Sales Prediction: The Dataset used in these models tells about whether a person of certain age having certain income purchases a product or not. We need to predict whether a targeted audience will purchase the product or not.

In [1]:
```
# import the library
import pandas as pd
# load the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
```

In [2]:
```
# display the first five rows
dataset.head()
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [3]:
```
# check the null values
dataset.isnull().sum()
```

Out[3]:
```
User ID             0
Gender              0
Age                 0
EstimatedSalary     0
Purchased           0
dtype: int64
```

In [4]:
```
# ID is not helpful in predicting , so we will drop ID
dataset.drop(columns=['User ID'],inplace = True)
```

In [5]:
```
# Define X and y
# X is independent variable/features (Gender , Age and Estimated Salary)
# y is depencdent variable/label/target (Purchased)
X = dataset.iloc[:,0:3]
y = dataset.iloc[:,3]
```

In [6]:
```
# Character values in gender to be converted to numeric
# import the library
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder() # instantiating object of Labelencoder class.
X['Gender'] = le.fit_transform(X['Gender'])
X.head()
```

Out[6]:

	Gender	Age	EstimatedSalary
0	1	19	19000
1	1	35	20000
2	0	26	43000
3	0	27	57000
4	1	19	76000

In [7]:
```
# Split into train and test
# train will be used for training and test for testing
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=5)
```

In [8]:
```
dataset.describe()
```

Out[8]:

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

In [9]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64 
 2   EstimatedSalary  400 non-null    int64 
 3   Purchased        400 non-null    int64 
dtypes: int64(3), object(1)
memory usage: 12.6+ KB
```

In [10]:
```
# We need to standardise the values
# import the library
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [12]:
```
# Build the model
#import the library
import tensorflow as tf
from keras.layers import *
from keras.models import *
```

In [13]:
```
model = Sequential()
# input layer
model.add(Dense(16,input_dim=3,activation='relu'))
# hidden layer
model.add(Dense(16,activation='relu'))
model.add(Flatten())
# output layer
model.add(Dense(1 , activation= 'sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 16)                64        

 dense_1 (Dense)             (None, 16)                272       

 flatten (Flatten)           (None, 16)                0         

 dense_2 (Dense)             (None, 1)                 17        

=================================================================
Total params: 353 (1.38 KB)
Trainable params: 353 (1.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [14]:
```
# Compile the model
model.compile(loss='binary_crossentropy',optimizer='adam',metrics='accuracy')
```

In [15]:
```
# use fit to train the model
model.fit(X_train,y_train,batch_size=16,epochs=10)
```

```
Epoch 1/10
20/20 [==============================] - 1s 3ms/step - loss: 0.7318 - accuracy: 0.2344
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.6931 - accuracy: 0.4750
Epoch 3/10
20/20 [==============================] - 0s 2ms/step - loss: 0.6577 - accuracy: 0.7375
Epoch 4/10
20/20 [==============================] - 0s 2ms/step - loss: 0.6208 - accuracy: 0.8156
Epoch 5/10
20/20 [==============================] - 0s 2ms/step - loss: 0.5818 - accuracy: 0.8625
Epoch 6/10
20/20 [==============================] - 0s 2ms/step - loss: 0.5429 - accuracy: 0.8750
Epoch 7/10
20/20 [==============================] - 0s 2ms/step - loss: 0.5011 - accuracy: 0.8719
Epoch 8/10
20/20 [==============================] - 0s 2ms/step - loss: 0.4613 - accuracy: 0.8781
Epoch 9/10
20/20 [==============================] - 0s 2ms/step - loss: 0.4226 - accuracy: 0.8781
Epoch 10/10
20/20 [==============================] - 0s 2ms/step - loss: 0.3872 - accuracy: 0.8844
```

Out[15]:
```
<keras.src.callbacks.History at 0x78e7ddd75570>
```

In [19]:
```
# predict the values
y_pred = model.predict(X_test)
y_pred = (y_pred>0.5)
```

```
3/3 [==============================] - 0s 4ms/step
```

In [20]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
confusion_matrix(y_test,y_pred)
```

Out[20]:
```
array([[49,  4],
       [ 2, 25]])
```

In [21]:
```
accuracy_score(y_test,y_pred)
```

Out[21]:
```
0.925
```