Simple Logistic Regression

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

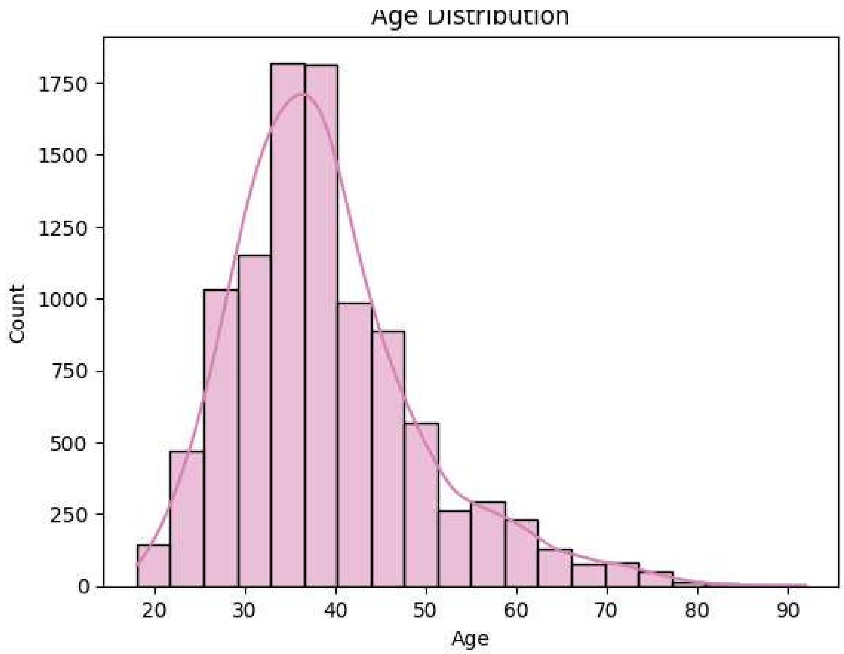
```
data = pd.read_csv('/content/drive/MyDrive/ITC Projects/Project_4/Churn_Modelling.csv')
data.head()
print(data.columns)
print(data.info())
print(data.duplicated().sum())
data = data.drop(['RowNumber', 'CustomerId', 'Surname'], axis = 1)
print(data.shape)
```

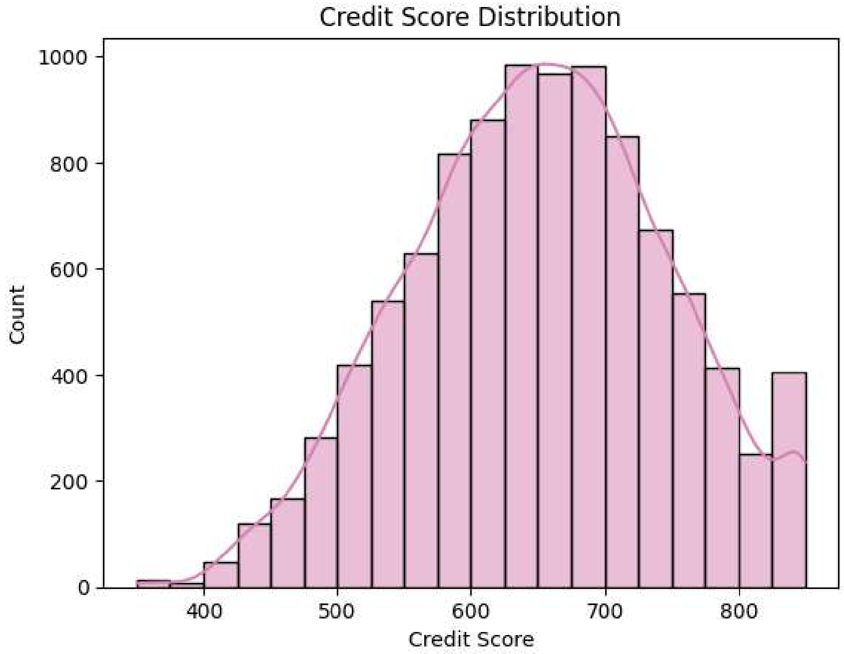
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
None
0
(10000, 11)
```

Exploratory Data Analysis

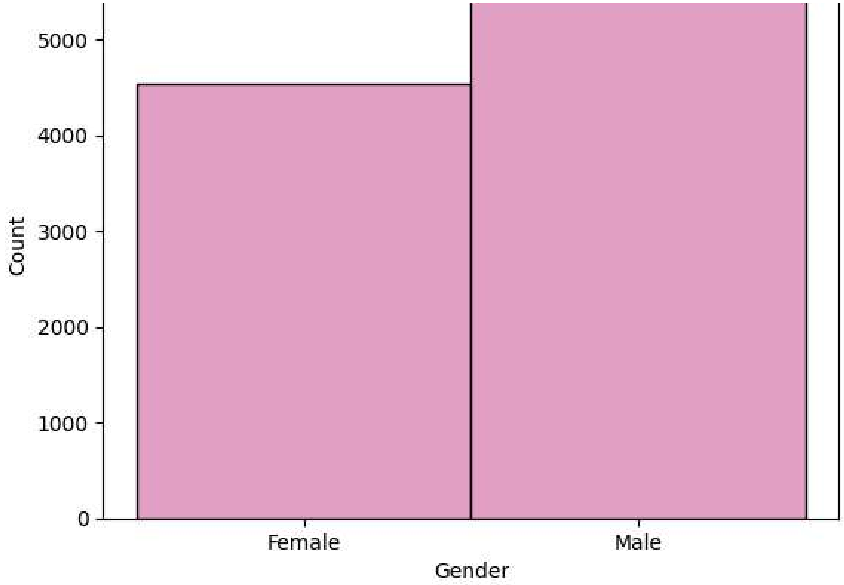
Univariate Analysis

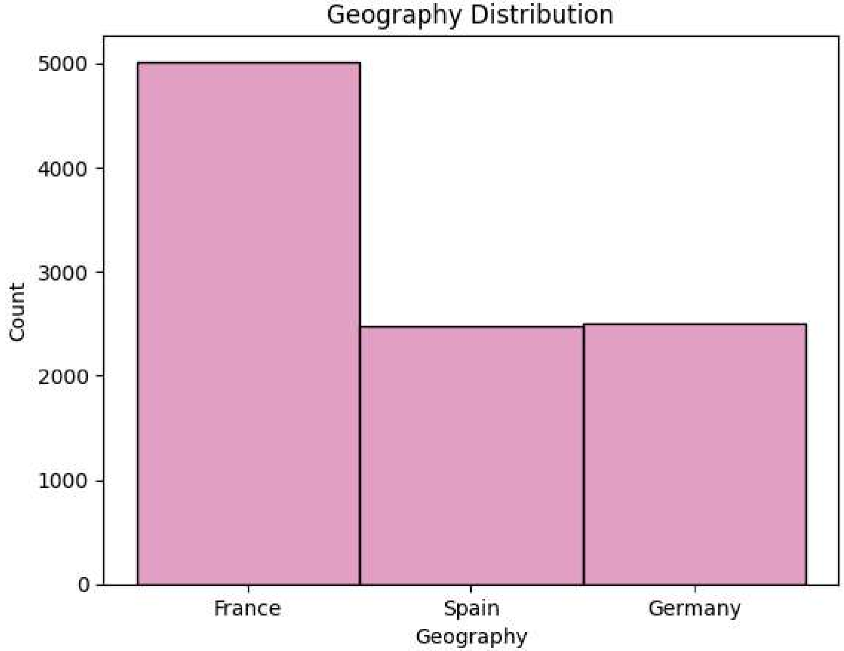
```
import seaborn as sns
sns.histplot(data['Age'], bins = 20, color = '#D982B5', kde = True)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Age Distribution')
plt.show()

sns.histplot(data['CreditScore'], bins = 20, color = '#D982B5', kde = True)
plt.xlabel('Credit Score')
plt.ylabel('Count')
plt.title('Credit Score Distribution')
plt.show()

sns.histplot(data['Gender'], bins = 2, color = '#D982B5')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Distribution')
plt.show()

sns.histplot(data['Geography'], bins = 2, color = '#D982B5')
plt.xlabel('Geography')
plt.ylabel('Count')
plt.title('Geography Distribution')
plt.show()

avg_age_exited = data.groupby('Exited')['Age'].mean()
plt.bar(avg_age_exited.index, avg_age_exited.values, color = '#D982B5')
plt.xlabel('Exited (0 = No, 1 = Yes)')
plt.ylabel('Average Age')
plt.title('Average Age of Customers per Exit Status')
plt.xticks([0, 1], ['0', '1'])
plt.show()

target = data['Exited'].value_counts()
fig1, ax1 = plt.subplots()
ax1.pie(data['Exited'].value_counts(), labels=target.index, autopct='%1.1f%%', colors = ['#D982B5', '#FF9999'], shadow=None)
plt.show()
```

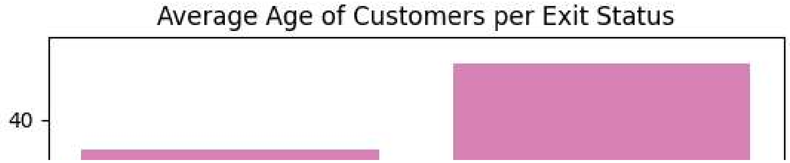

Age Distribution
Count
Age
0
250
500
750
1000
1250
1500
1750
20
30
40
50
60
70
80
90

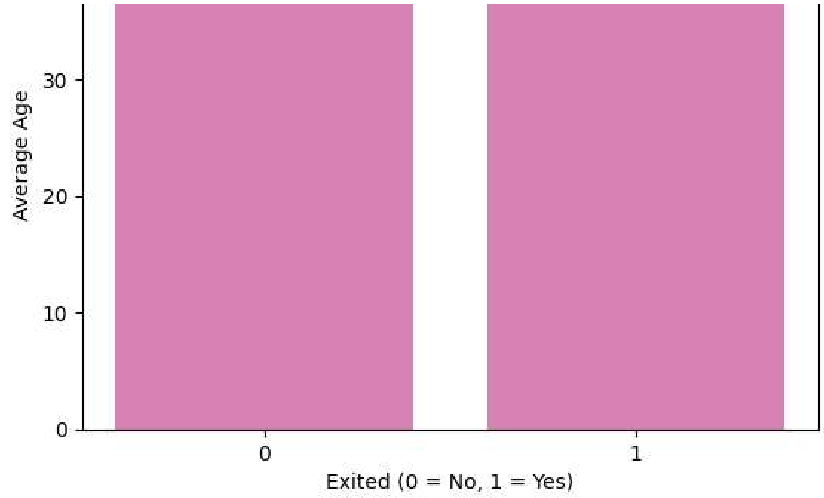

Credit Score Distribution
Count
Credit Score
0
200
400
600
800
1000
400
500
600
700
800

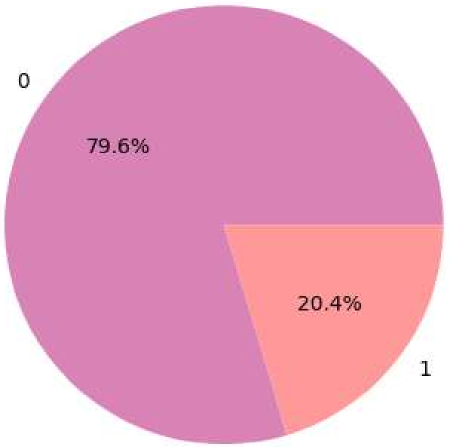

Gender Distribution


5000
4000
3000
2000
1000
0
Count
Female
Male
Gender


Geography Distribution
5000
4000
3000
2000
1000
0
Count
France
Spain
Germany
Geography


Average Age of Customers per Exit Status
40


Average Age
30
20
10
0
0
1
Exited (0 = No, 1 = Yes)


0
79.6%
20.4%
1

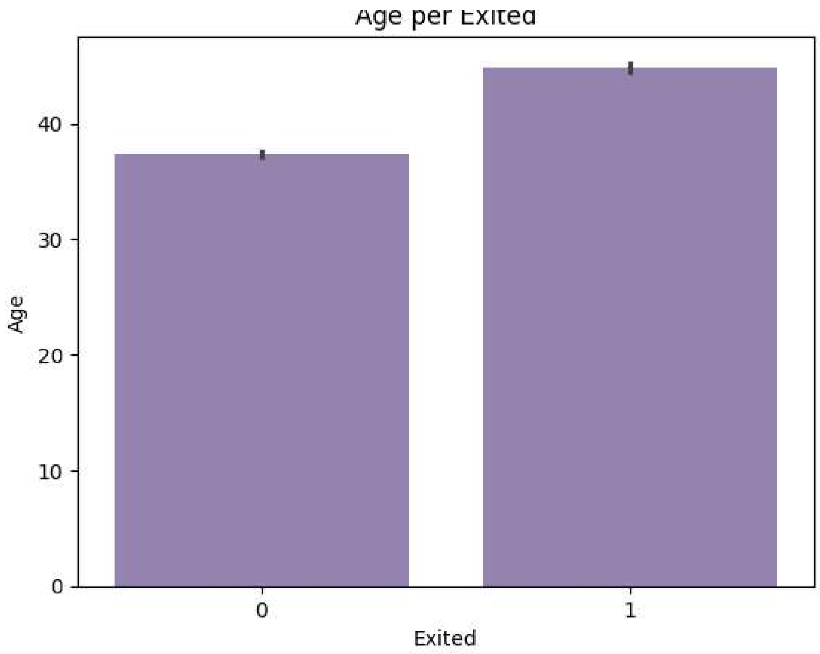
```
Count_Exited = data['Exited'].value_counts()
Count_Gender = data['Gender'].value_counts()
Count_Geography = data['Geography'].value_counts()

print(Count_Exited)
print(Count_Gender)
print(Count_Geography)
```

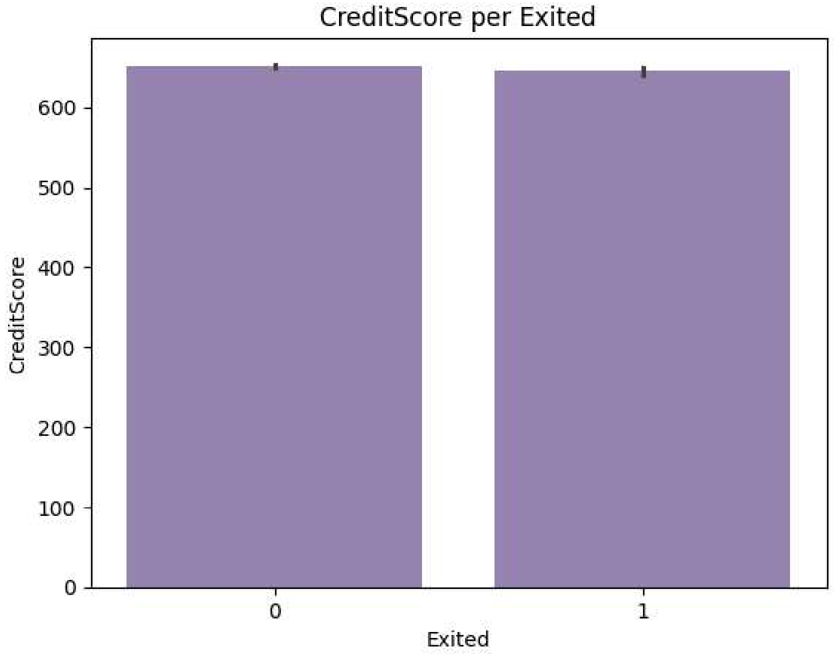
```
Exited
0    7963
1    2037
Name: count, dtype: int64
Gender
Male      5457
Female    4543
Name: count, dtype: int64
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```

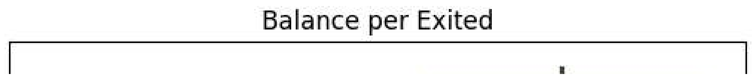
Bivariate Analysis

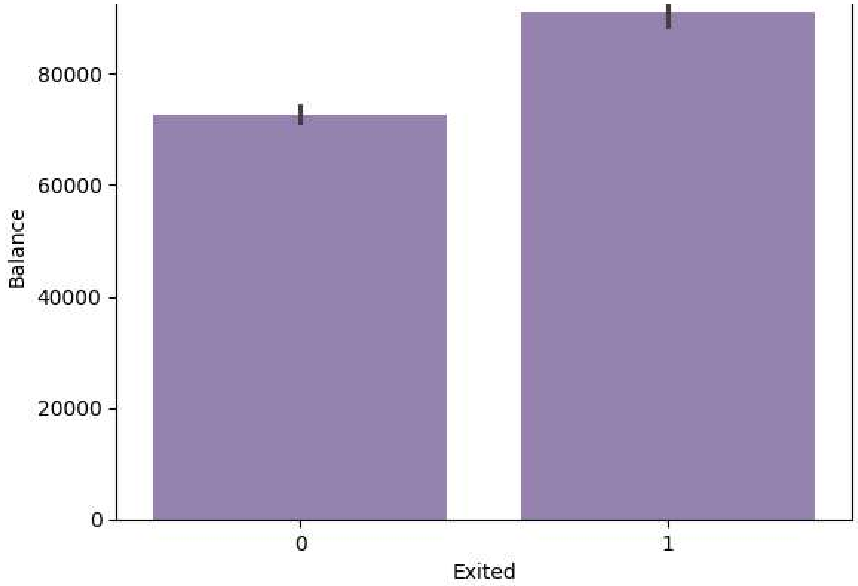
```
sns.barplot(x = 'Exited', y = 'Age', data = data,  color = '#967BB6')
plt.xlabel('Exited')
plt.ylabel('Age')
plt.title('Age per Exited')
plt.show()

sns.barplot(x = 'Exited', y = 'CreditScore', data = data, color = '#967BB6')
plt.xlabel('Exited')
plt.ylabel('CreditScore')
plt.title('CreditScore per Exited')
plt.show()

sns.barplot(x = 'Exited', y = 'Balance', data = data, color = '#967BB6')
plt.xlabel('Exited')
plt.ylabel('Balance')
plt.title('Balance per Exited')
plt.show()

sns.barplot(x = 'Exited', y = 'NumOfProducts', data = data, color = '#967BB6')
plt.xlabel('Exited')
plt.ylabel('Number of Products')
plt.title('Number of Products per Exited')
plt.show()

sns.barplot(x = 'Exited', y = 'EstimatedSalary', data = data, color = '#967BB6')
```

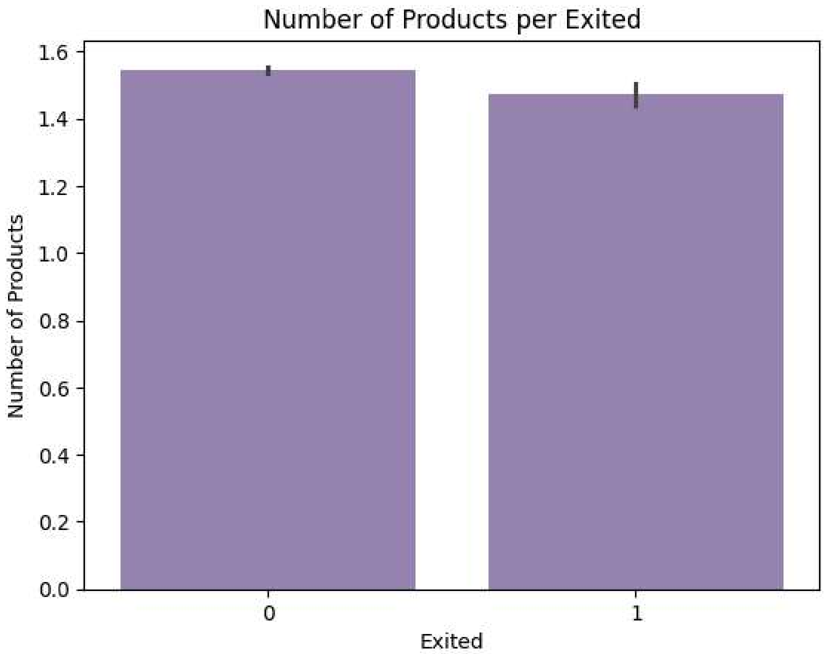
```
plt.xlabel('Exited')
plt.ylabel('Estimated Salary')
plt.title('Estimated Salary per Exited')
plt.show()
```


Age per Exited
Age
Exited
0
1
0
10
20
30
40

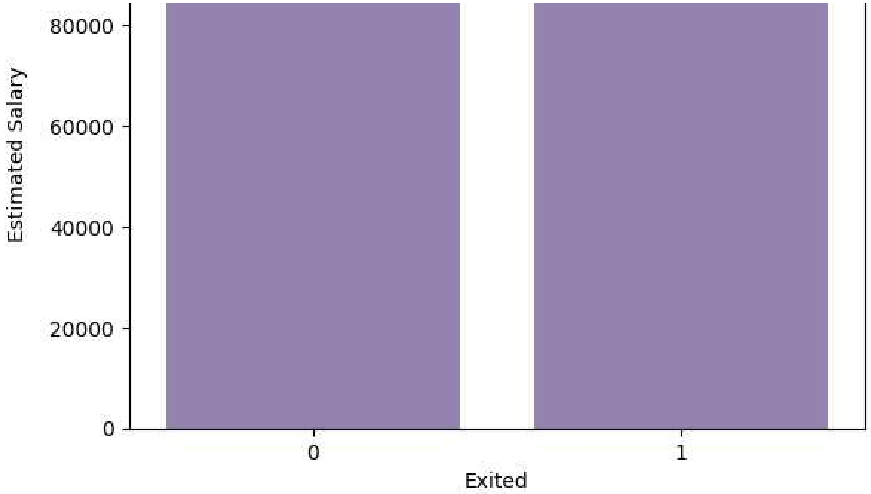

CreditScore per Exited
CreditScore
Exited
0
1
0
100
200
300
400
500
600


Balance per Exited


80000
60000
40000
20000
0
Balance
0
1
Exited


Number of Products per Exited
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Number of Products
0
1
Exited


Estimated Salary per Exited
100000

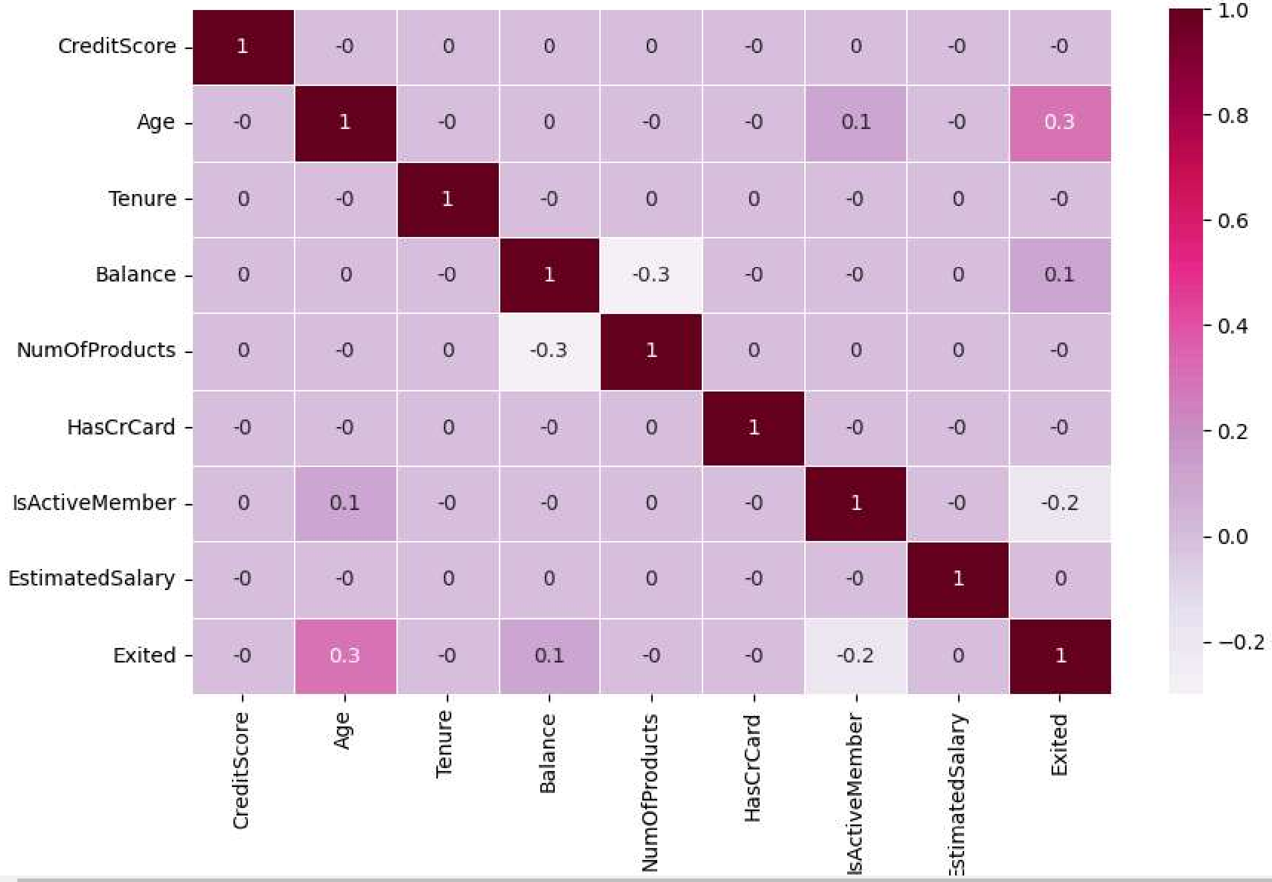

80000
60000
40000
20000
0
Estimated Salary
0
1
Exited

```
Avg_creditScore_by_Exited = data.groupby('Exited')['CreditScore'].mean()
Avg_Age_by_Exited = data.groupby('Exited')['Age'].mean()
Avg_Balance_by_Exited = data.groupby('Exited')['Balance'].mean()
Avg_NumOfProducts_by_Exited = data.groupby('Exited')['NumOfProducts'].mean()
Avg_EstimatedSalary_by_Exited = data.groupby('Exited')['EstimatedSalary'].mean()

print(Avg_creditScore_by_Exited)
print(Avg_Age_by_Exited)
print(Avg_Balance_by_Exited)
print(Avg_NumOfProducts_by_Exited)
print(Avg_EstimatedSalary_by_Exited)
```

```
Exited
0    651.853196
1    645.351497
Name: CreditScore, dtype: float64
Exited
0    37.408389
1    44.837997
Name: Age, dtype: float64
Exited
0    72745.296779
1    91108.539337
Name: Balance, dtype: float64
Exited
0    1.544267
1    1.475209
Name: NumOfProducts, dtype: float64
Exited
0     99738.391772
1    101465.677531
Name: EstimatedSalary, dtype: float64
```

```
data1 = data[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Exited']]
plt.figure(figsize = (10,6))
sns.heatmap(round(data1.corr(),1), cmap="PuRd", annot =True, linewidths= 0.5)
```

<Axes: >




```
Geography_Encoded= pd.get_dummies(data['Geography'],  dtype = int)
Gender_Encoded= pd.get_dummies(data['Gender'], dtype = int)

df = pd.concat([data, Geography_Encoded], axis = 1)
df = pd.concat([df, Gender_Encoded], axis = 1)

df = df.drop(['Geography', 'Gender'], axis = 1)
df.head()
```

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

Next steps: View recommended plots New interactive sheet

```
#MinMaxScaling
data_scaled = (df - df.min())/(df.max()-df.min())
data_scaled
```

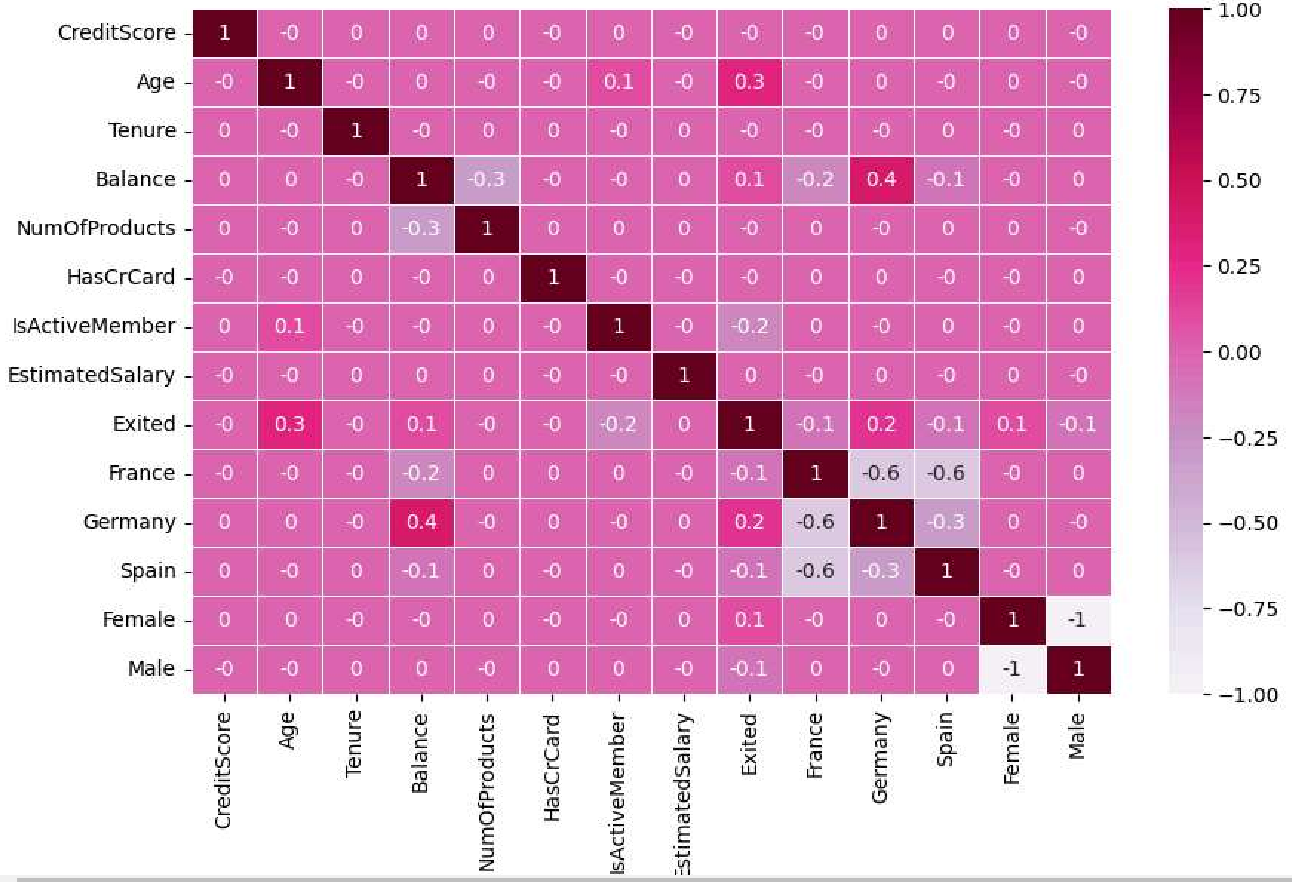
	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	France	Germany	Spain	Female	Male
0	0.538	0.324324	0.2	0.000000	0.000000	1.0	1.0	0.506735	1.0	1.0	0.0	0.0	1.0	0.0
1	0.516	0.310811	0.1	0.334031	0.000000	0.0	1.0	0.562709	0.0	0.0	0.0	1.0	1.0	0.0
2	0.304	0.324324	0.8	0.636357	0.666667	1.0	0.0	0.569654	1.0	1.0	0.0	0.0	1.0	0.0
3	0.698	0.283784	0.1	0.000000	0.333333	0.0	0.0	0.469120	0.0	1.0	0.0	0.0	1.0	0.0
4	1.000	0.337838	0.2	0.500246	0.000000	1.0	1.0	0.395400	0.0	0.0	0.0	1.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0.842	0.283784	0.5	0.000000	0.333333	1.0	0.0	0.481341	0.0	1.0	0.0	0.0	0.0	1.0
9996	0.332	0.229730	1.0	0.228657	0.000000	1.0	1.0	0.508490	0.0	1.0	0.0	0.0	0.0	1.0
9997	0.718	0.243243	0.7	0.000000	0.000000	0.0	1.0	0.210390	1.0	1.0	0.0	0.0	1.0	0.0
9998	0.844	0.324324	0.3	0.299226	0.333333	1.0	0.0	0.464429	1.0	0.0	1.0	0.0	0.0	1.0
9999	0.884	0.135135	0.4	0.518708	0.000000	1.0	0.0	0.190914	0.0	1.0	0.0	0.0	1.0	0.0

10000 rows × 14 columns

Next steps: View recommended plots New interactive sheet

```
plt.figure(figsize = (10,6))
sns.heatmap(round(data_scaled.corr(),1), cmap = "PuRd", annot =True, linewidths= 0.5)
```

<Axes: >



```
data_scaled = data_scaled.drop(['France', 'Male'], axis = 1)
```

```
data_scaled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      10000 non-null  float64
 1   Age              10000 non-null  float64
 2   Tenure           10000 non-null  float64
 3   Balance          10000 non-null  float64
 4   NumOfProducts    10000 non-null  float64
 5   HasCrCard        10000 non-null  float64
 6   IsActiveMember   10000 non-null  float64
```

```
 7   EstimatedSalary  10000 non-null  float64
 8   Exited           10000 non-null  float64
 9   Germany          10000 non-null  float64
 10  Spain            10000 non-null  float64
 11  Female           10000 non-null  float64
dtypes: float64(12)
memory usage: 937.6 KB
```

Simple Logistic Regression

Useful Functions

```
def sigmoid(h):
  return 1/(1+np.exp(-h))

def cross_entropy(y,p_hat):
  return -(1/len(y))*np.sum(y*np.log(p_hat)+(1-y)*np.log(1-p_hat))

def accuracy (y,y_hat):
  return np.mean(y==y_hat)
```

Logistic Regression Class

```
class LogisticRegression():
  def __init__(self, thresh = 0.5):
     # thresh is hyperparameter
     self.thresh = thresh
     self.W = None
     self.b = None

  def fit(self, X, y, eta = 1e-3, epochs = 1e3, show_curve = False):
    epochs = int(epochs)
    N, D = X.shape

    #Initialize Weights and Biases
    self.W = np.random.randn(D)
    self.b = np.random.randn(1)

    J = np.zeros(epochs)

    #SGD
    for epoch in range(epochs):
      p_hat = self.__forward__(X)
      J[epoch] = cross_entropy(y, p_hat)
```

```
      #Weight Update Rules
      self.W -= eta*(1/N)*X.T@(p_hat-y)
      self.b -= eta*(1/N)*np.sum(p_hat-y)

    if show_curve:
      plt.figure()
      plt.plot(J)
      plt.xlabel('epochs')
      plt.ylabel('$\mathcal{J}$')
      plt.title("Training Curve")

  def __forward__ (self, X):
    return sigmoid(X@self.W +self.b)

  def predict(self, X):
    return (self.__forward__(X) >= self.thresh).astype(np.int32)
```

Train Test Split

```
#With Scaling

train_ratio = 0.8

train_data_log_reg = data_scaled.sample(frac=train_ratio, random_state=42)
test_data_log_reg = data_scaled.drop(train_data_log_reg.index)

print(train_data_log_reg.shape)
print(test_data_log_reg.shape)



y_train = train_data_log_reg['Exited']
X_train = train_data_log_reg.drop(['Exited'], axis = 1)
y_test = test_data_log_reg['Exited']
X_test = test_data_log_reg.drop(['Exited'], axis = 1)


print(y_train.shape)
print(X_train.shape)
print(y_test.shape)
print(X_test.shape)


# converting X_train, X_test, y_train, y_test to NumPy arrays
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
```

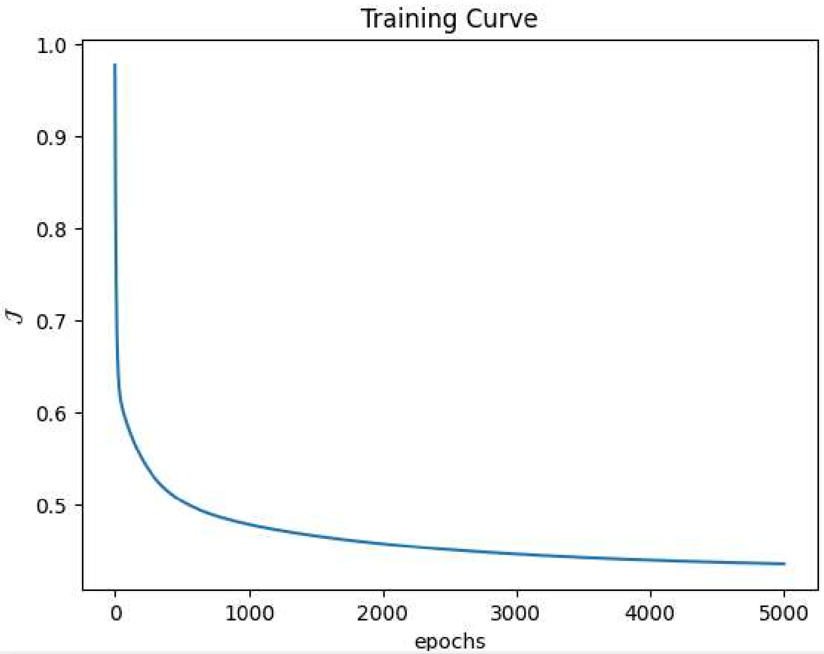
```
y_test = np.array(y_test)
y_train = y_train.astype(int)
```

```
(8000, 12)
(2000, 12)
(8000,)
(8000, 11)
(2000,)
(2000, 11)
```

Implementation of Binary Logistic Regression

```
LR = LogisticRegression(thresh = 0.51)
LR.fit(X_train,y_train, eta=9e-2, epochs=5e3,show_curve=True)

y_train_hat = LR.predict(X_train)

print(f"Training Accuracy: {accuracy(y_train, y_train_hat): 0.4f}")
print(LR.W, LR.b)
```

```
Training Accuracy:  0.8114
[-0.58028663  3.37011577 -0.17381636  0.51848868 -0.43894914 -0.13440265
 -0.92612008 -0.02721149  0.73951948 -0.00691515  0.48093174] [-1.95364916]
```

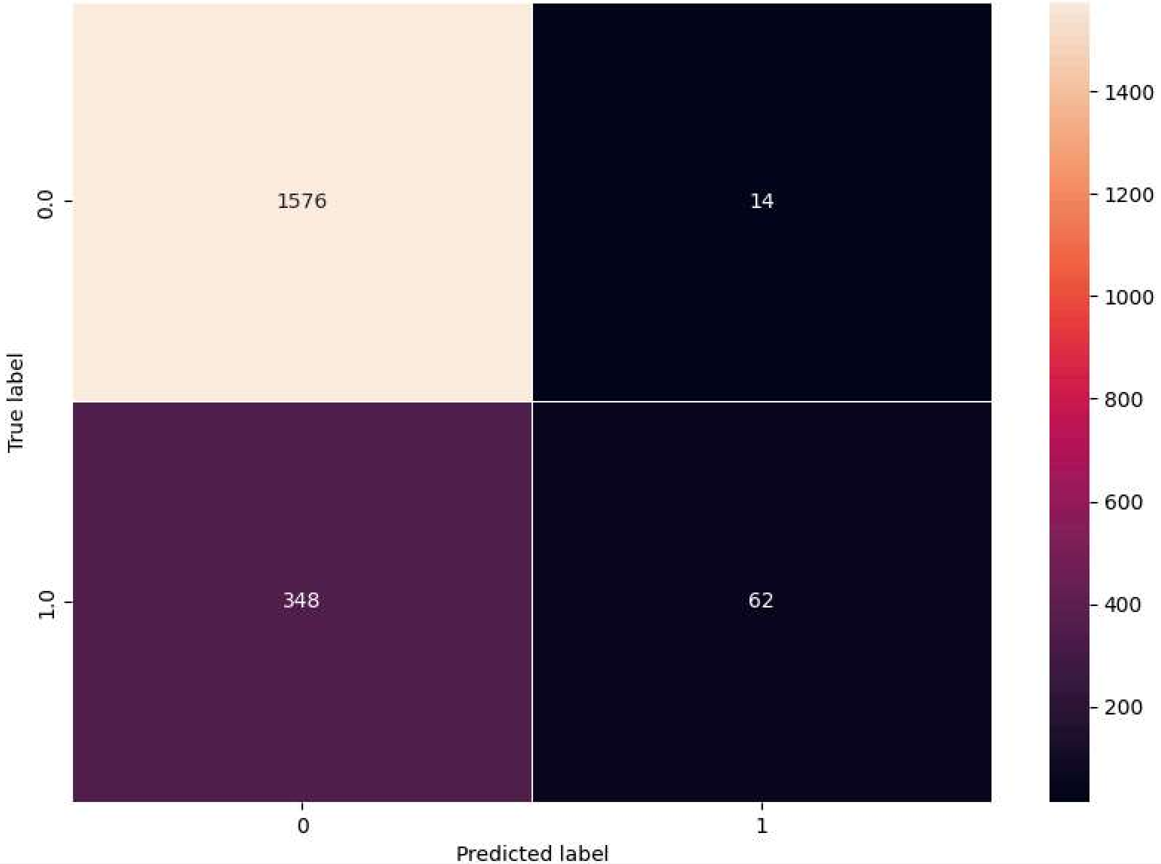


```
y_test_hat = LR.predict(X_test)
print(f"Test Accuracy: {accuracy(y_test, y_test_hat): 0.4f}")
```

```
Test Accuracy:  0.8190
```

```
plt.figure(figsize=(10,7))
y_actu = pd.Series(y_test, name='Actual')
y_pred = pd.Series(y_test_hat, name='Predicted')
cm = pd.crosstab(y_actu, y_pred)
ax = sns.heatmap(cm, annot=True, fmt="d", linewidths= .5)
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 47.7222222222222, 'Predicted label')
```



Artificial Neural Network with Variable Architecture

Useful Activation Functions

```
# Activations

def linear(H):
  return H

def ReLU(H):
  return H*(H>0)


def softmax(H):
  eH=np.exp(H)
  return eH/eH.sum(axis=1, keepdims=True)


#Misc

def one_hot(y):
  N=len(y)
  K=len(set(y))
  Y = np.zeros((N,K))

  for i in range(N):
    Y[i,y[i]]=1

  return Y
```

Useful Loss Functions

```
#Loss Functions

def cross_entropy(Y, P_hat):
  return -(1/len(Y))*np.sum(Y*np.log(P_hat))

def OLS(Y, Y_hat):
  return (1/(2*len(Y)))*np.sum((Y-Y_hat)**2)
```

Derivatives

```
def derivative(Z, a):

  if a == linear:
    return 1
  elif a == sigmoid:
    return Z*(1-Z)
  elif a==np.tanh:
    return 1-Z*Z
  elif a==ReLU:
    return (Z>0).astype(int)

  else:
    ValueError("Unknown Activation")
```

Useful Metrics

```
def accuracy(y,y_hat):
  return np.mean(y==y_hat)

def R2(y,y_hat):
  return 1-np.sum((y-y_hat)**2)/np.sum((y - y.mean())**2)
```

ANN Class

```
class ANN():

  def __init__(self, architecture, activations=None, mode=0): #architecture--No. of Neurons in the hidden layers
    self.mode=mode #mode: 0 → Classification (default, uses softmax in the output layer). 1 → Regression (uses linear activation in the output layer).
    self.architecture = architecture
    self.activations = activations
    self.L = len(architecture)+1            #self.L: Total number of layers = hidden layers + output layer.

  def fit (self, X, y, eta=1e-3, epochs=1e3,show_curve=False):

    epochs= int(epochs)
    if self.mode:
      Y=y
      K=1
    else:
      Y = one_hot(y)
```