D208 Performance Assessment Task II

Darian Gurrola

Course Instructor: Dr. Keiona Middleton

A1.

One of the challenges that many telecommunications companies face is customer retention. Customers commonly change providers for various reasons ranging from affordability to wider service offerings. In this analysis we will ask "What factors are most responsible for customer churn?". The independent variables below will be used in the analysis.

A2.

The goal of this analysis is to determine which factors have the most influence on customer churn. The telecommunications company will use this information to retain customers and prevent them seeking out competitors.

B1.

Logistic regression is one of the most useful tools for predicting binary outcomes. In order to do this effectively, the model makes several assumptions. These assumptions are independence of observations, absence of multicollinearity in independent variables, absence of extreme outliers, and a linear relationship between independent variables and the logit of the dependent variable (Statology, 2020).

Two of these assumptions, independence of observations and absence of extreme outliers, are very straightforward. Independence of observations means that the each data point in the dataset must be unrelated. Absence of extreme outliers is required to prevent the results of a regression from being skewed.

Lack of multicollinearity in the independent variables is a another critical component of logistic regression. Multicollinearity occurs when there is high correlation between multiple independent variables. If this is not dealt with, it can cause the regression model to make unreliable predictions.

The last assumption is that there is a linear relationship between the independent variables and the logit of the dependent variable. The logit, or log odds, is the logarithm of the odds of the probability of an event occurring. This is important because the goal of logistic regression is to predict a binary outcome.

B2.

For this analysis, I will be using Python to perform multiple logistic regression. Python is one of the most popular tools for predictive modeling and machine learning. One of the benefits of this language is the wide variety of libraries. I plan on using several libraries throughout the different phases of the analysis. Pandas will be used to import and manipulate data from the "churn_clean" csv file. Numpy will be used to perform statistical calculations. Seaborn and Matplotlib will be used to generate visualizations for each of the variables. Lastly, Scikit-learn will be used to develop the logistic regression model.

Another benefit of python is its speed. Although R has the advantage of being specialized towards data science, Python is able to render data at a much faster speed (Turing). This should prove very useful when performing the complex calculations required for logistic regression.

B3.

Multiple logistic regression is the appropriate technique for this analysis because it can be used to analyze the relationship between independent variables and a dependent categorical variable (Walwadkar, 2022). This method is effective because the dependent variable, "churn", has only two possible values, "Yes" and "No". Multiple logistic regression will allow us to utilize a variety of continuous and categorical variables to predict a binary outcome.

C1.

Before performing the regression analysis, the data must be sufficiently cleaned. This process will involve the detection treatment of duplicates, missing values, and outliers, as well as the re-expression of categorical variables. The churn data has been imported into a python variable named "df_churn".

The first part of the data cleaning process is to detect duplicates, missing values, and outliers. To identify duplicates in "df_churn", I combined the "duplicated()" and "value_counts()" methods from the pandas library. The resulting output indicated that there were no duplicate rows found in the dataset.

To detect missing values, I used the "isnull()" function from pandas, along with the "sum()" function on "df_churn". The output shows that there were 2,129 missing values in the InternetService variable. Lastly, I detected outliers by generating boxplots for each quantitative variable. This was done using the "boxplot()" function from the seaborn library. The resulting output showed that there were outliers in "Population", "Children", "Income", "Outage_sec_perweek", "Email", "Contacts", and "Yearly_equip_failure". To supplement the boxplots, I created a function called "boxplot_info()". This function accepts a variable from df_churn as an input and provides a detailed output of boxplot and outlier information.

The next step in the data cleaning process is to treat the data quality issues mentioned previously. Duplicates do not need to be treated because they are not present in the dataset. To treat the missing values in "InternetService" I used the "fillna()" method and imputed missing values with the mode. This was done because "InternetService" is a categorical variable.

After reviewing the boxplots for each numerical variable, I chose to retain all outliers. This was because the values were plausible and not extreme enough to exclude. I do not believe that retaining these values will violate the outlier assumption in B1. I also did not want to reduce the sample size or potentially introduce bias into the dataset.

Please see the annotated below, which was used to detect and treat data quality issues.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import
variance_inflation_factor
import sklearn

df_churn = pd.read_csv('churn_clean.csv')

#Detect duplicate rows in df_churn
print(df_churn.duplicated().value_counts())
```

```
False    10000
Name: count, dtype: int64
```

```
#Detect missing values in df_churn
df_churn.isnull().sum()
```

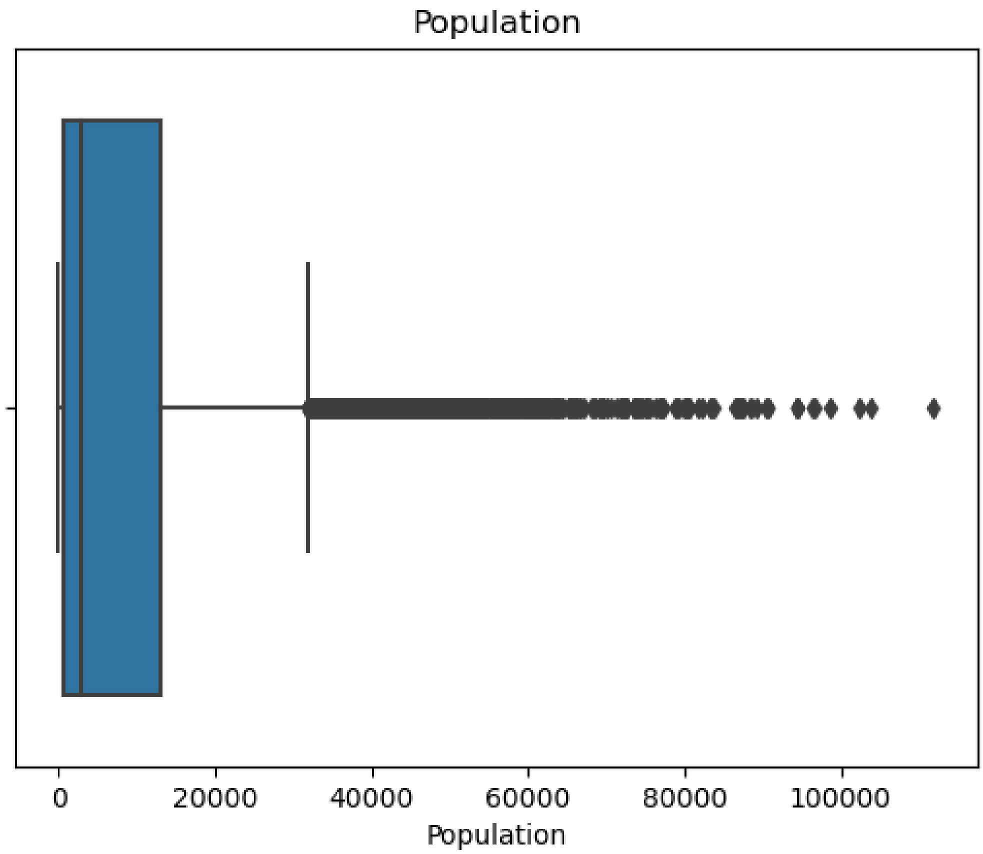
```
CaseOrder                  0
Customer_id                0
Interaction                0
UID                        0
City                       0
State                      0
County                     0
Zip                        0
Lat                        0
Lng                        0
Population                 0
Area                       0
TimeZone                   0
Job                        0
Children                   0
Age                        0
Income                     0
Marital                    0
Gender                     0
Churn                      0
Outage_sec_perweek         0
Email                      0
Contacts                   0
Yearly_equip_failure       0
Techie                     0
Contract                   0
Port_modem                 0
Tablet                     0
InternetService         2129
```

```
Phone                          0
Multiple                       0
OnlineSecurity                 0
OnlineBackup                   0
DeviceProtection               0
TechSupport                    0
StreamingTV                    0
StreamingMovies                0
PaperlessBilling               0
PaymentMethod                  0
Tenure                         0
MonthlyCharge                  0
Bandwidth_GB_Year              0
Item1                          0
Item2                          0
Item3                          0
Item4                          0
Item5                          0
Item6                          0
Item7                          0
Item8                          0
dtype: int64
```

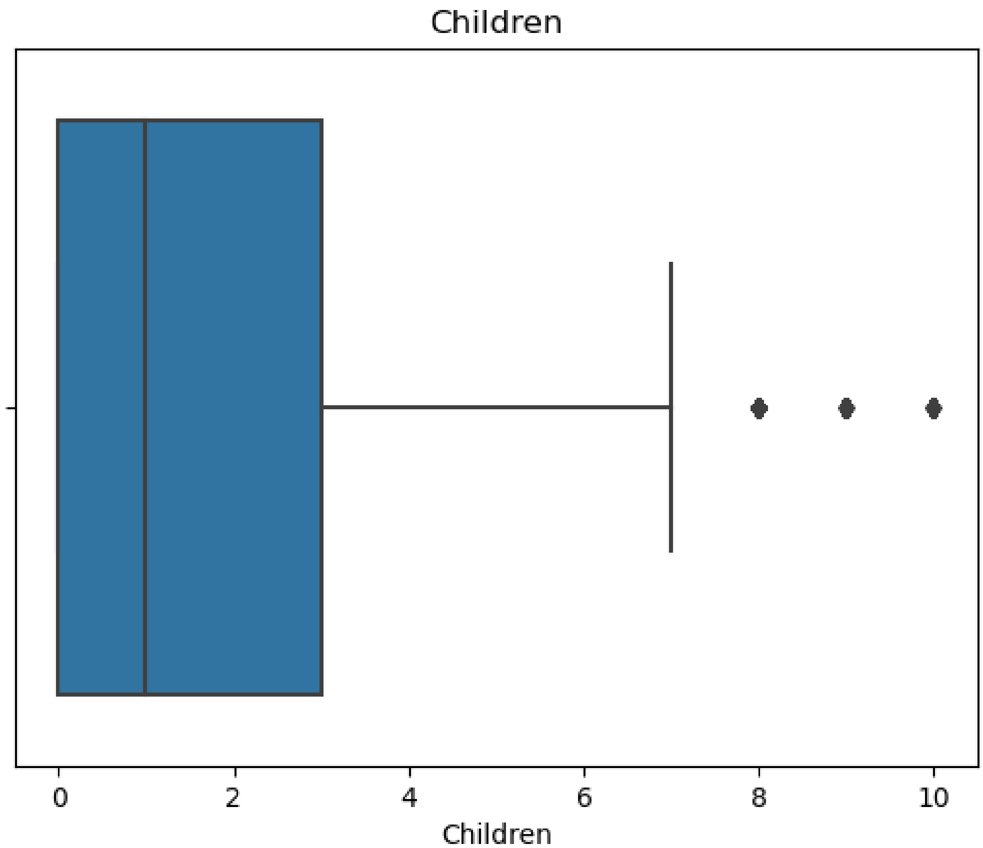
```
#Create function to provide boxplot information
def boxplot_info(input):

    #obtain values of column and ignore nulls
    data = input.dropna().values

    #generate q1 and q3 using pandas.DataFrame.quantile.
    q1 = input.quantile(0.25)
    print("Q1: " + str(q1))
    q3 = input.quantile(0.75)
    print("Q3: " + str(q3))

    #Calculate interquartile range for boxplot by subtracting Q1 from
Q3
    iqr = q3 - q1
    print("IQR: " + str(iqr))

    #Calculate whisker values of boxplot.
    whisker_lower = q1 - (1.5 * iqr)
    print("Lower Whisker: " + str(whisker_lower))
    whisker_upper = q3 + (1.5 * iqr)
    print("Upper Whisker: " + str(whisker_upper))

     #Find number of outliers outside of Q1 and Q3.  Print total
number of outliers in column.
    outliers_min = (input < whisker_lower).sum()
    print("Number of outliers lower than boxplot minimum: " +
```

```
str(outliers_min))
    outliers_max = (input > whisker_upper).sum()
    print("Number of outliers greater than boxplot maximum: " +
str(outliers_max))
    outliers_total = outliers_min + outliers_max
    print("Total number of Outliers: " + str(outliers_total))
    max_outlier = max(data)
    print("Highest Outlier: " + str(max_outlier))
    min_outlier = min(data)
    print("Lowest Outlier: " + str(min_outlier))

#Detect outliers in Population variable
population_boxplot = sns.boxplot(x="Population", data =
df_churn).set_title("Population")

#Generate boxplot info for Population using boxplot_info function
boxplot_info(df_churn['Population'])
```

```
Q1: 738.0
Q3: 13168.0
IQR: 12430.0
Lower Whisker: -17907.0
Upper Whisker: 31813.0
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 937
Total number of Outliers: 937
Highest Outlier: 111850
Lowest Outlier: 0
```

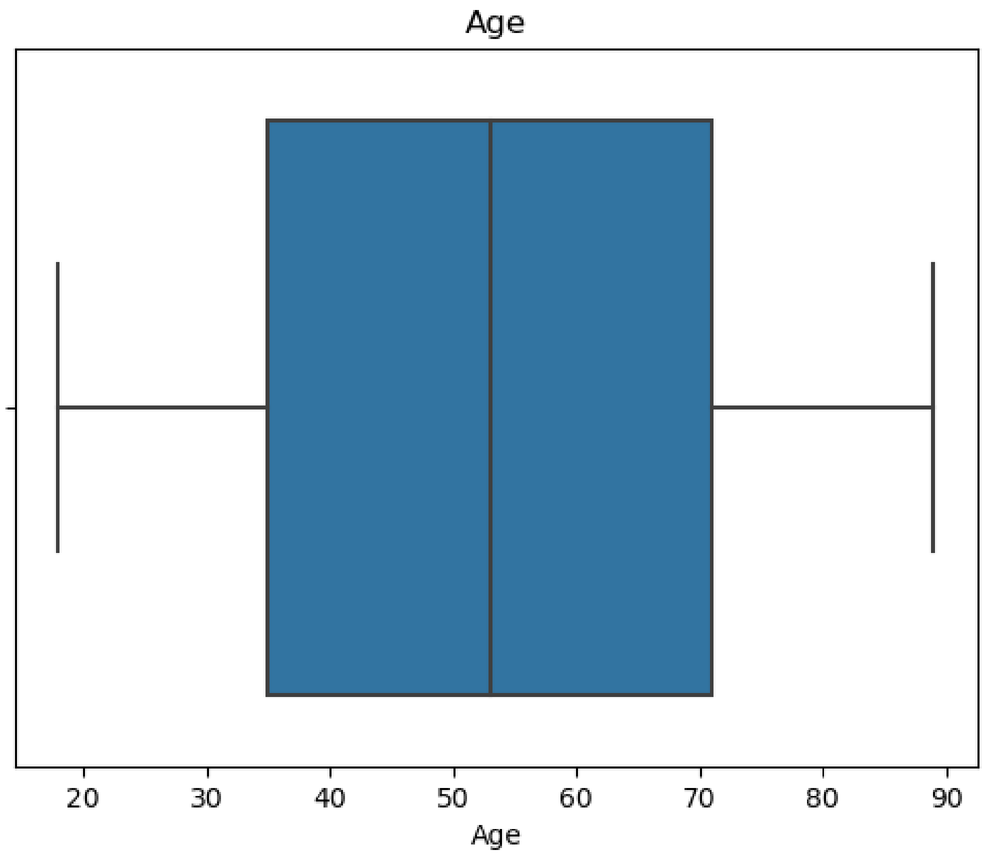



```
#Detect outliers in Children variable
population_boxplot = sns.boxplot(x="Children", data =
df_churn).set_title("Children")

#Generate boxplot info for Children using boxplot_info function
boxplot_info(df_churn['Children'])

Q1: 0.0
Q3: 3.0
IQR: 3.0
Lower Whisker: -4.5
Upper Whisker: 7.5
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 401
Total number of Outliers: 401
Highest Outlier: 10
Lowest Outlier: 0
```

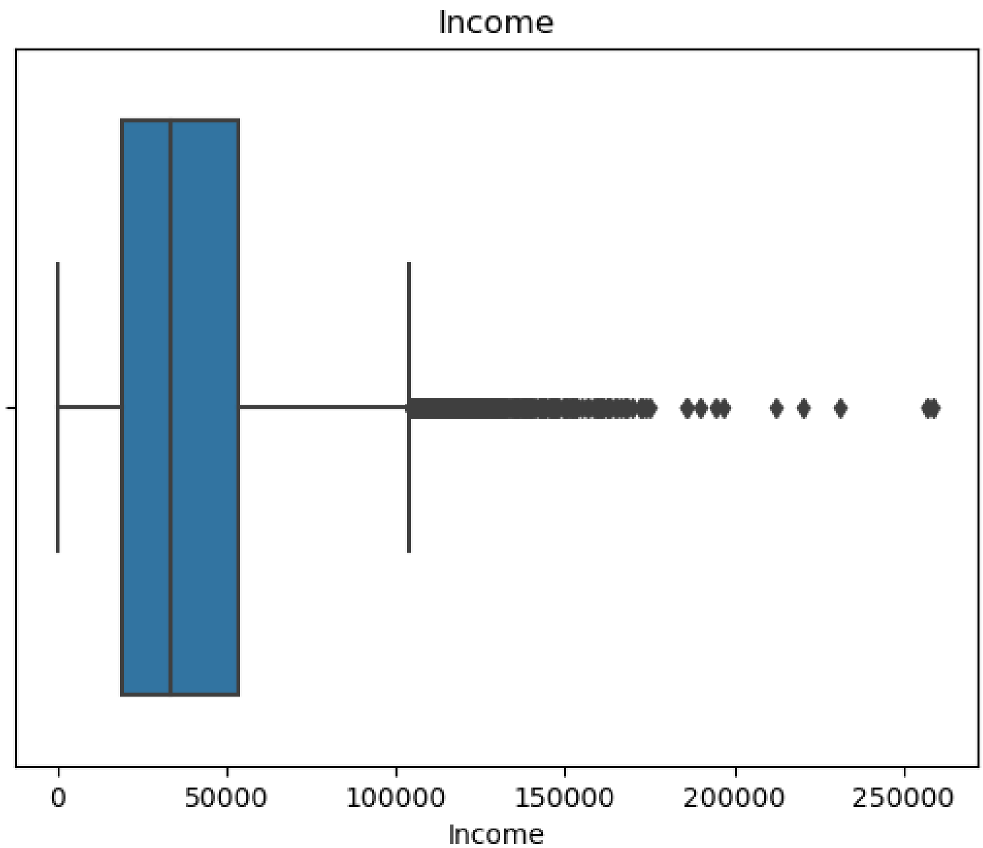



```
#Detect outliers in Age variable
population_boxplot = sns.boxplot(x="Age", data =
df_churn).set_title("Age")

#Generate boxplot info for Population using boxplot_info function
boxplot_info(df_churn['Age'])
```

```
Q1: 35.0
Q3: 71.0
IQR: 36.0
Lower Whisker: -19.0
Upper Whisker: 125.0
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 89
Lowest Outlier: 18
```

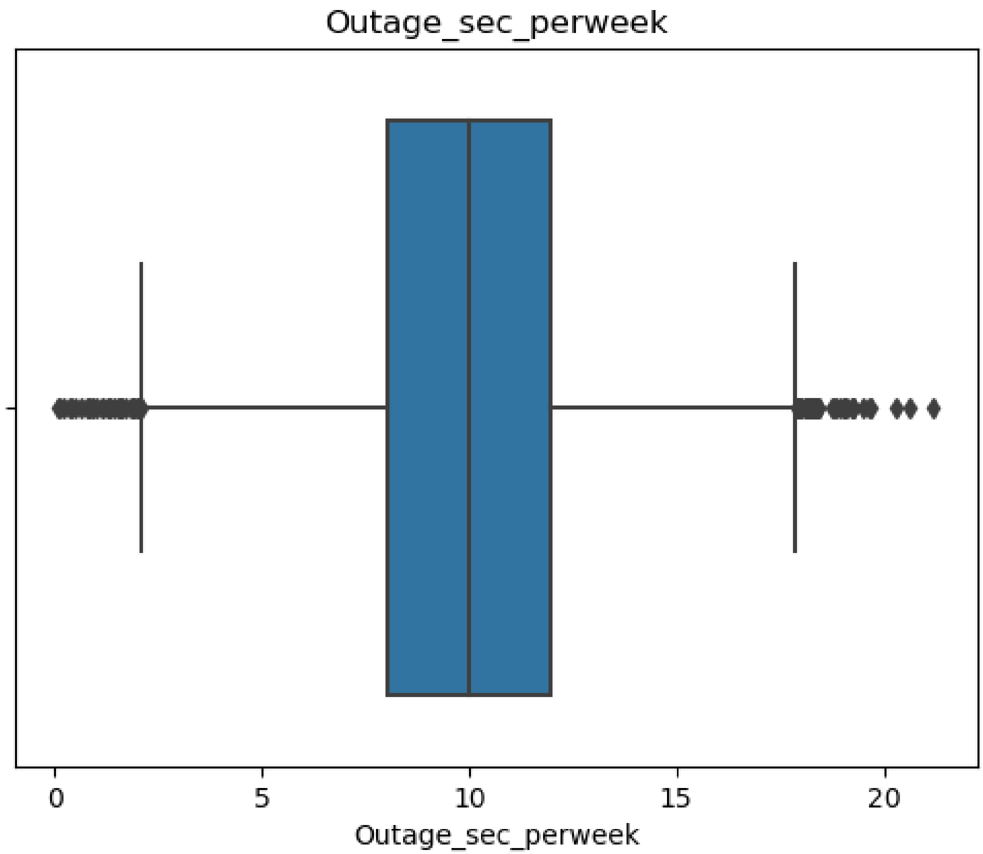



```
#Generate boxplot for Income variable
income_boxplot = sns.boxplot(x="Income", data =
df_churn).set_title("Income")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Income'])

Q1: 19224.7175
Q3: 53246.17
IQR: 34021.4525
Lower Whisker: -31807.46125
Upper Whisker: 104278.34875
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 336
Total number of Outliers: 336
Highest Outlier: 258900.7
Lowest Outlier: 348.67
```

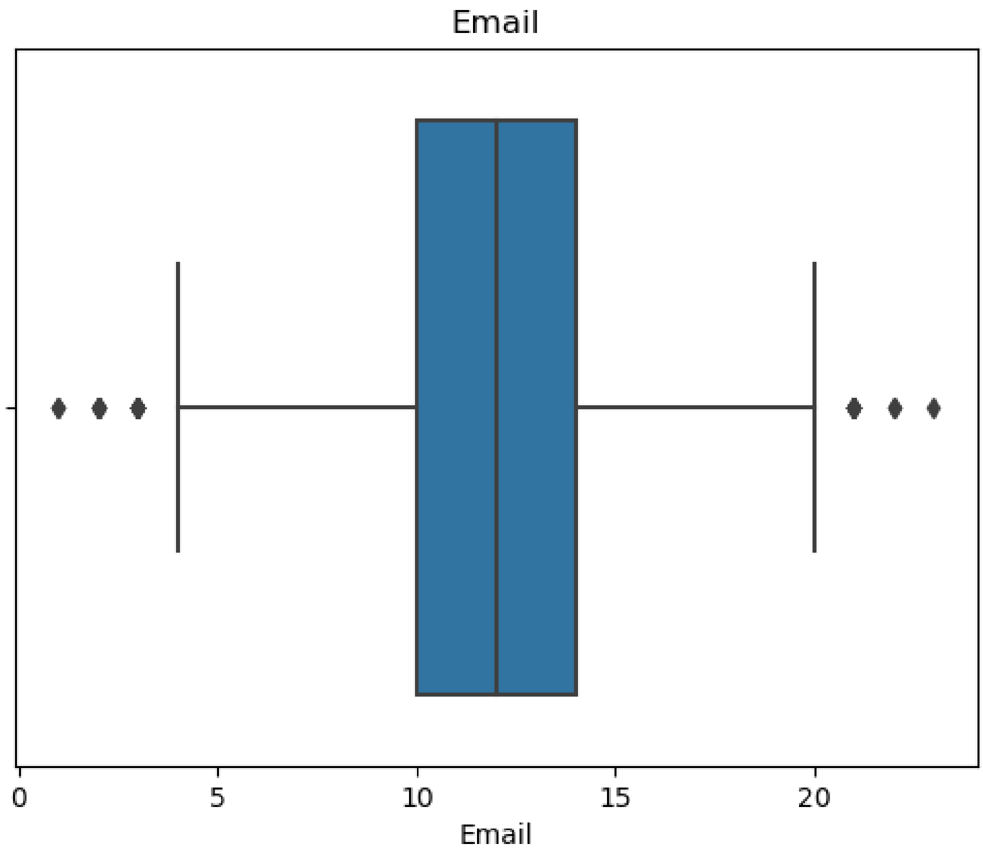



```
#Generate boxplot for Outage_sec_perweek variable
outage_boxplot = sns.boxplot(x="Outage_sec_perweek", data =
df_churn).set_title("Outage_sec_perweek")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Outage_sec_perweek'])
```

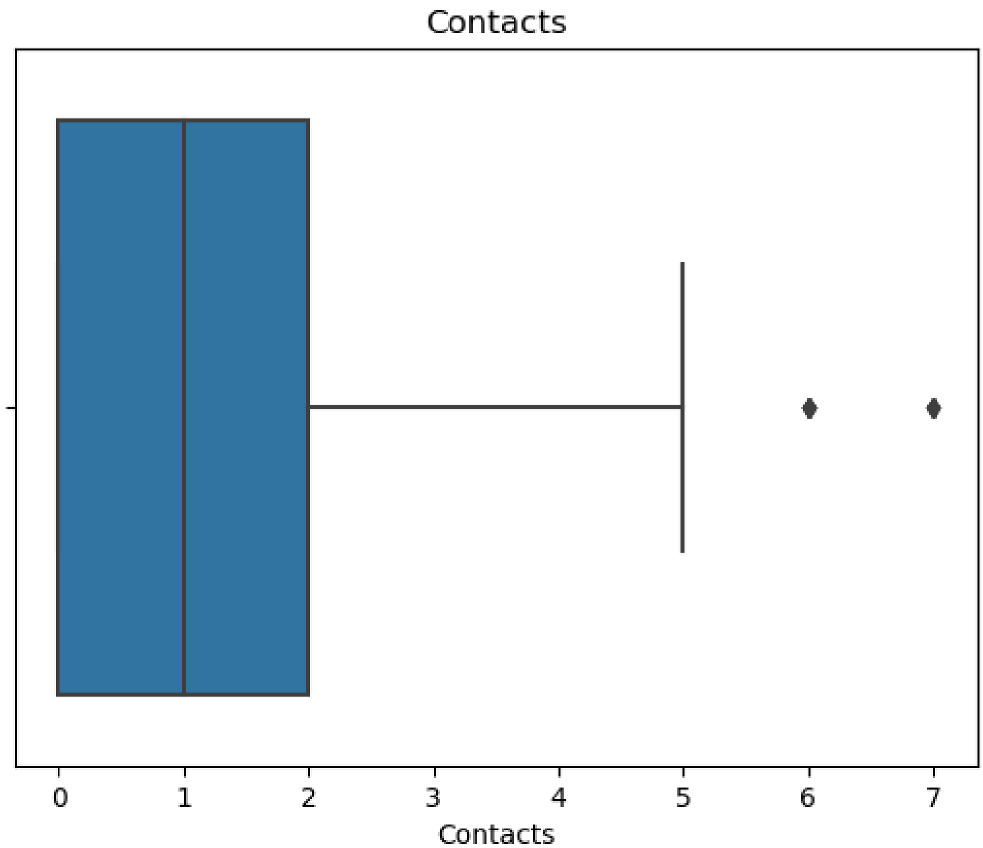
```
Q1: 8.018214
Q3: 11.969485
IQR: 3.951271
Lower Whisker: 2.0913075
Upper Whisker: 17.8963915
Number of outliers lower than boxplot minimum: 33
Number of outliers greater than boxplot maximum: 43
Total number of Outliers: 76
Highest Outlier: 21.20723
Lowest Outlier: 0.09974694
```




```
#Generate boxplot for Email variable
email_boxplot = sns.boxplot(x="Email", data =
df_churn).set_title("Email")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Email'])

Q1: 10.0
Q3: 14.0
IQR: 4.0
Lower Whisker: 4.0
Upper Whisker: 20.0
Number of outliers lower than boxplot minimum: 23
Number of outliers greater than boxplot maximum: 15
Total number of Outliers: 38
Highest Outlier: 23
Lowest Outlier: 1
```

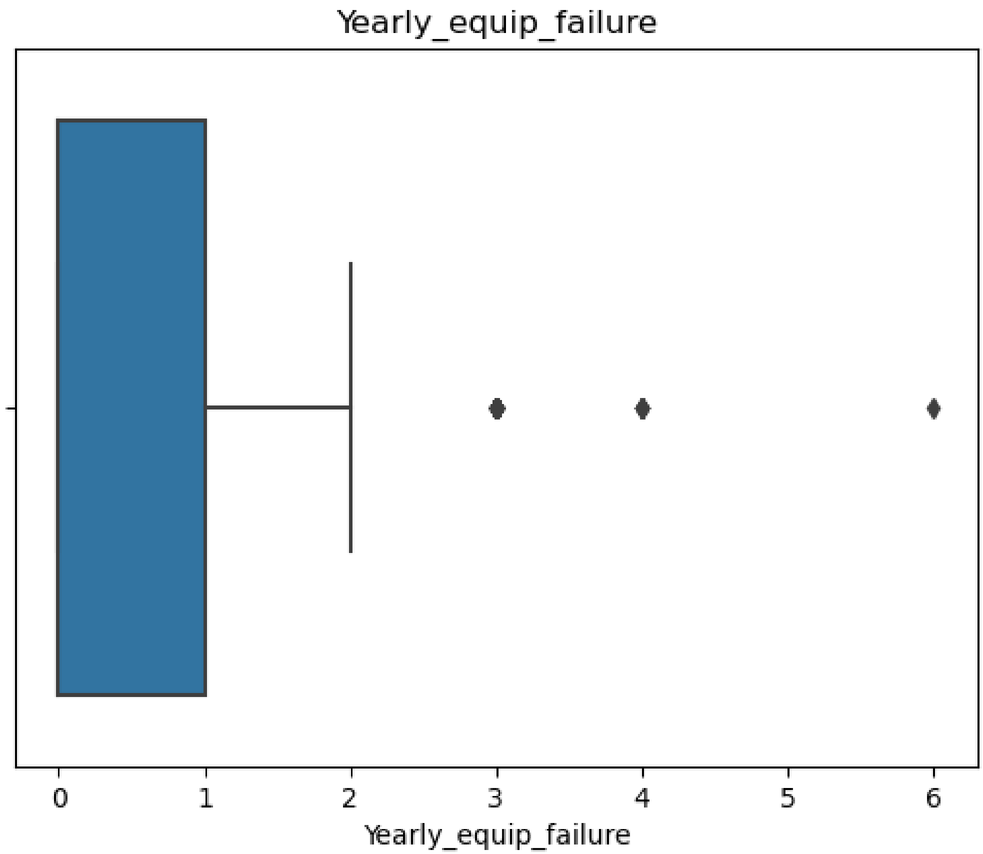



```
#Generate boxplot for Contacts variable
Contacts_boxplot = sns.boxplot(x="Contacts", data =
df_churn).set_title("Contacts")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Contacts'])

Q1: 0.0
Q3: 2.0
IQR: 2.0
Lower Whisker: -3.0
Upper Whisker: 5.0
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 8
Total number of Outliers: 8
Highest Outlier: 7
Lowest Outlier: 0
```



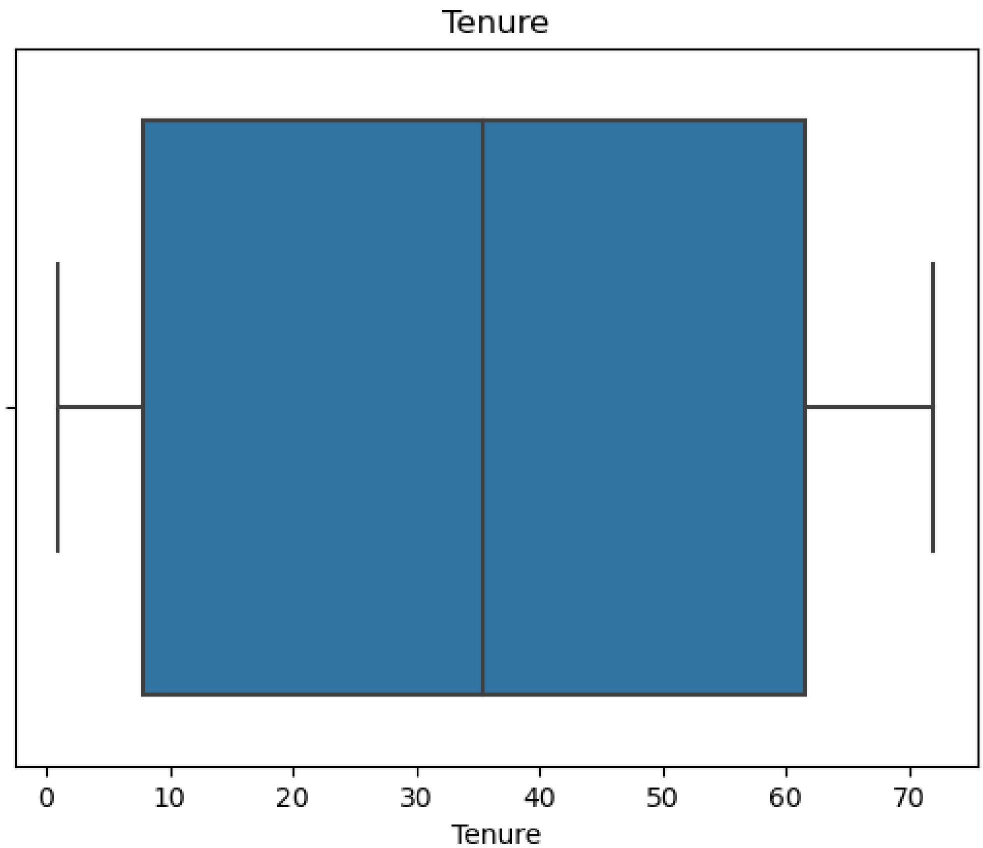

```
#Generate boxplot for Yearly_equip_failure variable
failure_boxplot = sns.boxplot(x="Yearly_equip_failure", data =
df_churn).set_title("Yearly_equip_failure")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Yearly_equip_failure'])
```

```
Q1: 0.0
Q3: 1.0
IQR: 1.0
Lower Whisker: -1.5
Upper Whisker: 2.5
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 94
Total number of Outliers: 94
Highest Outlier: 6
Lowest Outlier: 0
```



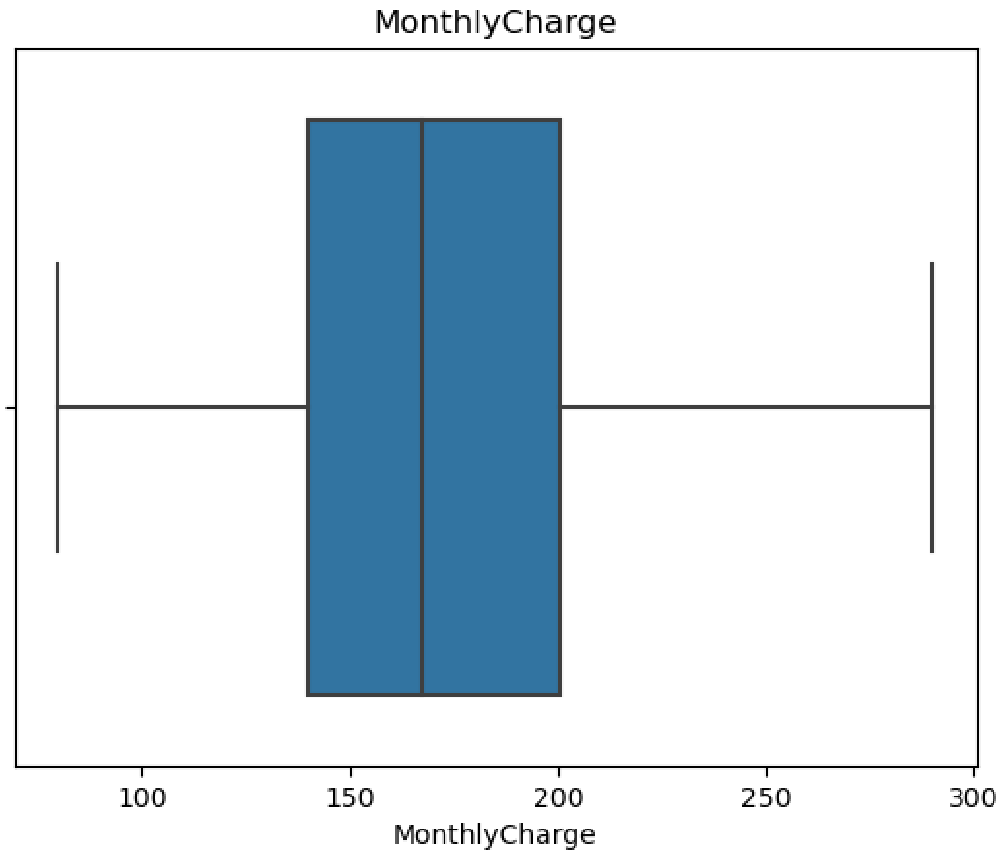

```
#Generate boxplot for Tenure variable
tenure_boxplot = sns.boxplot(x="Tenure", data =
df_churn).set_title("Tenure")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Tenure'])
```

```
Q1: 7.91769359175
Q3: 61.479795
IQR: 53.56210140825
Lower Whisker: -72.42545852062501
Upper Whisker: 141.822947112375
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 71.99928
Lowest Outlier: 1.00025934
```

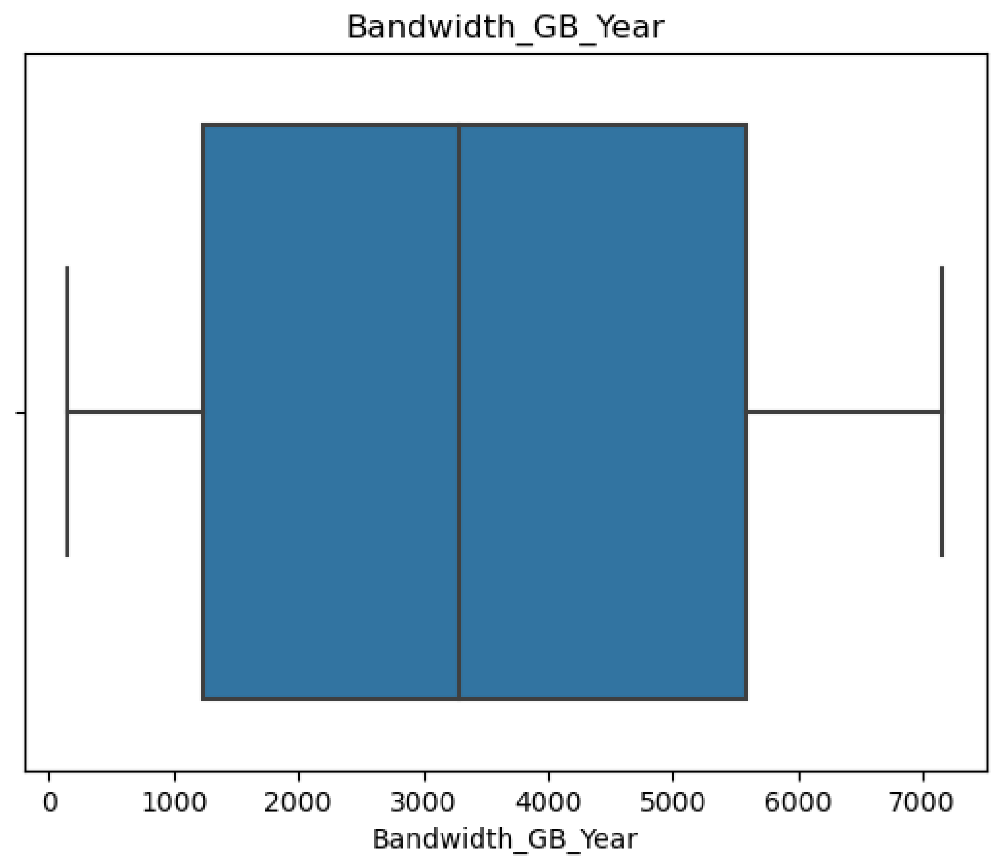



```
#Generate boxplot for MonthlyCharge variable
MonthlyCharge_boxplot = sns.boxplot(x="MonthlyCharge", data =
df_churn).set_title("MonthlyCharge")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['MonthlyCharge'])
```

```
Q1: 139.979239
Q3: 200.734725
IQR: 60.75548599999999
Lower Whisker: 48.84601000000002
Upper Whisker: 291.867954
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 290.160419
Lowest Outlier: 79.97886
```




```
#Generate boxplot for Bandwidth_GB_Year variable
bandwidth_boxplot = sns.boxplot(x="Bandwidth_GB_Year", data =
df_churn).set_title("Bandwidth_GB_Year")

#Generate boxplot info using boxplot_info() function
boxplot_info(df_churn['Bandwidth_GB_Year'])

Q1: 1236.470827
Q3: 5586.1413695
IQR: 4349.6705425
Lower Whisker: -5288.03498675
Upper Whisker: 12110.64718325
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 7158.98153
Lowest Outlier: 155.5067148
```




```
#Treat missing values in InternetService with mode imputation
df_churn['InternetService'] =
df_churn['InternetService'].fillna(df_churn['InternetService'].mode()
[0])
```

C2.

Please see the summary statistics for the dependent and independent variables below. For the numerical variables, I used the "describe()" method to obtain basic information such as the mean and standard deviation. For the categorical variables, I used the "value_counts()" method and multiplied by 100 to obtain percentages for category.

```
#Describe dependent variable, Churn, using percentages for categories
df_churn['Churn'].value_counts(normalize = True) * 100

Churn
No     73.5
Yes    26.5
Name: proportion, dtype: float64
```

```
#Describe independent variable, Contract, using percentages for
categories
df_churn['Contract'].value_counts(normalize = True) * 100
```

```
Contract
Month-to-month    54.56
Two Year          24.42
One year          21.02
Name: proportion, dtype: float64
```

```
#Describe independent variable, InternetService, using percentages for
categories
df_churn['InternetService'].value_counts(normalize = True) * 100
```

```
InternetService
Fiber Optic    65.37
DSL            34.63
Name: proportion, dtype: float64
```

```
#Describe independent variable, TechSupport, using percentages for
categories
df_churn['TechSupport'].value_counts(normalize = True) * 100
```

```
TechSupport
No     62.5
Yes    37.5
Name: proportion, dtype: float64
```

```
#Describe independent variable, Item1, using percentages for
categories
df_churn['Item1'].value_counts(normalize = True) * 100
```

```
Item1
3    34.48
4    33.58
2    13.93
5    13.59
1     2.24
6     1.99
7     0.19
Name: proportion, dtype: float64
```

```
#Describe independent variable, Item2, using percentages for
categories
df_churn['Item2'].value_counts(normalize = True) * 100
```

```
Item2
3    34.15
4    34.12
5    13.68
2    13.60
```

```
1     2.17
6     2.15
7     0.13
Name: proportion, dtype: float64
```

```python
#Describe independent variable, Item3, using percentages for categories
df_churn['Item3'].value_counts(normalize = True) * 100
```

```
Item3
3    34.35
4    34.10
2    14.24
5    13.13
6     2.03
1     2.02
7     0.12
8     0.01
Name: proportion, dtype: float64
```

```python
#Describe independent variable, Item4, using percentages for categories
df_churn['Item4'].value_counts(normalize = True) * 100
```

```
Item4
4    34.52
3    34.30
2    13.50
5    13.35
1     2.21
6     2.03
7     0.09
Name: proportion, dtype: float64
```

```python
#Describe independent variable, Outage_sec_perweek
df_churn['Outage_sec_perweek'].describe()
```

```
count    10000.000000
mean        10.001848
std          2.976019
min          0.099747
25%          8.018214
50%         10.018560
75%         11.969485
max         21.207230
Name: Outage_sec_perweek, dtype: float64
```

```python
#Describe independent variable, Contacts
df_churn['Contacts'].describe()
```

```
count    10000.000000
mean         0.994200
std          0.988466
min          0.000000
25%          0.000000
50%          1.000000
75%          2.000000
max          7.000000
Name: Contacts, dtype: float64
```

```python
#Describe independent variable, Yearly_equip_failure
df_churn['Yearly_equip_failure'].describe()
```

```
count    10000.000000
mean         0.398000
std          0.635953
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          6.000000
Name: Yearly_equip_failure, dtype: float64
```

```python
#Describe independent variable, Tenure
df_churn['Tenure'].describe()
```

```
count    10000.000000
mean        34.526188
std         26.443063
min          1.000259
25%          7.917694
50%         35.430507
75%         61.479795
max         71.999280
Name: Tenure, dtype: float64
```

```python
#Describe independent variable, MonthlyCharge
df_churn['MonthlyCharge'].describe()
```

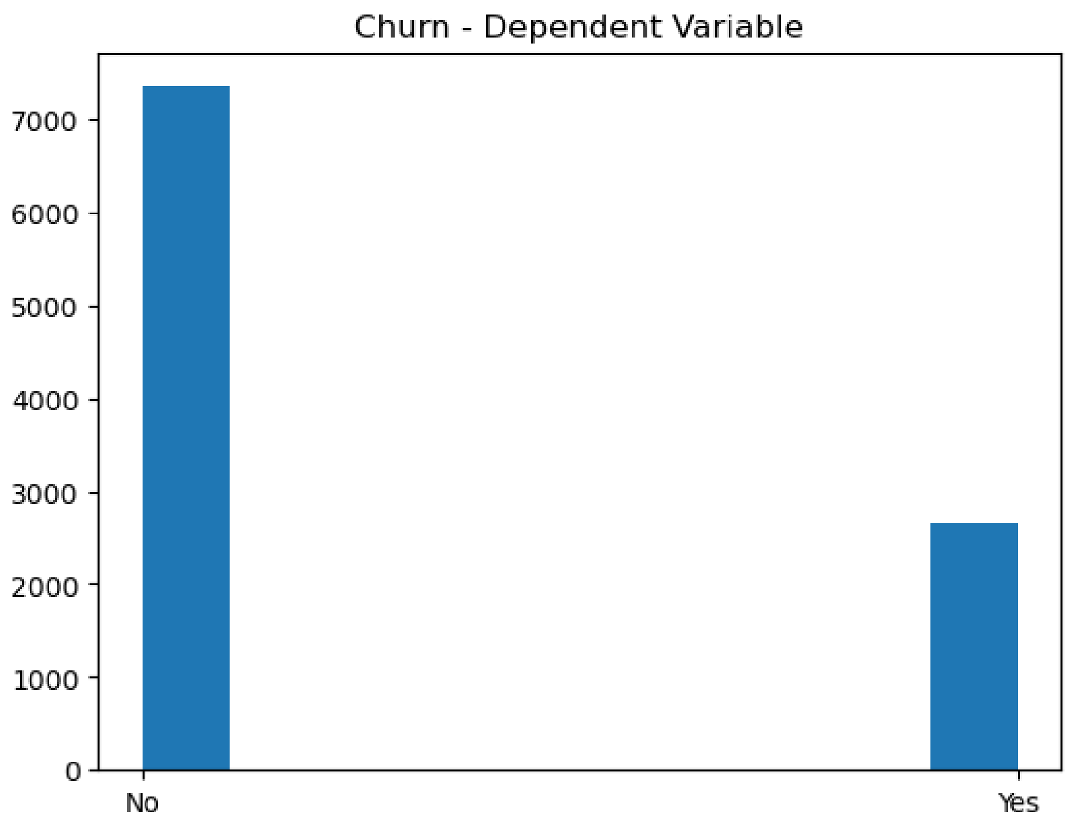
```
count    10000.000000
mean       172.624816
std         42.943094
min         79.978860
25%        139.979239
50%        167.484700
75%        200.734725
max        290.160419
Name: MonthlyCharge, dtype: float64
```

C3.

Please see univariate and bivariate visualizations of dependent and independent variables below. I generated histograms for the univariate visualizations, boxplots for continuous bivariate visualizations, and crosstabulations for categorical bivariate visualizations. The dependent variable "Churn" is included in all bivariate visualizations.

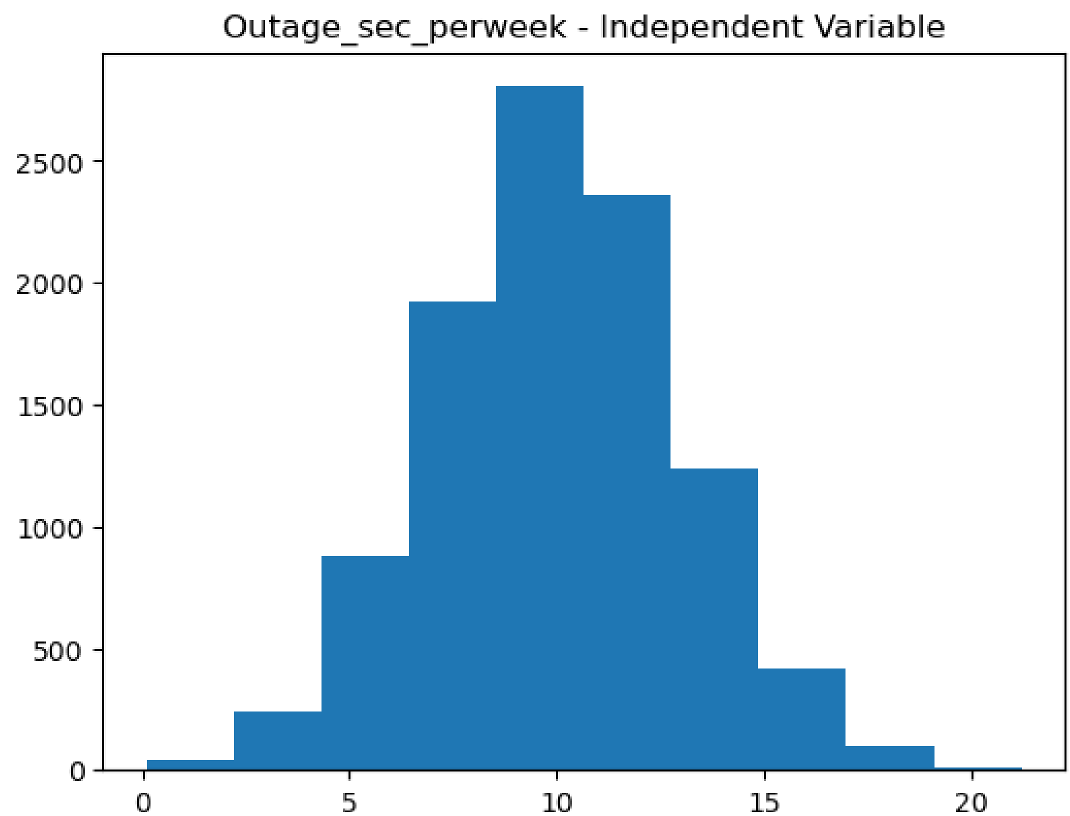
```
#Univariate distribution of dependent variable, Churn
plt.hist(df_churn['Churn'])
plt.title('Churn - Dependent Variable')
```

```
Text(0.5, 1.0, 'Churn - Dependent Variable')
```



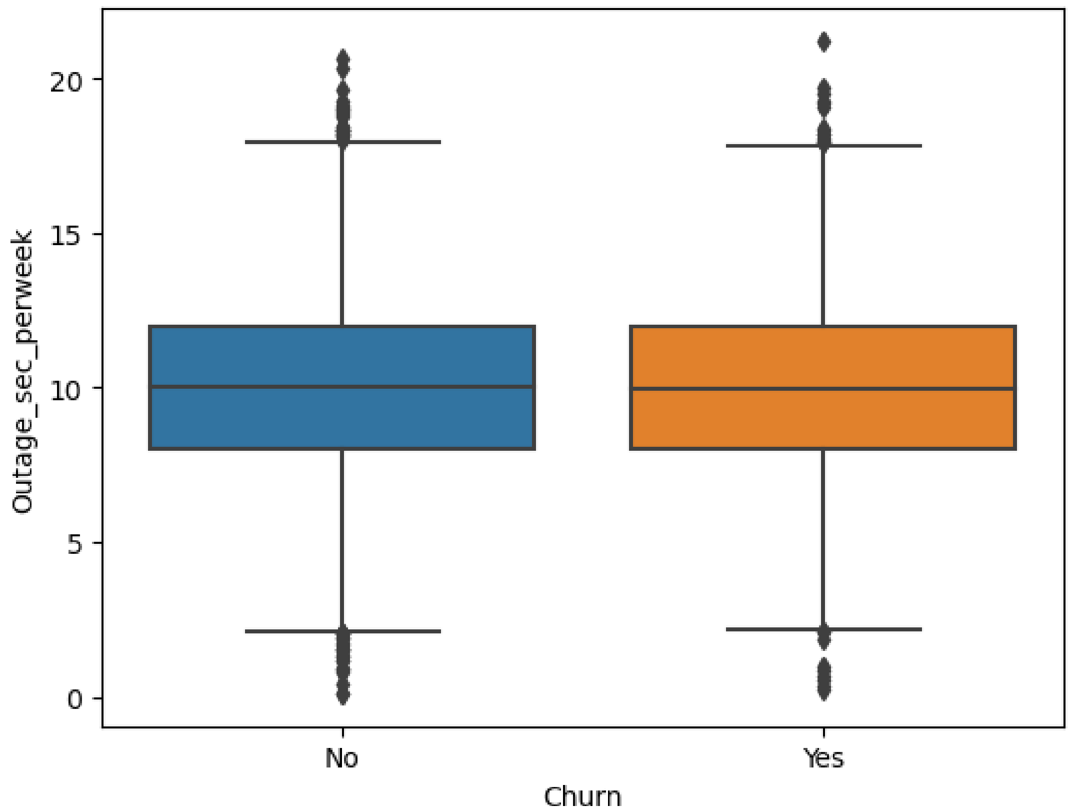

```
#Univariate distribution of dependent variable, Outage_sec_perweek
plt.hist(df_churn['Outage_sec_perweek'])
plt.title('Outage_sec_perweek - Independent Variable')
```

```
Text(0.5, 1.0, 'Outage_sec_perweek - Independent Variable')
```



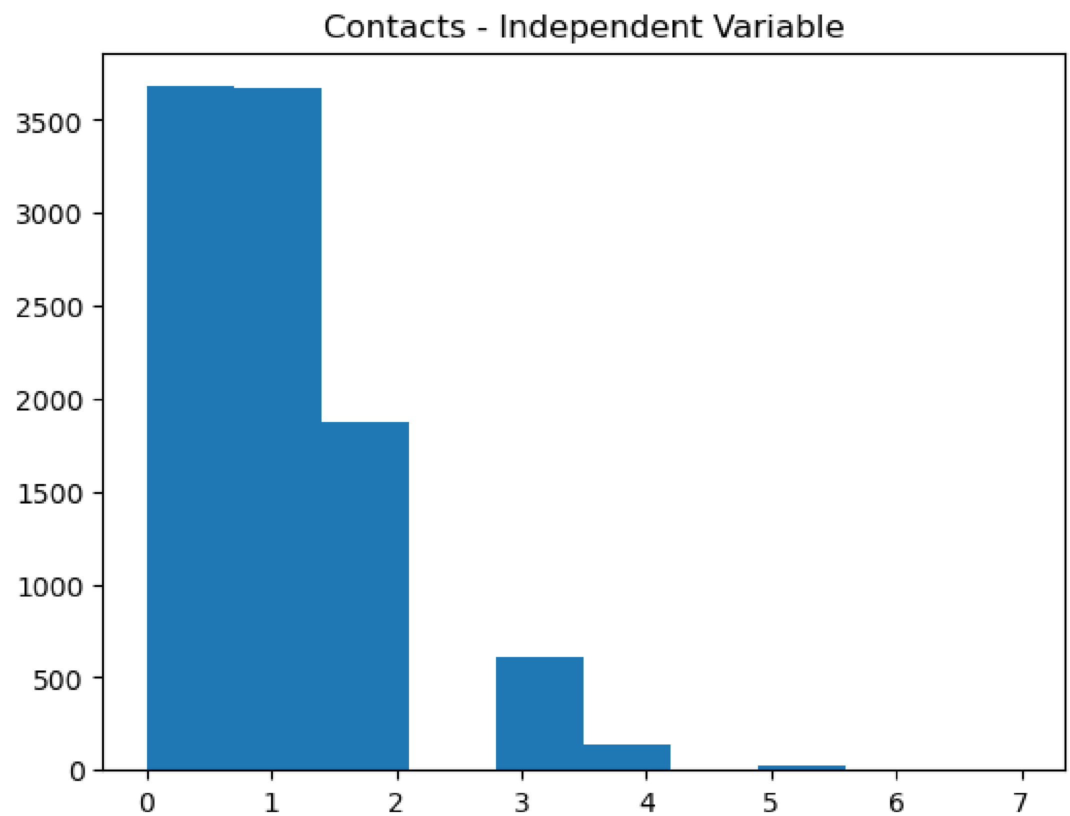

```
# Bivariate distribution between Outage_sec_perweek and Churn
sns.boxplot(x="Churn", y="Outage_sec_perweek", data = df_churn)
```

```
<Axes: xlabel='Churn', ylabel='Outage_sec_perweek'>
```



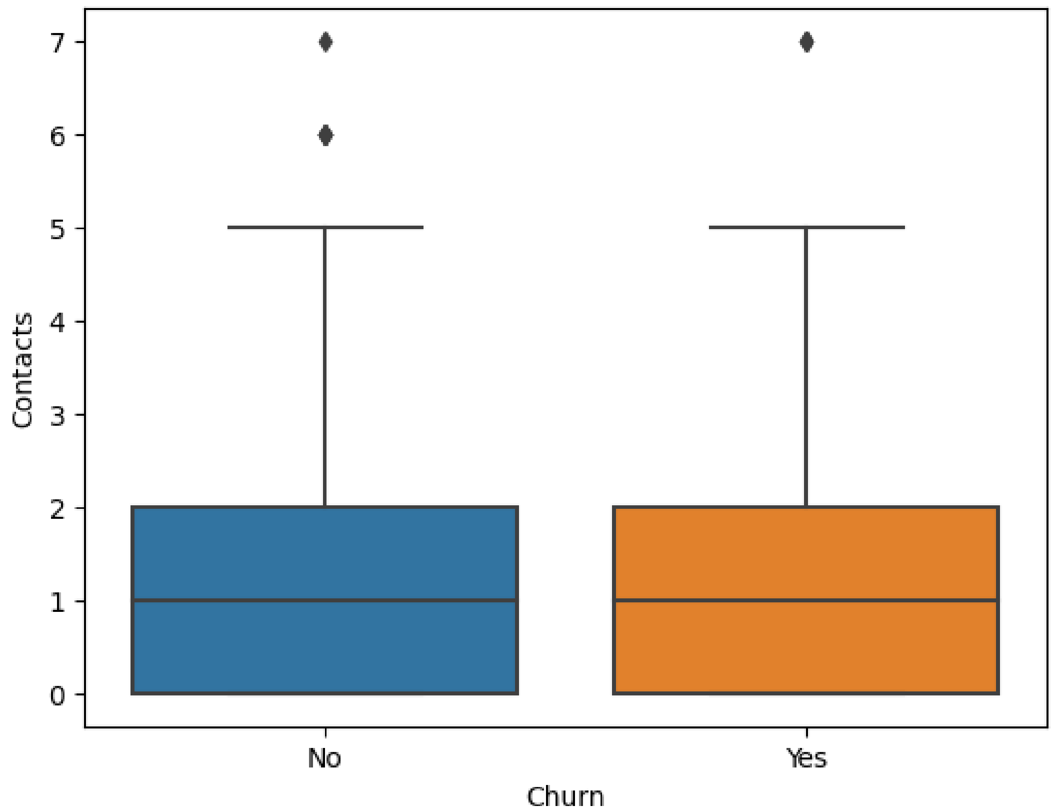

```
#Univariate distribution of Independent variable, Contacts
plt.hist(df_churn['Contacts'])
plt.title('Contacts - Independent Variable')
```

```
Text(0.5, 1.0, 'Contacts - Independent Variable')
```

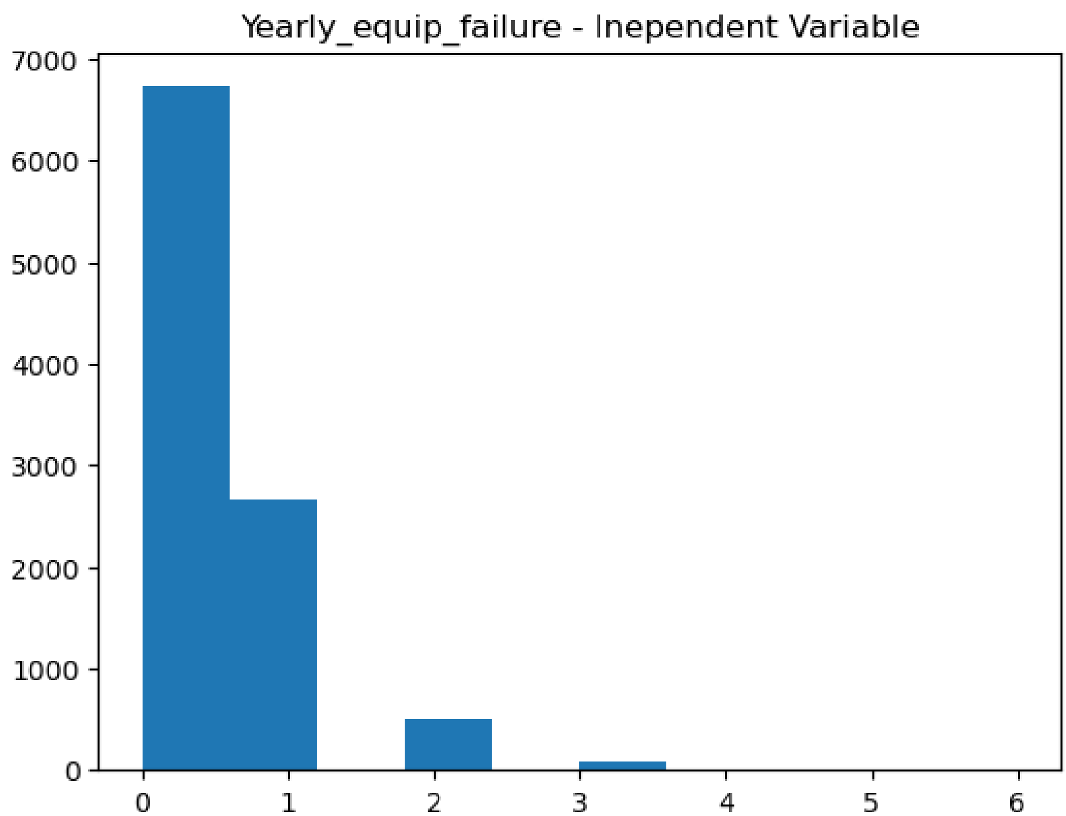



```
# Bivariate distribution between Contacts and Churn
sns.boxplot(x="Churn", y="Contacts", data = df_churn)
```

```
<Axes: xlabel='Churn', ylabel='Contacts'>
```

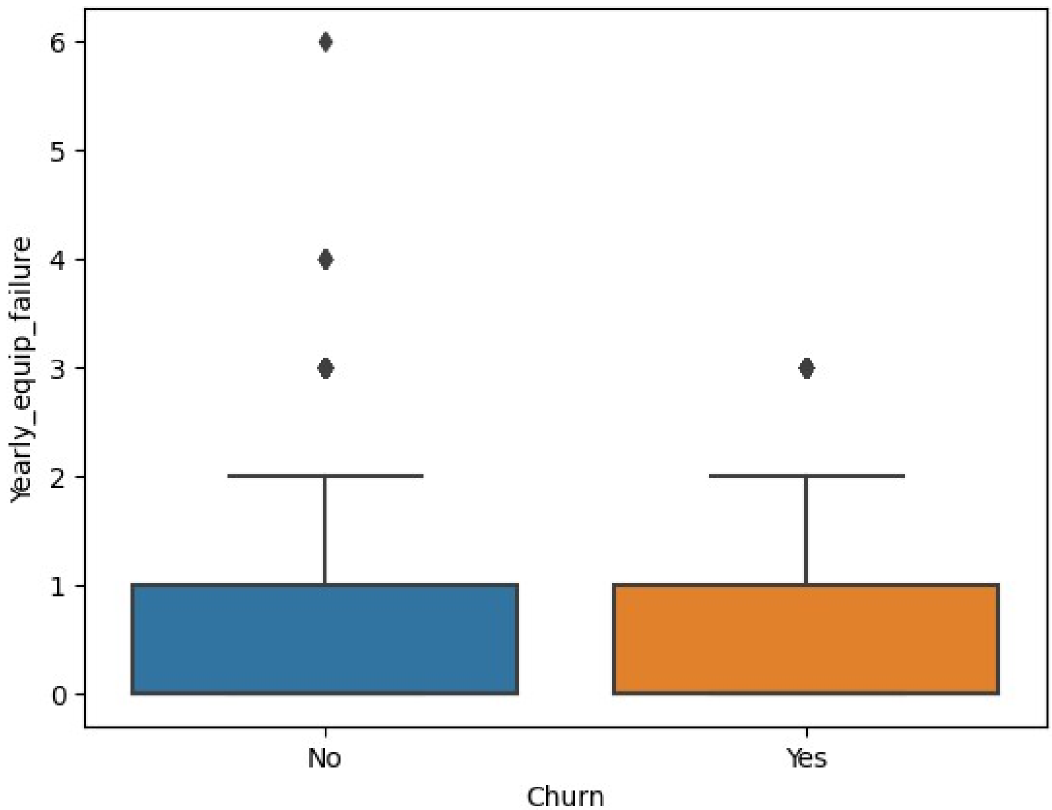



```
#Univariate distribution of independent variable, Yearly_equip_failure
plt.hist(df_churn['Yearly_equip_failure'])
plt.title('Yearly_equip_failure - Inependent Variable')

Text(0.5, 1.0, 'Yearly_equip_failure - Inependent Variable')
```

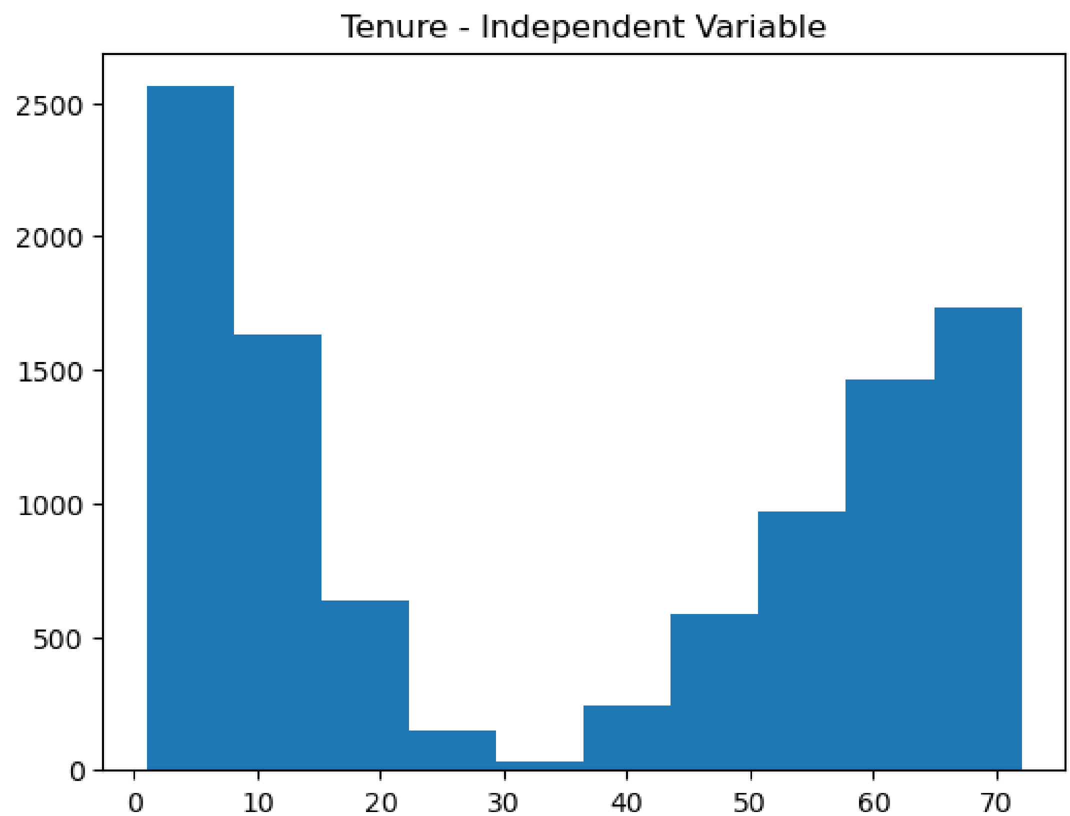



```
# Bivariate distribution between Yearly_equip_failure and Churn
sns.boxplot(x="Churn", y="Yearly_equip_failure", data = df_churn)
```

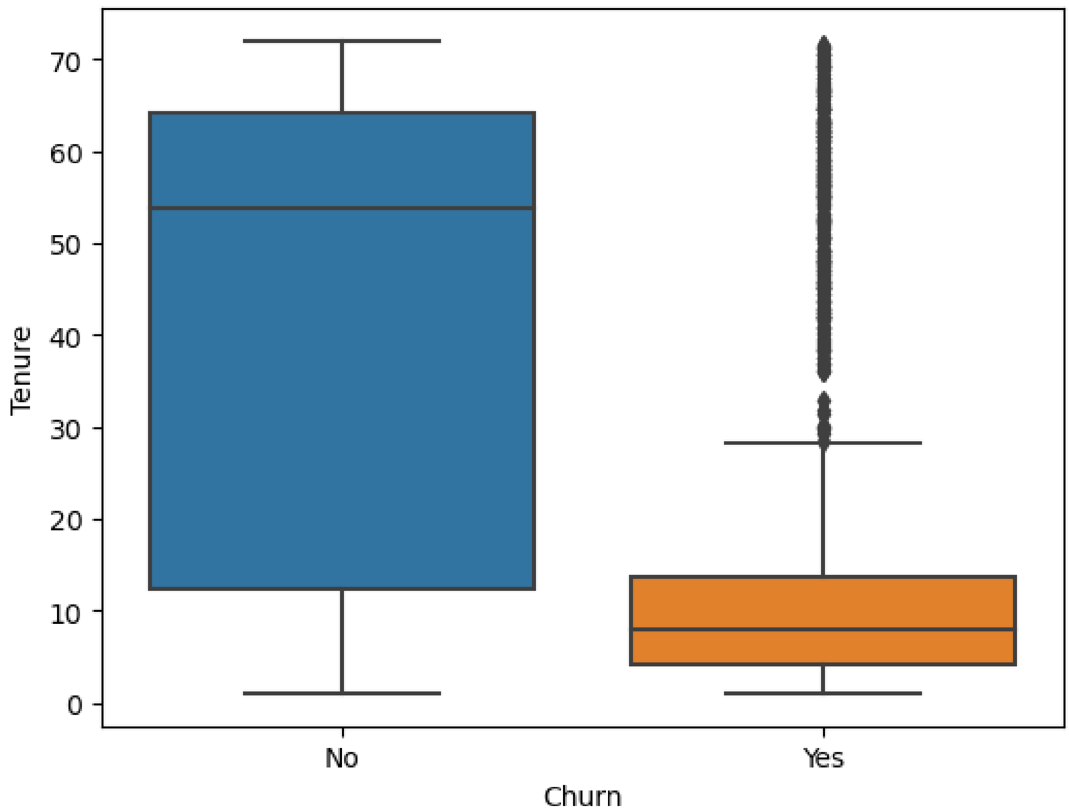
```
<Axes: xlabel='Churn', ylabel='Yearly_equip_failure'>
```




```
#Univariate distribution of independent variable, Tenure
plt.hist(df_churn['Tenure'])
plt.title('Tenure - Independent Variable')

Text(0.5, 1.0, 'Tenure - Independent Variable')
```

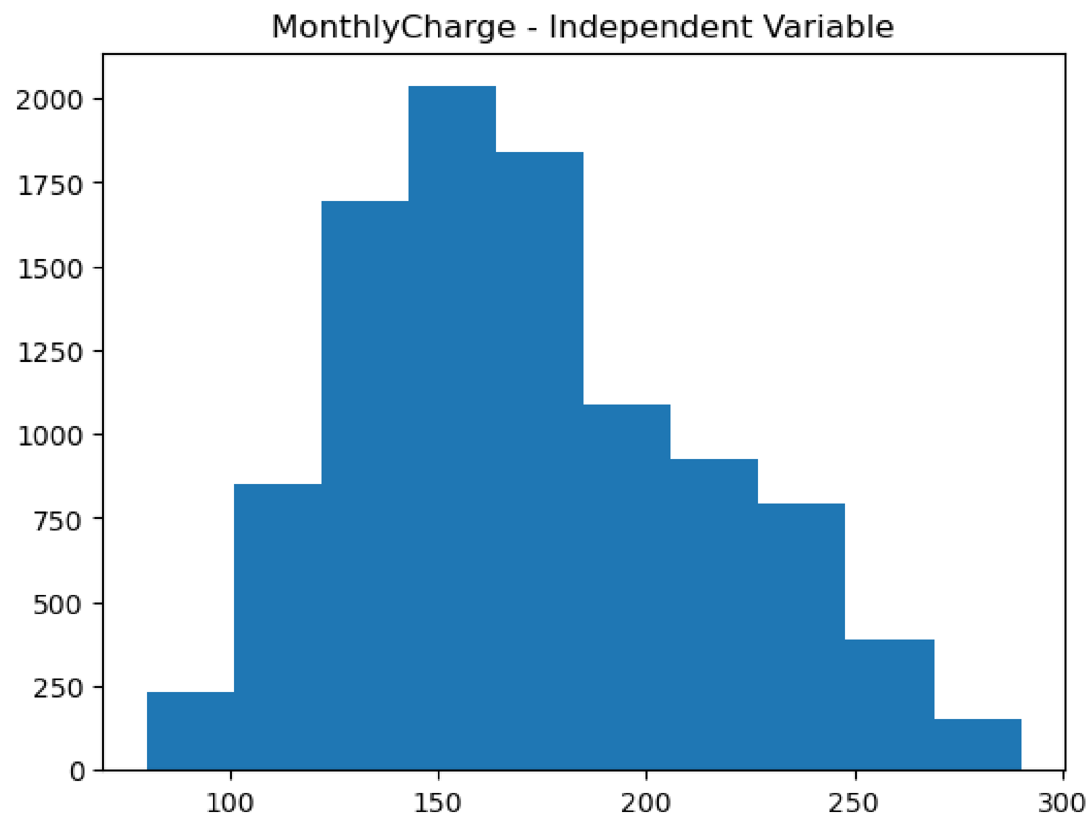



```
# Bivariate distribution between Tenure and Churn
sns.boxplot(x="Churn", y="Tenure", data = df_churn)

<Axes: xlabel='Churn', ylabel='Tenure'>
```



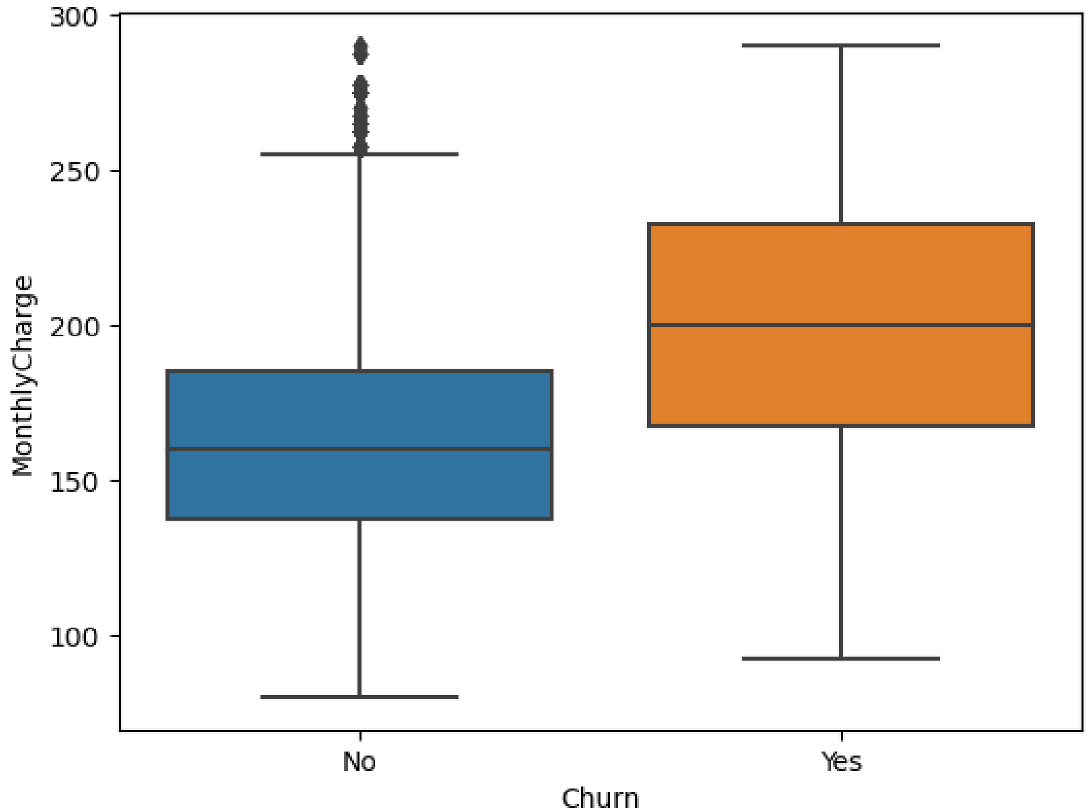

```
#Univariate distribution of independent variable, MonthlyCharge
plt.hist(df_churn['MonthlyCharge'])
plt.title('MonthlyCharge - Independent Variable')
```

```
Text(0.5, 1.0, 'MonthlyCharge - Independent Variable')
```

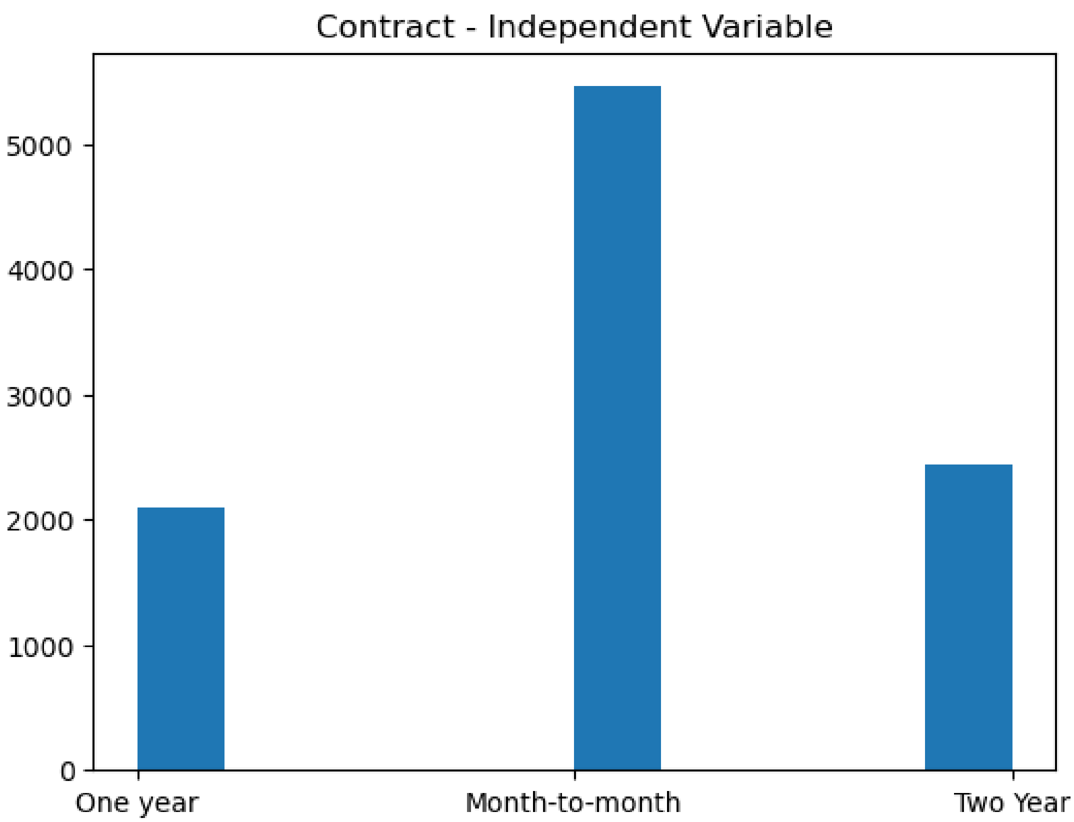



```
# Bivariate distribution between MonthlyCharge and Churn
sns.boxplot(x="Churn", y="MonthlyCharge", data = df_churn)
```

```
<Axes: xlabel='Churn', ylabel='MonthlyCharge'>
```




```
#Univariate distribution of independent variable, Contract
plt.hist(df_churn['Contract'])
plt.title('Contract - Independent Variable')

Text(0.5, 1.0, 'Contract - Independent Variable')
```

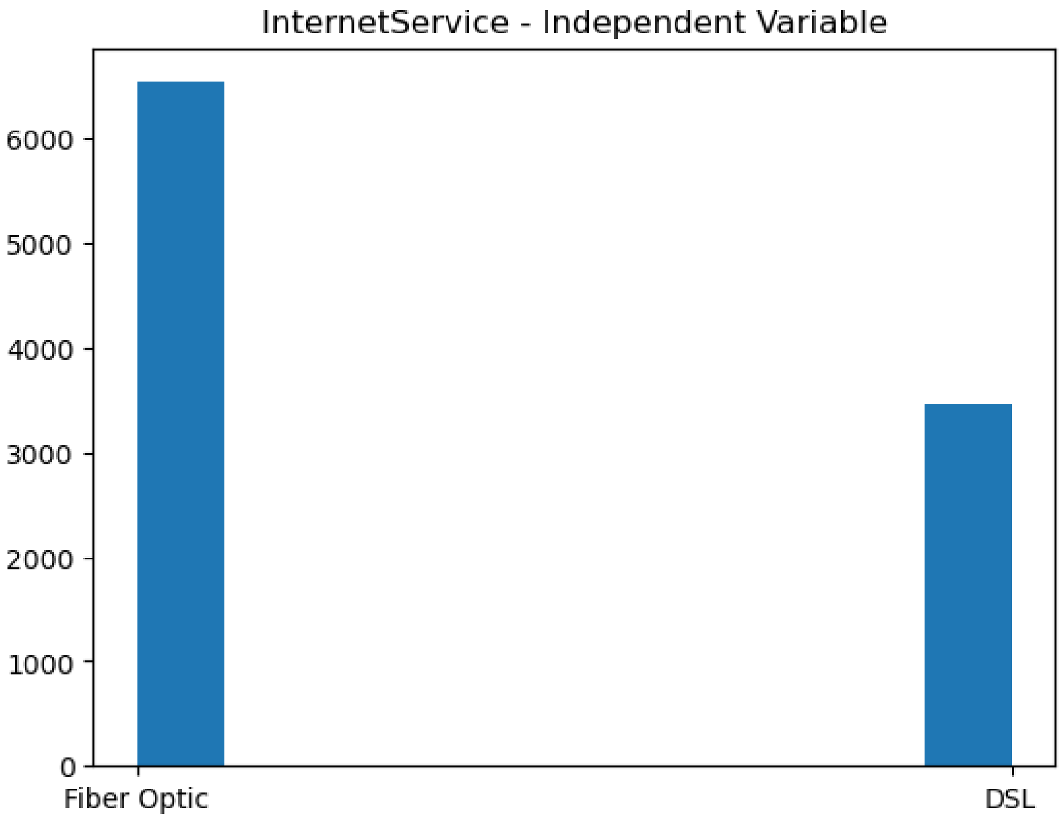



```
# Bivariate visualization for Contract and Churn
pd.crosstab(df_churn["Contract"], df_churn["Churn"])
```

Churn	No	Yes
Contract		
Month-to-month	3422	2034
One year	1795	307
Two Year	2133	309

```
#Univariate distribution of independent variable, InternetService
plt.hist(df_churn['InternetService'])
plt.title('InternetService - Independent Variable')
```

```
Text(0.5, 1.0, 'InternetService - Independent Variable')
```

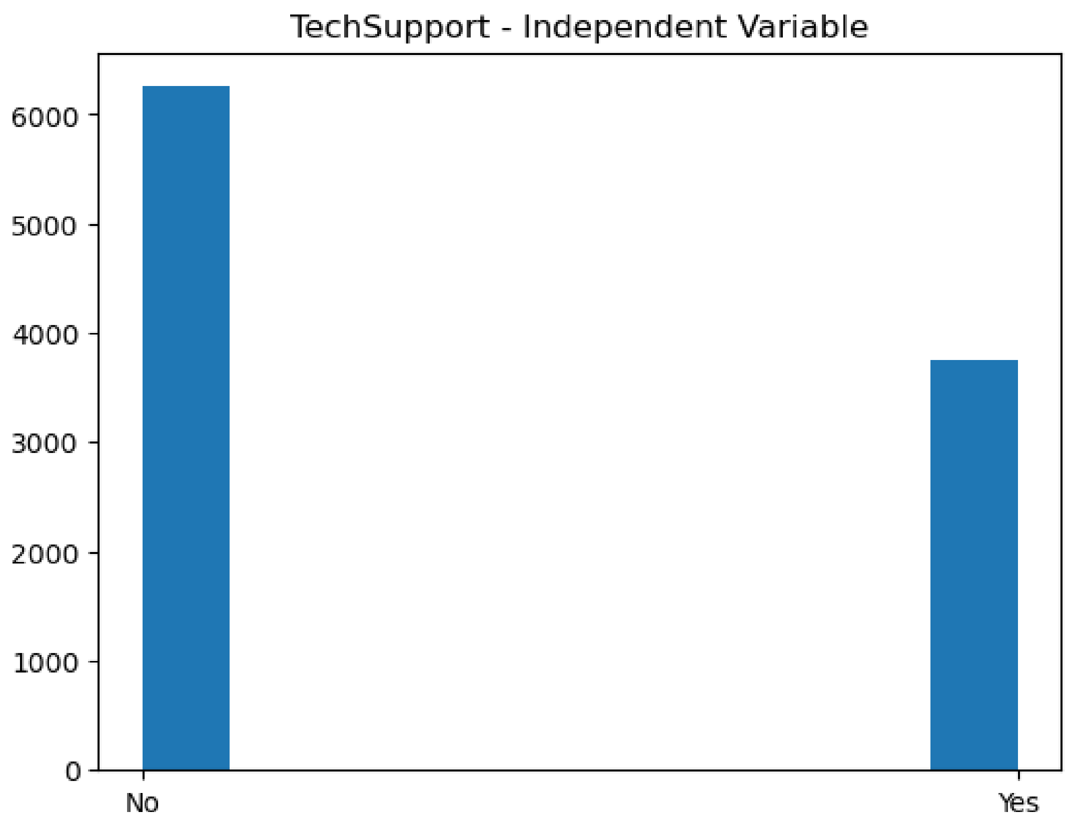



```
# Bivariate distribution for InternetService and Churn
pd.crosstab(df_churn["InternetService"], df_churn["Churn"])
```

Churn	No	Yes
InternetService		
DSL	2349	1114
Fiber Optic	5001	1536

```
#Univariate distribution of independent variable, TechSupport
plt.hist(df_churn['TechSupport'])
plt.title('TechSupport - Independent Variable')
```

```
Text(0.5, 1.0, 'TechSupport - Independent Variable')
```

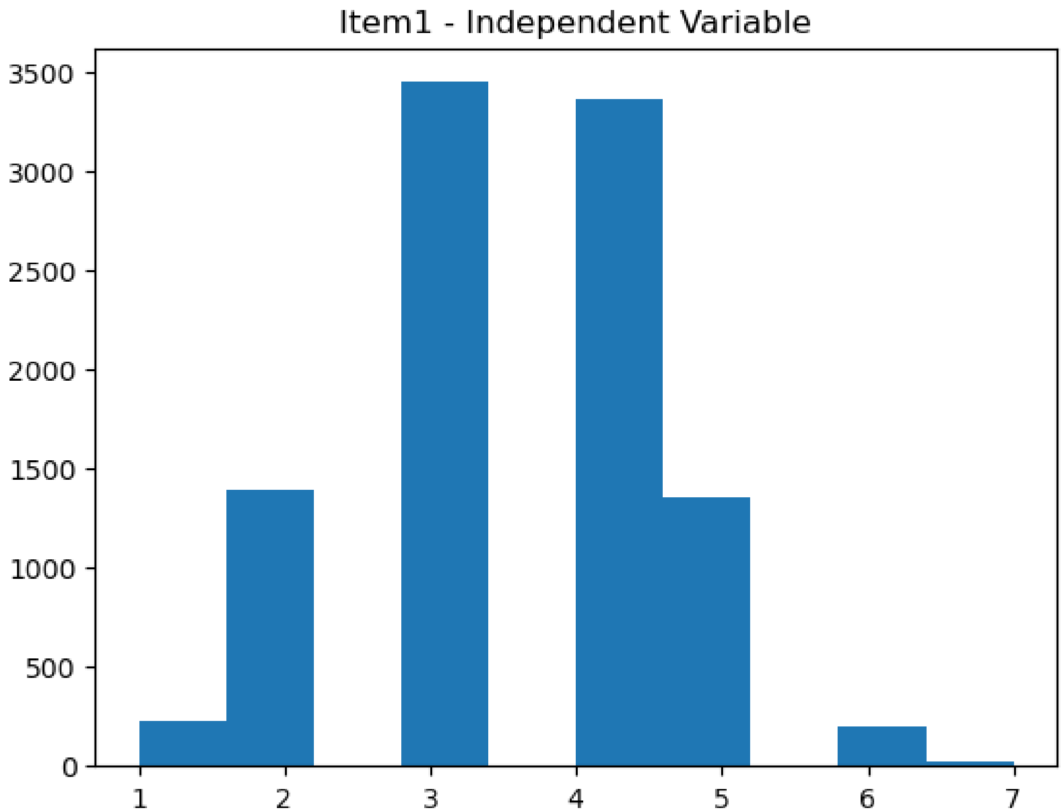



```
# Bivariate distribution for TechSupport and Churn
pd.crosstab(df_churn["TechSupport"], df_churn["Churn"])
```

Churn	No	Yes
TechSupport		
No	4634	1616
Yes	2716	1034

```
#Univariate distribution of independent variable, Item1
plt.hist(df_churn['Item1'])
plt.title('Item1 - Independent Variable')
```

```
Text(0.5, 1.0, 'Item1 - Independent Variable')
```

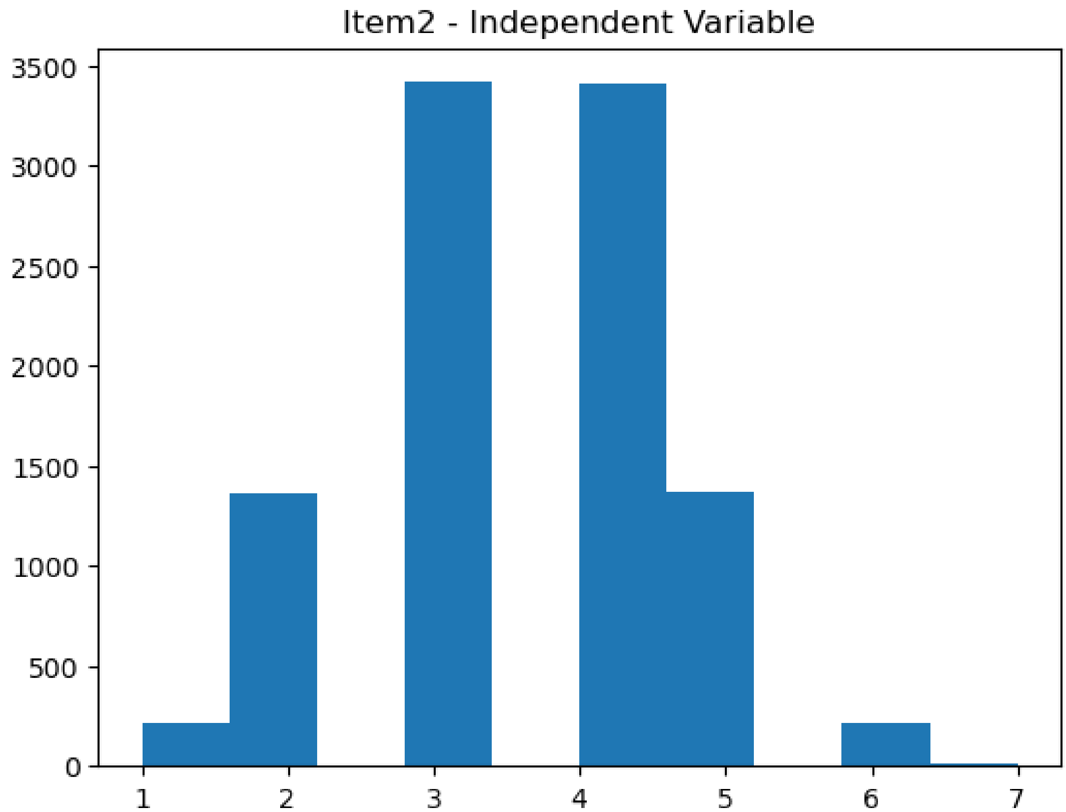



```
# Bivariate distribution for Item1 and Churn
pd.crosstab(df_churn["Item1"], df_churn["Churn"])
```

Churn	No	Yes
Item1		
1	158	66
2	1002	391
3	2562	886
4	2473	885
5	994	365
6	146	53
7	15	4

```
#Univariate distribution of independent variable, Item2
plt.hist(df_churn['Item2'])
plt.title('Item2 - Independent Variable')
```

```
Text(0.5, 1.0, 'Item2 - Independent Variable')
```

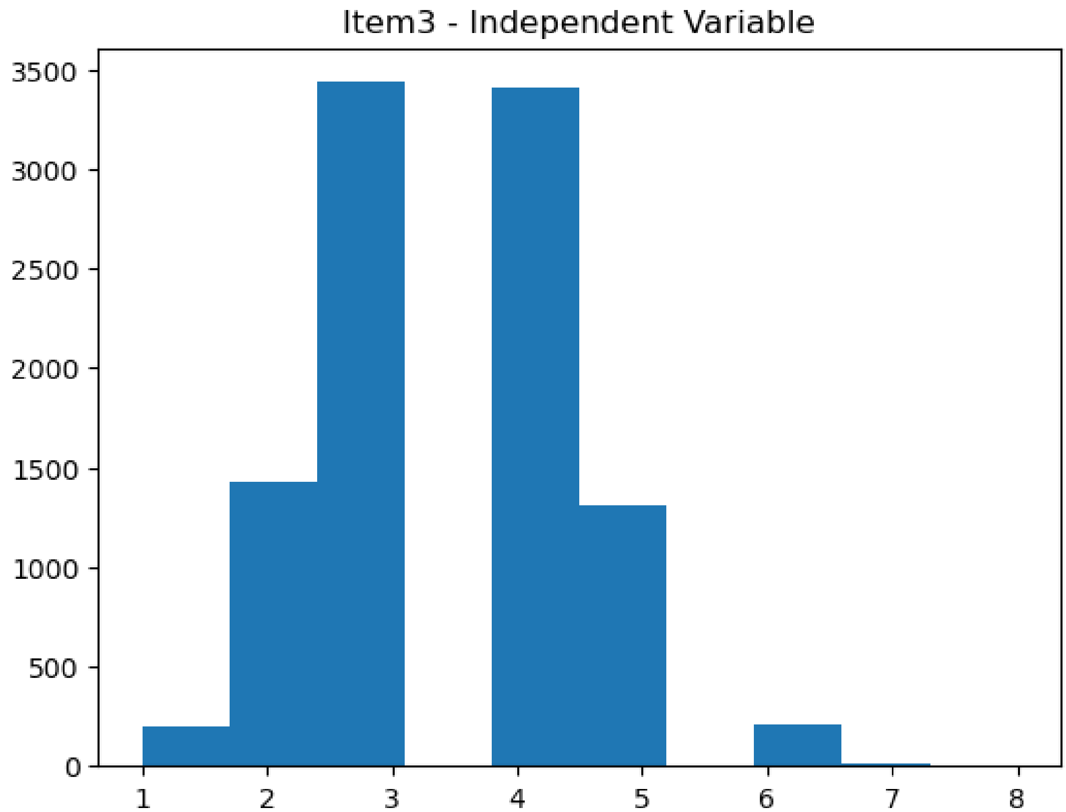



```
# Bivariate distribution for Item2 and Churn
pd.crosstab(df_churn["Item2"], df_churn["Churn"])
```

Churn	No	Yes
Item2		
1	160	57
2	973	387
3	2519	896
4	2507	905
5	1025	343
6	155	60
7	11	2

```
#Univariate distribution of independent variable, Item3
plt.hist(df_churn['Item3'])
plt.title('Item3 - Independent Variable')
```

```
Text(0.5, 1.0, 'Item3 - Independent Variable')
```

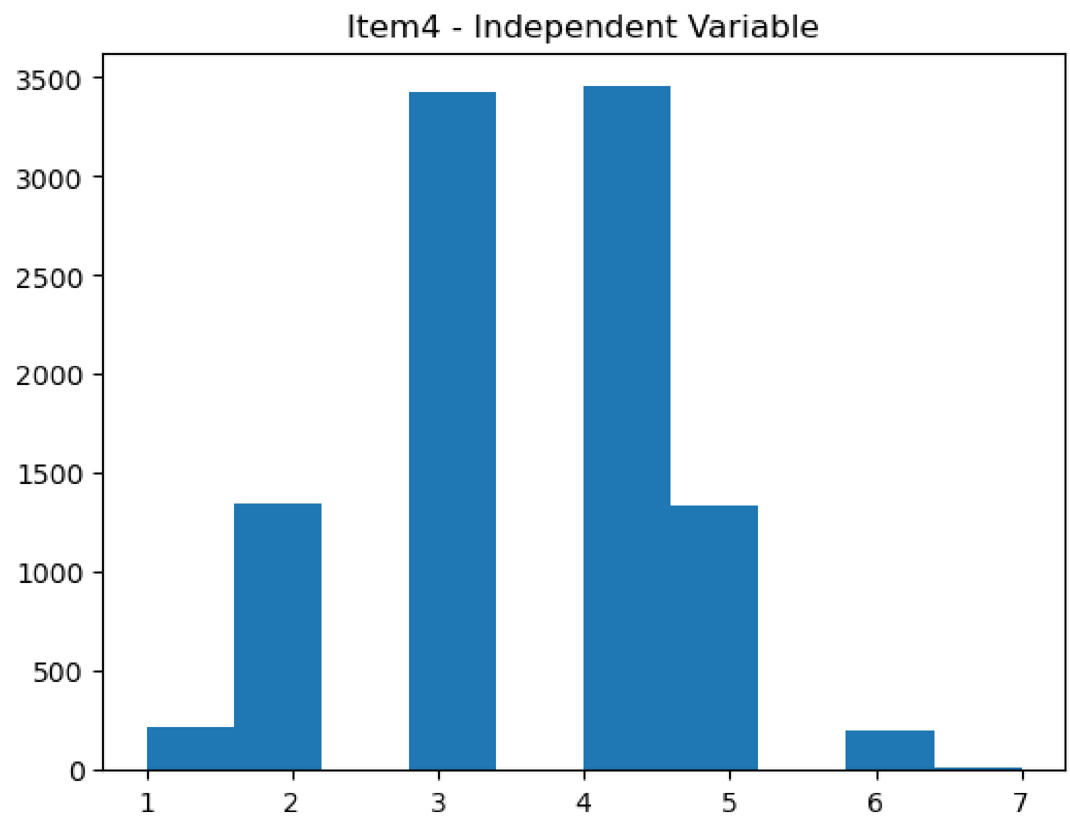



```
# Bivariate distribution for Item3 and Churn
pd.crosstab(df_churn["Item3"], df_churn["Churn"])
```

Churn	No	Yes
Item3		
1	146	56
2	1017	407
3	2540	895
4	2527	883
5	960	353
6	149	54
7	10	2
8	1	0

```
#Univariate distribution of independent variable, Item4
plt.hist(df_churn['Item4'])
plt.title('Item4 - Independent Variable')
```

```
Text(0.5, 1.0, 'Item4 - Independent Variable')
```




```
# Bivariate distribution for Item4 and Churn
pd.crosstab(df_churn["Item4"], df_churn["Churn"])
```

Churn	No	Yes
Item4		
1	162	59
2	990	360
3	2524	906
4	2523	929
5	998	337
6	145	58
7	8	1

C4.

After treating missing values and outliers, I decided to transform df_churn by re-expressing some of the categorical variables. The goal of the transformation was to prepare the selected variables for use in my logistic regression model later on. This regression will allow us to analyze the relationship between the independent variables and the dependent variable "Churn".

I decided to re-express my dependent variable "Churn" and my independent variable "TechSupport" using ordinal encoding. To do this, I created a unique dictionary for both. Each dictionary assigns a value of 1 to "Yes" and 0 to "No". I then used the pandas "replace()" method and passed the unique dictionary as a parameter. This replaced the original "Yes" and "No" with the numerical values contained in the dictionaries. To confirm that the values had been changed, I printed the unique values of the "Churn" and "TechSupport" variables. The resulting output for each variable showed 1 and 0.

I chose to re-express the variable "InternetService" using one-hot encoding. This was because the two categories "DSL" and "Fiber Optic" represent labels rather than a particular order. To do this, I used the "get_dummies()" method from the pandas library. This generated a new dataframe containing the dummy variables "dummy_DSL" and "dummy_Fiber Optic". Because the values of the resulting dataframe are "True" and "False" I used the "astype" method to convert them to 1 and 0. This was done to make the values compatible with the logistic regression model I plan to develop later on. Before adding the dummy variables to "df_churn", I chose to exclude "dummy_DSL", which will serve as the base category for "InternetService". It is recommended to exclude one of the dummy variables to avoid redundancy (Shmueli, 2015).

I also re-expressed "Contract" using one-hot encoding. I chose this method because the values "Month-to-month", "One year" and "Two Year" also represent labels rather than a particular order. To do this, I used the "get_dummies()" method from the pandas library. This generated a new dataframe containing the dummy variables "dummy_Month-to-month", "dummy_One year", and "dummy_Two Year". Because the values of the resulting dataframe are "True" and "False" I used the "astype" method to convert them to 1 and 0. This was done to make the values compatible with the logistic regression model I plan to develop later on. Before adding the dummy variables to "df_churn", I chose to exclude "dummy_Month-to-month", which will serve as the base category for "Contract".

Lastly, I chose to rename a few of the categorical variables that will be used in the regression model. "Item1", "Item2", "Item3", and "Item4" were changed to "Responses","Fixes","Replacements", and "Reliability". This was done to make the variables easier to interpret. The final step I took was to change the data types of all variables to float so they would be compatible with logistic regression.

Now that the variables have been transformed, we are ready to proceed with the initial regression model. Please see the annotated code below, which was used to transform the selected categorical variables.

```
#Re-express dependent variable, Churn, as numeric

#Find unique values of variable
print(df_churn["Churn"].unique())

#Create dictionary to store numeric values for variable
dict_churn = {"Churn":
                    {"Yes":1,
                     "No":0
                    }
                }

#Replace categorical values with numeric values from dictionary
df_churn.replace(dict_churn, inplace=True)

#Change variable to float for compatability with logistic regression
```

```
df_churn["Churn"] = df_churn["Churn"].astype(int)

#Confirm categorical values have been replaced
print(df_churn["Churn"].unique())
```

```
['No' 'Yes']
[0 1]
```

```
#Re-express Contract as numeric using one-hot encoding

#Use pd.get_dummies to turn Contract variable into 3 dummy variables
df_contract = pd.get_dummies(df_churn["Contract"], prefix="dummy")

#Change data type of dummy variables from boolean to float
df_contract = df_contract.astype(float)

#Join dummy_One year and dummy_Two Year to df_churn.  Use dummy_Month-
to-month as base category.
df_churn = df_churn.join(df_contract[["dummy_One year", "dummy_Two
Year"]])

#Re-express InternetService as numeric using one-hot encoding

#Use pd.get_dummies to turn InternetService variable into 2 dummy
variable
df_internet = pd.get_dummies(df_churn["InternetService"],
prefix="dummy")

#Change data type of dummy variables from boolean to float
df_internet = df_internet.astype(float)

#Join dummy_Fiber Optic to df_churn
df_churn = df_churn.join(df_internet["dummy_Fiber Optic"])

#Re-express TechSupport as numeric using ordinal encoding

#Find unique values of  variable
print(df_churn["TechSupport"].unique())

#Create dictionary to store numeric values for variable
dict_techsupport = {"TechSupport":
                    {"Yes":1,
                     "No":0,
                    }
                }

#Replace categorical values with numeric values from dictionary
df_churn.replace(dict_techsupport, inplace=True)

#Confirm categorical values have been replaced
print(df_churn["TechSupport"].unique())
```

```
['No' 'Yes']
[0 1]

#Rename Item columns in df_churn
df_churn = df_churn.rename(columns =
{'Item1':'Responses','Item2':'Fixes','Item3':'Replacements','Item4':'R
eliability'})
```

C5.

Please see the attached csv file containing the prepared data.

```
df_churn.to_csv('churn_prepared.csv')
```

D1.

To build the multiple logistic regression model, I first assigned the independent variables to a dataframe called "X" and added a constant using the "sm.add_constant()" method. The dependent variable, "Churn", was added to a dataframe called "y". I then used the "Logit()" method from statsmodels.api and passed the dataframes as parameters. To generate a summary of the model, I used the "summary2()" method.

```
#Create initial logistic regression model [In-text citation: (LaRose
et al, 2019)]
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Responses","Fixes","Replacements","Reliability"]
])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.246969
         Iterations 8

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
=====================================================================
Model:                 Logit              Method:            MLE
Dependent Variable:    Churn              Pseudo R-squared:  0.573
Date:                  2024-02-05 19:05   AIC:               4967.3750
```

```
No. Observations:      10000            BIC:                5068.3198
Df Model:              13               Log-Likelihood:     -2469.7
Df Residuals:          9986             LL-Null:            -5782.2
Converged:             1.0000           LLR p-value:        0.0000
No. Iterations:        8.0000           Scale:              1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                -4.9120   0.2916 -16.8432 0.0000 -5.4836 -4.3404
Outage_sec_perweek   -0.0012   0.0121  -0.0958 0.9237 -0.0250  0.0226
Contacts              0.0493   0.0360   1.3689 0.1710 -0.0213  0.1200
Yearly_equip_failure -0.0219   0.0567  -0.3861 0.6994 -0.1331  0.0893
Tenure               -0.1019   0.0025 -40.5148 0.0000 -0.1069 -0.0970
MonthlyCharge         0.0492   0.0013  38.4982 0.0000  0.0467  0.0517
dummy_One year       -2.9832   0.1143 -26.0934 0.0000 -3.2072 -2.7591
dummy_Two Year       -3.0552   0.1109 -27.5538 0.0000 -3.2726 -2.8379
dummy_Fiber Optic    -1.6949   0.0821 -20.6383 0.0000 -1.8559 -1.5339
TechSupport          -0.3598   0.0747  -4.8139 0.0000 -0.5063 -0.2133
Responses            -0.0552   0.0494  -1.1177 0.2637 -0.1519  0.0416
Fixes                -0.0240   0.0480  -0.5013 0.6162 -0.1180  0.0699
Replacements          0.0328   0.0439   0.7458 0.4558 -0.0534  0.1189
Reliability          -0.0327   0.0348  -0.9416 0.3464 -0.1008  0.0354
=====================================================================

"""
```

D2.

Now that the initial model has been built, I plan to reduce the amount of features using a few different methods. This step is imperative because having too many explanatory variables can make a regression model unreliable. This will help us to identify which factors are indeed the most responsible in predicting customer churn.

The first method I will use is to check for multicollinearity. This occurs when there is a high degree of correlation between two or more independent variables. This poses a problem because it can lead to inaccurate regression coefficients. To do this, I will calculate the variance inflation factor (VIF) for all features in my initial model. The first step will be to assign the features to a new dataframe and use the "variance_inflation_factor()" method to calculate the VIF for each feature. Typically, if a predictor has a VIF of 10 or greater, there is a high level of correlation with another predictor (LaRose et al, 2019). I will remove the variable with the highest VIF greater than 10 and recalculate VIF for the remaining variables. This process will be repeated until there are no remaining variable with a VIF greater than 10.

After removing variables with high VIF, I will use backward stepwise regression to eliminate variables that are not statistically significant. I will start off by removing the variable with the largest p-value that is greater than .05. Then I will run a new iteration of the model using the "Logit()" method and recalculate the p-values for each variable. This process will be repeated until all of the remaining independent variables have a p-value less than 0.05. These variables will be included in the reduced logistic regression model.

Lastly, I will compare the pseudo R-squared of the initial and reduced models. Unlike the R-squared used in linear regression, a pseudo R-squared provides no value unless it is compared to another pseudo R-squared value. Pseudo r-squared values range from 0 to 1, with higher values indicating a better fit (UCLA Statistical Methods and Data Analytics).

D3.

Please see the annotated model evaluation process below. This process follows the steps outlined in section D2. Please see the output of each model iteration below. The reduced logistic regression model is located at the end of this section.

Calculate VIF for all independent variables

```
#Assign independent variables to dataframe X
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Responses","Fixes","Replacements","Reliability"]
])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

	feature	VIF
0	Outage_sec_perweek	10.062425
1	Contacts	1.981046
2	Yearly_equip_failure	1.382617
3	Tenure	2.630068
4	MonthlyCharge	13.493832
5	dummy_One year	1.376736
6	dummy_Two Year	1.438965
7	dummy_Fiber Optic	2.872565
8	TechSupport	1.626659
9	Responses	25.102549
10	Fixes	22.969036
11	Replacements	18.970597
12	Reliability	10.141270

Remove Reponses (VIF = 25.102549) and recalculate VIF

```
#Assign independent variables to dataframe X, remove Responses
X =
```

```
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Fixes","Replacements","Reliability"]])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

```
                 feature        VIF
0     Outage_sec_perweek  10.052907
1               Contacts   1.980515
2   Yearly_equip_failure   1.381201
3                 Tenure   2.629971
4          MonthlyCharge  13.421418
5          dummy_One year   1.376498
6          dummy_Two Year   1.438877
7       dummy_Fiber Optic   2.869853
8            TechSupport   1.625990
9                  Fixes  16.150396
10          Replacements  15.982121
11           Reliability  10.114242
```

Remove Fixes (VIF = 16.150396) and recalculate VIF

```
#Assign independent variables to dataframe X, remove Responses
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

```
                 feature        VIF
0     Outage_sec_perweek   9.908351
1               Contacts   1.977029
2    Yearly_equip_failure   1.380827
3                 Tenure   2.624860
4          MonthlyCharge  13.131709
5          dummy_One year   1.376427
6          dummy_Two Year   1.438423
7       dummy_Fiber Optic   2.868109
8            TechSupport   1.625615
9           Replacements   9.795162
10           Reliability   9.956283
```

Remove MonthlyCharge (VIF = 13.131709) and recalculate VIF

```
#Assign independent variables to dataframe X, remove Responses
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","dummy_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)

                feature       VIF
0    Outage_sec_perweek  8.928276
1              Contacts  1.962045
2  Yearly_equip_failure  1.378012
3                Tenure  2.598159
4        dummy_One year  1.370356
5        dummy_Two Year  1.433700
6     dummy_Fiber Optic  2.762826
7           TechSupport  1.582278
8          Replacements  8.952135
9           Reliability  9.065586
```

Remove Responses, Fixes, and MonthlyCharge (VIF > 10) from initial model. Perform new regression to check for features with largest p-value greater than 0.05

```
#Iterate on logistic regression model, remove Responses, Fixes, and
MonthlyCharge (VIF > 10)
X =
```

```
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","dummy_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.387693
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                           Results: Logit
=====================================================================
Model:                 Logit              Method:             MLE
Dependent Variable:    Churn              Pseudo R-squared:   0.330
Date:                  2024-02-05 19:05   AIC:                7775.8533
No. Observations:      10000              BIC:                7855.1670
Df Model:              10                 Log-Likelihood:     -3876.9
Df Residuals:          9989               LL-Null:            -5782.2
Converged:             1.0000             LLR p-value:        0.0000
No. Iterations:        7.0000             Scale:              1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                 1.7702   0.1834   9.6498 0.0000  1.4107  2.1298
Outage_sec_perweek    0.0053   0.0095   0.5618 0.5743 -0.0133  0.0239
Contacts              0.0387   0.0286   1.3522 0.1763 -0.0174  0.0947
Yearly_equip_failure -0.0398   0.0450  -0.8839 0.3767 -0.1279  0.0484
Tenure               -0.0624   0.0015 -42.0431 0.0000 -0.0654 -0.0595
dummy_One year       -1.7213   0.0783 -21.9968 0.0000 -1.8746 -1.5679
dummy_Two Year       -1.8604   0.0770 -24.1537 0.0000 -2.0114 -1.7095
dummy_Fiber Optic    -0.7526   0.0597 -12.6072 0.0000 -0.8696 -0.6356
TechSupport           0.0954   0.0584   1.6337 0.1023 -0.0191  0.2099
Replacements         -0.0189   0.0275  -0.6856 0.4930 -0.0729  0.0351
Reliability          -0.0251   0.0276  -0.9100 0.3628 -0.0791  0.0290
=====================================================================

"""
```

Remove Outage_sec_perweek (p > 0.05) and check p-values again

```
#Iterate on logistic regression model, remove Outage_sec_perweek (p-
value > 0.05)
X =
```

```
pd.DataFrame(df_churn[["Contacts","Yearly_equip_failure","Tenure","dummy_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.387708
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                           Results: Logit
=====================================================================
Model:                 Logit              Method:             MLE
Dependent Variable:    Churn              Pseudo R-squared:   0.329
Date:                  2024-02-05 19:05   AIC:                7774.1689
No. Observations:      10000              BIC:                7846.2723
Df Model:              9                  Log-Likelihood:     -3877.1
Df Residuals:          9990               LL-Null:            -5782.2
Converged:             1.0000             LLR p-value:        0.0000
No. Iterations:        7.0000             Scale:              1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                 1.8238   0.1568  11.6305 0.0000  1.5164  2.1311
Contacts              0.0389   0.0286   1.3610 0.1735 -0.0171  0.0950
Yearly_equip_failure -0.0394   0.0450  -0.8758 0.3811 -0.1275  0.0487
Tenure               -0.0624   0.0015 -42.0431 0.0000 -0.0654 -0.0595
dummy_One year       -1.7210   0.0782 -21.9940 0.0000 -1.8744 -1.5676
dummy_Two Year       -1.8600   0.0770 -24.1520 0.0000 -2.0110 -1.7091
dummy_Fiber Optic    -0.7522   0.0597 -12.6012 0.0000 -0.8692 -0.6352
TechSupport           0.0952   0.0584   1.6296 0.1032 -0.0193  0.2097
Replacements         -0.0191   0.0275  -0.6940 0.4877 -0.0731  0.0349
Reliability          -0.0251   0.0276  -0.9119 0.3618 -0.0792  0.0289
=====================================================================

"""
```

Remove Replacements (p > 0.05) and check p-values again

```
#Iterate on logistic regression model, remove Replacements (p-value >
0.05)
X =
pd.DataFrame(df_churn[["Contacts","Yearly_equip_failure","Tenure","dum
```

```
my_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.387733
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
=====================================================================
Model:                 Logit              Method:            MLE
Dependent Variable:    Churn              Pseudo R-squared:  0.329
Date:                  2024-02-05 19:05   AIC:               7772.6507
No. Observations:      10000              BIC:               7837.5437
Df Model:              8                  Log-Likelihood:    -3877.3
Df Residuals:          9991               LL-Null:           -5782.2
Converged:             1.0000             LLR p-value:       0.0000
No. Iterations:        7.0000             Scale:             1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                 1.7566   0.1232  14.2568 0.0000  1.5151  1.9981
Contacts              0.0391   0.0286   1.3689 0.1710 -0.0169  0.0952
Yearly_equip_failure -0.0390   0.0450  -0.8677 0.3856 -0.1271  0.0491
Tenure               -0.0624   0.0015 -42.0456 0.0000 -0.0654 -0.0595
dummy_One year       -1.7224   0.0782 -22.0153 0.0000 -1.8757 -1.5690
dummy_Two Year       -1.8601   0.0770 -24.1560 0.0000 -2.0110 -1.7092
dummy_Fiber Optic    -0.7522   0.0597 -12.6020 0.0000 -0.8692 -0.6352
TechSupport           0.0946   0.0584   1.6199 0.1053 -0.0199  0.2091
Reliability          -0.0249   0.0276  -0.9037 0.3662 -0.0790  0.0291
=====================================================================

"""
```

Remove Yearly_equip_failure (p > 0.05) and check p-values again

```
#Iterate on logistic regression model, remove Yearly_equip_failure (p-
value > 0.05)
X = pd.DataFrame(df_churn[["Contacts","Tenure","dummy_One
year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Reliability"]])
X = sm.add_constant(X)
```

```
y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.387770
         Iterations 7
```

```
<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:              Logit            Method:           MLE
Dependent Variable: Churn            Pseudo R-squared: 0.329
Date:               2024-02-05 19:05 AIC:              7771.4055
No. Observations:   10000            BIC:              7829.0882
Df Model:           7                Log-Likelihood:   -3877.7
Df Residuals:       9992             LL-Null:          -5782.2
Converged:          1.0000           LLR p-value:      0.0000
No. Iterations:     7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.7423   0.1221  14.2740 0.0000  1.5030  1.9815
Contacts           0.0392   0.0286   1.3703 0.1706 -0.0169  0.0952
Tenure            -0.0625   0.0015 -42.0539 0.0000 -0.0654 -0.0595
dummy_One year    -1.7232   0.0782 -22.0267 0.0000 -1.8765 -1.5699
dummy_Two Year    -1.8600   0.0770 -24.1580 0.0000 -2.0109 -1.7091
dummy_Fiber Optic -0.7523   0.0597 -12.6039 0.0000 -0.8693 -0.6353
TechSupport        0.0935   0.0584   1.6013 0.1093 -0.0209  0.2079
Reliability       -0.0250   0.0276  -0.9061 0.3649 -0.0790  0.0291
==================================================================

"""
```

Remove Reliability (p > 0.05) and check p-values again

```
#Iterate on logistic regression model, remove Yearly_equip_failure (p-
value > 0.05)
X = pd.DataFrame(df_churn[["Contacts","Tenure","dummy_One
year","dummy_Two Year","dummy_Fiber Optic","TechSupport"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.387811
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                Logit            Method:           MLE
Dependent Variable:   Churn            Pseudo R-squared: 0.329
Date:                 2024-02-05 19:05 AIC:              7770.2266
No. Observations:     10000            BIC:              7820.6990
Df Model:             6                Log-Likelihood:   -3878.1
Df Residuals:         9993             LL-Null:          -5782.2
Converged:            1.0000           LLR p-value:      0.0000
No. Iterations:       7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.6550   0.0746  22.1819 0.0000  1.5087  1.8012
Contacts           0.0391   0.0286   1.3678 0.1714 -0.0169  0.0951
Tenure            -0.0625   0.0015 -42.0491 0.0000 -0.0654 -0.0595
dummy_One year    -1.7221   0.0782 -22.0201 0.0000 -1.8754 -1.5688
dummy_Two Year    -1.8598   0.0770 -24.1570 0.0000 -2.0107 -1.7089
dummy_Fiber Optic -0.7524   0.0597 -12.6067 0.0000 -0.8694 -0.6355
TechSupport        0.0926   0.0584   1.5866 0.1126 -0.0218  0.2070
==================================================================

"""
```

Remove Contacts ($p > 0.05$) and check p-values

```
#Iterate on logistic regression model, remove Contacts (p-value >
0.05)
X = pd.DataFrame(df_churn[["Tenure","dummy_One year","dummy_Two
Year","dummy_Fiber Optic","TechSupport"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.387905
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
```

```
==================================================================
Model:                 Logit            Method:           MLE
Dependent Variable:    Churn            Pseudo R-squared: 0.329
Date:                  2024-02-05 19:05 AIC:              7770.0934
No. Observations:      10000            BIC:              7813.3555
Df Model:              5                Log-Likelihood:   -3879.0
Df Residuals:          9994             LL-Null:          -5782.2
Converged:             1.0000           LLR p-value:      0.0000
No. Iterations:        7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.6938   0.0691  24.5100 0.0000  1.5584  1.8293
Tenure            -0.0624   0.0015 -42.0467 0.0000 -0.0653 -0.0595
dummy_One year    -1.7199   0.0782 -22.0043 0.0000 -1.8731 -1.5667
dummy_Two Year    -1.8592   0.0770 -24.1535 0.0000 -2.0101 -1.7083
dummy_Fiber Optic -0.7534   0.0597 -12.6253 0.0000 -0.8703 -0.6364
TechSupport        0.0916   0.0584   1.5703 0.1163 -0.0227  0.2060
==================================================================

"""
```

Remove TechSupport(p > 0.05) and check p-values

```
#Iterate on logistic regression model, remove TechSupport (p-value >
0.05)
X = pd.DataFrame(df_churn[["Tenure","dummy_One year","dummy_Two
Year","dummy_Fiber Optic"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.388028
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                 Logit            Method:           MLE
Dependent Variable:    Churn            Pseudo R-squared: 0.329
Date:                  2024-02-05 19:05 AIC:              7770.5569
No. Observations:      10000            BIC:              7806.6086
Df Model:              4                Log-Likelihood:   -3880.3
Df Residuals:          9995             LL-Null:          -5782.2
```

```
Converged:              1.0000           LLR p-value:      0.0000
No. Iterations:         7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.7298   0.0653  26.4813 0.0000  1.6018  1.8579
Tenure            -0.0624   0.0015 -42.0470 0.0000 -0.0653 -0.0595
dummy_One year    -1.7200   0.0781 -22.0110 0.0000 -1.8732 -1.5669
dummy_Two Year    -1.8592   0.0770 -24.1583 0.0000 -2.0101 -1.7084
dummy_Fiber Optic -0.7557   0.0596 -12.6686 0.0000 -0.8726 -0.6387
==================================================================

"""
```

Reduced logistic regression model

```
#Iterate on logistic regression model, remove TechSupport (p-value >
0.05)
X = pd.DataFrame(df_churn[["Tenure","dummy_One year","dummy_Two
Year","dummy_Fiber Optic"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_reduced = sm.Logit(y, X).fit()

mdl_reduced.summary2()

Optimization terminated successfully.
         Current function value: 0.388028
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                  Logit            Method:           MLE
Dependent Variable:     Churn            Pseudo R-squared: 0.329
Date:                   2024-02-05 19:05 AIC:              7770.5569
No. Observations:       10000            BIC:              7806.6086
Df Model:               4                Log-Likelihood:   -3880.3
Df Residuals:           9995             LL-Null:          -5782.2
Converged:              1.0000           LLR p-value:      0.0000
No. Iterations:         7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.7298   0.0653  26.4813 0.0000  1.6018  1.8579
Tenure            -0.0624   0.0015 -42.0470 0.0000 -0.0653 -0.0595
dummy_One year    -1.7200   0.0781 -22.0110 0.0000 -1.8732 -1.5669
```

```
dummy_Two Year    -1.8592   0.0770 -24.1583 0.0000 -2.0101 -1.7084
dummy_Fiber Optic -0.7557   0.0596 -12.6686 0.0000 -0.8726 -0.6387
=====================================================================

"""
```

E1.

After checking the variance inflation factor and p-value of each variable, I was able to iterate on my initial model several times until I was left with a complete reduced model. The VIF calculations indicated that "Responses", "Fixes", and "MonthlyCharge" needed to be removed due to high correlation with other predictors. Additionally, I was left with only statistically significant variables because I eliminated all independent variables with p-values greater than 0.05. The final model contains 4 independent variables from the 15 (including dummy variables) in the initial model. These variables are "Tenure", "dummy_One year", "dummy_Two Year", and "dummy_Fiber Optic".

To compare these two models, I decided to review their pseudo R-squared values. As mentioned previously, a single pseudo R-squared value means nothing on its own. However, a comparison of the values between the two models will inform us which one is superior. We can see from calculations above that the initial model has a pseudo R-squared of 0.573. The reduced model has a pseudo R-squared of 0.329. Based on this information, we can conclude that the initial model has a better fit than the reduced model.

A comparison of AIC values in both models confirms this conclusion. The initial model had an AIC of 4967.38 while the reduced model had an AIC of 7770.56. The lower value for the initial model shows that it is indeed a better fit

E2.

Below are the output and calculations of the logistic regression analysis, the confusion matrix, and the accuracy score of the model.

To create the confusion matrix, I used the "pred_table()" method from statsmodels.api on the reduced model. The output shows that there were 6,521 true positives, 829 false positives, 985 false negatives, and 1,665 true negatives. I calculated the accuracy of the model by adding the number of true negatives and true positives and dividing by the total number of predictions. The resulting accuracy score was 0.8186.

Initial Model

```
#Create initial logistic regression model [In-text citation: (LaRose et al, 2019)]
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_failure","Tenure","MonthlyCharge","dummy_One year","dummy_Two Year","dummy_Fiber Optic","TechSupport","Responses","Fixes","Replacements","Reliability"]])
```

```
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.246969
         Iterations 8

<class 'statsmodels.iolib.summary2.Summary'>
"""
                           Results: Logit
=====================================================================
Model:                 Logit             Method:             MLE
Dependent Variable:    Churn             Pseudo R-squared:   0.573
Date:                  2024-02-05 19:05  AIC:                4967.3750
No. Observations:      10000             BIC:                5068.3198
Df Model:              13                Log-Likelihood:     -2469.7
Df Residuals:          9986              LL-Null:            -5782.2
Converged:             1.0000            LLR p-value:        0.0000
No. Iterations:        8.0000            Scale:              1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                -4.9120   0.2916 -16.8432 0.0000 -5.4836 -4.3404
Outage_sec_perweek   -0.0012   0.0121  -0.0958 0.9237 -0.0250  0.0226
Contacts              0.0493   0.0360   1.3689 0.1710 -0.0213  0.1200
Yearly_equip_failure -0.0219   0.0567  -0.3861 0.6994 -0.1331  0.0893
Tenure               -0.1019   0.0025 -40.5148 0.0000 -0.1069 -0.0970
MonthlyCharge         0.0492   0.0013  38.4982 0.0000  0.0467  0.0517
dummy_One year       -2.9832   0.1143 -26.0934 0.0000 -3.2072 -2.7591
dummy_Two Year       -3.0552   0.1109 -27.5538 0.0000 -3.2726 -2.8379
dummy_Fiber Optic    -1.6949   0.0821 -20.6383 0.0000 -1.8559 -1.5339
TechSupport          -0.3598   0.0747  -4.8139 0.0000 -0.5063 -0.2133
Responses            -0.0552   0.0494  -1.1177 0.2637 -0.1519  0.0416
Fixes                -0.0240   0.0480  -0.5013 0.6162 -0.1180  0.0699
Replacements          0.0328   0.0439   0.7458 0.4558 -0.0534  0.1189
Reliability          -0.0327   0.0348  -0.9416 0.3464 -0.1008  0.0354
=====================================================================

"""
```

Model Reduction using Variance Inflation Factor

```
#Assign independent variables to dataframe X
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
```

```
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Responses","Fixes","Replacements","Reliability"]
])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

```
                  feature        VIF
0     Outage_sec_perweek  10.062425
1               Contacts   1.981046
2   Yearly_equip_failure   1.382617
3                 Tenure   2.630068
4          MonthlyCharge  13.493832
5          dummy_One year   1.376736
6          dummy_Two Year   1.438965
7       dummy_Fiber Optic   2.872565
8            TechSupport   1.626659
9              Responses  25.102549
10                 Fixes  22.969036
11          Replacements  18.970597
12           Reliability  10.141270
```

```
#Assign independent variables to dataframe X, remove Responses
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Fixes","Replacements","Reliability"]])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

```
                  feature        VIF
0     Outage_sec_perweek  10.052907
```

```
1                Contacts   1.980515
2   Yearly_equip_failure   1.381201
3                  Tenure   2.629971
4           MonthlyCharge  13.421418
5          dummy_One year   1.376498
6          dummy_Two Year   1.438877
7       dummy_Fiber Optic   2.869853
8             TechSupport   1.625990
9                   Fixes  16.150396
10           Replacements  15.982121
11            Reliability  10.114242
```

```
#Assign independent variables to dataframe X, remove Responses
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","MonthlyCharge","dummy_One year","dummy_Two
Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

```
                 feature        VIF
0     Outage_sec_perweek   9.908351
1                Contacts   1.977029
2   Yearly_equip_failure   1.380827
3                  Tenure   2.624860
4           MonthlyCharge  13.131709
5          dummy_One year   1.376427
6          dummy_Two Year   1.438423
7       dummy_Fiber Optic   2.868109
8             TechSupport   1.625615
9            Replacements   9.795162
10            Reliability   9.956283
```

```
#Assign independent variables to dataframe X, remove Responses
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","dummy_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])

#Create VIF dataframe [In-text citation: GeeksforGeeks]
```

```
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

#Calculate VIF for each independent variable
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

#print VIF data
print(vif_data)
```

```
                feature       VIF
0    Outage_sec_perweek  8.928276
1              Contacts  1.962045
2  Yearly_equip_failure  1.378012
3                Tenure  2.598159
4        dummy_One year  1.370356
5        dummy_Two Year  1.433700
6     dummy_Fiber Optic  2.762826
7           TechSupport  1.582278
8          Replacements  8.952135
9           Reliability  9.065586
```

Backwards Stepwise Regression

```
#Iterate on logistic regression model, remove Responses, Fixes, and
MonthlyCharge (VIF > 10)
X =
pd.DataFrame(df_churn[["Outage_sec_perweek","Contacts","Yearly_equip_f
ailure","Tenure","dummy_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Replacements","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.387693
         Iterations 7
```

```
<class 'statsmodels.iolib.summary2.Summary'>
"""
                           Results: Logit
=====================================================================
Model:                 Logit              Method:             MLE
Dependent Variable:    Churn              Pseudo R-squared:   0.330
Date:                  2024-02-05 19:05   AIC:                7775.8533
No. Observations:      10000              BIC:                7855.1670
```

```
Df Model:                 10                 Log-Likelihood:    -3876.9
Df Residuals:             9989               LL-Null:           -5782.2
Converged:                1.0000             LLR p-value:       0.0000
No. Iterations:           7.0000             Scale:             1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                 1.7702   0.1834   9.6498 0.0000  1.4107  2.1298
Outage_sec_perweek    0.0053   0.0095   0.5618 0.5743 -0.0133  0.0239
Contacts              0.0387   0.0286   1.3522 0.1763 -0.0174  0.0947
Yearly_equip_failure -0.0398   0.0450  -0.8839 0.3767 -0.1279  0.0484
Tenure               -0.0624   0.0015 -42.0431 0.0000 -0.0654 -0.0595
dummy_One year       -1.7213   0.0783 -21.9968 0.0000 -1.8746 -1.5679
dummy_Two Year       -1.8604   0.0770 -24.1537 0.0000 -2.0114 -1.7095
dummy_Fiber Optic    -0.7526   0.0597 -12.6072 0.0000 -0.8696 -0.6356
TechSupport           0.0954   0.0584   1.6337 0.1023 -0.0191  0.2099
Replacements         -0.0189   0.0275  -0.6856 0.4930 -0.0729  0.0351
Reliability          -0.0251   0.0276  -0.9100 0.3628 -0.0791  0.0290
=====================================================================

"""

#Iterate on logistic regression model, remove Outage_sec_perweek (p-value > 0.05)
X = 
pd.DataFrame(df_churn[["Contacts","Yearly_equip_failure","Tenure","dummy_One year","dummy_Two Year","dummy_Fiber Optic","TechSupport","Replacements","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.387708
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                           Results: Logit
=====================================================================
Model:                    Logit              Method:            MLE
Dependent Variable:       Churn              Pseudo R-squared:  0.329
Date:                     2024-02-05 19:05   AIC:               7774.1689
No. Observations:         10000              BIC:               7846.2723
Df Model:                 9                  Log-Likelihood:    -3877.1
Df Residuals:             9990               LL-Null:           -5782.2
```

```
Converged:               1.0000           LLR p-value:       0.0000
No. Iterations:          7.0000           Scale:             1.0000
---------------------------------------------------------------------
                       Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
const                  1.8238   0.1568  11.6305 0.0000  1.5164  2.1311
Contacts               0.0389   0.0286   1.3610 0.1735 -0.0171  0.0950
Yearly_equip_failure -0.0394   0.0450  -0.8758 0.3811 -0.1275  0.0487
Tenure               -0.0624   0.0015 -42.0431 0.0000 -0.0654 -0.0595
dummy_One year       -1.7210   0.0782 -21.9940 0.0000 -1.8744 -1.5676
dummy_Two Year       -1.8600   0.0770 -24.1520 0.0000 -2.0110 -1.7091
dummy_Fiber Optic    -0.7522   0.0597 -12.6012 0.0000 -0.8692 -0.6352
TechSupport            0.0952   0.0584   1.6296 0.1032 -0.0193  0.2097
Replacements         -0.0191   0.0275  -0.6940 0.4877 -0.0731  0.0349
Reliability          -0.0251   0.0276  -0.9119 0.3618 -0.0792  0.0289
=====================================================================

"""

#Iterate on logistic regression model, remove Replacements (p-value >
0.05)
X =
pd.DataFrame(df_churn[["Contacts","Yearly_equip_failure","Tenure","dum
my_One year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.387733
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                           Results: Logit
=====================================================================
Model:                   Logit             Method:            MLE
Dependent Variable:      Churn             Pseudo R-squared:  0.329
Date:                    2024-02-05 19:05  AIC:               7772.6507
No. Observations:        10000             BIC:               7837.5437
Df Model:                8                 Log-Likelihood:    -3877.3
Df Residuals:            9991              LL-Null:           -5782.2
Converged:               1.0000            LLR p-value:       0.0000
No. Iterations:          7.0000            Scale:             1.0000
---------------------------------------------------------------------
```

```
                         Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
----------------------------------------------------------------------
const                    1.7566   0.1232  14.2568 0.0000  1.5151  1.9981
Contacts                 0.0391   0.0286   1.3689 0.1710 -0.0169  0.0952
Yearly_equip_failure    -0.0390   0.0450  -0.8677 0.3856 -0.1271  0.0491
Tenure                  -0.0624   0.0015 -42.0456 0.0000 -0.0654 -0.0595
dummy_One year          -1.7224   0.0782 -22.0153 0.0000 -1.8757 -1.5690
dummy_Two Year          -1.8601   0.0770 -24.1560 0.0000 -2.0110 -1.7092
dummy_Fiber Optic       -0.7522   0.0597 -12.6020 0.0000 -0.8692 -0.6352
TechSupport              0.0946   0.0584   1.6199 0.1053 -0.0199  0.2091
Reliability             -0.0249   0.0276  -0.9037 0.3662 -0.0790  0.0291
======================================================================

"""
```

```
#Iterate on logistic regression model, remove Yearly_equip_failure (p-
value > 0.05)
X = pd.DataFrame(df_churn[["Contacts","Tenure","dummy_One
year","dummy_Two Year","dummy_Fiber
Optic","TechSupport","Reliability"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.387770
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
===================================================================
Model:                 Logit            Method:            MLE
Dependent Variable:    Churn            Pseudo R-squared:  0.329
Date:                  2024-02-05 19:05 AIC:               7771.4055
No. Observations:      10000            BIC:               7829.0882
Df Model:              7                Log-Likelihood:    -3877.7
Df Residuals:          9992             LL-Null:           -5782.2
Converged:             1.0000           LLR p-value:       0.0000
No. Iterations:        7.0000           Scale:             1.0000
-------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-------------------------------------------------------------------
const              1.7423   0.1221  14.2740 0.0000  1.5030  1.9815
Contacts           0.0392   0.0286   1.3703 0.1706 -0.0169  0.0952
Tenure            -0.0625   0.0015 -42.0539 0.0000 -0.0654 -0.0595
```

```
dummy_One year    -1.7232   0.0782 -22.0267 0.0000 -1.8765 -1.5699
dummy_Two Year    -1.8600   0.0770 -24.1580 0.0000 -2.0109 -1.7091
dummy_Fiber Optic -0.7523   0.0597 -12.6039 0.0000 -0.8693 -0.6353
TechSupport        0.0935   0.0584   1.6013 0.1093 -0.0209  0.2079
Reliability       -0.0250   0.0276  -0.9061 0.3649 -0.0790  0.0291
==================================================================

"""

#Iterate on logistic regression model, remove Yearly_equip_failure (p-
value > 0.05)
X = pd.DataFrame(df_churn[["Contacts","Tenure","dummy_One
year","dummy_Two Year","dummy_Fiber Optic","TechSupport"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()

Optimization terminated successfully.
         Current function value: 0.387811
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                 Logit            Method:           MLE
Dependent Variable:    Churn            Pseudo R-squared: 0.329
Date:                  2024-02-05 19:05 AIC:              7770.2266
No. Observations:      10000            BIC:              7820.6990
Df Model:              6                Log-Likelihood:   -3878.1
Df Residuals:          9993             LL-Null:          -5782.2
Converged:             1.0000           LLR p-value:      0.0000
No. Iterations:        7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.6550   0.0746  22.1819 0.0000  1.5087  1.8012
Contacts           0.0391   0.0286   1.3678 0.1714 -0.0169  0.0951
Tenure            -0.0625   0.0015 -42.0491 0.0000 -0.0654 -0.0595
dummy_One year    -1.7221   0.0782 -22.0201 0.0000 -1.8754 -1.5688
dummy_Two Year    -1.8598   0.0770 -24.1570 0.0000 -2.0107 -1.7089
dummy_Fiber Optic -0.7524   0.0597 -12.6067 0.0000 -0.8694 -0.6355
TechSupport        0.0926   0.0584   1.5866 0.1126 -0.0218  0.2070
==================================================================

"""
```

```python
#Iterate on logistic regression model, remove Contacts (p-value >
0.05)
X = pd.DataFrame(df_churn[["Tenure","dummy_One year","dummy_Two
Year","dummy_Fiber Optic","TechSupport"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()

mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.387905
         Iterations 7

<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                Logit            Method:           MLE
Dependent Variable:   Churn            Pseudo R-squared: 0.329
Date:                 2024-02-05 19:05 AIC:              7770.0934
No. Observations:     10000            BIC:              7813.3555
Df Model:             5                Log-Likelihood:   -3879.0
Df Residuals:         9994             LL-Null:          -5782.2
Converged:            1.0000           LLR p-value:      0.0000
No. Iterations:       7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.6938   0.0691  24.5100 0.0000  1.5584  1.8293
Tenure            -0.0624   0.0015 -42.0467 0.0000 -0.0653 -0.0595
dummy_One year    -1.7199   0.0782 -22.0043 0.0000 -1.8731 -1.5667
dummy_Two Year    -1.8592   0.0770 -24.1535 0.0000 -2.0101 -1.7083
dummy_Fiber Optic -0.7534   0.0597 -12.6253 0.0000 -0.8703 -0.6364
TechSupport        0.0916   0.0584   1.5703 0.1163 -0.0227  0.2060
==================================================================

"""
```

```python
#Iterate on logistic regression model, remove TechSupport (p-value >
0.05)
X = pd.DataFrame(df_churn[["Tenure","dummy_One year","dummy_Two
Year","dummy_Fiber Optic"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_initial = sm.Logit(y, X).fit()
```

```
mdl_initial.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.388028
         Iterations 7
```

```
<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                Logit            Method:           MLE
Dependent Variable:   Churn            Pseudo R-squared: 0.329
Date:                 2024-02-05 19:05 AIC:              7770.5569
No. Observations:     10000            BIC:              7806.6086
Df Model:             4                Log-Likelihood:   -3880.3
Df Residuals:         9995             LL-Null:          -5782.2
Converged:            1.0000           LLR p-value:      0.0000
No. Iterations:       7.0000           Scale:            1.0000
------------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const               1.7298   0.0653  26.4813 0.0000  1.6018  1.8579
Tenure             -0.0624   0.0015 -42.0470 0.0000 -0.0653 -0.0595
dummy_One year     -1.7200   0.0781 -22.0110 0.0000 -1.8732 -1.5669
dummy_Two Year     -1.8592   0.0770 -24.1583 0.0000 -2.0101 -1.7084
dummy_Fiber Optic -0.7557   0.0596 -12.6686 0.0000 -0.8726 -0.6387
==================================================================

"""
```

Reduced Logistic Regression Model

```
#Iterate on logistic regression model, remove TechSupport (p-value >
0.05)
X = pd.DataFrame(df_churn[["Tenure","dummy_One year","dummy_Two
Year","dummy_Fiber Optic"]])
X = sm.add_constant(X)

y = pd.DataFrame(df_churn[["Churn"]])

mdl_reduced = sm.Logit(y, X).fit()

mdl_reduced.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.388028
         Iterations 7
```

```
<class 'statsmodels.iolib.summary2.Summary'>
"""
                          Results: Logit
==================================================================
Model:                 Logit            Method:            MLE
Dependent Variable:    Churn            Pseudo R-squared:  0.329
Date:                  2024-02-05 19:05 AIC:               7770.5569
No. Observations:      10000            BIC:               7806.6086
Df Model:              4                Log-Likelihood:    -3880.3
Df Residuals:          9995             LL-Null:           -5782.2
Converged:             1.0000           LLR p-value:       0.0000
No. Iterations:        7.0000           Scale:             1.0000
------------------------------------------------------------------
                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
------------------------------------------------------------------
const              1.7298   0.0653  26.4813 0.0000  1.6018  1.8579
Tenure            -0.0624   0.0015 -42.0470 0.0000 -0.0653 -0.0595
dummy_One year    -1.7200   0.0781 -22.0110 0.0000 -1.8732 -1.5669
dummy_Two Year    -1.8592   0.0770 -24.1583 0.0000 -2.0101 -1.7084
dummy_Fiber Optic -0.7557   0.0596 -12.6686 0.0000 -0.8726 -0.6387
==================================================================

"""
```

Confusion Matrix

```
#Generate confusion matrix for reduced model
mdl_reduced.pred_table()
```

```
array([[6521.,  829.],
       [ 985., 1665.]])
```

Accuracy Calculation

```
#Calculate Accuracy Score for Reduced Model

#Calculate numerator
TP_TN = 6521 + 1665

#Calculate denominator
All_predictions = 6521 + 829 + 985 + 1665

#Divide true positive + true negative by total number of predictions
accuracy_score = TP_TN / All_predictions
print("Accuracy Score of Reduced Model: " + str(accuracy_score))
```

```
Accuracy Score of Reduced Model: 0.8186
```

E3.

Please see the attached code used to implement the linear regression model. The file name is "Task2_E3.ipynb".

F1.

Now that we have arrived at a reduced regression model, we can analyze the results and discuss key insights.

Regression Equation

The logistic regression demonstrates the relationship between the independent variables ("Tenure", "dummy_One year", "dummy_Two Year", and "dummy_Fiber Optic") and the log odds of "Churn".

$$ln\frac{p}{1-p}=1.7298-.0624(Tenure)-1.72(dummyOneyear)-1.8592(dummyTwoYear)-.7557(dummy$$

Interpretation of Coefficients

From the results of the final regression model, we can see the intercept of the equation is 1.7298. This represents the log-odds of "Churn" when all independent variables have a value of 0 (Saini, 2024). The coefficients, represent the amount by which the log odds of "Churn" will change if there is an increase of one unit of a given variable. To summarize, this is how an increase in one unit of each independent variable would affect the dependent variable.

Statistical and Practical Significance

One of the ways we can assess the statistical significance of the model is to look at the p-values of the variables. From the reduced multiple linear regression in D3, we can see that all variables have a p-value less than 0.05. We can interpret this to mean that all of the variables are statistically significant(Singh, 2020). The LLR p-value metric similarly allows us to evaluate the statistical significance of the model as a whole. Because this value is also less than my chosen alpha value of 0.05, we can interpret this to mean that the regression model itself is statistically significant.

Although this model is statistically significant, it has very limited practical significance. For one, it is rather obvious that the company should increase tenure for as long as possible to reduce the likelihood of churn. In other words, the model implies that the company needs to keep customers to prevent them from leaving. This seems redundant and not very insightful. Additionally, the small number of independent variables might also make it difficult to provide reliable predictions. There are only four independent variables in the final model and two of these variables directly influence each other ("dummy_One year" and dummy_Two Year").

After reviewing the final regression equation, we could infer that the company should focus on signing customers to one or two year fiber optic internet contracts and then retaining them for as long as possible. This just seems to limited to be useful in real life.

Limitations

Although logistic regression is a useful tools for predicting binary outcomes, it does have some limitations. One challenge in performing logistic regression is the presence of multicollinearity in the dataset. As we saw in the model reduction phase, 3 of the independent variables needed to be removed due to high correlation with other variables. It is possible that the presence of multicollinearity still had an impact on the results of the regression analysis.

The data cleaning methods used might have also had a negative impact on the model. By treating the 2,129 missing values in "InternetService" with modal imputation, I likely skewed the data which could have some repercussions. Additionally, my decision to retain the outliers in the data could cause it to lose accuracy.

Lastly, the methods I chose to reduce the model may have had a negative effect on its ability to make accurate predictions. In the initial model, I started off with 13 variables (9 independent variables plus 3 dummy variables). After using VIF to eliminate multicollinearity and removing variables with p-values greater than 0.05, I was left with 4 variables in the final model. It is possible that I removed too many variables with this strategy, which means the model might be too simple to make very accurate predictions.

F2.

Now that the model has been evaluated, the next step should be to apply the insights gained from the analysis. As mentioned previously, the final model does not appear to hold practical significance. The model may have been impacted by the presence of multicollinearity in the initial variables, which violates one of the key assumptions. It might be worthwhile to pursue a different regression method that does not adhere to the same assumptions. Additionally, it might be a good idea to pursue a different method of model reduction. The final model contained only 4 variables, 3 of which were dummy variables. A different method might allow us to retain more variables that the company can influence.

G.

Please see the link below for the panopto recording.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6b864611-9792-4d17-ae36-b10e00466d2d

H. Sources

Detecting Multicollinearity with VIF – Python. (n.d.). GeeksforGeeks. https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/

Chantal D. Larose, & Daniel T. Larose. (2019). Data Science Using Python and R. Wiley.

I. Sources

The 6 Assumptions of Logistic Regression (With Examples). (2020, October 13). Statology. https://www.statology.org/assumptions-of-logistic-regression/

R vs Python for Data Science: Which One Is the Best Programming Language for Data Scientists in 2024? (n.d.). Turing. https://www.turing.com/kb/best-programming-language-for-data-science-r-vs-python

Walwadkar, D. (2022, April 26). Regression A to Z : Choosing the Correct Type of Regression Analysis. Medium. https://medium.com/@dnyaneshwalwadkar/regression-a-to-z-choosing-the-correct-type-of-regression-analysis-4cfb29ae5a1

Shmueli, Galit. "Categorical predictors: how many dummies to use in regression vs. k-nearest neighbors." BzST | Business Analytics, Statistics, Teaching, 19 Aug. 2015, www.bzst.com/2015/08/categorical-predictors-how-many-dummies.html.

Chantal D. Larose, & Daniel T. Larose. (2019). Data Science Using Python and R. Wiley.

UCLA Statistical Consulting Group. (n.d.). FAQ: What are Pseudo R-Squareds?. UCLA Statistical Methods and Analytics. https://stats.oarc.ucla.edu/other/mult-pkg/faq/general/faq-what-are-pseudo-r-squareds/

Bressler, N. (2-22, November 23). How to Check the Accuracy of Your Machine Learning Model. Deep Checks. https://deepchecks.com/how-to-check-the-accuracy-of-your-machine-learning-model/#:~:text=Accuracy%20score%20in%20machine%20learning%20is%20an%20evaluation%20metric%20that,the%20total%20number%20of%20p

Saini, A. (2024, January 5). A Beginner's Guide to Logistic Regression. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/

Singh, A. (2020, March 4). Understanding the P-Value in Regression. Medium. https://medium.com/analytics-vidhya/understanding-the-p-value-in-regression-1fc2cd2568af#:~:text=The%20P%2DValue%20as%20you,observed%20in%20the%20sample%20also