CUSTOMER CHURN PREDICTION

Aim

The primary objective of this project is to develop a predictive model capable of identifying customers at risk of churning. By accurately predicting customer churn, businesses can implement targeted retention strategies, reduce customer attrition, and improve overall profitability.

Theory

Customer Churn refers to the loss of customers over a specified period. Predicting customer churn involves identifying patterns and trends in customer behavior that indicate an increased likelihood of churn. This information can be leveraged to implement timely interventions and retention strategies.

Machine Learning provides a powerful framework for building predictive models. In this context, we employ **classification algorithms** to categorize customers as either 'churning' or 'not churning'.

Key Concepts Implemented:

- **Data Preprocessing:** Involves handling missing values, outliers, and converting categorical data into numerical format suitable for machine learning algorithms. This step ensures data quality and consistency.
- **Feature Engineering:** Creating new features or transforming existing ones to improve model performance. For instance, creating features based on customer tenure, service usage patterns, or demographic information.
- **Model Selection:** Choosing an appropriate machine learning algorithm based on dataset characteristics and problem requirements. Common choices for churn prediction include Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting.
- **Model Training:** Feeding the preprocessed data to the chosen algorithm to learn patterns and relationships between features and the target variable (churn).
- **Model Evaluation:** Assessing the model's performance using metrics like accuracy, precision, recall, F1-score, and confusion matrix. This helps in understanding the model's strengths and weaknesses.

Algorithm (To be detailed based on specific implementation)

To predict customer churn, we use a machine learning model called Random Forest. First, we prepare the customer data by cleaning it and converting information into a format the model can understand. Then, we split the data into training and testing sets. The model learns patterns from the training data to predict whether a customer will churn or not. We evaluate the model's accuracy using metrics like precision, recall, and F1-score. Finally, we visualize the results using a confusion matrix to understand how well the model performs.

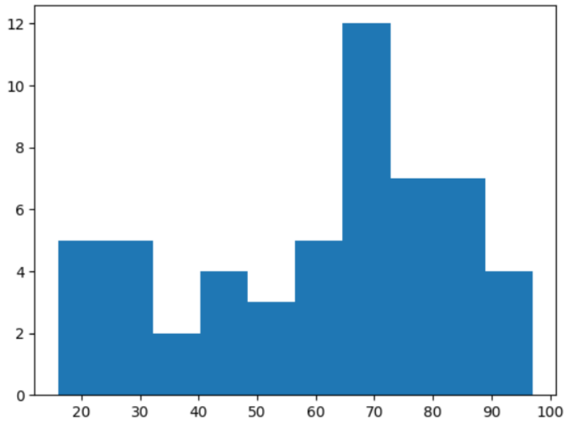
- **Data Preparation:** Describe the steps involved in data cleaning, preprocessing, and feature engineering.

- **Model Selection:** Explain the rationale for choosing the specific algorithm.
- **Model Training:** Detail the training process, including hyperparameter tuning and optimization techniques.
- **Model Evaluation:** Discuss the evaluation metrics used and their interpretation.

[1]:
```
print("hello")
```
```
hello
```

[2]:
```
import pandas as pd
```

[3]:
```
import matplotlib.pyplot as plt

# create data
data = [32, 96, 45, 67, 76, 28, 79, 62, 43, 81, 70,

        61, 95, 44, 60, 69, 71, 23, 69, 54, 76, 67,
        82, 97, 26, 34, 18, 16, 59, 88, 29, 30, 66,
        23, 65, 72, 20, 78, 49, 73, 62, 87, 37, 68,
        81, 80, 77, 92, 81, 52, 43, 68, 71, 86]

# create histogram
plt.hist(data)

# display histogram
plt.show()
```



[4]:
```
df = pd.read_csv(r"E:\Jupyter\Churn-Data.csv")
```

[5]:
```
df.head()
```

[5]:

	cID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	TV_Stre
0	4223-BKEOR	Female	0	No	Yes	21	Yes	No	DSL	Yes	...	Yes	No	
1	6035-RIIOM	Female	0	No	No	54	Yes	Yes	Fiber optic	No	...	No	No	
2	3797-VTIDR	Male	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	
3	2568-BRGYX	Male	0	No	No	4	Yes	No	Fiber optic	No	...	No	No	
4	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	...	No	Yes	

5 rows × 21 columns

[6]:
```
df.drop('cID' ,axis='columns',inplace=True)
```

[7]:
```
df.dtypes
```

[7]:
```
gender                object
SeniorCitizen          int64
Partner               object
Dependents            object
tenure                 int64
PhoneService          object
MultipleLines         object
InternetService       object
OnlineSecurity        object
OnlineBackup          object
DeviceProtection      object
TechSupport           object
TV_Streaming          object
Movie_Streaming       object
Contract              object
PaperlessBilling      object
Method_Payment        object
Charges_Month        float64
TotalCharges          object
Churn                 object
dtype: object
```

[8]:
```
df.TotalCharges.values
```

[8]:
```
array(['1336.8', '5129.45', '23.45', ..., '306.05', '1200.15', '457.3'],
      dtype=object)
```

[9]:
```
df.Charges_Month.values
```

[9]:
```
array([64.85, 97.2 , 23.45, ..., 21.15, 99.45, 19.8 ])
```

[10]:
```
pd.to_numeric(df.TotalCharges,errors='coerce').isnull()
```

[10]:
```
0       False
1       False
2       False
3       False
4        True
        ...
5629    False
5630    False
5631    False
5632    False
5633    False
Name: TotalCharges, Length: 5634, dtype: bool
```

[11]:
```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]
```

[11]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
4	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	Yes
282	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	Yes
2419	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service
2734	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
2903	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
3974	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	No
5023	Male	0	No	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
5030	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
5343	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	Yes
5599	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service

[12]:
```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()].shape
```

[12]: (10, 20)

[13]:
```
df.shape
```

[13]: (5634, 20)

[14]:
```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

[14]: (5624, 20)

[15]:
```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
C:\Users\Arka Singha\AppData\Local\Temp\ipykernel_10220\973151263.py: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-versus-a-copy
  df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

[16]:
```
df_copy = df1.copy()
df_copy['TotalCharges'] = pd.to_numeric(df_copy['TotalCharges'])
```

[17]:
```
df_copy.dtypes
```

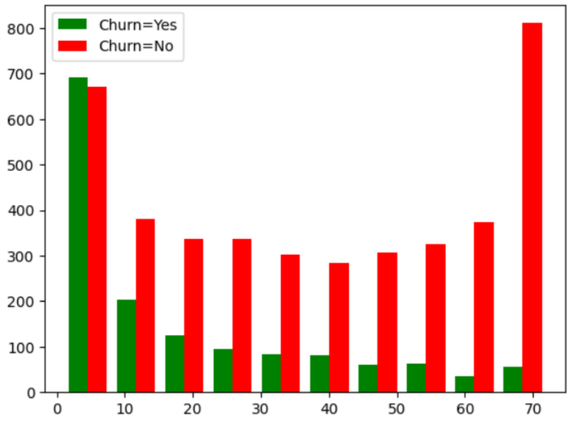
[17]:
```
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
TV_Streaming         object
Movie_Streaming      object
Contract             object
PaperlessBilling     object
Method_Payment       object
Charges_Month       float64
TotalCharges        float64
Churn                object
dtype: object
```

[18]:
```
df1.TotalCharges.dtypes
```

[18]: dtype('float64')

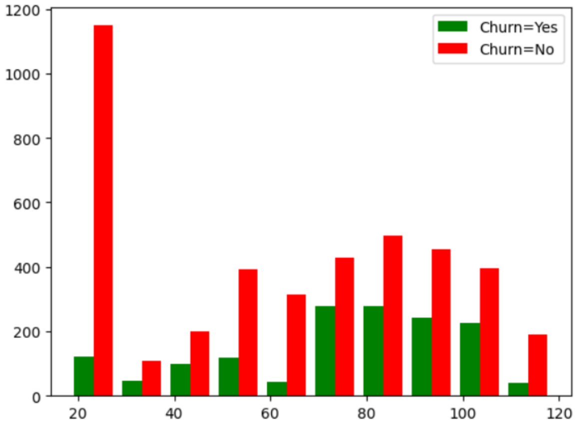
[19]:
```
tenure_churn_No = df1[df1.Churn=='No'].tenure
tenure_churn_Yes = df1[df1.Churn=='Yes'].tenure

plt.hist([tenure_churn_Yes, tenure_churn_No], color=['green','red'], label=['Churn=Yes', 'Churn=No'])
plt.legend()
```

[19]: <matplotlib.legend.Legend at 0x27e687a7590>



[20]:
```python
 mc_churn_No = df1[df1.Churn=='No'].Charges_Month
mc_churn_Yes = df1[df1.Churn=='Yes'].Charges_Month

plt.hist([mc_churn_Yes, mc_churn_No], color=['green','red'], label=['Churn=Yes', 'Churn=No'])
plt.legend()
```

[20]: <matplotlib.legend.Legend at 0x27e6aa06870>



[21]:
```python
def print_unique_col_values(df):
    for column in df:
        if df[column].dtypes=='object':
            print(f'{column} : {df[column].unique()}')
```

[22]:
```python
print_unique_col_values(df1)
```

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

[23]:
```python
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
```

```
C:\Users\Arka Singha\AppData\Local\Temp\ipykernel_10220\2045096646.py: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-versus-a-copy
  df1.replace('No internet service','No',inplace=True)
C:\Users\Arka Singha\AppData\Local\Temp\ipykernel_10220\2045096646.py: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-versus-a-copy
  df1.replace('No phone service','No',inplace=True)
```

[24]:
```
print_unique_col_values(df1)
```

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

[25]:
```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines',
                  'OnlineSecurity','OnlineBackup','DeviceProtection','TechSupport',
                  'TV_Streaming','Movie_Streaming','PaperlessBilling','Churn']

for col in yes_no_columns:
    df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

[26]:
```
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

```
gender: ['Female' 'Male']
SeniorCitizen: [0 1]
Partner: [0 1]
Dependents: [1 0]
tenure: [21 54  1  4  7 32 72 19 10 45 40 47 36 69 71 35  3 68 42  8 46 12 26 49
 33 31 66 58 13 57  6 59 15 27 34 18  5 39 29  2 63 20 14 56 37 24 52 43
 11 16 50 38 23 55 48 53 70 22 28 44 65 64 60 51  9 25 61 30 17 41 67 62]
PhoneService: [1 0]
MultipleLines: [0 1]
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: [1 0]
OnlineBackup: [0 1]
DeviceProtection: [1 0]
TechSupport: [0 1]
TV_Streaming: [0 1]
Movie_Streaming: [1 0]
Contract: ['One year' 'Two year' 'Month-to-month']
PaperlessBilling: [0 1]
Method_Payment: ['Mailed check' 'Bank transfer (automatic)' 'Electronic check'
 'Credit card (automatic)']
Charges_Month: [64.85 97.2  23.45 ... 59.25 35.35 21.15]
TotalCharges: [1336.8  5129.45   23.45 ...  306.05 1200.15  457.3 ]
Churn: [0 1]
```

[27]:
```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

[28]:
```
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [0 1]
Dependents: [1 0]
tenure: [21 54  1  4  7 32 72 19 10 45 40 47 36 69 71 35  3 68 42  8 46 12 26 49
 33 31 66 58 13 57  6 59 15 27 34 18  5 39 29  2 63 20 14 56 37 24 52 43
 11 16 50 38 23 55 48 53 70 22 28 44 65 64 60 51  9 25 61 30 17 41 67 62]
PhoneService: [1 0]
MultipleLines: [0 1]
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: [1 0]
OnlineBackup: [0 1]
DeviceProtection: [1 0]
TechSupport: [0 1]
TV_Streaming: [0 1]
Movie_Streaming: [1 0]
Contract: ['One year' 'Two year' 'Month-to-month']
PaperlessBilling: [0 1]
Method_Payment: ['Mailed check' 'Bank transfer (automatic)' 'Electronic check'
 'Credit card (automatic)']
Charges_Month: [64.85 97.2  23.45 ... 59.25 35.35 21.15]
TotalCharges: [1336.8  5129.45   23.45 ...  306.05 1200.15  457.3 ]
Churn: [0 1]
```

[29]:
```
df2 = pd.get_dummies(data=df1,columns=['InternetService','Contract','Method_Payment'])
df2.columns
```

[29]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'TV_Streaming', 'Movie_Streaming',
       'PaperlessBilling', 'Charges_Month', 'TotalCharges', 'Churn',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'Method_Payment_Bank transfer (automatic)',
       'Method_Payment_Credit card (automatic)',
       'Method_Payment_Electronic check', 'Method_Payment_Mailed check'],
      dtype='object')
```

[30]:
```
df2.sample(4)
```

[30]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceProtection	...	InternetService_DSL	Intern
5092	1	0	1	1	10	1	0	0	1	1	...	True	
2172	1	0	0	1	72	1	0	0	0	0	...	False	
5214	1	0	1	1	48	1	1	1	1	0	...	False	
2912	1	0	0	1	39	0	0	0	0	1	...	True	

4 rows × 27 columns

[31]:
```
df2.dtypes
```

[31]:
```
gender                                      int64
SeniorCitizen                               int64
Partner                                     int64
Dependents                                  int64
tenure                                      int64
PhoneService                                int64
MultipleLines                               int64
OnlineSecurity                              int64
OnlineBackup                                int64
DeviceProtection                            int64
TechSupport                                 int64
TV_Streaming                                int64
Movie_Streaming                             int64
PaperlessBilling                            int64
Charges_Month                             float64
TotalCharges                              float64
Churn                                       int64
InternetService_DSL                          bool
InternetService_Fiber optic                  bool
InternetService_No                           bool
Contract_Month-to-month                      bool
Contract_One year                            bool
Contract_Two year                            bool
Method_Payment_Bank transfer (automatic)     bool
Method_Payment_Credit card (automatic)       bool
Method_Payment_Electronic check              bool
Method_Payment_Mailed check                  bool
dtype: object
```

[32]:
```
cols_to_scale = ['tenure','Charges_Month','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

[33]:
```
df2.sample(3)
```

[33]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceProtection	...	InternetService_DSL	Inte
3088	1	0	1	0	0.323944	1	1	0	0	1	...	False	
4882	0	0	0	0	0.323944	1	1	0	1	1	...	False	
3695	1	0	1	0	0.971831	1	1	0	1	1	...	False	

3 rows × 27 columns

[34]:
```
x = df2.drop('Churn',axis='columns')
y = df2['Churn']
```

[35]:
```
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)
```

[36]:
```
x_train.shape
```

[36]:
```
(4499, 26)
```

[37]:
```python
x_test.shape
```

[37]: (1125, 26)

[38]:
```python
x_train[:10]
```

[38]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceProtection	...	InternetService_DSL	Inte
1707	1	0	0	0	0.521127	1	1	0	0	0	...	False	
1373	1	0	0	0	0.464789	1	1	1	1	0	...	True	
1311	1	0	1	1	0.225352	1	0	1	1	0	...	True	
151	1	0	1	1	0.760563	1	0	0	0	0	...	False	
2456	1	0	1	1	0.267606	1	1	0	0	1	...	False	
3998	1	0	0	0	0.760563	1	0	0	1	1	...	True	
4968	1	0	0	0	0.126761	1	0	0	0	0	...	False	
5512	0	0	0	0	0.957746	1	1	0	1	0	...	False	
3549	0	0	0	1	0.309859	1	0	0	0	0	...	False	
4739	0	0	0	0	0.492958	1	1	0	0	0	...	False	

10 rows × 26 columns

[39]:
```python
len(x_train.columns)
```

[39]: 26

[40]:
```python
import tensorflow as tf
from tensorflow import keras
```

[41]:
```python
model = keras.Sequential([
    keras.layers.Dense(20, input_shape=(26,),activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid'),
])
```

```
C:\Users\Arka Singha\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shap
e`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[42]:
```python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[43]:
```python
model.fit(x_train, y_train, epochs=100)
```

```
Epoch 1/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.5486 - loss: 0.6944
Epoch 2/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7776 - loss: 0.4672
Epoch 3/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7848 - loss: 0.4338
Epoch 4/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7954 - loss: 0.4256
Epoch 5/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8044 - loss: 0.4189
Epoch 6/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8010 - loss: 0.4134
Epoch 7/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7872 - loss: 0.4346
Epoch 8/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8052 - loss: 0.4180
Epoch 9/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8064 - loss: 0.4143
Epoch 10/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8151 - loss: 0.4024
Epoch 11/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8062 - loss: 0.4195
Epoch 12/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8037 - loss: 0.4058
Epoch 13/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8016 - loss: 0.4181
Epoch 14/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8093 - loss: 0.4120
Epoch 15/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8138 - loss: 0.4007
Epoch 16/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7972 - loss: 0.4171
Epoch 17/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8135 - loss: 0.4040
```

```
Epoch 18/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8048 - loss: 0.4153
Epoch 19/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8150 - loss: 0.4042
Epoch 20/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8149 - loss: 0.4007
Epoch 21/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8115 - loss: 0.4044
Epoch 22/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8185 - loss: 0.3948
Epoch 23/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8184 - loss: 0.4028
Epoch 24/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8216 - loss: 0.3927
Epoch 25/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8237 - loss: 0.3927
Epoch 26/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8140 - loss: 0.3938
Epoch 27/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8102 - loss: 0.3949
Epoch 28/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8213 - loss: 0.3913
Epoch 29/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8247 - loss: 0.3905
Epoch 30/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8155 - loss: 0.3865
Epoch 31/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8193 - loss: 0.3866
Epoch 32/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8146 - loss: 0.3956
Epoch 33/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8200 - loss: 0.3860
Epoch 34/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8091 - loss: 0.3895
Epoch 35/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8200 - loss: 0.3795
```

```
Epoch 36/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8200 - loss: 0.3873
Epoch 37/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8172 - loss: 0.3970
Epoch 38/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8240 - loss: 0.3721
Epoch 39/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8207 - loss: 0.3723
Epoch 40/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8240 - loss: 0.3772
Epoch 41/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8109 - loss: 0.3936
Epoch 42/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8238 - loss: 0.3826
Epoch 43/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8190 - loss: 0.3843
Epoch 44/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8237 - loss: 0.3783
Epoch 45/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8231 - loss: 0.3806
Epoch 46/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8254 - loss: 0.3734
Epoch 47/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8240 - loss: 0.3836
Epoch 48/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8172 - loss: 0.3851
Epoch 49/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8216 - loss: 0.3775
Epoch 50/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8135 - loss: 0.3853
Epoch 51/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8208 - loss: 0.3788
Epoch 52/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8212 - loss: 0.3780
Epoch 53/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8189 - loss: 0.3787
```

```
Epoch 54/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8175 - loss: 0.3780
Epoch 55/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8231 - loss: 0.3795
Epoch 56/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8279 - loss: 0.3798
Epoch 57/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8270 - loss: 0.3721
Epoch 58/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8106 - loss: 0.4024
Epoch 59/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8161 - loss: 0.3758
Epoch 60/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8213 - loss: 0.3766
Epoch 61/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8169 - loss: 0.3866
Epoch 62/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8243 - loss: 0.3675
Epoch 63/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8278 - loss: 0.3677
Epoch 64/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8272 - loss: 0.3701
Epoch 65/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8205 - loss: 0.3817
Epoch 66/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8219 - loss: 0.3732
Epoch 67/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8253 - loss: 0.3719
Epoch 68/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8319 - loss: 0.3711
Epoch 69/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8374 - loss: 0.3647
Epoch 70/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8307 - loss: 0.3678
Epoch 71/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8354 - loss: 0.3526
```

```
Epoch 72/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8388 - loss: 0.3540
Epoch 73/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8237 - loss: 0.3685
Epoch 74/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8317 - loss: 0.3548
Epoch 75/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8307 - loss: 0.3647
Epoch 76/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8237 - loss: 0.3633
Epoch 77/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8252 - loss: 0.3728
Epoch 78/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8257 - loss: 0.3614
Epoch 79/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8334 - loss: 0.3576
Epoch 80/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8248 - loss: 0.3587
Epoch 81/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8260 - loss: 0.3746
Epoch 82/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8339 - loss: 0.3589
Epoch 83/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8220 - loss: 0.3672
Epoch 84/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8338 - loss: 0.3554
Epoch 85/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8265 - loss: 0.3690
Epoch 86/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8314 - loss: 0.3570
Epoch 87/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8249 - loss: 0.3644
Epoch 88/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8235 - loss: 0.3627
Epoch 89/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8371 - loss: 0.3626
```

```
Epoch 90/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8394 - loss: 0.3589
Epoch 91/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8371 - loss: 0.3504
Epoch 92/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8336 - loss: 0.3572
Epoch 93/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8294 - loss: 0.3602
Epoch 94/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8268 - loss: 0.3694
Epoch 95/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8287 - loss: 0.3680
Epoch 96/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8317 - loss: 0.3535
Epoch 97/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8332 - loss: 0.3584
Epoch 98/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8292 - loss: 0.3618
Epoch 99/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8274 - loss: 0.3546
Epoch 100/100
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8313 - loss: 0.3636
```

[43]:
```
<keras.src.callbacks.history.History at 0x27e05753710>
```

[44]:
```
model.evaluate(x_test, y_test)
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7891 - loss: 0.4513
```

[44]:
```
[0.4954747259616852, 0.7671111226081848]
```

[45]:
```
yp = model.predict(x_test)
yp[:5]
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

[45]:
```
array([[0.36580497],
       [0.6883788 ],
       [0.08368747],
       [0.6980473 ],
       [0.07836749]], dtype=float32)
```

[46]:
```
y_test[:10]
```

[46]:
```
5128    0
2438    1
1388    0
3415    1
1112    0
3535    0
1805    0
2258    0
1771    0
1309    0
Name: Churn, dtype: int64
```

[47]:
```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

[48]:
```
y_pred[:10]
```

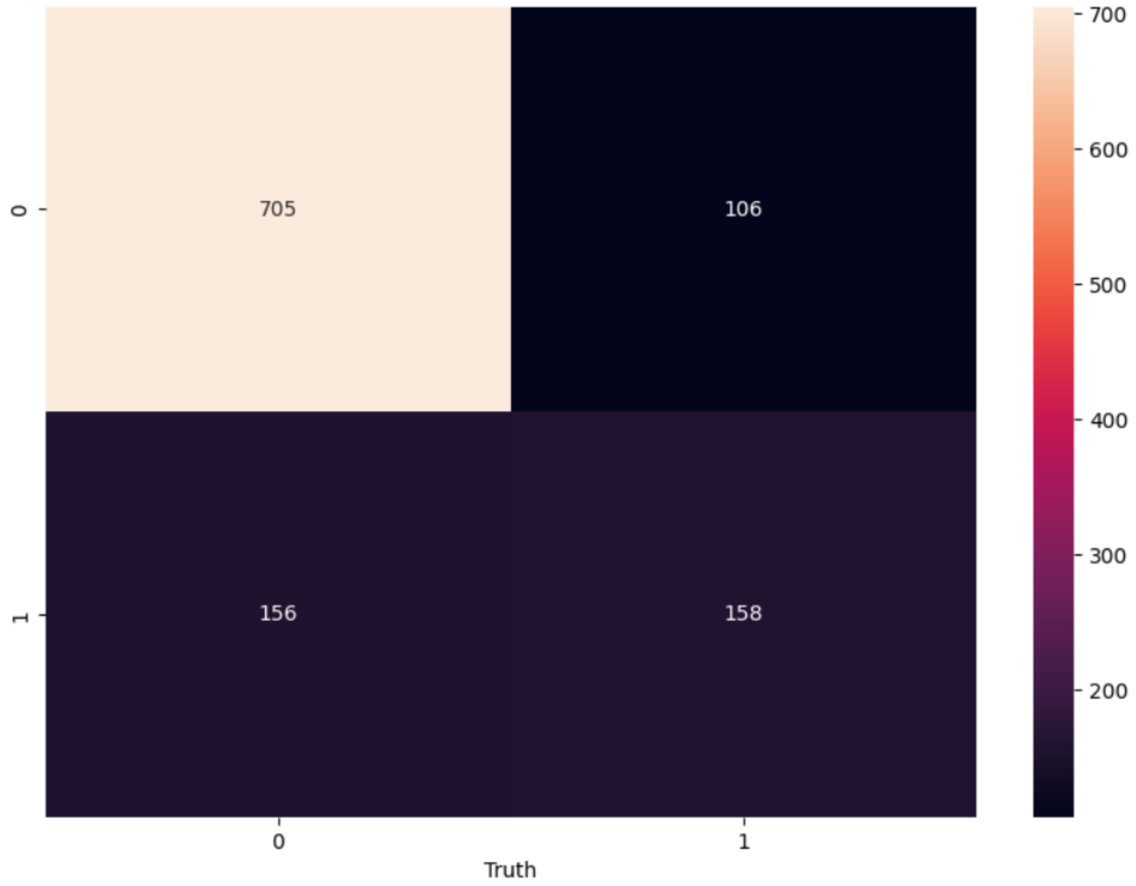
[48]:
```
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0]
```

[49]:
```
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       811
           1       0.60      0.50      0.55       314

    accuracy                           0.77      1125
   macro avg       0.71      0.69      0.70      1125
weighted avg       0.76      0.77      0.76      1125
```

[50]:
```
import seaborn as sn
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.xlabel('Truth')
```

[50]:
```
Text(0.5, 47.7222222222222, 'Truth')
```



[61]:
```
round((705+158)/(705+106+156+158),2)
```

[61]:
```
0.77
```

Conclusion

The developed customer churn prediction model effectively identifies customers at risk of churning. By leveraging machine learning techniques and incorporating relevant customer data, the model provides valuable insights into customer behaviour. The model's performance demonstrates its ability to accurately predict churn.