Lab 9: Pre-Processing with Scikit-Learn and Pandas

The objective of this notebook is to learn about **pre-processing** with the **Scikit-Learn** and **Pandas** libraries. Then, train a simple binary classifier on the pre-processed dataset.

In this lab, we will train a binary classification model that predicts which **passengers survived** the **Titanic shipwreck** link.

The sinking of the Titanic is one of the most infamous shipwrecks in history.

On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this notebook, you are asked to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

You can find a detailed **tutorial** here.

Outline

First, run the following cell to import some useful libraries to complete this Lab. If not already done, you must install them in your virtual environment

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

pd.options.display.max_columns= 50
pd.options.display.max_rows= None

import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml

from sklearn.impute import SimpleImputer
```

1. Load dataset

Firstly, you will load the **Titanic** dataset used in this lab into a DataFrame `df` .

Scikit-Learn comes with built-in datasets for the **Titanic dataset**. The next cell loads the titanic dataset from Scikit-Learn and stores it in a Pandas DataFrame.

In [2]:
```
df, y = fetch_openml('titanic', version=1, as_frame=True, parser='auto', ret
df["survived"] = y
```

Run the next cell to look at the first 5 rows of the dataset.

In [3]:
```
df.head()
```

Out[3]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boa
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	1
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN

In [4]:
```
print("Number of samples:", len(df))
```

```
Number of samples: 1309
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['pclass', 'name', 'sex', 'age', 'sibsp', 'parch', 'ticket', 'fare',
       'cabin', 'embarked', 'boat', 'body', 'home.dest', 'survived'],
      dtype='object')
```

The dataset is composed of 1309 samples. Each row contains information on each passenger. Specifically, the dataset contains the following attributes:

- **pclass**: Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- **name**: Passenger name
- **sex**: Passenger sex
- **age**: Passenger age
- **sibsp**: Number of Siblings/Spouses Aboard
- **parch**: Number of Parents/Children Aboard

- **ticket**: Ticket Number
- **fare**: Passenger Fare
- **cabin**: Cabin
- **embarked**: Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- **boat**: Lifeboat (if survived)
- **body**: Body number (if did not survive and body was recovered). It could be another target.
- **home.dest**: Destination
- **survival** (target): Survival (0 = No; 1 = Yes)

Note that **boat** and **body** must be removed from input features because provide information about the target variable (i.e., they have values only if target is survived).

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype   
---  ------     --------------  -----   
 0   pclass     1309 non-null   int64   
 1   name       1309 non-null   object  
 2   sex        1309 non-null   category
 3   age        1046 non-null   float64 
 4   sibsp      1309 non-null   int64   
 5   parch      1309 non-null   int64   
 6   ticket     1309 non-null   object  
 7   fare       1308 non-null   float64 
 8   cabin      295 non-null    object  
 9   embarked   1307 non-null   category
 10  boat       486 non-null    object  
 11  body       121 non-null    float64 
 12  home.dest  745 non-null    object  
 13  survived   1309 non-null   category
dtypes: category(3), float64(3), int64(3), object(5)
memory usage: 116.8+ KB
```

In [7]:
```
df.describe()
```

Out[7]:

	pclass	age	sibsp	parch	fare	body
count	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	121.000000
mean	2.294882	29.881135	0.498854	0.385027	33.295479	160.809917
std	0.837836	14.413500	1.041658	0.865560	51.758668	97.696922
min	1.000000	0.166700	0.000000	0.000000	0.000000	1.000000
25%	2.000000	21.000000	0.000000	0.000000	7.895800	72.000000
50%	3.000000	28.000000	0.000000	0.000000	14.454200	155.000000
75%	3.000000	39.000000	1.000000	0.000000	31.275000	256.000000
max	3.000000	80.000000	8.000000	9.000000	512.329200	328.000000

2. Data pre-processing

Firstly, you will perform the pre-processing of the dataset.

2.1 Train and Test splitting with Stratification

In [8]:
```
df["survived"].value_counts()
```

Out[8]:
```
0    809
1    500
Name: survived, dtype: int64
```

The dataset is a slightly **imbalance**.

In [9]:
```
df.head()
```

Out[9]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boa
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	:
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	1
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	Nal
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	Nal
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	Nal

Exercise 2.6.1

Extract the input features in `X` and the target values in `y`.

In [10]:
```
#### START CODE HERE ####
#### Approximately 2 line ####
X = df.loc[:, "pclass":"home.dest"]
y = df['survived']
#### END CODE HERE ####
```

In [11]:
```
X.head()
```

Out[11]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boa
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	1
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	Nal
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	Nal
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	Nal

Exercise 2.1.2

Split the dataset into **train** and **test**. In this case, the dataset is **imbalance**. Therefore, it is recommended to split using stratification (i.e., the class label distribution will be preserved during the splitting).

Split with 80% for training and 20% for validation. Shuffle the dataset before splitting.

In [12]:
```
#### START CODE HERE ####
#### Approximately 1 line ####

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shu

#### END CODE HERE ####
```

In [13]:
```
print(f"Number of training examples: {len(X_train)}")
print(f"Number of testing examples: {len(X_test)}")
```

```
Number of training examples: 1047
Number of testing examples: 262
```

2.2 Handling missing values

Exercise 2.2.1

Count the number of **null values** in training and test, and store them in the variables `nan_count_train` and `nan_count_test`.

In [14]:
```
#### START CODE HERE ####
#### Approximately 2 line ####

nan_count_train = X_train.isna().sum()
```

```
nan_count_test = X_test.isna().sum()

#### END CODE HERE ####
```

In [15]:
```
print("Train")
print(nan_count_train)
```

```
Train
pclass         0
name           0
sex            0
age          209
sibsp          0
parch          0
ticket         0
fare           1
cabin        822
embarked       0
boat         658
body         955
home.dest    450
dtype: int64
```

In [16]:
```
print("Test")
print(nan_count_test)
```

```
Test
pclass         0
name           0
sex            0
age           54
sibsp          0
parch          0
ticket         0
fare           0
cabin        192
embarked       2
boat         165
body         233
home.dest    114
dtype: int64
```

Sometimes, the **missing values** are not in the *nan* format.

The next cell prints the format of *nan* values.

In [17]:
```
print('Data types of missing values')
for col in X_train.columns[X_train.isnull().any()]:
    print(col, X_train[col][X_train[col].isnull()].values[0])
```

```
Data types of missing values
age nan
fare nan
cabin nan
boat nan
body nan
home.dest nan
```

In this case, all *nan* values are in the *nan* format.

Exercise 2.2.2

Fill **null values** in the column `age` with the **mean** of the column `age` in the training and test set. Please compute the mean only on the training!

In [18]:
```
print(f'Number of null values in Train before pre-processing: {X_train.age.i
print(f'Number of null values in Test before pre-processing: {X_test.age.isn

#### START CODE HERE ####
#### Approximately 1 line ####

X_train['age'].fillna(X_train['age'].mean(), inplace=True)
X_test['age'].fillna(X_train['age'].mean(), inplace=True)

#### END CODE HERE ####

print(f'Number of null values in Train after pre-processing: {X_train.age.is
print(f'Number of null values in Test after pre-processing: {X_test.age.isnu
```

```
Number of null values in Train before pre-processing: 209/1047
Number of null values in Test before pre-processing: 54/262
Number of null values in Train after pre-processing: 0/1047
Number of null values in Test after pre-processing: 0/262
```

Exercise 2.2.3

Fill **null values** in the column `fare` with the **median** of the column `fare` in the training and test set. Please compute the median only on the training!

In [19]:
```
print(f'Number of null values in Train before pre-processing: {X_train.fare.
print(f'Number of null values in Test before pre-processing: {X_test.fare.is

#### START CODE HERE ####
#### Approximately 1 line ####

X_train['fare'].fillna(X_train['fare'].median(), inplace=True)
X_test['fare'].fillna(X_train['fare'].median(), inplace=True)

#### END CODE HERE ####

print(f'Number of null values in Train after pre-processing: {X_train.fare.i
print(f'Number of null values in Test after pre-processing: {X_test.fare.isn
```

```
Number of null values in Train before pre-processing: 1/1047
Number of null values in Test before pre-processing: 0/262
Number of null values in Train after pre-processing: 0/1047
Number of null values in Test after pre-processing: 0/262
```

Exercise 2.2.4

Fill **null values** in the column `embarked` with the **most frequent value** of the column `embarked`. Please compute the most frequent only on the training!

In [20]:
```
print(f'Number of null values in Train before pre-processing: {X_train.embar
print(f'Number of null values in Test before pre-processing: {X_test.embarke

#### START CODE HERE ####
#### Approximately 3 line ####

imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
X_train['embarked'] = imp.fit_transform(X_train[['embarked']])
X_test['embarked'] = imp.transform(X_test[['embarked']])
```

```
#### END CODE HERE ####

print(f'Number of null values in Train after pre-processing: {X_train.embark
print(f'Number of null values in Test after pre-processing: {X_test.embarked
```

```
Number of null values in Train before pre-processing: 0/1047
Number of null values in Test before pre-processing: 2/262
Number of null values in Train after pre-processing: 0/1047
Number of null values in Test after pre-processing: 0/262
```

2.3 Features selection

Exercise 2.3.1

Remove columns *cabin*, *body*, *boat*, and *home.dest* from the train and test sets because they contain info about the target variable (i.e., the model could "cheat" predicting the target label based on the info in these attributes).

In [21]:
```
#### START CODE HERE ####
#### Approximately 2 line ####

X_train = X_train.drop(columns=['cabin', 'body', 'boat', 'home.dest'])
X_test = X_test.drop(columns=['cabin', 'body', 'boat', 'home.dest'])

#### END CODE HERE ####

X_train.head()
```

Out[21]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	embarked
999	3	McCarthy, Miss. Catherine 'Katie'	female	29.604316	0	0	383123	7.7500	Q
392	2	del Carlo, Mrs. Sebastiano (Argenia Genovesi)	female	24.000000	1	0	SC/PARIS 2167	27.7208	C
628	3	Andersson, Miss. Sigrid Elisabeth	female	11.000000	4	2	347082	31.2750	S
1165	3	Saad, Mr. Khalil	male	25.000000	0	0	2672	7.2250	C
604	3	Abelseth, Miss. Karen Marie	female	16.000000	0	0	348125	7.6500	S

Exercise 2.3.2

Remove other columns that you think are useless features in predicting which people were more likely to survive.

In [22]:
```
#### START CODE HERE ####
#### Approximately 2 line ####
```

```
X_train = X_train.drop(columns=['name','ticket'])
X_test = X_test.drop(columns=['name','ticket'])

#### END CODE HERE ####

X_train.head()
```

Out[22]:

	pclass	sex	age	sibsp	parch	fare	embarked
999	3	female	29.604316	0	0	7.7500	Q
392	2	female	24.000000	1	0	27.7208	C
628	3	female	11.000000	4	2	31.2750	S
1165	3	male	25.000000	0	0	7.2250	C
604	3	female	16.000000	0	0	7.6500	S

The next cell plots the **correlation heatmap** using `Seaborn` and `df.corr()`. You will probably need to install `Seaborn` using the command `pip install seaborn`, and then restart your kernel.

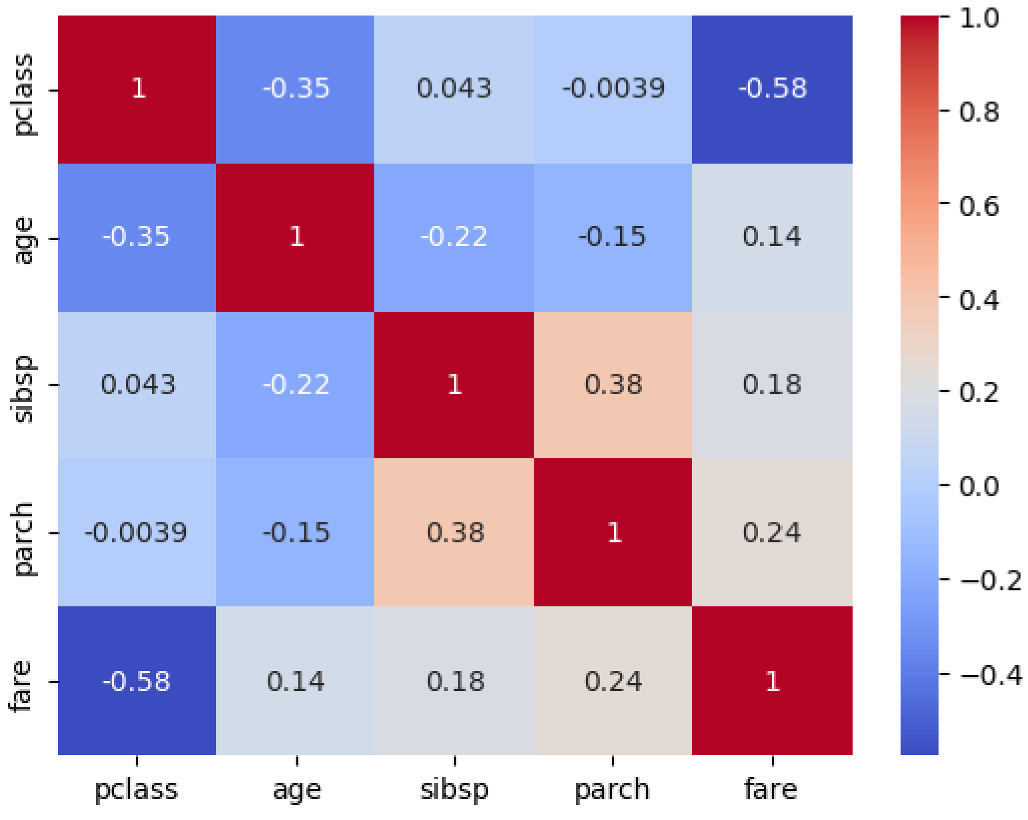
In [23]:

```
import seaborn as sns
#%config InlineBackend.figure_format = 'svg'

df_show = pd.concat([X_train[['pclass', 'sex', 'age', 'sibsp', 'parch', 'far

g = sns.heatmap(df_show.corr(),
                annot=True,
                cmap = "coolwarm")
```

```
/var/folders/ck/5bn3d96976q9mdgwzsdcxtmw0000gn/T/ipykernel_38416/3734267867.
py:6: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  g = sns.heatmap(df_show.corr(),
```



2.4 Features engineering (optional)

Exercise 2.4.1

If you want, you can create new columns here from the ones available.

In [24]:
```
#### START CODE HERE ####

#### END CODE HERE ####

X_train.head()
```

Out[24]:

	pclass	sex	age	sibsp	parch	fare	embarked
999	3	female	29.604316	0	0	7.7500	Q
392	2	female	24.000000	1	0	27.7208	C
628	3	female	11.000000	4	2	31.2750	S
1165	3	male	25.000000	0	0	7.2250	C
604	3	female	16.000000	0	0	7.6500	S

2.5 Discretization

The next cell performs the **discretization** of the age column with **fixed-intervals**. You can learn more about **discretization** here.

In [25]:
```
age_category = ['Child (0-14]', 'Young (14-24]', 'Adults (24-50]', 'Senior (

X_train['age_disc']=pd.cut(x=X_train['age'], bins=[0,14,24,50,100],labels=ag
X_train = X_train.drop(columns=['age']) # Remove the old age column

X_test['age_disc']=pd.cut(x=X_test['age'], bins=[0,14,24,50,100],labels=age_
X_test = X_test.drop(columns=['age']) # Remove the old age column
```

In [26]:
```
X_train.head()
```

Out[26]:

	pclass	sex	sibsp	parch	fare	embarked	age_disc
999	3	female	0	0	7.7500	Q	Adults (24-50]
392	2	female	1	0	27.7208	C	Young (14-24]
628	3	female	4	2	31.2750	S	Child (0-14]
1165	3	male	0	0	7.2250	C	Adults (24-50]
604	3	female	0	0	7.6500	S	Young (14-24]

In [27]:
```
X_test.head()
```

Out[27]:

	pclass	sex	sibsp	parch	fare	embarked	age_disc
1028	3	female	1	0	24.1500	Q	Adults (24-50]
1121	3	male	1	1	22.3583	C	Adults (24-50]
1155	3	male	0	0	7.7750	S	Adults (24-50]
1251	3	male	0	0	8.0500	S	Adults (24-50]
721	3	male	0	0	7.4958	S	Adults (24-50]

2.7 One-hot encoding

The following cells perform the **one-hot encoding** of the categorical features using the `OneHotEncoder` of the **Scikit-Learn** library. You can also use a similar approach using the `get_dummies` function of **Pandas**.

You can learn the differences between `OneHotEncoder` and `get_dummies` here.

When building the `OneHotEncoder` object, the `handle_unknown` parameter is set to `'ignore'`.

In [28]:
```
X_train.head()
```

Out[28]:

	pclass	sex	sibsp	parch	fare	embarked	age_disc
999	3	female	0	0	7.7500	Q	Adults (24-50]
392	2	female	1	0	27.7208	C	Young (14-24]
628	3	female	4	2	31.2750	S	Child (0-14]
1165	3	male	0	0	7.2250	C	Adults (24-50]
604	3	female	0	0	7.6500	S	Young (14-24]

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

ohe = OneHotEncoder(handle_unknown='ignore')
```

In [30]:
```
categorical_columns = ['sex', 'embarked']
```

In [31]:
```
ohe.fit(X_train[categorical_columns]) # Fit on training data

temp_df = pd.DataFrame(data=ohe.transform(X_train[categorical_columns]).toar
                       columns=ohe.get_feature_names_out()) # Create a new D

X_train.drop(columns=categorical_columns, axis=1, inplace=True) # Remove the
X_train = pd.concat([X_train.reset_index(drop=True), temp_df], axis=1)

X_train.head()
```

Out[31]:

	pclass	sibsp	parch	fare	age_disc	sex_female	sex_male	embarked_C	embarked_
0	3	0	0	7.7500	Adults (24-50]	1.0	0.0	0.0	1.
1	2	1	0	27.7208	Young (14-24]	1.0	0.0	1.0	0.
2	3	4	2	31.2750	Child (0-14]	1.0	0.0	0.0	0.
3	3	0	0	7.2250	Adults (24-50]	0.0	1.0	1.0	0.
4	3	0	0	7.6500	Young (14-24]	1.0	0.0	0.0	0.

In [32]:

```
temp_df = pd.DataFrame(data=ohe.transform(X_test[categorical_columns]).toarr
                       columns=ohe.get_feature_names_out()) # Not fit on tes

X_test.drop(columns=categorical_columns, axis=1, inplace=True)
X_test = pd.concat([X_test.reset_index(drop=True), temp_df], axis=1)

X_test.head()
```

Out[32]:

	pclass	sibsp	parch	fare	age_disc	sex_female	sex_male	embarked_C	embarked_
0	3	1	0	24.1500	Adults (24-50]	1.0	0.0	0.0	1
1	3	1	1	22.3583	Adults (24-50]	0.0	1.0	1.0	0
2	3	0	0	7.7750	Adults (24-50]	0.0	1.0	0.0	0
3	3	0	0	8.0500	Adults (24-50]	0.0	1.0	0.0	0
4	3	0	0	7.4958	Adults (24-50]	0.0	1.0	0.0	0

2.7 Ordinal Encoding

When the categorical feature is ordinal we can use ordinal Encoding. Since the order among the categories is important, encoding should reflect the sequence.

In [33]:

```
age_category
```

Out[33]:

```
['Child (0-14]', 'Young (14-24]', 'Adults (24-50]', 'Senior (50+]']
```

In [34]:

```
from sklearn.preprocessing import OrdinalEncoder

ord_enc = OrdinalEncoder(categories=[age_category]) # Should be a list becua


ord_enc.fit(X_train.loc[:, ["age_disc"]]) # Fit on training data

ord_enc
```

Out[34]:

```
▾ OrdinalEncoder
OrdinalEncoder(categories=[['Child (0-14]', 'Young (14-24]', 'Adult
s (24-50]',
                            'Senior (50+]']])
```

In [35]:

```
X_train["age_disc_enc"] = ord_enc.transform(X_train.loc[:, ["age_disc"]])

X_train.head()
```

Out[35]:

	pclass	sibsp	parch	fare	age_disc	sex_female	sex_male	embarked_C	embarked_
0	3	0	0	7.7500	Adults (24-50]	1.0	0.0	0.0	1.
1	2	1	0	27.7208	Young (14-24]	1.0	0.0	1.0	0.
2	3	4	2	31.2750	Child (0-14]	1.0	0.0	0.0	0.
3	3	0	0	7.2250	Adults (24-50]	0.0	1.0	1.0	0.
4	3	0	0	7.6500	Young (14-24]	1.0	0.0	0.0	0.

In [36]:

```
X_train.drop(columns=["age_disc"], axis=1, inplace=True)

X_train.head()
```

Out[36]:

	pclass	sibsp	parch	fare	sex_female	sex_male	embarked_C	embarked_Q	embark
0	3	0	0	7.7500	1.0	0.0	0.0	1.0	
1	2	1	0	27.7208	1.0	0.0	1.0	0.0	
2	3	4	2	31.2750	1.0	0.0	0.0	0.0	
3	3	0	0	7.2250	0.0	1.0	1.0	0.0	
4	3	0	0	7.6500	1.0	0.0	0.0	0.0	

In [37]:

```
X_test["age_disc_enc"] = ord_enc.transform(X_test.loc[:, ["age_disc"]])
X_test.drop(columns=["age_disc"], axis=1, inplace=True)

X_test.head()
```

Out[37]:

	pclass	sibsp	parch	fare	sex_female	sex_male	embarked_C	embarked_Q	embark
0	3	1	0	24.1500	1.0	0.0	0.0	1.0	
1	3	1	1	22.3583	0.0	1.0	1.0	0.0	
2	3	0	0	7.7750	0.0	1.0	0.0	0.0	
3	3	0	0	8.0500	0.0	1.0	0.0	0.0	
4	3	0	0	7.4958	0.0	1.0	0.0	0.0	

2.8 Normalization/Standardization

Exercise 2.8.1

Perform **Min-Max** normalization of the *numerical features*. Remember to **fit** on the training and not on the test. Note that `age_disc_enc` in this case is categorical but can be normalized too.

In [38]:
```
from sklearn.preprocessing import MinMaxScaler

numerical_features = ["pclass", "sibsp", "parch", "fare", "age_disc_enc"]

#### START CODE HERE ####
#### Approximately 4 line ####

minmax_s = MinMaxScaler()

minmax_s.fit(X_train[numerical_features])

X_train[numerical_features] = minmax_s.transform(X_train[numerical_features]
X_test[numerical_features] = minmax_s.transform(X_test[numerical_features])

#### END CODE HERE ####
```

In [39]:
```
X_train.head()
```

Out[39]:

	pclass	sibsp	parch	fare	sex_female	sex_male	embarked_C	embarked_Q	em
0	1.0	0.000	0.000000	0.015127	1.0	0.0	0.0	1.0	
1	0.5	0.125	0.000000	0.054107	1.0	0.0	1.0	0.0	
2	1.0	0.500	0.222222	0.061045	1.0	0.0	0.0	0.0	
3	1.0	0.000	0.000000	0.014102	0.0	1.0	1.0	0.0	
4	1.0	0.000	0.000000	0.014932	1.0	0.0	0.0	0.0	

In [40]:
```
X_test.head()
```

Out[40]:

	pclass	sibsp	parch	fare	sex_female	sex_male	embarked_C	embarked_Q	en
0	1.0	0.125	0.000000	0.047138	1.0	0.0	0.0	1.0	
1	1.0	0.125	0.111111	0.043640	0.0	1.0	1.0	0.0	
2	1.0	0.000	0.000000	0.015176	0.0	1.0	0.0	0.0	
3	1.0	0.000	0.000000	0.015713	0.0	1.0	0.0	0.0	
4	1.0	0.000	0.000000	0.014631	0.0	1.0	0.0	0.0	

3. Model Training and Evaluation

Now, you can **train** and **evaluate** a **binary classification** model on the pre-processed dataset.

3.1 Training

In [ ]:
```
```

3.2 Evaluation

In [ ]: