```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline
```

```
df = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn.csv")
df
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	
...	...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	

7043 rows × 21 columns

```
df.drop('customerID',axis='columns',inplace=True)
```

```
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         object
Churn                object
dtype: object
```

```
df.TotalCharges.values
```

```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

```
pd.to_numeric(df.TotalCharges)
```

```
    ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call last)
    /usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
    pandas._libs.lib.maybe_convert_numeric()

    ValueError: Unable to parse string " "

    During handling of the above exception, another exception occurred:

    ValueError                                Traceback (most recent call last)
    ───────────────────────────────── 2 frames ──────────────────────────────────
    /usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
    pandas._libs.lib.maybe_convert_numeric()
```

```
pd.to_numeric(df.TotalCharges,errors='coerce').isnull()
```

```
    0       False
    1       False
    2       False
    3       False
    4       False
            ...
    7038    False
    7039    False
    7040    False
    7041    False
    7042    False
    Name: TotalCharges, Length: 7043, dtype: bool
```

```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
488	Female	0	Yes	Yes	0	No	No phone service	DSL
753	Male	0	No	Yes	0	Yes	No	No
936	Female	0	Yes	Yes	0	Yes	No	DSL
1082	Male	0	Yes	Yes	0	Yes	Yes	No
1340	Female	0	Yes	Yes	0	No	No phone service	DSL
3331	Male	0	Yes	Yes	0	Yes	No	No
3826	Male	0	Yes	Yes	0	Yes	Yes	No
4380	Female	0	Yes	Yes	0	Yes	No	No
5218	Male	0	Yes	Yes	0	Yes	No	No
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL
6754	Male	0	No	Yes	0	Yes	Yes	DSL

```
df.shape
```

```
    (7043, 20)
```

```
df.iloc[488].TotalCharges
```

```
    ' '
```

```
df[df.TotalCharges!=' '].shape
```

```
    (7032, 20)
```

```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

```
(7032, 20)
```

```
df1.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         object
Churn                object
dtype: object
```

```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
<ipython-input-143-b67e0c3d31a6>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
df1.TotalCharges.values
```

```
array([  29.85, 1889.5 ,  108.15, ...,  346.45,  306.6 , 6844.5 ])
```

```
df1[df1.Churn=='No']
```

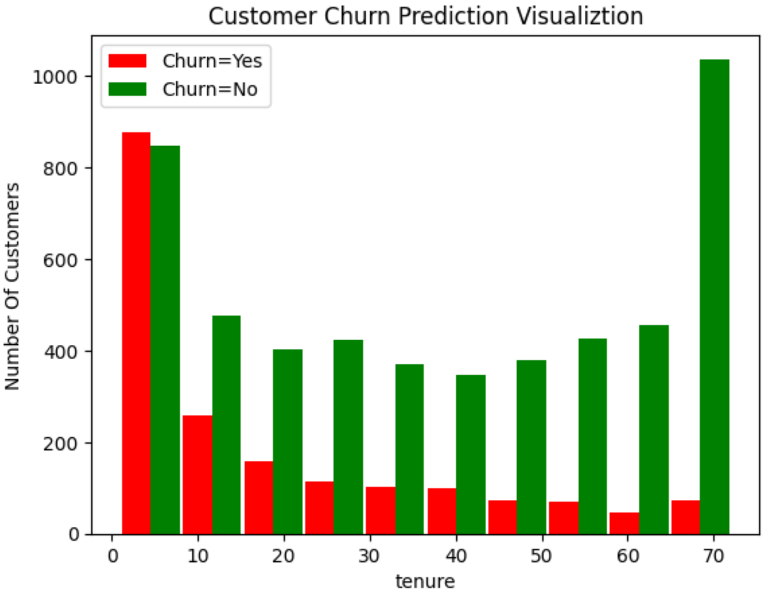
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	Female	0	Yes	No	1	No	No phone service	DSL
1	Male	0	No	No	34	Yes	No	DSL
3	Male	0	No	No	45	No	No phone service	DSL
6	Male	0	No	Yes	22	Yes	Yes	Fiber optic
7	Female	0	No	No	10	No	No phone service	DSL
...	...	...	...	...	...	...	...	...
7037	Female	0	No	No	72	Yes	No	No
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic
7040	Female	0	Yes	Yes	11	No	No phone service	DSL
7042	Male	0	No	No	66	Yes	No	Fiber optic

5163 rows × 20 columns

Visualization

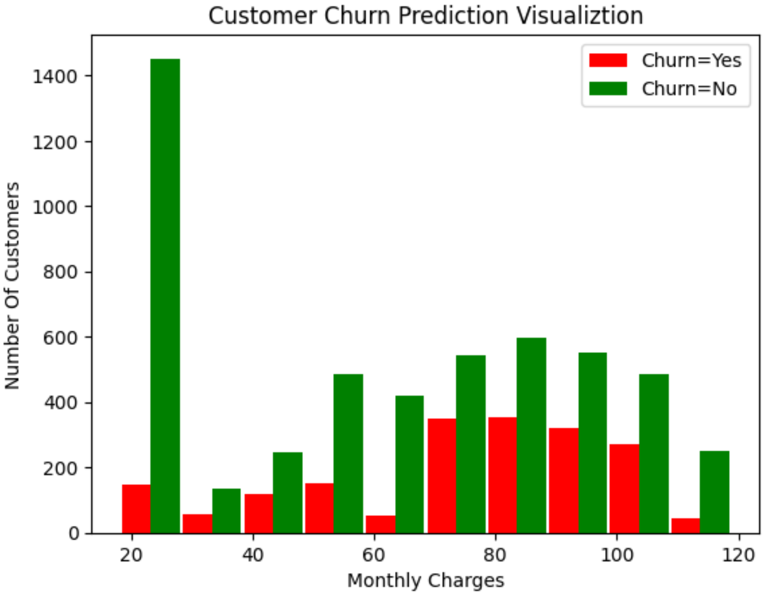
```
tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure

plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")

plt.hist([tenure_churn_yes, tenure_churn_no], rwidth=0.95, color=['red','green'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f1c98544880>
```



```
mc_churn_no = df1[df1.Churn=='No'].MonthlyCharges
mc_churn_yes = df1[df1.Churn=='Yes'].MonthlyCharges

plt.xlabel("Monthly Charges")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")

plt.hist([mc_churn_yes, mc_churn_no], rwidth=0.95, color=['red','green'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f1c997ff370>
```



```
def print_unique_col_values(df):
       for column in df:
            if df[column].dtypes=='object':
                print(f'{column}: {df[column].unique()}')
```

```
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No phone service' 'No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['Yes' 'No' 'No internet service']
DeviceProtection: ['No' 'Yes' 'No internet service']
TechSupport: ['No' 'Yes' 'No internet service']
StreamingTV: ['No' 'Yes' 'No internet service']
StreamingMovies: ['No' 'Yes' 'No internet service']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

```
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
```

```
<ipython-input-150-104b877f3854>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  df1.replace('No internet service','No',inplace=True)
<ipython-input-150-104b877f3854>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  df1.replace('No phone service','No',inplace=True)
```

```
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes']
OnlineBackup: ['Yes' 'No']
DeviceProtection: ['No' 'Yes']
TechSupport: ['No' 'Yes']
StreamingTV: ['No' 'Yes']
StreamingMovies: ['No' 'Yes']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','OnlineBackup',
                  'DeviceProtection','TechSupport','StreamingTV','StreamingMovies','PaperlessBilling','Churn']
for col in yes_no_columns:
    df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
<ipython-input-152-34dfac0bf179>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

```
gender: ['Female' 'Male']
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService: [0 1]
```

```
MultipleLines: [0 1]
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: [1 0]
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges: [  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
Churn: [0 1]
```

```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
<ipython-input-154-ba153b6b6960>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
df1.gender.unique()
```

```
array([1, 0])
```

One Hot Encoding

```
df2 = pd.get_dummies(data=df1, columns=['InternetService','Contract','PaymentMethod'])
df2.columns
```

```
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')
```

```
df2.sample(5)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity
4308	0	0	0	0	25	1	0	0
2540	1	0	0	0	70	1	1	1
1231	0	0	0	0	20	1	0	1
3582	1	0	0	0	3	1	0	0
3626	0	0	0	1	23	1	0	0

5 rows × 27 columns

```
df2.dtypes
```

```
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
```

Scaling some columns

```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

```
for col in df2:
    print(f'{col}: {df2[col].unique()}')
```

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [0.         0.46478873 0.01408451 0.61971831 0.09859155 0.29577465
 0.12676056 0.38028169 0.85915493 0.16901408 0.21126761 0.8028169
 0.67605634 0.33802817 0.95774648 0.71830986 0.98591549 0.28169014
 0.15492958 0.4084507  0.64788732 1.         0.22535211 0.36619718
 0.05633803 0.63380282 0.14084507 0.97183099 0.87323944 0.5915493
 0.1971831  0.83098592 0.23943662 0.91549296 0.11267606 0.02816901
 0.42253521 0.69014085 0.88732394 0.77464789 0.08450704 0.57746479
 0.47887324 0.66197183 0.3943662  0.90140845 0.52112676 0.94366197
 0.43661972 0.76056338 0.50704225 0.49295775 0.56338028 0.07042254
 0.04225352 0.45070423 0.92957746 0.30985915 0.78873239 0.84507042
 0.18309859 0.26760563 0.73239437 0.54929577 0.81690141 0.32394366
 0.6056338  0.25352113 0.74647887 0.70422535 0.35211268 0.53521127]
PhoneService: [0 1]
MultipleLines: [0 1]
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
PaperlessBilling: [1 0]
MonthlyCharges: [0.11542289 0.38507463 0.35422886 ... 0.44626866 0.25820896 0.60149254]
TotalCharges: [0.0012751  0.21586661 0.01031041 ... 0.03780868 0.03321025 0.78764136]
Churn: [0 1]
InternetService_DSL: [1 0]
InternetService_Fiber optic: [0 1]
InternetService_No: [0 1]
Contract_Month-to-month: [1 0]
Contract_One year: [0 1]
Contract_Two year: [0 1]
PaymentMethod_Bank transfer (automatic): [0 1]
PaymentMethod_Credit card (automatic): [0 1]
PaymentMethod_Electronic check: [1 0]
PaymentMethod_Mailed check: [0 1]
```

```
df2
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity
0	1	0	1	0	0.000000	0	0	0
1	0	0	0	0	0.464789	1	0	1
2	0	0	0	0	0.014085	1	0	1
3	0	0	0	0	0.619718	0	0	1
4	1	0	0	0	0.014085	1	0	0
...	...	...	...	...	...	...	...	...
7038	0	0	1	1	0.323944	1	1	1
7039	1	0	1	1	1.000000	1	1	0
7040	1	0	1	1	0.140845	0	0	1
7042	0	0	0	0	0.915493	1	0	1

Train and test split

```
X = df2.drop('Churn',axis='columns')
y = df2['Churn']

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)
```

```
X_train.shape
```

```
(5625, 26)
```

```
X_test.shape
```

```
(1407, 26)
```

```
X_train[:10]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity
5664	1	1	0	0	0.126761	1	0	0
101	1	0	1	1	0.000000	1	0	0
2621	0	0	1	0	0.985915	1	0	0
392	1	1	0	0	0.014085	1	0	0
1327	0	0	1	0	0.816901	1	1	0
3607	1	0	0	0	0.169014	1	0	1
2773	0	0	1	0	0.323944	0	0	0
1936	1	0	1	0	0.704225	1	0	1
5387	0	0	0	0	0.042254	0	0	0
4331	0	0	0	0	0.985915	1	1	0

10 rows × 26 columns

```
import tensorflow as tf
from tensorflow import keras


model = keras.Sequential([
    keras.layers.Dense(26, input_shape=(26,), activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# opt = keras.optimizers.Adam(learning_rate=0.01)

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

model.fit(X_train, y_train, epochs=120)
```

```
176/176 [==============================] - 0s 2ms/step - loss: 0.3537 - accuracy: 0.8352
Epoch 93/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3537 - accuracy: 0.8315
Epoch 94/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3533 - accuracy: 0.8348
Epoch 95/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3520 - accuracy: 0.8366
Epoch 96/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3525 - accuracy: 0.8347
Epoch 97/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3516 - accuracy: 0.8316
Epoch 98/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3514 - accuracy: 0.8336
Epoch 99/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3516 - accuracy: 0.8359
Epoch 100/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3509 - accuracy: 0.8348
Epoch 101/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3511 - accuracy: 0.8313
Epoch 102/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3501 - accuracy: 0.8359
Epoch 103/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3500 - accuracy: 0.8322
Epoch 104/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3492 - accuracy: 0.8356
Epoch 105/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3481 - accuracy: 0.8395
Epoch 106/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3473 - accuracy: 0.8366
Epoch 107/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3474 - accuracy: 0.8359
Epoch 108/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3465 - accuracy: 0.8370
Epoch 109/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3457 - accuracy: 0.8400
Epoch 110/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3472 - accuracy: 0.8388
Epoch 111/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.8370
Epoch 112/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3466 - accuracy: 0.8368
Epoch 113/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3461 - accuracy: 0.8402
Epoch 114/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3447 - accuracy: 0.8368
Epoch 115/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3450 - accuracy: 0.8375
Epoch 116/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3456 - accuracy: 0.8373
Epoch 117/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3434 - accuracy: 0.8409
Epoch 118/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3436 - accuracy: 0.8389
Epoch 119/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3427 - accuracy: 0.8373
Epoch 120/120
176/176 [==============================] - 0s 2ms/step - loss: 0.3424 - accuracy: 0.8389
<keras.callbacks.History at 0x7f1c92c45bd0>
```

```
model.evaluate(X_test, y_test)
```

```
44/44 [==============================] - 0s 2ms/step - loss: 0.5073 - accuracy: 0.7676
[0.5072710514068604, 0.7675906419754028]
```

```
yp = model.predict(X_test)
yp[:5]
```

```
44/44 [==============================] - 0s 2ms/step
array([[0.44723594],
       [0.4067413 ],
       [0.01504415],
       [0.91444534],
       [0.22173353]], dtype=float32)
```

```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

```
y_pred[:10]
```

```
    [0, 0, 0, 1, 0, 1, 0, 0, 0, 0]
```

```
y_test[:10]
```

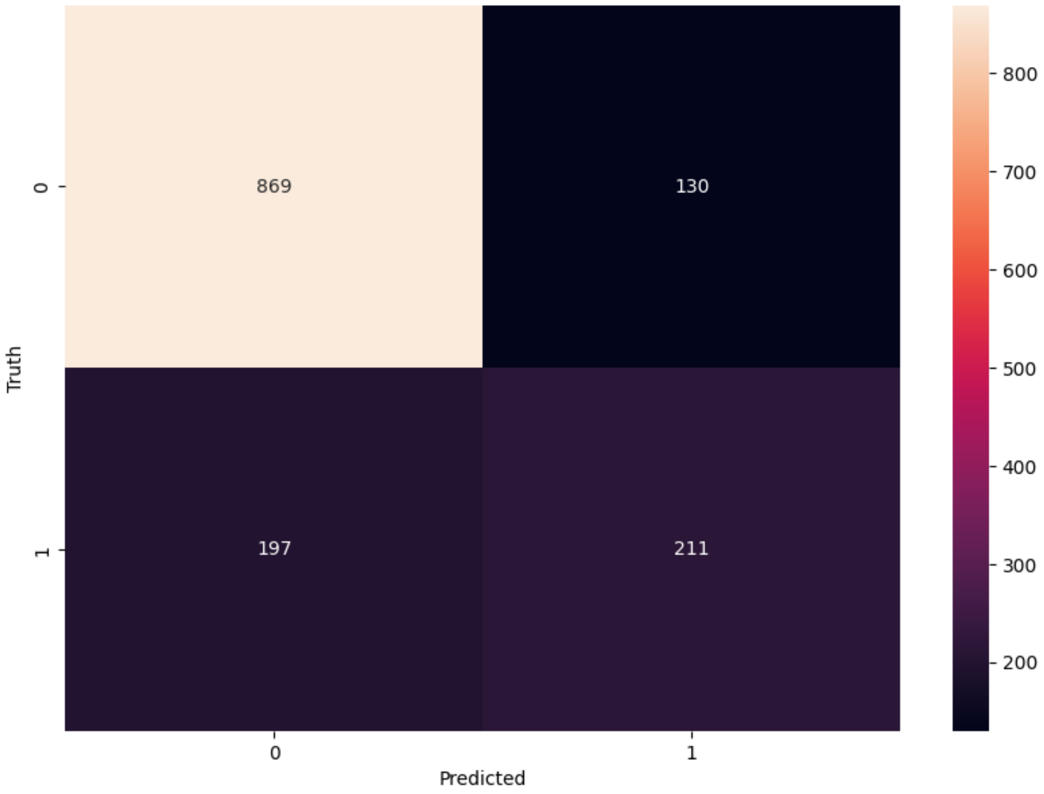
```
    2660    0
    744     0
    5579    1
    64      1
    3287    1
    816     1
    2670    0
    5920    0
    1023    0
    6087    0
    Name: Churn, dtype: int64
```

```
from sklearn.metrics import confusion_matrix , classification_report

print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.82      0.87      0.84       999
               1       0.62      0.52      0.56       408

        accuracy                           0.77      1407
       macro avg       0.72      0.69      0.70      1407
    weighted avg       0.76      0.77      0.76      1407
```

```
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)

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

```
    Text(95.72222222222221, 0.5, 'Truth')
```



```
y_test.shape
```

```
    (1407,)
```

```
#accuracy
round((862+229)/(862+229+137+179),2)
```

```
    0.78
```

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

```
round(862/(862+179),2)
```

```
    0.83
```

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

```
round(229/(229+137),2)
```

```
    0.63
```

Recall for 0 class

```
round(862/(862+137),2)
```

```
    0.86
```

```
round(229/(229+179),2)
```

```
    0.56
```

✓ 0s completed at 8:31 PM ● ×