titanic

April 3, 2025

[150]:
```
import numpy as np
import pandas as pd
```

[151]:
```
raw_train = pd.read_csv("train.csv")
```

[152]:
```
raw_train.head()
```

[152]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

[153]:
```
raw_train.shape
```

[153]: (891, 12)

[154]:
```
raw_train.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
```

[155]:
```
raw_train.describe()
```

[155]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

[156]:
```
all_cols = list(raw_train)
num_data = [i for i in list(raw_train) if raw_train[i].dtypes != "object"]
cat_data = [i for i in list(raw_train) if raw_train[i].dtypes == "object"]
print(f"The numerical attributes are: {num_data}")
```

```
The numerical attributes are: ['PassengerId', 'Survived', 'Pclass', 'Age',
'SibSp', 'Parch', 'Fare']
```

Dividing the dataset into numerical dataset and categorical dataset

[157]:
```
raw_copied = raw_train.copy()
num_dataset = raw_train.copy()
cat_dataset = raw_train.copy()
for i in cat_data:
    num_dataset.drop(columns=i, inplace=True)
for i in num_data:
    cat_dataset.drop(columns=i, inplace=True)
```

[158]:
```
num_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 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   Age          714 non-null    float64
 4   SibSp        891 non-null    int64  
 5   Parch        891 non-null    int64  
 6   Fare         891 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
```

[159]:
```
cat_dataset
```

[159]:

	Name	Sex	Ticket	Cabin	Embarked
0	Braund, Mr. Owen Harris	male	A/5 21171	NaN	S
1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	PC 17599	C85	C
2	Heikkinen, Miss. Laina	female	STON/O2. 3101282	NaN	S
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	C123	S
4	Allen, Mr. William Henry	male	373450	NaN	S
..	…	…	…	…	…
886	Montvila, Rev. Juozas	male			
887	Graham, Miss. Margaret Edith	female			
888	Johnston, Miss. Catherine Helen "Carrie"	female			
889	Behr, Mr. Karl Howell	male			
890	Dooley, Mr. Patrick	male			

```
886            211536   NaN        S
887            112053   B42        S
888        W./C. 6607   NaN        S
889            111369  C148        C
890            370376   NaN        Q

[891 rows x 5 columns]
```

0.0.1 Convert sex to integer counterparts for more efficient data processing by model

[160]:
```
for i in range (len(cat_dataset["Sex"])):
    if cat_dataset["Sex"].iloc[i] == 'male':
        cat_dataset["Sex"].iloc[i] = np.float32(1)
    else:
        cat_dataset["Sex"].iloc[i] = np.float32(0)
```

```
C:\Users\dasas\AppData\Local\Temp\ipykernel_12380\1782178432.py:3:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  cat_dataset["Sex"].iloc[i] = np.float32(1)
C:\Users\dasas\AppData\Local\Temp\ipykernel_12380\1782178432.py:5:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  cat_dataset["Sex"].iloc[i] = np.float32(0)
```

[161]:
```
cat_dataset["Sex"] = cat_dataset["Sex"].astype(float)
cat_dataset
```

[161]:

	Name	Sex	Ticket	\
0	Braund, Mr. Owen Harris	1.0	A/5 21171	
1	Cumings, Mrs. John Bradley (Florence Briggs Th…	0.0	PC 17599	
2	Heikkinen, Miss. Laina	0.0	STON/O2. 3101282	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0.0	113803	
4	Allen, Mr. William Henry	1.0	373450	
..	…	…	…	
886	Montvila, Rev. Juozas	1.0	211536	
887	Graham, Miss. Margaret Edith	0.0	112053	
888	Johnston, Miss. Catherine Helen "Carrie"	0.0	W./C. 6607	
889	Behr, Mr. Karl Howell	1.0	111369	
890	Dooley, Mr. Patrick	1.0	370376	

	Cabin	Embarked
0	NaN	S
1	C85	C
2	NaN	S
3	C123	S
4	NaN	S
..	…	…
886	NaN	S
887	B42	S
888	NaN	S
889	C148	C
890	NaN	Q

[891 rows x 5 columns]

[162]:
```
num_dataset = num_dataset.join(cat_dataset["Sex"])
num_dataset
```

[162]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex
0	1	0	3	22.0	1	0	7.2500	1.0
1	2	1	1	38.0	1	0	71.2833	0.0
2	3	1	3	26.0	0	0	7.9250	0.0
3	4	1	1	35.0	1	0	53.1000	0.0

```
4              5         0       3  35.0      0      0   8.0500  1.0
..           ...       ...     ...   ...    ...    ...      ...  ...
886          887         0       2  27.0      0      0  13.0000  1.0
887          888         1       1  19.0      0      0  30.0000  0.0
888          889         0       3   NaN      1      2  23.4500  0.0
889          890         1       1  26.0      0      0  30.0000  1.0
890          891         0       3  32.0      0      0   7.7500  1.0

[891 rows x 8 columns]
```

[163]:
```
cat_dataset.drop(columns=["Sex"], inplace=True)
# cat_dataset  #print the dataset to see the desired output if necessary
```

Whenever conversion is happening from pandas dataframe to numpy array the data type changes to object for the entire array because numpy array can only have 1 datatype and mixture of datatypes forces numpy to cast all the data to object type To avoid this I did not create a class for sex datatype conversion and did the conversion utilizing pandas and numpy

construct the correlation matrix for the transformed data

[164]:
```
corr_matrix = num_dataset.corr(numeric_only=True)
corr_matrix["Survived"].sort_values(ascending=False)
```

[164]:
```
Survived       1.000000
Fare           0.257307
Parch          0.081629
PassengerId   -0.005007
SibSp         -0.035322
Age           -0.077221
Pclass        -0.338481
Sex           -0.543351
Name: Survived, dtype: float64
```

0.0.2 Transform the Embarked column with OneHotEncoder

The approach I am using is:

- Firstly clean the data by filling the nan columns
- To fill the nan values i am using the most frequent value of the column

[165]:
```
cat_dataset['Embarked']
```

[165]:
```
0      S
1      C
2      S
3      S
4      S
      ..
886    S
```

```
887    S
888    S
889    C
890    Q
Name: Embarked, Length: 891, dtype: object
```

[166]:
```
cat_dataset['Embarked'].value_counts().idxmax()
```

[166]:
```
'S'
```

class to fill the nan values present in Embarked column

[167]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class FillnaEmbarked(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        X["Embarked"] = X["Embarked"].fillna(X["Embarked"].value_counts().
 ↪idxmax())
        return X.to_numpy()
```

[168]:
```
fillna_embark = FillnaEmbarked()
raw_train_embark_arr = fillna_embark.transform(cat_dataset)
raw_train_embark_filled = pd.DataFrame(raw_train_embark_arr,␣
 ↪columns=list(cat_dataset))
raw_train_embark_filled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      891 non-null    object
 1   Ticket    891 non-null    object
 2   Cabin     204 non-null    object
 3   Embarked  891 non-null    object
dtypes: object(4)
memory usage: 28.0+ KB
```

seperate the embarked values using OneHotEncoder for ease of model

[169]:
```
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder()
encoded_embarked = one_hot_encoder.
 ↪fit_transform(raw_train_embark_filled[["Embarked"]])
```

[170]:
```
encoded_embarked.toarray()
```

[170]:
```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       …,
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

Steps to add the OneHotEncoded values to the original dataset

- extract the categories to a python list in the created one_hot_encoder object
- construct a pandas dataset with the one_hot_encoder values
- join the previously present numerical dataset and the newly formed one hot encoded embarked dataset

[171]:
```
one_hot_columns = []
for i in one_hot_encoder.categories_[0]:
    one_hot_columns.append(i)
    print(i)
print(one_hot_columns)
```

```
C
Q
S
['C', 'Q', 'S']
```

[172]:
```
embarked_dataset = pd.DataFrame(encoded_embarked.toarray(),␣
 ↪columns=one_hot_columns)
embarked_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   C       891 non-null    float64
 1   Q       891 non-null    float64
 2   S       891 non-null    float64
dtypes: float64(3)
memory usage: 21.0 KB
```

[173]:
```
num_dataset = num_dataset.join(embarked_dataset)
num_dataset
```

[173]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex	C	\
0	1	0	3	22.0	1	0	7.2500	1.0	0.0	

```
1              2         1       1  38.0      1      0  71.2833  0.0  1.0
2              3         1       3  26.0      0      0   7.9250  0.0  0.0
3              4         1       1  35.0      1      0  53.1000  0.0  0.0
4              5         0       3  35.0      0      0   8.0500  1.0  0.0
..           …       …     …   …    …    …       …  …  …
886          887         0       2  27.0      0      0  13.0000  1.0  0.0
887          888         1       1  19.0      0      0  30.0000  0.0  0.0
888          889         0       3   NaN      1      2  23.4500  0.0  0.0
889          890         1       1  26.0      0      0  30.0000  1.0  1.0
890          891         0       3  32.0      0      0   7.7500  1.0  0.0

       Q    S
0    0.0  1.0
1    0.0  0.0
2    0.0  1.0
3    0.0  1.0
4    0.0  1.0
..   …  …
886  0.0  1.0
887  0.0  1.0
888  0.0  1.0
889  0.0  0.0
890  1.0  0.0

[891 rows x 11 columns]
```

[174]:
```
raw_train_embark_filled.drop(columns=["Embarked"], inplace=True)
raw_train_embark_filled
```

[174]:
```
                                                Name            Ticket Cabin
0                            Braund, Mr. Owen Harris         A/5 21171   NaN
1    Cumings, Mrs. John Bradley (Florence Briggs Th…          PC 17599   C85
2                               Heikkinen, Miss. Laina  STON/O2. 3101282   NaN
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)            113803  C123
4                             Allen, Mr. William Henry            373450   NaN
..                                                 …               …   …
886                             Montvila, Rev. Juozas            211536   NaN
887                      Graham, Miss. Margaret Edith            112053   B42
888          Johnston, Miss. Catherine Helen "Carrie"        W./C. 6607   NaN
889                             Behr, Mr. Karl Howell            111369  C148
890                               Dooley, Mr. Patrick            370376   NaN

[891 rows x 3 columns]
```

[175]:
```
corr_mat = num_dataset.corr()
corr_matrix["Survived"].sort_values()
```

[175]:
```
Sex           -0.543351
Pclass        -0.338481
Age           -0.077221
SibSp         -0.035322
PassengerId   -0.005007
Parch          0.081629
Fare           0.257307
Survived       1.000000
Name: Survived, dtype: float64
```

0.0.3 Split the data into test set and train set

as we see in the last correlation matrix that the survived column has very high dependency on the sex feature so we are dividing the dataset based on the sex parameter

split the data in a ratio of 7:3

[176]:
```
from sklearn.model_selection import StratifiedShuffleSplit
train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.3,␣
 ↪random_state=42)
for train_index, test_index in train_test_split.split(num_dataset,␣
 ↪num_dataset["Sex"]):
    X_train = num_dataset.loc[train_index]
    X_test = num_dataset.loc[test_index]
```

extract the labels form both the train set and the test set

[177]:
```
y_train = X_train["Survived"]
y_test = X_test["Survived"]
```

[178]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 623 entries, 97 to 421
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  623 non-null    int64
 1   Survived     623 non-null    int64
 2   Pclass       623 non-null    int64
 3   Age          507 non-null    float64
 4   SibSp        623 non-null    int64
 5   Parch        623 non-null    int64
 6   Fare         623 non-null    float64
 7   Sex          623 non-null    float64
 8   C            623 non-null    float64
 9   Q            623 non-null    float64
 10  S            623 non-null    float64
```

```
dtypes: float64(6), int64(5)
memory usage: 58.4 KB
```

[179]:
```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 268 entries, 154 to 563
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  268 non-null    int64
 1   Survived     268 non-null    int64
 2   Pclass       268 non-null    int64
 3   Age          207 non-null    float64
 4   SibSp        268 non-null    int64
 5   Parch        268 non-null    int64
 6   Fare         268 non-null    float64
 7   Sex          268 non-null    float64
 8   C            268 non-null    float64
 9   Q            268 non-null    float64
 10  S            268 non-null    float64
dtypes: float64(6), int64(5)
memory usage: 25.1 KB
```

0.0.4 Tackle the null values present in the age column

i am using simple imputer of sklearn with strategy as median for the age nan values

[180]:
```
X_train
```

[180]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex	C	Q	S
97	98	1	1	23.0	0	1	63.3583	1.0	1.0	0.0	0.0
198	199	1	3	NaN	0	0	7.7500	0.0	0.0	1.0	0.0
10	11	1	3	4.0	1	1	16.7000	0.0	0.0	0.0	1.0
808	809	0	2	39.0	0	0	13.0000	1.0	0.0	0.0	1.0
206	207	0	3	32.0	1	0	15.8500	1.0	0.0	0.0	1.0
..	…	…	…	…	…	…	…	…	…	…	…
131	132	0	3	20.0	0	0	7.0500	1.0	0.0	0.0	1.0
692	693	1	3	NaN	0	0	56.4958	1.0	0.0	0.0	1.0
231	232	0	3	29.0	0	0	7.7750	1.0	0.0	0.0	1.0
870	871	0	3	26.0	0	0	7.8958	1.0	0.0	0.0	1.0
421	422	0	3	21.0	0	0	7.7333	1.0	0.0	0.0	1.0

```
..   …  …
131  0.0  1.0
692  0.0  1.0
231  0.0  1.0
870  0.0  1.0
421  1.0  0.0

[623 rows x 11 columns]
```

[181]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
imputer.fit(X_train)
imputed_train_dataset = imputer.transform(X_train)
```

[182]:
```
imputer.statistics_
```

[182]:
```
array([458. ,   0. ,   3. ,  28. ,   0. ,   0. ,  14.5,   1. ,   0. ,
         0. ,   1. ])
```

filled all null and nan values in the age column with the meadian of the whole column

[183]:
```
imputed_train_dataset
```

[183]:
```
array([[ 98.,   1.,   1., …,   1.,   0.,   0.],
       [199.,   1.,   3., …,   0.,   1.,   0.],
       [ 11.,   1.,   3., …,   0.,   0.,   1.],
       …,
       [232.,   0.,   3., …,   0.,   0.,   1.],
       [871.,   0.,   3., …,   0.,   0.,   1.],
       [422.,   0.,   3., …,   0.,   1.,   0.]])
```

[184]:
```
X_train = pd.DataFrame(imputed_train_dataset, columns=list(X_train))
```

0.0.5 The name and cabin are not sounding very useful for our model so dropping them will be a good idea

any ways the cabin column has a lot of nan values which will be dificult to tackle. So we are not including those columns from the cat_dataset into the num_dataset

0.0.6 Having a look at the ticket feature

define the class to separate the ticket code from ticket number (modification) i am planning to drop the ticket code feature so necessary changes are made in this class

[185]:
```
''' class SeperateTicket(BaseEstimator, TransformerMixin):
    def __init__(self):
        super().__init__()
```

```
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        alpha_part = []
        num_part = []
        for i in range(len(X)):
            total = X["Ticket"][i].split()
            if len(total) > 2:
                total.pop(1)
            if len(total) == 1:
                if(total[0].isalpha()):
                    total.append('0')
                else:
                    total.insert(0, 'No Code')
            for i in total:
                if(i[0].isdigit()):
                    num_part.append(int(i))
        ticket_dict = {
            "ticket_num": num_part,
        }
        ticket_df = pd.DataFrame(ticket_dict)
        X.drop(columns=["Ticket"], inplace=True)
        X = X.join(ticket_df)
        return X.to_numpy() '''
```

[185]:
```
' class SeperateTicket(BaseEstimator, TransformerMixin):\n    def
__init__(self):\n        super().__init__()\n    def fit(self, X, y=None):\n
return self\n    def transform(self, X, y=None):\n        alpha_part = []\n
num_part = []\n        for i in range(len(X)):\n            total =
X["Ticket"][i].split()\n            if len(total) > 2:\n
total.pop(1)\n            if len(total) == 1:\n
if(total[0].isalpha()):\n                    total.append(\'0\')\n
else:\n                    total.insert(0, \'No Code\')\n            for i in
total:\n                if(i[0].isdigit()):\n
num_part.append(int(i))\n        ticket_dict = {\n            "ticket_num":
num_part,\n        }\n        ticket_df = pd.DataFrame(ticket_dict)\n
X.drop(columns=["Ticket"], inplace=True)\n        X = X.join(ticket_df)\n
return X.to_numpy() '
```

use the class to separate ticket code and ticket number

- construct new dataframe with the obtained ticket code and ticket number
- delete the past ticket column
- restore to proper datatype

[186]:
```
''' all_cols.remove("Ticket")
all_cols.extend(["ticket_num"])
```

```
ticket_seperator = SeperateTicket()
seperated_arr = ticket_seperator.transform(X_train)
X_train = pd.DataFrame(seperated_arr, columns=all_cols)
revert_datatype(X_train, int_features_index=[0, 1, 2, 3, 5, 6, 8, 9, 10, 11],␣
↪float_features_index=[4, 7])
X_train '''
```

[186]:
```
' all_cols.remove("Ticket")\nall_cols.extend(["ticket_num"])\nticket_seperator =
SeperateTicket()\nseperated_arr = ticket_seperator.transform(X_train)\nX_train =
pd.DataFrame(seperated_arr, columns=all_cols)\nrevert_datatype(X_train,
int_features_index=[0, 1, 2, 3, 5, 6, 8, 9, 10, 11], float_features_index=[4,
7])\nX_train '
```

[187]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 623 entries, 0 to 622
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  623 non-null    float64
 1   Survived     623 non-null    float64
 2   Pclass       623 non-null    float64
 3   Age          623 non-null    float64
 4   SibSp        623 non-null    float64
 5   Parch        623 non-null    float64
 6   Fare         623 non-null    float64
 7   Sex          623 non-null    float64
 8   C            623 non-null    float64
 9   Q            623 non-null    float64
 10  S            623 non-null    float64
dtypes: float64(11)
memory usage: 53.7 KB
```

[188]:
```
corr_matr = X_train.corr(numeric_only=True)
corr_matr["Survived"].sort_values(ascending=False)
```

[188]:
```
Survived       1.000000
Fare           0.244125
C              0.177686
Parch          0.040049
Q             -0.011346
PassengerId   -0.040355
SibSp         -0.054082
Age           -0.069049
S             -0.147534
Pclass        -0.315752
```

```
Sex           -0.542131
Name: Survived, dtype: float64
```

0.0.7 Data Visualization

based on sex

graph showing male and female survival status

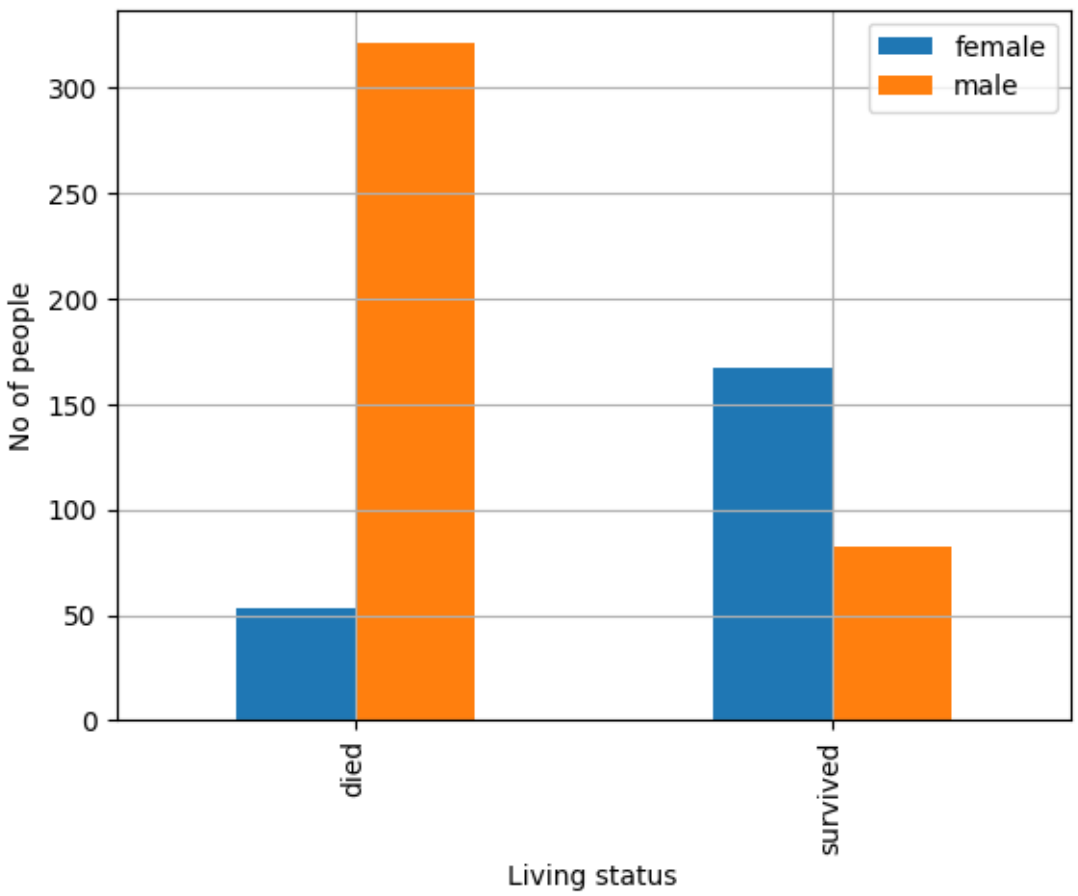
[189]:
```
x = X_train.groupby(["Survived", "Sex"]).size().unstack(fill_value=0)
x.columns = ['female', 'male']
x.index = ['died', 'survived']
x
```

[189]:

	female	male
died	53	321
survived	167	82

[190]:
```
%matplotlib inline
import matplotlib.pyplot as plt
x.plot(kind='bar')
plt.grid(True)
plt.ylabel("No of people")
plt.xlabel("Living status")
```

[190]:
```
Text(0.5, 0, 'Living status')
```



graph showing male and female survival status per total number of male and female respectively

[191]:
```
total_males = X_train[X_train["Sex"] == 1].shape[0]
total_females = X_train[X_train["Sex"] == 0].shape[0]
print(f"Total male: {total_males}\nTotal females: {total_females}")
```

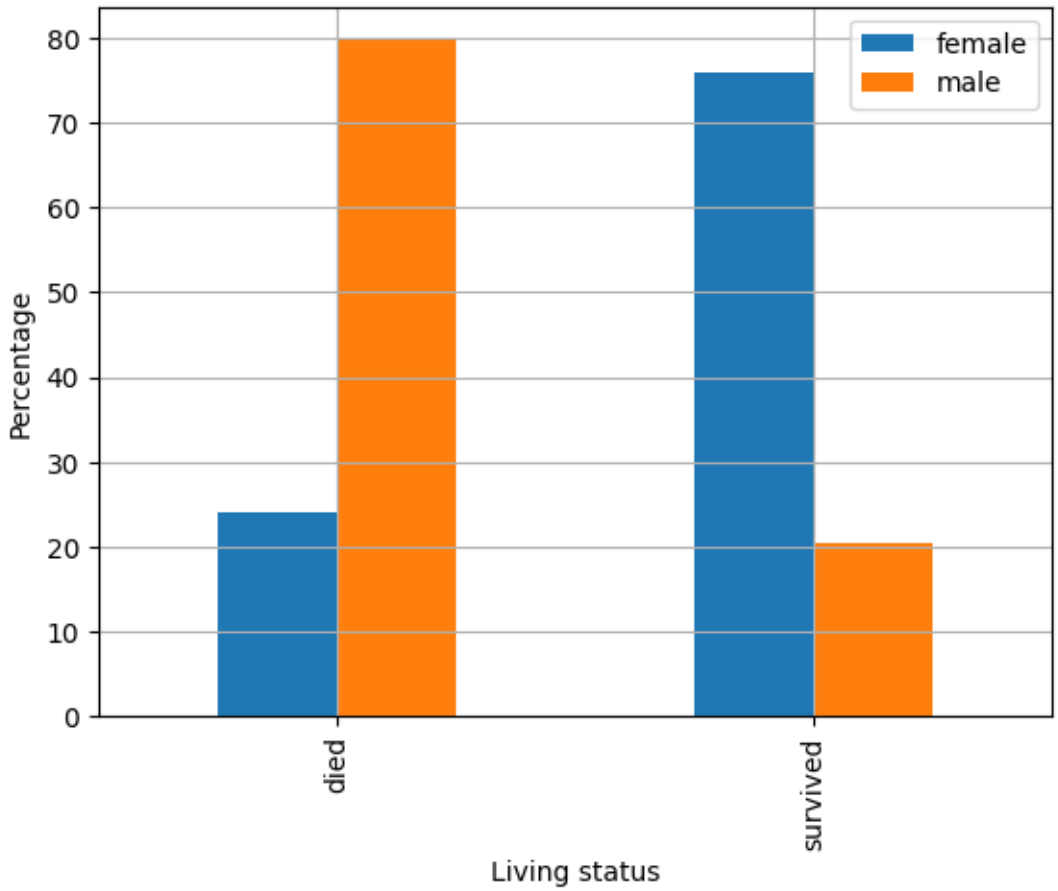
```
Total male: 403
Total females: 220
```

[192]:
```
x["female"] = x["female"]/total_females*100
x["male"] = x["male"]/total_males*100
print(x)
```

```
              female       male
died       24.090909  79.652605
survived   75.909091  20.347395
```

[193]:
```
%matplotlib inline
import matplotlib.pyplot as plt
x.plot(kind='bar')
plt.grid(True)
plt.ylabel("Percentage")
plt.xlabel("Living status")
```

[193]:
```
Text(0.5, 0, 'Living status')
```



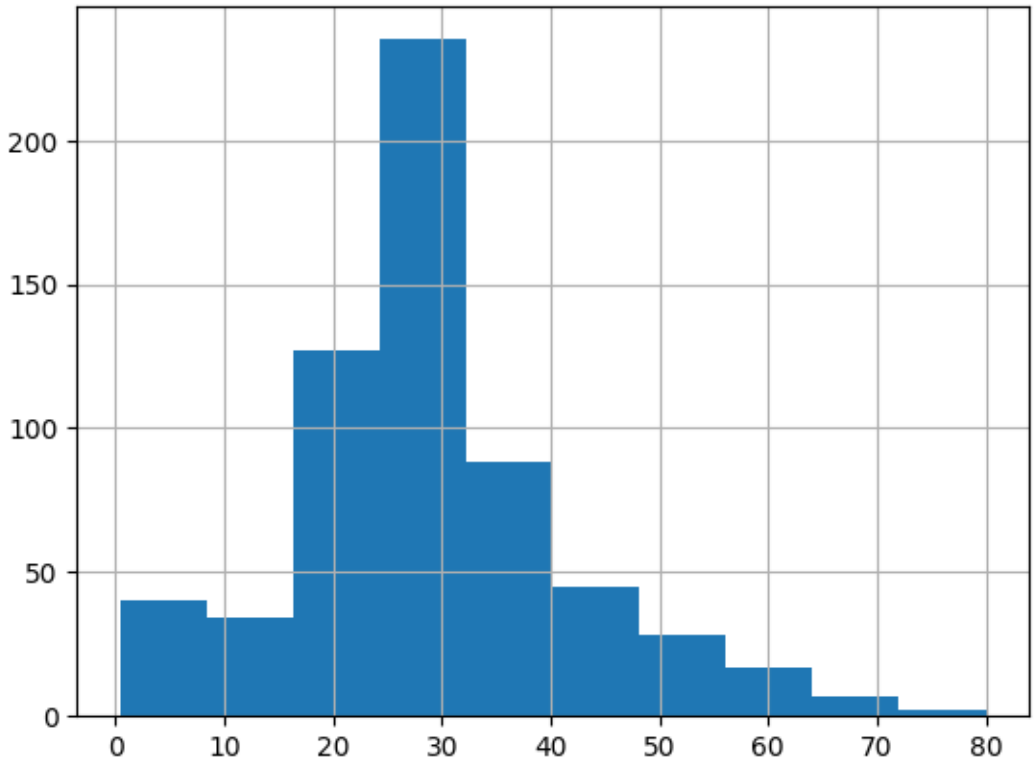
Surprisingly it is seen that almost 80% males died while almost 75% females survived

based on age

divide data according to age category

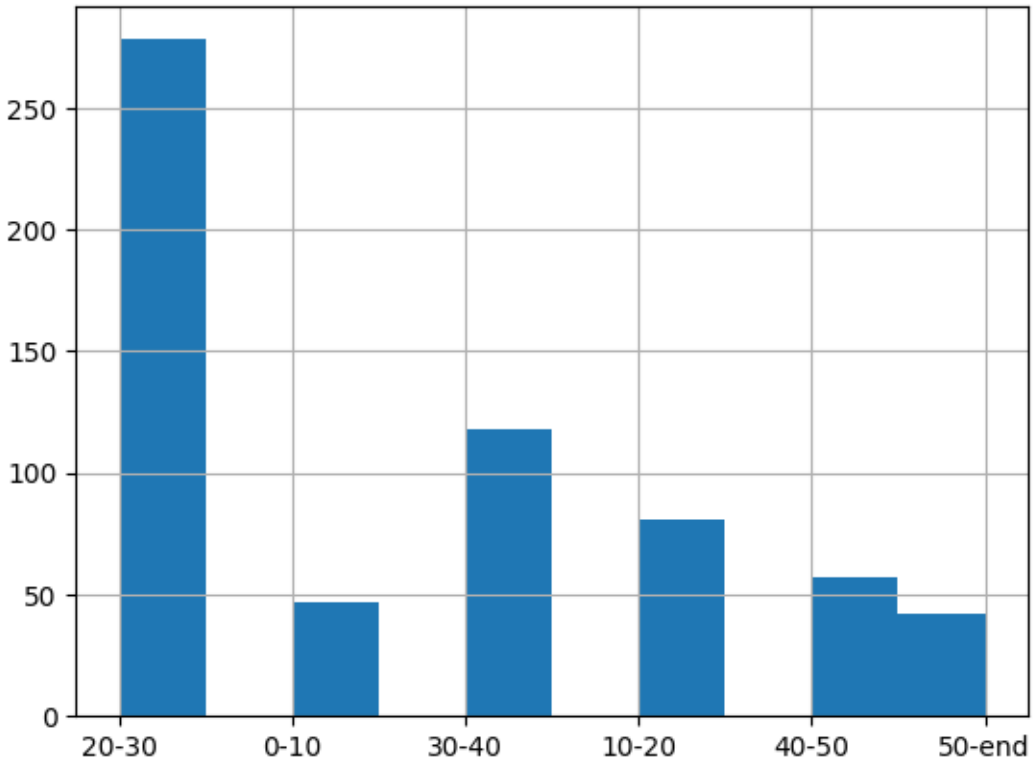
[194]:
```
X_train["Age"].hist(bins = 10)
```

[194]:
```
<Axes: >
```



[195]:
```
X_train["age_category"] = pd.cut(
                                 X_train["Age"],
                                 bins=[0., 10., 20., 30., 40., 50., np.inf],
                                 labels=["0-10", "10-20", "20-30", "30-40",␣
 ↪"40-50", "50-end"]
                                 )
X_train["age_category"].hist()
```

[195]: <Axes: >



it is evident that the ship contained most of the people in age group 20-30 years

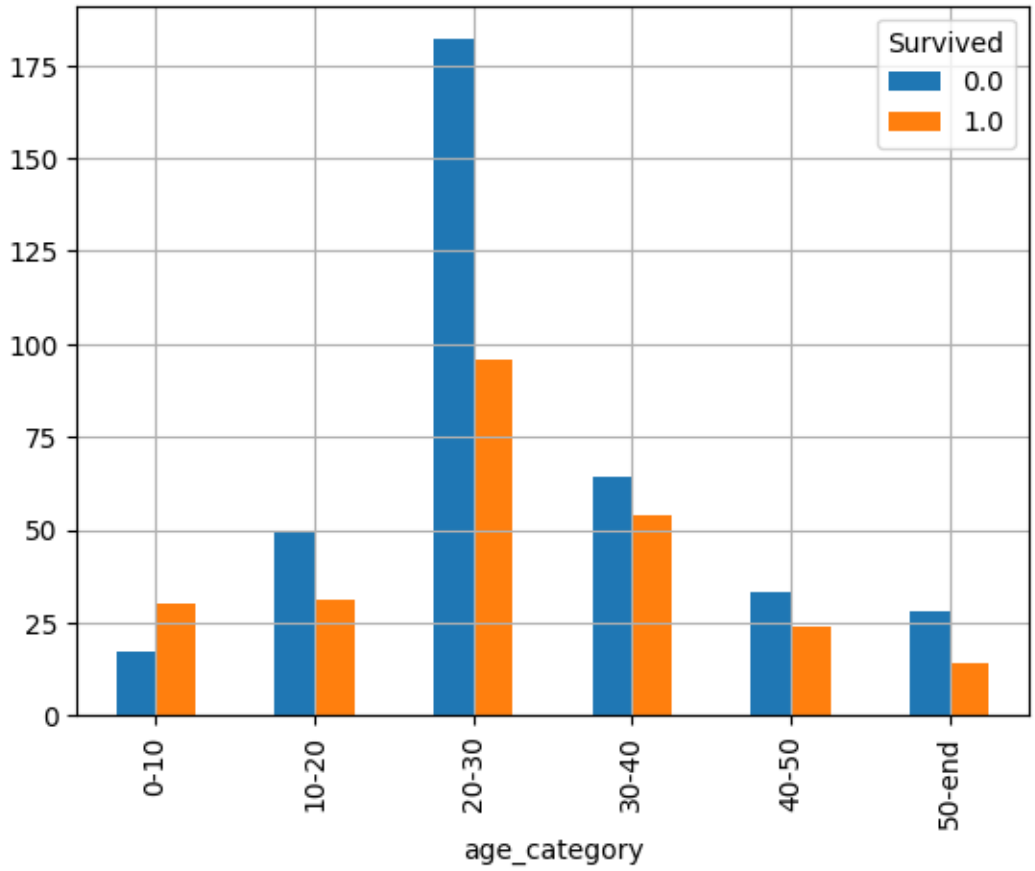
[196]:
```
x = X_train.groupby(["age_category", "Survived"]).size().unstack(fill_value=0)
# x.columns = ['female', 'male']
# x.index = ['died', 'survived']
x
```

```
C:\Users\dasas\AppData\Local\Temp\ipykernel_12380\1017478552.py:1:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  x = X_train.groupby(["age_category", "Survived"]).size().unstack(fill_value=0)
```

[196]:

Survived	0.0	1.0
age_category		
0-10	17	30
10-20	50	31
20-30	182	96
30-40	64	54
40-50	33	24
50-end	28	14

[197]:
```
x.plot(kind="bar")
plt.grid(True)
```



[198]:
```
x.iloc[0,1]
```

[198]: 30

survival or death percentage according to age category

[199]:
```
age_total_count = []
for i in range(len(x)):
    count = x.iloc[i].loc[0] + x.iloc[i].loc[1]
    age_total_count.append(count)
    # x.iloc[i].loc[0] = x.iloc[i].loc[0]/count*100
    # x.iloc[i].loc[1] = x.iloc[i].loc[1]/count*100
for i in age_total_count:
    print(int(i))
```

```
47
81
```

```
278
118
57
42
```

[200]:
```
for i in range(len(x)):
    x.iloc[i, 0] = float(int(x.iloc[i, 0])/int(age_total_count[i]))*100
    x.iloc[i, 1] = float(int(x.iloc[i, 1])/int(age_total_count[i]))*100
```

```
C:\Users\dasas\AppData\Local\Temp\ipykernel_12380\1547325009.py:2:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value '36.17021276595745' has
dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  x.iloc[i, 0] = float(int(x.iloc[i, 0])/int(age_total_count[i]))*100
C:\Users\dasas\AppData\Local\Temp\ipykernel_12380\1547325009.py:3:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value '63.829787234042556' has
dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  x.iloc[i, 1] = float(int(x.iloc[i, 1])/int(age_total_count[i]))*100
```

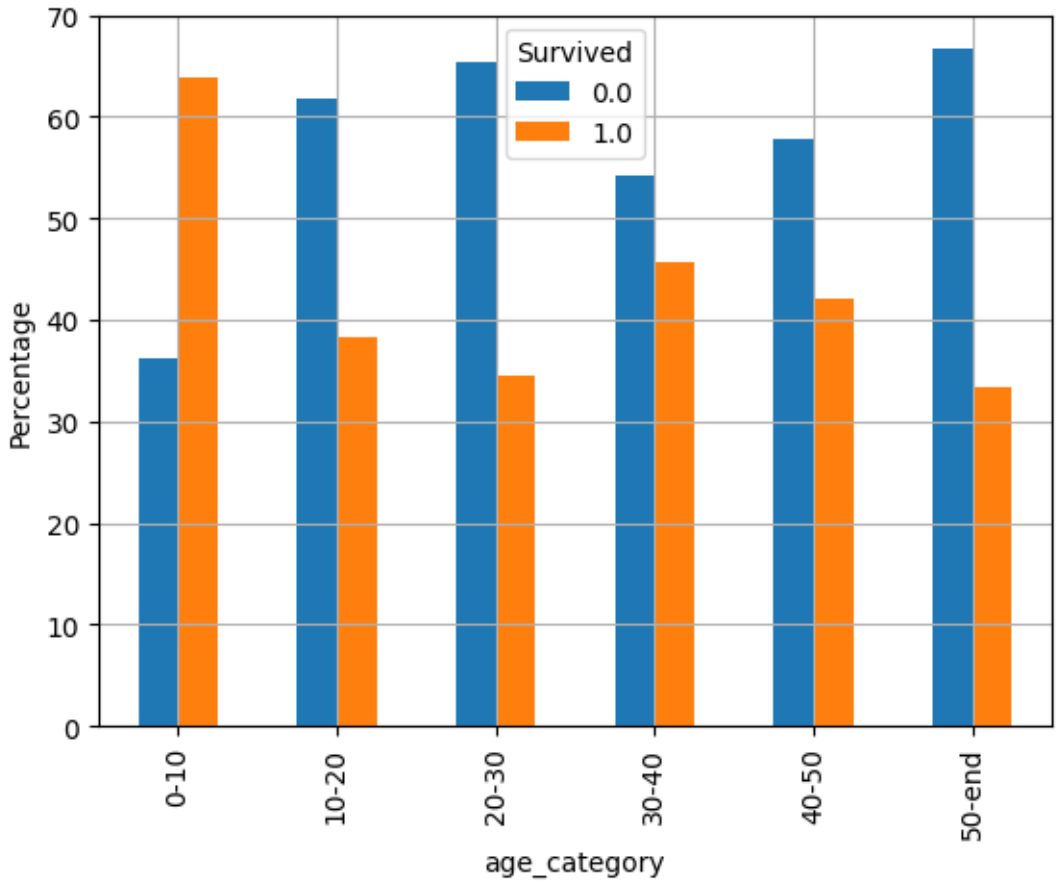
[201]:
```
x
```

[201]:

Survived	0.0	1.0
age_category		
0-10	36.170213	63.829787
10-20	61.728395	38.271605
20-30	65.467626	34.532374
30-40	54.237288	45.762712
40-50	57.894737	42.105263
50-end	66.666667	33.333333

[202]:
```
x.plot(kind="bar")
plt.grid(True)
plt.ylabel("Percentage")
```

[202]:
```
Text(0, 0.5, 'Percentage')
```



so it can clearly be concluded that children of age group 0 to 10 are the maximum to survive and the people within the age 50 till end are the maximum to die

based on both sex and age group

[203]:
```
x = X_train.groupby(["Sex", "Survived", "age_category"]).size().
  ↪unstack(fill_value=0)
# x.columns = ["age_category", "Sex", "dead", "alive"]
x
```

```
C:\Users\dasas\AppData\Local\Temp\ipykernel_12380\3905749618.py:1:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  x = X_train.groupby(["Sex", "Survived",
"age_category"]).size().unstack(fill_value=0)
```

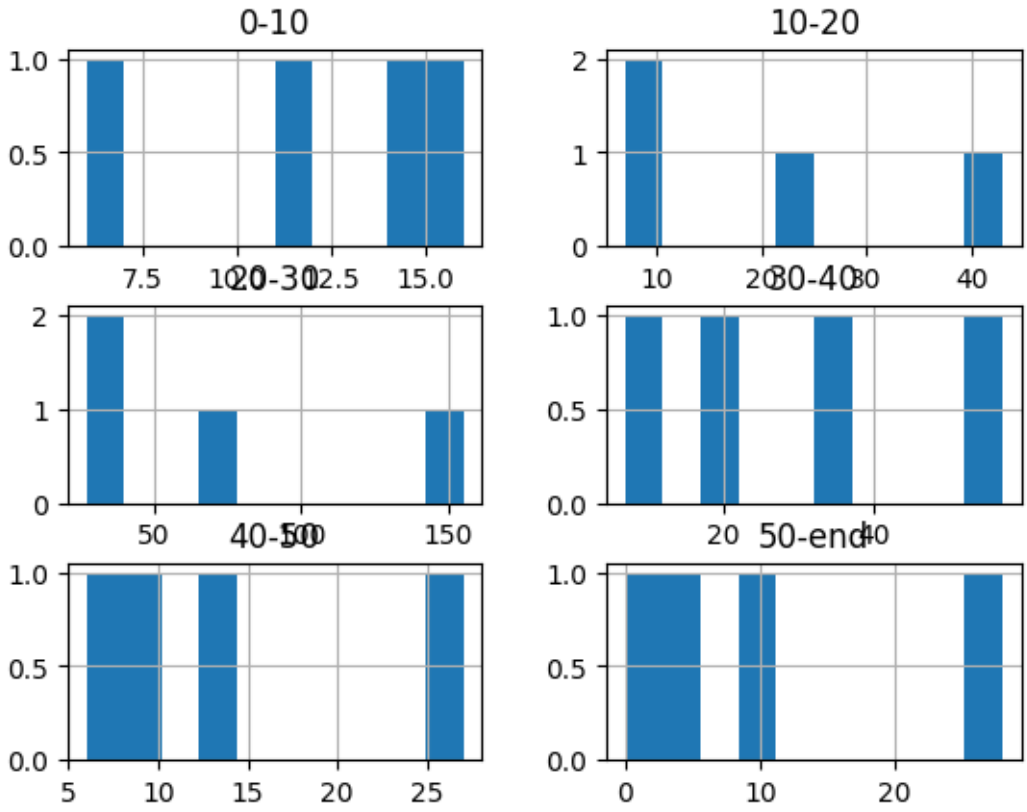
[203]:

	age_category	0-10	10-20	20-30	30-40	40-50	50-end
Sex	Survived						
0.0	0.0	6	7	27	7	6	0

```
     1.0          16      24      69      35      14        9
1.0  0.0          11      43     155      57      27       28
     1.0          14       7      27      19      10        5
```

[204]:
```
x.hist()
```

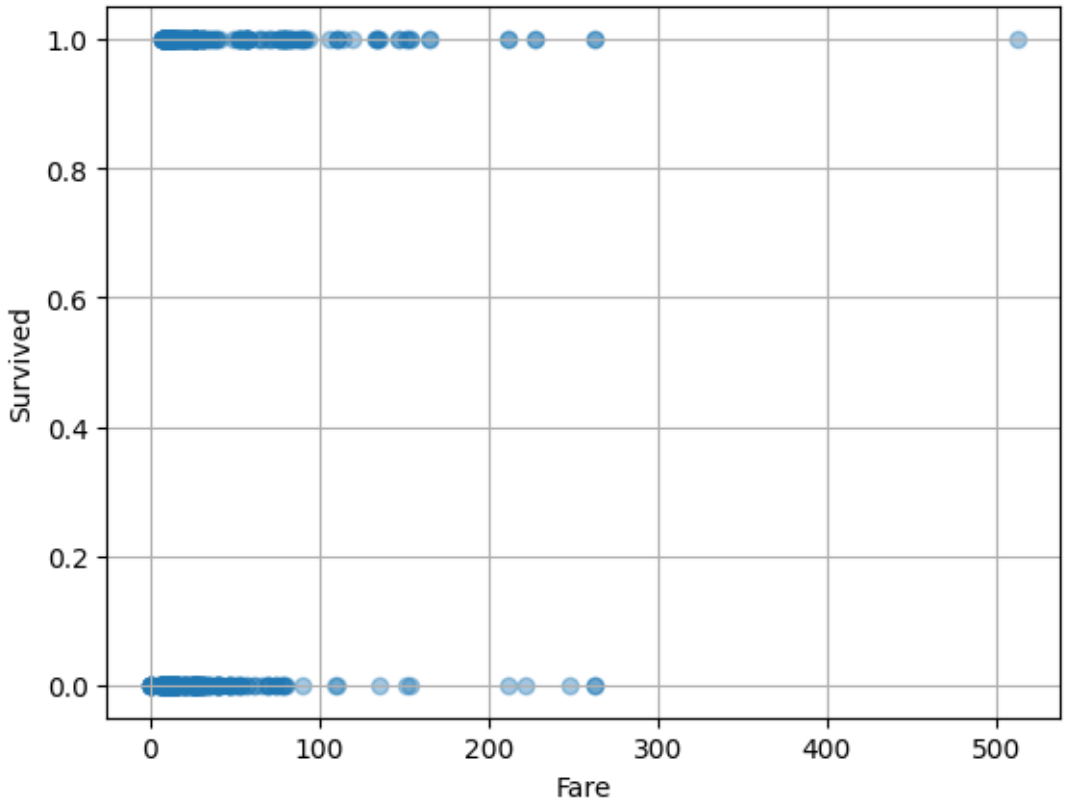
[204]:
```
array([[<Axes: title={'center': '0-10'}>,
        <Axes: title={'center': '10-20'}>],
       [<Axes: title={'center': '20-30'}>,
        <Axes: title={'center': '30-40'}>],
       [<Axes: title={'center': '40-50'}>,
        <Axes: title={'center': '50-end'}>]], dtype=object)
```



not so conclusive

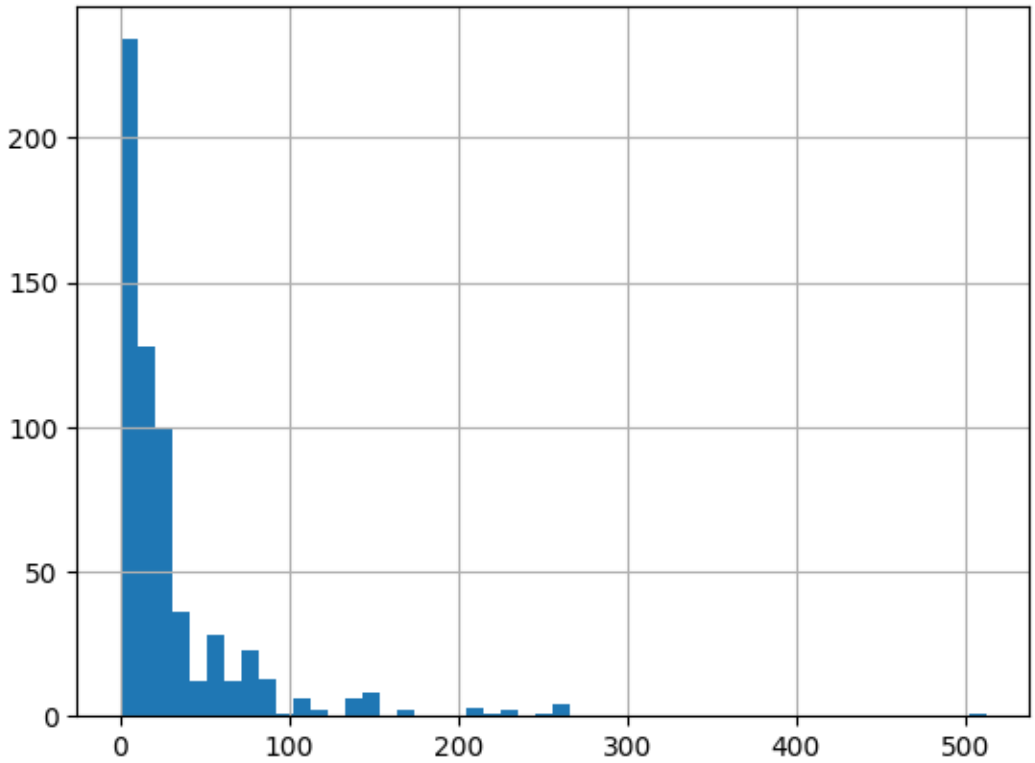
based on fare

[205]:
```
plt.scatter(X_train["Fare"], X_train["Survived"],  alpha=0.4)
plt.xlabel("Fare")
plt.ylabel("Survived")
plt.grid(True)
```



[206]:
```
X_train["Fare"].hist(bins = 50)
```

[206]: <Axes: >



not so conclusive

Drop the age category column as it was only added for our convenience

[207]:
```
X_train.drop(columns=["age_category"], inplace=True)
```

[208]:
```
X_train
```

[208]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex	C	\
0	98.0	1.0	1.0	23.0	0.0	1.0	63.3583	1.0	1.0	
1	199.0	1.0	3.0	28.0	0.0	0.0	7.7500	0.0	0.0	
2	11.0	1.0	3.0	4.0	1.0	1.0	16.7000	0.0	0.0	
3	809.0	0.0	2.0	39.0	0.0	0.0	13.0000	1.0	0.0	
4	207.0	0.0	3.0	32.0	1.0	0.0	15.8500	1.0	0.0	
..	…	…	…	…	…	…	…	…	…	
618	132.0	0.0	3.0	20.0	0.0	0.0	7.0500	1.0	0.0	
619	693.0	1.0	3.0	28.0	0.0	0.0	56.4958	1.0	0.0	
620	232.0	0.0	3.0	29.0	0.0	0.0	7.7750	1.0	0.0	
621	871.0	0.0	3.0	26.0	0.0	0.0	7.8958	1.0	0.0	
622	422.0	0.0	3.0	21.0	0.0	0.0	7.7333	1.0	0.0	

	Q	S
0	0.0	0.0

```
1     1.0  0.0
2     0.0  1.0
3     0.0  1.0
4     0.0  1.0
..    …  …
618   0.0  1.0
619   0.0  1.0
620   0.0  1.0
621   0.0  1.0
622   1.0  0.0

[623 rows x 11 columns]
```

0.0.8 Model Training

Stochastic Gradient Descent classifier model

Training

[209]:
```
from sklearn.linear_model import SGDClassifier

sgd_classifier = SGDClassifier(random_state=42)
sgd_classifier.fit(X_train.values, y_train.values)
```

[209]:
```
SGDClassifier(random_state=42)
```

[210]:
```
X_train.values[0].reshape(1, -1)
```

[210]:
```
array([[98.    ,  1.    ,  1.    , 23.    ,  0.    ,  1.    , 63.3583,
         1.    ,  1.    ,  0.    ,  0.    ]])
```

[211]:
```
sgd_classifier.predict(X_train.values[120].reshape(1, -1))[0]
```

[211]:
```
1
```

[212]:
```
print(y_train.iloc[0])
```

```
1
```

Testing the model on various performance measurement parameters

[213]:
```
from sklearn.model_selection import cross_val_score

cross_val_score(sgd_classifier, X_train.values, y_train.values,␣
 ↪scoring="accuracy", cv=3)
```

[213]:
```
array([0.43269231, 0.625     , 0.68115942])
```

[214]:
```
cross_val_score(sgd_classifier, X_train.values, y_train.values,␣
↪scoring="recall", cv=3)
```

[214]:
```
array([0.97590361, 0.09638554, 0.55421687])
```

[215]:
```
cross_val_score(sgd_classifier, X_train.values, y_train.values,␣
↪scoring="precision", cv=3)
```

[215]:
```
array([0.41116751, 0.72727273, 0.61333333])
```

[216]:
```
cross_val_score(sgd_classifier, X_train.values, y_train.values, scoring="f1",␣
↪cv=3)
```

[216]:
```
array([0.57857143, 0.17021277, 0.58227848])
```

[217]:
```
from sklearn.model_selection import cross_val_predict
cross_predictions = cross_val_predict(sgd_classifier, X_train.values, y_train.
↪values, cv=3)
```

[218]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train.values, cross_predictions)
```

[218]:
```
array([[226, 148],
       [114, 135]], dtype=int64)
```

[219]:
```
from sklearn.metrics import precision_score, recall_score

precision_score(y_train.values, cross_predictions)
```

[219]:
```
0.47703180212014135
```

[220]:
```
recall_score(y_train.values, cross_predictions)
```

[220]:
```
0.5421686746987951
```

[221]:
```
from sklearn.metrics import f1_score

f1_score(y_train.values, cross_predictions)
```

[221]:
```
0.5075187969924813
```

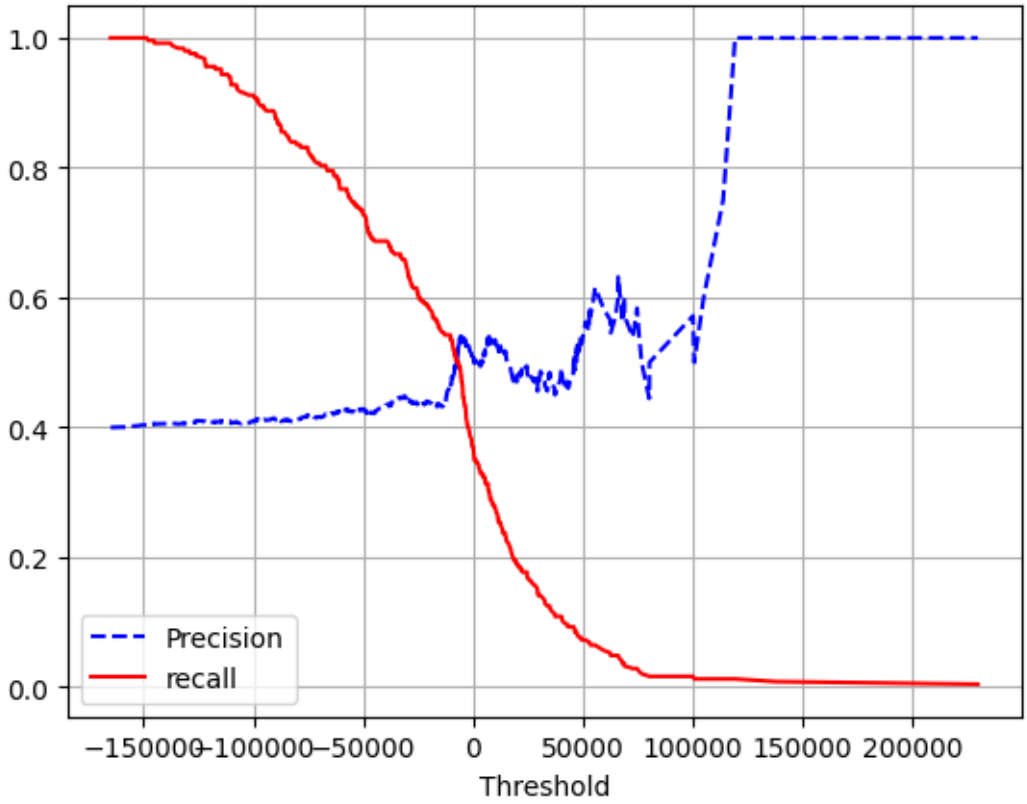
[222]:
```
y_scores = cross_val_predict(sgd_classifier, X_train.values, y_train.values,␣
↪method="decision_function")
```

[223]:
```
from sklearn.metrics import precision_recall_curve
```

```
precision, recall, threshold = precision_recall_curve(y_train.values, y_scores)
```

[224]:
```
def plot_precision_recall_threshold(precision, recall, threshold):
    plt.plot(threshold, precision[:-1], "b--", label="Precision")
    plt.plot(threshold, recall[:-1], "r-", label="recall")
    plt.xlabel("Threshold")
    plt.legend()
    plt.grid(True)
    plt.show()
```

[225]:
```
plot_precision_recall_threshold(precision=precision, recall=recall,␣
↪threshold=threshold)
```



[226]:
```
precision[np.argmax(threshold >= -7000)]
```

[226]: 0.508130081300813

[227]:
```
recall[np.argmax(threshold >= -7000)]
```

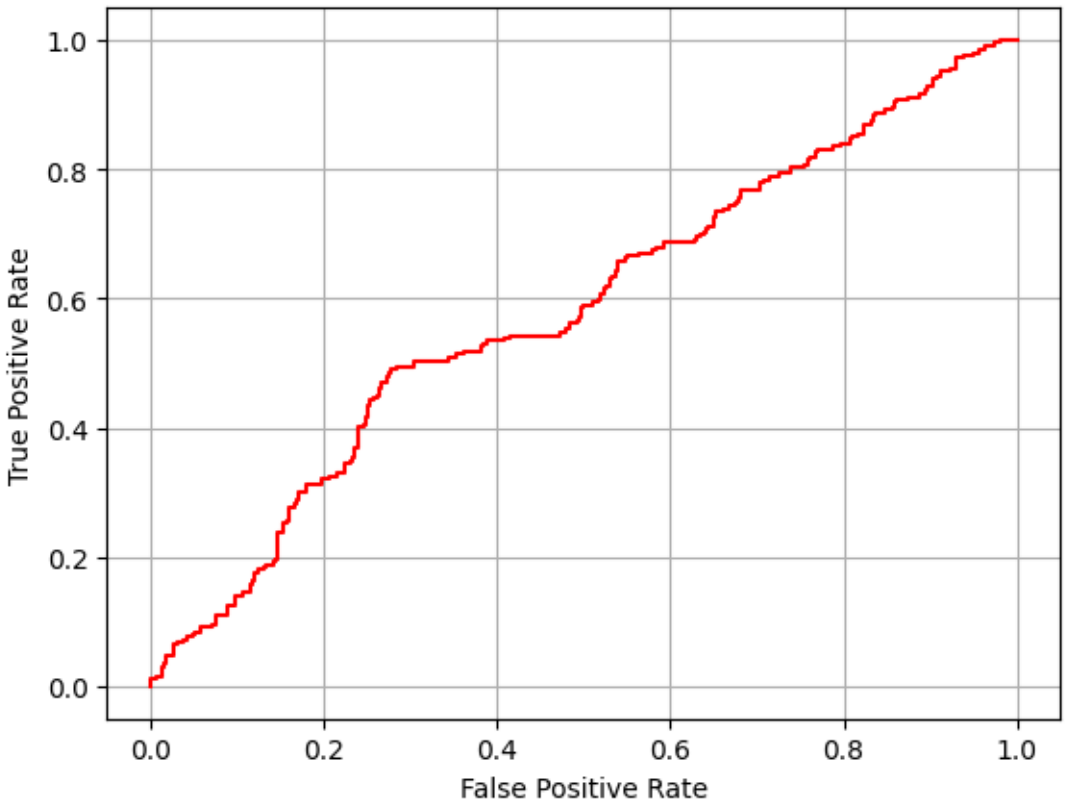
[227]: 0.5020080321285141

[228]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train.values, y_scores)
```

[229]:
```
def plot_roc(fpr, tpr):
    plt.plot(fpr, tpr, "r-")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.grid(True)
    plt.show()
```

[230]:
```
plot_roc(fpr, tpr)
```



[231]:
```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train.values, y_scores)
```

[231]: 0.5816420763266972

Conclusion: The SGDClassifier does not look much promising

Random Forest Classifier

Training

[232]:
```
from sklearn.ensemble import RandomForestClassifier

rfc_classifier = RandomForestClassifier(random_state=42)
rfc_classifier.fit(X_train.values, y_train.values)
```

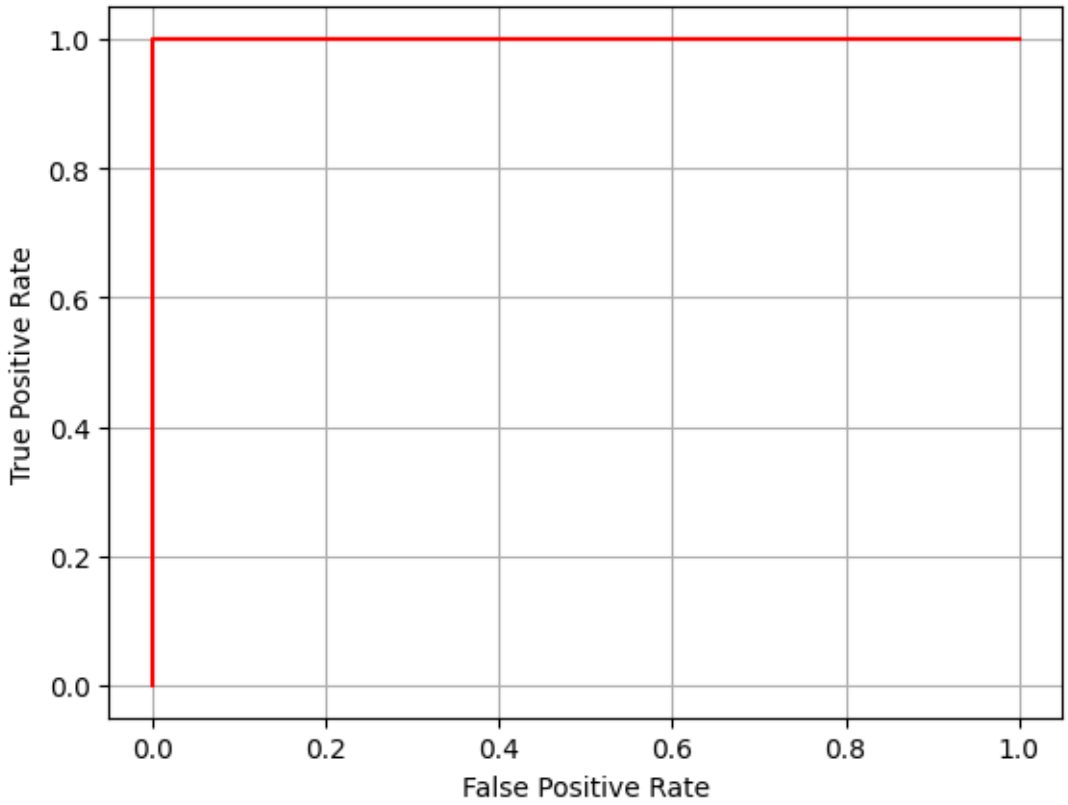
[232]:
```
RandomForestClassifier(random_state=42)
```

[233]:
```
y_predict_proba = cross_val_predict(rfc_classifier, X_train.values, y_train.
  ↪values, method="predict_proba", cv=3)
y_predict_proba
```

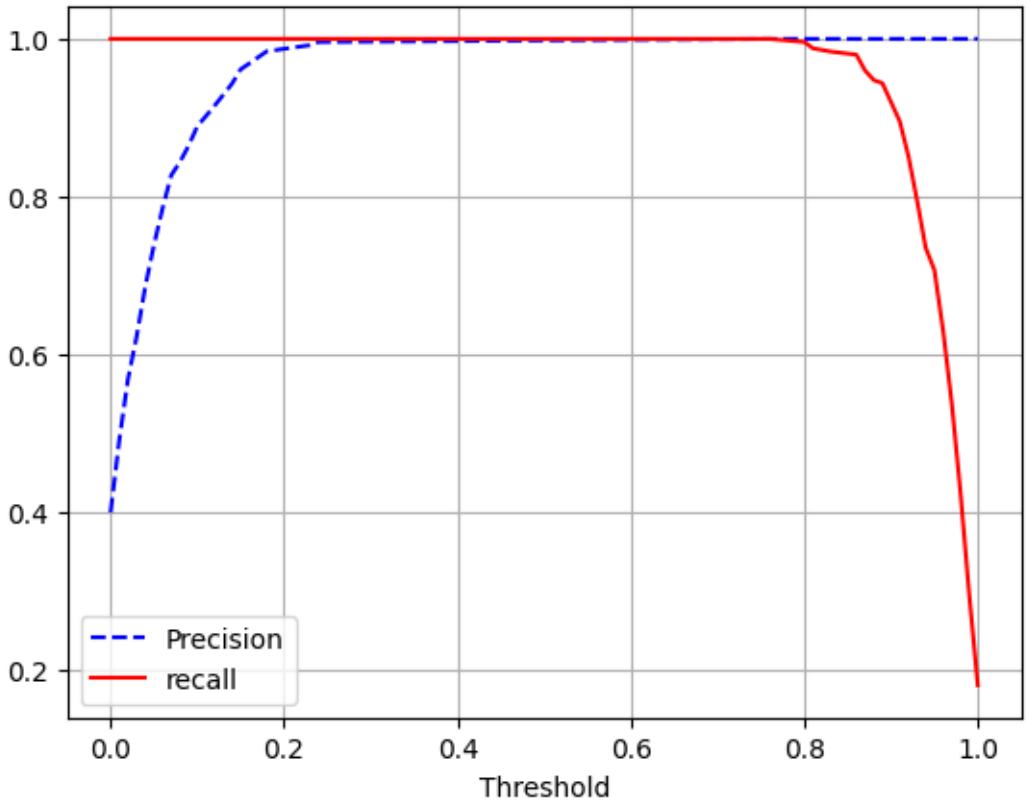
[233]:
```
array([[0.1 , 0.9 ],
       [0.02, 0.98],
       [0.07, 0.93],
       …,
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ]])
```

[234]:
```
y_scores_rfc = y_predict_proba[:,1]
```

[235]:
```
fpr_rfc, tpr_rfc, threshold_rfc = roc_curve(y_train.values, y_scores_rfc)
plot_roc(fpr=fpr_rfc, tpr=tpr_rfc)
```



[236]:
```
precision_rfc, recall_rfc, threshold_prc_rfc = precision_recall_curve(y_train.
  ↪values, y_scores_rfc)
plot_precision_recall_threshold(precision_rfc, recall_rfc, threshold_prc_rfc)
```



[237]:
```
cross_val_score(rfc_classifier, X_train.values, y_train.values, cv=3,␣
↪scoring="recall")
```

[237]:
```
array([1., 1., 1.])
```

[238]:
```
y_rfc_pred = cross_val_predict(rfc_classifier, X_train.values, y_train.values,␣
↪cv=3)
confusion_matrix(y_train.values, y_rfc_pred)
```

[238]:
```
array([[374,   0],
       [  0, 249]], dtype=int64)
```

Testing the random forest classifier model on test dataset

[239]:
```
imputed_test = imputer.transform(X_test)
X_test = pd.DataFrame(imputed_test, columns=list(X_test))
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 268 entries, 0 to 267
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
```

```
---  ------       --------------  -----
 0   PassengerId  268 non-null    float64
 1   Survived     268 non-null    float64
 2   Pclass       268 non-null    float64
 3   Age          268 non-null    float64
 4   SibSp        268 non-null    float64
 5   Parch        268 non-null    float64
 6   Fare         268 non-null    float64
 7   Sex          268 non-null    float64
 8   C            268 non-null    float64
 9   Q            268 non-null    float64
 10  S            268 non-null    float64
dtypes: float64(11)
memory usage: 23.2 KB
```

[240]:
```
y_test_predictions = rfc_classifier.predict(X_test.values)
```

[241]:
```
confusion_matrix(y_test.values, y_test_predictions)
```

[241]:
```
array([[175,   0],
       [  0,  93]], dtype=int64)
```

[242]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_neighbours = KNeighborsClassifier()
knn_neighbours.fit(X_train.values, y_train.values)
```

[242]:
```
KNeighborsClassifier()
```

[243]:
```
cross_val_score(knn_neighbours, X_train.values, y_train.values, cv=3,␣
↪scoring="accuracy")
```

[243]:
```
array([0.61538462, 0.625     , 0.58454106])
```

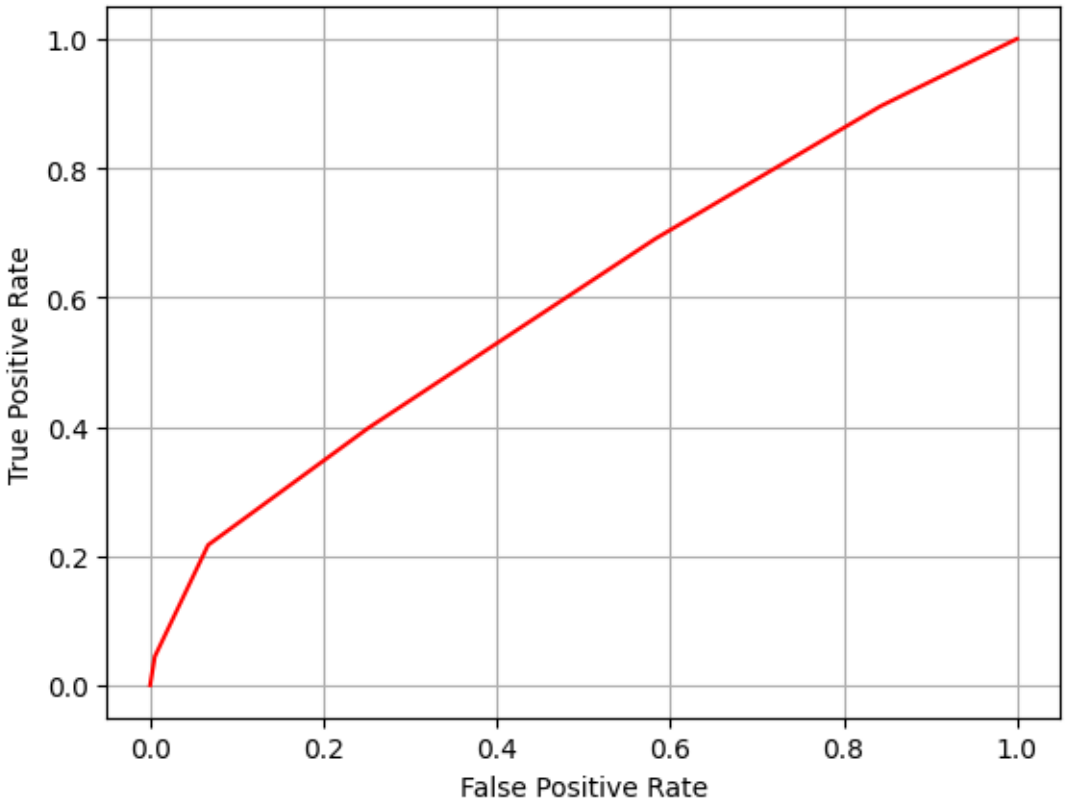
[244]:
```
y_prediction_knn = cross_val_predict(knn_neighbours, X_train.values, y_train.
↪values, cv=3)
confusion_matrix(y_train.values, y_prediction_knn)
```

[244]:
```
array([[280,  94],
       [150,  99]], dtype=int64)
```

[245]:
```
y_scores_knn = cross_val_predict(knn_neighbours, X_train.values, y_train.
↪values, cv=3, method="predict_proba")
```

[246]:
```
y_scores_knn = y_scores_knn[:, 1]
```

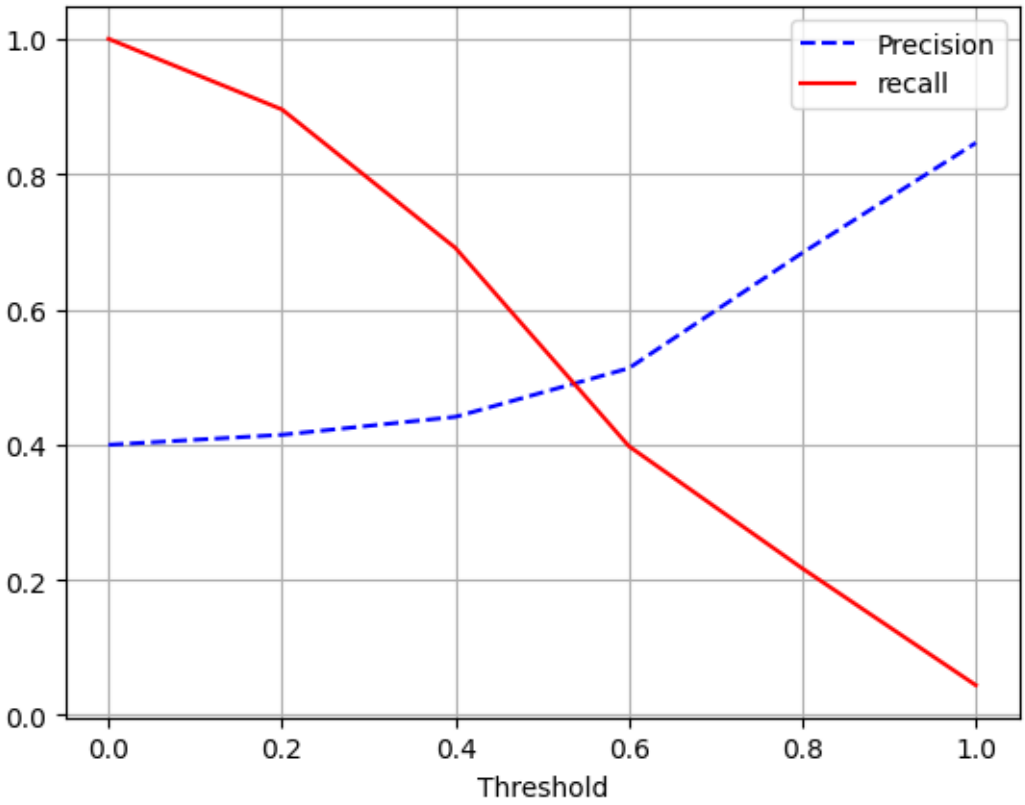
[247]:
```
fpr_knn, tpr_knn, threshold_knn = roc_curve(y_train.values, y_scores_knn)
plot_roc(fpr_knn, tpr_knn)
```



[250]:
```
roc_auc_score(y_train.values, y_scores_knn)
```

[250]:
```
0.600482142473638
```

[249]:
```
precision_knn, recall_knn, threshold_prc_knn = precision_recall_curve(y_train.
 ↪values, y_scores_knn)
plot_precision_recall_threshold(precision=precision_knn, recall=recall_knn,␣
 ↪threshold=threshold_prc_knn)
```


Precision
recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Threshold