D208 – Assessment NBM2 TASK 2: LOGISTIC REGRESSION FOR PREDICTIVE MODELING

PART I – RESEARCH QUESTION

PART A1 – SUMMARIZE ONE RESEARCH QUESTION THAT IS RELEVANT TO A REAL WORLD ORGANIZATIONAL SITUATION CAPTURED IN THE DATASET THAT I HAVE SELECTED AND I WILL ANSWER USING LOGISTIC REGRESSION

Question: A telecommunication company has requested a profile prediction of customers who are most likely to terminate their contract, churn. Since maintaining a customer is much cheaper than acquiring a new one, the analyst will have to review the customer data provided to identify which variables would predict customer churn the best. With that information, the telecommunication company will be able to create an effective plan to avoid losing current customers. The model will predict if the customer will churn or not. Putting that determined customer in one of the two groups: "Yes" (churn) or "No" (no churn). Since this is a classification problem, using logistic regression is the right approach for this problem.

PART A2 – GOALS OF THE DATA ANALYSIS

The main goal of this analysis is to determine what is the customer profile who are more likely to churn and with this information create a machine learning algorithm to predict customer churn.

PART II – METHOD JUSTIFICATION

PART B1 – SUMMARIZE THE ASSUMPTIONS OF A LOGISTIC REGRESSION MODEL [1]

1-) Independece of errors

2-) Linearity in the logit for continuous variables

3-) No multicollinearity

4-) Absence of strongly influencial outliers

PART B2 – DESCRIBE THE BENEFITS OF USING THE TOOL YOU HAVE CHOSEN IN SUPPORT OF VARIOUS PHASES OF THE ANALYSIS

I have started D206 using PYTHON and I intend to continue using this free tool. PYTHON has been largely used in DATA ANALYTICS projects. As I stated in my previous report on D206 "PyCharm is an integrated development environment used in computer programming, specifically for the Python language". It is a free and powerful language. Python is able to read Excel files, clean untidy data, manage missing values and more. I have done all kinds of graphs,

from pie charts and histograms to scatter plots in Python, it also offers packages to perform PCA and Factor analysis, Regression and Decision tree modeling to generate a prediction function.

PART B3 – EXPLAIN WHY LOGISTIC REGRESSION IS AN APPROPRIATE TECHNIQUE TO ANALYZE THE RESEARCH QUESTION SUMMARIZED IN PART I

I am going to use logistic regression to predict if a customer will churn or not (which is a classification problem). The developed model will try to predict the probability of a customer to belong to one group (churn) or another (not churn). The definition of logistic regression is the method is used to model the probability of effects from causes [1]. Logistic Regression is used for categorical Y-variables, in our case, Churn.

PART III – DATA PREPARATION

PART C – SUMMARIZE THE DATA PREP PROCESS FOR LOGISTIC REGRESSION ANALYSIS BY DOING THE FOLLOWING:

PART C1 – DESCRIBE YOUR DATA PREP GOALS AND THE DATA MANIPULATIONS THAT WILL BE USED TO ACHIEVE THE GOALS

Starting in D206, I worked with the "churn dataset". The first step was to drop some variables presented in the original dataset that would not help to predict customer churn. Those variables dropped were: CaseOrder, Interaction, City, State, County, Zip, Lat, Lng and Population and also the very first column with only numbers and no description was also dropped at this first step. In addition to these columns, I am deleting the variables "TimeZone", "Job" and "Customer_ID".

Second step was to rename the last 8 columns with the appropriate customer survey label:

```
#Start cleaning up the data, getting rid of irrelevant data
df = churn_df.drop(churn_df.columns[0], axis = 1)
df.head()
print(df)
```

```
#Removing the 9 columns: CaseOrder, Interaction, City, Zip, Lat and Lng
df_clean = df.drop(columns=['CaseOrder', 'Interaction', 'City', 'State',
'County', 'Zip', 'Lat', 'Lng', 'Population'])
print(df_clean)
```

```
#Renaming the last 8 survey columns for a more descriptive value
df_clean.rename(columns = {'item1':'CS Responses', 'item2':'CS Fixes',
'item3':'CS Replacements', 'item4':'CS Reliability', 'item5':'CS Options',
'item6':'CS Respectfulness', 'item7':'CS Courteous', 'item8':'CS Listening'},
inplace=True)
print(df_clean)
```

These steps are not part of my current code since I have been working with the clean churn data set.

We have numerical and categorical variables in the dataset. Mathematical models cannot be done with categorical variables so for the ones with two levels of category, YES and NO, I converted them to 1 and 0, respectively.

```
#Changing Variables from NO/YES to 0/1
churn_df2.Churn.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.Phone.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.PaperlessBilling.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.Techie.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.Port_modem.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.Tablet.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.Multiple.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.OnlineSecurity.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.OnlineBackup.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.DeviceProtection.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.TechSupport.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.StreamingTV.replace({"Yes":1, "No":0}, inplace = True)
churn_df2.StreamingMovies.replace({"Yes":1, "No":0}, inplace = True)
```

For categorical variables with 3 levels or more, I created dummy variables and later concatenated both data sets and dropped the original variables:

```
#Creating a dummy variable for some of the categorical variables with 3 or
more levels

dummy1 = pd.get_dummies(churn_df2[['Marital', 'Contract', 'PaymentMethod',
'Gender', 'InternetService', 'Area', 'Employment']])

#Adding the results to the master dataframe
churn_df2 = pd.concat([churn_df2, dummy1], axis=1)

#We have created dummies for the below variables, so we can drop them
churn_df2 = churn_df2.drop(['Marital', 'Contract', 'PaymentMethod', 'Gender',
'InternetService', 'Area','Employment'], 1)
```

PART C2 – DISCUSS THE SUMMARY STATS, INCLUDING THE TARGET VARIABLE AND ALL PREDICTOR VARIABLES THAT YOU WILL NEED TO GATHER FROM THE DATA SET TO ANSWER THE RESEARCH QUESTION

```
# Loading the Clean Churn Dataset from D206
df = pd.read_csv('/Users/bia/Desktop/churn_clean.csv')
df_stats = df.describe()
# print(df_stats)
```

	Children	Age	...	Courteous	Listening
count	7505.000000	7525.000000	...	10000.000000	10000.000000
mean	2.095936	53.275748	...	3.509500	3.495600
std	2.154758	20.753928	...	1.028502	1.028633
min	0.000000	18.000000	...	1.000000	1.000000
25%	0.000000	35.000000	...	3.000000	3.000000
50%	1.000000	53.000000	...	4.000000	3.000000
75%	3.000000	71.000000	...	4.000000	4.000000
max	10.000000	89.000000	...	7.000000	8.000000

[8 rows x 18 columns]

Picture 1: Summary Stats of the data

Calculating the churn rate:

```
#Calculating the Churn Rate
churn_rate=df.Churn.value_counts() / len(df)
print(churn_rate)
```

Picture 2: Churn Rate

We can clearly see that the data provided is not equally distributed. We have the majority of cases of customers who haven't churned (73.5%) and only **26.5%** of total customers have churned. Ideally, we would have an equal ratio of cases to assure a good fit for the model prediction of churn rate.

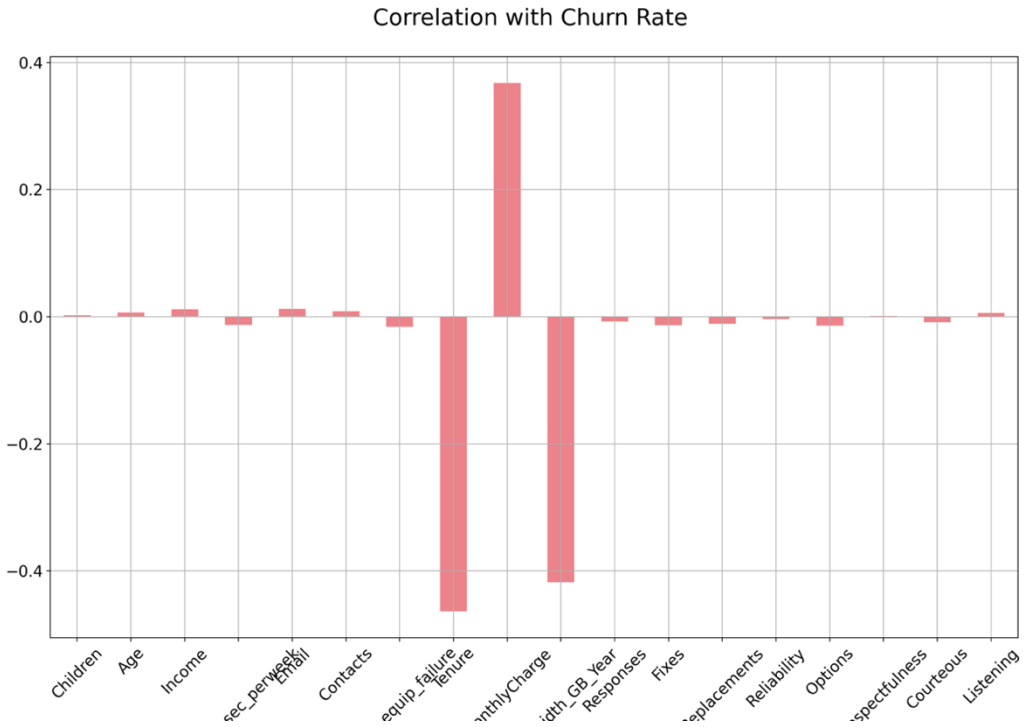
From my reports delivered in **D206** & **D207**, I could find some relationship in between these variables and churn rate:

Num: Tenure, Bandwidth_GB_Year, MonthlyCharge,
Cat: StreamingTV, StreamingMovies, Multiple, InternetService and Contract

For the numerical variables and categorical with 2 or less values that I had to encode in D206, I generated a correlation plot so I could visualize which variables had a greater influence on the churn rate. I found that Bandwidth, Tenure, MonthlyCharge, StreamingTV, StreamingMovies

and Multiple were the key variables. I also included InternetService because from the bar plot against churn rate, we can clearly see that customers with DSL internet churn more and Contract because customers on a month-to-month clearly are more likely to churn.

```
#Correclation Matrix for Numeric and label encoded columns
sns.heatmap(churn_df2.corr(), annot=True)
plt.show()
```

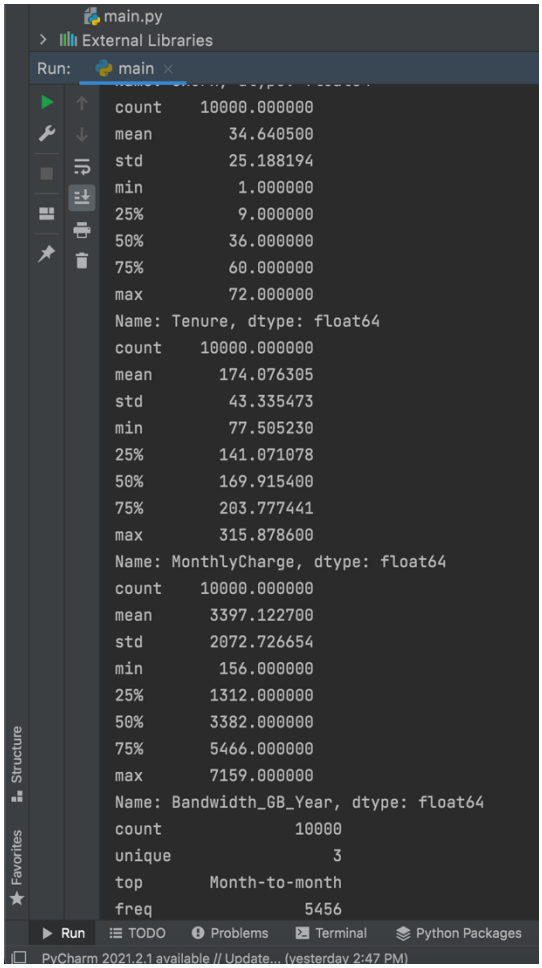


Picture 3: Correlation Plot

From the correlation plot above we can see that variables such as MonthlyCharge, StreamingMovies and StreamingTV, Multiple, InternetService, Contract and Tenure

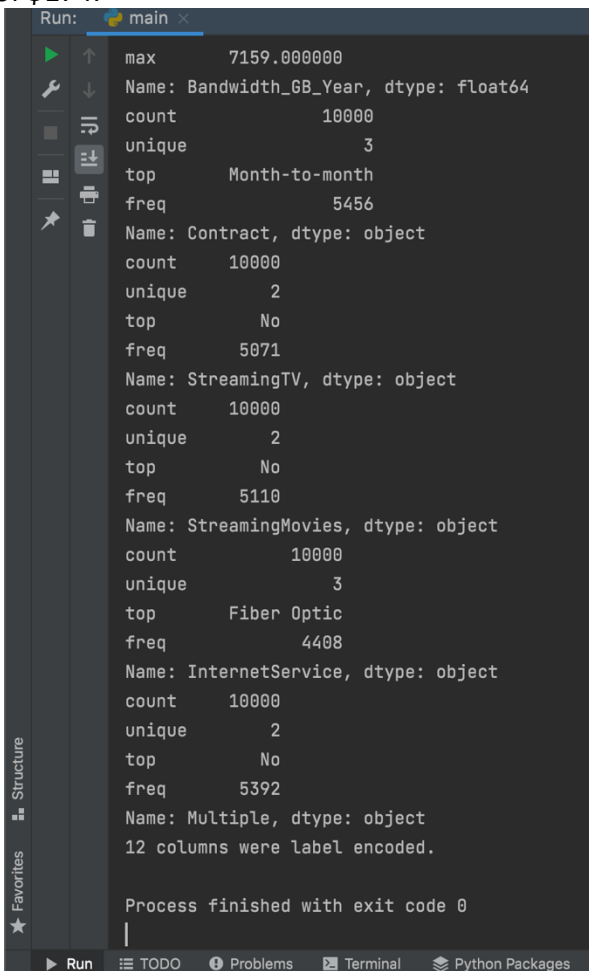
```
#Basic Stats of important variables

#Tenure
df_stats_tenure = churn_df['Tenure'].describe()
print(df_stats_tenure)

#Monthly Charges
df_stats_charge = churn_df['MonthlyCharge'].describe()
print(df_stats_charge)

#Bandwidth
df_stats_band = churn_df['Bandwidth_GB_Year'].describe()
print(df_stats_band)
```

```
#Contract
df_stats_contract = churn_df['Contract'].describe()
print(df_stats_contract)

#Streaming TV
df_stats_sttv = churn_df['StreamingTV'].describe()
print(df_stats_sttv)

#Straeming Movies
df_stats_stmovies = churn_df['StreamingMovies'].describe()
print(df_stats_stmovies)

#Internet
df_stats_internet = churn_df['InternetService'].describe()
print(df_stats_internet)

#Multiple
df_stats_multiple = churn_df['Multiple'].describe()
print(df_stats_multiple)
```



Picture 4: Displaying Basic Stats I

Tenure: we notice that the average customer is retained for an average of 34 months with a monthly charge average of $174.



Picture 5: Displaying Basic Stats II

Most customers are on a month-to-month contract. Which I have already explained in D207 that it's not good for customer retention. We will also see in sections to come in this report that customers with a month-to-month contract churn more than customers with longer contracts.

PART C3 – EXPLAIN THE STEPS USED TO PREPARE THE DATA FOR THE ANALYSIS, INCLUDING THE ANNOTATED CODE

The dataset has is originally presented with missing values. I replaced numerical missing values with the median of each column and categorical missing values with the mode of each variable (the most common value).

Searching for missing values:

```
null_values = df_clean.isna().any()
print(null_values)
```

Numerical:

```
#Filling the missing data with the median of each variable
#We saw that the columns Children, Age, Income, Tenure and␣Bandwidth_GB_Year
have missing values

na_cols = df_clean.isna().any()
na_cols = na_cols[na_cols == True].reset_index()
na_cols = na_cols["index"].tolist()
for col in df_clean.columns[1:]:
    if col in na_cols:
        if df_clean[col].dtype != 'object':
            df_clean[col] =
df_clean[col].fillna(df_clean[col].median()).round(0)
```

Categorical:
PHONE → YES
TECHIE → NO

```
#Phone and Techie Columns are categorical with missing values as well
print(df_clean['Phone'].unique())
print(df_clean['Techie'].unique())
#Since We have "YES" "NO" and "NAN" we will need to replace the nan values
for something
df_stats_phone = df_clean['Phone'].describe()
print(df_stats_phone)

df_stats_phone = df_clean['Techie'].describe()
print(df_stats_phone)

#Since these are categorical columns, I am going to replace the "NAN" values
for whatever shows more
#Phone --> "YES"
#Techie --> "NO"

df_clean = df_clean.fillna(df.mode().iloc[0])
```

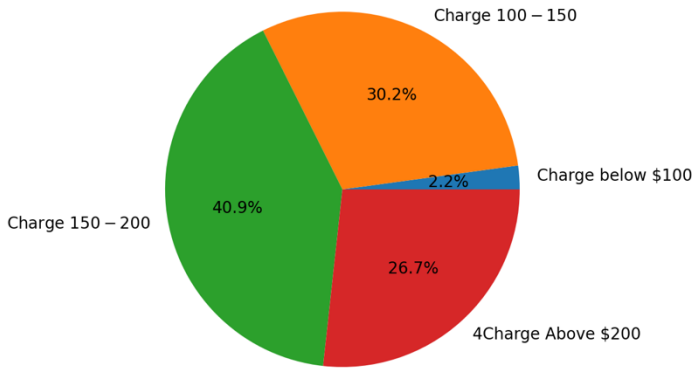
PART C4 – GENERATE UNIVARIATE AND BIVARIATE VISUALIZATIONS FOR THE DISTRIBUTIONS OF VARIABLES IN THE CLEANED DATA SET. INCLUDE THE TARGET VARIABLE IN YOUR BIVARIATE VISUALIZATIONS

As demonstrated in my report in D207:

1. Univariate Stats: the simplest form of statistics. I am going to generate pie charts for all important variables in this analysis:

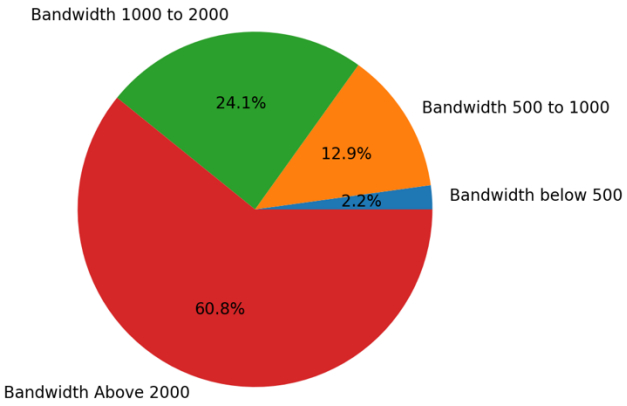
```
#Monthly Charge Evaluation
def monthly_charge(df_clean):
    if df_clean['MonthlyCharge'] <= 100:
        return "Charge Level 1"
```

```
    elif (df_clean['MonthlyCharge'] > 101) & (df_clean['MonthlyCharge'] <=
150):
        return "Charge Level 2"
    elif (df_clean['MonthlyCharge'] > 151) & (df_clean['MonthlyCharge'] <=
200):
        return "Charge Level 3"
    elif (df_clean['MonthlyCharge'] > 201):
        return "Charge Level 4"

df_clean_6 = df_clean.copy()
df_clean_6['Monthly_Charge'] = df_clean_6.apply(lambda
df_clean_6:monthly_charge(df_clean_6), axis=1)

charge1_count = df_clean_6['Monthly_Charge'].value_counts()['Charge Level 1']
charge2_count = df_clean_6['Monthly_Charge'].value_counts()['Charge Level 2']
charge3_count = df_clean_6['Monthly_Charge'].value_counts()['Charge Level 3']
charge4_count = df_clean_6['Monthly_Charge'].value_counts()['Charge Level 4']

charge_total = charge1_count + charge2_count + charge3_count + charge4_count

pct_charge1 = (charge1_count / charge_total) * 100
pct_charge2 = (charge2_count / charge_total) * 100
pct_charge3 = (charge3_count / charge_total) * 100
pct_charge4 = (charge4_count / charge_total) * 100

#Pie Chart of Income

plt.figure(figsize=(5,5))
labels = ["Charge below 100", "Charge $101-$150", "Charge $151 - $200",
"Charge above $200"]
values = [pct_charge1, pct_charge2, pct_charge3, pct_charge4]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



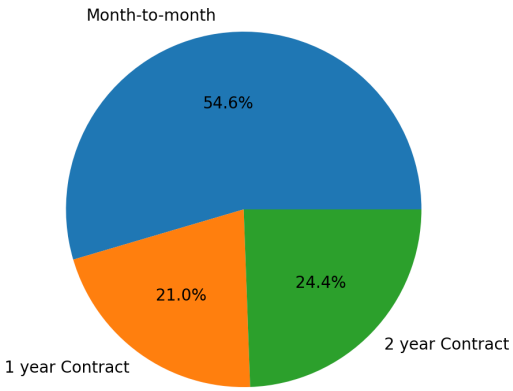
Picture 6: MonthlyCharge Pie Chart

```
#Bandwidth_GB_Year Aalysis
print(df_clean['Bandwidth_GB_Year'].unique())
def bandwidth(df_clean):
    if df_clean['Bandwidth_GB_Year'] <= 500:
        return "Bandwidth Below 500"
    elif (df_clean['Bandwidth_GB_Year'] > 500) &
(df_clean['Bandwidth_GB_Year'] <= 1000):
        return "Bandwidth 500 - 1000"
    elif (df_clean['Bandwidth_GB_Year'] > 1000) &
(df_clean['Bandwidth_GB_Year'] <= 2000):
        return "Bandwidth 1000 - 2000"
    elif (df_clean['Bandwidth_GB_Year'] > 2000):
        return "Bandwidth Above 2000"

df_clean_10 = df_clean.copy()
df_clean_10['Bandwidth_GB_Year'] = df_clean_10.apply(lambda
df_clean_10:bandwidth(df_clean_10), axis=1)

bd1_count = df_clean_10['Bandwidth_GB_Year'].value_counts()['Bandwidth Below
500']
bd2_count = df_clean_10['Bandwidth_GB_Year'].value_counts()['Bandwidth 500 -
1000']
bd3_count = df_clean_10['Bandwidth_GB_Year'].value_counts()['Bandwidth 1000 -
2000']
bd4_count = df_clean_10['Bandwidth_GB_Year'].value_counts()['Bandwidth Above
2000']

bd_total = bd1_count + bd2_count + bd3_count + bd4_count

pct_bd1 = (bd1_count / bd_total) * 100
pct_bd2 = (bd2_count / bd_total) * 100
pct_bd3 = (bd3_count / bd_total) * 100
pct_bd4 = (bd4_count / bd_total) * 100
#Pie Chart of Monthly Charge

plt.figure(figsize=(5,5))
labels = ["Bandwidth below 500", "Bandwidth 500 to 1000", "Bandwidth 1000 to
2000", "Bandwidth Above 2000"]
values = [pct_bd1, pct_bd2, pct_bd3, pct_bd4]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



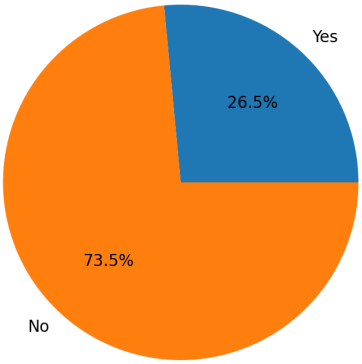
Picture 7: Bandwidth_GB_Year Pie Chart

```
print(df_clean['Contract'].unique())

month_to_month_count= df_clean['Contract'].value_counts()['Month-to-month']
one_year_count= df_clean['Contract'].value_counts()['One year']
two_year_count = df_clean['Contract'].value_counts()['Two Year']

contract_total = month_to_month_count + one_year_count + two_year_count

contract1_pct = (month_to_month_count / contract_total) * 100
contract2_pct = (one_year_count / contract_total) * 100
contract3_pct = (two_year_count / contract_total) * 100

#Pie Chart of types of Contracts

plt.figure(figsize=(5,5))
labels = ["Month-to-month", "1 year Contract", "2 year Contract"]
values = [contract1_pct, contract2_pct, contract3_pct]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



Picture 8:Contract Pie Chart

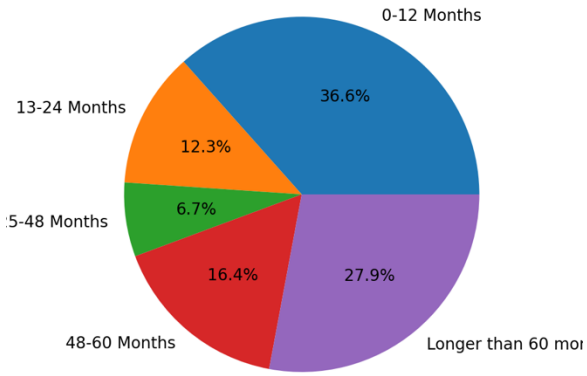
```
#Creating a Pie Chart to Visualize the Churn rate

plt.figure(figsize=(5,5))
labels = ["Yes", "No"]
values = [26.5, 73.5]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



Picture 9:Churn Pie Chart

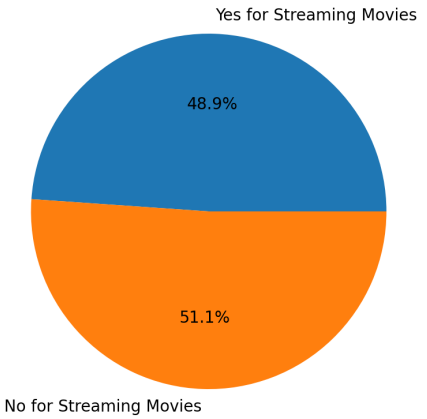
```
#Tenure Evaluation: I am going to create tenure intervals (0-12 months, 13-24
months, 25-48 months, 49-60 months, greater than 60 months)

def tenure_bands(df_clean):
    if df_clean['Tenure'] <= 12:
```

```
        return "Tenure_0-12"
    elif (df_clean['Tenure'] > 12) & (df_clean['Tenure'] <= 24):
        return "Tenure_13-24"
    elif (df_clean['Tenure'] > 24) & (df_clean['Tenure'] <= 48):
        return "Tenure_25-48"
    elif (df_clean['Tenure'] > 48) & (df_clean['Tenure'] <= 60):
        return "Tenure_49-60"
    elif df_clean['Tenure'] > 60:
        return "Tenure_gt_60"

df_clean_2 = df_clean.copy()
df_clean_2['tenure_group'] = df_clean_2.apply(lambda
df_clean_2:tenure_bands(df_clean_2), axis=1)
df_clean_2['tenure_group']

band1_count= df_clean_2['tenure_group'].value_counts()['Tenure_0-12']
band2_count = df_clean_2['tenure_group'].value_counts()['Tenure_13-24']
band3_count= df_clean_2['tenure_group'].value_counts()['Tenure_25-48']
band4_count= df_clean_2['tenure_group'].value_counts()['Tenure_49-60']
band5_count= df_clean_2['tenure_group'].value_counts()['Tenure_gt_60']

band_total = band1_count + band2_count + band3_count + band4_count +
band5_count

pct_band1 = (band1_count / band_total) * 100
pct_band2 = (band2_count / band_total) * 100
pct_band3 = (band3_count / band_total) * 100
pct_band4 = (band4_count / band_total) * 100
pct_band5 = (band5_count / band_total) * 100

#Pie Chart of Tenure Time

plt.figure(figsize=(5,5))
labels = ["0-12 Months", "13-24 Months", "25-48 Months", "48-60 Months",
"Longer than 60 months"]
values = [pct_band1, pct_band2, pct_band3, pct_band4, pct_band5]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



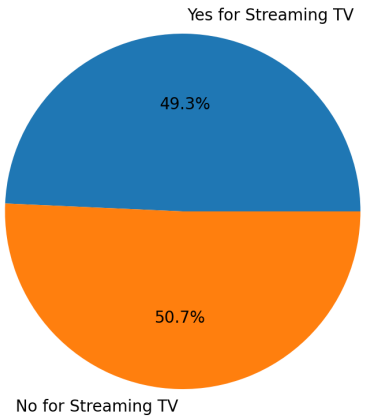
Picture 10: Tenure Pie Chart

```
#Streaming Movies
sm_yes_count = churn_df['StreamingMovies'].value_counts()['Yes']
sm_no_count = churn_df['StreamingMovies'].value_counts()['No']

streamingm_total = sm_yes_count + sm_no_count

sm_yes_pct = (sm_yes_count / streamingm_total) * 100
sm_no_pct = (sm_no_count / streamingm_total) * 100

#Pie Chart of type of Streaming Movies
plt.figure(figsize=(5,5))
labels = ["Yes for Streaming Movies", "No for Streaming Movies"]
values = [sm_yes_pct, sm_no_pct]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



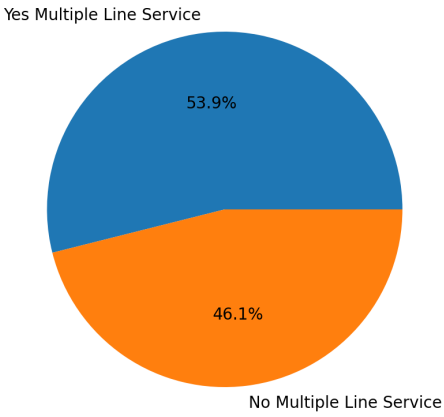
Picture 11: StreamingMovies Pie Chart

```
#Streaming TV
st_yes_count = churn_df['StreamingTV'].value_counts()['Yes']
st_no_count = churn_df['StreamingTV'].value_counts()['No']

streamingtv_total = st_yes_count + st_no_count

st_yes_pct = (st_yes_count / streamingtv_total) * 100
st_no_pct = (st_no_count / streamingtv_total) * 100

#Pie Chart of type of Streaming TV

plt.figure(figsize=(5,5))
labels = ["Yes for Streaming TV", "No for Streaming TV"]
values = [st_yes_pct, st_no_pct]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



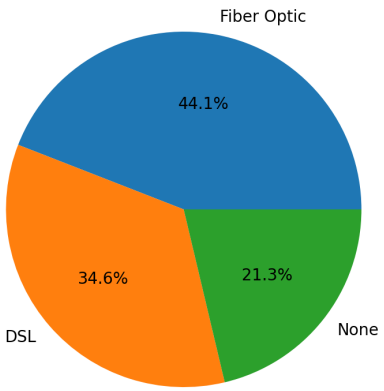
Picture 12: StreamingTV Pie Chart

```
#Multiple lines service
no_multiple_count = churn_df['Multiple'].value_counts()['Yes']

yes_multiple_count = churn_df['Multiple'].value_counts()['No']

multiple_total = no_multiple_count + yes_multiple_count

no_multiple_pct = (no_multiple_count / multiple_total) *100
yes_multiple_pct = (yes_multiple_count / multiple_total) *100


#Pie Chart of Multiple
plt.figure(figsize=(5,5))
labels = ["Yes Multiple Line Service", "No Multiple Line Service"]
values = [yes_multiple_pct, no_multiple_pct]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



Picture 13:Multiple Pie Chart

```
#Type of Internet Service
print(df_clean['InternetService'].unique())

fiber_optic_count = df_clean['InternetService'].value_counts()['Fiber Optic']
dsl_count = df_clean['InternetService'].value_counts()['DSL']
none_count = df_clean['InternetService'].value_counts()['None']

service_total = fiber_optic_count + dsl_count + none_count

fiber_pct = (fiber_optic_count / service_total) * 100
dsl_pct = (dsl_count / service_total) * 100
none_pct = (none_count / service_total) * 100

#Pie Chart of type of Internet Service

plt.figure(figsize=(5,5))
labels = ["Fiber Optic", "DSL", "None"]
values = [fiber_pct, dsl_pct, none_pct]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

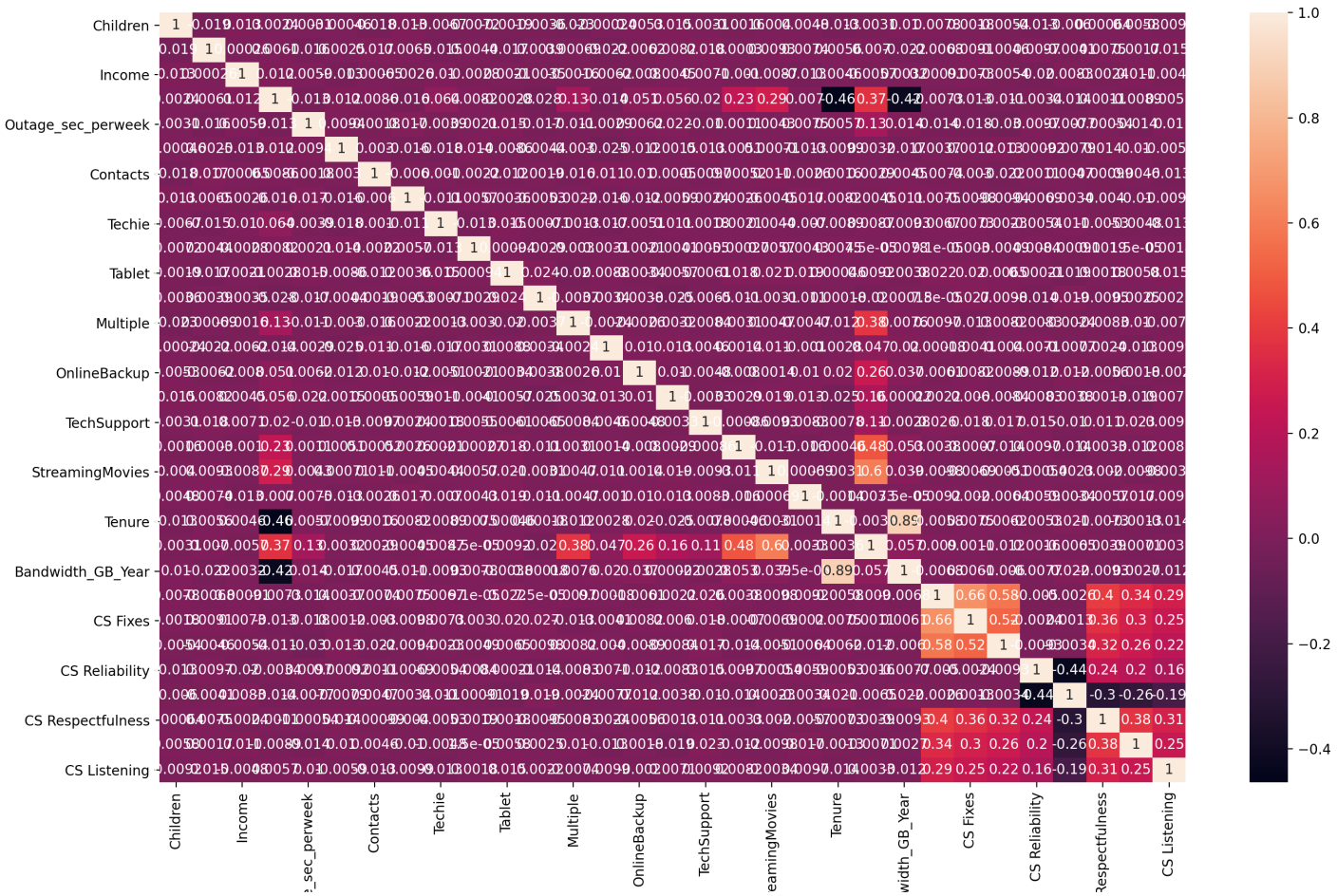


Picture 14: InternetService Pie Chart

2. **Bivariate Stats**: After analyzing all important variables independently, it's time to analyze if there is a relationship between two variables. When two variables have a correlation it means that when one variable changes the second variable will also change by a certain amount.

I am going to create graphs of each variable against the churn rate as I presented in my report from D206 [4]. And also a correlation matrix.

```
#Creating a label encoder object
le = LabelEncoder()
churn_df2['Churn'] = le.fit_transform(churn_df2['Churn'])
# Label Encoding will be used for categorical variables with 2 or less unique
values
le_count = 0
for col in churn_df.columns[1:]:
    if churn_df2[col].dtype == 'object':
        if len(list(churn_df2[col].unique())) <= 2:
            le.fit(churn_df2[col])
            churn_df2[col] = le.transform(churn_df2[col])
            le_count += 1
print('{} columns were label encoded.'.format(le_count))
#**********************************************************************
****************************************

#Correclation Matrix for all Numeric Columns
sns.heatmap(churn_df.corr(), annot=True)
plt.show()
```



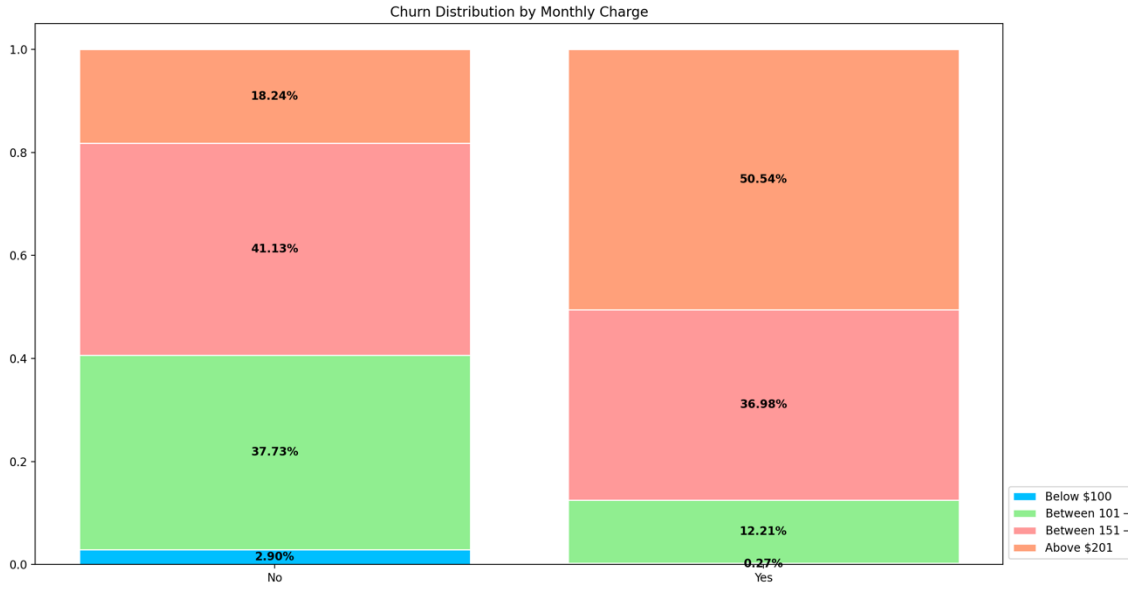
Picture 15: Correlation Matrix

From the correlation matrix above we see that the customer survey variables are all correlated to each other and "Bandwidth_GB_Year" is highly correlated with "Tenure". I am going to drop Bandwidth and customer survey variables at a later stage of the modelling.

Monthly Charge and Churn Rate:

```
#Monthly Charge and Churn Rate
no_churn = ((df_clean_2[df_clean_2['Churn']
=='No']['Monthly_Charge'].value_counts()) /(df_clean_2[df_clean_2['Churn']==
'No']['Monthly_Charge'].value_counts().sum()))
yes_churn = ((df_clean_2[df_clean_2['Churn']
=='Yes']['Monthly_Charge'].value_counts())
/(df_clean_2[df_clean_2['Churn']=='Yes']['Monthly_Charge'].value_counts().sum
()))

# Getting values from the group and categories
x_labels = df_clean_2['Churn'].value_counts().keys().tolist()
level1 = [no_churn['Charge below $100'], yes_churn['Charge below $100']]
level2 = [no_churn['Charge $100 - $150'], yes_churn['Charge $100 - $150']]
level3 = [no_churn['Charge $150 - $200'], yes_churn['Charge $150 - $200']]
```

```
level4 = [no_churn['Charge Above $200'], yes_churn['Charge Above $200']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, level1, color='#00BFFF', label=('Charge below $100'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, level2, bottom=level1, color='lightgreen',
label=('Charge $100 - $150'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, level3, bottom=np.array(level2) + np.array(level1),
color='#FF9999', label=('Charge $150 - $200'), edgecolor='white',
width=barWidth)
ax4 = plt.bar(x_labels, level4, bottom=np.array(level1) + np.array(level2) +
np.array(level3), color='#FFA07A', label=('Charge Above $200'),
edgecolor='white', width=barWidth)

plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by Monthly Charge')

for r1, r2, r3, r4 in zip(ax1, ax2, ax3, ax4):
    h1 = r1.get_height()
    h2 = r2.get_height()
    h3 = r3.get_height()
    h4 = r4.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r3.get_x() + r3.get_width() / 2., h1 + h2 + h3 / 2.,
'{:.2%}'.format(h3), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r4.get_x() + r4.get_width() / 2., h1 + h2 + h3 + h4 / 2.,
'{:.2%}'.format(h4), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



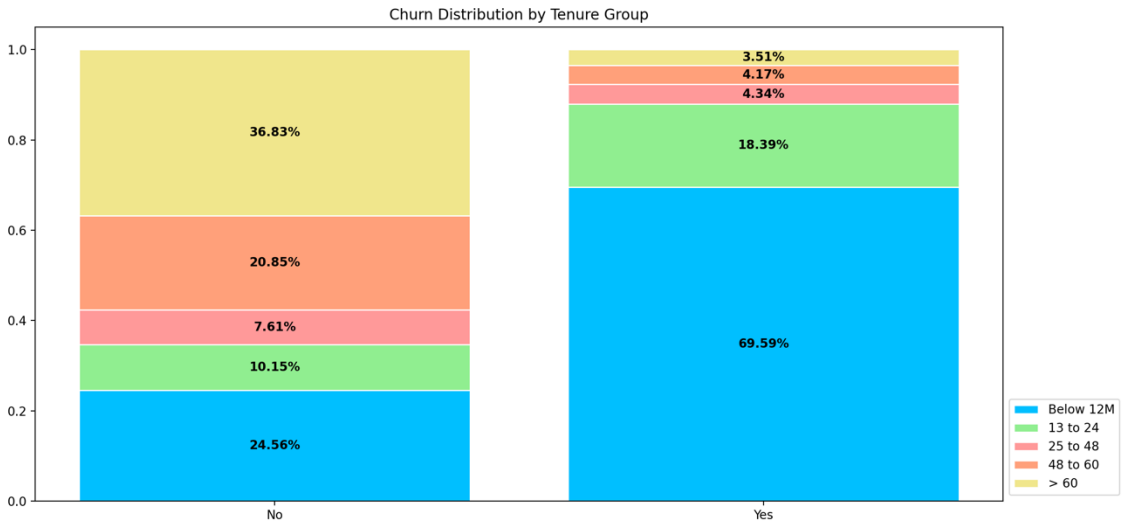
Picture 16: MonthlyCharge and Churn Rate

This graph above shows that we see a big customer churn when monthly charges are above $200 per month.

Tenure and Churn Rate:

```
#Tenure and Churn Rate
no_churn =
((df_clean_3[df_clean_3['Churn']=='No']['tenure_group'].value_counts())
/(df_clean_3[df_clean_3['Churn']=='No']['tenure_group'].value_counts().sum())
)
yes_churn =
((df_clean_3[df_clean_3['Churn']=='Yes']['tenure_group'].value_counts())
/(df_clean_3[df_clean_3['Churn']=='Yes']['tenure_group'].value_counts().sum()
))

# Getting values from the group and categories
x_labels = churn_df['Churn'].value_counts().keys().tolist()
t_0_12 = [no_churn['Tenure_0-12'], yes_churn['Tenure_0-12']]
t_13_24 = [no_churn['Tenure_13-24'], yes_churn['Tenure_13-24']]
t_25_48 = [no_churn['Tenure_25-48'], yes_churn['Tenure_25-48']]
t_49_60 = [no_churn['Tenure_49-60'], yes_churn['Tenure_49-60']]
t_gt_60 = [no_churn['Tenure_gt_60'], yes_churn['Tenure_gt_60']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, t_0_12, color='#00BFFF', label=('Below 12M'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, t_13_24, bottom=t_0_12, color='lightgreen',
label=('13 to 24'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, t_25_48, bottom=np.array(t_0_12) + np.array(t_13_24),
```

```
color='#FF9999',  label=('25 to 48'), edgecolor='white', width=barWidth)
ax4 = plt.bar(x_labels, t_49_60, bottom=np.array(t_0_12) + np.array(t_13_24)
+ np.array(t_25_48), color='#FFA07A', label=('48 to 60'), edgecolor='white',
width=barWidth)
ax5 = plt.bar(x_labels, t_gt_60, bottom=np.array(t_0_12) + np.array(t_13_24)
+ np.array(t_25_48) + np.array(t_49_60), color='#F0E68C', label=('> 60'),
edgecolor='white', width=barWidth)

plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by Tenure Group')

for r1, r2, r3, r4, r5 in zip(ax1, ax2, ax3, ax4, ax5):
    h1 = r1.get_height()
    h2 = r2.get_height()
    h3 = r3.get_height()
    h4 = r4.get_height()
    h5 = r5.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r3.get_x() + r3.get_width() / 2., h1 + h2 + h3 / 2.,
'{:.2%}'.format(h3), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r4.get_x() + r4.get_width() / 2., h1 + h2 + h3 + h4 / 2.,
'{:.2%}'.format(h4), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r5.get_x() + r5.get_width() / 2., h1 + h2 + h3 + h4 + h5 / 2.,
'{:.2%}'.format(h5), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



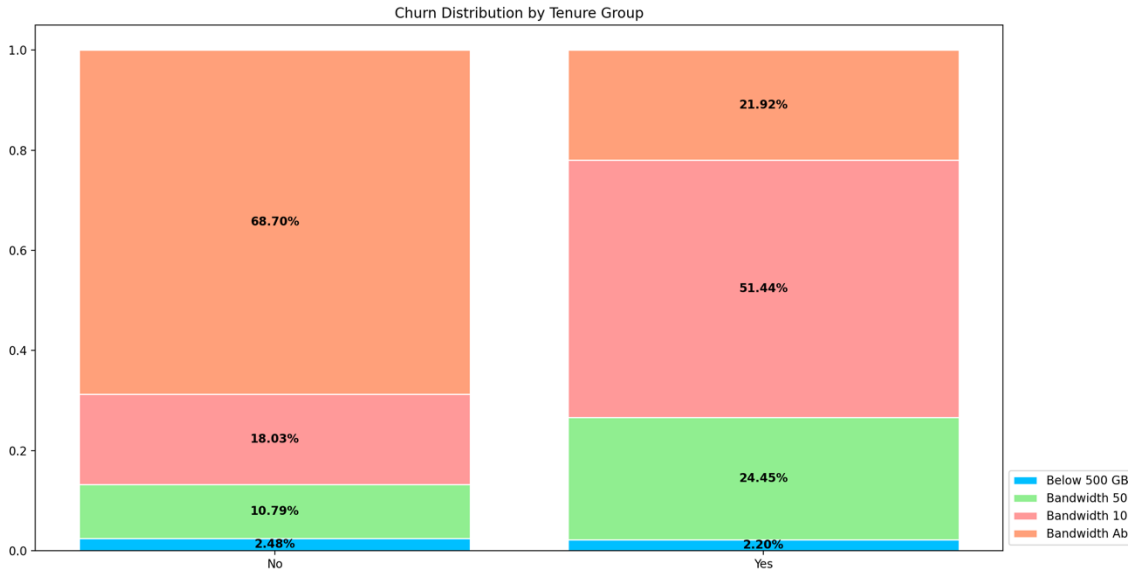
Picture 17: Tenure and Churn Rate

The graph above tells us that customers who have been a short period of time with the company (below 12 months) tend to churn more.

Bandwidth and Churn Rate:

```
#Bandwidth and churn
no_churn =
((df_clean_4[df_clean_4['Churn']=='No']['Bandwidth_GB_Year'].value_counts())
/(df_clean_4[df_clean_4['Churn']=='No']['Bandwidth_GB_Year'].value_counts().s
um()))
yes_churn =
((df_clean_4[df_clean_4['Churn']=='Yes']['Bandwidth_GB_Year'].value_counts())
/(df_clean_4[df_clean_4['Churn']=='Yes']['Bandwidth_GB_Year'].value_counts().
sum()))

# Getting values from the group and categories
x_labels = churn_df['Churn'].value_counts().keys().tolist()
b_500 = [no_churn['Bandwidth Below 500'], yes_churn['Bandwidth Below 500']]
b_500_1000 = [no_churn['Bandwidth 500 - 1000'], yes_churn['Bandwidth 500 -
1000']]
b_1000_2000 = [no_churn['Bandwidth 1000 - 2000'], yes_churn['Bandwidth 1000 -
2000']]
b_gt_2000 = [no_churn['Bandwidth Above 2000'], yes_churn['Bandwidth Above
2000']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, b_500, color='#00BFFF', label=('Below 500 GB'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, b_500_1000, bottom=b_500, color='lightgreen',
label=('Band 500-1000'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, b_1000_2000, bottom=np.array(b_500) +
np.array(b_500_1000), color='#FF9999',  label=('Band 1000-2000'),
edgecolor='white', width=barWidth)
ax4 = plt.bar(x_labels, b_gt_2000, bottom=np.array(b_500) +
np.array(b_500_1000) + np.array(b_1000_2000), color='#FFA07A', label=('Above
2000'), edgecolor='white', width=barWidth)

plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by Bandwidth_GB_Year')

for r1, r2, r3, r4 in zip(ax1, ax2, ax3, ax4):
    h1 = r1.get_height()
    h2 = r2.get_height()
    h3 = r3.get_height()
    h4 = r4.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r3.get_x() + r3.get_width() / 2., h1 + h2 + h3 / 2.,
'{:.2%}'.format(h3), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r4.get_x() + r4.get_width() / 2., h1 + h2 + h3 + h4 / 2.,
```

```
'{:.2%}'.format(h4), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



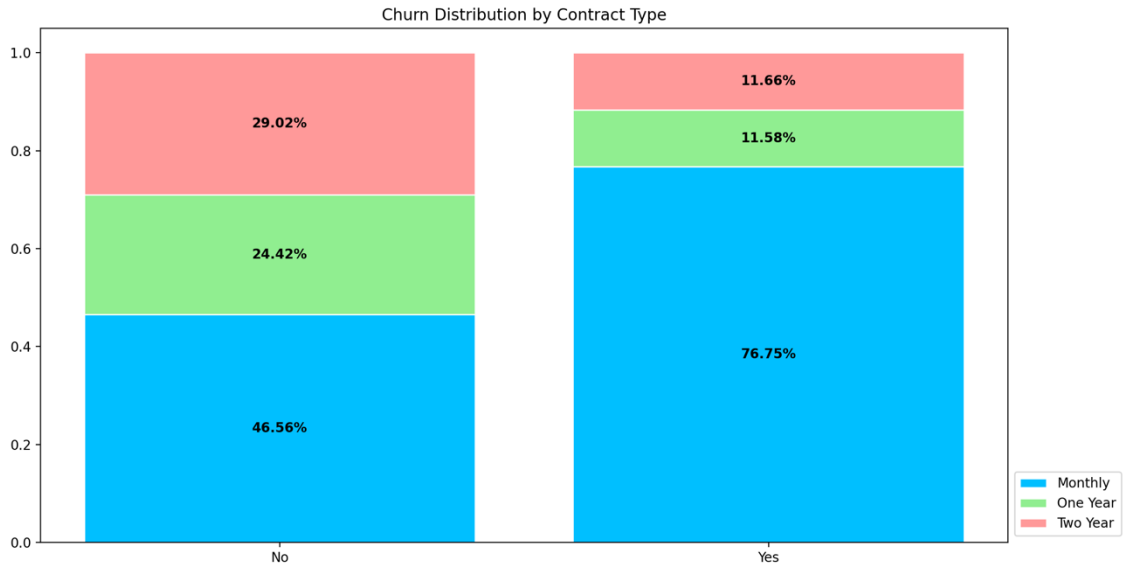
Picture 18: Bandwidth and Churn Rate

The previous graph shows that customers who signed up for having 1000-2000 GB used per year churn more.

Contract and Churn Rate:

```
# #Contract Type and Churn Rate
no_churn = ((churn_df[churn_df['Churn']=='No']['Contract'].value_counts())
/(churn_df[churn_df['Churn']=='No']['Contract'].value_counts().sum()))
yes_churn = ((churn_df[churn_df['Churn']=='Yes']['Contract'].value_counts())
/(churn_df[churn_df['Churn']=='Yes']['Contract'].value_counts().sum()))

# Getting values from the group and categories
x_labels = churn_df['Churn'].value_counts().keys().tolist()
monthly = [no_churn['Month-to-month'], yes_churn['Month-to-month']]
one_year = [no_churn['One year'], yes_churn['One year']]
two_year = [no_churn['Two Year'], yes_churn['Two Year']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, monthly, color='#00BFFF', label=('Monthly'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, one_year, bottom=monthly, color='lightgreen',
label=('One Year'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, two_year, bottom=np.array(one_year) +
np.array(monthly), color='#FF9999', label=('Two Year'), edgecolor='white',
width=barWidth)
```

```
plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by Contract Type')

for r1, r2, r3 in zip(ax1, ax2, ax3):
    h1 = r1.get_height()
    h2 = r2.get_height()
    h3 = r3.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r3.get_x() + r3.get_width() / 2., h1 + h2 + h3 / 2.,
'{:.2%}'.format(h3), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



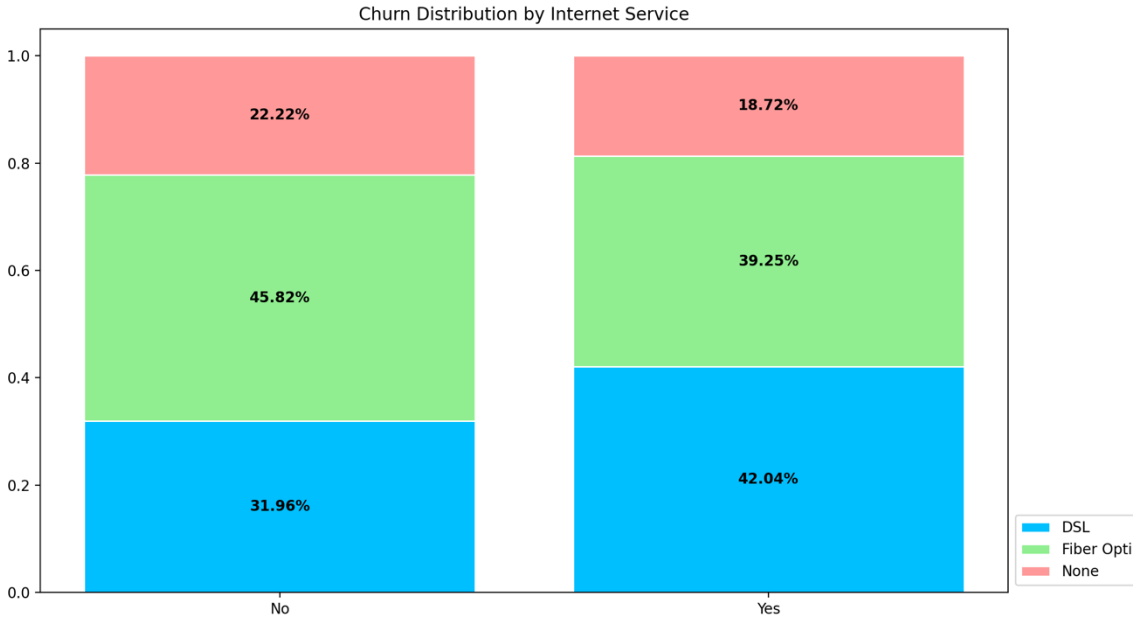
Picture 19: Contract and Churn Rate

We can clearly see that customers on a month-to-month contract churn more than other customers.

Internet Service and Churn Rate:

```
#Internet Service and Churn Rate
no_churn =
((churn_df[churn_df['Churn']=='No']['InternetService'].value_counts())
/(churn_df[churn_df['Churn']=='No']['InternetService'].value_counts().sum()))
yes_churn =
((churn_df[churn_df['Churn']=='Yes']['InternetService'].value_counts())
/(churn_df[churn_df['Churn']=='Yes']['InternetService'].value_counts().sum())
)

# Getting values from the group and categories
x_labels = churn_df['Churn'].value_counts().keys().tolist()
```

```
dsl = [no_churn['DSL'], yes_churn['DSL']]
fiber_optic = [no_churn['Fiber Optic'], yes_churn['Fiber Optic']]
none = [no_churn['None'], yes_churn['None']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, dsl, color='#00BFFF', label=('DSL'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, fiber_optic, bottom=dsl, color='lightgreen',
label=('Fiber Optic'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, none, bottom=np.array(fiber_optic) + np.array(dsl),
color='#FF9999', label=('None'), edgecolor='white', width=barWidth)

plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by Internet Service')

for r1, r2, r3 in zip(ax1, ax2, ax3):
    h1 = r1.get_height()
    h2 = r2.get_height()
    h3 = r3.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
    plt.text(r3.get_x() + r3.get_width() / 2., h1 + h2 + h3 / 2.,
'{:.2%}'.format(h3), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



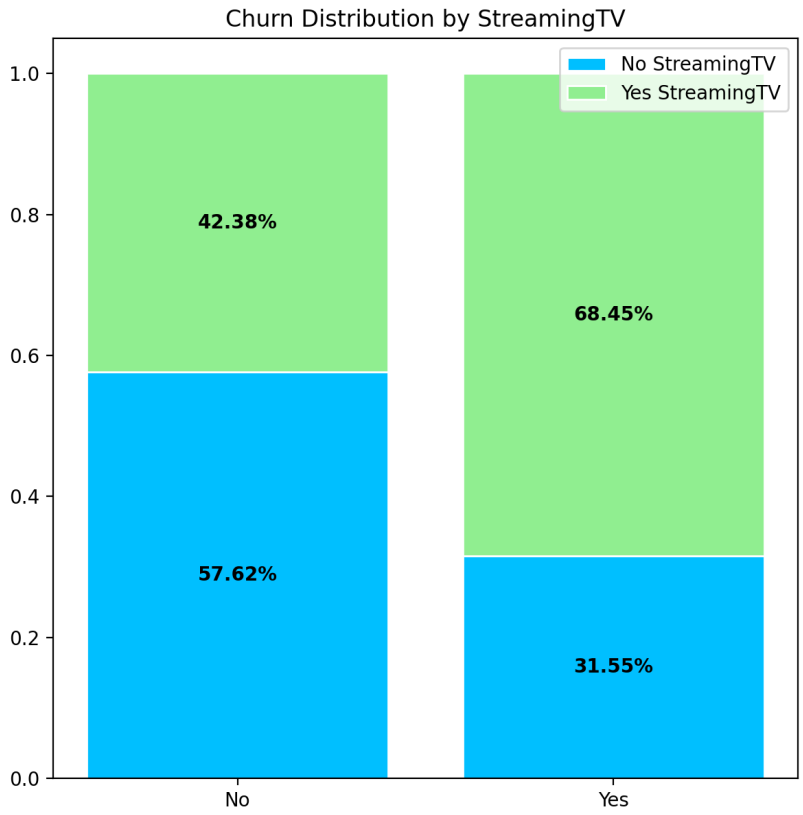
Picture 20: Internet Service and Churn Rate

The internet and churn graph tells us that customers with DSL service churn more than other customers.

Streaming TV and Churn Rate:

```
# #StreamingTV and Churn Rate
no_churn = ((churn_df[churn_df['Churn']=='No']['StreamingTV'].value_counts())
/(churn_df[churn_df['Churn']=='No']['StreamingTV'].value_counts().sum()))
yes_churn =
((churn_df[churn_df['Churn']=='Yes']['StreamingTV'].value_counts())
/(churn_df[churn_df['Churn']=='Yes']['StreamingTV'].value_counts().sum()))

# Getting values from the group and categories
x_labels = churn_df['Churn'].value_counts().keys().tolist()
yes_st = [no_churn['Yes'], yes_churn['Yes']]
no_st = [no_churn['No'], yes_churn['No']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, yes_st, color='#00BFFF', label=('Yes StreamingTV'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, no_st, bottom=yes_st, color='lightgreen', label=('No
StreamingTV'), edgecolor='white', width=barWidth)

plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by StreamingTV')

for r1, r2 in zip(ax1, ax2):
```

```
    h1 = r1.get_height()
    h2 = r2.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



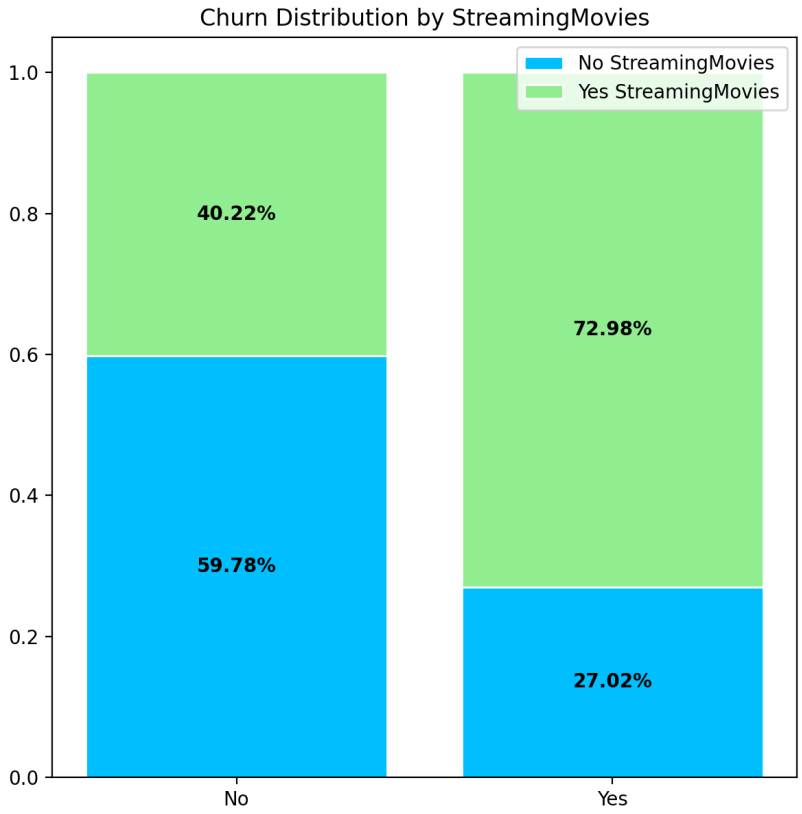
Picture 21: StreamingTV and Churn Rate

Now we can see that customers with streaming tv service churn more.

Streaming Movies and Churn Rate:

```
#StreamingMovies and Churn Rate
no_churn =
((churn_df[churn_df['Churn']=='No']['StreamingMovies'].value_counts())
/(churn_df[churn_df['Churn']=='No']['StreamingMovies'].value_counts().sum()))
yes_churn =
((churn_df[churn_df['Churn']=='Yes']['StreamingMovies'].value_counts())
/(churn_df[churn_df['Churn']=='Yes']['StreamingMovies'].value_counts().sum())
)

# Getting values from the group and categories
```

```
x_labels = churn_df['Churn'].value_counts().keys().tolist()
yes_sm = [no_churn['Yes'], yes_churn['Yes']]
no_sm = [no_churn['No'], yes_churn['No']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, yes_sm, color='#00BFFF', label=('Yes
StreamingMovies'), edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, no_sm, bottom=yes_sm, color='lightgreen', label=('No
StreamingMovies'), edgecolor='white', width=barWidth)

plt.legend(loc='lower left', bbox_to_anchor=(1,0))
plt.title('Churn Distribution by StreamingMovies')

for r1, r2 in zip(ax1, ax2):
    h1 = r1.get_height()
    h2 = r2.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



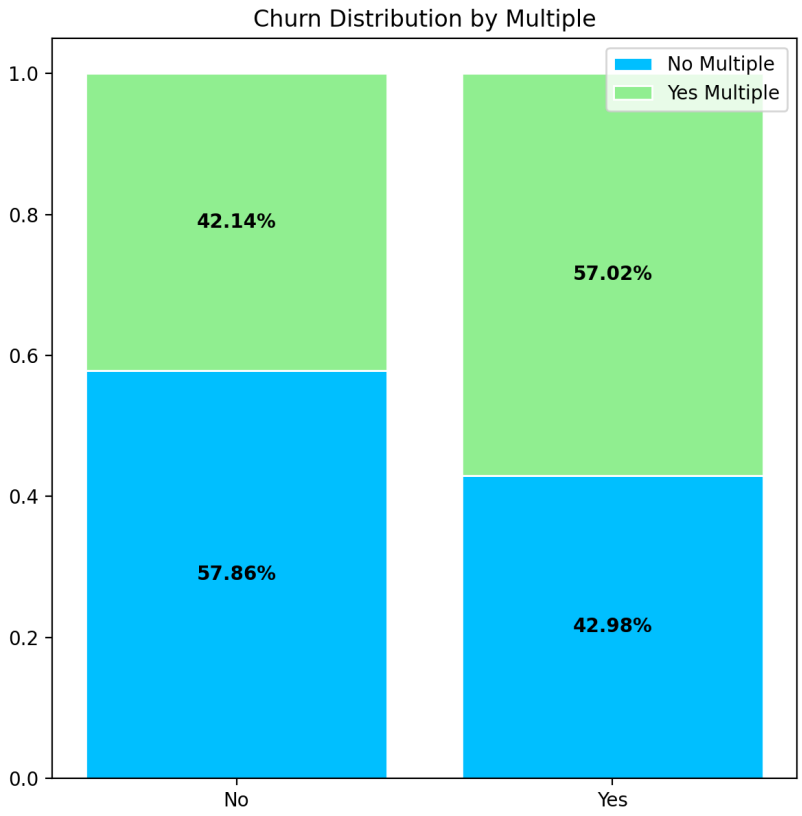
Picture 22: StreamingMovies and Churn Rate

Now we can see that customers with streaming movies service churn more.

Multiple and Churn Rate:

```
# Getting values from the group and categories
#Churn is my X axis
x_labels = churn_df['Churn'].value_counts().keys().tolist()
#Y axis will be my other variables
n_var = [no_churn['No'], yes_churn['No']]
y_var = [no_churn['Yes'], yes_churn['Yes']]

# Multiple and Churn
barWidth = 0.8
plt.figure(figsize=(7, 7))
ax1 = plt.bar(x_labels, n_var, color='#00BFFF', label=('No Multiple'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, y_var, bottom=n_var, color='lightgreen', label=('Yes
Multiple'), edgecolor='white', width=barWidth)
plt.legend()
plt.title('Churn Distribution by Multiple')
```

```
for r1, r2 in zip(ax1, ax2):
    h1 = r1.get_height()
    h2 = r2.get_height()
    plt.text(r1.get_x() + r1.get_width() / 2., h1 / 2., '{:.2%}'.format(h1),
ha='center', va='center', color='black', fontweight='bold')
    plt.text(r2.get_x() + r2.get_width() / 2., h1 + h2 / 2.,
'{:.2%}'.format(h2), ha='center', va='center', color='black',
fontweight='bold')
plt.show()
```



Picture 23: Multiple and Churn Rate

Customers who have signed up for multiple lines service also churn more.

PART C5 – PROVIDE A COPY OF THE PREPARED DATASET

The prepared data set will be attached as “prepared_churn_data.csv”.

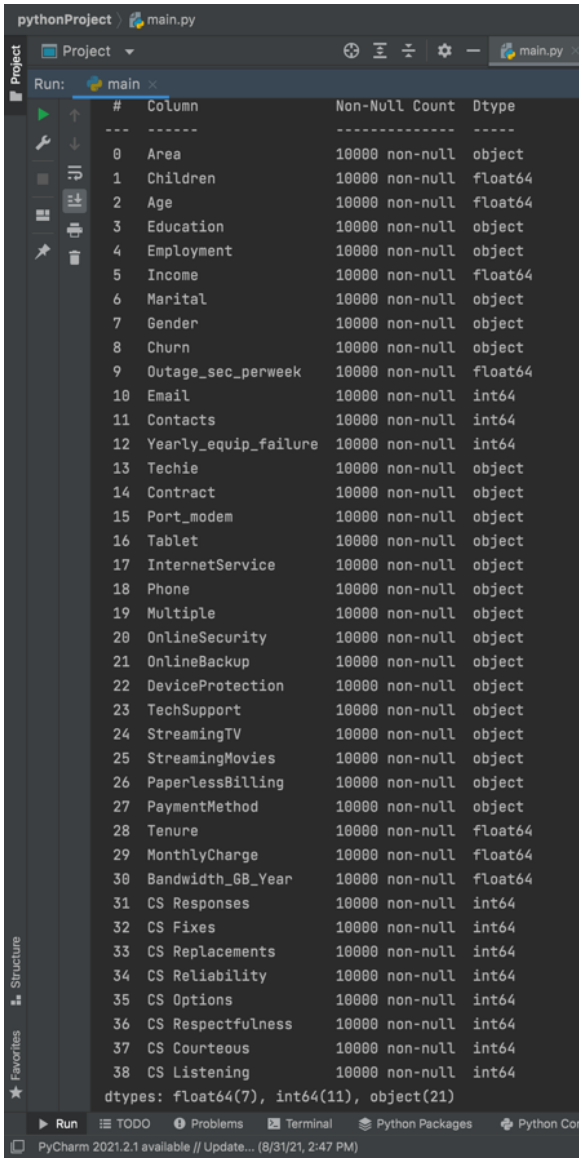
PART IV - MODEL COMPARISON AND ANALYSIS

PART D – COMPARE AN INITIAL AND A REDUCED LOGISTIC REGRESSION MODEL BY THE DOING THE FOLLOWING:

PART D1. Construct an initial logistic regression model from *all* predictors that were identified in Part C2.

We can formulate this customer churn prediction as a logistic regression task. This technique is used to estimate the relationship between a dependent variable (churn) and independent variables that might predict the churn.

```
#Additional info about the remaining data:
data_types = churn_df2.info()
print(data_types)
```



Picture 24: Types of data

Checking if the continuous variables present outliers:

```
# Checking for outliers in the continuous variables
numerical_churn_df2 = churn_df2[['Tenure','MonthlyCharge',
'Bandwidth_GB_Year']]


# Checking for outliers at 25%, 50%, 75%, 90%, 95% and 99%
print(numerical_churn_df2.describe(percentiles=[.25, .5, .75, .90, .95,
.99]))
```

```
              Tenure        Tenure  ...  Bandwidth_GB_Year  Bandwidth_GB_Year
count  10000.000000  10000.000000  ...       10000.000000       10000.000000
mean      34.640500     34.640500  ...        3397.122700        3397.122700
std       25.188194     25.188194  ...        2072.726654        2072.726654
min        1.000000      1.000000  ...         156.000000         156.000000
25%        9.000000      9.000000  ...        1312.000000        1312.000000
50%       36.000000     36.000000  ...        3382.000000        3382.000000
75%       60.000000     60.000000  ...        5466.000000        5466.000000
90%       67.000000     67.000000  ...        6094.100000        6094.100000
95%       70.000000     70.000000  ...        6343.050000        6343.050000
99%       72.000000     72.000000  ...        6717.010000        6717.010000
max       72.000000     72.000000  ...        7159.000000        7159.000000

[11 rows x 6 columns]
```

Picture 25: Outliers check for continuous variables

These variables do not have outliers since they get higher with the percentiles.

Logistic Regression Model:

```
#LOGISTIC REGRESSION CODE
#Y variable is the target: Churn
y = churn_df2.Churn.values
#Removing Churn from remaining data
X = churn_df2.drop('Churn', axis = 1)
#Saving column titles to a list
cols = X.columns

#Train - Test - Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3,
random_state = 33)

#Num Feature Scalling
scaler = StandardScaler()
X_train[['Tenure','MonthlyCharge','Bandwidth_GB_Year']] =
scaler.fit_transform(X_train[['Tenure','MonthlyCharge','Bandwidth_GB_Year']])
X_test[['Tenure','MonthlyCharge','Bandwidth_GB_Year']] =
scaler.fit_transform(X_test[['Tenure','MonthlyCharge','Bandwidth_GB_Year']])
```

```
#Building the model, training it and creating predictions
model = LogisticRegression()

#Fitting the model to our X and y training sets
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

```
Performance Evaluation
#Find residual differences between train data and predicted train data
residuals_train = np.abs(y_train, y_pred_train)

#Print the number of times our model was correct ('0') and incorrect ('1')
model_count_train = pd.Series(residuals_train).value_counts()
print('Residual Training Data is ',model_count_train)

#Print normalized amount of times our model was correct (percentage)
model_count_train = pd.Series(residuals_train).value_counts(normalize = True)
print('Normalized Residual Training Data is ',model_count_train)
```

```
Residual Training Data is  0    5189
1    1811
dtype: int64
Normalized Residual Training Data is  0    0.741286
1    0.258714
dtype: float64
Residual Testing data is  0    2161
1     839
dtype: int64
Normalized Residual Testing data is  0    0.720333
1    0.279667
dtype: float64
```

Picture 26: Model built based on all dataset

From the results above we see that model has 74% accuracy.

Visualizing the model coefficients:

```
#Create lists
column_labels = cols.tolist()
coef = model.coef_.squeeze().tolist()

#Visualization of features and coef together as requested per evaluator
```

```
labels_coef = list(zip(column_labels, coef))

# Verify the result
print(labels_coef)
```

Picture 27: Logistic Regression Coefficients

These are the output with feature names and coefficients as requested:

[('Unnamed: 0', -0.0003240798620417291), ('Children', -0.04240207633459349), ('Age', -0.0004921655113257889), ('Income', 3.222891357912277e-06), ('Outage_sec_perweek', -0.15868061801624675), ('Email', -0.018711002968459282), ('Contacts', 0.02449417720569007), ('Yearly_equip_failure', -0.03163504251803353), ('Techie', 0.10770514946631739), ('Port_modem', 0.03956139784436012), ('Tablet', 0.019915135504599783), ('Phone', -0.008406709352333697), ('Multiple', 0.3214120742842189), ('OnlineSecurity', -0.03455027665144141), ('OnlineBackup', 0.1600123664474774), ('DeviceProtection', 0.11258879729278393), ('TechSupport', 0.03046650986760705), ('StreamingTV', 0.5584984619419376), ('StreamingMovies', 0.6815028272215595), ('PaperlessBilling', 0.03261154017881548), ('Tenure', -0.703361848932789), ('MonthlyCharge', 1.7206408920147533), ('Bandwidth_GB_Year', -0.4910741882180901), ('CS Responses', -0.022705771676829533), ('CS Fixes', 0.015304689165337388), ('CS Replacements', -0.022198597393459637), ('CS Reliability', -0.00648125297528355), ('CS Options', 0.05829178838015728), ('CS Respectfulness', -0.0029124327333827768), ('CS Courteous', -0.03475555537050152), ('CS Listening', 0.030570840406970463), ('Marital_Divorced', -0.0026774228490833714), ('Marital_Married', -0.016534761525023933), ('Marital_Never Married', -0.03596571113481555), ('Marital_Separated', 0.056814154473559655), ('Marital_Widowed', 0.015864236014941295), ('Contract_Month-to-month', 0.6488447453770555), ('Contract_One year', -0.28753703779385625), ('Contract_Two Year', -0.34380721257291863), ('PaymentMethod_Bank Transfer(automatic)', -0.05511146820744128), ('PaymentMethod_Credit Card (automatic)', 0.0012666761248624355), ('PaymentMethod_Electronic Check', 0.07661857117803456), ('PaymentMethod_Mailed Check', -0.005273284113986753), ('Gender_Female', -0.028056820892499404), ('Gender_Male', 0.053592543879634746), ('Gender_Prefer not to answer', -0.008035228007641346), ('InternetService_DSL', 0.22327296014599898), ('InternetService_Fiber Optic', -0.15511915373939267), ('InternetService_None', -0.05065331142215622), ('Area_Rural', 0.016276004668895894), ('Area_Suburban', -0.028821465748243243), ('Area_Urban', 0.03004595605952443), ('Employment_Full Time', 0.011427128438470517), ('Employment_Part Time', -0.007004699481682707), ('Employment_Retired', -0.007158140163882258), ('Employment_Student', 0.027255366848456728), ('Employment_Unemployed', -0.007019160659866237)]

Model Intercept is: [**0.0175035**]

Calculating the Confusion Matrix:

```
#Confusion Matrix
confusion_matrix = confusion_matrix(y_test, y_pred_test)
print('Confusion Matrix: \n', confusion_matrix)
```

```
dtype: float64
Confusion Matrix:
 [[1974  187]
 [ 321  518]]
```

Picture 28: Confusion Matrix Output

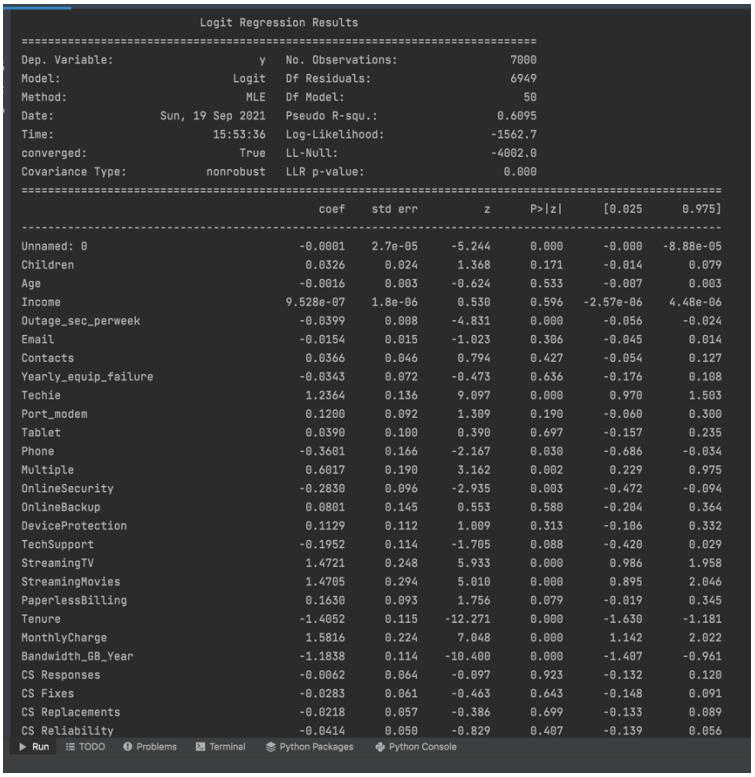
As we can see with the calculated confusion matrix here, my model was right 1974 (TN) times predicting that the customer was not going to churn, and also right 518 (TP) times predicting that the customer was going to churn. 321 (FP) times my model predicted that a customer was going to churn but it didn't happen and 187 (FN) times my model predicted that a customer was not going to churn but they did.

Logistic Regression Equation I: all data set

Y = 0.0175035 -0.04240207633459349* Children -0.0004921655113257889* Age + 3.222891357912277e-06*Income -0.15868061801624675* Outage_sec_perweek -0.018711002968459282* Email + 0.02449417720569007* Contacts -0.03163504251803353* Yearly_equip_failure + 0.10770514946631739* Techie + 0.03956139784436012* Port_modem + 0.019915135504599783*Tablet -0.008406709352333697* Phone + 0.3214120742842189* Multiple -0.03455027665144141* OnlineSecurity + 0.1600123664474774* OnlineBackup + 0.11258879729278393* DeviceProtection + 0.03046650986760705* TechSupport + 0.5584984619419376* StreamingTV + 0.6815028272215595* StreamingMovies', + 0.03261154017881548* PaperlessBilling -0.703361848932789* Tenure + 1.7206408920147533* MonthlyCharge -0.4910741882180901*Bandwidth_GB_Year - 0.022705771676829533* CS Responses + 0.015304689165337388* CS Fixes - 0.022198597393459637* CS Replacements -0.00648125297528355* CS Reliability + 0.05829178838015728* CS Options -0.0029124327333827768* CS Respectfulness + 0.03475555537050152* CS Courteous + 0.030570840406970463* CS Listening - 0.0026774228490833714* Marital_Divorced -0.016534761525023933* Marital_Married - 0.03596571113481555* Marital_Never Married + 0.056814154473559655* Marital_Separated + 0.015864236014941295* Marital_Widowed + 0.6488447453770555* Contract_Month-to-month -0.28753703779385625*Contract_One year -0.34380721257291863* Contract_Two Year -0.05511146820744128* PaymentMethod_Bank Transfer(automatic) +

0.0012666761248624355* PaymentMethod_Credit Card (automatic) + 0.07661857117803456* PaymentMethod_Electronic Check -0.005273284113986753* PaymentMethod_Mailed Check - 0.028056820892499404* Gender_Female + 0.053592543879634746* Gender_Male - 0.008035228007641346* Gender_Prefer not to answer + 0.22327296014599898* InternetService_DSL -0.15511915373939267* InternetService_Fiber Optic - 0.05065331142215622* InternetService_None + 0.016276004668895894* Area_Rural - 0.028821465748243243* Area_Suburban + 0.03004595605952443* Area_Urban + 0.011427128438470517* Employment_Full Time -0.007004699481682707* Employment_Part Time -0.007158140163882258* Employment_Retired + 0.027255366848456728* Employment_Student -0.00701916065986623 7*Employment_Unemployed

With this model described above we have a **74% accuracy.**



Logit Regression Results

Dep. Variable:	y	No. Observations:	7000
Model:	Logit	Df Residuals:	6949
Method:	MLE	Df Model:	50
Date:	Sun, 19 Sep 2021	Pseudo R-squ.:	0.6095
Time:	15:53:36	Log-Likelihood:	-1562.7
converged:	True	LL-Null:	-4002.0
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
Unnamed: 0	-0.0001	2.7e-05	-5.244	0.000	-0.000	-8.88e-05
Children	0.0326	0.024	1.368	0.171	-0.014	0.079
Age	-0.0016	0.003	-0.624	0.533	-0.007	0.003
Income	9.528e-07	1.8e-06	0.530	0.596	-2.57e-06	4.48e-06
Outage_sec_perweek	-0.0399	0.008	-4.831	0.000	-0.056	-0.024
Email	-0.0154	0.015	-1.023	0.306	-0.045	0.014
Contacts	0.0366	0.046	0.794	0.427	-0.054	0.127
Yearly_equip_failure	-0.0343	0.072	-0.473	0.636	-0.176	0.108
Techie	1.2364	0.136	9.097	0.000	0.970	1.503
Port_modem	0.1200	0.092	1.309	0.190	-0.060	0.300
Tablet	0.0390	0.100	0.390	0.697	-0.157	0.235
Phone	-0.3601	0.166	-2.167	0.030	-0.686	-0.034
Multiple	0.6017	0.190	3.162	0.002	0.229	0.975
OnlineSecurity	-0.2830	0.096	-2.935	0.003	-0.472	-0.094
OnlineBackup	0.0801	0.145	0.553	0.580	-0.204	0.364
DeviceProtection	0.1129	0.112	1.009	0.313	-0.106	0.332
TechSupport	-0.1952	0.114	-1.705	0.088	-0.420	0.029
StreamingTV	1.4721	0.248	5.933	0.000	0.986	1.958
StreamingMovies	1.4705	0.294	5.010	0.000	0.895	2.046
PaperlessBilling	0.1630	0.093	1.756	0.079	-0.019	0.345
Tenure	-1.4052	0.115	-12.271	0.000	-1.630	-1.181
MonthlyCharge	1.5816	0.224	7.048	0.000	1.142	2.022
Bandwidth_GB_Year	-1.1838	0.114	-10.400	0.000	-1.407	-0.961
CS Responses	-0.0062	0.064	-0.097	0.923	-0.132	0.120
CS Fixes	-0.0283	0.061	-0.463	0.643	-0.148	0.091
CS Replacements	-0.0218	0.057	-0.386	0.699	-0.133	0.089
CS Reliability	-0.0414	0.050	-0.829	0.407	-0.139	0.056

Run TODO Problems Terminal Python Packages Python Console

Picture 29: All Variables Log Reg Model Results Part I

PaperlessBilling	0.1630	0.093	1.756	0.079	-0.019	0.345
Tenure	-1.4052	0.115	-12.271	0.000	-1.630	-1.181
MonthlyCharge	1.5816	0.224	7.048	0.000	1.142	2.022
Bandwidth_GB_Year	-1.1838	0.114	-10.400	0.000	-1.407	-0.961
CS Responses	-0.0062	0.064	-0.097	0.923	-0.132	0.120
CS Fixes	-0.0283	0.061	-0.463	0.643	-0.148	0.091
CS Replacements	-0.0218	0.057	-0.386	0.699	-0.133	0.089
CS Reliability	-0.0414	0.050	-0.829	0.407	-0.139	0.056
CS Options	-0.0103	0.053	-0.194	0.846	-0.114	0.094
CS Respectfulness	0.0195	0.053	0.367	0.714	-0.085	0.124
CS Courteous	0.0345	0.051	0.672	0.502	-0.066	0.135
CS Listening	-0.0112	0.047	-0.237	0.813	-0.104	0.081
Marital_Divorced	-0.6807	2.8e+06	-2.43e-07	1.000	-5.49e+06	5.49e+06
Marital_Married	-0.7135	2.02e+06	-3.52e-07	1.000	-3.97e+06	3.97e+06
Marital_Never Married	-0.7331	2.73e+06	-2.69e-07	1.000	-5.35e+06	5.35e+06
Marital_Separated	-0.5375	2.32e+06	-2.32e-07	1.000	-4.54e+06	4.54e+06
Marital_Widowed	-0.4482	2.63e+06	-1.71e-07	1.000	-5.15e+06	5.15e+06
Contract_Month-to-month	1.5981	1.7e+06	9.41e-07	1.000	-3.33e+06	3.33e+06
Contract_One year	-1.7661	1.77e+06	-9.96e-07	1.000	-3.47e+06	3.47e+06
Contract_Two Year	-1.7655	1.68e+06	-1.05e-06	1.000	-3.29e+06	3.29e+06
PaymentMethod_Bank Transfer(automatic)	-1.2679	nan	nan	nan	nan	nan
PaymentMethod_Credit Card (automatic)	-0.9592	nan	nan	nan	nan	nan
PaymentMethod_Electronic Check	-0.5596	nan	nan	nan	nan	nan
PaymentMethod_Mailed Check	-0.8534	nan	nan	nan	nan	nan
Gender_Female	-0.0258	2.45e+06	-1.05e-08	1.000	-4.81e+06	4.81e+06
Gender_Male	0.2382	3.13e+06	7.61e-08	1.000	-6.14e+06	6.14e+06
Gender_Prefer not to answer	-0.1181	2.97e+06	-3.97e-08	1.000	-5.83e+06	5.83e+06
InternetService_DSL	0.4569	nan	nan	nan	nan	nan
InternetService_Fiber Optic	-1.7585	nan	nan	nan	nan	nan
InternetService_None	-0.6199	nan	nan	nan	nan	nan
Area_Rural	-0.6339	1.72e+06	-3.69e-07	1.000	-3.37e+06	3.37e+06
Area_Suburban	-0.7601	1.7e+06	-4.48e-07	1.000	-3.33e+06	3.33e+06
Area_Urban	-0.5276	1.74e+06	-3.03e-07	1.000	-3.41e+06	3.41e+06
Employment_Full Time	-0.4698	nan	nan	nan	nan	nan
Employment_Part Time	-0.4250	nan	nan	nan	nan	nan
Employment_Retired	-0.3889	nan	nan	nan	nan	nan
Employment_Student	-0.2614	nan	nan	nan	nan	nan
Employment_Unemployed	-0.3765	nan	nan	nan	nan	nan

Picture 30: All Variables Log Reg Model Results Part II

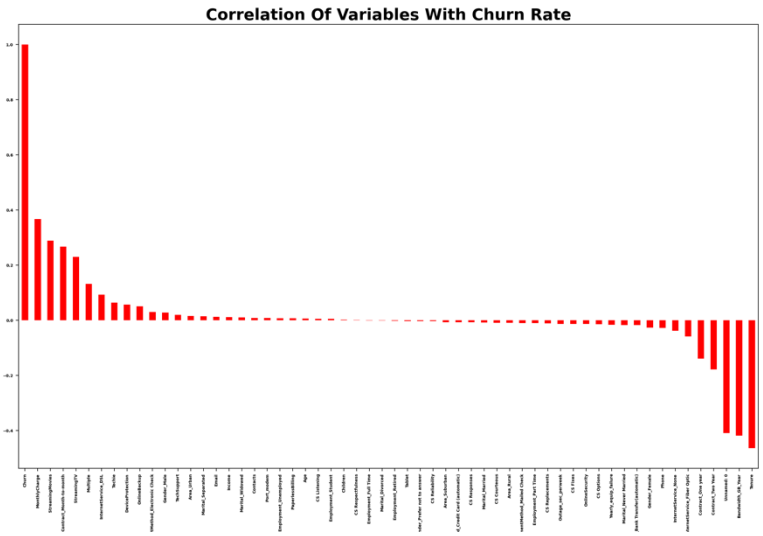
```
1    0.279667
dtype: float64
Confusion Matrix:
 [[1974  187]
 [ 321  518]]
Accuracy is  0.8306666666666667
Sensitivity is  0.6174016686531585
Specificity is  0.9134659879685331
Precision is  0.7347517730496453
F1 Score is  0.6709844559585492

Process finished with exit code 0
```

Picture 31: Performance Metrics of the initial model

PART D2. Justify a statistically based variable selection procedure and a model evaluation metric to reduce the initial model in a way that aligns with the research question.

Lets analyze a correlation matrix with all variables and try to drop a few with high correlation to each other:

```
#Plotting features and Churn correlations in descending order
churn_df2.corr()['Churn'].sort_values(ascending = False).plot(kind ='bar',
figsize = (10, 5), color = 'Red')
plt.title('Correlation Of Variables With Churn Rate', fontsize = 20,
fontweight = 'bold')
plt.xticks(fontsize = 5, fontweight = 'bold')
plt.yticks(fontweight = 'bold', fontsize = 5)
plt.show()
```



Picture 32: Correlation Plot with Churn Rate

From the correlation plot above we can notice that the variables MonthlyCharge, StreamingMovies, Contract Month-to-month, StreamingTV, Multiple, InternetService_DSL, Techie, DeviceProtection, Contract_One Year, Contract_Two Year, Bandwidth_GB_Year and Tenure have a bigger correlation with churn.

From the results of the first model we analyze all p-values and dropped everything that was over 0.05. Analyzing the initial model's p-values, we keep these variables: Outage_sec_perweek, Techie, Phone, Multiple, OnlineSecurity, StreamingTV, StreamingMovies, Tenure, MonthlyCharge, Bandwidth.

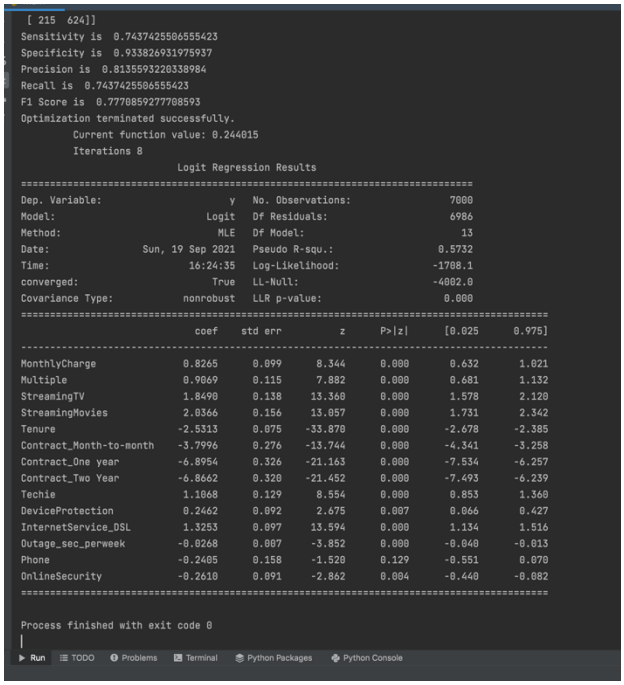
From these two analysis, my reduced final model will have: Outage_sec_perweek, Techie, Phone, Multiple, OnlineSecurity, StreamingTV, StreamingMovies, Tenure, MonthlyCharge, Contract Month-to-month, InternetService_DSL, DeviceProtection, Contract_One Year, Contract_Two Year

As presented in the first correlation matrix plotted for this report, Bandwidth and Tenure are highly correlated and I chose to drop bandwidth. Also the 8 survey columns are all correlated and I am dropping them as well.

```
#From the heatmap we can drop bandwidth and also all costumer survey answers
# #Dropping Bandwidth due to its highly correlation with Tenure
churn_df2 = churn_df2.drop(columns=['Bandwidth_GB_Year'])
# #Dropping Customer Survey Answers
churn_df2 = churn_df2.drop(columns=['CS Responses', 'CS Fixes', 'CS
Replacements', 'CS Reliability', 'CS Options',
                                    'CS Respectfulness', 'CS Courteous', 'CS
Listening'])

churn_reduced_analysis = churn_df2[['MonthlyCharge','Multiple',
'StreamingTV', 'StreamingMovies', 'Tenure',
                                    'Contract_Month-to-month', 'Churn',
                                    'Contract_One year', 'Contract_Two Year',
'Techie', 'DeviceProtection',
                                    'InternetService_DSL']]

churn_df2 = churn_reduced_analysis
```

```
#LOGISTIC REGRESSION CODE
#Y variable is the target: Churn
y = churn_df2.Churn.values
#Removing Churn from remaining data
X = churn_df2.drop('Churn', axis = 1)
#X = churn_reduced_analysis

#Saving column titles to a list
cols = X.columns

# print('Columns of CHURN_DF2 are: ',cols)


#Train - Test - Split: 70%-30%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3,
random_state = 33)

#Num Feature Scalling _except the Bandwidth this time
scaler = StandardScaler()

#X_train[['Tenure','MonthlyCharge','Bandwidth_GB_Year']] =
scaler.fit_transform(X_train[['Tenure','MonthlyCharge','Bandwidth_GB_Year']])
#X_test[['Tenure','MonthlyCharge','Bandwidth_GB_Year']] =
scaler.fit_transform(X_test[['Tenure','MonthlyCharge','Bandwidth_GB_Year']])

X_train[['Tenure','MonthlyCharge']] =
scaler.fit_transform(X_train[['Tenure','MonthlyCharge']])
X_test[['Tenure','MonthlyCharge']] =
scaler.fit_transform(X_test[['Tenure','MonthlyCharge']])
```

```
# #Building the model, training it and creating predictions
model = LogisticRegression()

#Fitting the model to our X and y training sets
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

#Create lists
column_labels = cols.tolist()
coef = model.coef_.squeeze().tolist()

#Visualization of features and coef together as requested by evaluator
labels_coef = list(zip(column_labels, coef))

#Verify the result
print(labels_coef)

print('Model Intercept is: ' , model.intercept_)

#Performance Evaluation
#Find residual differences between train data and predicted train data
residuals_train = np.abs(y_train, y_pred_train)

#Print the number of times our model was correct ('0') and incorrect ('1')
model_count_train = pd.Series(residuals_train).value_counts()
print('Residual Training Data is ',model_count_train)

#Print normalized amount of times our model was correct (percentage)
model_count_train = pd.Series(residuals_train).value_counts(normalize = True)
print('Normalized Residual Training Data is ',model_count_train)

#Train Set Results: 5189 correct and 1811 Incorrect (74.1% Accuracy)

#Doing the same but for the test set
#Performance Evaluation
# Find residual differences between test data and predicted test data
residuals_test = np.abs(y_test, y_pred_test)

#Print the number of times our model was correct ('0') and incorrect ('1')
model_count_test = pd.Series(residuals_test).value_counts()
print('Residual Testing data is ', model_count_test)

# Print normalized amount of times our model was correct (percentage)
model_count_test = pd.Series(residuals_test).value_counts(normalize = True)
print('Normalized Residual Testing data is ', model_count_test)

#For the test set: 2161 correct and 839 Incorrect --> Accuracy of 72%

#Confusion Matrix
confusion_matrix = metrics.confusion_matrix(y_test, model.predict(X_test))
print('Confusion Matrix: \n', confusion_matrix)
```



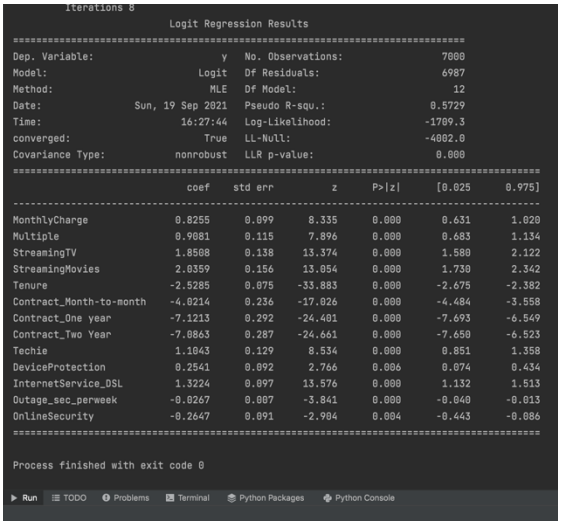
```
 [ 215  624]]
Sensitivity is  0.7437425506555423
Specificity is  0.933826931975937
Precision is  0.8135593220338984
Recall is  0.7437425506555423
F1 Score is  0.7770859277708593
Optimization terminated successfully.
         Current function value: 0.244015
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 7000
Model:                          Logit   Df Residuals:                     6986
Method:                           MLE   Df Model:                           13
Date:                Sun, 19 Sep 2021   Pseudo R-squ.:                 0.5732
Time:                        16:24:35   Log-Likelihood:                -1708.1
converged:                       True   LL-Null:                       -4002.0
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
MonthlyCharge                 0.8265      0.099      8.344      0.000       0.632       1.021
Multiple                      0.9069      0.115      7.882      0.000       0.681       1.132
StreamingTV                   1.8490      0.138     13.360      0.000       1.578       2.120
StreamingMovies               2.0366      0.156     13.057      0.000       1.731       2.342
Tenure                       -2.5313      0.075    -33.870      0.000      -2.678      -2.385
Contract_Month-to-month      -3.7996      0.276    -13.744      0.000      -4.341      -3.258
Contract_One year            -6.8954      0.326    -21.163      0.000      -7.534      -6.257
Contract_Two Year            -6.8662      0.320    -21.452      0.000      -7.493      -6.239
Techie                        1.1068      0.129      8.554      0.000       0.853       1.360
DeviceProtection              0.2462      0.092      2.675      0.007       0.066       0.427
InternetService_DSL           1.3253      0.097     13.594      0.000       1.134       1.516
Outage_sec_perweek           -0.0268      0.007     -3.852      0.000      -0.040      -0.013
Phone                        -0.2405      0.158     -1.520      0.129      -0.551       0.070
OnlineSecurity               -0.2610      0.091     -2.862      0.004      -0.440      -0.082
==============================================================================================

Process finished with exit code 0
```

Picture 33: Performance of the reduced model

We can see that the p-value for 'Phone' is high so we are going to drop this variable!

The only change is this:

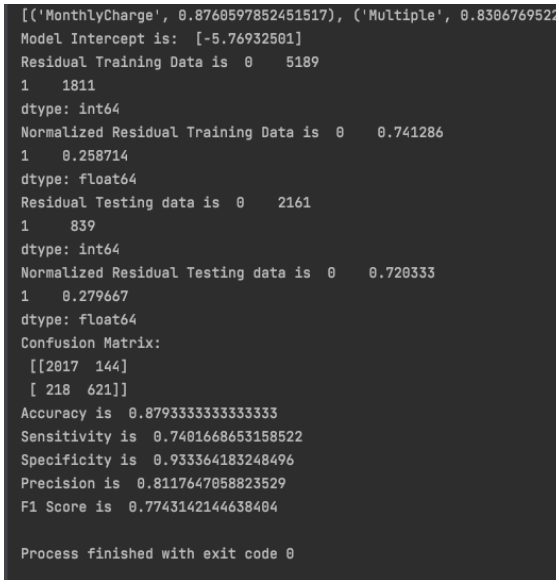
```
churn_reduced_analysis = churn_df2[['MonthlyCharge','Multiple',
'StreamingTV', 'StreamingMovies', 'Tenure',
                                    'Contract_Month-to-month', 'Churn',
                                    'Contract_One year', 'Contract_Two Year',
'Techie', 'DeviceProtection',

'InternetService_DSL','Outage_sec_perweek', 'OnlineSecurity']]
```



```
         Iterations 8
                          Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 7000
Model:                          Logit   Df Residuals:                     6987
Method:                           MLE   Df Model:                           12
Date:                Sun, 19 Sep 2021   Pseudo R-squ.:                 0.5729
Time:                        16:27:44   Log-Likelihood:                -1709.3
converged:                       True   LL-Null:                       -4002.0
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================================
                                 coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
MonthlyCharge                  0.8255      0.099      8.335      0.000       0.631       1.020
Multiple                       0.9081      0.115      7.896      0.000       0.683       1.134
StreamingTV                    1.8508      0.138     13.374      0.000       1.580       2.122
StreamingMovies                2.0359      0.156     13.054      0.000       1.730       2.342
Tenure                        -2.5285      0.075    -33.883      0.000      -2.675      -2.382
Contract_Month-to-month       -4.0214      0.236    -17.026      0.000      -4.484      -3.558
Contract_One year             -7.1213      0.292    -24.401      0.000      -7.693      -6.549
Contract_Two Year             -7.0863      0.287    -24.661      0.000      -7.650      -6.523
Techie                         1.1043      0.129      8.534      0.000       0.851       1.358
DeviceProtection               0.2541      0.092      2.766      0.006       0.074       0.434
InternetService_DSL            1.3224      0.097     13.576      0.000       1.132       1.513
Outage_sec_perweek            -0.0267      0.007     -3.841      0.000      -0.040      -0.013
OnlineSecurity                -0.2647      0.091     -2.904      0.004      -0.443      -0.086
==============================================================================================

Process finished with exit code 0
```

Picture 34: Model Performance

We can see that these set of variables work much better!



```
[('MonthlyCharge', 0.8760597852451517), ('Multiple', 0.830676952
Model Intercept is:  [-5.76932501]
Residual Training Data is  0    5189
1    1811
dtype: int64
Normalized Residual Training Data is  0    0.741286
1    0.258714
dtype: float64
Residual Testing data is  0    2161
1     839
dtype: int64
Normalized Residual Testing data is  0    0.720333
1    0.279667
dtype: float64
Confusion Matrix:
 [[2017  144]
 [ 218  621]]
Accuracy is  0.8793333333333333
Sensitivity is  0.7401668653158522
Specificity is  0.933364183248496
Precision is  0.8117647058823529
F1 Score is  0.7743142144638404

Process finished with exit code 0
```

Picture 35: Reduced Model Log Reg

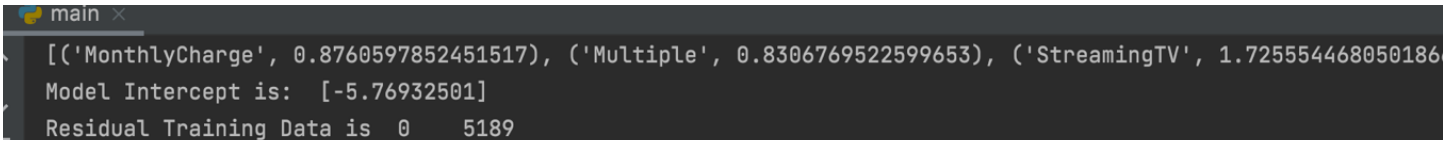
This confusion matrix shows that the reduced model was right in predicting 2017 (TN) times that customers would not churn, and 621 (TP) times that they would. 218 (FP) times the model was wrong, predicting customers were going to churn but they didn't and 144 (FN) times the model said customers were not going to churn but they did.

PART D3. Provide a reduced multiple regression model that includes *both* categorical and continuous variables.

The output of the reduced model presented above is:

[('MonthlyCharge', 0.8760597852451517), ('Multiple', 0.8306769522599653), ('StreamingTV', 1.7255544680501866), ('StreamingMovies', 1.902052729660154), ('Tenure', -2.4754806297623526), ('Contract_Month-to-month', 1.9487646815398803), ('Contract_One year', -1.050491217863562), ('Contract_Two Year', -1.019858255215286), ('Techie', 1.0472422567866986), ('DeviceProtection', 0.22700780092858222), ('InternetService_DSL', 1.311543203512211), ('Outage_sec_perweek', -0.027415836594221678), ('OnlineSecurity', -0.27180211642215435)]

Model Intercept is: [-5.76932501]



Picture 36: Logistic Regression Model with reduced variables

The reduced set of variables logistic regression equation is:

Y = -5.76932501+ 0.8760597852451517* MonthlyCharge + 0.8306769522599653* Multiple + 1.7255544680501866* StreamingTV + 1.902052729660154* StreamingMovies - 2.4754806297623526* Tenure + 1.9487646815398803* Contract_Month-to-month - 1.050491217863562* Contract_One year -1.019858255215286* Contract_Two Year + 1.0472422567866986* Techie + 0.22700780092858222* DeviceProtection + 1.311543203512211* InternetService_DSL -0.027415836594221678*Outage_sec_perweek - 0.27180211642215435*OnlineSecurity

This model accuracy is also 74% using less variables. We pretty much got the same results with way less variables!

PART E – ANALYZE THE DATASET USING YOUR REDUCED LOGISTIC REGRESSION MODEL BY DOING THE FOLLOWING:

PART E1. EXPLAIN YOUR DATA ANALYSIS PROCESS BY COMPARING THE INITIAL AND REDUCED LOGISTIC REGRESSION MODELS, INCLUDING THE FOLLOWING ELEMENTS:

- **the logic of the variable selection technique**

To select the reduced set of variables I analyzed two things: first it was the "Correlation Plot With Churn Rate" presented in this document (Picture 32) and also the results of the initial model, more specifically the p-values of each feature, presented in Pictures 29 and 30 of this report. The correlation plot presented features and their correlation significance with churn.

The closer this correlation number is to 1, the stronger the correlation. After analyzing the plot, I decided that the variables **MonthlyCharge, StreamingMovies, Contract Month-to-month, StreamingTV, Multiple, InternetService_DSL, Techie, DeviceProtection, Contract_One Year, Contract_Two Year, Bandwidth_GB_Year and Tenure** have a strong correlation with churn and because of this factor I included them in the reduced model.

The other analysis is the p-values of all features from the initial model. All p-values above 0.05 were discarded. For that reason, I kept these features: **Outage_sec_perweek, Techie, Phone, Multiple, OnlineSecurity, StreamingTV, StreamingMovies, Tenure, MonthlyCharge and Bandwidth_GB_Year**.

Since we also found out using the correlation matrix presented previously in this report that Tenure and Bandwidth have a high correlation, I dropped Bandwidth.

Combining the analysis shown above, my reduced model has the following features:

After analyzing again the p-values of these features in a reduced model, I noticed that the p-value of the variable Phone was high so I dropped this variables and created another model. The final set of variables is:

Outage_sec_perweek,
Techie,
Multiple,
OnlineSecurity,
StreamingTV,
StreamingMovies,
Tenure,
MonthlyCharge,
Contract Month-to-month,
InternetService_DSL,
DeviceProtection,
Contract_One Year,
Contract_Two Year.

- **the model evaluation metric**

The metrics to evaluate a logistic regression model are:

-) Precision
-) Accuracy
-) Sensitivity or Recall
-) Specificity
-) F1 Score

The metrics for the initial model are presented in Picture 31:

Precision: 0.734
Accuracy: 0.83
Sensitivity: 0.617
Specificity: 0.913
F1 Score: 0.671

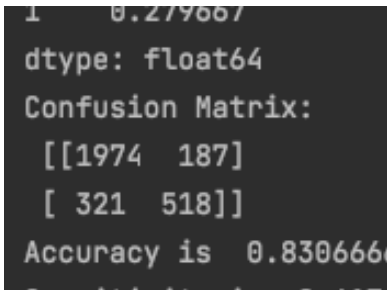
The metrics for the final reduced model are presented in Picture 35:

Precision: 0.812
Accuracy: 0.879
Sensitivity: 0.74
Specificity: 0.933
F1 Score: 0.774

We can clearly see that all metrics for the reduced model are better than the initial model!

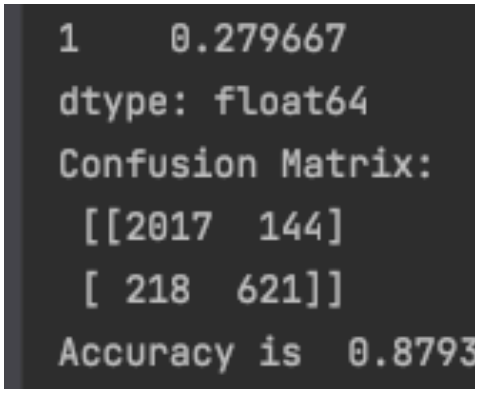
PART E2. PROVIDE THE OUTPUT AND ANY CALCULATIONS OF THE ANALYSIS YOU PERFORMED, INCLUDING A CONFUSION MATRIX:

Confusion Matrix Initial Model:



Picture 37: Confusion Matrix - Initial Model

Confusion Matrix Reduced Model:



Picture 38: Confusion Matrix - Final Reduced Model

We can noticed that we have a higher number of true negatives and true negatives in the reduced model's confusion matrix than in the initial model.

PART E3. PROVIDE THE CODE USED TO SUPPORT THE IMPLEMENTATION OF THE LOGISTIC REGRESSION MODELS: included along this report

Part V: DATA SUMMARY AND IMPLICATIONS

PART F. SUMMARIZE YOUR FINDINGS AND ASSUMPTIONS BY DOING THE FOLLOWING:

PART F1. DISCUSS THE RESULTS OF YOUR DATA ANALYSIS, INCLUDING THE FOLLOWING ELEMENTS:

- a regression equation for the reduced model:

Y = -5.76932501+ 0.8760597852451517* MonthlyCharge + 0.8306769522599653* Multiple + 1.7255544680501866* StreamingTV + 1.902052729660154* StreamingMovies - 2.4754806297623526* Tenure + 1.9487646815398803* Contract_Month-to-month - 1.050491217863562* Contract_One year -1.019858255215286* Contract_Two Year + 1.0472422567866986* Techie + 0.22700780092858222* DeviceProtection + 1.311543203512211* InternetService_DSL -0.027415836594221678*Outage_sec_perweek - 0.27180211642215435*OnlineSecurity

- an interpretation of coefficients of the statistically significant variables of the model: for every 1 unit of

MonthlyCharge: customer churn will increase **0.8760597852451517 units**

Multiple: customer churn will increase **0.8306769522599653 units**

StreamingTV: customer churn will increase **1.7255544680501866 units**

StreamingMovies: customer churn will increase **1.902052729660154 units**

Tenure: customer churn will decrease **2.4754806297623526 units**

Contract Month to Month: customer churn will increase **1.9487646815398803 units**

Contract_One year: customer churn will decrease **1.050491217863562 units**

Contract_Two year: customer churn will decrease **1.019858255215286 units**

Techie: customer churn will increase **1.0472422567866986units**

DeviceProtection**:** customer churn will increase **0.22700780092858222 units**

InternetService_DSL: customer churn will increase **1.311543203512211 units**
Outage_sec_perweek**:** customer churn will decrease **0.027415836594221678 units**
OnlineSecurity**:** customer churn will decrease **0.27180211642215435 units**

- the statistical and practical significance of the model

p-values: are statistically significant at 0 for all variables, as shown in **Picture 34**.

```
#Obtaining P-Values
import statsmodels.api as sm
logit_model=sm.Logit(y_train,X_train)
result=logit_model.fit()
print(result.summary())
```

Metrics of the model: [3]

```
TP = confusion_matrix[1,1]
TN = confusion_matrix[0,0]
FP = confusion_matrix[0,1]
FN = confusion_matrix[1,0]

#Accuracy:
accuracy = (TN + TP)/(TN+TP+FN+FP)
print('Accuracy is ', accuracy)

#Sensitivity
sensitivity = TP / float(TP+FN)
print('Sensitivity is ', sensitivity)

#Specifity
specificity = TN / float(TN+FP)
print('Specificity is ', specificity)
```

```
#Precision
precision = TP / float(TP + FP)
print('Precision is ', precision)

#F1 Score
f1score = 2*precision*sensitivity / float(precision+sensitivity)
print('F1 Score is ', f1score)
```

```
[('MonthlyCharge', 0.8760597852451517), ('Multiple', 0.6
Model Intercept is:  [-5.76932501]
Residual Training Data is  0    5189
1    1811
dtype: int64
Normalized Residual Training Data is  0    0.741286
1    0.258714
dtype: float64
Residual Testing data is  0    2161
1     839
dtype: int64
Normalized Residual Testing data is  0    0.720333
1    0.279667
dtype: float64
Confusion Matrix:
 [[2017  144]
 [ 218  621]]
Accuracy is  0.8793333333333333
Sensitivity is  0.7401668653158522
Specificity is  0.933364183248496
Precision is  0.8117647058823529
F1 Score is  0.7743142144638404

Process finished with exit code 0
```

Picture 39: Performance Metrics of the Reduced Model

- the limitations of the data analysis:

As mentioned before the churn rate is smaller compared with customers who haven't churned. Maybe a bigger ration in between those who churn and those who don't, would provide a more accurate model.

PART F2. RECOMMEND A COURSE OF ACTION BASED ON YOUR RESULTS

Based on the results we obtained with the logistic regression equation, the course of action would be to sign up customers using one or two year contracts (high negative coefficients). DSL internet might be unsatisfactory for customers with multiple lines or streaming services.

PART VI – DEMONSTRATION

PART G – VIDEO

The Panopto video will be uploaded here:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=28f6482c-1a5a-4b2b-a3c2-adaa016008dd

PART H – THIRD PARTY CODE
N/A

PART I – SOURCES

[1] (2011, Oct 10th) STOLTZFUS, JILL C Logistic Regression: a brief primer https://pubmed.ncbi.nlm.nih.gov/21996075/

[2] 2021, July BISCHOFF, BIANCA D206 Assessment Report

[3] 2020, Sept 13th, JAYASWAL, Vaibhav Performance Metrics: Confusion Matrix, Precision, Recall and F1 Score https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262