#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

In [1]:
```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

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

Out[3]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

In [4]:
```
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):
Unnamed: 0           200000 non-null int64
key                  200000 non-null object
fare_amount          200000 non-null float64
pickup_datetime      200000 non-null object
pickup_longitude     200000 non-null float64
pickup_latitude      200000 non-null float64
dropoff_longitude    199999 non-null float64
dropoff_latitude     199999 non-null float64
passenger_count      200000 non-null int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [5]:
```
df.columns #TO get number of columns in the dataset
```

Out[5]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [6]:
```
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as it isn't required
```

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

Out[7]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

In [8]:
```
df.shape #To get the total (Rows,Columns)
```

Out[8]:
```
(200000, 7)
```

In [9]:
```
df.dtypes #To get the type of each column
```

Out[9]:
```
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

In [10]:
```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
```

In [11]:
```
df.dtypes
```

Out[11]:
```
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
```

Filling Missing values

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [13]:
```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

To segregate each time of date and time

In [15]:
```
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)
```

In [16]:
```
df.head()
```

Out[16]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	1	19	7	5	2015	3
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	1	20	17	7	2009	4
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	1	21	24	8	2009	0
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	3	8	26	6	2009	4
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	5	17	28	8	2014	3

Here we are going to use Heversine formula to calculate the distance between two points and journey, using the longitude and latitude values.

Heversine formula hav(θ) = sin**2(θ/2).

In [19]:
```
from math import *
# function to calculate the travel distance from the longitudes and latitudes
def distance_transform(longitude1, latitude1, longitude2, latitude2):
    travel_dist = []

    for pos in range(len(longitude1)):
        long1,lati1,long2,lati2 = map(radians,[longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]])
        dist_long = long2 - long1
        dist_lati = lati2 - lati1
        a = sin(dist_lati/2)**2 + cos(lati1) * cos(lati2) * sin(dist_long/2)**2
        c = 2 * asin(sqrt(a))*6371
        travel_dist.append(c)

    return travel_dist
```

In [20]:
```
df['dist_travel_km'] = distance_transform(df['pickup_longitude'].to_numpy(),
                                                df['pickup_latitude'].to_numpy(),
                                                df['dropoff_longitude'].to_numpy(),
                                                df['dropoff_latitude'].to_numpy()
                                              )
```

In [21]:
```
df.head()
```

Out[21]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	1	19	7	5	2015	3	1.683323
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	1	20	17	7	2009	4	2.457590
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	1	21	24	8	2009	0	5.036377
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	3	8	26	6	2009	4	1.661683
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	5	17	28	8	2014	3	4.475450

In [22]:
```
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

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

In [23]:
```
df.head()
```

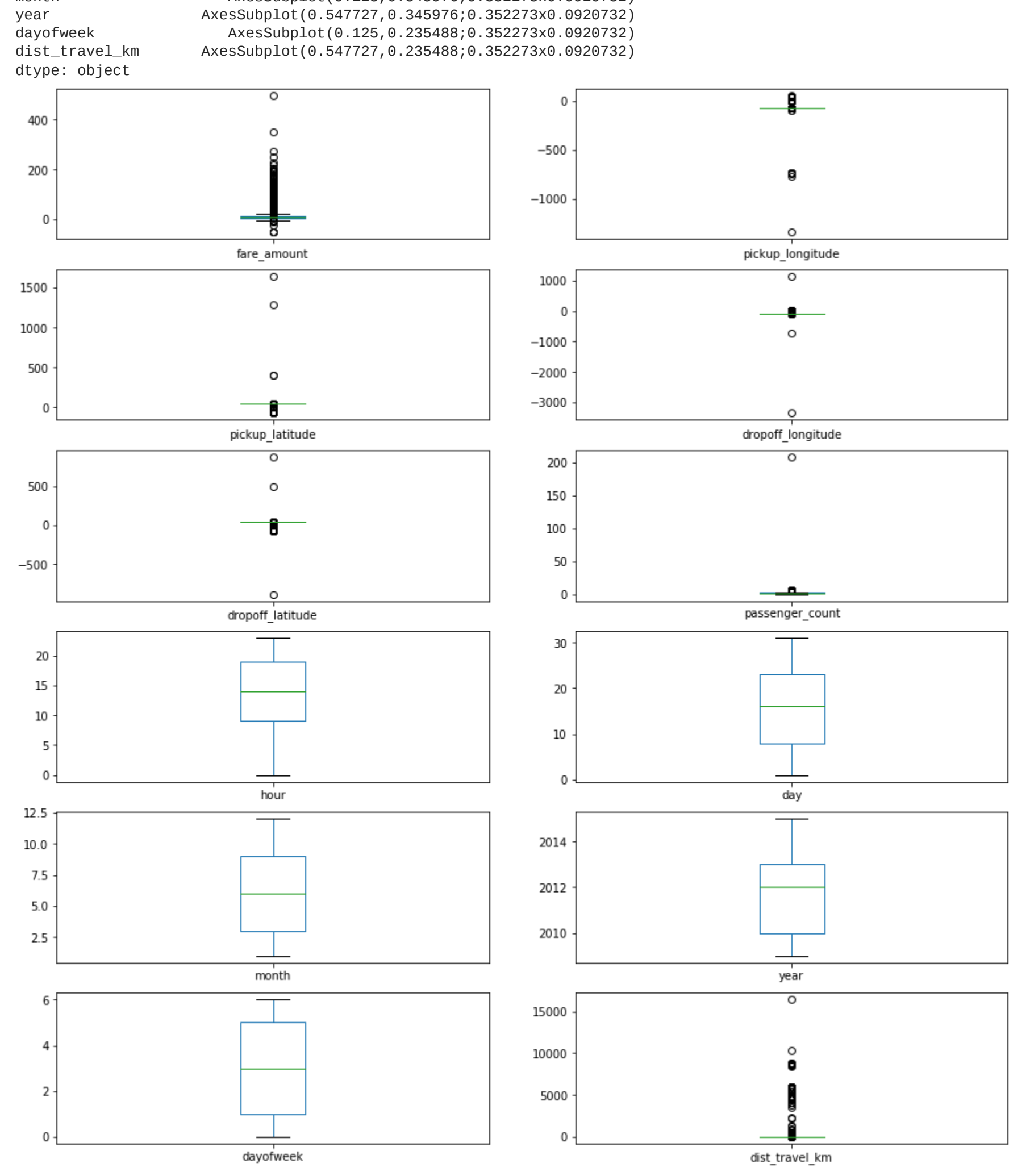
Out[23]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1	19	7	5	2015	3	1.683323
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1	20	17	7	2009	4	2.457590
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1	21	24	8	2009	0	5.036377
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3	8	26	6	2009	4	1.661683
4	16.0	-73.925023	40.744085	-73.973082	40.761247	5	17	28	8	2014	3	4.475450

Checking outliers and filling them

In [24]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to check the outliers
```

Out[24]:
```
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)
dist_travel_km       AxesSubplot(0.547727,0.235488;0.352273x0.0920732)
dtype: object
```

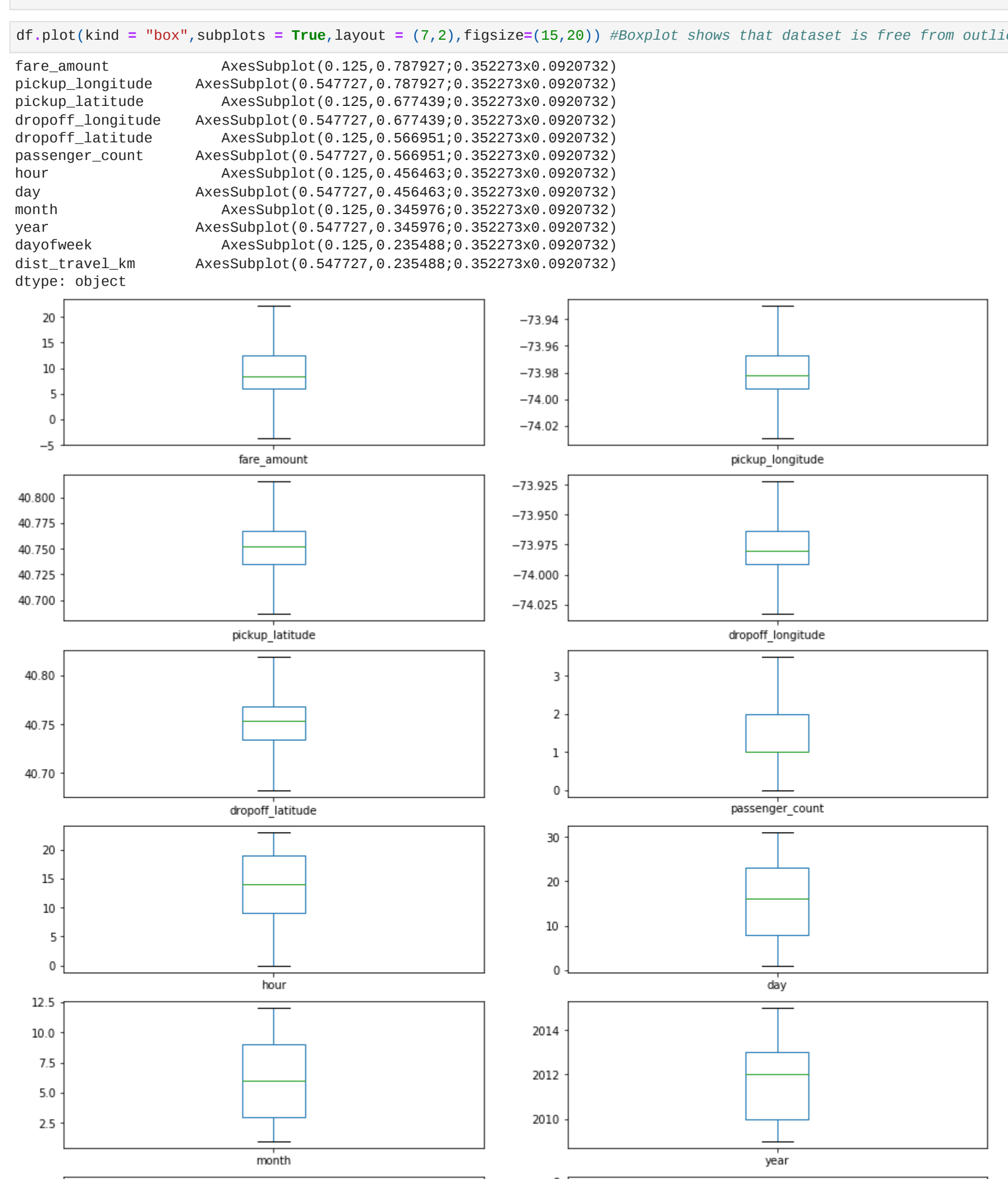


In [25]:
```
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

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

In [26]:
```
df = treat_outliers_all(df , df.iloc[: , 0::])
```

In [27]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot shows that dataset is free from outliers
```

Out[27]:
```
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)
dist_travel_km       AxesSubplot(0.547727,0.235488;0.352273x0.0920732)
dtype: object
```



In [28]:
```
#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)
```

In [29]:
```
#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)
                                    ]
```

In [30]:
```
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

In [31]:
```
df.head()
```

Out[31]:

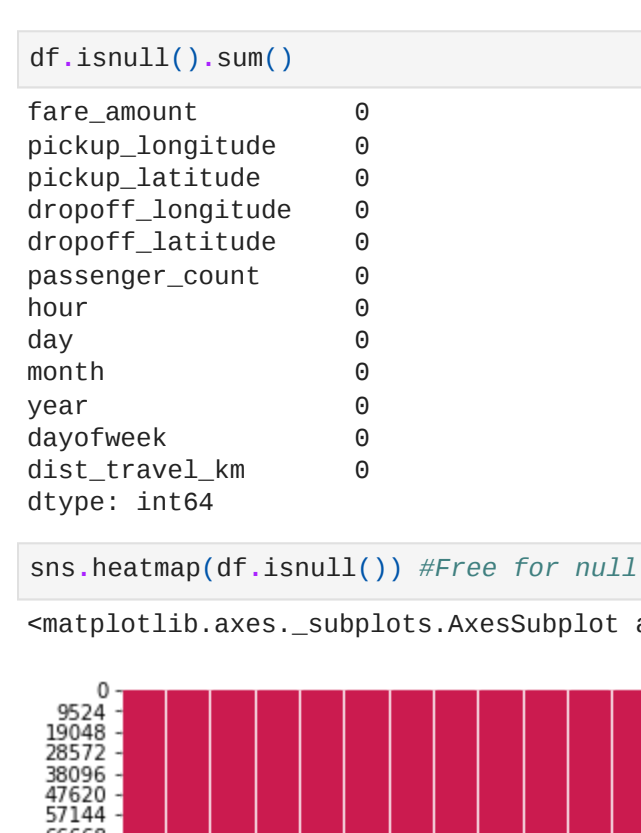
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0	19	7	5	2015	3	1.683323
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0	20	17	7	2009	4	2.457590
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0	21	24	8	2009	0	5.036377
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3.0	8	26	6	2009	4	1.661683
4	16.0	-73.929786	40.744085	-73.973082	40.761247	3.5	17	28	8	2014	3	4.475450

In [32]:
```
df.isnull().sum()
```

Out[32]:
```
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
```

In [33]:
```
sns.heatmap(df.isnull()) #Free for null values
```

Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x8d8af2a080>
```



In [34]:
```
corr = df.corr() #Function to find the correlation
```

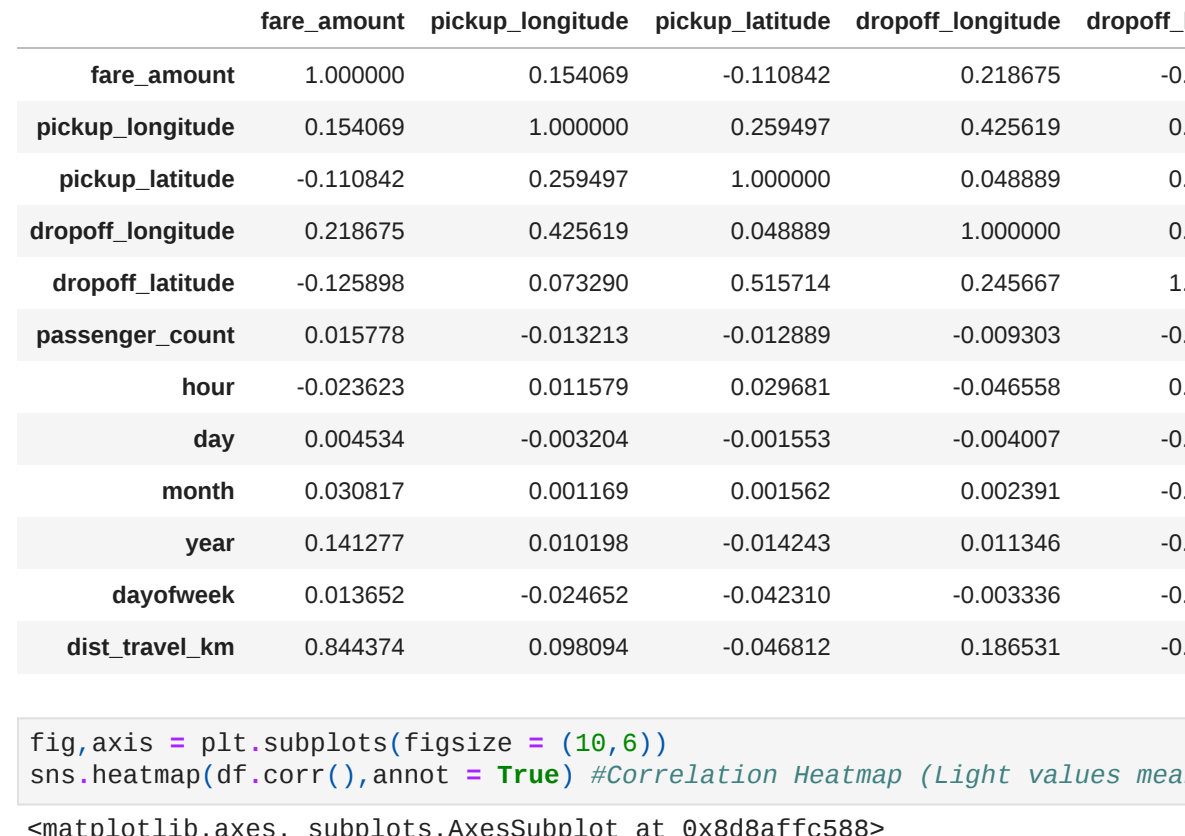
In [35]:
```
corr
```

Out[35]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
fare_amount	1.000000	0.154069	-0.110842	0.218675	-0.125898	0.015778	-0.023623	0.004534	0.030817	0.141277	0.013652	0.844374
pickup_longitude	0.154069	1.000000	0.259497	0.425619	0.073290	-0.013213	0.011579	-0.003204	0.001169	0.010198	-0.024652	0.098094
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	0.515714	-0.012889	0.029681	-0.001553	0.001562	-0.014243	-0.042310	-0.046812
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	0.245667	-0.009303	-0.046558	-0.004007	0.002391	0.011346	-0.003336	0.186531
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	1.000000	-0.006308	0.019783	-0.003479	-0.001193	-0.009603	-0.031919	-0.038900
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	-0.006308	1.000000	0.020274	0.002712	0.010351	-0.009749	0.048550	0.009709
hour	-0.023623	0.011579	0.029681	-0.046558	0.019783	0.020274	1.000000	0.004677	-0.003926	0.002156	-0.086947	-0.038366
day	0.004534	-0.003204	-0.001553	-0.004007	-0.003479	0.002712	0.004677	1.000000	-0.017360	-0.012170	0.005617	0.003062
month	0.030817	0.001169	0.001562	0.002391	-0.001193	0.010351	-0.003926	-0.017360	1.000000	-0.115859	-0.008786	0.011628
year	0.141277	0.010198	-0.014243	0.011346	-0.009603	-0.009749	0.002156	-0.012170	-0.115859	1.000000	0.006113	0.024278
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	-0.031919	0.048550	-0.086947	0.005617	-0.008786	0.006113	1.000000	0.027053
dist_travel_km	0.844374	0.098094	-0.046812	0.186531	-0.038900	0.009709	-0.038366	0.003062	0.011628	0.024278	0.027053	1.000000

In [36]:
```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means highly correlated)
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x8d8affc588>
```



Dividing the dataset into feature and target values

In [37]:
```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','passenger_count','hour','day','month','year','dayofweek','dist_travel_km']]
```

In [38]:
```
y = df['fare_amount']
```

Dividing the dataset into training and testing dataset

In [39]:
```
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

In [40]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [41]:
```
regression.fit(X_train,y_train)
```

Out[41]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [42]:
```
regression.intercept_ #To find the linear intercept
```

Out[42]:
```
2809.192377415925
```

In [43]:
```
regression.coef_ #To find the linear coeeficient
```

Out[43]:
```
array([ 1.75328304e+01, -9.83172673e+00,  1.54611809e+01, -1.69707270e+01,
        5.40456388e-02,  9.46950748e-03,  1.66720620e-03,  5.40917698e-02,
        3.61743634e-01, -3.69474342e-02,  2.00077959e+00])
```

In [44]:
```
prediction = regression.predict(X_test) #To predict the target values
```

In [45]:
```
print(prediction)
```

```
[10.80422002  4.74707896  9.95283165 ...  5.89597937 17.00144322
  5.38487972]
```

In [46]:
```
y_test
```

Out[46]:
```
16850      8.50
181076     4.10
70798      9.30
87421     12.90
169443    22.25
18976     11.00
50921     13.70
199564    14.50
125215     5.30
67510      8.50
85217     22.25
156903    21.50
116795     4.10
112179    16.90
124459     3.70
173299    22.25
51448     19.70
99502     22.25
174467    10.90
78880     20.50
26798     22.25
38501      4.50
63091     12.90
171207    22.25
142238     8.50
101106     7.30
120177     4.50
154585    14.50
75840      5.50
85918     14.00
          ...
104227    10.10
14172     19.70
49985      3.70
183045     6.50
11927     12.90
93684      4.50
101795    13.70
21444      6.10
85147      8.50
81311      8.00
157686    11.70
194074     6.50
132558    10.50
132616    11.70
188536     5.70
179629     8.90
11277      3.70
147880     7.30
116553     5.70
157394     6.50
103519    13.30
41348     12.90
12608      4.50
6820       5.50
84612      5.00
168836     3.70
39719     21.00
124536     4.90
90432     22.10
12543      4.90
Name: fare_amount, Length: 66000, dtype: float64
```

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

In [47]:
```
from sklearn.metrics import r2_score
```

In [48]:
```
r2_score(y_test,prediction)
```

Out[48]:
```
0.7471032194200018
```

In [49]:
```
from sklearn.metrics import mean_squared_error
```

In [50]:
```
MSE = mean_squared_error(y_test,prediction)
```

In [51]:
```
MSE
```

Out[51]:
```
7.464818887848474
```

In [52]:
```
RMSE = np.sqrt(MSE)
```

In [53]:
```
RMSE
```

Out[53]:
```
2.7321820744321696
```

Random Forest Regression

In [54]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [55]:
```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of trees you want to build before making the prediction
```

In [56]:
```
rf.fit(X_train,y_train)
```

Out[56]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [57]:
```
y_pred = rf.predict(X_test)
```

In [58]:
```
y_pred
```

Out[58]:
```
array([ 9.7025,  4.744 ,  9.202 , ...,  6.468 , 16.2802,  4.47  ])
```

Metrics evaluatin for Random Forest

In [59]:
```
R2_Random = r2_score(y_test,y_pred)
```

In [60]:
```
R2_Random
```

Out[60]:
```
0.8024361566950065
```

In [64]:
```
MSE_Random = mean_squared_error(y_test,y_pred)
MSE_Random
```

Out[64]:
```
5.831542440662031
```

In [65]:
```
RMSE_Random = np.sqrt(MSE_Random)
RMSE_Random
```

Out[65]:
```
2.4148586792319815
```

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

In [19]:
```
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
```

In [20]:
```
df=pd.read_csv('emails.csv')
```

In [21]:
```
df.head()
```

Out[21]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Prediction
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	0
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	0

5 rows × 3002 columns

In [22]:
```
df.columns
```

Out[22]:
```
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)
```

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
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
```

In [24]:
```
df.dropna(inplace = True)
```

In [25]:
```
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
```

In [26]:
```
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

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

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

In [36]:
```
print("Prediction",y_pred)
```

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

In [37]:
```
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
KNN accuracy =  0.8009020618556701
```

In [39]:
```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

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

SVM classifier

In [27]:
```
# cost C = 1
model = SVC(C = 1)

# fit
model.fit(X_train, y_train)

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

In [28]:
```
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
```

Out[28]:
```
array([[1091,    6],
       [  90,  365]])
```

In [29]:
```
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.

In [46]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

In [47]:
```
df = pd.read_csv("Churn_Modelling.csv")
```

Preprocessing.

In [48]:
```
df.head()
```

Out[48]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalar
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.8
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.5
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.5
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.6
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1

In [49]:
```
df.shape
```

Out[49]:
```
(10000, 14)
```

In [50]:
```
df.describe()
```

Out[50]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000

In [51]:
```
df.isnull()
```

Out[51]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedS
0	False	False	False	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	False	False	False	False	False	False	False	False	False	False	False	False	F
9996	False	False	False	False	False	False	False	False	False	False	False	False	F
9997	False	False	False	False	False	False	False	False	False	False	False	False	F
9998	False	False	False	False	False	False	False	False	False	False	False	False	F
9999	False	False	False	False	False	False	False	False	False	False	False	False	F

10000 rows × 14 columns

In [52]:
```
df.isnull().sum()
```

Out[52]:
```
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
```

In [53]:
```
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
```

In [54]:
```
df.dtypes
```

Out[54]:
```
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
```

In [55]:
```
df.columns
```

Out[55]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [56]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary columns
```

In [57]:
```
df.head()
```

Out[57]:

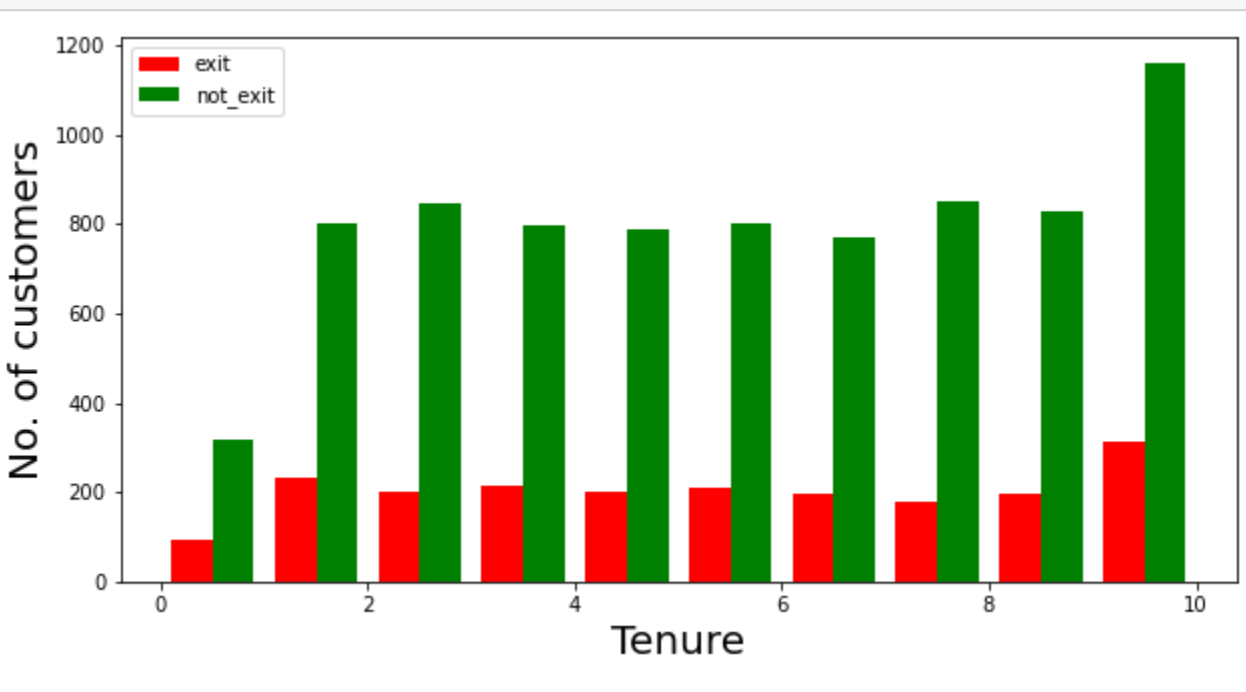
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Visualization

In [101]:
```
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()
```

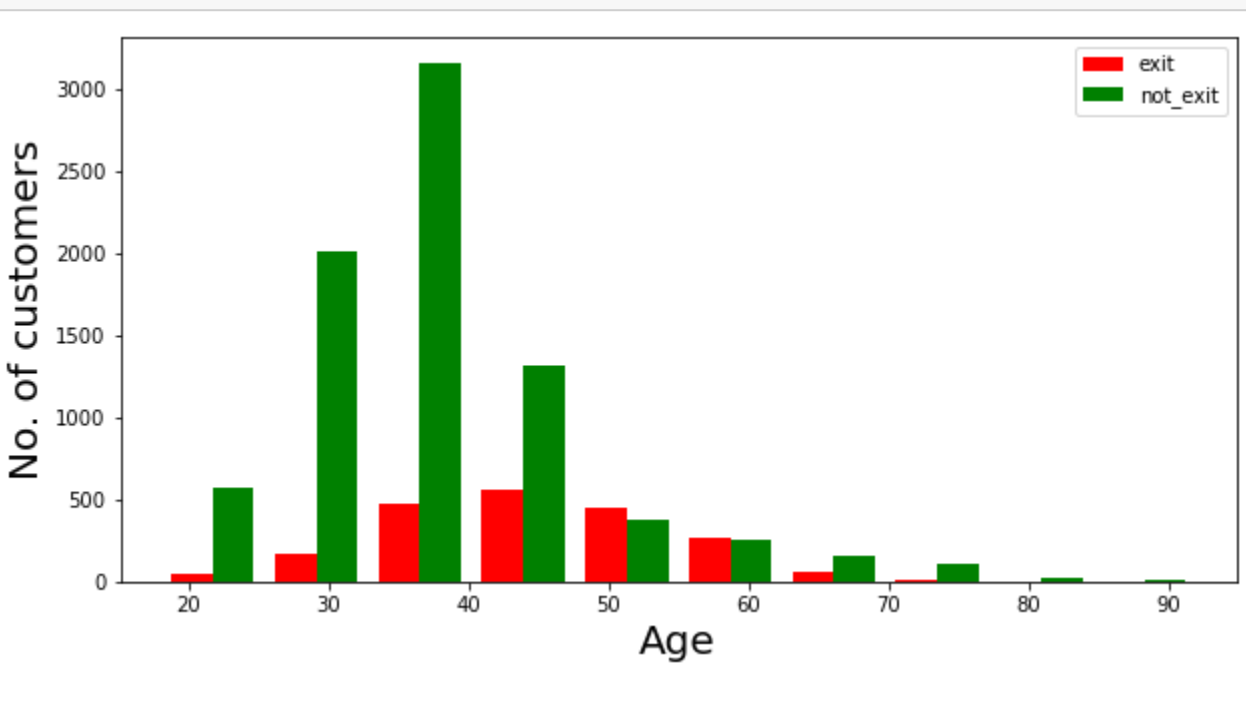
In [102]:
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

In [103]:
```
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```



In [105]:
```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

In [106]:
```
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



Converting the Categorical Variables

In [59]:
```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalar
y']]
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
```

In [61]:
```
df = pd.concat([df,gender,states], axis = 1)
```

Splitting the training and testing Dataset

In [62]:
```
df.head()
```

Out[62]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Male	Germany	Spain
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1	0	0	0
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0	0	0	1
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1	0	0	0
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0	0	0	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0	0	0	1

In [63]:
```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male'
,'Germany','Spain']]
```

In [64]:
```
y = df['Exited']
```

In [65]:
```
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

In [66]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [67]:
```
X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [68]:
```
X_train
```

Out[68]:
```
array([[ 4.56838557e-01, -9.45594735e-01,  1.58341939e-03, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [-2.07591864e-02, -2.77416637e-01,  3.47956411e-01, ...,
        -1.09507222e+00, -5.81969145e-01,  1.74334114e+00],
       [-1.66115021e-01,  1.82257167e+00, -1.38390855e+00, ...,
        -1.09507222e+00, -5.81969145e-01, -5.73611200e-01],
       ...,
       [-3.63383654e-01, -4.68324665e-01,  1.73344838e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [ 4.67221117e-01, -1.42286480e+00,  1.38707539e+00, ...,
         9.13181783e-01, -5.81969145e-01,  1.74334114e+00],
       [-8.82511636e-01,  2.95307447e-01, -6.91162564e-01, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01]])
```

In [69]:
```
X_test
```

Out[69]:
```
array([[ 3.63395520e-01,  1.99853433e-01,  1.58341939e-03, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [-4.15243057e-02,  4.86215475e-01,  1.58341939e-03, ...,
        -1.09507222e+00, -5.81969145e-01,  1.74334114e+00],
       [-1.87923736e+00, -3.72870651e-01, -1.38390855e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       ...,
       [-6.02182526e-01, -5.63778679e-01, -1.73028154e+00, ...,
        -1.09507222e+00, -5.81969145e-01, -5.73611200e-01],
       [ 1.51585964e+00, -6.59232693e-01,  1.73344838e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01],
       [-5.19122049e-01,  1.04399419e-01,  1.73344838e+00, ...,
         9.13181783e-01, -5.81969145e-01, -5.73611200e-01]])
```

Building the Classifier Model using Keras

In [70]:
```
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.
```

In [71]:
```
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

In [72]:
```
classifier = Sequential()
```

In [74]:
```
#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"))
```

In [75]:
```
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform"))   #Adding second hidden layers
```

In [76]:
```
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform")) #Final neuron will be having
 siigmoid function
```

In [77]:
```
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
```

In [79]:
```
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 6)                 72        
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 42        
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 7         
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

In [89]:
```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training dataset
```

```
Epoch 1/50
700/700 [==============================] - 0s 674us/step - loss: 0.4293 - accuracy: 0.7947
Epoch 2/50
700/700 [==============================] - 0s 647us/step - loss: 0.4239 - accuracy: 0.7947
Epoch 3/50
700/700 [==============================] - 0s 657us/step - loss: 0.4203 - accuracy: 0.8067
Epoch 4/50
700/700 [==============================] - 0s 664us/step - loss: 0.4167 - accuracy: 0.8260
Epoch 5/50
700/700 [==============================] - 0s 674us/step - loss: 0.4153 - accuracy: 0.8287
Epoch 6/50
700/700 [==============================] - 0s 653us/step - loss: 0.4137 - accuracy: 0.8310
Epoch 7/50
700/700 [==============================] - 0s 658us/step - loss: 0.4125 - accuracy: 0.8317
Epoch 8/50
700/700 [==============================] - 1s 842us/step - loss: 0.4116 - accuracy: 0.8306
Epoch 9/50
700/700 [==============================] - 0s 671us/step - loss: 0.4103 - accuracy: 0.8331
Epoch 10/50
700/700 [==============================] - 0s 682us/step - loss: 0.4100 - accuracy: 0.8326
Epoch 11/50
700/700 [==============================] - 0s 690us/step - loss: 0.4093 - accuracy: 0.8337
Epoch 12/50
700/700 [==============================] - 0s 688us/step - loss: 0.4087 - accuracy: 0.8339
Epoch 13/50
700/700 [==============================] - 0s 675us/step - loss: 0.4081 - accuracy: 0.8341
Epoch 14/50
700/700 [==============================] - 1s 722us/step - loss: 0.4071 - accuracy: 0.8331
Epoch 15/50
700/700 [==============================] - 1s 811us/step - loss: 0.4065 - accuracy: 0.8341
Epoch 16/50
700/700 [==============================] - 0s 711us/step - loss: 0.4056 - accuracy: 0.8356
Epoch 17/50
700/700 [==============================] - 0s 702us/step - loss: 0.4046 - accuracy: 0.8366
Epoch 18/50
700/700 [==============================] - 0s 688us/step - loss: 0.4035 - accuracy: 0.8343
Epoch 19/50
700/700 [==============================] - 1s 715us/step - loss: 0.4024 - accuracy: 0.8363
Epoch 20/50
700/700 [==============================] - 0s 714us/step - loss: 0.4020 - accuracy: 0.8337
Epoch 21/50
700/700 [==============================] - 0s 705us/step - loss: 0.4010 - accuracy: 0.8374
Epoch 22/50
700/700 [==============================] - 1s 720us/step - loss: 0.4003 - accuracy: 0.8370
Epoch 23/50
700/700 [==============================] - 0s 692us/step - loss: 0.3993 - accuracy: 0.8374
Epoch 24/50
700/700 [==============================] - 0s 709us/step - loss: 0.3990 - accuracy: 0.8356
Epoch 25/50
700/700 [==============================] - 1s 871us/step - loss: 0.3984 - accuracy: 0.8366
Epoch 26/50
700/700 [==============================] - 1s 719us/step - loss: 0.3984 - accuracy: 0.8367
Epoch 27/50
700/700 [==============================] - 1s 719us/step - loss: 0.3980 - accuracy: 0.8366
Epoch 28/50
700/700 [==============================] - 0s 695us/step - loss: 0.3981 - accuracy: 0.8366
Epoch 29/50
700/700 [==============================] - 0s 667us/step - loss: 0.3976 - accuracy: 0.8374
Epoch 30/50
700/700 [==============================] - 0s 669us/step - loss: 0.3972 - accuracy: 0.8373
Epoch 31/50
700/700 [==============================] - 0s 670us/step - loss: 0.3970 - accuracy: 0.8370
Epoch 32/50
700/700 [==============================] - 1s 720us/step - loss: 0.3972 - accuracy: 0.8376
Epoch 33/50
700/700 [==============================] - 0s 675us/step - loss: 0.3965 - accuracy: 0.8367
Epoch 34/50
700/700 [==============================] - 0s 680us/step - loss: 0.3961 - accuracy: 0.8364
Epoch 35/50
700/700 [==============================] - 0s 685us/step - loss: 0.3962 - accuracy: 0.8379
Epoch 36/50
700/700 [==============================] - 1s 771us/step - loss: 0.3960 - accuracy: 0.8370
Epoch 37/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3963 - accuracy: 0.8366
Epoch 38/50
700/700 [==============================] - 1s 764us/step - loss: 0.3962 - accuracy: 0.8373
Epoch 39/50
700/700 [==============================] - 1s 823us/step - loss: 0.3950 - accuracy: 0.8384
Epoch 40/50
700/700 [==============================] - 1s 759us/step - loss: 0.3956 - accuracy: 0.8361
Epoch 41/50
700/700 [==============================] - 1s 773us/step - loss: 0.3949 - accuracy: 0.8366
Epoch 42/50
700/700 [==============================] - 0s 695us/step - loss: 0.3953 - accuracy: 0.8369
Epoch 43/50
700/700 [==============================] - 0s 701us/step - loss: 0.3952 - accuracy: 0.8369
Epoch 44/50
700/700 [==============================] - 0s 707us/step - loss: 0.3952 - accuracy: 0.8366
Epoch 45/50
700/700 [==============================] - 0s 680us/step - loss: 0.3955 - accuracy: 0.8376
Epoch 46/50
700/700 [==============================] - 0s 665us/step - loss: 0.3947 - accuracy: 0.8373
Epoch 47/50
700/700 [==============================] - 0s 708us/step - loss: 0.3947 - accuracy: 0.8371
Epoch 48/50
700/700 [==============================] - 0s 681us/step - loss: 0.3944 - accuracy: 0.8371
Epoch 49/50
700/700 [==============================] - 0s 678us/step - loss: 0.3947 - accuracy: 0.8383
Epoch 50/50
700/700 [==============================] - 1s 869us/step - loss: 0.3944 - accuracy: 0.8370
```

Out[89]:
```
<tensorflow.python.keras.callbacks.History at 0x1fb1eb93df0>
```

In [90]:
```
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

In [97]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
```

In [92]:
```
cm = confusion_matrix(y_test,y_pred)
```

In [93]:
```
cm
```

Out[93]:
```
array([[2328,   72],
       [ 425,  175]], dtype=int64)
```

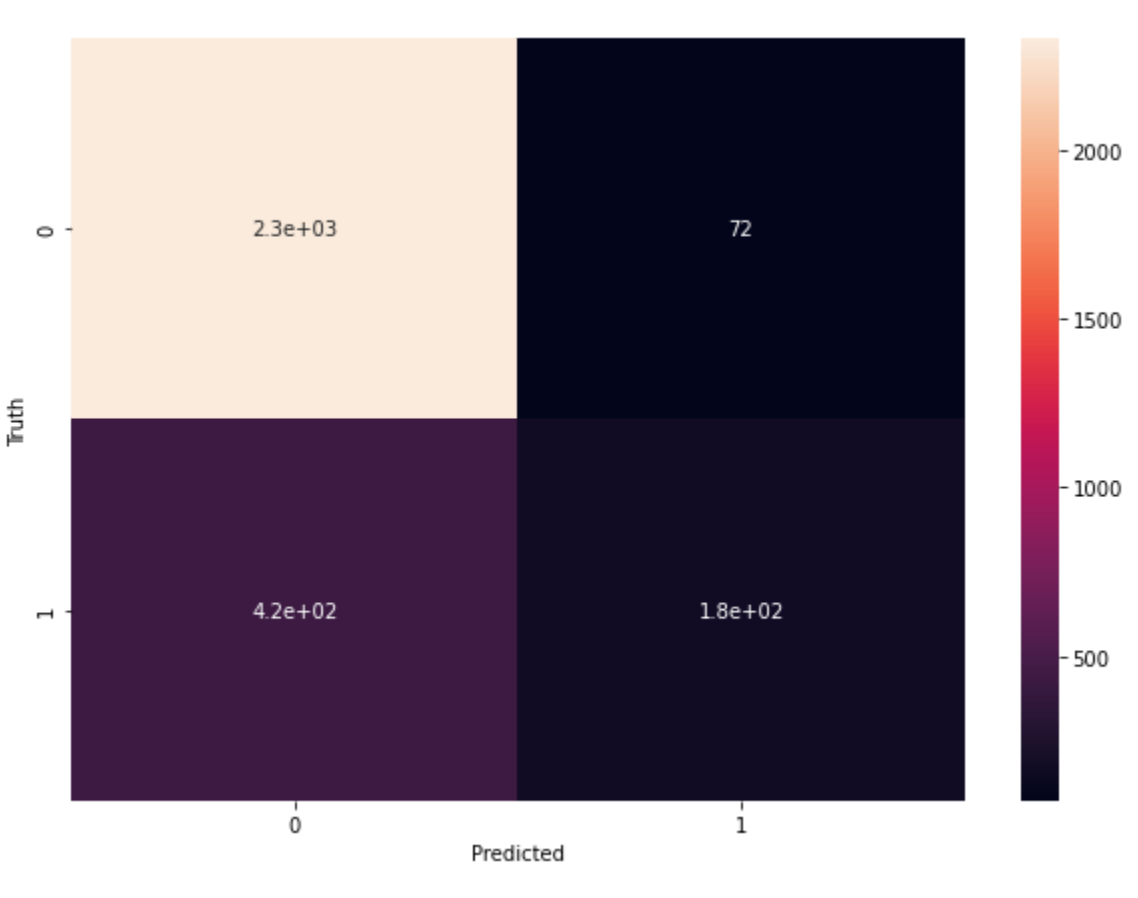
In [94]:
```
accuracy = accuracy_score(y_test,y_pred)
```

In [95]:
```
accuracy
```

Out[95]:
```
0.8343333333333334
```

In [98]:
```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[98]:
```
Text(69.0, 0.5, 'Truth')
```



In [100]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.97      0.90      2400
           1       0.71      0.29      0.41       600

    accuracy                           0.83      3000
   macro avg       0.78      0.63      0.66      3000
weighted avg       0.82      0.83      0.81      3000
```

Prcatical No. 4: Manual Content

Guru Gobind Singh Foundation's

Guru Gobind Singh College of Engineering and Research Center, Nashik

Experiment No: 04

Title of Experiment: 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 $x=2$.

Student Name:			
Class:	**BE (Computer)**		
Div:	-	**Batch:**	**CO**
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			

Marks (Grade) Attainment of CO Marks out of 10

A	P	W	T	Total

CO Mapped	CO3: Select and apply appropriately supervised machine learning algorithms for real time applications.
Signature of Subject Teacher	

TITLE: 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.

AIM: The aim of this practical is to illustrate the implementation of the Gradient Descent algorithm for locating local minima of a given function. This involves understanding the gradient descent method, performing iterations to converge towards a local minimum, and evaluating the effectiveness of the algorithm in the context of the function $y = (x + 3)^2$ starting from the initial point (x = 2).

OBJECTIVES: Based on above main aim following are the objectives

1. **Understanding Optimization Techniques**: Develop a solid understanding of optimization methods, with a focus on the Gradient Descent algorithm. Grasp the concept of finding local minima in mathematical functions through iterative updates.

2. **Algorithm Implementation**: Implement the Gradient Descent algorithm practically. Apply it to the function $y = (x + 3)^2$, beginning with an initial value of (x = 2), and observe how the algorithm converges towards a local minimum.

3. **Analyzing Convergence**: Examine the iterative behavior of the Gradient Descent algorithm. Gain insights into how the algorithm iteratively refines the value of (x) by utilizing the gradient of the function.

4. **Algorithm Evaluation**: Evaluate the effectiveness of the Gradient Descent algorithm in successfully identifying the local minimum of the given function. Compare the obtained result with the anticipated theoretical solution.

Prerequisite:

Basic of Python, Concept of Gradient Descent Algorithm

Software Requirements:

Anaconda with Python 3.7

Hardware Requirement:

PIV, 2GB RAM, 500 GB HDD

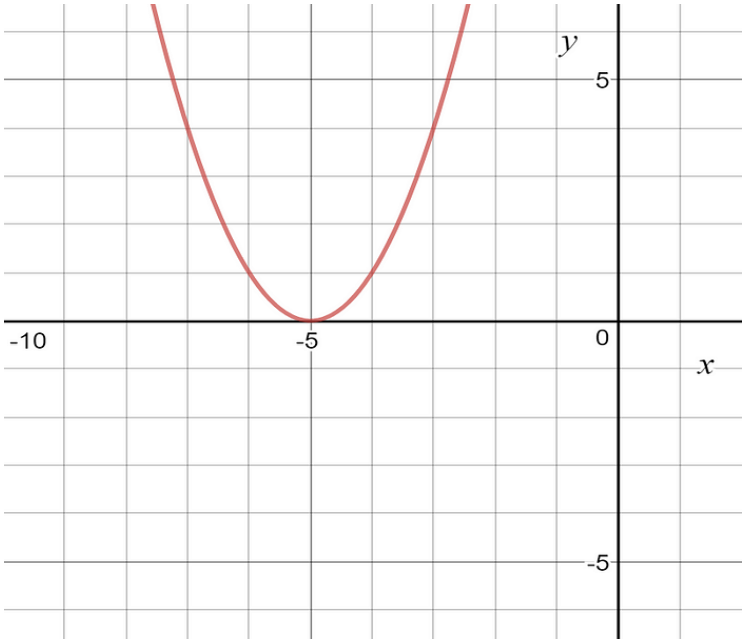
Theory Concepts:

What is gradient descent ?

It is an optimization algorithm to find the minimum of a function. We start with a random point on the function and move in the **negative direction** of the **gradient of the function** to reach the **local/global minima**.

Example:

Question : Find the local minima of the function $y=(x+5)^2$ starting from the point x=3



Solution : We know the answer just by looking at the graph. $y = (x+5)^2$ reaches it's minimum value when x = -5 (i.e when x=-5, y=0). Hence x=-5 is the local and global minima of the function.

Now, let's see how to obtain the same numerically using gradient descent.

Step 1 : Initialize x =3. Then, find the gradient of the function, dy/dx = 2*(x+5).

Step 2 : Move in the direction of the negative of the gradient (Why?). But wait, how much to move? For that, we require a learning rate. Let us assume the **learning rate → 0.01**

Step 3 : Let's perform 2 iterations of gradient descent

Initialize Parameters :

$$X_0 = 3$$

$$Learning\ rate = 0.01$$

$$\frac{dy}{dx} = \frac{d}{dx}(x+5)^2 = 2 * (x+5)$$

Iteration 1 :

$$X_1 = X_0 - (learning\ rate) * (\frac{dy}{dx})$$

$$X_1 = 3 - (0.01) * \big(2 * (3+5)\big) = 2.84$$

Iteration 2 :

$$X_2 = X_1 - (learning\ rate) * (\frac{dy}{dx})$$

$$X_2 = 2.84 - (0.01) * \big(2 * (2.84+5)\big) = 2.6832$$

Step 4 : We can observe that the X value is slowly decreasing and should converge to -5 (the local minima).

Implementing Gradient Descent in Python

Importing libraries

```
import numpy as np
import matplotlib.pyplot as plt
```

We then define the function f(x) and its derivative f'(x) –

```
def f(x):
  return (x+3)**2

def df(x):
  return 2*x + 6
```

F(x) is the function that has to be decreased, and df is its derivative (x). The gradient descent approach makes use of the derivative to guide itself toward the minimum by revealing the function's slope along the way.

The gradient descent function is then defined.

```
def gradient_descent(initial_x, learning_rate, num_iterations):
  x = initial_x
  x_history = [x]

  for i in range(num_iterations):
     gradient = df(x)
     x = x - learning_rate * gradient
     x_history.append(x)

  return x, x_history
```

The starting value of x, the learning rate, and the desired number of iterations are sent to the gradient descent function. In order to save the values of x after each iteration, it initializes x to its original value and generates an empty list. The method then executes the gradient descent for the provided number of iterations, changing x every iteration in accordance with the equation x = x - learning rate * gradient. The function produces a list of every iteration's x values together with the final value of x.

The gradient descent function may now be used to locate the local minimum of f(x) –

```
initial_x = 2
learning_rate = 0.1
num_iterations = 50

x, x_history = gradient_descent(initial_x, learning_rate, num_iterations)

print("Local minimum: {:.2f}".format(x))
```

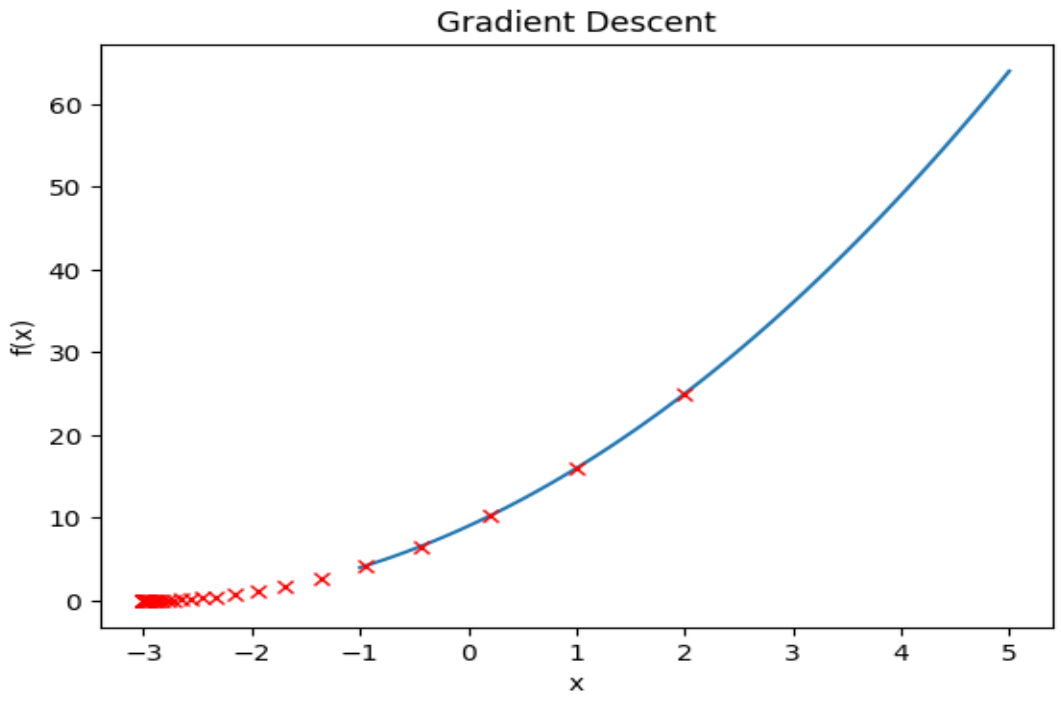
Output

```
Local minimum: -3.00
```

In this illustration, x is set to 2 at the beginning, with a learning rate of 0.1, and 50 iterations are run. Finally, we publish the value of x, which ought to be close to the local minimum at x=-3.

Plotting the function f(x) and the values of x for each iteration allows us to see the gradient descent process in action –

```
#Create a range of x values to plot
x_vals = np.linspace(-1, 5, 100)

#Plot the function f(x)
plt.plot(x_vals, f(x_vals))

# Plot the values of x at each iteration
plt.plot(x_history, f(np.array(x_history)), 'rx')

#Label the axes and add a title
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent')

#Show the plot
plt.show()
```



Conclusion: Thus we have implemented Gradient Descent Algorithm to find the local minima of a function $y=(x+3)^2$ starting from the point x=2 that is -3.

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.

In [198]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Importing the required libraries.
```

In [199]:
```python
from sklearn.cluster import KMeans, k_means #For clustering
from sklearn.decomposition import PCA #Linear Dimensionality reduction.
```

In [200]:
```python
df = pd.read_csv("sales_data_sample.csv") #Loading the dataset.
```

Preprocessing

In [201]:
```python
df.head()
```

Out[201]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINI
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airp Aven
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 rue l'Abba
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 rue Colonel Pier Av
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillsi I
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong :

5 rows × 25 columns

In [202]:
```python
df.shape
```

Out[202]:
```
(2823, 25)
```

In [203]:
```python
df.describe()
```

Out[203]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000
mean	10258.725115	35.092809	83.658544	6.466171	3553.889072	2.717676	7.092455	2003.81509	100.715551
std	92.085478	9.741443	20.174277	4.225841	1841.865106	1.203878	3.656633	0.69967	40.187912
min	10100.000000	6.000000	26.880000	1.000000	482.130000	1.000000	1.000000	2003.00000	33.000000
25%	10180.000000	27.000000	68.860000	3.000000	2203.430000	2.000000	4.000000	2003.00000	68.000000
50%	10262.000000	35.000000	95.700000	6.000000	3184.800000	3.000000	8.000000	2004.00000	99.000000
75%	10333.500000	43.000000	100.000000	9.000000	4508.000000	4.000000	11.000000	2004.00000	124.000000
max	10425.000000	97.000000	100.000000	18.000000	14082.800000	4.000000	12.000000	2005.00000	214.000000

In [204]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDERNUMBER       2823 non-null   int64
 1   QUANTITYORDERED   2823 non-null   int64
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCTLINE       2823 non-null   object
 11  MSRP              2823 non-null   int64
 12  PRODUCTCODE       2823 non-null   object
 13  CUSTOMERNAME      2823 non-null   object
 14  PHONE             2823 non-null   object
 15  ADDRESSLINE1      2823 non-null   object
 16  ADDRESSLINE2      302 non-null    object
 17  CITY              2823 non-null   object
 18  STATE             1337 non-null   object
 19  POSTALCODE        2747 non-null   object
 20  COUNTRY           2823 non-null   object
 21  TERRITORY         1749 non-null   object
 22  CONTACTLASTNAME   2823 non-null   object
 23  CONTACTFIRSTNAME  2823 non-null   object
 24  DEALSIZE          2823 non-null   object
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [205]:
```python
df.isnull().sum()
```

Out[205]:
```
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
PHONE                  0
ADDRESSLINE1           0
ADDRESSLINE2        2521
CITY                   0
STATE               1486
POSTALCODE            76
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

In [206]:
```python
df.dtypes
```

Out[206]:
```
ORDERNUMBER           int64
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
ORDERDATE            object
STATUS               object
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
PRODUCTLINE          object
MSRP                  int64
PRODUCTCODE          object
CUSTOMERNAME         object
PHONE                object
ADDRESSLINE1         object
ADDRESSLINE2         object
CITY                 object
STATE                object
POSTALCODE           object
COUNTRY              object
TERRITORY            object
CONTACTLASTNAME      object
CONTACTFIRSTNAME     object
DEALSIZE             object
dtype: object
```

In [207]:
```python
df_drop  = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRITORY', 'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CUSTOMERNAME', 'ORDERNUMBER']
df = df.drop(df_drop, axis=1) #Dropping the categorical uneccessary columns along with columns having null values. Can't fill the null values are there are alot of null values.
```

In [208]:
```python
df.isnull().sum()
```

Out[208]:
```
QUANTITYORDERED    0
PRICEEACH          0
ORDERLINENUMBER    0
SALES              0
ORDERDATE          0
QTR_ID             0
MONTH_ID           0
YEAR_ID            0
PRODUCTLINE        0
MSRP               0
PRODUCTCODE        0
COUNTRY            0
DEALSIZE           0
dtype: int64
```

In [209]:
```python
df.dtypes
```

Out[209]:
```
QUANTITYORDERED      int64
PRICEEACH          float64
ORDERLINENUMBER      int64
SALES              float64
ORDERDATE           object
QTR_ID               int64
MONTH_ID             int64
YEAR_ID              int64
PRODUCTLINE         object
MSRP                 int64
PRODUCTCODE         object
COUNTRY             object
DEALSIZE            object
dtype: object
```

In [ ]:
```python
# Checking the categorical columns.
```

In [210]:
```python
df['COUNTRY'].unique()
```

Out[210]:
```
array(['USA', 'France', 'Norway', 'Australia', 'Finland', 'Austria', 'UK',
       'Spain', 'Sweden', 'Singapore', 'Canada', 'Japan', 'Italy',
       'Denmark', 'Belgium', 'Philippines', 'Germany', 'Switzerland',
       'Ireland'], dtype=object)
```

In [211]:
```python
df['PRODUCTLINE'].unique()
```

Out[211]:
```
array(['Motorcycles', 'Classic Cars', 'Trucks and Buses', 'Vintage Cars',
       'Planes', 'Ships', 'Trains'], dtype=object)
```

In [212]:
```python
df['DEALSIZE'].unique()
```

Out[212]:
```
array(['Small', 'Medium', 'Large'], dtype=object)
```

In [213]:
```python
productline = pd.get_dummies(df['PRODUCTLINE']) #Converting the categorical columns.
Dealsize = pd.get_dummies(df['DEALSIZE'])
```

In [214]:
```python
df = pd.concat([df,productline,Dealsize], axis = 1)
```

In [215]:
```python
df_drop  = ['COUNTRY','PRODUCTLINE','DEALSIZE'] #Dropping Country too as there are alot of countries.
df = df.drop(df_drop, axis=1)
```

In [216]:
```python
df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes #Converting the datatype.
```

In [217]:
```python
df.drop('ORDERDATE', axis=1, inplace=True) #Dropping the Orderdate as Month is already included.
```

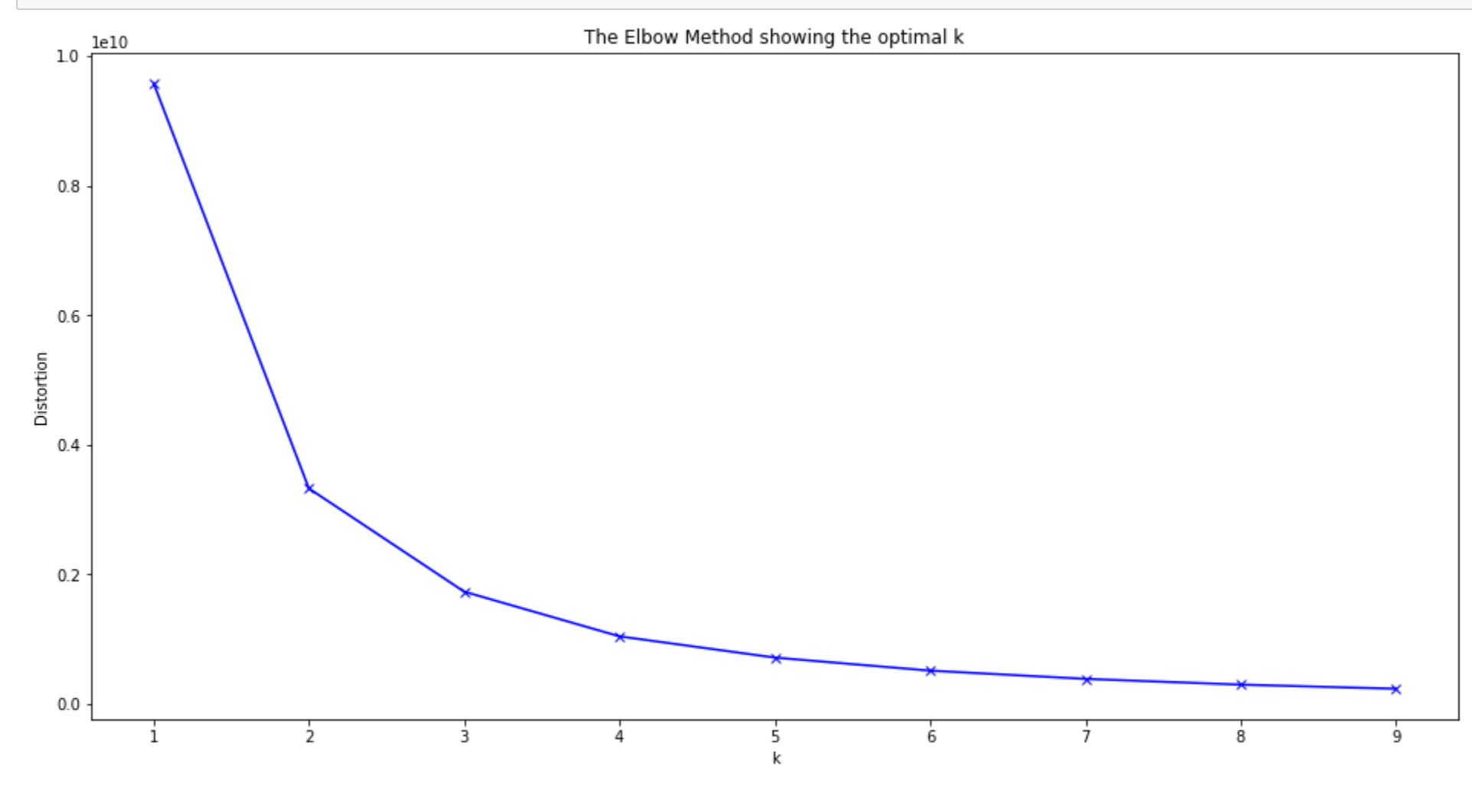
In [218]:
```python
df.dtypes #All the datatypes are converted into numeric
```

Out[218]:
```
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
MSRP                  int64
PRODUCTCODE            int8
Classic Cars          uint8
Motorcycles           uint8
Planes                uint8
Ships                 uint8
Trains                uint8
Trucks and Buses      uint8
Vintage Cars          uint8
Large                 uint8
Medium                uint8
Small                 uint8
dtype: object
```

Plotting the Elbow Plot to determine the number of clusters.

In [219]:
```python
distortions = [] # Within Cluster Sum of Squares from the centroid
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)   #Appeding the intertia to the Distortions
```

In [220]:
```python
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



As the number of k increases Inertia decreases.

Observations: A Elbow can be observed at 3 and after that the curve decreases gradually.

In [221]:
```python
X_train = df.values #Returns a numpy array.
```

In [222]:
```python
X_train.shape
```

Out[222]:
```
(2823, 19)
```

In [223]:
```python
model = KMeans(n_clusters=3,random_state=2) #Number of cluster = 3
model = model.fit(X_train) #Fitting the values to create a model.
predictions = model.predict(X_train) #Predicting the cluster values (0,1,or 2)
```

In [225]:
```python
unique,counts = np.unique(predictions,return_counts=True)
```

In [226]:
```python
counts = counts.reshape(1,3)
```

In [227]:
```python
counts_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3'])
```

In [228]:
```python
counts_df.head()
```

Out[228]:

	Cluster1	Cluster2	Cluster3
0	1083	1367	373

Visualization

In [229]:
```python
pca = PCA(n_components=2) #Converting all the features into 2 columns to make it easy to visualize using Principal Component Analysis.
```

In [230]:
```python
reduced_X = pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2']) #Creating a DataFrame.
```

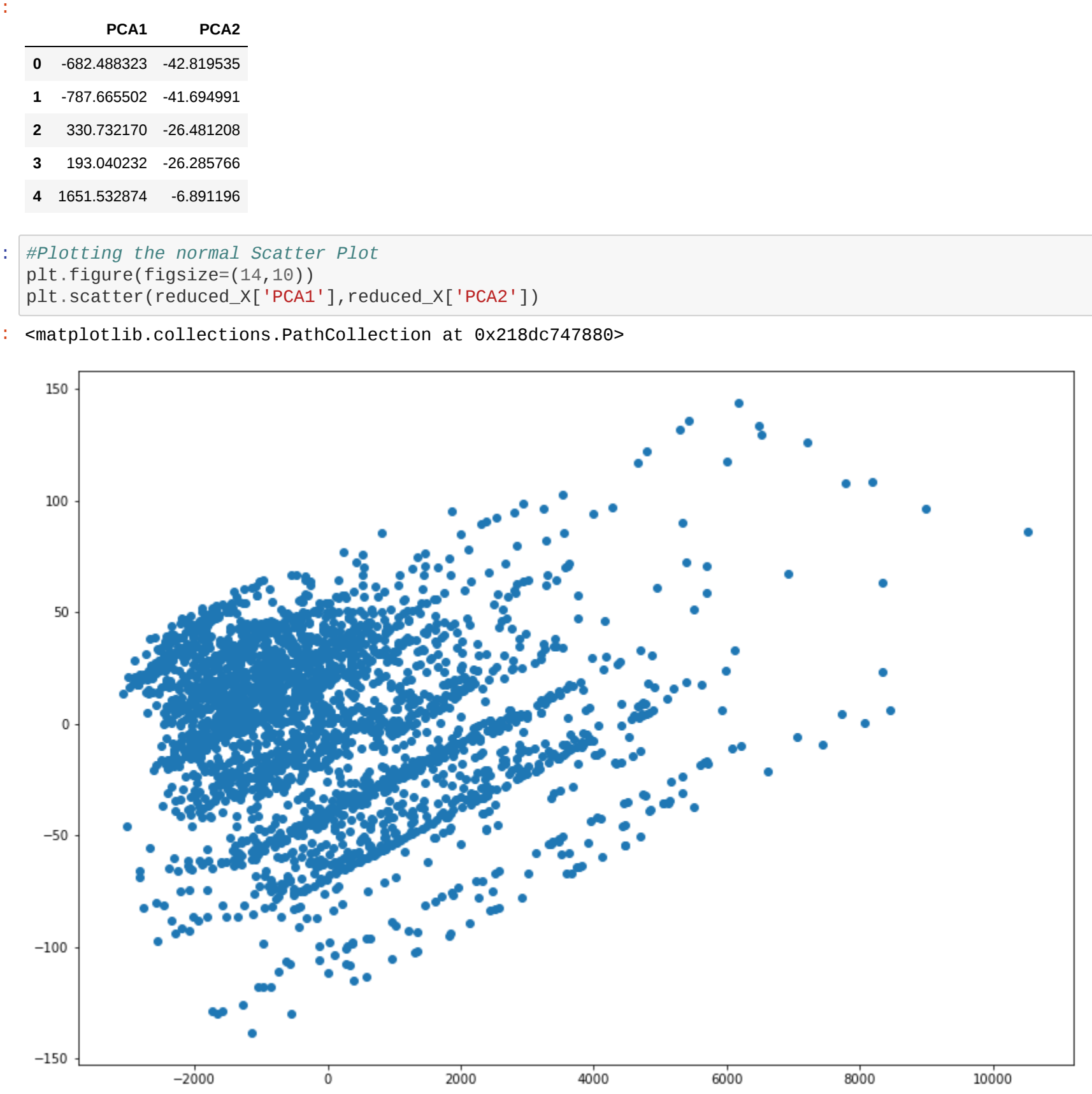
In [231]:
```python
reduced_X.head()
```

Out[231]:

	PCA1	PCA2
0	-682.488323	-42.819535
1	-787.665502	-41.694991
2	330.732170	-26.481208
3	193.040232	-26.285766
4	1651.532874	-6.891196

In [232]:
```python
#Plotting the normal Scatter Plot
plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
```

Out[232]:
```
<matplotlib.collections.PathCollection at 0x218dc747880>
```



In [233]:
```python
model.cluster_centers_ #Finding the centriods. (3 Centriods in total. Each Array contains a centroids for particular feature )
```

Out[233]:
```
array([[ 3.72031394e+01,  9.52120960e+01,  6.44967682e+00,
         4.13868425e+03,  2.72022161e+00,  7.09879963e+00,
         2.00379409e+03,  1.13248384e+02,  5.04469067e+01,
         3.74884580e-01,  1.15420129e-01,  9.41828255e-02,
         8.21791320e-02,  1.84672207e-02,  1.16343490e-01,
         1.98522622e-01,  2.08166817e-17,  1.00000000e+00,
        -6.66133815e-16],
       [ 3.08302853e+01,  7.00755230e+01,  6.67300658e+00,
         2.12409474e+03,  2.71762985e+00,  7.09509876e+00,
         2.00381127e+03,  7.84784199e+01,  6.24871982e+01,
         2.64813460e-01,  1.21433797e-01,  1.29480614e-01,
         1.00219459e-01,  3.87710315e-02,  9.21726408e-02,
         2.53108998e-01,  6.93889390e-18,  6.21799561e-02,
         9.37820044e-01],
       [ 4.45871314e+01,  9.98931099e+01,  5.75603217e+00,
         7.09596863e+03,  2.71045576e+00,  7.06434316e+00,
         2.00389008e+03,  1.45823056e+02,  3.14959786e+01,
         5.33512064e-01,  1.07238606e-01,  7.23860590e-02,
         2.14477212e-02,  1.07238606e-02,  1.31367292e-01,
         1.23324397e-01,  4.20911528e-01,  5.79088472e-01,
         5.55111512e-17]])
```

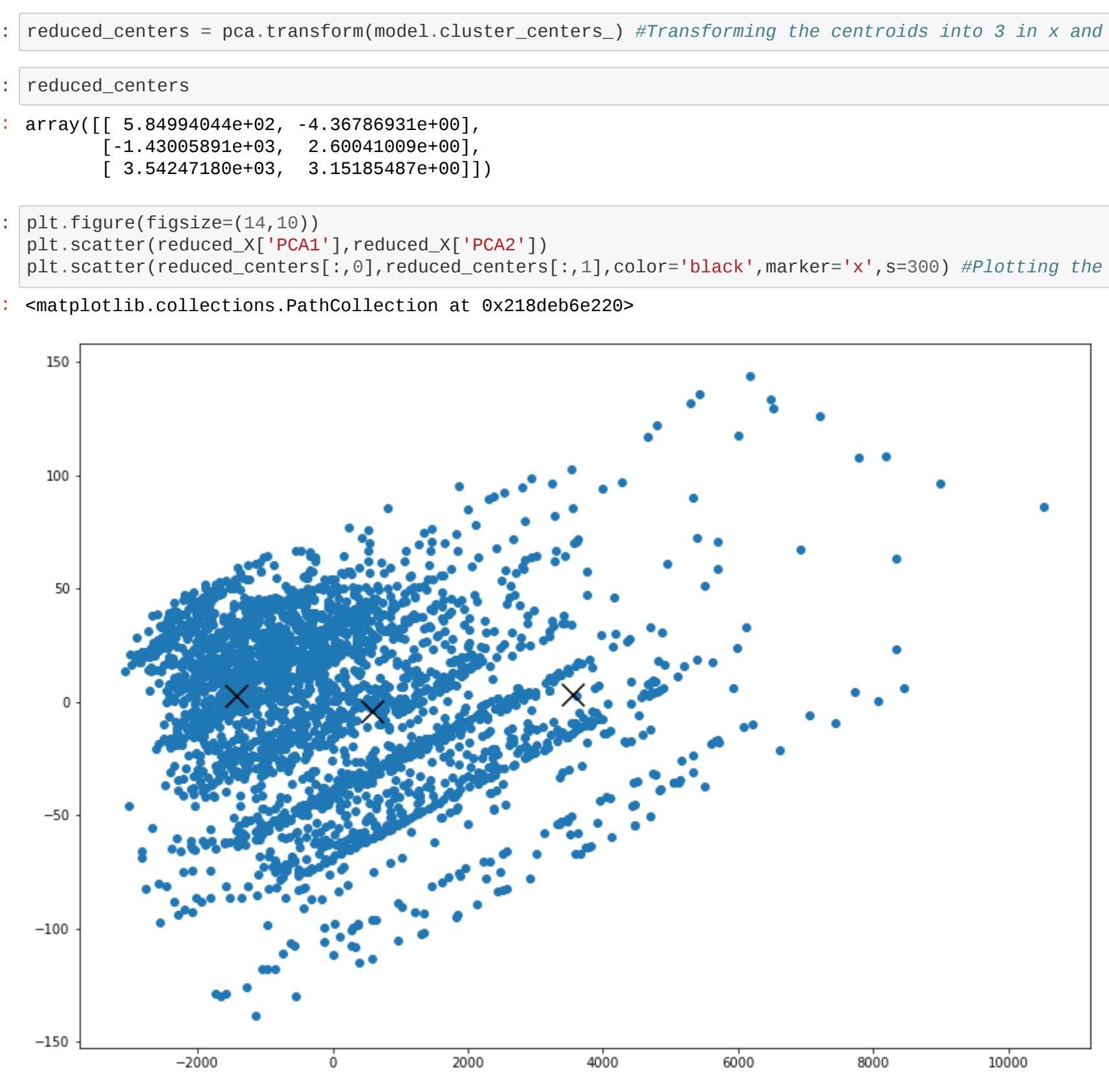
In [234]:
```python
reduced_centers = pca.transform(model.cluster_centers_) #Transforming the centroids into 3 in x and y coordinates
```

In [235]:
```python
reduced_centers
```

Out[235]:
```
array([[ 5.84994044e+02, -4.36786931e+00],
       [-1.43005891e+03,  2.60041009e+00],
       [ 3.54247180e+03,  3.15185487e+00]])
```

In [236]:
```python
plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300) #Plotting the centriods
```

Out[236]:
```
<matplotlib.collections.PathCollection at 0x218deb6e220>
```



In [237]:
```python
reduced_X['Clusters'] = predictions #Adding the Clusters to the reduced dataframe.
```

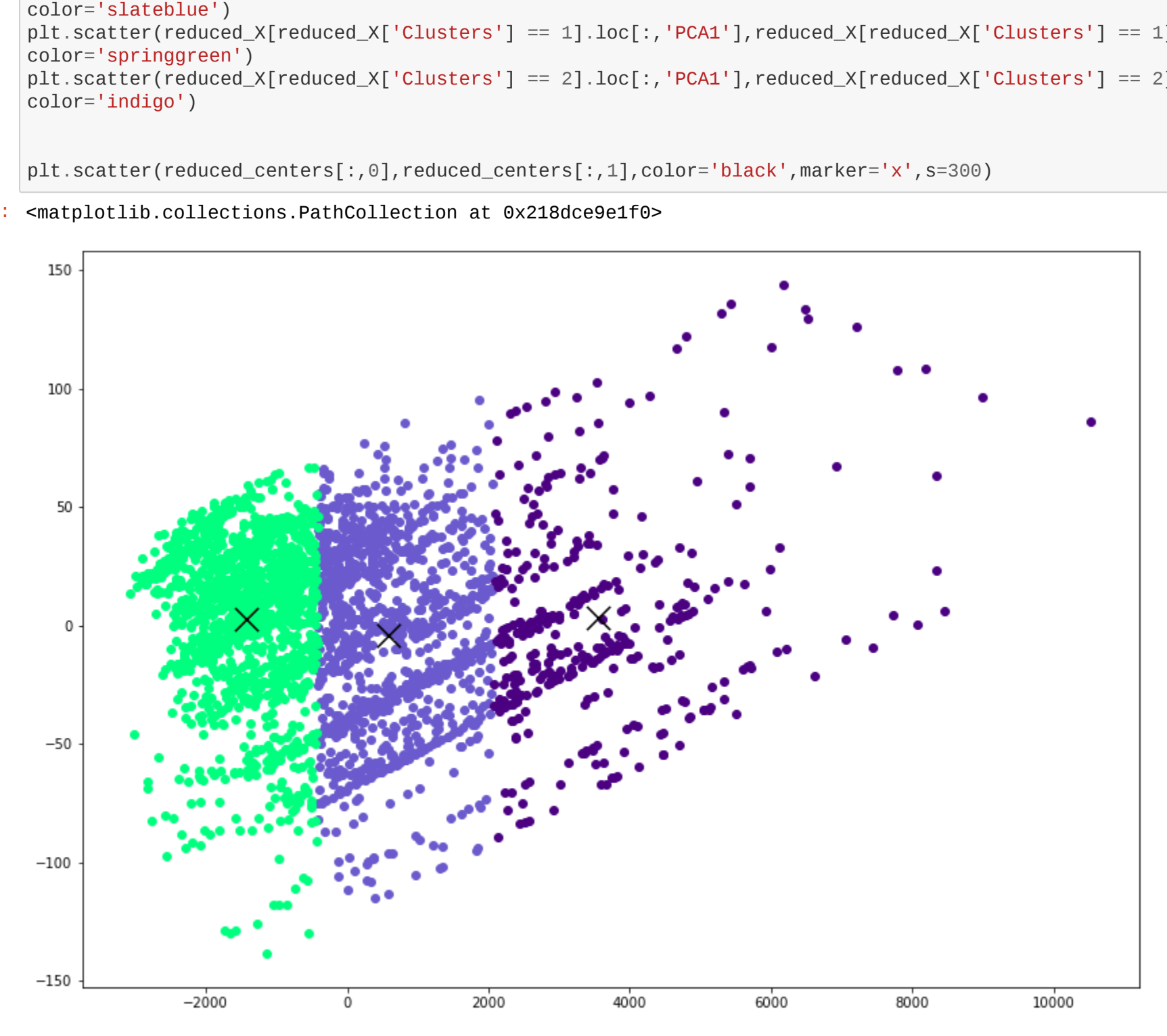
In [238]:
```python
reduced_X.head()
```

Out[238]:

	PCA1	PCA2	Clusters
0	-682.488323	-42.819535	1
1	-787.665502	-41.694991	1
2	330.732170	-26.481208	0
3	193.040232	-26.285766	0
4	1651.532874	-6.891196	0

In [239]:
```python
#Plotting the clusters
plt.figure(figsize=(14,10))
#                     taking the cluster number and first column           taking the same cluster number and second column      Assigning the color
plt.scatter(reduced_X[reduced_X['Clusters'] == 0].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] == 0].loc[:,'PCA2'],color='slateblue')
plt.scatter(reduced_X[reduced_X['Clusters'] == 1].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] == 1].loc[:,'PCA2'],color='springgreen')
plt.scatter(reduced_X[reduced_X['Clusters'] == 2].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] == 2].loc[:,'PCA2'],color='indigo')


plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300)
```

Out[239]:
```
<matplotlib.collections.PathCollection at 0x218dce9e1f0>
```



In [ ]:

GURU GOBIND SINGH FOUNDATION'S

Guru Gobind Singh College of Engineering & Research Centre, Nashik

Department of Computer Engineering

Academic Year 2023-24

Final year Sem I

Machine Learning

Miniproject Report

Class: BE Computer

Guided By:

Mr. P. K. Bachhav

Title: Loan Prediction Using Machine Learning

Group Members:

Roll No.	Name
B-28	Ruchita Rupesh Raut
B-16	Divesh Sanjay Patil

Mini-Project Report

Name of Program: Computer Engineering **Academic Year:**2023-24
Semester: BECO-Sem I C**ourse code:** 410246
Name of Course: Laboratory Practice-III (Machine Learning)

Title of Mini-Project: "Loan Prediction Using Machine Learning"

1.0 Rationale:

The rationale for utilizing machine learning in loan prediction is grounded in the need for more accurate, data-driven, and efficient lending practices. Traditional loan approval processes, which often rely heavily on manual assessments and subjective judgment, can be error-prone and time-consuming. Machine learning offers a solution by leveraging historical data to create predictive models that can analyze an applicant's financial history, creditworthiness, and various risk factors in a systematic and unbiased manner. These models can identify patterns and relationships that are not easily discernible through human assessment, leading to more informed lending decisions. By using machine learning in loan prediction, financial institutions can mitigate risks associated with loan defaults, optimize their lending portfolios, and improve the overall quality of their loan portfolios. Additionally, this approach can enhance the accessibility of loans for qualified borrowers, fostering financial inclusion and fairness. In summary, the rationale for loan prediction using machine learning lies in its potential to enhance the accuracy and efficiency of lending processes, reduce risks, and promote a more inclusive and data-driven approach to financial services.

2.0 Aim /Benefits of Mini-Project:

The aim of loan prediction using Machine Learning is to develop a predictive model that can assess the creditworthiness of loan applicants with greater accuracy and efficiency. This model leverages historical data, including information about previous loan applicants, their financial profiles, and whether they repaid their loans on time or defaulted. By analyzing this data, the machine learning model can identify patterns and factors that are indicative of a borrower's likelihood to repay a loan successfully or become a defaulter. The ultimate goal is to assist financial institutions, such as banks and lending agencies, in making more informed lending decisions, reducing the risk of loan defaults, and ensuring that loans are allocated to individuals and businesses who are more likely to meet their repayment obligations. This not only benefits the lenders by minimizing financial losses but also aids borrowers in accessing loans that align with their financial capabilities. Machine learning plays a crucial role in automating and optimizing the loan approval process, making it more efficient and equitable.

Course Outcomes achieved (COs):

a) Apply preprocessing techniques on datasets (CO1)
b) Implement and evaluate linear regression and random forest regression models (CO2)

4.0 Literature Review: -

The use of machine learning in loan prediction has gained significant interest in the financial industry. Researchers have developed predictive models using various algorithms, such as decision trees, random forests, support vector machines, and neural networks, to assess creditworthiness.These models use historical data on loan applicants to make informed decisions. The literature highlights the potential benefits of machine learning in improving loan approval processes and identifying non-linear and complex data patterns. Researchers have also investigated the ethical and fairness aspects of machine learning-based loan prediction to prevent biases or discrimination in lending. The literature highlights the growing importance of machine learning in transforming lending practices and addressing risk management concerns in the banking and financial sectors.

5.0 Actual Methodology followed:

In this Loan Prediction using Machine Learning Mini Project, the methodology begins with loading the dataset and conducting an initial exploration to understand its structure and content. Data preprocessing steps follow, addressing missing values and creating new features that categorize passenger ages and titles. These categorical variables are then one-hot encoded for machine learning. Data visualization is employed to visually analyze survival rates across different passenger groups. The dataset is split into features and the target variable for training and validation. Decision tree is a type of supervised education algorithm (having a pre- defined target variable) that is generally used in category problems. In this approach, we disassociate the population or sample into two or added homogeneous sets (or sub-populations) based on the most significant splitter/ differentiator in input variables.Decision trees use multiple algorithms to decide to disunite a bump into two or added sub- knots. The creation of sub- knots increases the unsophistication of attendant sub- knots. In other words, we can say that chasteness of the bump increases with respect to the target variable.This comprehensive methodology covers data analysis, preprocessing, machine learning, and model evaluation, ultimately allowing practical predictions based on historical data.

6.0 Actual Code of Program:

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

In [2]:
```
df = pd.read_csv("loan_approval_dataset.csv")
```

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

Out[3]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_te
0	1	2	Graduate	No	9600000	29900000	
1	2	0	Not Graduate	Yes	4100000	12200000	
2	3	3	Graduate	No	9100000	29700000	
3	4	3	Graduate	No	8200000	30700000	
4	5	5	Not Graduate	Yes	9800000	24200000	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype 
---                           --------------  ----- 
 0   loan_id                   4269 non-null   int64 
 1    no_of_dependents         4269 non-null   int64 
 2    education                4269 non-null   object
 3    self_employed            4269 non-null   object
 4    income_annum             4269 non-null   int64 
 5    loan_amount              4269 non-null   int64 
 6    loan_term                4269 non-null   int64 
 7    cibil_score              4269 non-null   int64 
 8    residential_assets_value 4269 non-null   int64 
 9    commercial_assets_value  4269 non-null   int64 
 10   luxury_assets_value      4269 non-null   int64 
 11   bank_asset_value         4269 non-null   int64 
 12   loan_status              4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score
count	4269.000000	4269.000000	4.269000e+03	4.269000e+03	4269.000000	4269.000000
mean	2135.000000	2.498712	5.059124e+06	1.513345e+07	10.900445	599.936051
std	1232.498479	1.695910	2.806840e+06	9.043363e+06	5.709187	172.430401
min	1.000000	0.000000	2.000000e+05	3.000000e+05	2.000000	300.000000
25%	1068.000000	1.000000	2.700000e+06	7.700000e+06	6.000000	453.000000
50%	2135.000000	3.000000	5.100000e+06	1.450000e+07	10.000000	600.000000
75%	3202.000000	4.000000	7.500000e+06	2.150000e+07	16.000000	748.000000
max	4269.000000	5.000000	9.900000e+06	3.950000e+07	20.000000	900.000000

In [6]:
```
df.shape
```

Out[6]: (4269, 13)

In [7]:
```
df['Movable_assets'] = df[' bank_asset_value'] + df[' luxury_assets_value']


df['Immovable_assets'] = df[' residential_assets_value'] + df[' commercial_asset
```

In [8]:
```
df.drop(columns=[' bank_asset_value',' luxury_assets_value', ' residential_asset
```

In [9]:
```
def uniquevals(col):
    print(f'Unique Values in {col} is : {df[col].unique()}')

def valuecounts(col):
    print(f'Valuecounts of {col} is: {len(df[col].value_counts())}')




for col in df.columns:
    valuecounts(col)
#     uniquevals(col)
    print("-"*75)
```

```
Valuecounts of loan_id is: 4269
--------------------------------------------------------------------------------
Valuecounts of  no_of_dependents is: 6
--------------------------------------------------------------------------------
Valuecounts of  education is: 2
--------------------------------------------------------------------------------
Valuecounts of  self_employed is: 2
--------------------------------------------------------------------------------
Valuecounts of  income_annum is: 98
--------------------------------------------------------------------------------
Valuecounts of  loan_amount is: 378
--------------------------------------------------------------------------------
Valuecounts of  loan_term is: 10
--------------------------------------------------------------------------------
Valuecounts of  cibil_score is: 601
--------------------------------------------------------------------------------
Valuecounts of  loan_status is: 2
--------------------------------------------------------------------------------
Valuecounts of Movable_assets is: 484
--------------------------------------------------------------------------------
Valuecounts of Immovable_assets is: 406
--------------------------------------------------------------------------------
```

In [10]:
```
catvars  =  df.select_dtypes(include=['object']).columns
numvars  =  df.select_dtypes(include  =  ['int32','int64','float32','float64']).colu

catvars,numvars
```

Out[10]:
```
(Index([' education', ' self_employed', ' loan_status'], dtype='object'),
 Index(['loan_id', ' no_of_dependents', ' income_annum', ' loan_amount',
        ' loan_term', ' cibil_score', 'Movable_assets', 'Immovable_assets'],
       dtype='object'))
```

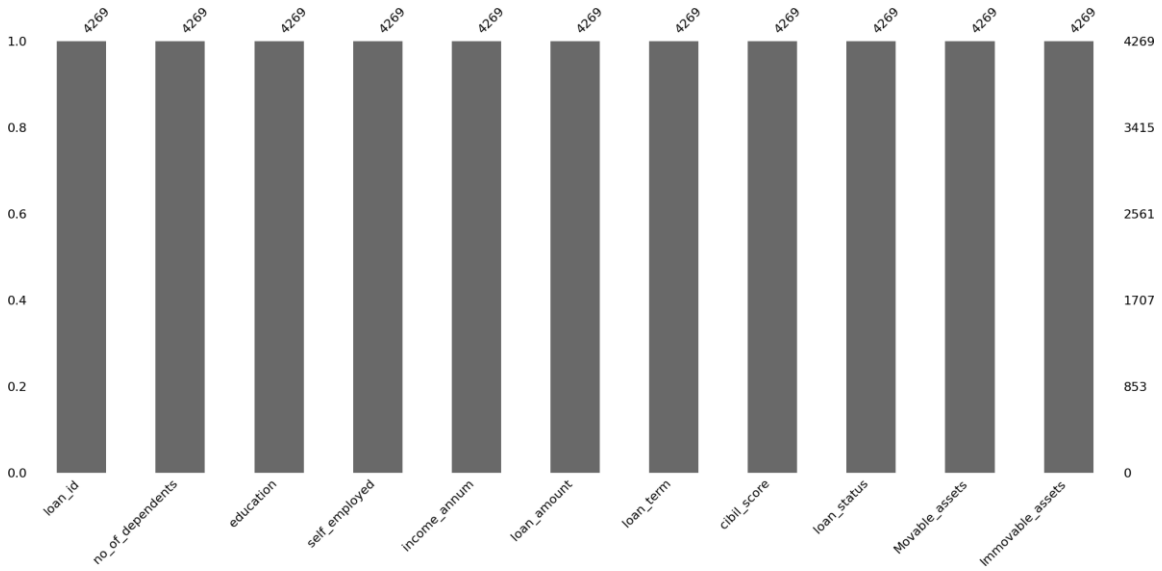
In [11]:
```
pip install missingno
```

```
Requirement already satisfied: missingno in c:\users\parag\appdata\local\progra
ms\python\python310\lib\site-packages (0.5.2)
Requirement already satisfied: numpy in c:\users\parag\appdata\local\programs\p
ython\python310\lib\site-packages (from missingno) (1.24.2)
Requirement already satisfied: matplotlib in c:\users\parag\appdata\local\progr
ams\python\python310\lib\site-packages (from missingno) (3.7.1)
Requirement already satisfied: scipy in c:\users\parag\appdata\local\programs\p
ython\python310\lib\site-packages (from missingno) (1.10.1)
Requirement already satisfied: seaborn in c:\users\parag\appdata\local\programs
\python\python310\lib\site-packages (from missingno) (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\parag\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib->missingno) (1.0.
7)
Requirement already satisfied: cycler>=0.10 in c:\users\parag\appdata\local\pro
grams\python\python310\lib\site-packages (from matplotlib->missingno) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\parag\appdata\loca
l\programs\python\python310\lib\site-packages (from matplotlib->missingno) (4.3
9.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\parag\appdata\loca
l\programs\python\python310\lib\site-packages (from matplotlib->missingno) (1.
4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\parag\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib->missingno) (23.
0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\parag\appdata\local\pr
ograms\python\python310\lib\site-packages (from matplotlib->missingno) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\parag\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib->missingno) (3.0.
9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\parag\appdata\l
ocal\programs\python\python310\lib\site-packages (from matplotlib->missingno)
(2.8.2)
Requirement already satisfied: pandas>=0.25 in c:\users\parag\appdata\local\pro
grams\python\python310\lib\site-packages (from seaborn->missingno) (2.0.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\parag\appdata\local\pro
grams\python\python310\lib\site-packages (from pandas>=0.25->seaborn->missingn
o) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\users\parag\appdata\local\p
rograms\python\python310\lib\site-packages (from pandas>=0.25->seaborn->missing
no) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\parag\appdata\local\program
s\python\python310\lib\site-packages (from python-dateutil>=2.7->matplotlib->mi
ssingno) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [12]:
```
import missingno as msno
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
```

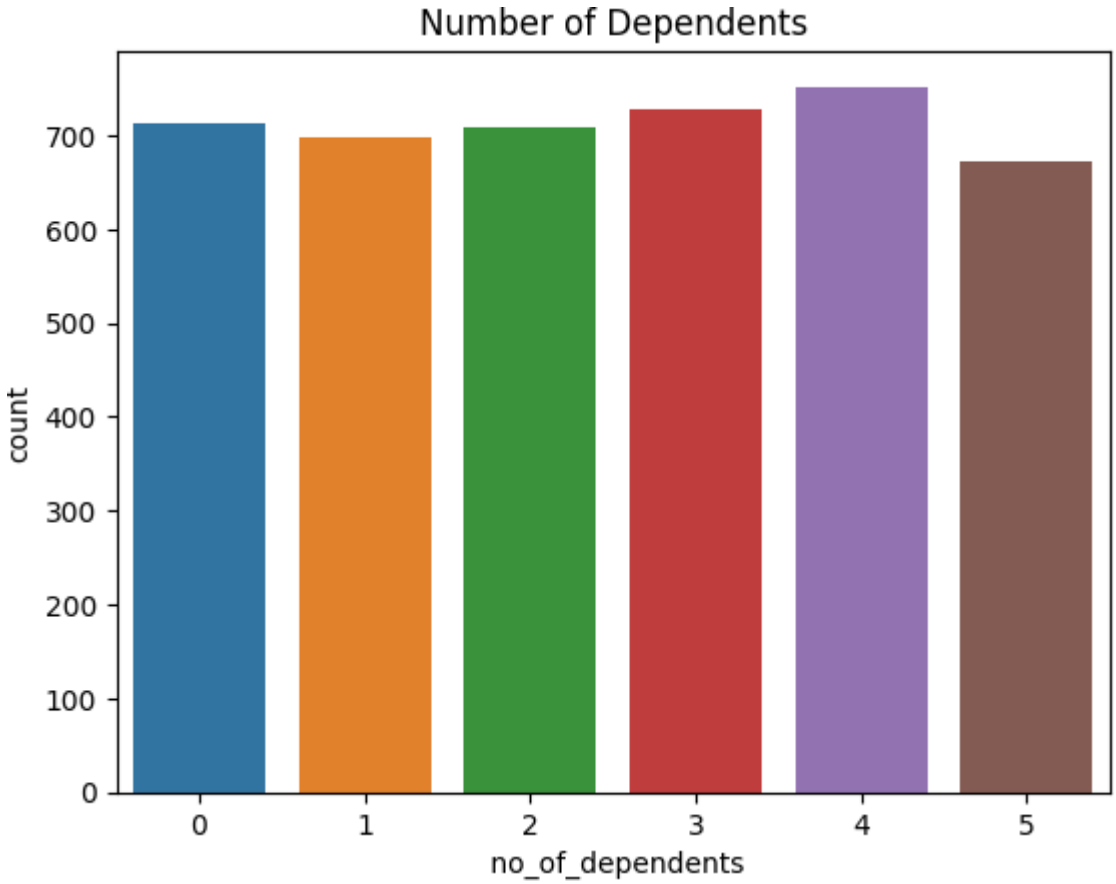
In [13]:
```
msno.bar(df)
```

Out[13]:
```
<Axes: >
```



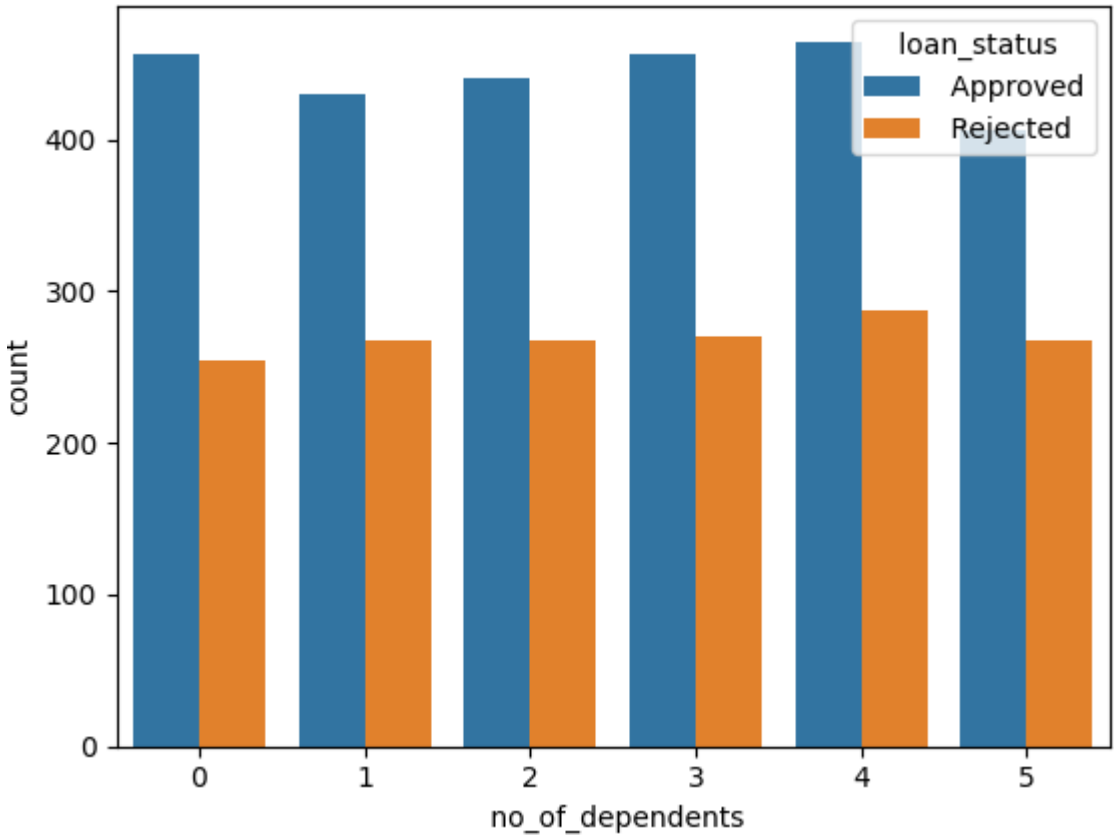
In [14]:
```
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.countplot(x = ' no_of_dependents', data = df).set_title('Number of Dependent
```

Out[14]: Text(0.5, 1.0, 'Number of Dependents')



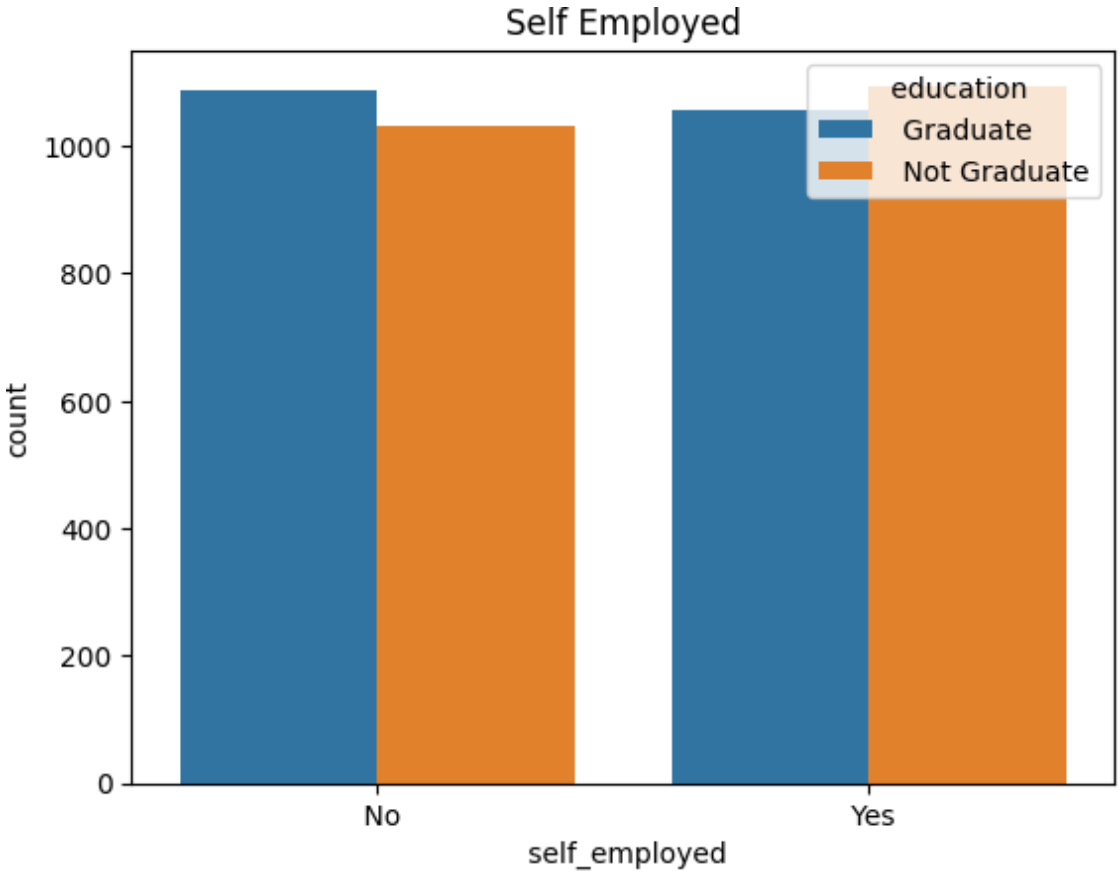
In [15]:
```
sns.countplot(x = ' no_of_dependents', data = df, hue = ' loan_status')
```

Out[15]: <Axes: xlabel=' no_of_dependents', ylabel='count'>



In [16]:
```
sns.countplot(x=' self_employed', data = df, hue = ' education').set_title('Self
```

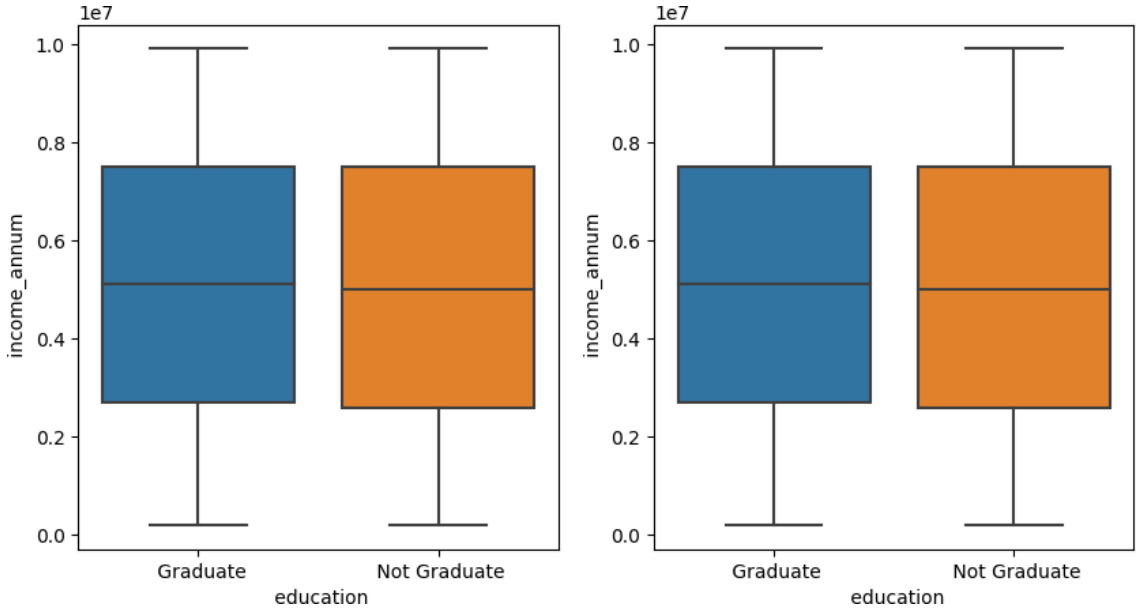
Out[16]:
```
Text(0.5, 1.0, 'Self Employed')
```



In [17]:
```
fig, ax = plt.subplots(1,2,figsize=(10, 5))
sns.boxplot(x = ' education', y = ' income_annum', data = df, ax=ax[0])
```

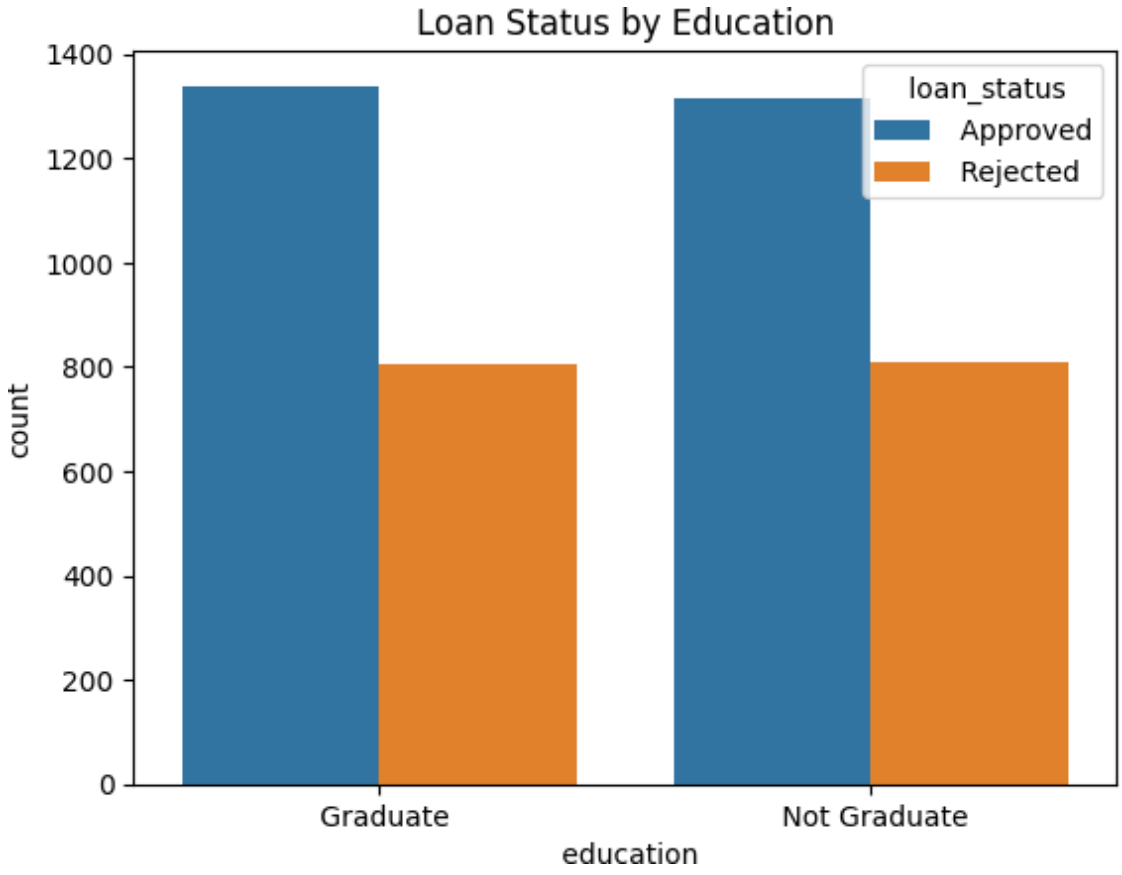
```
sns.boxplot(x = ' education', y = ' income_annum', data = df, ax=ax[1])
```

Out[17]: <Axes: xlabel=' education', ylabel=' income_annum'>



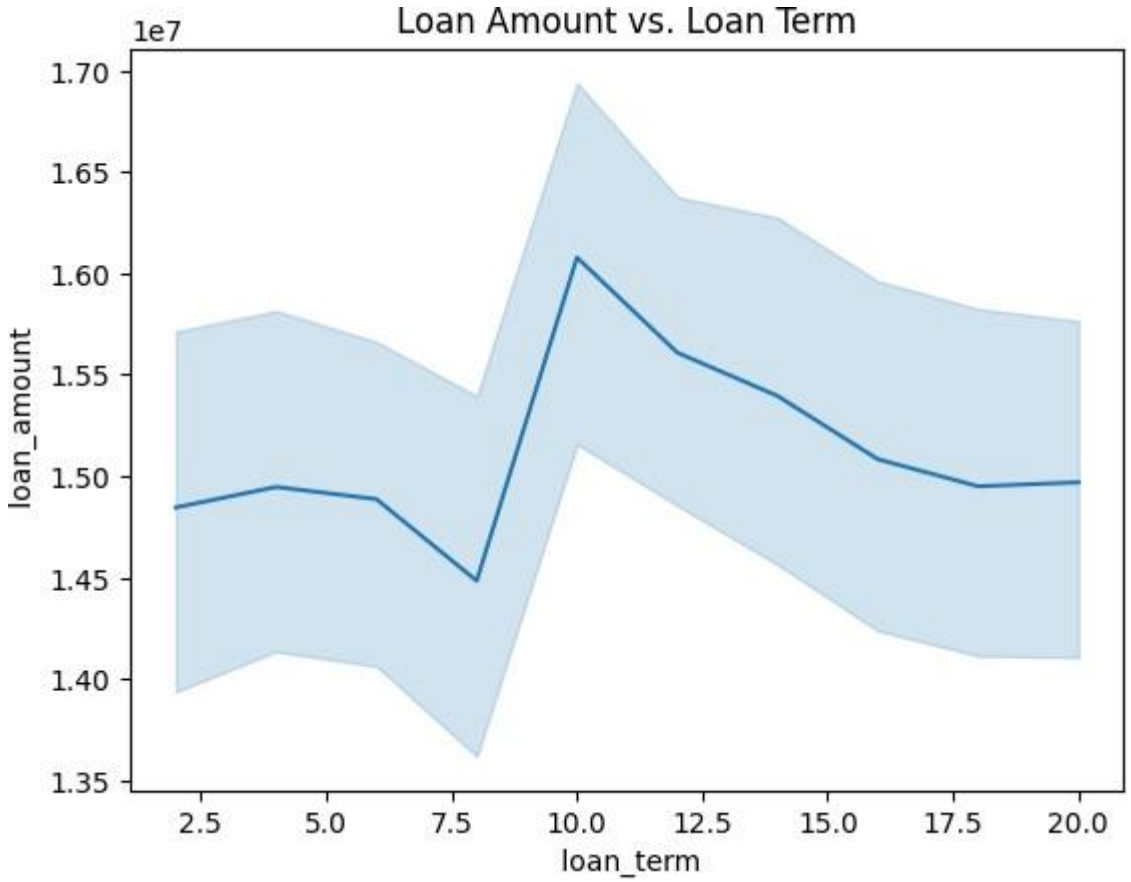
In [18]:
```
sns.countplot(x = ' education', hue = ' loan_status', data = df).set_title('Loan
```

Out[18]: Text(0.5, 1.0, 'Loan Status by Education')



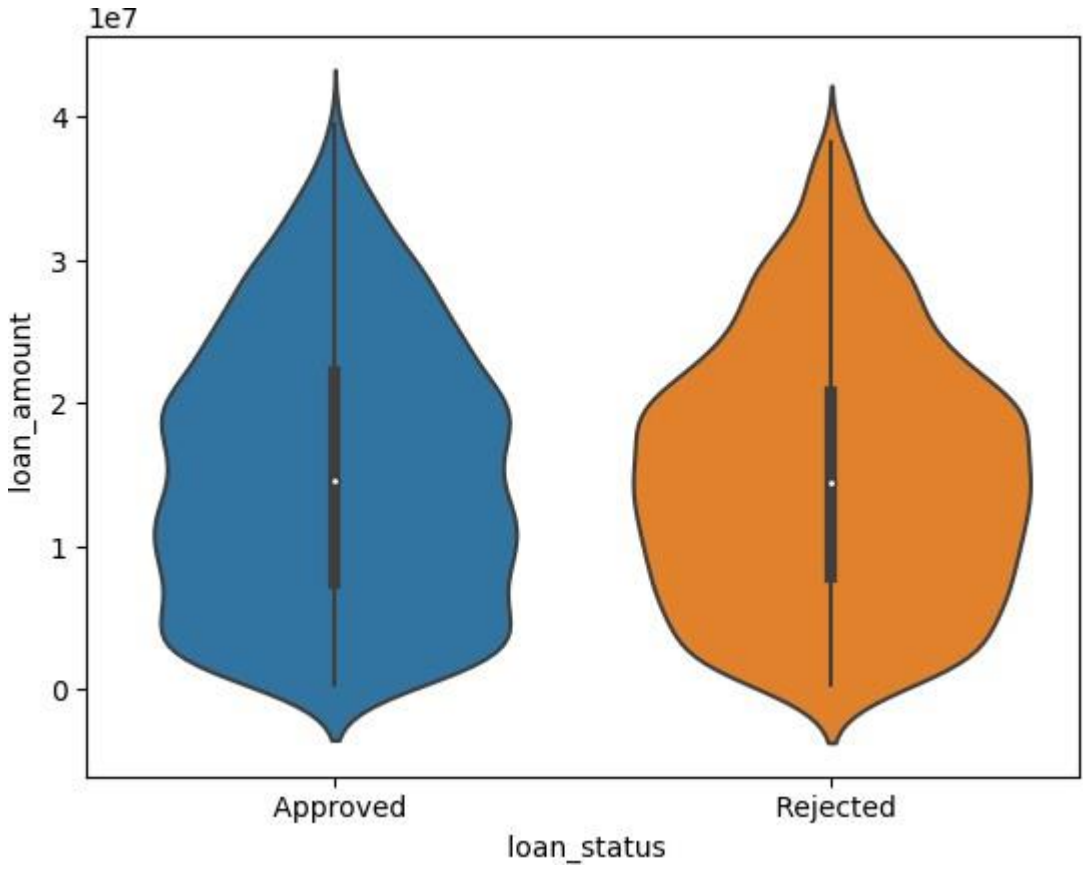
In [19]:
```
sns.lineplot(x = ' loan_term', y = ' loan_amount', data = df).set_title('Loan Am
```

Out[19]: Text(0.5, 1.0, 'Loan Amount vs. Loan Term')



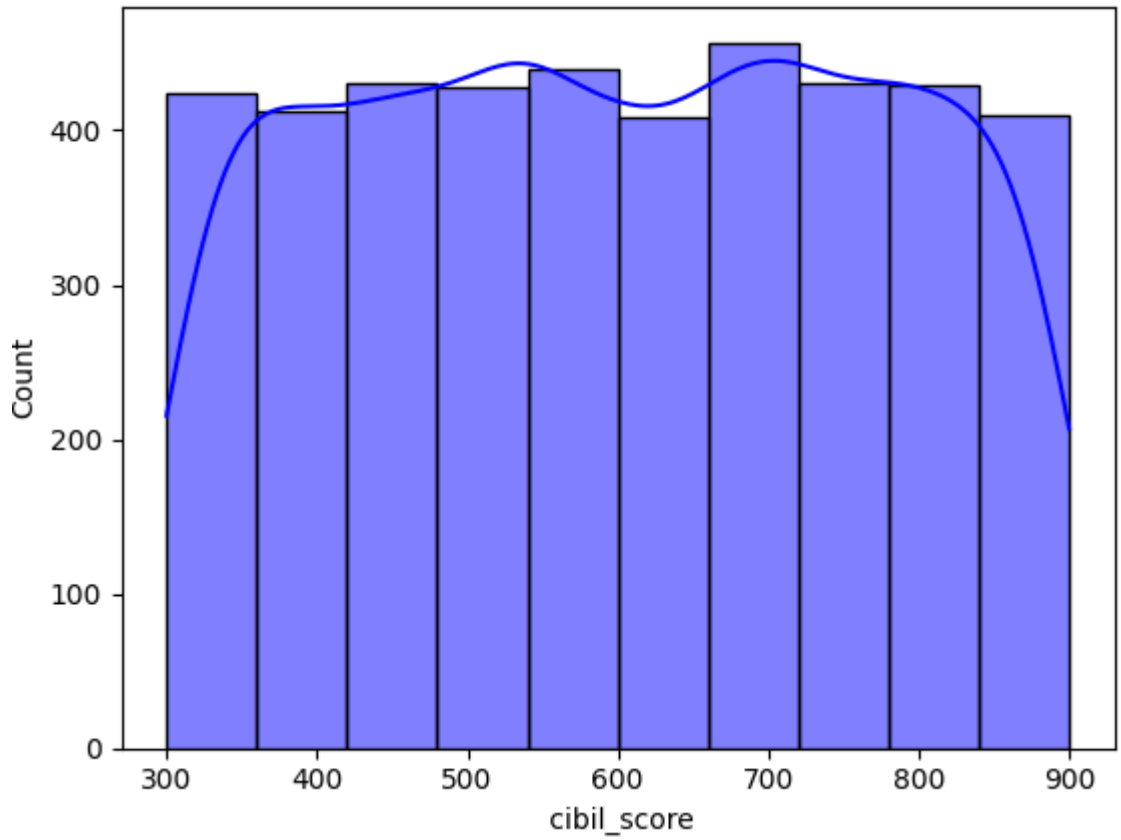
In [20]:
```
sns.violinplot(x=' loan_status', y=' loan_amount', data=df)
```

Out[20]:
```
<Axes: xlabel=' loan_status', ylabel=' loan_amount'>
```



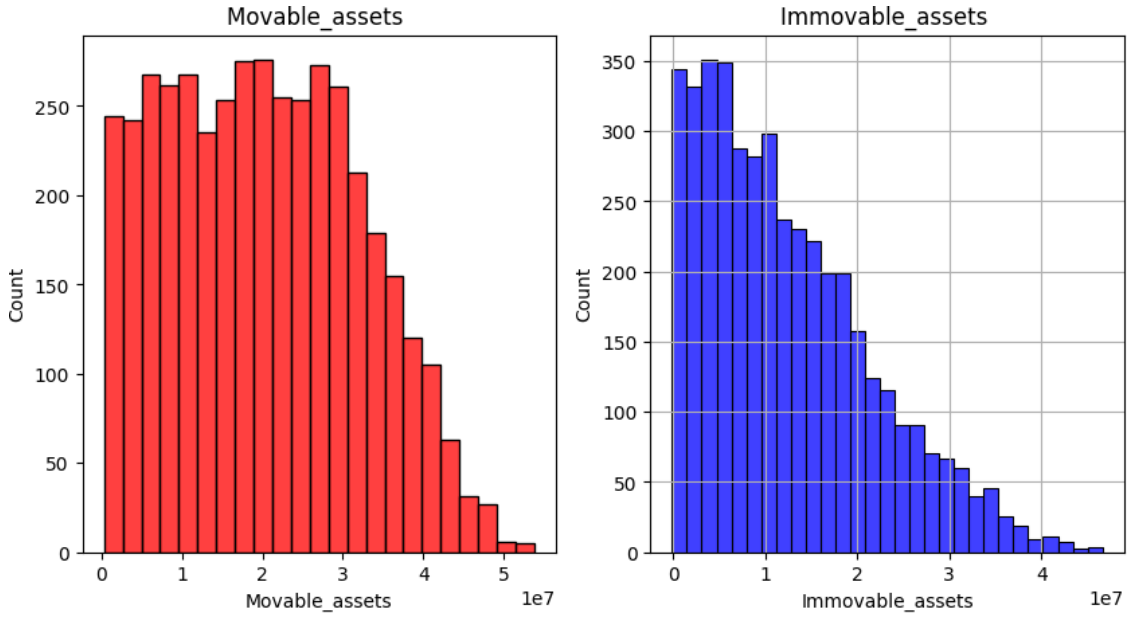
In [21]:
```
sns.histplot(df[" cibil_score"],bins=10, kde=True, color='blue')
```

Out[21]: <Axes: xlabel=' cibil_score', ylabel='Count'>



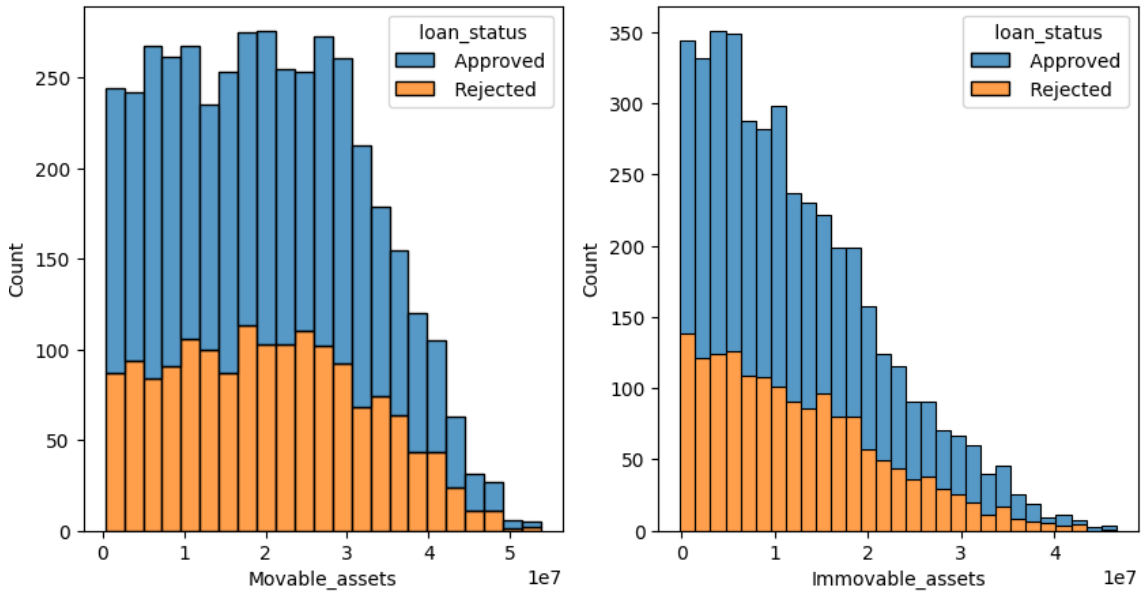
In [22]:
```
fig, ax = plt.subplots(1,2,figsize=(10,5))
plt.subplot(1, 2, 1)
sns.histplot(df['Movable_assets'], ax=ax[0], color='red')
plt.title("Movable_assets ")

plt.subplot(1, 2, 2)
plt.grid()
sns.histplot(df['Immovable_assets'], ax=ax[1], color='blue')
plt.title("Immovable_assets ")
```

Out[22]: Text(0.5, 1.0, 'Immovable_assets ')



In [23]:
```
fig, ax = plt.subplots(1,2,figsize=(10,5))
sns.histplot(x = 'Movable_assets', data = df, ax=ax[0], hue = ' loan_status', m
sns.histplot(x = 'Immovable_assets', data = df, ax=ax[1], hue = ' loan_status'
```

Out[23]:
```
<Axes: xlabel='Immovable_assets', ylabel='Count'>
```



In [24]:
```
df[' education'] = df[' education'].map({' Not Graduate':0, ' Graduate':1})
df[' self_employed'] = df[' self_employed'].map({' No':0, ' Yes':1})
df[' loan_status'] = df[' loan_status'].map({' Rejected':0, ' Approved':1})
```

In [25]:
```
pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\parag\appdata\local\programs
\python\python310\lib\site-packages (0.0.post5)
Note: you may need to restart the kernel to use updated packages.
```

In [26]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.compose import ColumnTransformer
from sklearn import tree
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop(' loan_status', axis
```

In [28]:
```
from sklearn.linear_model import LogisticRegression

lgr = LogisticRegression()

lgr.fit(X_train, y_train)

predictions = lgr.predict(X_test)
```

In [29]:
```
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6276346604215457
```

In [30]:
```
from sklearn.svm import SVC
model = SVC()
```

```
model.fit(X_train, y_train)

predictions = model.predict(X_test)
```

In [31]:
```
from sklearn.ensemble import RandomForestClassifier

# Create a random forest classifier
rfc = RandomForestClassifier()
```

In [32]:
```
rfc.fit(X_train, y_train)
```

Out[32]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [33]:
```
rfc_pred = rfc.predict(X_test)
```

In [34]:
```
accuracy = accuracy_score(y_test, rfc_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9800936768149883
```

In [37]:
```
from sklearn.tree import DecisionTreeClassifier

# Create decision tree object
dtree = DecisionTreeClassifier()
```

In [38]:
```
dtree.fit(X_train, y_train)
```

Out[38]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [39]:
```
dtree_pred = dtree.predict(X_test)
```

In [40]:
```
dtree.score(X_train, y_train)
```

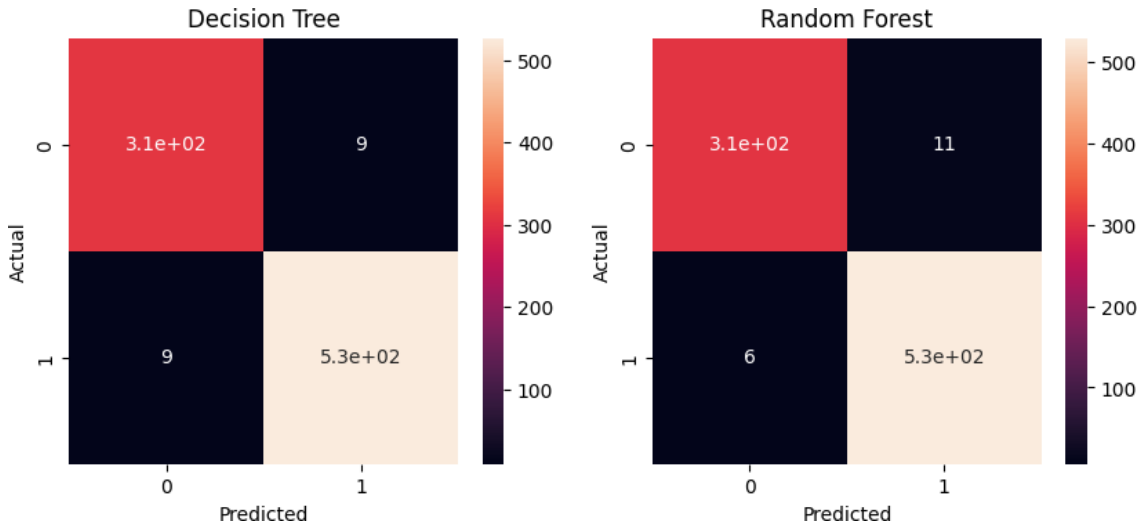
Out[40]:
```
1.0
```

In [41]:
```
accuracy = accuracy_score(y_test, dtree_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9789227166276346
```

In [42]:
```
from sklearn.metrics import confusion_matrix

fig, ax = plt.subplots(1,2,figsize=(10,4))
sns.heatmap(confusion_matrix(y_test, dtree_pred), annot=True, ax=ax[0]).set_titl
ax[0].set_xlabel('Predicted')
ax[0].set_ylabel('Actual')
sns.heatmap(confusion_matrix(y_test, rfc_pred), annot=True, ax=ax[1]).set_title(
ax[1].set_xlabel('Predicted')
ax[1].set_ylabel('Actual')
```

Out[42]:
```
Text(518.4494949494949, 0.5, 'Actual')
```


Decision Tree
3.1e+02
9
9
5.3e+02
Actual
Predicted
0
1
500
400
300
200
100
Random Forest
3.1e+02
11
6
5.3e+02
Actual
Predicted
0
1
500
400
300
200
100

In [ ]:

In [ ]:

7.0. Actual Resources used:

S. No.	Name of Resource/material	Specifications	Qty	Remarks
1	Computer System	Windows OS,i3 processor ,8 GB RAM	01	-
2	Software	Jupyter Notebook, MS Office word	01	-
3	Printer, Pages	Canon LaserJet	01	-

8.0. Skill Developed / Learning outcome from this Mini-Project:

1. Understanding the concept of random forest classifier.
2. Improved Data Analysis Skills.
3. Improved Data Preprocessing Skills.
4. Understanding the concept of Data splitting.
5. Understanding the concept of Feature Selection and Feature Engineering.

9.0. Applications of Mini Project:

1. **Auto Financing:**

   Machine Learning models can assist in determining whether an applicant is eligible for an auto loan based on factors such as credit score, income, and vehicle value.

2. **Real Estate:**

   Machine Learning models can help assess the creditworthiness of real estate investors applying for investment loans.

3. **Student Loans:**

   Machine Learning can be used by educational institutions and lenders to assess student loan eligibility based on factors like educational history and future earning potential.

4. **Small Business Loans:**

   Machine Learning models can evaluate the creditworthiness of small business owners applying for loans to support their enterprises.