In [203…
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [204…
```
df=pd.read_csv(r"C:\Users\Shruthy\WA_Fn-UseC_-Telco-Customer-Churn.csv")
df.head(1)
```

Out[204]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	T
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	

1 rows × 21 columns

In [205…
```
#drop customer id
df.drop('customerID',axis=1,inplace=True)
```

In [206…
```
df.head(1)
```

Out[206]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	Te
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	

In [207…
```
df.dtypes
```

Out[207]:
```
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges        object
Churn               object
dtype: object
```

In [208…
```
# Convert 'TotalCharges' column from object to numerical
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# 'errors='coerce'' handles any invalid values by converting them to NaN

# Print the updated DataFrame
print(df.dtypes)
```

```
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges       float64
Churn               object
dtype: object
```

In [209…
```
df.shape
```

Out[209]:
```
(7043, 20)
```

In [210…
```
df.drop_duplicates()
df.shape
```

Out[210]:
```
(7043, 20)
```

In [211…
```
df.isnull().sum()
```

Out[211]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges       11
Churn               0
dtype: int64
```

In [212…
```
# Remove null values from 'TotalCharges' column
df = df.dropna(subset=['TotalCharges'])

print(df.isnull().sum())
```

```
gender             0
SeniorCitizen      0
Partner            0
Dependents         0
tenure             0
PhoneService       0
MultipleLines      0
InternetService    0
OnlineSecurity     0
OnlineBackup       0
DeviceProtection   0
TechSupport        0
StreamingTV        0
StreamingMovies    0
Contract           0
PaperlessBilling   0
PaymentMethod      0
MonthlyCharges     0
TotalCharges       0
Churn              0
dtype: int64
```

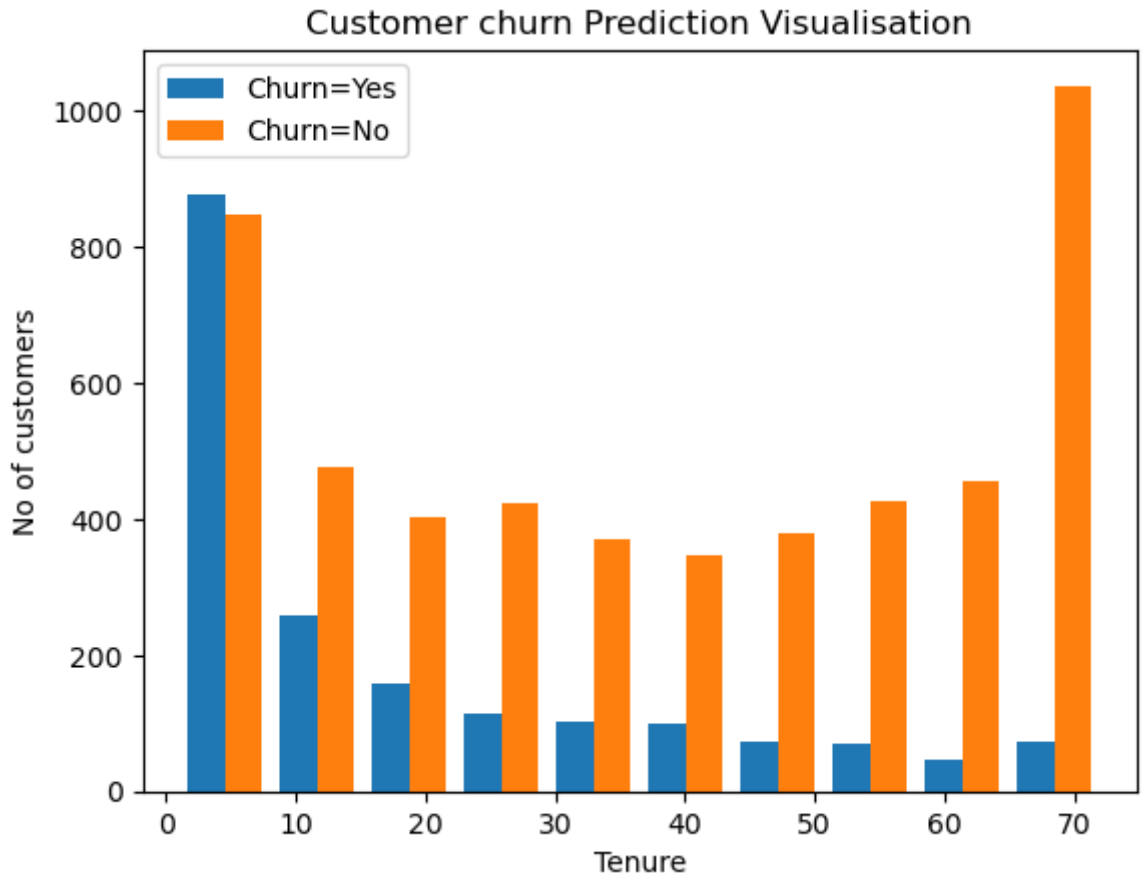
In [213…
```
df.shape
```

Out[213]:
```
(7032, 20)
```

EDA

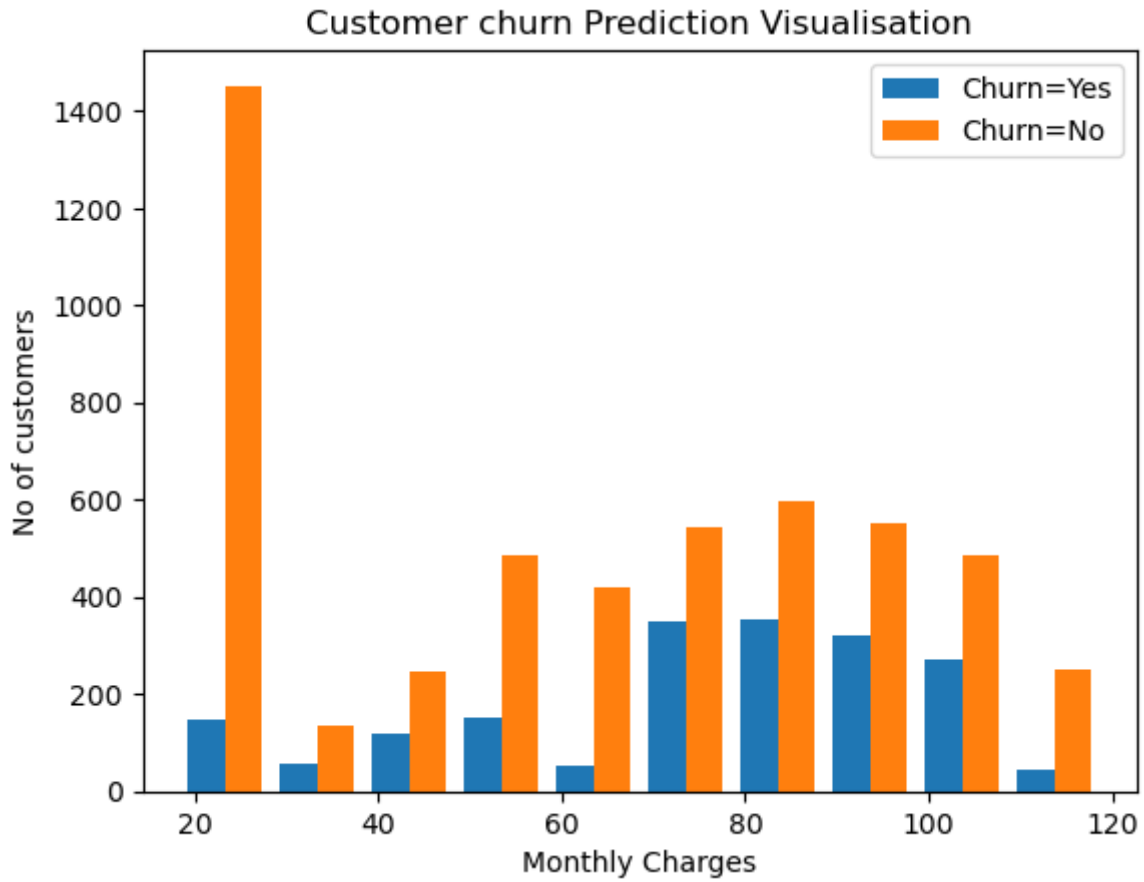
In [214…
```
tenure_churn_no=df[df.Churn=='No'].tenure
tenure_churn_yes=df[df.Churn=='Yes'].tenure
plt.hist([tenure_churn_yes,tenure_churn_no],label=['Churn=Yes','Churn=No'])
plt.legend()
plt.xlabel('Tenure')
plt.ylabel('No of customers')
plt.title('Customer churn Prediction Visualisation')
```

Out[214]:
```
Text(0.5, 1.0, 'Customer churn Prediction Visualisation')
```



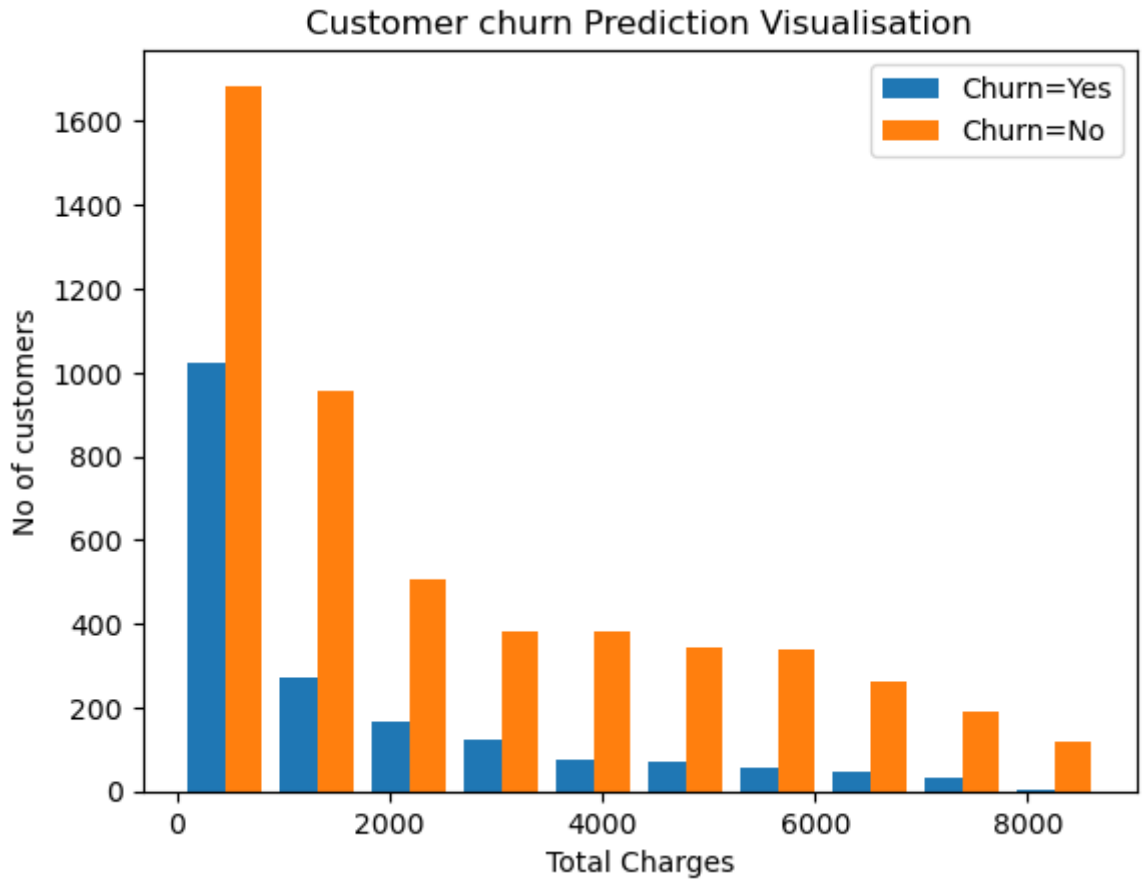
In [215…
```
monthly_charges_no=df[df.Churn=='No'].MonthlyCharges
monthly_charges_yes=df[df.Churn=='Yes'].MonthlyCharges
plt.hist([monthly_charges_yes,monthly_charges_no],label=['Churn=Yes','Churn=No'])
plt.legend()
plt.xlabel('Monthly Charges')
plt.ylabel('No of customers')
plt.title('Customer churn Prediction Visualisation')
```

Out[215]:
```
Text(0.5, 1.0, 'Customer churn Prediction Visualisation')
```



In [216…
```
Total_charges_no=df[df.Churn=='No'].TotalCharges
Total_charges_yes=df[df.Churn=='Yes'].TotalCharges
plt.hist([Total_charges_yes,Total_charges_no],label=['Churn=Yes','Churn=No'])
plt.legend()
plt.xlabel('Total Charges')
plt.ylabel('No of customers')
plt.title('Customer churn Prediction Visualisation')
```

Out[216]:
```
Text(0.5, 1.0, 'Customer churn Prediction Visualisation')
```



In [217…

```
df.nunique()
```

Out[217]:
```
gender              2
SeniorCitizen       2
Partner             2
Dependents          2
tenure             72
PhoneService        2
MultipleLines       3
InternetService     3
OnlineSecurity      3
OnlineBackup        3
DeviceProtection    3
TechSupport         3
StreamingTV         3
StreamingMovies     3
Contract            3
PaperlessBilling    2
PaymentMethod       4
MonthlyCharges   1584
TotalCharges     6530
Churn               2
dtype: int64
```

In [219…
```
df.replace('No phone service','No',inplace=True)
df.replace('No internet service','No',inplace=True)
```

In [220…
```
df.columns
```

Out[220]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
       'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [221…
```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','OnlineBackup',
                  'DeviceProtection','TechSupport','StreamingTV','StreamingMovies','PaperlessBilling','Churn']
for col in yes_no_columns:
    df[col].replace({'Yes': 1,'No': 0},inplace=True)
```

In [222…
```
df['gender'].unique()
```

Out[222]:
```
array(['Female', 'Male'], dtype=object)
```

In [223…
```
df['gender'].replace({'Female':1,'Male':0},inplace=True)
```

In [224…
```
df['gender'].unique()
```

Out[224]:
```
array([1, 0], dtype=int64)
```

In [225…
```
df['Dependents'].unique()
```

Out[225]:
```
array([0, 1], dtype=int64)
```

In [226…
```
df1=pd.get_dummies(data=df,columns=['InternetService','Contract','PaymentMethod'])
df1.columns
```

Out[226]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

In [227…
```
df1.dtypes
```

Out[227]:
```
gender                                     int64
SeniorCitizen                              int64
Partner                                    int64
Dependents                                 int64
tenure                                     int64
PhoneService                               int64
MultipleLines                              int64
OnlineSecurity                             int64
OnlineBackup                               int64
DeviceProtection                           int64
TechSupport                                int64
StreamingTV                                int64
StreamingMovies                            int64
PaperlessBilling                           int64
MonthlyCharges                           float64
TotalCharges                             float64
Churn                                      int64
InternetService_DSL                        uint8
InternetService_Fiber optic                uint8
InternetService_No                         uint8
Contract_Month-to-month                    uint8
Contract_One year                          uint8
Contract_Two year                          uint8
PaymentMethod_Bank transfer (automatic)    uint8
PaymentMethod_Credit card (automatic)      uint8
PaymentMethod_Electronic check             uint8
PaymentMethod_Mailed check                 uint8
dtype: object
```

In [228…
```
col_to_scale=['tenure','MonthlyCharges','TotalCharges']
#convert values to 0-1
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
df1[col_to_scale] = scaler.fit_transform(df1[col_to_scale])
```

In [229…
```
df1[col_to_scale]
```

Out[229]:

	tenure	MonthlyCharges	TotalCharges
0	0.000000	0.115423	0.001275
1	0.464789	0.385075	0.215867
2	0.014085	0.354229	0.010310
3	0.619718	0.239303	0.210241
4	0.014085	0.521891	0.015330
...	...	...	...
7038	0.323944	0.662189	0.227521
7039	1.000000	0.845274	0.847461
7040	0.140845	0.112935	0.037809
7041	0.042254	0.558706	0.033210
7042	0.915493	0.869652	0.787641

7032 rows × 3 columns

In [230…
```
X=df1.drop('Churn',axis='columns')
y=df1['Churn']
```

In [231…
```
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 [232…
```
X_train.shape
```

Out[232]:
```
(5625, 26)
```

In [233…
```
X_test.shape
```

Out[233]:
```
(1407, 26)
```

In [234…
```
len(X_train.columns)
```

Out[234]: 26

In [235…

```
import tensorflow as tf
from tensorflow import keras


model = keras.Sequential([
    keras.layers.Dense(20, input_shape=(26,), activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
176/176 [==============================] - 0s 843us/step - loss: 0.5559 - accuracy: 0.7035
Epoch 2/100
176/176 [==============================] - 0s 911us/step - loss: 0.4379 - accuracy: 0.7938
Epoch 3/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4228 - accuracy: 0.7984
Epoch 4/100
176/176 [==============================] - 0s 927us/step - loss: 0.4174 - accuracy: 0.8032
Epoch 5/100
176/176 [==============================] - 0s 838us/step - loss: 0.4147 - accuracy: 0.8034
Epoch 6/100
176/176 [==============================] - 0s 834us/step - loss: 0.4137 - accuracy: 0.8036
Epoch 7/100
176/176 [==============================] - 0s 839us/step - loss: 0.4121 - accuracy: 0.8075
Epoch 8/100
176/176 [==============================] - 0s 818us/step - loss: 0.4108 - accuracy: 0.8071
Epoch 9/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4101 - accuracy: 0.8092
Epoch 10/100
176/176 [==============================] - 0s 959us/step - loss: 0.4102 - accuracy: 0.8073
Epoch 11/100
176/176 [==============================] - 0s 844us/step - loss: 0.4085 - accuracy: 0.8092
Epoch 12/100
176/176 [==============================] - 0s 868us/step - loss: 0.4078 - accuracy: 0.8092
Epoch 13/100
176/176 [==============================] - 0s 844us/step - loss: 0.4072 - accuracy: 0.8103
Epoch 14/100
176/176 [==============================] - 0s 878us/step - loss: 0.4066 - accuracy: 0.8101
Epoch 15/100
176/176 [==============================] - 0s 864us/step - loss: 0.4063 - accuracy: 0.8096
Epoch 16/100
176/176 [==============================] - 0s 957us/step - loss: 0.4051 - accuracy: 0.8121
Epoch 17/100
176/176 [==============================] - 0s 850us/step - loss: 0.4051 - accuracy: 0.8119
Epoch 18/100
176/176 [==============================] - 0s 864us/step - loss: 0.4037 - accuracy: 0.8139
Epoch 19/100
176/176 [==============================] - 0s 935us/step - loss: 0.4036 - accuracy: 0.8098
Epoch 20/100
176/176 [==============================] - 0s 841us/step - loss: 0.4029 - accuracy: 0.8105
Epoch 21/100
176/176 [==============================] - 0s 836us/step - loss: 0.4023 - accuracy: 0.8137
Epoch 22/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4019 - accuracy: 0.8130
```

```
Epoch 23/100
176/176 [==============================] - 0s 837us/step - loss: 0.4011 - accuracy: 0.8126
Epoch 24/100
176/176 [==============================] - 0s 834us/step - loss: 0.4013 - accuracy: 0.8142
Epoch 25/100
176/176 [==============================] - 0s 847us/step - loss: 0.4004 - accuracy: 0.8144
Epoch 26/100
176/176 [==============================] - 0s 1ms/step - loss: 0.4003 - accuracy: 0.8139
Epoch 27/100
176/176 [==============================] - 0s 827us/step - loss: 0.4002 - accuracy: 0.8132
Epoch 28/100
176/176 [==============================] - 0s 965us/step - loss: 0.3996 - accuracy: 0.8112
Epoch 29/100
176/176 [==============================] - 0s 980us/step - loss: 0.3990 - accuracy: 0.8148
Epoch 30/100
176/176 [==============================] - 0s 871us/step - loss: 0.3989 - accuracy: 0.8140
Epoch 31/100
176/176 [==============================] - 0s 839us/step - loss: 0.3986 - accuracy: 0.8142
Epoch 32/100
176/176 [==============================] - 0s 915us/step - loss: 0.3977 - accuracy: 0.8158
Epoch 33/100
176/176 [==============================] - 0s 789us/step - loss: 0.3974 - accuracy: 0.8156
Epoch 34/100
176/176 [==============================] - 0s 807us/step - loss: 0.3974 - accuracy: 0.8178
Epoch 35/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3975 - accuracy: 0.8156
Epoch 36/100
176/176 [==============================] - 0s 854us/step - loss: 0.3969 - accuracy: 0.8158
Epoch 37/100
176/176 [==============================] - 0s 833us/step - loss: 0.3961 - accuracy: 0.8164
Epoch 38/100
176/176 [==============================] - 0s 841us/step - loss: 0.3955 - accuracy: 0.8151
Epoch 39/100
176/176 [==============================] - 0s 821us/step - loss: 0.3956 - accuracy: 0.8153
Epoch 40/100
176/176 [==============================] - 0s 845us/step - loss: 0.3949 - accuracy: 0.8176
Epoch 41/100
176/176 [==============================] - 0s 972us/step - loss: 0.3948 - accuracy: 0.8164
Epoch 42/100
176/176 [==============================] - 0s 950us/step - loss: 0.3944 - accuracy: 0.8155
Epoch 43/100
176/176 [==============================] - 0s 903us/step - loss: 0.3939 - accuracy: 0.8153
Epoch 44/100
176/176 [==============================] - 0s 902us/step - loss: 0.3943 - accuracy: 0.8162
```

```
Epoch 45/100
176/176 [==============================] - 0s 848us/step - loss: 0.3938 - accuracy: 0.8164
Epoch 46/100
176/176 [==============================] - 0s 802us/step - loss: 0.3928 - accuracy: 0.8190
Epoch 47/100
176/176 [==============================] - 0s 872us/step - loss: 0.3926 - accuracy: 0.8156
Epoch 48/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3927 - accuracy: 0.8171
Epoch 49/100
176/176 [==============================] - 0s 834us/step - loss: 0.3922 - accuracy: 0.8172
Epoch 50/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3915 - accuracy: 0.8165
Epoch 51/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3916 - accuracy: 0.8187
Epoch 52/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3913 - accuracy: 0.8181
Epoch 53/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3909 - accuracy: 0.8192
Epoch 54/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3906 - accuracy: 0.8185
Epoch 55/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3905 - accuracy: 0.8172
Epoch 56/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3900 - accuracy: 0.8192
Epoch 57/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3899 - accuracy: 0.8167
Epoch 58/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3900 - accuracy: 0.8187
Epoch 59/100
176/176 [==============================] - 0s 963us/step - loss: 0.3895 - accuracy: 0.8194
Epoch 60/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3892 - accuracy: 0.8185
Epoch 61/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3889 - accuracy: 0.8192
Epoch 62/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3891 - accuracy: 0.8187
Epoch 63/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3885 - accuracy: 0.8208
Epoch 64/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3881 - accuracy: 0.8213
Epoch 65/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3874 - accuracy: 0.8192
Epoch 66/100
176/176 [==============================] - 0s 930us/step - loss: 0.3881 - accuracy: 0.8196
```

```
Epoch 67/100
176/176 [==============================] - 0s 909us/step - loss: 0.3872 - accuracy: 0.8188
Epoch 68/100
176/176 [==============================] - 0s 894us/step - loss: 0.3870 - accuracy: 0.8208
Epoch 69/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3870 - accuracy: 0.8217
Epoch 70/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3869 - accuracy: 0.8203
Epoch 71/100
176/176 [==============================] - 0s 994us/step - loss: 0.3866 - accuracy: 0.8201
Epoch 72/100
176/176 [==============================] - 0s 828us/step - loss: 0.3868 - accuracy: 0.8208
Epoch 73/100
176/176 [==============================] - 0s 841us/step - loss: 0.3865 - accuracy: 0.8210
Epoch 74/100
176/176 [==============================] - 0s 804us/step - loss: 0.3860 - accuracy: 0.8220
Epoch 75/100
176/176 [==============================] - 0s 804us/step - loss: 0.3856 - accuracy: 0.8210
Epoch 76/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3850 - accuracy: 0.8201
Epoch 77/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3854 - accuracy: 0.8208
Epoch 78/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3855 - accuracy: 0.8201
Epoch 79/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3853 - accuracy: 0.8208
Epoch 80/100
176/176 [==============================] - 0s 977us/step - loss: 0.3849 - accuracy: 0.8212
Epoch 81/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3848 - accuracy: 0.8197
Epoch 82/100
176/176 [==============================] - 0s 852us/step - loss: 0.3843 - accuracy: 0.8224
Epoch 83/100
176/176 [==============================] - 0s 893us/step - loss: 0.3849 - accuracy: 0.8220
Epoch 84/100
176/176 [==============================] - 0s 964us/step - loss: 0.3839 - accuracy: 0.8238
Epoch 85/100
176/176 [==============================] - 0s 901us/step - loss: 0.3839 - accuracy: 0.8197
Epoch 86/100
176/176 [==============================] - 0s 845us/step - loss: 0.3837 - accuracy: 0.8229
Epoch 87/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3841 - accuracy: 0.8203
Epoch 88/100
176/176 [==============================] - 0s 988us/step - loss: 0.3835 - accuracy: 0.8235
```

```
Epoch 89/100
176/176 [==============================] - 0s 853us/step - loss: 0.3839 - accuracy: 0.8212
Epoch 90/100
176/176 [==============================] - 0s 783us/step - loss: 0.3830 - accuracy: 0.8228
Epoch 91/100
176/176 [==============================] - 0s 978us/step - loss: 0.3832 - accuracy: 0.8208
Epoch 92/100
176/176 [==============================] - 0s 934us/step - loss: 0.3825 - accuracy: 0.8228
Epoch 93/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3825 - accuracy: 0.8203
Epoch 94/100
176/176 [==============================] - 0s 869us/step - loss: 0.3823 - accuracy: 0.8222
Epoch 95/100
176/176 [==============================] - 0s 806us/step - loss: 0.3825 - accuracy: 0.8233
Epoch 96/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3821 - accuracy: 0.8240
Epoch 97/100
176/176 [==============================] - 0s 848us/step - loss: 0.3818 - accuracy: 0.8220
Epoch 98/100
176/176 [==============================] - 0s 828us/step - loss: 0.3824 - accuracy: 0.8222
Epoch 99/100
176/176 [==============================] - 0s 1ms/step - loss: 0.3818 - accuracy: 0.8245
Epoch 100/100
176/176 [==============================] - 0s 887us/step - loss: 0.3822 - accuracy: 0.8249
```

Out[235]:
```
<keras.callbacks.History at 0x283f005d280>
```

In [236…
```
model.evaluate(X_test,y_test)
```

```
44/44 [==============================] - 0s 1ms/step - loss: 0.4564 - accuracy: 0.7783
```

Out[236]:
```
[0.45642077922821045, 0.778251588344574]
```

In [239…
```
y_pred=model.predict(X_test)
y_pred[:2]
```

```
44/44 [==============================] - 0s 1ms/step
```

Out[239]:
```
array([[0.24458884],
       [0.45865902]], dtype=float32)
```

In [240…
```
y_test
```

Out[240]:
```
2660    0
744     0
5579    1
64      1
3287    1
       ..
2024    0
4396    1
4081    1
1297    0
4899    1
Name: Churn, Length: 1407, dtype: int64
```

In [242…
```
y_prediction=[]
for elements in y_pred:
    if elements>0.5:
         y_prediction.append(1)
    else:
         y_prediction.append(0)
```

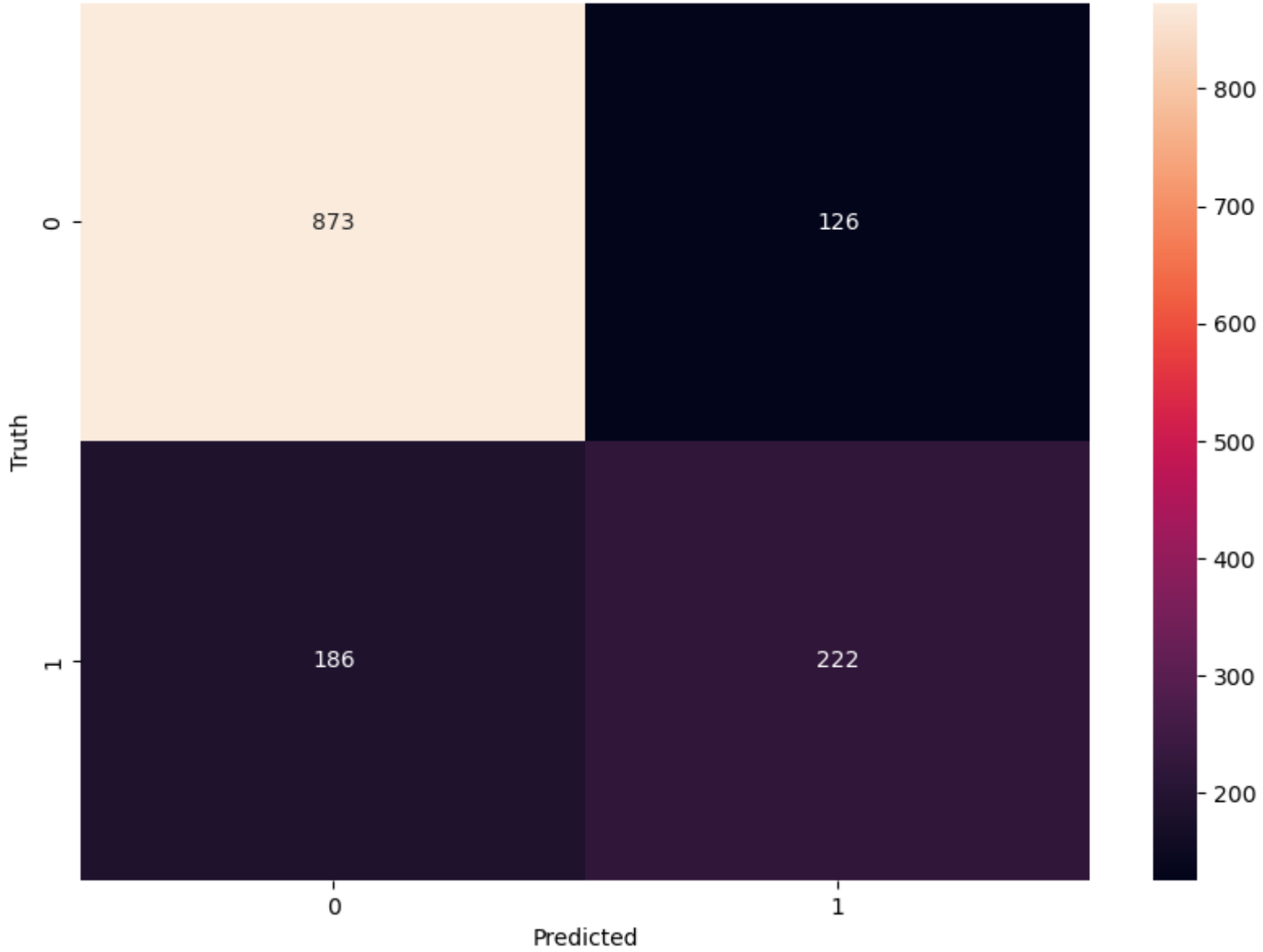
In [244…
```
y_prediction[:5]
```

Out[244]:
```
[0, 0, 0, 1, 0]
```

In [246…
```
import seaborn as sn
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_prediction)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[246]:
```
Text(95.72222222222221, 0.5, 'Truth')
```



ACCURACY

In [247…

```
round((873+222)/(873+222+186+126),2)
```

Out[247]: 0.78

Precision for 0 class. i.e. Precision for customers who did not churn

In [249…
```
round(873/(873+186),2)
```

Out[249]: 0.82

Precision for 1 class. i.e. Precision for customers who actually churned

In [250…
```
round(222/(222+126),2)
```

Out[250]: 0.64

Recall for 0

In [251…
```
round(873/(873+126),2)
```

Out[251]: 0.87

Recall for 1

In [252…
```
round(222/(222+186),2)
```

Out[252]: 0.54

In [ ]:
```

```