Titanic Survival Classification

The survival chances of the Titanic passengers using the given information about their sex, age, etc. As this is a classification task we will be using random forest.

There will be three main steps:

Feature Engineering

Imputation

Training and Prediction

Importing Libraries and Initial setup

Python 3

import warnings

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

plt.style.use('fivethirtyeight')

%matplotlib inline

warnings.filterwarnings('ignore')

[**LEARN MORE**](https://adclick.g.doubleclick.net/pcs/click?xai=AKAOjsusfUbCleQ_TVfBzUx-lssZsq51cZOIp6IvDCMl6nPBbiLUmumhl8YiMz5tWzPssxQNIKAASMv8GUP02c7QmX8ZKMBdKZQurvJbSz_qprIjtfTKxakuaHe2S9NjyIzOsS-Eeu5WvXbsgdmJ4wkOTqdnDe6oB7oAoGeBhvOGBBCMz64qCJO4Jm14AaDaRK_Q_1EG7Wi14jccWQyyfypGsjeukO8seN-sdU5DlXH62mE3q3HpzRPMYUzUKwFURrXrVO2YNF-2kq_iyd-JfFpJJlqjiIjIPV-bmiwn5fyKa0ytjmwEVk68anQhAZ8Bekx5PaIkjFCMSIpwsKJzCHQ56vjjLFhNyg&sai=AMfl-YTfw8rw1G6Dd62PBI_469CmD0XSXDpL3Q4JDPyH2jlEk7Ewv8XoZyo4-VurzFB1OUF_clzYGPMGtzBsLayUEzbsfnGjhLNoUnXEtQK_SYn87Zc79XPVBQDU9Ujx2Q&sig=Cg0ArKJSzApphXpLA5LeEAE&fbs_aeid=%5bgw_fbsaeid%5d&urlfix=1&adurl=https://www.geeksforgeeks.org/courses/data-science-live?utm_source=geeksforgeeks&utm_medium=in_article_native&utm_campaign=targetcourse)

Python3

train **=** pd.read\_csv('train.csv')

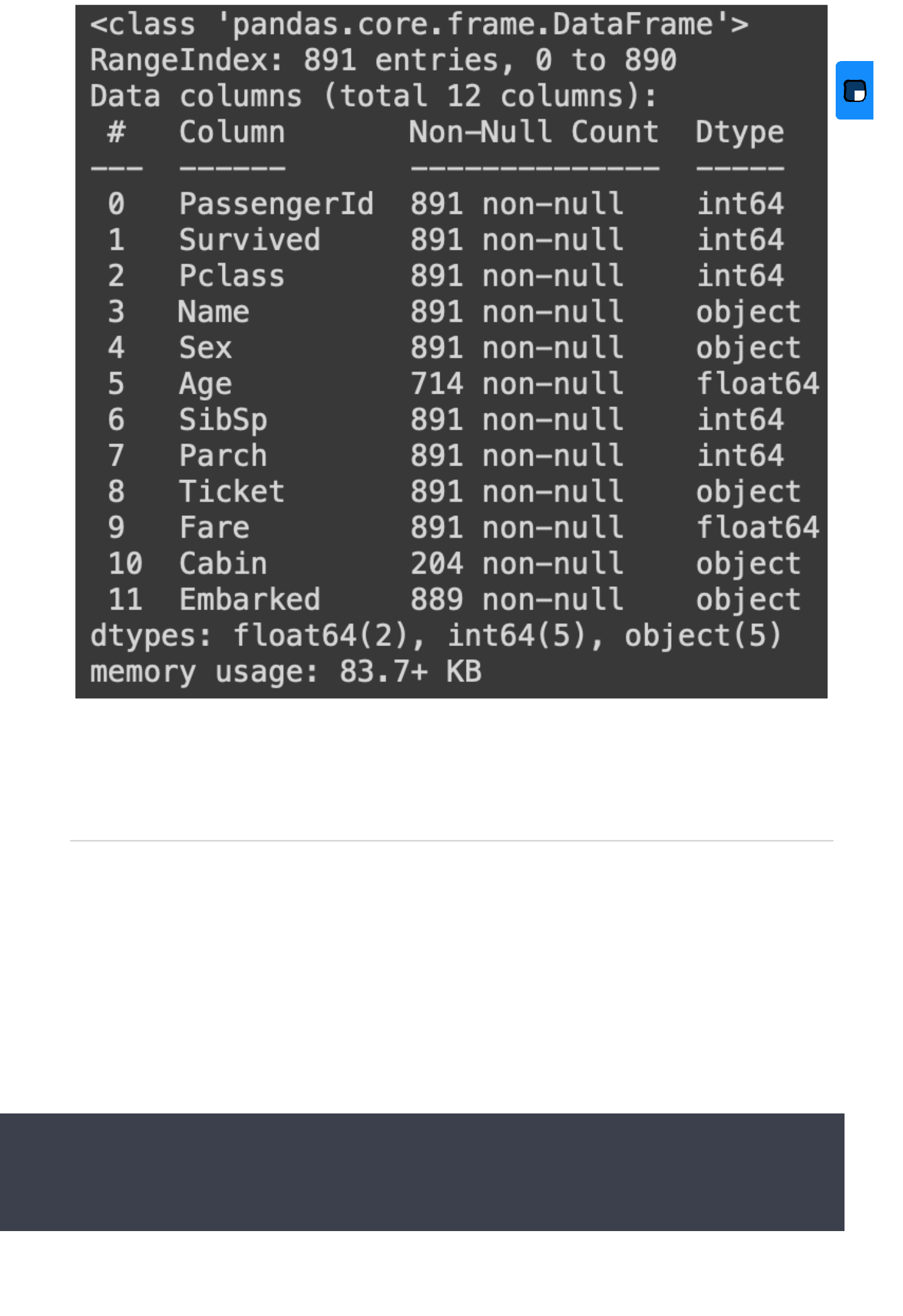
test **=** pd.read\_csv('test.csv')

* To know number of columns and rows train.shape
* (891, 12)

To know the information about each column like the data type, etc we use the df.info() function.

Python3

train.info()



Now let’s see if there are any NULL values present in the dataset. This can be checked using the isnull() function. It yields the following output.

Python3

train.isnull().sum()



Visualization

Now let us visualize the data using some pie charts and histograms to get a proper understanding of the data.

Let us first visualize the number of survivors and death counts.

Python3

f, ax **=** plt.subplots(1, 2, figsize**=**(12, 4))

train['Survived'].value\_counts().plot.pie(

explode**=**[0, 0.1], autopct**=**'%1.1f%%', ax**=**ax[0], shadow**=**False)

ax[0].set\_title('Survivors (1) and the dead (0)')

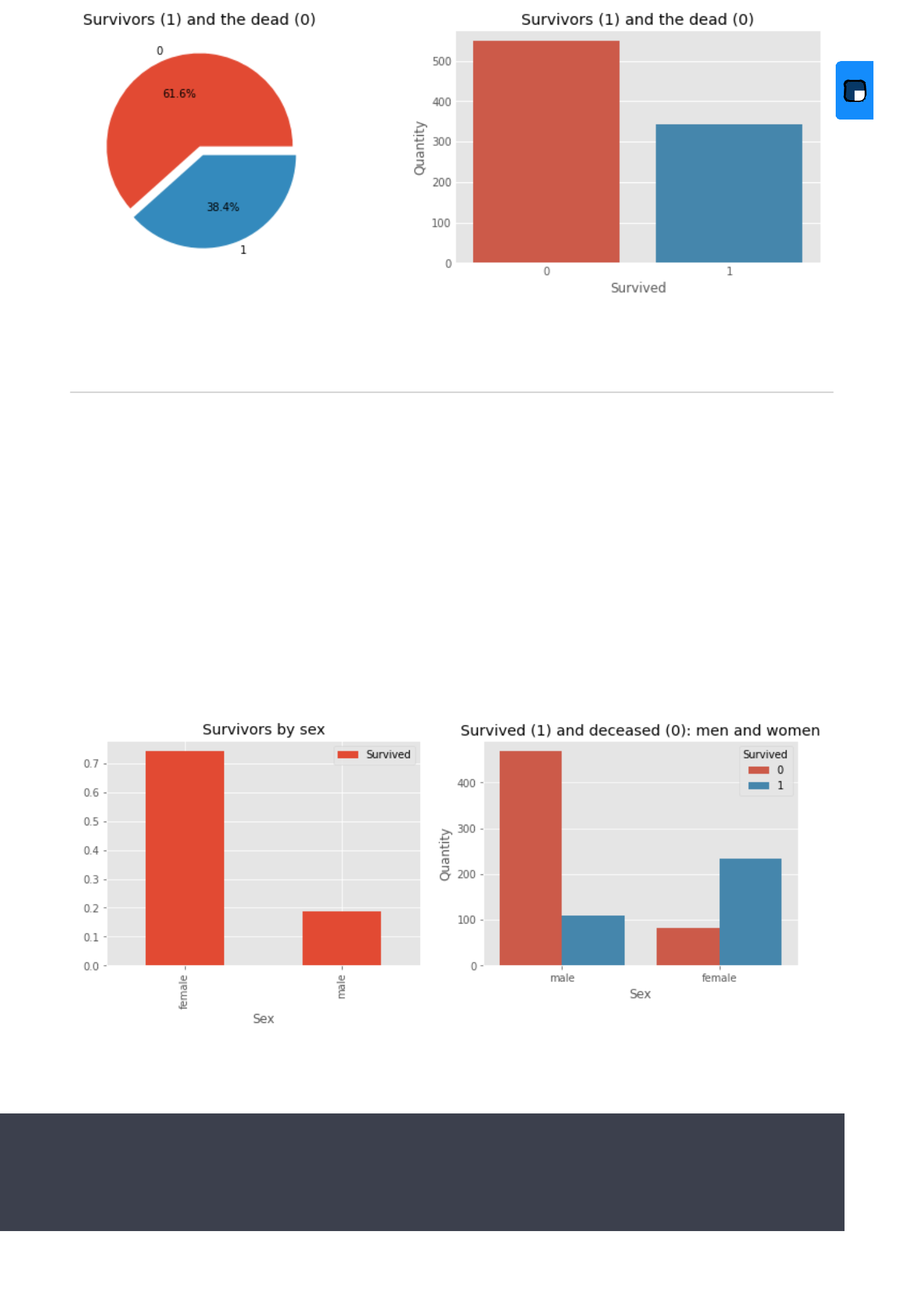
ax[0].set\_ylabel('')

sns.countplot('Survived', data**=**train, ax**=**ax[1])

ax[1].set\_ylabel('Quantity')

ax[1].set\_title('Survivors (1) and the dead (0)')

plt.show()



Sex feature

Python3

f, ax **=** plt.subplots(1, 2, figsize**=**(12, 4))

train[['Sex', 'Survived']].groupby(['Sex']).mean().plot.bar(ax**=**ax[0])

ax[0].set\_title('Survivors by sex')

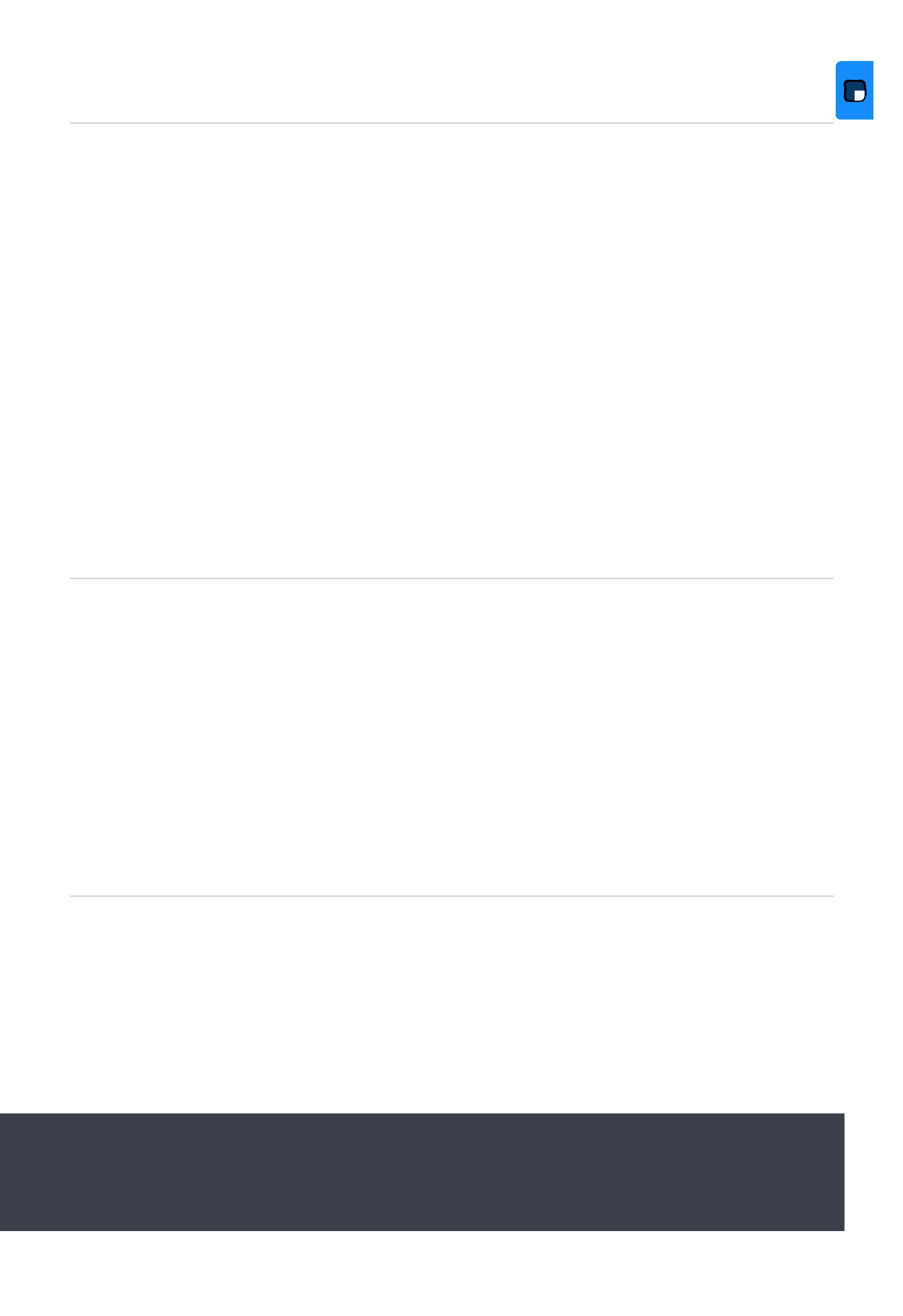
sns.countplot('Sex', hue**=**'Survived', data**=**train, ax**=**ax[1])

ax[1].set\_ylabel('Quantity')

ax[1].set\_title('Survived (1) and deceased (0): men and women')

plt.show()

Now let’s see which columns should we drop and/or modify for the model to predict the testing data. The main tasks in this step is to drop unnecessary features and to convert string data into the numerical category for easier training.



We’ll start off by dropping the Cabin feature since not a lot more useful information can be extracted from it. But we will make a new column from the Cabins column to see if there was cabin information allotted or not.

Python3

* Create a new column cabinbool indicating
* if the cabin value was given or was NaN

train["CabinBool"] **=** (train["Cabin"].notnull().astype('int'))

test["CabinBool"] **=** (test["Cabin"].notnull().astype('int'))

* Delete the column 'Cabin' from test
* and train dataset

train **=** train.drop(['Cabin'], axis**=**1)

test **=** test.drop(['Cabin'], axis**=**1)

We can also drop the Ticket feature since it’s unlikely to yield any useful information

Python3

train **=** train.drop(['Ticket'], axis**=**1)

test **=** test.drop(['Ticket'], axis**=**1)

There are missing values in the Embarked feature. For that, we will replace the NULL values with ‘S’ as the number of Embarks for ‘S’ are higher than the other two.

Python3

* replacing the missing values in
* the Embarked feature with S

train **=** train.fillna({"Embarked": "S"})

.

We will now sort the age into groups. We will combine the age groups of the people and categorize them into the same groups. BY doing so we will be having fewer categories and will have a better prediction since it will be a categorical dataset.

Python3

* sort the ages into logical categories train["Age"] **=** train["Age"].fillna(**-**0.5) test["Age"] **=** test["Age"].fillna(**-**0.5)

bins **=** [**-**1, 0, 5, 12, 18, 24, 35, 60, np.inf] labels **=** ['Unknown', 'Baby', 'Child', 'Teenager',

'Student', 'Young Adult', 'Adult', 'Senior'] train['AgeGroup'] **=** pd.cut(train["Age"], bins, labels**=**labels) test['AgeGroup'] **=** pd.cut(test["Age"], bins, labels**=**labels)

In the ‘title’ column for both the test and train set, we will categorize them into an equal number of classes. Then we will assign numerical values to the title for convenience of model training.

Python3

|  |
| --- |
| # create a combined group of both datasets  combine = [train, test]    # extract a title for each Name in the  # train and test datasets  for dataset in combine:      dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)    pd.crosstab(train['Title'], train['Sex'])    # replace various titles with more common names  for dataset in combine:      dataset['Title'] = dataset['Title'].replace(['Lady', 'Capt', 'Col',                                                   'Don', 'Dr', 'Major',                                                   'Rev', 'Jonkheer', 'Dona'],                                                  'Rare')        dataset['Title'] = dataset['Title'].replace(          ['Countess', 'Lady', 'Sir'], 'Royal')      dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')      dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')      dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')    train[['Title', 'Survived']].groupby(['Title'], as\_index=False).mean()    # map each of the title groups to a numerical value  title\_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3,                   "Master": 4, "Royal": 5, "Rare": 6}  for dataset in combine:      dataset['Title'] = dataset['Title'].map(title\_mapping)      dataset['Title'] = dataset['Title'].fillna(0) |

dataset['Title'] **=** dataset['Title'].replace(

['Countess', 'Lady', 'Sir'], 'Royal')

dataset['Title'] **=** dataset['Title'].replace('Mlle', 'Miss')

dataset['Title'] **=** dataset['Title'].replace('Ms', 'Miss')

dataset['Title'] **=** dataset['Title'].replace('Mme', 'Mrs')

train[['Title', 'Survived']].groupby(['Title'], as\_index**=**False).mean()

* map each of the title groups to a numerical value title\_mapping **=** {"Mr": 1, "Miss": 2, "Mrs": 3,

"Master": 4, "Royal": 5, "Rare": 6} **for** dataset **in** combine:

dataset['Title'] **=** dataset['Title'].map(title\_mapping)

dataset['Title'] **=** dataset['Title'].fillna(0)

Now using the title information we can fill in the missing age values.

Python3

mr\_age **=** train[train["Title"] **==** 1]["AgeGroup"].mode() # Young Adult miss\_age **=** train[train["Title"] **==** 2]["AgeGroup"].mode() # Student mrs\_age **=** train[train["Title"] **==** 3]["AgeGroup"].mode() # Adult master\_age **=** train[train["Title"] **==** 4]["AgeGroup"].mode() # Baby royal\_age **=** train[train["Title"] **==** 5]["AgeGroup"].mode() # Adult rare\_age **=** train[train["Title"] **==** 6]["AgeGroup"].mode() # Adult

age\_title\_mapping **=** {1: "Young Adult", 2: "Student",

3: "Adult", 4: "Baby", 5: "Adult", 6: "Adult"}

**for** x **in** range(len(train["AgeGroup"])):

**if** train["AgeGroup"][x] **==** "Unknown":

train["AgeGroup"][x] **=** age\_title\_mapping[train["Title"][x]]

**for** x **in** range(len(test["AgeGroup"])):

**if** test["AgeGroup"][x] **==** "Unknown":

test["AgeGroup"][x] **=** age\_title\_mapping[test["Title"][x]]

Now assign a numerical value to each age category. Once we have mapped the age into different categories we do not need the age feature. Hence drop

Python3

# map each Age value to a numerical value

age\_mapping **=** {'Baby': 1, 'Child': 2, 'Teenager': 3,

'Student': 4, 'Young Adult': 5, 'Adult': 6,

'Senior': 7}

train['AgeGroup'] **=** train['AgeGroup'].map(age\_mapping)

test['AgeGroup'] **=** test['AgeGroup'].map(age\_mapping)

train.head()

* dropping the Age feature for now, might change train **=** train.drop(['Age'], axis**=**1)

test **=** test.drop(['Age'], axis**=**1)

Drop the name feature since it contains no more useful information.

Python3

train **=** train.drop(['Name'], axis**=**1)

test **=** test.drop(['Name'], axis**=**1)

Assign numerical values to sex and embarks categories\

Python3

sex\_mapping **=** {"male": 0, "female": 1}

train['Sex'] **=** train['Sex'].map(sex\_mapping)

test['Sex'] **=** test['Sex'].map(sex\_mapping)

embarked\_mapping **=** {"S": 1, "C": 2, "Q": 3}

train['Embarked'] **=** train['Embarked'].map(embarked\_mapping)

test['Embarked'] **=** test['Embarked'].map(embarked\_mapping)

Fill in the missing Fare value in the test set based on the mean fare for that P-class

Python3

for x in range(len(test["Fare"])):

    if pd.isnull(test["Fare"][x]):

        pclass = test["Pclass"][x]  # Pclass = 3

        test["Fare"][x] = round(

            train[train["Pclass"] == pclass]["Fare"].mean(), 4)

# map Fare values into groups of

# numerical values

train['FareBand'] = pd.qcut(train['Fare'], 4,

                            labels=[1, 2, 3, 4])

test['FareBand'] = pd.qcut(test['Fare'], 4,

                           labels=[1, 2, 3, 4])

# drop Fare values

train = train.drop(['Fare'], axis=1)

test = test.drop(['Fare'], axis=1)

Now we are done with the feature engineering

Model Training

We will be using Random forest as the algorithm of choice to perform model training. Before that, we will split the data in an 80:20 ratio as a train-test split. For that, we will use the train\_test\_split() from the sklearn library.

Python3

**from** sklearn.model\_selection **import** train\_test\_split

* Drop the Survived and PassengerId
* column from the trainset

predictors **=** train.drop(['Survived', 'PassengerId'], axis**=**1)

target **=** train["Survived"]

x\_train, x\_val, y\_train, y\_val **=** train\_test\_split( predictors, target, test\_size**=**0.2, random\_state**=**0)

Now import the random forest function from the ensemble module of sklearn and for the training set.

Python3

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

randomforest = RandomForestClassifier()

# Fit the training data along with its output

randomforest.fit(x\_train, y\_train)

y\_pred = randomforest.predict(x\_val)

# Find the accuracy score of the model

acc\_randomforest = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_randomforest)

With this, we got an accuracy of 83.25%

Prediction

We are provided with the testing dataset on which we have to perform the prediction. To predict, we will pass the test dataset into our trained model and save it into a CSV file containing the information, passengerid and survival. PassengerId will be the passengerid of the passengers in the test data and the survival will column will be either 0 or 1.

Python3

ids **=** test['PassengerId']

predictions **=** randomforest.predict(test.drop('PassengerId', axis**=**1))

* set the output as a dataframe and convert
* to csv file named resultfile.csv

output **=** pd.DataFrame({'PassengerId': ids, 'Survived': predictions})

output.to\_csv('resultfile.csv', index**=**False)

This will create a resultfile.csv which looks like this

