exp-ml-1

October 14, 2024

[1]:
```python
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
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import r2_score, mean_squared_error
```

[7]:
```python
data = pd.read_csv("uber.csv")
data
```

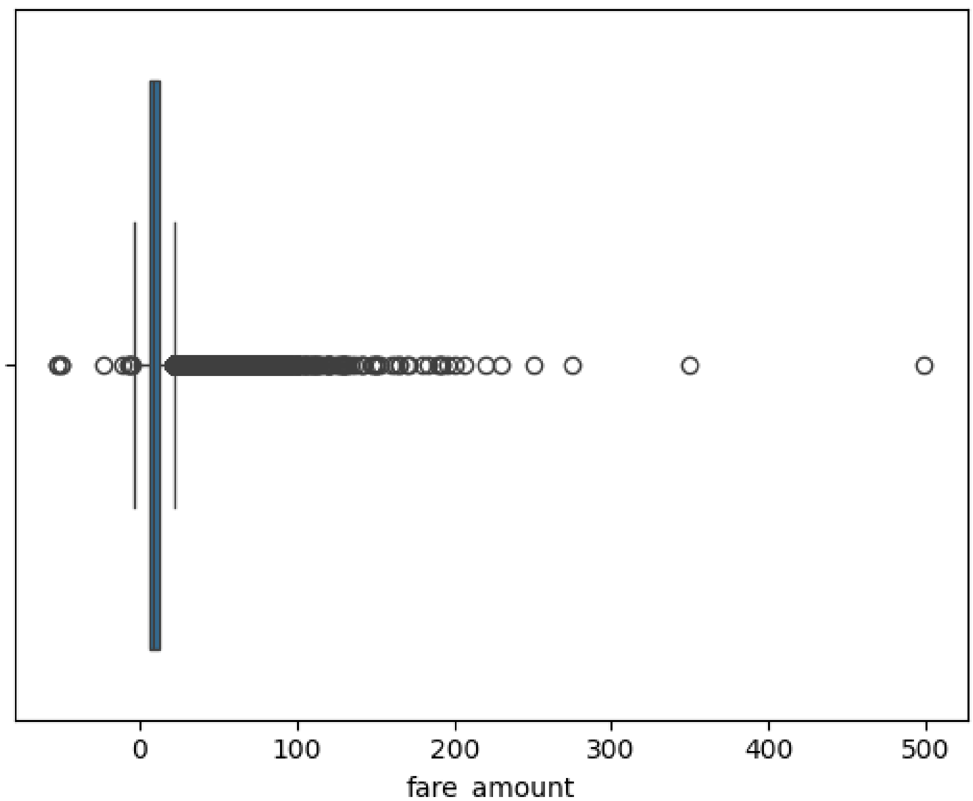
[7]:
```
        Unnamed: 0                            key  fare_amount  \
0         24238194    2015-05-07 19:52:06.0000003          7.5
1         27835199    2009-07-17 20:04:56.0000002          7.7
2         44984355   2009-08-24 21:45:00.00000061         12.9
3         25894730    2009-06-26 08:22:21.0000001          5.3
4         17610152  2014-08-28 17:47:00.000000188         16.0
…              …                            …          …
199995    42598914   2012-10-28 10:49:00.00000053          3.0
199996    16382965    2014-03-14 01:09:00.0000008          7.5
199997    27804658   2009-06-29 00:42:00.00000078         30.9
199998    20259894    2015-05-20 14:56:25.0000004         14.5
199999    11951496   2010-05-15 04:08:00.00000076         14.1

                pickup_datetime  pickup_longitude  pickup_latitude  \
0       2015-05-07 19:52:06 UTC        -73.999817        40.738354
1       2009-07-17 20:04:56 UTC        -73.994355        40.728225
2       2009-08-24 21:45:00 UTC        -74.005043        40.740770
3       2009-06-26 08:22:21 UTC        -73.976124        40.790844
4       2014-08-28 17:47:00 UTC        -73.925023        40.744085
…                           …               …              …
199995  2012-10-28 10:49:00 UTC        -73.987042        40.739367
199996  2014-03-14 01:09:00 UTC        -73.984722        40.736837
199997  2009-06-29 00:42:00 UTC        -73.986017        40.756487
199998  2015-05-20 14:56:25 UTC        -73.997124        40.725452
```

```
199999  2010-05-15 04:08:00 UTC        -73.984395        40.720077

        dropoff_longitude  dropoff_latitude  passenger_count
0               -73.999512         40.723217                1
1               -73.994710         40.750325                1
2               -73.962565         40.772647                1
3               -73.965316         40.803349                3
4               -73.973082         40.761247                5
…                      …               …               …
199995          -73.986525         40.740297                1
199996          -74.006672         40.739620                1
199997          -73.858957         40.692588                2
199998          -73.983215         40.695415                1
199999          -73.985508         40.768793                1

[200000 rows x 9 columns]
```

[8]:
```python
# 1. Pre-process the dataset

# Remove unnecessary column
data["pickup_datetime"] = pd.to_datetime(data["pickup_datetime"])

missing_values = data.isnull().sum()
print("Missing values in the dataset:")
print(missing_values)

# Handle missing values
# We can choose to drop rows with missing values or fill them with appropriate␣
↪values.

data.dropna(inplace=True)

# To fill missing values with the mean value of the column:
# data.fillna(data.mean(), inplace=True)

# Ensure there are no more missing values
missing_values = data.isnull().sum()
print("Missing values after handling:")
print(missing_values)

# 2. Identify outliers
# visualization to detect outliers.
sns.boxplot(x=data["fare_amount"])
plt.show()
```

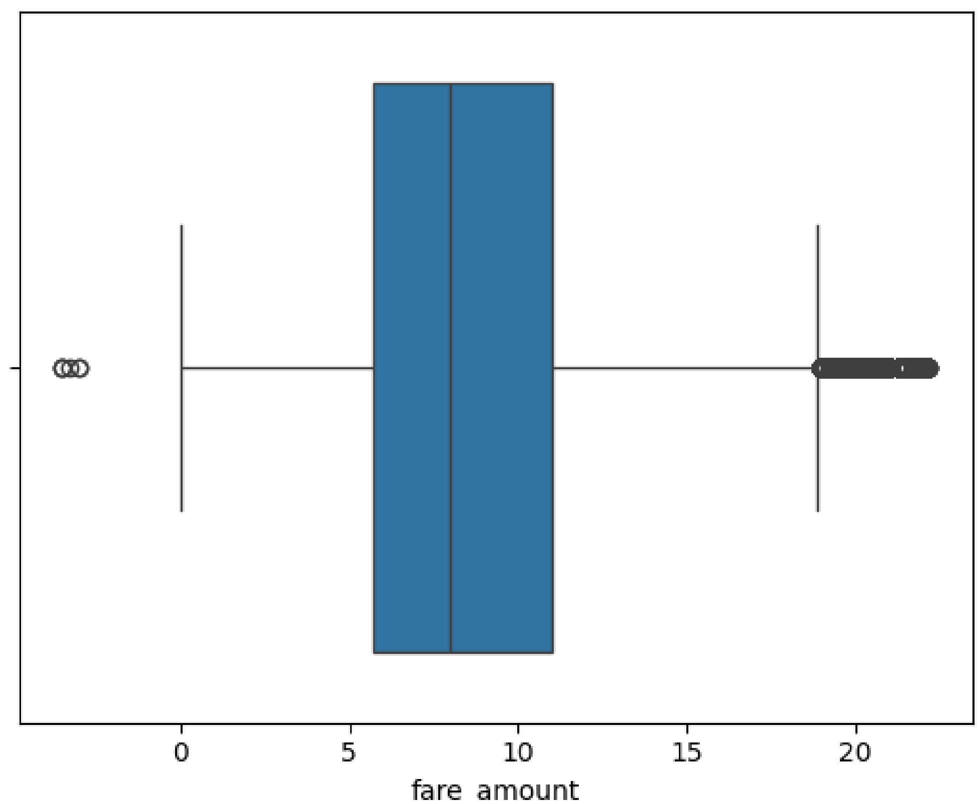
```
Missing values in the dataset:
Unnamed: 0           0
```

```
key                   0
fare_amount           0
pickup_datetime       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     1
dropoff_latitude      1
passenger_count       0
dtype: int64
Missing values after handling:
Unnamed: 0            0
key                   0
fare_amount           0
pickup_datetime       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
passenger_count       0
dtype: int64
```



[9]:

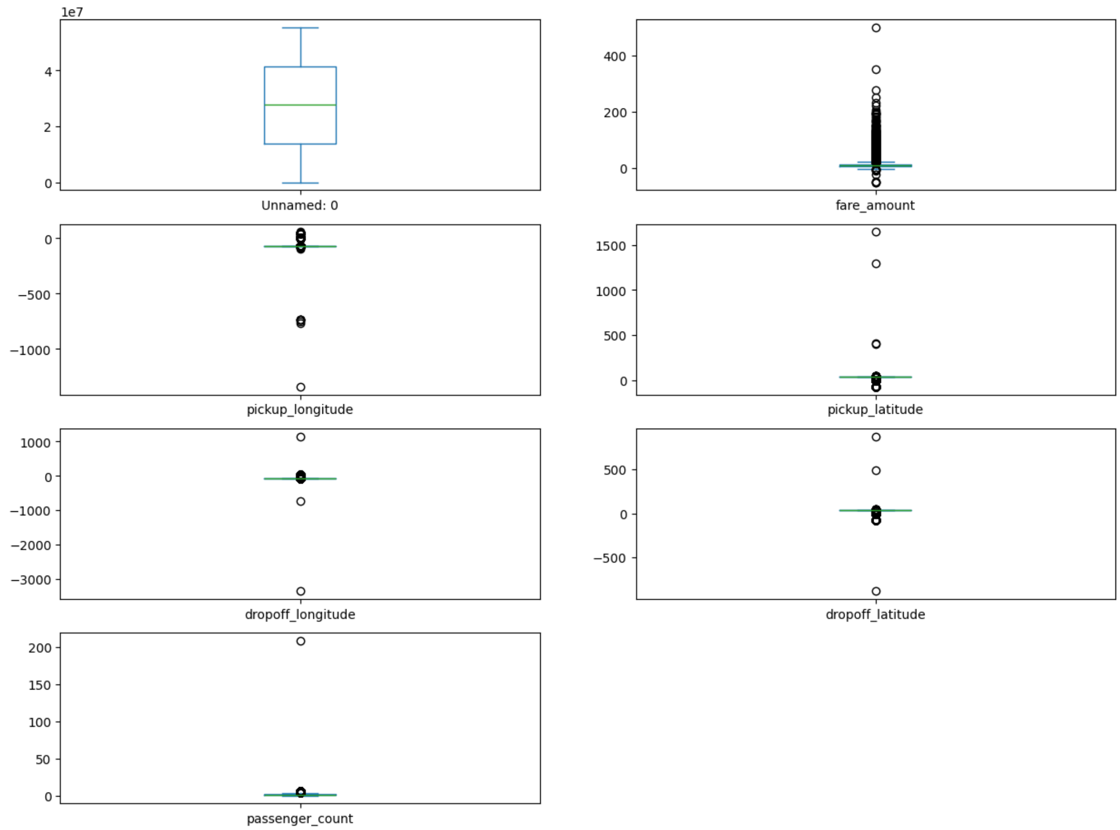
```
# Calculate the IQR for the 'fare_amount' column
Q1 = data["fare_amount"].quantile(0.25)
Q3 = data["fare_amount"].quantile(0.75)
IQR = Q3 - Q1

# Define a threshold (e.g., 1.5 times the IQR) to identify outliers
threshold = 1.5
lower_bound = Q1 - threshold * IQR
upper_bound = Q3 + threshold * IQR

# Remove outliers
data_no_outliers = data[(data["fare_amount"] >= lower_bound) &
    (data["fare_amount"] <= upper_bound)]

# Visualize the 'fare_amount' distribution without outliers
sns.boxplot(x=data_no_outliers["fare_amount"])
plt.show()
```



[10]:

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

[10]:
```
Unnamed: 0                 Axes(0.125,0.786098;0.352273x0.0939024)
fare_amount             Axes(0.547727,0.786098;0.352273x0.0939024)
pickup_longitude           Axes(0.125,0.673415;0.352273x0.0939024)
pickup_latitude         Axes(0.547727,0.673415;0.352273x0.0939024)
dropoff_longitude          Axes(0.125,0.560732;0.352273x0.0939024)
dropoff_latitude        Axes(0.547727,0.560732;0.352273x0.0939024)
passenger_count            Axes(0.125,0.448049;0.352273x0.0939024)
dtype: object
```



[12]:
```python
# 4. Implement linear regression and random forest regression models
# Split the data into features and target variable
X = data[['pickup_longitude', 'pickup_latitude', 'dropoff_longitude',␣
 ↪'dropoff_latitude', 'passenger_count']]
y = data['fare_amount']  #Target

y
```

[12]:
```
0          7.5
1          7.7
2         12.9
3          5.3
```

```
4          16.0
           …
199995      3.0
199996      7.5
199997     30.9
199998     14.5
199999     14.1
Name: fare_amount, Length: 199999, dtype: float64
```

[13]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[14]:
```
# Create and train the linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

[14]:
```
LinearRegression()
```

[ ]:
```
# Create and train the random forest regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

[ ]:
```
# 5. Evaluate the models
# Predict the values
y_pred_lr = lr_model.predict(X_test)
y_pred_lr
print("Linear Model:",y_pred_lr)
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Model:", y_pred_rf)
```

[ ]:
```
# Calculate R-squared (R2) and Root Mean Squared Error (RMSE) for both models
r2_lr = r2_score(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
```

[ ]:
```
# Compare the scores
print("Linear Regression - R2:", r2_lr)
print("Linear Regression - RMSE:", rmse_lr)
```

[ ]:
```
# Compare the scores
print("Linear Regression - R2:", r2_lr)
print("Linear Regression - RMSE:", rmse_lr)
```

[ ]:
```

```

exp-ml-2

October 14, 2024

[1]:

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

[2]:

```python
# Load the dataset
data = pd.read_csv("emails.csv")  # Replace with the actual path to the dataset
data
```

[2]:

```
      Email No.  the  to  ect  and  for  of    a  you  hou  …  connevey  \
0       Email 1    0   0    1    0    0   0    2    0    0  …         0
1       Email 2    8  13   24    6    6   2  102    1   27  …         0
2       Email 3    0   0    1    0    0   0    8    0    0  …         0
3       Email 4    0   5   22    0    5   1   51    2   10  …         0
4       Email 5    7   6   17    1    5   2   57    0    9  …         0
…           …    …  ..   …    …    …   ..   …    …    …   …       …
5167 Email 5168    2   2    2    3    0   0   32    0    0  …         0
5168 Email 5169   35  27   11    2    6   5  151    4    3  …         0
5169 Email 5170    0   0    1    1    0   0   11    0    0  …         0
5170 Email 5171    2   7    1    0    2   1   28    2    0  …         0
5171 Email 5172   22  24    5    1    6   5  148    8    2  …         0

     jay  valued  lay  infrastructure  military  allowing  ff  dry  \
0      0       0    0               0         0         0   0    0
1      0       0    0               0         0         0   1    0
2      0       0    0               0         0         0   0    0
3      0       0    0               0         0         0   0    0
4      0       0    0               0         0         1   0    0
…    …       …    …               …         …         …  ..    …
5167   0       0    0               0         0         0   0    0
5168   0       0    0               0         0         0   1    0
5169   0       0    0               0         0         0   0    0
5170   0       0    0               0         0         0   1    0
5171   0       0    0               0         0         0   0    0
```

```
      Prediction
0              0
1              0
2              0
3              0
4              0
…            …
5167           0
5168           0
5169           1
5170           1
5171           0

[5172 rows x 3002 columns]
```

[3]:
```python
# 1. Data Preprocessing - Handle missing values if necessary
data.drop(['Email No.'],axis=1, inplace=True)
# 2. Feature Selection/Engineering - Select relevant features
```

[4]:
```python
# 3. Split the data into training and testing sets
X = data.drop("Prediction", axis=1)  # Features
y = data["Prediction"]  # Target variable
print("Features: ",X)
print("Target: ",y)
```

```
Features:         the  to  ect  and  for  of    a  you  hou  in  …
enhancements  \
0       0   0    1    0    0   0    2    0    0   0  …             0
1       8  13   24    6    6   2  102    1   27  18  …             0
2       0   0    1    0    0   0    8    0    0   4  …             0
3       0   5   22    0    5   1   51    2   10   1  …             0
4       7   6   17    1    5   2   57    0    9   3  …             0
…   …  ..  …  …  …  ..  …  …  …  ..  …             …
5167    2   2    2    3    0   0   32    0    0   5  …             0
5168   35  27   11    2    6   5  151    4    3  23  …             0
5169    0   0    1    1    0   0   11    0    0   1  …             0
5170    2   7    1    0    2   1   28    2    0   8  …             0
5171   22  24    5    1    6   5  148    8    2  23  …             0

      connevey  jay  valued  lay  infrastructure  military  allowing  ff  dry
0            0    0       0    0               0         0         0   0    0
1            0    0       0    0               0         0         0   1    0
2            0    0       0    0               0         0         0   0    0
3            0    0       0    0               0         0         0   0    0
4            0    0       0    0               0         0         0   1    0
…          …  …     …  …             …       …       …  ..  …
5167         0    0       0    0               0         0         0   0    0
```

```
5168          0     0       0     0               0         0         0    1    0
5169          0     0       0     0               0         0         0    0    0
5170          0     0       0     0               0         0         0    1    0
5171          0     0       0     0               0         0         0    0    0

[5172 rows x 3000 columns]
Target:  0       0
1       0
2       0
3       0
4       0
       ..
5167    0
5168    0
5169    1
5170    1
5171    0
Name: Prediction, Length: 5172, dtype: int64
```

[5]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[6]:
```python
# 4. Model Building
# K-Nearest Neighbors
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Support Vector Machine
svm_model = SVC()
svm_model.fit(X_train, y_train)
```

[6]:
```
SVC()
```

[7]:
```python
# 5. Model Evaluation
# K-Nearest Neighbors
knn_predictions = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
knn_report = classification_report(y_test, knn_predictions)
```

[8]:
```python
print(knn_predictions)
```

```
[0 0 1 … 0 0 0]
```

[9]:
```python
# Print or visualize the evaluation results
print("K-Nearest Neighbors Accuracy:")
print(knn_accuracy)
print("K-Nearest Neighbors Classification Report:")
```

```
print(knn_report)
```

```
K-Nearest Neighbors Accuracy:
0.8608247422680413
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.87      0.90      1097
           1       0.73      0.83      0.78       455

    accuracy                           0.86      1552
   macro avg       0.83      0.85      0.84      1552
weighted avg       0.87      0.86      0.86      1552
```

[10]:
```
# Support Vector Machine
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
svm_report = classification_report(y_test, svm_predictions)
```

[11]:
```
print(svm_predictions)
```

```
[0 0 1 … 0 0 0]
```

[13]:
```
print("Support Vector Machine Accuracy:")
print(svm_accuracy)
print("Support Vector Machine Classification Report:")
print(svm_report)
```

```
Support Vector Machine Accuracy:
0.803479381443299
Support Vector Machine Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.99      0.88      1097
           1       0.92      0.36      0.52       455

    accuracy                           0.80      1552
   macro avg       0.85      0.67      0.70      1552
weighted avg       0.83      0.80      0.77      1552
```

[ ]:

exp-ml-3

October 14, 2024

[21]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
import matplotlib.pyplot as plt
```

[4]:
```
# 1. Read the dataset
data = pd.read_csv("Churn_Modelling.csv")  # Replace with the actual path to␣
 ↪the dataset
data
```

[4]:
```
      RowNumber  CustomerId    Surname  CreditScore Geography  Gender  Age  \
0             1    15634602   Hargrave          619    France  Female   42
1             2    15647311       Hill          608     Spain  Female   41
2             3    15619304       Onio          502    France  Female   42
3             4    15701354       Boni          699    France  Female   39
4             5    15737888   Mitchell          850     Spain  Female   43
…           …           …         …          …         …      …   …
9995       9996    15606229   Obijiaku          771    France    Male   39
9996       9997    15569892  Johnstone          516    France    Male   35
9997       9998    15584532        Liu          709    France  Female   36
9998       9999    15682355  Sabbatini          772   Germany    Male   42
9999      10000    15628319     Walker          792    France  Female   28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0          2       0.00              1          1               1
1          1   83807.86              1          0               1
2          8  159660.80              3          1               0
3          1       0.00              2          0               0
4          2  125510.82              1          1               1
…        …         …            …         …             …
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
```

```
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
0           101348.88       1
1           112542.58       0
2           113931.57       1
3            93826.63       0
4            79084.10       0
…                 …       …
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

[10000 rows x 14 columns]
```

[5]:
```
# 2. Distinguish features and target
X = data.drop("Exited", axis=1)  # Features
y = data["Exited"]  # Target variable
```

[6]:
```
X
```

[6]:
```
      RowNumber  CustomerId    Surname  CreditScore Geography  Gender  Age  \
0             1    15634602  Hargrave          619    France  Female   42
1             2    15647311      Hill          608     Spain  Female   41
2             3    15619304      Onio          502    France  Female   42
3             4    15701354      Boni          699    France  Female   39
4             5    15737888  Mitchell          850     Spain  Female   43
…           …          …         …           …         …      …  …
9995       9996    15606229   Obijiaku          771    France    Male   39
9996       9997    15569892  Johnstone          516    France    Male   35
9997       9998    15584532        Liu          709    France  Female   36
9998       9999    15682355  Sabbatini          772   Germany    Male   42
9999      10000    15628319     Walker          792    France  Female   28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0          2       0.00              1          1               1
1          1   83807.86              1          0               1
2          8  159660.80              3          1               0
3          1       0.00              2          0               0
4          2  125510.82              1          1               1
…        …          …            …          …               …
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
```

```
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary
0           101348.88
1           112542.58
2           113931.57
3            93826.63
4            79084.10
…                 …
9995         96270.64
9996        101699.77
9997         42085.58
9998         92888.52
9999         38190.78

[10000 rows x 13 columns]
```

[7]:
```python
# 2. Divide the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[11]:
```python
data.head()
```

[11]:
```
   CreditScore Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
0          619    France  Female   42       2       0.00              1
1          608     Spain  Female   41       1   83807.86              1
2          502    France  Female   42       8  159660.80              3
3          699    France  Female   39       1       0.00              2
4          850     Spain  Female   43       2  125510.82              1

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

[12]:
```python
data.isna().any()
data.isna().sum()
```

[12]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
```

```
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

[13]:
```python
print(data.shape)
data.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
```

[14]:
```python
data.describe()
```

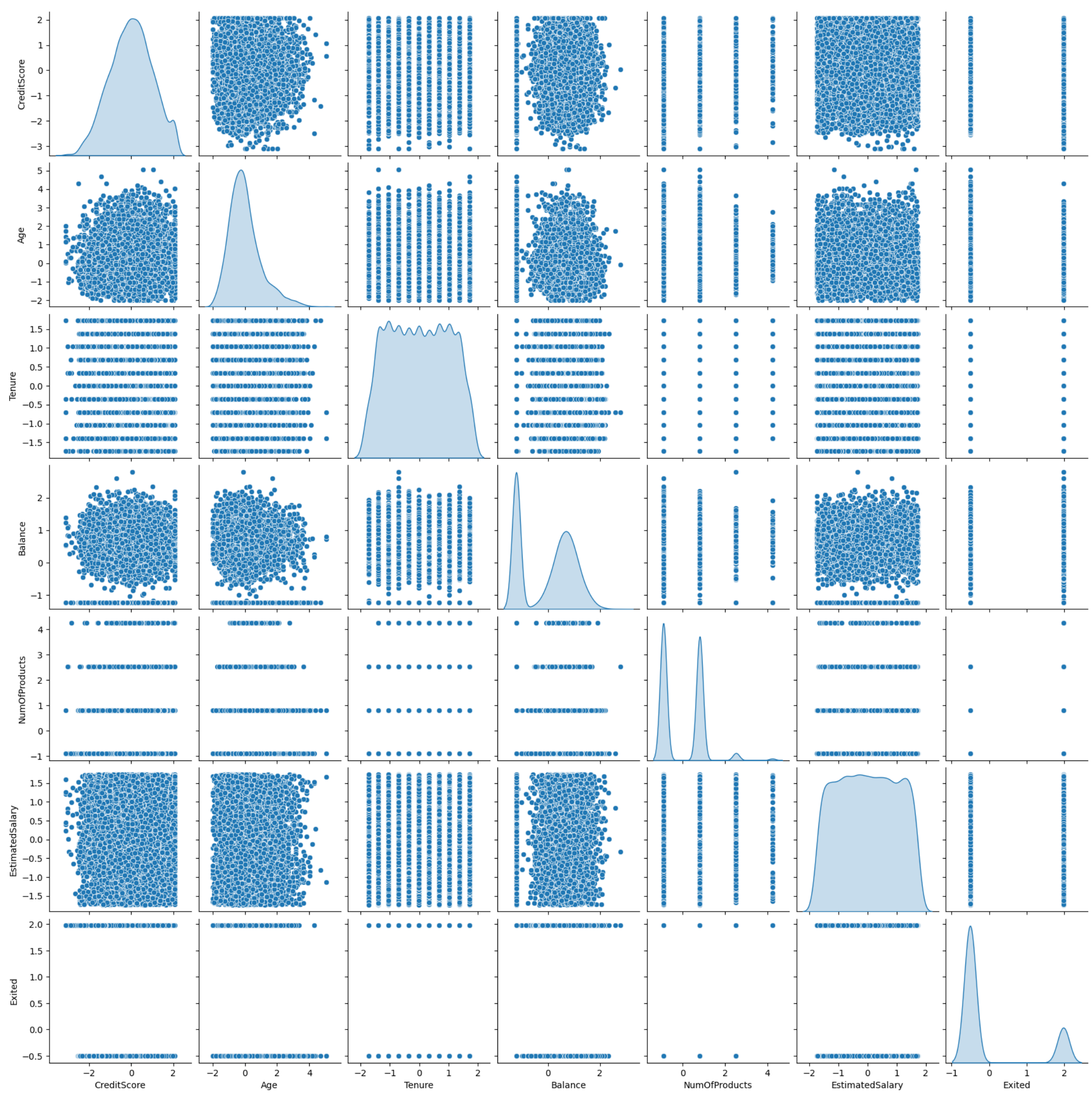
[14]:
```
        CreditScore           Age        Tenure        Balance  NumOfProducts  \
count  10000.000000  10000.000000  10000.000000   10000.000000   10000.000000
mean     650.528800     38.921800      5.012800   76485.889288       1.530200
std       96.653299     10.487806      2.892174   62397.405202       0.581654
min      350.000000     18.000000      0.000000       0.000000       1.000000
25%      584.000000     32.000000      3.000000       0.000000       1.000000
50%      652.000000     37.000000      5.000000   97198.540000       1.000000
75%      718.000000     44.000000      7.000000  127644.240000       2.000000
max      850.000000     92.000000     10.000000  250898.090000       4.000000

         HasCrCard  IsActiveMember  EstimatedSalary        Exited
count  10000.00000    10000.000000     10000.000000  10000.000000
mean       0.70550        0.515100    100090.239881      0.203700
std        0.45584        0.499797     57510.492818      0.402769
min        0.00000        0.000000        11.580000      0.000000
```

25%	0.00000	0.000000	51002.110000	0.000000
50%	1.00000	1.000000	100193.915000	0.000000
75%	1.00000	1.000000	149388.247500	0.000000
max	1.00000	1.000000	199992.480000	1.000000

[17]:

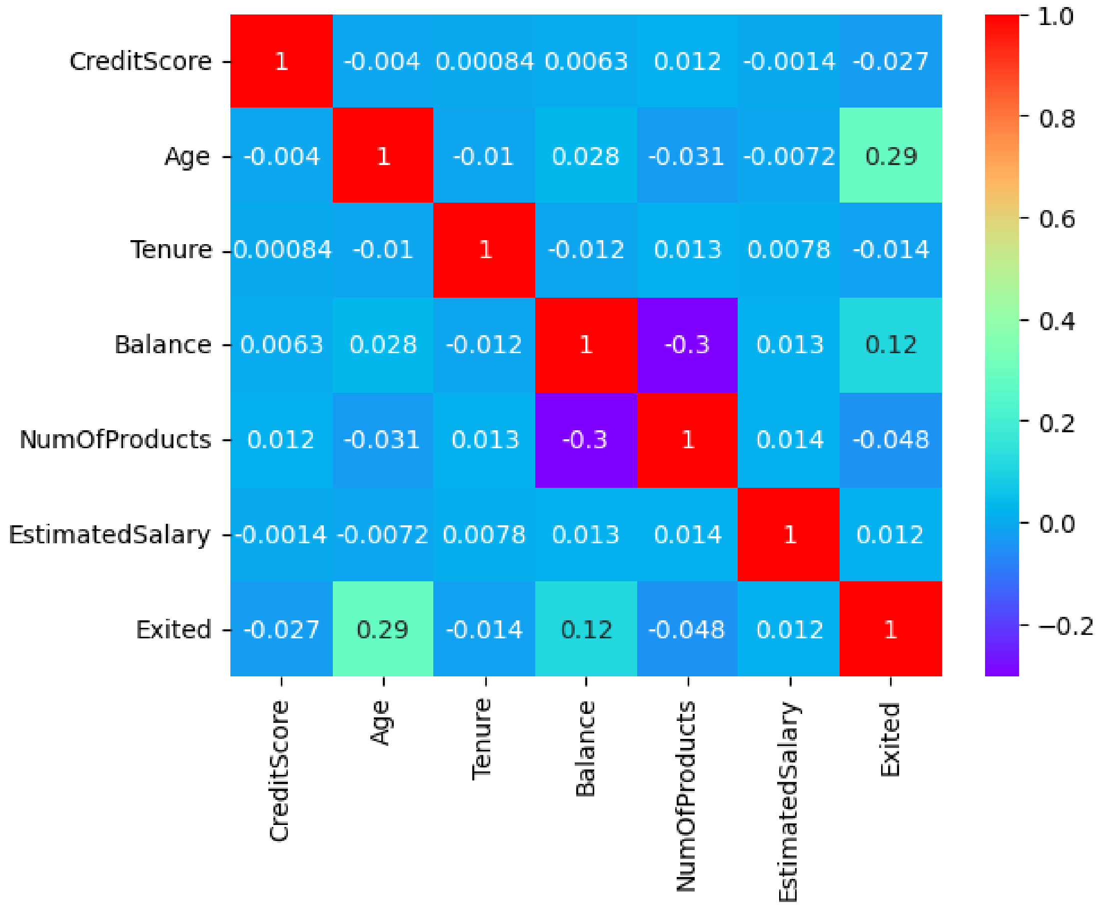
```
## Scale the data
scaler=StandardScaler()
## Extract only the Numerical Columns to perform Bivariate Analysis
subset=data.drop(['Geography','Gender','HasCrCard','IsActiveMember'],axis=1)
scaled=scaler.fit_transform(subset)
scaled_df=pd.DataFrame(scaled,columns=subset.columns)
sns.pairplot(scaled_df,diag_kind='kde')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

[17]: <seaborn.axisgrid.PairGrid at 0x1d6810d5790>

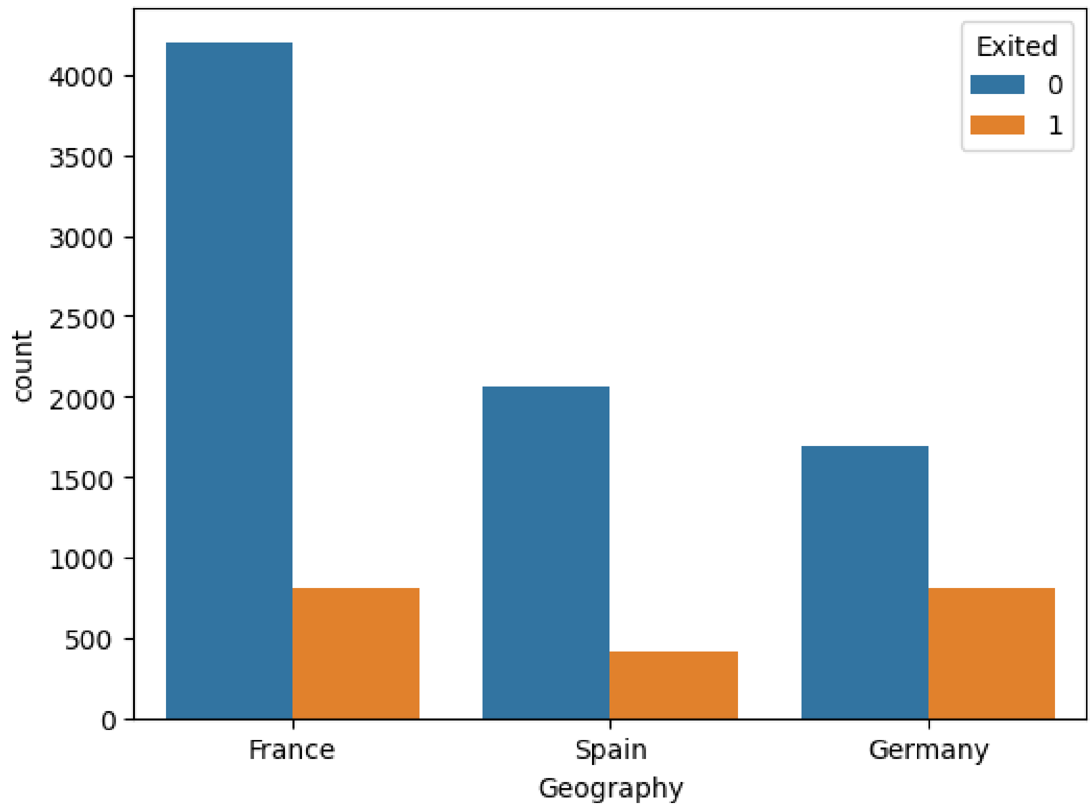


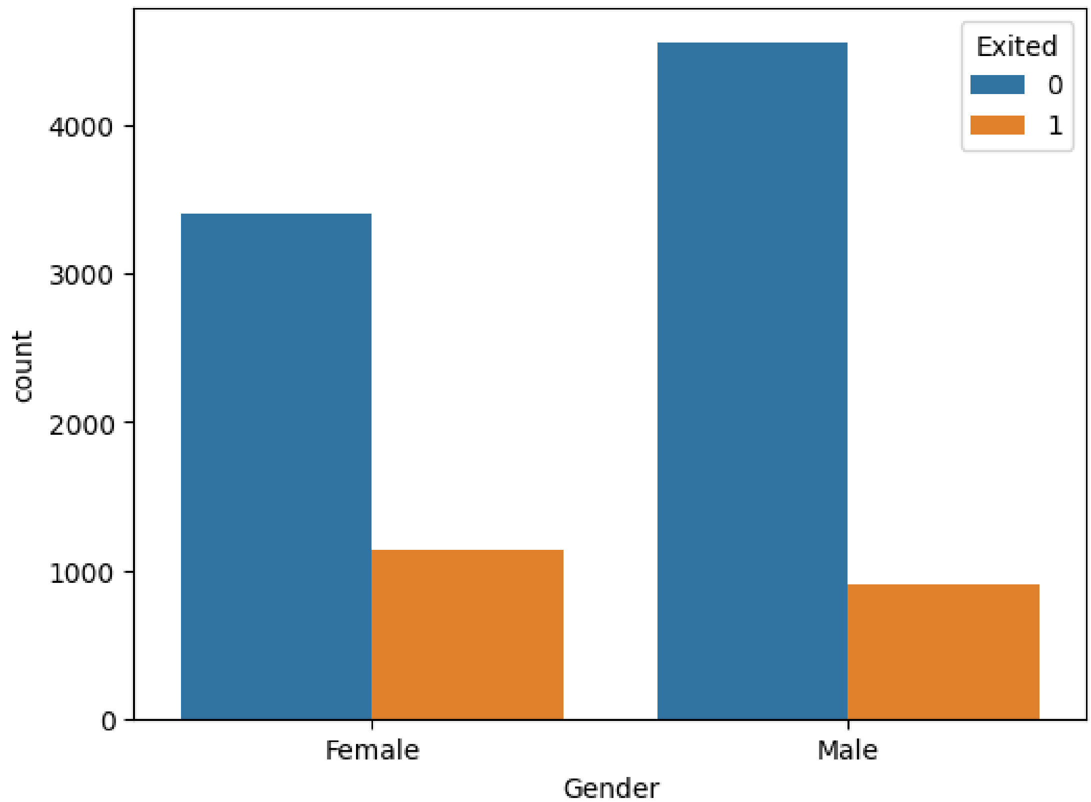
[18]:
```
sns.heatmap(scaled_df.corr(),annot=True,cmap='rainbow')
```

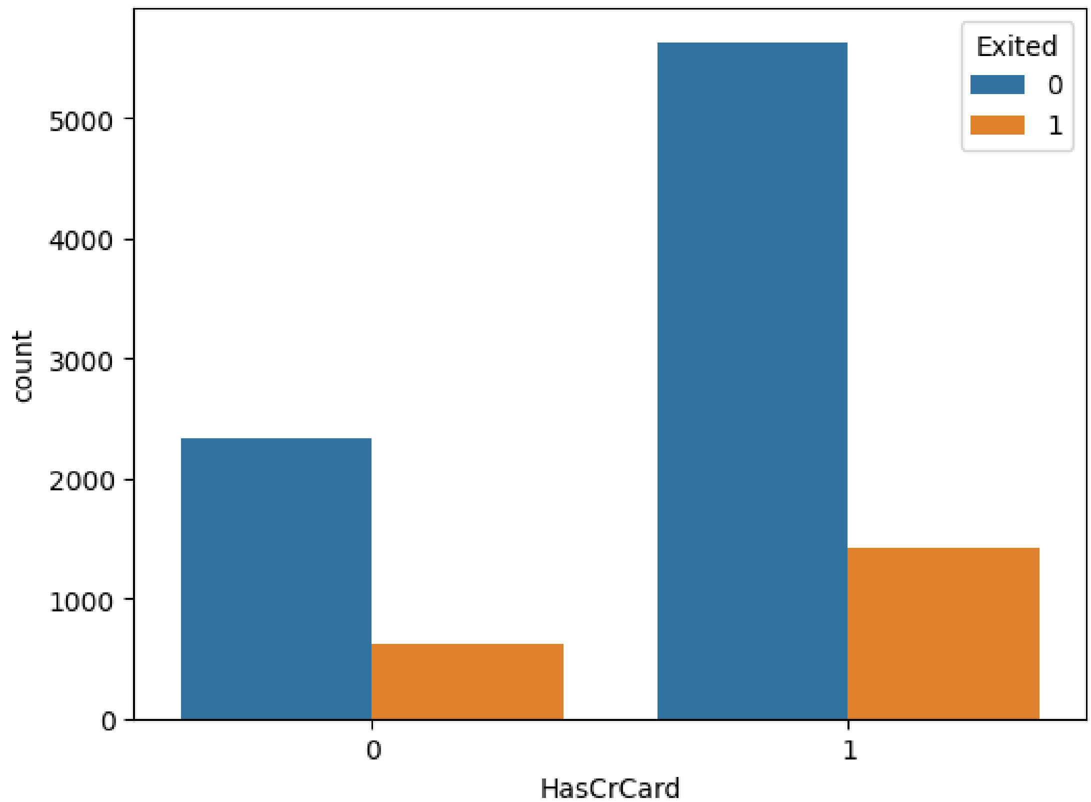
[18]: <Axes: >

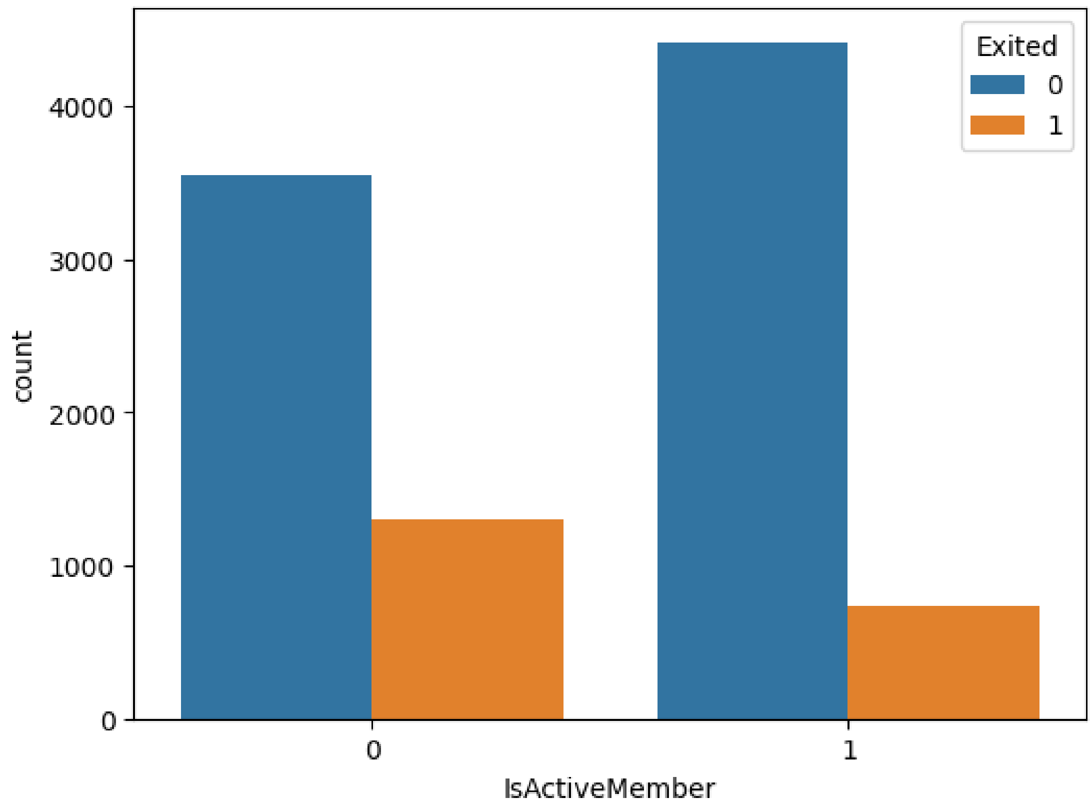


[22]:
```
## Categorical Features vs Target Variable
sns.countplot(x='Geography',data=data,hue='Exited')
plt.show()
sns.countplot(x='Gender',data=data,hue='Exited')
plt.show()
sns.countplot(x='HasCrCard',data=data,hue='Exited')
plt.show()
sns.countplot(x='IsActiveMember',data=data,hue='Exited')
plt.show()
```


Exited
0
1
4000
3500
3000
2500
2000
1500
1000
500
0
count
France
Spain
Germany
Geography


Exited
0
1
4000
3000
2000
1000
0
count
Female
Male
Gender


Exited
0
1
5000
4000
3000
2000
1000
0
count
0
1
HasCrCard



[25]:
```
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=5)
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.
 ↪10,random_state=5)
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
```

[26]:
```
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()
```

[26]:

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

  Geography  Gender  HasCrCard  IsActiveMember
0    France    Male          1               1
1     Spain  Female          1               0
2   Germany  Female          1               0
3    France    Male          1               1
4     Spain    Male          1               1
```

[27]:

```
## 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'])
```

[28]:

```
## 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'])
```

[29]:

```
## 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()
```

[29]:

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

   HasCrCard  IsActiveMember  Geography_France  Geography_Germany  \
```

```
0          1               1             True               False
1          1               0            False               False
2          1               0            False                True
3          1               1             True               False
4          1               1            False               False

   Geography_Spain  Gender_Female  Gender_Male
0            False          False         True
1             True           True        False
2            False           True        False
3            False          False         True
4             True          False         True
```

[30]:
```python
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'))
```

[31]:
```python
import tensorflow as tf
optimizer=tf.keras.optimizers.Adam(0.01)
model.
 ↪compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
```

[32]:
```python
model.fit(X_train_num_df,y_train,epochs=100,batch_size=10,verbose=1)
```

```
Epoch 1/100
810/810              2s 981us/step -
accuracy: 0.8120 - loss: 0.4428
Epoch 2/100
810/810              1s 1ms/step -
accuracy: 0.8546 - loss: 0.3542
Epoch 3/100
810/810              1s 797us/step -
accuracy: 0.8493 - loss: 0.3578
Epoch 4/100
810/810              1s 879us/step -
accuracy: 0.8586 - loss: 0.3458
Epoch 5/100
810/810              1s 1ms/step -
accuracy: 0.8568 - loss: 0.3542
Epoch 6/100
810/810              1s 909us/step -
accuracy: 0.8638 - loss: 0.3400
Epoch 7/100
```

```
810/810             1s 1ms/step -
accuracy: 0.8610 - loss: 0.3436
Epoch 8/100
810/810             1s 976us/step -
accuracy: 0.8689 - loss: 0.3364
Epoch 9/100
810/810             1s 768us/step -
accuracy: 0.8661 - loss: 0.3352
Epoch 10/100
810/810             1s 772us/step -
accuracy: 0.8691 - loss: 0.3245
Epoch 11/100
810/810             1s 1ms/step -
accuracy: 0.8708 - loss: 0.3289
Epoch 12/100
810/810             1s 865us/step -
accuracy: 0.8663 - loss: 0.3312
Epoch 13/100
810/810             1s 794us/step -
accuracy: 0.8698 - loss: 0.3249
Epoch 14/100
810/810             1s 725us/step -
accuracy: 0.8612 - loss: 0.3397
Epoch 15/100
810/810             1s 716us/step -
accuracy: 0.8636 - loss: 0.3346
Epoch 16/100
810/810             1s 957us/step -
accuracy: 0.8636 - loss: 0.3314
Epoch 17/100
810/810             1s 796us/step -
accuracy: 0.8633 - loss: 0.3295
Epoch 18/100
810/810             1s 994us/step -
accuracy: 0.8613 - loss: 0.3302
Epoch 19/100
810/810             1s 884us/step -
accuracy: 0.8625 - loss: 0.3317
Epoch 20/100
810/810             1s 979us/step -
accuracy: 0.8763 - loss: 0.3156
Epoch 21/100
810/810             1s 851us/step -
accuracy: 0.8660 - loss: 0.3360
Epoch 22/100
810/810             1s 740us/step -
accuracy: 0.8607 - loss: 0.3388
Epoch 23/100
```

```
810/810             1s 764us/step -
accuracy: 0.8632 - loss: 0.3390
Epoch 24/100
810/810             1s 768us/step -
accuracy: 0.8628 - loss: 0.3294
Epoch 25/100
810/810             1s 717us/step -
accuracy: 0.8662 - loss: 0.3315
Epoch 26/100
810/810             1s 734us/step -
accuracy: 0.8685 - loss: 0.3291
Epoch 27/100
810/810             1s 754us/step -
accuracy: 0.8678 - loss: 0.3241
Epoch 28/100
810/810             1s 710us/step -
accuracy: 0.8679 - loss: 0.3330
Epoch 29/100
810/810             1s 1ms/step -
accuracy: 0.8713 - loss: 0.3242
Epoch 30/100
810/810             1s 812us/step -
accuracy: 0.8719 - loss: 0.3200
Epoch 31/100
810/810             1s 738us/step -
accuracy: 0.8629 - loss: 0.3244
Epoch 32/100
810/810             1s 754us/step -
accuracy: 0.8655 - loss: 0.3283
Epoch 33/100
810/810             1s 850us/step -
accuracy: 0.8708 - loss: 0.3211
Epoch 34/100
810/810             1s 869us/step -
accuracy: 0.8720 - loss: 0.3163
Epoch 35/100
810/810             1s 977us/step -
accuracy: 0.8631 - loss: 0.3313
Epoch 36/100
810/810             1s 999us/step -
accuracy: 0.8699 - loss: 0.3218
Epoch 37/100
810/810             1s 931us/step -
accuracy: 0.8707 - loss: 0.3212
Epoch 38/100
810/810             1s 909us/step -
accuracy: 0.8721 - loss: 0.3168
Epoch 39/100
```

```
810/810             1s 910us/step -
accuracy: 0.8721 - loss: 0.3207
Epoch 40/100
810/810             1s 792us/step -
accuracy: 0.8693 - loss: 0.3210
Epoch 41/100
810/810             1s 1ms/step -
accuracy: 0.8569 - loss: 0.3405
Epoch 42/100
810/810             1s 762us/step -
accuracy: 0.8746 - loss: 0.3176
Epoch 43/100
810/810             1s 680us/step -
accuracy: 0.8696 - loss: 0.3259
Epoch 44/100
810/810             1s 711us/step -
accuracy: 0.8611 - loss: 0.3351
Epoch 45/100
810/810             1s 867us/step -
accuracy: 0.8675 - loss: 0.3214
Epoch 46/100
810/810             1s 746us/step -
accuracy: 0.8671 - loss: 0.3205
Epoch 47/100
810/810             1s 763us/step -
accuracy: 0.8725 - loss: 0.3174
Epoch 48/100
810/810             1s 695us/step -
accuracy: 0.8725 - loss: 0.3233
Epoch 49/100
810/810             1s 643us/step -
accuracy: 0.8685 - loss: 0.3310
Epoch 50/100
810/810             1s 655us/step -
accuracy: 0.8747 - loss: 0.3152
Epoch 51/100
810/810             1s 849us/step -
accuracy: 0.8647 - loss: 0.3288
Epoch 52/100
810/810             1s 1ms/step -
accuracy: 0.8663 - loss: 0.3190
Epoch 53/100
810/810             1s 926us/step -
accuracy: 0.8729 - loss: 0.3101
Epoch 54/100
810/810             1s 722us/step -
accuracy: 0.8688 - loss: 0.3257
Epoch 55/100
```

```
810/810              1s 812us/step -
accuracy: 0.8653 - loss: 0.3206
Epoch 56/100
810/810              1s 734us/step -
accuracy: 0.8717 - loss: 0.3258
Epoch 57/100
810/810              1s 699us/step -
accuracy: 0.8616 - loss: 0.3270
Epoch 58/100
810/810              1s 817us/step -
accuracy: 0.8643 - loss: 0.3231
Epoch 59/100
810/810              1s 1ms/step -
accuracy: 0.8761 - loss: 0.3093
Epoch 60/100
810/810              1s 1ms/step -
accuracy: 0.8668 - loss: 0.3252
Epoch 61/100
810/810              1s 1ms/step -
accuracy: 0.8637 - loss: 0.3276
Epoch 62/100
810/810              1s 1ms/step -
accuracy: 0.8688 - loss: 0.3241
Epoch 63/100
810/810              1s 1ms/step -
accuracy: 0.8644 - loss: 0.3307
Epoch 64/100
810/810              1s 1ms/step -
accuracy: 0.8658 - loss: 0.3240
Epoch 65/100
810/810              1s 992us/step -
accuracy: 0.8745 - loss: 0.3198
Epoch 66/100
810/810              1s 961us/step -
accuracy: 0.8586 - loss: 0.3365
Epoch 67/100
810/810              1s 944us/step -
accuracy: 0.8682 - loss: 0.3162
Epoch 68/100
810/810              1s 808us/step -
accuracy: 0.8675 - loss: 0.3207
Epoch 69/100
810/810              1s 904us/step -
accuracy: 0.8621 - loss: 0.3342
Epoch 70/100
810/810              1s 930us/step -
accuracy: 0.8586 - loss: 0.3359
Epoch 71/100
```

```
810/810             1s 1ms/step -
accuracy: 0.8706 - loss: 0.3193
Epoch 72/100
810/810             1s 1ms/step -
accuracy: 0.8635 - loss: 0.3239
Epoch 73/100
810/810             1s 885us/step -
accuracy: 0.8756 - loss: 0.3081
Epoch 74/100
810/810             1s 1ms/step -
accuracy: 0.8602 - loss: 0.3381
Epoch 75/100
810/810             1s 1ms/step -
accuracy: 0.8643 - loss: 0.3317
Epoch 76/100
810/810             1s 974us/step -
accuracy: 0.8730 - loss: 0.3187
Epoch 77/100
810/810             1s 987us/step -
accuracy: 0.8662 - loss: 0.3264
Epoch 78/100
810/810             1s 978us/step -
accuracy: 0.8731 - loss: 0.3162
Epoch 79/100
810/810             1s 1ms/step -
accuracy: 0.8751 - loss: 0.3123
Epoch 80/100
810/810             1s 1ms/step -
accuracy: 0.8678 - loss: 0.3264
Epoch 81/100
810/810             1s 939us/step -
accuracy: 0.8638 - loss: 0.3301
Epoch 82/100
810/810             1s 1ms/step -
accuracy: 0.8692 - loss: 0.3203
Epoch 83/100
810/810             1s 1ms/step -
accuracy: 0.8668 - loss: 0.3263
Epoch 84/100
810/810             1s 1ms/step -
accuracy: 0.8653 - loss: 0.3251
Epoch 85/100
810/810             1s 1ms/step -
accuracy: 0.8711 - loss: 0.3169
Epoch 86/100
810/810             1s 1ms/step -
accuracy: 0.8674 - loss: 0.3206
Epoch 87/100
```

```
810/810              1s 971us/step -
accuracy: 0.8783 - loss: 0.3129
Epoch 88/100
810/810              1s 1ms/step -
accuracy: 0.8683 - loss: 0.3225
Epoch 89/100
810/810              1s 1ms/step -
accuracy: 0.8721 - loss: 0.3165
Epoch 90/100
810/810              1s 1ms/step -
accuracy: 0.8647 - loss: 0.3268
Epoch 91/100
810/810              1s 1ms/step -
accuracy: 0.8768 - loss: 0.3166
Epoch 92/100
810/810              1s 1ms/step -
accuracy: 0.8683 - loss: 0.3260
Epoch 93/100
810/810              1s 1ms/step -
accuracy: 0.8661 - loss: 0.3276
Epoch 94/100
810/810              1s 1ms/step -
accuracy: 0.8708 - loss: 0.3194
Epoch 95/100
810/810              1s 1ms/step -
accuracy: 0.8576 - loss: 0.3367
Epoch 96/100
810/810              1s 1ms/step -
accuracy: 0.8719 - loss: 0.3115
Epoch 97/100
810/810              1s 1ms/step -
accuracy: 0.8663 - loss: 0.3268
Epoch 98/100
810/810              1s 1ms/step -
accuracy: 0.8712 - loss: 0.3151
Epoch 99/100
810/810              1s 1ms/step -
accuracy: 0.8613 - loss: 0.3282
Epoch 100/100
810/810              1s 1ms/step -
accuracy: 0.8675 - loss: 0.3260
```

[32]: <keras.src.callbacks.history.History at 0x1d69617b200>

[34]:
```
from sklearn.metrics import confusion_matrix
y_pred_val = model.predict(X_val)  # Replace model with your trained model␣
 ↪variable
```

```
y_pred_val = (y_pred_val > 0.5).astype(int)  # Convert probabilities to binary␣
↪predictions

# Now you can compute the confusion matrix
cm_val = confusion_matrix(y_val, y_pred_val)
print("Confusion Matrix:")
print(cm_val)

# Optional: Print the classification report for more detailed evaluation
print("\nClassification Report:")
print(classification_report(y_val, y_pred_val))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[34], line 2
      1 from sklearn.metrics import confusion_matrix
----> 2 y_pred_val = model.predict(X_val)  # Replace model with your trained␣
↪model variable
      3 y_pred_val = (y_pred_val > 0.5).astype(int)  # Convert probabilities to␣
↪binary predictions
      5 # Now you can compute the confusion matrix

File␣
↪~\AppData\Roaming\Python\Python312\site-packages\keras\src\utils\traceback_utils.
↪py:122, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
    120     # To get the full stack trace, call:
    121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
    123 finally:
    124     del filtered_tb

File ~\AppData\Roaming\Python\Python312\site-packages\optree\ops.py:747, in␣
↪tree_map(func, tree, is_leaf, none_is_leaf, namespace, *rests)
    745 leaves, treespec = _C.flatten(tree, is_leaf, none_is_leaf, namespace)
    746 flat_args = [leaves] + [treespec.flatten_up_to(r) for r in rests]
--> 747 return treespec.unflatten(map(func, *flat_args))

File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\generic.py:
↪6640, in NDFrame.astype(self, dtype, copy, errors)
   6634     results = [
   6635         ser.astype(dtype, copy=copy, errors=errors) for _, ser in self.
↪items()
   6636     ]
   6638 else:
   6639     # else, only a single dtype is given
-> 6640     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
```

```
   6641     res = self._constructor_from_mgr(new_data, axes=new_data.axes)
   6642     return res.__finalize__(self, method="astype")

File␣
 ↪~\AppData\Roaming\Python\Python312\site-packages\pandas\core\internals\managers.
 ↪py:430, in BaseBlockManager.astype(self, dtype, copy, errors)
    427 elif using_copy_on_write():
    428     copy = False
--> 430 return self.apply(
    431     "astype",
    432     dtype=dtype,
    433     copy=copy,
    434     errors=errors,
    435     using_cow=using_copy_on_write(),
    436 )

File␣
 ↪~\AppData\Roaming\Python\Python312\site-packages\pandas\core\internals\managers.
 ↪py:363, in BaseBlockManager.apply(self, f, align_keys, **kwargs)
    361         applied = b.apply(f, **kwargs)
    362     else:
--> 363         applied = getattr(b, f)(**kwargs)
    364     result_blocks = extend_blocks(applied, result_blocks)
    366 out = type(self).from_blocks(result_blocks, self.axes)

File␣
 ↪~\AppData\Roaming\Python\Python312\site-packages\pandas\core\internals\blocks
 ↪py:758, in Block.astype(self, dtype, copy, errors, using_cow, squeeze)
    755         raise ValueError("Can not squeeze with more than one column.")
    756     values = values[0, :]  # type: ignore[call-overload]
--> 758 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
    760 new_values = maybe_coerce_values(new_values)
    762 refs = None

File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\dtypes\astype
 ↪py:237, in astype_array_safe(values, dtype, copy, errors)
    234     dtype = dtype.numpy_dtype
    236 try:
--> 237     new_values = astype_array(values, dtype, copy=copy)
    238 except (ValueError, TypeError):
    239     # e.g. _astype_nansafe can fail on object-dtype of strings
    240     #  trying to convert to float
    241     if errors == "ignore":

File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\dtypes\astype
 ↪py:182, in astype_array(values, dtype, copy)
    179     values = values.astype(dtype, copy=copy)
    181 else:
```

```
--> 182     values = _astype_nansafe(values, dtype, copy=copy)
    184 # in pandas we don't store numpy str dtypes, so convert to object
    185 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\dtypes\astype
  ↪py:133, in _astype_nansafe(arr, dtype, copy, skipna)
    129     raise ValueError(msg)
    131 if copy or arr.dtype == object or dtype == object:
    132     # Explicit copy, or required since NumPy can't view from / to object.
--> 133     return arr.astype(dtype, copy=True)
    135 return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: 'Kerr'
```

[ ]:
```
Accuracy=782/900
print("Accuracy of the Model on the Validation Data set is 86.89%")
```

[ ]:
```
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))
```

[ ]:
```

```

exp-ml-4

October 14, 2024

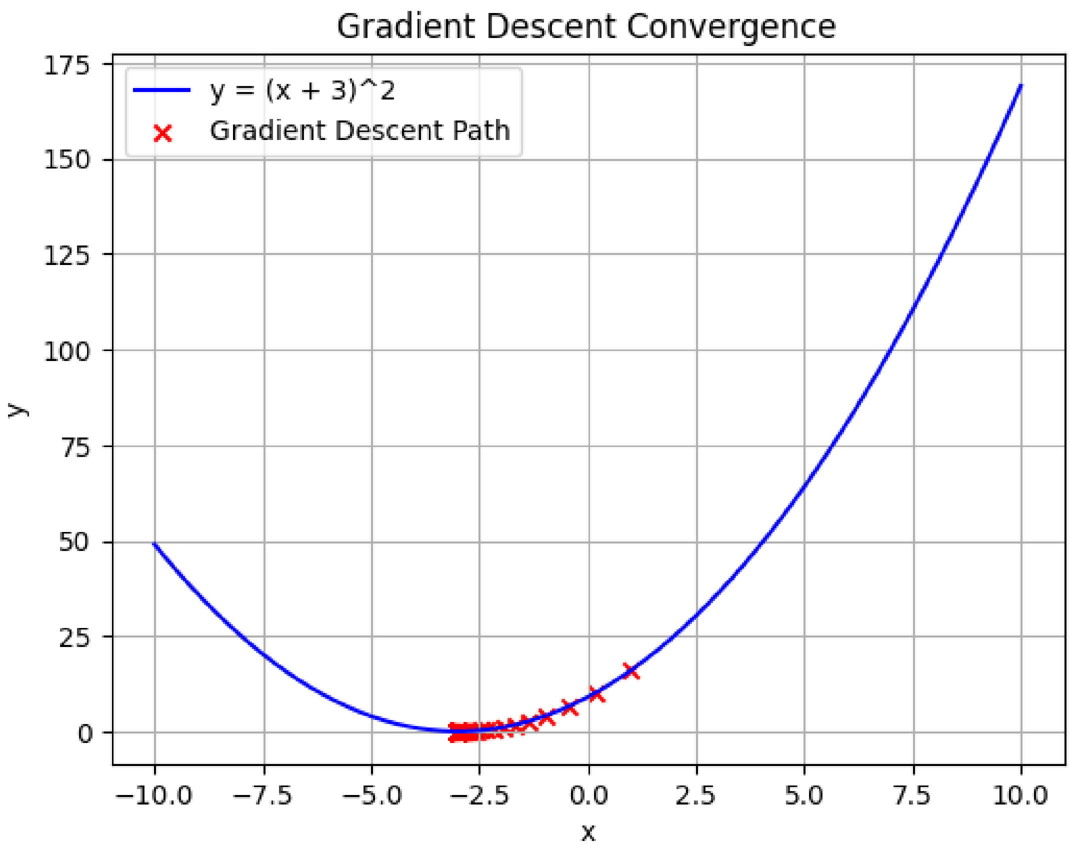
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
def gradient_descent(learning_rate, max_iterations, initial_x):
    x = initial_x
    x_history = []  # Create a list to store the history of x values
    for _ in range(max_iterations):
        gradient = 2 * (x + 3)  # Compute the gradient of the function
        x = x - learning_rate * gradient  # Update x using the gradient and␣
↪learning rate
        x_history.append(x)  # Append the current x to the history list

    return x, x_history
```

[5]:
```
# Parameters for Gradient Descent
learning_rate = 0.1
max_iterations = 1000
initial_x = 2

# Run Gradient Descent to find the local minimum
local_minimum, x_history = gradient_descent(learning_rate, max_iterations,␣
↪initial_x)

print(f"Local Minimum at x = {local_minimum}")

# Plot the graph to visualize the convergence
x_values = np.linspace(-10, 10, 400)  # Generate x values for the graph
y_values = (x_values + 3)**2  # Calculate corresponding y values

plt.plot(x_values, y_values, label='y = (x + 3)^2', color='blue')
plt.scatter(x_history, [(x + 3)**2 for x in x_history], label='Gradient Descent␣
↪Path', color='red', marker='x')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Gradient Descent Convergence')
plt.grid(True)
```

```
plt.show()
```

```
Local Minimum at x = -2.9999999999999999
```




```
[ ]:
```

exp-ml-5

October 14, 2024

[1]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score
```

[2]:
```python
data=pd.read_csv("diabetes.csv")
data
```

[2]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …              …              …      …     …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     Pedigree  Age  Outcome
0       0.627   50        1
1       0.351   31        0
2       0.672   32        1
3       0.167   21        0
4       2.288   33        1
..        …    …        …
763     0.171   63        0
764     0.340   27        0
765     0.245   30        0
766     0.349   47        1
767     0.315   23        0
```

```
[768 rows x 9 columns]
```

[3]:
```
X = data.drop("Outcome", axis=1)  # Features
y = data["Outcome"]  # Target variable
```

[4]:
```
X
```

[4]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …       …            …            …      …   …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     Pedigree  Age
0       0.627   50
1       0.351   31
2       0.672   32
3       0.167   21
4       2.288   33
..        …  …
763     0.171   63
764     0.340   27
765     0.245   30
766     0.349   47
767     0.315   23

[768 rows x 8 columns]
```

[5]:
```
# 2. Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[6]:
```
# 3. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[7]:
```
X_train
```

[7]:
```
array([[-0.52639686, -1.15139792, -3.75268255, …, -4.13525578,
        -0.49073479, -1.03594038],
       [ 1.58804586, -0.27664283,  0.68034485, …, -0.48916881,
         2.41502991,  1.48710085],
       [-0.82846011,  0.56687102, -1.2658623 , …, -0.42452187,
         0.54916055, -0.94893896],
       …,
       [ 1.8901091 , -0.62029661,  0.89659009, …,  1.76054443,
         1.981245  ,  0.44308379],
       [-1.13052335,  0.62935353, -3.75268255, …,  1.34680407,
        -0.78487662, -0.33992901],
       [-1.13052335,  0.12949347,  1.43720319, …, -1.22614383,
        -0.61552223, -1.03594038]])
```

[8]:
```python
# 4. Implement K-Nearest Neighbors (KNN)
k = 3  # Choose the number of neighbors (k) based on your needs
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```

[8]:
```
KNeighborsClassifier(n_neighbors=3)
```

[11]:
```python
# 5. Predict and Evaluate
y_pred = knn.predict(X_test)
y_pred
```

[11]:
```
array([0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
      dtype=int64)
```

[12]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score

# Assuming y_test and y_pred are already defined
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculate accuracy, error rate, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print results
```

```python
print("Confusion Matrix:")
print(conf_matrix)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
[[81 18]
 [27 28]]
Accuracy: 0.7077922077922078
Error Rate: 0.29220779220779225
Precision: 0.6086956521739131
Recall: 0.509090909090909
```