Mini Project ANN(Shoaib Ghulam Sadar ALi Khan Osama Saleem Huzaifa Abdali)

In [2]:

```
# Part 1 - Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [3]:

```
# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
```

In [4]:

```
# Encoding categorical data
# Encode before splitting because matrix X and independent variable Y mu
# Found two categorical data (country, gender)
# create dummy variables, avoid dummy variable trap
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
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])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
```

```
D:\Anaconda\lib\site-packages\sklearn\preprocessing\_encoders.py:371: Futur
eWarning: The handling of integer data will change in version 0.22. Current
ly, the categories are determined based on the range [0, max(values)], whil
e in the future they will be determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the ca
tegories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
D:\Anaconda\lib\site-packages\sklearn\preprocessing\_encoders.py:392: Depre
cationWarning: The 'categorical_features' keyword is deprecated in version
0.20 and will be removed in 0.22. You can use the ColumnTransformer instea
d.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

In [5]:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
```

In [8]:

```

# Feature Scaling
# lots of high computation to ease calculation, we don't want one indepe
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [9]:

```
# Part 2 - Making the ANN

# Importing the Keras libraries and package
# Sequential module - initialize neural network
# Dense - layers of ANN
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

In [10]:

```
# Initialising the ANN
classifier = Sequential()
```

In [11]:

```
# Adding the input layer and the first hidden layer with dropout
# Take average of input + output for units/output_dim param in Dense
# input_dim is necessary for the first layer as it was just initialized
classifier.add(Dense(6, input_dim = 11, kernel_initializer = 'glorot_uni
classifier.add(Dropout(p = 0.1))
```

```
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:5: UserWarning: Update
your `Dropout` call to the Keras 2 API: `Dropout(rate=0.1)`
  """
```

In [12]:

```
# Adding the second hidden layer with dropout
# doesn't need the input_dim params
# kernel_initializer updates weights
# activation function - rectifier
classifier.add(Dense(6, kernel_initializer = 'glorot_uniform', activatio
classifier.add(Dropout(p = 0.1))
```

```
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:6: UserWarning: Update
your `Dropout` call to the Keras 2 API: `Dropout(rate=0.1)`
```

In [13]:

```
# Adding the output layer
# dependent variable with more than two categories (3), output_dim needs
classifier.add(Dense(1, kernel_initializer = 'glorot_uniform', activatio
```

In [14]:

```
# Compiling the ANN - applying Stochastic Gradient Descent to whole ANN
# Several different SGD algorithms
# mathematical details based on the loss function
# binary_crossentropy, categorical_cross_entropy
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', met
```

```
WARNING:tensorflow:From D:\Anaconda\lib\site-packages\tensorflow\python\ops
\nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tensorflow.pyt
hon.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

In [15]:

```
# Fitting the ANN to the Training Set
# X_train, y_train, Batch size, Epochs (whole training set)
classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
```

```
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:3: UserWarning: The `nb
_epoch` argument in `fit` has been renamed `epochs`.
  This is separate from the ipykernel package so we can avoid doing imports
until

WARNING:tensorflow:From D:\Anaconda\lib\site-packages\keras\backend\tensorf
low_backend.py:422: The name tf.global_variables is deprecated. Please use
tf.compat.v1.global_variables instead.

Epoch 1/100
8000/8000 [==============================] - 5s 656us/step - loss: 0.5544 -
accuracy: 0.7444
Epoch 2/100
8000/8000 [==============================] - 3s 400us/step - loss: 0.4705 -
accuracy: 0.7974
Epoch 3/100
8000/8000 [==============================] - 3s 428us/step - loss: 0.4469 -
accuracy: 0.8008
Epoch 4/100
8000/8000 [==============================] - 3s 391us/step - loss: 0.4210 -
accuracy: 0.8236
Epoch 5/100
8000/8000 [==============================] - 3s 368us/step - loss: 0.4051 -
accuracy: 0.8367
Epoch 6/100
8000/8000 [==============================] - 3s 411us/step - loss: 0.3889 -
accuracy: 0.8426
Epoch 7/100
8000/8000 [==============================] - 3s 379us/step - loss: 0.3871 -
accuracy: 0.8395
Epoch 8/100
8000/8000 [==============================] - 3s 406us/step - loss: 0.3838 -
accuracy: 0.8403
Epoch 9/100
8000/8000 [==============================] - 3s 429us/step - loss: 0.3802 -
accuracy: 0.8425
Epoch 10/100
8000/8000 [==============================] - 3s 406us/step - loss: 0.3753 -
accuracy: 0.8466
Epoch 11/100
8000/8000 [==============================] - 3s 397us/step - loss: 0.3793 -
accuracy: 0.8426
Epoch 12/100
8000/8000 [==============================] - 4s 461us/step - loss: 0.3741 -
accuracy: 0.8471
Epoch 13/100
8000/8000 [==============================] - 3s 362us/step - loss: 0.3735 -
accuracy: 0.8455
Epoch 14/100
8000/8000 [==============================] - 3s 398us/step - loss: 0.3747 -
accuracy: 0.8444
Epoch 15/100
8000/8000 [==============================] - 4s 453us/step - loss: 0.3728 -
```

```
accuracy: 0.8474
Epoch 16/100
8000/8000 [==============================] - 3s 391us/step - loss: 0.3728 -
accuracy: 0.8469
Epoch 17/100
8000/8000 [==============================] - 3s 417us/step - loss: 0.3630 -
accuracy: 0.8514
Epoch 18/100
8000/8000 [==============================] - 3s 341us/step - loss: 0.3745 -
accuracy: 0.8444
Epoch 19/100
8000/8000 [==============================] - 3s 411us/step - loss: 0.3659 -
accuracy: 0.8499
Epoch 20/100
8000/8000 [==============================] - 4s 491us/step - loss: 0.3668 -
accuracy: 0.8491
Epoch 21/100
8000/8000 [==============================] - 3s 366us/step - loss: 0.3702 -
accuracy: 0.8466
Epoch 22/100
8000/8000 [==============================] - 3s 369us/step - loss: 0.3670 -
accuracy: 0.8489
Epoch 23/100
8000/8000 [==============================] - 3s 408us/step - loss: 0.3620 -
accuracy: 0.8481
Epoch 24/100
8000/8000 [==============================] - 3s 334us/step - loss: 0.3714 -
accuracy: 0.8472
Epoch 25/100
8000/8000 [==============================] - 3s 325us/step - loss: 0.3689 -
accuracy: 0.8476
Epoch 26/100
8000/8000 [==============================] - 3s 342us/step - loss: 0.3664 -
accuracy: 0.8494
Epoch 27/100
8000/8000 [==============================] - 3s 334us/step - loss: 0.3670 -
accuracy: 0.8505
Epoch 28/100
8000/8000 [==============================] - 3s 337us/step - loss: 0.3643 -
accuracy: 0.8494
Epoch 29/100
8000/8000 [==============================] - 3s 323us/step - loss: 0.3634 -
accuracy: 0.8497
Epoch 30/100
8000/8000 [==============================] - 3s 324us/step - loss: 0.3659 -
accuracy: 0.8499
Epoch 31/100
8000/8000 [==============================] - 3s 332us/step - loss: 0.3661 -
accuracy: 0.8484
Epoch 32/100
8000/8000 [==============================] - 3s 389us/step - loss: 0.3631 -
accuracy: 0.8503
Epoch 33/100
8000/8000 [==============================] - 4s 474us/step - loss: 0.3693 -
accuracy: 0.8435
Epoch 34/100
8000/8000 [==============================] - 4s 554us/step - loss: 0.3618 -
```

```
accuracy: 0.8504
Epoch 35/100
8000/8000 [==============================] - 4s 457us/step - loss: 0.3630 -
accuracy: 0.8465
Epoch 36/100
8000/8000 [==============================] - 3s 429us/step - loss: 0.3582 -
accuracy: 0.8494
Epoch 37/100
8000/8000 [==============================] - 3s 403us/step - loss: 0.3621 -
accuracy: 0.8459
Epoch 38/100
8000/8000 [==============================] - 3s 402us/step - loss: 0.3649 -
accuracy: 0.8460
Epoch 39/100
8000/8000 [==============================] - 4s 450us/step - loss: 0.3687 -
accuracy: 0.8456
Epoch 40/100
8000/8000 [==============================] - 4s 454us/step - loss: 0.3644 -
accuracy: 0.8469
Epoch 41/100
8000/8000 [==============================] - 4s 442us/step - loss: 0.3587 -
accuracy: 0.8534
Epoch 42/100
8000/8000 [==============================] - 4s 458us/step - loss: 0.3621 -
accuracy: 0.8516
Epoch 43/100
8000/8000 [==============================] - 3s 423us/step - loss: 0.3582 -
accuracy: 0.8524
Epoch 44/100
8000/8000 [==============================] - 4s 442us/step - loss: 0.3641 -
accuracy: 0.8491
Epoch 45/100
8000/8000 [==============================] - 3s 386us/step - loss: 0.3622 -
accuracy: 0.8482
Epoch 46/100
8000/8000 [==============================] - 4s 464us/step - loss: 0.3642 -
accuracy: 0.8469
Epoch 47/100
8000/8000 [==============================] - 3s 374us/step - loss: 0.3568 -
accuracy: 0.8546
Epoch 48/100
8000/8000 [==============================] - 3s 410us/step - loss: 0.3615 -
accuracy: 0.8482
Epoch 49/100
8000/8000 [==============================] - 3s 407us/step - loss: 0.3626 -
accuracy: 0.8496
Epoch 50/100
8000/8000 [==============================] - 3s 344us/step - loss: 0.3595 -
accuracy: 0.8484
Epoch 51/100
8000/8000 [==============================] - 3s 386us/step - loss: 0.3618 -
accuracy: 0.8489
Epoch 52/100
8000/8000 [==============================] - 3s 390us/step - loss: 0.3607 -
accuracy: 0.8481
Epoch 53/100
8000/8000 [==============================] - 3s 387us/step - loss: 0.3568 -
```

```
accuracy: 0.8518
Epoch 54/100
8000/8000 [==============================] - 4s 462us/step - loss: 0.3605 -
accuracy: 0.84990s - loss: 0.3600 - accu
Epoch 55/100
8000/8000 [==============================] - 4s 465us/step - loss: 0.3640 -
accuracy: 0.8461
Epoch 56/100
8000/8000 [==============================] - 3s 403us/step - loss: 0.3595 -
accuracy: 0.8499
Epoch 57/100
8000/8000 [==============================] - 3s 347us/step - loss: 0.3598 -
accuracy: 0.8501
Epoch 58/100
8000/8000 [==============================] - 4s 459us/step - loss: 0.3591 -
accuracy: 0.8501
Epoch 59/100
8000/8000 [==============================] - 4s 456us/step - loss: 0.3596 -
accuracy: 0.8497
Epoch 60/100
8000/8000 [==============================] - 3s 404us/step - loss: 0.3598 -
accuracy: 0.8506
Epoch 61/100
8000/8000 [==============================] - 4s 485us/step - loss: 0.3580 -
accuracy: 0.8522
Epoch 62/100
8000/8000 [==============================] - 4s 560us/step - loss: 0.3576 -
accuracy: 0.8504
Epoch 63/100
8000/8000 [==============================] - 4s 526us/step - loss: 0.3612 -
accuracy: 0.8482
Epoch 64/100
8000/8000 [==============================] - 4s 443us/step - loss: 0.3600 -
accuracy: 0.8530
Epoch 65/100
8000/8000 [==============================] - 4s 457us/step - loss: 0.3562 -
accuracy: 0.8518
Epoch 66/100
8000/8000 [==============================] - 4s 475us/step - loss: 0.3614 -
accuracy: 0.8490
Epoch 67/100
8000/8000 [==============================] - 4s 484us/step - loss: 0.3598 -
accuracy: 0.8504
Epoch 68/100
8000/8000 [==============================] - 4s 444us/step - loss: 0.3573 -
accuracy: 0.8511
Epoch 69/100
8000/8000 [==============================] - 4s 479us/step - loss: 0.3582 -
accuracy: 0.8524
Epoch 70/100
8000/8000 [==============================] - 4s 450us/step - loss: 0.3623 -
accuracy: 0.8479
Epoch 71/100
8000/8000 [==============================] - 4s 453us/step - loss: 0.3631 -
accuracy: 0.8475
Epoch 72/100
8000/8000 [==============================] - 4s 476us/step - loss: 0.3624 -
```

```
accuracy: 0.8508
Epoch 73/100
8000/8000 [==============================] - 4s 462us/step - loss: 0.3588 -
accuracy: 0.8518
Epoch 74/100
8000/8000 [==============================] - 3s 407us/step - loss: 0.3638 -
accuracy: 0.8487
Epoch 75/100
8000/8000 [==============================] - 3s 425us/step - loss: 0.3602 -
accuracy: 0.8495
Epoch 76/100
8000/8000 [==============================] - 3s 434us/step - loss: 0.3610 -
accuracy: 0.8496
Epoch 77/100
8000/8000 [==============================] - 3s 384us/step - loss: 0.3595 -
accuracy: 0.8491
Epoch 78/100
8000/8000 [==============================] - 3s 424us/step - loss: 0.3611 -
accuracy: 0.8496
Epoch 79/100
8000/8000 [==============================] - 3s 422us/step - loss: 0.3566 -
accuracy: 0.8499
Epoch 80/100
8000/8000 [==============================] - 3s 345us/step - loss: 0.3573 -
accuracy: 0.8540
Epoch 81/100
8000/8000 [==============================] - 3s 431us/step - loss: 0.3594 -
accuracy: 0.8495
Epoch 82/100
8000/8000 [==============================] - 3s 421us/step - loss: 0.3586 -
accuracy: 0.8500
Epoch 83/100
8000/8000 [==============================] - ETA: 0s - loss: 0.3602 - accur
acy: 0.85 - 3s 390us/step - loss: 0.3607 - accuracy: 0.8520
Epoch 84/100
8000/8000 [==============================] - 3s 387us/step - loss: 0.3560 -
accuracy: 0.8506
Epoch 85/100
8000/8000 [==============================] - 3s 428us/step - loss: 0.3571 -
accuracy: 0.8512
Epoch 86/100
8000/8000 [==============================] - 3s 417us/step - loss: 0.3588 -
accuracy: 0.8509
Epoch 87/100
8000/8000 [==============================] - 3s 378us/step - loss: 0.3551 -
accuracy: 0.8540
Epoch 88/100
8000/8000 [==============================] - 3s 403us/step - loss: 0.3538 -
accuracy: 0.8539
Epoch 89/100
8000/8000 [==============================] - 3s 368us/step - loss: 0.3553 -
accuracy: 0.8508
Epoch 90/100
8000/8000 [==============================] - 3s 398us/step - loss: 0.3597 -
accuracy: 0.8514
Epoch 91/100
8000/8000 [==============================] - 3s 423us/step - loss: 0.3565 -
```

```
accuracy: 0.8509
Epoch 92/100
8000/8000 [==============================] - 4s 441us/step - loss: 0.3585 -
accuracy: 0.8533
Epoch 93/100
8000/8000 [==============================] - 3s 409us/step - loss: 0.3583 -
accuracy: 0.8539
Epoch 94/100
8000/8000 [==============================] - 3s 426us/step - loss: 0.3565 -
accuracy: 0.8510
Epoch 95/100
8000/8000 [==============================] - 3s 398us/step - loss: 0.3566 -
accuracy: 0.8506
Epoch 96/100
8000/8000 [==============================] - 3s 423us/step - loss: 0.3605 -
accuracy: 0.8499
Epoch 97/100
8000/8000 [==============================] - 3s 391us/step - loss: 0.3556 -
accuracy: 0.8536
Epoch 98/100
8000/8000 [==============================] - 3s 433us/step - loss: 0.3578 -
accuracy: 0.8528
Epoch 99/100
8000/8000 [==============================] - 3s 417us/step - loss: 0.3555 -
accuracy: 0.8510
Epoch 100/100
8000/8000 [==============================] - 3s 358us/step - loss: 0.3605 -
accuracy: 0.8497
```

Out[15]:
```
<keras.callbacks.callbacks.History at 0x19c3b4ef7f0>
```

In [17]:
```python
# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
# Training set, see if the new data probability is right
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

In [18]:
```python
# Predicting a single new observation
new_prediction = classifier.predict(sc.transform(np.array([[0, 0, 600, 1
new_prediction = (new_prediction > 0.5)
```

```
D:\Anaconda\lib\site-packages\sklearn\utils\validation.py:595: DataConversi
onWarning: Data with input dtype int32 was converted to float64 by Standard
Scaler.
  warnings.warn(msg, DataConversionWarning)
```

In [19]:
```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Apply the ANN Model

In [20]:

```
# Part 4 - Evaluating, Imrpvoing and Tuning the ANN

# Evaluating the ANN
# Keras wrapper and Sci-kit Learn for k-Fold Cross Validation
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers import Dense
def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(6, input_dim = 11, kernel_initializer = 'glorot
    classifier.add(Dense(6, kernel_initializer = 'glorot_uniform', activ
    classifier.add(Dense(1, kernel_initializer = 'glorot_uniform', activ
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    return classifier
```

Apply K-Fold Cross Validation

In [21]:

```
# k-Fold cross validator to check if the real relevant accuracy or the s
#ß and where we are in bias-variance tradeoffs
classifier = KerasClassifier(build_fn = build_classifier, batch_size = 1
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_
mean = accuracies.mean()
variable = accuracies.std()
```

In [22]:

```
# overfitting is when it's trained to much on the training set, less per
# and training set, high variance in, Dropout Regulariatio to reduce ove

# Tuning the ANN, parameters learned during training (weights), stay fix
# parameter tuning best value of these hyperparameters, GridSearchCV wit
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
def build_classifier(optimzer):
    classifier = Sequential()
    classifier.add(Dense(6, input_dim = 11, kernel_initializer = 'glorot
    classifier.add(Dense(6, kernel_initializer = 'glorot_uniform', activ
    classifier.add(Dense(1, kernel_initializer = 'glorot_uniform', activ
    classifier.compile(optimizer = optimzer, loss = 'binary_crossentropy
    return classifier

```

In [23]:

```
classifier = KerasClassifier(build_fn = build_classifier)
parameters = {'batch_size': [25, 32],
              'nb_epoch': [100, 500],
              'optimzer': ['adam', 'rmsprop']}
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

```
Epoch 1/1
7200/7200 [==============================] - 2s 255us/step - loss: 0.6099
- accuracy: 0.7079
Epoch 1/1
7200/7200 [==============================] - 2s 334us/step - loss: 0.5758
- accuracy: 0.7539
Epoch 1/1
7200/7200 [==============================] - 2s 323us/step - loss: 0.5770
- accuracy: 0.7618
Epoch 1/1
7200/7200 [==============================] - 2s 300us/step - loss: 0.5400
- accuracy: 0.7839
Epoch 1/1
7200/7200 [==============================] - 2s 313us/step - loss: 0.5648
- accuracy: 0.7835
Epoch 1/1
7200/7200 [==============================] - 2s 294us/step - loss: 0.5426
- accuracy: 0.7596
Epoch 1/1
```

Describe The Dataset

In [28]:

```python
import pandas

data = pd.read_csv('Churn_Modelling.csv')
peek = data.head(20)
print(peek)
```

```
    RowNumber  CustomerId    Surname  CreditScore Geography  Gender  Age  \
0           1    15634602   Hargrave          619    France  Female   42   
1           2    15647311       Hill          608     Spain  Female   41   
2           3    15619304       Onio          502    France  Female   42   
3           4    15701354       Boni          699    France  Female   39   
4           5    15737888   Mitchell          850     Spain  Female   43   
5           6    15574012        Chu          645     Spain    Male   44   
6           7    15592531   Bartlett          822    France    Male   50   
7           8    15656148     Obinna          376   Germany  Female   29   
8           9    15792365         He          501    France    Male   44   
9          10    15592389         H?          684    France    Male   27   
10         11    15767821     Bearce          528    France    Male   31   
11         12    15737173    Andrews          497     Spain    Male   24   
12         13    15632264        Kay          476    France  Female   34   
13         14    15691483       Chin          549    France  Female   25   
14         15    15600882      Scott          635     Spain  Female   35   
15         16    15643966    Goforth          616   Germany    Male   45   
16         17    15737452      Romeo          653   Germany    Male   58   
17         18    15788218  Henderson          549     Spain  Female   24   
18         19    15661507    Muldrow          587     Spain    Male   45   
19         20    15568982        Hao          726    France  Female   24   

    Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0        2       0.00              1          1               1   
1        1   83807.86              1          0               1   
2        8  159660.80              3          1               0   
3        1       0.00              2          0               0   
4        2  125510.82              1          1               1   
5        8  113755.78              2          1               0   
6        7       0.00              2          1               1   
7        4  115046.74              4          1               0   
8        4  142051.07              2          0               1   
9        2  134603.88              1          1               1   
10       6  102016.72              2          0               0   
11       3       0.00              2          1               0   
12      10       0.00              2          1               0   
13       5       0.00              2          0               0   
14       7       0.00              2          1               1   
15       3  143129.41              2          0               1   
16       1  132602.88              1          1               0   
17       9       0.00              2          1               1   
18       6       0.00              1          0               0   
19       6       0.00              2          1               1   

    EstimatedSalary  Exited  
0         101348.88       1  
1         112542.58       0  
2         113931.57       1  
3          93826.63       0  
```

```
4         79084.10       0
5        149756.71       1
6         10062.80       0
7        119346.88       1
8         74940.50       0
9         71725.73       0
10        80181.12       0
11        76390.01       0
12        26260.98       0
13       190857.79       0
14        65951.65       0
15        64327.26       0
16         5097.67       1
17        14406.41       0
18       158684.81       0
19        54724.03       0
```

Dimensions of Your Data

In [30]:

```
shape = data.shape
print(shape)
```

```
(10000, 14)
```

Data Type For Each Attribute

In [31]:

```
types = data.dtypes
print(types)
```

```
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
```

Descriptive Statistics

In [32]:

```
pandas.set_option('display.width', 100)
pandas.set_option('precision', 3)
description = data.describe()
print(description)
```

```
        RowNumber  CustomerId  CreditScore        Age     Tenure     Balance  
NumOfProducts  \
count  10000.000   1.000e+04    10000.000  10000.000  10000.000   10000.000  
10000.000
mean    5000.500   1.569e+07      650.529     38.922      5.013   76485.889  
1.530
std     2886.896   7.194e+04       96.653     10.488      2.892   62397.405  
0.582
min        1.000   1.557e+07      350.000     18.000      0.000       0.000  
1.000
25%     2500.750   1.563e+07      584.000     32.000      3.000       0.000  
1.000
50%     5000.500   1.569e+07      652.000     37.000      5.000   97198.540  
1.000
75%     7500.250   1.575e+07      718.000     44.000      7.000  127644.240  
2.000
max    10000.000   1.582e+07      850.000     92.000     10.000  250898.090  
4.000

        HasCrCard  IsActiveMember  EstimatedSalary     Exited
count  10000.000       10000.000        10000.000  10000.000
mean       0.706           0.515       100090.240      0.204
std        0.456           0.500        57510.493      0.403
min        0.000           0.000           11.580      0.000
25%        0.000           0.000        51002.110      0.000
50%        1.000           1.000       100193.915      0.000
75%        1.000           1.000       149388.247      0.000
max        1.000           1.000       199992.480      1.000
```

Correlation Between Attributes

In [33]:

```
pandas.set_option('display.width', 100)
pandas.set_option('precision', 3)
correlations = data.corr(method='pearson')
print(correlations)
```

```
                RowNumber  CustomerId  CreditScore        Age      Tenure
Balance  NumOfProducts  \
RowNumber         1.000e+00       0.004    5.840e-03  7.826e-04 -6.495e-03
-0.009          0.007
CustomerId        4.202e-03       1.000    5.308e-03  9.497e-03 -1.488e-02
-0.012          0.017
CreditScore       5.840e-03       0.005    1.000e+00 -3.965e-03  8.419e-04
0.006          0.012
Age               7.826e-04       0.009   -3.965e-03  1.000e+00 -9.997e-03
0.028         -0.031
Tenure           -6.495e-03      -0.015    8.419e-04 -9.997e-03  1.000e+00
-0.012          0.013
Balance          -9.067e-03      -0.012    6.268e-03  2.831e-02 -1.225e-02
1.000         -0.304
NumOfProducts     7.246e-03       0.017    1.224e-02 -3.068e-02  1.344e-02
-0.304          1.000
HasCrCard         5.987e-04      -0.014   -5.458e-03 -1.172e-02  2.258e-02
-0.015          0.003
IsActiveMember    1.204e-02       0.002    2.565e-02  8.547e-02 -2.836e-02
-0.010          0.010
EstimatedSalary  -5.988e-03       0.015   -1.384e-03 -7.201e-03  7.784e-03
0.013          0.014
Exited           -1.657e-02      -0.006   -2.709e-02  2.853e-01 -1.400e-02
0.119         -0.048

                 HasCrCard  IsActiveMember  EstimatedSalary  Exited
RowNumber        5.987e-04           0.012           -0.006  -0.017
CustomerId      -1.403e-02           0.002            0.015  -0.006
CreditScore     -5.458e-03           0.026           -0.001  -0.027
Age             -1.172e-02           0.085           -0.007   0.285
Tenure           2.258e-02          -0.028            0.008  -0.014
Balance         -1.486e-02          -0.010            0.013   0.119
NumOfProducts    3.183e-03           0.010            0.014  -0.048
HasCrCard        1.000e+00          -0.012           -0.010  -0.007
IsActiveMember  -1.187e-02           1.000           -0.011  -0.156
EstimatedSalary -9.933e-03          -0.011            1.000   0.012
Exited          -7.138e-03          -0.156            0.012   1.000
```