Data Pre-Processing

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
```

In [2]:
```
# Load the dataset
df = pd.read_csv(r"C:\Users\rpsie\Downloads\Sunbase Assignment\customer_churn_large_dataset.csv")
df
```

Out[2]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	1	Customer_1	63	Male	Los Angeles	17	73.36	236	0
1	2	Customer_2	62	Female	New York	1	48.76	172	0
2	3	Customer_3	24	Female	Los Angeles	5	85.47	460	0
3	4	Customer_4	36	Female	Miami	3	97.94	297	1
4	5	Customer_5	46	Female	Miami	19	58.14	266	0
...	...	...	...	...	...	...	...	...	...
99995	99996	Customer_99996	33	Male	Houston	23	55.13	226	1
99996	99997	Customer_99997	62	Female	New York	19	61.65	351	0
99997	99998	Customer_99998	64	Male	Chicago	17	96.11	251	1
99998	99999	Customer_99999	51	Female	New York	20	49.25	434	1
99999	100000	Customer_100000	27	Female	Los Angeles	19	76.57	173	1

100000 rows × 9 columns

In [3]:
```
df.head(10)
```

Out[3]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	1	Customer_1	63	Male	Los Angeles	17	73.36	236	0
1	2	Customer_2	62	Female	New York	1	48.76	172	0
2	3	Customer_3	24	Female	Los Angeles	5	85.47	460	0
3	4	Customer_4	36	Female	Miami	3	97.94	297	1
4	5	Customer_5	46	Female	Miami	19	58.14	266	0
5	6	Customer_6	67	Male	New York	15	82.65	456	1
6	7	Customer_7	30	Female	Chicago	3	73.79	269	0
7	8	Customer_8	67	Female	Miami	1	97.70	396	1
8	9	Customer_9	20	Female	Miami	10	42.45	150	1
9	10	Customer_10	53	Female	Los Angeles	12	64.49	383	1

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

Out[4]:

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	50000.500000	44.027020	12.490100	65.053197	274.393650	0.497790
std	28867.657797	15.280283	6.926461	20.230696	130.463063	0.499998
min	1.000000	18.000000	1.000000	30.000000	50.000000	0.000000
25%	25000.750000	31.000000	6.000000	47.540000	161.000000	0.000000
50%	50000.500000	44.000000	12.000000	65.010000	274.000000	0.000000
75%	75000.250000	57.000000	19.000000	82.640000	387.000000	1.000000
max	100000.000000	70.000000	24.000000	100.000000	500.000000	1.000000

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
CustomerID                  0
Name                        0
Age                         0
Gender                      0
Location                    0
Subscription_Length_Months  0
Monthly_Bill                0
Total_Usage_GB              0
Churn                       0
dtype: int64
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   CustomerID                 100000 non-null  int64
 1   Name                       100000 non-null  object
 2   Age                        100000 non-null  int64
 3   Gender                     100000 non-null  object
 4   Location                   100000 non-null  object
 5   Subscription_Length_Months 100000 non-null  int64
 6   Monthly_Bill               100000 non-null  float64
 7   Total_Usage_GB             100000 non-null  int64
 8   Churn                      100000 non-null  int64
dtypes: float64(1), int64(5), object(3)
memory usage: 6.9+ MB
```

In [7]:
```
df['Gender'].value_counts()
```

Out[7]:
```
Female    50216
Male      49784
Name: Gender, dtype: int64
```

In [8]:
```
df['Churn'].value_counts()
```

Out[8]:
```
0    50221
1    49779
Name: Churn, dtype: int64
```

In [9]:
```
df['Location'].value_counts()
```

Out[9]:
```
Houston        20157
Los Angeles    20041
Miami          20031
Chicago        19958
New York       19813
Name: Location, dtype: int64
```

In [10]:
```
# Handle missing data
imputer = SimpleImputer(strategy='median')
df[['Age', 'Monthly_Bill', 'Total_Usage_GB']] = imputer.fit_transform(df[['Age', 'Monthly_Bill', 'Total_Usage_GB']])
```

In [12]:
```
# Split data into features (X) and target (y)
X = df.drop('Churn', axis=1)
y = df['Churn']
print(X.head())
print(y.head())
```

```
   CustomerID   Name   Age  Gender  Location  Subscription_Length_Months  \
0           1      0  63.0       1         2                          17
1           2  11112  62.0       0         4                           1
2           3  22223  24.0       0         2                           5
3           4  33334  36.0       0         3                           3
4           5  44445  46.0       0         3                          19

   Monthly_Bill  Total_Usage_GB
0         73.36           236.0
1         48.76           172.0
2         85.47           460.0
3         97.94           297.0
4         58.14           266.0
0    0
1    0
2    0
3    1
4    0
Name: Churn, dtype: int64
```

In [32]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,stratify=y)
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
70000
30000
70000
30000
```

Feature Engineering

In [56]:
```
# Encode categorical variables
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Location'] = label_encoder.fit_transform(df['Location'])
df['Name'] = label_encoder.fit_transform(df['Name'])
df.head()
```

Out[56]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	1	0	63.0	1	2	17	73.36	236.0	0
1	2	11112	62.0	0	4	1	48.76	172.0	0
2	3	22223	24.0	0	2	5	85.47	460.0	0
3	4	33334	36.0	0	3	3	97.94	297.0	1
4	5	44445	46.0	0	3	19	58.14	266.0	0

In [54]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print('X_train_scaled',X_train_scaled)
print('X_test_scaled',X_test_scaled)
```

```
X_train_scaled [[-1.0837931  -1.39563574  1.04732722 ...  0.94240111 -0.50951727
  -0.40679669]
 [ 1.6901857   1.5863186  -0.71903127 ...  1.08708096 -0.87571583
  -0.88959773]
 [-0.55955307 -0.81316303  0.06601695 ... -0.64907719 -0.21537394
   0.0222798 ]
 ...
 [-0.74989636 -1.02463468 -0.98071401 ...  0.36368173  0.02596177
   1.48573708]
 [-1.51902168  0.25869471 -0.45734853 ...  1.5211205  -0.26127215
  -0.00836852]
 [ 0.60105648  0.47625685  0.7202238  ...  0.94240111  0.92813291
   0.35941132]]
X_test_scaled [[ 1.28148182  1.23220482  1.63611339 ...  0.21900188  0.30233805
   0.73485324]
 [ 0.56932106  0.44102888 -0.26108648 ...  1.5211205  -0.13986399
  -1.36455669]
 [-1.38920755  1.70093063 -0.78445196 ... -0.3597175   0.7667489
   0.81147404]
 ...
 [-1.48617881  0.62360528 -0.98071401 ... -0.21503765  0.64484722
   1.70027532]
 [-0.77270295 -1.0500002  -0.26108648 ...  0.50836157  1.1995739
  -0.06200308]
 [-0.87386175 -1.1623629  -0.39192785 ... -0.21503765  0.34182038
  -1.61740533]]
```

Model Building and Evaluation

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Choose the model
model = RandomForestClassifier()

# Train the model
model.fit(X_train_scaled, y_train)

# Validate the test model
y_pred = model.predict(X_test_scaled)

# Evaluate the test model
test_accuracy = accuracy_score(y_test, y_pred)
test_report = classification_report(y_test, y_pred)

print(f'Accuracy: {test_accuracy}')
print(test_report)

# Validate the test model
y_pred = model.predict(X_train_scaled)

# Evaluate the test model
train_accuracy = accuracy_score(y_train, y_pred)
train_report = classification_report(y_train, y_pred)

print(f'Accuracy: {train_accuracy}')
print(train_report)
```

```
Accuracy: 0.49866666666666665
              precision    recall  f1-score   support

           0       0.50      0.53      0.52     15066
           1       0.50      0.46      0.48     14934

    accuracy                           0.50     30000
   macro avg       0.50      0.50      0.50     30000
weighted avg       0.50      0.50      0.50     30000

Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     35155
           1       1.00      1.00      1.00     34845

    accuracy                           1.00     70000
   macro avg       1.00      1.00      1.00     70000
weighted avg       1.00      1.00      1.00     70000
```

Model Deployement

In [75]:
```
input=X_test_scaled[1] # Provide index number of Test Dataset
array=np.array(input)
reshaped_array=array.reshape(1,-1)
prediction=model.predict(reshaped_array)
print(prediction)


if (prediction==0):
    print("Churned")
else:
    print("Not Churned")
```

```
[1]
Not Churned
```

In [ ]:
```

```