Importing all necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Dropout
from tensorflow import keras
from tensorflow.keras.callbacks import EarlyStopping
```

Reading the dataset

In [2]:
```
df=pd.read_csv('E-commerce_Shipping_Data.csv')
df
```

Out[2]:

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost
0	1	D	Flight	4	2	
1	2	F	Flight	4	5	
2	3	A	Flight	2	2	
3	4	B	Flight	3	3	
4	5	C	Flight	2	2	
...	...	...	...	...	...	...
10994	10995	A	Ship	4	1	
10995	10996	B	Ship	4	1	
10996	10997	C	Ship	5	4	
10997	10998	F	Ship	5	2	
10998	10999	D	Ship	2	5	

10999 rows × 12 columns

Checking for null values present in dataset

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
ID                     0
Warehouse_block        0
Mode_of_Shipment       0
Customer_care_calls    0
Customer_rating        0
Cost_of_the_Product    0
Prior_purchases        0
Product_importance     0
Gender                 0
Discount_offered       0
Weight_in_gms          0
Reached.on.Time_Y.N    0
dtype: int64
```

As we can see there are no null values present in the dataset

Checking for missing values in dataset

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10999 entries, 0 to 10998
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   ID                   10999 non-null  int64 
 1   Warehouse_block      10999 non-null  object
 2   Mode_of_Shipment     10999 non-null  object
 3   Customer_care_calls  10999 non-null  int64 
 4   Customer_rating      10999 non-null  int64 
 5   Cost_of_the_Product  10999 non-null  int64 
 6   Prior_purchases      10999 non-null  int64 
 7   Product_importance   10999 non-null  object
 8   Gender               10999 non-null  object
 9   Discount_offered     10999 non-null  int64 
 10  Weight_in_gms        10999 non-null  int64 
 11  Reached.on.Time_Y.N  10999 non-null  int64 
dtypes: int64(8), object(4)
memory usage: 1.0+ MB
```

As we can see here there are no missing values in the dataset

Checking for descriptive statisitcs

In [5]:
```
df.describe()
```

Out[5]:

	ID	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases
count	10999.00000	10999.000000	10999.000000	10999.000000	10999.000000
mean	5500.00000	4.054459	2.990545	210.196836	3.567597
std	3175.28214	1.141490	1.413603	48.063272	1.522860
min	1.00000	2.000000	1.000000	96.000000	2.000000
25%	2750.50000	3.000000	2.000000	169.000000	3.000000
50%	5500.00000	4.000000	3.000000	214.000000	3.000000
75%	8249.50000	5.000000	4.000000	251.000000	4.000000
max	10999.00000	7.000000	5.000000	310.000000	10.000000

Descriptive statistics shows the mean and median values of the columns in dataset. This shows that all the columns have normal skewness except 2 columns which shows left and right skewness but does not affect the output categorical columns

Converting the categorical data into numerical values

In [6]:
```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
df[['Warehouse_block','Mode_of_Shipment','Product_importance','Gender']]=oe.fi
```

In [7]:
```
df
```

Out[7]:

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost
0	1	3.0	0.0	4	2	
1	2	4.0	0.0	4	5	
2	3	0.0	0.0	2	2	
3	4	1.0	0.0	3	3	
4	5	2.0	0.0	2	2	
...	...	...	...	...	...	
10994	10995	0.0	2.0	4	1	
10995	10996	1.0	2.0	4	1	
10996	10997	2.0	2.0	5	4	
10997	10998	4.0	2.0	5	2	
10998	10999	3.0	2.0	2	5	

10999 rows × 12 columns

As we can see all the categorical columns have been converted into their numerical form

Checking the correlation between columns

In [8]:
```
df.corr().style.background_gradient()
```

Out[8]:

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Cust
ID	1.000000	0.000070	-0.002459	0.188998	
Warehouse_block	0.000070	1.000000	0.000617	0.014496	
Mode_of_Shipment	-0.002459	0.000617	1.000000	-0.020164	
Customer_care_calls	0.188998	0.014496	-0.020164	1.000000	
Customer_rating	-0.005722	0.010169	0.001679	0.012209	
Cost_of_the_Product	0.196791	-0.006679	0.006681	0.323182	
Prior_purchases	0.145369	-0.005262	-0.001640	0.180771	
Product_importance	0.029081	0.004260	0.004911	0.006273	
Gender	-0.001695	-0.003700	-0.011288	0.002545	
Discount_offered	-0.598278	0.009569	0.009364	-0.130750	
Weight_in_gms	0.278312	0.004086	-0.000797	-0.276615	
Reached.on.Time_Y.N	-0.411822	0.005214	-0.000535	-0.067126	

As we can see, only the column Discount_offered is affecting the output target column strongly

Dropping Unecessary columns

In [9]:
```
df.drop(['ID'],axis=1,inplace=True)
```

In [10]:
```
df
```

Out[10]:

	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_
0	3.0	0.0	4	2	
1	4.0	0.0	4	5	
2	0.0	0.0	2	2	
3	1.0	0.0	3	3	
4	2.0	0.0	2	2	
...	...	...	...	...	
10994	0.0	2.0	4	1	
10995	1.0	2.0	4	1	
10996	2.0	2.0	5	4	
10997	4.0	2.0	5	2	
10998	3.0	2.0	2	5	

10999 rows × 11 columns

Splitting the dataset

In [11]:
```
x=df.iloc[:,0:-1].values
x
```

Out[11]:
```
array([[3.000e+00, 0.000e+00, 4.000e+00, ..., 0.000e+00, 4.400e+01,
        1.233e+03],
       [4.000e+00, 0.000e+00, 4.000e+00, ..., 1.000e+00, 5.900e+01,
        3.088e+03],
       [0.000e+00, 0.000e+00, 2.000e+00, ..., 1.000e+00, 4.800e+01,
        3.374e+03],
       ...,
       [2.000e+00, 2.000e+00, 5.000e+00, ..., 0.000e+00, 4.000e+00,
        1.155e+03],
       [4.000e+00, 2.000e+00, 5.000e+00, ..., 1.000e+00, 2.000e+00,
        1.210e+03],
       [3.000e+00, 2.000e+00, 2.000e+00, ..., 0.000e+00, 6.000e+00,
        1.639e+03]])
```

In [12]:
```
y=df.iloc[:,-1].values
y
```

Out[12]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

Data Scaling

In [13]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
```

Splitting the dataset into training and testing dataset

In [14]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=0)
```

In [15]:
```
xtrain
```

Out[15]:
```
array([[ 1.11803399,  0.63834175, -0.04771132, ..., -0.99176046,
        -0.57842252, -0.17306685],
       [ 1.11803399,  0.63834175, -0.92379938, ..., -0.99176046,
        -0.64013267,  0.96189107],
       [-1.56534517,  0.63834175, -0.92379938, ...,  1.00830799,
         2.38366454, -1.25176609],
       ...,
       [ 1.11803399, -0.68290796, -0.04771132, ...,  1.00830799,
        -0.64013267,  0.58459094],
       [ 1.11803399, -0.68290796, -0.04771132, ...,  1.00830799,
        -0.70184282, -1.16615504],
       [-1.56534517, -0.68290796, -0.04771132, ..., -0.99176046,
         0.22380939,  0.04952188]])
```

In [16]:
```
ytrain
```

Out[16]:
```
array([1, 1, 1, ..., 0, 1, 1], dtype=int64)
```

In [17]:
```
xtest
```

Out[17]:
```
array([[-1.56534517, -2.00415767,  0.82837675, ...,  1.00830799,
        -0.57842252,  0.49714537],
       [ 1.11803399, -2.00415767,  1.70446482, ..., -0.99176046,
        -0.70184282, -1.41136955],
       [ 1.11803399,  0.63834175, -1.79988745, ...,  1.00830799,
        -0.20816164, -0.13759942],
       ...,
       [ 1.11803399,  0.63834175, -0.92379938, ..., -0.99176046,
        -0.57842252,  0.49408783],
       [ 1.11803399, -0.68290796,  0.82837675, ..., -0.99176046,
        -0.51671238,  1.07868901],
       [-1.56534517,  0.63834175,  0.82837675, ...,  1.00830799,
        -0.57842252, -1.00532857]])
```

In [18]:
```
ytest
```

Out[18]:
```
array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
```

Building the model

In [19]:
```
ann=Sequential()
```

Adding the hidden and output Layers

In [30]:
```
ann.add(Dense(500,activation='relu'))
ann.add(Dropout(rate=0.3))
ann.add(Dense(300,activation='relu'))
ann.add(Dropout(rate=0.2))
ann.add(Dense(100,activation='relu'))
ann.add(Dropout(rate=0.1))
ann.add(Dense(1,activation='sigmoid'))
```

In [31]:
```
es=EarlyStopping(monitor='val_loss',patience=25,verbose=1,mode='min')
```

In [32]:
```
ann.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Training the model

In [33]:
```
history=ann.fit(xtrain,ytrain,epochs=300,batch_size=100,validation_data=(xtest
```

```
Epoch 1/300
77/77 [==============================] - 2s 7ms/step - loss: 0.5288 - accurac
y: 0.6668 - val_loss: 0.5504 - val_accuracy: 0.6748
Epoch 2/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4797 - accurac
y: 0.7101 - val_loss: 0.5657 - val_accuracy: 0.6706
Epoch 3/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4762 - accurac
y: 0.7207 - val_loss: 0.5833 - val_accuracy: 0.6624
Epoch 4/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4684 - accurac
y: 0.7253 - val_loss: 0.5448 - val_accuracy: 0.6733
Epoch 5/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4683 - accurac
y: 0.7254 - val_loss: 0.5851 - val_accuracy: 0.6597
Epoch 6/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4636 - accurac
y: 0.7317 - val_loss: 0.5742 - val_accuracy: 0.6764
Epoch 7/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4608 - accurac
y: 0.7346 - val_loss: 0.5611 - val_accuracy: 0.6715
Epoch 8/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4603 - accurac
y: 0.7279 - val_loss: 0.5738 - val_accuracy: 0.6627
Epoch 9/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4539 - accurac
y: 0.7332 - val_loss: 0.6106 - val_accuracy: 0.6694
Epoch 10/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4597 - accurac
y: 0.7268 - val_loss: 0.5898 - val_accuracy: 0.6570
Epoch 11/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4542 - accurac
y: 0.7400 - val_loss: 0.5779 - val_accuracy: 0.6667
Epoch 12/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4534 - accurac
y: 0.7349 - val_loss: 0.6298 - val_accuracy: 0.6518
Epoch 13/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4492 - accurac
y: 0.7431 - val_loss: 0.5993 - val_accuracy: 0.6591
Epoch 14/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4515 - accurac
y: 0.7383 - val_loss: 0.6172 - val_accuracy: 0.6548
Epoch 15/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4514 - accurac
y: 0.7379 - val_loss: 0.6275 - val_accuracy: 0.6715
Epoch 16/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4489 - accurac
y: 0.7448 - val_loss: 0.5789 - val_accuracy: 0.6700
Epoch 17/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4430 - accurac
y: 0.7463 - val_loss: 0.5856 - val_accuracy: 0.6664
Epoch 18/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4423 - accurac
y: 0.7444 - val_loss: 0.6066 - val_accuracy: 0.6745
Epoch 19/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4450 - accurac
y: 0.7458 - val_loss: 0.5702 - val_accuracy: 0.6658
Epoch 20/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4406 - accurac
y: 0.7517 - val_loss: 0.6331 - val_accuracy: 0.6618
Epoch 21/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4425 - accurac
y: 0.7510 - val_loss: 0.5899 - val_accuracy: 0.6618
Epoch 22/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4396 - accurac
y: 0.7535 - val_loss: 0.6142 - val_accuracy: 0.6727
Epoch 23/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4409 - accurac
y: 0.7527 - val_loss: 0.6167 - val_accuracy: 0.6521
Epoch 24/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4394 - accurac
y: 0.7537 - val_loss: 0.6059 - val_accuracy: 0.6506
Epoch 25/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4341 - accurac
```

```
y: 0.7570 - val_loss: 0.6181 - val_accuracy: 0.6488
Epoch 26/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4333 - accurac
y: 0.7593 - val_loss: 0.6517 - val_accuracy: 0.6639
Epoch 27/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4312 - accurac
y: 0.7605 - val_loss: 0.6165 - val_accuracy: 0.6664
Epoch 28/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4318 - accurac
y: 0.7549 - val_loss: 0.6241 - val_accuracy: 0.6497
Epoch 29/300
77/77 [==============================] - 0s 5ms/step - loss: 0.4289 - accurac
y: 0.7583 - val_loss: 0.6387 - val_accuracy: 0.6545
Epoch 29: early stopping
```

Making predictions

In [34]:
```
ypred=ann.predict(xtest)
```

```
104/104 [==============================] - 0s 884us/step
```

In [35]:
```
ypred=np.where(ypred>0.5,1,0)
ypred
```

Out[35]:
```
array([[1],
       [1],
       [1],
       ...,
       [0],
       [0],
       [1]])
```

Checking Accuracy ¶

In [36]:
```
from sklearn.metrics import classification_report
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.58      0.66      0.62      1379
           1       0.73      0.65      0.69      1921

    accuracy                           0.65      3300
   macro avg       0.65      0.66      0.65      3300
weighted avg       0.66      0.65      0.66      3300
```

Accuracy of our model is 65%

In [ ]:
```

```