Assignment -2

Data Visualization and Pre-processing

Assignment Date	3 October 2022
Student Name	GOWTHAM N
Student Roll Number	737819ECR043

Tasks:-

1. Download the dataset: Dataset
2. Load the dataset.

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files  Churn_Mod...)_ass_2.csv
• Churn_Modelling (1)_ass_2.csv(text/csv) - 684858 bytes, last modified: 9/21/2022 - 100% done
Saving Churn_Modelling (1)_ass_2.csv to Churn_Modelling (1)_ass_2.csv
```

3. Perform Below Visualizations.

● Univariate Analysis

If we analyse data over a single variable/column from a dataset, it is known as Univariate Analysis.

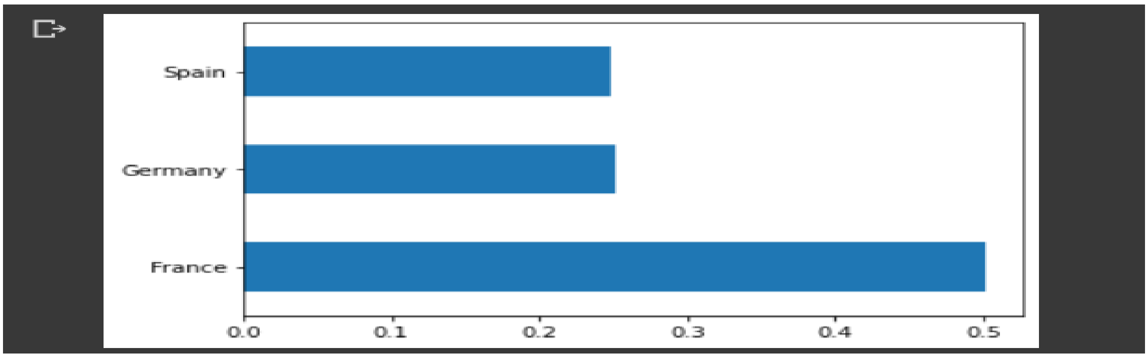
Now, let's analyse the geography category by using plots. Since Geography is a category, we will plot the bar plot.

```
[6] df=pd.read_csv("Churn_Modelling (1)_ass_2.csv")

Series: df.Geography
Series with shape (10000,) and dtype object

df.Geography.value_counts(normalize=True).plot.barh()
plt.show()
```

The output looks likes this,



By the above bar plot, we can infer that the data set contains more number of France peoples are there compared to other categories.

- Bi - Variate Analysis

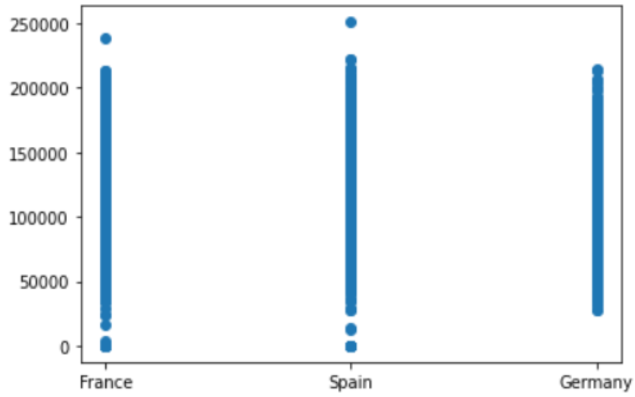
Analysing the two numeric variables from a dataset is known as numeric-numeric analysis.

Let's take three columns 'Balance', 'Age' and 'Geography' from our dataset and see what we can infer by plotting to scatter plot between Geography balance and Age balance.

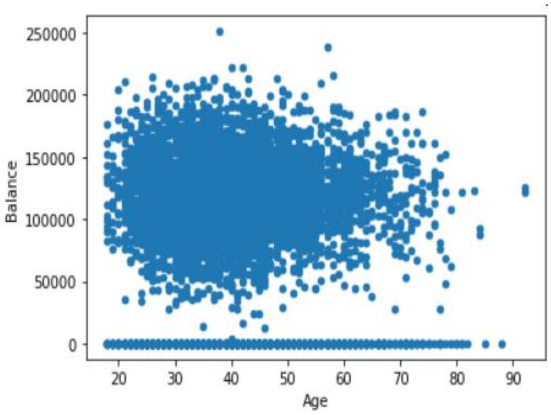
```
#plot the scatter plot of Geography and Balance variable in data
plt.scatter(df.Geography,df.Balance)
plt.show()

#plot the scatter plot of age and Balance variable in data
df.plot.scatter(x="Age",y="Balance")
plt.show()
```

The output looks like this,



Geography vs balance



Age vs balance

- Multi - Variate Analysis

Multivariate analysis is based in observation and analysis of more than one statistical outcome variable at a time.

Let's take columns from 'Geography' to 'Balance' from our dataset and see what we can infer by plotting to scatter plot.

Code for taking required columns from our dataset, and output for that code

```
df.loc[:, "Geography":"Balance"]
```

	Geography	Gender	Age	Tenure	Balance
0	France	Female	42	2	0.00
1	Spain	Female	41	1	83807.86
2	France	Female	42	8	159660.80
3	France	Female	39	1	0.00
4	Spain	Female	43	2	125510.82
...	...	...	...	...	...
9995	France	Male	39	5	0.00
9996	France	Male	35	10	57369.61
9997	France	Female	36	7	0.00
9998	Germany	Male	42	3	75075.31
9999	France	Female	28	4	130142.79

10000 rows × 5 columns

To make a matrix scatterplot of just these 5 variables using the scatter_matrix() function we type:

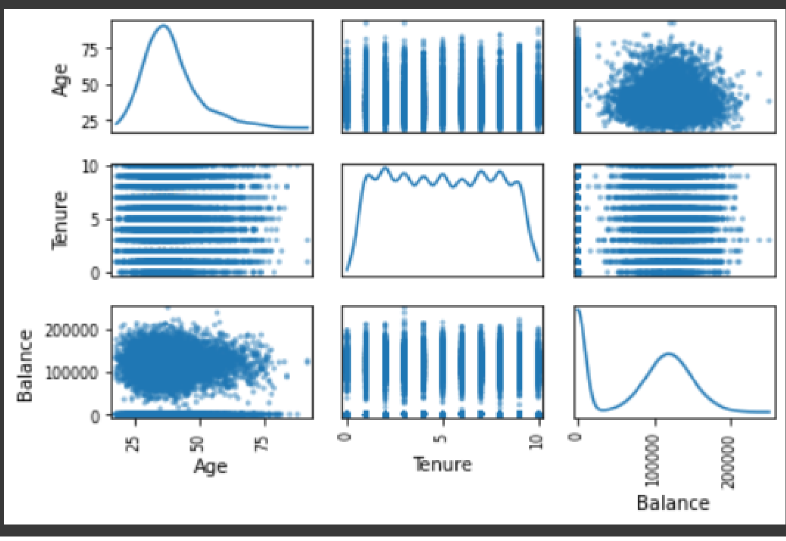
pd.plotting.scatter_matrix(df.loc[:, "Geography":"Balance"], diagonal="kde")

plt.tight_layout()

plt.show()

The output looks like this,

```
pd.plotting.scatter_matrix(df.loc[:, "Geography":"Balance"], diagonal="kde")
plt.tight_layout()
plt.show()
```



4. Perform descriptive statistics on the dataset.

The describe() function computes a summary of statistics pertaining to the Data Frame columns.

The output of descriptive statistics on the dataset look like,

```
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

5. Handle the Missing values.

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not.

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values.

```
df.isnull()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	True	True	True	True	True	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9996	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9997	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9998	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9999	True	True	True	True	True	True	True	True	True	True	True	True	True	True

10000 rows × 14 columns

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values.

```
df.notnull()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	True	True	True	True	True	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9996	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9997	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9998	True	True	True	True	True	True	True	True	True	True	True	True	True	True
9999	True	True	True	True	True	True	True	True	True	True	True	True	True	True

10000 rows × 14 columns

Filling missing values using fillna(), replace() and interpolate():

In order to fill null values in a datasets, we use fillna(), replace() and interpolate() function these function replace NaN values with some value of their own.

Filling null values with a single value

```
df.fillna(0)
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 14 columns

Filling null values with the previous ones

```
df.fillna(method ='pad')
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 14 columns

Filling null value with the next ones

```
df.fillna(method ='bfill')
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

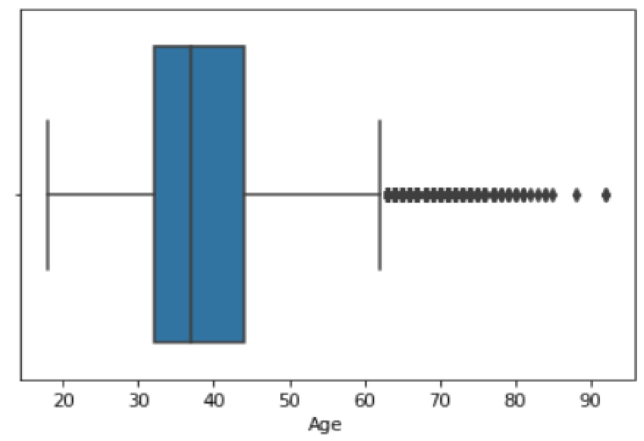
10000 rows × 14 columns

```
6. Find the outliers and replace the outliers
```

Outliers can be detected using visualization, Using Box Plot - It captures the summary of the data effectively and efficiently with only a simple box and whiskers. Boxplot summarizes sample data using 25th, 50th, and 75th percentiles. One can just get insights(quartiles, median, and outliers) into the dataset by just looking at its boxplot.

```
# Box Plot
import seaborn as sns
sns.boxplot(df['Age'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fe09fec2190>
```



In the above graph, can clearly see that values above 60 are acting as the outliers. The outliers indexes are shown below,

```
# Position of the Outlier
print(np.where(df['Age']>60))
```

```
(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,
```

Replacing of outliers is done by .loc function, all the outliers are replaced with the max age value as 60. The output looks like,

```
df.Age.loc[df.Age > 60]  = 60
df.describe()
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1732: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
  self._setitem_single_block(indexer, value, name)
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.573300	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	9.543906	2.892174	62397.405202
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	60.000000	10.000000	250898.090000

7. Check for Categorical columns and perform encoding.

Categorical encoding is a process of converting categories to numbers.

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Understanding the datatypes of each columns:

```
print(df.dtypes)
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

As you can see here, Geography, is the categorical feature as it is represented by the object data type. Now, let us implement label encoding:

```
# Import label encoder
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'Geography'.
df['Geography']= label_encoder.fit_transform(df['Geography'])
print(df.head())
```

```
   RowNumber  CustomerId   Surname  CreditScore  Geography  Gender  Age  \
0          1    15634602  Hargrave          619          0  Female   42
1          2    15647311      Hill          608          2  Female   41
2          3    15619304      Onio          502          0  Female   42
3          4    15701354      Boni          699          0  Female   39
4          5    15737888  Mitchell          850          2  Female   43
```

As you can see here, label encoding uses alphabetical ordering. Hence, France has been encoded with 0, the Germany with 1, and Spain with 2.

8. Split the data into dependent and independent variables.

For dependent variable X, it takes all the rows in the dataset and it takes all the columns up to the one before the last column.

```
[100] #Splitting the Dataset into the Independent Feature Matrix:
      X = df.iloc[:, :-1].values
      print(X)

      [[1 15634602 'Hargrave' ... 1 1 101348.88]
       [2 15647311 'Hill' ... 0 1 112542.58]
       [3 15619304 'Onio' ... 1 0 113931.57]
       ...
       [9998 15584532 'Liu' ... 0 1 42085.58]
       [9999 15682355 'Sabbatini' ... 1 0 92888.52]
       [10000 15628319 'Walker' ... 1 0 38190.78]]
```

For independent variable Y, it takes all the rows, but only column 4 from the dataset.

```
[101] #Extracting the Dataset to Get the Dependent Vector
      Y = df.iloc[:, -1].values
      print(Y)

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

9. Scale the independent variables

When a dataset has values of different columns at drastically different scales, it gets tough to analyze the trends and patterns and comparison of the features or columns. So, in cases where all the columns have a significant difference in their scales, are needed to be modified in such a way that all those values fall into the same scale. This process is called Scaling.

Min-Max Normalization

Here, all the values are scaled in between the range of [0,1] where 0 is the minimum value and 1 is the maximum value. The age and customer ID columns are scaled in below figure,

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['Age', 'CustomerId']] = scaler.fit_transform(df[['Age', 'CustomerId']])
df
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure
0	1	0.275616	Hargrave	619	0	Female	0.571429	2
1	2	0.326454	Hill	608	2	Female	0.547619	1
2	3	0.214421	Onio	502	0	Female	0.571429	8
3	4	0.542636	Boni	699	0	Female	0.500000	1
4	5	0.688778	Mitchell	850	2	Female	0.595238	2
...	...	...	...	...	...	...	...	...
9995	9996	0.162119	Obijiaku	771	0	Male	0.500000	5
9996	9997	0.016765	Johnstone	516	0	Male	0.404762	10
9997	9998	0.075327	Liu	709	0	Female	0.428571	7
9998	9999	0.466637	Sabbatini	772	1	Male	0.571429	3
9999	10000	0.250483	Walker	792	0	Female	0.238095	4

10000 rows × 14 columns

10. Split the data into training and testing

training set—a subset to train a model

test set—a subset to test the trained model

1. Loading the dataset
2. Splitting
   Let's split this data into labels and features. Now, what's that? Using features, we predict labels. I mean using features (the data we use to predict labels), we predict labels (the data we want to predict).
   Balance is a label to predict balance in y; we use the drop() function to take all other data in x. Then, we split the data.

```
[121] import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.datasets import load_iris
      df.head()
      y=df.Balance
      x=df.drop('Balance',axis=1)
      x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
      x_train.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
6087	6088	0.660261	Chukwudi	561	0	Female	0.214286	9	1	1	0	153080.40	1
2488	2489	0.021789	Baranov	645	2	Female	0.071429	1	2	0	0	28726.07	0
4052	4053	0.210809	Douglas	616	1	Male	0.547619	10	2	1	1	114072.91	0
2490	2491	0.269772	Robinson	696	1	Female	0.404762	4	1	1	0	69079.85	0
714	715	0.967675	Yuan	650	2	Female	0.166667	3	3	1	0	16649.31	1

```
x_train.shape
```

```
(8000, 13)
```

```
x_test.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
9289	9290	0.444456	Jen	561	0	Female	0.309524	1	2	1	1	65234.60	0
2687	2688	0.453808	Oliver	508	0	Male	0.619048	3	2	0	0	67234.33	0
3383	3384	0.400278	T'ang	698	1	Male	0.500000	9	2	0	1	53289.49	0
745	746	0.297445	Smith	606	0	Male	0.523810	5	2	1	1	70899.27	0
772	773	0.294465	Cartwright	589	0	Male	0.333333	2	2	0	1	9468.64	0

```
x_test.shape
```

```
(2000, 13)
```

The line test_size=0.2 suggests that the test data should be 20% of the dataset and the rest should be train data. With the outputs of the shape() functions, you can see that we have 2000 rows in the test data and 8000 in the training data.