Assignment -2

Data Visualization and Pre-processing

Assignment Date	28 September 2022
Student Name	Manikandan.R
Student Roll Number	820419104031
Maximum Marks	2 Marks

1.

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

Matplotlib is building the font cache; this may take a moment.

2. Load the dataset.

Solution:

```
data = pd.read_csv("Churn_Modelling.csv")
data.head()
```

Output:

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

3. Perform Below Visualizations.

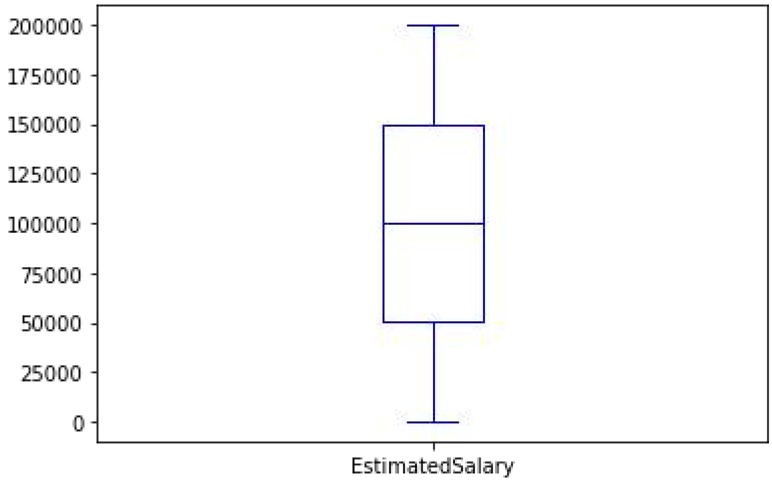
Univariate Analysis

Solution:

```
data.boxplot(column=['EstimatedSalary'], grid=False, color='blue')
```

Output:

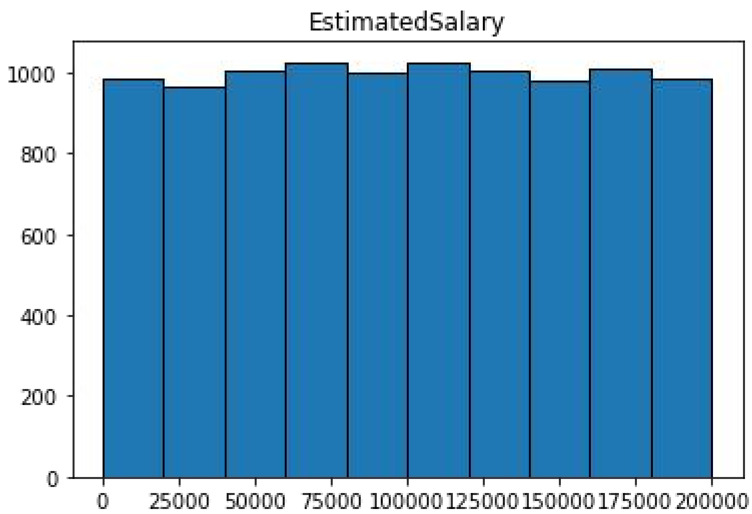
<AxesSubplot:>



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

Output:

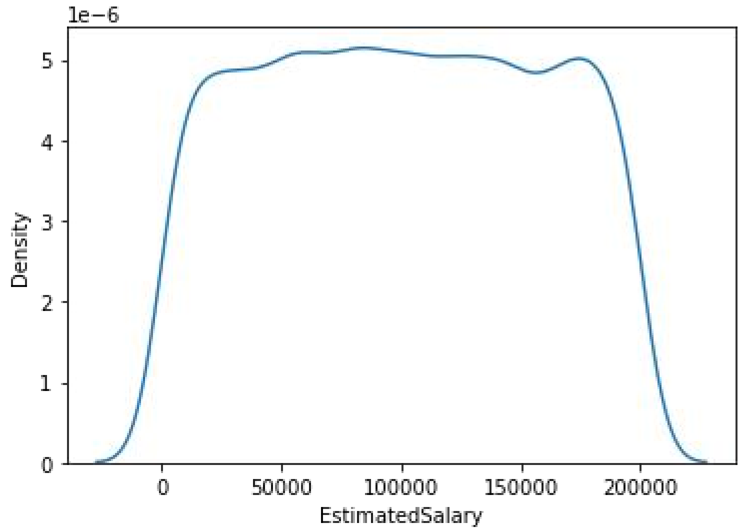
array([[<AxesSubplot:title={'center':'EstimatedSalary'}>]], dtype=object)



sns.kdeplot(data['EstimatedSalary'])

Output:

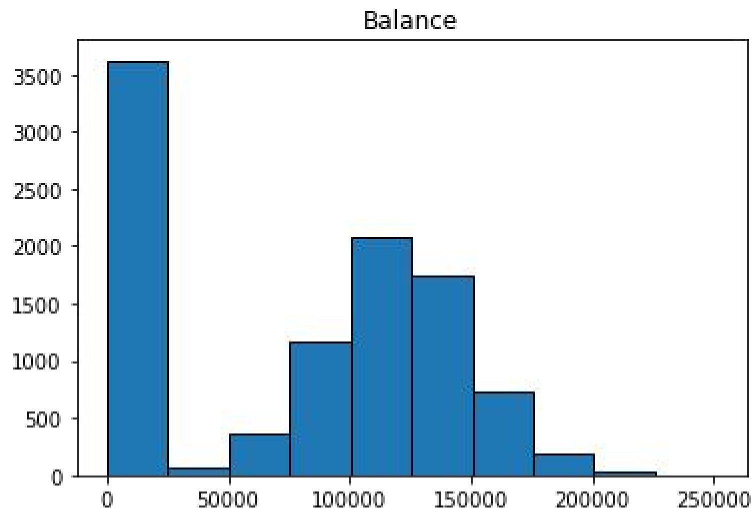
<AxesSubplot:xlabel='EstimatedSalary', ylabel='Density'>




```
data.hist(column='Balance', grid=False, edgecolor='black')
```

Output:

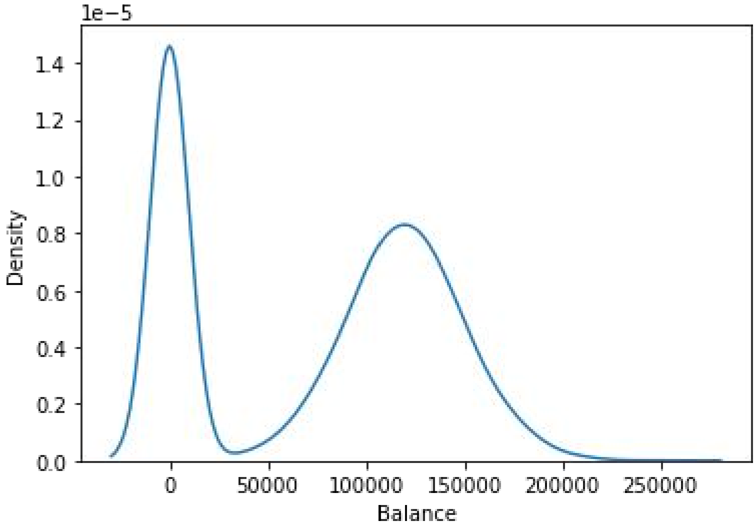
array([[<AxesSubplot:title={'center':'Balance'}>]], dtype=object)




```
import seaborn as sns
sns.kdeplot(data['Balance'])
```

Output:

<AxesSubplot:xlabel='Balance', ylabel='Density'>



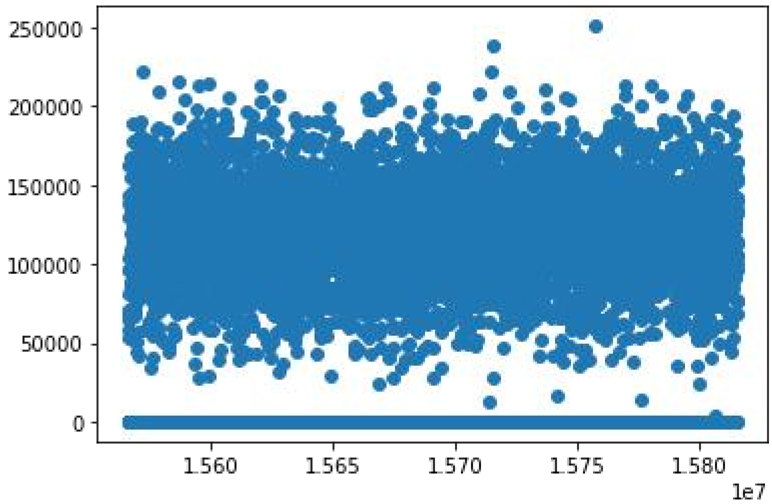
Bi - Variate Analysis

Solution:

```
plt.scatter(data.CustomerId, data.Balance)
```

Output:

<matplotlib.collections.PathCollection at 0xc33f130>




```
data.corr()
```

Output:

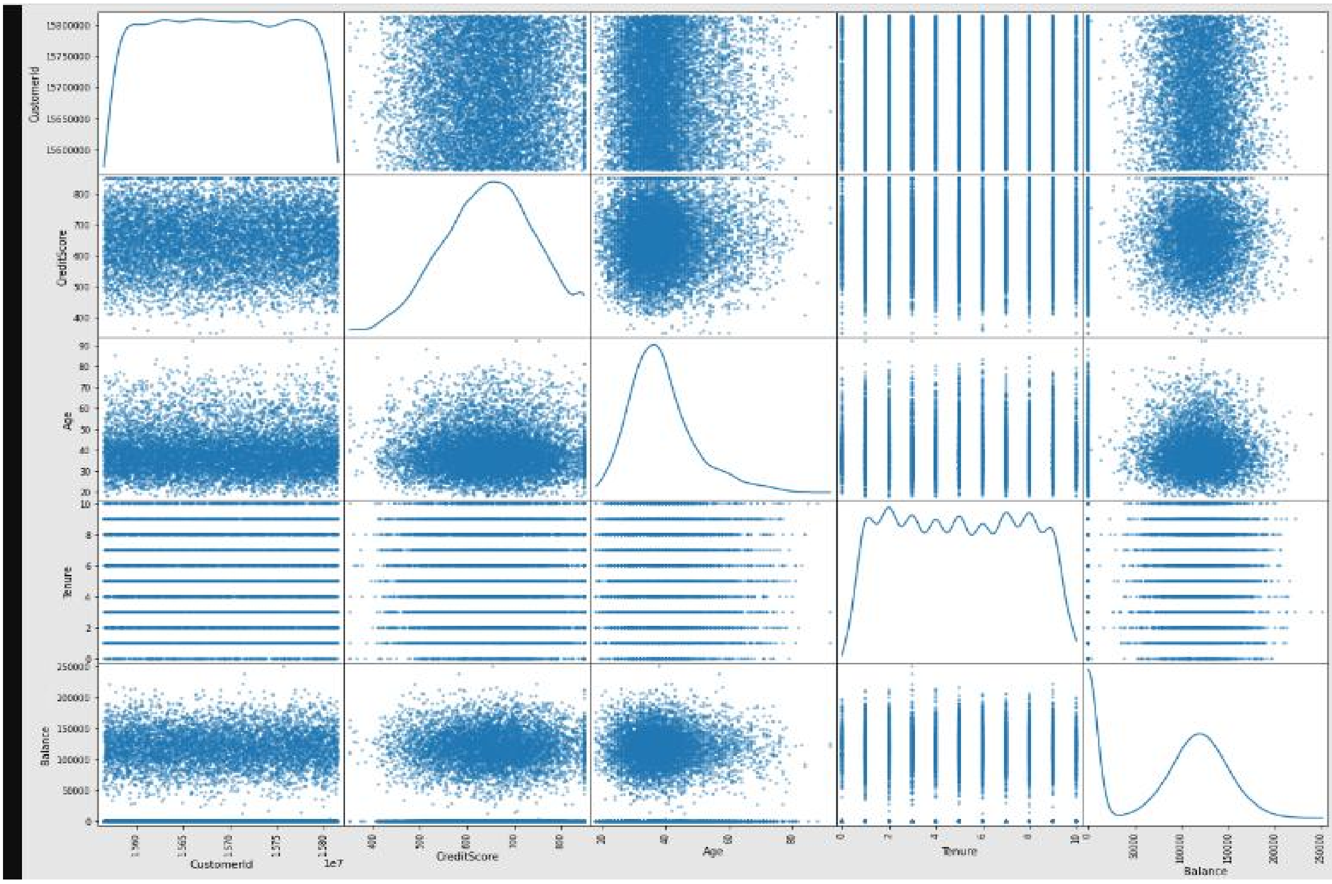
	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
RowNumber	1.000000	0.004202	0.005840	0.000783	-0.006495	-0.009067	0.007246	0.000599	0.012044	-0.005988	-0.016571
CustomerId	0.004202	1.000000	0.005308	0.009497	-0.014883	-0.012419	0.016972	-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

Multi - Variate Analysis

Solution:

```
pd.plotting.scatter_matrix(data.loc[:, "CustomerId":"Balance"],
diagonal="kde",figsize=(20,15))
plt.show()
```

Output:



4. Perform descriptive statistics on the dataset

Solution:

```
data[['CreditScore', 'Balance', 'EstimatedSalary']].mean()
```

Output:

```
CreditScore            650.528800
Balance              76485.889288
EstimatedSalary     100090.239881
dtype: float64
```

```
data[['CreditScore', 'Balance', 'EstimatedSalary']].median()
```

Output:

```
: CreditScore            652.000
  Balance              97198.540
  EstimatedSalary     100193.915
  dtype: float64
```

```
data[['CreditScore', 'Balance', 'EstimatedSalary']].mode()
```

Output:

	CreditScore	Balance	EstimatedSalary
0	850	0.0	24924.92

```
data[['CreditScore', 'Balance', 'EstimatedSalary']].quantile()
```

Output:

```
CreditScore            652.000
Balance              97198.540
EstimatedSalary     100193.915
Name: 0.5, dtype: float64
```

```
data[['CreditScore', 'Balance', 'EstimatedSalary']].std()
```

Output:

```
CreditScore            96.653299
Balance              62397.405202
EstimatedSalary      57510.492818
dtype: float64
```

```
data[['CreditScore', 'Balance', 'EstimatedSalary']].min()
```

Output:

```
CreditScore         350.00
Balance               0.00
EstimatedSalary      11.58
dtype: float64
```

data[['CreditScore', 'Balance', 'EstimatedSalary']].max()

Output:

```
: CreditScore           850.00
  Balance            250898.09
  EstimatedSalary    199992.48
  dtype: float64
```

data[['CreditScore', 'Balance', 'EstimatedSalary']].skew()

Output:

```
: CreditScore       -0.071607
  Balance           -0.141109
  EstimatedSalary    0.002085
  dtype: float64
```

data.info()

Output:

```
<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: 976.6+ KB
```

data.shape

Output:

```
(10000, 14)
```

data.describe()

Output:

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

5. Handle the Missing values.

Solution:

There is no missing values

data.isnull().sum()

Output:

```
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
```

6. Find the outliers and replace the outliers.

Solution:

data.describe()

Output:

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

```
numeric_col = ['RowNumberCustomerId','CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Exited']
categorical_col = ['Surname', 'Geography', 'Gender']
```

```
print(data['CreditScore'].skew())
data['CreditScore'].describe()
```

Output:

```
-0.07160660820092675

count    10000.000000
mean       650.528800
std         96.653299
min        350.000000
25%        584.000000
50%        652.000000
75%        718.000000
max        850.000000
Name: CreditScore, dtype: float64
```

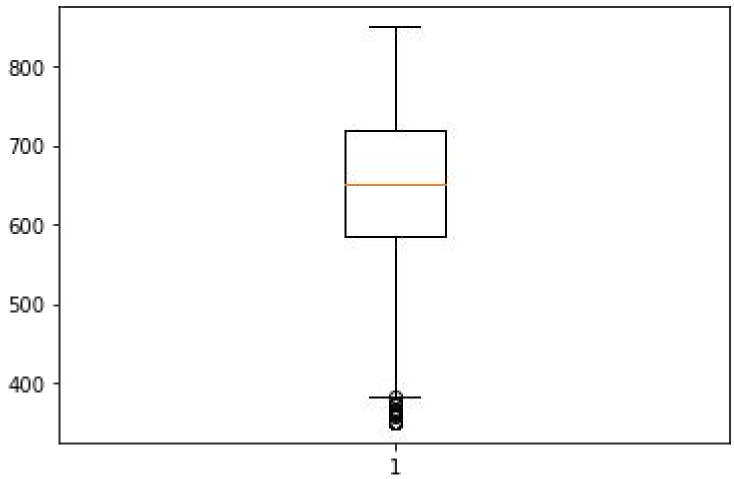
```
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

Output:

```
RowNumber             4999.5000
CustomerId          124705.5000
CreditScore            134.0000
Age                     12.0000
Tenure                   4.0000
Balance             127644.2400
NumOfProducts            1.0000
HasCrCard                1.0000
IsActiveMember           1.0000
EstimatedSalary      98386.1375
Exited                   0.0000
dtype: float64
```

plt.boxplot(data["CreditScore"])
plt.show()

Output:



print(data['CreditScore'].quantile(0.50))
print(data['CreditScore'].quantile(0.95))
data['CreditScore'] = np.where(data['CreditScore'] > 325, 140, data['CreditScore'])
data.describe()

Output:

```
652.0
812.0
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
count	10000.00000	1.000000e+04	10000.0	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	140.0	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881
std	2886.89568	7.193619e+04	0.0	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818
min	1.00000	1.556570e+07	140.0	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000
25%	2500.75000	1.562853e+07	140.0	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000
50%	5000.50000	1.569074e+07	140.0	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000
75%	7500.25000	1.575323e+07	140.0	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500
max	10000.00000	1.581569e+07	140.0	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000

7. Check for Categorical columns and perform encoding

Solution:

```
X = data.iloc[:, 10:20].values
y = data.iloc[:, 13].values
```

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

labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

# remove categorical_features, it works 100% perfectly
onehotencoder = OneHotEncoder()
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
```

8. Split the data into dependent and independent variables.

Solution:

```
X= data.iloc[:,3:-1]
y=data.iloc[:,-1]
X.head()
```

Output:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	140	France	Female	42	2	0.00	1	1	1	101348.88
1	140	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	140	France	Female	42	8	159660.80	3	1	0	113931.57
3	140	France	Female	39	1	0.00	2	0	0	93826.63
4	140	Spain	Female	43	2	125510.82	1	1	1	79084.10

```
X = data.iloc[:, 10:20].values
y = data.iloc[:, 13].values
```

print(X)

Output:

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

[

print(y)

```
[1 0 1 ... 1 1 0]
```

9. Scale the independant variables

Solution:

from sklearn.preprocessing **import** StandardScaler

sc **=** StandardScaler()x_train **=** sc**.**fit_transform(x_train)x_test **=** sc**.**fit_transform(x_test)

x_train **=** pd**.**DataFrame(x_train)x_train**.**head()

Output:

	0	1	2	3
0	-1.553624	-1.034460	-1.640810	1.976962
1	0.643657	-1.034460	-0.079272	-0.505827
2	0.643657	0.966688	-0.996840	-0.505827
3	0.643657	0.966688	-1.591746	1.976962
4	0.643657	0.966688	1.283302	-0.505827

10. Split the data into training and testing

Solution:

from sklearn.model_selection **import** train_test_splitx_train, x_test, y_train, y_test **=** train_test_split(X, y, test_size **=** 0.25, random_state **=** 0)

print(x_train**.**shape)

print(y_train**.**shape)

print(x_test**.**shape)

print(y_test**.**shape)

Output:

```
(7500, 4)
(7500,)
(2500, 4)
(2500,)
```