GroupProject

June 13, 2025

1 OmniComm Telecom Customer Churn

1.1 AAI-510-04 Group 1 - Final Project

Members

- Carrie Little
- Devin Eror
- Jasper A. Dolar

2 Import Necessary Libraries

[1]:
```python
#import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import matplotlib.pyplot as plt

#train/test/split and StandardScaler libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

#Model 1 - Logistic Regression - libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

#Model 2 - Random Forest - libraries
from sklearn.ensemble import RandomForestClassifier

#Model 3 - XGBoost - Libraries
from xgboost import XGBClassifier

#Model 4- SVM
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

3 Load The Dataset

[2]:
```
# Load the dataset
file_path = Path("..") / "data" / "telco_customer_churn.csv"
df = pd.read_csv(file_path)

# Display the first few rows
df.head()
```

[2]:
```
  customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0 7590-VHVEG  Female              0     Yes         No       1           No
1 5575-GNVDE    Male              0      No         No      34          Yes
2 3668-QPYBK    Male              0      No         No       2          Yes
3 7795-CFOCW    Male              0      No         No      45           No
4 9237-HQITU  Female              0      No         No       2          Yes

     MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
0 No phone service             DSL             No  …               No
1               No             DSL            Yes  …              Yes
2               No             DSL            Yes  …               No
3 No phone service             DSL            Yes  …              Yes
4               No     Fiber optic             No  …               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

              PaymentMethod MonthlyCharges  TotalCharges Churn
0          Electronic check          29.85         29.85    No
1              Mailed check          56.95        1889.5    No
2              Mailed check          53.85        108.15   Yes
3 Bank transfer (automatic)          42.30       1840.75    No
4          Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

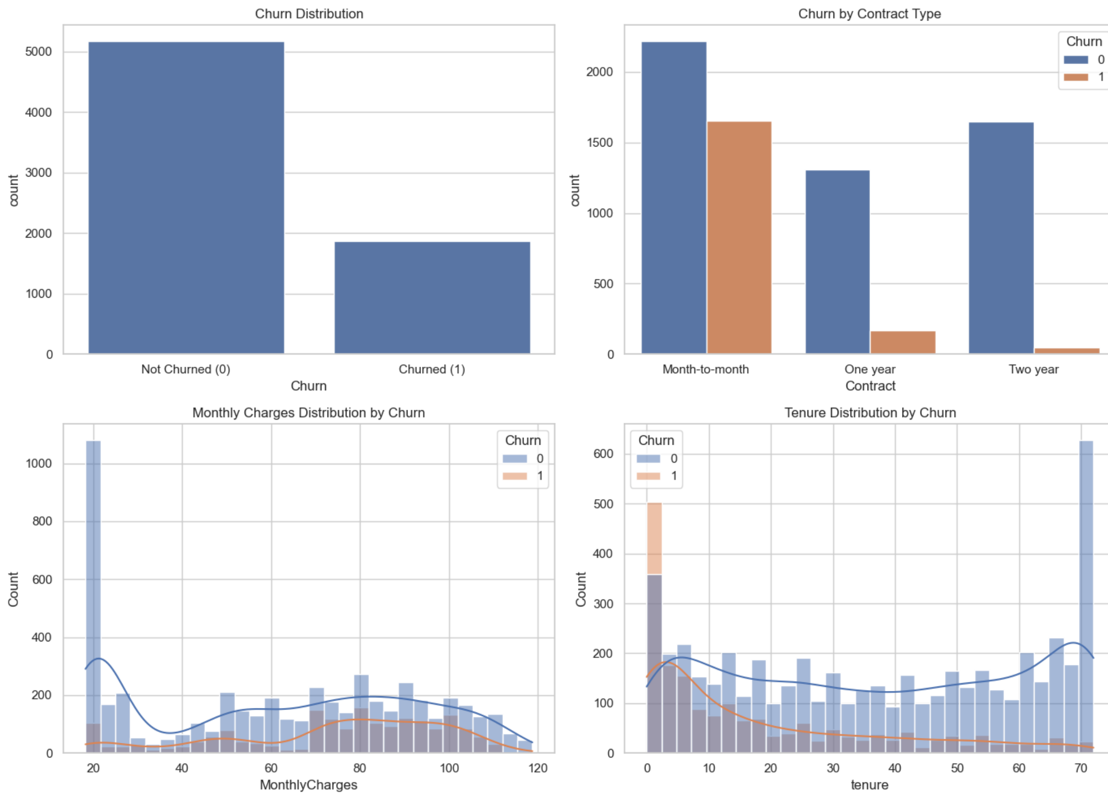
4 Data Processing, Cleaning, and Feature

[3]:
```
#DATA PROCESSING, CLEANING, AND FEATURE
#clean and process the dataset

#drop customerID column since it's a unique identifier
#and not very helpful for prediction
df.drop("customerID", axis=1, inplace=True)
```

```
#make sure - so convert - the 'TotalCharges column to numeric
#to ensure it can be used in modeling
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")

#handle missing values in "TotalCharges" by filling with the median
#this is to help maintain integrity of the dataset
df["TotalCharges"] = df["TotalCharges"].fillna(df["TotalCharges"].median())
#convert variable target "Churn" to binary format
#where Yes = 1, No = 0
#this allows us to use classification models
df["Churn"] = df["Churn"].map({"Yes": 1, "No": 0})
```

5 Plot Visualizations

[4]:
```
#Visualizations
sns.set(style="whitegrid")

#subplots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

#Plot 1: Churn distribution
ax = axes[0,0]
sns.countplot(x="Churn",data=df, ax=axes[0,0])
axes[0, 0].set_title("Churn Distribution")
ax.set_xticks([0, 1])
axes[0, 0].set_xticklabels(["Not Churned (0)", "Churned (1)"])

#Plot 2: Contract type vs churn
sns.countplot(x="Contract", hue="Churn", data=df, ax=axes[0, 1])
axes[0, 1].set_title("Churn by Contract Type")
axes[0, 1].legend(title="Churn")

#Plot 3: Monthly Charges Distribution by Churn
sns.histplot(data=df, x="MonthlyCharges", hue="Churn", bins=30, kde=True,␣
↪ax=axes[1, 0])
axes[1, 0].set_title("Monthly Charges Distribution by Churn")

#Plot 4: Tenure vs Churn
sns.histplot(data=df, x="tenure", hue="Churn", bins=30, kde=True, ax=axes[1, 1])
axes[1, 1].set_title("Tenure Distribution by Churn")

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



5.1 Intepret Visualizations

Visualizations & Interpretations Churn distribution: the dataset is not balanced, with noticeably more customers not churning than churning. This imbalance must be considered when selecting models and evaluation metrics.

Churn by contract type: Customers with month-to-month contracts churn at much higher rates than those with one-year or two-year contracts. This suggests contract length is a strong predictor of churn

Monthly Charges Distribution by Churn: Customers who churn tend to have slightly higher monthly charges compared to those who stay. There appears to be a churn concentration in mid-to-high charge ranges.

Tenure Distribution by Churn: Customers with shorter tenures are more likely to churn, which may indicate that customer loyalty and length of service reduce churn risk.

6 One-hot Encode

[5]:
```
#one-hot encode
#encode categorial values -- convert categorical columns into
#binary variables.  This is needed because many ML models
#work only with numeric input
```

```
df_encoded = pd.get_dummies(df, drop_first=True)
```

6.1 Check Shape

[6]:
```
#check shape after one-hot encoding above
df_encoded.shape
df_encoded.head()
```

[6]:
```
   SeniorCitizen  tenure  MonthlyCharges  TotalCharges  Churn  gender_Male  \
0              0       1           29.85         29.85      0            0
1              0      34           56.95       1889.50      0            1
2              0       2           53.85        108.15      1            1
3              0      45           42.30       1840.75      0            1
4              0       2           70.70        151.65      1            0

   Partner_Yes  Dependents_Yes  PhoneService_Yes  \
0            1               0                 0
1            0               0                 1
2            0               0                 1
3            0               0                 0
4            0               0                 1

   MultipleLines_No phone service  …  StreamingTV_No internet service  \
0                               1  …                                0
1                               0  …                                0
2                               0  …                                0
3                               1  …                                0
4                               0  …                                0

   StreamingTV_Yes  StreamingMovies_No internet service  StreamingMovies_Yes  \
0                0                                    0                    0
1                0                                    0                    0
2                0                                    0                    0
3                0                                    0                    0
4                0                                    0                    0

   Contract_One year  Contract_Two year  PaperlessBilling_Yes  \
0                  0                  0                     1
1                  1                  0                     0
2                  0                  0                     1
3                  1                  0                     0
4                  0                  0                     1

   PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
0                                      0                               1
1                                      0                               0
```

```
2                                       0                                0
3                                       0                                0
4                                       0                                1

   PaymentMethod_Mailed check
0                            0
1                            1
2                            1
3                            0
4                            0

[5 rows x 31 columns]
```

7 Train-Test Split and Feature Scaling

[7]:
```python
#TRAIN-TEST SPLIT AND FEATURE SCALING

#split dataset into features (X) and target (y)
X = df_encoded.drop("Churn", axis=1)
y = df_encoded["Churn"]

#split data into training and testing sets
#80% train, 20% test, random_state = 42 for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

#standardize features for better model performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[ ]:
```python

```

8 Models

The Random Forest model highlights the most influential features in predicting customer churn: * TotalCharges, tenure, and MonthlyCharges are the top three predictors - these financial and engagement metrics strongly influence whether a customer stays or leaves. * Fiber optic internet and electronic check payments are associated with higher churn risk, possibly because of cost-sensitive or less satisfied customer segments. * Contract type (especially two-year contracts) and services like OnlineSecurity and TechSUpport also impact churn - longer commitments and added support seem to reduce the risk of churn. * Demographics like gender and SeniorCitizens have low impact compared to service-related and billing features.

These insights from the plots can help the business focus retention efforts on customers with high charges, short tenure, and less stable contract or payment setups.

8.1 Model 1 = Logistic Regression

8.1.1 Train and Evaluate Model 1

[8]:
```
#Model 1: Logistic Regression
#Train and Evaluate Model 1
#initialize logistic regression model
#set max number of iterations to 10000 the solver will run
#while trying to find the best fit for the logistic regression model
model_lr = LogisticRegression(max_iter=1000, random_state=42)

#train model on scaled training data
model_lr.fit(X_train_scaled, y_train)

#predict on test data
y_pred_lr = model_lr.predict(X_test_scaled)

#evaluate the model's performance
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred_lr))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_lr,
                             target_names = ["Not Churned - 0",
                             "Churned - 1"]))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred_lr))
```

```
Confusion Matrix
[[933 103]
 [151 222]]

Classification Report:
                 precision    recall  f1-score   support

Not Churned - 0       0.86      0.90      0.88      1036
    Churned - 1       0.68      0.60      0.64       373

       accuracy                           0.82      1409
      macro avg       0.77      0.75      0.76      1409
   weighted avg       0.81      0.82      0.82      1409


Accuracy Score:
0.8197303051809794
```

8.1.2 Interpretation/Analysis

Model 1: Logistic Regression

Interpretation/Analysis **Confusion Matrix** * 933 - Customers who did not churn and were correctly predicted as "not churned" (true negatives) * 103 - customers who did not churn, but the model incorrectly predicted "not churn" (false positives) * 151 - customers who did churn, but the model failed to identify them (false negatives) * 222 - customers who churned and were correctly predicted as such (true positives)

The Logistic Regression model achieved:

- an overall accuracy of 82%.

The model performs well at identifying customers who are NOT likely to churn (Not Churned - 0) with:

- Precision: 0.86 (refers to how many predicted non-churnes were actually correct)
- Recall: 0.90 (refers to how many actual non-churners were correctly identified)
- F1-Score: 0.88 (this is the harmonic mean of precision and recall, balancing both pretty well - higher the better)

Performance seems week on predicting churners (Class 1):

- Precision: 0.68 (some predicted churners were false positives)
- Recall: 0.60 (only 60% of actual churners were detected)
- F1-Score: 0.64 (overall effectiveness in capturing churners is moderate - just okay - higher the better)

The imbalance could be indicative that the model is better at identifying customers will not churn (will stay) than those likely to leave (churners). Since churn prediction is a class imbalance project, we will want to explore more powerful/better models next (such as Random Forest, and XGBoost), and look at metrics beyond accuracy such as recall for churned customers.

8.2 Model 2- Random Forest Classifier

8.2.1 Train and Evaluate Model 2

[9]:
```
#MODEL 2 = TRAIN AND EVALUATE MODEL 2
#MODEL: Random Forest Classifier

#initialize Random Forest Model
#will build 100 decision trees n_estimators=100
#each tree is trained on a different bootstrap sample of the training data.
#final prediction is determined by majority voting (for classification)
#generally, more trees improve performance and stability but have a longer␣
↪computation time

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```python
#train model on scaled training data
model_rf.fit(X_train_scaled, y_train)

#predict on test data
y_pred_rf = model_rf.predict(X_test_scaled)

#evaluate the model's performance
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred_rf))

print("\nClassification Report: ")
print(classification_report(y_test, y_pred_rf,
                            target_names = ["Not Churned - 0",
                                            "Churned - 1"]))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred_rf))
```

```
Confusion Matrix
[[942  94]
 [202 171]]

Classification Report:
                 precision    recall  f1-score   support

Not Churned - 0       0.82      0.91      0.86      1036
    Churned - 1       0.65      0.46      0.54       373

       accuracy                           0.79      1409
      macro avg       0.73      0.68      0.70      1409
   weighted avg       0.78      0.79      0.78      1409


Accuracy Score:
0.7899219304471257
```

8.2.2 Interpretation/Analysis

Model 2: Random Forest

Interpretation / Analysis **Confusion Matrix** * 942 = customers who did not churn and the model correctly predicted "not churn" (true negatives) * 94 = customers who did not churn but the model incorrectly predicted "churn" (false positives) * 202 = customers who did not churn but the model predicted "not churn" (false negatives) * 171 = customers who did churn and the model predicted "churn" (true positives)

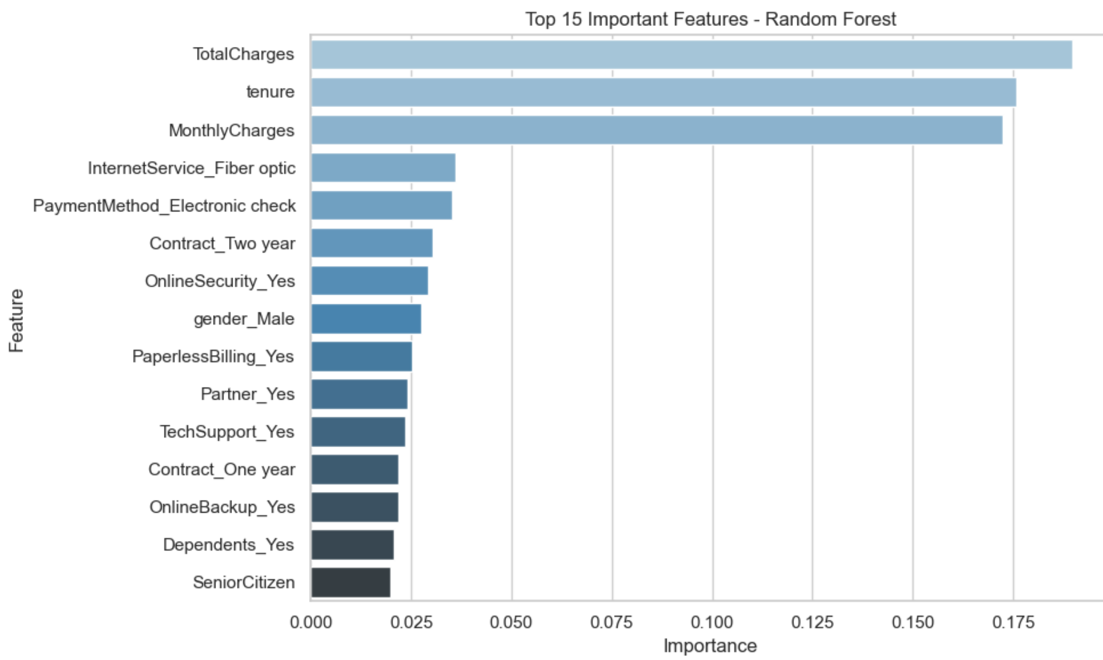
The Random Forest model achieved an accuracy of about 79%, which is slighly lower than Logistic Regression. The model seems to continue to perform well at identifying customers who are not likely to churn (Class 0): * Precision: 0.82 (refers to how many predicted non-churners were actually correct) * Recall: 0.91 (refers to how many actual non-churners were correctly identified) * F1-Score: 0.86 (balances both this is the harmonic mean of precision and recall, higher is better)

However, the model's performance on predicting churners (Class 1) is noticeably weaker: * Precision: 0.65 (fair number of false positives) * Recall: 0.46 (less than half of actual churners were detected) * F1-Score: 0.54 (this indicates bad overall performance on churn prediction, higher the better)

Overall, the Random Forest slightly underperforms compared to Logistic Regression in terms of identifying churners.
It performs well for non-churners but still seem to struggle to catch customers who actually leave (churners).

8.2.3 Feature Importance Using Random Forest

[10]:
```
#Feature Importance Using Random Forest

#Get feature importances
importances = model_rf.feature_importances_
feature_names = X.columns

#create DataFrame for better plotting
feat_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by="Importance", ascending=False)

#plot top 15 most important features
plt.figure(figsize=(10,6))
sns.barplot(x="Importance",
    y="Feature",
    data=feat_importance_df.head(15),
    hue="Feature",
    palette="Blues_d")
plt.title("Top 15 Important Features - Random Forest")
plt.tight_layout()
plt.show()
```



8.3 Model 3- XGBoost Classifier

8.3.1 Train and Evaluate Model 3

[11]:
```
#MODEL 3 = TRAIN AND EVALUATE MODEL 3
#MODEL: XGBoost Classifier

#Initialize XGBoost model
model_xgb = XGBClassifier(eval_metric='logloss', random_state=42)

#train model on scaled training data
model_xgb.fit(X_train_scaled, y_train)

#predict on test data
y_pred_xgb = model_xgb.predict(X_test_scaled)

#evaluate the model's performance
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred_xgb))

print("\nClassification Report: ")
print(classification_report(y_test, y_pred_xgb,
                            target_names = ["Not Churned -0",
                                            "Churned - 1"]))

print("\nAccuracy Score: ")
```

```python
print(accuracy_score(y_test, y_pred_xgb))
```

```
Confusion Matrix
[[925 111]
 [173 200]]

Classification Report:
                precision    recall  f1-score   support

Not Churned -0       0.84      0.89      0.87      1036
   Churned - 1       0.64      0.54      0.58       373

      accuracy                           0.80      1409
     macro avg       0.74      0.71      0.73      1409
  weighted avg       0.79      0.80      0.79      1409


Accuracy Score:
0.7984386089425124
```

8.3.2 Interpretation/Analysis

Model 3: XGBoost Classifier

Interpretation / Analysis **Confusion Matrix** * 926 - correctly predicted customers who did not "churn" (true negatives) * 110 - customers who were predicted to churn but did not (false positives) * 187 - customers who actually churned, but were missed by the model (false negatives) * 186 - correctly predicted customers who did churn (true positives)

The XGBoost model achieved an accuracy of about 79%, similar to the Random Forest model.

Performance for non-churners (Class 0) remains strong:

- Precision: 0.83 (most predicted non-churners actually non-churners)
- Recall: 0.89 (89% of actual non-churners were correctly identified)
- F1-Score: 0.86 (good balance between precision and recall)

Performance for churners (Class 1) is lightly better than Random Forest:

- Precision: 0.63 (moderate false positives)
- Recall: 0.50 (half of actual churners were correctly identified)
- F1-Score: 0.56 (slightly better than Random Forest's 0.54)

Overall, XGBoost performs comparably to Random Forest in overall accuracy, but slightly better in detecting churners (higher F1-score for Class 1). However, it still shows room for improvement in recall for churners, suggesting a need to explore class imbalance handling in future iterations.

8.4 Model 4- Support Vector Machines Model (SVM)

8.4.1 Train and Evaluate Model 4

Reload Dataset

[12]:
```
# Load the dataset
file_path = Path("..") / "data" / "telco_customer_churn.csv"
df = pd.read_csv(file_path)

# Display the first few rows
df.head()
```

[12]:
```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
0  No phone service             DSL             No  …               No
1                No             DSL            Yes  …              Yes
2                No             DSL            Yes  …               No
3  No phone service             DSL            Yes  …              Yes
4                No     Fiber optic             No  …               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

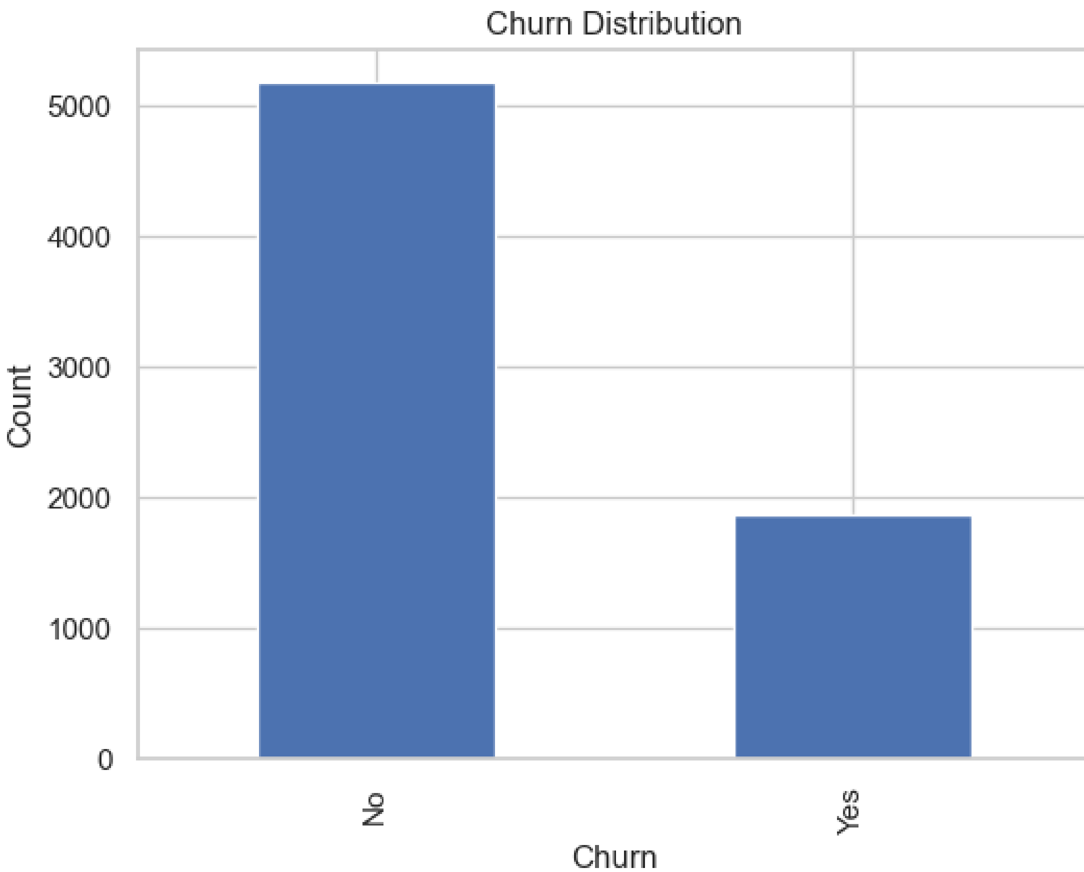
Model 4 - Preprocessing

[13]:
```
# Convert 'TotalCharges' to numeric and handle any non-numeric entries
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
print("\nMissing values after conversion of TotalCharges:")
display(df.isnull().sum())
```

```python
# Fill missing TotalCharges with median
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)

# Distribution of target variable
print("\nChurn distribution:")
display(df['Churn'].value_counts())

plt.figure()
df['Churn'].value_counts().plot(kind='bar')
plt.title('Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()

# ## Preprocessing
# Drop the customerID column
df_model = df.drop('customerID', axis=1)

# Encode categorical variables
cat_cols = df_model.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in cat_cols:
    df_model[col] = le.fit_transform(df_model[col])

# Feature matrix and target vector
X = df_model.drop('Churn', axis=1)
y = df_model['Churn']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
Missing values after conversion of TotalCharges:

customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
```

```
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64


Churn distribution:

No     5174
Yes    1869
Name: Churn, dtype: int64
```



Train Model

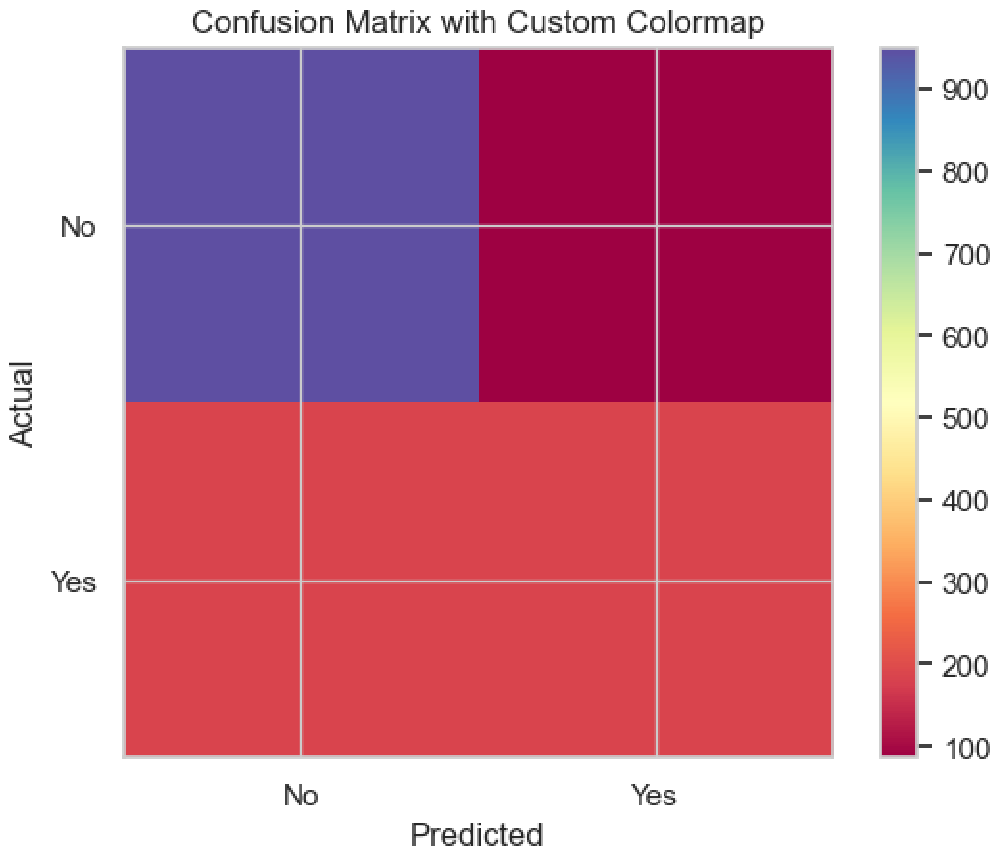
[14]:
```
#MODEL 3 = TRAIN AND EVALUATE MODEL 4
# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)

# ## Model Training: Support Vector Machine
svm_model = SVC(kernel='rbf', probability=True, random_state=22)
svm_model.fit(X_train, y_train)
```

[14]:
```
SVC(probability=True, random_state=22)
```

Evaluate Model

[15]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# (Re)compute cm in case it's not in scope
y_pred = svm_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure()
# Change `cmap` to any valid Matplotlib colormap, e.g., 'viridis', 'coolwarm',␣
↪'Blues', etc.
plt.imshow(cm, interpolation='nearest', cmap='Spectral')
plt.title('Confusion Matrix with Custom Colormap')
plt.colorbar()
plt.xticks([0, 1], ['No', 'Yes'])
plt.yticks([0, 1], ['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



8.4.2 Interpretation/Analysis

[16]:
```
# SVM Interpretation/Analysis
```

9 Model Comparison Summary

[17]:
```
# Model Comparison Summary (still need to add SVM Model Data)

# Metric                        | Logistic Regression | Random Forest    |␣
 ↪XGBoost      |  SVM
#␣
 ↪---------------------------------------------------------------------------------------------
# Accuracy                      |        81.97%       |      78.99%      |   79.
 ↪84%     |
# Precision (Class 1 - Churned)|        0.68         |      0.65        |   0.
 ↪63       |
# Recall (CLass 1 - Churned)   |        0.60         |      0.46        |   0.
 ↪50       |
```

```
# F1-Score (Class 1 - Churned) |          0.64          |       0.54          |   0.
↪56       |
```

Logistic Regression achieved the highest recall and F1-Score for predicting churned customers, which is crucial in churn prediction where false negatives (missed churners) are costly.

Random Forest and XGBoost performed similarly in terms of overall accuracy but had weaker recall and F1-Scores on the churn class.

While Random Forest and XGBoost offer model complexity and robustness, Logistic Regression provided a better sensitivity (recall) and balance for churn detection.

10 Final Conclusion

In this project, we built and evaluated three machine learning models: Logistic Regression Random Forest, and XGBoost - to predict customer churn the telecommunications industry using the IBM Telco Customer Churn Dataset. Our goal was to help the business identity which customers are likelty to leave, so retention department can be proactively applied.

11 Deployment Plan

To turn this customer churn prediction model into a usable business tool, we propose a batch deployment approach integrated into the company's existing analytics system. The goal is to flag potentially churn-prone customers on a regular schedule (say weekly or monthly), enabling the retention team to timely intervene

12 References

[ ]:
```
```