TELECOM CHURN DEPI GRADUATION PROJET

OUR TEAM

MOHAMED AHMED

MOSTAFA MOHAMED

NADA ELGENDY

HASSAN ABDELSATTAR

ESLAM MOHAMED

TABLE OF CONTENT

INTRODUCTION AND BUSINESS PROBLEM

This project focuses on analyzing customer churn in a telecom company using customer demographics, account details, and service usage data. The goal is to identify patterns and key factors influencing churn. Customer churn leads to revenue loss and high acquisition costs, so the aim is to develop a predictive model that identifies at-risk customers, helping the company implement retention strategies to minimize churn.

DATA OVERVIEW

The dataset contains information about 7,043 telecom customers and their service usage patterns, demographics, and whether they churned or not. It has 21 columns, covering aspects like customer demographics, account details, services they have subscribed to, and their churn status.

```
df.dtypes

customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

DATA CLEANING

Missing Data Handling

From the initial data overview, we noticed that the TotalCharges feature was mistakenly stored as an object data type instead of numeric, likely due to missing or inconsistent values.

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce') #
↪coerce => If the data null will put NAN instead of error
```

DATA CLEANING

Handling Missing Values in TotalCharges

Upon further inspection, we found 11 rows with missing values in the TotalCharges column. Additionally, all of these customers had tenure values of 0 months, indicating they were in their first month of service. Given the small number of affected rows (11 out of 7043), we decided to drop these rows, as their removal will have minimal impact on the overall dataset.

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

```
MonthlyCharges       0
TotalCharges        11
Churn                0
```

```
# Drop rows where TotalCharges has missing values (NaN)
df.dropna(subset=['TotalCharges'], inplace=True)
```

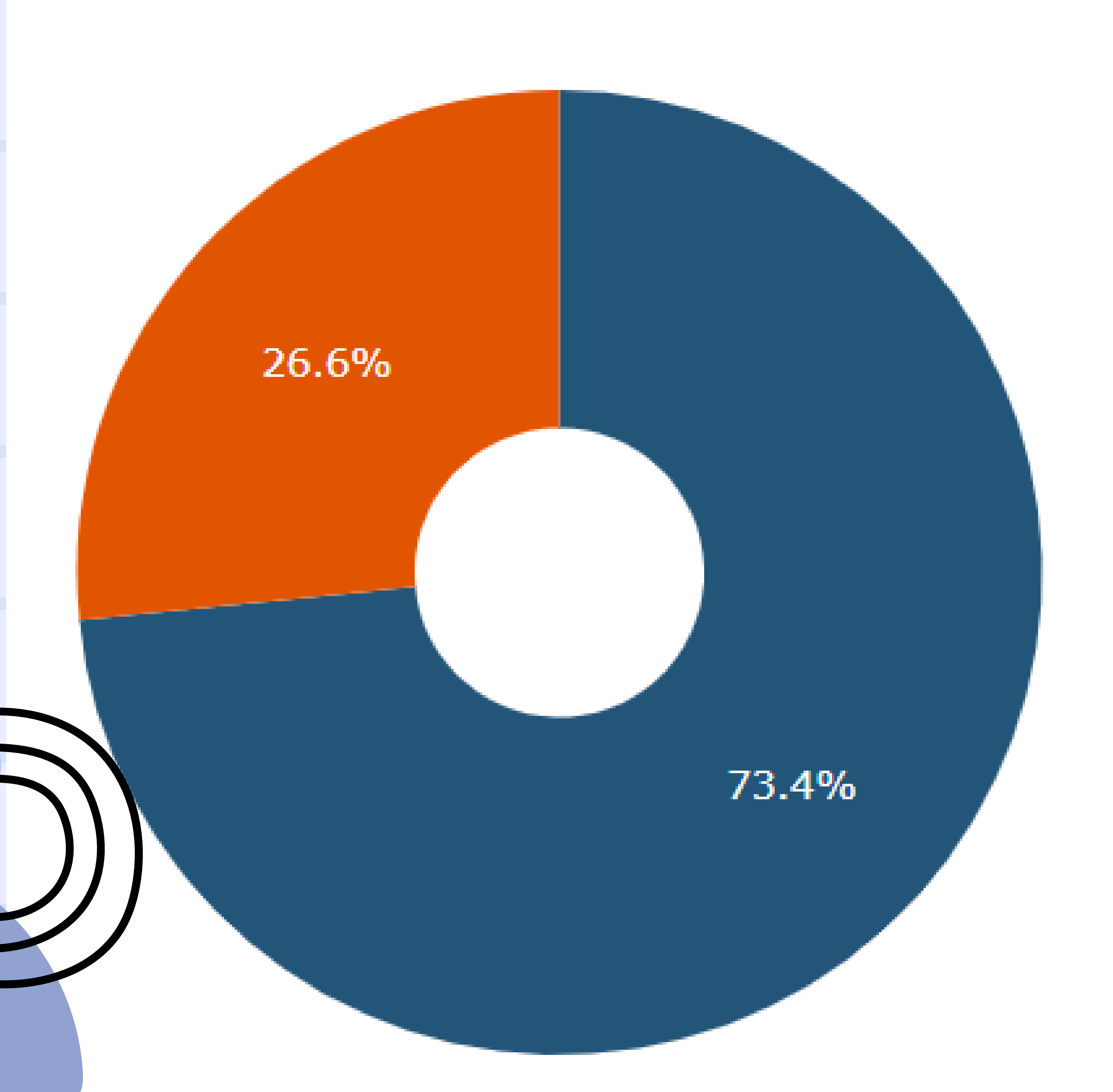
DATA CLEANING

Handling Duplicates

At this stage, we have not encountered any duplicate rows in the dataset. However, once we drop the customerID column, there is a possibility that duplicates may appear. We will carefully check for duplicates again after this step and remove any to maintain data integrity.

DATA EXPLORATION

Distribution of Target Variable (Churn)

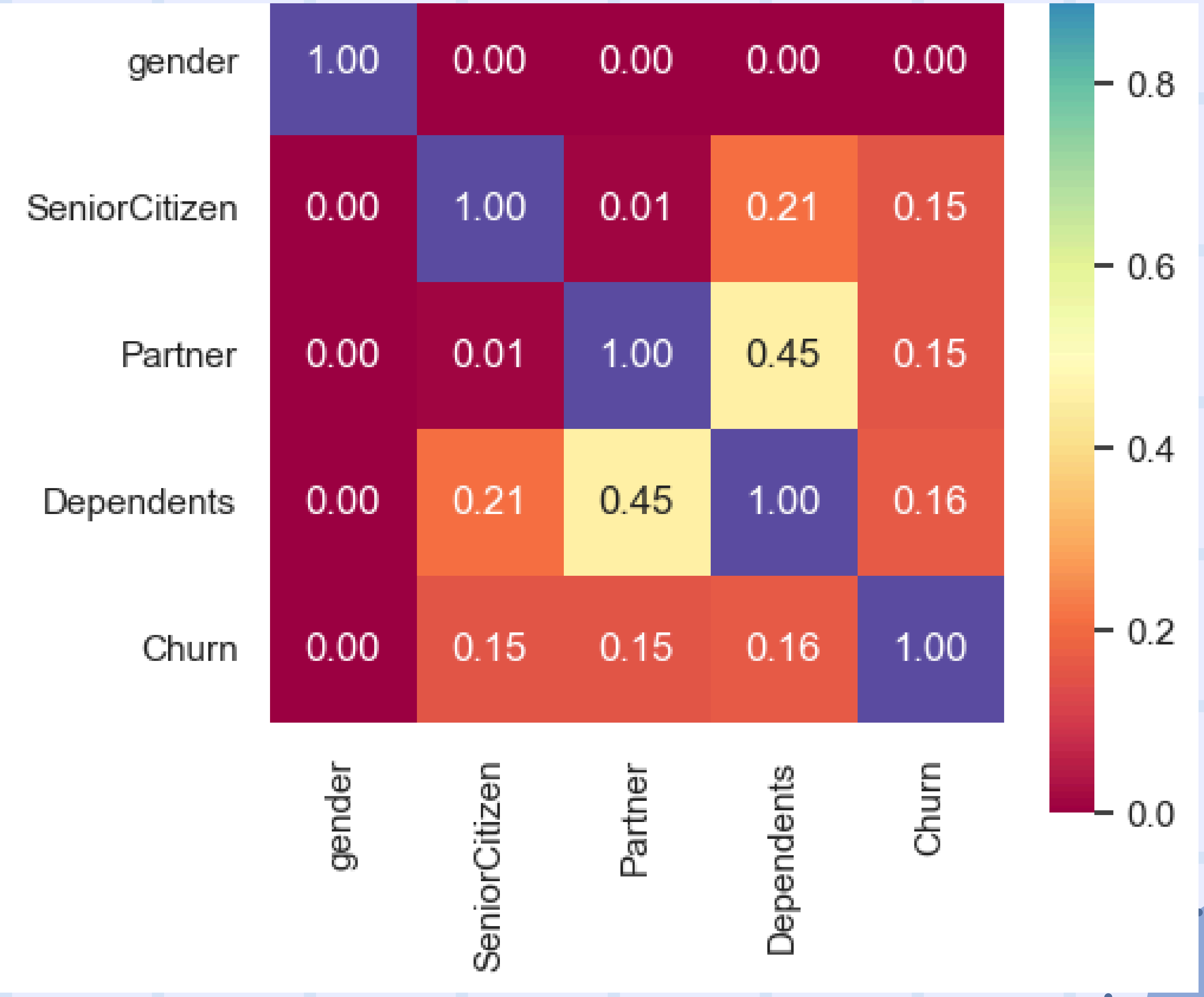


- he dataset is somewhat imbalanced, with a significantly higher number of customers who did not churn (73.4 %).
- This imbalance is common in customer churn datasets but We will have to take that into account while splitting the dataset.

DATA EXPLORATION

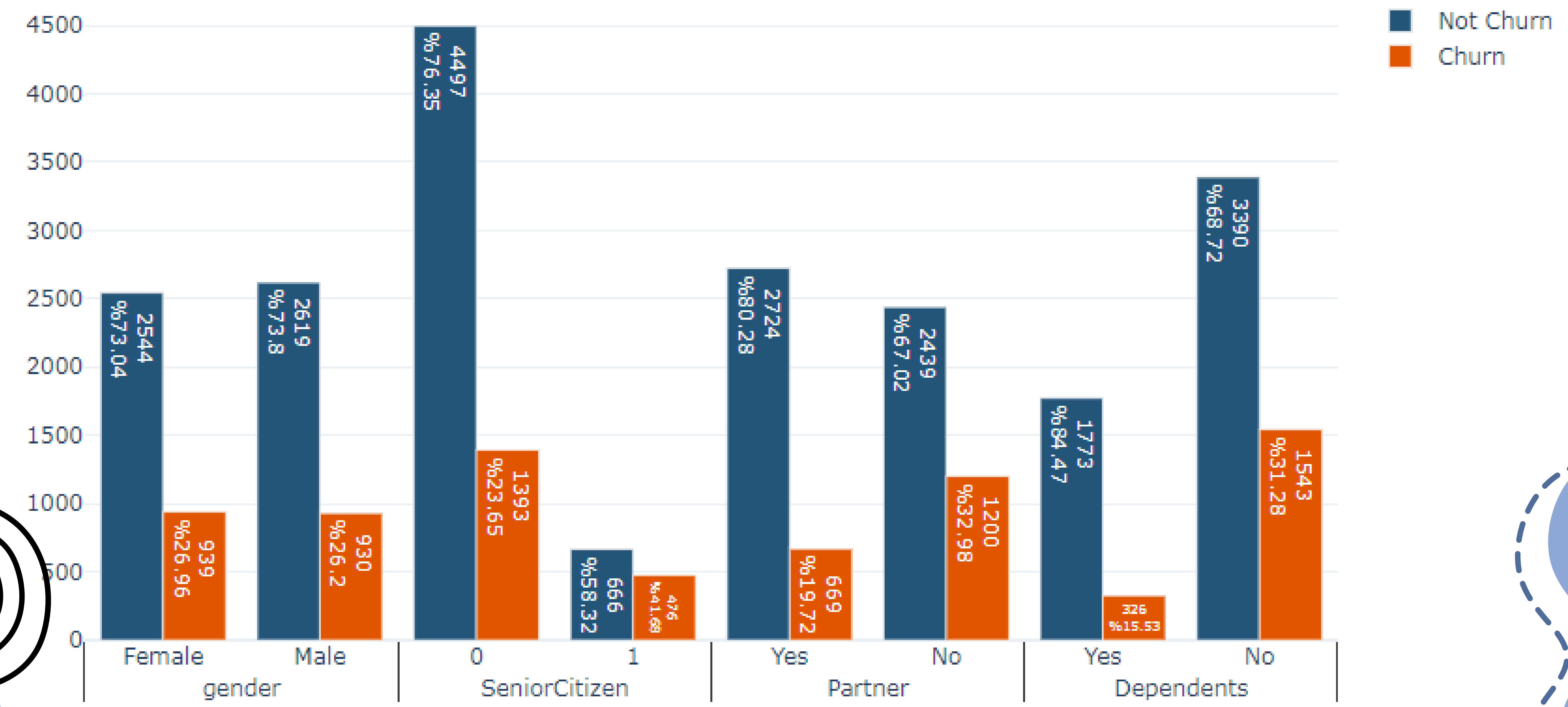
Customer Information

Analysis to discover the correlations and relation to churn



DATA EXPLORATION

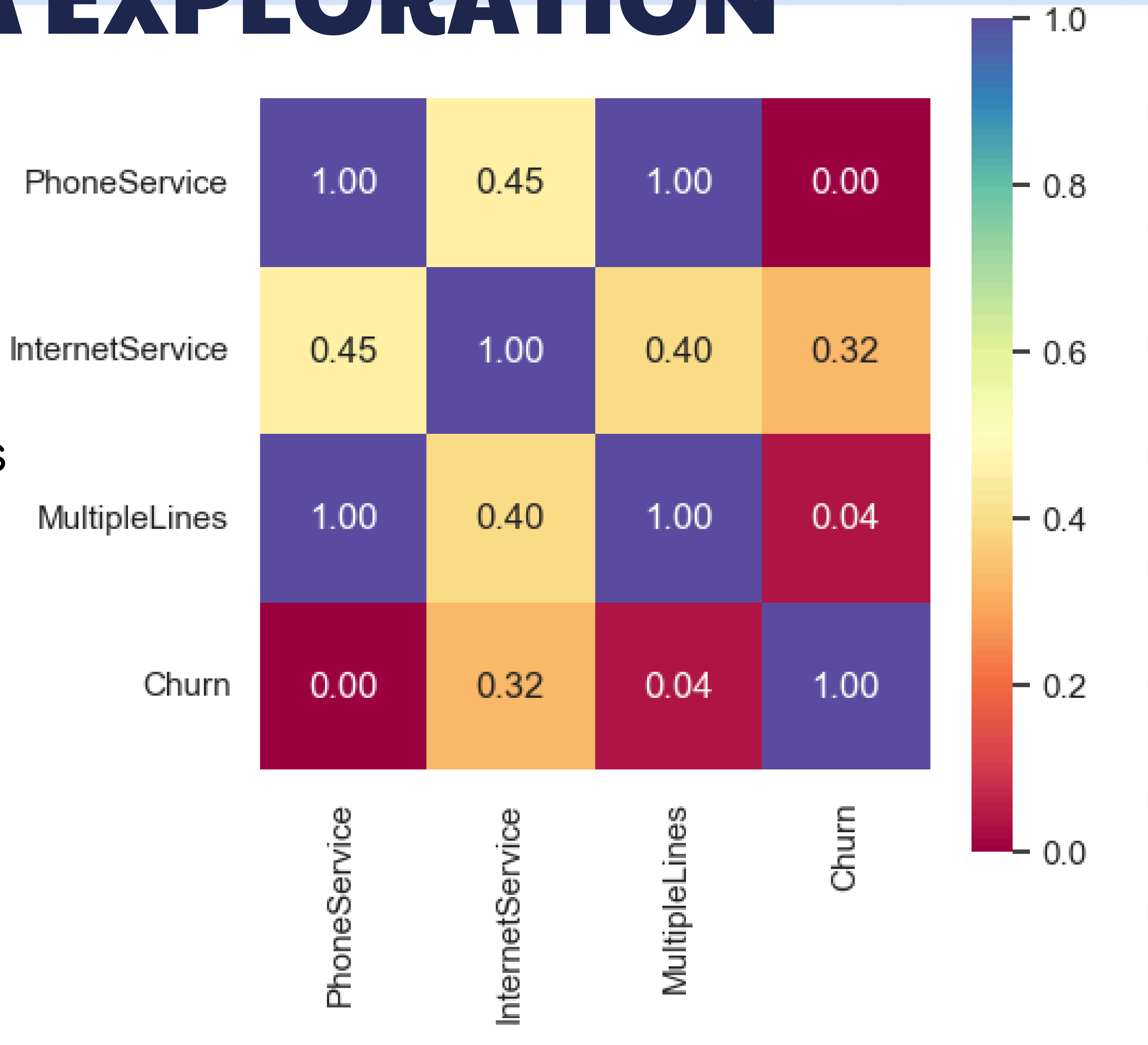
Customer Information



DATA EXPLORATION

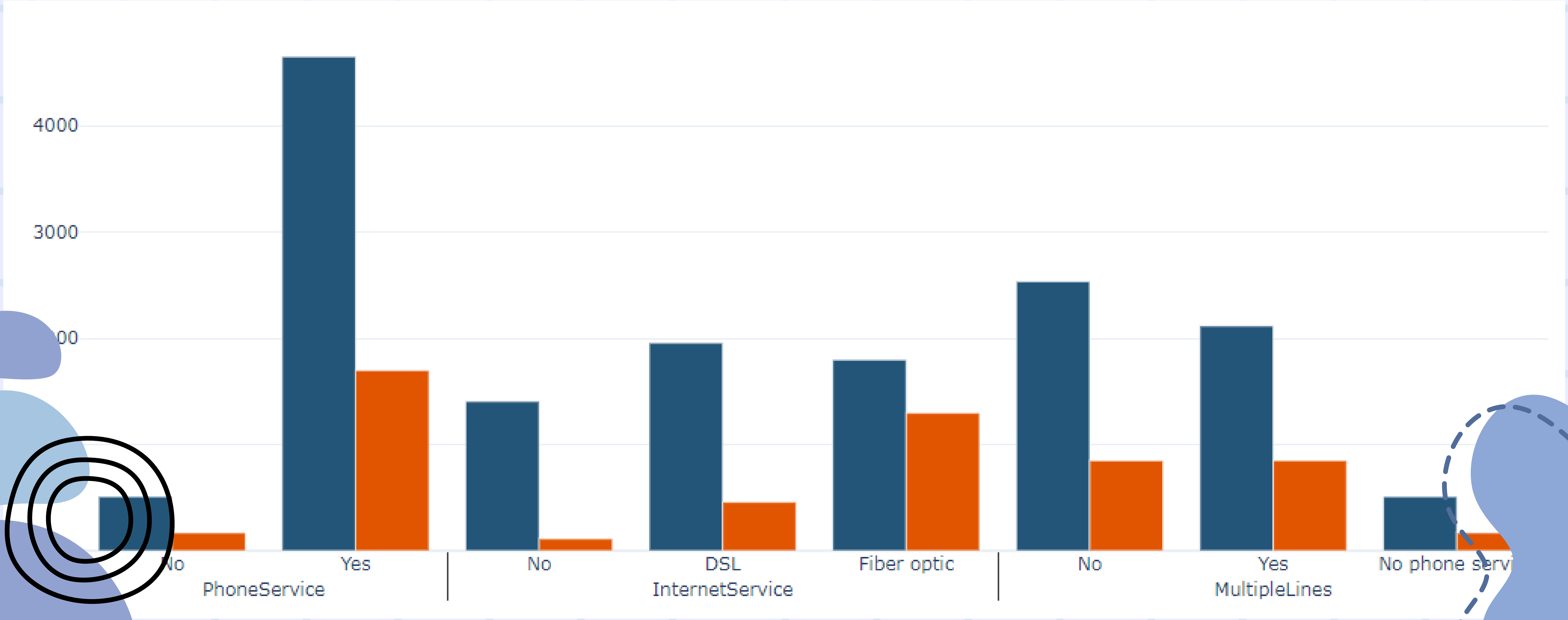
Service

This analysis refers to Service. We will analyze service information to discover the correlations and relation to churn.



DATA EXPLORATION

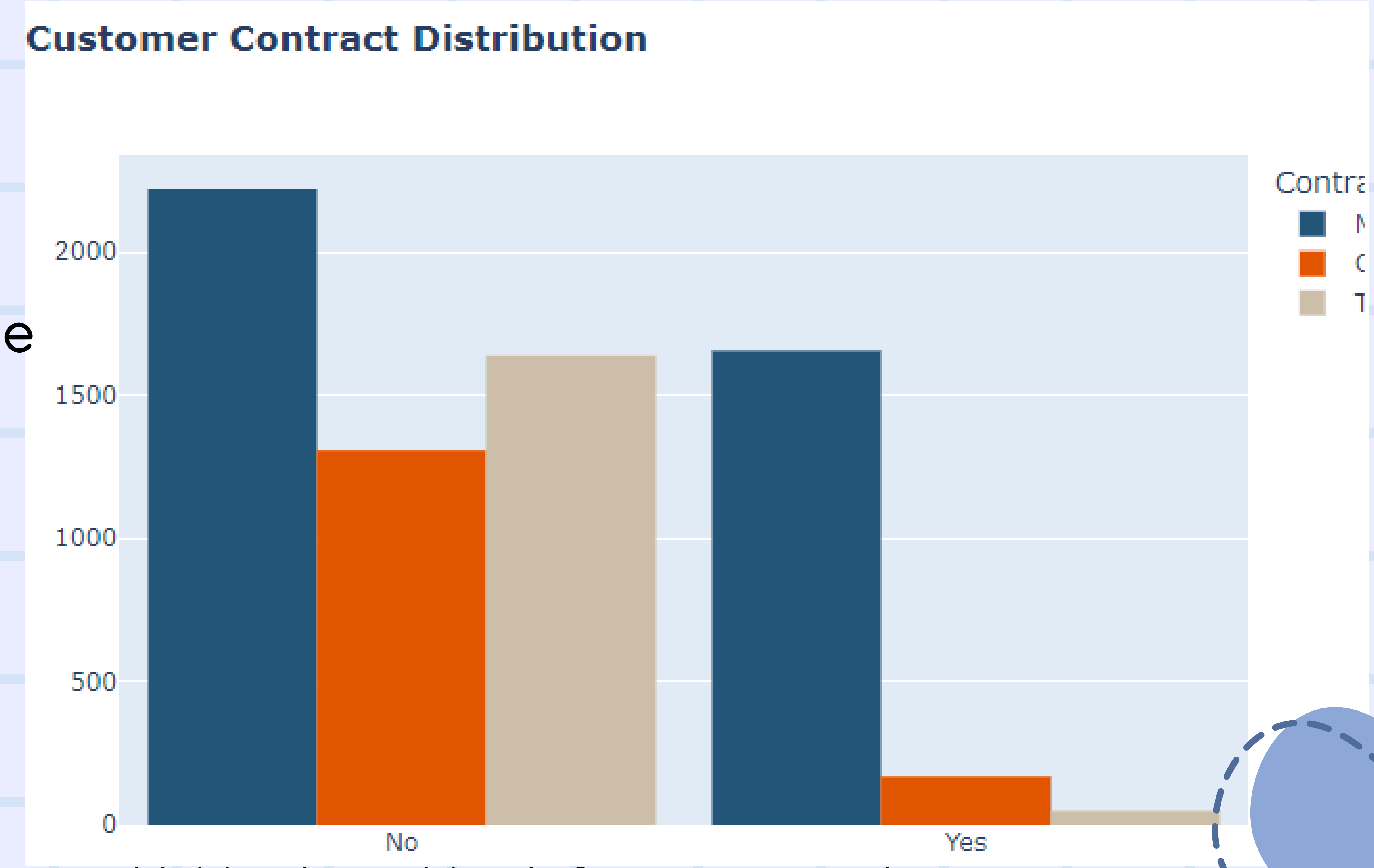
Services



DATA EXPLORATION

Account

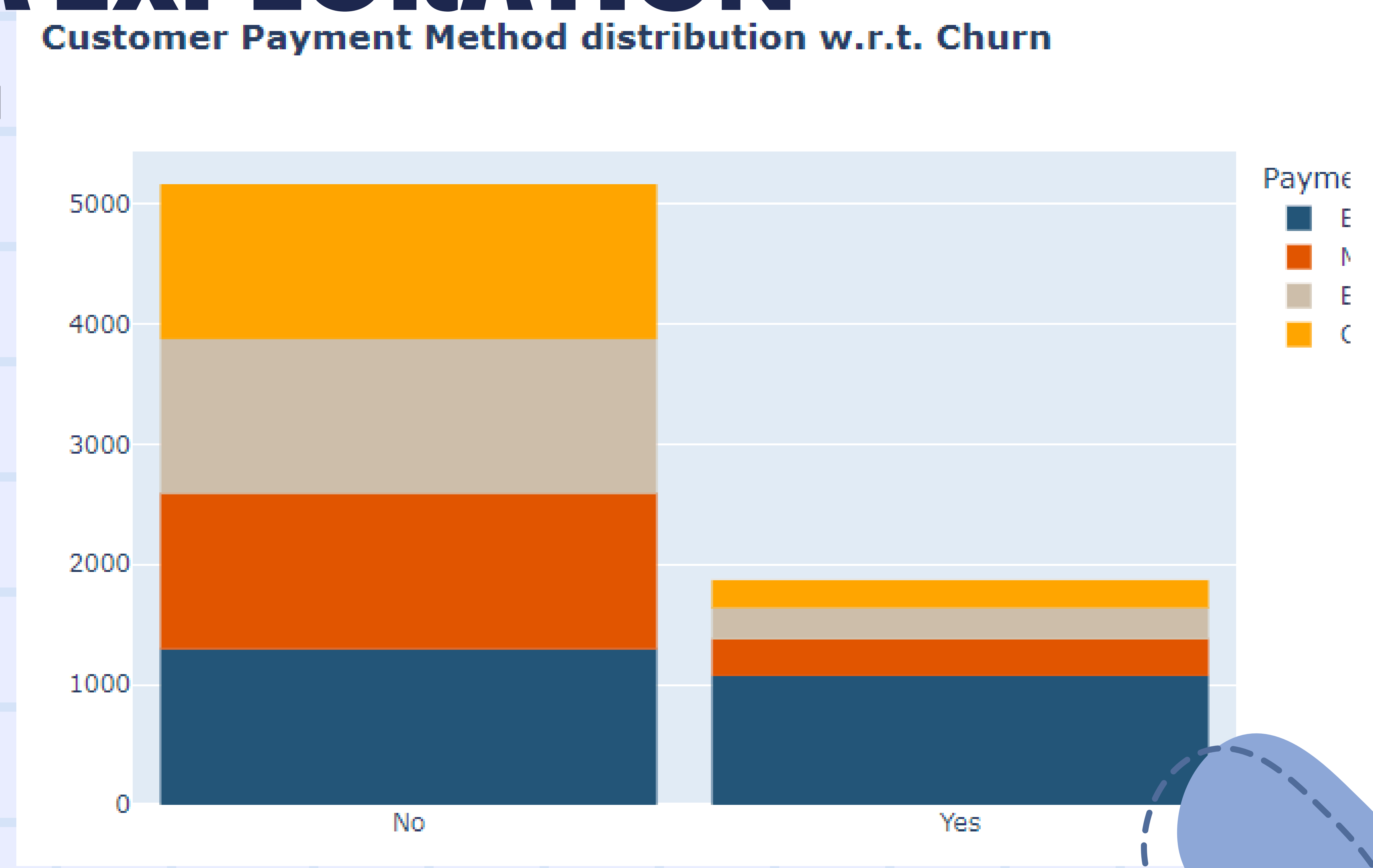
These analysis refers to customer Account and the process he follow to use the service. We will analyze Account information to discover the correlations and relation to churn.



About 75% of customer with Month-to-Month Contract opted to move out as compared to 13% of customrs with One Year Contract and 3% with Two Year Contract.

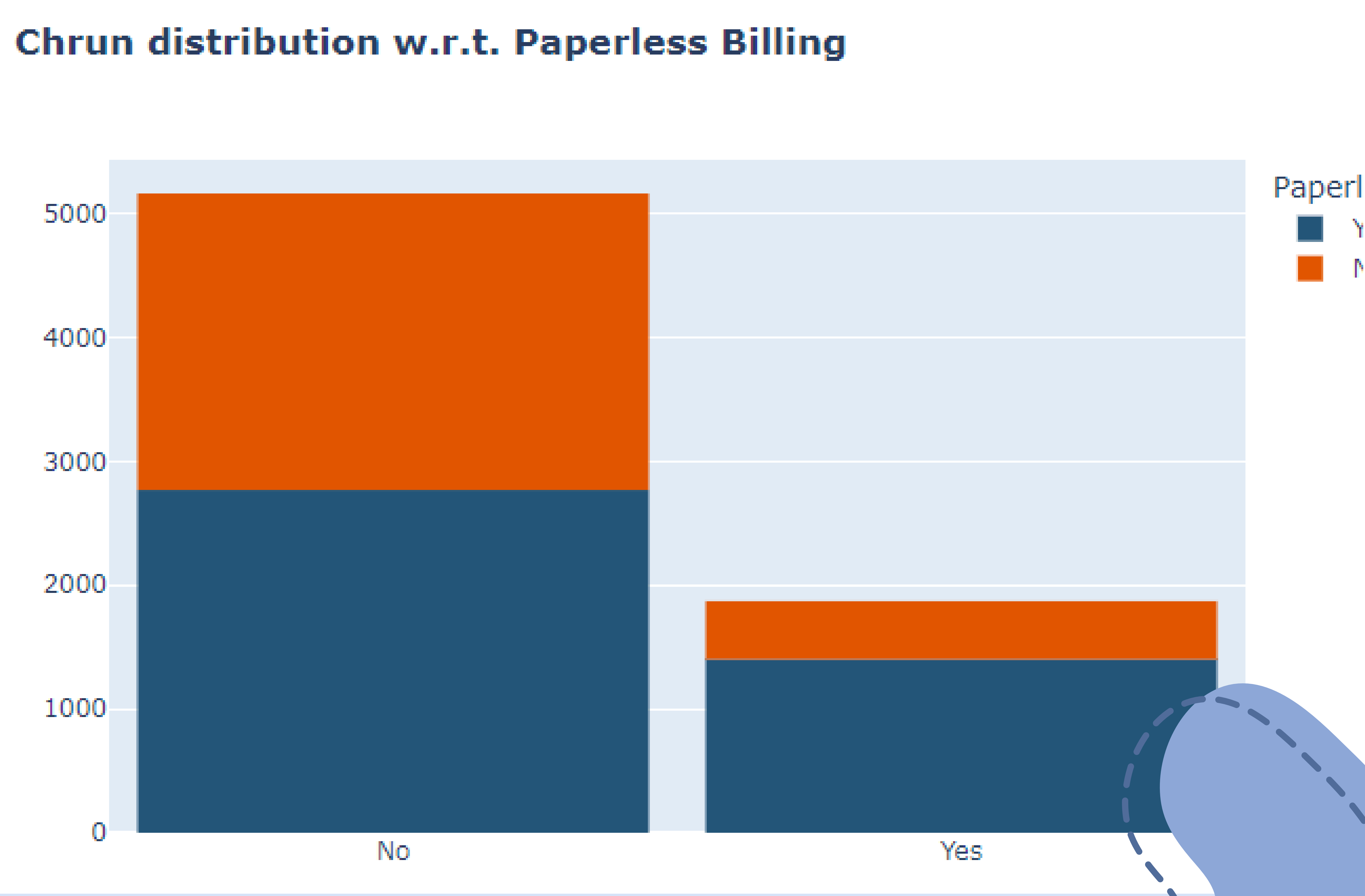
DATA EXPLORATION

- Major customers who moved out were having Electronic Check as Payment Method.
- Customers who opted for Credit-Card automatic transfer or Bank Automatic Transfer and Mailed Check as Payment Method were less likely to move out.



DATA EXPLORATION

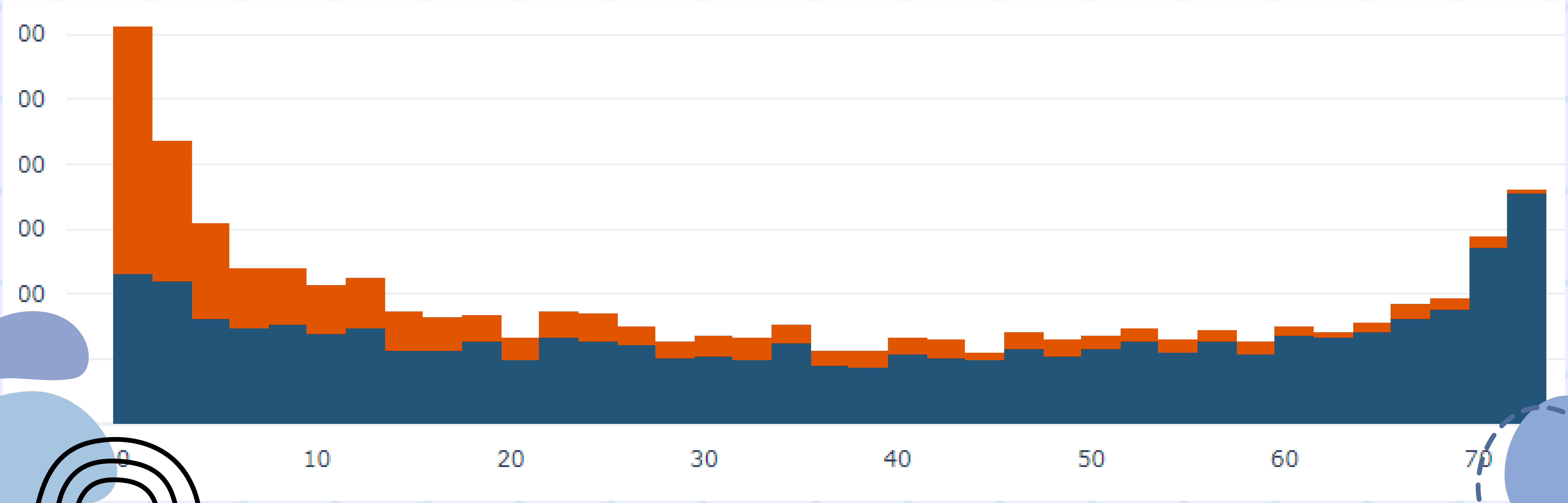
Customers with Paperless Billing are most likely to churn.



DATA EXPLORATION

For Numerical features:

```
num_viz('tenure').show()
```

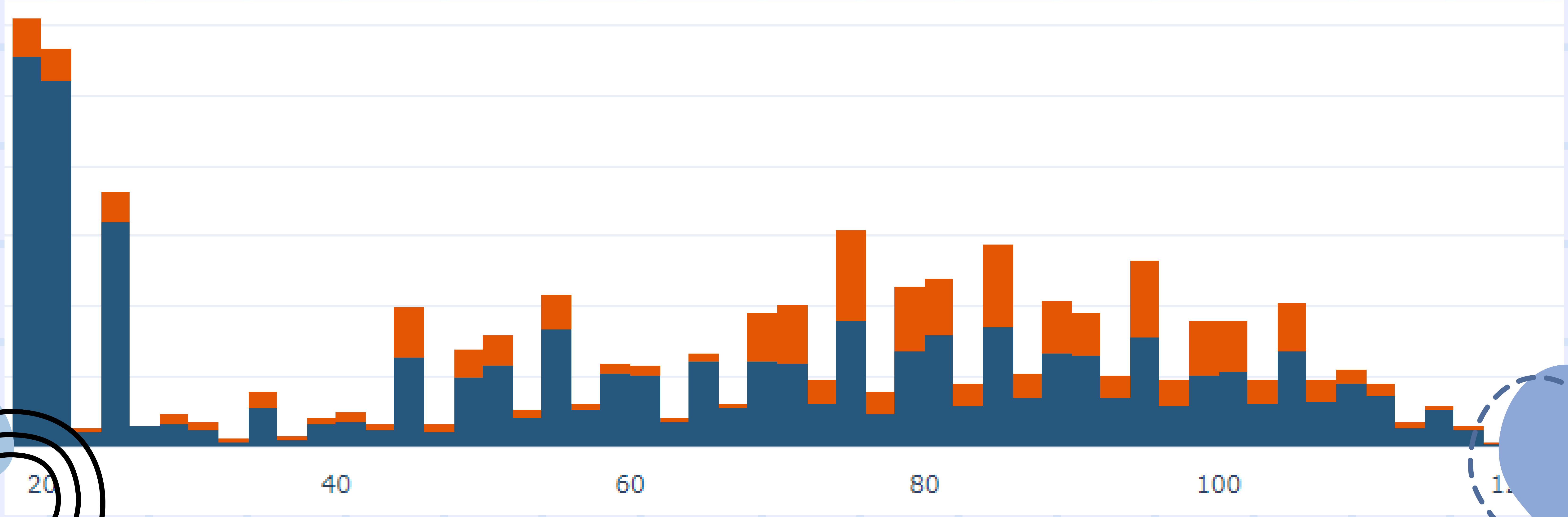


New customers are more likely to churn.

DATA EXPLORATION

For Numerical features:

```
num_viz('MonthlyCharges').show()
```



Customers with higher Monthly Charges are also more likely to churn.

DATA EXPLORATION

KEY INSIGHTS

- Customer Distribution:
  - Gender and partner status have no significant effect on churn.
  - Most customers are not senior citizens, and many have dependents.
- Service Subscription Patterns:
  - A small percentage of customers lack phone service.
  - Most customers do not subscribe to optional services like Online Security, Online Backup, Device Protection, Tech Support, Streaming TV, or Streaming Movies.
- Churn Characteristics:
- Higher Churn Rates:
  - Senior citizens, single people, and those without dependents are more likely to churn.
  - Customers using fiber optic internet show the highest churn rates.
  - Customers with month-to-month contracts and paperless billing have significantly higher churn rates.
  - Most churned customers have higher monthly charges but lower total charges, indicating early churn.
- Lower Churn Rates:
  - Yearly or biyearly contract customers are less likely to churn.
  - Customers with online security and related services (e.g., Online Backup, Device Protection, Tech Support) have lower churn rates.

DATA EXPLORATION

KEY INSIGHTS

Payment and Billing:

- Most churned customers use electronic checks, while non-churned customers are evenly distributed across payment methods.
- Customers with paperless billing are more prone to churn.

Tenure and Charges:

- Churn rates decrease as customer tenure increases.
- Churned customers tend to have higher monthly charges across all demographic groups.
- Churn likelihood increases with higher monthly and total charges.

DATA EXPLORATION

RECOMMENDATIONS

- Enhance Customer Retention for Senior Citizens and Singles:
  - Develop targeted retention programs for senior citizens and single customers, offering incentives like discounts or personalized plans to reduce churn rates in these segments.
- Promote Long-Term Contracts:
  - Encourage customers to switch from month-to-month contracts to longer-term contracts (yearly or biyearly) by offering attractive discounts or bundled services, as longer contracts show lower churn rates.
- Increase Value for High-Charge Customers:
  - Provide more value for customers with higher monthly charges, such as premium services or loyalty rewards. These customers are at a higher risk of early churn, so adding perceived value could improve retention.
- Improve Service Uptake:
  - Highlight the benefits of subscribing to services like Online Security, Tech Support, and Device Protection, as customers who use these services show lower churn rates.

DATA EXPLORATION

RECOMMENDATIONS

- Address Payment Method Issues:
  - Investigate why electronic check users are more prone to churn and consider offering alternative payment options with better incentives for these customers.
- Targeted Campaigns for High-Tenure Customers:
  - Create targeted campaigns for new customers to reach high-tenure status, as higher tenure correlates with lower churn rates. Focus on providing early value and engagement to retain them longer.
- Analyze and Address Billing Preferences:
  - Assess why paperless billing is associated with higher churn and explore ways to improve the experience for customers using paperless billing, such as more user-friendly payment interfaces or billing transparency.

By addressing these key areas, the company can reduce churn, improve customer satisfaction, and increase long-term revenue.

DATA PREPARATION:

A) Checking the Unique Values for Object Features:

```python
# Select only object columns
object_columns = df.select_dtypes(include=['object'])

# Get unique values and their counts for each object column
unique_values_info = {col: {
    "unique_values": object_columns[col].unique(),
    "num_unique": object_columns[col].nunique()
} for col in object_columns.columns}

# Print the unique values and their counts
for column, info in unique_values_info.items():
    print(f"Unique values in '{column}':")
    print(info["unique_values"])
    print(f"Number of unique values: {info['num_unique']}")
    print()  # Print a new line for better readability
```

```
Unique values in 'gender':
['Female' 'Male']
Number of unique values: 2

Unique values in 'Partner':
['Yes' 'No']
Number of unique values: 2

Unique values in 'Dependents':
['No' 'Yes']
Number of unique values: 2

Unique values in 'PhoneService':
['No' 'Yes']
Number of unique values: 2

Unique values in 'PaymentMethod
['Electronic check' 'Mailed che
 'Credit card (automatic)']
Number of unique values: 4

Unique values in 'Churn':
['No' 'Yes']
Number of unique values: 2
```

DATA PREPARATION:

B) Standardizing Categorical Features for Consistency:

– In the `OnlineSecurity` and Similar Features, we observed three unique values:
  – **Yes**: Indicates the customer has OnlineSecurity.
  – **No**: Indicates the customer does not have OnlineSecurity.
  – **No internet service**: Indicates the customer does not have any Internet service.

– To enhance data consistency:
  – We will replace the **No internet service** value with **No**, as it aligns with the binary nature of the feature.
  – This transformation will improve the clarity and usability of the data for analysis and machine learning models.

– Additionally, since we have a separate column **InternetService** that indicates whether the customer has internet service, standardizing these values helps maintain uniformity across related features.

DATA PREPARATION:

B) Standardizing Categorical Features for Consistency:

```
df.loc[df['InternetService'] == 'No',
              ['InternetService', 'OnlineSecurity', 'OnlineBackup',
               'DeviceProtection', 'TechSupport', 'StreamingTV',
               'StreamingMovies']]
```

	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovi
11	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet servi
16	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet servi
21	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet servi
22	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet servi
33	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet servi

```
# List of columns to replace 'No internet service' and 'No phone service:' with 'No'
columns_to_replace = [
    'MultipleLines',
    'OnlineSecurity',
    'OnlineBackup',
    'DeviceProtection',
    'TechSupport',
    'StreamingTV',
    'StreamingMovies'
]

# Replace 'No internet service' with 'No' in specified columns
for column in columns_to_replace:
    df[column] = df[column].replace({'No internet service': 'No', 'No phone service': 'No'})
```

DATA PREPARATION:

C) Convert Binary Categorical Variables:

```
# Convert 'Yes'/'No' columns to 0 and 1
binary_columns = [
    'Partner',
    'Dependents',
    'PhoneService',
    'MultipleLines',
    'OnlineSecurity',
    'OnlineBackup',
    'DeviceProtection',
    'TechSupport',
    'StreamingTV',
    'StreamingMovies',
    'PaperlessBilling',
    'Churn'
]

# Mapping Yes/No to 1/0
for col in binary_columns:
    df[col] = df[col].map({'Yes': 1, 'No': 0})
```

MACHINE LEARNING

- Feature Engineering
- Data Preprocessing
- Model Selection
- Modeling
- Model Performance

FEATURE ENGINEERING

One-Hot Encoding and Creating Dummy Variables:

```
# One-hot encode the multi-category columns
df = pd.get_dummies(df, columns=['InternetService', 'Contract', 'PaymentMethod'], drop_first=True)
# 'drop_first=True' ensures that one category is dropped to avoid multicollinearity
# Convert all boolean columns to integers (True -> 1, False -> 0)
df = df.astype({col: 'int64' for col in df.select_dtypes(include='bool').columns})
```

```
print(df.info())
print(df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7032 entries, 0 to 7042
Data columns (total 25 columns):
```

```
0   customerID                             7032 non-null   object
1   gender                                 7032 non-null   int64
2   SeniorCitizen                          7032 non-null   int64
3   Partner                                7032 non-null   int64
4   Dependents                             7032 non-null   int64
5   tenure                                 7032 non-null   int64
6   PhoneService                           7032 non-null   int64
7   MultipleLines                          7032 non-null   int64
8   OnlineSecurity                         7032 non-null   int64
9   OnlineBackup                           7032 non-null   int64
10  DeviceProtection                       7032 non-null   int64
11  TechSupport                            7032 non-null   int64
12  StreamingTV                            7032 non-null   int64
13  StreamingMovies                        7032 non-null   int64
14  PaperlessBilling                       7032 non-null   int64
15  MonthlyCharges                         7032 non-null   float64
16  TotalCharges                           7032 non-null   float64
17  Churn                                  7032 non-null   int64
18  InternetService_Fiber optic            7032 non-null   int64
19  InternetService_No                     7032 non-null   int64
20  Contract_One year                      7032 non-null   int64
21  Contract_Two year                      7032 non-null   int64
22  PaymentMethod_Credit card (automatic)  7032 non-null   int64
23  PaymentMethod_Electronic check         7032 non-null   int64
```

DATA PREPROCESSING

Splitting the Data into Features and Target Variable In this step, we separate the features (independent variables) from the target variable (dependent variable) that we want to predict. The features will be stored in `X`, while the target variable `Churn` will be stored in `y`.

```
X= df.drop(['Churn'],axis=1)
y = df['Churn']
```

Splitting the Data into Training and Testing Sets In this step, we split our feature set `X` and target variable `y` into training and testing sets. We use 10% of the data for testing and 90% for training. The `random_state` parameter ensures that the split is reproducible. We also extract the `customerID` from the test set to keep track of the customers, and then drop the `customerID` column from both the training and testing feature sets for model training.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=101)
Test_CustomerIDs = X_test['customerID']
X_train.drop(['customerID'],axis=1,inplace=True)
X_test.drop(['customerID'],axis=1,inplace=True)
```

DATA PREPROCESSING

Standardizing Numerical Features In this step, we standardize the numerical features in our DataFrame `df` using the `StandardScaler` from Scikit-learn. This transformation rescales the data to have a mean of 0 and a standard deviation of 1, which is important for many machine learning algorithms. After standardization, we create a new DataFrame `df_std` to hold the standardized values. Finally, we visualize the distribution of each standardized feature using a distribution plot for better understanding of the data's characteristics.

```
scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
```

MODELING SELECTION

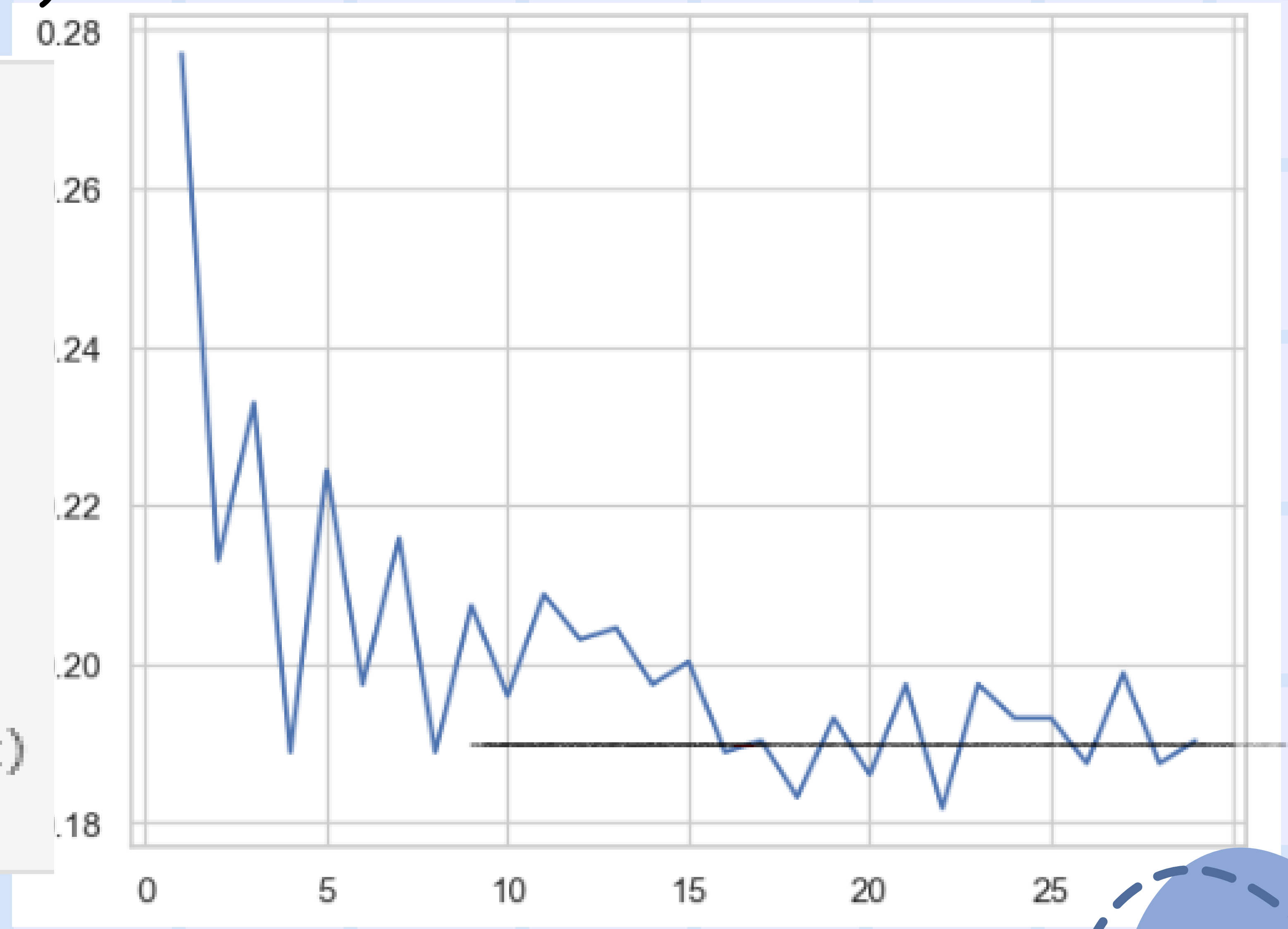
we will begin our modeling selection process by implementing the K-Nearest Neighbors (KNN)

```
test_error_rates = []


for k in range(1,30):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(scaled_X_train,y_train)

    y_pred_test = knn_model.predict(scaled_X_test)

    test_error = 1 - accuracy_score(y_test,y_pred_test)
    test_error_rates.append(test_error)
```



MODEL EVALUATION

We finalize the KNN model with \`n_neighbors=17\`, fit it to the training data, and make predictions on the test set.
The confusion matrix visualizes the classification performance, and the classification report provides precision, recall, and F1-score metrics for both classes.

Model Evaluation

```
Final_model = KNeighborsClassifier(n_neighbors=17)
Final_model.fit(scaled_X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=17)
```

```
y_pred = Final_model.predict(scaled_X_test)
y_pred_proba = Final_model.predict_proba(scaled_X_test)
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[489,  68],
       [ 66,  81]])
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       557
           1       0.54      0.55      0.55       147

    accuracy                           0.81       704
   macro avg       0.71      0.71      0.71       704
weighted avg       0.81      0.81      0.81       704
```

MODELING SELECTION

Random Forest

- Splits the data into training and testing sets, using 30% of the data for testing and 70% for training.

```
# Step 3: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- Initializes and Fits a `Random Forest classifier` to the training data to obtain feature importances.

```
# Step 4: Fit the initial Random Forest model to get feature importances
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

RandomForestClassifier?Documentation for RandomForestClassifieriFitted

```
RandomForestClassifier(random_state=42)
```

- Extracts and Displays the feature importances from the trained Random Forest model, sorting them in descending order.

```
# Get feature importance
feature_importances = model.feature_importances_
feature_names = X.columns
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
```

MODELING SELECTION

Random Forest

Random Forest Model Implementation

- Selects relevant features from the DataFrame df, Removes duplicate entries based on customerID, Converts the Churn column to a categorical type, and prints the class distribution of churn.

```
# Selected features
selected_features = ["customerID", "TotalCharges", "MonthlyCharges", "tenure", "Churn"]

# Filter the data
data_imputed = df[selected_features].drop_duplicates(subset='customerID')
data_imputed['Churn'] = data_imputed['Churn'].astype('category')

# Check the class distribution
print("Class distribution in the original dataset:\n", data_imputed['Churn'].value_counts())
```

```
Class distribution in the original dataset:
 Churn
0    5163
1    1869
Name: count, dtype: int64
```

- Performs a stratified split of the dataset into training and testing sets while maintaining the class distribution of churn. It also removes the `customerID` column from both sets for modeling purposes.

MODELING SELECTION

Random Forest

- Performs a stratified split of the dataset into training and testing sets while maintaining the class distribution of churn.
  It also removes the `customerID` column from both sets for modeling purposes.

154]:
```
# Stratified split to maintain class distribution
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=123)
for train_index, test_index in sss.split(data_imputed, data_imputed['Churn']):
    train_set = data_imputed.iloc[train_index]
    test_set = data_imputed.iloc[test_index]

# Drop customerID from training and testing sets
train_set = train_set.drop(columns=['customerID'])
test_CIF = test_set['customerID']
test_set = test_set.drop(columns=['customerID'])
```

- Checks if both classes ('Yes' and 'No') are present in the training set. If one class is missing, it raises an error and prints the class distribution.

155]:
```
# Check if both classes are present after the split
train_set_majority = train_set[train_set['Churn'] == 0]
train_set_minority = train_set[train_set['Churn'] == 1]

# Add a check to ensure both classes are present
if len(train_set_majority) == 0 or len(train_set_minority) == 0:
    print("Class distribution in training set:\n", train_set['Churn'].value_counts())
    raise ValueError("One of the classes is missing in the training set.")
```

MODELING EVALUATION

Random Forest

```
# Predictions on the test set
X_test = test_set.drop(columns=['Churn'])
y_test = test_set['Churn']

predicted_probs = rf_model.predict_proba(X_test)
predicted_classes = rf_model.predict(X_test)

# Confusion matrix
confusion = confusion_matrix(y_test, predicted_classes)
print("Confusion Matrix:\n", confusion)
```

```
Confusion Matrix:
 [[846 187]
 [182 192]]
```

- Calculates various performance metrics (accuracy, precision, recall, F1-score) based on the predictions made on the test set and prints them out.

```
# Metrics calculations
accuracy = accuracy_score(y_test, predicted_classes)
precision = precision_score(y_test, predicted_classes, pos_label=1)
recall = recall_score(y_test, predicted_classes, pos_label=1)
f1 = f1_score(y_test, predicted_classes, pos_label=1)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Accuracy: 0.7377
Precision: 0.5066
Recall: 0.5134
```

SUMMARY

it was noticed that the KNN model was more likely to predict the customer churn better then the Random Forest Model

THANK YOU