**EXPLORATORY DATA ANALYSIS WITH PYTHON**

For this EDA we will be working with the Titanic Data Set from Kaggle. This is a very famous data set and very often is a student's first step in machine learning.

We'll be trying to predict a classification – survival or deceased. Let's begin our understanding of implementing Logistic Regression in Python for classification.

We'll use a "semi-cleaned" version of the titanic data set, if you use the data set hosted directly on Kaggle, you may need to do some additional cleaning which is not shown in this lecture notebook.

**IMPORT LIBRARIES**

Let’s import some libraries to get started

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

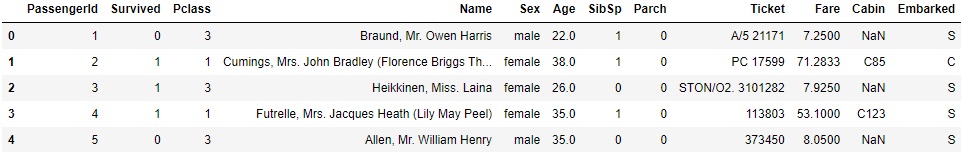
%matplotlib inline

**READ DATA**

Let’s start by reading in the titanic\_train.csv file into a pandas dataframe.

train = pd.read\_csv(‘titanic\_train.csv’)

train.head()



We can see in our data we have different features like ‘PassengerId’, ‘Survived’, ‘Pclass’, ‘Name’, ‘Sex’, ‘Age’, ‘SibSp’(This stands for Siblings and Spouse), ‘Parch’(This stands for Parent and Child), ‘Ticket’, ‘Cabin’, ‘Embarked’.

The main aim of this dataset is that we have to predict, based on the information we have, whether the passenger has survived or not

**EXPLORATORY DATA ANALYSIS**

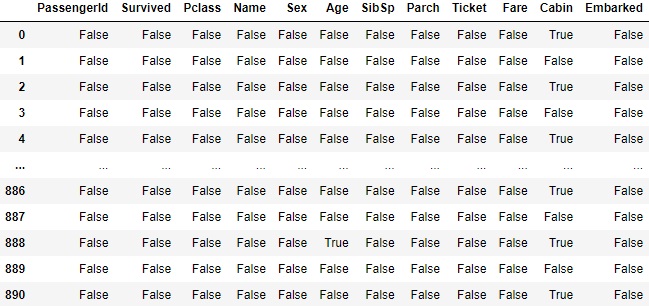
First of all we will not consider ‘PassengerId’, ‘Name’ and ‘Ticket’ feature as they contains unique values.

Now let’s begin some exploratory data analysis. We’ll start by finding out how many NaN values are present. NaN value is basically the null value or missing dataset. So we need to check for missing data in our dataset.

**MISSING DATA**

We can check missing data with help of in – built function isnull(). isnull() function helps to find out whether each and every value in each and every row is either true or false. If it is true, that means the value is null.

train.isnull()



We can see over here in the first row of ‘Cabin’ feature we have the value as true. If we check in the head section we can see there the value is NaN. Similarly there are many columns and rows which may have values as true or false. But is this a good way of finding missing values. It becomes difficult as if we have a dataset with thousands of rows, we have to scroll down to see them all one by one. Also Jupyter Notebook doesn’t displays whole dataset. It skips some rows. In order to get exact number of missing values we can add .sum() function to isnull() function.

train.isnull().sum()

Survived 0

Pclass 0

Name 0

Sex 0

Age 177

SibSp 0

Parch 0

Ticket 0

Fare 0

Cabin 687

Embarked 2

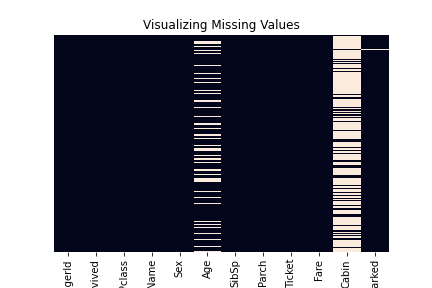
dtype: int64

We can see the ‘Age’, ‘Cabin’ and ‘Embarked’ feature has 177, 687 and 2 missing values respectively. We can also visualise these missing values on a seaborn heatmap.

sns.heatmap(train.isnull(),yticklabels =False,cbar=False)

plt.title('Visualizing Missing Values')

plt.savefig('Visualizing Missing Values.png')



We can see in the heatmap, the white lines are missing values. We can observe most of the missing values are in ‘Age’ and ‘Cabin’ feature.

Roughly 20 percent of the ‘Age’ data is missing. The proportion of ‘Age’ missing is likely small enough for reasonable replacement with some form of imputation.

Looking at the ‘Cabin’ column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0". Let’s continue by visualizing some more features of the dataset by using seaborn.

**UNIVARIATE AND BIVARIATE ANALYSIS OF FEATURES OF DATASET**

We can further explore the data by analysing each (or useful) features separately and by analysing relation between 2 or more features. This can be done by visualizing the features using seaborn or matplotlib libraries. We will be using the seaborn library.

First we will visualize the target feature ‘Survived’ using the count plot of seaborn library. In the ‘Survived’ feature we can see values in form of 0 and 1. Here 0 means person was unable to survive while 1 means person has survived.

**Univariate Analysis of ‘Survived’ feature**

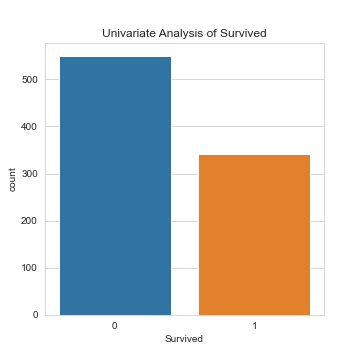
plt.figure(figsize=(5,5))

sns.set\_style('whitegrid')

sns.countplot(x='Survived',data=train)

plt.title('Univariate Analysis of Survived')

plt.savefig('Univariate Analysis of Survived.png')



We can see in the count plot above more than 500 people didn’t survive and more than 300 people survived. Note that generally the univariate analysis is done for target feature only but we can also analyse other features like ‘SibSp’ or ‘Fare’.

Now we will visualize relation of ‘Survived’ feature with other features of the dataset. Basically we will be doing bivariate analysis of ‘Survived’ feature with other features one by one.

**Bivariate analysis of ‘Survived’ and ‘Sex’ features**

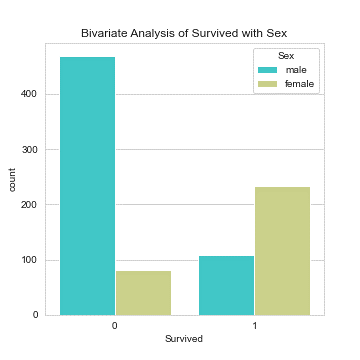
plt.figure(figsize=(5,5))

sns.set\_style('whitegrid')

sns.countplot(x='Survived',hue='Sex',data=train,palette='rainbow')

plt.title('Bivariate Analysis of Survived with Sex')

plt.savefig('Bivariate Analysis of Survived.png')



We can observe that most of the females have survived and most of the males haven’t survived.

**Bivariate Analysis of ‘Survived’ and ‘Pclass’ feature**

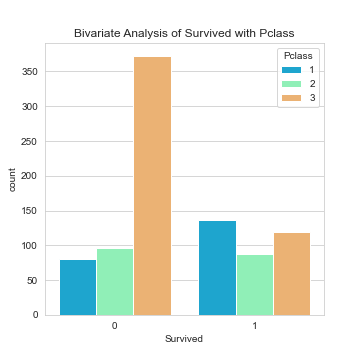
plt.figure(figsize=(5,5))

sns.set\_style('whitegrid')

sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')

plt.title('Bivariate Analysis of Survived with Pclass')

plt.savefig('Bivariate Analysis of Survived.png')



We can observe that most of the first class passenger survived while most of the third class passenger couldn’t survived.

Now we will see if the ‘Age’ feature(having null values) have a normal distribution or not. We can use the distribution plot of seaborn or a histogram to visualise it. Also we will be dropping the null values during this visualization.

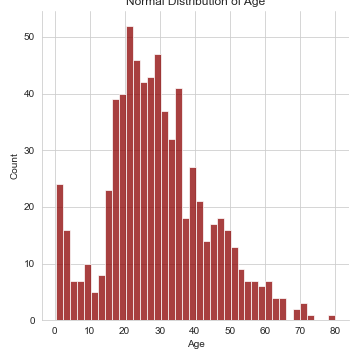
plt.figure(figsize=(5,5))

sns.set\_style('whitegrid')

sns.displot(train['Age'].dropna(),kde=False,color='darkred',bins=40)

plt.title('Bivariate Analysis of Survived with Pclass')

plt.savefig('Bivariate Analysis of Survived.png')



We can see maximum number of people were between the age group of 17 to 30 and there are very few people above age of 60. We can also observe it is also a kind of normal distribution as it is forming a bell curve.

**DATA CLEANING**

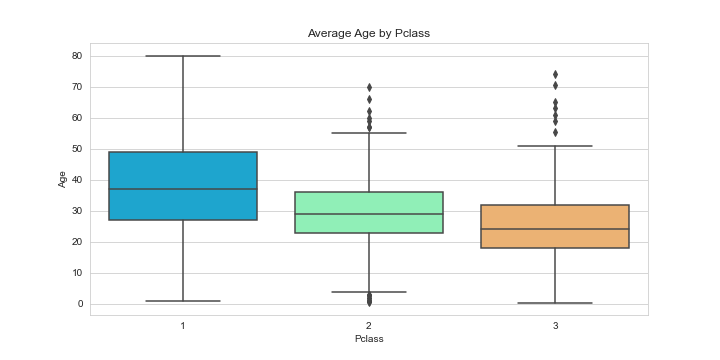
We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class. For example:

plt.figure(figsize=(15,10))

sns.boxplot(x='Pclass',y='Age',data=train,palette='rainbow')

plt.title('Average Age by Pclass')

plt.savefig('Average Age by Pclass.png')



We can see average age of First Class, Second Class and Third Class Passengers is around 37, 29 and 24 respectively. Based on this passenger class and age we are going to replace the NaN value in the ‘Age’ column. In order to do that we have to create a function.

def impute\_age(cols):

if pd.isnull('Age'):

if Pclass == 1:

return 37

elif Pclass == 2:

return 29

else:

return 24

else:

return 'Age'

Here the function we have created states that if the passenger class is first the NaN value will be replaced by average age of passenger of first class i.e. 37 same is for second and third class passengers. If there is no missing value then it will return the value.

Now we will apply this function in the age column;

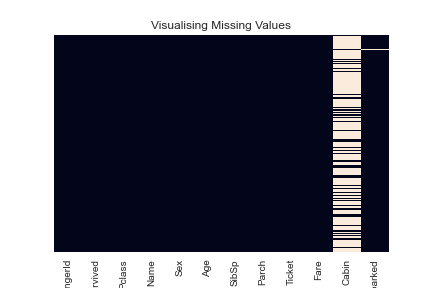
train['Age'] = train[['Age','Pclass']].apply(impute\_agee,axis=1)

Now we will again use the heatmap to check for missing values;

sns.heatmap(train.isnull(),cbar=False,yticklabels=False)

plt.title('Visualising Missing Values')

plt.savefig('Visualising Missing Values.png')



We can see now there are no missing values in ‘Age’ column. The next thing is we have to replace NaN values in ‘Cabin’ but the problem with this feature is that there are too many missing values and in order to fill them we have to use feature engineering but for now we will drop this entire column.

train.drop('Cabin',axis=1,inplace=True)

train.head()

Also there were 2 missing values in ‘Embarked’ column. We will fill them with ‘Q’

train['Embarked'] = train['Embarked'].fillna('Q')

So if we check the heatmap again we can see there are no missing values. Now we have to change the categorical variables into integer format before we start building the model. We are going to handle these categorical features by using an in – built function get dummies of pandas.

We will replace objects in ‘Embarked’ and ‘Sex’ features with integers and store them into new columns

embarked = pd.get\_dummies(train['Embarked'])

sex = pd.get\_dummies(train['Sex'])

We will drop all the columns that are not required i.e. 'Sex', 'Embarked', 'Name' and 'Ticket'

train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)

train.head()

Now we will append the sex and embarked feature into our training data before we start building our model. We will use concatenation function of pandas library.

train=pd.concat([train,sex,embarked],axis=1)

train.head()

**BUILDING A LOGISTIC REGRESSION MODEL**

Let's start by splitting our data into a training set and test set.

**TRAIN TEST SPLIT**

Before doing the train test split we will create a train data where we will drop the target variable i.e. ‘Survived’ from the training dataset

train.drop('Survived',axis=1)

train.head()

Now we have complete training dataset

train['Survived'].head()

This is our output dataset

Now we will split the dataset into train and test data

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train.drop('Survived',

axis=1),train['Survived'],

test\_size=.30,random\_state=101)

**TRAINING AND PREDICTING**

We will be using logistic regression model for making predictions

from sklearn.linear\_model import LogisticRegression

logmodel = LogisticRegression()

logmodel.fit(X\_train,y\_train)

Now we will make prediction using our model and test data

predictions = logmodel.predict(X\_test)

To check accuracy of our prediction we will use confusion metrix and accuracy score

from sklearn.metrics import confusion\_matrix

accuracy = confusion\_matrix(y\_test,predictions)

accuracy

array([[134, 20],

[ 40, 74]], dtype=int64)

Here diagonal values are right and else are wrong

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(y\_test,predictions)

accuracy

0.7761194029850746

We can see we have a good prediction score just by applying a simple logistic regression model. We can also try different techniques to build our model.

Now finally let’s print our predictions

predictions

array([0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,

1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,

0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,

1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,

0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,

0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,

1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,

0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,

1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,

0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1,

0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,

1, 0, 0, 1], dtype=int64)