<u>Report: Data Wrangling on Bank Customer Churn dataset</u>

The Dataset is about bank customers churning and can be found on Kaggle:

https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

Disclaimer: The dataset above is simulated.

Before I started my notebook, I made sure that the dataset was in the same directory as my notebook. The first thing that I did after reading in the csv file was to check out the shape of the dataset. As we can see below, the bank customer churn dataset is of the shape 10000 rows and 14 columns.

[58]:
```python
# Load the dataset from local directory into a Pandas dataframe called 'df'
df = pd.read_csv('Churn_Modelling.csv', index_col=None)
```

[59]:
```python
# View the shape of the data using .shape
df.shape
```

[59]:
```
(10000, 14)
```

I wanted to see what the data looks like so I used the **df.head()** function to get a glimpse into the first 5 rows of the dataset. It can be seen that of the 14 columns, 13 columns are feature columns and the '***Exited***' column is the response column.

[61]:
```python
# View the data
df.head()
```

[61]:

	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

Now that we know what the data looks like, the next thing I will check is if there are any null values in our data. To check for null values in the columns, I called the function **.isnull().any()** which would return **True** for any column that contains a null value.

Fortunately, our dataset does not contain any null values as it can be seen below and this is because the dataset was from Kaggle, and it was already very clean. This is not often the case with real world data.

```
[60]: # Check to see if there are any null values in our dataset
      df.isnull().any()
```

```
[60]: 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
```

After checking for null values, a second look at the data reveals to us that the column '**RowNumber**' is redundant and can be taken out.

```
[61]: # View the data
      df.head()
```

[61]:

	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

The function **.drop()** is used on the DataFrame to drop the column **'RowNumber'** and then the data is viewed again using **.head()**. After dropping, this is what our new dataframe looks like. The new shape of the dataframe is 10000 rows and 13 columns.

```
[64]: #Drop the RowNumber column as it is redundant
      df.drop('RowNumber', axis = 1, inplace=True)
```

```
[65]: df.head()
```

[65]:

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

The next thing that needs to be done is converting the categorical columns; 'Gender' and 'Geography' into numerical values. This is done because during modelling, some actions can not be performed on categorical values. To do this, first I call the function **'.value_count()'** on the columns which lists all the unique values and their counts in the column.

```
[66]: print(df['Gender'].value_counts())
      print(df['Geography'].value_counts())
```

```
Male      5457
Female    4543
Name: Gender, dtype: int64
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

Next, I used the **'.replace()'** method to convert the **'Geography'** column into 3 numerical values and the **'Gender'** column into 2 numerical values. After replacing the categorical values with numerical values, this is how the data looks like.

```
[67]: df['Geography'].replace(['France', 'Germany', 'Spain'], [0, 1, 2], inplace=True)
      df['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)
      df.head()
```

[67]:

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

For visual purposes, I moved the response variable column **'Exited'** to the left side of the table. I find it quicker to view the data this way, and also makes splitting the dataset into train/test sets easier at a later stage.

Data Rearrangement

For visual purposes, I like to move the response variable, in this case 'Exited', to the left side of the table. I find it quicker to view it this way, and also makes the dataset splitting into train/test set easier later on.

68]:
```
first_column = df['Exited']
df.drop('Exited', axis=1,inplace=True)
df.insert(0, 'Exited', first_column)
df.head()
```

68]:

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

The last thing that remains to be done is to check for outliers in the data. In this case, we use **'.describe()'** method and look for any extreme values in the min and max fields of the output. For our data, there seems to be no outliers.

[69]:
```
df.describe()
```

[69]:

	Exited	CustomerId	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
count	10000.000000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000
mean	0.203700	1.569094e+07	650.528800	0.746300	0.454300	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881
std	0.402769	7.193619e+04	96.653299	0.827529	0.497932	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818
min	0.000000	1.556570e+07	350.000000	0.000000	0.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000
25%	0.000000	1.562853e+07	584.000000	0.000000	0.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000
50%	0.000000	1.569074e+07	652.000000	0.000000	0.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000
75%	0.000000	1.575323e+07	718.000000	1.000000	1.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500
max	1.000000	1.581569e+07	850.000000	2.000000	1.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000