.8092524
 0.77334756 0.80029502 0.80814136 0.80698638 0.77110665 0.79463938
 0.79691796 0.8013684  0.77111292 0.79917143 0.81488293 0.80920846
        nan        nan        nan        nan]
  warnings.warn(
```

[83]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [2, 5, 10, 25, 50],
                         'max_features': [1, 2, 3, 4, 5, 6, 7, 8],
                         'min_samples_split': [2, 10, 20, 50]},
             scoring='accuracy')
```

12 13. Find best parameters

[84]:
```
# check best params
clf.best_params_
```

[84]:
```
{'max_depth': 10, 'max_features': 7, 'min_samples_split': 20}
```

[85]:
```
clf.best_score_
```

[85]:
```
0.8227104387671835
```

[86]:
```
best_model = clf.best_estimator_
```

[87]:
```
# since our best model only requires 4 features, we need to select them
best_model.feature_importances_
```

[87]:
```
array([0.16047751, 0.47264177, 0.12496358, 0.03481855, 0.01861867,
       0.17510981, 0.0133701 ])
```

[88]:
```
X.columns
```

[88]:
```
Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
       'Embarked'],
      dtype='object')
```

[89]:
```
df_test
```

[89]:
```
             Pclass  Sex        Age  SibSp  Parch      Fare  Embarked
PassengerId
892               3    0  34.500000      0      0    7.8292         2
893               3    1  47.000000      1      0    7.0000         0
```

```
894                  2    0   62.000000      0       0     9.6875          2
895                  3    0   27.000000      0       0     8.6625          0
896                  3    1   22.000000      1       1    12.2875          0
…                 …   …         …      …     …       …       …
1305                 3    0   29.699118      0       0     8.0500          0
1306                 1    1   39.000000      0       0   108.9000          1
1307                 3    0   38.500000      0       0     7.2500          0
1308                 3    0   29.699118      0       0     8.0500          0
1309                 3    0   29.699118      1       1    22.3583          1

[418 rows x 7 columns]
```

[90]:
```
# lets make predictions
predictions= best_model.predict(df_test)
```

[91]:
```
# write new model to kaggle

# Let's create a function
def write_results_to_disk(test_data, predictions, csv_name):
    data = test_data.copy(deep=True)
    data["Survived"] = predictions
    data = data[["Survived"]]
    data.reset_index(inplace=True)
    data.to_csv(csv_name, index=False)
    print(f"Successfully written {csv_name}")
```

[92]:
```
write_results_to_disk(test_data=df_test, predictions=predictions,␣
 ↪csv_name="submission_v2.csv")
```

```
Successfully written submission_v2.csv
```

13 14. Let's Improve the features

[93]:
```
importances = best_model.feature_importances_
```

As we can see from above the 1st, 2nd, 3rd and 4rd have the most total importance

importances

[94]:
```
features_to_keep = ["Pclass", "Sex", "Age", "Fare"]
```

[95]:
```
X
```

[95]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	0	22.000000	1	0	7.2500	0
1	1	1	1	38.000000	1	0	71.2833	1
2	1	3	1	26.000000	0	0	7.9250	0

```
3          1     1   1  35.000000     1      0  53.1000        0
4          0     3   0  35.000000     0      0   8.0500        0
..       …    …  …      …    …    …      …       …
886        0     2   0  27.000000     0      0  13.0000        0
887        1     1   1  19.000000     0      0  30.0000        0
888        0     3   1  29.699118     1      2  23.4500        0
889        1     1   0  26.000000     0      0  30.0000        1
890        0     3   0  32.000000     0      0   7.7500        2

[891 rows x 8 columns]
```

[96]:
```
# so if we train the same model
MAX_DEPTH = [2, 5, 10, 25, 50]
MIN_SAMPLE_SPLIT = [2, 10, 20, 50]
MAX_FEATURES = [ 1,2,3,4]
parameters = {'max_depth':MAX_DEPTH, 'min_samples_split':MIN_SAMPLE_SPLIT,␣
 ↪"max_features":MAX_FEATURES}
clf = GridSearchCV(DTclf, parameters, cv=5, scoring="accuracy")
clf.fit(X[features_to_keep], y)
```

[96]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [2, 5, 10, 25, 50],
                         'max_features': [1, 2, 3, 4],
                         'min_samples_split': [2, 10, 20, 50]},
             scoring='accuracy')
```

[97]:
```
clf.best_params_
```

[97]:
```
{'max_depth': 10, 'max_features': 3, 'min_samples_split': 10}
```

[98]:
```
clf.best_score_
```

[98]:
```
0.8182035026049841
```

[99]:
```
best_model_2 = clf.best_estimator_
```

[100]:
```
# lets make predictions
predictions_2= best_model_2.predict(df_test[features_to_keep])
```

[101]:
```
write_results_to_disk(test_data=df_test, predictions=predictions_2,␣
 ↪csv_name="submission_v3.csv")
```

```
Successfully written submission_v3.csv
```

14 Before moving one let's build our pipeline in order to perform easier iterations

[102]:
```
def remove_columns(data, cols):
    for col in cols:
        try:
            data = data.drop(col, axis=1) # can use inplace=True
        except KeyError:
            print(f"{col} not in data")
    return data
```

[103]:
```
def age_avg_filling(data, population_average):
    # as expected we have some missing data.
    # to fill the Age, we need to use the average that we computed in the train␣
 ↪dataset
    data.loc[:, ["Age"]] = data.Age.fillna(value=population_average)
    return data
```

[104]:
```
def fare_median_filling(data, population_median):
    # as expected we have some missing data.
    # to fill the Age, we need to use the average that we computed in the train␣
 ↪dataset
    data.loc[:, ["Fare"]] = data.Fare.fillna(value=population_median)
    return data
```

[105]:
```
from sklearn import preprocessing
def label_encode(data, col_to_encode):
    le = preprocessing.LabelEncoder()
    data[col_to_encode] = le.fit_transform(data[col_to_encode])
    return data
```

[106]:
```
class PredictPipeline:
    @staticmethod
    def predict(predict_data, population_average, population_median, model):

        # preprocessing
        data = age_avg_filling(data=predict_data,␣
 ↪population_average=population_average)
        data = fare_median_filling(data=data,␣
 ↪population_median=population_median)

        # feature engineering
        data = label_encode(data=data, col_to_encode="Sex")
        data = label_encode(data=data, col_to_encode="Embarked")
```

```
        return model.predict(data)

    @staticmethod
    def write_results_to_disk(test_data, predictions, csv_name):
        data = test_data.copy(deep=True)
        data["Survived"] = predictions
        data = data[["Survived"]]
        data.reset_index(inplace=True)
        data.to_csv(csv_name, index=False)
        print(f"Successfully written {csv_name}")
```

[107]:
```
# generate results
results = PredictPipeline.predict(predict_data=df_test,
                        population_average=age_mean,
                        population_median=fare_median,
                        model=best_model)
```

[108]:
```
# write predictions
PredictPipeline.write_results_to_disk(test_data=df_test,
                                      predictions=results,
                                      csv_name="submission_v7.csv")
```

```
Successfully written submission_v7.csv
```

15 Let's test diferent models

[109]:
```
# To train diferent models using k-fold validation let's do
# let's compute accury using scikit learn
from sklearn.metrics import accuracy_score

def k_fold_validation(k, X, model):
    kf = KFold(n_splits=k, random_state=None)
    acc_score = []

    for train_index , test_index in kf.split(X):

        X_train_fold , X_test_fold = X.iloc[train_index,:],X.iloc[test_index,:]
        y_train_fold , y_test_fold = y.iloc[train_index] , y.iloc[test_index]

        model.fit(X_train_fold,y_train_fold)
        pred_values = model.predict(X_test_fold)

        acc = accuracy_score(pred_values , y_test_fold)
        acc_score.append(acc)

    avg_acc_score = sum(acc_score)/k
```

```
    return avg_acc_score, model
```

[110]:
```
from sklearn.ensemble import RandomForestClassifier
```

[111]:
```
# Instantiate Random Forest Classifier
RFclf = RandomForestClassifier()
```

[112]:
```
score, random_forest = k_fold_validation(k=3, X=X.drop("Survived", axis=1),␣
↪model=RFclf)
```

```
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
```

[113]:
```
# generate results
results = PredictPipeline.predict(predict_data=df_test,
                        population_average=age_mean,
                        population_median=fare_median,
                        model=random_forest)
# write predictions
PredictPipeline.write_results_to_disk(test_data=df_test,
                                      predictions=results,
                                      csv_name="submission_random_forest.csv")
```

```
Successfully written submission_random_forest.csv
```

16 XGBoost

[114]:
```
import xgboost as xgb
```

[115]:
```
boost_clf = xgb.XGBClassifier()
```

[116]:
```
score, bost_model = k_fold_validation(k=3, X=X.drop("Survived", axis=1),␣
↪model=boost_clf)
score
```

[116]: 0.8047138047138048

[117]:
```
# generate results
results = PredictPipeline.predict(predict_data=df_test,
                        population_average=age_mean,
                        population_median=fare_median,
                        model=bost_model)
# write predictions
PredictPipeline.write_results_to_disk(test_data=df_test,
                                      predictions=results,
                                      csv_name="submission_boost.csv")
```

```
Successfully written submission_boost.csv
```

17 KNN

[118]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[119]:
```
knn_clf = KNeighborsClassifier()
```

[120]:
```
score, KNN_model = k_fold_validation(k=3, X=X.drop("Survived", axis=1),␣
 ↪model=knn_clf)
score
```

```
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:198: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:198: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/titanic/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:198: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

[120]: 0.6992143658810326

[121]:
```
# generate results
results = PredictPipeline.predict(predict_data=df_test,
                        population_average=age_mean,
                        population_median=fare_median,
                        model=KNN_model)
# write predictions
PredictPipeline.write_results_to_disk(test_data=df_test,
```

```
                                              predictions=results,
                                              csv_name="submission_KNN.csv")
```

```
Successfully written submission_KNN.csv
```

18 Let's improve features

[122]:
```
# Let's convert age into bins
X['AgeBand'] = pd.cut(X['Age'], 5)
```

[123]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

[124]:
```
age_label_encoder = le.fit(X['AgeBand'])

X['Age'] = age_label_encoder.transform(X['AgeBand'])
X.drop("AgeBand", axis=1, inplace=True)
```

18.1 Create Family size feature

[125]:
```
X['FamilySize'] = X['SibSp'] + X['Parch'] + 1
X['IsAlone'] = 0
X.loc[X['FamilySize'] == 1, 'IsAlone'] = 1
```

[126]:
```
X[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

[126]:

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

[127]:
```
# Let's delete FamilySize, SibSp, Parch
X.drop(["FamilySize", "SibSp", "Parch"], axis=1, inplace=True)
```

19 Convert Fare to Band

[128]:
```
X['FareBand'] = pd.qcut(X['Fare'], 4)
X[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().
  ↪sort_values(by='FareBand', ascending=True)
```

[128]:

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

[129]:
```
fare_label_encoder = le.fit(X['FareBand'])
X['Fare'] = fare_label_encoder.transform(X['FareBand'])
```

[130]:
```
X.drop("FareBand", axis=1, inplace=True)
```

[137]:
```
X.describe()
```

[137]:
```
         Survived      Pclass         Sex         Age        Fare    Embarked  \
count  891.000000  891.000000  891.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642    0.352413    1.290685    1.497194    0.361392
std      0.486592    0.836071    0.477990    0.812620    1.118156    0.635673
min      0.000000    1.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    2.000000    0.000000    1.000000    0.500000    0.000000
50%      0.000000    3.000000    0.000000    1.000000    1.000000    0.000000
75%      1.000000    3.000000    1.000000    2.000000    2.000000    1.000000
max      1.000000    3.000000    1.000000    4.000000    3.000000    2.000000

          IsAlone
count  891.000000
mean     0.602694
std      0.489615
min      0.000000
25%      0.000000
50%      1.000000
75%      1.000000
max      1.000000
```

[142]:
```
def evaluate_models(data):
    rf_res  = k_fold_validation(k=5, X=data, model=RFclf)
    dt_res  = k_fold_validation(k=5, X=data, model=DTclf)
    xgb_res = k_fold_validation(k=5, X=data, model=boost_clf)
    return {"DT": dt_res, "RF": rf_res, "xgb": xgb_res}
```

[143]:
```
models_results = evaluate_models(X.drop("Survived", axis=1))
```

```
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
```

```
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
/tmp/ipykernel_95041/23553612.py:14: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  model.fit(X_train_fold,y_train_fold)
```

[144]:
```
models_results
```

[144]:
```
{'DT': (0.80585022911305, DecisionTreeClassifier()),
 'RF': (0.8092272926997678, RandomForestClassifier()),
 'xgb': (0.8159751428033394,
  XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                early_stopping_rounds=None, enable_categorical=False,
                eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
                importance_type=None, interaction_constraints='',
                learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
                max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
                missing=nan, monotone_constraints='()', n_estimators=100,
                n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
                reg_alpha=0, reg_lambda=1, …))}
```

[151]:
```
df_test
```

[151]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
PassengerId							
892	3	0	34.500000	0	0	7.8292	2
893	3	1	47.000000	1	0	7.0000	0
894	2	0	62.000000	0	0	9.6875	2
895	3	0	27.000000	0	0	8.6625	0
896	3	1	22.000000	1	1	12.2875	0
…	…	…	…	…	…	…	…
1305	3	0	29.699118	0	0	8.0500	0
1306	1	1	39.000000	0	0	108.9000	1
1307	3	0	38.500000	0	0	7.2500	0
1308	3	0	29.699118	0	0	8.0500	0
1309	3	0	29.699118	1	1	22.3583	1

[418 rows x 7 columns]

20 Let's update our snipet to train

[148]:
```
# need to define functions for age binning
def age_band(data):
    data['AgeBand'] = pd.cut(data['Age'], 5)
    data['Age'] = le.fit_transform(data['AgeBand'])
    data.drop("AgeBand", axis=1, inplace=True)
    return data

def fare_band(data):
    data['FareBand'] = pd.qcut(X['Fare'], 4)
    data['Fare'] = le.fit_transform(data['FareBand'])
    data.drop("FareBand", axis=1, inplace=True)
    return data

def create_is_alone(data):
    data['FamilySize'] = data['SibSp'] + data['Parch'] + 1
    data['IsAlone'] = 0
    data.loc[data['FamilySize'] == 1, 'IsAlone'] = 1
    return data
```

[156]:
```
class PredictPipeline:
    @staticmethod
    def predict(predict_data, population_average,␣
 ↪population_median,cols_to_remove, model):
        # preprocessing
        data = age_avg_filling(data=predict_data,␣
 ↪population_average=population_average)
        data = fare_median_filling(data=data,␣
 ↪population_median=population_median)

        # feature engineering
        data = label_encode(data=data, col_to_encode="Sex")
        data = label_encode(data=data, col_to_encode="Embarked")

        data = age_band(data=data)
        data = fare_band(data=data)
        data = create_is_alone(data=data)

        # remove columns
        data = remove_columns(data=data, cols=cols_to_remove)

        return model.predict(data)

    @staticmethod
    def write_results_to_disk(test_data, predictions, csv_name):
        data = test_data.copy(deep=True)
```

```
        data["Survived"] = predictions
        data = data[["Survived"]]
        data.reset_index(inplace=True)
        data.to_csv(csv_name, index=False)
        print(f"Successfully written {csv_name}")
```

[157]:
```
# generate results
results = PredictPipeline.predict(predict_data=df_test,
                        population_average=age_mean,
                        population_median=fare_median,
                        cols_to_remove = ["FamilySize", "SibSp", "Parch"],
                        model=models_results["xgb"][1])
# write predictions
PredictPipeline.write_results_to_disk(test_data=df_test,
                                      predictions=results,
                                      csv_name="submission_xgboost.csv")
```

```
Successfully written submission_xgboost.csv
```

[ ]:
```

```