Assignment -2

Python Programming

Assignment Date	22 September 2022
Student Name	SURYA PRAKASH S
Maximum Marks	2 Marks

Question-1:

Downloading the dataset

Question-2:

Load the dataset:

Solution:

```
import pandas as pd df=pd.read_csv("/content/Churn_Modelling.csv")
```

Question-3:

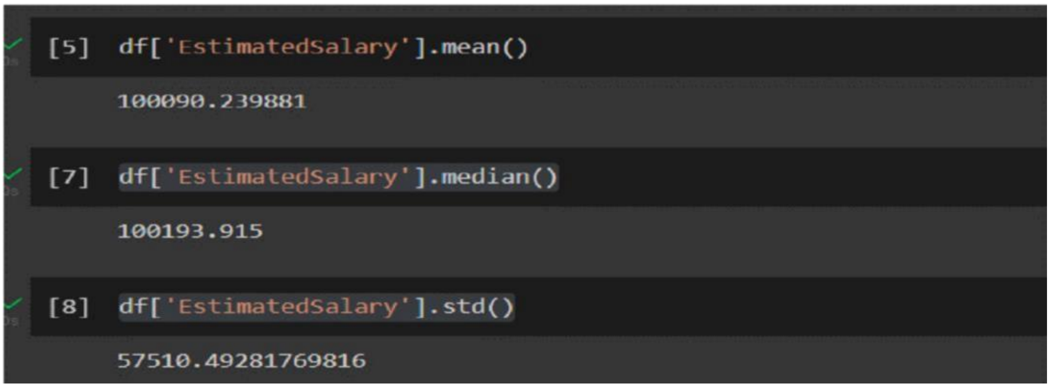
Perform Below Visualizations-Univariate Analysis, Bi - Variate Analysis and Multi - Variate Analysis

Solution:

Univariate Analysis:

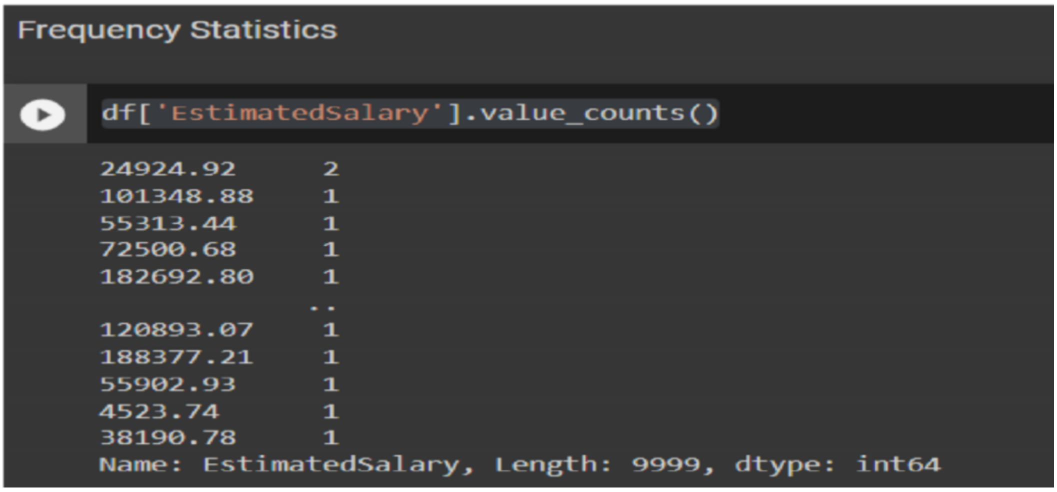
1.Summary Statistics

```
df['EstimatedSalary'].mean()

df['EstimatedSalary'].median()

df['EstimatedSalary'].std()
```



```
[5]  df['EstimatedSalary'].mean()

     100090.239881

[7]  df['EstimatedSalary'].median()

     100193.915

[8]  df['EstimatedSalary'].std()

     57510.49281769816
```

2.Frequency Statistics   df['EstimatedSalary'].value_counts()

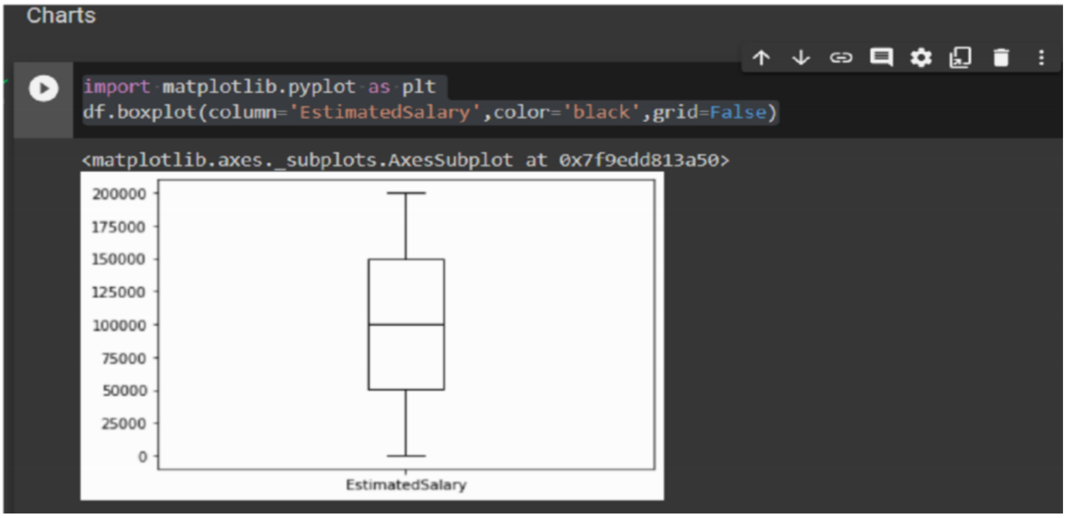


Frequency Statistics

```
df['EstimatedSalary'].value_counts()

24924.92     2
101348.88    1
55313.44     1
72500.68     1
182692.80    1
            ..
120893.07    1
188377.21    1
55902.93     1
4523.74      1
38190.78     1
Name: EstimatedSalary, Length: 9999, dtype: int64
```

3.Charts   import matplotlib.pyplot as plt

df.boxplot(column='EstimatedSalary',color='black',grid=False)



Charts

```
import matplotlib.pyplot as plt
df.boxplot(column='EstimatedSalary',color='black',grid=False)

<matplotlib.axes._subplots.AxesSubplot at 0x7f9edd813a50>
```

Bivariate Analysis:

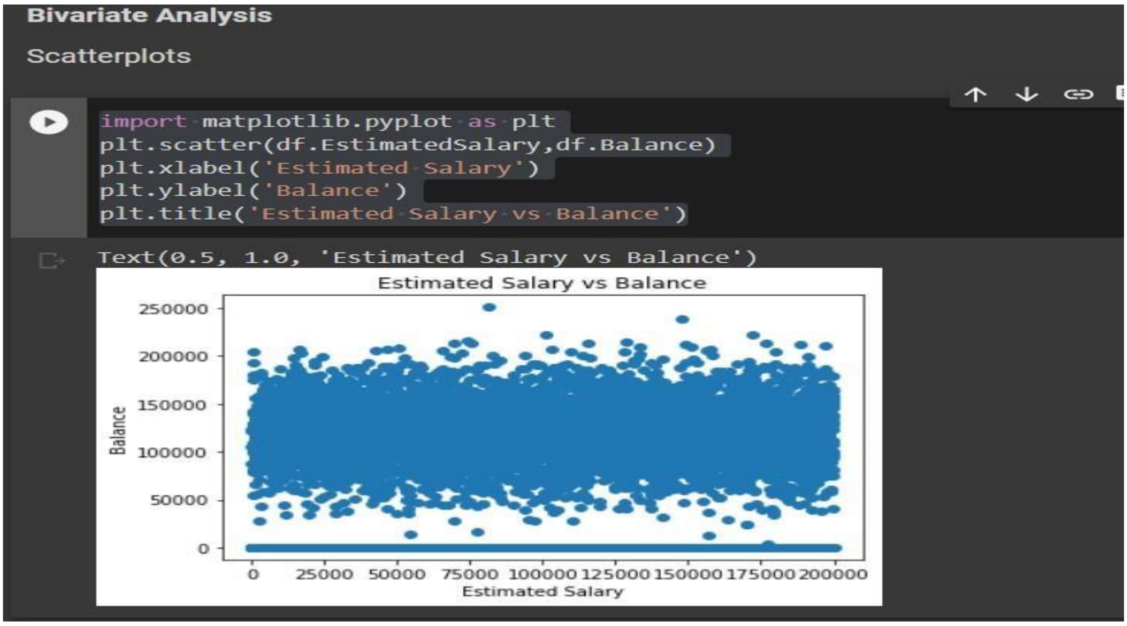
1.Scatterplot import matplotlib.pyplot as plt

plt.scatter(df.EstimatedSalary,df.Balance)

plt.xlabel('Estimated Salary')

plt.ylabel('Balance')

plt.title('Estimated Salary vs Balance')



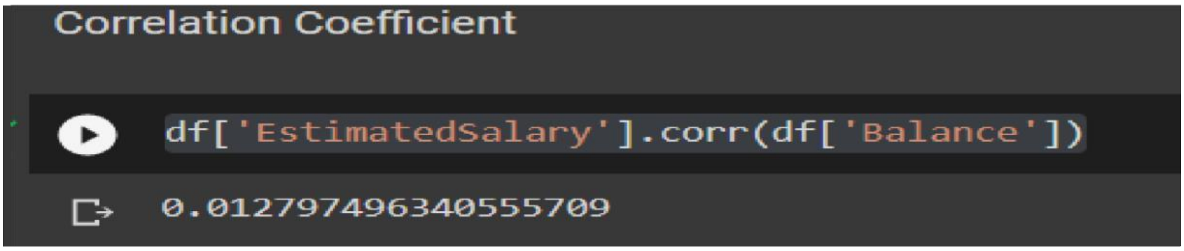
Bivariate Analysis

Scatterplots

```
import matplotlib.pyplot as plt
plt.scatter(df.EstimatedSalary,df.Balance)
plt.xlabel('Estimated Salary')
plt.ylabel('Balance')
plt.title('Estimated Salary vs Balance')
```

```
Text(0.5, 1.0, 'Estimated Salary vs Balance')
```

2. Correlation Coefficient
df['EstimatedSalary'].corr(df['Balance'])



Correlation Coefficient

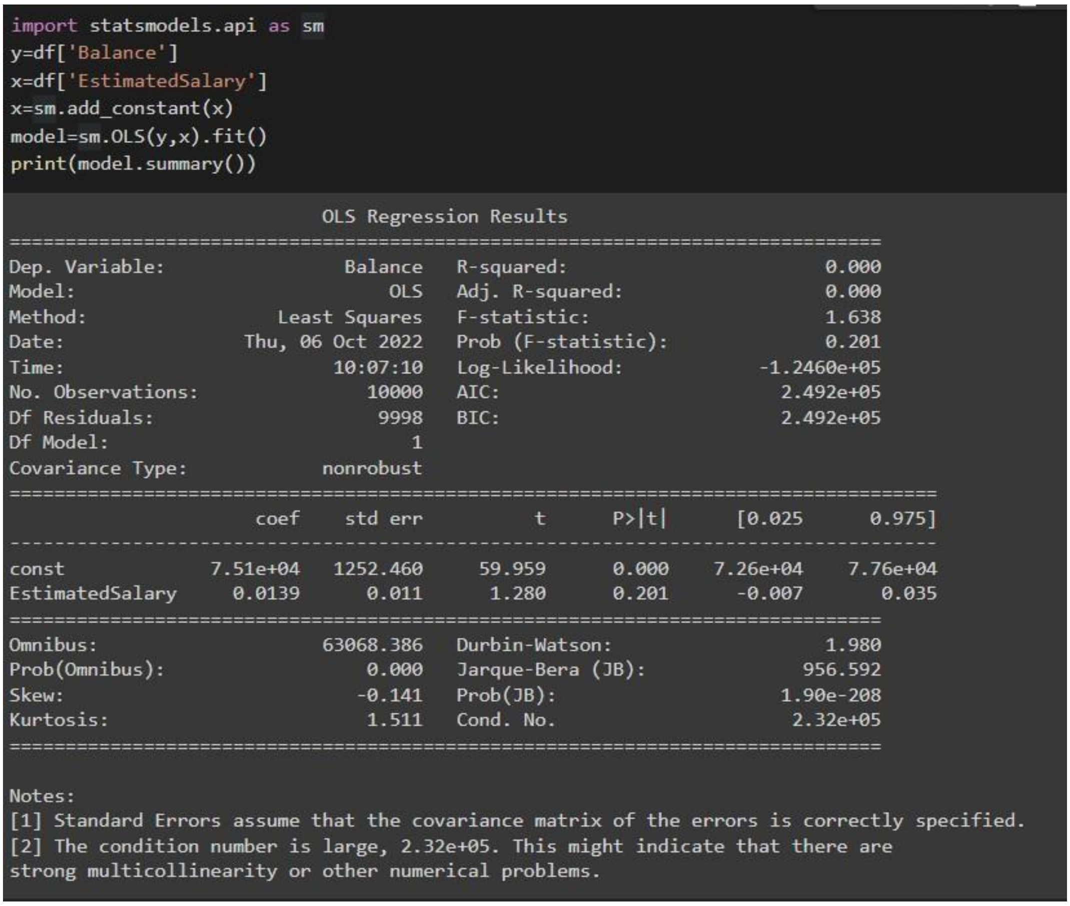
```
df['EstimatedSalary'].corr(df['Balance'])
```

```
0.012797496340555709
```

3. Simple Linear Regression
   import statsmodels.api as sm
y=df['Balance']

x=df['EstimatedSalary']   x=sm.add_constant(x)
model=sm.OLS(y,x).fit()

   print(model.summary())

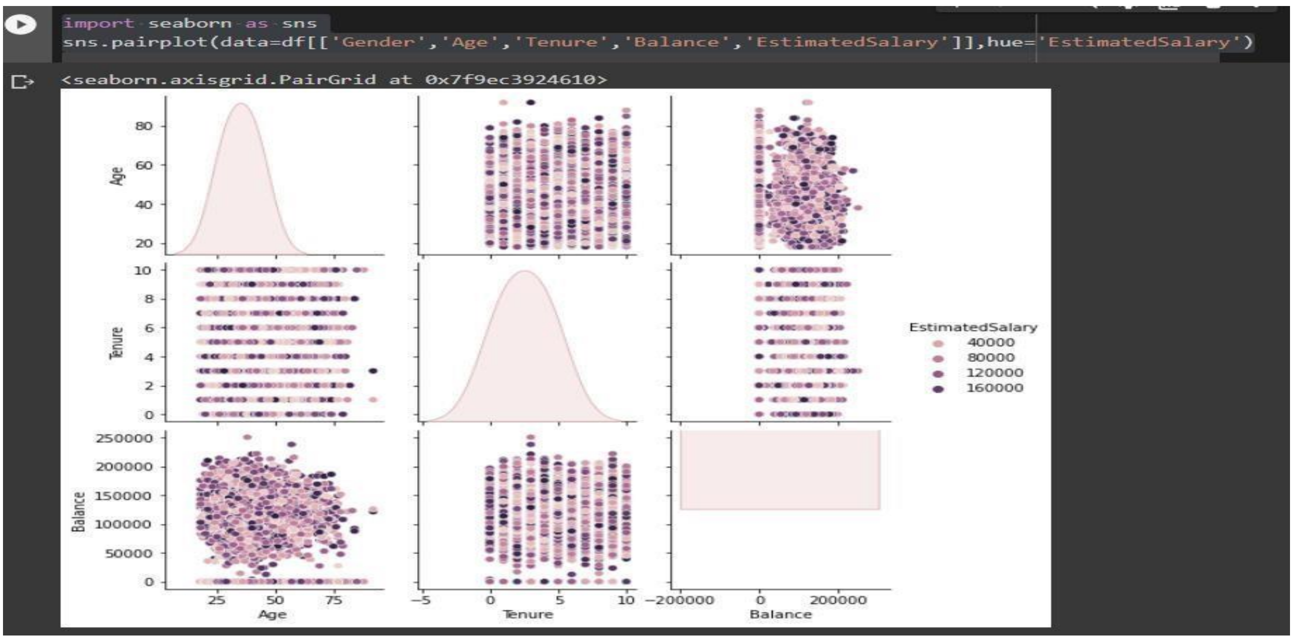


```
import statsmodels.api as sm
y=df['Balance']
x=df['EstimatedSalary']
x=sm.add_constant(x)
model=sm.OLS(y,x).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Balance   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                     1.638
Date:                Thu, 06 Oct 2022   Prob (F-statistic):              0.201
Time:                        10:07:10   Log-Likelihood:            -1.2460e+05
No. Observations:               10000   AIC:                         2.492e+05
Df Residuals:                    9998   BIC:                         2.492e+05
Df Model:                           1
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const             7.51e+04   1252.460     59.959      0.000    7.26e+04    7.76e+04
EstimatedSalary     0.0139      0.011      1.280      0.201      -0.007       0.035
==============================================================================
Omnibus:                    63068.386   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              956.592
Skew:                          -0.141   Prob(JB):                    1.90e-208
Kurtosis:                       1.511   Cond. No.                     2.32e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.32e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Multivariate Analysis: import seaborn
as sns

sns.pairplot(data=df[['Gender','Age','Tenure','Balance','EstimatedSalary']],hue='EstimatedSal ary')



```
import seaborn as sns
sns.pairplot(data=df[['Gender','Age','Tenure','Balance','EstimatedSalary']],hue='EstimatedSalary')
```

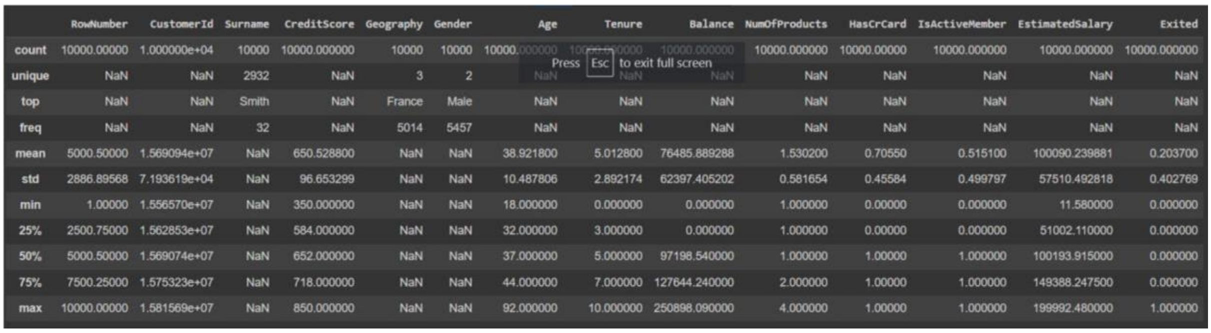
```
<seaborn.axisgrid.PairGrid at 0x7f9ec3924610>
```

Question-4:

Perform descriptive statistics on the dataset.

Solution:

df.describe(include='all')



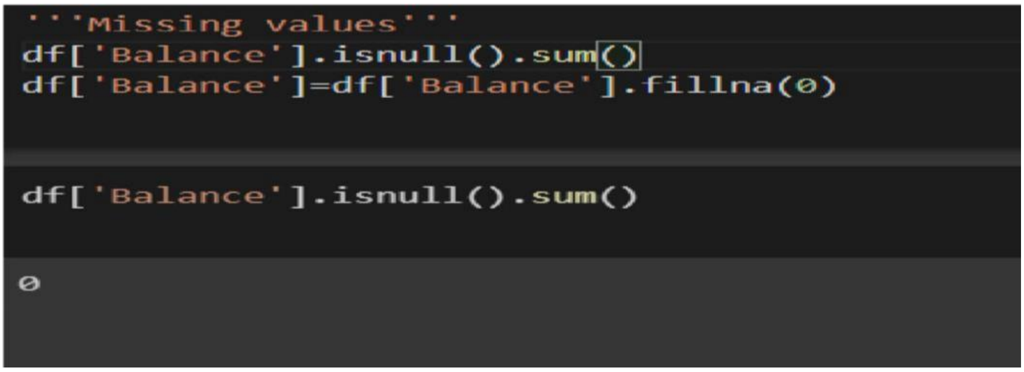
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000	10000.000000	10000	10000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
unique	NaN	NaN	2932	NaN	3	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	Smith	NaN	France	Male	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	32	NaN	5014	5457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	5000.50000	1.569094e+07	NaN	650.528800	NaN	NaN	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	NaN	96.653299	NaN	NaN	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	NaN	350.000000	NaN	NaN	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	NaN	584.000000	NaN	NaN	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	NaN	652.000000	NaN	NaN	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	NaN	718.000000	NaN	NaN	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	NaN	850.000000	NaN	NaN	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

Press Esc to exit full screen

Question-5:

Handle the Missing values.

Solution: df['Balance'].isnull().sum() df['Balance']=df['Balance'].fillna(0)



```
'''Missing values'''
df['Balance'].isnull().sum()
df['Balance']=df['Balance'].fillna(0)
```

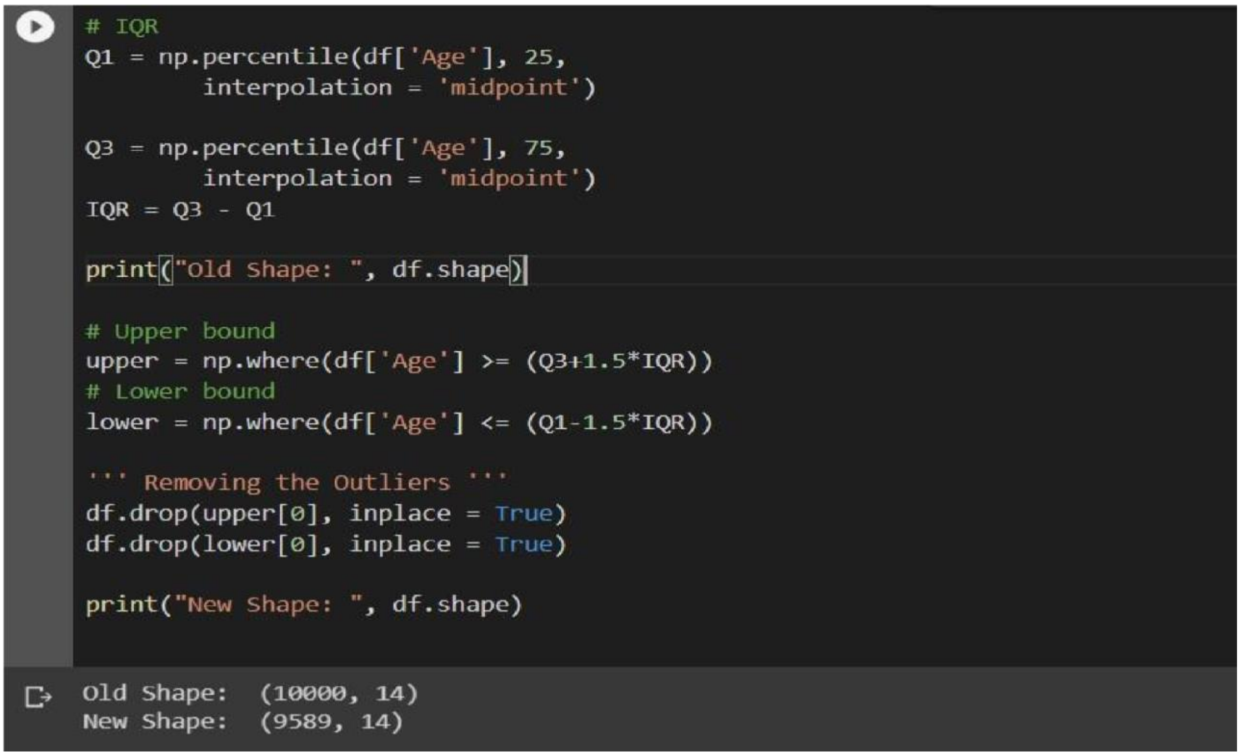
```
df['Balance'].isnull().sum()
```

```
0
```

Question-6:

Find the outliers and replace the outliers Solution:

```
# IQR
Q1 = np.percentile(df['Age'], 25,interpolation = 'midpoint')
Q3 = np.percentile(df['Age'], 75,interpolation = 'midpoint')
IQR = Q3 - Q1  print("Old Shape: ", df.shape)
# Upper bound  upper = np.where(df['Age']
>= (Q3+1.5*IQR))
# Lower bound  lower = np.where(df['Age']
<= (Q1-1.5*IQR))
''' Removing the Outliers ''' df.drop(upper[0], inplace
= True)
df.drop(lower[0], inplace = True)

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



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

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

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

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

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

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

```
Old Shape:  (10000, 14)
New Shape:  (9589, 14)
```

Question-7:

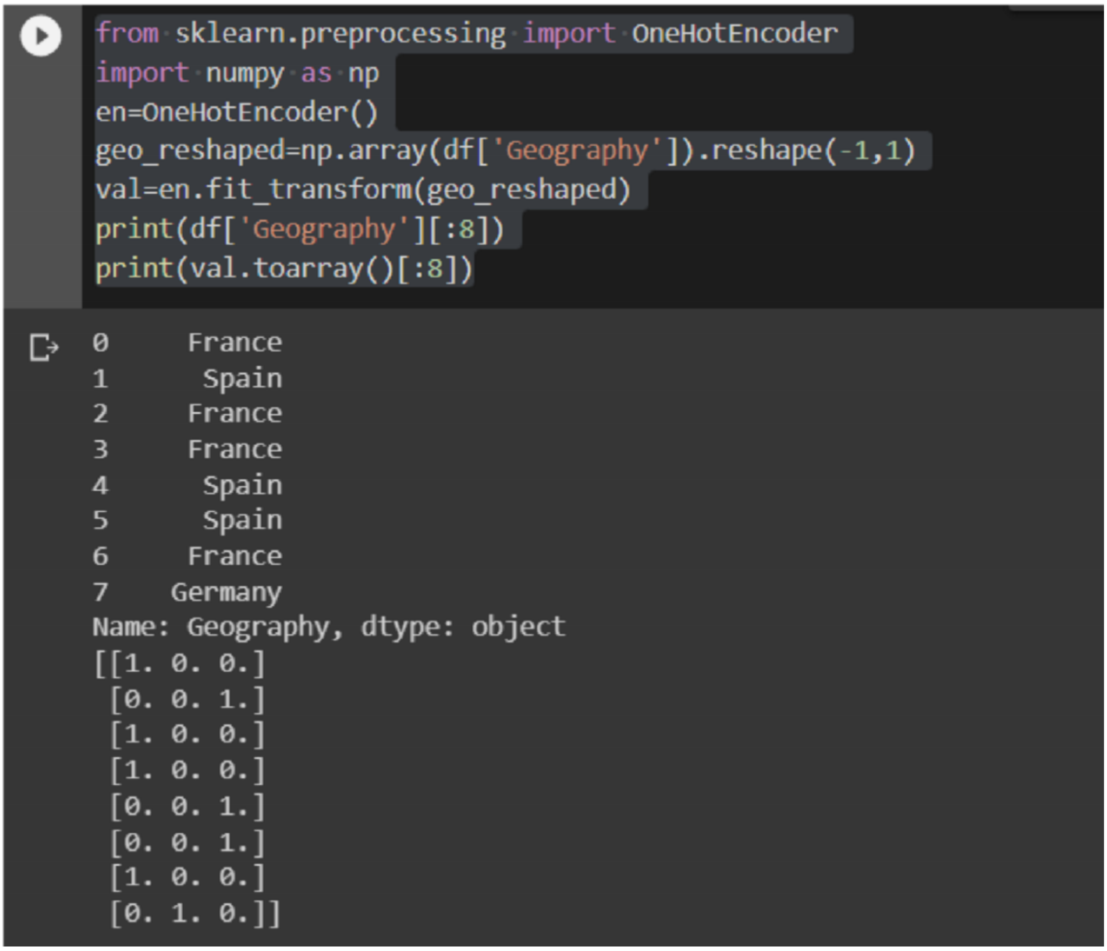
Check for Categorical columns and perform encoding

Solution:

from sklearn.preprocessing import OneHotEncoder

import numpy as np en=OneHotEncoder()

geo_reshaped=np.array(df['Geography']).reshape(-1,1) val=en.fit_transform(geo_reshaped)
print(df['Geography'][:8]) print(val.toarray()[:8])



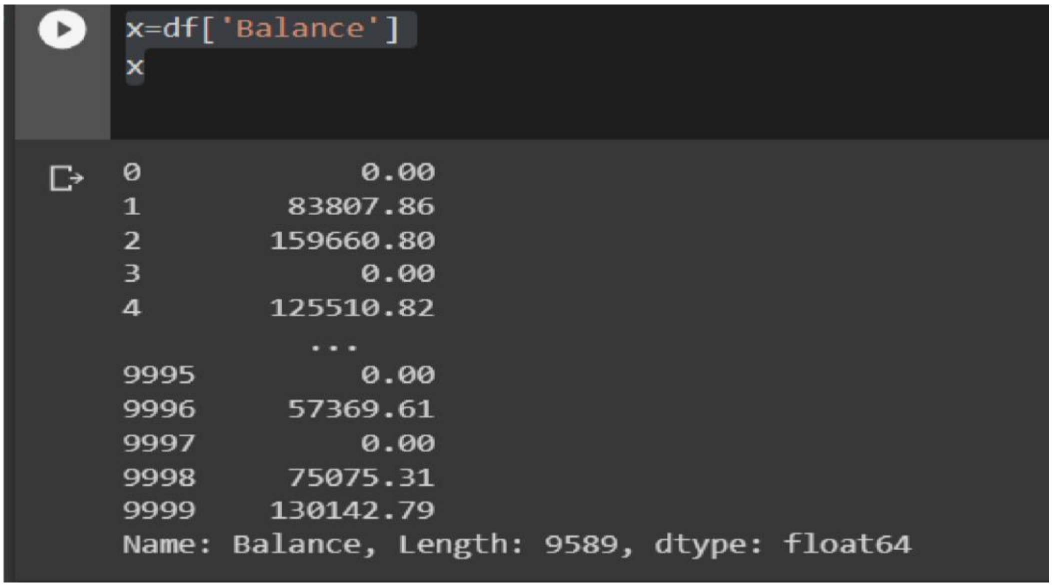
```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
en=OneHotEncoder()
geo_reshaped=np.array(df['Geography']).reshape(-1,1)
val=en.fit_transform(geo_reshaped)
print(df['Geography'][:8])
print(val.toarray()[:8])
```

```
0     France
1      Spain
2     France
3     France
4      Spain
5      Spain
6     France
7    Germany
Name: Geography, dtype: object
[[1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]]
```

Question-8:

Split the data into dependent and independent variables.

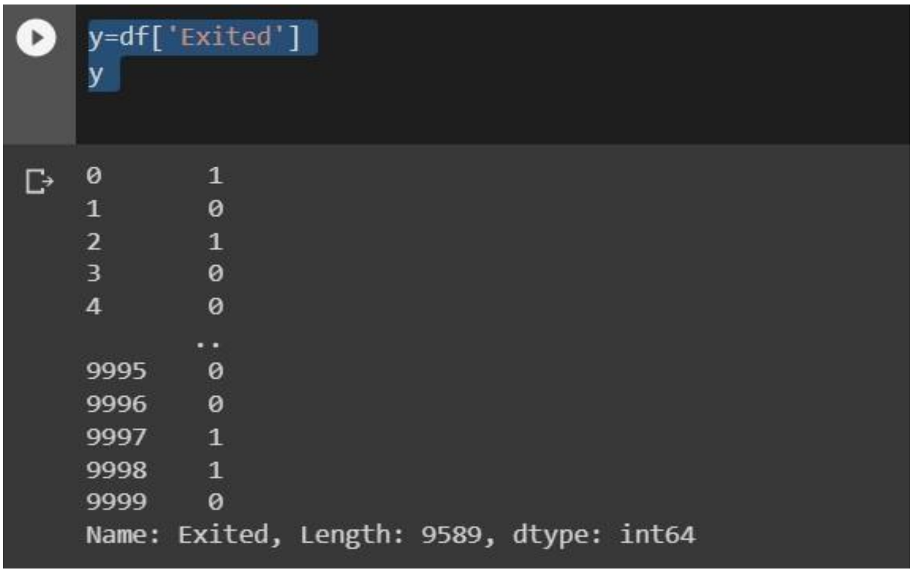
Solution: x=df['Balance']
x



```
x=df['Balance']
x
```

```
0            0.00
1        83807.86
2       159660.80
3            0.00
4       125510.82
          ...
9995         0.00
9996     57369.61
9997         0.00
9998     75075.31
9999    130142.79
Name: Balance, Length: 9589, dtype: float64
```

y=df['Exited'] y



```
y=df['Exited']
y
```

```
0       1
1       0
2       1
3       0
4       0
       ..
9995    0
9996    0
9997    1
9998    1
9999    0
Name: Exited, Length: 9589, dtype: int64
```

Question-9:

Scale the independent variables

Solution:

from sklearn.preprocessing import StandardScaler

x = df[‘Balance’] scaler=StandardScaler()

x=scaler.fit_transform(x)

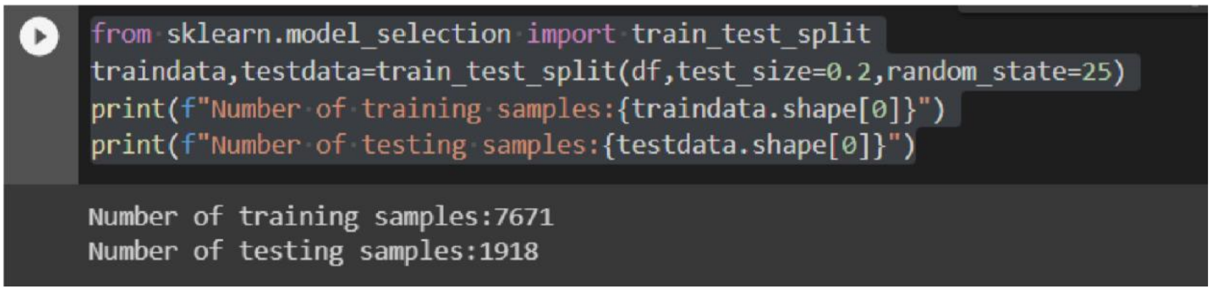
Question-10:

Split the data into training and testing

Solution:

from sklearn.model_selection import train_test_split traindata,testdata=train_test_split(df,test_size=0.2,random_state=25)

print(f"Number of training samples:{traindata.shape[0]}") print(f"Number of testing samples:{testdata.shape[0]}")



```
from sklearn.model_selection import train_test_split
traindata,testdata=train_test_split(df,test_size=0.2,random_state=25)
print(f"Number of training samples:{traindata.shape[0]}")
print(f"Number of testing samples:{testdata.shape[0]}")
```

```
Number of training samples:7671
Number of testing samples:1918
```