ASSIGNMENT 2

Project Name: Intelligent Vehicle Damage Assessment & Cost Estimator for Insurance Companies

Name : Barath Raj R.S

Roll Number : 720719104034

Download the dataset

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

2.Load the Data Set

```
dataset = pd.read_csv('/content/Churn_Modelling.csv')
```

```
#dropping row number columns as we already have index column by default
dataset.drop(['RowNumber'], axis=1,inplace=True)
```

```
dataset
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1
2	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0
3	15701354	Boni	699	France	Female	39	1	0.00	2	0	0
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
995	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0
996	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1
997	15584532	Liu	709	France	Female	36	7	0.00	1	0	1
998	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0
999	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0

000 rows × 13 columns

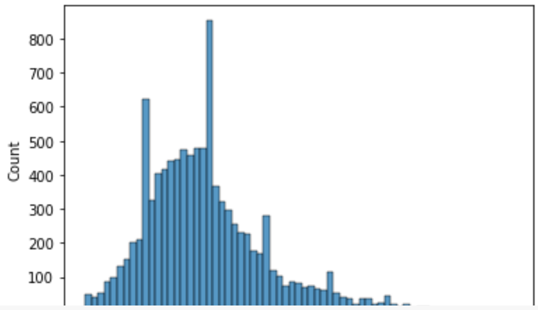
3 . Visualizations

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

Univariate Analysis

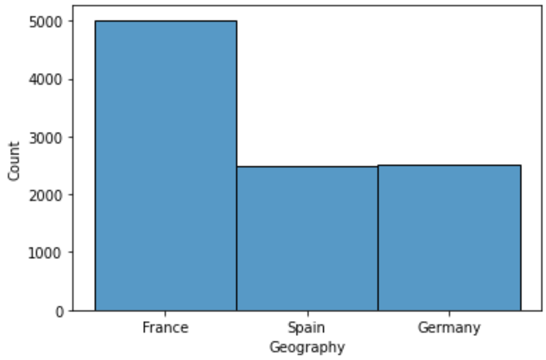
```
# plt.scatter(churn.index,churn["Age"])
# plt.show()

# Age Histogram
sns.histplot(x='Age', data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff375466990>
```



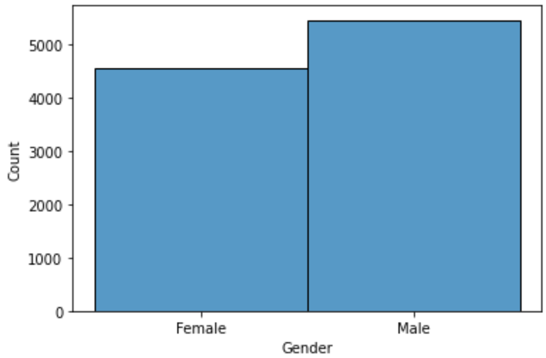

```
# Geography Histogram
sns.histplot(x='Geography', data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff3752b0e50>
```



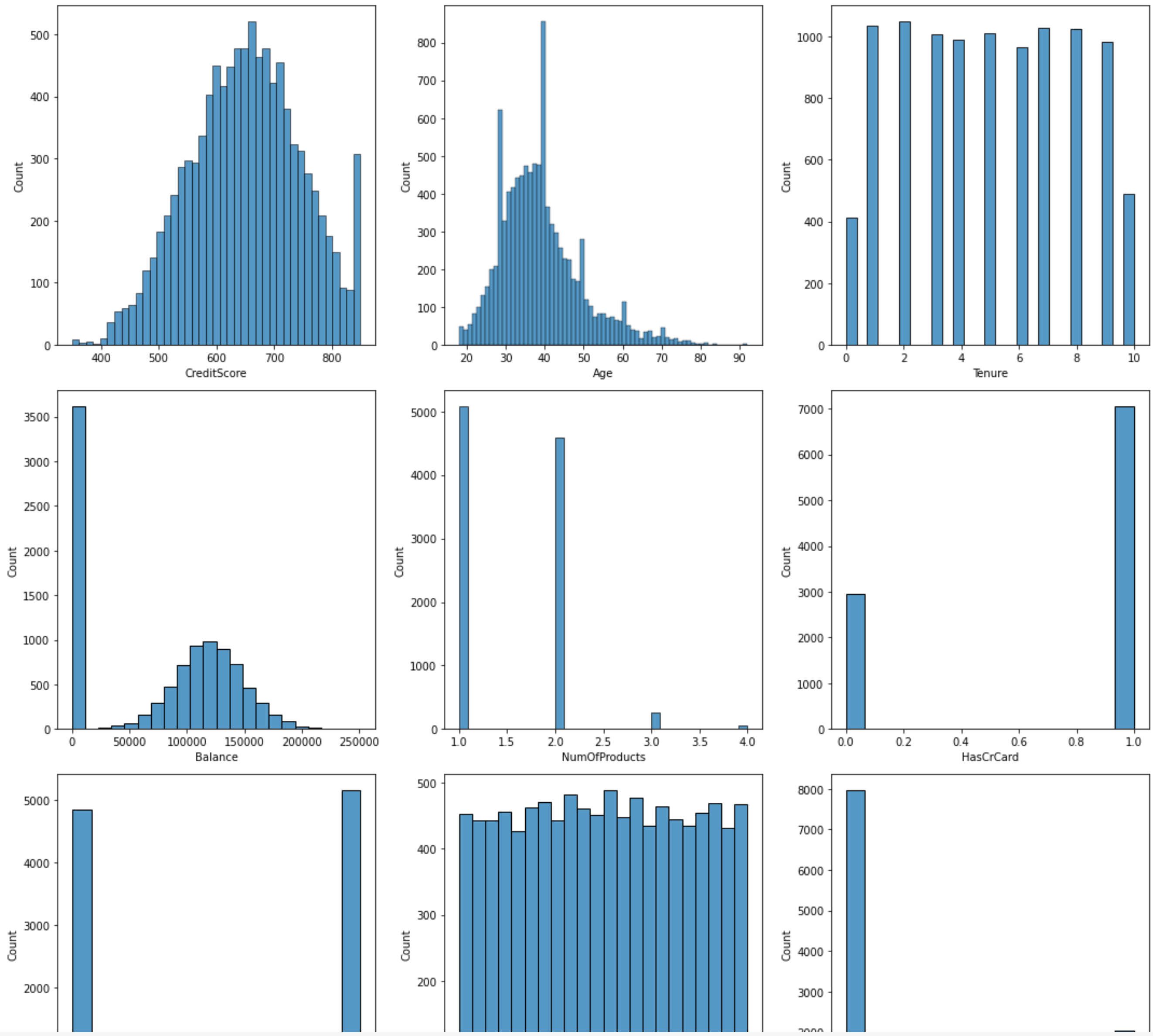

```
# Geography Histogram
sns.histplot(x='Gender', data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff3752c2d10>
```



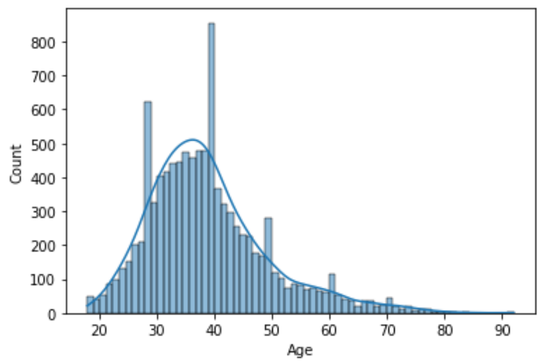

```
cols = 3
rows = 3
num_cols = dataset.select_dtypes(exclude='object').columns #exclude string based columns namely Surname, Geography, Gender
print(num_cols)
fig = plt.figure(figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols[1:]): #exclude Customer ID

    ax=fig.add_subplot(rows,cols,i+1)

    sns.histplot(x = dataset[col], ax = ax)

fig.tight_layout()
plt.show()
```

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



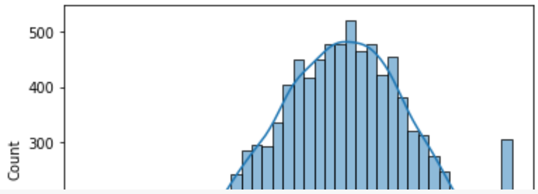

```
# sns.kdeplot(x='Age', data=churn, hue='Exited')
sns.histplot(x='Age', data=dataset, kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff374dbf790>
```



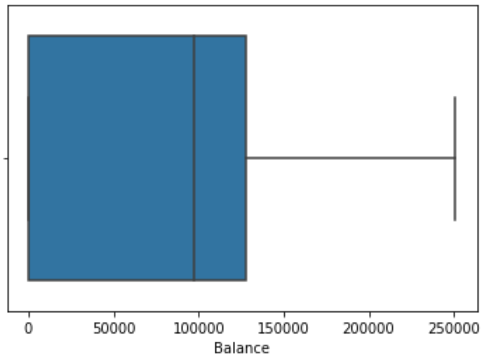

```
# sns.kdeplot(x='Age', data=churn, hue='IsActiveMember')
sns.histplot(x='CreditScore', data=dataset, kde=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff3752b6690>



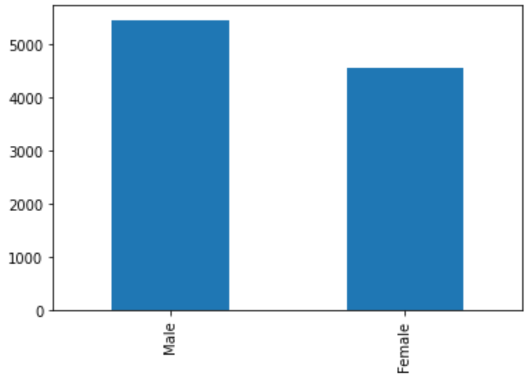

```
sns.boxplot(x=dataset['Balance'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff371cf2e90>



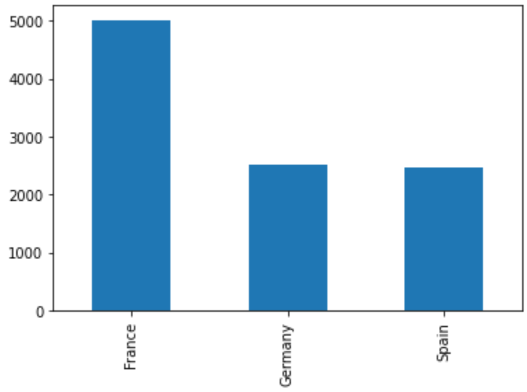

```
dataset['Gender'].value_counts().plot.bar()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff371c2c190>




```
dataset['Geography'].value_counts().plot.bar()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff371cd1f90>



Bi - Variate Analysis

```
g = sns.pairplot(dataset, diag_kind="kde", markers="+",
                 plot_kws=dict(s=50, edgecolor="b", linewidth=1),
                 diag_kws=dict(shade=True))
```

```
import matplotlib.pyplot as plt

#create scatterplot of hours vs. score
```

```
plt.scatter(dataset.Age[:30], dataset.Balance[:30])
plt.title('Age vs. Balance')
plt.xlabel('Age')
plt.ylabel('Balance')
```

```
dataset.plot.hexbin(x='Age', y='CreditScore', gridsize=10)
```

Multi-variate Analysis

```
dataset.corr()
```

```
sns.set(font_scale=0.50)
plt.figure(figsize=(8,4))
sns.heatmap(dataset.corr(),cmap='RdBu_r', annot=True, vmin=-1, vmax=1)
```

```
#Three variables - Multivaraiate
sns.barplot(x='Age', y='Geography', data=dataset, palette='bright',hue='Gender')
```

4 . Descriptive statistics

```
import statistics as st
```

```
dataset[['Age', 'Balance', 'EstimatedSalary']].mean()
```

```
dataset.info()
```

```
dataset.describe()
```

```
dataset['Age'].median()
```

```
standard_deviation = dataset['CreditScore'].std()
print(standard_deviation)
```

```
st.mode(dataset['Geography'])
```

```
st.median(dataset['Age'])
```

```
st.variance(dataset['CreditScore'])
```

5 . Handle Missing Values

```
dataset.isnull().sum() #no missing values
```

6 . Find and replace outliers

```
sns.boxplot(dataset['CreditScore'],data=dataset)
```

```
dataset['Balance'].hist()
```

```
for col in num_cols[1:]:
  print('skewness value of ',col,dataset[col].skew())

#Skewness should be in the range of -1 to 1, any columns with skewness outside of that range would have outliers
```

```
Q1=dataset['Age'].quantile(0.25)
```

```
Q3=dataset['Age'].quantile(0.75)
IQR=Q3-Q1
```

```
IQR
```

```
#Values above than the upper bound and below than the lower bound are considered outliers

upper = dataset['Age'] >= (Q3+1.5*IQR)

# print("Upper bound:",upper)
print(np.where(upper))

lower = dataset['Age'] <= (Q1-1.5*IQR)
# print("Lower bound:", lower)
print(np.where(lower))
```

```
#Removing outliers based off Age column

# IQR
Q1 = np.percentile(dataset['Age'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(dataset['Age'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", dataset.shape)

# Upper bound
upper = np.where(dataset['Age'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(dataset['Age'] <= (Q1-1.5*IQR))

''' Removing the Outliers '''
dataset.drop(upper[0], inplace = True)
dataset.drop(lower[0], inplace = True)

print("New Shape: ", dataset.shape)
```

```
dataset
```

```
for col in num_cols[1:]:
  print('skewness value of ',col,dataset[col].skew())

# Now we have reduced the Age column's skewness values within -1 to 1 range
# We left the Exited column's skewness value as it is the dependent varaible
```

7 . Check for Categorical columns and perform encoding

Label encoding and One Hot encoding

```
dataset.reset_index(inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
categorical_feature_mask = dataset.dtypes==object
categorical_cols = dataset.columns[categorical_feature_mask].tolist()
```

```
categorical_cols=categorical_cols[1:]
categorical_cols
```

```
le = LabelEncoder()
dataset[categorical_cols] = dataset[categorical_cols].apply(lambda col: le.fit_transform(col))
dataset[categorical_cols].head(10)
```

```
categorical_feature_mask
```

```
enc=OneHotEncoder()
enc_data=pd.DataFrame(enc.fit_transform(dataset[['Geography','Gender']]).toarray())
enc_data
```

```
#First three columns of enc_data is for Geography and the next two columns is for Gender, we can replace the already existing cate

#Dropping already existing Geography and Gender columns
dataset.drop(['Geography'], axis=1,inplace=True)
dataset.drop(['Gender'], axis=1,inplace=True)

dataset.insert(2, "Geography_France", enc_data.iloc[:,0], True)
dataset.insert(3, "Geography_Germany", enc_data.iloc[:,1], True)
dataset.insert(4, "Geography_Spain", enc_data.iloc[:,2], True)
dataset.insert(5, "Gender_Female", enc_data.iloc[:,3], True)
dataset.insert(6, "Gender_Male", enc_data.iloc[:,4], True)
```

```
dataset
```

```
# We drop some irrelevant columns that does not contribute to prediction
dataset.drop(columns="CustomerId",axis=1,inplace=True)
dataset.drop(columns="Surname",axis=1,inplace=True)
dataset.drop(columns="index",axis=1,inplace=True)
```

```
dataset
```

8 . Split the data into dependent and independent variables

```
X= dataset.iloc[:,:-1].values #Indepedent variables
y= dataset.iloc[:,-1].values #Dependent varaibles
```

```
X
```

```
array([[1.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 1.0000000e+00,
        1.0000000e+00, 1.0134888e+05],
       [0.0000000e+00, 0.0000000e+00, 1.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 1.1254258e+05],
       [1.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 1.0000000e+00,
        0.0000000e+00, 1.1393157e+05],
       ...,
       [1.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 0.0000000e+00,
        1.0000000e+00, 4.2085580e+04],
       [0.0000000e+00, 1.0000000e+00, 0.0000000e+00, ..., 1.0000000e+00,
        0.0000000e+00, 9.2888520e+04],
       [1.0000000e+00, 0.0000000e+00, 0.0000000e+00, ..., 1.0000000e+00,
        0.0000000e+00, 3.8190780e+04]])
```

```
y
```

```
array([1, 0, 1, ..., 1, 1, 0])
```

9 . Scale the independent variable

```
from sklearn.preprocessing import StandardScaler
scale= StandardScaler()
X = scale.fit_transform(X)
X
```

```
array([[ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
         0.99573337,  0.01997639],
       [-1.0028197 , -0.57955796,  1.74527693, ..., -1.54891873,
         0.99573337,  0.21465635],
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
        -1.00428491,  0.23881355],
       ...,
       [ 0.99718823, -0.57955796, -0.57297497, ..., -1.54891873,
```

```
        0.99573337, -1.01072631],
       [-1.0028197 ,  1.72545295, -0.57297497, ...,  0.64561166,
        -1.00428491, -0.12716553],
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
        -1.00428491, -1.07846436]])
```

```
from sklearn.model_selection import train_test_split

# We use train_test_split function to split the data such that 25% is used for testing while the remaining 75% is used for trainin
X_train, X_test, y_train, y_test = train_test_split(X,y , random_state=104,test_size=0.25, shuffle=True)

X_train
```

```
array([[ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
         0.99573337, -1.74019169],
       [-1.0028197 , -0.57955796,  1.74527693, ..., -1.54891873,
        -1.00428491, -1.39787901],
       [-1.0028197 ,  1.72545295, -0.57297497, ..., -1.54891873,
         0.99573337, -1.48817335],
       ...,
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
        -1.00428491,  0.71481237],
       [ 0.99718823, -0.57955796, -0.57297497, ..., -1.54891873,
        -1.00428491,  0.60834563],
       [-1.0028197 ,  1.72545295, -0.57297497, ...,  0.64561166,
         0.99573337,  0.0525285 ]])
```

```
X_test
```

```
array([[-1.0028197 , -0.57955796,  1.74527693, ..., -1.54891873,
        -1.00428491, -0.90389608],
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
         0.99573337, -0.54087223],
       [-1.0028197 , -0.57955796,  1.74527693, ...,  0.64561166,
         0.99573337, -1.02004733],
       ...,
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
         0.99573337, -0.23978536],
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
         0.99573337, -0.17457887],
       [ 0.99718823, -0.57955796, -0.57297497, ...,  0.64561166,
        -1.00428491, -0.0121091 ]])
```

```
y_train
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
y_test
```

```
array([0, 1, 0, ..., 0, 0, 1])
```