```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
import seaborn as sb
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

```python
df = pd.read_excel("/content/customer_churn_large_dataset.xlsx")
df
```

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

```python
#Changing into One hot encoding
df = pd.get_dummies(df, columns = ['Location', 'Gender'])
print(df)
```

```
       CustomerID             Name  Age  Subscription_Length_Months  \
0               1       Customer_1   63                          17
1               2       Customer_2   62                           1
2               3       Customer_3   24                           5
3               4       Customer_4   36                           3
4               5       Customer_5   46                          19
...           ...              ...  ...                         ...
99995       99996   Customer_99996   33                          23
99996       99997   Customer_99997   62                          19
99997       99998   Customer_99998   64                          17
99998       99999   Customer_99999   51                          20
99999      100000  Customer_100000   27                          19

       Monthly_Bill  Total_Usage_GB  Churn  Location_Chicago  \
0             73.36             236      0                 0
1             48.76             172      0                 0
2             85.47             460      0                 0
3             97.94             297      1                 0
4             58.14             266      0                 0
...             ...             ...    ...               ...
99995         55.13             226      1                 0
99996         61.65             351      0                 0
99997         96.11             251      1                 1
99998         49.25             434      1                 0
99999         76.57             173      1                 0

       Location_Houston  Location_Los Angeles  Location_Miami  \
0                     0                     1               0
1                     0                     0               0
2                     0                     1               0
3                     0                     0               1
4                     0                     0               1
...                 ...                   ...             ...
```

```
99995               1                   0              0
99996               0                   0              0
99997               0                   0              0
99998               0                   0              0
99999               0                   1              0

       Location_New York  Gender_Female  Gender_Male
0                      0              0            1
1                      1              1            0
2                      0              1            0
3                      0              1            0
4                      0              1            0
...                  ...            ...          ...
99995                  0              0            1
99996                  1              1            0
99997                  0              0            1
99998                  1              1            0
99999                  0              1            0

[100000 rows x 14 columns]
```

```python
#Finding correlation between all the columns
df.corr()
```

```
<ipython-input-81-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr i
  df.corr()
```

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage
CustomerID	1.000000	-0.001085	0.005444	0.001265	-0.004
Age	-0.001085	1.000000	0.003382	0.001110	0.001
Subscription_Length_Months	0.005444	0.003382	1.000000	-0.005294	-0.002
Monthly_Bill	0.001265	0.001110	-0.005294	1.000000	0.003
Total_Usage_GB	-0.004025	0.001927	-0.002203	0.003187	1.000
Churn	-0.004586	0.001559	0.002328	-0.000211	-0.002
Location_Chicago	-0.000666	0.006068	0.002187	-0.005772	-0.000
Location_Houston	-0.001390	0.001795	-0.001842	0.001856	-0.002
Location_Los Angeles	0.002501	-0.004971	-0.001234	0.003444	-0.001
Location_Miami	0.001617	0.001079	0.005508	-0.002521	0.001
Location_New York	-0.002069	-0.003982	-0.004630	0.002992	0.002
Gender_Female	0.000131	-0.000832	-0.000320	-0.002239	0.001
Gender_Male	-0.000131	0.000832	0.000320	0.002239	-0.001

```python
#To give brief about the dataset
df.describe()
```

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.0
mean	50000.500000	44.027020	12.490100	65.053197	274.393650	0.4
std	28867.657797	15.280283	6.926461	20.230696	130.463063	0.4
min	1.000000	18.000000	1.000000	30.000000	50.000000	0.0
25%	25000.750000	31.000000	6.000000	47.540000	161.000000	0.0
50%	50000.500000	44.000000	12.000000	65.010000	274.000000	0.0
75%	75000.250000	57.000000	19.000000	82.640000	387.000000	1.0
max	100000.000000	70.000000	24.000000	100.000000	500.000000	1.0

```python
#To find outliners
q1 = df['Subscription_Length_Months'].quantile(0.25)
q3 = df['Subscription_Length_Months'].quantile(0.75)
iqr = q3 - q1

# Define the lower and upper bounds for outliers
```

```
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Identify the outliers
outliers = df[df['Subscription_Length_Months'] < lower_bound] | df[df['Subscription_Length_Months'] > upper_bound]

# Print the outliers
print(outliers)
```

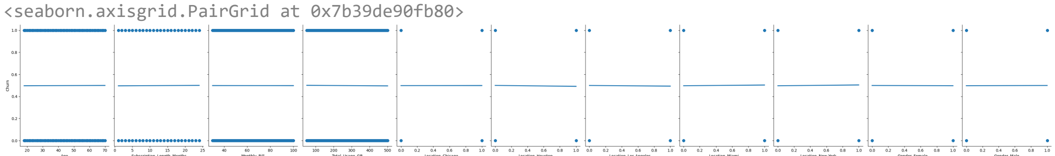
```
Empty DataFrame
Columns: [Female, Male, Chicago, Houston, Los Angeles, Miami, New York, CustomerID, Name, Age, Subscription_Length_Months, Monthly_Bill,
Index: []
```

```
#Finding outliners in Total_Usage_GB
q1 = df['Total_Usage_GB'].quantile(0.25)
q3 = df['Total_Usage_GB'].quantile(0.75)
iqr = q3 - q1

# Define the lower and upper bounds for outliers
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Identify the outliers
outliers = df[df['Total_Usage_GB'] < lower_bound] | df[df['Total_Usage_GB'] > upper_bound]

# Print the outliers
print(outliers)
```

```
Empty DataFrame
Columns: [Female, Male, Chicago, Houston, Los Angeles, Miami, New York, CustomerID, Name, Age, Subscription_Length_Months, Monthly_Bill,
Index: []
```

```
sns.pairplot(df, x_vars=['Age', 'Subscription_Length_Months',
       'Monthly_Bill', 'Total_Usage_GB', 'Location_Chicago',
       'Location_Houston', 'Location_Los Angeles', 'Location_Miami',
       'Location_New York', 'Gender_Female', 'Gender_Male'], y_vars='Churn', height=5, aspect=0.7, kind='reg')
# There is not much relation between the churn and diiferent columns
```

```
<seaborn.axisgrid.PairGrid at 0x7b39de90fb80>
```



```
dataplot = sb.heatmap(df.corr(),annot=True )
```

```
<ipython-input-108-bc08916b5934>:1: FutureWarning: The default value of numeric_only in DataFrame.corr
  dataplot = sb.heatmap(df.corr(),annot=True )
```


```python
columns = ['Age', 'Subscription_Length_Months',
       'Monthly_Bill', 'Total_Usage_GB', 'Location_Chicago',
       'Location_Houston', 'Location_Los Angeles', 'Location_Miami',
       'Location_New York', 'Gender_Female', 'Gender_Male']
```

```python
target_column_name = 'Churn'
```

```python
#Splitting training and test data
X_train, X_test, y_train, y_test = train_test_split(df[columns], df[target_column_name], test_size=0.2)
```

```python
X_train
```

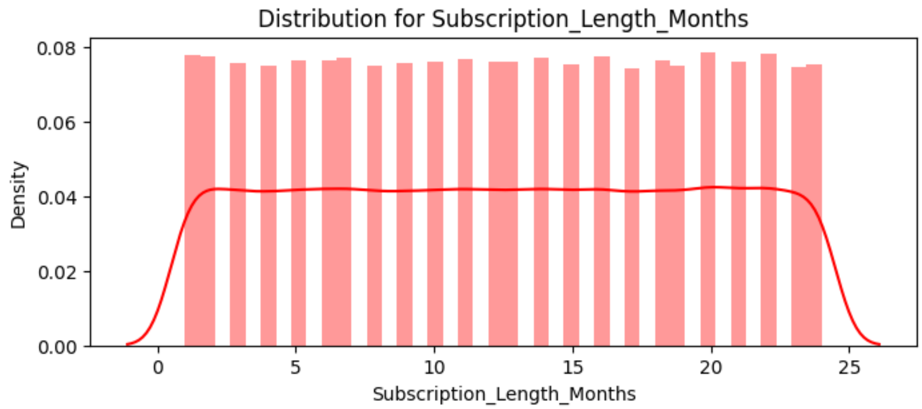
	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Location_Chicago	Location_Housto
38964	29	8	44.13	132	1	
6892	20	23	90.88	322	0	
27556	65	5	74.00	202	0	
83906	23	8	70.74	69	0	
24157	19	22	87.33	211	0	
...	...	...	...	...	...	
73949	45	18	32.36	402	0	
89128	58	3	48.05	277	1	
56975	68	13	95.53	79	1	
700	65	15	99.20	237	0	
30884	41	11	39.92	485	0	

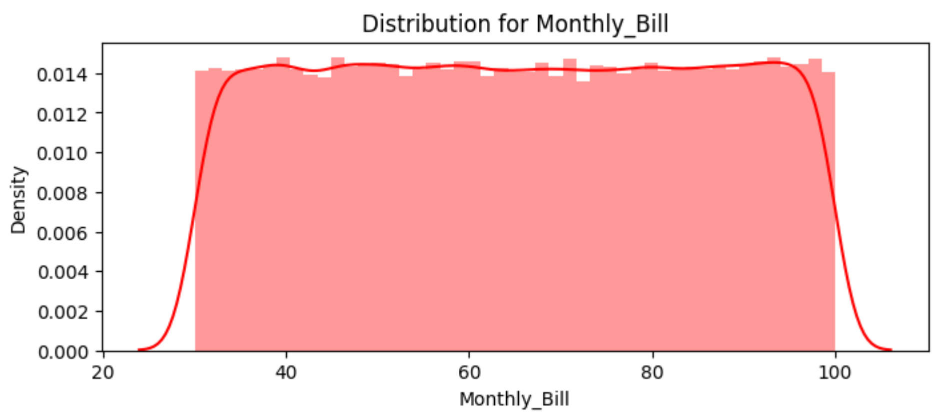
80000 rows × 11 columns

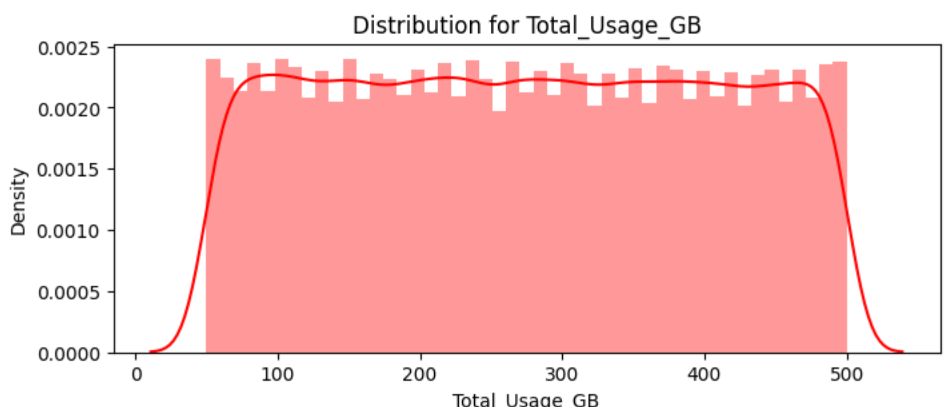
```python
def distplot(feature, frame, color='r'):
    plt.figure(figsize=(8,3))
    plt.title("Distribution for {}".format(feature))
    ax = sns.distplot(frame[feature], color= color)
```

```python
num_cols = ["Subscription_Length_Months", 'Monthly_Bill', 'Total_Usage_GB']
for feat in num_cols: distplot(feat, df)
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(frame[feature], color= color)
<ipython-input-93-8c8257b32bab>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(frame[feature], color= color)
```








```
#Train a KNN classification model with scikit-learn
knn_model = KNeighborsClassifier(n_neighbors = 11)
knn_model.fit(X_train,y_train)
predicted_y = knn_model.predict(X_test)
accuracy_knn = knn_model.score(X_test,y_test)
print("KNN accuracy:",accuracy_knn)
```

```
KNN accuracy: 0.5063
```

```
#Train a SVC classification model with scikit-learn
svc_model = SVC(random_state = 1)
```

```
svc_model.fit(X_train,y_train)
predict_y = svc_model.predict(X_test)
accuracy_svc = svc_model.score(X_test,y_test)
print("SVM accuracy is :",accuracy_svc)
```

```
print(classification_report(y_test, predict_y))
```

```
#Train a Random Forest classification model with scikit-learn
model_rf = RandomForestClassifier(n_estimators=500 , oob_score = True, n_jobs = -1,
                                  random_state =50,max_leaf_nodes = 30)
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))

```

```
0.49965
```

```
print(classification_report(y_test, prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.50      0.68      0.58     10080
           1       0.49      0.32      0.39      9920

    accuracy                           0.50     20000
   macro avg       0.50      0.50      0.48     20000
weighted avg       0.50      0.50      0.48     20000
```

```
#Train a XGB classification model with scikit-learn
model = XGBClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

```
0.50445
```

```
#Fine tuning XB Classification model to get better accuracy score
model = XGBClassifier(learning_rate=0.0001,
                      colsample_bytree = 0.4,
                      subsample = 0.8,
                      objective='binary:logistic',
                      n_estimators=1000,
                      reg_alpha = 0.3,
                      max_depth=5,
                      gamma=100)
model.fit(X_train, y_train)
preds = model.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

```
0.504
```

```
#Train a MLP classification model with scikit-learn
mlp_classifier = MLPClassifier(hidden_layer_sizes=(8, 20), activation='relu', solver='adam', max_iter=50000)
mlp_classifier.set_params(alpha=0.0001, batch_size=10, learning_rate_init=0.0001)
mlp_classifier.fit(X_train, y_train)
y_pred = mlp_classifier.predict(X_test)
accuracy = mlp_classifier.score(X_test, y_test)

print('Accuracy:', accuracy)
```

```
Accuracy: 0.50395
```

```
#Train a Logistic Regression model with scikit-learn
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

```
Logistic Regression accuracy is : 0.5021
```

```
lr_pred= lr_model.predict(X_test)
```

```
lr_pred= lr_model.predict(X_test)
report = classification_report(y_test,lr_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.51      0.60      0.55     10080
           1       0.50      0.40      0.45      9920

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

```
#To improve the accuracy we can use Pytorch Neural Network and can overfit the data ,
#but that would be good to only improve the accuracy score but it will give inaccurate result for new data.
#Out of all the models implemented KNN Classification works best for the data as it gives highest Accuracy score of 50.63%
```