Bank Customer Churn Analysis

This case study is a part of the Google Data Analytics profissional certificate.

Ask:

- Problem: It is much more expensive to sign in a new client than keeping an existing one. It is advantageous for banks to know what leads a client towards the decision to leave the company.
- Goal: The goal of this analysis is to identify the key reasons and frequency of customers leaving the company, and to explore trends and attempt to find a solution for decreasing customer churn rates.

Churn prevention allows companies to develop loyalty programs and retention campaigns to keep as many customers as possible. The dataset used in this case study comes from kaggle, press here for further information.

Prepare:

First, I'll start by importing the necessary libraries for completing this project, then I'll import the data.

Libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
from sklearn.preprocessing import OneHotEncoder, normalize
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import relu, sigmoid
```

Data:

```
df = pd.read_csv('Customer-Churn-Records.csv')
df = df.drop(['RowNumber', 'Surname'], axis = 1)
df.head()

   CustomerId  CreditScore Geography  Gender  Age  Tenure    Balance

0    15634602          619    France  Female   42       2
0.00  \
```

```
1    15647311          608      Spain  Female   41        1   83807.86

2    15619304          502     France  Female   42        8  159660.80

3    15701354          699     France  Female   39        1       0.00

4    15737888          850      Spain  Female   43        2  125510.82


   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


   Complain  Satisfaction Score Card Type  Point Earned
0         1                   2   DIAMOND           464
1         1                   3   DIAMOND           456
2         1                   3   DIAMOND           377
3         0                   5      GOLD           350
4         0                   5      GOLD           425
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   CustomerId          10000 non-null  int64
 1   CreditScore         10000 non-null  int64
 2   Geography           10000 non-null  object
 3   Gender              10000 non-null  object
 4   Age                 10000 non-null  int64
 5   Tenure              10000 non-null  int64
 6   Balance             10000 non-null  float64
 7   NumOfProducts       10000 non-null  int64
 8   HasCrCard           10000 non-null  int64
 9   IsActiveMember      10000 non-null  int64
 10  EstimatedSalary     10000 non-null  float64
 11  Exited              10000 non-null  int64
 12  Complain            10000 non-null  int64
 13  Satisfaction Score  10000 non-null  int64
 14  Card Type           10000 non-null  object
```

```
 15  Point Earned        10000 non-null  int64
dtypes: float64(2), int64(11), object(3)
memory usage: 1.2+ MB
```

```
pd.unique(df['Geography'])
```

```
array(['France', 'Spain', 'Germany'], dtype=object)
```

Before we move into the actual analysis, I would like to consider the quality of the data. To do that I will use the ROCCC acronym introduced in the google course:

- **Reliable**: Unfortunately, little information is provided on the dataset in its Kaggle page. Therefore, we could say its reliability is unkown.
- **Original**: The data does seem to have been collected by the bank itself though. So, I would assume it indeed is original.
- **Comprehensive**: Based on the data having *10000* records, it seems to be very comprehensive.
- **Current**: The data has last been updated on 2022, therefore it isn't current.
- **Cited**: As has already been said, the origin of the data is not declared in the Kaggle page, so it is not cited.

That would emply that the data's quality isn't great.

Process and Analyse:

since the data seems to already be clean, it would be unnecessary to have an entire step for processing the data. Therefore, I decided to join processing of the data with the analysis.

Our key metric will be churn rate in %. This will be messured over different geographic locations and for customers who had different experiences with the bank.

```
TotalChurn = round(np.sum(df['Exited'])/df.shape[0],3)*100
print("The total churn rate is: ", TotalChurn, "%")
```

```
The total churn rate is:  20.4 %
```

Geographic analysis

```
GeographicChurn = df[['Geography',
'Exited']].groupby('Geography').sum()
GeographicChurn['ChurnRate']=
round(GeographicChurn['Exited']/(df['Geography'].value_counts()),3)*10
0
GeographicChurn
```

	Exited	ChurnRate
Geography		
France	811	16.2
Germany	814	32.4
Spain	413	16.7

```
GeographicComplain = df[['Geography',
'Complain']].groupby('Geography').sum()
GeographicComplain['ComplainFreq'] =
round(GeographicComplain['Complain']/df['Geography'].value_counts(),3)
*100
GeographicComplain
```

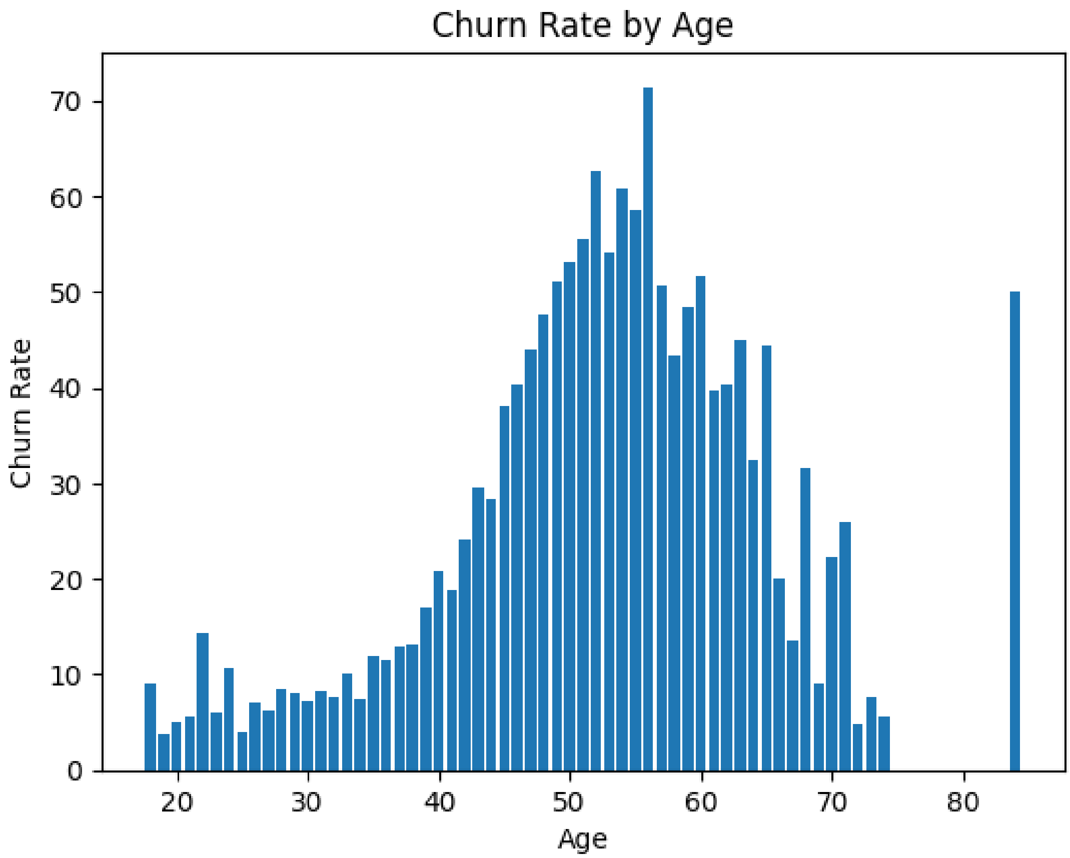
	Complain	ComplainFreq
Geography		
France	812	16.2
Germany	819	32.6
Spain	413	16.7

Firstly, it is very clear that clients from germany leave the bank far more frequently. The fact that the complain frequency is almost identical to the Churn rate shows a clear relationship. Indicating that clients who complain may not be having their issues resolved.

Age & Tenure analysis

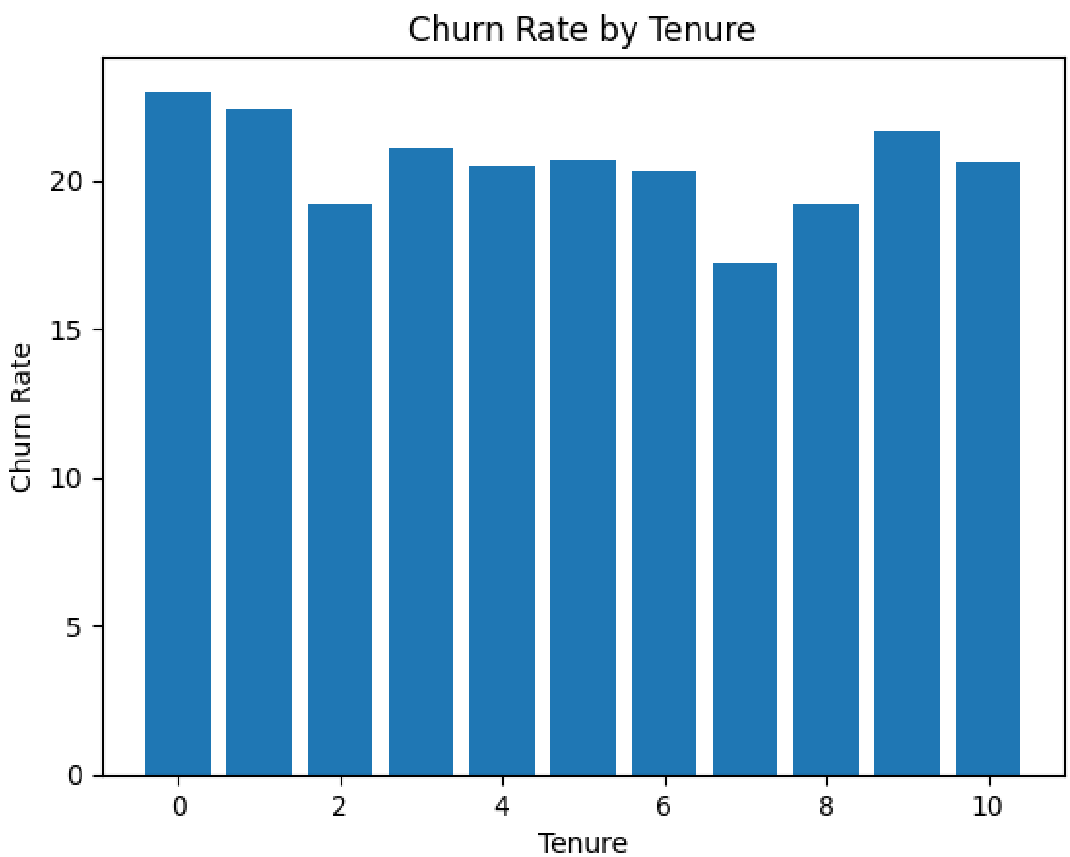
```
AgeChurn = (df[['Age', 'Exited']].groupby('Age').sum())
AgeChurn = AgeChurn[AgeChurn['Exited']!=0]
AgeChurn['ChurnRate']=
round(AgeChurn['Exited']/(df['Age'].value_counts()),3)*100
AgeChurn = AgeChurn.reset_index()

plt.bar(AgeChurn['Age'],AgeChurn['ChurnRate'])
plt.xlabel('Age')
plt.ylabel('Churn Rate')
plt.title('Churn Rate by Age')
plt.show()
```




```
TenureChurn = (df[['Tenure', 'Exited']].groupby('Tenure').sum())
TenureChurn = TenureChurn[TenureChurn['Exited']!=0]
TenureChurn['ChurnRate']=
round(TenureChurn['Exited']/(df['Tenure'].value_counts()),3)*100
TenureChurn = TenureChurn.reset_index()

plt.bar(TenureChurn['Tenure'],TenureChurn['ChurnRate'])
plt.xlabel('Tenure')
plt.ylabel('Churn Rate')
plt.title('Churn Rate by Tenure')
plt.show()
```



We saw that clients between the ages of 40 to 65 left the bank far more often than other age ranges. We also saw that Churn rate isn't affected by tenure.

Behavioral analysis of clients

First we check to see the relationship between a member being active and the churn rate.

```
ActiveChurn = (df[['IsActiveMember',
'Exited']].groupby('IsActiveMember').sum())
ActiveChurn = ActiveChurn[ActiveChurn['Exited']!=0]
ActiveChurn['ChurnRate']=
round(ActiveChurn['Exited']/(df['IsActiveMember'].value_counts()),3)*1
00
ActiveChurn
```

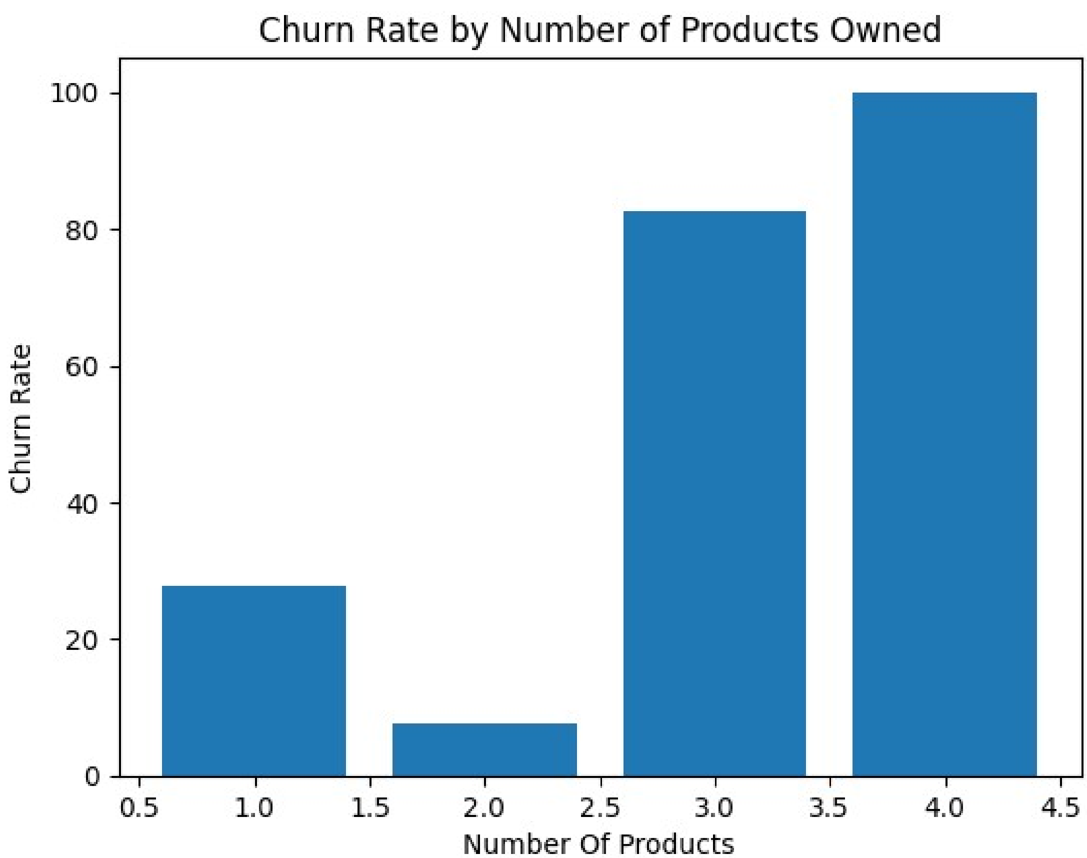
	Exited	ChurnRate
IsActiveMember		
0	1303	26.9
1	735	14.3

It is clear that inactive members are twice as likely to leave the bank.

Now, we will check to see if there is any relationship between the number of products owned by a member and the frequency of members leaving the bank.

```
ProdChurn = (df[['NumOfProducts',
'Exited']].groupby('NumOfProducts').sum())
ProdChurn['ChurnRate']=
round(ProdChurn['Exited']/(df['NumOfProducts'].value_counts()),3)*100
ProdChurn = ProdChurn.reset_index()

plt.bar(ProdChurn['NumOfProducts'],ProdChurn['ChurnRate'])
plt.xlabel('Number Of Products')
plt.ylabel('Churn Rate')
plt.title('Churn Rate by Number of Products Owned')
plt.show()
```



It seems that about a third of clients with one product leave the bank. On the other hand, less than a tenth of clients with two products leave the bank. That may be because clients with a single product are less dependent on the services of the bank than those with two products. The Churn rate of customers with 3 and 4 products, however, is surprisingly high. That might be due to these products being underdeveloped or perhaps the clients don't want to be buying 3 or 4 different products and would rather buy a single one. Eitherway, more information is required to find this out.

Advanced Analysis of the set

First, I will one-hot encode our catigorical feilds and drop the necessary feilds to be able to apply machine learning to the dataset:

```
ohe = OneHotEncoder()
fa = ohe.fit_transform(df[['Gender','Geography','Card
Type']]).toarray()
fl = np.concatenate(ohe.categories_)
features= pd.DataFrame(fa, columns=fl)

dataset = pd.concat([df,features], axis =1)
dataset = dataset.drop(['CustomerId','Gender','Geography','Card Type',
'Exited'], axis=1)
Exited = df['Exited']
dataset_= df.drop(['CustomerId','Gender','Geography','Card Type'],
axis=1)
```

I have made two datasets, with and without the catigorical data. Now, I will utilise a Kmean clustering to try to group similar clients. This could help the bank maintain different types of clients.

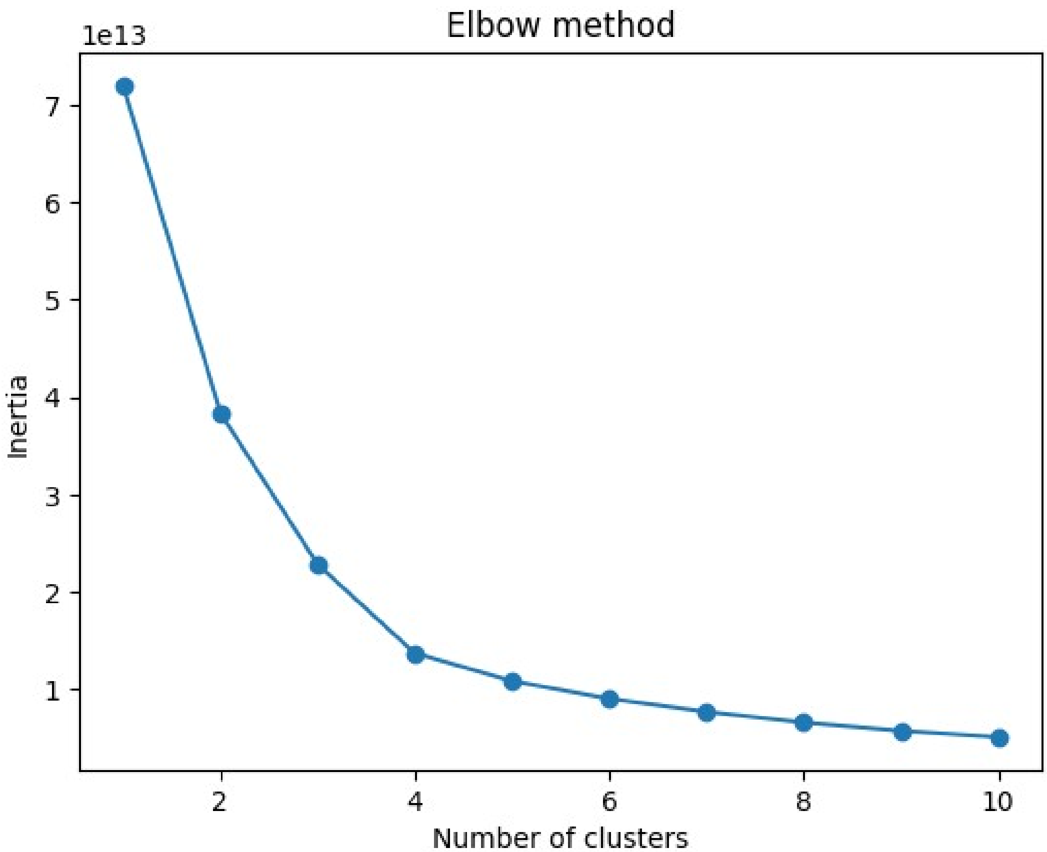
```
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(dataset_)
    inertias.append(kmeans.inertia_)
```

```
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
```

```
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



It seems like 4 is a fair number of clusters. We will split our data based on it, and save it for further analysis to find similarities between them.

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(dataset_)
df['Class']=kmeans.labels_
dataset_['Class']=kmeans.labels_

C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)

dataset_.groupby('Class').mean()

       CreditScore        Age    Tenure        Balance  NumOfProducts

Class

0       650.828489  38.988183  5.002555  121953.442609       1.402108
\
1       651.626549  39.405414  4.961840  121880.264804       1.367906
```

```
2        648.706504  38.423057  5.069804    2619.445812        1.762031

3        650.080021  38.530858  5.054916    2293.096313        1.770921


       HasCrCard  IsActiveMember  EstimatedSalary    Exited  Complain

Class

0       0.694666        0.512935    149744.909055  0.243373  0.244970
\
1       0.699935        0.513699     50472.479997  0.237443  0.237769

2       0.709677        0.510841    149222.657758  0.149656  0.149127

3       0.728033        0.525105     49750.219179  0.138598  0.139121


       Satisfaction Score  Point Earned
Class
0                2.998084    611.972852
1                3.037182    605.710372
2                2.993654    597.984135
3                3.021967    607.305439
```

Seems like the algorithm only grouped them by balance, number of productsm and estimated salary, since the rest is almost identical in values between all the classes. Lets look a bit closer at that.

```
s = (round(dataset_[['Class','Exited']].groupby('Class').sum()
['Exited']/dataset_['Class'].value_counts(),3)*100)
s.name ="ChurnRate"
pd.concat([dataset_[['Class',
'Balance','EstimatedSalary']].groupby('Class').mean(),
           dataset_[['Class',
'NumOfProducts']].groupby('Class').median(),
          s], axis=1)
```

```
             Balance  EstimatedSalary  NumOfProducts  ChurnRate
Class
0      121953.442609    149744.909055            1.0       24.3
1      121880.264804     50472.479997            1.0       23.7
2        2619.445812    149222.657758            2.0       15.0
3        2293.096313     49750.219179            2.0       13.9
```

It seems like people with a higher balance seem to face more problems in their accounts. That would explain the higher complain and churn rate.

Share:

This step will be done using tablaue.

Act:

In the act phase, I create an assisting tool which I think would be useful to stakeholders to figure out which clients are likely to leave the company and take action before losing our clients. The tool will be a neural network trained on historical data and built to predict the likelyhood of a client leaving the bank.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   CreditScore         10000 non-null  int64  
 1   Age                 10000 non-null  int64  
 2   Tenure              10000 non-null  int64  
 3   Balance             10000 non-null  float64
 4   NumOfProducts       10000 non-null  int64  
 5   HasCrCard           10000 non-null  int64  
 6   IsActiveMember      10000 non-null  int64  
 7   EstimatedSalary     10000 non-null  float64
 8   Complain            10000 non-null  int64  
 9   Satisfaction Score  10000 non-null  int64  
 10  Point Earned        10000 non-null  int64  
 11  Female              10000 non-null  float64
 12  Male                10000 non-null  float64
 13  France              10000 non-null  float64
 14  Germany             10000 non-null  float64
 15  Spain               10000 non-null  float64
 16  DIAMOND             10000 non-null  float64
 17  GOLD                10000 non-null  float64
 18  PLATINUM            10000 non-null  float64
 19  SILVER              10000 non-null  float64
dtypes: float64(11), int64(9)
memory usage: 1.5 MB
```

```
# Normalisation of the data
dataset[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
         'EstimatedSalary', 'Satisfaction Score']] =
normalize(dataset[['CreditScore', 'Age', 'Tenure', 'Balance',

'NumOfProducts','EstimatedSalary', 'Satisfaction Score']])
# Splitting data to training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(dataset, Exited,
test_size=0.33)

model = Sequential(
    [
        tf.keras.Input(shape=(20,)),
        Dense(50, activation='relu'),
        Dense(25, activation='relu'),
        Dense(15, activation='relu'),
        Dense(10, activation='relu'),
        Dense(1, activation='sigmoid')

    ]
)

model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

history = model.fit(
    X_train,y_train,
    epochs=50
)
```

```
Epoch 1/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 1s 668us/step - loss: 2.3240
Epoch 2/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - loss: 0.5325
Epoch 3/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - loss: 0.5151
Epoch 4/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 604us/step - loss: 0.5140
Epoch 5/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 603us/step - loss: 0.5188
Epoch 6/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - loss: 0.5063
Epoch 7/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - loss: 0.4951
Epoch 8/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.4907
Epoch 9/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 598us/step - loss: 0.4901
Epoch 10/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - loss: 0.4725
Epoch 11/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.4250
Epoch 12/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 0.3503
Epoch 13/50
```

```
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.2783
Epoch 14/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.1605
Epoch 15/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - loss: 0.0308
Epoch 16/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 0.0183
Epoch 17/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.0247
Epoch 18/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 0.0122
Epoch 19/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - loss: 0.0113
Epoch 20/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.0128
Epoch 21/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - loss: 0.0260
Epoch 22/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.0061
Epoch 23/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 618us/step - loss: 0.0103
Epoch 24/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.0178
Epoch 25/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 707us/step - loss: 0.0376
Epoch 26/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0084
Epoch 27/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 626us/step - loss: 0.0126
Epoch 28/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 646us/step - loss: 0.0143
Epoch 29/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 612us/step - loss: 0.0100
Epoch 30/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - loss: 0.0168
Epoch 31/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - loss: 0.0119
Epoch 32/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 590us/step - loss: 0.0209
Epoch 33/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.0195
Epoch 34/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 0.0149
Epoch 35/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - loss: 0.0142
Epoch 36/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.0149
Epoch 37/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - loss: 0.0153
```

```
Epoch 38/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.0176
Epoch 39/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 596us/step - loss: 0.0087
Epoch 40/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - loss: 0.0692
Epoch 41/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - loss: 0.0081
Epoch 42/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - loss: 0.0108
Epoch 43/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - loss: 0.0127
Epoch 44/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 612us/step - loss: 0.0131
Epoch 45/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - loss: 0.0137
Epoch 46/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.0165
Epoch 47/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - loss: 0.0091
Epoch 48/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 708us/step - loss: 0.0169
Epoch 49/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 701us/step - loss: 0.0187
Epoch 50/50
210/210 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.0061

predictions= model.predict(X_test)
predictions= pd.Series(np.round(predictions)[:,0])
y_test.reset_index(drop=True, inplace=True)
print("Our Model is",
round((predictions==y_test).sum()/y_test.shape[0],3)*100, "%
accurate.")

104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 996us/step
Our Model is 99.9 % accurate.
```

Now, the network has been trained and is ready to predict our data for use in customer attrition prevention.