

## LP3 – MACHINE LEARNING

Vaibhav Dhande BE 1 405A002

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.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

In [2]:

```
df = pd.read_csv('https://raw.githubusercontent.com/sahil-gidwani/ML/main/data
df.info()
```

```
<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
```

### 1. Pre-process the dataset.

In [3]:

```
df.shape
```

Out[3]:

```
(200000, 9)
```

In [4]: df.head()

Out[4]:

		Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194		2015-05-07 19:52:06.00000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.73835
1	27835199		2009-07-17 20:04:56.00000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.72822
2	44984355		2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.74077
3	25894730		2009-06-26 08:22:21.00000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.79084
4	17610152		2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.74408

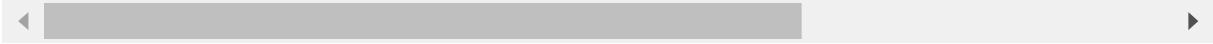


In [5]: df.isnull()

Out[5]:

		Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	drop
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
...		...	...	...	...	...	...	...
199995		False	False	False	False	False	False	False
199996		False	False	False	False	False	False	False
199997		False	False	False	False	False	False	False
199998		False	False	False	False	False	False	False
199999		False	False	False	False	False	False	False

200000 rows × 9 columns



In [6]: `df.drop(columns=["Unnamed: 0", "key"], inplace=True)  
df.head()`

Out[6]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.7
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.7
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.7
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.8
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.7

◀ ▶

In [7]: `df.isnull().sum()`

Out[7]:

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

In [8]: `df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(), inplace = True)  
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(), inplace = True)`

In [9]: `df.dtypes`

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

In [10]: `# From the above output, we see that the data type of 'pickup_datetime' is 'object'  
# But 'pickup_datetime' is a date time stamp variable, which is wrongly interpreted`

◀ ▶

```
In [11]: df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
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
```

```
In [12]: # we will extract time feature from the 'pickup_datetime'
# we will add a variable which measures the distance between pickup and drop
```

```
In [13]: 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 [14]: df
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	drop
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	
...	...	...	...	...	...	...
199995	3.0	2012-10-28 10:49:00+00:00	-73.987042	40.739367	-73.986525	
199996	7.5	2014-03-14 01:09:00+00:00	-73.984722	40.736837	-74.006672	
199997	30.9	2009-06-29 00:42:00+00:00	-73.986017	40.756487	-73.858957	
199998	14.5	2015-05-20 14:56:25+00:00	-73.997124	40.725452	-73.983215	
199999	14.1	2010-05-15 04:08:00+00:00	-73.984395	40.720077	-73.985508	

200000 rows × 12 columns



In [15]: `df = df.drop(["pickup_datetime"], axis =1)`  
`df`

Out[15]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pass
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	
...	...	...	...	...	...	...
199995	3.0	-73.987042	40.739367	-73.986525	40.740297	
199996	7.5	-73.984722	40.736837	-74.006672	40.739620	
199997	30.9	-73.986017	40.756487	-73.858957	40.692588	
199998	14.5	-73.997124	40.725452	-73.983215	40.695415	
199999	14.1	-73.984395	40.720077	-73.985508	40.768793	

200000 rows × 11 columns

In [16]: `# function to calculate the travel distance from the longitudes and latitudes`  
`from math import *`

```

def distance_formula(longitude1, latitude1, longitude2, latitude2):
    travel_dist = []

    for pos in range (len(longitude1)):
        lon1, lan1, lon2, lan2 = map(radians, [longitude1[pos], latitude1[pos],
                                                longitude2[pos], latitude2[pos]])
        dist_lon = lon2 - lon1
        dist_lan = lan2 - lan1

        a = sin(dist_lan/2)**2 + cos(lan1) * cos(lan2) * sin(dist_lon/2)**2

        #radius of earth = 6371
        c = 2 * asin(sqrt(a)) * 6371
        travel_dist.append(c)

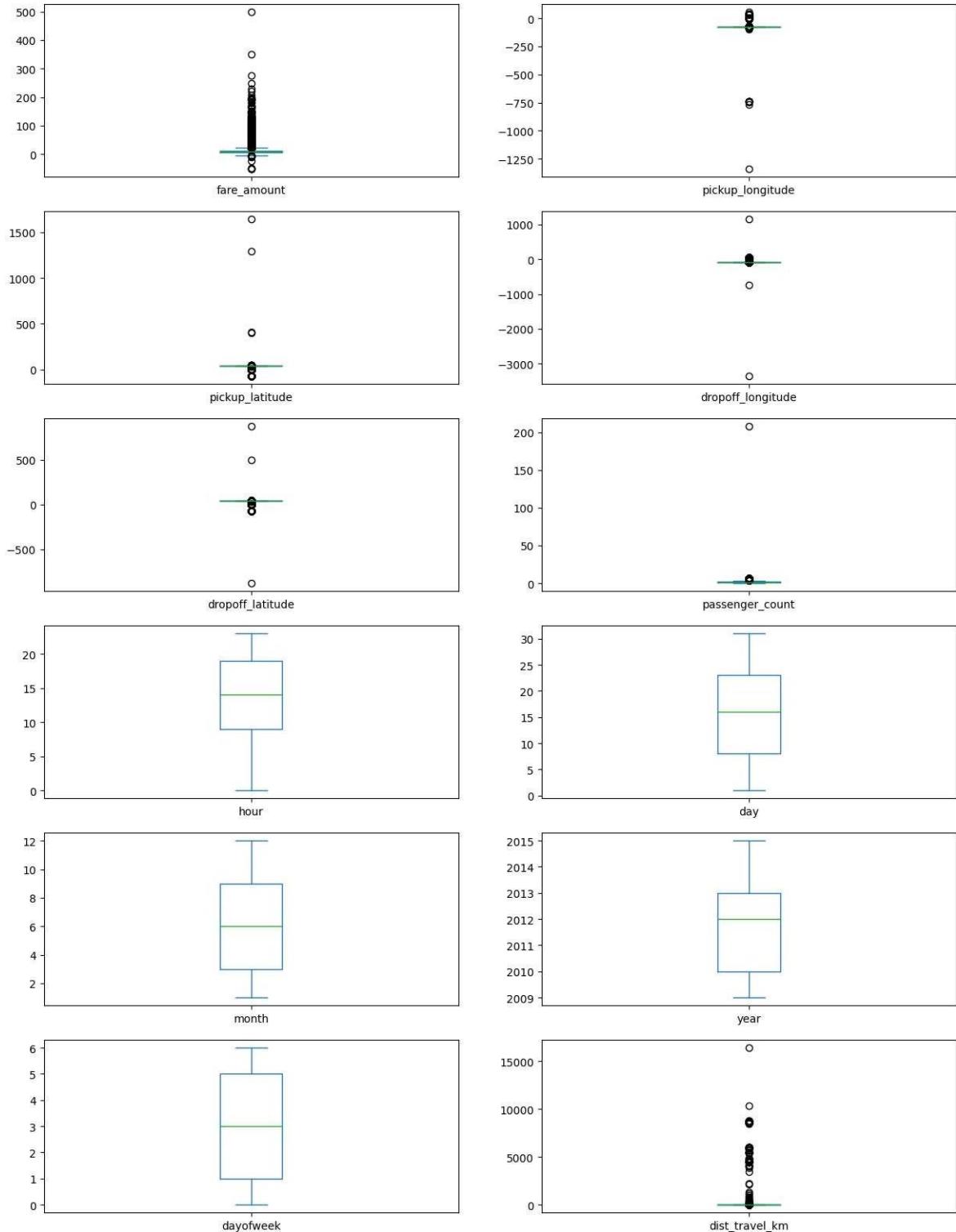
    return travel_dist

```

In [17]: `df['dist_travel_km'] = distance_formula(df.pickup_longitude.to_numpy(), df.pic`

## 2. Identify outliers.

```
In [18]: df.plot(kind = "box", subplots = True, layout = (6,2), figsize=(15,20)) #Boxplot  
plt.show()
```



In [19]: #Using the InterQuartile Range to fill the values

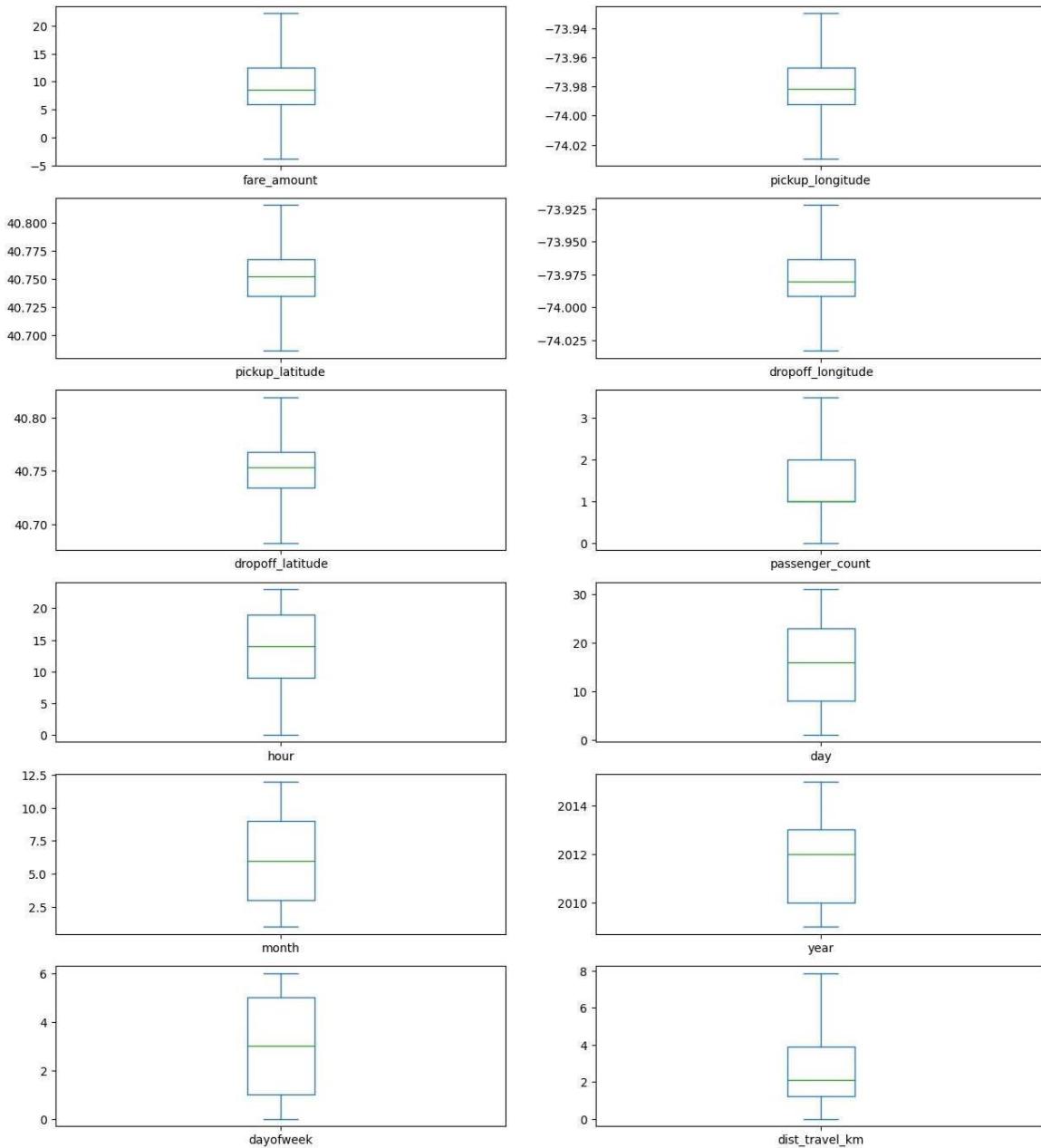
```
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

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

In [20]: df = treat\_outliers\_all(df , df.iloc[:, 0::])

In [21]: *#Boxplot shows that dataset is free from outliers*

```
df.plot(kind = "box", subplots = True, layout = (7,2), figsize=(15,20))  
plt.show()
```



### 3. Check the correlation.

In [22]: *#Function to find the correlation*  
corr = df.corr()  
corr

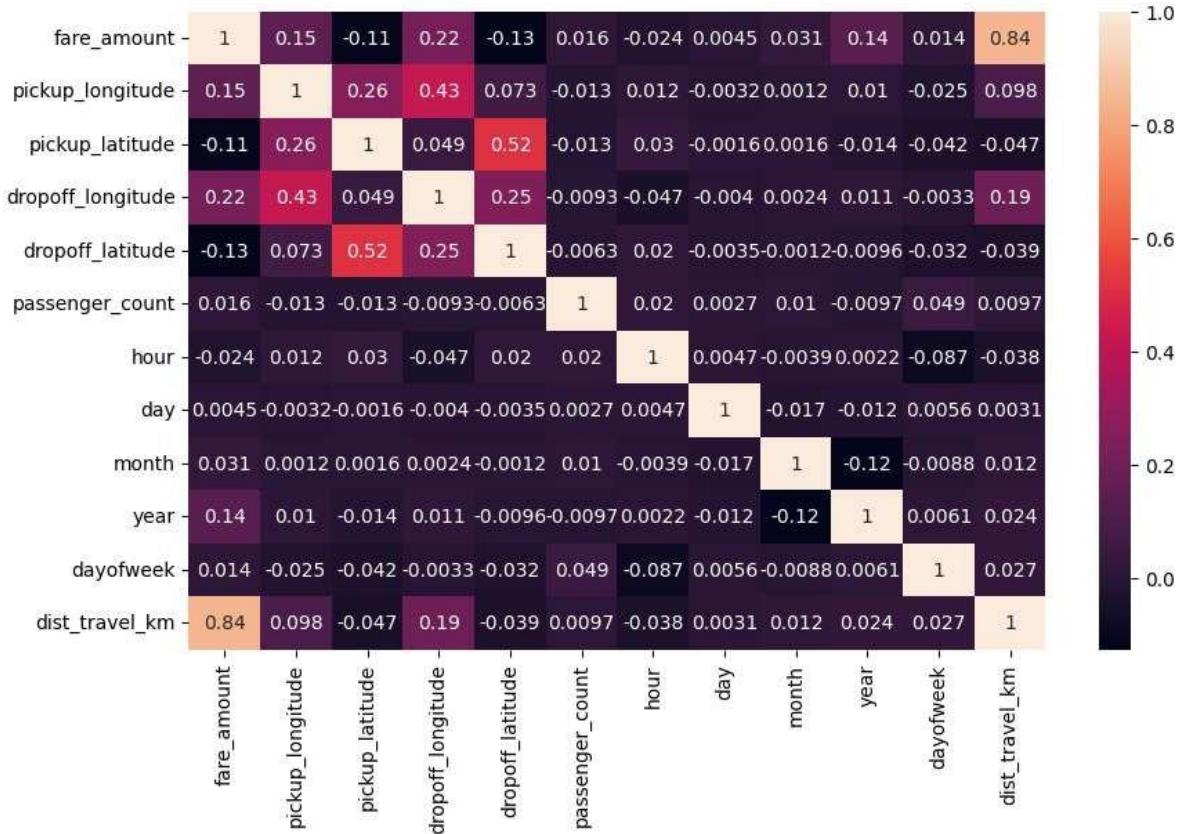
Out[22]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
fare_amount	1.000000	0.154069	-0.110842	0.218675	-0.125
pickup_longitude	0.154069	1.000000	0.259497	0.425619	0.073
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	0.515
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	0.245
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	1.000
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	-0.006
hour	-0.023623	0.011579	0.029681	-0.046558	0.019
day	0.004534	-0.003204	-0.001553	-0.004007	-0.003
month	0.030817	0.001169	0.001562	0.002391	-0.001
year	0.141277	0.010198	-0.014243	0.011346	-0.009
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	-0.031
dist_travel_km	0.844374	0.098094	-0.046812	0.186531	-0.038

In [23]:

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

Out[23]: <Axes: >



## 4. Implement linear regression and random forest regression models.

In [24]:

```
# Dividing the dataset into feature and target values
df_x = df[['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'passenger_count']]
df_y = df['fare_amount']
```

In [25]:

```
# Dividing the dataset into training and testing dataset
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2,
```

In [26]: df

Out[26]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pass
0	7.50	-73.999817	40.738354	-73.999512	40.723217	
1	7.70	-73.994355	40.728225	-73.994710	40.750325	
2	12.90	-74.005043	40.740770	-73.962565	40.772647	
3	5.30	-73.976124	40.790844	-73.965316	40.803349	
4	16.00	-73.929786	40.744085	-73.973082	40.761247	
...	...	...	...	...	...	...
199995	3.00	-73.987042	40.739367	-73.986525	40.740297	
199996	7.50	-73.984722	40.736837	-74.006672	40.739620	
199997	22.25	-73.986017	40.756487	-73.922036	40.692588	
199998	14.50	-73.997124	40.725452	-73.983215	40.695415	
199999	14.10	-73.984395	40.720077	-73.985508	40.768793	

200000 rows × 12 columns



In [27]: from sklearn.linear\_model import LinearRegression

```
# initialize the Linear regression model
reg = LinearRegression()

# Train the model with our training data
reg.fit(x_train, y_train)
```

Out[27]: LinearRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [28]: y\_pred\_lin = reg.predict(x\_test)
print(y\_pred\_lin)

```
[ 6.27615184  5.09986098  9.43641238 ... 11.07663949 12.15392248
 11.41496075]
```

```
In [29]: from sklearn.ensemble import RandomForestRegressor

#Here n_estimators means number of trees you want to build before making the p
rf = RandomForestRegressor(n_estimators=100)
rf.fit(x_train,y_train)
```

Out[29]: RandomForestRegressor()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [30]: y_pred_rf = rf.predict(x_test)
print(y_pred_rf)
```

[ 5.105 6.561 9.5875 ... 11.415 11.6615 13.659 ]

## 5. Evaluate the models and compare their respective scores like R2, RMSE, etc

```
In [31]: cols = ['Model', 'RMSE', 'R-Squared']

# create a empty dataframe of the columns
# columns: specifies the columns to be selected
result_tabulation = pd.DataFrame(columns = cols)
```

```
In [32]: from sklearn import metrics
from sklearn.metrics import r2_score

reg_RMSE = np.sqrt(metrics.mean_squared_error(y_test, y_pred_lin))
reg_squared = r2_score(y_test, y_pred_lin)

full_metrics = pd.Series({'Model': "Linear Regression", 'RMSE' : reg_RMSE, 'R-'

# append our result table using append()
# ignore_index=True: does not use the index Labels
# python can only append a Series if ignore_index=True or if the Series has a
result_tabulation = result_tabulation.append(full_metrics, ignore_index = True

# print the result table
result_tabulation
```

<ipython-input-32-3a51173ca998>:12: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

result\_tabulation = result\_tabulation.append(full\_metrics, ignore\_index = True)

Out[32]:

	Model	RMSE	R-Squared
0	Linear Regression	2.703957	0.753906

```
In [33]: rf_RMSE = np.sqrt(metrics.mean_squared_error(y_test, y_pred_rf))
rf_squared = r2_score(y_test, y_pred_rf)

full_metrics = pd.Series({'Model': "Random Forest ", 'RMSE':rf_RMSE, 'R-Square':rf_squared})
# append our result table using append()
# ignore_index=True: does not use the index Labels
# python can only append a Series if ignore_index=True or if the Series has a
result_tabulation = result_tabulation.append(full_metrics, ignore_index = True)

# print the result table
result_tabulation
```

<ipython-input-33-37c52c9744f2>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

```
result_tabulation = result_tabulation.append(full_metrics, ignore_index = True)
```

Out[33]:

	Model	RMSE	R-Squared
0	Linear Regression	2.703957	0.753906
1	Random Forest	2.363103	0.812040

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> (<https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv>)

```
In [15]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

```
In [16]: df = pd.read_csv('emails.csv')
df
```

Out[16]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastru
0	Email 1	0	0	1	0	0	0	2	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
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5167	Email 5168	2	2	2	3	0	0	32	0	0	...	0	0	0	0	0
5168	Email 5169	35	27	11	2	6	5	151	4	3	...	0	0	0	0	0
5169	Email 5170	0	0	1	1	0	0	11	0	0	...	0	0	0	0	0
5170	Email 5171	2	7	1	0	2	1	28	2	0	...	0	0	0	0	0
5171	Email 5172	22	24	5	1	6	5	148	8	2	...	0	0	0	0	0

5172 rows × 3002 columns



In [17]: df.shape

Out[17]: (5172, 3002)

In [18]: df.isnull().any()

Out[18]: Email No. False  
the False  
to False  
ect False  
and False  
...  
military False  
allowing False  
ff False  
dry False  
Prediction False  
Length: 3002, dtype: bool

In [19]: df.drop(columns='Email No.', inplace=True)  
df

Out[19]:

	the	to	ect	and	for	of	a	you	hou	in	...	connevey	jay	valued	lay	infrastructu
0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	0	0
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	0	0
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	0	0
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5167	2	2	2	3	0	0	32	0	0	5	...	0	0	0	0	0
5168	35	27	11	2	6	5	151	4	3	23	...	0	0	0	0	0
5169	0	0	1	1	0	0	11	0	0	1	...	0	0	0	0	0
5170	2	7	1	0	2	1	28	2	0	8	...	0	0	0	0	0
5171	22	24	5	1	6	5	148	8	2	23	...	0	0	0	0	0

5172 rows × 3001 columns

In [20]: df.columns

Out[20]: Index(['the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou', 'in',  
...  
'connevey', 'jay', 'valued', 'lay', 'infrastructure', 'military',  
'allowing', 'ff', 'dry', 'Prediction'],  
dtype='object', length=3001)

In [21]: `df.Prediction.unique()`

Out[21]: `array([0, 1])`

In [22]: `df['Prediction'] = df['Prediction'].replace({0:'Not spam', 1:'Spam'})`

In [23]: `df`

Out[23]:

	the	to	ect	and	for	of	a	you	hou	in	...	connevey	jay	valued	lay	infrastructu
0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	0	0
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	0	0
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	0	0
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5167	2	2	2	3	0	0	32	0	0	5	...	0	0	0	0	0
5168	35	27	11	2	6	5	151	4	3	23	...	0	0	0	0	0
5169	0	0	1	1	0	0	11	0	0	1	...	0	0	0	0	0
5170	2	7	1	0	2	1	28	2	0	8	...	0	0	0	0	0
5171	22	24	5	1	6	5	148	8	2	23	...	0	0	0	0	0

5172 rows × 3001 columns



## KNN

In [24]: `X = df.drop(columns='Prediction',axis = 1)`  
`Y = df['Prediction']`

In [25]: `X.columns`

Out[25]: `Index(['the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou', 'in', ... 'enhancements', 'connevey', 'jay', 'valued', 'lay', 'infrastructure', 'military', 'allowing', 'ff', 'dry'], dtype='object', length=3000)`

In [26]: Y.head()

```
Out[26]: 0    Not spam
1    Not spam
2    Not spam
3    Not spam
4    Not spam
Name: Prediction, dtype: object
```

In [27]: x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

```
In [28]: KN = KNeighborsClassifier
knn = KN(n_neighbors=7)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

```
In [29]: print("Prediction: \n")
print(y_pred)
```

Prediction:

```
['Not spam' 'Spam' 'Not spam' ... 'Not spam' 'Not spam' 'Not spam']
```

In [30]: # Accuracy

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

KNN accuracy: 0.8714975845410629

In [31]: C = metrics.confusion\_matrix(y\_test,y\_pred)
print("Confusion matrix: ", C)

```
Confusion matrix: [[635  84]
 [ 49 267]]
```

## SVM Classifier

```
In [32]: model = SVC(C = 1)    # cost C = 1

model.fit(x_train, y_train)

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

In [33]: kc = metrics.confusion\_matrix(y\_test, y\_pred)
print("SVM accuracy: ", kc)

```
SVM accuracy: [[700  19]
 [189 127]]
```

In [33]:

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import io
```

## Read the Dataset

```
In [2]: df=pd.read_csv("https://raw.githubusercontent.com/sahil-gidwani/ML/main/dataset.csv")
df.head()
```

Out[2]:

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

◀ ▶

## 2. Drop the Columns which are unique for all users

```
In [3]: df=df.drop(['RowNumber','CustomerId','Surname'],axis=1)
df.head()
```

Out[3]:

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

◀ ▶

```
In [4]: df.isna().any()
df.isna().sum()
```

```
Out[4]: CreditScore      0
Geography        0
Gender           0
Age              0
Tenure           0
Balance          0
NumOfProducts    0
HasCrCard        0
IsActiveMember   0
EstimatedSalary  0
Exited           0
dtype: int64
```

## Bivariate Analysis

```
In [5]: print(df.shape)
df.info()
```

```
(10000, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
 --- 
 0   CreditScore       10000 non-null   int64  
 1   Geography         10000 non-null   object  
 2   Gender            10000 non-null   object  
 3   Age               10000 non-null   int64  
 4   Tenure            10000 non-null   int64  
 5   Balance           10000 non-null   float64 
 6   NumOfProducts     10000 non-null   int64  
 7   HasCrCard         10000 non-null   int64  
 8   IsActiveMember    10000 non-null   int64  
 9   EstimatedSalary   10000 non-null   float64 
 10  Exited            10000 non-null   int64  
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

In [6]: df.describe()

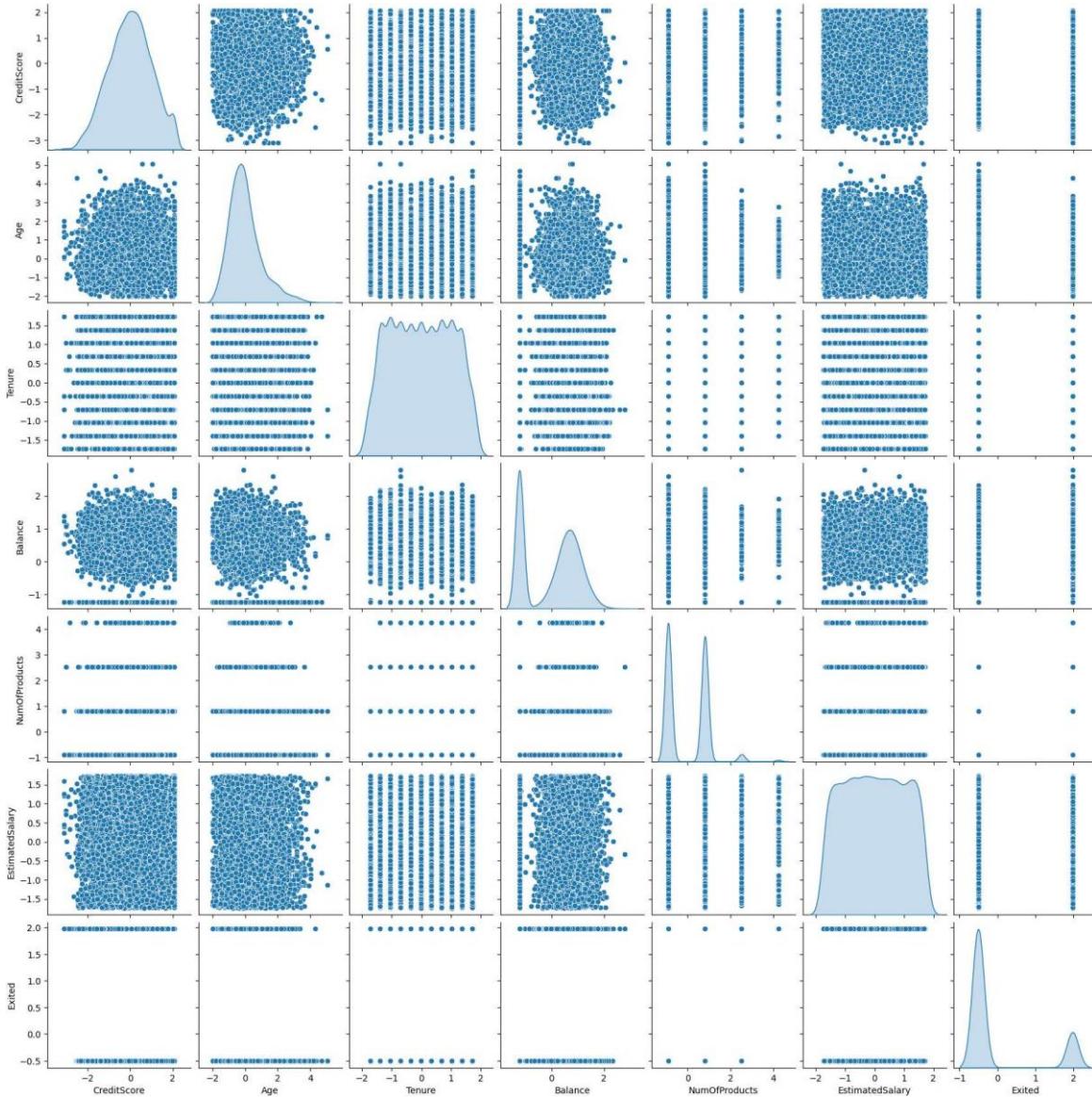
Out[6]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	EstimatedSalary
<b>count</b>	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
<b>mean</b>	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	10000.000000
<b>std</b>	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	10000.000000
<b>min</b>	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	10000.000000
<b>25%</b>	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	10000.000000
<b>50%</b>	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	10000.000000
<b>75%</b>	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	10000.000000
<b>max</b>	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	10000.000000

Before performing Bivariate analysis, Lets bring all the features to the same range

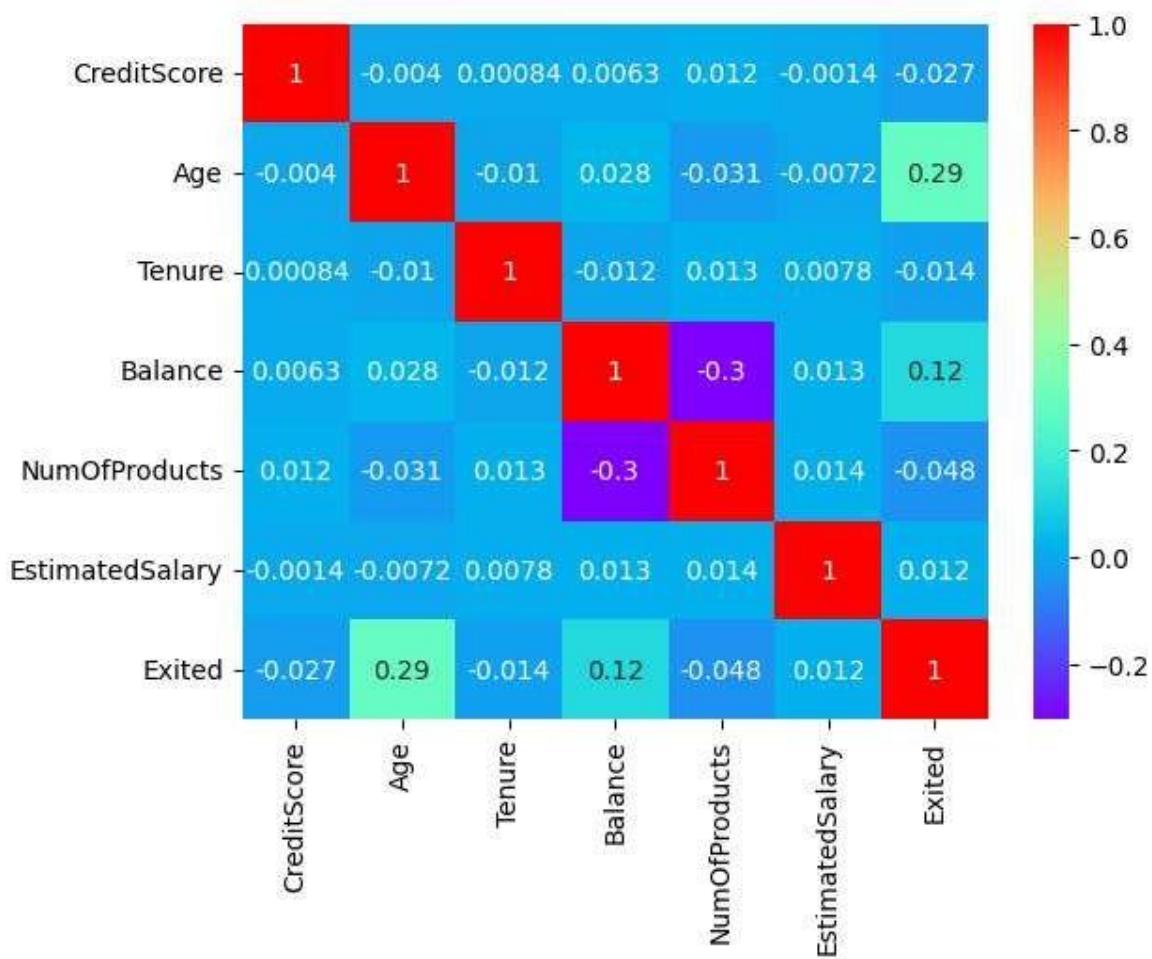
```
In [7]: ## Scale the data
scaler=StandardScaler()
## Extract only the Numerical Columns to perform Bivariate Analysis
subset=df.drop(['Geography','Gender','HasCrCard','IsActiveMember'],axis=1)
scaled=scaler.fit_transform(subset)
scaled_df=pd.DataFrame(scaled,columns=subset.columns)
sns.pairplot(scaled_df,diag_kind='kde')
```

Out[7]: <seaborn.axisgrid.PairGrid at 0x7828c8f03f40>



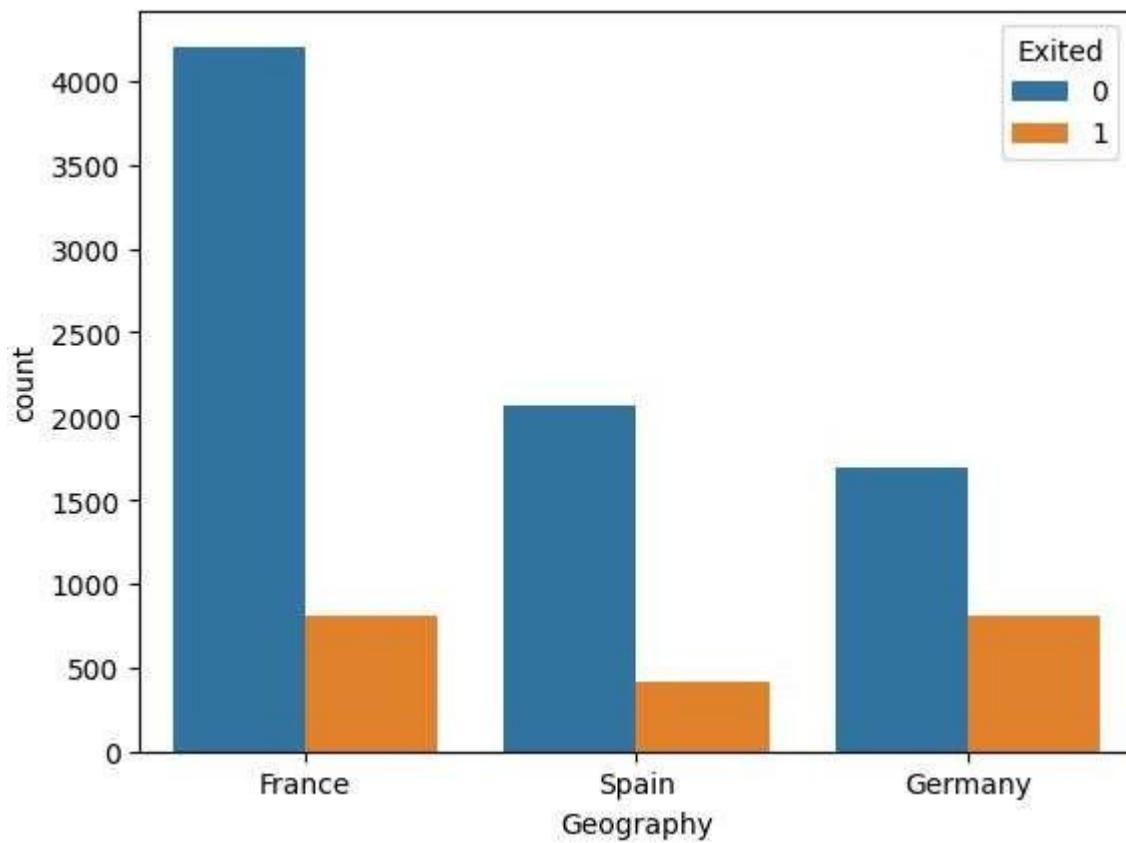
```
In [8]: sns.heatmap(scaled_df.corr(), annot=True, cmap='rainbow')
```

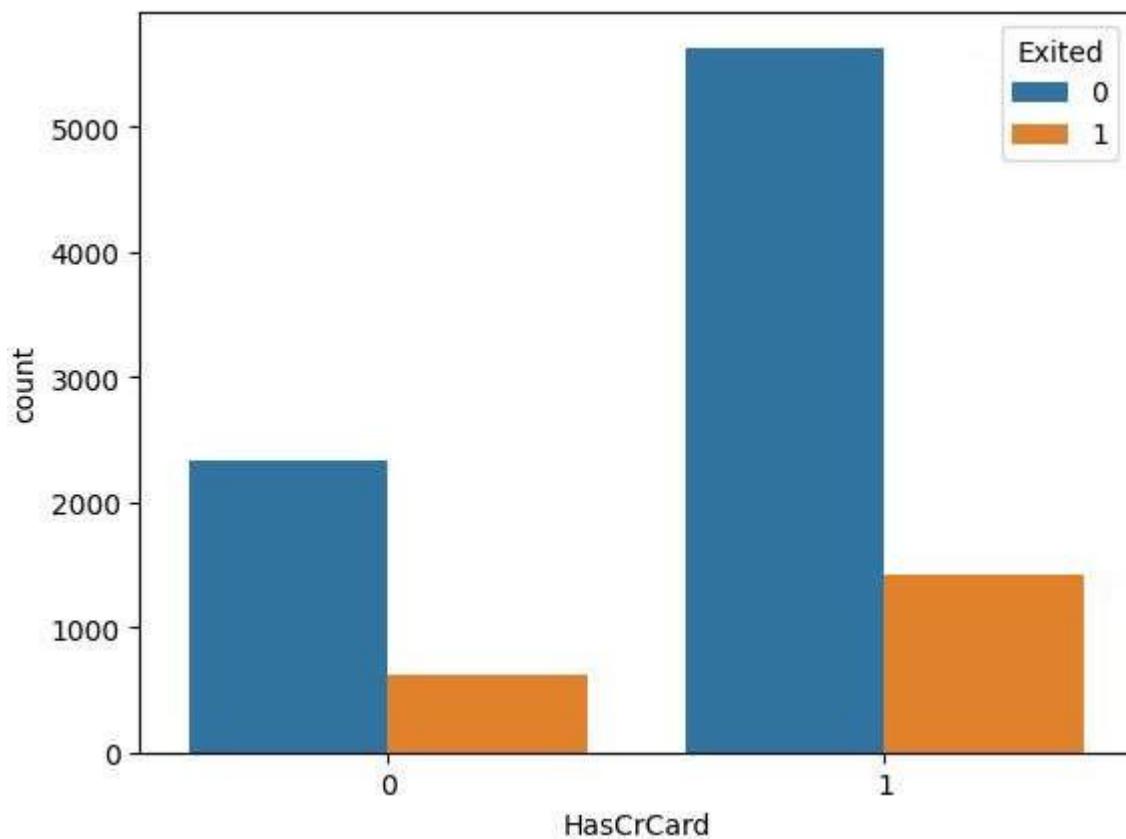
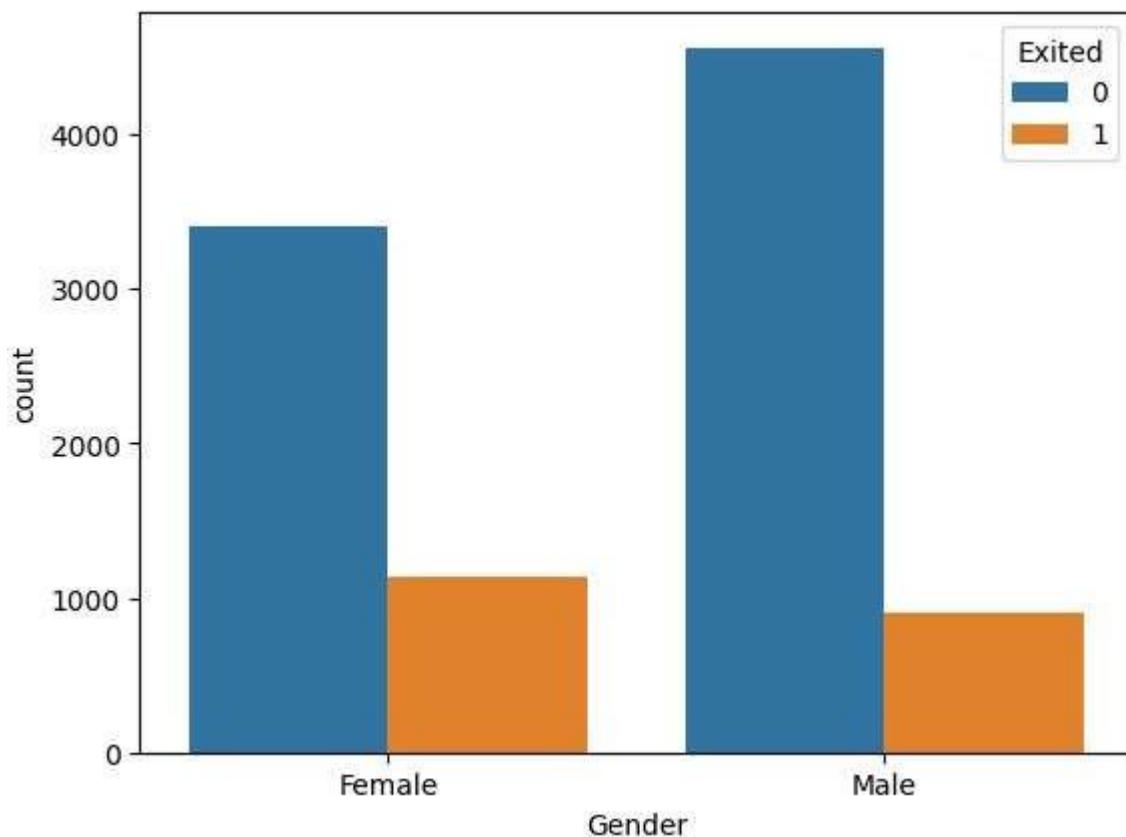
Out[8]: <Axes: >

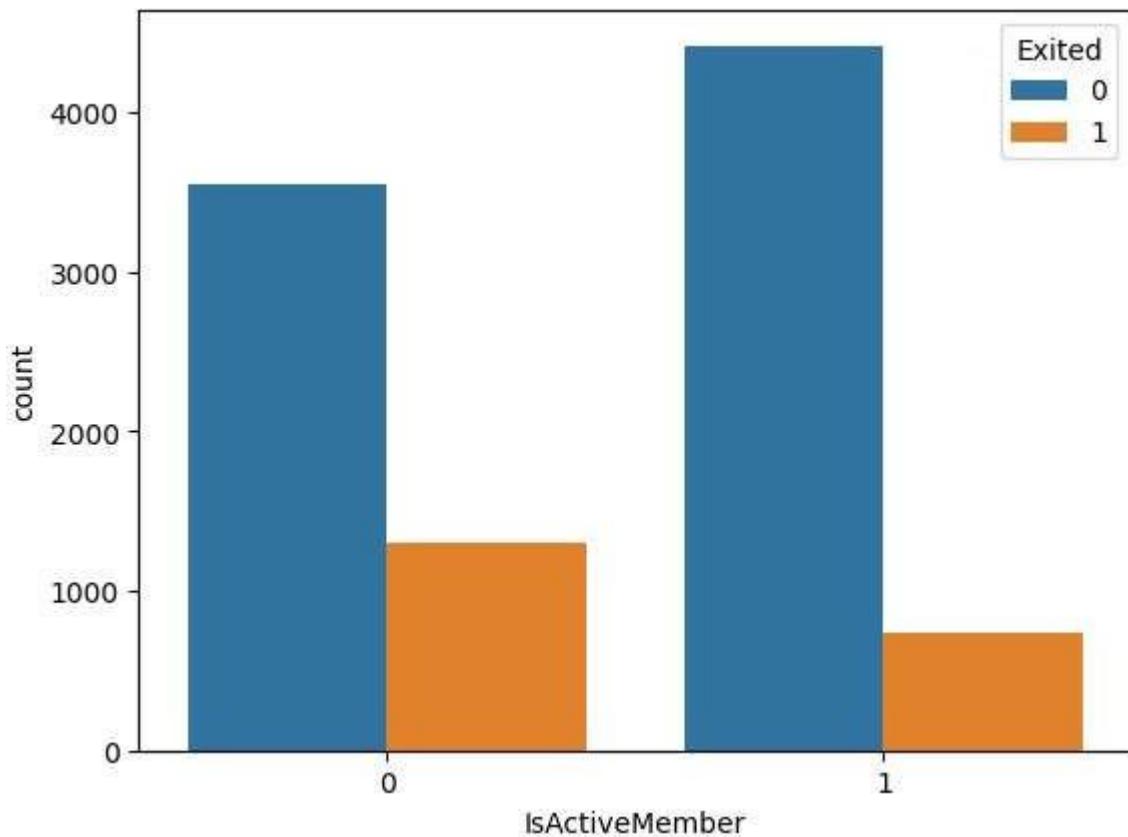


**From the above plots, We can see that there is no significant Linear relationship between the features**

```
In [9]: ## Categorical Features vs Target Variable  
sns.countplot(x='Geography',data=df,hue='Exited')  
plt.show()  
sns.countplot(x='Gender',data=df,hue='Exited')  
plt.show()  
sns.countplot(x='HasCrCard',data=df,hue='Exited')  
plt.show()  
sns.countplot(x='IsActiveMember',data=df,hue='Exited')  
plt.show()
```

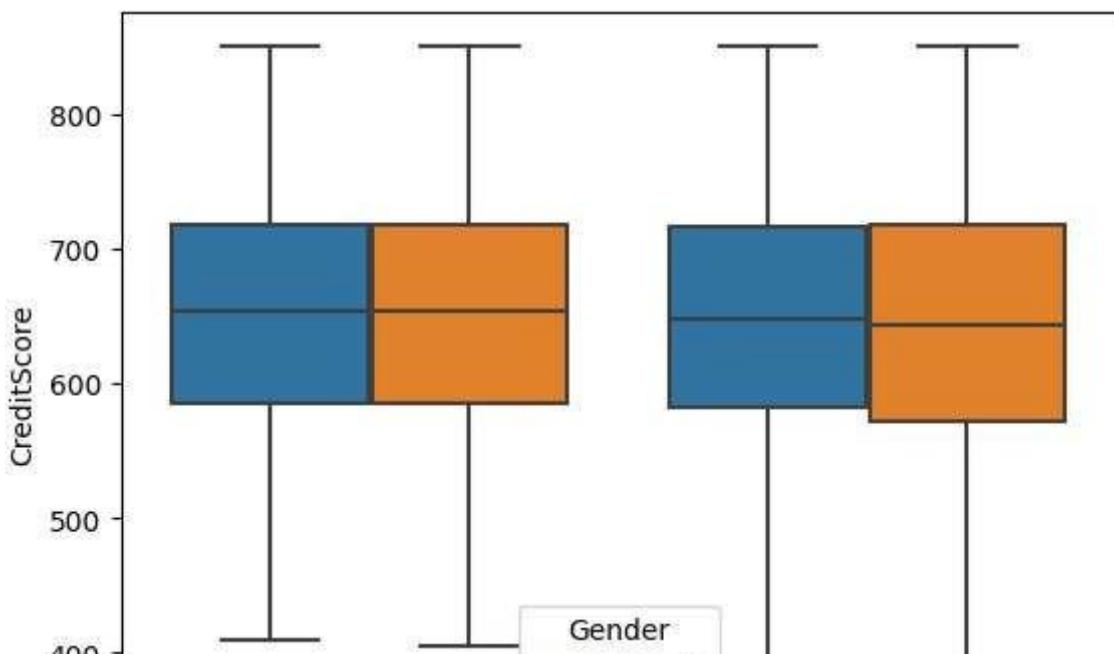






**Analysing the Numerical Features relationship with the Target variable. Here 'Exited' is the Target Feature.**

```
In [10]: subset = subset.drop('Exited', axis=1)
for i in subset.columns:
    sns.boxplot(x=df['Exited'], y=df[i], hue=df['Gender'])
    plt.show()
```



## Insights from Bivariate Plots

1. The Avg Credit Score seem to be almost the same for Active and Churned customers
2. Young People seem to stick to the bank compared to older people
3. The Average Bank Balance is high for Churned Customers
4. The churning rate is high with German Customers
5. The Churning rate is high among the Non-Active Members

## 4. Distinguish the Target and Feature Set and divide the dataset into Training and Test sets

```
In [11]: X=df.drop('Exited',axis=1)  
y=df.pop('Exited')
```

```
In [12]: from sklearn.model_selection import train_test_split  
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.10,random_state  
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.10,ra  
print("X_train size is {}".format(X_train.shape[0]))  
print("X_val size is {}".format(X_val.shape[0]))  
print("X_test size is {}".format(X_test.shape[0]))
```

```
X_train size is 8100  
X_val size is 900  
X_test size is 1000
```

```
In [13]: ## Standardising the train, Val and Test data
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
num_cols=['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']
num_subset=scaler.fit_transform(X_train[num_cols])
X_train_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_train_num_df['Geography']=list(X_train['Geography'])
X_train_num_df['Gender']=list(X_train['Gender'])
X_train_num_df['HasCrCard']=list(X_train['HasCrCard'])
X_train_num_df['IsActiveMember']=list(X_train['IsActiveMember'])
X_train_num_df.head()
## Standardise the Validation data
num_subset=scaler.fit_transform(X_val[num_cols])
X_val_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_val_num_df['Geography']=list(X_val['Geography'])
X_val_num_df['Gender']=list(X_val['Gender'])
X_val_num_df['HasCrCard']=list(X_val['HasCrCard'])
X_val_num_df['IsActiveMember']=list(X_val['IsActiveMember'])
## Standardise the Test data
num_subset=scaler.fit_transform(X_test[num_cols])
X_test_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_test_num_df['Geography']=list(X_test['Geography'])
X_test_num_df['Gender']=list(X_test['Gender'])
X_test_num_df['HasCrCard']=list(X_test['HasCrCard'])
X_test_num_df['IsActiveMember']=list(X_test['IsActiveMember'])
```

```
In [14]: ## Convert the categorical features to numerical
X_train_num_df=pd.get_dummies(X_train_num_df,columns=['Geography','Gender'])
X_test_num_df=pd.get_dummies(X_test_num_df,columns=['Geography','Gender'])
X_val_num_df=pd.get_dummies(X_val_num_df,columns=['Geography','Gender'])
X_train_num_df.head()
```

Out[14]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	HasCrCard	IsActiveMember
0	-1.178587	-1.041960	-1.732257	0.198686	0.820905	1.560315	1	1
1	-0.380169	-1.326982	1.730718	-0.022020	-0.907991	-0.713592	1	1
2	-0.349062	1.808258	-0.693364	0.681178	0.820905	-1.126515	1	1
3	0.625629	2.378302	-0.347067	-1.229191	0.820905	-1.682740	1	1
4	-0.203895	-1.136967	1.730718	0.924256	-0.907991	1.332535	1	1

## Initialise and build the Model

```
In [15]: from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model=Sequential()
model.add(Dense(7,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
In [16]: import tensorflow as tf
optimizer=tf.keras.optimizers.Adam(0.01)
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=[ 'accuracy'])
```

```
In [17]: model.fit(X_train_num_df,y_train,epochs=100,batch_size=10,verbose=1)
```

```
Epoch 1/100
810/810 [=====] - 2s 2ms/step - loss: 0.3977 - accuracy: 0.8241
Epoch 2/100
810/810 [=====] - 2s 2ms/step - loss: 0.3599 - accuracy: 0.8552
Epoch 3/100
810/810 [=====] - 2s 2ms/step - loss: 0.3529 - accuracy: 0.8559
Epoch 4/100
810/810 [=====] - 2s 3ms/step - loss: 0.3527 - accuracy: 0.8559
Epoch 5/100
810/810 [=====] - 2s 3ms/step - loss: 0.3455 - accuracy: 0.8610
Epoch 6/100
810/810 [=====] - 2s 3ms/step - loss: 0.3469 - accuracy: 0.8577
Epoch 7/100
810/810 [=====] - 2s 3ms/step - loss: 0.3476 - accuracy: 0.8577
```

## Predict the Results using 0.5 threshold

```
In [18]: y_pred_val=model.predict(X_val_num_df)
y_pred_val[y_pred_val>0.5]=1
y_pred_val[y_pred_val <0.5]=0
```

```
29/29 [=====] - 0s 3ms/step
```

```
In [19]: y_pred_val=y_pred_val.tolist()
X_compare_val=X_val.copy()
X_compare_val['y_actual']=y_val
X_compare_val['y_pred']=y_pred_val
X_compare_val.head(10)
```

Out[19]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsFraud
340	642	Germany	Female	40	6	129502.49		2	0
8622	706	Germany	Male	36	9	58571.18		2	1
8401	535	Spain	Male	58	1	0.00		2	1
4338	714	Spain	Male	25	2	0.00		1	1
8915	606	France	Male	36	1	155655.46		1	1
2624	605	Spain	Female	29	3	116805.82		1	0
2234	720	France	Female	38	10	0.00		2	1
349	582	France	Male	39	5	0.00		2	1
3719	850	France	Female	62	1	124678.35		1	1
2171	526	Germany	Male	58	9	190298.89		2	1

## Confusion Matrix of the Validation set

```
In [20]: from sklearn.metrics import confusion_matrix
cm_val=confusion_matrix(y_val,y_pred_val)
cm_val
```

```
Out[20]: array([[689,  27],
 [101,  83]])
```

```
In [26]: # Extract TP, TN, FP, FN from the confusion matrix
TP = cm_val[1, 1] # True Positives
TN = cm_val[0, 0] # True Negatives
FP = cm_val[0, 1] # False Positives
FN = cm_val[1, 0] # False Negatives

# Calculate accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)

print("Accuracy:", accuracy)
```

Accuracy: 0.8577777777777778

```
In [22]: loss1,accuracy1=model.evaluate(X_train_num_df,y_train,verbose=False)
loss2,accuracy2=model.evaluate(X_val_num_df,y_val,verbose=False)
print("Train Loss {}".format(loss1))
print("Train Accuracy {}".format(accuracy1))
print("Val Loss {}".format(loss2))
print("Val Accuracy {}".format(accuracy2))
```

```
Train Loss 0.32568806409835815
Train Accuracy 0.8662962913513184
Val Loss 0.35947299003601074
Val Accuracy 0.857777743339539
```

**Since our Training Accuracy and Validation Accuracy are pretty close, we can conclude that our model generalises well. So, lets apply the model on the Test set and make predictions and evaluate the model against the Test.**

```
In [23]: from sklearn import metrics
y_pred_test=model.predict(X_test_num_df)
y_pred_test[y_pred_test>0.5]=1
y_pred_test[y_pred_test <0.5]=0
cm_test=metrics.confusion_matrix(y_test,y_pred_test)
cm_test
print("Test Confusion Matrix")
```

```
32/32 [=====] - 0s 2ms/step
Test Confusion Matrix
```

```
In [24]: cm_test
```

```
Out[24]: array([[755,  39],
 [119,  87]])
```

```
In [25]: loss3,accuracy3=model.evaluate(X_test_num_df,y_test,verbose=False)
print("Test Accuracy is {}".format(accuracy3))
print("Test loss is {}".format(loss3))
```

```
Test Accuracy is 0.8420000076293945
Test loss is 0.3749040365219116
```

```
In [25]:
```



```
In [2]: import numpy as np
import matplotlib.pyplot as plt

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

def df(x):
    return 2 * (x + 3)

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

initial_x = 0
learning_rate = 0.1
num_iterations = 50

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

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

# Create a range of x values to plot
x_vals = np.linspace(-10, 4, 100)

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

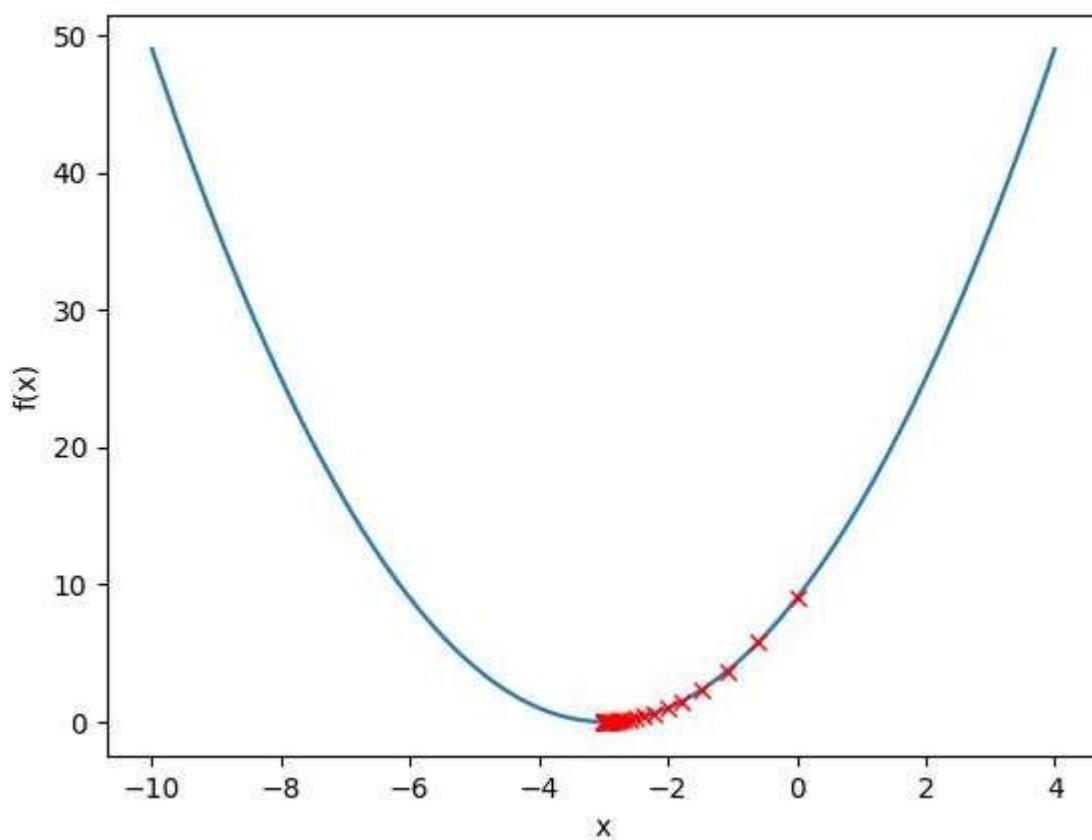
# Plot the values of x at each iteration
plt.plot(x_history, f(np.array(x_history)), 'rx')

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

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

Local minimum: -3.00

## Gradient Descent



In [ ]:

# Diabetes classification using KNN

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

```
In [2]: data = pd.read_csv("https://raw.githubusercontent.com/sahil-gidwani/ML/main/datasets/diabetes.csv")
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
In [3]: data.isnull().any()
```

Out[3]:

Pregnancies	False
Glucose	False
BloodPressure	False
SkinThickness	False
Insulin	False
BMI	False
Pedigree	False
Age	False
Outcome	False
dtype: bool	

```
In [4]: data.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
Pedigree	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

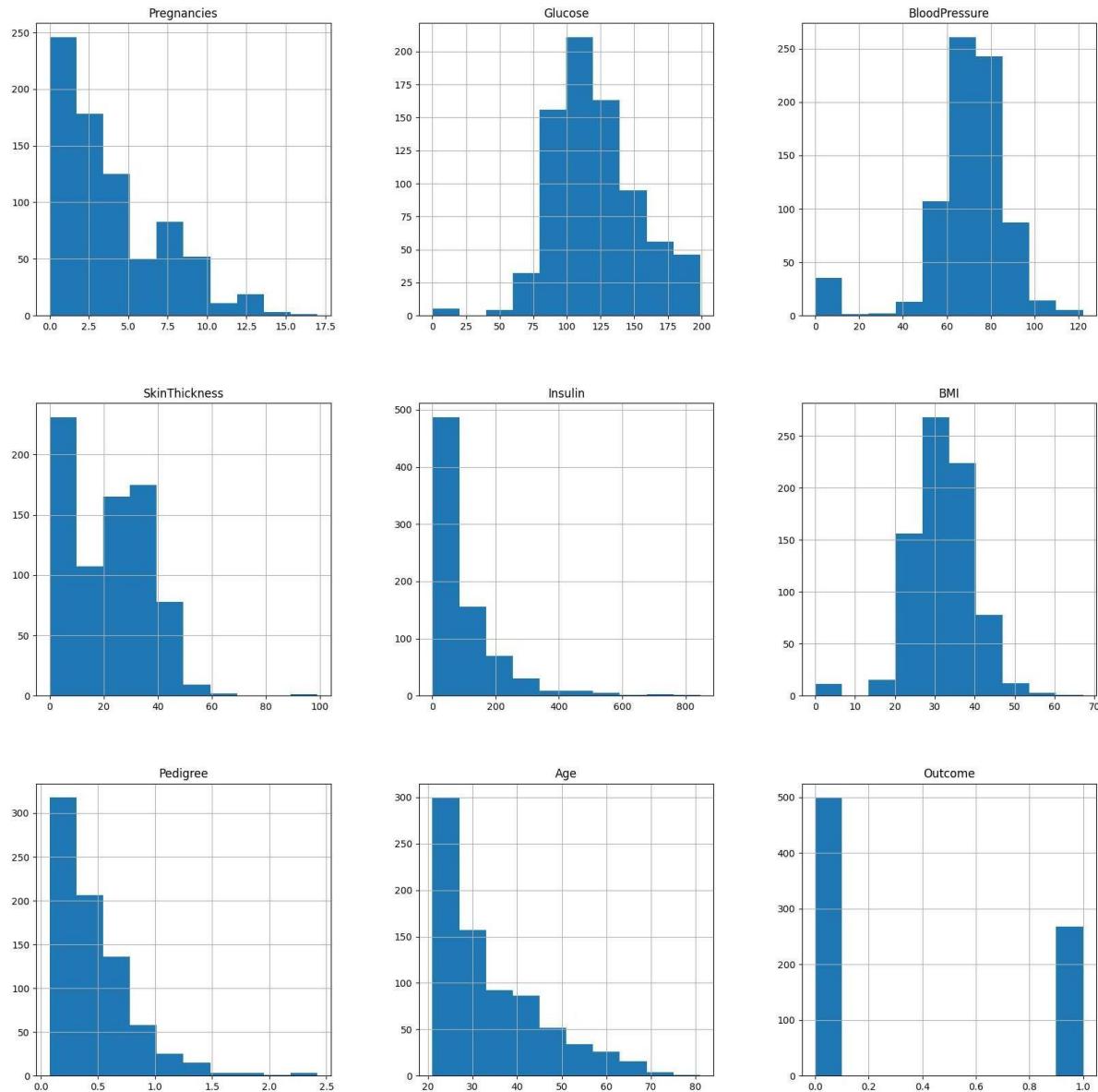
Glucose, BloodPressure, SkinThickness, Insulin, BMI  
columns have values 0 which does not make sense , hence are missing values

```
In [5]: data_copy = data.copy(deep = True)
data_copy[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = data_
data_copy.isnull().sum()
```

```
Out[5]: Pregnancies      0
Glucose          5
BloodPressure    35
SkinThickness   227
Insulin         374
BMI            11
Pedigree        0
Age             0
Outcome         0
dtype: int64
```

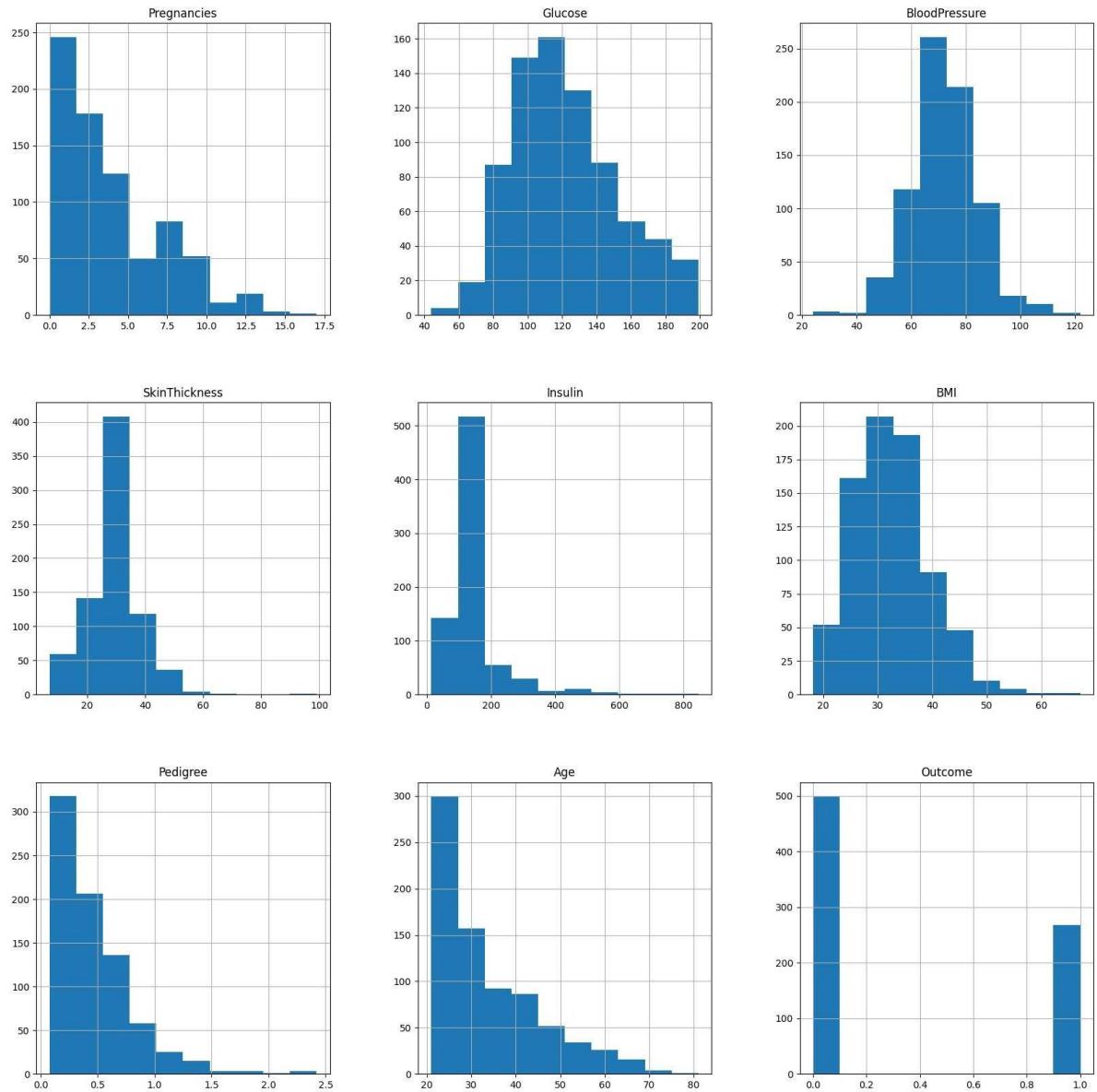
To fill these Nan values the data distribution needs to be understood

In [6]: `p = data.hist(figsize = (20,20))`

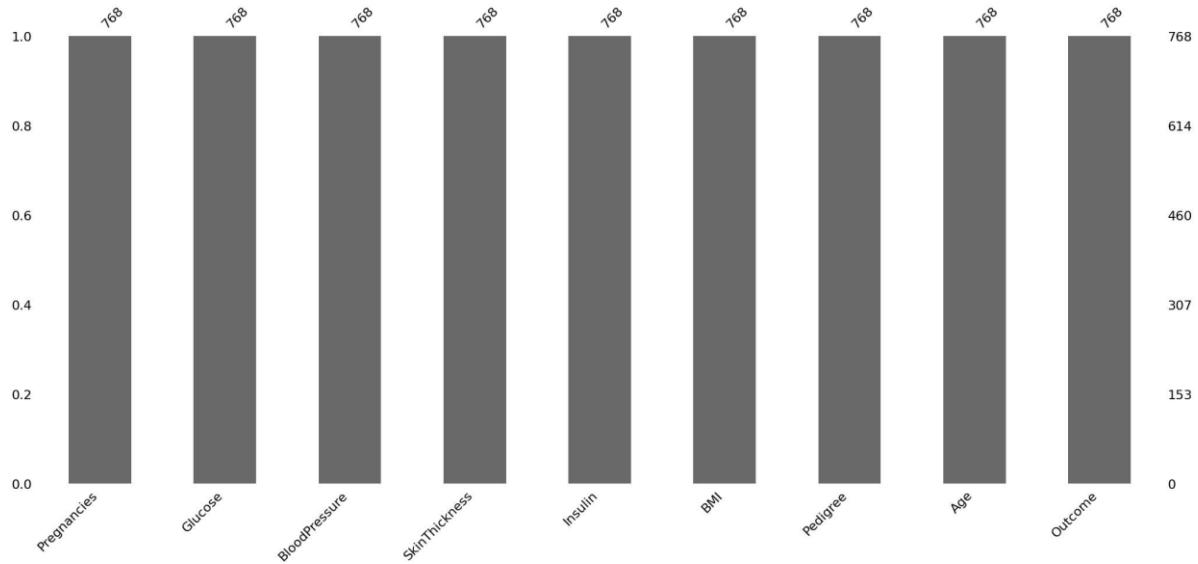


In [7]: `data_copy['Glucose'].fillna(data_copy['Glucose'].mean(), inplace = True)  
data_copy['BloodPressure'].fillna(data_copy['BloodPressure'].mean(), inplace = True)  
data_copy['SkinThickness'].fillna(data_copy['SkinThickness'].median(), inplace = True)  
data_copy['Insulin'].fillna(data_copy['Insulin'].median(), inplace = True)  
data_copy['BMI'].fillna(data_copy['BMI'].median(), inplace = True)`

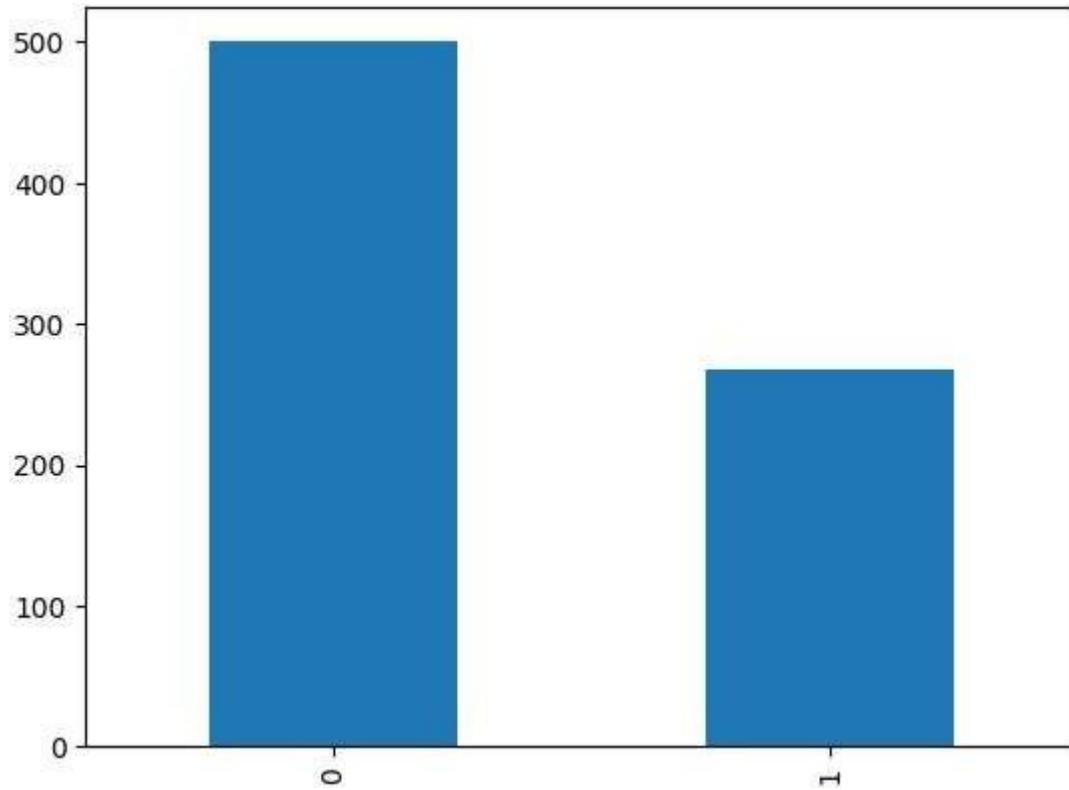
```
In [8]: p = data_copy.hist(figsize = (20,20))
```



```
In [9]: import missingno as msno
p = msno.bar(data)
```

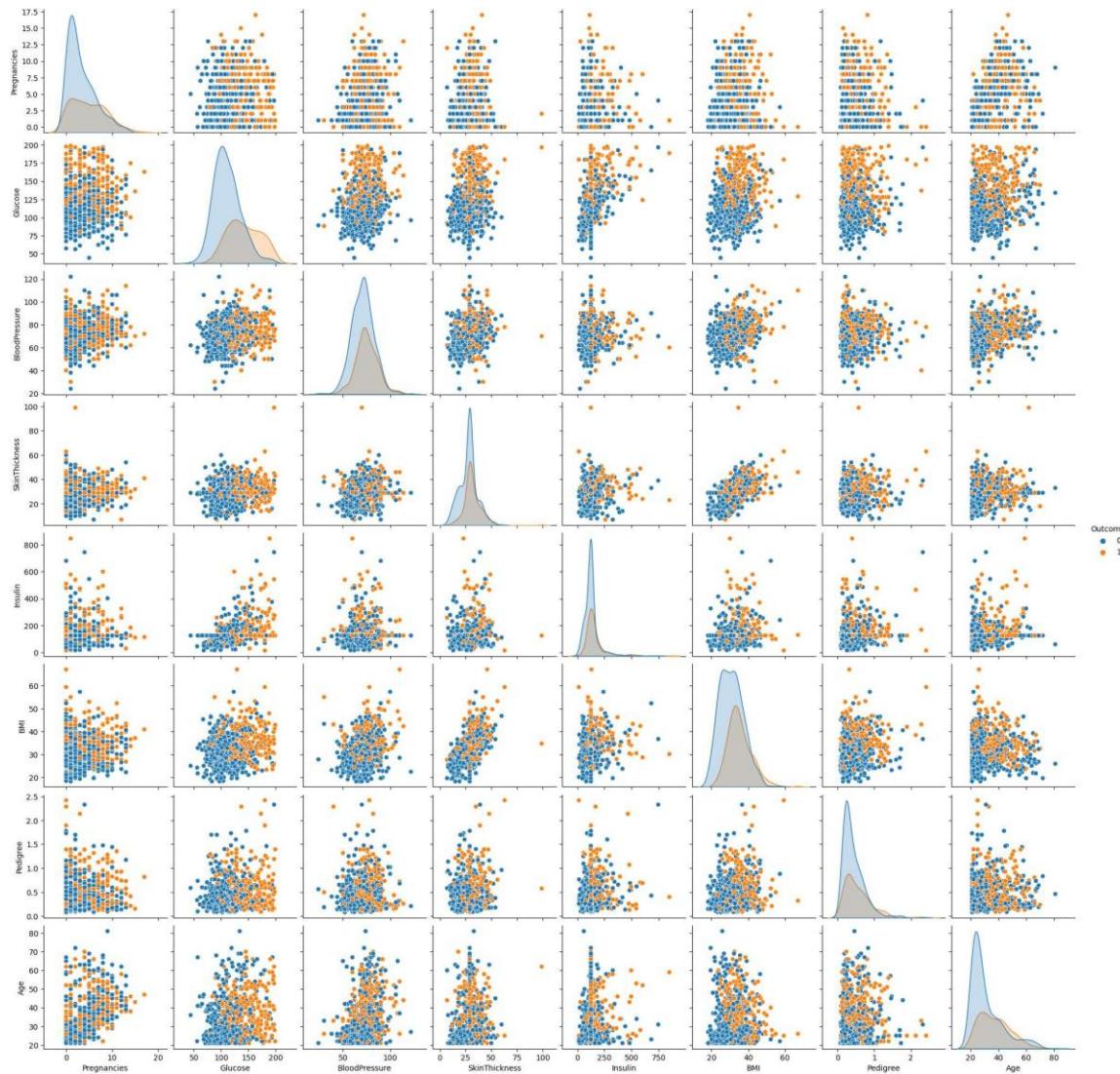


```
In [10]: p=data.Outcome.value_counts().plot(kind="bar")
```

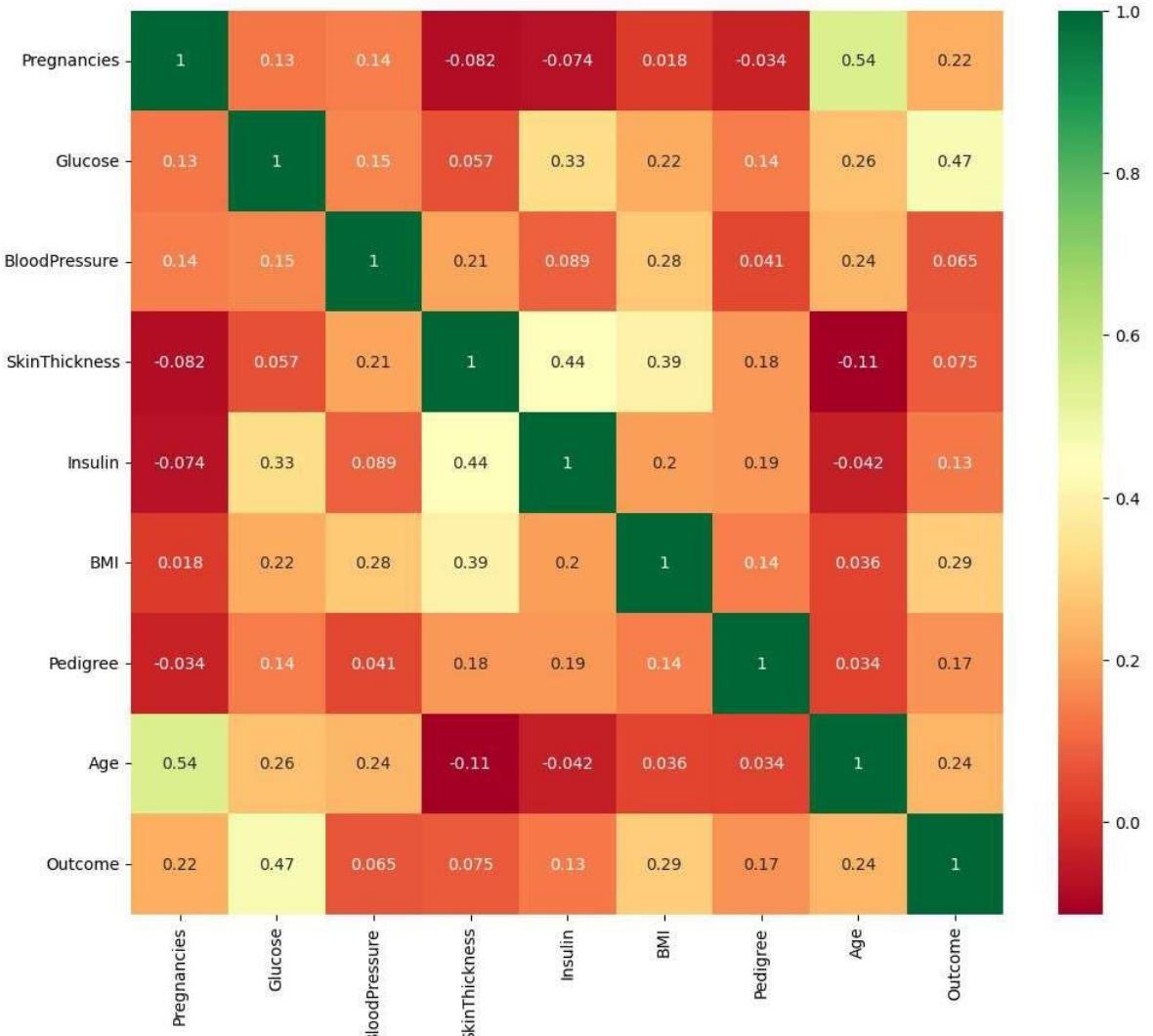


The above graph shows that the data is biased towards datapoints having outcome value as 0 where it means that diabetes was not present actually. The number of non-diabetics is almost twice the number of diabetic patients

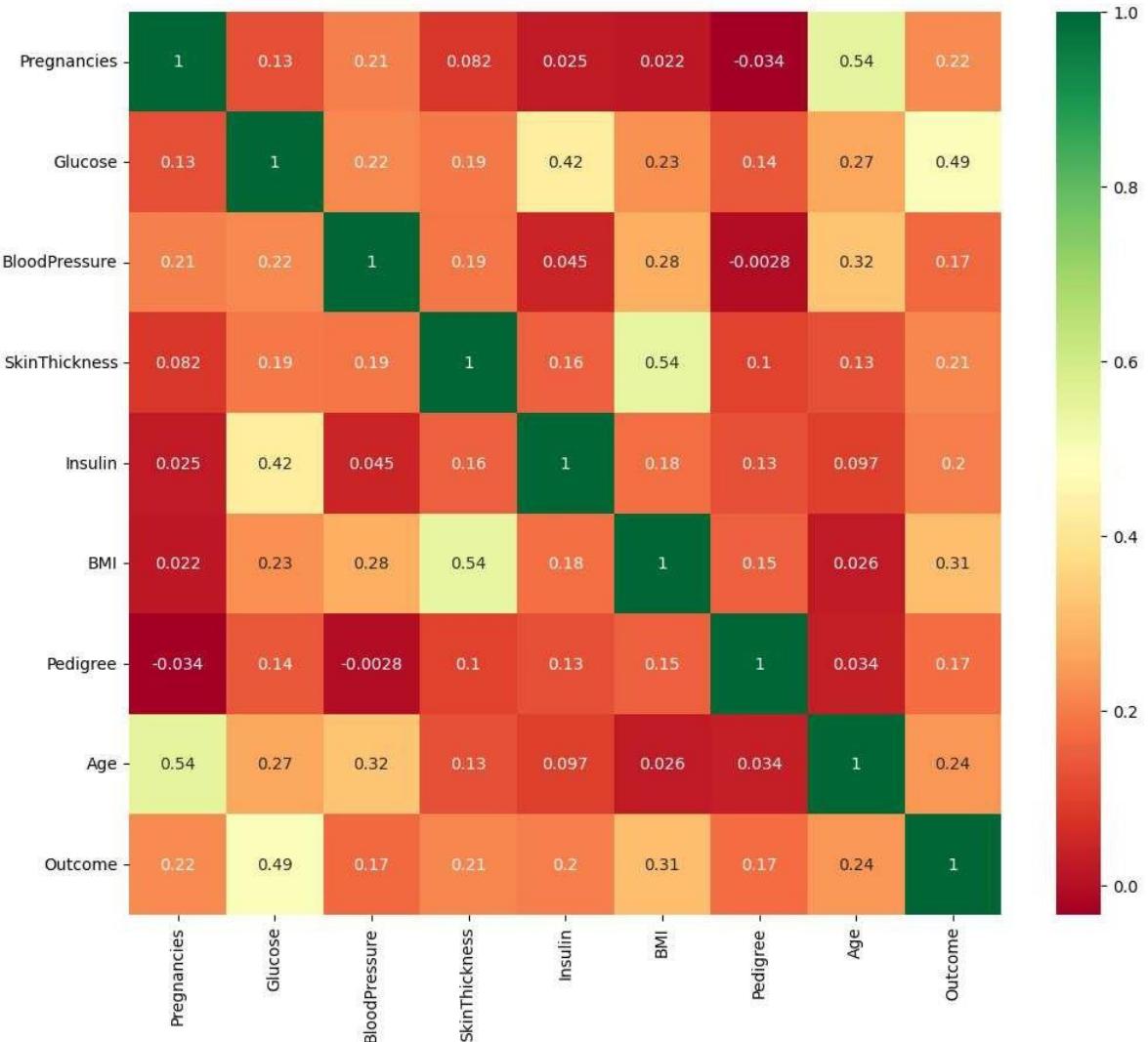
```
In [11]: import seaborn as sns  
p=sns.pairplot(data_copy, hue = 'Outcome')
```



```
In [12]: import matplotlib.pyplot as plt
plt.figure(figsize=(12,10)) # on this line I just set the size of figure to 1
p=sns.heatmap(data.corr(), annot=True, cmap ='RdYlGn') # seaborn has very simp
```



```
In [13]: plt.figure(figsize=(12,10)) # on this Line I just set the size of figure to 1
p=sns.heatmap(data_copy.corr(), annot=True,cmap ='RdYlGn') # seaborn has very
```



```
In [14]: from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = pd.DataFrame(sc_X.fit_transform(data_copy.drop(["Outcome"], axis =1)),columns
'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

```
In [15]: X.head()
```

```
Out[15]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	-
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	-
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	-

```
In [16]: y = data_copy.Outcome
```

```
In [17]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, ran
```

```
In [18]: from sklearn.neighbors import KNeighborsClassifier
```

```
train_scores = []
test_scores = []

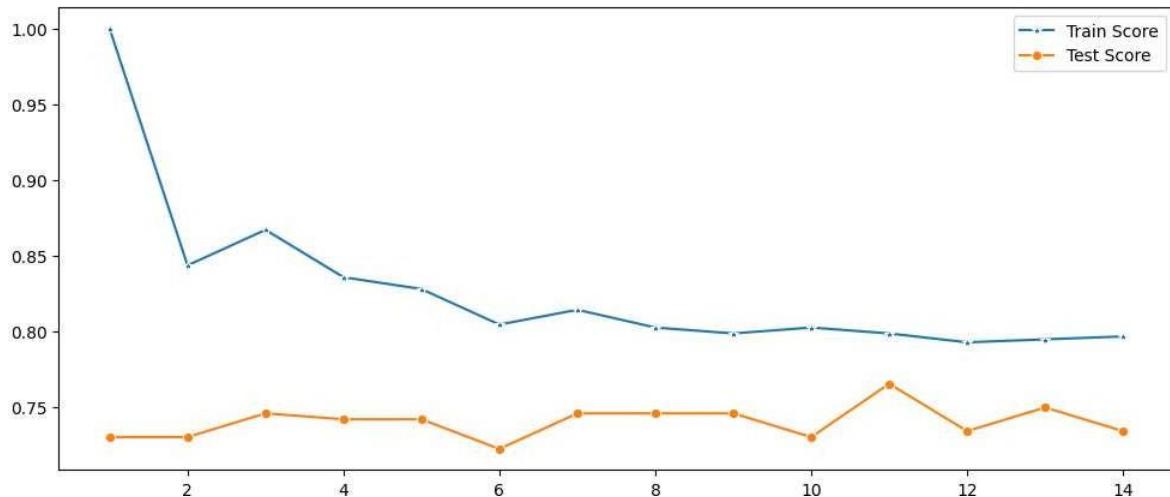
for i in range(1,15):
    knn = KNeighborsClassifier(i)
    knn.fit(X_train, y_train)
    train_scores.append(knn.score(X_train, y_train))
    test_scores.append(knn.score(X_test, y_test))
```

```
In [19]: max_test_score = max(test_scores)
```

```
In [20]: test_score_index = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100, list(map(lambda
```

Max test score 76.5625 % and k = [11]

```
In [23]: plt.figure(figsize=(12,5))
p = sns.lineplot(x=range(1, 15), y=train_scores, marker='*', label='Train Score')
p = sns.lineplot(x=range(1, 15), y=test_scores, marker='o', label='Test Score')
```



```
In [24]: # K=11
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

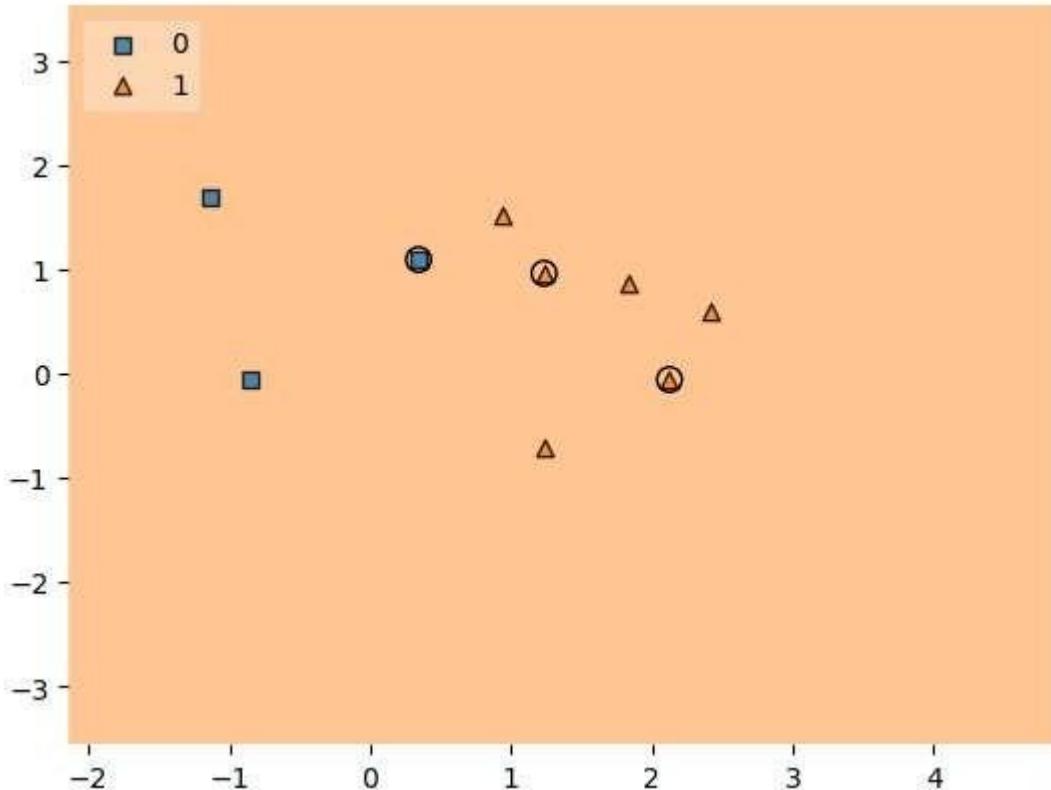
Out[24]: 0.765625

```
In [25]: from mlxtend.plotting import plot_decision_regions
value = 20000
width = 20000

plot_decision_regions(X.values, y.values, clf = knn, legend = 2,filler_feature_
filler_feature_ranges={2: width, 3: width, 4: width, 5: width}
X_highlight=X_test.values)
plt.title("KNN with diabetes data")
plt.show()
```

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names  
warnings.warn(

KNN with diabetes data

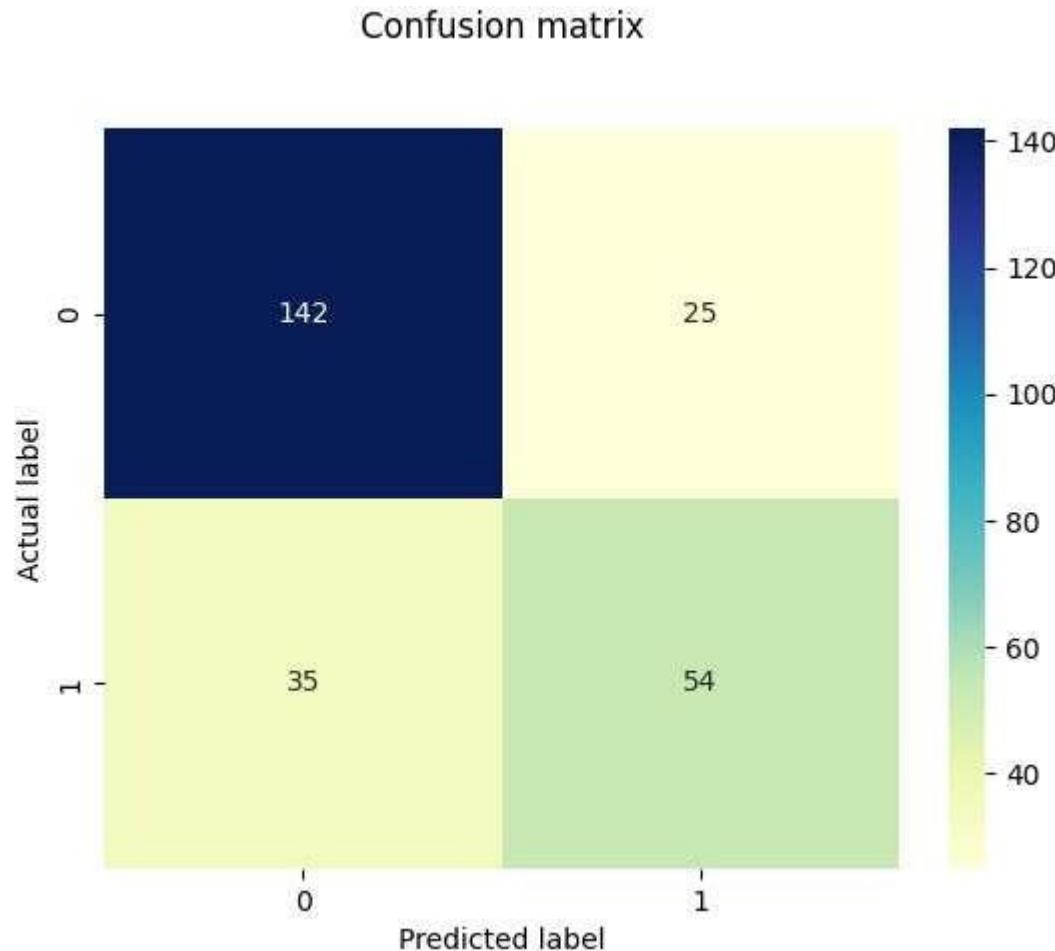


```
In [26]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
y_pred = knn.predict(X_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
```

```
In [27]: p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[27]: Text(0.5, 23.52222222222222, 'Predicted label')



```
In [28]: def model_evaluation(y_test, y_pred, model_name):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta = 2.0)

    results = pd.DataFrame([[model_name, acc, prec, rec, f1, f2]],
                           columns = ["Model", "Accuracy", "Precision", "Recall",
                                      "F1 Score", "F2 Score"])
    results = results.sort_values(["Precision", "Recall", "F2 Score"], ascending=False)
    return results

model_evaluation(y_test, y_pred, "KNN")
```

Out[28]:

	Model	Accuracy	Precision	Recall	F1 Score	F2 Score
0	KNN	0.765625	0.683544	0.606742	0.642857	0.62069

```
In [29]: # Alternate way
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

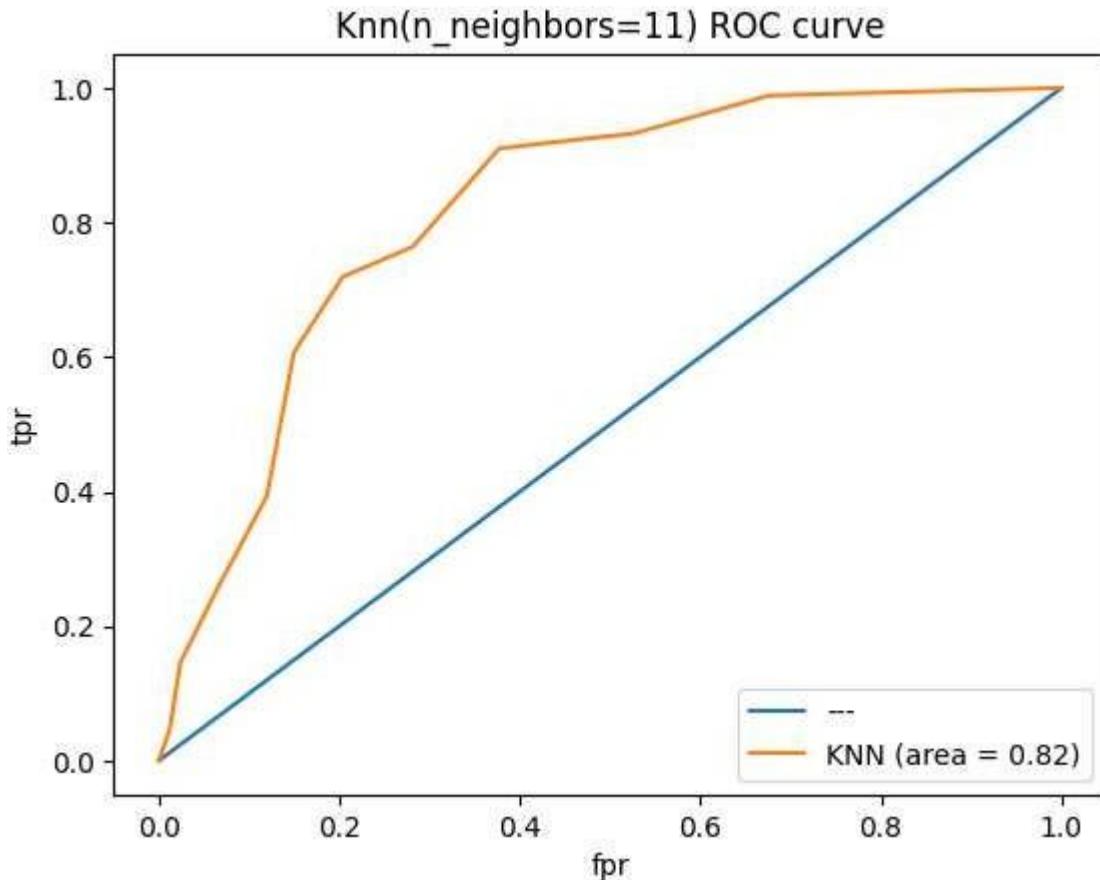
	precision	recall	f1-score	support
0	0.80	0.85	0.83	167
1	0.68	0.61	0.64	89
accuracy			0.77	256
macro avg	0.74	0.73	0.73	256
weighted avg	0.76	0.77	0.76	256

```
In [30]: from sklearn.metrics import auc, roc_auc_score, roc_curve

y_pred_proba = knn.predict_proba(X_test)[:, -1]
fpr, tpr, threshold = roc_curve(y_test, y_pred_proba)
```

```
In [31]: classifier_roc_auc = roc_auc_score(y_test, y_pred_proba)
plt.plot([0,1],[0,1], label = "----")

plt.plot(fpr, tpr, label ='KNN (area = %0.2f)' % classifier_roc_auc)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title('Knn(n_neighbors=11) ROC curve')
plt.legend(loc="lower right", fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")
plt.show()
```



In [32]: #Hyper parameters tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
parameters_grid = {"n_neighbors": np.arange(0,50)}
knn= KNeighborsClassifier()
knn_GSV = GridSearchCV(knn, param_grid=parameters_grid, cv = 5)
knn_GSV.fit(X, y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
5 fits failed out of a total of 250.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting erro
r_score='raise'.
```

Below are more details about the failures:

---

```
---
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
  dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classifica
  tion.py", line 213, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, i
  n _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validati
  on.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'n_neighbors' para
meter of KNeighborsClassifier must be an int in the range [1, inf) or None. G
ot 0 instead.

    warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:95
2: UserWarning: One or more of the test scores are non-finite: [      nan 0.
68759019 0.71362363 0.73312962 0.7369663  0.73441134
  0.73700025 0.74609965 0.74870554 0.75395977 0.74743231 0.76436635
  0.75787285 0.76699771 0.75524998 0.76306765 0.76828792 0.76698073
  0.77086835 0.76698073 0.76438333 0.76696376 0.76307614 0.76566505
  0.76176895 0.77218403 0.76566505 0.76958662 0.77088532 0.76568203
  0.76045327 0.76306765 0.76305916 0.76047874 0.7696036  0.76437484
  0.76046176 0.76047874 0.76046176 0.75526696 0.76176895 0.76438333
  0.75655717 0.75916306 0.75785587 0.75525847 0.75264409 0.75135387
  0.75265258 0.75136236]
    warnings.warn(
```

```
Out[32]: GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
                      param_grid={'n_neighbors': array([ 0,  1,  2,  3,  4,  5,  6,  
                         7,  8,  9, 10, 11, 12, 13, 14, 15, 16,  
                         17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,  
                         34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with [nbviewer.org](#).

```
In [33]: print("Best Params" ,knn_GSV.best_params_)  
print("Best score" ,knn_GSV.best_score_)
```

```
Best Params {'n_neighbors': 25}  
Best score 0.7721840251252015
```

```
In [ ]:
```