Assignment - 2

PROJECT NAME:	CAR RESALE VALUE PREDICTION
NAME:	VIGNESH K
ROLLNO:	921319104220

Data Visualization and Pre-processing

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

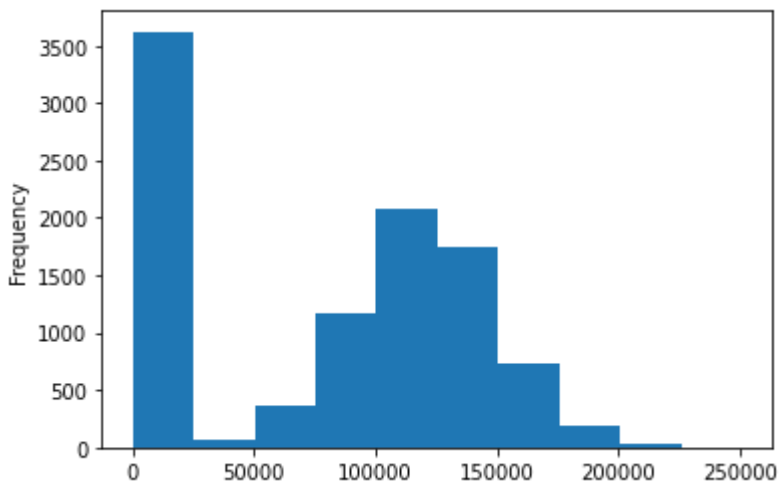
In [7]:
```
df=pd.read_csv("/Churn_Modelling.csv")
```

In [9]:
```
df.head()
```

Out[9]:

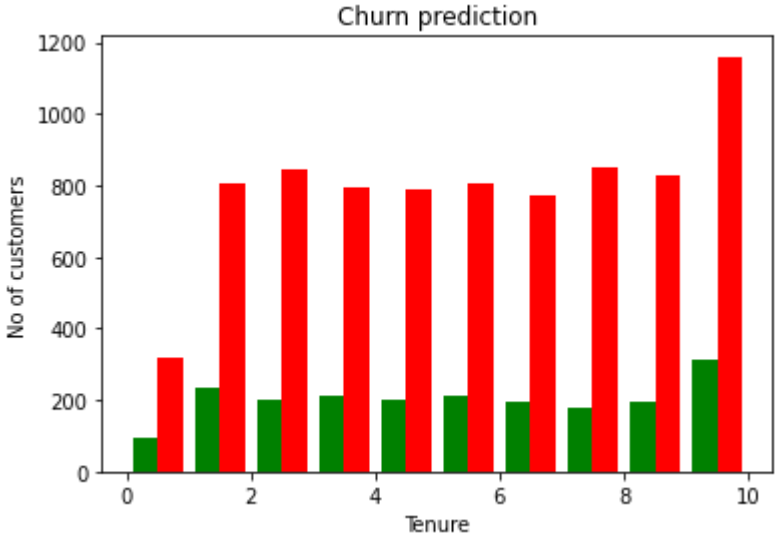
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1	15634602	Hargrave	619	France	Female	42	2	0
1	2	15647311	Hill	608	Spain	Female	41	1	83807
2	3	15619304	Onio	502	France	Female	42	8	159660
3	4	15701354	Boni	699	France	Female	39	1	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510

In [ ]:
```
#Univariate Analysis
df["Balance"].plot(kind='hist');
```



In [ ]:
```
#Bi-Variate Analysis
cy=df[df.Exited==1].Tenure
cn=df[df.Exited==0].Tenure
plt.title("Churn prediction")
plt.xlabel("Tenure")
plt.ylabel("No of customers")
plt.hist([cy,cn],color=['green','red'],label=["churn=yes"])
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

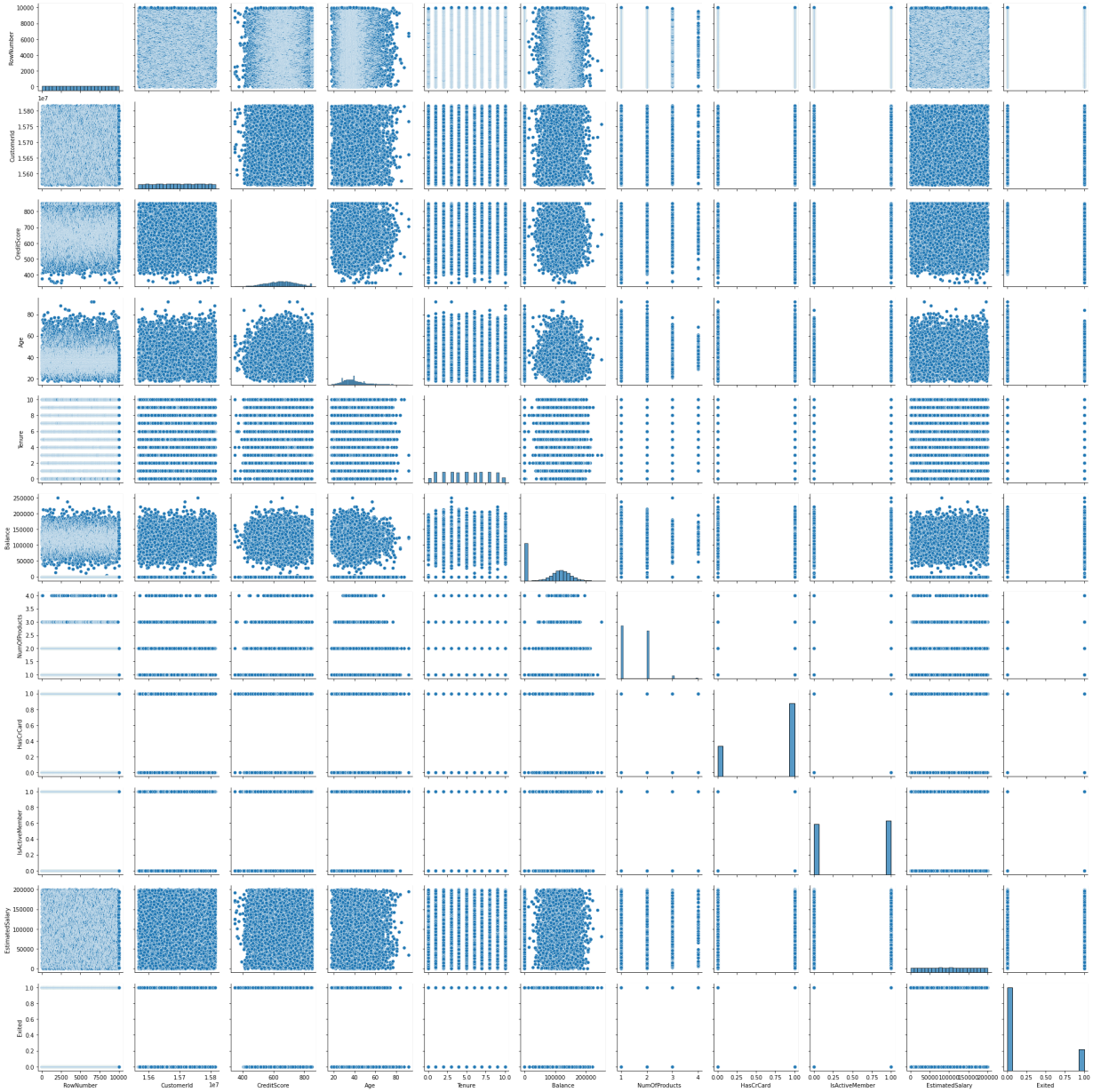


In [ ]:

```
#Multi-variate Analysis
sns.pairplot(df)
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x7f54396e8e10>
```



In [ ]:
```
#Descriptive statistics
df.describe()
```

Out[ ]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balanc
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405201
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

In [ ]:
```
#Handle The Missing values
df.isnull().any()
```

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

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

Out[ ]:
```
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 [ ]:
```
#Find the outliers
df.skew()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning:
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=Non
e') is deprecated; in a future version this will raise TypeError.  Select only
valid columns before calling the reduction.
```

Out[ ]:
```
RowNumber          0.000000
CustomerId         0.001149
CreditScore       -0.071607
Age                1.011320
Tenure             0.010991
Balance           -0.141109
NumOfProducts      0.745568
HasCrCard         -0.901812
IsActiveMember    -0.060437
EstimatedSalary    0.002085
Exited             1.471611
dtype: float64
```

In [ ]:
```
#Split data into dependent and independent variables
x=df.iloc[:,3:13].values
y=df.iloc[:,13:14].values
x.shape
```

Out[ ]:
```
(10000, 10)
```

In [ ]:
```
y.shape
```

Out[ ]:
```
(10000, 1)
```

In [ ]:
```
#Categorical colums and encoding
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer([("oh",OneHotEncoder(),[1,2])],remainder="passthrough")
x=ct.fit_transform(x)
x.shape
```

Out[ ]:
```
(10000, 13)
```

In [ ]:
```
df["Gender"].unique()
```

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

In [ ]:
```
#Split the data into training and testing
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state
x_train.shape
```

Out[ ]:
```
(8000, 13)
```

In [ ]:
```
x_test.shape
```

Out[ ]:
```
(2000, 13)
```

In [ ]:
```
#Scale the independent variables
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

In [ ]:
```
import joblib
joblib.dump(ct,"churn.pkl")
```

Out[ ]:
```
['churn.pkl']
```

In [ ]:
```
joblib.dump(sc,"churnsc.pkl")
```

Out[ ]:
```
['churnsc.pkl']
```