Importing Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import warnings
warnings.filterwarnings('ignore')
```

Loading the Dataset

```
data = pd.read_csv("Customer Churn Prediction.csv")
data
```

iceRegistered	ViewingHoursPerWeek	...	ContentDownloadsPerMonth	GenrePreference	UserRating	SupportTicketsPerMonth	Gender	WatchlistSize	ParentalControl
Mobile	36.758104	...	10	Sci-Fi	2.176498	4	Male	3	No
Tablet	32.450568	...	18	Action	3.478632	8	Male	23	No
Computer	7.395160	...	23	Fantasy	4.238824	6	Male	1	Yes
Tablet	27.960389	...	30	Drama	4.276013	2	Male	24	Yes
TV	20.083397	...	20	Comedy	3.616170	4	Female	0	No
...	...	...	...	...	...	...	...	...	...
Computer	13.502729	...	47	Sci-Fi	3.697451	1	Male	8	Yes
TV	24.963291	...	35	Comedy	1.449742	4	Male	20	No
TV	10.628728	...	44	Action	4.012217	6	Male	13	Yes

Top 5 rows

```
data.head()
```

	AccountAge	MonthlyCharges	TotalCharges	SubscriptionType	PaymentMethod	PaperlessBilling	ContentType	MultiDeviceAccess	DeviceRegistered	ViewingHour
0	20	11.055215	221.104302	Premium	Mailed check	No	Both	No	Mobile	
1	57	5.175208	294.986882	Basic	Credit card	Yes	Movies	No	Tablet	
2	73	12.106657	883.785952	Basic	Mailed check	Yes	Movies	No	Computer	
3	32	7.263743	232.439774	Basic	Electronic check	No	TV Shows	No	Tablet	
4	57	16.953078	966.325422	Premium	Electronic check	Yes	TV Shows	No	TV	

5 rows × 21 columns

Info of dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243787 entries, 0 to 243786
Data columns (total 21 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   AccountAge               243787 non-null  int64
 1   MonthlyCharges           243787 non-null  float64
 2   TotalCharges             243787 non-null  float64
 3   SubscriptionType         243787 non-null  object
 4   PaymentMethod            243787 non-null  object
 5   PaperlessBilling         243787 non-null  object
 6   ContentType              243787 non-null  object
 7   MultiDeviceAccess        243787 non-null  object
 8   DeviceRegistered         243787 non-null  object
 9   ViewingHoursPerWeek      243787 non-null  float64
 10  AverageViewingDuration   243787 non-null  float64
 11  ContentDownloadsPerMonth 243787 non-null  int64
 12  GenrePreference          243787 non-null  object
 13  UserRating               243787 non-null  float64
 14  SupportTicketsPerMonth   243787 non-null  int64
 15  Gender                   243787 non-null  object
 16  WatchlistSize            243787 non-null  int64
```

Columns of Dataset

```
data.columns
```

```
Index(['AccountAge', 'MonthlyCharges', 'TotalCharges', 'SubscriptionType',
       'PaymentMethod', 'PaperlessBilling', 'ContentType', 'MultiDeviceAccess',
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration',
       'ContentDownloadsPerMonth', 'GenrePreference', 'UserRating',
       'SupportTicketsPerMonth', 'Gender', 'WatchlistSize', 'ParentalControl',
       'SubtitlesEnabled', 'CustomerID', 'Churn'],
      dtype='object')
```

Statistical Info

```
data.describe()
```

	AccountAge	MonthlyCharges	TotalCharges	ViewingHoursPerWeek	AverageViewingDuration	ContentDownloadsPerMonth	UserRating	SupportTicketsP
count	243787.000000	243787.000000	243787.000000	243787.000000	243787.000000	243787.000000	243787.000000	24378
mean	60.083758	12.490695	750.741017	20.502179	92.264061	24.503513	3.002713	
std	34.285143	4.327615	523.073273	11.243753	50.505243	14.421174	1.155259	
min	1.000000	4.990062	4.991154	1.000065	5.000547	0.000000	1.000007	
25%	30.000000	8.738543	329.147027	10.763953	48.382395	12.000000	2.000853	
50%	60.000000	12.495555	649.878487	20.523116	92.249992	24.000000	3.002261	
75%	90.000000	16.238160	1089.317362	30.219396	135.908048	37.000000	4.002157	
max	119.000000	19.989957	2378.723844	39.999723	179.999275	49.000000	4.999989	

```
data.shape
```

```
(243787, 21)
```

Handling Missing Values

```
data.isnull().sum()
```

```
AccountAge                0
MonthlyCharges            0
TotalCharges              0
SubscriptionType          0
PaymentMethod             0
PaperlessBilling          0
ContentType               0
MultiDeviceAccess         0
DeviceRegistered          0
ViewingHoursPerWeek       0
AverageViewingDuration    0
ContentDownloadsPerMonth  0
GenrePreference           0
UserRating                0
SupportTicketsPerMonth    0
Gender                    0
WatchlistSize             0
ParentalControl           0
SubtitlesEnabled          0
CustomerID                0
Churn                     0
dtype: int64
```

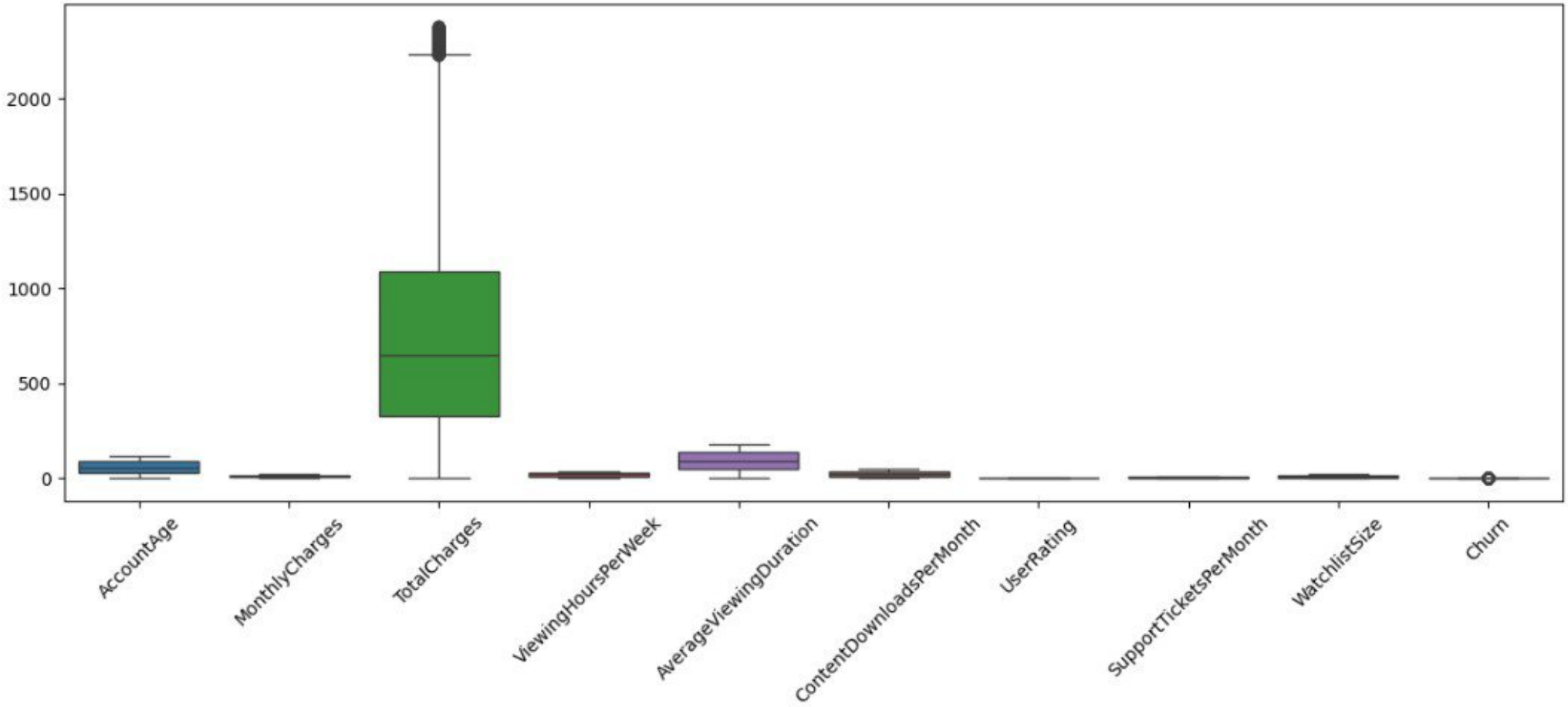
Handling Duplicates

```
data.duplicated().sum()
```

```
np.int64(0)
```

Checking Outliers via Box Plot

```
plt.figure(figsize=(15, 5))
sns.boxplot(data)
plt.xticks(rotation=45)
plt.show()
```

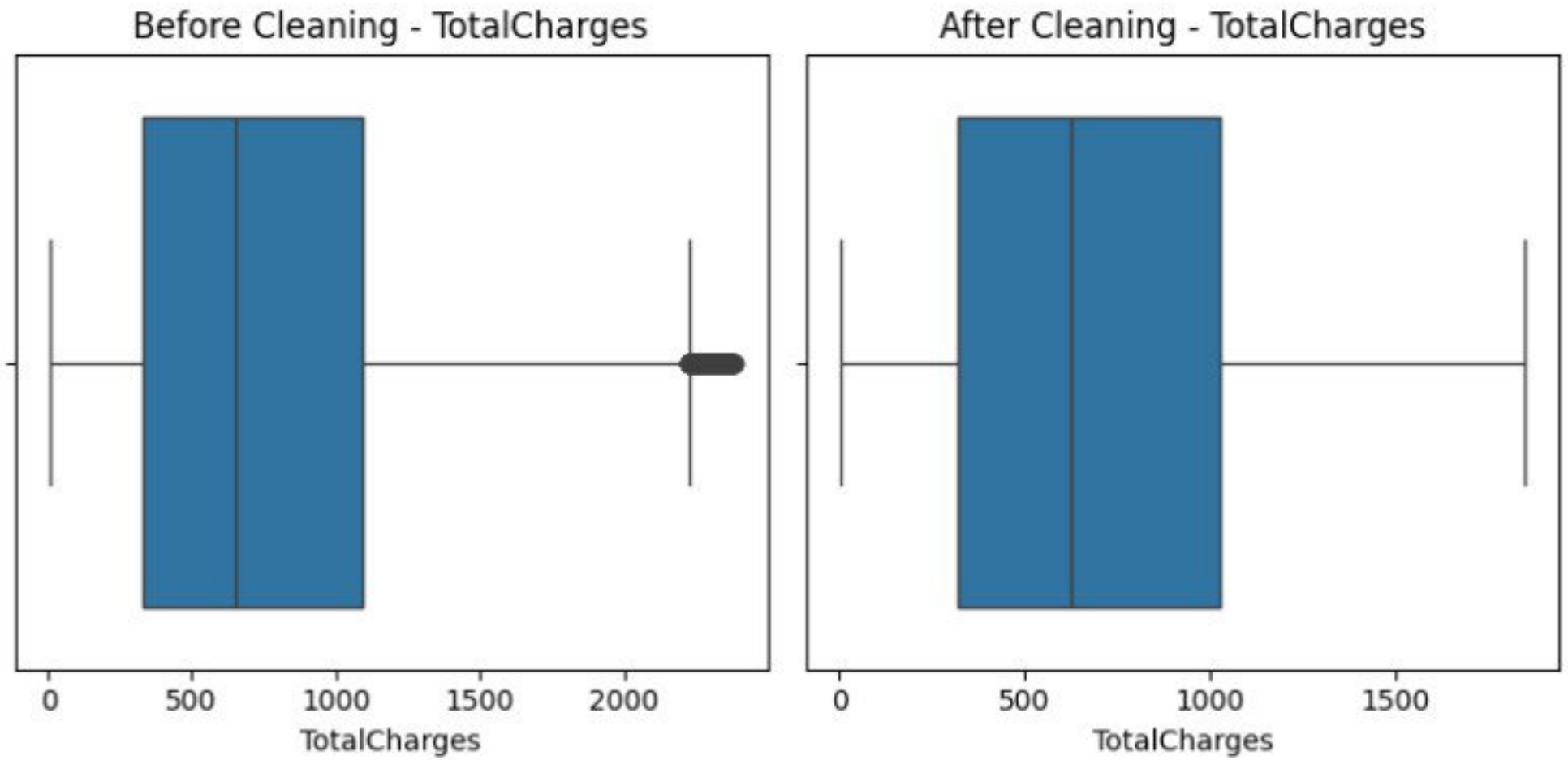


Handling Outliers

```
## handling outliers in (Total CHarges)
# calculate the q1,q3,iqr

q1 = data['TotalCharges'].quantile(0.25)
q3 = data['TotalCharges'].quantile(0.75)
iqr = q3-q1

lowerbound = q1-1.0 * iqr
upperbound = q3+1.0 * iqr

outliers_Total_charges = data[(data['TotalCharges'] < lowerbound) | (data['TotalCharges'] > upperbound )]
```

```
remove_outliers_Totalcharges = data[(data['TotalCharges']>= lowerbound) & (data['TotalCharges']<= upperbound)]
```

```
plt.figure(figsize=(8, 4))


# Subplot 1 - Before Cleaning
plt.subplot(1, 2, 1)
sns.boxplot(x=data['TotalCharges'])
plt.title("Before Cleaning - TotalCharges")

# Subplot 2 - After Cleaning
plt.subplot(1, 2, 2)
sns.boxplot(x=remove_outliers_Totalcharges['TotalCharges'])
plt.title("After Cleaning - TotalCharges")

plt.tight_layout()
plt.show()
```

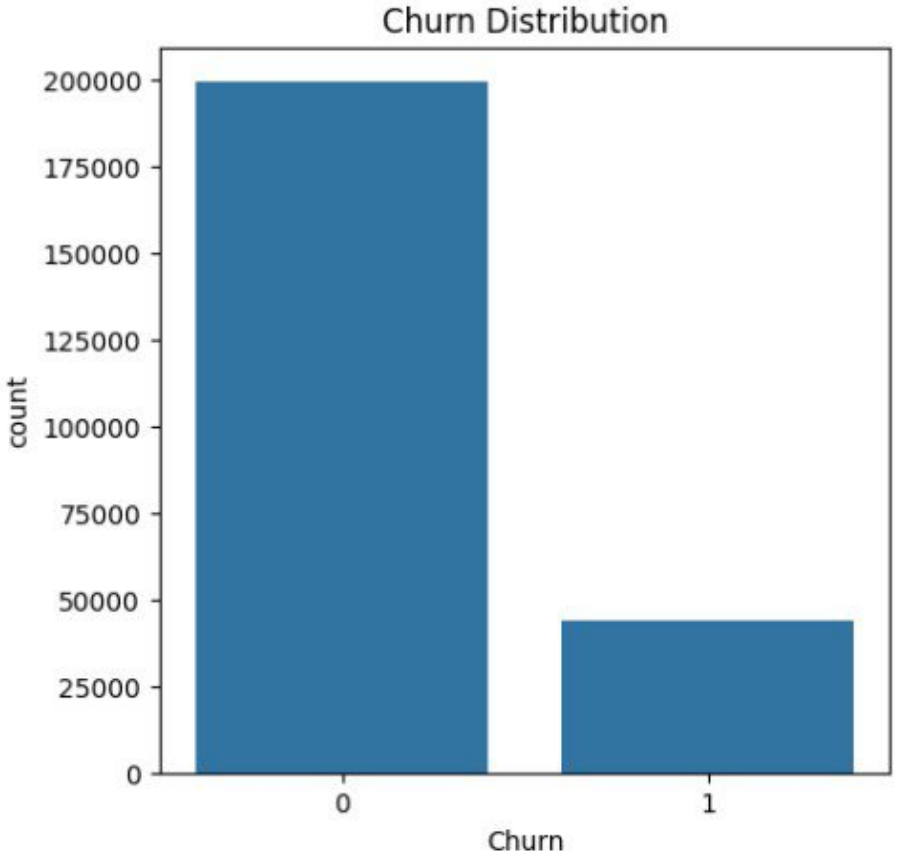

Before Cleaning - TotalCharges
0
500
1000
1500
2000
TotalCharges
After Cleaning - TotalCharges
0
500
1000
1500
TotalCharges

Understand insights from the data:

Visualizations:

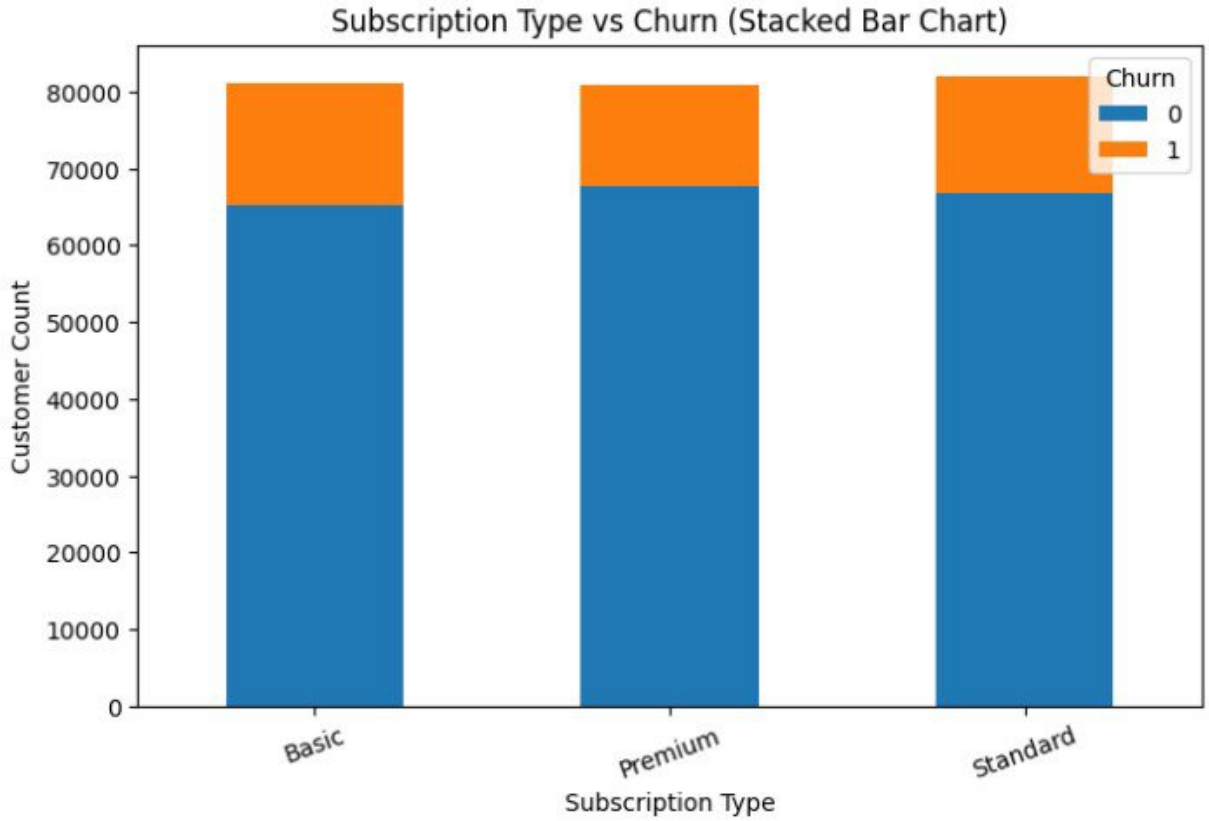
CountPlot

```
plt.figure(figsize=(5,5))
sns.countplot(x='Churn', data=data)
plt.title("Churn Distribution")
plt.show()
```



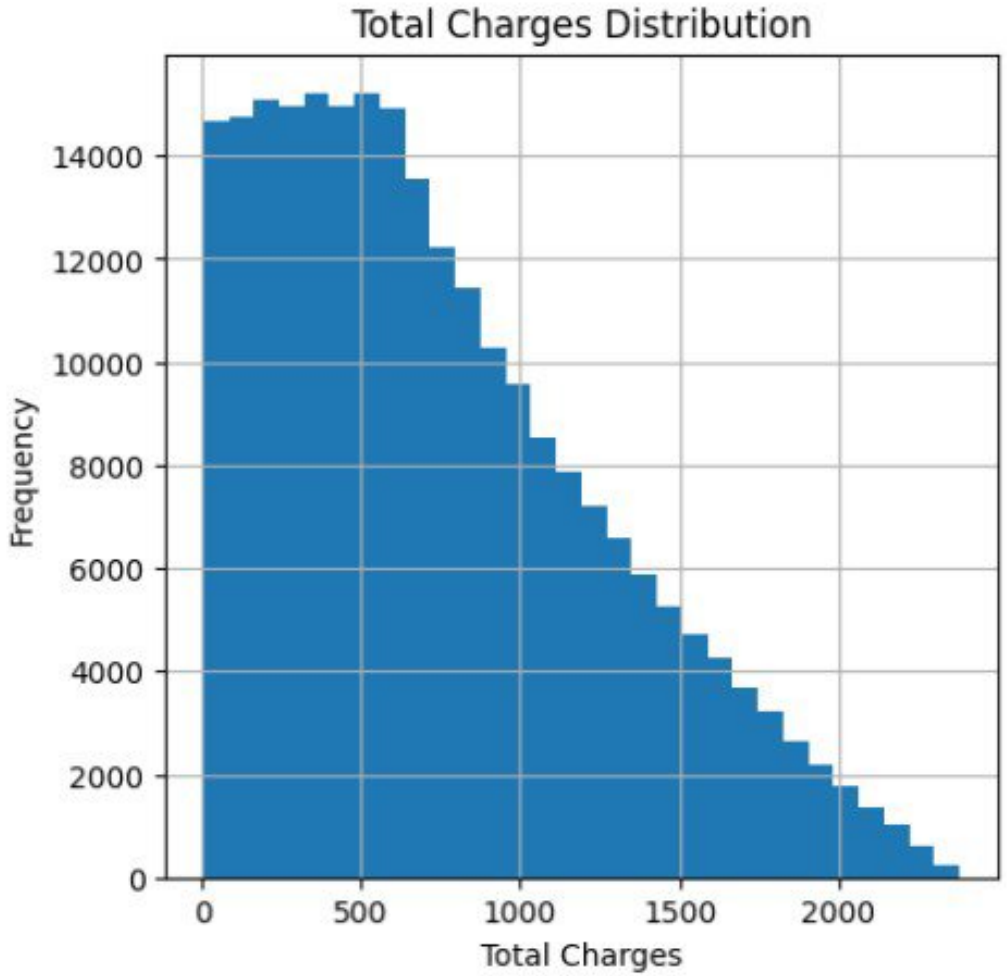
Stacked Bar Chart

```
# Prepare data for stacked chart
stack_data = pd.crosstab(data['SubscriptionType'], data['Churn'])

# Plot
stack_data.plot(kind='bar', stacked=True, figsize=(8,5))
plt.title("Subscription Type vs Churn (Stacked Bar Chart)")
plt.xlabel("Subscription Type")
plt.ylabel("Customer Count")
plt.legend(title="Churn")
plt.xticks(rotation=20)
plt.show()
```



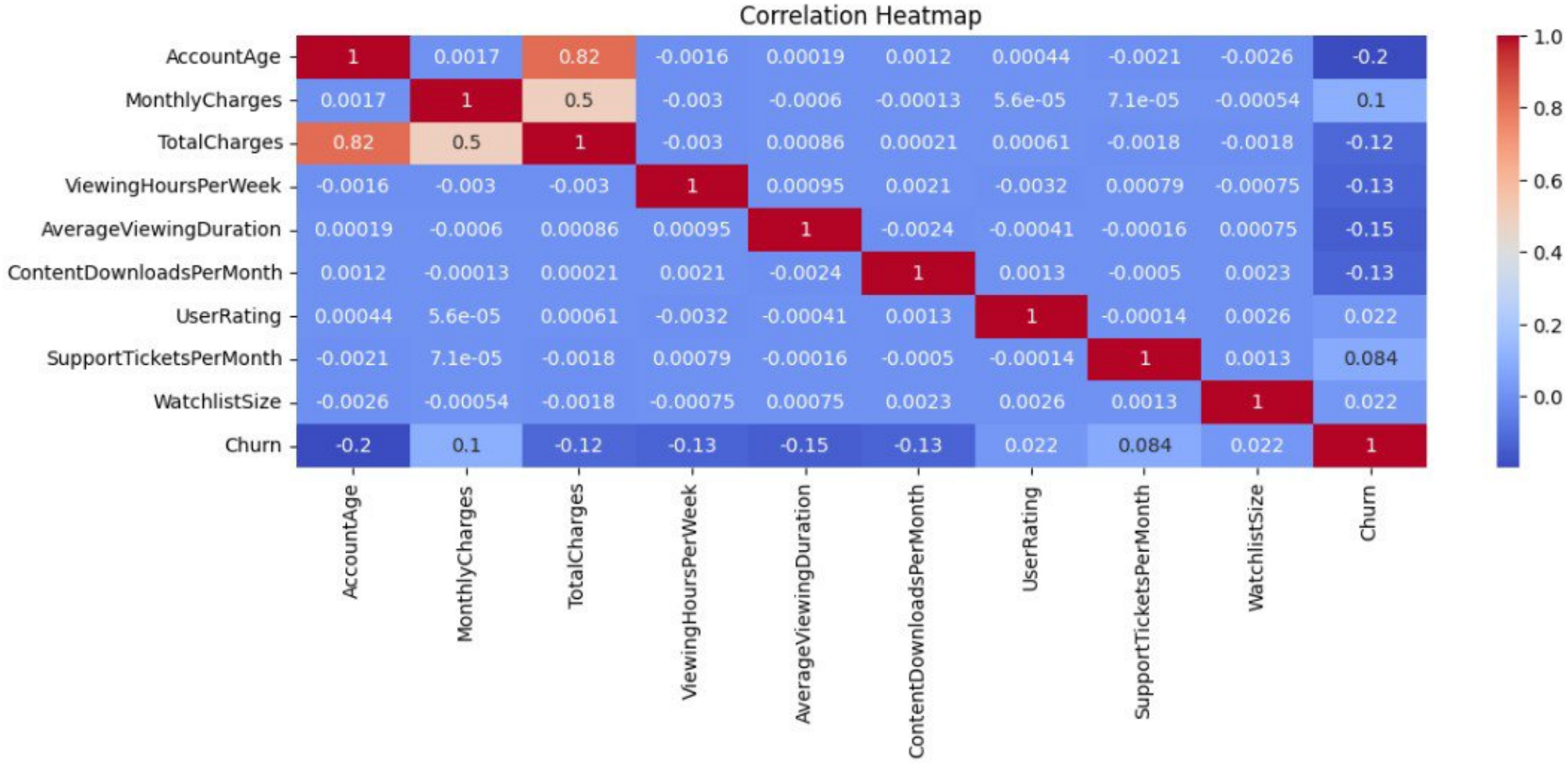
Histogram

```
data['TotalCharges'].hist(bins=30, figsize=(5,5))
plt.title("Total Charges Distribution")
plt.xlabel("Total Charges")
plt.ylabel("Frequency")
plt.show()
```



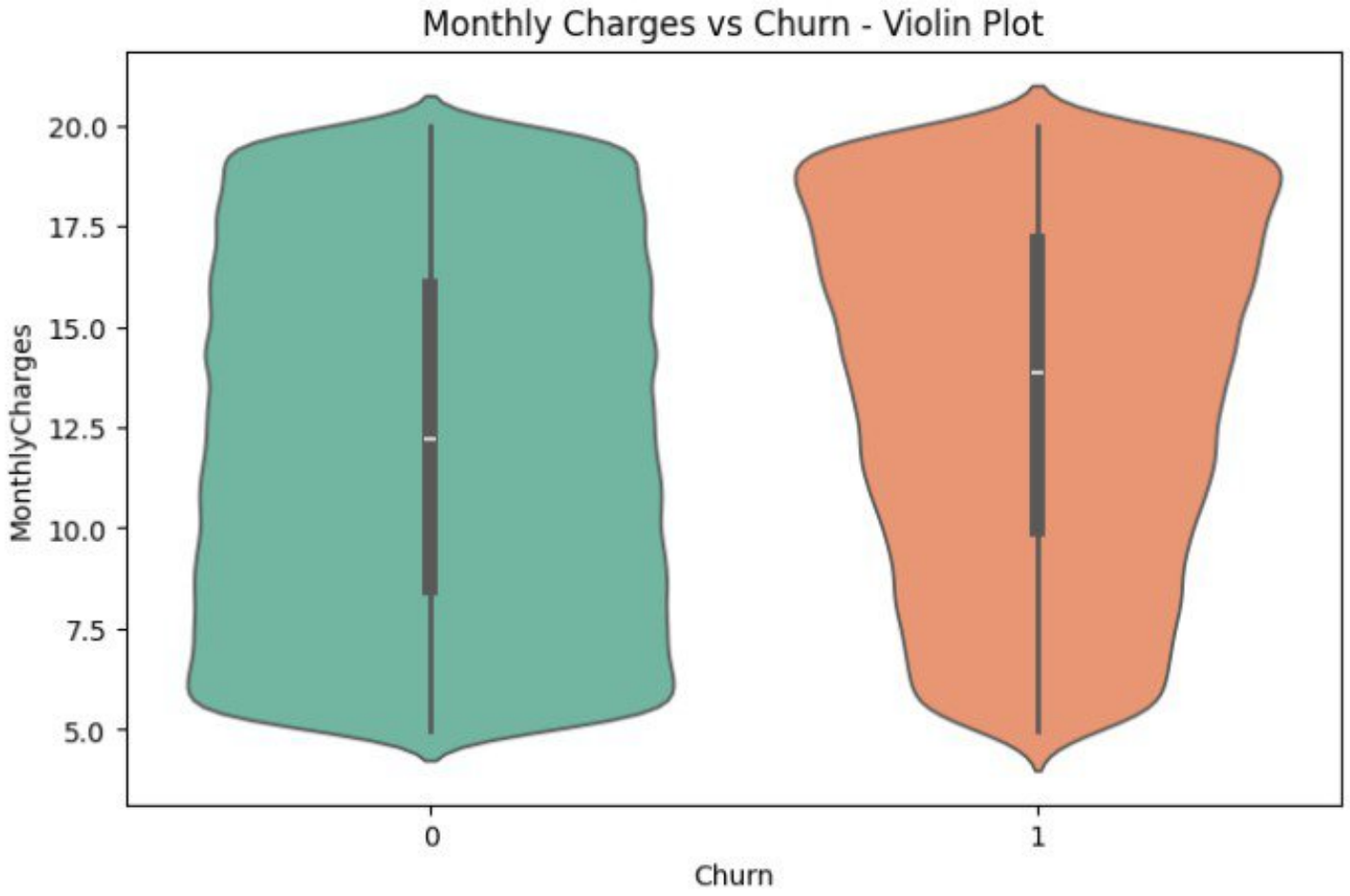
Correlation Heatmap

```
numeric_data = data.select_dtypes(include=['int64', 'float64'])
plt.figure(figsize=(13,4))
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



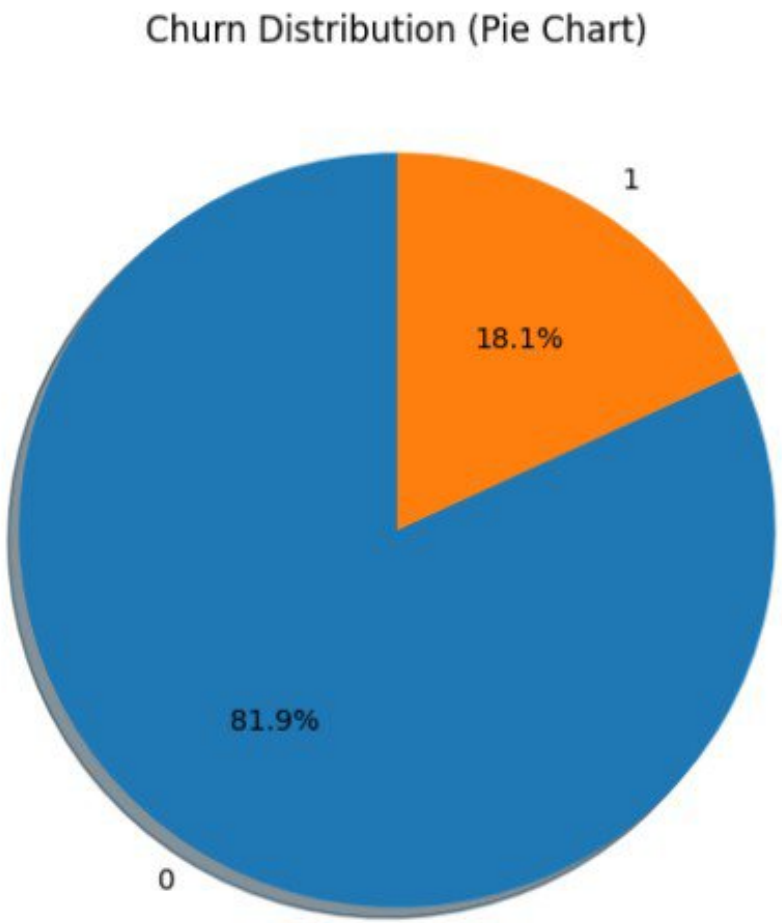
Violin Plot: Monthly Charges vs Churn

```
plt.figure(figsize=(8,5))
sns.violinplot(data=data, x='Churn', y='MonthlyCharges', palette='Set2')
plt.title("Monthly Charges vs Churn - Violin Plot")
plt.show()
```



Pie Chart

```
plt.figure(figsize=(6,6))
data['Churn'].value_counts().plot.pie(autopct='%1.1f%%', shadow=True, startangle=90)
plt.title("Churn Distribution (Pie Chart)")
plt.ylabel("")
plt.show()
```



Model Bulding

* Split data into features (X) and target (y)

* Split the data into training and testing

* Build each ML model

* Fit the model on training data

* Make predictions on test data

* Evaluate model performance

```
# Drop ID column
data = data.drop(columns=["CustomerID"])
```

```
# Split features and target
X = data.drop("Churn", axis=1)
y = data["Churn"]

# Identify columns
num_cols = X.select_dtypes(include=['int64','float64']).columns
cat_cols = X.select_dtypes(include=['object']).columns
```

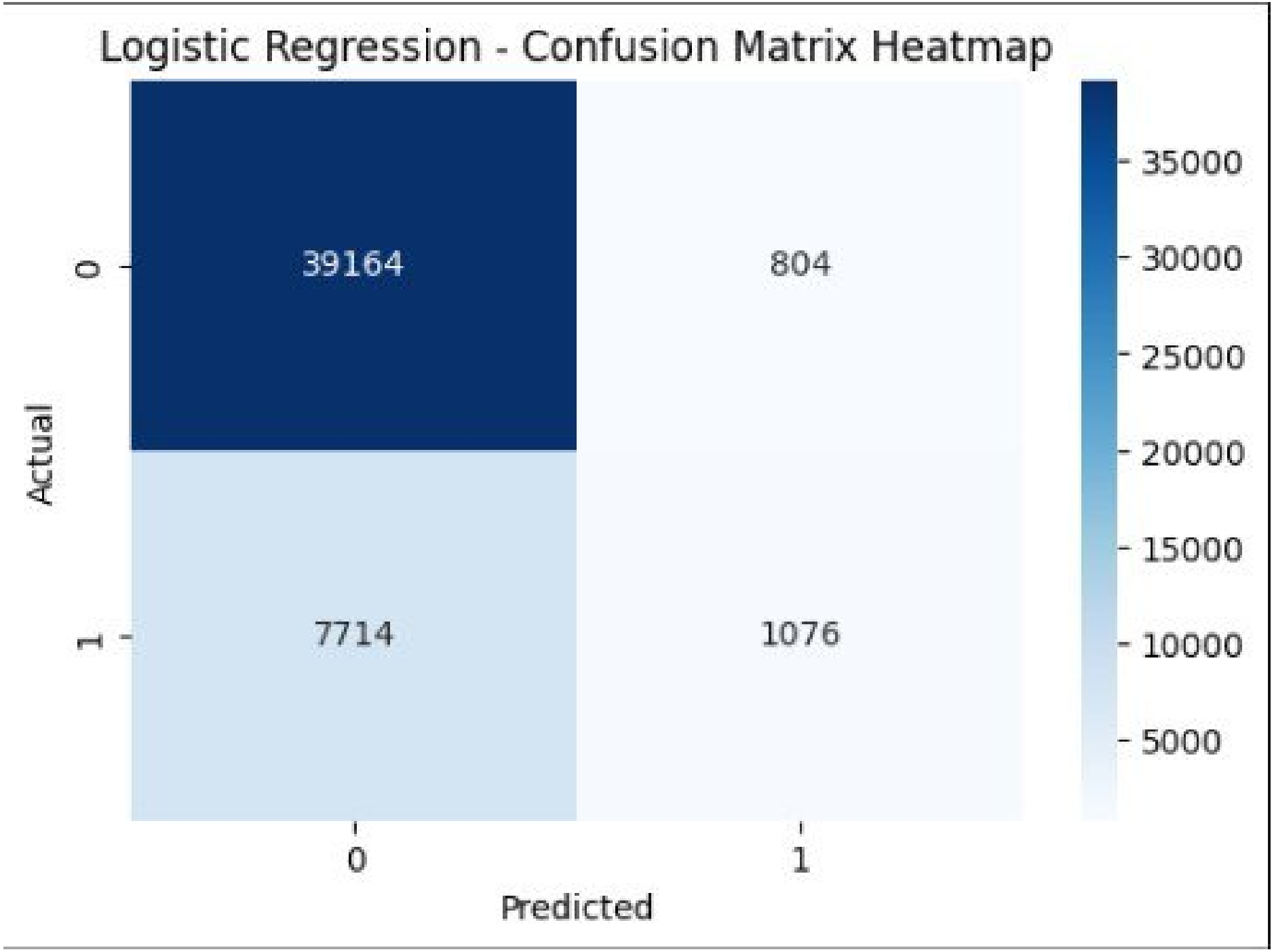
```
# ENCODING
encoder = OneHotEncoder(drop='first', handle_unknown='ignore')
encoded_cat = encoder.fit_transform(X[cat_cols]).toarray()

# Combine numeric + encoded categorical
X_encoded = np.hstack((X[num_cols].values, encoded_cat))

# SCALING
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)
```

```
#TRAIN TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split( X_scaled, y, test_size=0.2, random_state=42)
```

Logistic Regression

```
log_model = LogisticRegression(max_iter=200)
log_model.fit(X_train, y_train)
log_pred = log_model.predict(X_test)

print("Logistic Regression Accuracy:\t", accuracy_score(y_test, log_pred))
print("\nClassification Report: ", classification_report(y_test, log_pred))
print("\nConfusion Matrix:", confusion_matrix(y_test, log_pred))
cm =confusion_matrix(y_test, log_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression - Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

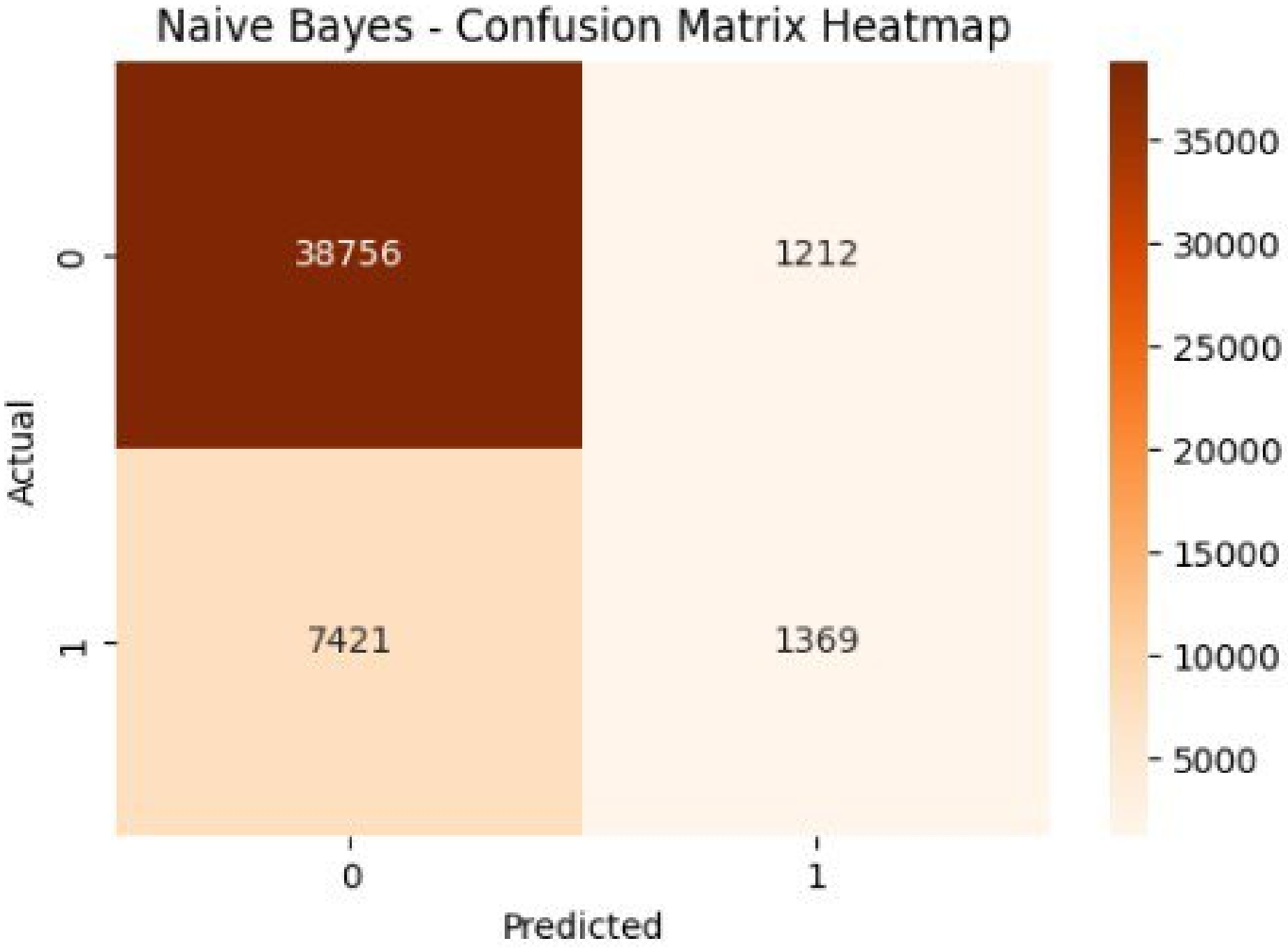
```
Logistic Regression Accuracy:    0.8253004635136798

Classification Report:                precision    recall  f1-score   support

           0       0.84      0.98      0.90     39968
           1       0.57      0.12      0.20      8790

    accuracy                           0.83     48758
   macro avg       0.70      0.55      0.55     48758
weighted avg       0.79      0.83      0.78     48758


Confusion Matrix: [[39164   804]
 [ 7714  1076]]
```


Logistic Regression - Confusion Matrix Heatmap
39164
804
7714
1076
0
1
Actual
0
1
Predicted
35000
30000
25000
20000
15000
10000
5000

Naive Bayes

```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)

print("Naive Bayes Accuracy:", accuracy_score(y_test, nb_pred))
print("\nNaive Bayes Classification Report:\n", classification_report(y_test, nb_pred))
print("\nNaive Bayes Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))

cm = confusion_matrix(y_test, nb_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges')
plt.title("Naive Bayes - Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

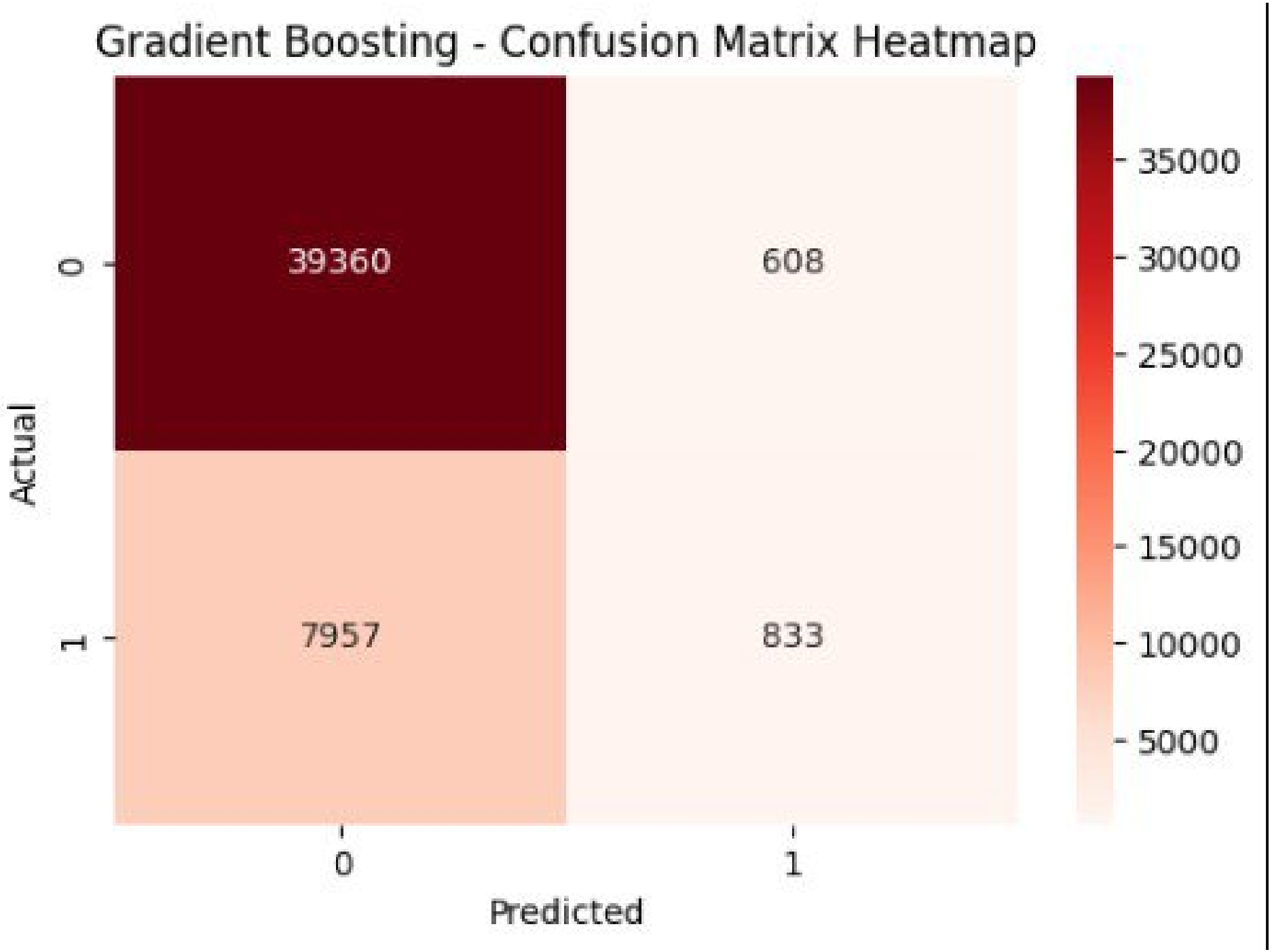
```
Naive Bayes Accuracy: 0.8229418762049304

Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.97      0.90     39968
           1       0.53      0.16      0.24      8790

    accuracy                           0.82     48758
   macro avg       0.68      0.56      0.57     48758
weighted avg       0.78      0.82      0.78     48758


Naive Bayes Confusion Matrix:
 [[38756  1212]
 [ 7421  1369]]
```


Naive Bayes - Confusion Matrix Heatmap
38756
1212
7421
1369
0
1
Actual
0
1
Predicted
35000
30000
25000
20000
15000
10000
5000

Gradient Boosting

```
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
gb_pred = gb_model.predict(X_test)

print("Gradient Boosting Accuracy:", accuracy_score(y_test, gb_pred))
print("Classification_Report: ", classification_report(y_test, gb_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, gb_pred))

cm = confusion_matrix(y_test, gb_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds')
plt.title("Gradient Boosting - Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

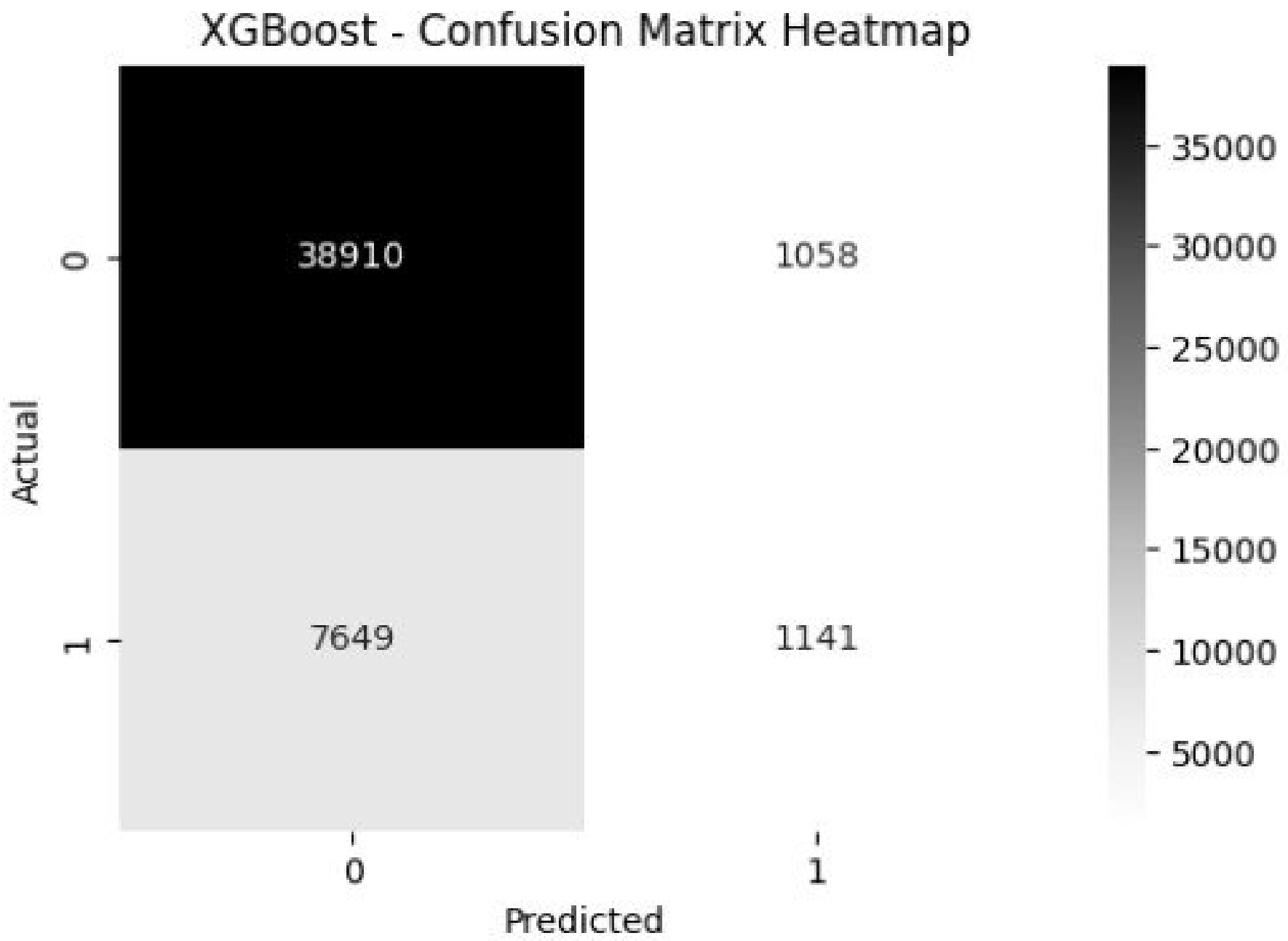
```
Gradient Boosting Accuracy: 0.8243365191353214
Classification_Report:                precision    recall  f1-score   support

           0       0.83      0.98      0.90     39968
           1       0.58      0.09      0.16      8790

    accuracy                           0.82     48758
   macro avg       0.70      0.54      0.53     48758
weighted avg       0.79      0.82      0.77     48758


Confusion Matrix:
 [[39360   608]
 [ 7957   833]]
```


Gradient Boosting - Confusion Matrix Heatmap
39360
608
7957
833
0
1
Actual
0
1
Predicted
35000
30000
25000
20000
15000
10000
5000

XGBoost

```
xgb_model = XGBClassifier(eval_metric='logloss', use_label_encoder=False)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)

print("XGBoost Accuracy:", accuracy_score(y_test, xgb_pred))
print("\nXGBoost Classification Report:\n", classification_report(y_test, xgb_pred))
print("\nXGBoost Confusion Matrix:\n", confusion_matrix(y_test, xgb_pred))

cm = confusion_matrix(y_test, xgb_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greys')
plt.title("XGBoost - Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

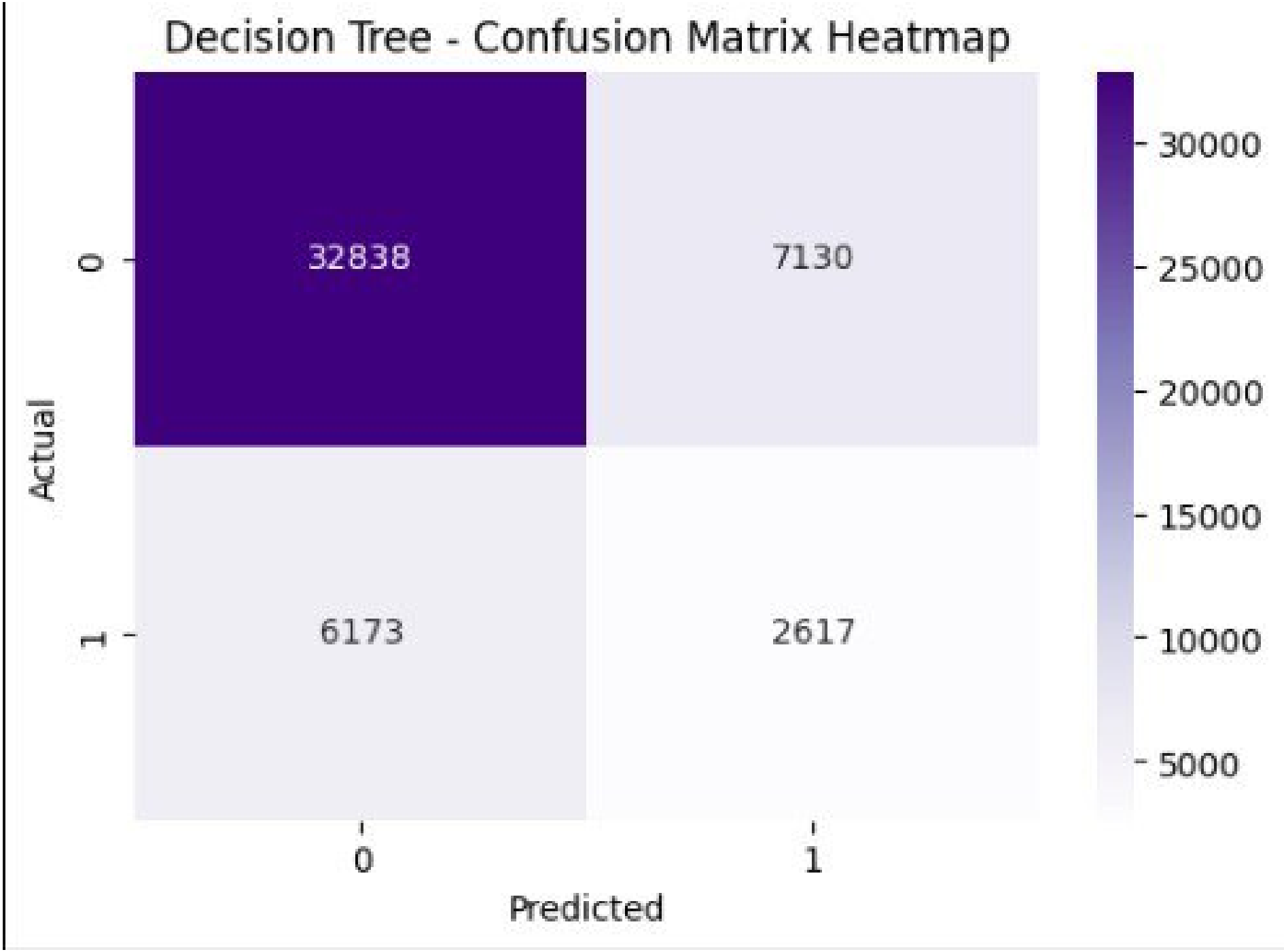
```
XGBoost Accuracy: 0.8214241765453875

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.97      0.90     39968
           1       0.52      0.13      0.21      8790

    accuracy                           0.82     48758
   macro avg       0.68      0.55      0.55     48758
weighted avg       0.78      0.82      0.77     48758


XGBoost Confusion Matrix:
 [[38910  1058]
 [ 7649  1141]]
```


XGBoost - Confusion Matrix Heatmap
38910
1058
7649
1141
0
1
Actual
0
1
Predicted
35000
30000
25000
20000
15000
10000
5000

Decision Tree

```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, dt_pred))
print("\nDecision Tree Classification Report:\n", classification_report(y_test, dt_pred))
print("\nDecision Tree Confusion Matrix:\n", confusion_matrix(y_test, dt_pred))
cm = confusion_matrix(y_test, dt_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples')
plt.title("Decision Tree - Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

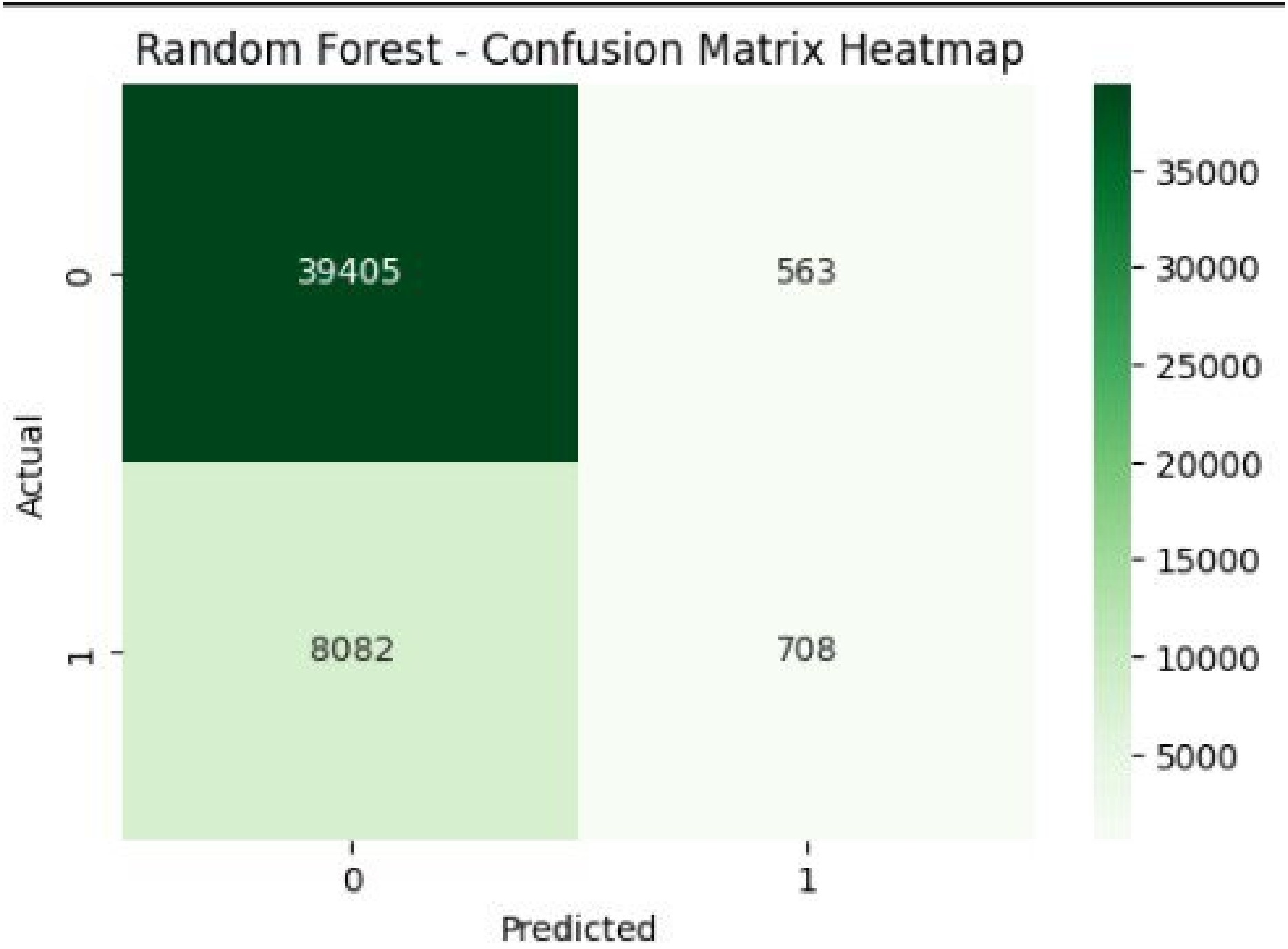
```
Decision Tree Accuracy: 0.7271627220148489

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.82      0.83     39968
           1       0.27      0.30      0.28      8790

    accuracy                           0.73     48758
   macro avg       0.56      0.56      0.56     48758
weighted avg       0.74      0.73      0.73     48758


Decision Tree Confusion Matrix:
 [[32838  7130]
 [ 6173  2617]]
```


Decision Tree - Confusion Matrix Heatmap
32838
7130
6173
2617
0
1
Actual
0
1
Predicted
30000
25000
20000
15000
10000
5000

Random Forest

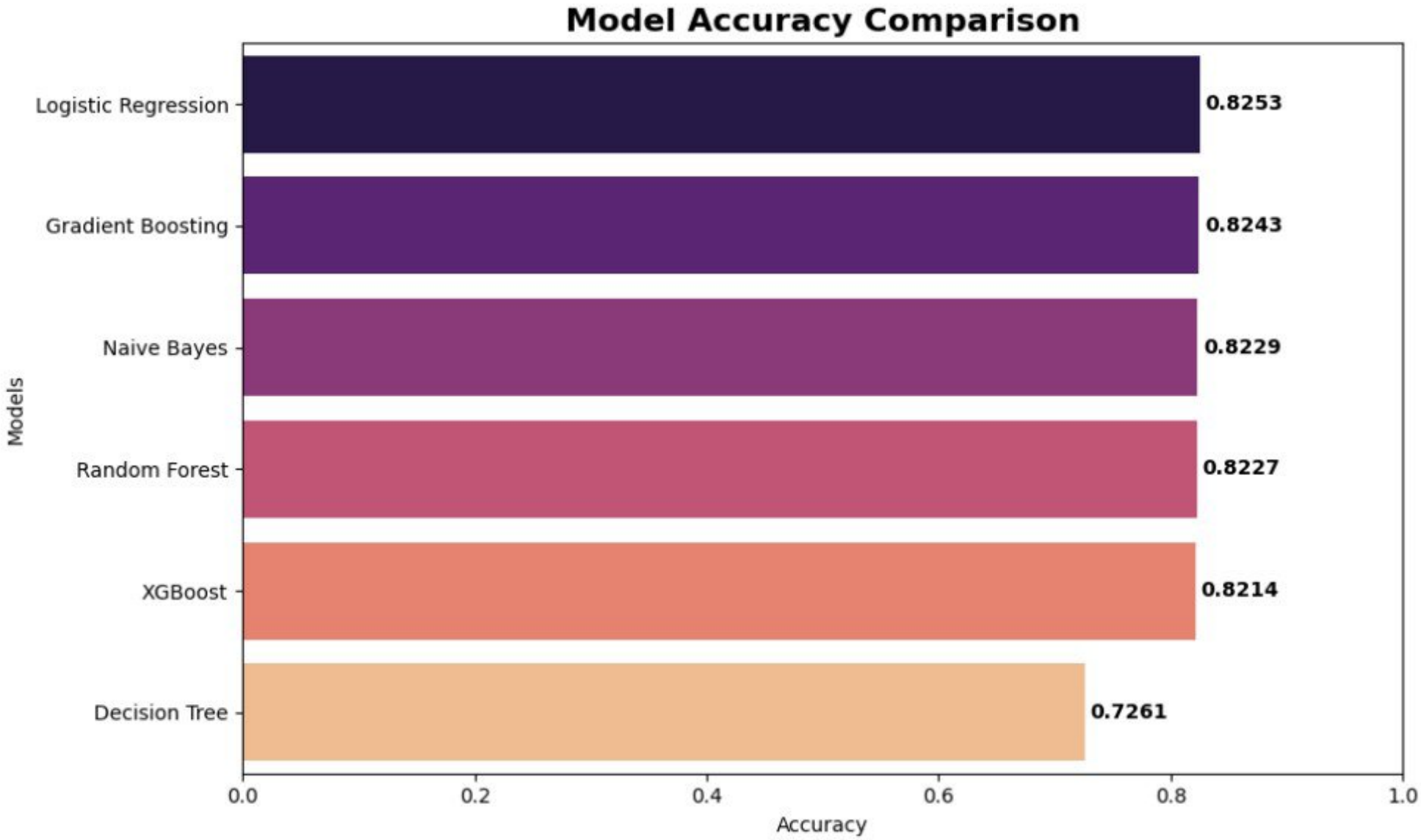
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
print("Classification Report: ", classification_report(y_test, rf_pred))
print("\Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))

cm = confusion_matrix(y_test, rf_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title("Random Forest - Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Random Forest Accuracy: 0.8226957627466261
Classification Report:                precision    recall  f1-score   support

           0       0.83      0.99      0.90     39968
           1       0.56      0.08      0.14      8790

    accuracy                           0.82     48758
   macro avg       0.69      0.53      0.52     48758
weighted avg       0.78      0.82      0.76     48758

\Confusion Matrix:
 [[39405   563]
 [ 8082   708]]
```


Random Forest - Confusion Matrix Heatmap
39405
563
8082
708
0
1
Actual
0
1
Predicted
35000
30000
25000
20000
15000
10000
5000

Model Accuracy comparison

```
accuracy_results = {
    "Logistic Regression": 0.8253004635136798,
    "Random Forest": 0.8226957627466261,
    "Gradient Boosting" : 0.8243365191353214,
    "Naive Bayes": 0.8229418762049304,
    "XGBoost": 0.8214241765453875,
    "Decision Tree": 0.726096230362197
}

accuracy_results = dict(sorted(accuracy_results.items(), key=lambda item: item[1], reverse=True))

# Plotting horizontal bar chart
plt.figure(figsize=(10,6))
sns.barplot(x=list(accuracy_results.values()), y=list(accuracy_results.keys()), palette="magma")

# Add accuracy labels on bars
for i, (model, acc) in enumerate(accuracy_results.items()):
    plt.text(acc + 0.005, i, f"{acc:.4f}", va='center', fontweight='bold')

# Find best model
best_model = max(accuracy_results, key=accuracy_results.get)
best_accuracy = accuracy_results[best_model]

plt.xlim(0,1)
plt.title("Model Accuracy Comparison", fontsize=16, fontweight='bold')
plt.xlabel("Accuracy")
plt.ylabel("Models")
plt.figtext(0.5, -0.05, f" Best Model: {best_model} with Accuracy = {accuracy_results[best_model]:.4f}",
            ha='center', fontsize=16, fontweight='bold', color='black')
plt.tight_layout()
plt.show()
```



Best Model: Logistic Regression with Accuracy = 0.8253

```
plt.figure(figsize=(12,6))
models = list(accuracy_results.keys())
acc = list(accuracy_results.values())

plt.plot(models, acc, marker='o', linewidth=2, color='green')
plt.fill_between(models, acc, color='lightgreen', alpha=0.5)
plt.ylim(0,1)
plt.title("Model Accuracy - Area + Line Chart", fontsize=16)
plt.ylabel("Accuracy")
plt.grid(alpha=0.7)
plt.tight_layout()
plt.show()
```