Artificial Neural Network

Importing the libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
tf.__version__
```

```
'2.4.1'
```

Importing the dataset

```
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

```
print(y)
```

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

Descriptive Statistics and Data visualization

Measures of Central Tendency

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
dataset.mean()
```

```
RowNumber          5.000500e+03
CustomerId         1.569094e+07
CreditScore        6.505288e+02
Age                3.892180e+01
Tenure             5.012800e+00
Balance            7.648589e+04
NumOfProducts      1.530200e+00
HasCrCard          7.055000e-01
IsActiveMember     5.151000e-01
EstimatedSalary    1.000902e+05
Exited             2.037000e-01
dtype: float64
```

```
dataset.median()
```

```
RowNumber          5.000500e+03
CustomerId         1.569074e+07
CreditScore        6.520000e+02
Age                3.700000e+01
Tenure             5.000000e+00
Balance            9.719854e+04
NumOfProducts      1.000000e+00
HasCrCard          1.000000e+00
IsActiveMember     1.000000e+00
EstimatedSalary    1.001939e+05
Exited             0.000000e+00
dtype: float64
```

Measures of Dispersion

```
dataset.std()
```

```
RowNumber           2886.895680
CustomerId         71936.186123
CreditScore           96.653299
Age                   10.487806
Tenure                 2.892174
Balance            62397.405202
NumOfProducts          0.581654
HasCrCard              0.455840
IsActiveMember         0.499797
EstimatedSalary    57510.492818
Exited                 0.402769
dtype: float64
```

```
dataset.var()
```

```
RowNumber          8.334167e+06
CustomerId         5.174815e+09
CreditScore        9.341860e+03
Age                1.099941e+02
Tenure             8.364673e+00
Balance            3.893436e+09
```

```
NumOfProducts      3.383218e-01
HasCrCard          2.077905e-01
IsActiveMember     2.497970e-01
EstimatedSalary    3.307457e+09
Exited             1.622225e-01
dtype: float64
```

```
dataset.skew()
```

```
RowNumber          0.000000
CustomerId         0.001149
CreditScore       -0.071607
Age                1.011320
Tenure             0.010991
Balance           -0.141109
NumOfProducts      0.745568
HasCrCard         -0.901812
IsActiveMember    -0.060437
EstimatedSalary    0.002085
Exited             1.471611
dtype: float64
```

Data visualization

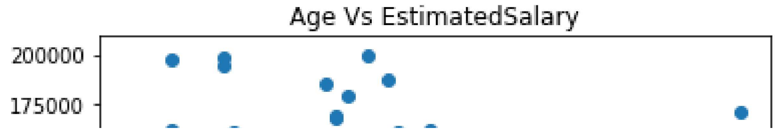
1. Scatter Plot

```
dataset.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

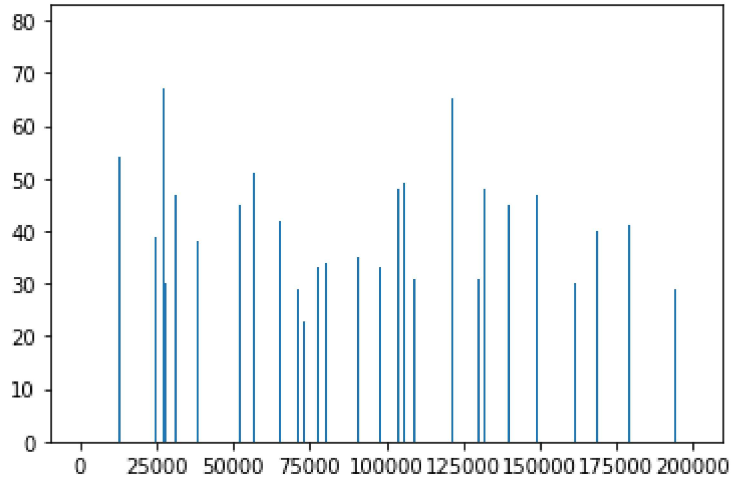
```
dfv=dataset.sample(100)
plt.xlabel('Age')
plt.ylabel('EstimatedSalary')
plt.title('Age Vs EstimatedSalary')
plt.scatter(dfv['Age'],dfv['EstimatedSalary'])
```

```
<matplotlib.collections.PathCollection at 0x7f58087f4e50>
```



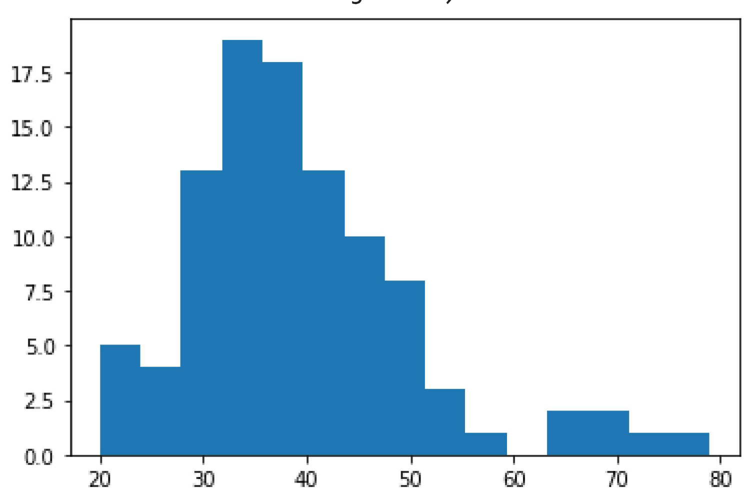

```
plt.bar(dfv['EstimatedSalary'], dfv['Age'], width=200)
```

```
<BarContainer object of 100 artists>
```



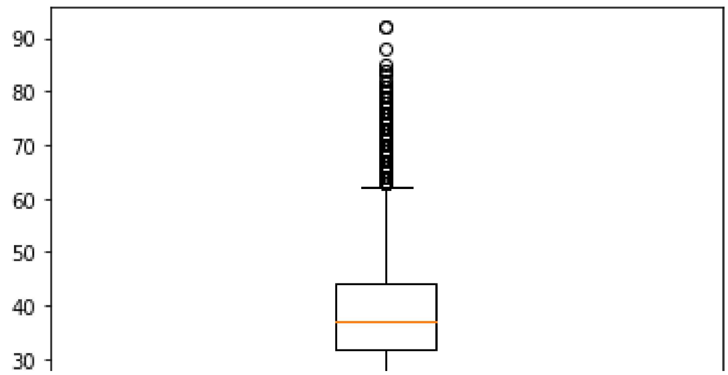

```
plt.hist(dfv['Age'], bins=15)
```

```
(array([ 5.,  4., 13., 19., 18., 13., 10.,  8.,  3.,  1.,  0.,  2.,  2.,
         1.,  1.]),
 array([20.        , 23.93333333, 27.86666667, 31.8       , 35.73333333,
        39.66666667, 43.6       , 47.53333333, 51.46666667, 55.4       ,
        59.33333333, 63.26666667, 67.2       , 71.13333333, 75.06666667,
        79.        ]),
 <a list of 15 Patch objects>)
```



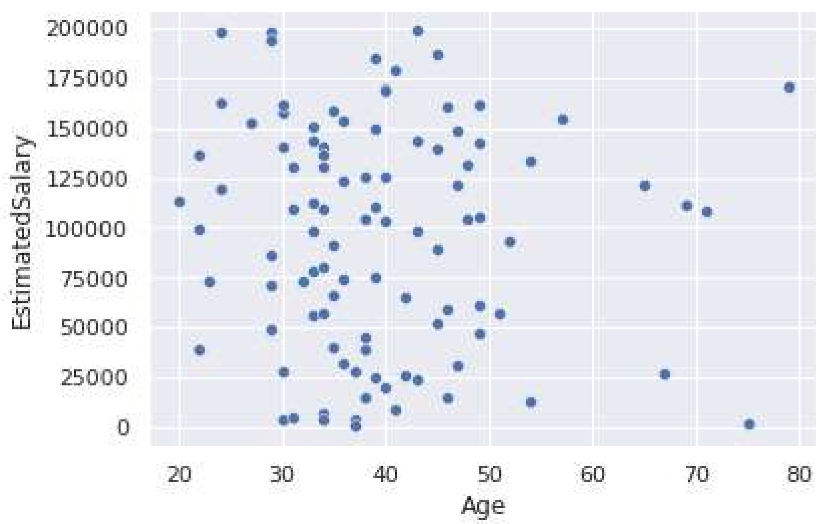

```
plt.boxplot(dataset['Age'])
```

```
{'boxes': [<matplotlib.lines.Line2D at 0x7f5808077890>],
 'caps': [<matplotlib.lines.Line2D at 0x7f580807c850>,
  <matplotlib.lines.Line2D at 0x7f580807cd90>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f5808082890>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f5808082350>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f58080aabd0>,
  <matplotlib.lines.Line2D at 0x7f580807c310>]}
```




```
import seaborn as sns
sns.set()
sns.scatterplot(dfv['Age'],dfv['EstimatedSalary'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass t
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f5801e20410>
```




```
sns.heatmap(dfv.corr(), annot=True, fmt='.2f')
```

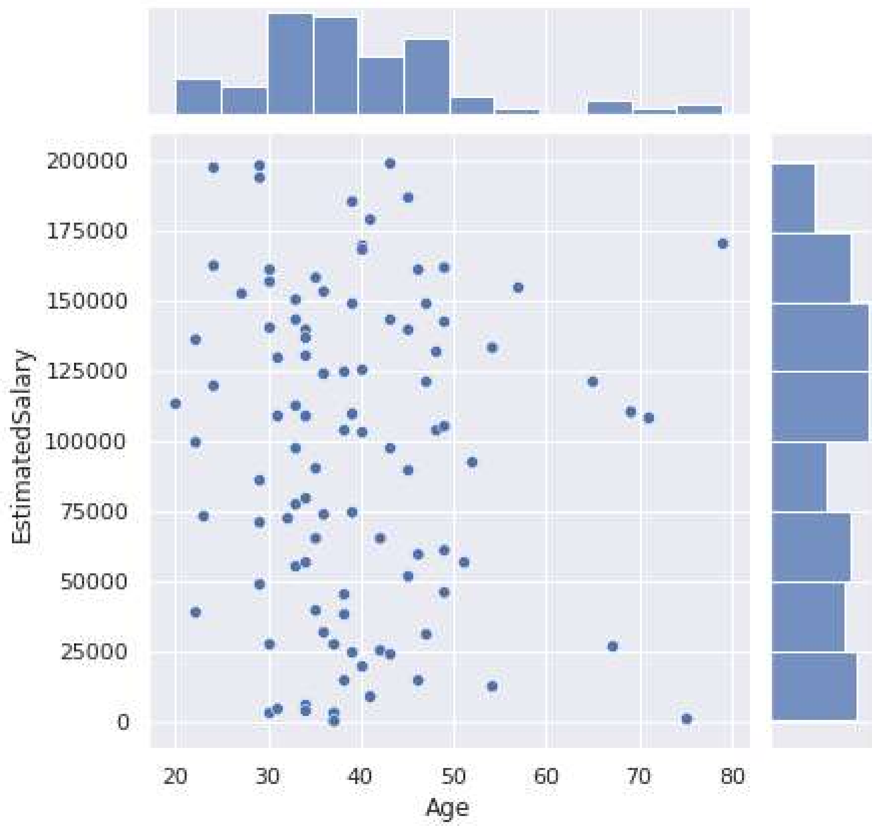
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f580156db90>
```


```
dfp=dfv[['Age','Tenure','EstimatedSalary','CreditScore']]
sns.pairplot(dfp)
```

```
<seaborn.axisgrid.PairGrid at 0x7f57f8bb6950>
```

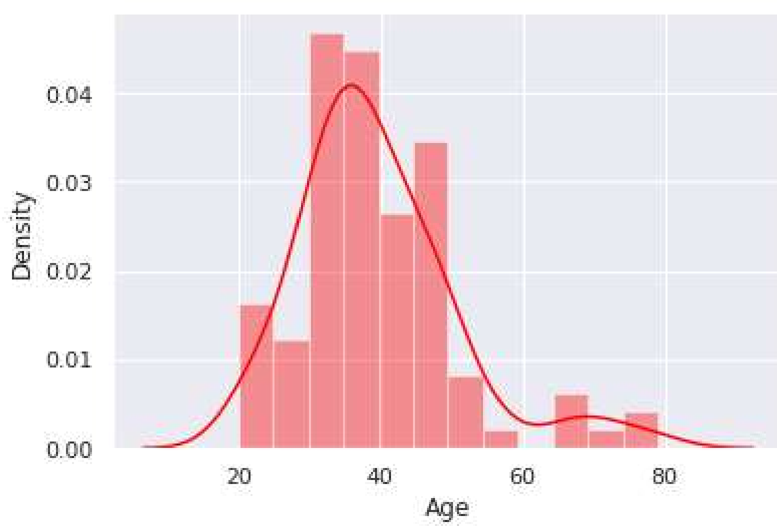
```
sns.jointplot(x='Age', y='EstimatedSalary', data=dfv)
```

```
<seaborn.axisgrid.JointGrid at 0x7f57f6cf7890>
```



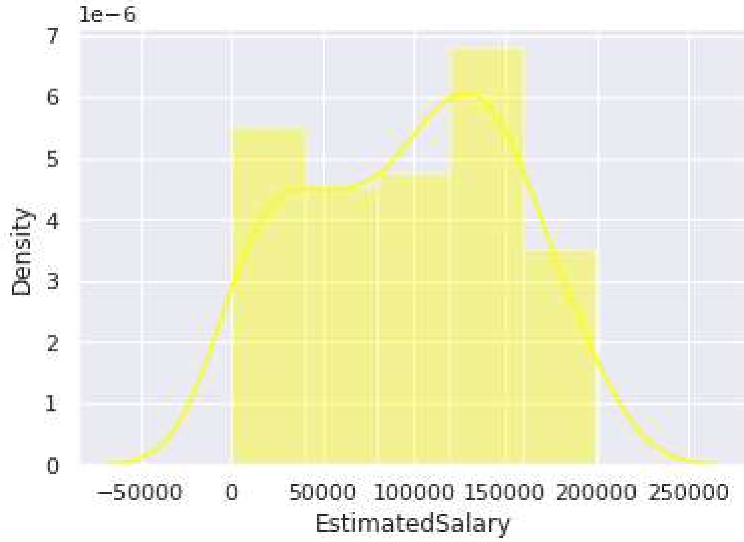
```
sns.distplot(dfv['Age'], color = 'Red', label = 'Age')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `d
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f6b5fc90>
```



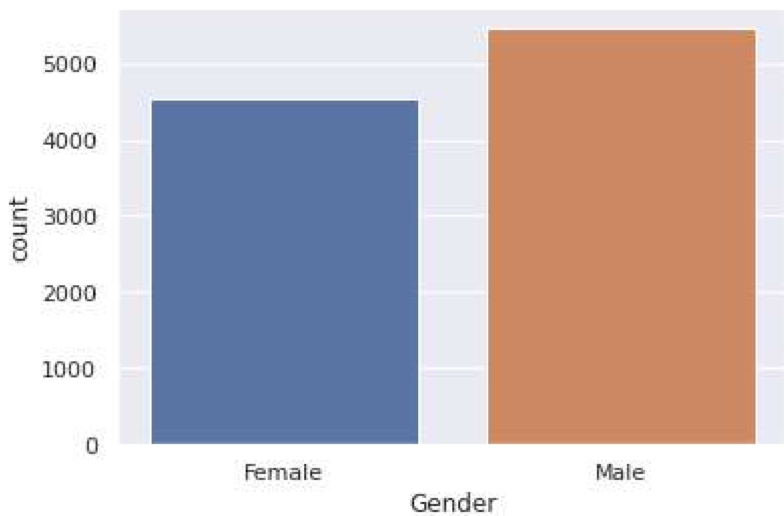
```
sns.distplot(dfv['EstimatedSalary'], color = 'Yellow', label = 'Salary')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `d
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f6cb76d0>
```




```
sns.countplot(x ='Gender', data = dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f49be9d0>
```

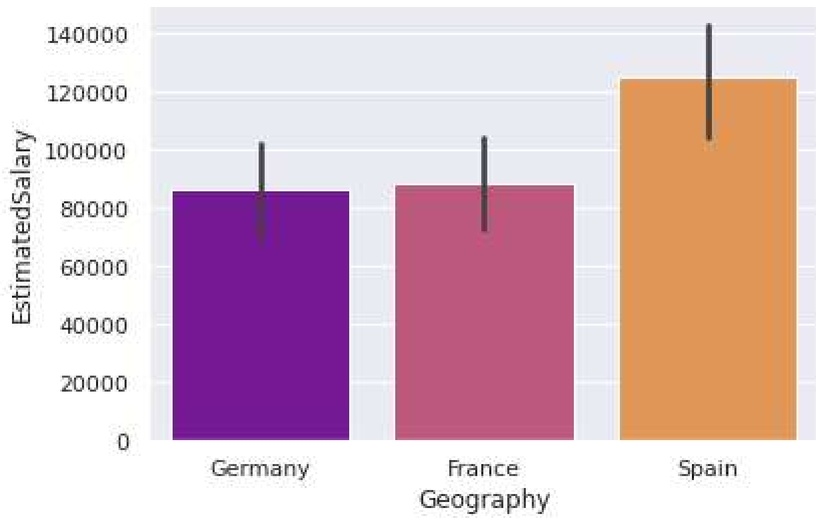



```
sns.set_style('darkgrid')
ac=0.14
sns.barplot(x ='Gender', y ='EstimatedSalary', data = dataset, palette ='plasma')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57e9d3f750>
```

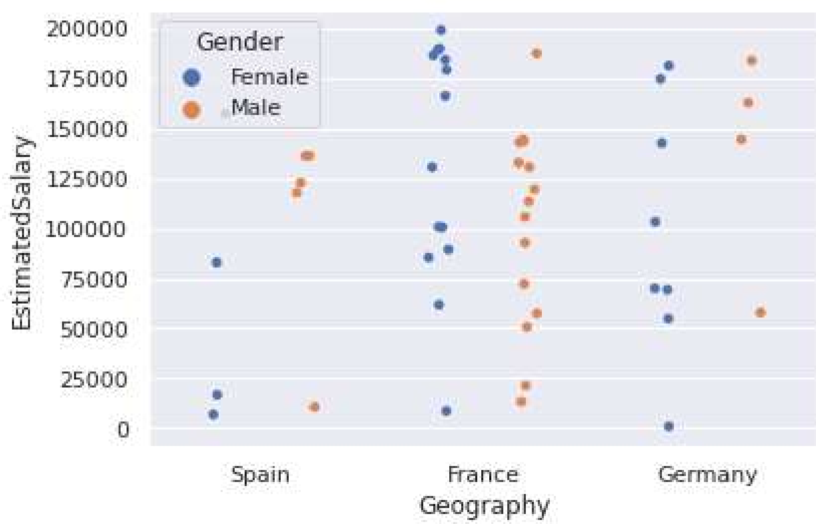

```
sns.barplot(x ='Geography', y ='EstimatedSalary', data = dfv, palette ='plasma')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f4961c90>
```



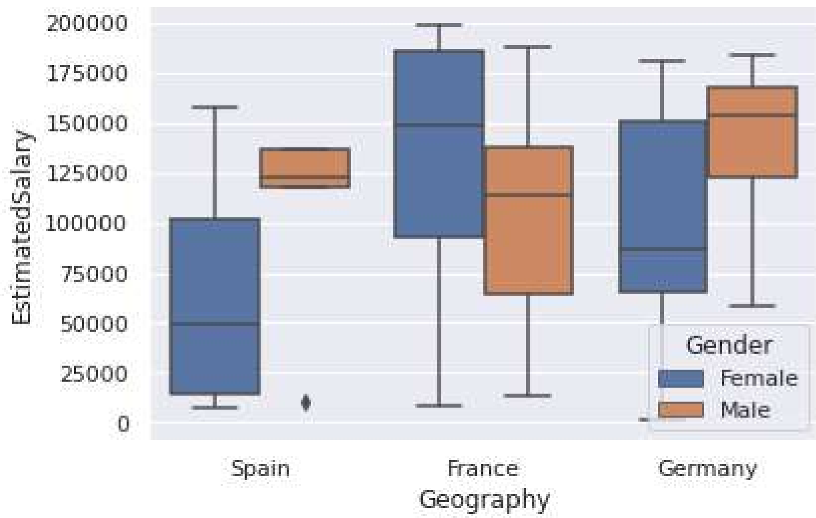

```
dfv1=dataset.sample(50)
sns.stripplot(x ='Geography', y ='EstimatedSalary', data = dfv1,
              jitter = True, hue ='Gender', dodge = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f48c7b90>
```



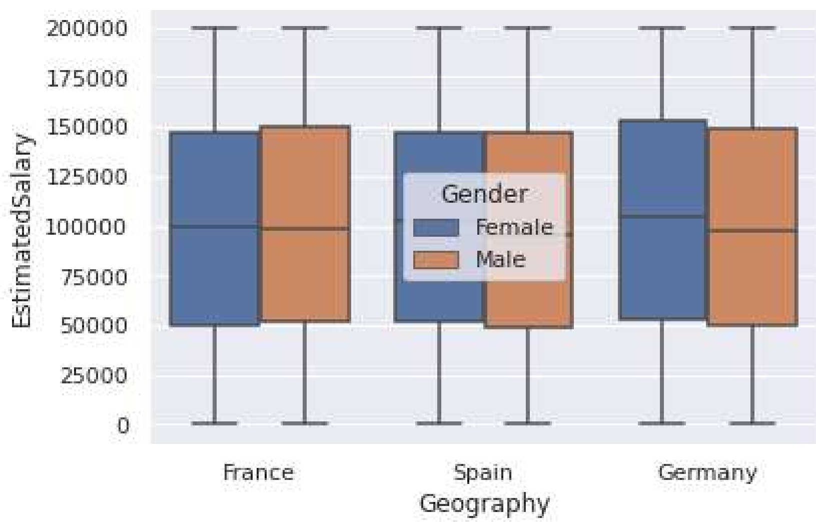

```
sns.boxplot(x ='Geography', y ='EstimatedSalary', data = dfv1, hue ='Gender')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f4858950>
```




```
sns.boxplot(x ='Geography', y ='EstimatedSalary', data = dataset, hue ='Gender')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f4761510>
```

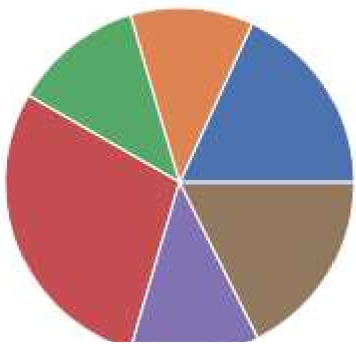



```
grp=dfv.groupby(['Gender','Geography']).EstimatedSalary.sum()
grp
```

```
Gender  Geography
Female  France       1754466.69
        Germany      1098005.85
        Spain        1173609.86
Male    France       2777209.15
        Germany      1154474.47
        Spain        1697592.87
Name: EstimatedSalary, dtype: float64
```

```
plt.pie(grp)
```

```
([<matplotlib.patches.Wedge at 0x7f57f45a8ed0>,
  <matplotlib.patches.Wedge at 0x7f57f45b72d0>,
  <matplotlib.patches.Wedge at 0x7f57f45b7390>,
  <matplotlib.patches.Wedge at 0x7f57f45b7a50>,
  <matplotlib.patches.Wedge at 0x7f57f45bf290>,
  <matplotlib.patches.Wedge at 0x7f57f45bfed0>],
 [Text(0.9255826115260446, 0.5943877768263975, ''),
  Text(0.07893729457242309, 1.0971640276301382, ''),
  Text(-0.6807536372164567, 0.8640454186074742, ''),
  Text(-1.0207159777132992, -0.41004742755050166, ''),
  Text(0.09937044375794502, -1.0955024029674008, ''),
  Text(0.9364228748040159, -0.577158729938119, '')])
```



DATA PREPROCESSING

```
dataset.isna()
```

```
dataset.isna().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             2
Balance            1
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
dropdf=dataset.dropna()
dropdf.isna().sum()
```

```
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
```

```
print(dataset.shape)
print(dropdf.shape)
```

```
(10000, 14)
(9997, 14)
```

```
print(dropdf)
```

```
      RowNumber  CustomerId    Surname  ...  IsActiveMember EstimatedSalary Exited
0             1    15634602   Hargrave  ...               1       101348.88      1
1             2    15647311       Hill  ...               1       112542.58      0
2             3    15619304       Onio  ...               0       113931.57      1
5             6    15574012        Chu  ...               0       149756.71      1
6             7    15592531   Bartlett  ...               1        10062.80      0
```

```
...          ...         ...        ...  ...             ...             ...    ...
9995        9996    15606229   Obijiaku  ...               0        96270.64      0
9996        9997    15569892  Johnstone  ...               1       101699.77      0
9997        9998    15584532        Liu  ...               1        42085.58      1
9998        9999    15682355  Sabbatini  ...               0        92888.52      1
9999       10000    15628319     Walker  ...               0        38190.78      0

[9997 rows x 14 columns]
```

```
#fill in missing values with mean
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(dataset.iloc[:, 7:9])
dataset.iloc[:, 7:9] = imputer.transform(dataset.iloc[:, 7:9])
print(dataset)
```

```
      RowNumber  CustomerId    Surname  ...  IsActiveMember EstimatedSalary Exited
0             1    15634602   Hargrave  ...               1       101348.88      1
1             2    15647311       Hill  ...               1       112542.58      0
2             3    15619304       Onio  ...               0       113931.57      1
3             4    15701354       Boni  ...               0        93826.63      0
4             5    15737888   Mitchell  ...               1        79084.10      0
...          ...         ...        ...  ...             ...             ...    ...
9995        9996    15606229   Obijiaku  ...               0        96270.64      0
9996        9997    15569892  Johnstone  ...               1       101699.77      0
9997        9998    15584532        Liu  ...               1        42085.58      1
9998        9999    15682355  Sabbatini  ...               0        92888.52      1
9999       10000    15628319     Walker  ...               0        38190.78      0

[10000 rows x 14 columns]
```

```
dataset.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
0	1	15634602	Hargrave	619	France	Female	42	2.000000	
1	2	15647311	Hill	608	Spain	Female	41	1.000000	838(
2	3	15619304	Onio	502	France	Female	42	8.000000	1596(
3	4	15701354	Boni	699	France	Female	39	1.000000	764(
4	5	15737888	Mitchell	850	Spain	Female	43	5.013003	12551

Encoding categorical data

Label Encoding the "Gender" column

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

```
print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

One Hot Encoding the "Geography" column

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrou
X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

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 = 0.2, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Part 2 - Building the ANN

Initializing the ANN

```
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the second hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the output layer

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the ANN

Compiling the ANN

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 72/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.794
Epoch 73/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.796
Epoch 74/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.799
Epoch 75/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.796
Epoch 76/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.794
Epoch 77/100
```

```
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.791
Epoch 78/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.793
Epoch 79/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.798
Epoch 80/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.799
Epoch 81/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.790
Epoch 82/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.808
Epoch 83/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.795
Epoch 84/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.796
Epoch 85/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.795
Epoch 86/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.804
Epoch 87/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.794
Epoch 88/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.795
Epoch 89/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.793
Epoch 90/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.798

Epoch 91/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.798
Epoch 92/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.792
Epoch 93/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.801
Epoch 94/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.795
Epoch 95/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.799
Epoch 96/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.797
Epoch 97/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.795
Epoch 98/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.790
Epoch 99/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.799
Epoch 100/100
250/250 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.793
<tensorflow.python.keras.callbacks.History at 0x7f57ec4cf050>
```

Part 4 - Making the predictions and evaluating the model

Predicting the result of a single observation

EXAMPLE OF PREDICTING SINGLE DATA

Use our ANN model to predict if the customer with the following informations will leave the bank:

Geography: France

Credit Score: 600

Gender: Male

Age: 40 years old

Tenure: 3 years

Balance: $ 60000

Number of Products: 2

Does this customer have a credit card ? Yes

Is this customer an Active Member: Yes

Estimated Salary: $ 50000

So, should we say goodbye to that customer ?

Solution

```
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)
```

```
[[False]]
```

Therefore, our ANN model predicts that this customer stays in the bank!

Important note 1: Notice that the values of the features were all input in a double pair of square brackets. That's because the "predict" method always expects a 2D array as the format of its inputs. And putting our values into a double pair of square brackets makes the input exactly a 2D array.

Important note 2: Notice also that the "France" country was not input as a string in the last column but as "1, 0, 0" in the first three columns. That's because of course the predict method expects the one-hot-encoded values of the state, and as we see in the first row of the matrix of features X, "France" was encoded as "1, 0, 0". And be careful to include these values in the first three columns, because the dummy variables are always created in the first columns.

Predicting the Test set results

```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)

print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
print(accuracy_score(y_test, y_pred)+ac)
```

```
0.9375
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1595    0]
 [ 405    0]]
```

0s completed at 3:43 PM