#Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression and random forest regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

```
df.head()

   Unnamed: 0                    key  fare_amount
pickup_datetime  \
0    24238194  2015-05-07 19:52:06          7.5  2015-05-07 19:52:06
UTC
1    27835199  2009-07-17 20:04:56          7.7  2009-07-17 20:04:56
UTC
2    44984355  2009-08-24 21:45:00         12.9  2009-08-24 21:45:00
UTC
3    25894730  2009-06-26 08:22:21          5.3  2009-06-26 08:22:21
UTC
4    17610152  2014-08-28 17:47:00         16.0  2014-08-28 17:47:00
UTC

   pickup_longitude  pickup_latitude  dropoff_longitude
dropoff_latitude  \
0        -73.999817        40.738354         -73.999512
40.723217
1        -73.994355        40.728225         -73.994710
40.750325
2        -74.005043        40.740770         -73.962565
40.772647
3        -73.976124        40.790844         -73.965316
40.803349
4        -73.925023        40.744085         -73.973082
40.761247
```

```
   passenger_count
0                1
1                1
2                1
3                3
4                5
```

```python
df.info() #To get the required information of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

```python
df.columns #TO get number of columns in the dataset
```

```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

```python
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column 
as it isn't required

df.head()
```

```
   fare_amount          pickup_datetime  pickup_longitude  
pickup_latitude  \
0          7.5  2015-05-07 19:52:06 UTC        -73.999817  
40.738354   
1          7.7  2009-07-17 20:04:56 UTC        -73.994355  
40.728225   
2         12.9  2009-08-24 21:45:00 UTC        -74.005043  
40.740770   
3          5.3  2009-06-26 08:22:21 UTC        -73.976124  
40.790844   
4         16.0  2014-08-28 17:47:00 UTC        -73.925023  
40.744085   

   dropoff_longitude  dropoff_latitude  passenger_count  
```

```
0         -73.999512        40.723217                1
1         -73.994710        40.750325                1
2         -73.962565        40.772647                1
3         -73.965316        40.803349                3
4         -73.973082        40.761247                5
```

```
df.shape #To get the total (Rows,Columns)
```

```
(200000, 7)
```

```
df.dtypes #To get the type of each column
```

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        200000 non-null  float64
 1   pickup_datetime    200000 non-null  object 
 2   pickup_longitude   200000 non-null  float64
 3   pickup_latitude    200000 non-null  float64
 4   dropoff_longitude  199999 non-null  float64
 5   dropoff_latitude   199999 non-null  float64
 6   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
```

```
df.describe() #To get statistics of each columns
```

```
         fare_amount  pickup_longitude  pickup_latitude
dropoff_longitude  \
count  200000.000000     200000.000000    200000.000000
199999.000000
mean       11.359955        -72.527638        39.935885         -
72.525292
std         9.901776         11.437787         7.720539
13.117408
min       -52.000000      -1340.648410       -74.015515       -
3356.666300
25%         6.000000        -73.992065        40.734796         -
73.991407
```

```
50%          8.500000        -73.981823        40.752592          -
73.980093
75%         12.500000        -73.967153        40.767158          -
73.963659
max        499.000000         57.418457      1644.421482
1153.572603

       dropoff_latitude  passenger_count
count     199999.000000    200000.000000
mean          39.923890         1.684535
std            6.794829         1.385997
min         -881.985513         0.000000
25%           40.733823         1.000000
50%           40.753042         1.000000
75%           40.768001         2.000000
max          872.697628       208.000000
```

Filling Missing values

```
df.isnull().sum()

fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64

df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inpl
ace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),
inplace = True)

df.isnull().sum()

fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64

df.dtypes

fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
```

```
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime,
errors='coerce')

df.dtypes

fare_amount                          float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude                     float64
pickup_latitude                      float64
dropoff_longitude                    float64
dropoff_latitude                     float64
passenger_count                        int64
dtype: object
```

To segregate each time of date and time

```
df= df.assign(hour = df.pickup_datetime.dt.hour,
             day= df.pickup_datetime.dt.day,
             month = df.pickup_datetime.dt.month,
             year = df.pickup_datetime.dt.year,
             dayofweek = df.pickup_datetime.dt.dayofweek)

df.head()

   fare_amount           pickup_datetime  pickup_longitude
pickup_latitude  \
0          7.5 2015-05-07 19:52:06+00:00        -73.999817
40.738354
1          7.7 2009-07-17 20:04:56+00:00        -73.994355
40.728225
2         12.9 2009-08-24 21:45:00+00:00        -74.005043
40.740770
3          5.3 2009-06-26 08:22:21+00:00        -73.976124
40.790844
4         16.0 2014-08-28 17:47:00+00:00        -73.925023
40.744085

   dropoff_longitude  dropoff_latitude  passenger_count  hour  day
month  \
0         -73.999512         40.723217                1    19    7
5
1         -73.994710         40.750325                1    20   17
7
2         -73.962565         40.772647                1    21   24
8
3         -73.965316         40.803349                3     8   26
```

```
6
4         -73.973082         40.761247                5    17   28
8

   year  dayofweek
0  2015          3
1  2009          4
2  2009          0
3  2009          4
4  2014          3
```

```python
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

df = df.drop('pickup_datetime',axis=1)

df.head()
```

```
   fare_amount  pickup_longitude  pickup_latitude  
dropoff_longitude  \
0          7.5        -73.999817        40.738354         -73.999512

1          7.7        -73.994355        40.728225         -73.994710

2         12.9        -74.005043        40.740770         -73.962565

3          5.3        -73.976124        40.790844         -73.965316

4         16.0        -73.925023        40.744085         -73.973082


   dropoff_latitude  passenger_count  hour  day  month  year  
dayofweek
0         40.723217                1    19    7      5  2015  
3
1         40.750325                1    20   17      7  2009  
4
2         40.772647                1    21   24      8  2009  
0
3         40.803349                3     8   26      6  2009  
4
4         40.761247                5    17   28      8  2014  
3
```

```python
df.dtypes
```

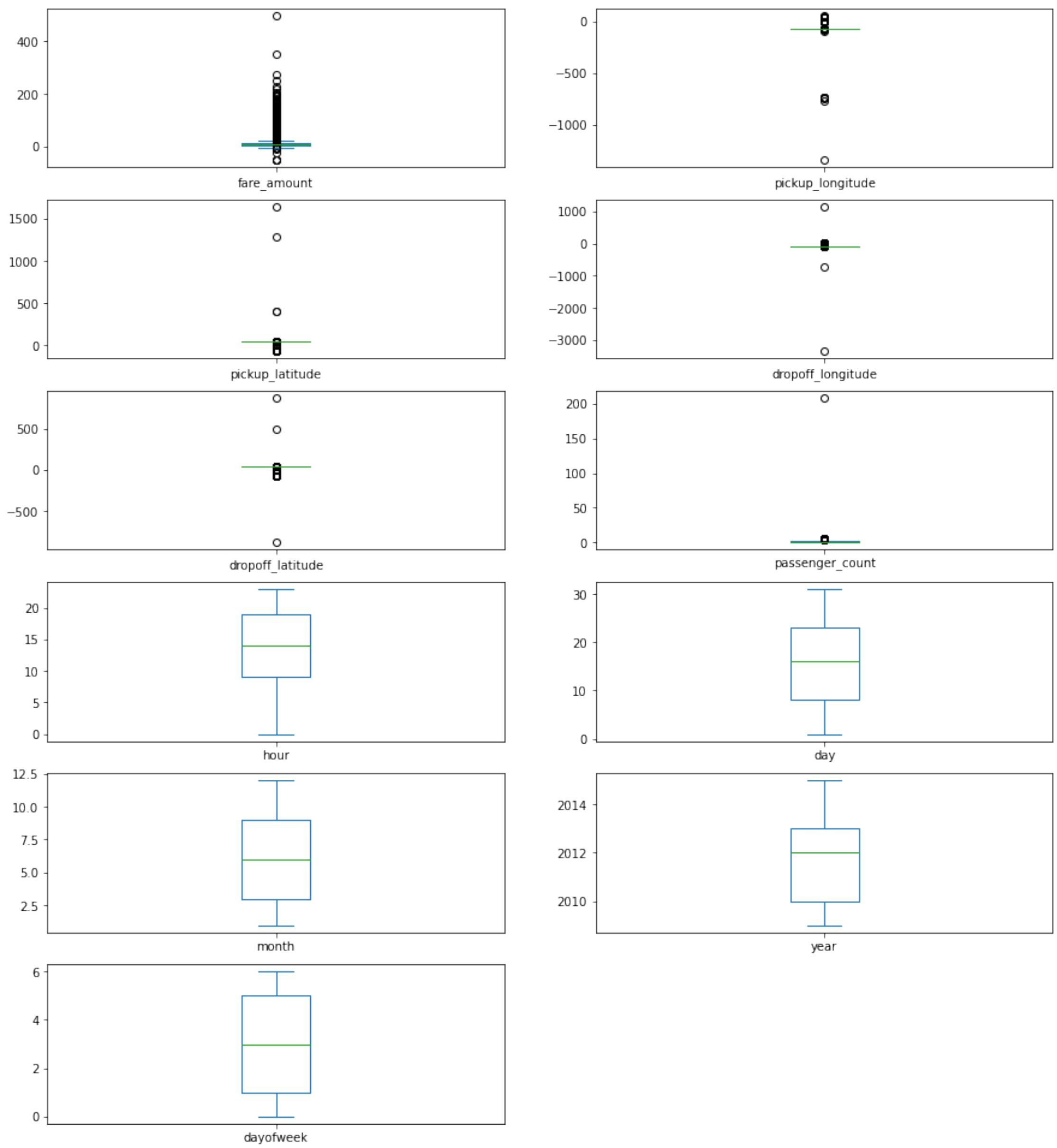
```
fare_amount          float64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
```

```
dropoff_latitude     float64
passenger_count        int64
hour                   int64
day                    int64
month                  int64
year                   int64
dayofweek              int64
dtype: object
```

Checking outliers and filling them

```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))
#Boxplot to check the outliers
```

```
fare_amount             AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude         AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude        AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                    AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                   AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek               AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```

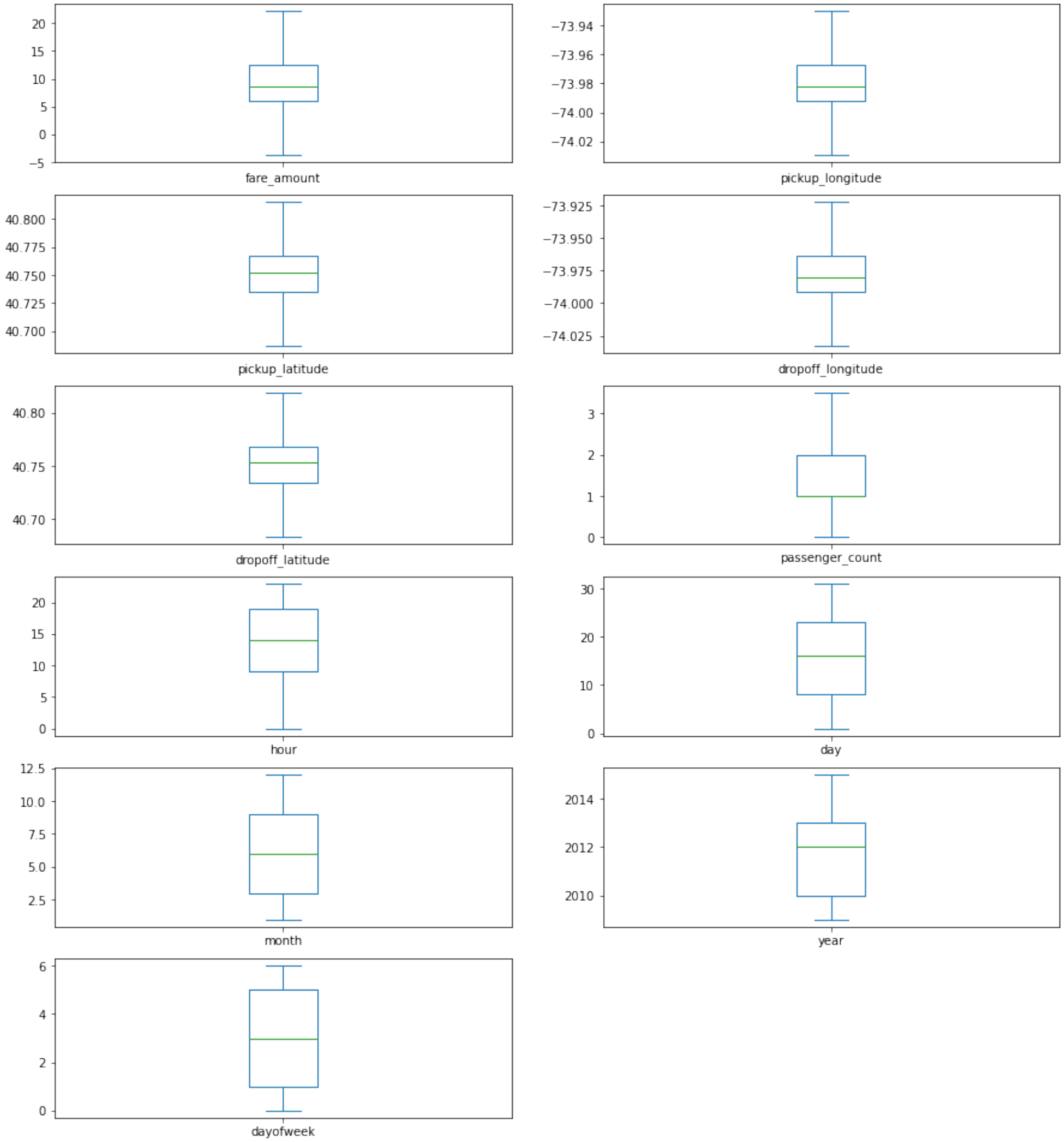



```python
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1
```

```python
df = treat_outliers_all(df , df.iloc[: , 0::])
```

```python
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))
#Boxplot shows that dataset is free from outliers
```

```
fare_amount              AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude      AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude          AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude     AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude         AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count       AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                     AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                   AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                    AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                  AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek                AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```




```
#pip install haversine
import haversine as hs  #Calculate the distance using Haversine to 
calculate the distance between to points. Can't use Eucladian as it is 
for flat surface.
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude']
[pos],df['pickup_latitude'][pos],df['dropoff_longitude']
[pos],df['dropoff_latitude'][pos]]
        loc1=(lati1,long1)
        loc2=(lati2,long2)
        c = hs.haversine(loc1,loc2)
        travel_dist.append(c)
```

```
print(travel_dist)
df['dist_travel_km'] = travel_dist
df.head()
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)


   fare_amount  pickup_longitude  pickup_latitude
dropoff_longitude  \
0          7.5        -73.999817        40.738354         -73.999512

1          7.7        -73.994355        40.728225         -73.994710

2         12.9        -74.005043        40.740770         -73.962565

3          5.3        -73.976124        40.790844         -73.965316

4         16.0        -73.929786        40.744085         -73.973082


   dropoff_latitude  passenger_count  hour  day  month  year
dayofweek  \
0         40.723217              1.0    19    7      5  2015
3
1         40.750325              1.0    20   17      7  2009
4
2         40.772647              1.0    21   24      8  2009
0
3         40.803349              3.0     8   26      6  2009
4
4         40.761247              3.5    17   28      8  2014
3

   dist_travel_km
0        1.683325
1        2.457593
2        5.036384
3        1.661686
4        4.116088
```

```
#Uber doesn't travel over 130 kms so minimize the distance
df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
print("Remaining observastions in the dataset:", df.shape)
```

```
Remaining observastions in the dataset: (200000, 12)
```

```
#Finding inccorect latitude (Less than or greater than 90) and
longitude (greater than or less than 180)
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |
(df.pickup_latitude < -90) |
                                   (df.dropoff_latitude > 90) |
(df.dropoff_latitude < -90) |
                                   (df.pickup_longitude > 180) |
(df.pickup_longitude < -180) |
                                   (df.dropoff_longitude > 90) |
(df.dropoff_longitude < -90)
                                    ]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')

df.head()
```

```
   fare_amount  pickup_longitude  pickup_latitude
dropoff_longitude  \
0          7.5        -73.999817        40.738354         -73.999512

1          7.7        -73.994355        40.728225         -73.994710

2         12.9        -74.005043        40.740770         -73.962565

3          5.3        -73.976124        40.790844         -73.965316

4         16.0        -73.929786        40.744085         -73.973082


   dropoff_latitude  passenger_count  hour  day  month  year
dayofweek  \
0         40.723217              1.0    19    7      5  2015
3
1         40.750325              1.0    20   17      7  2009
4
2         40.772647              1.0    21   24      8  2009
0
3         40.803349              3.0     8   26      6  2009
4
4         40.761247              3.5    17   28      8  2014
3

   dist_travel_km
0        1.683325
1        2.457593
```

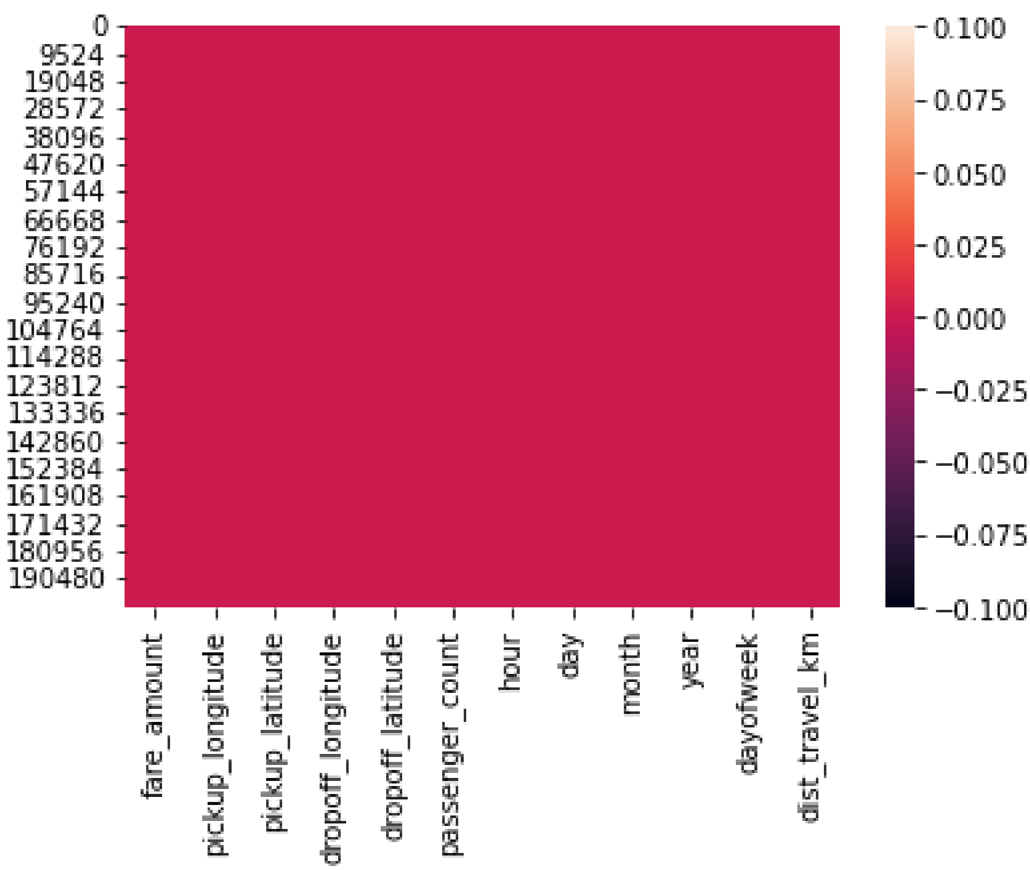
```
2        5.036384
3        1.661686
4        4.116088
```

```
df.isnull().sum()
```

```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
hour                 0
day                  0
month                0
year                 0
dayofweek            0
dist_travel_km       0
dtype: int64
```

```
sns.heatmap(df.isnull()) #Free for null values
```

```
<AxesSubplot:>
```




```
corr = df.corr() #Function to find the correlation
```

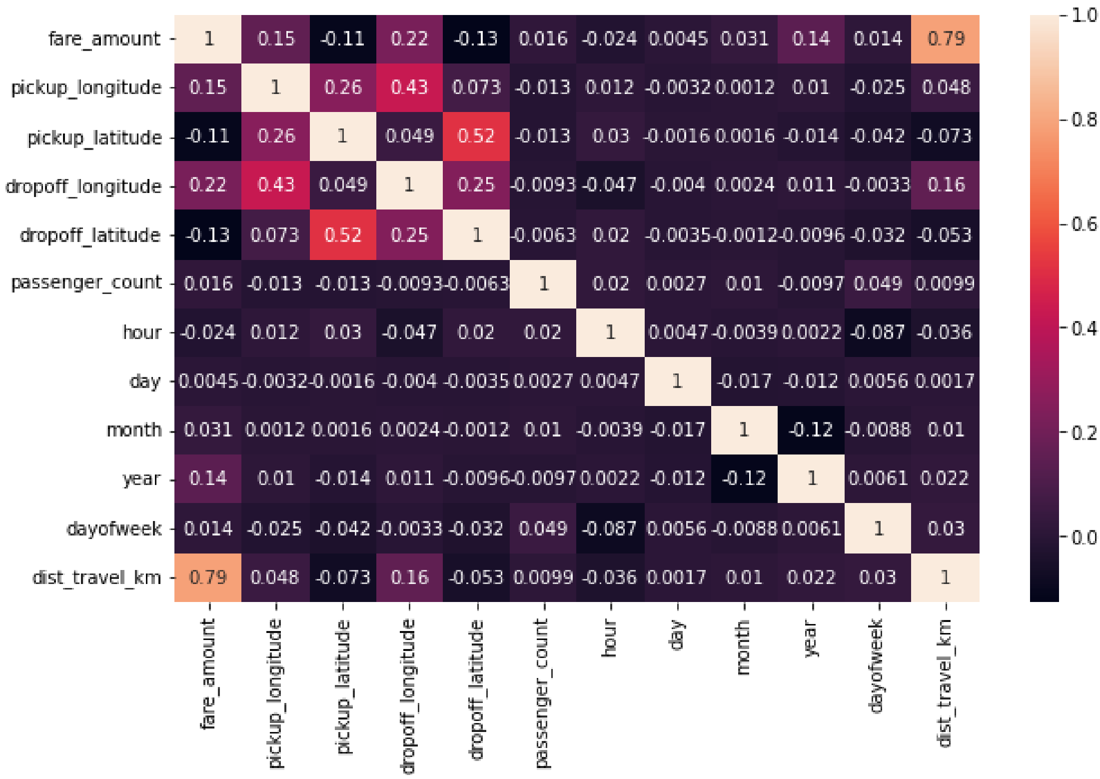
```
corr

                    fare_amount  pickup_longitude  pickup_latitude  \
fare_amount            1.000000          0.154069        -0.110842
pickup_longitude       0.154069          1.000000         0.259497
pickup_latitude       -0.110842          0.259497         1.000000
dropoff_longitude      0.218675          0.425619         0.048889
dropoff_latitude      -0.125898          0.073290         0.515714
passenger_count        0.015778         -0.013213        -0.012889
hour                  -0.023623          0.011579         0.029681
day                    0.004534         -0.003204        -0.001553
month                  0.030817          0.001169         0.001562
year                   0.141277          0.010198        -0.014243
dayofweek              0.013652         -0.024652        -0.042310
dist_travel_km         0.786385          0.048446        -0.073362

                    dropoff_longitude  dropoff_latitude
passenger_count  \
fare_amount                  0.218675         -0.125898
0.015778
pickup_longitude             0.425619          0.073290        -
0.013213
pickup_latitude              0.048889          0.515714        -
0.012889
dropoff_longitude            1.000000          0.245667        -
0.009303
dropoff_latitude             0.245667          1.000000        -
0.006308
passenger_count             -0.009303         -0.006308
1.000000
hour                        -0.046558          0.019783
0.020274
day                         -0.004007         -0.003479
0.002712
month                        0.002391         -0.001193
0.010351
year                         0.011346         -0.009603        -
0.009749
dayofweek                   -0.003336         -0.031919
0.048550
dist_travel_km               0.155191         -0.052701
0.009884

                        hour       day     month      year
dayofweek  \
fare_amount       -0.023623  0.004534  0.030817  0.141277   0.013652

pickup_longitude   0.011579 -0.003204  0.001169  0.010198  -0.024652

pickup_latitude    0.029681 -0.001553  0.001562 -0.014243  -0.042310
```

dropoff_longitude	-0.046558	-0.004007	0.002391	0.011346	-0.003336
dropoff_latitude	0.019783	-0.003479	-0.001193	-0.009603	-0.031919
passenger_count	0.020274	0.002712	0.010351	-0.009749	0.048550
hour	1.000000	0.004677	-0.003926	0.002156	-0.086947
day	0.004677	1.000000	-0.017360	-0.012170	0.005617
month	-0.003926	-0.017360	1.000000	-0.115859	-0.008786
year	0.002156	-0.012170	-0.115859	1.000000	0.006113
dayofweek	-0.086947	0.005617	-0.008786	0.006113	1.000000
dist_travel_km	-0.035708	0.001709	0.010050	0.022294	0.030382

	dist_travel_km
fare_amount	0.786385
pickup_longitude	0.048446
pickup_latitude	-0.073362
dropoff_longitude	0.155191
dropoff_latitude	-0.052701
passenger_count	0.009884
hour	-0.035708
day	0.001709
month	0.010050
year	0.022294
dayofweek	0.030382
dist_travel_km	1.000000

```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values
means highly correlated)
```

<AxesSubplot:>



Dividing the dataset into feature and target values

```
x =
df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_
latitude','passenger_count','hour','day','month','year','dayofweek','d
ist_travel_km']]

y = df['fare_amount']
```

Dividing the dataset into training and testing dataset

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()

regression.fit(X_train,y_train)

LinearRegression()

regression.intercept_ #To find the linear intercept

3683.734379131267

regression.coef_ #To find the linear coeeficient

array([ 2.54690644e+01, -7.25031311e+00,  2.01910609e+01, -
1.81689799e+01,
```

```
        6.49318535e-02,  8.88740039e-03,  3.96976218e-03,
6.07701750e-02,
        3.64995448e-01, -3.34018868e-02,  1.84796864e+00])
```

```
prediction = regression.predict(X_test) #To predict the target values
```

```
print(prediction)
```

```
[ 6.92808422  5.50169187  7.29033891 ...  7.34427831 11.48600676
  8.04489363]
```

```
y_test
```

```
23033      8.0
166557     4.5
188533     8.0
175085     7.5
69692     11.4
          ...
22917      8.1
42396     12.9
25947      8.0
66067      8.5
20658      8.5
Name: fare_amount, Length: 66000, dtype: float64
```

Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

```
from sklearn.metrics import r2_score
```

```
r2_score(y_test,prediction)
```

```
0.6640797581905353
```

```
from sklearn.metrics import mean_squared_error
```

```
MSE = mean_squared_error(y_test,prediction)
```

```
MSE
```

```
9.92519776977491
```

```
RMSE = np.sqrt(MSE)
```

```
RMSE
```

```
3.15042818832217
```

Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means
number of trees you want to build before making the prediction
```

```
rf.fit(X_train,y_train)
```

```
y_pred = rf.predict(X_test)

y_pred
```

Metrics evaluatin for Random Forest

```
R2_Random = r2_score(y_test,y_pred)

R2_Random

MSE_Random = mean_squared_error(y_test,y_pred)

MSE_Random

RMSE_Random = np.sqrt(MSE_Random)

RMSE_Random
```

Assignment 2

1. Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics

df=pd.read_csv('emails.csv')

df.head()

  Email No.  the  to  ect  and  for  of    a  you  hou  ...  connevey
jay  \
0   Email 1    0   0    1    0    0   0    2    0    0  ...         0
0
1   Email 2    8  13   24    6    6   2  102    1   27  ...         0
0
2   Email 3    0   0    1    0    0   0    8    0    0  ...         0
0
3   Email 4    0   5   22    0    5   1   51    2   10  ...         0
0
4   Email 5    7   6   17    1    5   2   57    0    9  ...         0
0

   valued  lay  infrastructure  military  allowing  ff  dry
Prediction
0       0    0               0         0         0   0    0
0
1       0    0               0         0         0   1    0
0
2       0    0               0         0         0   0    0
0
3       0    0               0         0         0   0    0
0
4       0    0               0         0         0   1    0
0

[5 rows x 3002 columns]
```

```
df.columns
```

```
Index(['Email No.', 'the', 'to', 'ect', 'and', 'for', 'of', 'a',
'you', 'hou',
       ...
       'connevey', 'jay', 'valued', 'lay', 'infrastructure',
'military',
       'allowing', 'ff', 'dry', 'Prediction'],
      dtype='object', length=3002)
```

```
df.isnull().sum()
```

```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

```
df.dropna(inplace = True)
```

```
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
```

```
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 42)
```

##KNN classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
print("Prediction",y_pred)
```

```
Prediction [0 0 1 ... 1 1 1]
```

```
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
KNN accuracy =  0.8009020618556701
```

```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

```
Confusion matrix [[804 293]
 [ 16 439]]
```

SVM classifier

```
# cost C = 1
model = SVC(C = 1)

# fit
model.fit(X_train, y_train)

# predict
y_pred = model.predict(X_test)
```

```
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
array([[1091,    6],
       [  90,  365]], dtype=int64)
```

```
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
SVM accuracy =  0.9381443298969072
```

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

```
df = pd.read_csv("Churn_Modelling.csv")
```

Preprocessing.

```
df.head()
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age
\
0          1    15634602  Hargrave          619    France  Female   42

1          2    15647311      Hill          608     Spain  Female   41

2          3    15619304      Onio          502    France  Female   42

3          4    15701354      Boni          699    France  Female   39

4          5    15737888  Mitchell          850     Spain  Female   43


   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
```

```
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

```
df.shape
```

```
(10000, 14)
```

```
df.describe()
```

```
         RowNumber    CustomerId   CreditScore           Age
Tenure  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000
10000.000000
mean    5000.50000  1.569094e+07    650.528800     38.921800
5.012800
std     2886.89568  7.193619e+04     96.653299     10.487806
2.892174
min        1.00000  1.556570e+07    350.000000     18.000000
0.000000
25%     2500.75000  1.562853e+07    584.000000     32.000000
3.000000
50%     5000.50000  1.569074e+07    652.000000     37.000000
5.000000
75%     7500.25000  1.575323e+07    718.000000     44.000000
7.000000
max    10000.00000  1.581569e+07    850.000000     92.000000
10.000000

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
75%    127644.240000       2.000000      1.00000        1.000000
max    250898.090000       4.000000      1.00000        1.000000

       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
std       57510.492818      0.402769
min          11.580000      0.000000
25%       51002.110000      0.000000
50%      100193.915000      0.000000
```

```
75%      149388.247500      0.000000
max      199992.480000      1.000000
```

```
df.isnull()
```

```
      RowNumber  CustomerId  Surname  CreditScore  Geography  Gender
Age  \
0         False       False    False        False      False   False
False
1         False       False    False        False      False   False
False
2         False       False    False        False      False   False
False
3         False       False    False        False      False   False
False
4         False       False    False        False      False   False
False
...         ...         ...      ...          ...        ...     ...
...
9995      False       False    False        False      False   False
False
9996      False       False    False        False      False   False
False
9997      False       False    False        False      False   False
False
9998      False       False    False        False      False   False
False
9999      False       False    False        False      False   False
False

      Tenure  Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0      False    False          False      False           False
1      False    False          False      False           False
2      False    False          False      False           False
3      False    False          False      False           False
4      False    False          False      False           False
...      ...      ...            ...        ...             ...
9995   False    False          False      False           False
9996   False    False          False      False           False
9997   False    False          False      False           False
9998   False    False          False      False           False
9999   False    False          False      False           False

      EstimatedSalary  Exited
0               False   False
1               False   False
2               False   False
3               False   False
4               False   False
...               ...     ...
```

```
9995            False   False
9996            False   False
9997            False   False
9998            False   False
9999            False   False

[10000 rows x 14 columns]
```

```
df.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.dtypes
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore',
'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

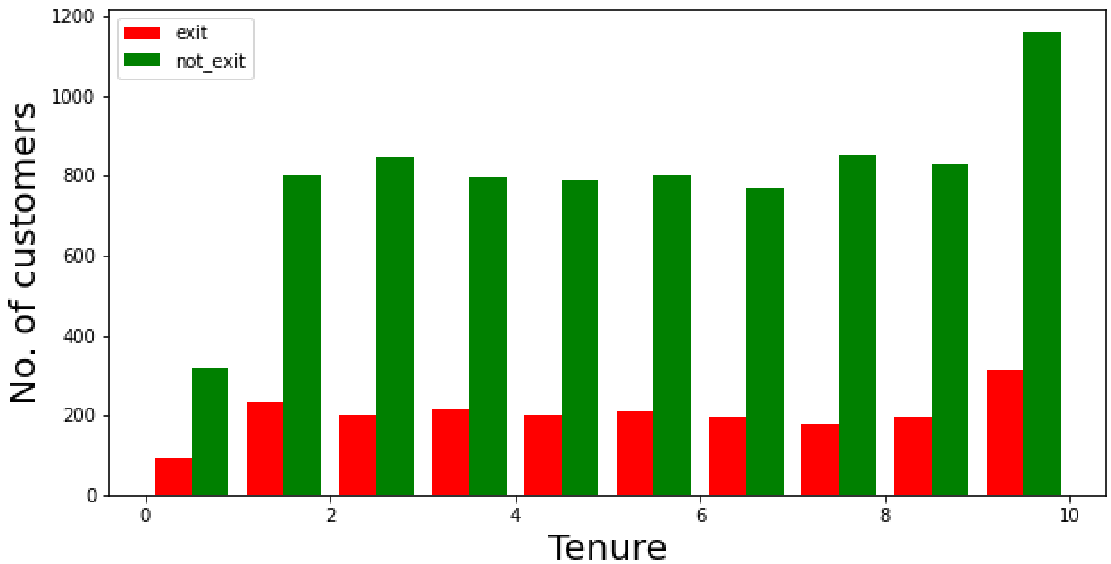
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1)
#Dropping the unnecessary columns
```

```
df.head()
```

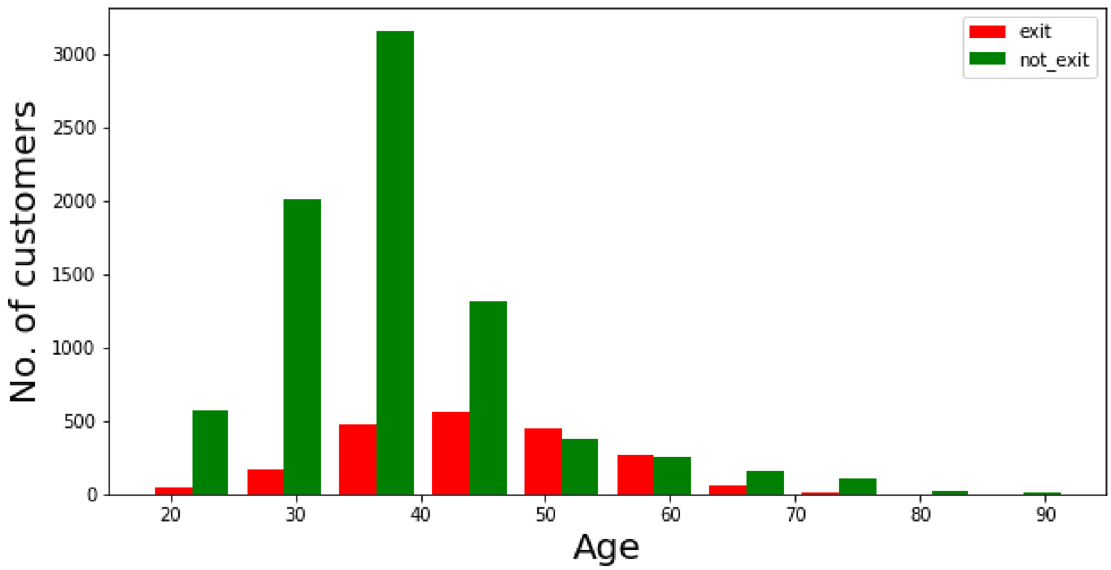
```
   CreditScore Geography  Gender  Age  Tenure    Balance
NumOfProducts  \
0          619    France  Female   42       2       0.00
1
1          608     Spain  Female   41       1   83807.86
1
2          502    France  Female   42       8  159660.80
3
3          699    France  Female   39       1       0.00
2
4          850     Spain  Female   43       2  125510.82
1

   HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1               1        101348.88       1
1          0               1        112542.58       0
2          1               0        113931.57       1
3          0               0         93826.63       0
4          1               1         79084.10       0
```

Visualization

```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 
'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()

df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']

visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```




```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']

visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



Converting the Categorical Variables

```
X =
df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','H
asCrCard','IsActiveMember','EstimatedSalary']]
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)


df = pd.concat([df,gender,states], axis = 1)
```

Splitting the training and testing Dataset

```
df.head()

   CreditScore Geography  Gender  Age  Tenure    Balance
NumOfProducts  \
0          619    France  Female   42       2       0.00
1
1          608     Spain  Female   41       1   83807.86
1
2          502    France  Female   42       8  159660.80
3
3          699    France  Female   39       1       0.00
2
4          850     Spain  Female   43       2  125510.82
1

   HasCrCard  IsActiveMember  EstimatedSalary  Exited  Male  Germany
Spain
0          1               1        101348.88       1     0        0
0
```

```
1          0               1        112542.58       0     0        0
1
2          1               0        113931.57       1     0        0
0
3          0               0         93826.63       0     0        0
0
4          1               1         79084.10       0     0        0
1
```

```
X =
df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard'
,'IsActiveMember','EstimatedSalary','Male','Germany','Spain']]

y = df['Exited']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

Normalizing the values with mean as 0 and Standard Deviation as 1

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train

array([[ 0.30685022, -0.36409498,  1.37874982, ...,  0.92216229,
        -0.5821891 ,  1.76089794],
       [ 0.47071357,  0.40957576, -0.34913058, ...,  0.92216229,
         1.71765497, -0.56789208],
       [-1.23961016, -0.36409498, -0.0035545 , ...,  0.92216229,
        -0.5821891 ,  1.76089794],
       ...,
       [ 0.45023065,  0.02274039, -0.0035545 , ...,  0.92216229,
        -0.5821891 , -0.56789208],
       [-0.0413594 , -0.84763919, -0.0035545 , ..., -1.08440782,
        -0.5821891 , -0.56789208],
       [-1.2293687 , -1.3311834 ,  1.37874982, ...,  0.92216229,
        -0.5821891 , -0.56789208]])

X_test

array([[ 0.2863673 ,  0.98982882, -1.04028274, ...,  0.92216229,
         1.71765497, -0.56789208],
       [ 0.80868173, -0.46080382,  1.37874982, ..., -1.08440782,
        -0.5821891 ,  1.76089794],
       [-0.13353254,  1.18324651, -1.38585882, ...,  0.92216229,
         1.71765497, -0.56789208],
       ...,
```

```
       [-0.31787881,  1.8602084 , -0.0035545 , ...,  0.92216229,
        -0.5821891 , -0.56789208],
       [ 0.81892319,  2.24704378, -1.04028274, ..., -1.08440782,
        -0.5821891 ,  1.76089794],
       [-0.51246654, -0.36409498, -0.34913058, ...,  0.92216229,
        -0.5821891 , -0.56789208]])
```

Building the Classifier Model using Keras

```
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors
initially.

from keras.models import Sequential #To create sequential neural
network
from keras.layers import Dense #To create hidden layers

classifier = Sequential()

#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units =
6,kernel_initializer = "uniform"))

classifier.add(Dense(activation = "relu",units = 6,kernel_initializer
= "uniform"))   #Adding second hidden layers

classifier.add(Dense(activation = "sigmoid",units =
1,kernel_initializer = "uniform")) #Final neuron will be having
siigmoid function

classifier.compile(optimizer="adam",loss =
'binary_crossentropy',metrics = ['accuracy']) #To compile the
Artificial Neural Network. Ussed Binary crossentropy as we just have
only two output

classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in
2nd layer and 1 neuron in last
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dense_1 (Dense)             (None, 6)                 42

 dense_2 (Dense)             (None, 1)                 7
```

```
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the
ANN to training dataset
```

```
Epoch 1/50
700/700 [==============================] - 1s 652us/step - loss:
0.4923 - accuracy: 0.7966
Epoch 2/50
700/700 [==============================] - 1s 763us/step - loss:
0.4267 - accuracy: 0.7966
Epoch 3/50
700/700 [==============================] - 0s 622us/step - loss:
0.4225 - accuracy: 0.7966
Epoch 4/50
700/700 [==============================] - 0s 651us/step - loss:
0.4182 - accuracy: 0.8129
Epoch 5/50
700/700 [==============================] - 0s 645us/step - loss:
0.4156 - accuracy: 0.8253
Epoch 6/50
700/700 [==============================] - 0s 655us/step - loss:
0.4132 - accuracy: 0.8294
Epoch 7/50
700/700 [==============================] - 0s 643us/step - loss:
0.4117 - accuracy: 0.8316
Epoch 8/50
700/700 [==============================] - 0s 633us/step - loss:
0.4102 - accuracy: 0.8331
Epoch 9/50
700/700 [==============================] - 0s 649us/step - loss:
0.4093 - accuracy: 0.8324
Epoch 10/50
700/700 [==============================] - 0s 662us/step - loss:
0.4077 - accuracy: 0.8346
Epoch 11/50
700/700 [==============================] - 0s 680us/step - loss:
0.4071 - accuracy: 0.8343
Epoch 12/50
700/700 [==============================] - 1s 787us/step - loss:
0.4064 - accuracy: 0.8356
Epoch 13/50
700/700 [==============================] - 1s 765us/step - loss:
0.4054 - accuracy: 0.8353
Epoch 14/50
700/700 [==============================] - 1s 754us/step - loss:
```

```
0.4050 - accuracy: 0.8366
Epoch 15/50
700/700 [==============================] - 1s 744us/step - loss:
0.4042 - accuracy: 0.8360
Epoch 16/50
700/700 [==============================] - 1s 773us/step - loss:
0.4038 - accuracy: 0.8357
Epoch 17/50
700/700 [==============================] - 1s 789us/step - loss:
0.4037 - accuracy: 0.8349
Epoch 18/50
700/700 [==============================] - 1s 763us/step - loss:
0.4031 - accuracy: 0.8366
Epoch 19/50
700/700 [==============================] - 1s 757us/step - loss:
0.4030 - accuracy: 0.8363
Epoch 20/50
700/700 [==============================] - 1s 751us/step - loss:
0.4024 - accuracy: 0.8360
Epoch 21/50
700/700 [==============================] - 1s 774us/step - loss:
0.4020 - accuracy: 0.8360
Epoch 22/50
700/700 [==============================] - 1s 771us/step - loss:
0.4019 - accuracy: 0.8331
Epoch 23/50
700/700 [==============================] - 1s 752us/step - loss:
0.4021 - accuracy: 0.8357
Epoch 24/50
700/700 [==============================] - 1s 752us/step - loss:
0.4015 - accuracy: 0.8363
Epoch 25/50
700/700 [==============================] - 1s 771us/step - loss:
0.4013 - accuracy: 0.8339
Epoch 26/50
700/700 [==============================] - 1s 763us/step - loss:
0.4010 - accuracy: 0.8337
Epoch 27/50
700/700 [==============================] - 1s 755us/step - loss:
0.4008 - accuracy: 0.8369
Epoch 28/50
700/700 [==============================] - 1s 758us/step - loss:
0.4003 - accuracy: 0.8364
Epoch 29/50
700/700 [==============================] - 1s 759us/step - loss:
0.4008 - accuracy: 0.8349
Epoch 30/50
700/700 [==============================] - 1s 775us/step - loss:
0.4007 - accuracy: 0.8354
Epoch 31/50
```

```
700/700 [==============================] - 1s 748us/step - loss:
0.3997 - accuracy: 0.8371
Epoch 32/50
700/700 [==============================] - 1s 757us/step - loss:
0.4001 - accuracy: 0.8331
Epoch 33/50
700/700 [==============================] - 1s 770us/step - loss:
0.3995 - accuracy: 0.8351
Epoch 34/50
700/700 [==============================] - 1s 776us/step - loss:
0.3999 - accuracy: 0.8359
Epoch 35/50
700/700 [==============================] - 1s 782us/step - loss:
0.3990 - accuracy: 0.8366
Epoch 36/50
700/700 [==============================] - 1s 769us/step - loss:
0.3997 - accuracy: 0.8359
Epoch 37/50
700/700 [==============================] - 1s 757us/step - loss:
0.3992 - accuracy: 0.8357
Epoch 38/50
700/700 [==============================] - 1s 774us/step - loss:
0.3991 - accuracy: 0.8347
Epoch 39/50
700/700 [==============================] - 1s 763us/step - loss:
0.3983 - accuracy: 0.8347
Epoch 40/50
700/700 [==============================] - 1s 751us/step - loss:
0.3982 - accuracy: 0.8353
Epoch 41/50
700/700 [==============================] - 1s 765us/step - loss:
0.3988 - accuracy: 0.8354
Epoch 42/50
700/700 [==============================] - 1s 744us/step - loss:
0.3982 - accuracy: 0.8339
Epoch 43/50
700/700 [==============================] - 1s 718us/step - loss:
0.3984 - accuracy: 0.8389
Epoch 44/50
700/700 [==============================] - 1s 789us/step - loss:
0.3982 - accuracy: 0.8369
Epoch 45/50
700/700 [==============================] - 1s 749us/step - loss:
0.3976 - accuracy: 0.8336
Epoch 46/50
700/700 [==============================] - 1s 761us/step - loss:
0.3983 - accuracy: 0.8346
Epoch 47/50
700/700 [==============================] - 1s 751us/step - loss:
0.3980 - accuracy: 0.8354
```

```
Epoch 48/50
700/700 [==============================] - 1s 757us/step - loss:
0.3980 - accuracy: 0.8353
Epoch 49/50
700/700 [==============================] - 1s 764us/step - loss:
0.3981 - accuracy: 0.8349
Epoch 50/50
700/700 [==============================] - 1s 746us/step - loss:
0.3979 - accuracy: 0.8357

<keras.callbacks.History at 0x2109341ca00>
```

```
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

```
from sklearn.metrics import
confusion_matrix,accuracy_score,classification_report
```

```
cm = confusion_matrix(y_test,y_pred)
```

```
cm
```

```
array([[2300,   87],
       [ 407,  206]], dtype=int64)
```

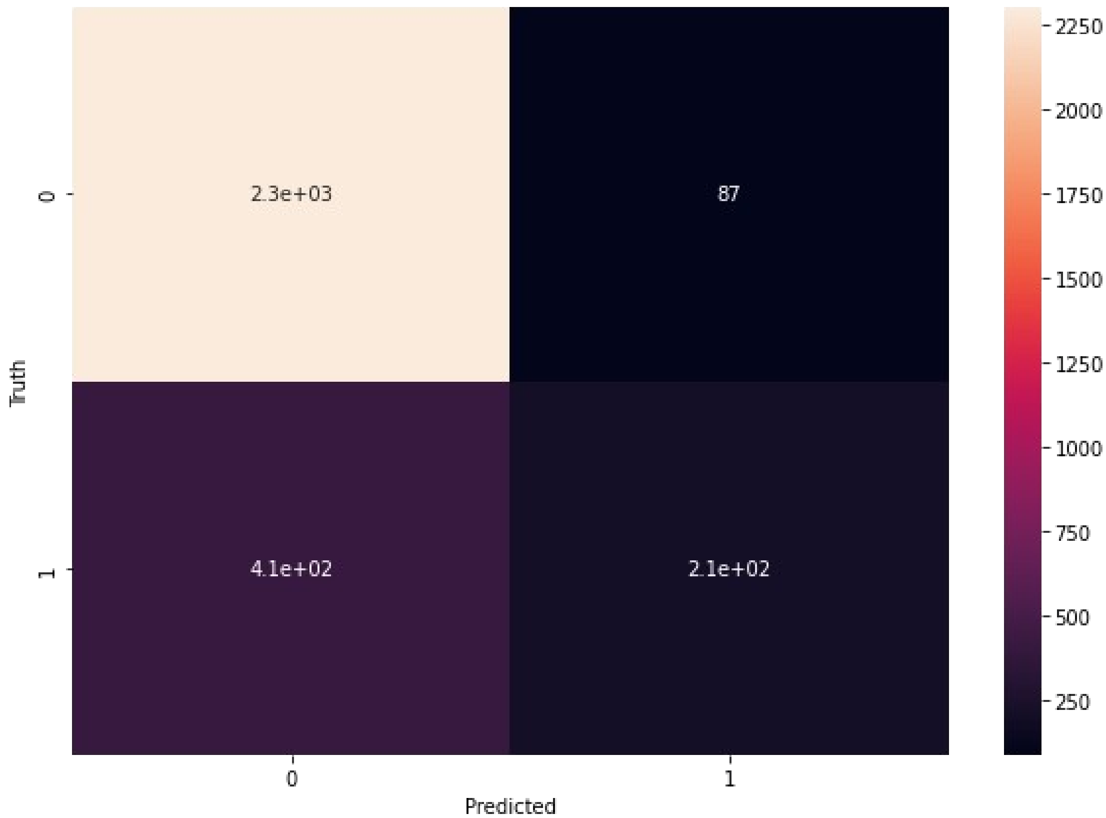
```
accuracy = accuracy_score(y_test,y_pred)
```

```
accuracy
```

```
0.8353333333333334
```

```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```




```
print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.85      0.96      0.90      2387
           1       0.70      0.34      0.45       613

    accuracy                           0.84      3000
   macro avg       0.78      0.65      0.68      3000
weighted avg       0.82      0.84      0.81      3000
```

Assignment 4 :

Code:

jupyter gradient descent assgnt (autosaved)

Logout

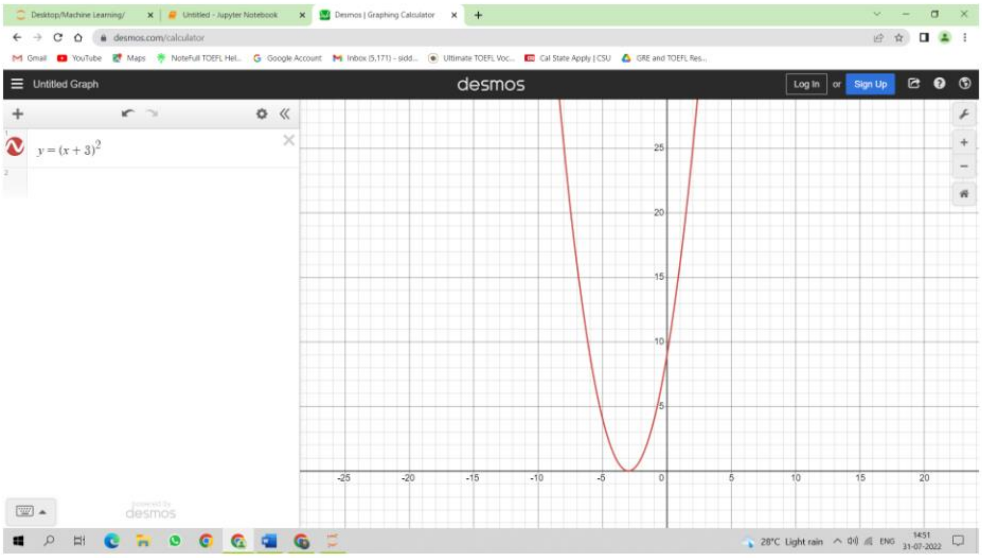
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

Implement Gradient Descent Algorithm to find the local minima of a function.

For example, find the local minima of the function y=(x+3)² starting from the point x=2.



We know the answer just by looking at the graph. y = (x+3)² reaches it's minimum value when x = -3 (i.e when x=-3, y=0). Hence x=-3 is the local and global minima of the function.Below is the implementation in python

In [1]:

```
current_x = 2 # The algorithm starts at x=3
rate = 0.01 # Learning rate
precision = 0.000001 #This tells us when to stop the algorithm
previous_step_size = 1
max_iters = 10000 # maximum number of iterations
iters = 0 #iteration counter
df = lambda x: 2*(x+3) #Gradient of our function
```

In [2]:

```
while previous_step_size > precision and iters < max_iters: #When Previous Step SIze will be less than Precision then we will rec
    previous_x = current_x #Store current x value in prev_x
    current_x = current_x - rate * df(previous_x) #Grad descent
    previous_step_size = abs(current_x - previous_x) #Change in x
    iters = iters+1 #iteration count
    print("Iteration",iters,"\nX value is",current_x) #Print iterations

print("The local minimum occurs at", current_x)
```

```
print("The local minimum occurs at", current_x)
```

```
Iteration 563
X value is -2.999942555213562
Iteration 564
X value is -2.999943704109291
Iteration 565
X value is -2.999944830027105
Iteration 566
X value is -2.999945933426563
Iteration 567
X value is -2.999947014758032
Iteration 568
X value is -2.9999480744628713
Iteration 569
X value is -2.999949112973614
Iteration 570
X value is -2.999950130714142
Iteration 571
X value is -2.999951128099859
The local minimum occurs at -2.999951128099859
```

In [ ]:

In [ ]:

Assignment 5

KNN algorithm on diabetes dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
```

```
df=pd.read_csv('diabetes.csv')
```

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'Pedigree', 'Age', 'Outcome'],
      dtype='object')
```

Check for null values. If present remove null values from the dataset

```
df.isnull().sum()
```

```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

Outcome is the label/target, other columns are features

```
X = df.drop('Outcome',axis = 1)
y = df['Outcome']
```

```
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 42)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("Confusion matrix: ")
cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)
```

```
Confusion matrix:
[[123  28]
 [ 37  43]]
```

```
print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

```
Acccuracy  0.7186147186147186
```

Classification error rate: proportion of instances misclassified over the whole set of instances. Error rate is calculated as the total number of two incorrect predictions (FN + FP) divided by the total number of a dataset (examples in the dataset.

Also error_rate = 1- accuracy

```
total_misclassified = cs[0,1] + cs[1,0]
print(total_misclassified)
total_examples = cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]
print(total_examples)
print("Error rate",total_misclassified/total_examples)
print("Error rate ",1-metrics.accuracy_score(y_test,y_pred))
```

```
65
231
Error rate 0.2813852813852814
Error rate  0.2813852813852814
```

```
print("Precision score",metrics.precision_score(y_test,y_pred))
```

```
Precision score 0.6056338028169014
```

```
print("Recall score ",metrics.recall_score(y_test,y_pred))
```

```
Recall score  0.5375
```

```
print("Classification report
",metrics.classification_report(y_test,y_pred))
```

```
Classification report                precision    recall  f1-score
support
```

0	0.77	0.81	0.79	151
1	0.61	0.54	0.57	80
accuracy			0.72	231
macro avg	0.69	0.68	0.68	231
weighted avg	0.71	0.72	0.71	231

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings("ignore")

titanic_data = pd.read_csv('train.csv')
titanic_test = pd.read_csv('test.csv')
titanic_data.head()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
titanic_data.shape
```

```
(891, 12)
```

```
titanic_data.describe()
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
```

```
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

```
titanic_data.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
```

```
titanic_data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
```

```
Cabin          687
Embarked         2
dtype: int64
```

```
titanic_data = titanic_data.drop(columns='Cabin', axis = 1)
```

```
titanic_data['Age'].fillna(titanic_data['Age'].mean(), inplace= True)
```

```
print(titanic_data['Embarked'].mode()[0])
```

```
S
```

```
titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0],
inplace= True)
```

```
titanic_data.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

```
titanic_data.shape
```

```
(891, 11)
```

```
titanic_data.corr()
```

```
             PassengerId  Survived    Pclass       Age     SibSp
Parch  \
PassengerId     1.000000 -0.005007 -0.035144  0.033207 -0.057527 -
0.001652
Survived       -0.005007  1.000000 -0.338481 -0.069809 -0.035322
0.081629
Pclass         -0.035144 -0.338481  1.000000 -0.331339  0.083081
0.018443
Age             0.033207 -0.069809 -0.331339  1.000000 -0.232625 -
0.179191
SibSp          -0.057527 -0.035322  0.083081 -0.232625  1.000000
0.414838
Parch          -0.001652  0.081629  0.018443 -0.179191  0.414838
1.000000
Fare            0.012658  0.257307 -0.549500  0.091566  0.159651
0.216225
```

```
                 Fare
PassengerId  0.012658
Survived     0.257307
Pclass      -0.549500
Age          0.091566
SibSp        0.159651
Parch        0.216225
Fare         1.000000
```

```
titanic_data['Survived'].value_counts()
```

```
0    549
1    342
Name: Survived, dtype: int64
```

```
titanic_data['Sex'].value_counts()
```

```
male      577
female    314
Name: Sex, dtype: int64
```

```
titanic_data.replace({'Sex':{'male':0,'female':1}}, inplace = True)
```

```
titanic_data['Embarked'].unique()
```

```
array(['S', 'C', 'Q'], dtype=object)
```

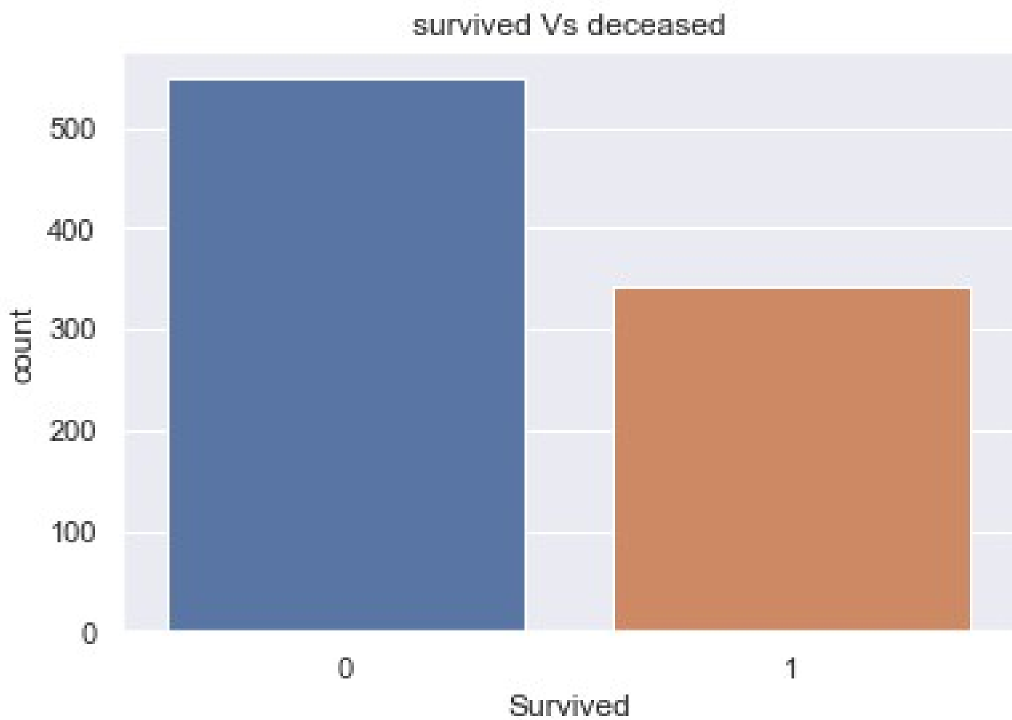
```
titanic_data.replace({'Embarked':{'S':0,'C':1, 'Q':2}}, inplace =
True)
```

```
titanic_data['Parch'].unique()
```

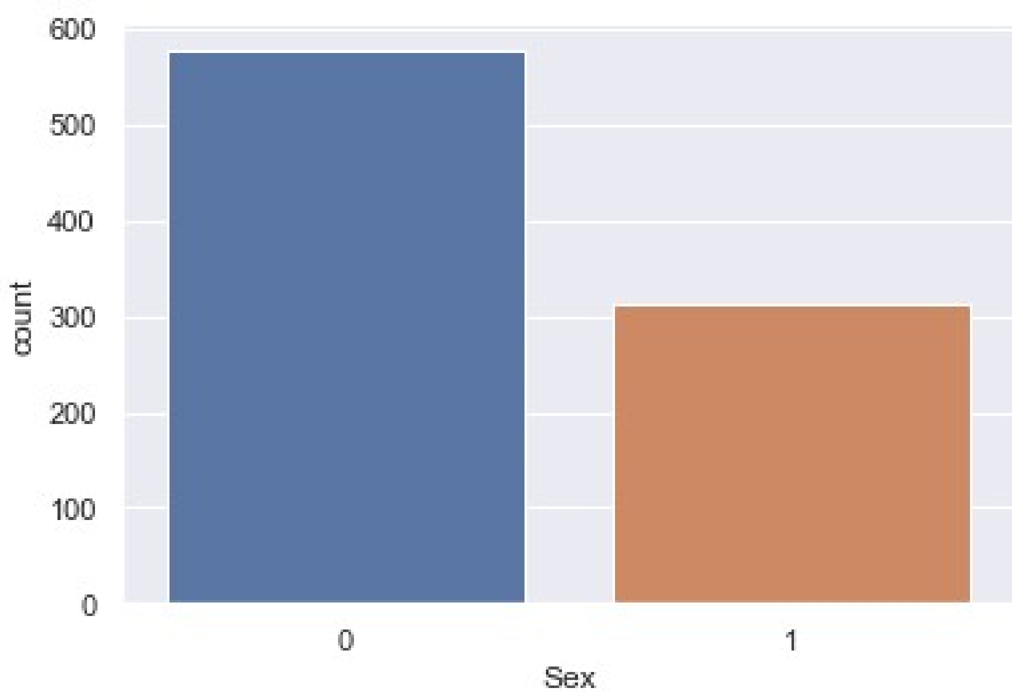
```
array([0, 1, 2, 5, 3, 4, 6], dtype=int64)
```

```
sns.set()
```

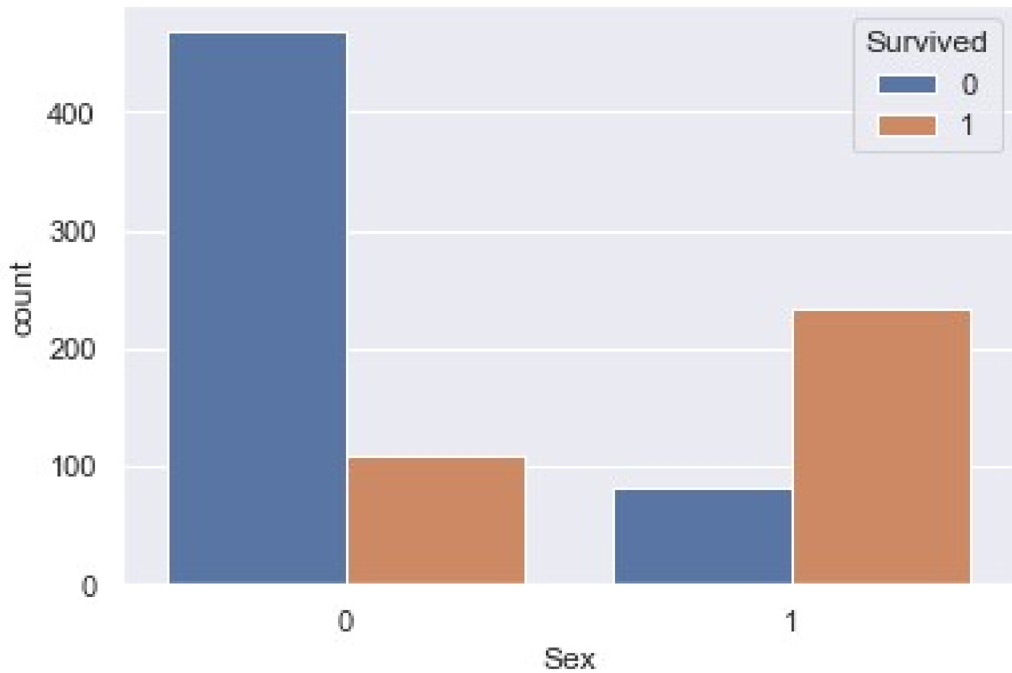
```
sns.countplot(x = titanic_data['Survived']).set_title('survived Vs
deceased');
```




```
sns.countplot(x = titanic_data['Sex']);
```



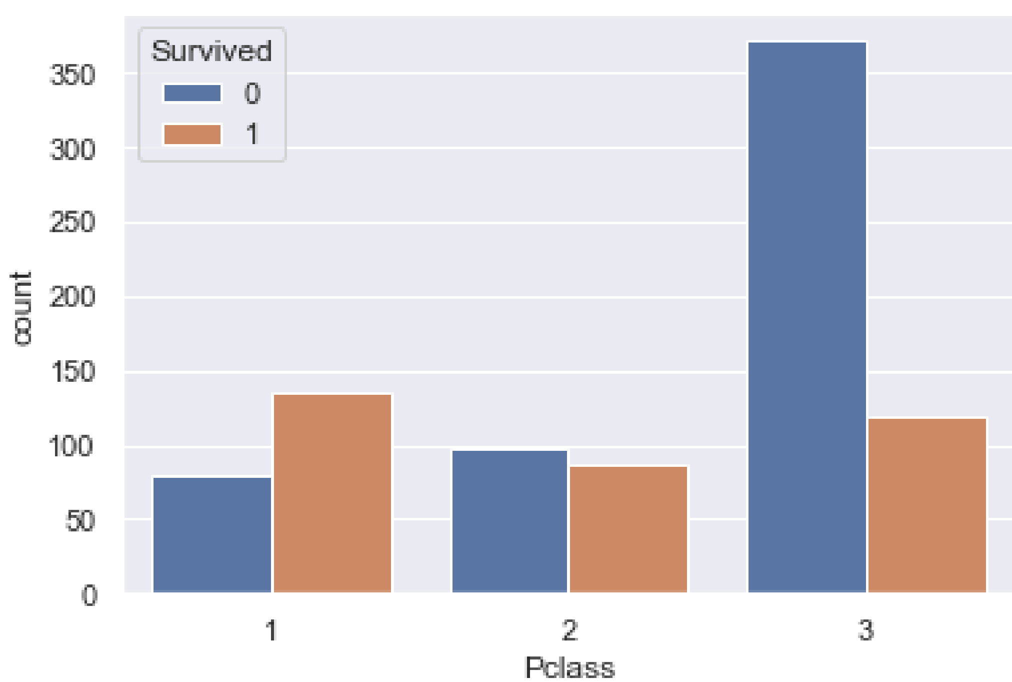

```
sns.countplot('Sex', hue='Survived', data = titanic_data);
```



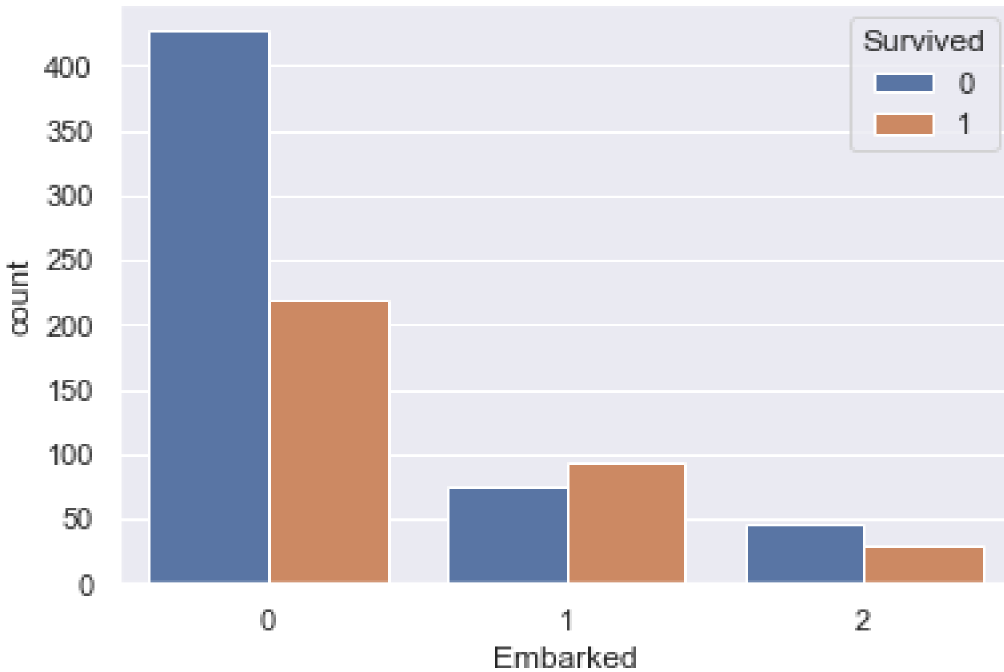

```
titanic_data['Pclass'].value_counts()
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

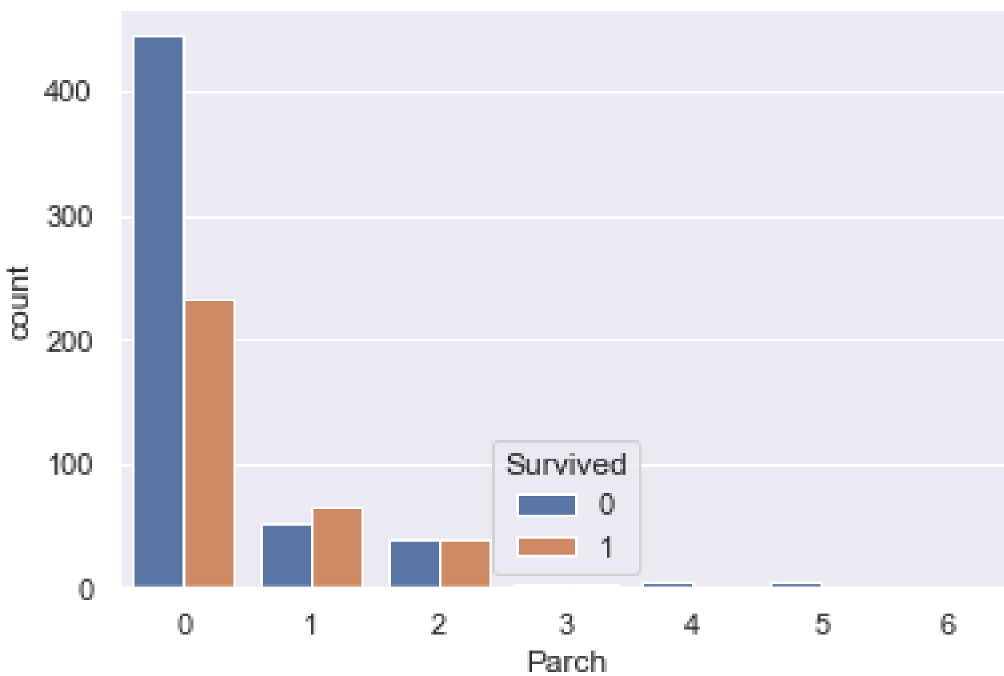
```
sns.countplot('Pclass', hue='Survived', data = titanic_data);
```



```
sns.countplot('Embarked', hue='Survived', data = titanic_data);
```




```
sns.countplot('Parch', hue='Survived', data = titanic_data);
```




```
titanic_data
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
```

```
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name  Sex        Age
SibSp  \
0                              Braund, Mr. Owen Harris    0  22.000000
1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...    1  38.000000
1
2                               Heikkinen, Miss. Laina    1  26.000000
0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)    1  35.000000
1
4                             Allen, Mr. William Henry    0  35.000000
0
..                                                 ...  ...        ...
...
886                              Montvila, Rev. Juozas    0  27.000000
0
887                       Graham, Miss. Margaret Edith    1  19.000000
0
888           Johnston, Miss. Catherine Helen "Carrie"    1  29.699118
1
889                              Behr, Mr. Karl Howell    0  26.000000
0
890                                Dooley, Mr. Patrick    0  32.000000
0

     Parch            Ticket     Fare  Embarked
0        0         A/5 21171   7.2500         0
1        0          PC 17599  71.2833         1
2        0  STON/O2. 3101282   7.9250         0
3        0            113803  53.1000         0
4        0            373450   8.0500         0
..     ...               ...      ...       ...
886      0            211536  13.0000         0
887      0            112053  30.0000         0
888      2        W./C. 6607  23.4500         0
889      0            111369  30.0000         1
890      0            370376   7.7500         2

[891 rows x 11 columns]
```

```
titanic_data.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex              int64
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Embarked         int64
dtype: object
```

```
X = titanic_data.drop(columns=
['PassengerId','Name','Ticket','Survived'],axis=1)
Y = titanic_data['Survived']
```

```
print(X,Y)
```

```
     Pclass  Sex        Age  SibSp  Parch     Fare  Embarked
0         3    0  22.000000      1      0   7.2500         0
1         1    1  38.000000      1      0  71.2833         1
2         3    1  26.000000      0      0   7.9250         0
3         1    1  35.000000      1      0  53.1000         0
4         3    0  35.000000      0      0   8.0500         0
..      ...  ...        ...    ...    ...      ...       ...
886       2    0  27.000000      0      0  13.0000         0
887       1    1  19.000000      0      0  30.0000         0
888       3    1  29.699118      1      2  23.4500         0
889       1    0  26.000000      0      0  30.0000         1
890       3    0  32.000000      0      0   7.7500         2

[891 rows x 7 columns] 0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=
0.2,random_state=2)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

```
(712, 7) (179, 7) (712,) (179,)
```

Model Training:

Logistic Regression

```
logreg = LogisticRegression()
```

```
logreg.fit(X_train,Y_train)
```

```
LogisticRegression()
```

Model Evaluation:

```
X_train_pred = logreg.predict(X_train)
X_train_pred.shape
```

```
(712,)
```

```
ac_training = accuracy_score(Y_train,X_train_pred)
print('Training Accuracy= ', round(ac_training * 100),'%')
```

```
Training Accuracy=  81 %
```

```
X_test_pred = logreg.predict(X_test)
X_test_pred.shape
```

```
(179,)
```

```
ac_testing = accuracy_score(Y_test,X_test_pred)
print('Testing Accuracy= ', round(ac_testing * 100),'%')
```

```
Testing Accuracy=  78 %
```

```
from sklearn.metrics import confusion_matrix
cf=confusion_matrix(Y_test,X_test_pred)
cf
```

```
array([[91,  9],
       [30, 49]], dtype=int64)
```