Importing the Neccessary Libraries

In [1]:
```
import opendatasets as od
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

Data Extraction From Kaggle

In [3]:
```
od.download('https://www.kaggle.com/datasets/mark18vi/telecom-churn-data')
```

```
Skipping, found downloaded files in ".\telecom-churn-data" (use force=True to force download)
```

In [4]:
```
data = pd.read_csv('telecom-churn-data/telecom_churn_me.csv',index_col=0)
```

Data Exploration and Cleaning

In [4]:
```
data
```

Out[4]:

	PTY_PROFILE_SUB_TYPE	SOCIO_ECONOMIC_SEGMENT	PARTY_NATIONALITY	PARTY_GENDER_CD	TARGET	YEAR_JOINED	CU
0	Residential	EMIRATI	United Arab Emirates	M	0	1994	
1	Prestige	EMIRATI	United Arab Emirates	M	0	1994	
2	Residential	EMIRATI	United Arab Emirates	M	0	1994	
3	Prestige	EMIRATI	United Arab Emirates	M	0	1994	
4	Residential	EMIRATI	United Arab Emirates	M	0	1994	
...	...	...	...	...	...	...	
1140610	Residential	EMIRATI	United Arab Emirates	M	0	2017	
1140611	Residential	YOUTH	United Arab Emirates	M	0	2017	
1140612	Consumer via Retailer	EXPATS	Comoros	M	0	2017	
1140613	Residential	EXPATS	Philippines	M	0	2017	
1140614	Residential	EXPATS	Syrian Arab Republic	M	0	2017	

1140604 rows × 27 columns

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1140604 entries, 0 to 1140614
Data columns (total 27 columns):
 #   Column                    Non-Null Count    Dtype  
---  ------                    --------------    -----  
 0   PTY_PROFILE_SUB_TYPE      1140604 non-null  object 
 1   SOCIO_ECONOMIC_SEGMENT    1140604 non-null  object 
 2   PARTY_NATIONALITY         1140604 non-null  object 
 3   PARTY_GENDER_CD           1140604 non-null  object 
 4   TARGET                    1140604 non-null  int64  
 5   YEAR_JOINED               1140604 non-null  int64  
 6   CURRENT_YEAR              1140604 non-null  int64  
 7   BILL_AMOUNT               1140604 non-null  float64
 8   PAID_AMOUNT               1140604 non-null  float64
 9   PAYMENT_TRANSACTIONS      1140604 non-null  int64  
 10  PARTY_REV                 1140604 non-null  float64
 11  PREPAID_LINES             1140604 non-null  int64  
 12  POSTPAID_LINES            1140604 non-null  int64  
 13  OTHER_LINES               1140604 non-null  int64  
 14  LINE_REV                  1140604 non-null  float64
 15  STATUS                    1140604 non-null  object 
 16  MOUS_TO_LOCAL_MOBILES     1140604 non-null  float64
 17  MOUS_FROM_LOCAL_MOBILES   1140604 non-null  float64
 18  MOUS_TO_LOCAL_LANDLINES   1140604 non-null  float64
 19  MOUS_FROM_LOCAL_LANDLINES  1140604 non-null  float64
 20  MOUS_TO_INT_NUMBER        1140604 non-null  float64
 21  MOUS_FROM_INT_NUMBER      1140604 non-null  float64
 22  DATA_IN_BNDL              1140604 non-null  float64
 23  DATA_OUT_BNDL             1140604 non-null  float64
 24  DATA_USG_PAYG             1140604 non-null  float64
 25  COMPLAINTS                1140604 non-null  int64  
 26  Years_stayed              1140604 non-null  int64  
dtypes: float64(13), int64(9), object(5)
memory usage: 243.7+ MB
```

In [6]:
```
data.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	m
TARGET	1140604.0	0.052783	0.223601	0.000000	0.000000	0.000000	0.000000	1.0000
YEAR_JOINED	1140604.0	2013.380685	6.082378	1994.000000	2013.000000	2016.000000	2017.000000	2018.0000
CURRENT_YEAR	1140604.0	2018.947217	0.223601	2018.000000	2019.000000	2019.000000	2019.000000	2019.0000
BILL_AMOUNT	1140604.0	381.180390	369.703950	-2810.494133	174.137757	290.723940	460.977100	27026.2861
PAID_AMOUNT	1140604.0	392.139052	372.560750	0.000000	181.666667	300.729167	476.423333	22743.6350
PAYMENT_TRANSACTIONS	1140604.0	1.346936	0.730928	0.000000	1.000000	1.000000	2.000000	30.0000
PARTY_REV	1140604.0	1910.808512	18370.153789	-2011.420000	423.187917	834.713333	1553.675000	545537.0616
PREPAID_LINES	1140604.0	2.144700	5.751809	0.000000	0.000000	1.000000	3.000000	956.0000
POSTPAID_LINES	1140604.0	3.989309	43.951146	1.000000	1.000000	2.000000	3.000000	1282.0000
OTHER_LINES	1140604.0	0.899103	7.864311	0.000000	0.000000	0.000000	1.000000	1853.0000
LINE_REV	1140604.0	382.747476	361.962754	-1367.315000	176.663333	295.601667	464.870417	26446.0300
MOUS_TO_LOCAL_MOBILES	1140604.0	393.181442	901.601659	0.000000	28.920000	170.380000	469.540000	41513.4450
MOUS_FROM_LOCAL_MOBILES	1140604.0	129.417629	298.945225	0.000000	0.425000	29.445000	141.895000	31391.9900
MOUS_TO_LOCAL_LANDLINES	1140604.0	17.171341	38.574464	0.000000	0.350000	7.160000	22.035000	21851.1200
MOUS_FROM_LOCAL_LANDLINES	1140604.0	36.186004	120.017158	0.000000	0.015000	10.025000	39.260000	20344.3350
MOUS_TO_INT_NUMBER	1140604.0	51.084417	114.449291	0.000000	0.000000	2.175000	54.090000	4021.3650
MOUS_FROM_INT_NUMBER	1140604.0	6.840673	35.604586	0.000000	0.000000	0.000000	1.710000	16166.9550
DATA_IN_BNDL	1140604.0	10491.521078	33572.216744	0.000000	708.101562	4394.218506	9955.910278	898452.5976
DATA_OUT_BNDL	1140604.0	0.151718	7.416291	0.000000	0.000000	0.000000	0.000000	1912.5585
DATA_USG_PAYG	1140604.0	168.279280	6918.395813	0.000000	0.000000	0.000000	0.000000	999000.0000
COMPLAINTS	1140604.0	0.078777	0.324774	0.000000	0.000000	0.000000	0.000000	13.0000
Years_stayed	1140604.0	5.566531	6.104279	0.000000	2.000000	3.000000	6.000000	25.0000

In [5]:
```
data.duplicated().value_counts()
```

Out[5]:
```
False    1140124
True         480
dtype: int64
```

In [6]:
```
data.drop_duplicates(inplace=True)
data.duplicated().value_counts()
```

Out[6]:
```
False    1140124
dtype: int64
```

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
PTY_PROFILE_SUB_TYPE          0
SOCIO_ECONOMIC_SEGMENT        0
PARTY_NATIONALITY             0
PARTY_GENDER_CD               0
TARGET                        0
YEAR_JOINED                   0
CURRENT_YEAR                  0
BILL_AMOUNT                   0
PAID_AMOUNT                   0
PAYMENT_TRANSACTIONS          0
PARTY_REV                     0
PREPAID_LINES                 0
POSTPAID_LINES                0
OTHER_LINES                   0
LINE_REV                      0
STATUS                        0
MOUS_TO_LOCAL_MOBILES         0
MOUS_FROM_LOCAL_MOBILES       0
MOUS_TO_LOCAL_LANDLINES       0
MOUS_FROM_LOCAL_LANDLINES     0
MOUS_TO_INT_NUMBER            0
MOUS_FROM_INT_NUMBER          0
DATA_IN_BNDL                  0
DATA_OUT_BNDL                 0
DATA_USG_PAYG                 0
COMPLAINTS                    0
Years_stayed                  0
dtype: int64
```

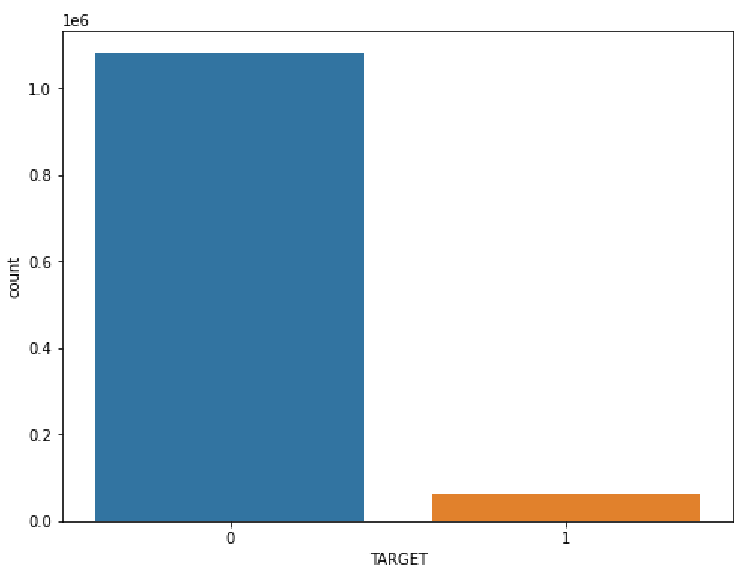
Exploratory Data Analysis

In [8]:
```
data['TARGET'].unique()
```

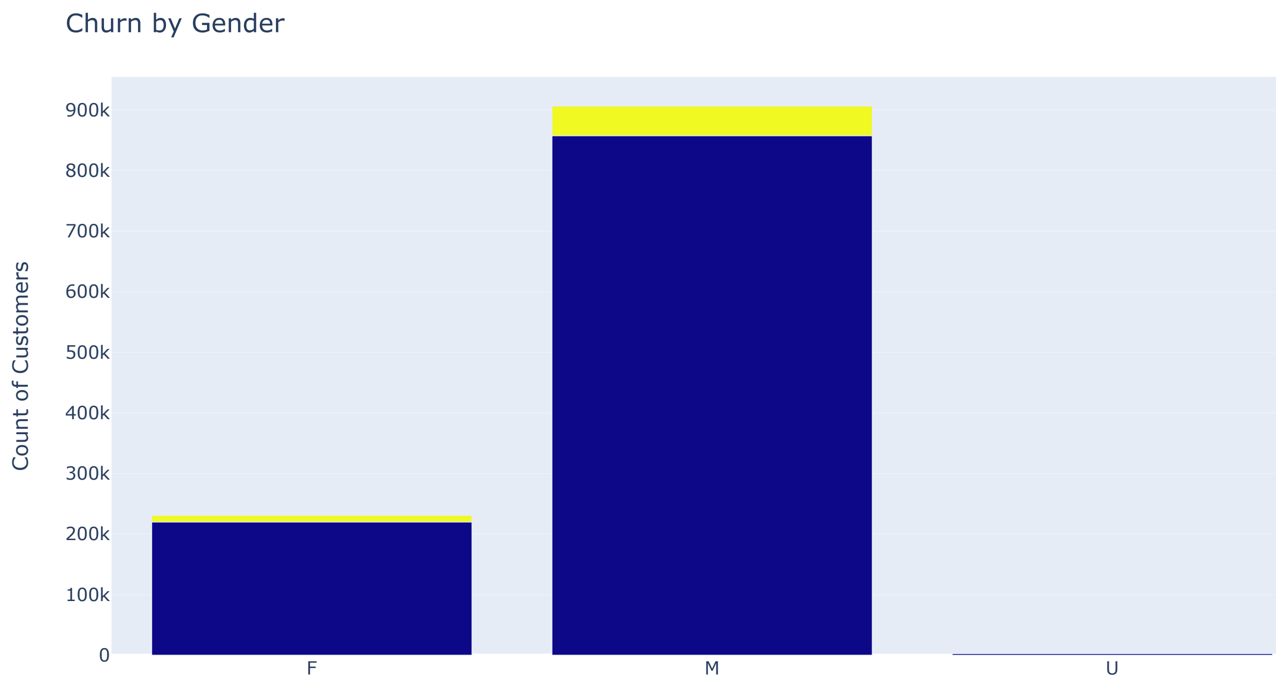
Out[8]:
```
array([0, 1], dtype=int64)
```

In [9]:
```
plt.figure(figsize=(8,6))
sns.countplot(x='TARGET',data=data)
```

Out[9]:
```
<AxesSubplot:xlabel='TARGET', ylabel='count'>
```



In [10]:
```
plt.figure(figsize=(10,6))
gender_group = data.groupby(['PARTY_GENDER_CD', 'TARGET'])['PARTY_GENDER_CD'].count().reset_index(name='count')
fig = px.bar(gender_group, x='PARTY_GENDER_CD', y='count', color='TARGET', barmode='group')
fig.update_layout(title_text='Churn by Gender', xaxis_title='Gender', yaxis_title='Count of Customers')
fig.show()
```

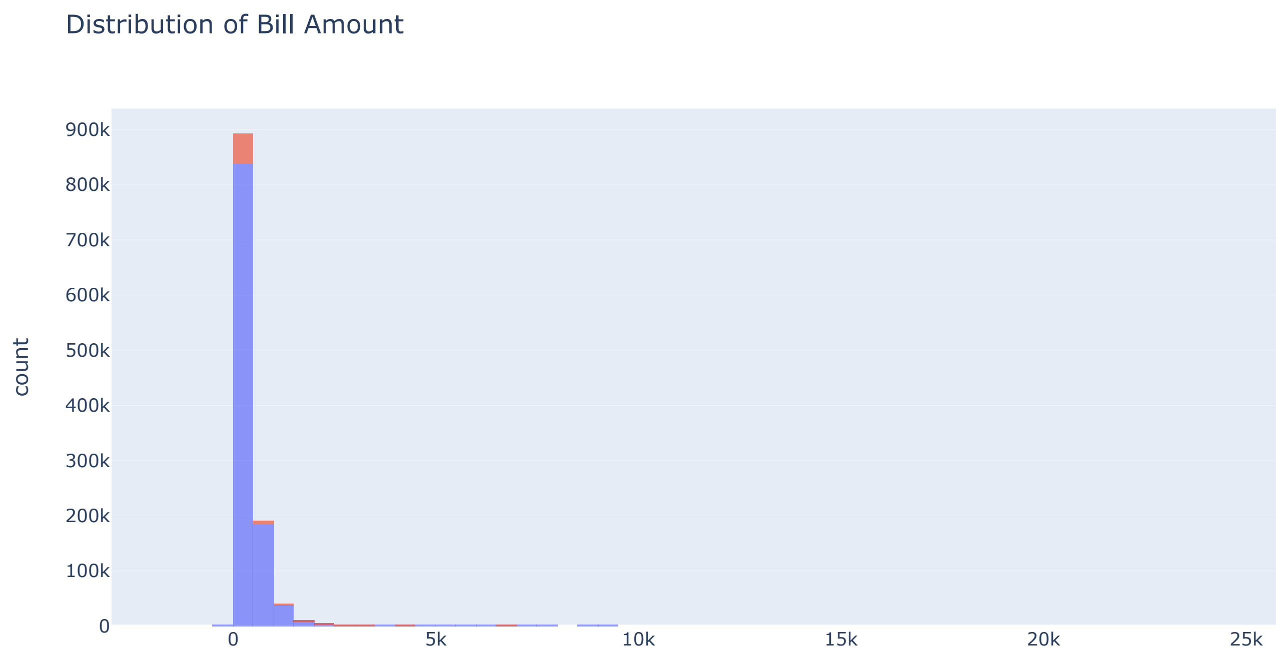



```
<Figure size 720x432 with 0 Axes>
```

This suggests that the majority of the churned customers are male (80%) while only a small percentage are female (20%). The distribution of gender among the customer base is skewed towards males and females, with fewer customers identifying as other.

In [11]:
```
fig = px.histogram(data, x="BILL_AMOUNT", color="TARGET", nbins=100,
                   title="Distribution of Bill Amount",
                   labels={"BILL_AMOUNT": "Bill Amount", "TARGET": "Churn"},
                   opacity=0.7)

fig.show()
```



The analysis of the bill amount column reveals that the majority of non-churned customers have a bill amount ranging from 0 to 2500 or around 250000. On the other hand, a significant number of non-churned customers have negative or defaulted bill amounts. In contrast, churned customers tend to have much lower bill amounts compared to the overall distribution. This suggests that bill amount may be a factor in customer churn.

In [12]:
```
fig = px.histogram(data, x="PAID_AMOUNT", color="TARGET", nbins=100,
                   title="Distribution of PAID Amount",
                   labels={"PAID_AMOUNT": "PAID Amount", "TARGET": "Churn"},
                   opacity=0.7)

fig.show()
```

It appears that there is a correlation between a customer's bill amount and their likelihood to churn. Customers with lower bill amounts tend to be more likely to churn compared to those with higher bill amounts. However, it is worth noting that even some high bill amount customers are still churning, indicating that there may be other factors at play. Further analysis is needed to determine the reasons behind these churns among high bill amount customers.

In [13]:
```
mean_churned = data[data["TARGET"] == 1]["BILL_AMOUNT"].mean()
mean_not_churned = data[data["TARGET"] == 0]["BILL_AMOUNT"].mean()

std_churned = data[data["TARGET"] == 1]["BILL_AMOUNT"].std()
std_not_churned = data[data["TARGET"] == 0]["BILL_AMOUNT"].std()

print("Mean Bill Amount for Churned Customers:", mean_churned)
print("Mean Bill Amount for Non-Churned Customers:", mean_not_churned)

print("Standard Deviation of Bill Amount for Churned Customers:", std_churned)
print("Standard Deviation of Bill Amount for Non-Churned Customers:", std_not_churned)
```

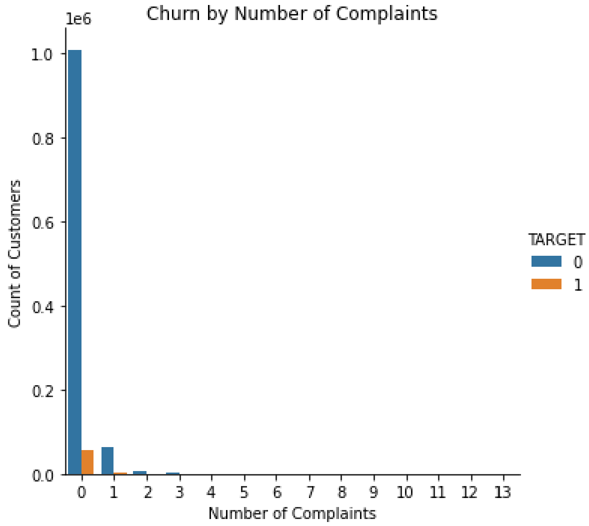
```
Mean Bill Amount for Churned Customers: 292.88632541951256
Mean Bill Amount for Non-Churned Customers: 386.26845276523153
Standard Deviation of Bill Amount for Churned Customers: 251.21456181459553
Standard Deviation of Bill Amount for Non-Churned Customers: 374.5872319648818
```

The mean bill amount for customers who have churned is significantly lower, at 292.92 Dollar, compared to the mean bill amount for customers who have not churned, at 386.29 Dollar. This suggests that customers who spend less on their monthly bill may be more likely to churn. Additionally, the standard deviation of the bill amount for churned customers is lower than that for non-churned customers, which suggests that the amount spent by churned customers is more consistent and predictable.

In [14]:
```
data['COMPLAINTS'].unique()
```

Out[14]:
```
array([ 0,  1,  2,  4,  3,  5, 13,  9,  6,  7, 10, 11,  8, 12],
      dtype=int64)
```

In [15]:
```
complaint_group = data.groupby(['COMPLAINTS', 'TARGET'])['COMPLAINTS'].count().reset_index(name='count')
sns.catplot(x='COMPLAINTS', y='count', hue='TARGET', kind='bar', data=complaint_group)
plt.title('Churn by Number of Complaints')
plt.xlabel('Number of Complaints')
plt.ylabel('Count of Customers')
plt.show()
```

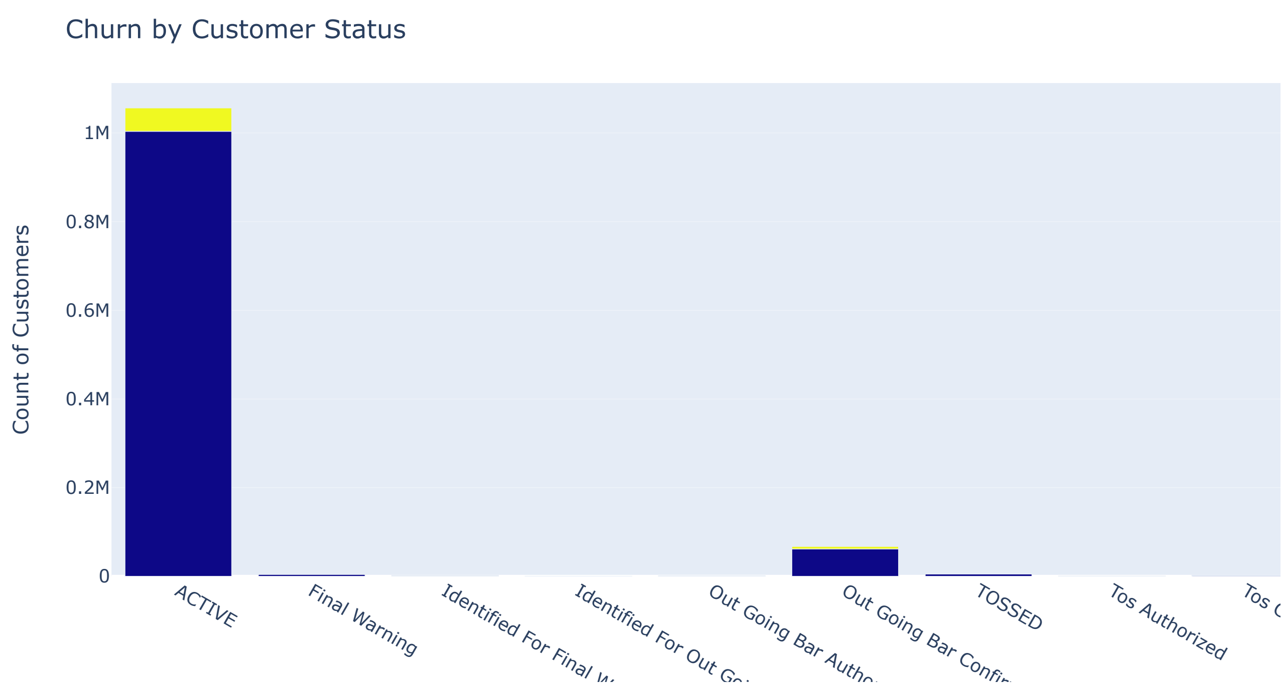


This plot indicates that the number of complaints alone may not be the sole reason for churn. While some customers may have left after filing a complaint, it is possible that there were other unresolved issues that led to their departure. Further investigation is needed to determine the root cause of churn in these cases, and to develop strategies for reducing churn in response to complaints.

In [16]:
```
data['STATUS'].unique()
```

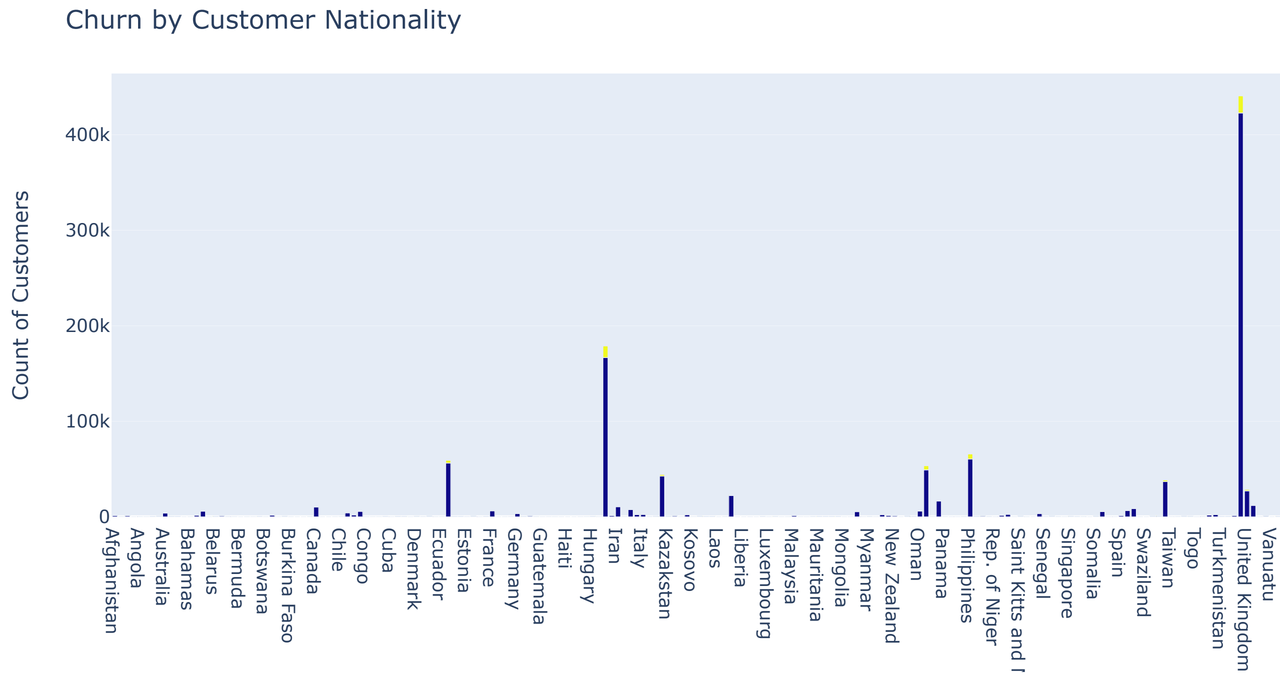
Out[16]:
```
array(['ACTIVE', 'Out Going Bar Confirmed', 'TOSSED', 'Final Warning',
       'Tos Confirmed', 'Out Going Bar Authorized',
       'Identified For Final Warning', 'Tos Authorized',
       'Identified For Out Going Bar'], dtype=object)
```

In [17]:
```
status_group = data.groupby(['STATUS', 'TARGET'])['STATUS'].count().reset_index(name='count')
fig = px.bar(status_group, x='STATUS', y='count', color='TARGET')
fig.update_layout(title='Churn by Customer Status', xaxis_title='Customer Status', yaxis_title='Count of Custome
fig.show()
```



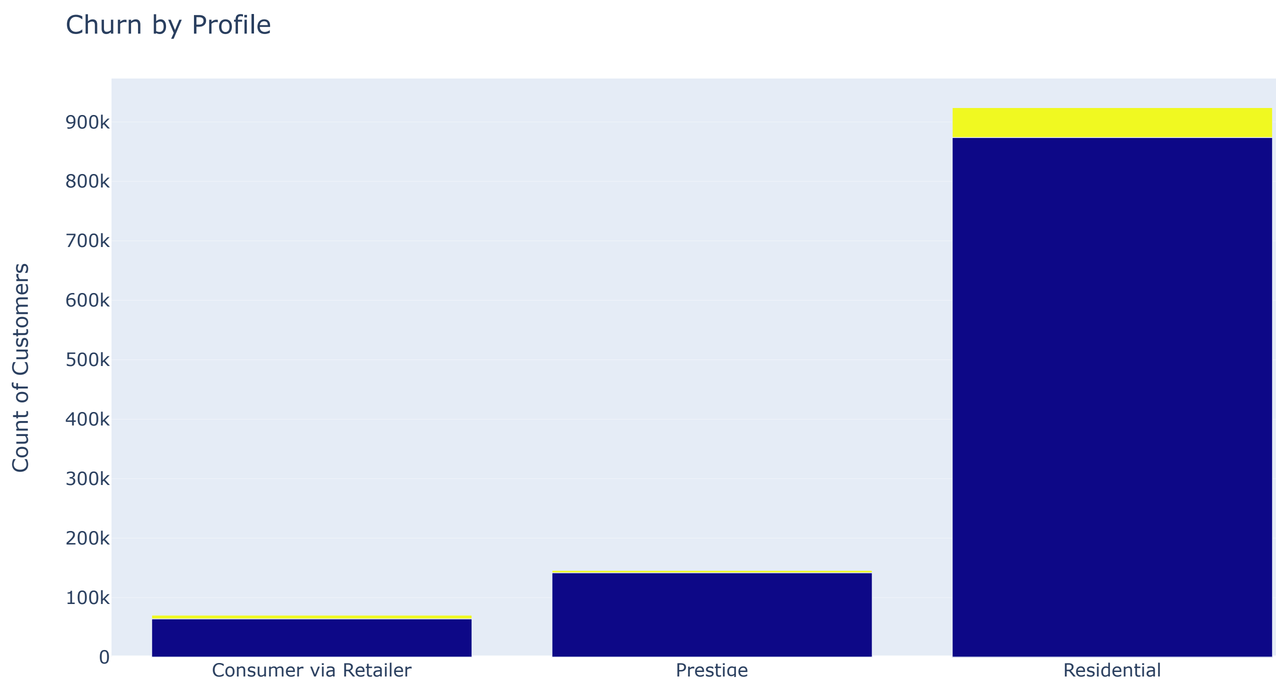
This indicates that a customer's status may not be the sole factor contributing to churn. The majority of churned customers were active, however 9.6% of churned customers were classified as outgoing barred or defaulted customers. Further analysis is needed to fully understand the relationship between outgoing barred and churn.

In [18]:
```
status_group = data.groupby(['PARTY_NATIONALITY', 'TARGET'])['PARTY_NATIONALITY'].count().reset_index(name='coun
fig = px.bar(status_group, x='PARTY_NATIONALITY', y='count', color='TARGET')
fig.update_layout(title='Churn by Customer Nationality', xaxis_title='Customer Nationality', yaxis_title='Count o
fig.show()
```



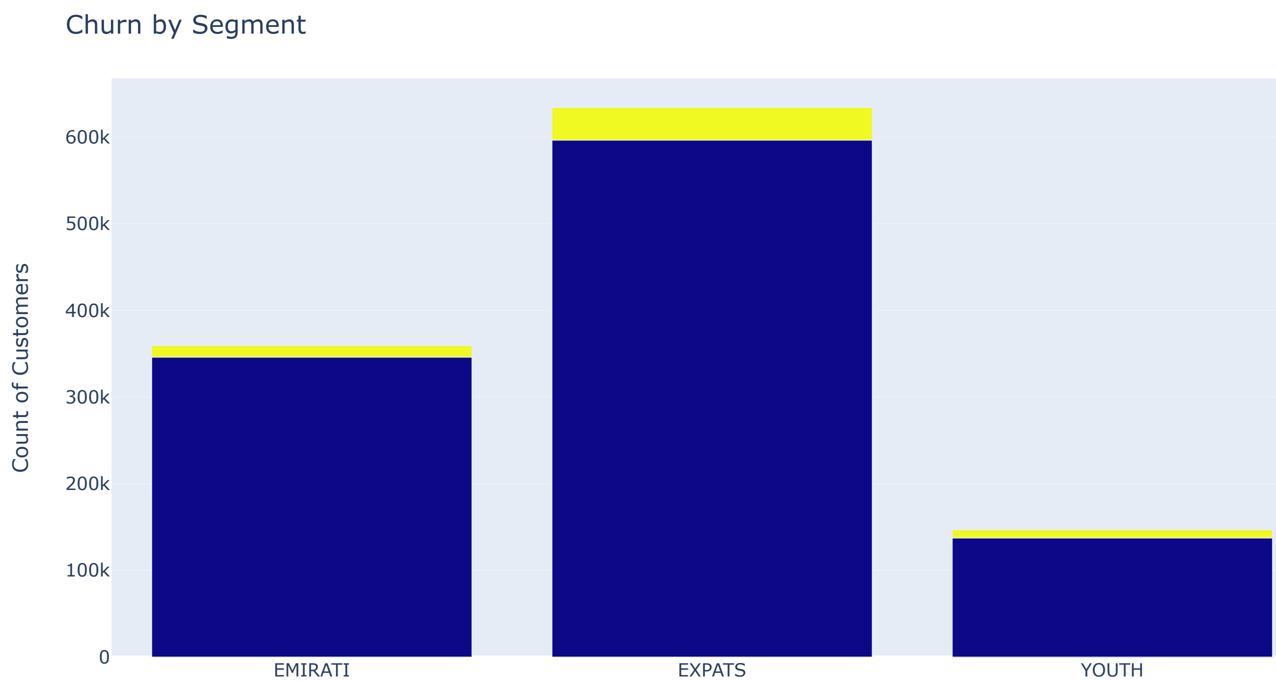
The country with the highest number of churned customers is the UAE, followed by India, the Philippines, Pakistan, Egypt,UK, and Jordan. Further analysis is needed to understand the specific reasons for customer churn in each of these countries, as internal factors and laws may play a role.

In [19]:
```
status_group = data.groupby(['PTY_PROFILE_SUB_TYPE', 'TARGET'])['PTY_PROFILE_SUB_TYPE'].count().reset_index(name=
fig = px.bar(status_group, x='PTY_PROFILE_SUB_TYPE', y='count', color='TARGET')
fig.update_layout(title='Churn by Profile', xaxis_title='Profile', yaxis_title='Count of Customers')
fig.show()
```



As we can see, residential customers have the highest churn rate, but consumers through retailers have a 10% churn rate, which is the highest among the three profile categories. It seems that it is impacted in a very high manner by churn.

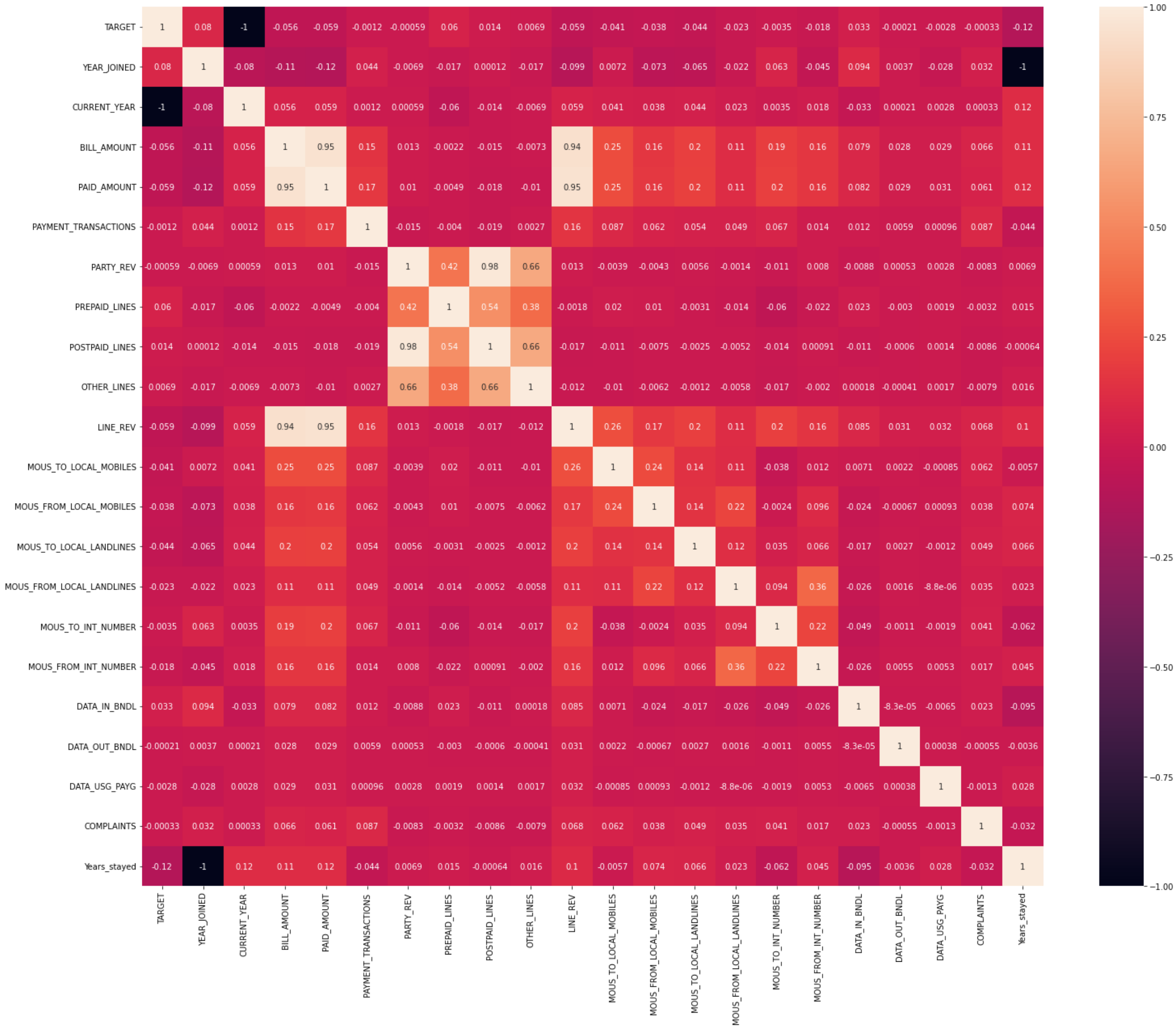
In [20]:
```
status_group = data.groupby(['SOCIO_ECONOMIC_SEGMENT', 'TARGET'])['SOCIO_ECONOMIC_SEGMENT'].count().reset_index(
fig = px.bar(status_group, x='SOCIO_ECONOMIC_SEGMENT', y='count', color='TARGET')
fig.update_layout(title='Churn by Segment', xaxis_title='Segment', yaxis_title='Count of Customers')
fig.show()
```



As we can see, expatriates hold the highest chance of churning. In addition, Emiratis and youth are comparatively at lower risk for churning. This can be a strong reason for churn to happen within the industry.

In [21]:
```
plt.figure(figsize=(25,20))
sns.heatmap(data.corr(),annot=True)
```

Out[21]:
```
<AxesSubplot:>
```



This observation indicates that the columns PARTY_REV, COMPLAINTS, PAYMENT_TRANSACTIONS, DATA_OUT_BNDL, OTHER_LINES, DATA_USG_PAYG, and MOU_TO_INT_NUMBER have a very weak correlation with the churn rate. It might not have a significant impact on the churn rate, and further analysis is needed to determine the actual reasons behind the churn. Additionally, other factors such as customer service, network quality, and pricing could also play a role in the churn rate.

Data Preparation

In [22]:
```
data.columns
```

Out[22]:
```
Index(['PTY_PROFILE_SUB_TYPE', 'SOCIO_ECONOMIC_SEGMENT', 'PARTY_NATIONALITY',
       'PARTY_GENDER_CD', 'TARGET', 'YEAR_JOINED', 'CURRENT_YEAR',
       'BILL_AMOUNT', 'PAID_AMOUNT', 'PAYMENT_TRANSACTIONS', 'PARTY_REV',
       'PREPAID_LINES', 'POSTPAID_LINES', 'OTHER_LINES', 'LINE_REV', 'STATUS',
       'MOUS_TO_LOCAL_MOBILES', 'MOUS_FROM_LOCAL_MOBILES',
       'MOUS_TO_LOCAL_LANDLINES', 'MOUS_FROM_LOCAL_LANDLINES',
       'MOUS_TO_INT_NUMBER', 'MOUS_FROM_INT_NUMBER', 'DATA_IN_BNDL',
       'DATA_OUT_BNDL', 'DATA_USG_PAYG', 'COMPLAINTS', 'Years_stayed'],
      dtype='object')
```

In [23]:
```
df = data.drop(['DATA_OUT_BNDL', 'DATA_USG_PAYG', 'COMPLAINTS','PAYMENT_TRANSACTIONS',
           'MOUS_TO_INT_NUMBER','PARTY_REV','STATUS','OTHER_LINES','PARTY_NATIONALITY',
                'POSTPAID_LINES','MOUS_FROM_INT_NUMBER','PARTY_GENDER_CD','DATA_OUT_BNDL','CURRENT_YEAR'],axis=1
```

In [24]:
```
df.columns
```

Out[24]:
```
Index(['PTY_PROFILE_SUB_TYPE', 'SOCIO_ECONOMIC_SEGMENT', 'TARGET',
       'YEAR_JOINED', 'BILL_AMOUNT', 'PAID_AMOUNT', 'PREPAID_LINES',
       'LINE_REV', 'MOUS_TO_LOCAL_MOBILES', 'MOUS_FROM_LOCAL_MOBILES',
       'MOUS_TO_LOCAL_LANDLINES', 'MOUS_FROM_LOCAL_LANDLINES', 'DATA_IN_BNDL',
       'Years_stayed'],
      dtype='object')
```

In [25]:
```
df['SOCIO_ECONOMIC_SEGMENT'] = df['SOCIO_ECONOMIC_SEGMENT'].astype('category')
df['SOCIO_ECONOMIC_SEGMENT'] = df['SOCIO_ECONOMIC_SEGMENT'].cat.codes

df['PTY_PROFILE_SUB_TYPE'] = df['PTY_PROFILE_SUB_TYPE'].astype('category')
df['PTY_PROFILE_SUB_TYPE'] = df['PTY_PROFILE_SUB_TYPE'].cat.codes
```

In [26]:
```
df.head().T
```

Out[26]:

	0	1	2	3	4
PTY_PROFILE_SUB_TYPE	2.000000	1.000000	2.000000	1.000000	2.000000
SOCIO_ECONOMIC_SEGMENT	0.000000	0.000000	0.000000	0.000000	0.000000
TARGET	0.000000	0.000000	0.000000	0.000000	0.000000
YEAR_JOINED	1994.000000	1994.000000	1994.000000	1994.000000	1994.000000
BILL_AMOUNT	931.208938	431.082618	50.619644	399.710034	612.665844
PAID_AMOUNT	812.175000	486.500000	52.815000	422.235000	825.888333
PREPAID_LINES	2.000000	6.000000	2.000000	3.000000	0.000000
LINE_REV	945.040000	493.815000	50.300000	406.586667	751.185000
MOUS_TO_LOCAL_MOBILES	1004.070000	159.050000	0.000000	288.805000	209.760000
MOUS_FROM_LOCAL_MOBILES	35.850000	10.595000	0.000000	158.500000	186.050000
MOUS_TO_LOCAL_LANDLINES	34.015000	7.715000	0.000000	2.670000	17.515000
MOUS_FROM_LOCAL_LANDLINES	72.075000	11.750000	0.000000	15.965000	28.685000
DATA_IN_BNDL	11944.079102	9903.157715	0.102539	3600.322266	3852.026367
Years_stayed	25.000000	25.000000	25.000000	25.000000	25.000000

In [29]:
```
df_scaled = MinMaxScaler().fit_transform(df)
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
df = df_scaled
```

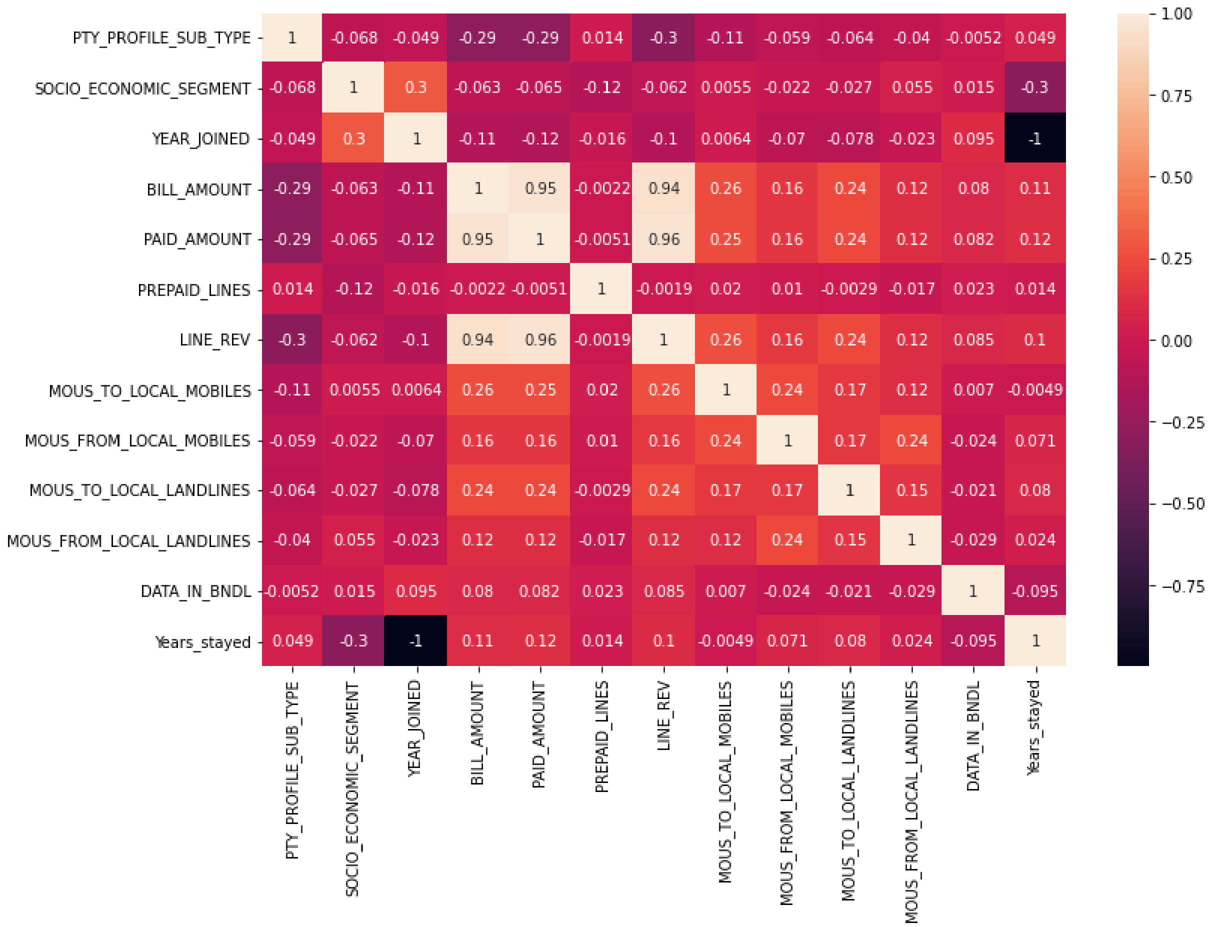
Model Training

In [30]:
```
X = df.drop(['TARGET'],axis=1)
y = df['TARGET']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42,shuffle=True,stratify=(
X_train,X_val,y_train,y_val =train_test_split(X_train,y_train,test_size=0.33,random_state=42)
```

In [31]:
```python
fig, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(X_train.corr(),annot=True)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81d7468520>



Logistic Regression

In [32]:
```python
lr = LogisticRegression(n_jobs=-1)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_val)
accuracy_lr = accuracy_score(y_val, y_pred_lr)
print("Accuracy of Logistic Regression:", accuracy_lr)
print("Classification Report of Logistic Regression:")
print(classification_report(y_val, y_pred_lr))
```

```
Accuracy of Logistic Regression: 0.9996152045762887
Classification Report of Logistic Regression:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    238858
         1.0       1.00      0.99      1.00     13224

    accuracy                           1.00    252082
   macro avg       1.00      1.00      1.00    252082
weighted avg       1.00      1.00      1.00    252082
```

Decision Tree

In [33]:
```
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_val)
accuracy_dt = accuracy_score(y_val, y_pred_dt)
print("Accuracy of Decision Tree:", accuracy_dt)
print("Classification Report of Decision Tree:")
print(classification_report(y_val, y_pred_dt))
```

```
Accuracy of Decision Tree: 0.9999047928848549
Classification Report of Decision Tree:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    238858
         1.0       1.00      1.00      1.00     13224

    accuracy                           1.00    252082
   macro avg       1.00      1.00      1.00    252082
weighted avg       1.00      1.00      1.00    252082
```

Random Forest

In [34]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_val)
accuracy_rf = accuracy_score(y_val, y_pred_rf)
print("Accuracy of Random Forest:", accuracy_rf)
print("Classification Report of Random Forest:")
print(classification_report(y_val, y_pred_rf))
```

```
Accuracy of Random Forest: 0.9951920406851739
Classification Report of Random Forest:
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00    238858
         1.0       1.00      0.91      0.95     13224

    accuracy                           1.00    252082
   macro avg       1.00      0.95      0.97    252082
weighted avg       1.00      1.00      1.00    252082
```

Gradient Boost

In [35]:
```
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_val)
accuracy_gb = accuracy_score(y_val, y_pred_gb)
print("Accuracy of Gradient Boosting:", accuracy_gb)
print("Classification Report of Gradient Boosting:")
print(classification_report(y_val, y_pred_gb))
```

```
Accuracy of Gradient Boosting: 0.9980244523607398
Classification Report of Gradient Boosting:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    238858
         1.0       1.00      0.96      0.98     13224

    accuracy                           1.00    252082
   macro avg       1.00      0.98      0.99    252082
weighted avg       1.00      1.00      1.00    252082
```

Evaluation Of Best Model

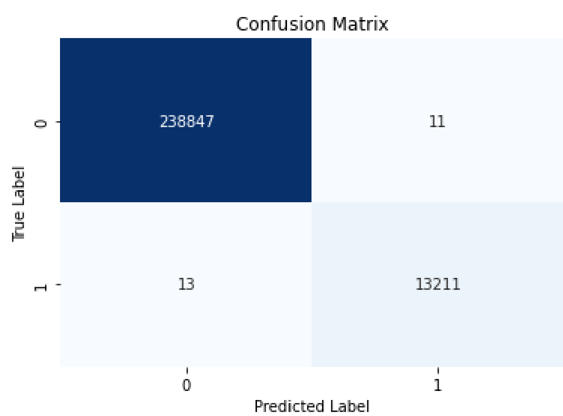
In [36]:
```
best_model = max(accuracy_lr, accuracy_dt, accuracy_rf, accuracy_gb)

if best_model == accuracy_lr:
    print("Logistic Regression is the best model with accuracy:", accuracy_lr)
    y_pred = y_pred_lr
elif best_model == accuracy_dt:
    print("Decision Tree is the best model with accuracy:", accuracy_dt)
    y_pred = y_pred_dt
elif best_model == accuracy_rf:
    print("Random Forest is the best model with accuracy:", accuracy_rf)
    y_pred = y_pred_rf
elif best_model == accuracy_gb:
    print("Gradient Boosting is the best model with accuracy:", accuracy_gb)
    y_pred = y_pred_gb
else:
    print('Error')
```

```
Decision Tree is the best model with accuracy: 0.9999047928848549
```

Confusion Matrix of Actual and Predicted Value

In [37]:
```
cm = confusion_matrix(y_val, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



In [37]: