ChurnRadar

August 3, 2023

1 Customer Churn Prediction for Telecommunications Company

[1]:
```
# !pip install --upgrade scikit-learn
import pandas
import random

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()

import matplotlib.pyplot as plot
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

import pickle
import requests
```

1.1 Step 1: Data Collection

[2]:
```
# # Creating a custom dataset using random function of 5000 instances.
# customerData = []
# for id in range(1000, 6000):
#     temp = []
#     temp.append(id) # Customer ID
#     # Null values will be assigned a lower weight than rest of the data, and␣
↪deliberatly lower weights will be assigned to certain categories to actually␣
↪make conclusions.
#     temp.append(random.choices(["Male", "Female", None], weights=[62, 35,␣
↪3])[0]) # Gender
```

```
#     temp.append(random.choices([random.randint(18, 90), None], weights=[90,␣
↪2])[0]) # Age
#     # Only making the age and gender contain null values, because the␣
↪supposed company doesn't require to enter personal details when subscribing␣
↪for its services (It also covers both categorical and numerical variables).

#     temp.append(random.randint(1, 96)) # Service Length (Months)
#     temp.append(random.choices(["One Year", "Two Year", "Five Year",␣
↪"Month-to-Month"], weights=[35, 15, 5, 45])[0]) # Contract Type
#     temp.append(random.randint(1, 100)) # Monthly Charges
#     temp.append(random.randint(1000, 10000)) # Total Charges
#     temp.append(random.choices(["Yes", "No"], weights=[68, 42])[0]) # Churn
#     customerData.append(temp)

# customerData = pandas.DataFrame(customerData, columns = ['Customer ID',␣
↪'Gender', 'Age', 'Service Length', 'Contract Type', 'Monthly Charges',␣
↪'Total Charges', 'Churn'])

# # Taking a sample of 30 rows from customer Data to create duplicate rows
# duplicationSample = customerData.sample(50, replace = False)
# customerData = pandas.concat([customerData, duplicationSample], ignore_index␣
↪= True)

# customerData.to_csv("data/CommLink_Telecom_Customer_Data.csv", index = False)

customerData = pandas.read_csv(r"data/CommLink_Telecom_Customer_Data.csv")
customerData = customerData.drop(columns = 'Customer ID')

customerData
```

[2]:

```
      Gender   Age  Service Length   Contract Type  Monthly Charges  \
0     Female  38.0              79  Month-to-Month               17
1     Female  80.0              43        One Year               56
2       Male  40.0              70  Month-to-Month                9
3       Male  34.0              78  Month-to-Month               89
4     Female  46.0              57        Two Year               89
…        …     …               …               …                …
5045    Male  39.0              27        One Year               83
5046  Female  35.0              16        One Year                7
5047    Male  53.0              69  Month-to-Month               51
5048    Male  79.0              27        One Year               97
5049    Male  57.0              48  Month-to-Month               84

      Total Charges Churn
0              6538   Yes
1              4264   Yes
2              3277   Yes
```

```
3                8182   Yes
4                4381    No
…              …   …
5045             9855   Yes
5046             3475   Yes
5047             8156   Yes
5048             3987   Yes
5049             6480   Yes

[5050 rows x 7 columns]
```

1.2 Step 2: Data Cleaning

[3]:
```python
# Checking for null and duplicate values

print(customerData.isna().sum().sum(), "null values found!")
if customerData.isna().sum().sum():
    for column in customerData.columns:
        # This line checks whether data type of the column is 'f' (float) or␣
 ↪'i' (integer)
        if customerData[column].dtype.kind in 'fi':
            customerData[column].fillna(customerData[column].median(), inplace␣
 ↪= True)
        # This line checks whether data type of the column is 'O' (object or␣
 ↪categorical)
        elif customerData[column].dtype.kind in 'O':
            customerData[column].fillna(customerData[column].mode()[0], inplace␣
 ↪= True)
    print("Null values imputed!")
    print(customerData.isna().sum().sum(), "null values left!\n")

print(customerData.duplicated().sum(), "duplicates found!")
if customerData.duplicated().sum():
    customerData = customerData.drop_duplicates().reset_index(drop = True)
    print("Duplicate values dropped!")
    print(customerData.duplicated().sum(), "duplicates left!")
```

```
256 null values found!
Null values imputed!
0 null values left!

50 duplicates found!
Duplicate values dropped!
0 duplicates left!
```

1.3 Step 3: Exploratory Data Analysis

[4]:
```
customerData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           5000 non-null   object
 1   Age              5000 non-null   float64
 2   Service Length   5000 non-null   int64
 3   Contract Type    5000 non-null   object
 4   Monthly Charges  5000 non-null   int64
 5   Total Charges    5000 non-null   int64
 6   Churn            5000 non-null   object
dtypes: float64(1), int64(3), object(3)
memory usage: 273.6+ KB
```

[5]:
```
customerData.describe()
```

[5]:

	Age	Service Length	Monthly Charges	Total Charges
count	5000.000000	5000.000000	5000.000000	5000.00000
mean	54.077000	48.105400	50.861600	5430.27000
std	20.664193	28.032628	28.655399	2587.67477
min	18.000000	1.000000	1.000000	1001.00000
25%	36.000000	24.000000	26.000000	3220.00000
50%	54.000000	48.000000	51.000000	5410.50000
75%	72.000000	73.000000	76.000000	7618.25000
max	90.000000	96.000000	100.000000	10000.00000

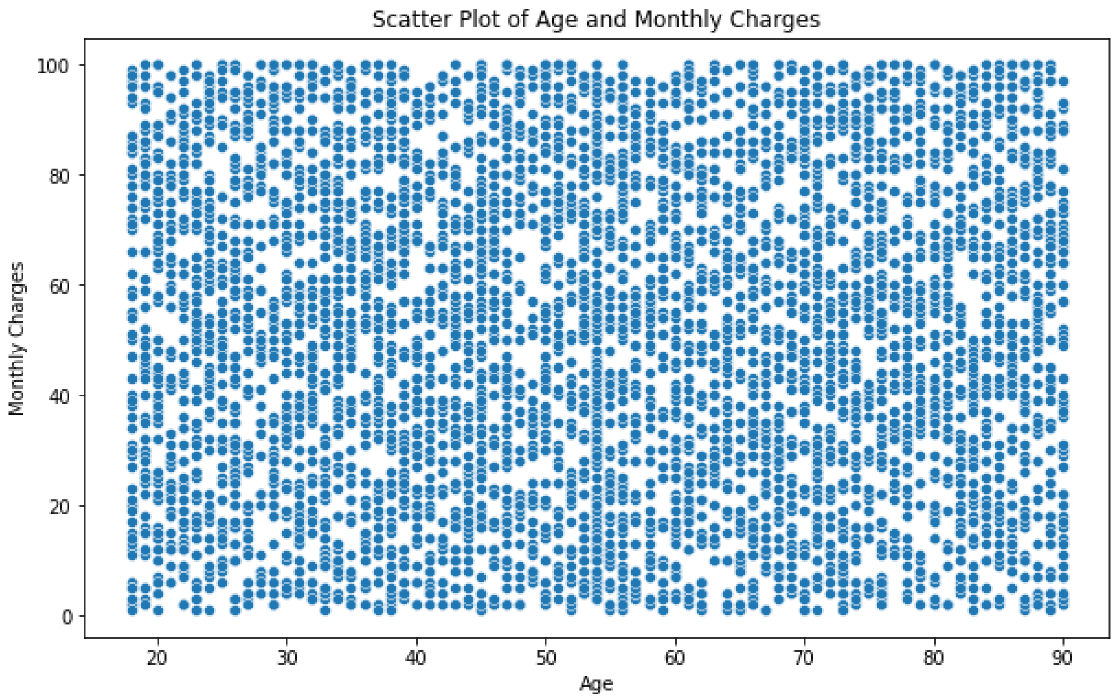
[6]:
```
customerData.describe(include = 'object')
```

[6]:

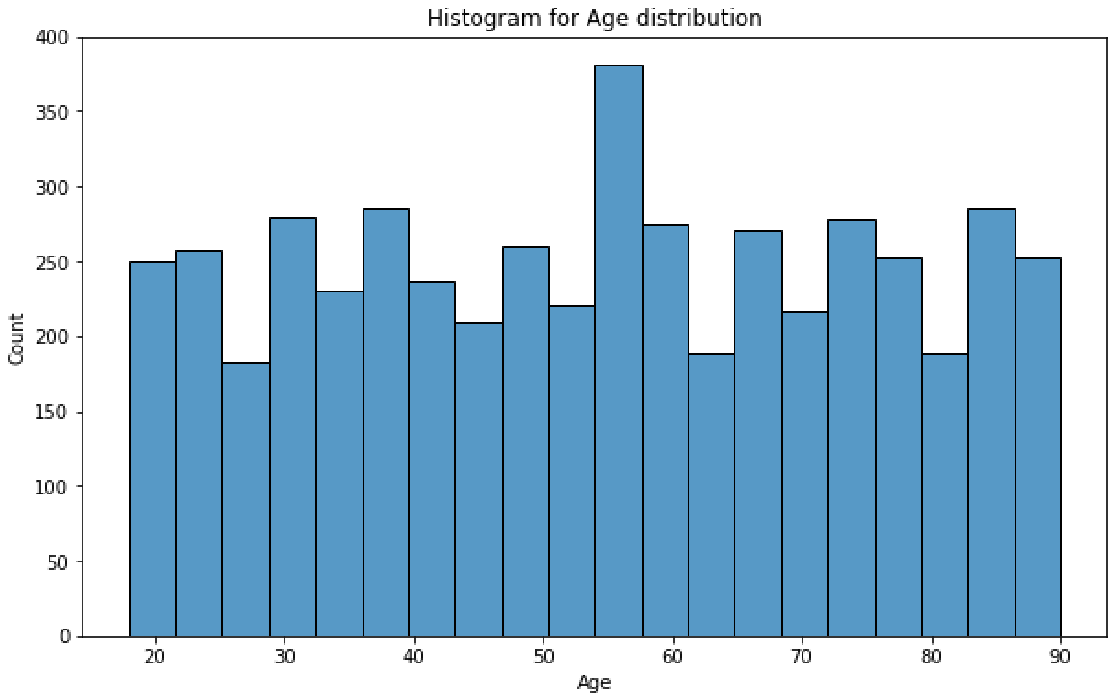
	Gender	Contract Type	Churn
count	5000	5000	5000
unique	2	4	2
top	Male	Month-to-Month	Yes
freq	3263	2283	3017

[7]:
```
# Plotting a scatter plot between Age and Monthly Charges
plot.figure(figsize = (10, 6))

sns.scatterplot(x = 'Age', y = 'Monthly Charges', data = customerData)

plot.xlabel('Age')
plot.ylabel('Monthly Charges')
plot.title('Scatter Plot of Age and Monthly Charges')
```

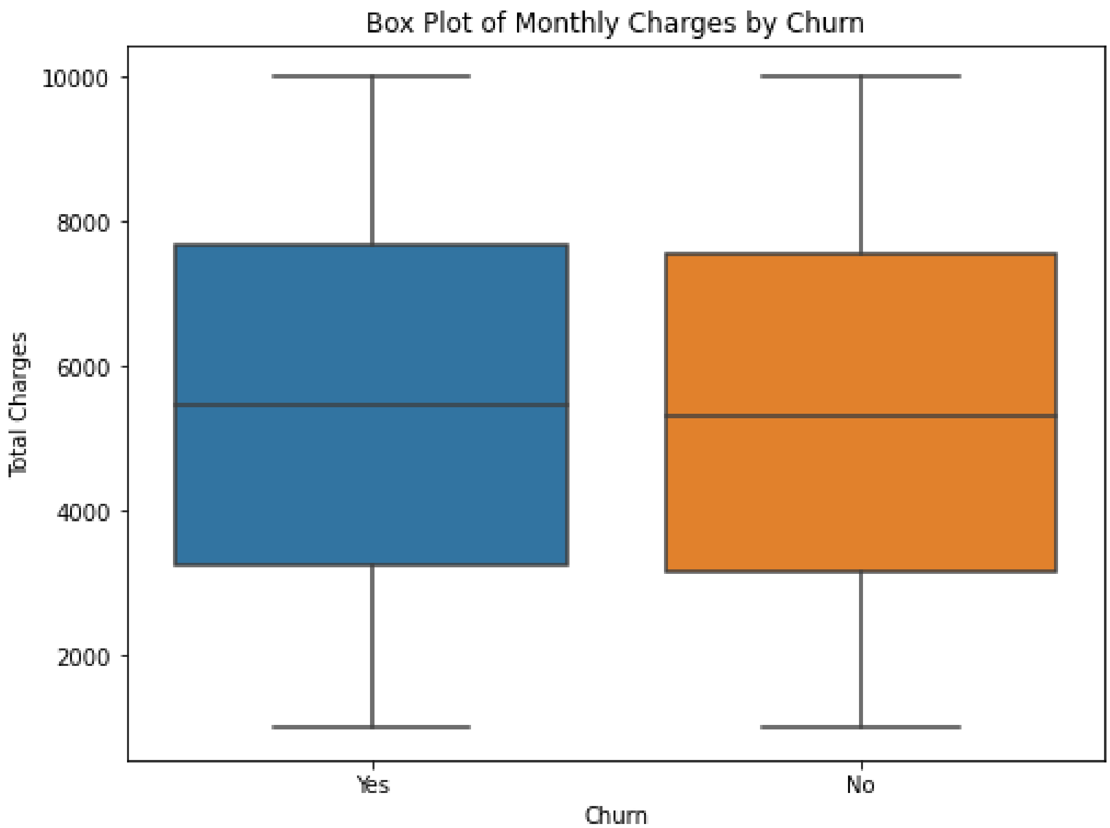
```
plot.show()
```



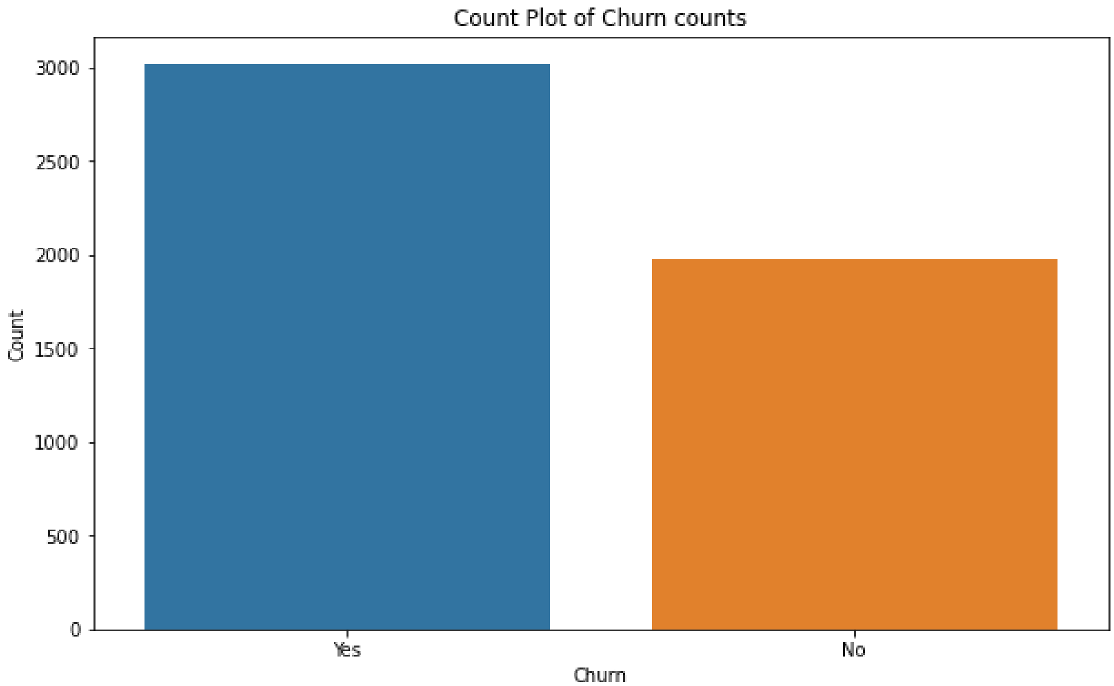
[8]:
```
# PLotting a histogram for Age distribution
plot.figure(figsize = (10, 6))

sns.histplot(customerData['Age'], bins = 20)

plot.xlabel('Age')
plot.ylabel('Count')
plot.title('Histogram for Age distribution')

plot.show()
```



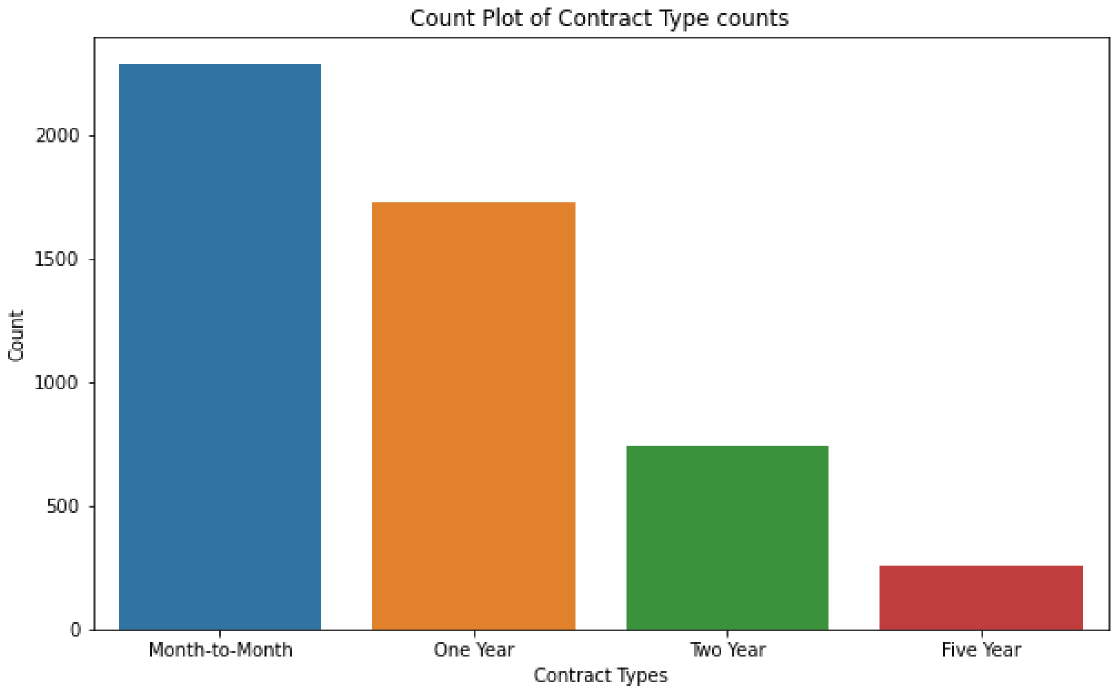
[9]:
```
# Plotting a box plot of Total Charges by Churn
plot.figure(figsize = (8, 6))

sns.boxplot(x = 'Churn', y = 'Total Charges', data = customerData)

plot.xlabel('Churn')
plot.ylabel('Total Charges')
plot.title('Box Plot of Monthly Charges by Churn')

plot.show()
```



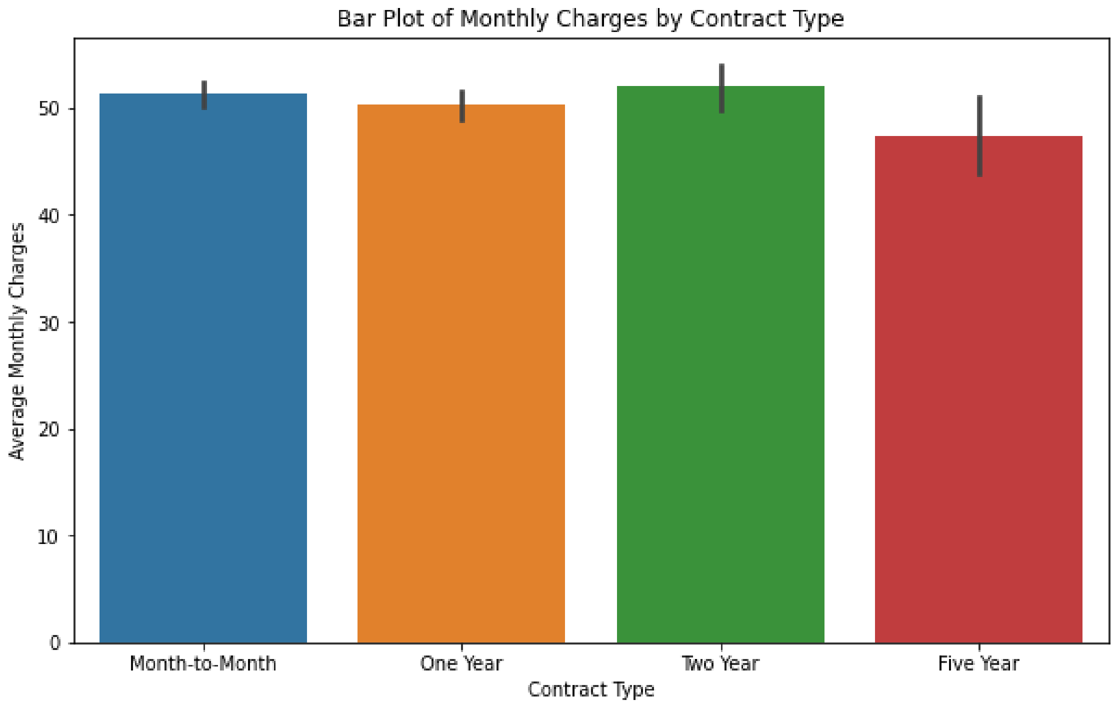
[10]:
```
# Plotting a count plot of amount of Churns
plot.figure(figsize = (10, 6))

sns.countplot(x = 'Churn', data = customerData)
plot.xlabel('Churn')
plot.ylabel('Count')
plot.title('Count Plot of Churn counts')

plot.show()
```



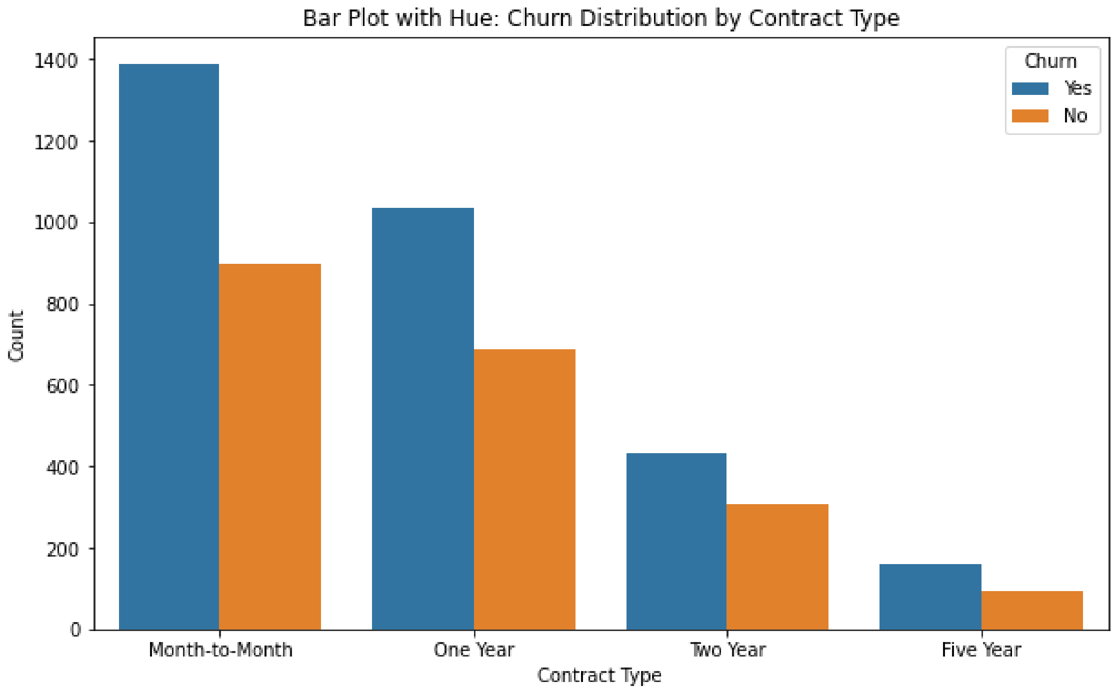
[11]:
```
# Plotting a count plot of each amount of Contract Types
plot.figure(figsize = (10, 6))

sns.countplot(x = 'Contract Type', data = customerData)
plot.xlabel('Contract Type')
plot.ylabel('Count')
plot.title('Count Plot of Contract Type counts')

plot.show()
```



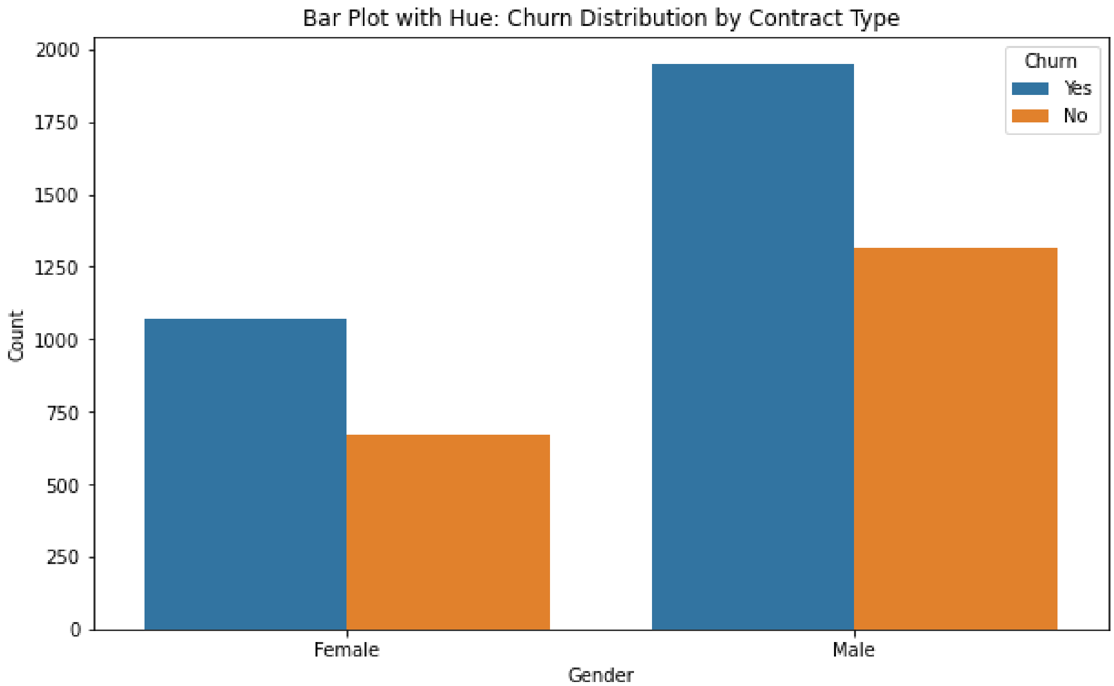
[12]:
```
# PLotting a bar plot of Contract Type with Monthly Charges
plot.figure(figsize=(10, 6))

sns.barplot(x='Contract Type', y='Monthly Charges', data=customerData)

plot.xlabel('Contract Type')
plot.ylabel('Average Monthly Charges')
plot.title('Bar Plot of Monthly Charges by Contract Type')

plot.show()
```



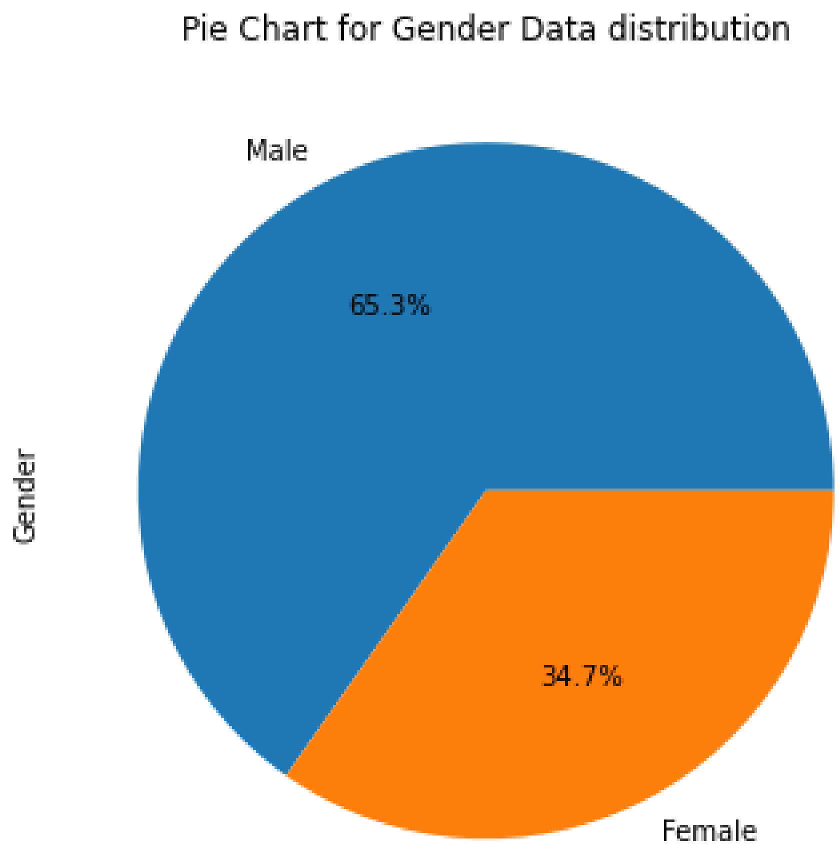
[13]:
```
# Plotting a count plot of Contract Types with Churn as hue
plot.figure(figsize = (10, 6))

sns.countplot(x = 'Contract Type', hue = 'Churn', data = customerData)

plot.xlabel('Contract Type')
plot.ylabel('Count')
plot.title('Bar Plot with Hue: Churn Distribution by Contract Type')

plot.show()
```



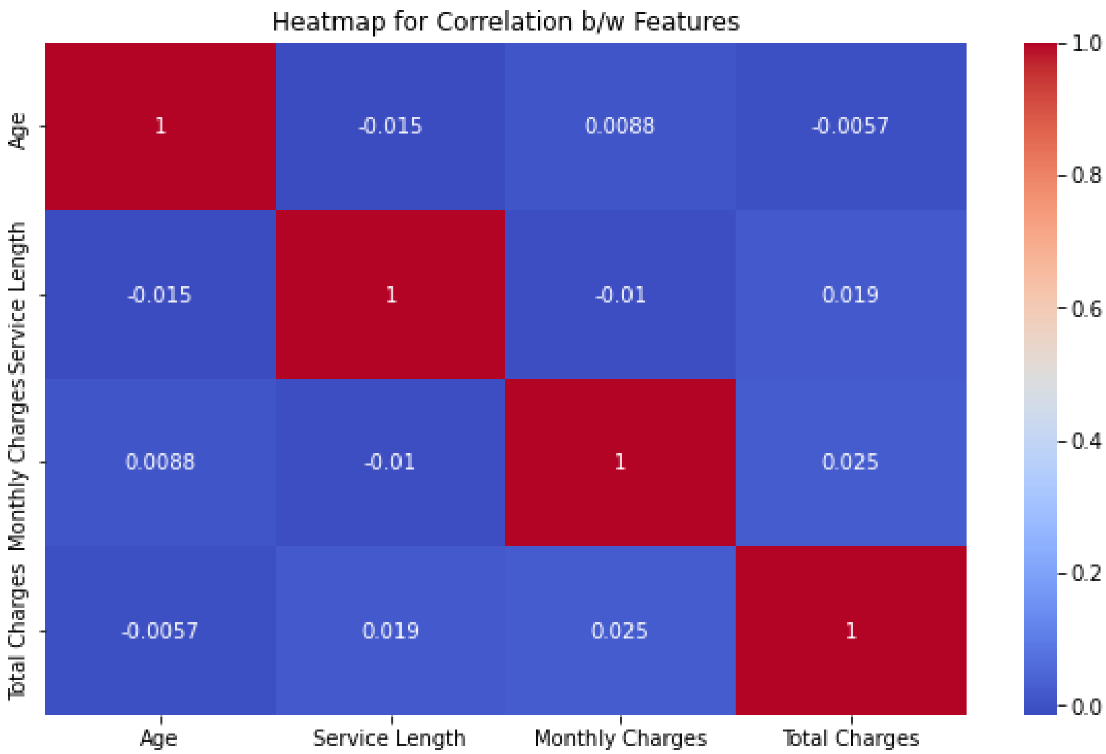
[14]:
```
# Plotting a count plot of Gender with Churn as hue
plot.figure(figsize = (10, 6))

sns.countplot(x = 'Gender', hue = 'Churn', data = customerData)

plot.xlabel('Gender')
plot.ylabel('Count')
plot.title('Bar Plot with Hue: Churn Distribution by Contract Type')

plot.show()
```



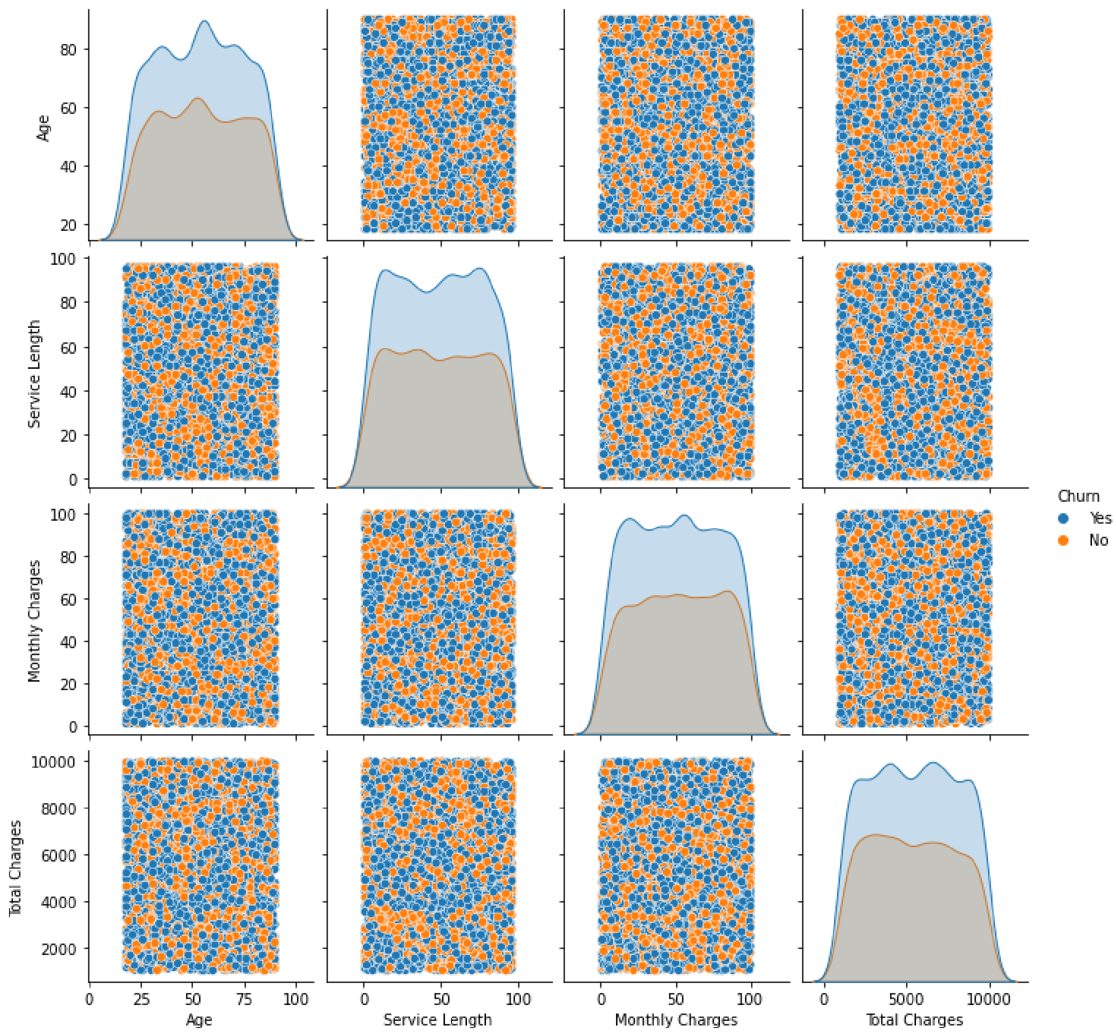
[15]:
```
# Plotting a pie chart to see gender distribution
plot.figure(figsize = (10, 6))

customerData['Gender'].value_counts().plot(kind = 'pie', autopct='%1.1f%%')
plot.title('Pie Chart for Gender Data distribution')

plot.show()
```



[16]:
```
# PLotting a heatmap for correlation between features
plot.figure(figsize = (10, 6))

sns.heatmap(customerData.corr(), annot = True, cmap = 'coolwarm')

plot.title('Heatmap for Correlation b/w Features')
plot.show()
```



[17]:
```
# Plotting a pair plot between different features with a hue of Churn
sns.pairplot(customerData, hue='Churn')

plot.show()
```



1.4 Step 4: Data Preprocessing

[18]:
```
# I will be applying Logistic regression to this dataset, so some␣
↪pre-processing steps are necessary.

#Logistic regression is sensitive to the scale of features, so it's essential␣
↪to apply feature scaling to numerical columns.
numerical = ['Age', 'Service Length', 'Monthly Charges', 'Total Charges']
customerData[numerical] = scaler.fit_transform(customerData[numerical])

# K-means algorithm cannot directly handle categorical variables, so I will␣
↪need to encode them into numerical representations. I will use label␣
↪encoding, which maps each category to a unique integer.
categorical = ['Gender', 'Contract Type']
customerData[categorical] = encoder.fit_transform(customerData[categorical])
```

```
# Preproccesed dataset
customerData
```

[18]:
```
      Gender       Age  Service Length  Contract Type  Monthly Charges  \
0        0.0  0.277778        0.821053            1.0         0.161616
1        0.0  0.861111        0.442105            2.0         0.555556
2        1.0  0.305556        0.726316            1.0         0.080808
3        1.0  0.222222        0.810526            1.0         0.888889
4        0.0  0.388889        0.589474            3.0         0.888889
…         …        …              …              …              …
4995     0.0  0.930556        0.505263            3.0         0.242424
4996     0.0  0.777778        0.368421            1.0         0.898990
4997     1.0  0.958333        0.600000            1.0         0.767677
4998     0.0  0.208333        0.526316            1.0         0.212121
4999     1.0  0.486111        0.768421            2.0         0.676768

      Total Charges Churn
0          0.615291   Yes
1          0.362596   Yes
2          0.252917   Yes
3          0.797978   Yes
4          0.375597    No
…               …    …
4995       0.757084   Yes
4996       0.616846   Yes
4997       0.254806    No
4998       0.001556    No
4999       0.347372   Yes

[5000 rows x 7 columns]
```

1.5 Step 5: Churn Prediction

[19]:
```
# Separating features and label from the dataset
features = customerData.drop(columns=['Churn'])
label = customerData['Churn']
```

[20]:
```
# Splitting the dataset using train-test split method by 75-25%
trainFeatures, testFeatures, trainLabel, testLabel = train_test_split(features,
 ↪label, test_size = 0.25, random_state = 19)

# Training the model
model.fit(trainFeatures, trainLabel)

# Making predictions
predictions = model.predict(testFeatures)
```

```
# Calculating accuracy
accuracy = accuracy_score(testLabel, predictions)
print("Accuracy:", accuracy)

# Creating a confusion matrix
conf_matrix = confusion_matrix(testLabel, predictions)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.6224

Confusion Matrix:
 [[  0 472]
 [  0 778]]
```

[21]:
```
# Combining the model, with the scaler and encoder to use same transformers to␣
↪create consistent results when send to the API.
modelTransformer = (model, scaler, encoder)

# Creating a pickle file of the model, so an API can be deployed, which takes␣
↪less time and doesn't need to perform all the steps every time.
with open('chum_predict.pkl', 'wb') as file:
    pickle.dump(modelTransformer, file)
```

1.6 Step 6: Create and Deploy an API

[ ]:
```
# This is a flask project. This is the code for an API (website) that take␣
↪requests at /process and return the predicted response (Yes or No). This is␣
↪only created on my local host and you can see the result of it in the cell␣
↪below.
import pickle, pandas
from flask import Flask, request, jsonify

app = Flask(__name__)

# Loading the prepared model's pickle file
with open('chum_predict.pkl', 'rb') as file:
    model, scaler, encoder = pickle.load(file)

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.json

        #Converting the JSON format to pandas dataframe for our model to read
        input_data = pandas.DataFrame([data])
```

```
        #Applying the appropriate transformation
        input_data[['Age', 'Service Length', 'Monthly Charges', 'Total␣
↪Charges']] = scaler.transform(input_data[['Age', 'Service Length', 'Monthly␣
↪Charges', 'Total Charges']])

        input_data[['Gender', 'Contract Type']] = encoder.
↪transform(input_data[['Gender', 'Contract Type']])

        #Rearranging the dataframe to match the column order when our model was␣
↪fit
        input_data = input_data[['Gender', 'Age', 'Service Length', 'Contract␣
↪Type', 'Monthly Charges',
       'Total Charges']]

        prediction = model.predict(input_data)

        if prediction[0] == 'Yes':
            return jsonify({'Churn': 'Yes'})
        elif prediction[0] == 'No':
            return jsonify({'Churn': 'No'})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    app.run(debug=True)
```

[22]:

```
# This is how the API is called.
data = {
    'Age': 35,
    'Service Length': 12,
    'Monthly Charges': 79,
    'Total Charges': 942,
    'Contract Type': 'One Year',
    'Gender': 'Female'
}

response = requests.post('http://127.0.0.1:5000/predict', json = data)
print("Data 1:", response.json())

data = {
    'Age': 67,
    'Service Length': 3,
    'Monthly Charges': 16,
    'Total Charges': 1520,
    'Contract Type': 'Month-to-Month',
    'Gender': 'Male'
```

```
}

response = requests.post('http://127.0.0.1:5000/predict', json = data)
print("Data 2:", response.json())

data = {
    'Age': 72,
    'Service Length': 12,
    'Monthly Charges': 305,
    'Total Charges': 3000,
    'Contract Type': 'Month-to-Month',
    'Gender': 'Male'
}

response = requests.post('http://127.0.0.1:5000/predict', json = data)
print("Data 3:", response.json())
```

```
Data 1: {'Churn': 'Yes'}
Data 2: {'Churn': 'Yes'}
Data 3: {'Churn': 'No'}
```

1.7 Step 7: EDA Summary Report

Here are some of the key insights: 1) Majority age category is close to 60, meaning most customers are around that age bracket of (50-60). 2) Most of the Chum values are Yes, meaning a lower customer retention. 3) Month-to-Month is the most popular contract, whereas Five-Year is least popular. That could be due to the long time commitment. 4) The customers are 'Male' dominated. 5) Female retention rate is lower than Male attrition rate.

Here are some recommendations: 1) Since female data churn rate is higher, company can offer targeted retention offers towards the female gender to cater their needs and preferences. 2) As customers like Month-to-Month contract, company can introduce additional incentives to longer-term contracts like One-Year or Two-Year by giving additional discounts or service options. 3) Five-Year contracts have the worst retention, so additional incentives on top of it, can be given to these customers, with offers which aren't offered by any other offer. 4) Since the majority of customers fall within the age bracket of 50-60, company should develop personalized retention strategies for this age group. 5) Actively seek customer feedback to identify areas for improvement. Prioritize issue resolution and customer support to address any dissatisfaction promptly.