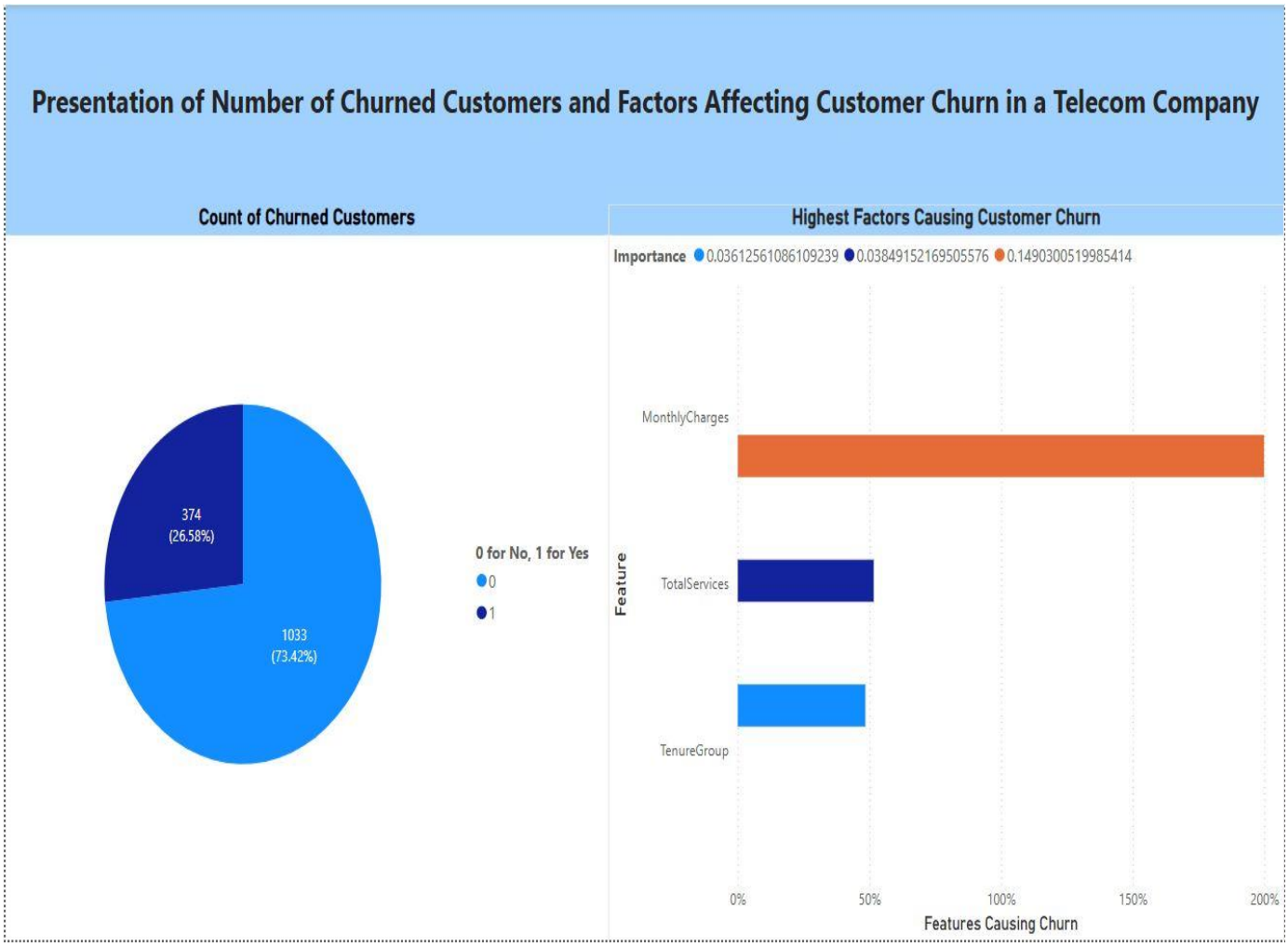
TELECOM COMPANY CUSTOMER CHURN



Began the project by importing all the necessary libraries and importing the raw dataset.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
```

```
df = pd.read_csv(r"C:\Users\PC\Desktop\TelecomProject\CustomerChurnRawData.csv")
```

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	St
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	

DATA CLEANING.

Began the data cleaning process by checking and clearing duplicates and null values, then changing data types which needed changing.

DATA CLEANING

```
### detecting duplicates

duplicates = df.duplicated()

duplicates.sum()
df = df.drop_duplicates()
```

```
### detecting Null and Missing Values

missing_values = ["NA", "", None, np.NaN]
missing = df.isin(missing_values)
missing.sum().sum()
```

```
0
```

```
### checking the total number of Null values. This should be used in a large dataset to confirm if there are Nulls. And then deleting the null values

df = df.dropna()
```

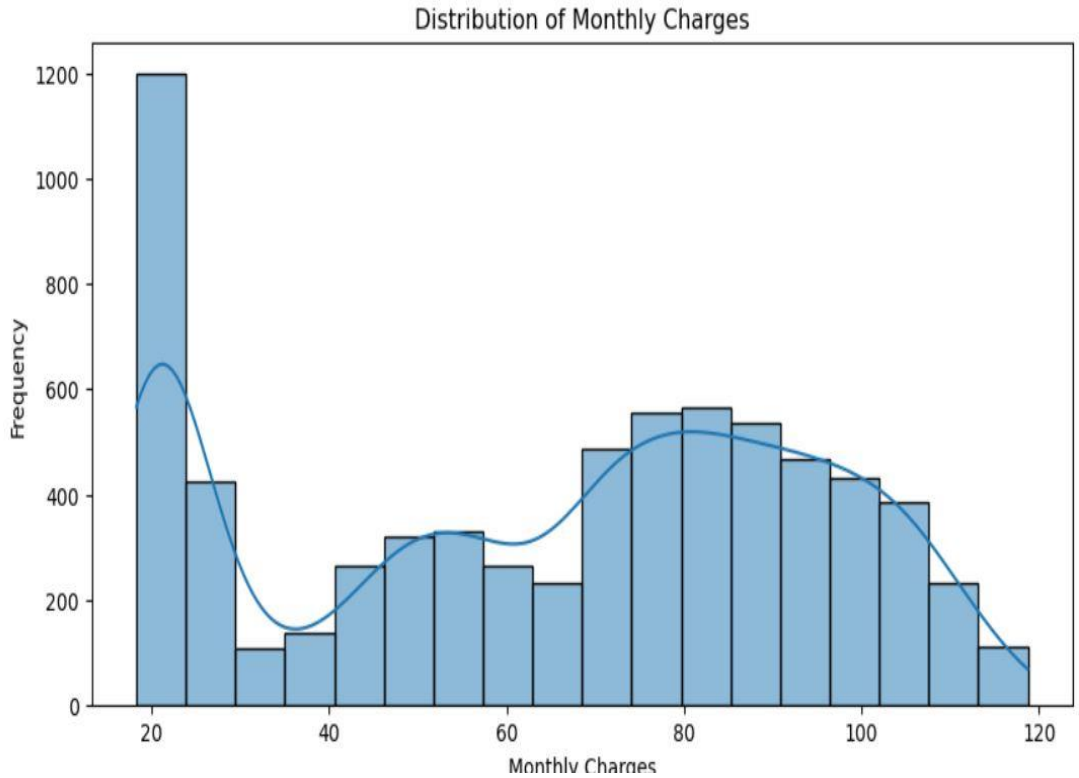
```
## changing the binary 0 and 1 to Yes and No

df["SeniorCitizen"] = df["SeniorCitizen"].astype('str').str.replace("0", "No")
df["SeniorCitizen"] = df["SeniorCitizen"].astype('str').str.replace("1", "Yes")
```

EXPLORATORY DATA ANALYSIS

While exploring the data, I carried out the following.

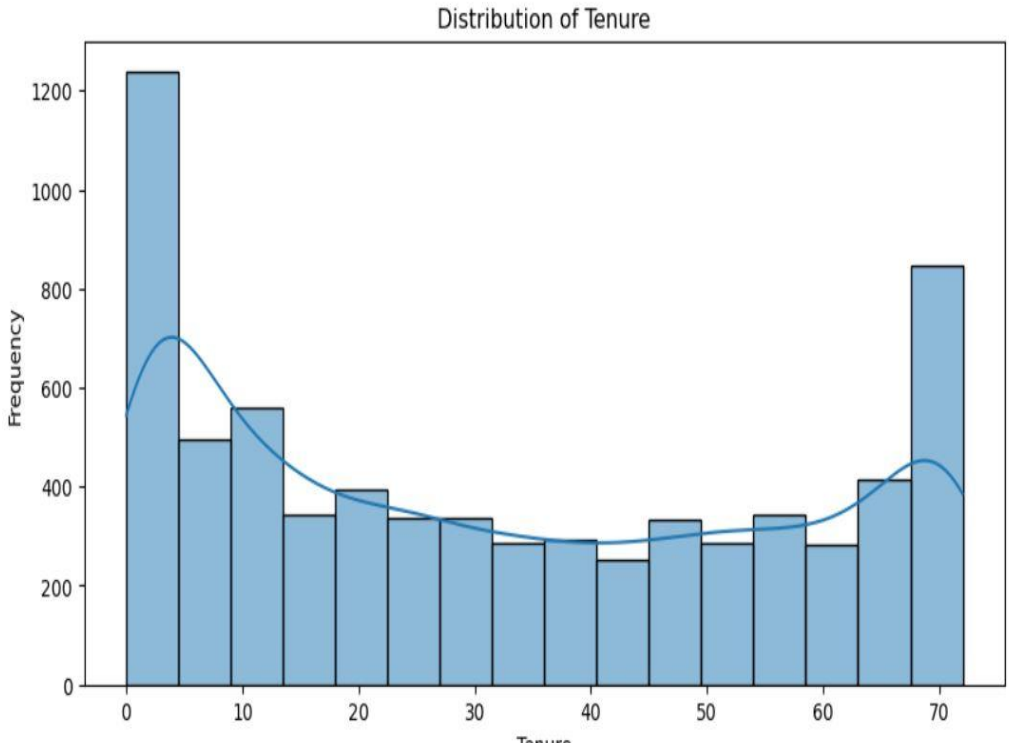
1. Checked the distribution of monthly charges using a histogram.

```
### Checking the distribution of Monthly Charges on the customers using a Histogram

plt.figure(figsize=(10, 5))
sns.histplot(df["MonthlyCharges"], kde=True)
plt.title("Distribution of Monthly Charges")
plt.xlabel("Monthly Charges")
plt.ylabel("Frequency")
plt.show()
```



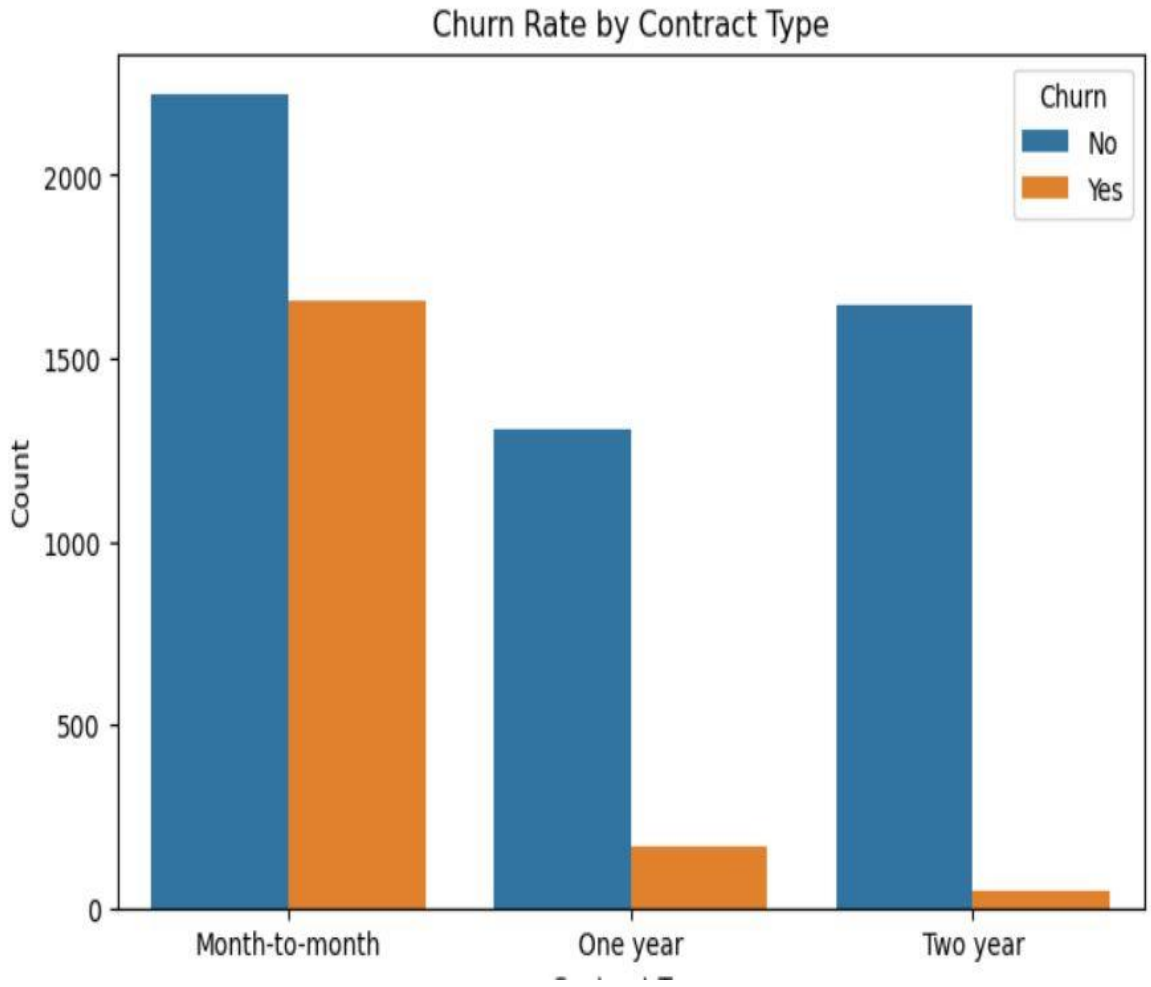
2. Checked the distribution of customer tenure.

```
### Checking the distribution of Customer Tenure with the Telco Company using a histogram

plt.figure(figsize=(10, 5))
sns.histplot(df['tenure'], kde=True)
plt.title("Distribution of Tenure")
plt.xlabel("Tenure")
plt.ylabel("Frequency")
plt.show()
```



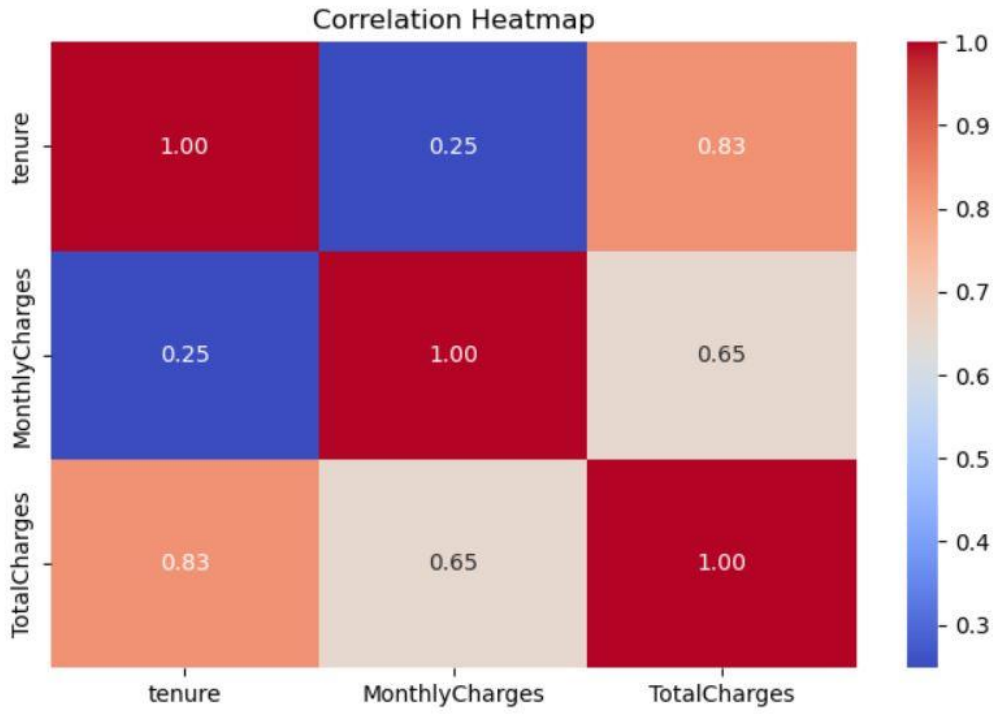
3. A count plot of contract type with the highest churn rates.

```
### Churn Rate per Contract Type


plt.figure(figsize=(8, 5))
sns.countplot(x=df['Contract'], hue=df['Churn'])
plt.title('Churn Rate by Contract Type')
plt.xlabel('Contract Type')
plt.ylabel('Count')
plt.legend(title='Churn', labels=['No', 'Yes'])
plt.show()
```



2. A correlation matrix for tenure, monthly charges, and total charges.

```
### Correlation Heatmap of Tenure, MonthlyCharges, and TotalCharges

correlation_matrix = df[['tenure', 'MonthlyCharges', 'TotalCharges']].corr()


plt.figure(figsize=(8, 5))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=True, fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



Feature Engineering.

Began the feature engineering process by gouping tenures and also grouping charges per month.

Feature Engineering

```
### Confirming that the entire dataframe does not contain nulls

df.isnull().sum().sum()
```

```
0
```

```
### Grouping the tenure based on duration

def tenure_group(tenure):
    if tenure <= 12:
        return '0-1 Year'
    elif tenure <=24:
        return '1-2 Years'
    elif tenure <=48:
        return '2-4 Years'
    else:
        return '4+ Years'


df['TenureGroup'] = df['tenure'].apply(tenure_group)
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_9360\3011640385.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['TenureGroup'] = df['tenure'].apply(tenure_group)
```

```
### Creating a Charge Per Month Column

df['ChargePerMonth'] = df['TotalCharges'] / (df['tenure'] + 1)
```

This was followed by getting the total services used by each customer, performing one-hot encoding on several columns, and dropping columns which won't be used for predictive analysis.

```
df['ChargePerMonth'] = df['TotalCharges'] / (df['tenure'] + 1)
```

```
### Creating a column for the total number of services used by a single customer

service_cols = ['PhoneService', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']

df['TotalServices'] = df[service_cols].apply(lambda x: sum(x == 'Yes'), axis=1)
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_9360\2763743354.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['TotalServices'] = df[service_cols].apply(lambda x: sum(x == 'Yes'), axis=1)
```

```
### One-Hot Encoding for Nominal Categories

df = pd.get_dummies(df, columns=['Contract', 'PaymentMethod', 'InternetService'], drop_first=True)
```

```
### Label Encoding for Tenure Group, which is an Ordinal Category

le = LabelEncoder()
df['TenureGroup'] = le.fit_transform(df['TenureGroup'])
```

```
# Removing unwanted columns so as to start creating predictive models

df = df.drop(columns=['SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'T
```

PREDICTIVE ANALYSIS.

Splitting the training and testing data in 80% and 20% respectively, and performing logistic regression, random forest, and XGBoost.

```
### Splitting Data into Train & Test Sets


x = df.drop(columns=['Churn', 'customerID'])

y = df['Churn']

# Splitting data (80% training, 20% testing)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Display dataset sizes

print('Training set size:', x_train.shape)
print('Testing set size:', x_test.shape)
```

```
## Logistic Regression


# training the model

log_model = LogisticRegression(max_iter=1000)
log_model.fit(x_train, y_train)

# making predictions

y_pred_log = log_model.predict(x_test)


# evaluating accuracy

accuracy_log = accuracy_score(y_test, y_pred_log)

print("Logistic Regression Accuracy:", accuracy_log)
```

```
Logistic Regression Accuracy: 0.7917555081734187
```

```
## Random Forest Classifier to handle Complex Relationships

# Train the model

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)

# make predictions

y_pred_rf = rf_model.predict(x_test)

# evaluate accuracy

accuracy_rf = accuracy_score(y_test, y_pred_rf)

print("Random Forest Accuracy:", accuracy_rf)
```

```
Random Forest Accuracy: 0.7512437810945274
```

```
## XGBoost (Optimized Gradient Boosting Model)

## Train the model

xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model.fit(x_train, y_train)

## make predictions

y_pred_xgb = xgb_model.predict(x_test)

## evaluate accuracy

accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print("XGBoost Accuracy:", accuracy_xgb)
```

```
C:\Users\PC\AppData\Roaming\Python\Python312\site-packages\xgboost\core.py:158: UserWarning: [22:07:44] WARNING: C:\buildkite-agent\builds\buildkite-win
ows-cpu-autoscaling-group-i-08cbc0333d8d4aae1-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost Accuracy: 0.7704335465529495
```

Performed Feature Importance on Random Forest and XGBoost due to the nature of the dataset, and converted the results of feature importance and predictions to CSV.

```
### Feature Importance of Random Forest

rf_importance = rf_model.feature_importances_
feature_names = x.columns

# Convert to DataFrame

importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': rf_importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)



importance_df.to_excel('RF_feature_importance.xlsx', index=False)
```

```
### Feature Importance from XGBoost

xgb_importance = pd.DataFrame({'Feature': x.columns, 'Importance': xgb_model.feature_importances_})
xgb_importance = xgb_importance.sort_values(by='Importance', ascending=False)

# Save to csv
xgb_importance.to_excel('xgb_feature_importance.xlsx', index=False)
```

```
# Save dataset with churn predictions

df_test_results = x_test.copy()
df_test_results['ActualChurn'] = y_test
df_test_results['PredictedChurn'] = y_pred_xgb # Best Model

df_test_results.to_excel('churn_predictions.xlsx', index=False)
print("Churn predictions saved for Tableau")
print(df_test_results)
```

```
Churn predictions saved for Tableau
```