importing essential libraries for build a ml model

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('Churn_Modelling.csv')
```

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

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In [4]:
```
#shape of the dataset
df.shape
```

Out[4]:
```
(10000, 14)
```

In [5]:
```
#give the information about the attributes of the dataset
df.info()
```

Out[5]:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1 MB
```

In [6]:
```
#checking for information null values
```

Output In [6]:
```
  Input In [6]
    #checking for information checking null values
                                                   ^
SyntaxError: invalid syntax
```

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

Out[7]:
```
RowNumber        0
CustomerId       0
Surname          0
CreditScore      0
Geography        0
Gender           0
Age              0
Tenure           0
Balance          0
NumOfProducts    0
HasCrCard        0
IsActiveMember   0
EstimatedSalary  0
Exited           0
dtype: int64
```

In [8]:
```
df.size
```

Out[8]:
```
140000
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [10]:
```
#drawing values which are not necessary for analysis
df.drop(columns=['RowNumber','CustomerId','Surname'],axis=1,inplace=True)
```

In [11]:
```
df.head()
```

Out[11]:

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

In [12]:
```
#give null value info
df.isnull().sum()
```

Out[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
```

In [13]:
```
#give statistical summary of data
df.describe()
```

Out[13]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.455835	0.499797	57512.654318	0.402769
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.000000	0.000000	11.580000	0.000000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.000000	0.000000	51002.110000	0.000000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.000000	1.000000	100193.915000	0.000000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.000000	1.000000	149388.247500	0.000000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.000000	1.000000	199992.480000	1.000000

In [14]:
```
df.table()
```

Out[14]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
8996	771	France	Male	39	5	0.00	2	1	0	96270.64	0
8997	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
8997	709	France	Female	36	7	0.00	1	0	1	42085.58	1
8998	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
8999	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

In [15]:
```
#EDA (exploratory data analysis)
df['Exited'].value_counts(normalize=True)*100
```

Out[15]:
```
0    79.63
1    20.37
Name: Exited, dtype: float64
```

In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [17]:
```
plt.figure(figsize = (5,5))
ax = sns.countplot(x = df['Exited'],data=df)
for label in ax.containers:ax.bar_label(label);
```

Out[17]:

In [18]:
```
plt.figure(figsize=(5,5))
target = df['Exited'].value_counts().values
target_name = df['Exited'].value_counts().index
plt.pie(x = target_val,labels=['Same','Exited'],autopct = '%1.2f%%',explode=(0,0.1))
plt.show()
```

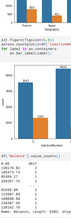
Out[18]:

In [19]:
```
print('Unewness:',df['CreditScore'].skew())
print('Kurto:',df['CreditScore'].kurt())
```

Out[19]:
```
Unewness: -0.07168969209687575
Kurto: -0.4252154
Name: CreditScore, dtype: int64
```

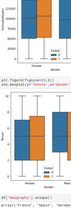
In [20]:
```
plt.figure(figsize=(8,5))
sns.distplot(x = df['CreditScore'])
```

Out[20]:



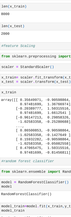
In [21]:
```
#handling categorical columns
print('Unewness:',df['Age'].skew())
plt.figure(figsize = (5,5))
sns.histplot(x = df['Age'],kde=True);
```

Out[21]:
```
Unewness: 1.0113351592174452
```



In [22]:
```
#  chart by group of age
df['Age Group'] = pd.cut(df['Age'],
bins=[0,30,40,50,60,np.inf],
labels=['<30', '30-40', '40-50', '50-60', '60+'])
plt.figure(figsize=(8,5))
ax = sns.countplot(x = 'Age_Group',hue='Exited',data=df)
plt.title('Churn by Age Group')
for label in ax.containers:
    ax.bar_label(label);
```

Out[22]:



In [23]:
```
plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
across.displot(df['Exited'],hue='Age',)
plt.subplot(1,2,2)
sns.boxplot(x = df['Exited'],y = df['Age'],hue=False,kde=On)
ax.set_xlabel('Age')
ax.set_title('Not Churn')
ax.set_ylabel('Age')
ax = sns.displot(df['Exited'][df['Exited']!='Age'],hue=False)
ax.set_xlabel('Age_class')
ax.set_ylabel('Count')
plt.show()
```

Out[23]:

In [24]:
```
#geographical column
plt.figure(figsize=(5,5))
ax = sns.countplot(x = df['Geography'],hue='Exited',data=df)
for label in ax.containers:
    ax.bar_label(label);
```

Out[24]:

In [25]:
```
plt.figure(figsize=(5,5))
across.countplot(df['IsActiveMember'],hue='Exited',data=df)
for label in ax.containers:
    ax.bar_label(label);
```

Out[25]:

In [26]:
```
df['Balance'].value_counts()
```

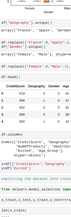
Out[26]:
```
0.00      3617
130170.82    2
105473.74    2
85304.27     2
159397.75    2
          ....
92204.80     1
114838.48    1
109928.09    1
150307.56    1
75075.31     1
Name: Balance, Length: 6382, dtype: int64
```

In [27]:
```
#due to more no of zero balance customers
df['Balance'].where(df['Balance']!=0,).value_counts()
```

Out[27]:
```
3.0    9365
2.0    3846
Name: Balance, dtype: int64
```

In [28]:
```
plt.figure(figsize=(5,4))
sns.distplot(df['Balance']);
```

Out[28]:



In [29]:
```
plt.figure(figsize=(5,5))
sns.boxplot(y='Tenure',x='Gender',hue='Exited',data=df);
```

Out[29]:

In [30]:
```
df['Geography'].unique()
```

Out[30]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [31]:
```
df.replace({'France':0,'Spain':1,'Germany':2},inplace=True)
df['Gender'].unique()
```

Out[31]:
```
array(['Female', 'Male'], dtype=object)
```

In [32]:
```
df.replace({'Female':0,'Male':1},inplace=True)
```

In [33]:
```
df.head()
```

Out[33]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Age_Group
0	619	0	0	42	2	0.00	1	1	1	101348.88	1	40-50
1	608	1	0	41	1	83807.86	1	0	1	112542.58	0	40-50
2	502	0	0	42	8	159660.80	3	1	0	113931.57	1	40-50
3	699	0	0	39	1	0.00	2	0	0	93826.63	0	30-40
4	850	1	0	43	2	125510.82	1	1	1	79084.10	0	40-50

In [34]:
```
df.columns
```

Out[34]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited', 'Age_Group'],
      dtype=object)
```

In [35]:
```
x=df[['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]
y=df['Exited']
```

In [36]:
```
#splitting the dataset into train tandet
```

In [37]:
```
from sklearn.model_selection import train_test_split
```

In [38]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [39]:
```
len(x_train)
```

Out[39]:
```
8000
```

In [40]:
```
len(x_test)
```

Out[40]:
```
2000
```

In [41]:
```
#feature Scaling
```

In [42]:
```
from sklearn.preprocessing import StandardScaler
```

In [43]:
```
scaler = StandardScaler()
```

In [44]:
```
x_train= scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [45]:
```
x_train
```

Out[45]:
```
array([[-0.18640971, -0.90500884,  0.91324755, ...,  0.64192567,
        0.97431099,  1.74176974],
       [-0.14481897, -0.90500884,  0.91324755, ...,  0.64192567,
        0.97431099,  0.18121161],
       [-0.94147213,  0.20803329,  0.91324755, ...,  0.64192567,
        0.97431099,  0.25208966],
       ...,
       [ 0.85930021, -0.90500884, -1.09499335, ...,  1.54915155,
        0.97431099,  0.24188127],
       [-0.19161202,  1.10204214,  0.91324755, ...,  0.64192567,
        0.97431099,  0.06882202],
       [ 0.49180225, -0.90500884,  0.91324755, ...,  0.64192567,
        0.97431099,  0.82166511]])
```

In [46]:
```
#random forest classifier
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [48]:
```
model = RandomForestClassifier()
model
```

Out[48]:
```
RandomForestClassifier()
```

In [49]:
```
model_trained=model.fit(x_train,y_train)
model_trained
```

Out[49]:
```
RandomForestClassifier()
```

In [50]:
```
model_test=model.predict(x_test)
model_test
```

Out[50]:
```
array([0, 0, 0, ..., 1, 0, 0], dtype=int64)
```

In [51]:
```
#model accuracy
```

In [52]:
```
from sklearn.metrics import accuracy_score
```

In [53]:
```
score = accuracy_score(y_test,model_test)
score
```

Out[53]:
```
0.8645
```

In [ ]:

In [ ]: