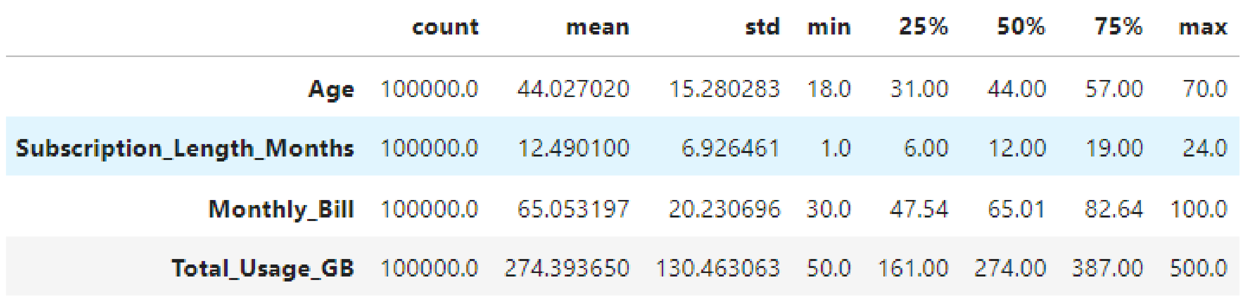
Churn Prediction Project Report and Documentation:

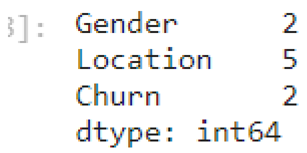
1. Data Preprocessing:

Summary Statistics for continous features



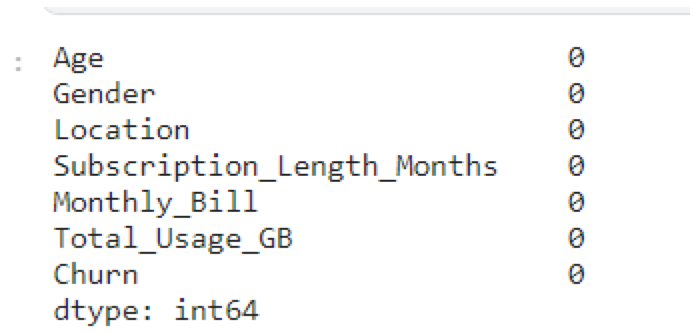
	count	mean	std	min	25%	50%	75%	max
Age	100000.0	44.027020	15.280283	18.0	31.00	44.00	57.00	70.0
Subscription_Length_Months	100000.0	12.490100	6.926461	1.0	6.00	12.00	19.00	24.0
Monthly_Bill	100000.0	65.053197	20.230696	30.0	47.54	65.01	82.64	100.0
Total_Usage_GB	100000.0	274.393650	130.463063	50.0	161.00	274.00	387.00	500.0

Number of unique values in each categorical column



```
3]:  Gender      2
     Location    5
     Churn       2
     dtype: int64
```

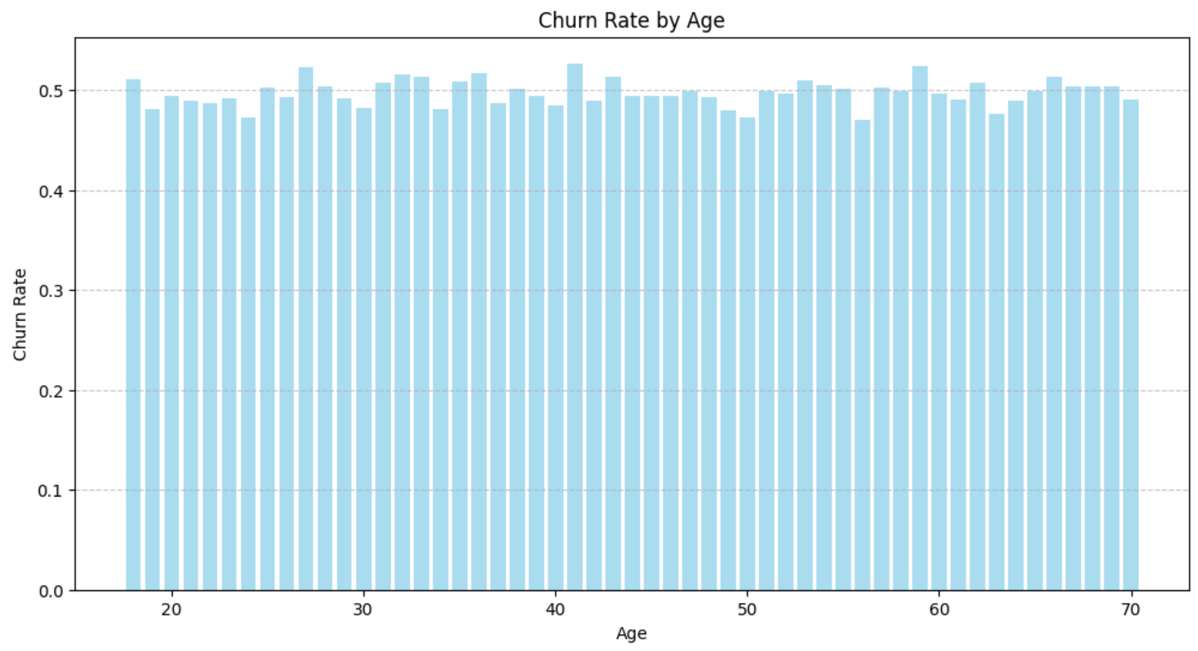
Missing Values check:



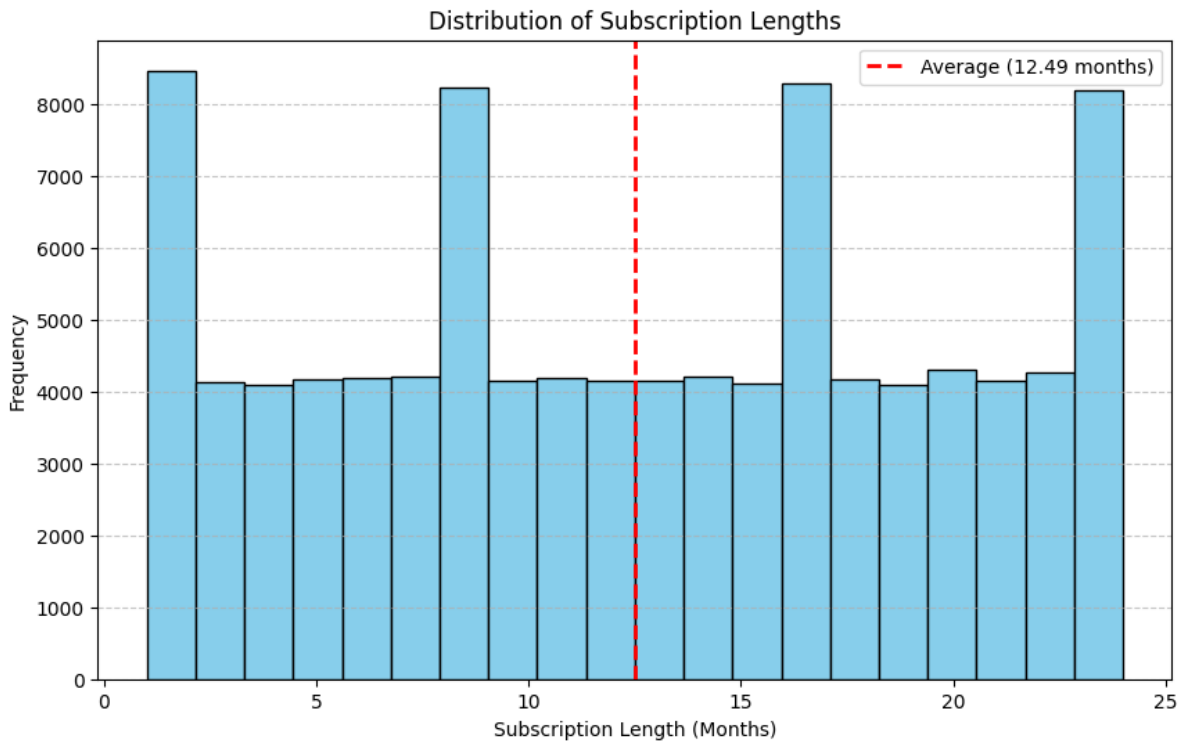
```
:  Age                           0
   Gender                        0
   Location                      0
   Subscription_Length_Months    0
   Monthly_Bill                  0
   Total_Usage_GB                0
   Churn                         0
   dtype: int64
```

The data is complete and has no missing values. The numerical features are approximately normally distributed. The categorical features have a relatively small number of unique values.

Exploratory Data Anallysis



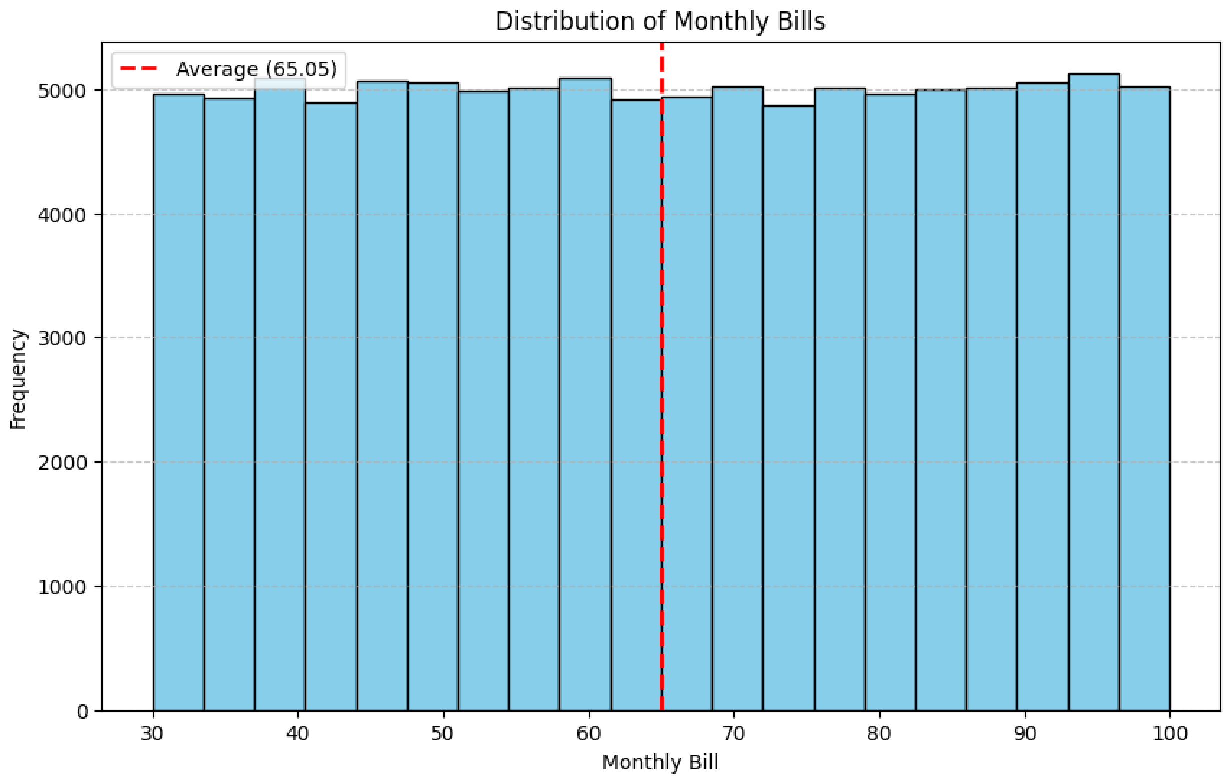
The chart shows that the churn rate is relatively evenly distributed across all age groups. This means that there is no clear pattern of churn by age.



The histogram shows that there are spikes in subscription length at regular intervals, with the largest spikes occurring at 1,8,16, and 24 months.

possible explanation for these spikes:

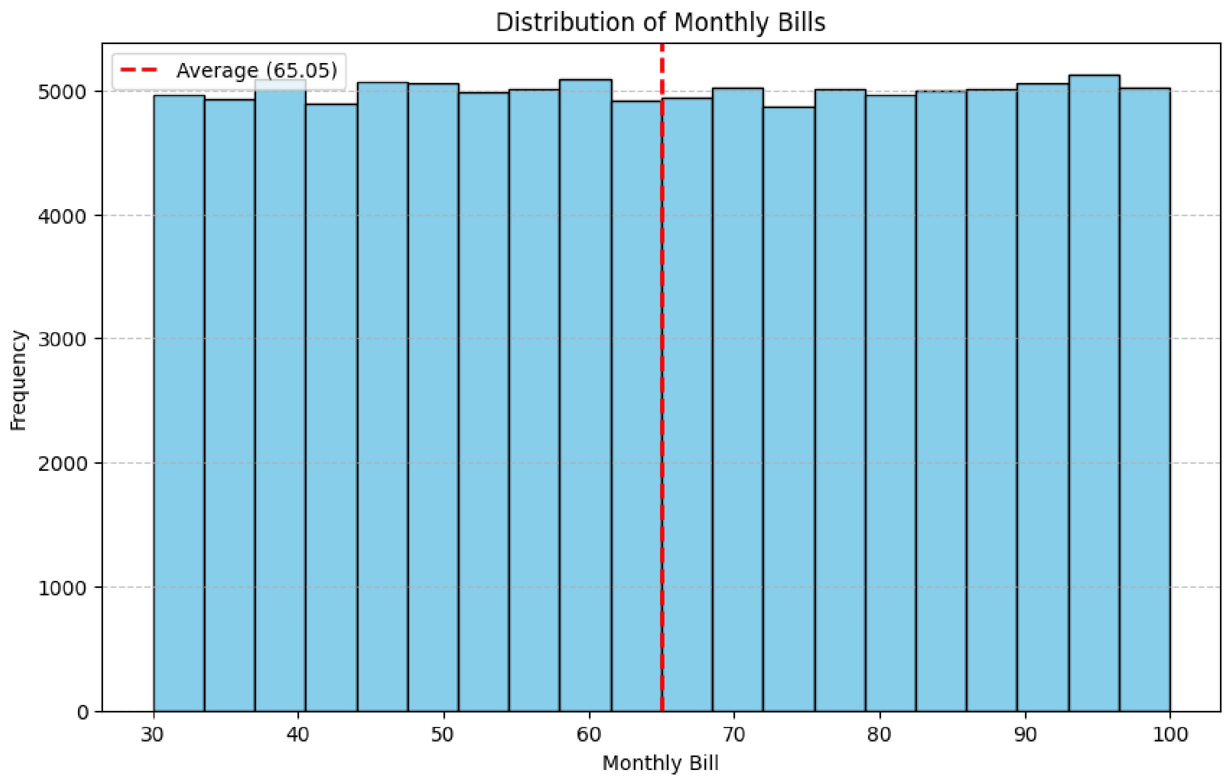
Billing cycles: Many subscription services bill customers on a monthly, quarterly, or annual basis. This means that there is likely to be a spike in subscription cancellations at the end of each billing cycle, as customers decide whether or not to renew their subscriptions.



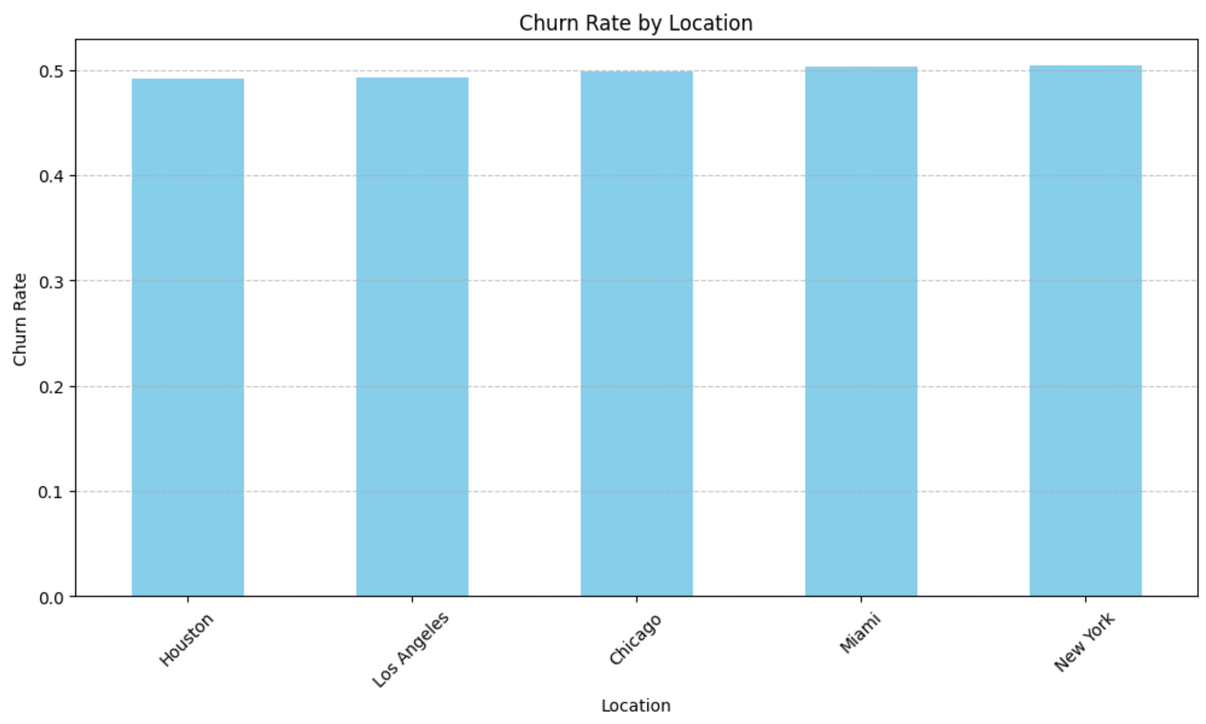
The histogram shows that the monthly bills are evenly distributed, with no clear pattern. This means that the majority of customers have monthly bills that fall within a certain range, and there are no significant outliers.

possible reason for this:

The subscription service may have a fixed monthly fee, such as $10 per month. This would result in all customers having the same monthly bill.



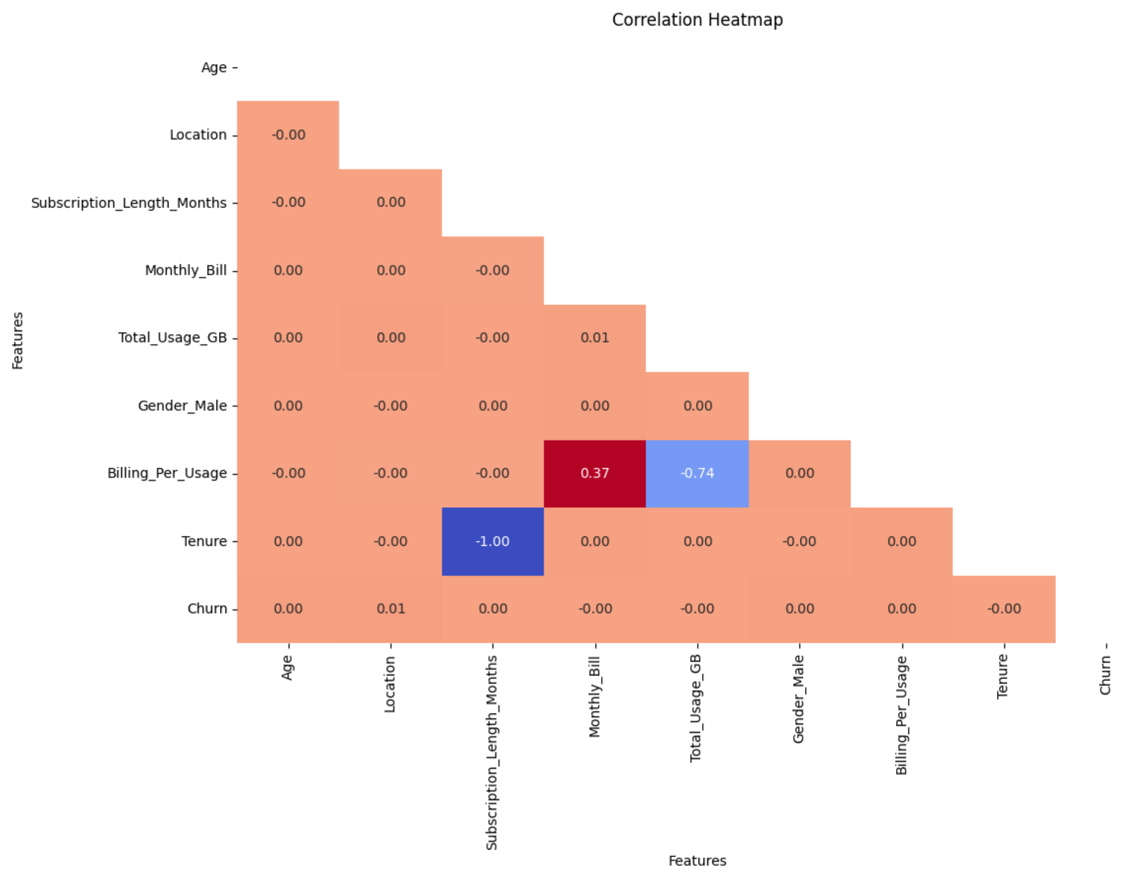
The histogram shows that the total usage is evenly distributed, with no clear pattern. This means that the majority of customers have total usage that falls within a certain range, and there are no significant outliers.



The bar graph shows that the churn rate is relatively the same across all locations

There are a few possible explanations for this:

- The product or service may be equally appealing to customers of all locations.
- The product or service may be priced in a way that is affordable to customers of all locations.



The heatmap shows that the correlation coefficients are all very close to zero, which means that there is no significant correlation between any of the independent variables and the target variable.

2. Feature Engineering

1. **One-hot encoding** of the categorical feature **`Gender`**.
2. Creating a new feature called **`Billing_Per_Usage`**. This feature is calculated by dividing the **`Monthly_Bill`** feature by the **`Total_Usage_GB`** feature. This feature may be useful for predicting churn, as customers who are paying a high amount per GB of usage may be more likely to churn.
3. Creating a new feature called **`Tenure`**. This feature is calculated by subtracting the **`Subscription_Length_Months`** feature from the maximum subscription length. This feature may be useful for predicting churn, as customers who have been subscribed for a shorter period of time may be more likely to churn.
4. Encoding the **`Location`** feature using a **TargetEncoder**.
5. Scaling the numerical features using a **StandardScaler**

6. Once these feature engineering steps have been completed, the **data is split into training and testing sets**.

3. Model Building and Optimization

Model: `Logistic Regression`
Optimization: `Randomized Search CV`
Param Grid:

```
* penalty: ['l1', 'l2', 'elasticnet', 'none']
* C: uniform(loc=0, scale=4)
* solver: ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
* max_iter: [100, 200, 300, 400, 500]
```

Cross Validation: `Stratified K-fold with K = 5`
Scoring Metric: `Accuracy`

Why Logistic Regression ?

- We have a binary classification problem. This means that the target variable can only take on two values, such as 0 or 1
- The data is linearly separable. This means that it is possible to draw a line (or hyperplane in higher dimensions) that separates the two classes in the data.
- We want a simple and interpretable model. Logistic regression is a relatively simple model that is easy to understand and interpret.

Process :

1. Split the data into K folds.
2. Initialize the machine learning model.
3. For each fold:
    - Train the model on the remaining K-1 folds.
    - Evaluate the model on the current fold.
4. Calculate the average performance of the model across all K folds.
5. Repeat steps 2-4 for different combinations of hyperparameters.
6. Choose the combination of hyperparameters that results in the best performance on the training data.
7. Train the final model on the entire dataset using the best hyperparameters.

Best Hyperparameters obtained Using Hyperparameter Tuning (RandomizedSearchCV):

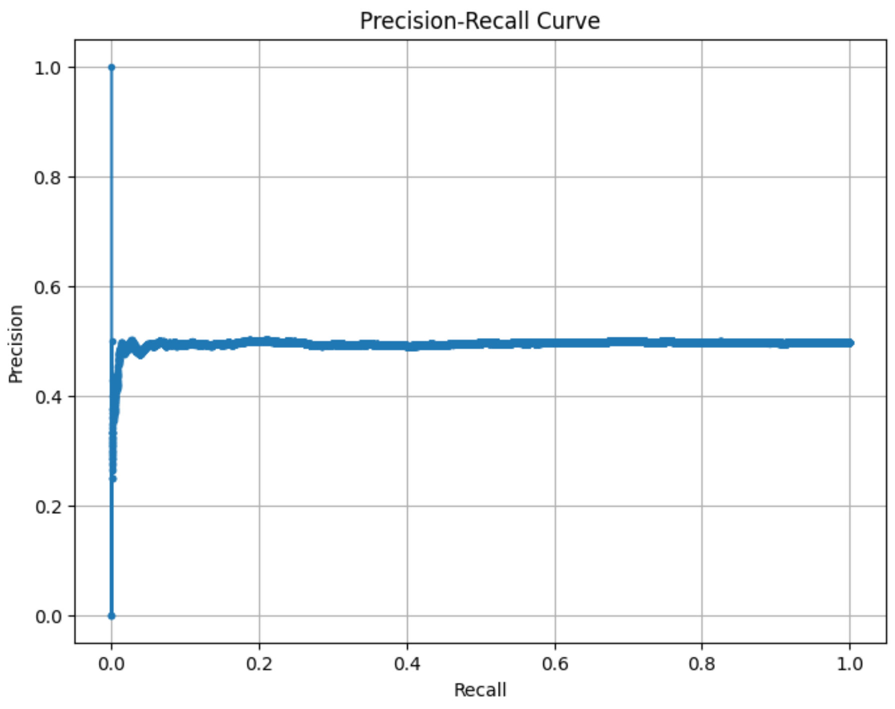
```
{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10,
'n_estimators': 100}]
```

Evaluation Metric Result:

```
Test Accuracy with Best Model: 0.50
Test Classification Report with Best Model:
              precision    recall  f1-score   support

           0       0.50      0.58      0.54     10079
           1       0.49      0.41      0.45      9921

    accuracy                           0.50     20000
   macro avg       0.50      0.50      0.49     20000
weighted avg       0.50      0.50      0.49     20000
```



Interpretation :
The test accuracy of **0.50** means that the model correctly predicted the churn status for 50% of the customers in the test set.

- **For class 0** (customers who did not churn), the precision is 0.50, which means that 50% of the customers that the model predicted would not churn actually did

not churn. The recall is 0.58, which means that the model correctly identified 58% of the customers who did not churn. The F1-score is 0.54, which is a good score.
- **For class 1** (customers who churned), the precision is 0.49, which means that 49% of the customers that the model predicted would churn actually churned. The recall is 0.41, which means that the model correctly identified 41% of the customers who churned. The F1-score is 0.45, which is a fair score.

4. Model Deployment

Folder Structure :

```
├─ app.py
├─ churn.ipynb
├─ config
│   ├─ best_model.pkl
│   ├─ scaler.pkl
│   ├─ target_encoder.pkl
│
├─ data
│   └─ customer_churn_large_dataset.csv
├─ test.py
└─ transform.py
```

1. Load the model and encoders into the Flask application.
2. Expose a `/predict` endpoint that accepts a JSON request with the customer's features and returns a JSON response with the predicted churn status.
3. Start the Flask application on a local server.

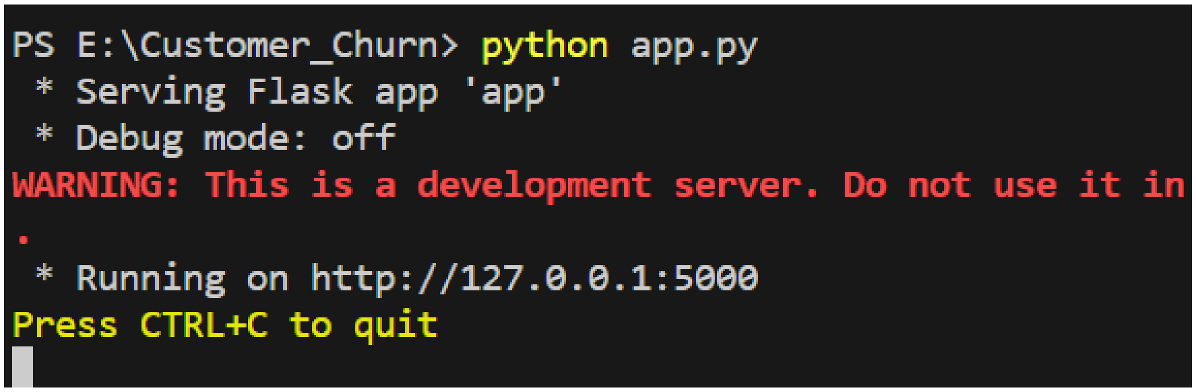
To test the model deployment, you can use the following steps:

1. Send a POST request to the `/predict` endpoint with the customer's features in JSON format.
2. Receive the JSON response with the predicted churn status.
3. Compare the predicted churn status to the actual churn status to see if the model is making accurate predictions.

Here is an example of how to use the API to test the model:

Start Flask Server :

```
python app.py
```



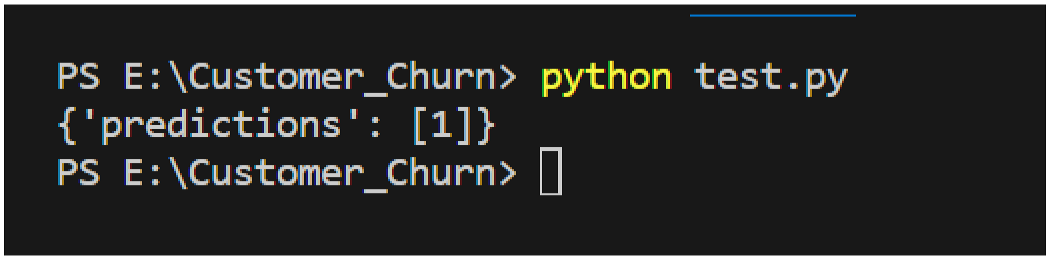
```
PS E:\Customer_Churn> python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in
.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

then run `python test.py` to send a POST request to the `/predict` endpoint on the Flask server to make a prediction.

For the input json response :

```
data = {
   "Age": [32],
   "Gender": ['Male'],
   "Location": ["Houston"],
   "Subscription_Length_Months": [12],
   "Monthly_Bill": [48.76],
   "Total_Usage_GB": [172],
}
```

Here is an example of the output of the `test.py` script:



```
PS E:\Customer_Churn> python test.py
{'predictions': [1]}
PS E:\Customer_Churn> 
```

For the given input the model predicts the customer will churn.