Assignment -2

DATA VISUALIZATION AND PREPROCESSING

Assignment Date	24 September 2022
Student Name	Mr.K.FRANCIS HUBAN
Student Roll Number	913319104018
Maximum Marks	2 Marks

Question-1:

DOWNLOAD THE DATA SET

The given data set

Question-2:

LOAD THE DATA SET

Solution

```
import numpy as np
import pandas as pd
DF=pd.read_csv("/content/Churn_Modelling.csv")
DF.head()
```

	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

Question-3:

Perform below visualization

- **Univariate analysis**
- **Bivariate analysis**
- **Multivariate analysis**

Solution

Univariate analysis

#Calculate Summary Statistics

```
import numpy as np

import pandas as pd

DF=pd.read_csv("/content/Churn_Modelling.csv")

print("mean",DF['EstimatedSalary'].mean())
```

```
print("median",DF['EstimatedSalary'].median())

print("mode",DF['EstimatedSalary'].mode())
```

```
mean 100090.239881
median 100193.915
mode 0    24924.92
dtype: float64
```

#frequency

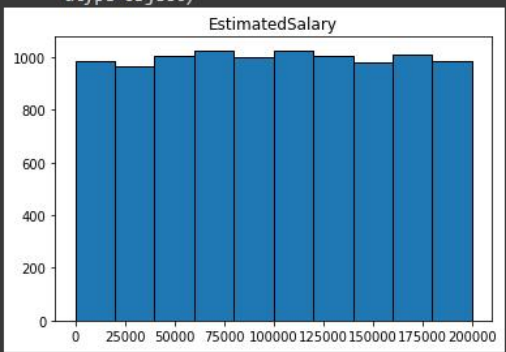
```
DF['Age'].value_counts()
```

```
37    478
38    477
35    474
36    456
34    447
     ...
92      2
82      1
88      1
85      1
83      1
Name: Age, Length: 70, dtype: int64
```

#create charts

```
DF.hist(column='EstimatedSalary', grid=False, edgecolor='black')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ed7d4d6d0>]],
      dtype=object)
```



Bivariate analysis

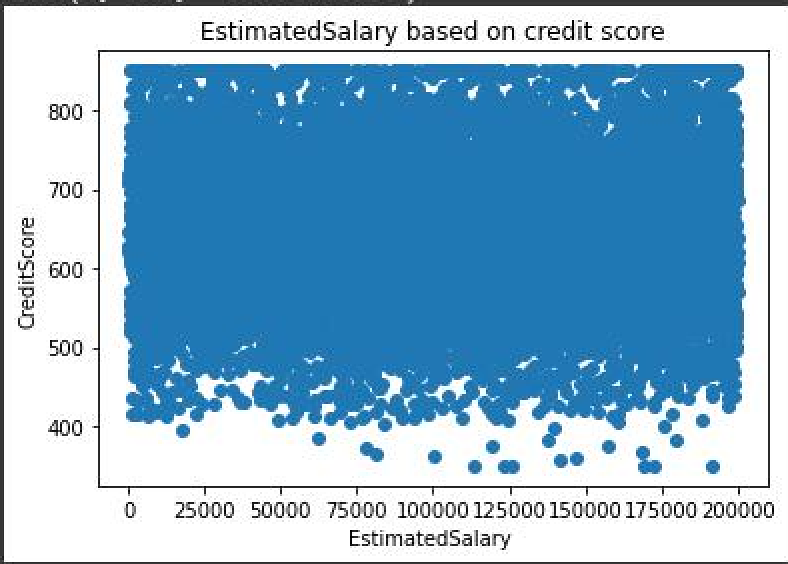
Scatter plot

```
import matplotlib.pyplot as plt

DF=pd.read_csv("/content/Churn_Modelling.csv")

plt.scatter(DF.EstimatedSalary, DF.CreditScore)

plt.title('EstimatedSalary based on credit score')
```

plt.xlabel('EstimatedSalary ')

plt.ylabel('CreditScore')

```
Text(0, 0.5, 'CreditScore')
```



Corelation coeficient

DF.corr()

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
RowNumber	1.000000	0.004202	0.005840	0.000783	-0.006495	-0.009067	0.007246	0.000599	0.012044	-0.005988	-0.016571
CustomerId	0.004202	1.000000	0.005308	0.009497	-0.014883	-0.012419	0.016972	-0.014025	0.001665	0.015271	-0.006248
CreditScore	0.005840	0.005308	1.000000	-0.003965	0.000842	0.006268	0.012238	-0.005458	0.025651	-0.001384	-0.027094
Age	0.000783	0.009497	-0.003965	1.000000	-0.009997	0.028308	-0.030680	-0.011721	0.085472	-0.007201	0.285323
Tenure	-0.006495	-0.014883	0.000842	-0.009997	1.000000	-0.012254	0.013444	0.022583	-0.028362	0.007784	-0.014001
Balance	-0.009067	-0.012419	0.006268	0.028308	-0.012254	1.000000	-0.304180	-0.014858	-0.010084	0.012797	0.118533
NumOfProducts	0.007246	0.016972	0.012238	-0.030680	0.013444	-0.304180	1.000000	0.003183	0.009612	0.014204	-0.047820
HasCrCard	0.000599	-0.014025	-0.005458	-0.011721	0.022583	-0.014858	0.003183	1.000000	-0.011866	-0.009933	-0.007138
IsActiveMember	0.012044	0.001665	0.025651	0.085472	-0.028362	-0.010084	0.009612	-0.011866	1.000000	-0.011421	-0.156128
EstimatedSalary	-0.005988	0.015271	-0.001384	-0.007201	0.007784	0.012797	0.014204	-0.009933	-0.011421	1.000000	0.012097
Exited	-0.016571	-0.006248	-0.027094	0.285323	-0.014001	0.118533	-0.047820	-0.007138	-0.156128	0.012097	1.000000

Simple linear regression

import statsmodels.api as sm

y = DF['EstimatedSalary']

x = DF['CreditScore']

x = sm.add_constant(x)

model = sm.OLS(y, x).fit()

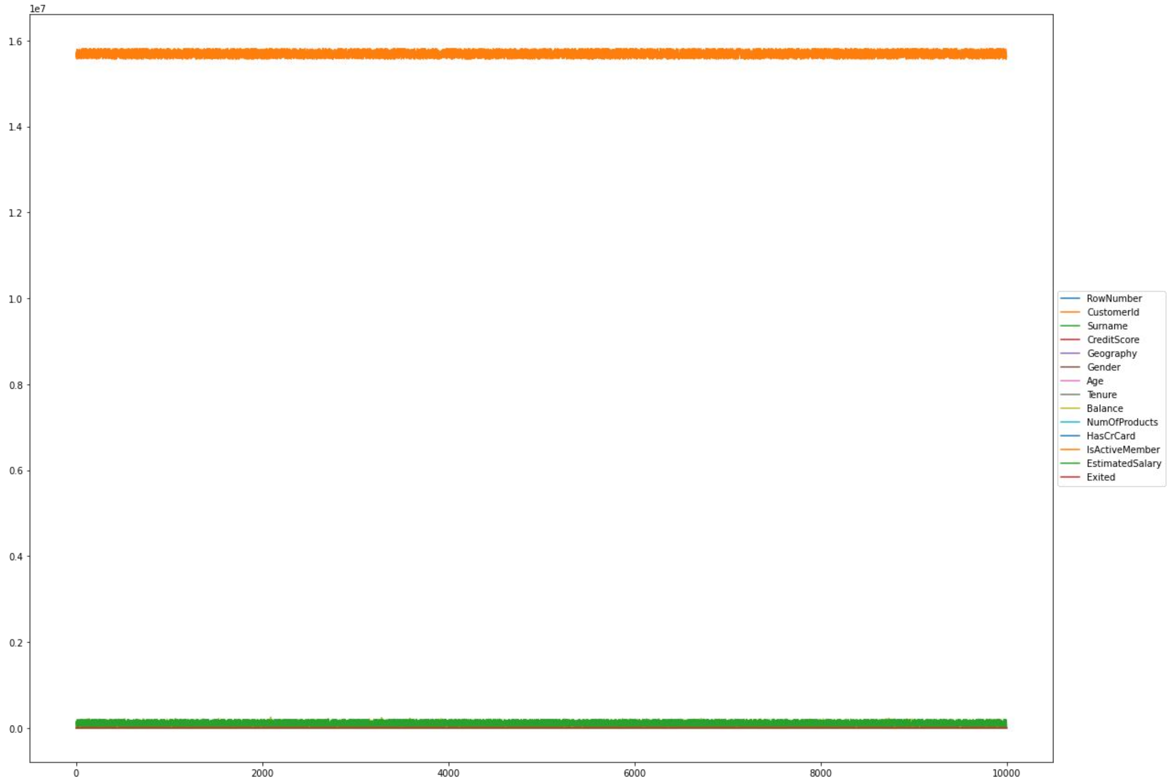
print(model.summary())

```
[ ]                             OLS Regression Results
    ==============================================================================
    Dep. Variable:        EstimatedSalary   R-squared:                       0.000
    Model:                            OLS   Adj. R-squared:                 -0.000
    Method:                 Least Squares   F-statistic:                   0.01916
    Date:                Wed, 28 Sep 2022   Prob (F-statistic):              0.890
    Time:                        06:38:48   Log-Likelihood:            -1.2379e+05
    No. Observations:               10000   AIC:                         2.476e+05
    Df Residuals:                    9998   BIC:                         2.476e+05
    Df Model:                           1
    Covariance Type:            nonrobust
    ===============================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
    -------------------------------------------------------------------------------
    const        1.006e+05   3913.640     25.712      0.000     9.3e+04    1.08e+05
    CreditScore    -0.8237      5.951     -0.138      0.890     -12.488      10.841
    ==============================================================================
    Omnibus:                     7392.705   Durbin-Watson:                   2.010
    Prob(Omnibus):                  0.000   Jarque-Bera (JB):              581.607
    Skew:                           0.002   Prob(JB):                    5.08e-127
    Kurtosis:                       1.819   Cond. No.                     4.48e+03
    ==============================================================================

    Notes:
    [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
    [2] The condition number is large, 4.48e+03. This might indicate that there are
    strong multicollinearity or other numerical problems.
    /usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of pandas all arguments of concat except for the argume
```

Multivariate analysis

ax = DF.plot(figsize=(20,15))

ax.legend(loc='center left', bbox_to_anchor=(1, 0.5));



Question-4:

Perform descriptive statistics on the dataset

Solution

DF.describe()

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

DF.describe(include=['object'])

	Surname	Geography	Gender
count	10000	10000	10000
unique	2932	3	2
top	Smith	France	Male
freq	32	5014	5457

Question-5:

Handle the missing values

Solution

DF.info()

```
<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
```

missing_values=DF.isnull().sum()

print(missing_values[missing_values>0]/len(DF)*100)

missing_values

```
Series([], dtype: float64)
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
```

<u>Question-6:</u>

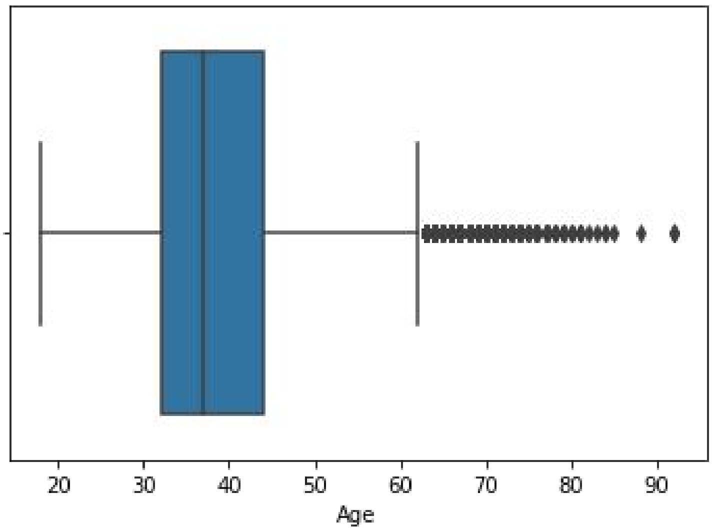
Find out the outliers

Solution

AGE OUTLIER

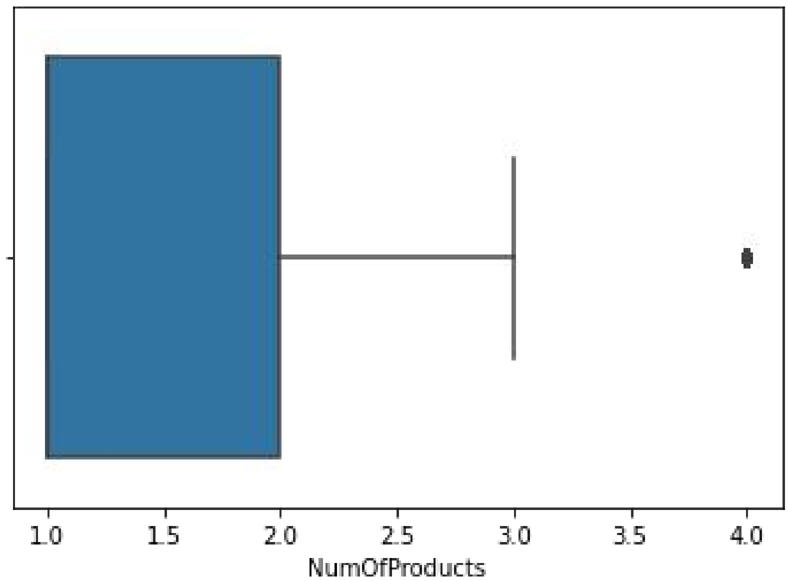
import seaborn as sns

sns.boxplot(DF['Age'])



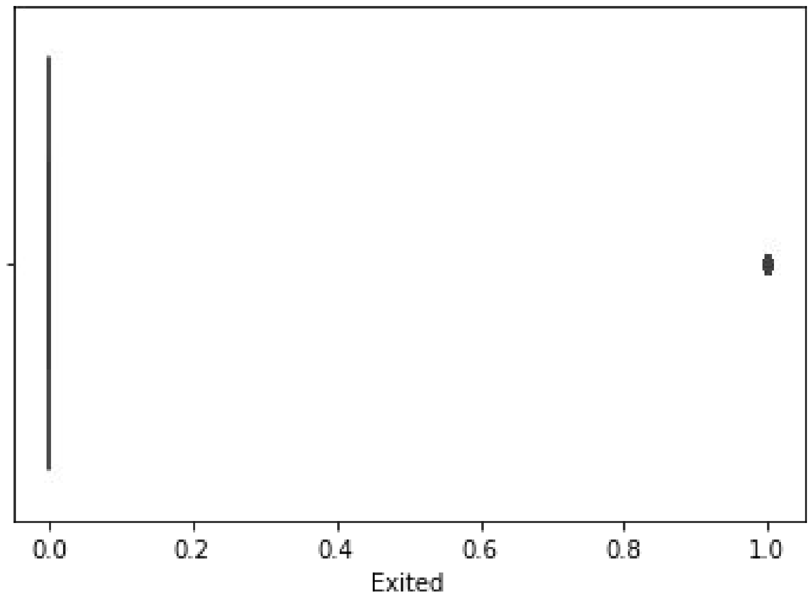
NUMOFPRODUCTS OUTLIER

sns.boxplot(DF['NumOfProducts'])



EXITED OUTLIER

```
sns.boxplot(DF['Exited'])
```



DETECTION OF **AGE OUTLIER**

```
a=np.where(DF['Age']>60)
print("OUTLIERS OF AGE\n",a)
```

```
 (array([  42,   44,   58,   85,  104,  158,  181,  230,  234,  243,  252,
         276,  310,  364,  371,  385,  387,  399,  416,  484,  538,  559,
         561,  567,  602,  612,  617,  630,  658,  678,  696,  736,  766,
         769,  807,  811,  823,  859,  884,  888,  921,  928,  948,  952,
         957,  963,  969,  997, 1009, 1039, 1040, 1055, 1114, 1118, 1192,
        1205, 1234, 1235, 1246, 1252, 1278, 1285, 1328, 1342, 1387, 1407,
        1410, 1433, 1439, 1457, 1519, 1543, 1588, 1607, 1614, 1642, 1790,
        1810, 1858, 1866, 1901, 1904, 1907, 1933, 1981, 1996, 2002, 2012,
        2039, 2053, 2078, 2094, 2103, 2108, 2154, 2159, 2164, 2244, 2261,
        2274, 2298, 2301, 2433, 2438, 2458, 2459, 2519, 2520, 2533, 2541,
        2553, 2599, 2615, 2659, 2670, 2713, 2717, 2760, 2772, 2777, 2778,
        2781, 2791, 2855, 2877, 2901, 2908, 2925, 2926, 3008, 3033, 3054,
        3110, 3142, 3166, 3192, 3203, 3229, 3305, 3308, 3311, 3314, 3317,
        3346, 3366, 3368, 3378, 3382, 3384, 3387, 3396, 3403, 3434, 3462,
        3497, 3499, 3527, 3531, 3541, 3549, 3559, 3563, 3573, 3575, 3593,
        3602, 3641, 3646, 3647, 3651, 3690, 3691, 3702, 3719, 3728, 3733,
        3761, 3774, 3813, 3826, 3880, 3881, 3888, 3909, 3910, 3927, 3940,
        3947, 3980, 3994, 4010, 4025, 4048, 4051, 4095, 4142, 4147, 4157,
        4162, 4170, 4241, 4244, 4256, 4273, 4280, 4297, 4313, 4318, 4335,
        4360, 4366, 4378, 4387, 4396, 4435, 4438, 4463, 4490, 4491, 4501,
        4506, 4559, 4563, 4590, 4595, 4644, 4678, 4698, 4747, 4751, 4801,
        4815, 4832, 4849, 4931, 4947, 4966, 4992, 5000, 5020, 5033, 5038,
        5068, 5132, 5136, 5148, 5159, 5197, 5223, 5225, 5235, 5255, 5299,
        5313, 5368, 5377, 5405, 5439, 5457, 5490, 5508, 5514, 5520, 5576,
        5577, 5581, 5639, 5651, 5655, 5660, 5664, 5671, 5683, 5698, 5742,
```

5577, 5581, 5639, 5651, 5655, 5660, 5664, 5671, 5683, 5698, 5742,

```
5777, 5783, 5817, 5825, 5840, 5867, 5907, 5957, 5996, 6046, 6116,
6152, 6166, 6167, 6171, 6173, 6212, 6230, 6278, 6289, 6315, 6357,
6366, 6373, 6375, 6410, 6443, 6515, 6530, 6532, 6581, 6612, 6626,
6706, 6709, 6715, 6721, 6759, 6763, 6812, 6899, 6970, 6997, 7008,
7057, 7058, 7063, 7071, 7078, 7094, 7138, 7139, 7142, 7156, 7194,
7202, 7238, 7243, 7272, 7302, 7362, 7375, 7392, 7499, 7514, 7523,
7526, 7548, 7552, 7623, 7624, 7629, 7668, 7687, 7692, 7694, 7709,
7715, 7719, 7720, 7727, 7773, 7776, 7784, 7788, 7802, 7813, 7851,
7894, 7898, 7909, 7933, 7956, 7995, 8019, 8037, 8094, 8098, 8156,
8170, 8193, 8207, 8215, 8217, 8304, 8321, 8385, 8394, 8444, 8458,
8467, 8469, 8478, 8488, 8562, 8568, 8577, 8602, 8674, 8686, 8689,
8711, 8759, 8761, 8763, 8768, 8787, 8793, 8822, 8865, 8900, 8917,
8930, 8970, 9018, 9021, 9062, 9080, 9102, 9112, 9116, 9162, 9174,
9223, 9261, 9279, 9284, 9292, 9309, 9318, 9321, 9324, 9332, 9333,
9351, 9380, 9391, 9402, 9425, 9428, 9438, 9472, 9490, 9506, 9555,
9557, 9582, 9587, 9589, 9593, 9595, 9646, 9671, 9673, 9681, 9686,
9688, 9718, 9733, 9734, 9736, 9747, 9753, 9765, 9832, 9879, 9894,
9897, 9936]),)
```

DETECTION OF **NUMOFPRODUCTS OUTLIER**

b=np.where(DF['NumOfProducts']>3)

print("OUTLIERS OF NUMOFPRODUCTS\n",b)

```
OUTLIERS OF NUMOFPRODUCTS
 (array([   7,   70, 1254, 1469, 1488, 1701, 1876, 2124, 2196, 2285, 2462,
        2499, 2509, 2541, 2614, 2617, 2872, 3152, 3365, 3841, 4013, 4014,
        4166, 4260, 4403, 4511, 4516, 4606, 4654, 4748, 4822, 5010, 5137,
        5235, 5386, 5700, 5904, 6150, 6172, 6279, 6750, 6875, 7257, 7457,
        7567, 7698, 7724, 7729, 8041, 8590, 8683, 8850, 8923, 9215, 9255,
        9323, 9370, 9411, 9540, 9565]),)
```

DETECTION OF **EXITED OUTLIER**

c=np.where(DF['Exited']>0)

print("OUTLIERS OF Exited\n",c)

```
OUTLIERS OF Exited
 (array([   0,    2,    5, ..., 9991, 9997, 9998]),)
```

<u>Question-7:</u>

Check the categorical columns and perform encoding

Solution

location=pd.get_dummies(km['Geography'])

from sklearn.preprocessing import LabelEncoder

from collections import Counter as count

le=LabelEncoder()

count(km['Geography'])

DF['Geography']=le.fit_transform(DF['Geography'])
count(DF['Geography'])

```
Counter({0: 5014, 2: 2477, 1: 2509})
```

Count(DF['Surname'])
DF'Surname']=le.fit_transform(DF['Surname'])
count(DF['Surname'])

```
Counter({1115: 1,
         1177: 17,
         2040: 8,
         289: 14,
         1822: 20,
         537: 22,
         177: 4,
         2000: 2,
         1146: 18,
         1081: 19,
         195: 1,
         83: 6,
         1369: 5,
         515: 16,
         2389: 29,
         1021: 1,
         2307: 1,
         1154: 16,
         1872: 1,
         1108: 12,
         1736: 19,
         697: 13,
         991: 2,
         1862: 1,
         2880: 14,
         1642: 24,
         2897: 20,
```

DF['Gender']=DF[‘Gender'].replace(['Male','Female'],[0,1])

DF

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	1115	619	0	1	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	1177	608	2	1	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	2040	502	0	1	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	289	699	0	1	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	1822	850	2	1	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	1999	771	0	0	39	5	0.00	2	1	0	96270.64	0
9996	9997	15569892	1336	516	0	0	35	10	57369.61	1	1	1	101699.77	0
9997	9998	15584532	1570	709	0	1	36	7	0.00	1	0	1	42085.58	1
9998	9999	15682355	2345	772	1	0	42	3	75075.31	2	1	0	92888.52	1
9999	10000	15628319	2751	792	0	1	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 14 columns

Question-8:

Split the data into dependent and independent variables

Solution

independent

DF['Gender']=DF['Gender'].replace(['Male','Female'],[0,1])

x=DF.iloc[:,2:]

print("\nindependent variable\n",x)

```
 independent variable
        Surname  CreditScore  Geography  Gender  Age  Tenure    Balance  \
0          1115          619          0       1   42       2       0.00
1          1177          608          2       1   41       1   83807.86
2          2040          502          0       1   42       8  159660.80
3           289          699          0       1   39       1       0.00
4          1822          850          2       1   43       2  125510.82
...         ...          ...        ...     ...  ...     ...        ...
9995       1999          771          0       0   39       5       0.00
9996       1336          516          0       0   35      10   57369.61
9997       1570          709          0       1   36       7       0.00
9998       2345          772          1       0   42       3   75075.31
9999       2751          792          0       1   28       4  130142.79

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

[10000 rows x 12 columns]
```

Dependent

y=DF.iloc[:,0:2]

print("dependent variables\n",y)

```
dependent variables
       RowNumber  CustomerId
0             1    15634602
1             2    15647311
2             3    15619304
3             4    15701354
4             5    15737888
...         ...         ...
9995       9996    15606229
9996       9997    15569892
9997       9998    15584532
9998       9999    15682355
9999      10000    15628319

[10000 rows x 2 columns]
```

Question-9:

Scale the independent variables

Solution

xtrain

from sklearn.preprocessing import MinMaxScaler

nm=MinMaxScaler()

n_xtrain=nm.fit_transform(X_train)

n_xtrain

```
array([[0.33879222, 0.974     , 1.        , ..., 1.        , 0.25485714,
        0.        ],
       [0.57795974, 1.        , 1.        , ..., 1.        , 0.51955874,
        0.        ],
       [0.97065848, 0.636     , 1.        , ..., 0.        , 0.53233635,
        1.        ],
       ...,
       [0.40361651, 0.55      , 1.        , ..., 1.        , 0.67404984,
        0.        ],
       [0.21050836, 0.324     , 0.5       , ..., 0.        , 0.07409993,
        0.        ],
       [0.5663596 , 0.356     , 0.5       , ..., 1.        , 0.00475092,
        0.        ]])
```

Xtest

n_X_test=nm.fit_transform(X_test)

n_X_test

```
array([[0.61659269, 0.352     , 0.5       , ..., 0.        , 0.66189298,
        0.        ],
       [0.28303175, 0.496     , 0.        , ..., 1.        , 0.37133981,
        0.        ],
       [0.95800615, 0.384     , 0.        , ..., 1.        , 0.10631272,
        0.        ],
       ...,
       [0.76681461, 0.874     , 0.        , ..., 1.        , 0.31051302,
        0.        ],
       [0.8477296 , 0.74      , 1.        , ..., 0.        , 0.68981209,
        0.        ],
       [0.94093547, 0.384     , 0.        , ..., 0.        , 0.62636535,
        0.        ]])
```

Question-10:

Split the data into training and testing

Solution

xtrain

from sklearn.model_selection import train_test_split

x=km.iloc[:,2:]

y=km.iloc[:,0:2]

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=11)

X_train

	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1264	993	837	2	0	31	9	104678.62	1	0	1	50972.60	0
5376	1694	850	2	0	38	1	146343.98	1	0	1	103902.11	0
2037	2845	668	2	1	24	7	173962.32	1	0	0	106457.11	1
6485	1016	640	1	0	26	5	90402.77	1	1	1	3298.65	0
1600	1037	517	0	0	28	2	115062.61	1	1	0	179056.23	0
...	...	...	...	...	...	...	...	...	...	...	...	...
1293	1067	641	0	0	30	2	87505.47	2	0	1	7278.57	0
4023	2611	535	0	0	38	8	85982.07	1	1	0	9238.35	0
7259	1183	625	2	0	32	7	106957.28	1	1	1	134794.02	0
5200	617	512	1	0	42	9	93955.83	2	1	0	14828.54	0
3775	1660	528	1	0	22	5	93547.23	2	0	1	961.57	0

7000 rows × 12 columns

X_test

	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
3104	1808	526	1	0	31	5	145537.21	1	1	0	132404.64	0
6353	831	598	0	0	35	8	114212.60	1	1	1	74322.85	0
8689	2808	542	0	0	67	10	129431.36	1	0	1	21343.74	0
5857	909	594	0	1	56	7	0.00	1	1	0	26215.85	1
6011	2113	520	1	1	45	1	123086.39	1	1	1	41042.40	1
...	...	...	...	...	...	...	...	...	...	...	...	...
8125	2496	629	1	1	38	9	123948.85	1	1	0	76053.07	0
8444	839	792	0	1	70	3	0.00	2	1	1	172240.27	0
2167	2248	787	0	0	33	1	126588.81	2	0	1	62163.53	0
8043	2485	720	2	0	31	4	141356.47	1	0	0	137985.69	0
4917	2758	542	0	0	32	7	107871.72	1	1	0	125302.64	0

y_train

	RowNumber	CustomerId
1264	1265	15732199
5376	5377	15602500
2037	2038	15678146
6485	6486	15635197
1600	1601	15748718
...	...	...
1293	1294	15687752
4023	4024	15629187
7259	7260	15718921
5200	5201	15641298
3775	3776	15709004

7000 rows × 2 columns

y_test

	RowNumber	CustomerId
3104	3105	15654230
6353	6354	15676353
8689	8690	15684769
5857	5858	15813659
6011	6012	15783007
...	...	...
8125	8126	15666982
8444	8445	15793641
2167	2168	15780846
8043	8044	15616525
4917	4918	15681991

3000 rows × 2 columns