Customer_churn_prediction

May 13, 2025

Customer Churn Prediction with XGBoost on Imbalanced Telecom Data

1 Setup

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import PowerTransformer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, ConfusionMatrixDisplay, roc_auc_score
```

[2]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

[3]:
```
# Load the data
df = pd.read_excel("data/CustomerChurn.xlsx")
```

2 Exploratory Data Analysis

[4]:
```
# Show the data table
df
```

[4]:
```
      LoyaltyID Customer ID Senior Citizen Partner Dependents  Tenure  \
0        318537  7590-VHVEG             No     Yes         No       1
1        152148  5575-GNVDE             No      No         No      34
2        326527  3668-QPYBK             No      No         No       2
3        845894  7795-CFOCW             No      No         No      45
4        503388  9237-HQITU             No      No         No       2
…           …           …              …       …          …       …
7038     810338  6840-RESVB             No     Yes        Yes      24
7039     230811  2234-XADUH             No     Yes        Yes      72
7040     155157  4801-JZAZL             No     Yes        Yes      11
```

```
7041     731782  8361-LTMKD            Yes     Yes         No        4
7042     353947  3186-AJIEK             No      No         No       66

     Phone Service    Multiple Lines Internet Service Online Security  …  \
0               No  No phone service              DSL              No  …
1              Yes                No              DSL             Yes  …
2              Yes                No              DSL             Yes  …
3               No  No phone service              DSL             Yes  …
4              Yes                No      Fiber optic              No  …
…            …                …              …                …   …
7038           Yes               Yes              DSL             Yes  …
7039           Yes               Yes      Fiber optic              No  …
7040            No  No phone service              DSL             Yes  …
7041           Yes               Yes      Fiber optic              No  …
7042           Yes                No      Fiber optic             Yes  …

     Device Protection Tech Support Streaming TV Streaming Movies  \
0                   No           No           No               No
1                  Yes           No           No               No
2                   No           No           No               No
3                  Yes          Yes           No               No
4                   No           No           No               No
…                  …            …            …                …
7038               Yes          Yes          Yes              Yes
7039               Yes           No          Yes              Yes
7040                No           No           No               No
7041                No           No           No               No
7042               Yes          Yes          Yes              Yes

            Contract Paperless Billing             Payment Method  \
0     Month-to-month               Yes           Electronic check
1           One year                No               Mailed check
2     Month-to-month               Yes               Mailed check
3           One year                No  Bank transfer (automatic)
4     Month-to-month               Yes           Electronic check
…               …                 …                         …
7038        One year               Yes               Mailed check
7039        One year               Yes    Credit card (automatic)
7040  Month-to-month               Yes           Electronic check
7041  Month-to-month               Yes               Mailed check
7042        Two year               Yes  Bank transfer (automatic)

     Monthly Charges  Total Charges Churn
0              29.85          29.85    No
1              56.95         1889.5    No
2              53.85         108.15   Yes
3              42.30        1840.75    No
```

```
4             70.70         151.65   Yes
…             …          …    …
7038          84.80         1990.5    No
7039         103.20         7362.9    No
7040          29.60         346.45    No
7041          74.40          306.6   Yes
7042         105.65         6844.5    No

[7043 rows x 21 columns]
```

[5]:
```python
# Print (rows, columns) for the data
df.shape
```

[5]:
```
(7043, 21)
```

[6]:
```python
# Print the List of Columns
list(df.columns)
```

[6]:
```
['LoyaltyID',
 'Customer ID',
 'Senior Citizen',
 'Partner',
 'Dependents',
 'Tenure',
 'Phone Service',
 'Multiple Lines',
 'Internet Service',
 'Online Security',
 'Online Backup',
 'Device Protection',
 'Tech Support',
 'Streaming TV',
 'Streaming Movies',
 'Contract',
 'Paperless Billing',
 'Payment Method',
 'Monthly Charges',
 'Total Charges',
 'Churn']
```

The data is sourced from IBM's Base Samples. It contains 7043 rows and 19 columns. It is fictional data on customer churn for a telecom company.

More information about the data can be found here: https://community.ibm.com/community/user/blogs/steven-macko/2019/07/11/telco-customer-churn-1113

2.1 Unique values

[7]:
```python
# Print all unique values for each column
for col in df.columns:
    print(f"{col}:")
    print(df[col].unique())
```

```
LoyaltyID:
[318537 152148 326527 … 155157 731782 353947]
Customer ID:
['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' … '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
Senior Citizen:
['No' 'Yes']
Partner:
['Yes' 'No']
Dependents:
['No' 'Yes']
Tenure:
[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
 39]
Phone Service:
['No' 'Yes']
Multiple Lines:
['No phone service' 'No' 'Yes']
Internet Service:
['DSL' 'Fiber optic' 'No']
Online Security:
['No' 'Yes' 'No internet service']
Online Backup:
['Yes' 'No' 'No internet service']
Device Protection:
['No' 'Yes' 'No internet service']
Tech Support:
['No' 'Yes' 'No internet service']
Streaming TV:
['No' 'Yes' 'No internet service']
Streaming Movies:
['No' 'Yes' 'No internet service']
Contract:
['Month-to-month' 'One year' 'Two year']
Paperless Billing:
['Yes' 'No']
Payment Method:
['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
```

```
Monthly Charges:
[29.85 56.95 53.85 … 63.1  44.2  78.7 ]
Total Charges:
[29.85 1889.5 108.15 … 346.45 306.6 6844.5]
Churn:
['No' 'Yes']
```

Some of the features have possible values that are dependent on another feature. For example, Multiple Lines has 'No phone service' as a possible value. When it takes values of 'Yes' or 'No', it implies the user has phone service.

When modeling, 'Phone Service' and 'Internet Service' woiuld have to be excluded in order to prevent issues with independence/autocorrelation. Considering how phone service and internet service could be determined from other columns, it would probably be better to remove those columns entirely.

2.2 Data types

[8]:
```
# Check data types for each column
df.dtypes
```

[8]:
```
LoyaltyID              int64
Customer ID           object
Senior Citizen        object
Partner               object
Dependents            object
Tenure                 int64
Phone Service         object
Multiple Lines        object
Internet Service      object
Online Security       object
Online Backup         object
Device Protection     object
Tech Support          object
Streaming TV          object
Streaming Movies      object
Contract              object
Paperless Billing     object
Payment Method        object
Monthly Charges      float64
Total Charges         object
Churn                 object
dtype: object
```

Most of the data is in object type. This makes sense as most of the data is categorical.

LoyaltyID and Tenure are in int64. LoyaltyID shoulde be changed to an object or string because it is an identifier rather than a numerical value.

Total Charges is an object, while Monthly Charges is in float64. Total Charges should be changed to float64 since the values are supposed to be numeric.

[9]:
```python
# Convert 'Total Charges' to numeric, coercing errors to NaN
df['Total Charges'] = pd.to_numeric(df['Total Charges'], errors='coerce')
```

[10]:
```python
# Convert LoyaltyID to an object
df['LoyaltyID'] = df['LoyaltyID'].astype('object')
```

[11]:
```python
# Verify that the data types were changed for 'LoyaltyID' and 'Total Charges'
df.dtypes
```

[11]:
```
LoyaltyID              object
Customer ID            object
Senior Citizen         object
Partner                object
Dependents             object
Tenure                  int64
Phone Service          object
Multiple Lines         object
Internet Service       object
Online Security        object
Online Backup          object
Device Protection      object
Tech Support           object
Streaming TV           object
Streaming Movies       object
Contract               object
Paperless Billing      object
Payment Method         object
Monthly Charges       float64
Total Charges         float64
Churn                  object
dtype: object
```

LoyaltyID and Total Charges were successfully changed into the appropriate data types.

2.3 Handle Missing Values

[12]:
```python
# Check for nulls
print('Number of Nulls:')
print(int(df.isna().sum().sum()))
```

```
Number of Nulls:
11
```

[13]:
```python
# Find all rows with missing values
df[df.isna().any(axis=1)]
```

[13]:
```
      LoyaltyID Customer ID Senior Citizen Partner Dependents  Tenure  \
488      344543  4472-LVYGI             No     Yes        Yes       0
753      150036  3115-CZMZD             No      No        Yes       0
936      497688  5709-LVOEQ             No     Yes        Yes       0
1082     158969  4367-NUYAO             No     Yes        Yes       0
1340     470044  1371-DWPAZ             No     Yes        Yes       0
3331     937662  7644-OMVMY             No     Yes        Yes       0
3826     821083  3213-VVOLG             No     Yes        Yes       0
4380     947028  2520-SGTTA             No     Yes        Yes       0
5218     135257  2923-ARZLG             No     Yes        Yes       0
6670     317862  4075-WKNIU             No     Yes        Yes       0
6754     392646  2775-SEFEE             No      No        Yes       0

     Phone Service    Multiple Lines Internet Service       Online Security  \
488             No  No phone service              DSL                   Yes
753            Yes                No               No  No internet service
936            Yes                No              DSL                   Yes
1082           Yes               Yes               No  No internet service
1340            No  No phone service              DSL                   Yes
3331           Yes                No               No  No internet service
3826           Yes               Yes               No  No internet service
4380           Yes                No               No  No internet service
5218           Yes                No               No  No internet service
6670           Yes               Yes              DSL                    No
6754           Yes               Yes              DSL                   Yes

      …    Device Protection         Tech Support         Streaming TV  \
488   …                  Yes                  Yes                  Yes
753   …  No internet service  No internet service  No internet service
936   …                  Yes                   No                  Yes
1082  …  No internet service  No internet service  No internet service
1340  …                  Yes                  Yes                  Yes
3331  …  No internet service  No internet service  No internet service
3826  …  No internet service  No internet service  No internet service
4380  …  No internet service  No internet service  No internet service
5218  …  No internet service  No internet service  No internet service
6670  …                  Yes                  Yes                  Yes
6754  …                   No                  Yes                   No

         Streaming Movies  Contract Paperless Billing  \
488                    No  Two year               Yes
753   No internet service  Two year                No
936                   Yes  Two year                No
1082  No internet service  Two year                No
1340                   No  Two year                No
3331  No internet service  Two year                No
3826  No internet service  Two year                No
```

```
4380  No internet service  Two year              No
5218  No internet service  One year             Yes
6670                   No  Two year              No
6754                   No  Two year             Yes

                 Payment Method Monthly Charges  Total Charges  Churn
488   Bank transfer (automatic)           52.55            NaN     No
753                Mailed check           20.25            NaN     No
936                Mailed check           80.85            NaN     No
1082               Mailed check           25.75            NaN     No
1340    Credit card (automatic)           56.05            NaN     No
3331               Mailed check           19.85            NaN     No
3826               Mailed check           25.35            NaN     No
4380               Mailed check           20.00            NaN     No
5218               Mailed check           19.70            NaN     No
6670               Mailed check           73.35            NaN     No
6754  Bank transfer (automatic)           61.90            NaN     No

[11 rows x 21 columns]
```

There are only null valuies for the Total Charges column. Since there happens to be values for the Monthly Charges for each row, we can impute the missing values with the corresponding Monthly Charges. What happened was probably that the users are new and just had their first Monthly Charge. We will impute the missing values with the corresponding Monthly Charge.

[14]:
```python
# Impute nulls with Monthly Charges
df.loc[df['Total Charges'].isna(), 'Total Charges'] = df.loc[df['Total␣
↪Charges'].isna(), 'Monthly Charges']
```

[15]:
```python
# Check for nulls
print('Number of Nulls:')
print(int(df.isna().sum().sum()))
```

```
Number of Nulls:
0
```

There are no rows with missing values in the data anymore. To further verify if the imputation was done correctly, let's find the rows that had their missing values imputed.

We will use the LoyaltyID of those rows to filter the data.

[16]:
```python
# Check some of the previous rows with missing values
df[df['LoyaltyID'].astype(str).isin(['344543', '150036', '497688'])]
```

[16]:
```
    LoyaltyID Customer ID Senior Citizen Partner Dependents  Tenure  \
488    344543  4472-LVYGI             No     Yes        Yes       0
753    150036  3115-CZMZD             No      No        Yes       0
936    497688  5709-LVOEQ             No     Yes        Yes       0
```

```
    Phone Service    Multiple Lines Internet Service      Online Security  \
488            No  No phone service              DSL                  Yes
753           Yes                No               No  No internet service
936           Yes                No              DSL                  Yes

    …    Device Protection         Tech Support         Streaming TV  \
488 …                  Yes                  Yes                  Yes
753 …  No internet service  No internet service  No internet service
936 …                  Yes                   No                  Yes

        Streaming Movies  Contract Paperless Billing  \
488                   No  Two year               Yes
753  No internet service  Two year                No
936                  Yes  Two year                No

                Payment Method Monthly Charges  Total Charges  Churn
488  Bank transfer (automatic)           52.55          52.55     No
753               Mailed check           20.25          20.25     No
936               Mailed check           80.85          80.85     No

[3 rows x 21 columns]
```

Here are some of the rows, and we can see that the Total Charges are now the same as the Monthly Charges for these users.

2.4 Remove Any Extra White Space in the Data

[17]:
```python
# Select columns that are objects
cat_cols = df.select_dtypes(include=['object']).columns

# Remove any extra white space
df[cat_cols] = df[cat_cols].apply(lambda x: x.astype(str).str.strip())
```

2.5 Check for Duplicate Rows

[18]:
```python
num_duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {num_duplicates}")
```

```
Number of duplicate rows: 0
```

2.6 Summary Statistics

[19]:
```python
# Summary Statistics
df.describe()
```

[19]:

	Tenure	Monthly Charges	Total Charges
count	7043.000000	7043.000000	7043.000000
mean	32.371149	64.761692	2279.798992

```
std      24.559481      30.090047   2266.730170
min       0.000000      18.250000     18.800000
25%       9.000000      35.500000    398.550000
50%      29.000000      70.350000   1394.550000
75%      55.000000      89.850000   3786.600000
max      72.000000     118.750000   8684.800000
```

The Summary Statistics table is only printed for the 3 non-object columns.

From the means, mins, and maxes, the values seem similar in magnitude have significant differences in magnitude between Total Charges and the other 2 features.

Tenure, although hard to tell just looking at feature name, is in months. The value of 72 represents 6 years. Half of the company's customers have stayed with them for around 2 and a half years.

[20]:
```python
# Filter the data for Total Charges over $8000 and having 72 months of Tenure
df[(df['Total Charges'] > 8000) & (df['Tenure'] == 72)]
```

[20]:
```
     LoyaltyID Customer ID Senior Citizen Partner Dependents  Tenure  \
193     113529  9680-NIAUV             No     Yes        Yes      72
198     868830  7255-SSFBC             No     Yes        Yes      72
369     116254  3520-FJGCV             No     Yes        Yes      72
437     173159  4376-KFVRS             No     Yes        Yes      72
464     289423  1480-BKXGA            Yes     Yes         No      72
…          …           …            …      …         …      …
5995    456645  2193-SFWQW             No     Yes        Yes      72
6118    717232  9924-JPRMC             No      No         No      72
6403    183948  3258-ZKPAI             No     Yes        Yes      72
6728    382019  2380-DAMQP             No     Yes         No      72
6768    546231  9739-JLPQJ             No     Yes        Yes      72

     Phone Service Multiple Lines Internet Service Online Security  …  \
193            Yes            Yes      Fiber optic             Yes  …
198            Yes            Yes      Fiber optic              No  …
369            Yes            Yes      Fiber optic             Yes  …
437            Yes            Yes      Fiber optic             Yes  …
464            Yes            Yes      Fiber optic             Yes  …
…              …              …                …               …  …
5995           Yes             No      Fiber optic             Yes  …
6118           Yes            Yes      Fiber optic             Yes  …
6403           Yes            Yes      Fiber optic             Yes  …
6728           Yes            Yes      Fiber optic             Yes  …
6768           Yes            Yes      Fiber optic             Yes  …

     Device Protection Tech Support Streaming TV Streaming Movies  Contract  \
193                Yes           No          Yes              Yes  Two year
198                Yes          Yes          Yes              Yes  Two year
369                Yes          Yes          Yes              Yes  Two year
437                Yes          Yes          Yes              Yes  Two year
```

```
464                 Yes          Yes          Yes              Yes  Two year
…                 …          …          …              …        …
5995                Yes          Yes          Yes              Yes  Two year
6118                Yes          Yes          Yes              Yes  Two year
6403                Yes          Yes          Yes              Yes  Two year
6728                Yes          Yes          Yes              Yes  Two year
6768                Yes          Yes          Yes              Yes  Two year

     Paperless Billing             Payment Method Monthly Charges  \
193                 No    Credit card (automatic)          109.70
198                Yes  Bank transfer (automatic)          112.25
369                Yes    Credit card (automatic)          112.60
437                Yes    Credit card (automatic)          114.05
464                Yes  Bank transfer (automatic)          116.05
…                 …                        …             …
5995                No  Bank transfer (automatic)          111.95
6118               Yes           Electronic check          118.20
6403               Yes  Bank transfer (automatic)          116.60
6728                No           Electronic check          115.15
6768                No    Credit card (automatic)          117.50

      Total Charges  Churn
193         8129.30     No
198         8041.65     No
369         8126.65     No
437         8468.20     No
464         8404.90     No
…               …    …
5995        8033.10     No
6118        8547.15     No
6403        8337.45     No
6728        8349.70     No
6768        8670.10     No

[62 rows x 21 columns]
```

[21]:
```
8684/72
```

[21]:
```
120.61111111111111
```

The range of values look reasonable. This is true even for the values of Total Charges at or close to the max. These rows translate to around $120 per month over their tenure.

Considering how reasonable the values appear, we will consider the data free from outliers and refrain from further action in order to preserve data integrity. We will also be using XGBoost for the model, which is robust to outliers when used for classification tasks such as our Customer Churn Prediction.

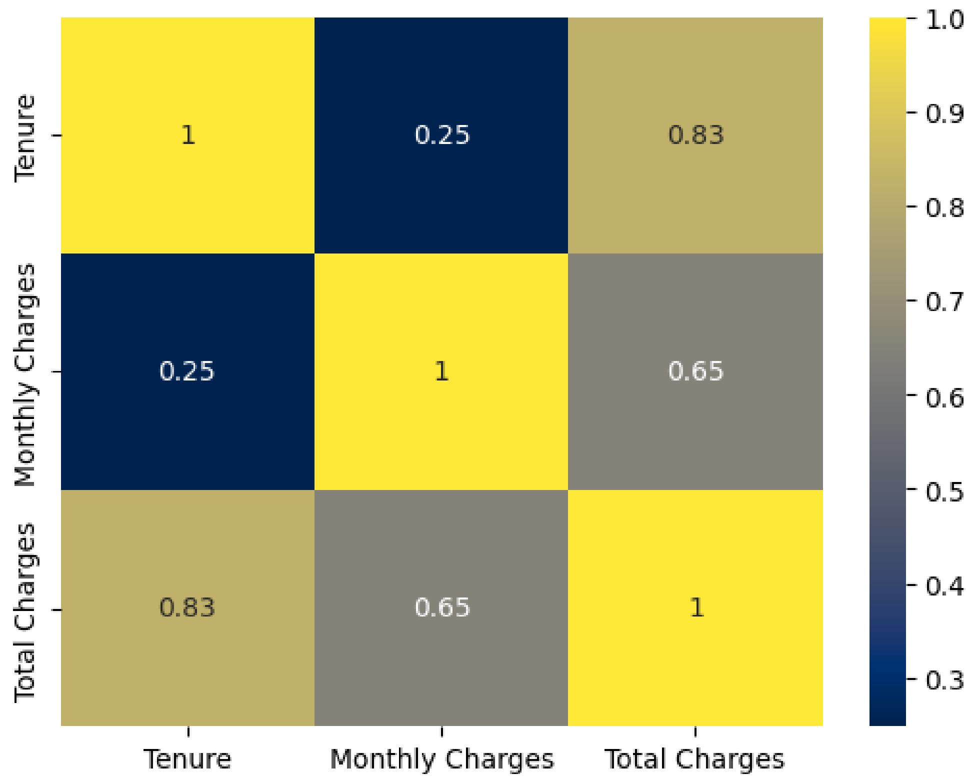
2.7 Correlation Heatmap

[22]:
```
# Select columns that are not objects
num_cols = df.select_dtypes(exclude=['object']).columns

# Find the correlation matrix
corr = df[num_cols].corr()

# Construct the Correlation Heatmap
sns.heatmap(corr, annot=True, fmt='.2g', cmap='cividis')
```

[22]:
```
<Axes: >
```



Tenure has high 0.83 correlation with Total Charges, which is reasonable considering the longer a customer stays with the company, the larger the accumulation of their charges.

Monthly Charges has a moderate 0.65 correlation with Total Charges. This likely means that customers have a tendency to change their plans, and by extension, their Monthly Charges rather than sticking to the same plan.

2.8 Plot Categorical Variable Counts

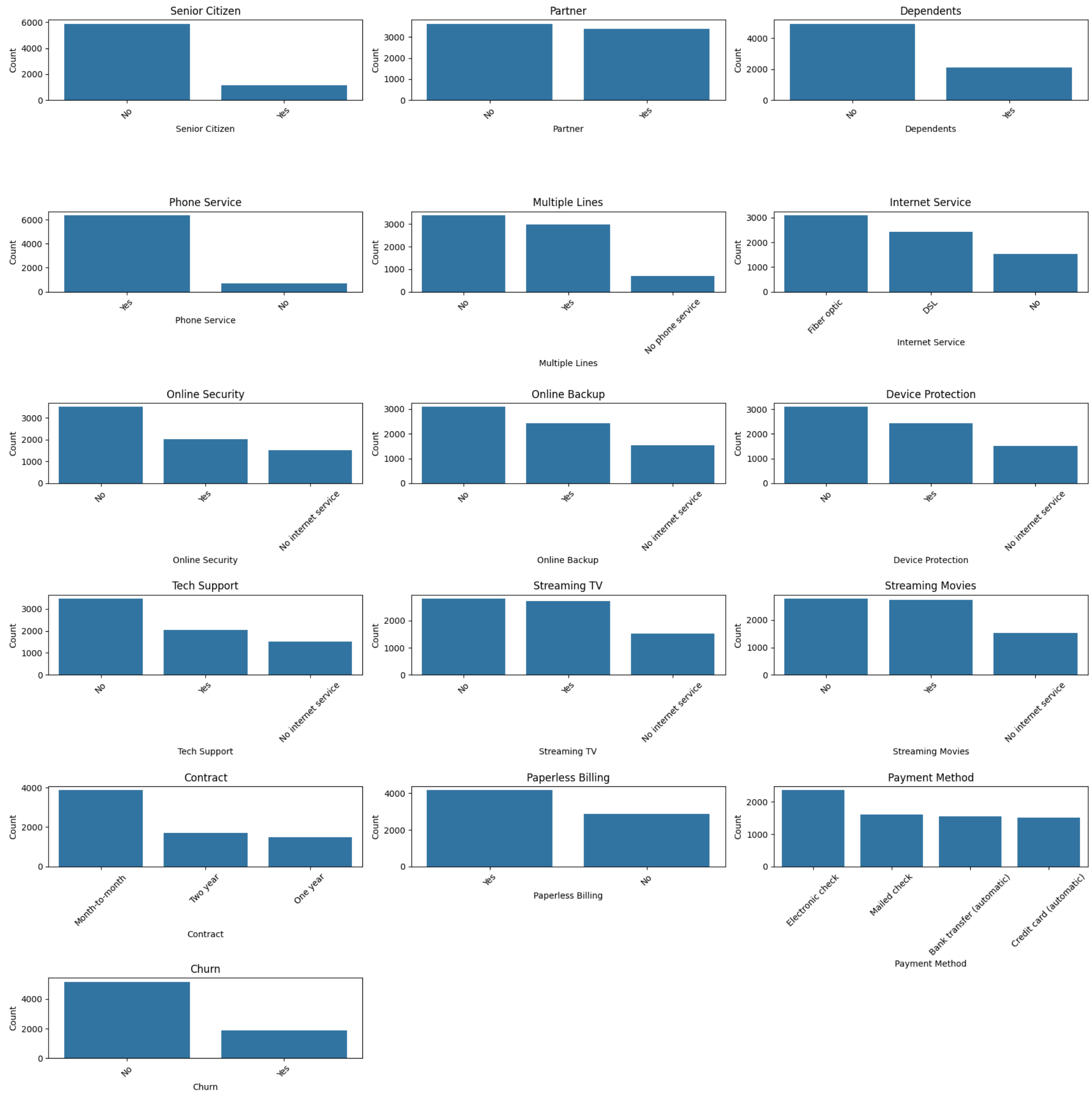
[23]:
```
# Find Categorical Columns to plot
cat_cols_to_plot = [col for col in df.select_dtypes(include=['object']).columns
                    if col not in ['LoyaltyID', 'Customer ID']]

# Set subplots
fig, axes = plt.subplots(6, 3, figsize=(18, 18))
axes = axes.flatten()

# Make the bar plots
for i, column in enumerate(cat_cols_to_plot):
    sns.countplot(data=df, x=column, ax=axes[i], order=df[column].
 ↪value_counts().index)
    axes[i].set_title(f"{column}")
    axes[i].set_ylabel('Count')
    axes[i].tick_params(axis='x', rotation=45)

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].set_visible(False)

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



From the plots of the counts for each feature, we see there is some imbalance for each feature except for Partner. Telecom data tends to have imbalanced data.

For the time being, we will proceed without taking action to balance the data, as it may be better to have the model work with what may be naturally imbalanced data. If we forcefully balance it, that may introduce biases.

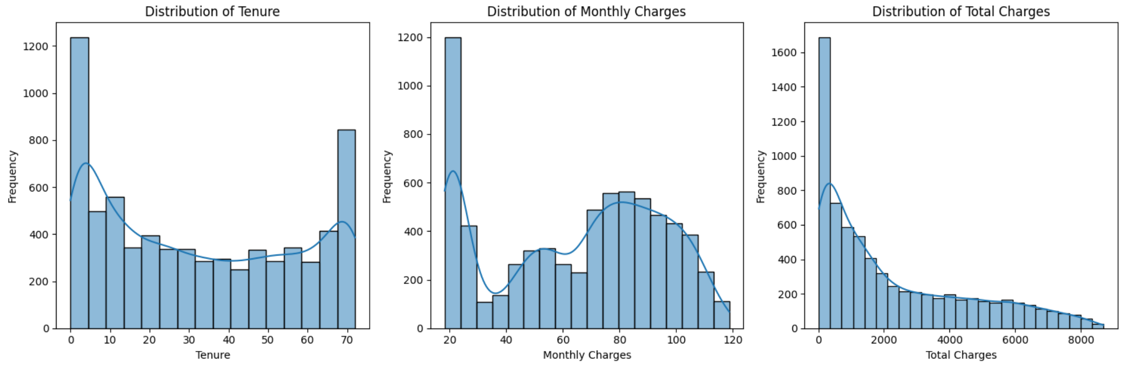
2.9 Plot Distributions of Numerical Features

[24]:
```
# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes = axes.flatten()
```

```
for i, col in enumerate(num_cols):
    sns.histplot(data=df, x=col, kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Frequency")

# Remove unused axes if any
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



The features do not have normal distributions. Tenure and Monthly Charges are multimodal, while Total Charges is right skewed.

3 Modeling

3.1 Transform Numerical Features

[25]:
```
# Apply Yeo-Johnson transformation to numerical features
pt = PowerTransformer(method='yeo-johnson')
df[num_cols] = pt.fit_transform(df[num_cols])
df.head()
```

[25]:
```
  LoyaltyID Customer ID Senior Citizen Partner Dependents    Tenure  \
0    318537  7590-VHVEG             No     Yes         No -1.644343
1    152148  5575-GNVDE             No      No         No  0.297205
2    326527  3668-QPYBK             No      No         No -1.495444
3    845894  7795-CFOCW             No      No         No  0.646327
4    503388  9237-HQITU             No      No         No -1.495444

  Phone Service    Multiple Lines Internet Service Online Security  …  \
0            No  No phone service              DSL              No  …
```

```
1          Yes                No             DSL             Yes  …
2          Yes                No             DSL             Yes  …
3           No  No phone service             DSL             Yes  …
4          Yes                No     Fiber optic              No  …

  Device Protection Tech Support Streaming TV Streaming Movies  \
0                No           No           No               No
1               Yes           No           No               No
2                No           No           No               No
3               Yes          Yes           No               No
4                No           No           No               No

         Contract Paperless Billing             Payment Method  \
0  Month-to-month               Yes           Electronic check
1        One year                No               Mailed check
2  Month-to-month               Yes               Mailed check
3        One year                No  Bank transfer (automatic)
4  Month-to-month               Yes           Electronic check

  Monthly Charges  Total Charges  Churn
0       -1.158541      -1.805206     No
1       -0.239492       0.256861     No
2       -0.342665      -1.381338    Yes
3       -0.731079       0.235865     No
4        0.213545      -1.244141    Yes

[5 rows x 21 columns]
```

The data is transformed to take into account the skewed distribution. Since we will be using XGBoost for the model, the data will not need to be scaled.

[26]:
```
# Make a copy of the dataframe
df_copy = df.copy()
```

3.2 Encode Categorical Features

[27]:
```
# Find Binary Columns
binary_cols = ['Senior Citizen', 'Partner', 'Dependents', 'Paperless Billing',␣
 ↪'Churn']

# Encoding Binary Columns
for col in binary_cols:
    df[col] = df[col].map({'No': 0, 'Yes': 1})
```

[28]:
```
# Set Additional Service Columns
add_service_cols = [
    'Multiple Lines', 'Online Security', 'Online Backup',
```

```
    'Device Protection', 'Tech Support', 'Streaming TV', 'Streaming Movies'
]

# Encoding Additional Service Columns
df = pd.get_dummies(df, columns=add_service_cols, drop_first=True, dtype=int)
```

[29]:
```
# Set Other Columns (Contract and Payment Method)
other_cols = ['Contract', 'Payment Method']

# Encoding Other Columns
df = pd.get_dummies(df, columns=other_cols, drop_first=True, dtype=int)
```

[30]:
```
# Print first 5 rows of data
df.head()
```

[30]:
```
   LoyaltyID Customer ID  Senior Citizen  Partner  Dependents    Tenure  \
0     318537  7590-VHVEG               0        1           0 -1.644343
1     152148  5575-GNVDE               0        0           0  0.297205
2     326527  3668-QPYBK               0        0           0 -1.495444
3     845894  7795-CFOCW               0        0           0  0.646327
4     503388  9237-HQITU               0        0           0 -1.495444

  Phone Service Internet Service  Paperless Billing  Monthly Charges  …  \
0            No              DSL                  1        -1.158541  …
1           Yes              DSL                  0        -0.239492  …
2           Yes              DSL                  1        -0.342665  …
3            No              DSL                  0        -0.731079  …
4           Yes      Fiber optic                  1         0.213545  …

   Tech Support_Yes  Streaming TV_No internet service  Streaming TV_Yes  \
0                 0                                 0                 0
1                 0                                 0                 0
2                 0                                 0                 0
3                 1                                 0                 0
4                 0                                 0                 0

   Streaming Movies_No internet service  Streaming Movies_Yes  \
0                                     0                     0
1                                     0                     0
2                                     0                     0
3                                     0                     0
4                                     0                     0

   Contract_One year  Contract_Two year  \
0                  0                  0
1                  1                  0
2                  0                  0
```

```
3                    1                  0
4                    0                  0

   Payment Method_Credit card (automatic)  Payment Method_Electronic check  \
0                                       0                                1
1                                       0                                0
2                                       0                                0
3                                       0                                0
4                                       0                                1

   Payment Method_Mailed check
0                            0
1                            1
2                            1
3                            0
4                            0

[5 rows x 31 columns]
```

3.3 Check Unique Values for Each Feature

[31]:

```
# Print all unique values for each column
for col in df.columns:
    print(f"{col}:")
    print(df[col].unique())
```

```
LoyaltyID:
['318537' '152148' '326527' … '155157' '731782' '353947']
Customer ID:
['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' … '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
Senior Citizen:
[0 1]
Partner:
[1 0]
Dependents:
[0 1]
Tenure:
[-1.6443435   0.29720527 -1.49544443  0.64632675 -0.92628704 -0.16269601
 -0.78871081  0.08053824  1.10445427 -0.60622285 -0.44435051  1.00352171
  0.76159927 -0.03718286  1.27288016  0.84473766  1.31927073 -0.2065339
 -0.66440415  0.15530631  0.70462831  1.34219847 -0.39396295  0.04208131
 -1.16936305  0.67564931 -0.72512119  1.29616592  1.12912864  0.58660355
 -0.49640807  1.05444632 -0.34510326  1.20189603 -0.8555903  -1.37221793
  0.19168912  0.78961168  1.15359039  0.95163168 -1.00148358  0.5561775
  0.33125677  0.7332748   0.11826746  1.17784456  0.43037665  1.2494094
  0.22744746  0.9253082   0.3978215   0.36478829  0.52535695 -1.08208957
 -1.26512968  0.26261052  1.22574947 -0.11990163  0.97770058  1.07956199
```

```
 -0.55030087 -0.25148998  0.8718688   0.4941272   1.02910155 -0.07808394
  0.61664894 -0.29764818  0.89872305  0.81732147  0.00285584 -1.84408848
  0.46247264]
Phone Service:
['No' 'Yes']
Internet Service:
['DSL' 'Fiber optic' 'No']
Paperless Billing:
[1 0]
Monthly Charges:
[-1.1585412  -0.23949171 -0.34266505 … -0.03598427 -0.66670841
  0.4741192 ]
Total Charges:
[-1.80520648  0.25686123 -1.38133786 … -0.85408054 -0.91709707
  1.48322527]
Churn:
[0 1]
Multiple Lines_No phone service:
[1 0]
Multiple Lines_Yes:
[0 1]
Online Security_No internet service:
[0 1]
Online Security_Yes:
[0 1]
Online Backup_No internet service:
[0 1]
Online Backup_Yes:
[1 0]
Device Protection_No internet service:
[0 1]
Device Protection_Yes:
[0 1]
Tech Support_No internet service:
[0 1]
Tech Support_Yes:
[0 1]
Streaming TV_No internet service:
[0 1]
Streaming TV_Yes:
[0 1]
Streaming Movies_No internet service:
[0 1]
Streaming Movies_Yes:
[0 1]
Contract_One year:
[0 1]
Contract_Two year:
```

```
[0 1]
Payment Method_Credit card (automatic):
[0 1]
Payment Method_Electronic check:
[1 0]
Payment Method_Mailed check:
[0 1]
```

The categories were all encoded. Drop_first was used to renmove the first category/dummary variable created to remove multicollinearity issues and prevent the dummy variable trap.

For example, the device_protection feature took values of 'Yes', 'No', or 'No internet service'.

The dummy variables in the dataframe are 'Yes' and 'No internet service'. The category of 'No' is implied from values of 0 in the 'Yes' dummy variable.

[32]:
```python
# Select All Features to be used in the model
features = df.drop(columns=['LoyaltyID', 'Customer ID', 'Phone Service',␣
↪'Internet Service'])
```

3.4 Split the Data

[33]:
```python
features
```

[33]:
```
      Senior Citizen  Partner  Dependents    Tenure  Paperless Billing  \
0                  0        1           0 -1.644343                  1
1                  0        0           0  0.297205                  0
2                  0        0           0 -1.495444                  1
3                  0        0           0  0.646327                  0
4                  0        0           0 -1.495444                  1
…                …      …         …       …                …
7038               0        1           1 -0.078084                  1
7039               0        1           1  1.342198                  1
7040               0        1           1 -0.725121                  1
7041               1        1           0 -1.265130                  1
7042               0        0           0  1.201896                  1

      Monthly Charges  Total Charges  Churn  Multiple Lines_No phone service  \
0           -1.158541      -1.805206      0                                1
1           -0.239492       0.256861      0                                0
2           -0.342665      -1.381338      1                                0
3           -0.731079       0.235865      0                                1
4            0.213545      -1.244141      1                                0
…                 …            …    …                             …
7038         0.671510       0.299107      0                                0
7039         1.260981       1.565492      0                                0
7040        -1.167241      -0.854081      0                                1
7041         0.334312      -0.917097      1                                0
7042         1.338863       1.483225      0                                0
```

```
      Multiple Lines_Yes  …  Tech Support_Yes  \
0                      0  …                 0
1                      0  …                 0
2                      0  …                 0
3                      0  …                 1
4                      0  …                 0
…                    …  …               …
7038                   1  …                 1
7039                   1  …                 0
7040                   0  …                 0
7041                   1  …                 0
7042                   0  …                 1

      Streaming TV_No internet service  Streaming TV_Yes  \
0                                    0                 0
1                                    0                 0
2                                    0                 0
3                                    0                 0
4                                    0                 0
…                                  …               …
7038                                 0                 1
7039                                 0                 1
7040                                 0                 0
7041                                 0                 0
7042                                 0                 1

      Streaming Movies_No internet service  Streaming Movies_Yes  \
0                                        0                     0
1                                        0                     0
2                                        0                     0
3                                        0                     0
4                                        0                     0
…                                      …                   …
7038                                     0                     1
7039                                     0                     1
7040                                     0                     0
7041                                     0                     0
7042                                     0                     1

      Contract_One year  Contract_Two year  \
0                     0                  0
1                     1                  0
2                     0                  0
3                     1                  0
4                     0                  0
…                   …                …
```

```
7038                    1                  0
7039                    1                  0
7040                    0                  0
7041                    0                  0
7042                    0                  1

      Payment Method_Credit card (automatic)  Payment Method_Electronic check  \
0                                          0                                1
1                                          0                                0
2                                          0                                0
3                                          0                                0
4                                          0                                1
…                                        …                              …
7038                                       0                                0
7039                                       1                                0
7040                                       0                                1
7041                                       0                                0
7042                                       0                                0

      Payment Method_Mailed check
0                               0
1                               1
2                               1
3                               0
4                               0
…                             …
7038                            1
7039                            0
7040                            0
7041                            1
7042                            0

[7043 rows x 27 columns]
```

3.5 Feature Selection

[34]:

```
# Set independent variables
X = features.drop(columns='Churn')

# Set dependent variable
y = features['Churn']

# Split the data at a 80:20 ratio and set the random state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

3.5.1 Fit the Model without Hyperparameter Tuning

[35]:
```
# Initialize the classifier
model = xgb.XGBClassifier(
    eval_metric='auc',
    random_state=42
)

# Fit the model
model.fit(X_train, y_train)

# Predict labels for the test set
y_pred = model.predict(X_test)
```

3.5.2 Model Validation

[36]:
```
# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.4f}")
```

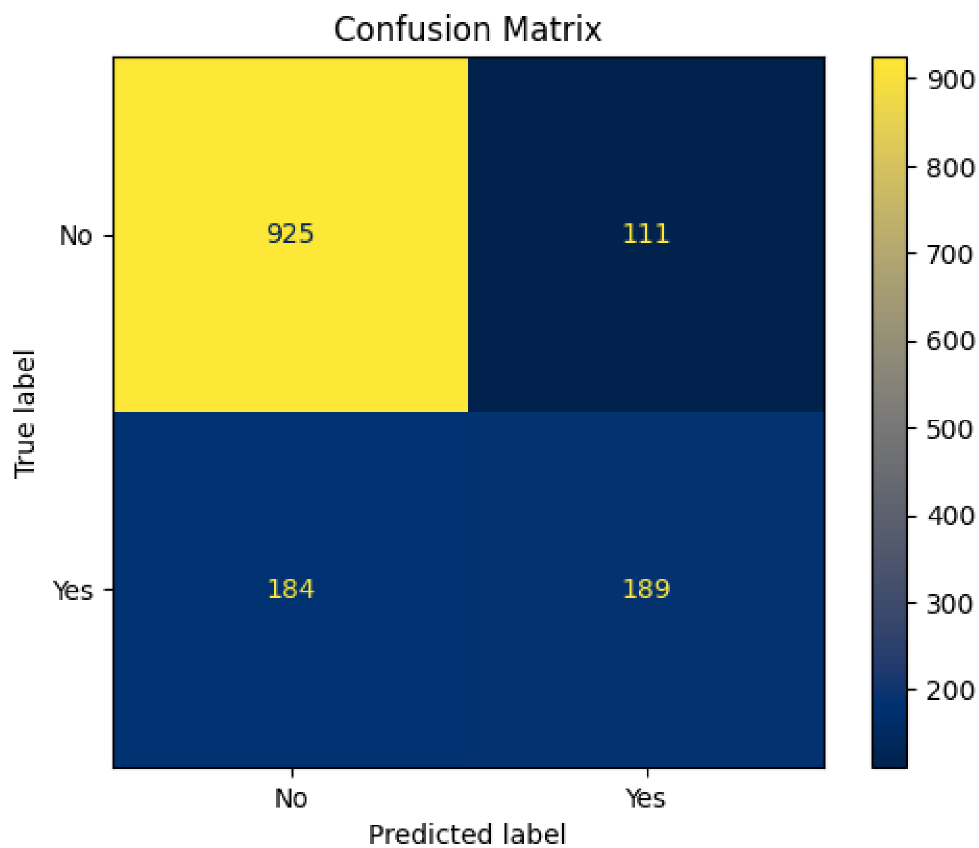
```
Accuracy: 0.7906
```

The accuracy is good, but the data is imbalanced and the accuracy may be from the model simply predicting 'No' for churn.

[37]:
```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the display
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["No", "Yes"])

# Plot the display
disp.plot(cmap='cividis')
plt.title("Confusion Matrix")
plt.show()
```



From the Confusion Matrix, the probability of the model making correct predictions for whether a customer will churn is akin to a coin-toss.

[38]:
```
# Classification report
print("Classification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1036
           1       0.63      0.51      0.56       373

    accuracy                           0.79      1409
   macro avg       0.73      0.70      0.71      1409
weighted avg       0.78      0.79      0.78      1409
```

For Not Churn (0):

`Precision: 83% of the people predicted not to churn actually did not churn.`

```
Recall: 89% of the people who actually churned were predicted correctly as not churning.
```

```
F1 Score: 0.86 means there is a good balance of precision and recall.
```

For Churn (1):

```
Precision: 63% of the people predicted to churn actually did churn.
```

```
Recall: 51% of churners were predicted correctly by the model.
```

```
F1 Score: 0.56 is average. It means there is a tradeoff between missing churners and falsely fl
```

[39]:
```
# AUC ROC score
y_pred_prob = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

```
ROC-AUC: 0.8364365418654962
```

The ROC AUC score is high and means that the model is good at distinguishing between the 2 classes.

3.6 Hyperparameter Tuning

We will use GridSearch to try and identify optimal hyperparameters for the model to try and improve performance.

"' # Find Hyperparameters param_grid = { 'learning_rate': [0.01, 0.05, 0.1, 0.2], 'max_depth': [3, 6, 10], 'n_estimators': [50, 100, 200], 'subsample': [0.8, 0.9, 1.0], 'colsample_bytree': [0.8, 0.9, 1.0], 'min_child_weight': [1, 3, 5] }

grid_search = GridSearchCV(estimator=xgb.XGBClassifier(eval_metric='auc', random_state=42), param_grid=param_grid, cv=5, n_jobs=-1) grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")

Best parameters: {'colsample_bytree': 0.9, 'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight': 3, 'n_estimators': 50, 'subsample': 1.0}

3.7 Fit the Model and Get Predictions

[40]:
```
# Initialize the classifier
model = xgb.XGBClassifier(
    colsample_bytree=0.9,
    learning_rate=0.1,
    max_depth=3,
    min_child_weight=3,
    n_estimators=50,
    subsample=1.0,
    eval_metric='auc',
    random_state=42
```

```
)

# Fit the model
model.fit(X_train, y_train)

# Predict labels for the test set
y_pred = model.predict(X_test)
```

3.8 Model Validation

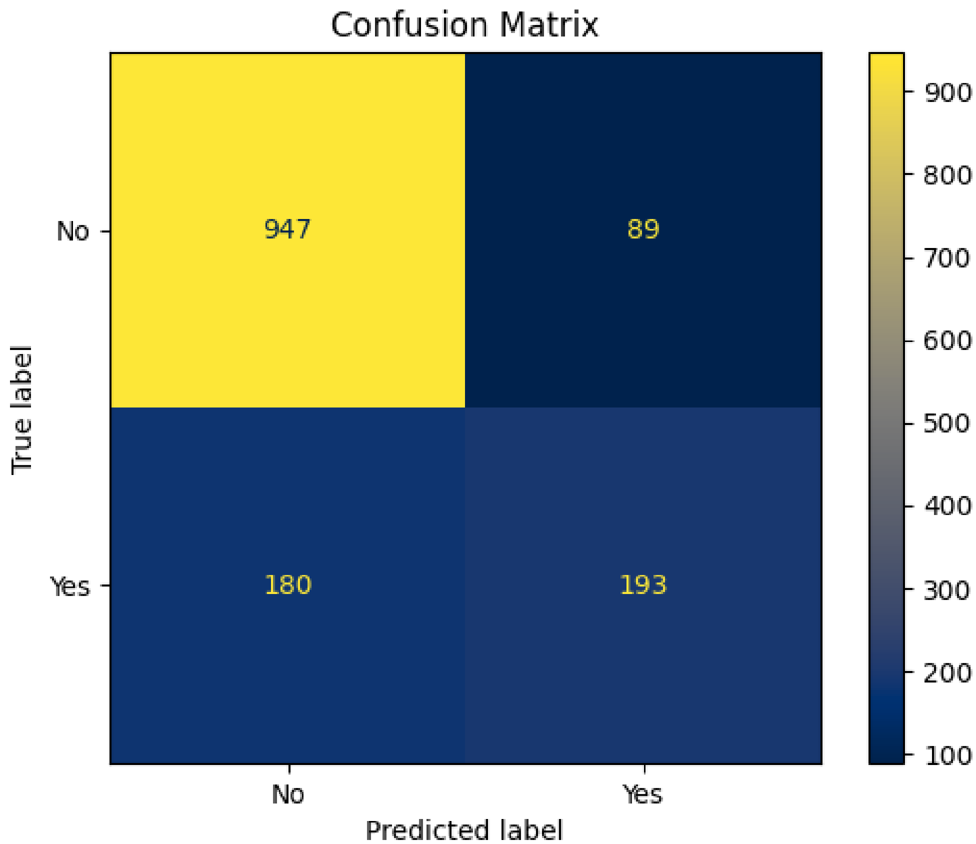
[41]:
```
# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.4f}")
```

```
Accuracy: 0.8091
```

[42]:
```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create the display
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["No", "Yes"])

# Plot the display
disp.plot(cmap='cividis')
plt.title("Confusion Matrix")
plt.show()
```



[43]:
```
# Classification report
print("Classification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.91      0.88      1036
           1       0.68      0.52      0.59       373

    accuracy                           0.81      1409
   macro avg       0.76      0.72      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

[44]:
```
# AUC ROC score
y_pred_prob = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

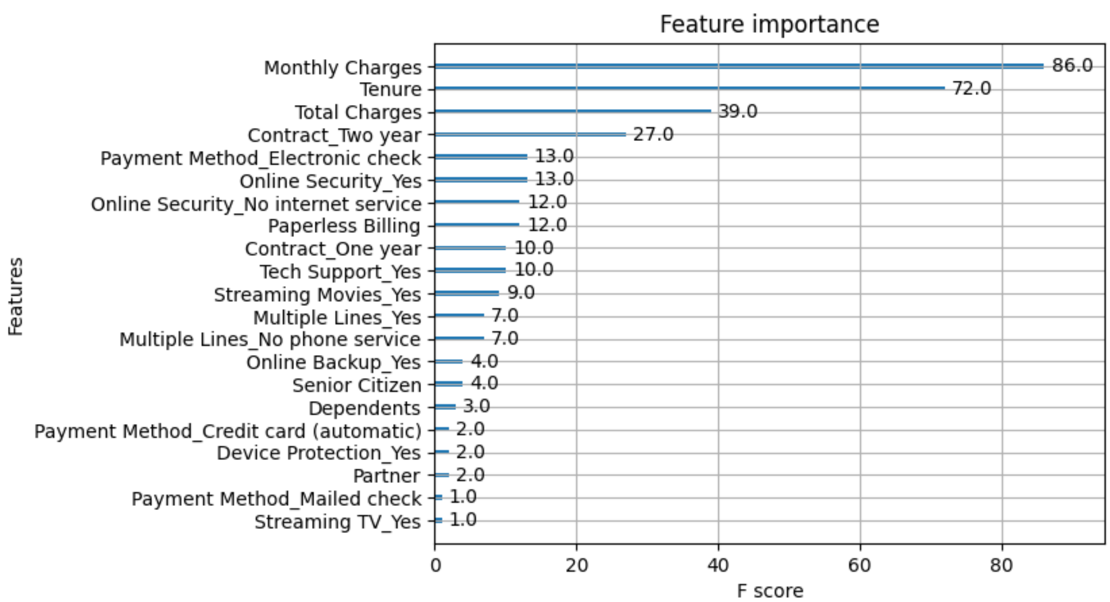
```
ROC-AUC: 0.861496837703272
```

We some small improvements across the board after doing some hyperparameter tuning. However, the improvement is only a few percentage points.

3.9 Feature Importance

[45]:
```python
# Plot Feature Importance
xgb.plot_importance(model)
```

[45]:
```
<Axes: title={'center': 'Feature importance'}, xlabel='F score',
ylabel='Features'>
```



The most important features in the model are Monthly Charges, Total Charges, and Tenure by far. Other features have less importance. If we add up all of the categories, Contract would come close in importance to Total Charges in importance.

4 Modeling With Fewer Features

4.1 Drop Unnecessary Features

The model may have too features. Removing some features that aren't necessary may improve performance. Let's remove Partner, Dependents, Phone Service, Online Security, Online Backup, Device Protection, Tech Support, Streaming TV, Streaming Movies, and Payment Method.

Phone Service is implied from values in Multiple Lines. Since we are removing the dependent features, we can leave Internet Service instead of removing it like before.

[46]:
```python
# Drop Columns
df_copy = df_copy.drop(columns=[
    'LoyaltyID',
    'Customer ID',
    'Partner',
    'Dependents',
    'Phone Service',
    'Online Security',
    'Online Backup',
    'Device Protection',
    'Tech Support',
    'Streaming TV',
    'Streaming Movies',
    'Payment Method',
])
```

[47]:
```python
# Print the first 5 rows of the data we'll be working with
df_copy.head()
```

[47]:
```
  Senior Citizen    Tenure    Multiple Lines Internet Service        Contract  \
0              No -1.644343  No phone service              DSL  Month-to-month
1              No  0.297205                No              DSL        One year
2              No -1.495444                No              DSL  Month-to-month
3              No  0.646327  No phone service              DSL        One year
4              No -1.495444                No      Fiber optic  Month-to-month

  Paperless Billing  Monthly Charges  Total Charges Churn
0               Yes        -1.158541      -1.805206    No
1                No        -0.239492       0.256861    No
2               Yes        -0.342665      -1.381338   Yes
3                No        -0.731079       0.235865    No
4               Yes         0.213545      -1.244141   Yes
```

4.2 Encode the Categorical Features

[48]:
```python
# Find Binary Columns
binary_cols = ['Senior Citizen', 'Paperless Billing', 'Churn']

# Encoding Binary Columns
for col in binary_cols:
    df_copy[col] = df_copy[col].map({'No': 0, 'Yes': 1})

# Encoding Other Columns
df_copy = pd.get_dummies(df_copy, columns=['Internet Service', 'Multiple␣
 ↪Lines', 'Contract'], drop_first=True, dtype=int)
```

```
[49]: # Print the first 5 rows of the encoded data frame
df_copy.head()
```

```
[49]:    Senior Citizen    Tenure  Paperless Billing  Monthly Charges  \
0                 0 -1.644343                  1        -1.158541
1                 0  0.297205                  0        -0.239492
2                 0 -1.495444                  1        -0.342665
3                 0  0.646327                  0        -0.731079
4                 0 -1.495444                  1         0.213545

   Total Charges  Churn  Internet Service_Fiber optic  Internet Service_No  \
0      -1.805206      0                             0                    0
1       0.256861      0                             0                    0
2      -1.381338      1                             0                    0
3       0.235865      0                             0                    0
4      -1.244141      1                             1                    0

   Multiple Lines_No phone service  Multiple Lines_Yes  Contract_One year  \
0                                1                   0                  0
1                                0                   0                  1
2                                0                   0                  0
3                                1                   0                  1
4                                0                   0                  0

   Contract_Two year
0                  0
1                  0
2                  0
3                  0
4                  0
```

4.3 Check Unique Values for Each Feature

```
[50]: # Print all unique values for each column
for col in df_copy.columns:
    print(f"{col}:")
    print(df_copy[col].unique())
```

```
Senior Citizen:
[0 1]
Tenure:
[-1.6443435   0.29720527 -1.49544443  0.64632675 -0.92628704 -0.16269601
 -0.78871081  0.08053824  1.10445427 -0.60622285 -0.44435051  1.00352171
  0.76159927 -0.03718286  1.27288016  0.84473766  1.31927073 -0.2065339
 -0.66440415  0.15530631  0.70462831  1.34219847 -0.39396295  0.04208131
 -1.16936305  0.67564931 -0.72512119  1.29616592  1.12912864  0.58660355
 -0.49640807  1.05444632 -0.34510326  1.20189603 -0.8555903  -1.37221793
  0.19168912  0.78961168  1.15359039  0.95163168 -1.00148358  0.5561775
```

```
  0.33125677  0.7332748   0.11826746  1.17784456  0.43037665  1.2494094
  0.22744746  0.9253082   0.3978215   0.36478829  0.52535695 -1.08208957
 -1.26512968  0.26261052  1.22574947 -0.11990163  0.97770058  1.07956199
 -0.55030087 -0.25148998  0.8718688   0.4941272   1.02910155 -0.07808394
  0.61664894 -0.29764818  0.89872305  0.81732147  0.00285584 -1.84408848
  0.46247264]
Paperless Billing:
[1 0]
Monthly Charges:
[-1.1585412  -0.23949171 -0.34266505 … -0.03598427 -0.66670841
  0.4741192 ]
Total Charges:
[-1.80520648  0.25686123 -1.38133786 … -0.85408054 -0.91709707
  1.48322527]
Churn:
[0 1]
Internet Service_Fiber optic:
[0 1]
Internet Service_No:
[0 1]
Multiple Lines_No phone service:
[1 0]
Multiple Lines_Yes:
[0 1]
Contract_One year:
[0 1]
Contract_Two year:
[0 1]
```

4.4 Feature Selection

[51]:
```
# Set independent variables
X2 = df_copy.drop(columns='Churn')

# Set dependent variable
y2 = df_copy['Churn']

# Split the data at a 70:30 ratio and set the random state for reproducibility
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size = 0.
  ↪2, random_state = 42)
```

4.5 Hyperparameter Tuning

"‘ # Find Hyperparameters param_grid = { 'learning_rate': [0.01, 0.05, 0.1, 0.2], 'max_depth': [3, 6, 10], 'n_estimators': [50, 100, 200], 'subsample': [0.8, 0.9, 1.0], 'colsample_bytree': [0.8, 0.9, 1.0], 'min_child_weight': [1, 3, 5] }

grid_search = GridSearchCV(estimator=xgb.XGBClassifier(eval_metric='auc', random_state=42), param_grid=param_grid, cv=5, n_jobs=-1) grid_search.fit(X2_train, y2_train)

print(f"Best parameters: {grid_search.best_params_}")

Best parameters: {'colsample_bytree': 0.9, 'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight': 1, 'n_estimators': 50, 'subsample': 1.0}

4.6 Fit the Model

[52]:
```
# Initialize the classifier
model2 = xgb.XGBClassifier(
    colsample_bytree=0.9,
    learning_rate=0.1,
    max_depth=3,
    min_child_weight=1,
    n_estimators=50,
    subsample=1.0,
    eval_metric='auc',
    random_state=42
)

# Fit the model
model2.fit(X2_train, y2_train)

# Predict labels for the test set
y2_pred = model2.predict(X2_test)
```

4.7 Model Validation

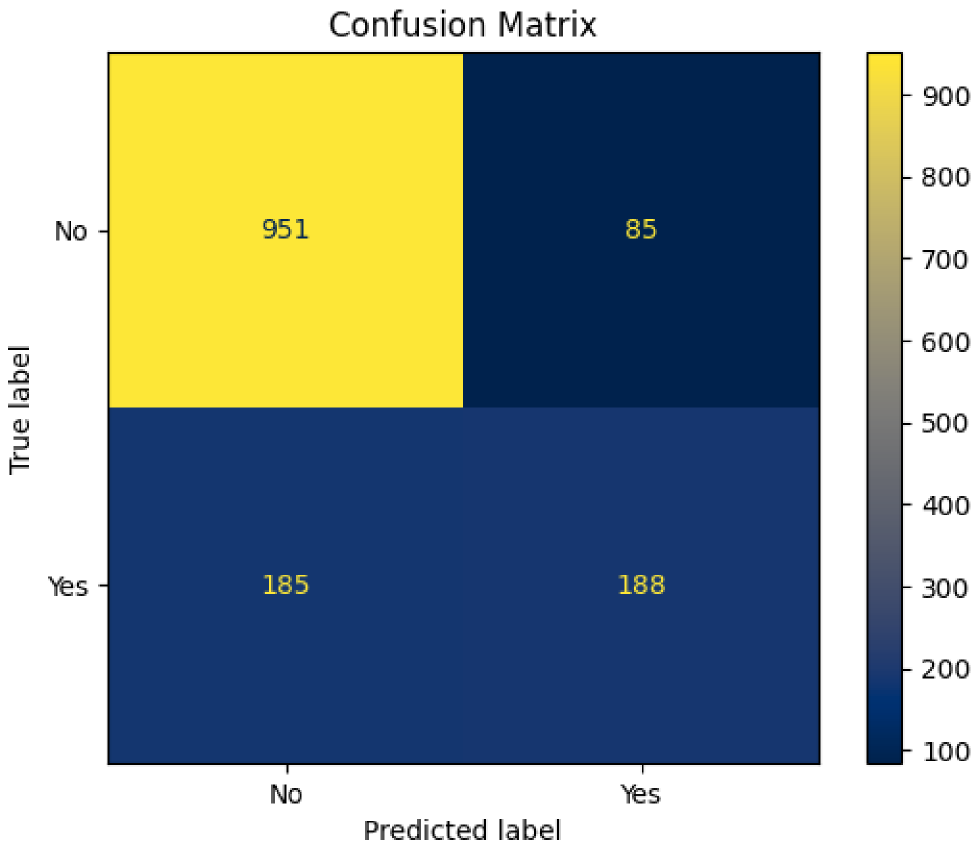
[53]:
```
# Accuracy
acc = accuracy_score(y2_test, y2_pred)
print(f"Accuracy: {acc:.4f}")
```

```
Accuracy: 0.8084
```

[54]:
```
# Create the confusion matrix
cm = confusion_matrix(y2_test, y2_pred)

# Create the display
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["No", "Yes"])

# Plot the display
disp.plot(cmap='cividis')
plt.title("Confusion Matrix")
plt.show()
```



[55]:
```
# Classification report
print("Classification Report:\n")
print(classification_report(y2_test, y2_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1036
           1       0.69      0.50      0.58       373

    accuracy                           0.81      1409
   macro avg       0.76      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

This model has similar performance overall compared to the hyperparameter tuned model. Unfortunately, it has a 2% decrease in Recall for churners, which is probably the most important metric for the model. The previous model would be preferred.

[56]:
```
# AUC ROC score
y2_pred_prob = model2.predict_proba(X2_test)[:, 1]
roc_auc = roc_auc_score(y2_test, y2_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

```
ROC-AUC: 0.8616119949900111
```

The ROC-AUC is slightly higher than the previous model and better distinguishes between classes.

Overall, this model would probably not be preferred over the previous model because we are most interested in correctly predicting churn.

5 Balanced Data Model

We will now use Synthetic Minority Oversampling Technique (SMOTE), a technique to deal with imbalanced data by generating some synthetic data for the minorities from existing data. Since the data is imbalanced, let's see if SMOTE brings any improvements to model performance. We will use the feature set from Model 1, as it had the best performance so far.

5.1 Resampling Data Using SMOTE

[57]:
```
# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to training data
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

5.2 Hyperparameter Tuning

"' # Find Hyperparameters param_grid = { 'learning_rate': [0.01, 0.05, 0.1, 0.2], 'max_depth': [3, 6, 10], 'n_estimators': [50, 100, 200], 'subsample': [0.8, 0.9, 1.0], 'colsample_bytree': [0.8, 0.9, 1.0], 'min_child_weight': [1, 3, 5] }

grid_search = GridSearchCV(estimator=xgb.XGBClassifier(eval_metric='error', random_state=42), param_grid=param_grid, cv=5, n_jobs=-1) grid_search.fit(X_resampled, y_resampled)

print(f"Best parameters: {grid_search.best_params_}")

Best parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 1, 'n_estimators': 100, 'subsample': 0.9}

5.3 Fit the Model and Get Predictions

5.4 Model Validation

[58]:
```
# Set the Hyperparameters for the SMOTE model
smote_model = xgb.XGBClassifier(
    colsample_bytree=0.8,
    learning_rate=0.1,
    max_depth=10,
```

```
    min_child_weight=1,
    n_estimators=100,
    subsample=0.9,
    eval_metric='error',
    random_state=42
)

# Fit the model
smote_model.fit(X_resampled, y_resampled)

# Predict labels for the test set
y_pred_smote = smote_model.predict(X_test)
```

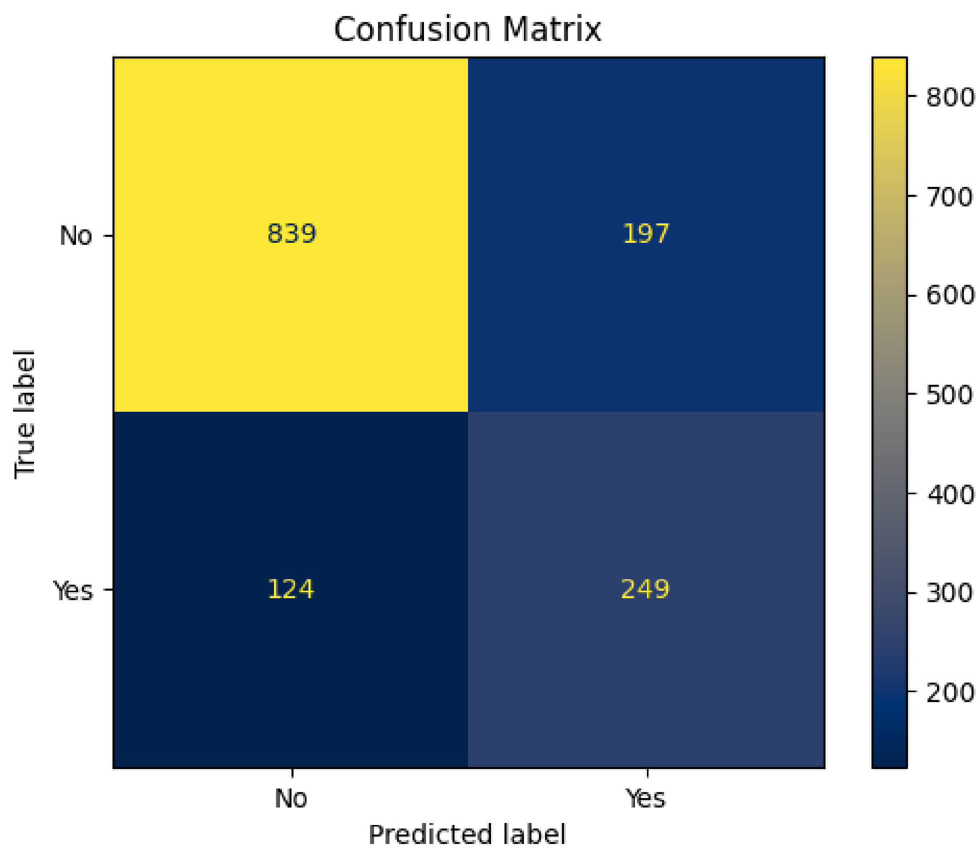
[59]:
```
# Accuracy
acc = accuracy_score(y_test, y_pred_smote)
print(f"Accuracy: {acc:.4f}")
```

```
Accuracy: 0.7722
```

[60]:
```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred_smote)

# Create the display
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["No", "Yes"])

# Plot the display
disp.plot(cmap='cividis')
plt.title("Confusion Matrix")
plt.show()
```



[61]:
```
# Classification report
print("Classification Report:\n")
print(classification_report(y_test, y_pred_smote))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.87      0.81      0.84      1036
           1       0.56      0.67      0.61       373

    accuracy                           0.77      1409
   macro avg       0.71      0.74      0.72      1409
weighted avg       0.79      0.77      0.78      1409
```

The model after using SMOTE has poor accuracy and recall when predicting whether a customer didn't churn. However, while the precision decreased for churning customers, the recall increased by 15%.

Overall, accuracy had decreased and the model sacrificed some performance when predicting if a customer didn't churn in exchange for catching more people who actully churned.

[62]:
```
# AUC ROC score
y_pred_prob = smote_model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

```
ROC-AUC: 0.8296306168290084
```

The ROC AUC score is slightly worse than the previous models but better than the one that did not have hyperparameter tuning done. The Score is still good.

6 Conclusion

An XGBoost Model was created for this imbalanced Telecom Customer Churn Dataset. The 2 best models were the models that had hyperparameter tuning and SMOTE. The model performed better with all of the reasonable features for predicting churn in the data. Removing features had slightly decreased performance by a negligible amount.

Between the 2 best models, the one with SMOTE makes a tradeoff between correctly predicting which customers will not churn for better performance in identifying customers that do churn. Considering that the customers that do churn are a minority, this model would likely be preferred by business.

7043 observations is small and a better model could be made with more data.