# D206 – Objective Assessment

## PART A – RESEARCH QUESTION

**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 main goal of this analysis is to determine which variables have a stronger relationship with the customer churn rate.

## PART B – REQUIRED VARIABLES

**Required Variables for Analysis**: The dataset contains 50 variables with 10,000 customers. The dependent variable is labeled as "**Churn**". This is a boolean type variable with values of YES (if the customer cancelled services within the last month) or NO (customer is still active with the company). The data has both numerical (number of children, customer's age, etc) and categorical (Churn is either Y/N, Payment Method and Type of Contract for example) attributes.

I am going to analyze all independent variables provided in the dataset and see which ones would make sense trying to predict churn rate. The variables that can be used for that intent are as follow:

### CUSTOMER DEMOGRAPHICS:

- **City:** Customer city of residence as listed on the billing statement
- **State:** Customer state of residence as listed on the billing statement
- **County:** Customer county of residence as listed on the billing statement
- **Zip:** Customer zip code of residence as listed on the billing statement
- **Lat, Lng:** GPS coordinates of customer residence as listed on the billing statement
- **Population:** Population within a mile radius of customer, based on census data
- **Area:** Area type (rural, urban, suburban), based on census data
- **TimeZone:** Time zone of customer residence based on customer's sign-up information
- **Job:** Job of the customer (or invoiced person) as reported in sign-up information
- **Children:** Number of children in customer's household as reported in sign-upinformation
- **Age:** Age of customer as reported in sign-up information
- **Education**: Highest degree earned by customer as reported in sign-up information
- **Employment**: Employment status of customer as reported in sign-up information
- **Income**: Annual income of customer as reported at time of sign-up

- **Marital:** Marital status of customer as reported in sign-up information
- **Gender:** Customer self-identification as male, female, or nonbinary

From this set of customer demographics, we know that "CaseOrder", "Interaction", "City", "state", "county", "zip", "lat", 'lng" and "Population" will not help us predict the churn rate. So we can start excluding these variables from the dataset.

- **Outage_sec_perweek:** Average number of seconds per week of system outages in the customer's neighborhood
- **Email:** Number of emails sent to the customer in the last year (marketing or correspondence)
- **Contacts:** Number of times customer contacted technical support
- **Yearly_equip_failure:** The number of times customer's equipment failed and had to be reset/replaced in the past year
- **Techie**: Whether the customer considers themselves technically inclined (based on customer questionnaire when they signed up for services) (yes, no)
- **Contract:** The contract term of the customer (month-to-month, one year, two year)
- **Port_modem:** Whether the customer has a portable modem (yes, no)
- **Tablet:** Whether the customer owns a tablet such as iPad, Surface, etc. (yes, no)
- **InternetService:** Customer's internet service provider (DSL, fiber optic, None)
- **Tenure**: Number of months the customer has stayed with the provider
- **Phone:** Whether the customer has a phone service (yes, no)
- **Multiple:** Whether the customer has multiple lines (yes, no)
- **OnlineSecurity:** Whether the customer has an online security add-on (yes, no)
- **OnlineBackup:** Whether the customer has an online backup add-on (yes, no)
- **DeviceProtection:** Whether the customer has device protection add-on (yes, no)
- **TechSupport:** Whether the customer has a technical support add-on (yes, no)
- **StreamingTV:** Whether the customer has streaming TV (yes, no)
- **StreamingMovies:** Whether the customer has streaming movies (yes, no)
- **PaperlessBilling:** Whether the customer has paperless billing (yes, no)
- **PaymentMethod:** The customer's payment method (electronic check, mailed check, bank (automatic bank transfer), credit card (automatic)
- **MonthlyCharge:** The amount charged to the customer monthly. This value reflects an average per customer.
- **Bandwidth_GB_Year:** The average amount of data used in GB, in a year by the customer

The next 8 columns come from a customer survey where the importance of key factors are rated on a on a scale of 1 to 8 (1 = most important, 8 = least important)

**Item1:** Timely response

**Item2:** Timely fixes
**Item3:** Timely replacements
**Item4:** Reliability
**Item5:** Options
**Item6:** Respectful response
**Item7:** Courteous exchange
**Item8:** Evidence of active listening

Data extracted from real processes are untidy, inconsistent, full of errors and sometimes even with missing values. So the first thing we need to work on is to clean the data. For that I choose to work with PyCharm (PyCharm is an integrated development environment used in computer programming, specifically for the Python language) [1]. The first step to my analysis will be preprocessing the data. The golden rule in predictive analysis is "Your model is only as good as your data". It is extremely important to treat the data and all the issues before any other step.

**PART C1 – PLAN TO FIND ANOMALIES**

**FIRST STEP: Importing the dataset and generate basic summary statistics.** I will be using PyCharm and using Pandas' commands.
**SECOND STEP:** Analyze the data for missing values, misspelled info
**THIRD STEP:** Fill in the missing data with the median of each variable
**FOURTH STEP:** Find outliers with the use of histograms to avoid analytical challenges
**FIFTH STEP:** Use PCA to reduce the number of features

**PART C2 - JUSTIFICATION:** This particular dataset has some missing values. Because of this I followed the steps of cleaning data as in the video "Python Statistics Essential Training". I have no coding background and since I have as a goal to learn both R and Python, I have decided to invest my time during D206 to learn Python and its libraries such as:

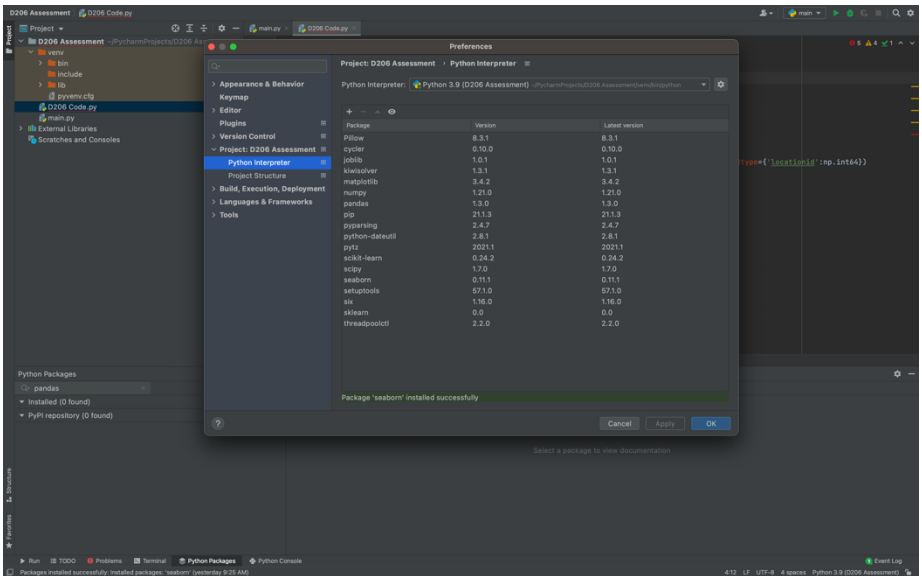
NUMPY: for mathematical operations
PANDAS: acts as a wrapper over Numpy and Matplotlib **libraries**, allowing you to access many of their methods with less code [2]
SCIKIT-LEARN: for PCA
MATPLOTLIB: data visualization
SCIPY: for linear algebra transformation

**PART C4 - CODE:**

**1-) To install the packages:** since I have decided to use PyCharm, one way to install the packages is:

PyCharm → Preferences → Project: D206 Assessment → Python Interpreter → +

I added the following packages: pandas, numpy, scipy, matplotlib and sklearn.




**Picture 1:** Screen Shot of Packages Installed in PyCharm

**2-) Import necessary packages:**

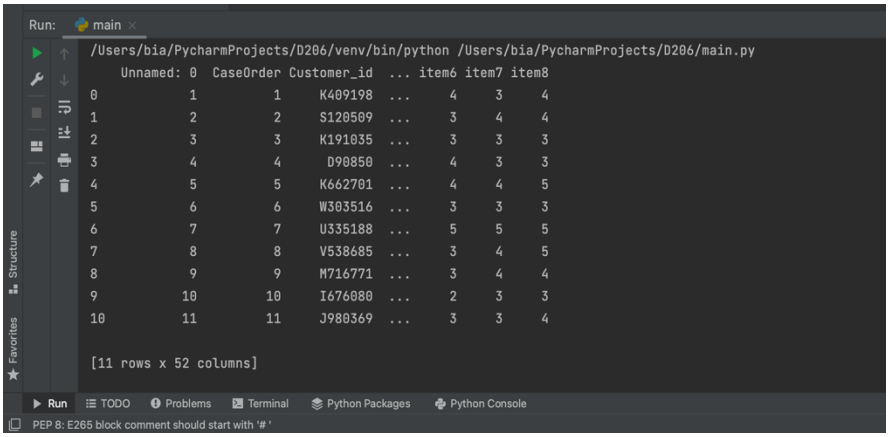
```
#Importing the packages:
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm, skew
from scipy import stats
# sklearn modules for data preprocessing:
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# sklearn modules for Model Selection:
from sklearn import svm, tree, linear_model, neighbors
from sklearn import naive_bayes, ensemble, discriminant_analysis,
gaussian_process
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# sklearn modules for Model Evaluation & Improvement:
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import f1_score, precision_score, recall_score,
fbeta_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import KFold
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
from sklearn.metrics import classification_report, precision_recall_curve
from sklearn.metrics import auc, roc_auc_score, roc_curve
from sklearn.metrics import make_scorer, recall_score, log_loss
from sklearn.metrics import average_precision_score
```

**3-) Importing the raw data:**

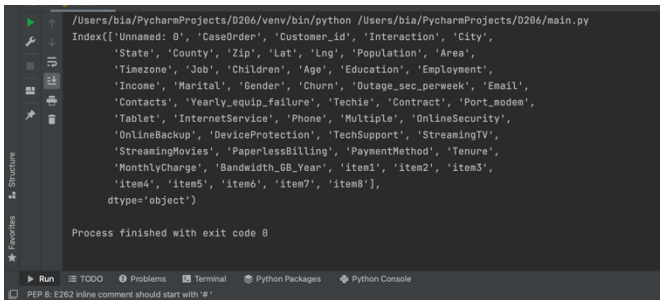
```
#Loading the Churn Dataset
churn_df=pd.read_csv('/Users/bia/Desktop/churn_raw_data.csv')
print(churn_df)
```

Since PyCharm doesn't allow the visualization of the entire data as it is, due to its limitation of 1000 rows x 15 columns, I edited the raw data spreadsheet to a smaller size just to make sure that my data was coming correctly:




**Picture 2:** Data Visualization

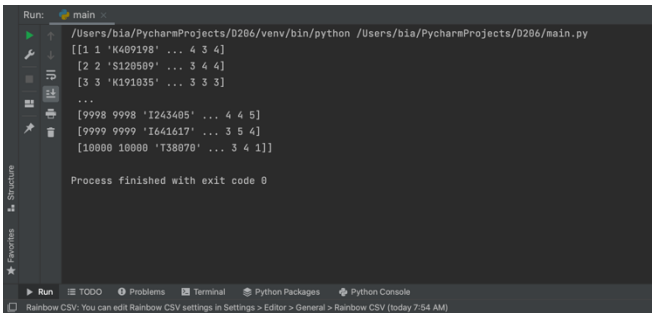
```
#Return the columns names
df=churn_df.columns
print(df)
```




**Picture 3**: Displaying the Name of All Columns

Data visualization as a numpy array: that is faster than doing it using Python lists since the library is densely packed in memory.

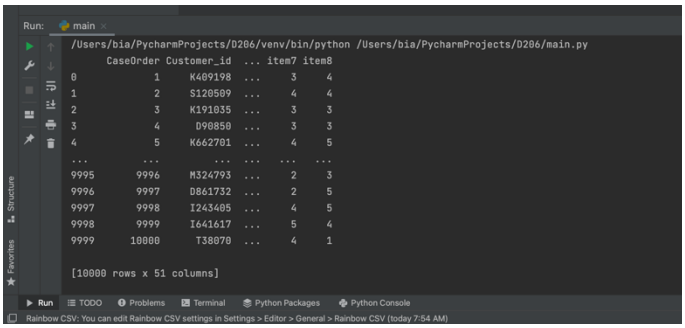
```
#Return all the data as an array
data=churn_df.values
print(data)
```




**Picture 4**: Displaying all the values as a numpy array

I am going to start removing data we won't need in this next step of the process. For example, the first column with only numbers and no description is irrelevant for our analysis. CaseOrder, Interaction, City, State, County, Zip, Lat, Lng and Population are also irrelevant.

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




**Picture 5**: Data without the first column

Removing 9 of the unnecessary columns:

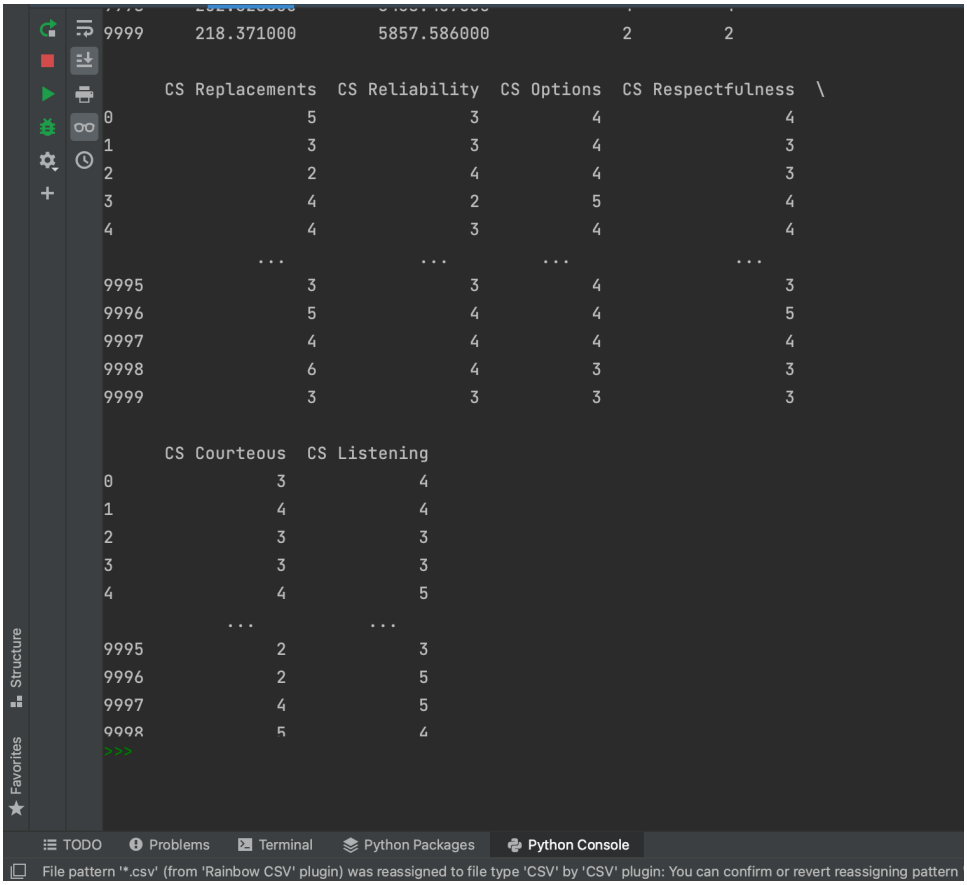
```
#Removing the 9 columns: CaseOrder, Interaction, City, Zip, Lat and Lng
df_clean = df.drop(columns=['CaseOrder', 'Interaction', 'City', 'State',
'County', 'Zip', 'Lat', 'Lng', 'Population'])
print(df_clean)
```

```
Python 3.9.6 (v3.9.6:db3ff76da1, Jun 28 2021, 11:49:53)
[Clang 6.0 (clang-600.0.57)] on darwin
>>> runfile('/Users/bia/PycharmProjects/D206/main.py', wdir='/Users/bia/PycharmProjects/D206')
Backend MacOSX is interactive backend. Turning interactive mode on.
     Customer_id      Area             Timezone  ... item6  item7  item8
0         K409198     Urban        America/Sitka  ...     4      3      4
1         S120509     Urban      America/Detroit  ...     3      4      4
2         K191035     Urban  America/Los_Angeles  ...     3      3      3
3          D90850  Suburban  America/Los_Angeles  ...     4      3      3
4         K662701  Suburban      America/Chicago  ...     4      4      5
           ...       ...                  ...  ...   ...    ...    ...
9995      M324793     Rural     America/New_York  ...     3      2      3
9996      D861732     Rural      America/Chicago  ...     5      2      5
9997      I243405     Rural      America/Chicago  ...     4      4      5
9998      I641617     Urban     America/New_York  ...     3      5      4
9999       T38070     Urban     America/New_York  ...     3      4      1

[10000 rows x 42 columns]
```

**Picture 6:** Data minus the 9 removed columns

The last 8 columns of the raw data file don't have a proper label, so I am going to add label to them in accordance with the data dictionary provided. All of them will have "CS" before since they come from Customer survey.

```
#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)
```




**Picture 7:** Adding a different label to better understand the independent variables

Calculating basic statistics of the data:

```
#Displaying Basic Statistics of the data
df_stats=df_clean.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 8:** Basic Statistics of the remaining dataset

```
#Displaying Basic Statistics of Tenure
df_stats_tenure = df_clean['Tenure'].describe()
print(df_stats_tenure)

#Displaying Basic Statistics of Monthly Charges
df_stats_charge = df_clean['MonthlyCharge'].describe()
print(df_stats_charge)
```

If we do a quick describe of “Tenure” and “MonthlyCharge” we see that customers stay at an average of **34 months** with this company paying an average of **$174** a month.

```
count    9069.000000
mean       34.498858
std        26.438904
min         1.000259
25%         7.890442
50%        36.196030
75%        61.426670
max        71.999280
Name: Tenure, dtype: float64
count    10000.000000
mean       174.076305
std         43.335473
min         77.505230
25%        141.071078
50%        169.915400
75%        203.777441
max        315.878600
Name: MonthlyCharge, dtype: float64
```

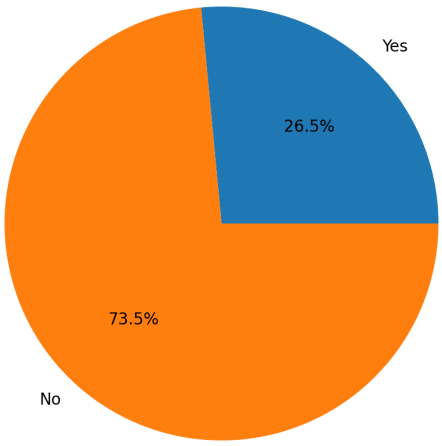
Calculating the churn rate:

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


**Picture 9:** Churn Rate

From the results above we can see that 73.5% of total customers didn't churn within the month period and 26.5% did. This represents a skew in the dataset. We need to keep that in mind for choosing the model. Machine learning works best when the number of instances in each class (churn and not churn) is similar. I also represented this result in the below pie chart. For that I used the MATPLOTLIB library.

```
#Creating a PieChart 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()
```



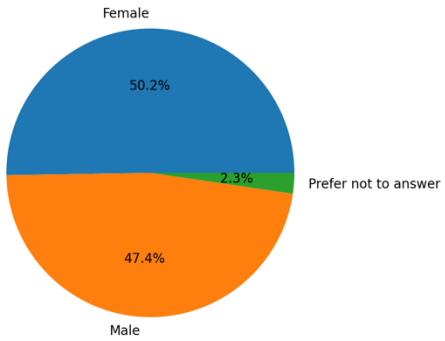

**Picture 10:** Pie Chart – Churn Rate

**Studying customer demographics:**

1. Gender

The first aspect of the customer that I am going to analyze is the gender ratio. I am going to calculate now what is our percentage of male, female and other clients. We have a very similar ratio of male and female.

```
#Gender Dist
print(df_clean['Gender'].unique())

male_count= df_clean['Gender'].value_counts()['Male']
female_count = df_clean['Gender'].value_counts()['Female']
none_count = df_clean['Gender'].value_counts()['Prefer not to answer']

gender_total = female_count + male_count + none_count

pct_female = (female_count / gender_total) * 100
pct_male = (male_count / gender_total) * 100
pct_none = (none_count / gender_total) * 100

#Pie Chart of Gender

plt.figure(figsize=(5,5))
labels = ["Female", "Male", "Prefer not to answer"]
values = [pct_female, pct_male, pct_none]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



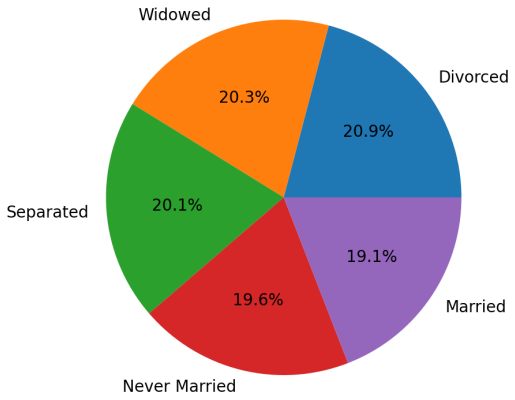

**Picture 11:** Gender Distribution

2. Marital Status:

To better understand the customer, I found the unique values for the marital data and created a pie chart to represent it.

```
#Marital Values
df_clean['Marital'].unique()
```

```
#Pie Charts for Marital Status
divorced_count= df_clean['Marital'].value_counts()['Divorced']
widowed_count = df_clean['Marital'].value_counts()['Widowed']
separated_count = df_clean['Marital'].value_counts()['Separated']
never_married_count = df_clean['Marital'].value_counts()['Never Married']
married_count = df_clean['Marital'].value_counts()['Married']

marital_total = divorced_count + widowed_count + separated_count +
never_married_count + married_count

pct_divorced = (divorced_count / marital_total) * 100
pct_widowed = (widowed_count / marital_total) * 100
pct_separated = (separated_count / marital_total) * 100
pct_never_married = (never_married_count / marital_total) * 100
pct_married = (married_count / marital_total) * 100

#Pie Chart of Marital Status
plt.figure(figsize=(5,5))
labels = ["Divorced", "Widowed", "Separated", "Never Married", "Married"]
values = [pct_divorced, pct_widowed, pct_separated, pct_never_married,
pct_married]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

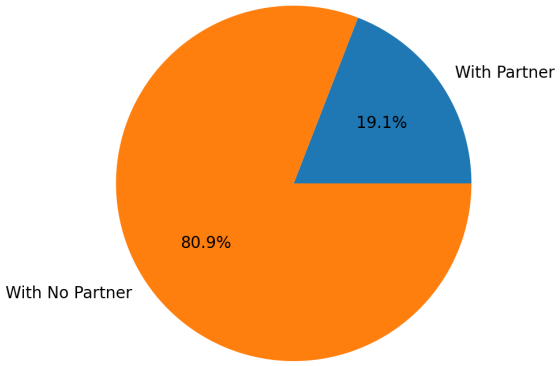



**Picture 12:** Marital Status Distribution

In the end, what really matters about the marital status will come to either “Customer Has a Partner” or “Customer Doesn’t Have a Partner”. Because of that I changed the chart a little bit to ease the final interpretation:

```
#Marital Status Calcs

divorced_count= df_clean['Marital'].value_counts()['Divorced']
widowed_count = df_clean['Marital'].value_counts()['Widowed']
```

```
separated_count = df_clean['Marital'].value_counts()['Separated']
never_married_count = df_clean['Marital'].value_counts()['Never Married']
married_count = df_clean['Marital'].value_counts()['Married']

marital_total = divorced_count + widowed_count + separated_count +
never_married_count + married_count

pct_divorced = (divorced_count / marital_total) * 100
pct_widowed = (widowed_count / marital_total) * 100
pct_separated = (separated_count / marital_total) * 100
pct_never_married = (never_married_count / marital_total) * 100
pct_married = (married_count / marital_total) * 100

pct_with_partner = pct_married

pct_no_partner = pct_divorced + pct_widowed + pct_separated +
pct_never_married


#Pie Chart of Marital Status

plt.figure(figsize=(5,5))
labels = ["With Partner", "With No Partner"]
values = [pct_with_partner, pct_no_partner]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```



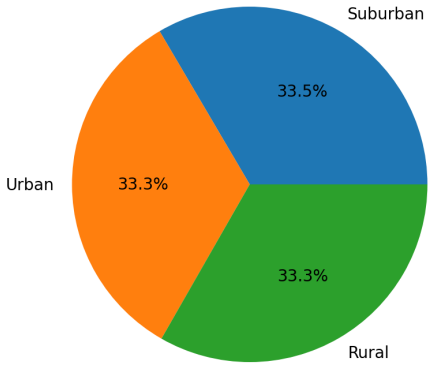

**Picture 13:** % of Customers with Partner and % With No Partners

3. Area:

```
#Area Values
df_clean['Area'].unique()
```

From this piece of code we see that customers either live in a suburban, urban or rural area. Since that might be a predictor of churn, I analyzed this data as well.

```
#Area Distribution
suburban_count= df_clean['Area'].value_counts()['Suburban']
urban_count = df_clean['Area'].value_counts()['Urban']
rural_count = df_clean['Area'].value_counts()['Rural']

area_total = suburban_count + urban_count + rural_count

pct_suburban = (suburban_count / area_total) * 100
pct_urban = (urban_count / area_total) * 100
pct_rural = (rural_count / area_total) * 100

#Pie Chart of Area Distribution
plt.figure(figsize=(5,5))
labels = ["Suburban", "Urban", "Rural"]
values = [pct_suburban, pct_urban, pct_rural]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

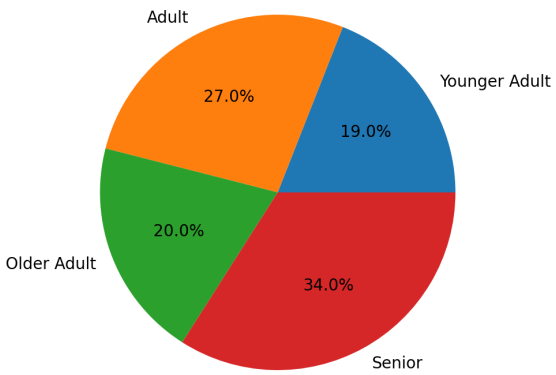



**Picture 14:** Area Distribution Pie Chart

4. Age Distribution:

```
#Customers age profile
df_clean['Age'].unique()
def age_intervals(df_clean):
    if df_clean['Age'] <= 30:
        return "Young Adult"
    elif (df_clean['Age'] > 31) & (df_clean['Age'] <= 50):
        return "Adult"
    elif (df_clean['Age'] > 51) & (df_clean['Age'] <= 65):
        return "Older Adult"
    elif (df_clean['Age'] > 65):
        return "Senior"
```

```
df_clean_4 = df_clean.copy()
df_clean_4['age_group'] = df_clean_4.apply(lambda
df_clean_4:age_intervals(df_clean_4), axis=1)
#df_clean_2['tenure_group']

age1_count= df_clean_4['age_group'].value_counts()['Young Adult']
age2_count = df_clean_4['age_group'].value_counts()['Adult']
age3_count= df_clean_4['age_group'].value_counts()['Older Adult']
age4_count= df_clean_4['age_group'].value_counts()['Senior']

age_total = age1_count + age2_count + age3_count + age4_count

pct_age1 = (age1_count / age_total) * 100
pct_age2 = (age2_count / age_total) * 100
pct_age3 = (age3_count / age_total) * 100
pct_age4 = (age4_count / age_total) * 100


#Pie Chart of Customers age
plt.figure(figsize=(5,5))
labels = ["Younger Adult", "Adult", "Older Adult", "Senior"]
values = [pct_age1, pct_age2, pct_age3, pct_age4]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```




**Picture 15:** Customer's age distribution

We also observe that we have some missing data in "Age".

```
#Unique values for Age Column

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

[68. 27. 50. 48. 83. nan 49. 86. 23. 56. 30. 39. 63. 60. 61. 52. 75. 77. 47. 70. 69. 45. 40. 82. 26. 25. 66. 72. 41. 44. 43. 84. 59. 31. 51. 58. 73. 33. 42. 81. 87. 54. 67. 46. 24. 20. 71. 32. 29. 80. 53. 79. 65. 35. 34. 74. 55. 76. 57. 38. 78. 19. 36. 88. 62. 37. 28. 22. 85. 89. 18. 21. 64.]

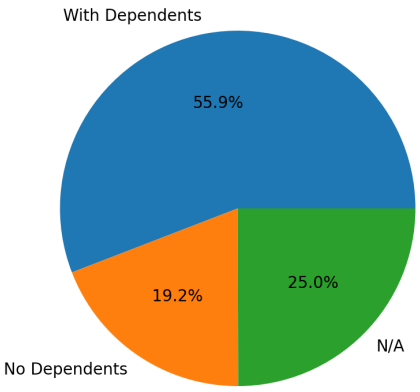
5. Dependents:

```
#Customers Dependents
def dep(df_clean):
    if df_clean['Children'] >= 1:
       return "Dependents"
    else:
        return "No Dependents"

df_clean_5 = df_clean.copy()
df_clean_5['dependents_group'] = df_clean_5.apply(lambda
df_clean_5:dep(df_clean_5), axis=1)

dep_count = df_clean_5['dependents_group'].value_counts()['Dependents']
no_dep_count = df_clean_5['dependents_group'].value_counts()['No Dependents']

dep_total = dep_count + no_dep_count

pct_dep = (dep_count / dep_total) * 100
pct_no_dep = (no_dep_count / dep_total) * 100

#Pie Chart of customers dependents
plt.figure(figsize=(5,5))
labels = ["With Dependents", "No Dependents"]
values = [pct_dep, pct_no_dep]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

We can observe some missing data for “Children” as well!

```
#Unique values for Children Column

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

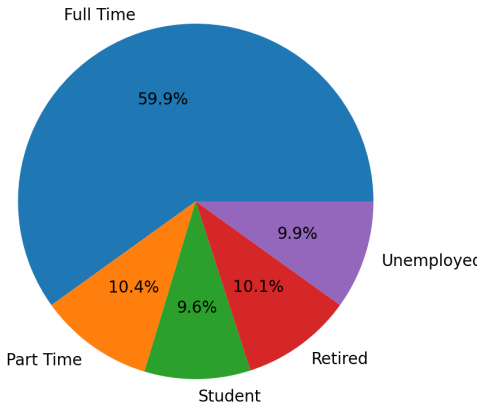
[nan 1. 4. 0. 3. 2. 7. 5. 9. 6. 10. 8.]




**Picture 16:** Customer Dependent Profile (25% of missing data)

6. Employment Status:

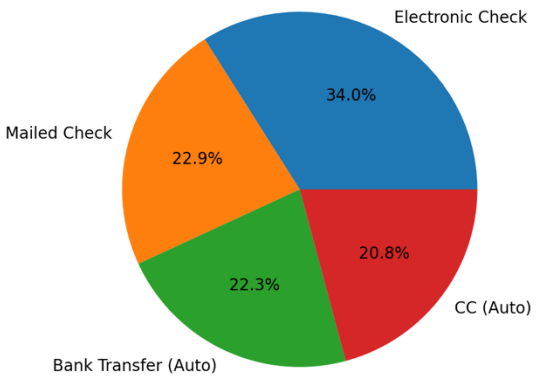
```
#Employment Values
df_clean['Employment'].unique()

#Customer Employment Profile
fulltime_count= df_clean['Employment'].value_counts()['Full Time']
parttime_count = df_clean['Employment'].value_counts()['Part Time']
student_count = df_clean['Employment'].value_counts()['Student']
retired_count = df_clean['Employment'].value_counts()['Retired']
unemployed_count = df_clean['Employment'].value_counts()['Unemployed']

employment_total = fulltime_count + parttime_count + student_count +
retired_count + unemployed_count

pct_fulltime = (fulltime_count / employment_total) * 100
pct_parttime = (parttime_count / employment_total) * 100
pct_student = (student_count / employment_total) * 100
pct_retired = (retired_count / employment_total) * 100
pct_unemployed = (unemployed_count / employment_total) * 100

#Pie Chart of Employment Distribution
plt.figure(figsize=(5,5))
labels = ["Full Time", "Part Time", "Student", "Retired", "Unemployed"]
values = [pct_fulltime, pct_parttime, pct_student, pct_retired,
pct_unemployed ]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```




**Picture 17:** Employment Distribution

7. Payment Method: most customers prefer to pay their bills electronic instead of setting it to auto payment, for example.

```
#Payment Method
df_clean['PaymentMethod'].unique()
eletronic_check_count= df_clean['PaymentMethod'].value_counts()['Electronic
Check']
mailed_check_count = df_clean['PaymentMethod'].value_counts()['Mailed Check']
bank_transfer_count = df_clean['PaymentMethod'].value_counts()['Bank
Transfer(automatic)']
cc_count = df_clean['PaymentMethod'].value_counts()['Credit Card
(automatic)']

payment_total = eletronic_check_count + mailed_check_count +
bank_transfer_count + cc_count

pct_eletronic_check = (eletronic_check_count / payment_total) * 100
pct_mailed_check = (mailed_check_count / payment_total) * 100
pct_bank_transfer = (bank_transfer_count / payment_total) * 100
pct_cc = (cc_count / payment_total) * 100


#Pie Chart of Employment Distribution
plt.figure(figsize=(5,5))
labels = ["Electronic Check", "Mailed Check", "Bank Transfer (Auto)", "CC
(Auto)"]
values = [pct_eletronic_check, pct_mailed_check, pct_bank_transfer, pct_cc ]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```

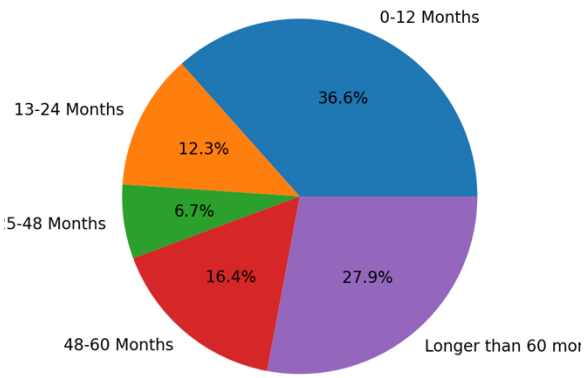



**Picture 18:** Payment Method Distribution

8. Tenure Time: Since tenure was presented as the length of time, in months, that a determined customer has been with the company, I converted the data to intervals so that I could analyze it in a better way. I divided the tenure in 5 different sub-divisions: new customers: they are having service with the company for less than 1 year, customers who have been with this company longer than 1 year but less than 2, customers in between 2 and 3 years, customers in between 3 and 4 years and finally people who are customers of the company longer than 5 years. Since the data is in months I converted it all to months. Most customers in our data is considered “new”. They have only been with this company less than 1 year.

```
#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()
```

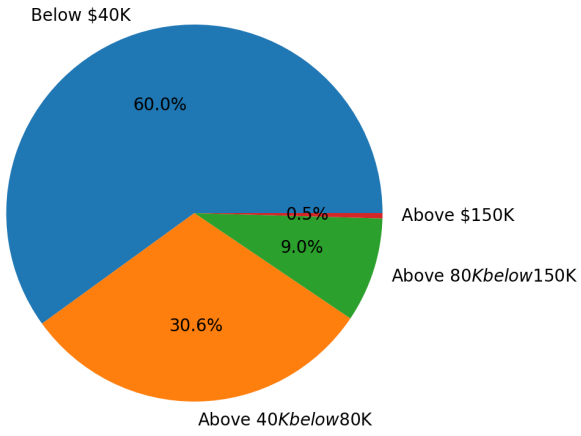



**Picture 19:** Percentage of customers in each tenure category

9. Income: I think the variable “Income” could also be a good indicator for customer churn rate. My next step then will be analyzing the income profile of the customers:

```
#Income Evaluation

def income(df_clean):
    if df_clean['Income'] <= 40000:
        return "Income Level 1"
    elif (df_clean['Income'] > 40001) & (df_clean['Income'] <= 80000):
        return "Income Level 2"
    elif (df_clean['Income'] > 80001) & (df_clean['Income'] <= 150000):
        return "Income Level 3"
```

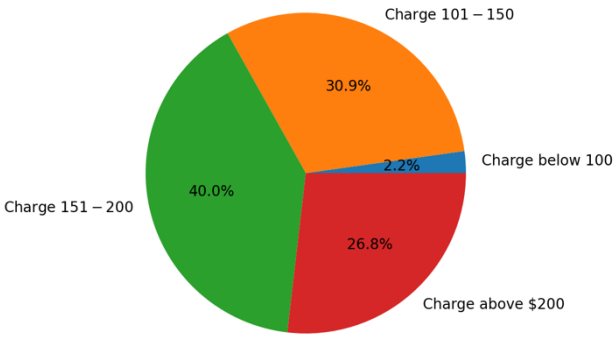
```
    elif (df_clean['Income'] > 150001):
        return "Income Level 4"

df_clean_3 = df_clean.copy()
df_clean_3['income_level'] = df_clean_3.apply(lambda
df_clean_3:income(df_clean_3), axis=1)
#df_clean_3['income_level']

inc1_count= df_clean_3['income_level'].value_counts()['Income Level 1']
inc2_count = df_clean_3['income_level'].value_counts()['Income Level 2']
inc3_count= df_clean_3['income_level'].value_counts()['Income Level 3']
inc4_count= df_clean_3['income_level'].value_counts()['Income Level 4']

inc_total = inc1_count + inc2_count + inc3_count + inc4_count

pct_inc1 = (inc1_count / inc_total) * 100
pct_inc2 = (inc2_count / inc_total) * 100
pct_inc3 = (inc3_count / inc_total) * 100
pct_inc4 = (inc4_count / inc_total) * 100

#Pie Chart of Income

plt.figure(figsize=(5,5))
labels = ["Income Level 1", "Income Level 2", "Income Level 3", "Income Level
4"]
values = [pct_inc1, pct_inc2, pct_inc3, pct_inc4]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```




**Picture 20:** Customer's Income Distribution

10. Monthly Charge 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()
```

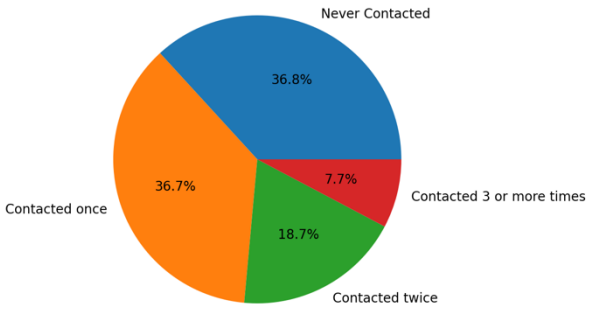



**Picture 21:** Monthly Charge Distribution

11. Contacts: One important factor might be how many times a customer needs to contact customer support. The more times the customer needs to call, most likely the same customer will churn. So we will also include the analysis of "Contacts"

```
#Number of times a customer had to call CS: "Contacts"
def number_contacts(df_clean):
    if df_clean['Contacts'] == 0:
        return "Never Contacted"
    elif (df_clean['Contacts'] == 1):
        return "Contacted once"
    elif (df_clean['Contacts'] ==2):
        return "Contacted twice"
    elif (df_clean['Contacts'] >= 3):
        return "Contacted 3 or more times"

df_clean_7 = df_clean.copy()
df_clean_7['Number_Contacts'] = df_clean_7.apply(lambda
df_clean_7:number_contacts(df_clean_7), axis=1)

contact1_count = df_clean_7['Number_Contacts'].value_counts()['Never
Contacted']
contact2_count = df_clean_7['Number_Contacts'].value_counts()['Contacted
once']
contact3_count = df_clean_7['Number_Contacts'].value_counts()['Contacted
twice']
contact4_count = df_clean_7['Number_Contacts'].value_counts()['Contacted 3 or
more times']

contact_total = contact1_count + contact2_count + contact3_count +
contact4_count

contact1_pct = (contact1_count / contact_total) * 100
contact2_pct = (contact2_count / contact_total) * 100
```

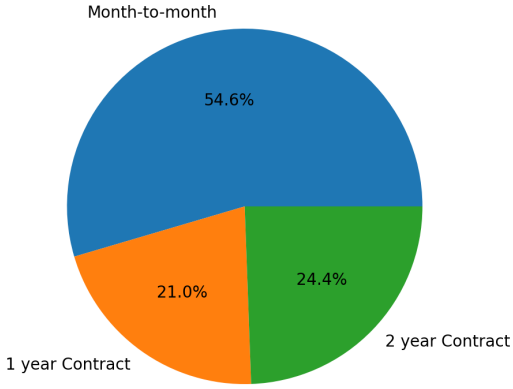
```
contact3_pct = (contact3_count / contact_total) * 100
contact4_pct = (contact4_count / contact_total) * 100

#Pie Chart of how many times customer contacted CS

plt.figure(figsize=(5,5))
labels = ["Never Contacted", "Contacted once", "Contacted twice", "Contacted
3 or more times"]
values = [contact1_pct, contact2_pct, contact3_pct, contact4_pct]
plt.pie(values, labels=labels, autopct="%.1f%%")
plt.show()
```




**Picture 22:** Percentage of customers contacting customer support

12. Type of Contract: I decided to include this variable in the analysis because it would make sense that customers who make automatic payments more often would churn less that customers who need to think about making the payments every month.

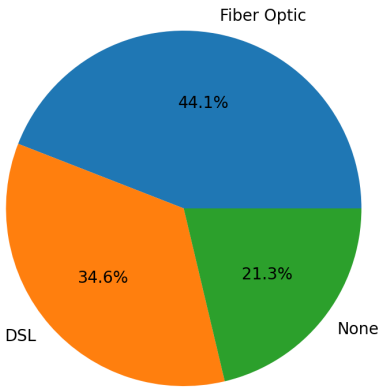
```
print(df_clean['Contract'].unique())

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 how many times customer contacted CS

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 23:** Type of Contract Distribution

13. Internet Service: Since service can be different depending on what you get, I am analyzing what kind of service customers decide to have. For example, fiber optic would be more expensive than DSL and maybe customers don't want to pay that much and decide to churn.

```
#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()
```

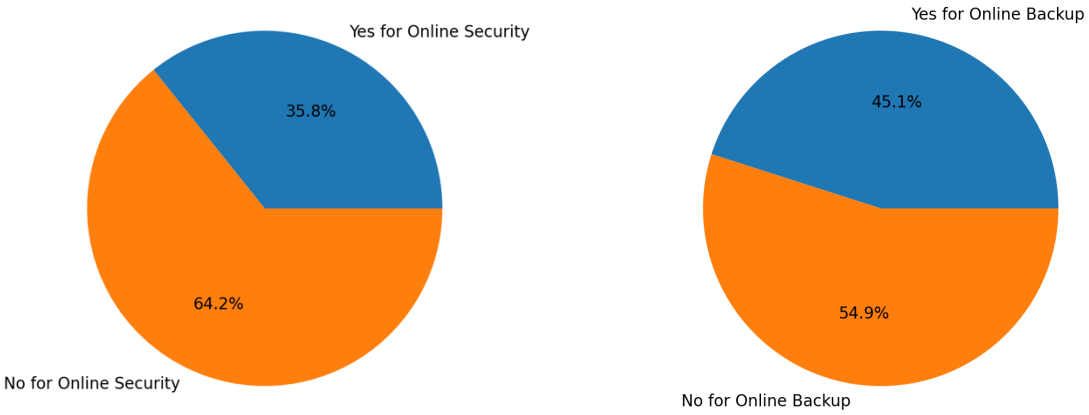



**Picture 24:** Type of Internet Service Provided

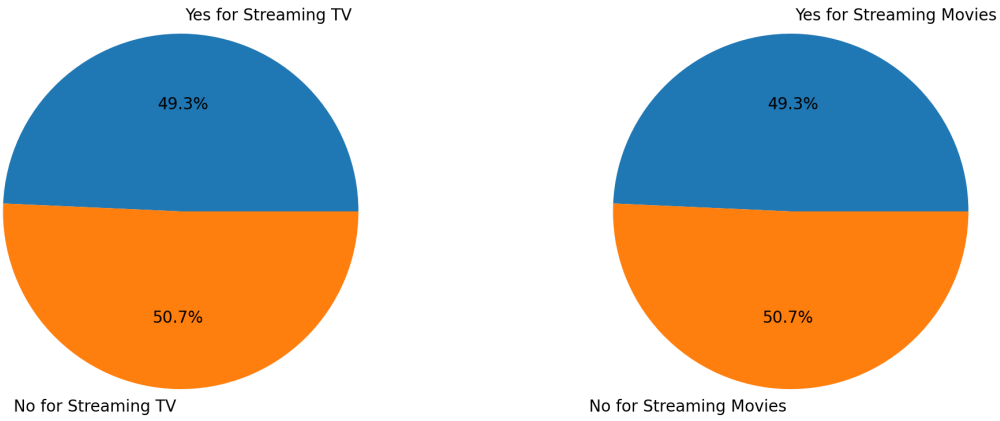
14. Service: That data is presented as Online Security, Online Backup, Device Protection, Tech Support, Streaming TV and Streaming Movies. All data here is either Yes or No. The only column that presents another value is “TechSupport” that also has “nan” (missing values).

```
print(df_clean['OnlineSecurity'].unique())
print(df_clean['OnlineBackup'].unique())
print(df_clean['DeviceProtection'].unique())
print(df_clean['TechSupport'].unique())
print(df_clean['StreamingTV'].unique())
print(df_clean['StreamingMovies'].unique())
```

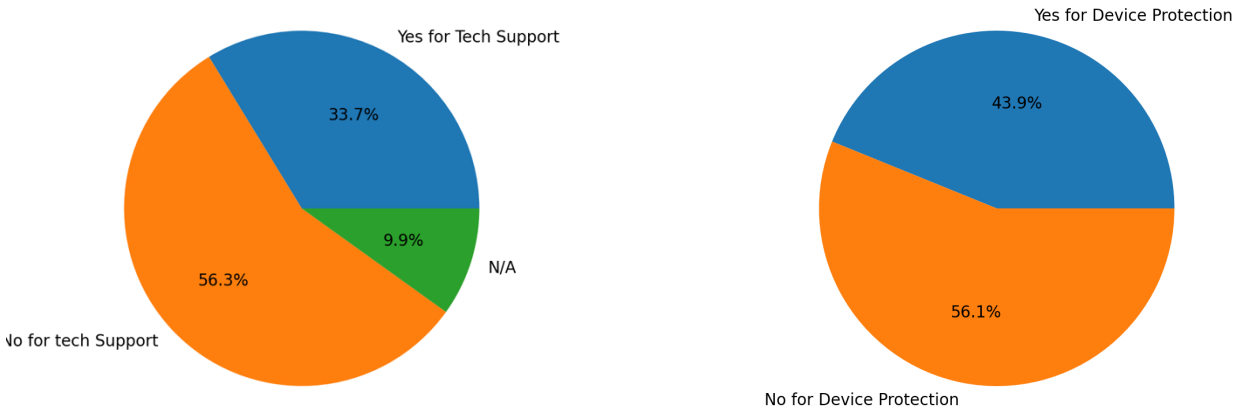
```
['Yes' 'No']
['Yes' 'No']
['No' 'Yes']
['No' 'Yes' nan]
['No' 'Yes']
['Yes' 'No']
```




**Picture 26:** Online Security and Online Backup Services




***Picture 25*:** Streaming TV and Movies

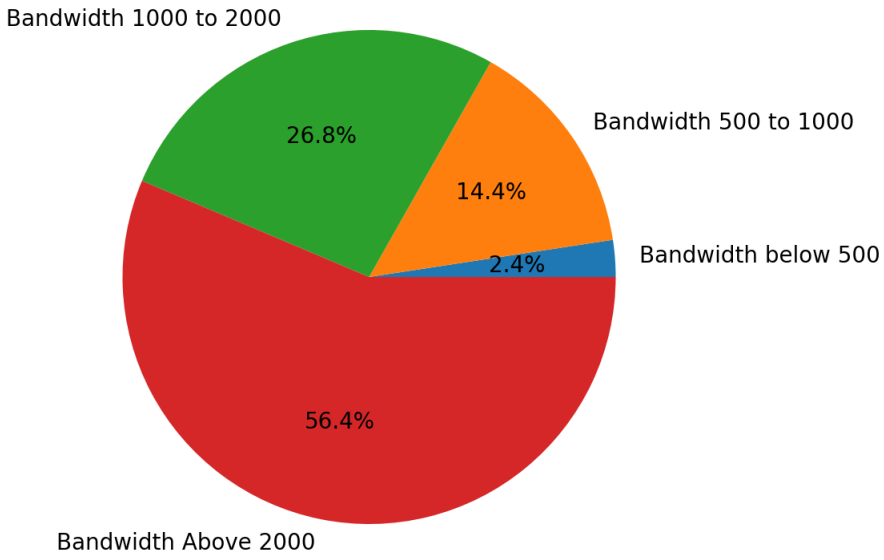



**Picture 27:** Tech Support and Device Protection Services

15. Bandwidth_GB_Year: this variable represents the average amount of data used in GB per year per customer

```
#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 28: Bandwidth Distribution**

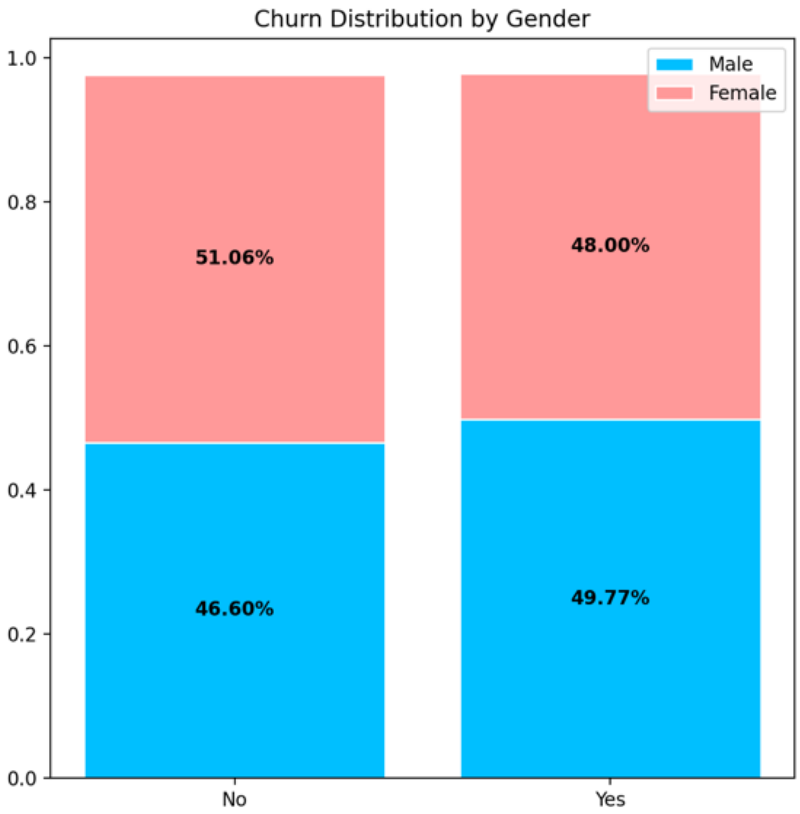
Next, I am going to analyze some key variables with the Churn Rate:

Next step is to plot each variable against the "Churn Rate". Depending on the response we can analyze it in more detail.

1. Gender Against Churn Rate:

Based on the picture on the next page we notice that gender might not be a good predictor for churn rate. The churn rate of females is slightly higher than males but not to the point that we could consider it significant. Also a little detail about this analysis is that some customers

gender info is labeled as “Prefer not the answer” so our percentages don’t add up to 100% because only customers labeled as either male or female were considered in this study.

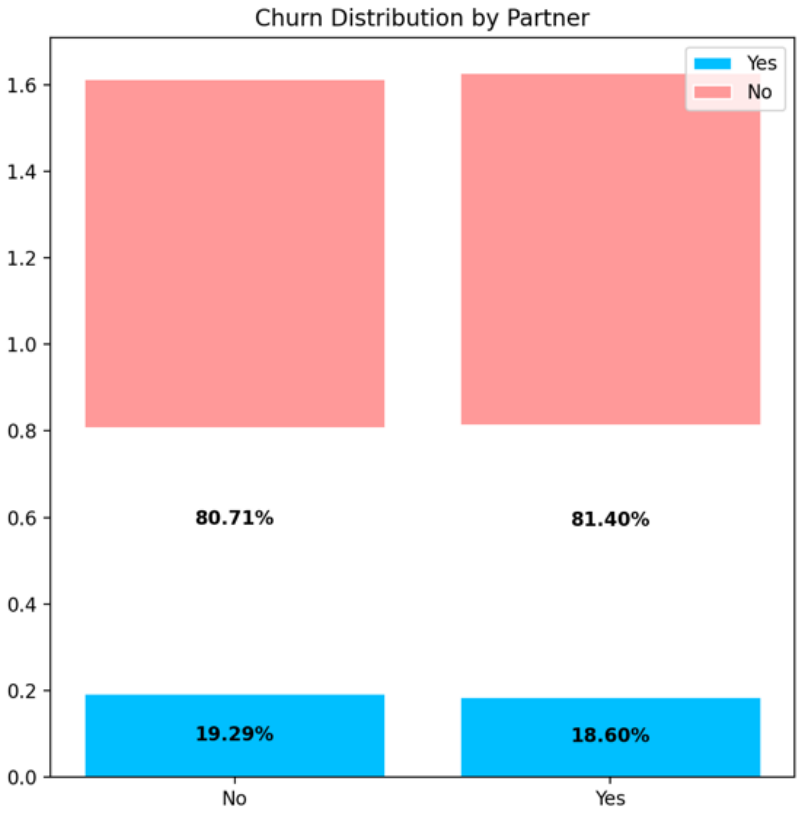



**Picture 29: Gender by Churn Rate**

2. Marital Status: I changed the “Marital” column to either present “Yes” for Married and “No” for the rest of the marital statuses. Since the churn and no churn rates are very similar, marital status might not be a good predictor for churn.

```
#Partner against Churn Rate
no_churn = ((df_clean_4[df_clean_4['Churn'] ==
'No']['partner'].value_counts()) / (df_clean_4[df_clean_4['Churn'] ==
'No']['partner'].value_counts().sum()))
yes_churn = ((df_clean_4[df_clean_4['Churn'] ==
'Yes']['partner'].value_counts()) / (df_clean_4[df_clean_4['Churn'] ==
'Yes']['partner'].value_counts().sum()))

# # Getting values from the group and categories
x_labels = df_clean_4['Churn'].value_counts().keys().tolist()
y_var = [no_churn['Yes'], yes_churn['Yes']]
n_var = [no_churn['No'], yes_churn['No']]
```

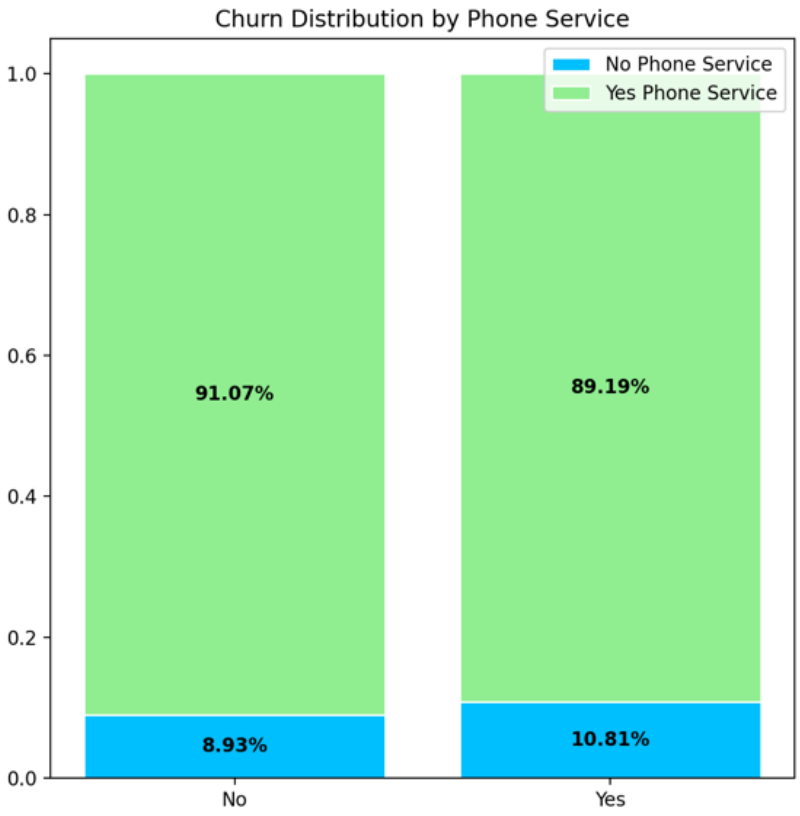
```
#
# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7, 7))
ax1 = plt.bar(x_labels, y_var, color='#00BFFF', label='Yes',
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, n_var, bottom= n_var, color='#FF9999', label='No',
edgecolor='white', width=barWidth)
plt.legend()
plt.title('Churn Distribution by Partner')

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 30:** Partner Against Marital Status

3. Phone Service Against Churn Rate: Again this variable presents very similar churn and no churn rate and probably wont be use to predict future churn rate.

```
#Phone Service
no_churn = ((df_clean[df_clean['Churn'] == 'No']['Phone'].value_counts()) /
(df_clean[df_clean['Churn'] == 'No']['Phone'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] == 'Yes']['Phone'].value_counts()) /
(df_clean[df_clean['Churn'] == 'Yes']['Phone'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

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

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()
```

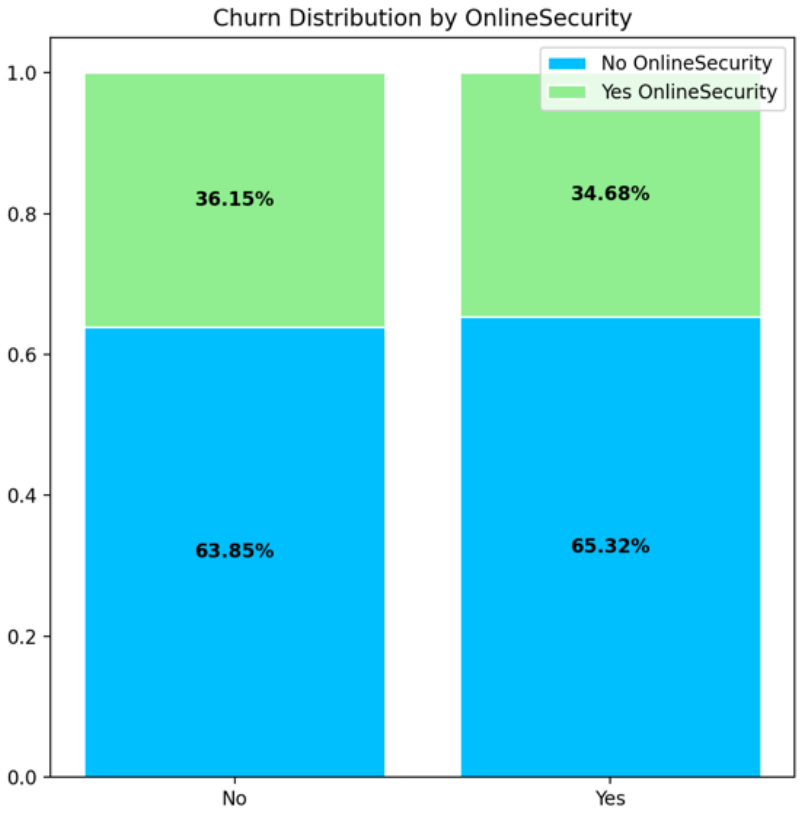



**Picture 31:** Phone Service and Churn Rate

4. Online Security Against Churn Rate: The result is very similar again so this variable wont be used to predict future customer churn.

```
#Online Security
no_churn = ((df_clean[df_clean['Churn'] ==
'No']['OnlineSecurity'].value_counts()) / (df_clean[df_clean['Churn'] ==
'No']['OnlineSecurity'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['OnlineSecurity'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['OnlineSecurity'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7, 7))
```

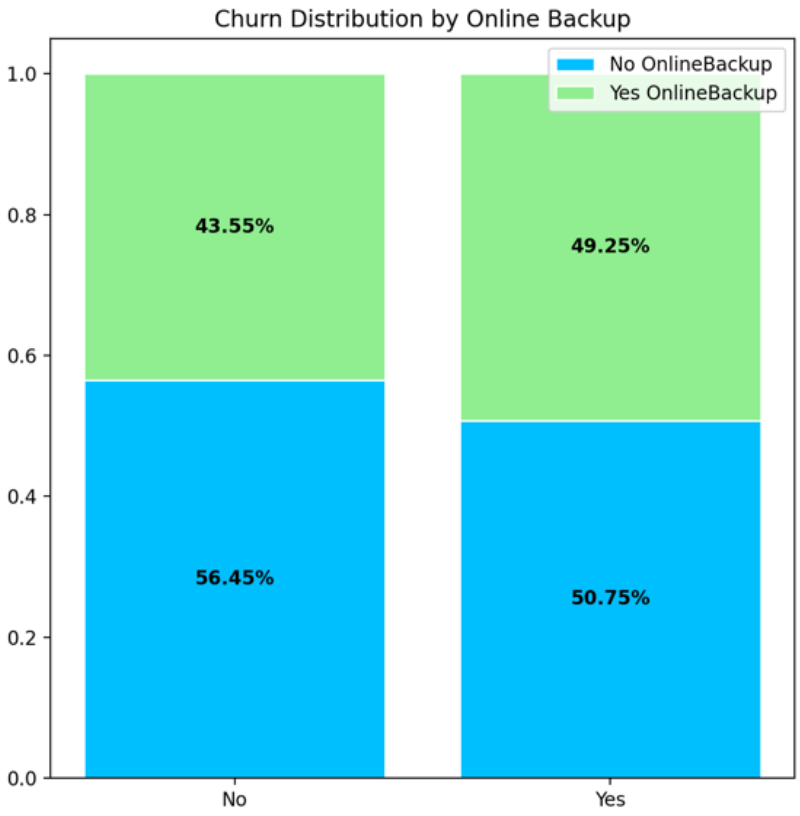
```
ax1 = plt.bar(x_labels, n_var, color='#00BFFF', label=('No OnlineSecurity'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, y_var, bottom=n_var, color='lightgreen', label=('Yes
OnlineSecurity'), edgecolor='white', width=barWidth)
plt.legend()
plt.title('Churn Distribution by OnlineSecurity')

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 32:** Online Security and Churn Rate

5. Online Backup Against Churn Rate: customers who have signed up for this service has a slightly higher churn rate.

```
#Online Backup
no_churn = ((df_clean[df_clean['Churn'] ==
'No']['OnlineBackup'].value_counts()) / (df_clean[df_clean['Churn'] ==
'No']['OnlineBackup'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['OnlineBackup'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['OnlineBackup'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

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

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()
```

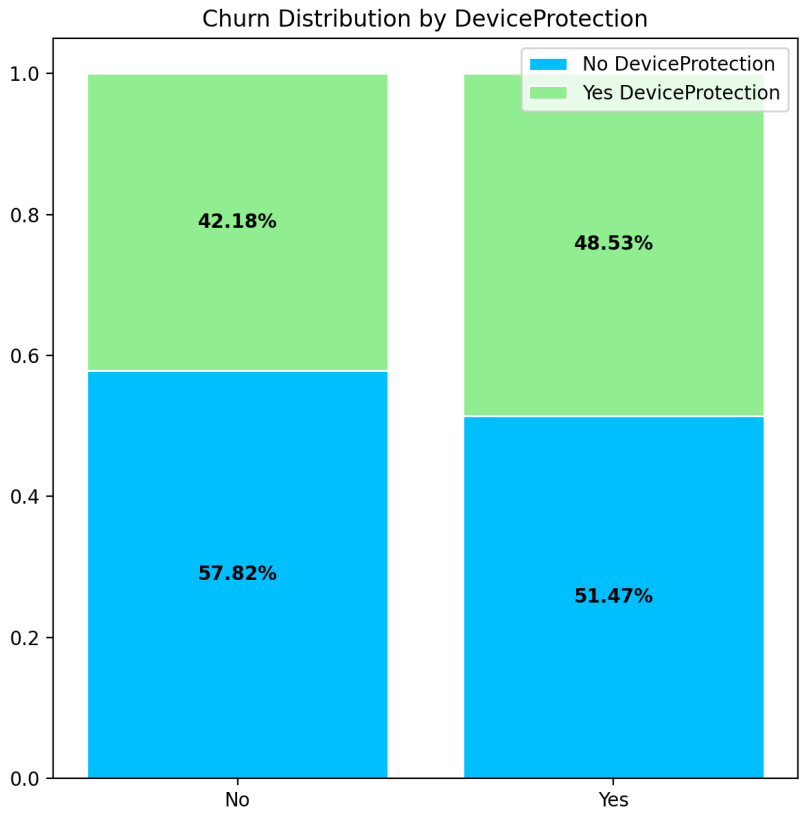



**Picture 33:** Online Backup Service and Churn Rate

6. Device Protection Against Churn Rate: customers who have signed up for this service has a slightly higher churn rate.

```
#Device Protection
no_churn = ((df_clean[df_clean['Churn'] ==
'No']['DeviceProtection'].value_counts()) / (df_clean[df_clean['Churn'] ==
'No']['DeviceProtection'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['DeviceProtection'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['DeviceProtection'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

# Plotting bars
barWidth = 0.8
```

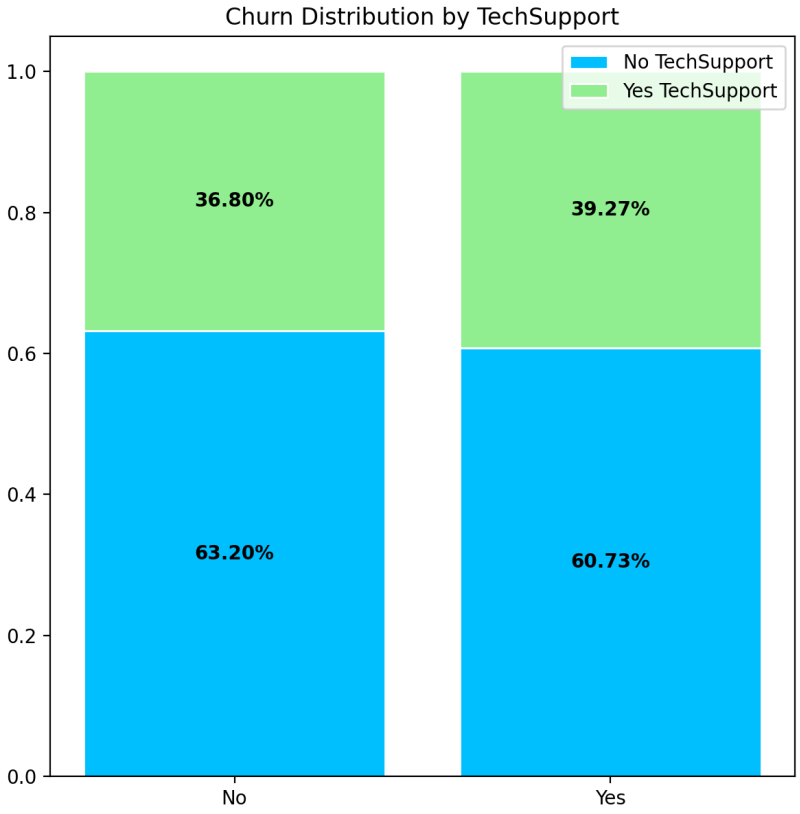
```
plt.figure(figsize=(7, 7))
ax1 = plt.bar(x_labels, n_var, color='#00BFFF', label=('No
DeviceProtection'), edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, y_var, bottom=n_var, color='lightgreen', label=('Yes
DeviceProtection'), edgecolor='white', width=barWidth)
plt.legend()
plt.title('Churn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

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 34:** Device Protection Against Churn Rate

7. Tech Support Against Churn Rate: percentages are very similar so this variable might not be used to predict churn rate.

```
#Tech Support
no_churn = ((df_clean[df_clean['Churn'] ==
'No']['TechSupport'].value_counts()) / (df_clean[df_clean['Churn'] ==
'No']['TechSupport'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['TechSupport'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['TechSupport'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

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

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()
```



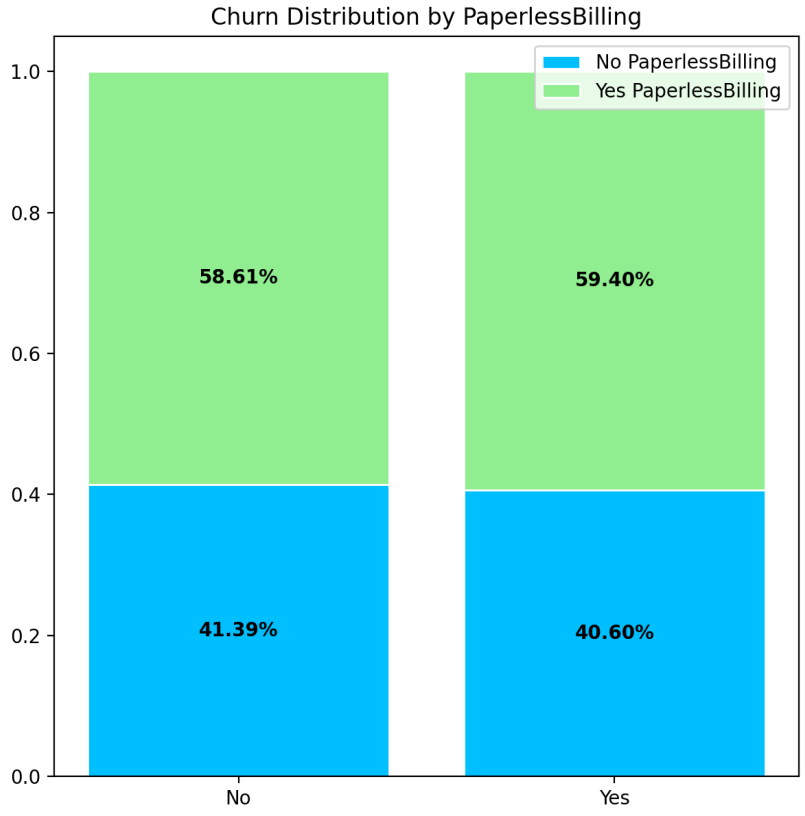

**Picture 35:** Tech Support and Churn Rate

Tech Support wouldn't be a good predictor of customer churn rate. The percentages of churn and no churn are very similar.

8. Paperless Billing: this variable will not be used to predict churn rate.

```
#Paperless Billing
no_churn = ((df_clean[df_clean['Churn'] ==
'No']['PaperlessBilling'].value_counts()) / (df_clean[df_clean['Churn'] ==
'No']['PaperlessBilling'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['PaperlessBilling'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['PaperlessBilling'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

# Plotting bars
```

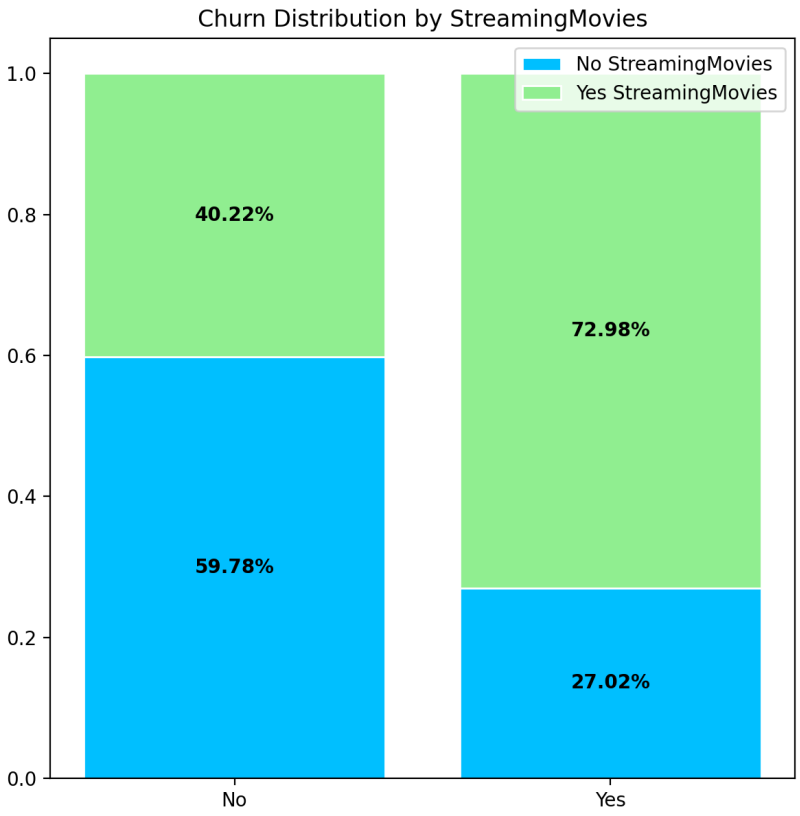
```
barWidth = 0.8
plt.figure(figsize=(7, 7))
ax1 = plt.bar(x_labels, n_var, color='#00BFFF', label=('No
PaperlessBilling'), edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, y_var, bottom=n_var, color='lightgreen', label=('Yes
PaperlessBilling'), edgecolor='white', width=barWidth)
plt.legend()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

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()
```



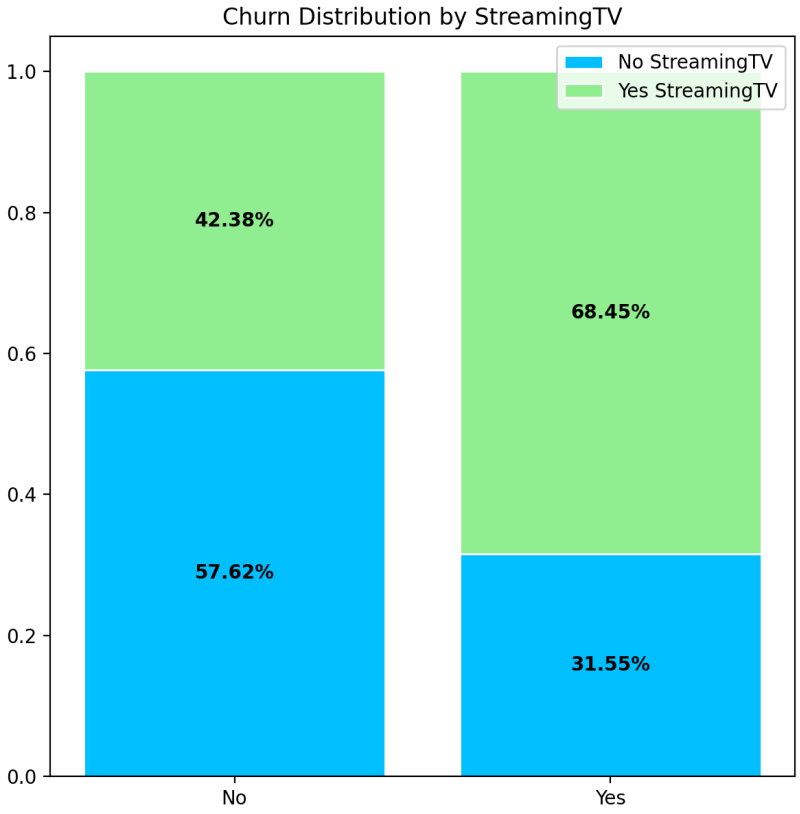

**Picture 36:** Paperless Billing Against Churn

9. Streaming TV and Streaming Movies Services: Customers who choose this service churn more than customers with no streaming service.

```
#Streaming TV and Movies
no_churn = ((df_clean[df_clean['Churn'] ==
'No']['StreamingMovies'].value_counts()) / (df_clean[df_clean['Churn'] ==
'No']['StreamingMovies'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['StreamingMovies'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['StreamingMovies'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7, 7))
ax1 = plt.bar(x_labels, n_var, color='#00BFFF', label=('No StreamingMovies'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, y_var, bottom=n_var, color='lightgreen', label=('Yes
StreamingMovies'), edgecolor='white', width=barWidth)
plt.legend()
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 37:** Streaming Movies Against Churn

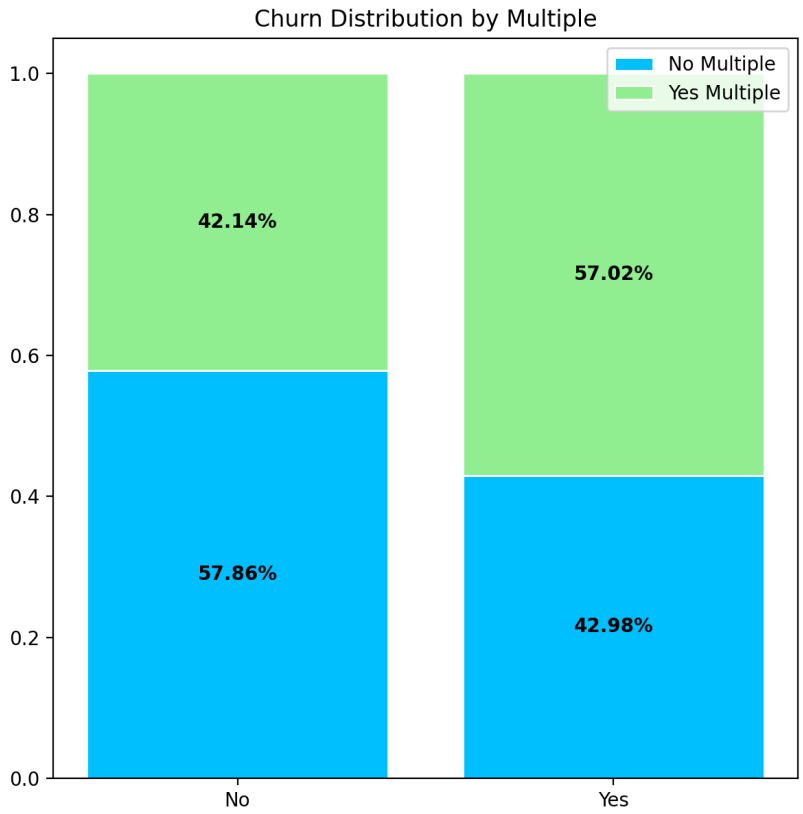



**Picture 38:** Streaming TV Against Churn

10. Multiple Lines: customers who choose to have multiple lines also churn more.

```
#Multiple lines service
no_churn = ((df_clean[df_clean['Churn'] == 'No']['Multiple'].value_counts())
/ (df_clean[df_clean['Churn'] == 'No']['Multiple'].value_counts().sum()))
yes_churn = ((df_clean[df_clean['Churn'] ==
'Yes']['Multiple'].value_counts()) / (df_clean[df_clean['Churn'] ==
'Yes']['Multiple'].value_counts().sum()))

# Getting values from the group and categories
#Churn is my X axis
x_labels = df_clean['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']]

# Plotting bars
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()
```

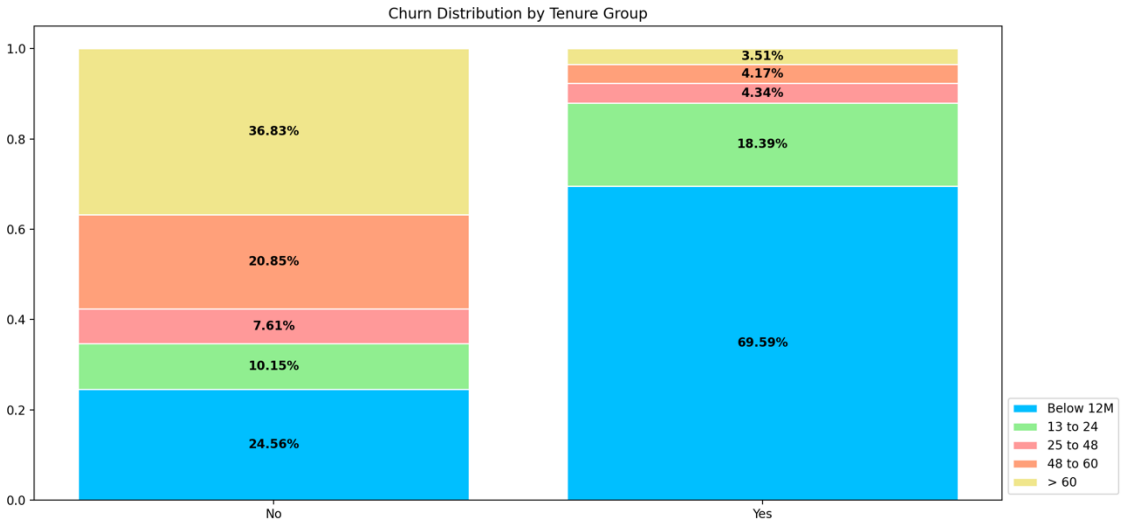



**Picture 39:** Multiple Line Service Against Churn Rate

11. Tenure Against Churn Rate: We clearly notice that the shorter the tenure, higher the churn rate. This result is exactly as expected.

```
#Tenure and Churn Rate
no_churn =
```

```
((df_clean_2[df_clean_2['Churn']=='No']['tenure_group'].value_counts())
/(df_clean_2[df_clean_2['Churn']=='No']['tenure_group'].value_counts().sum())
)
yes_churn =
((df_clean_2[df_clean_2['Churn']=='Yes']['tenure_group'].value_counts())
/(df_clean_2[df_clean_2['Churn']=='Yes']['tenure_group'].value_counts().sum()
))

# Getting values from the group and categories
x_labels = df_clean['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()
```

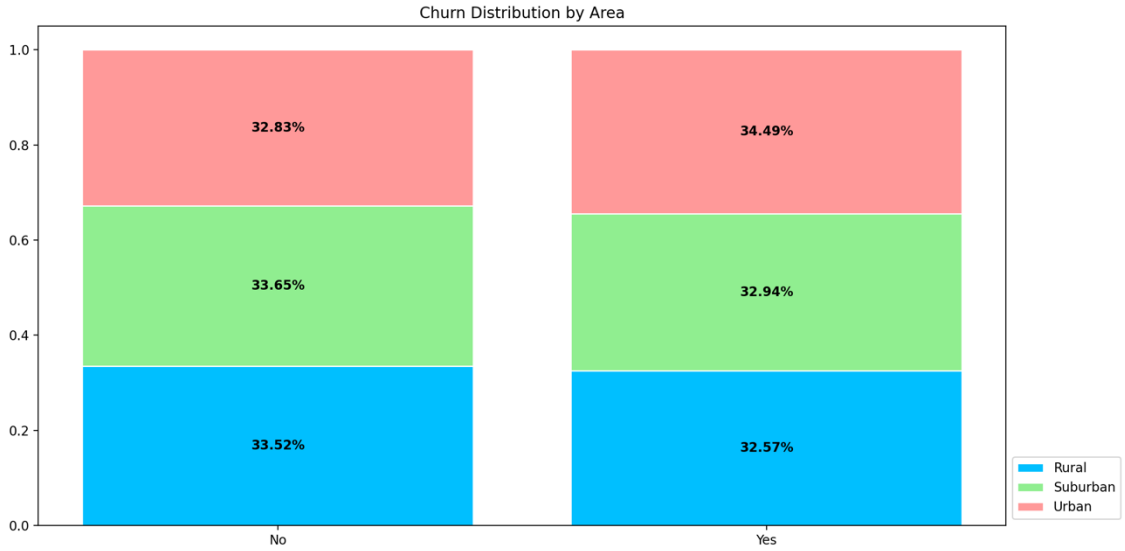



**Picture 40:** Tenure and Churn Rate

12. Area Against Churn Rate: area would not be a good predictor for future customers rate

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

# Getting values from the group and categories
x_labels = df_clean['Churn'].value_counts().keys().tolist()
rural = [no_churn['Rural'], yes_churn['Rural']]
suburban = [no_churn['Suburban'], yes_churn['Suburban']]
urban = [no_churn['Urban'], yes_churn['Urban']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, rural, color='#00BFFF', label=('Rural'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, suburban, bottom=rural, color='lightgreen',
label=('Suburban'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, urban, bottom=np.array(rural) + np.array(suburban),
color='#FF9999',  label=('Urban'), edgecolor='white', width=barWidth)

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

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()
```

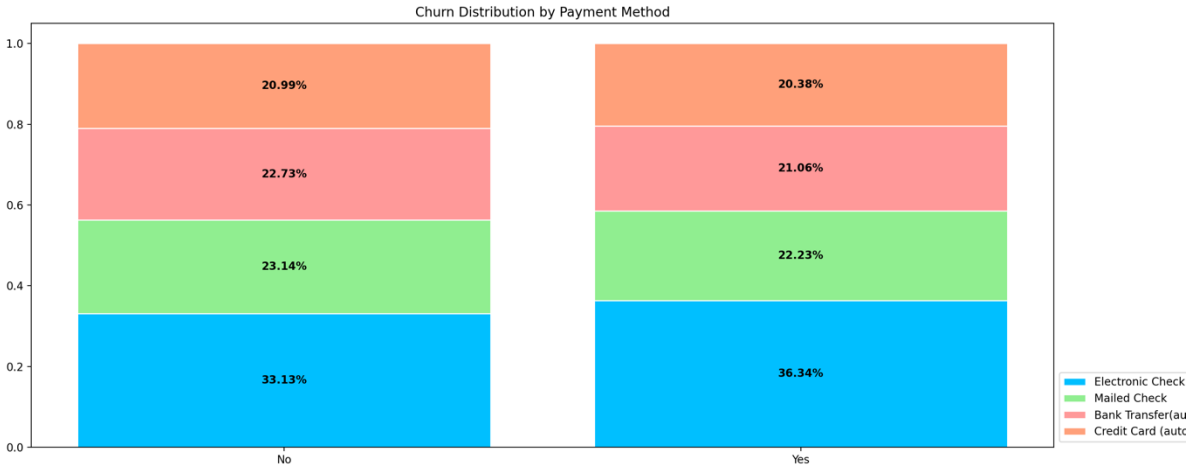



**Picture 41:** Area Against Churn Rate

13. Payment Type Against Churn Rate: payment type, surprisingly, doesn't show a good predictor for churn. I would think that if a customer needs to send money every month and opt out for auto payment, they would churn more often.

```
#Payment type and Churn Rate
no_churn =
((df_clean[df_clean['Churn']=='No']['PaymentMethod'].value_counts())
/(df_clean[df_clean['Churn']=='No']['PaymentMethod'].value_counts().sum()))
yes_churn =
((df_clean[df_clean['Churn']=='Yes']['PaymentMethod'].value_counts())
/(df_clean[df_clean['Churn']=='Yes']['PaymentMethod'].value_counts().sum()))

# Getting values from the group and categories
x_labels = df_clean['Churn'].value_counts().keys().tolist()
electornic_check = [no_churn['Electronic Check'], yes_churn['Electronic
Check']]
mailed_check = [no_churn['Mailed Check'], yes_churn['Mailed Check']]
bank_transfer = [no_churn['Bank Transfer(automatic)'], yes_churn['Bank
Transfer(automatic)']]
credit_card = [no_churn['Credit Card (automatic)'], yes_churn['Credit Card
(automatic)']]
```

```
# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, electornic_check, color='#00BFFF', label=('Electronic
Check'), edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, mailed_check, bottom=electornic_check,
color='lightgreen', label=('Mailed Check'), edgecolor='white',
width=barWidth)
ax3 = plt.bar(x_labels, bank_transfer, bottom=np.array(electornic_check) +
np.array(mailed_check), color='#FF9999', label=('Bank Transfer(automatic)'),
edgecolor='white', width=barWidth)
ax4 = plt.bar(x_labels, credit_card, bottom=np.array(electornic_check) +
np.array(mailed_check) + np.array(bank_transfer), color='#FFA07A',
label=('Credit Card (automatic)'), edgecolor='white', width=barWidth)

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

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()
```

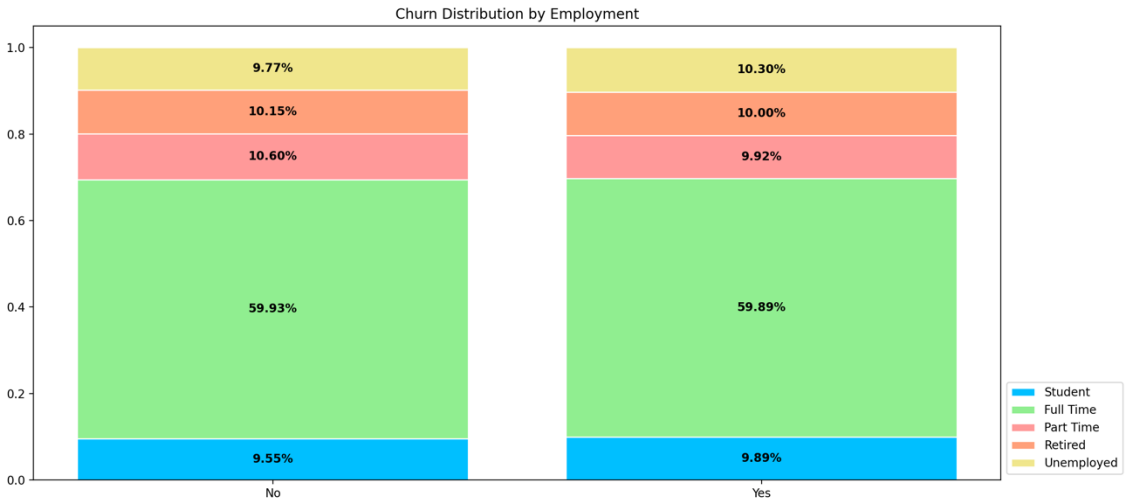



**Picture 42:** Payment Method Against Churn Rate

14. Employment Against Churn Rate: employment didn't show a great predictor for churn rate

```
15.  #Employment type and Churn Rate
   no_churn =
   ((df_clean[df_clean['Churn']=='No']['Employment'].value_counts())
   /(df_clean[df_clean['Churn']=='No']['Employment'].value_counts().sum())
   )
   yes_churn =
   ((df_clean[df_clean['Churn']=='Yes']['Employment'].value_counts())
   /(df_clean[df_clean['Churn']=='Yes']['Employment'].value_counts().sum()
   ))

   # Getting values from the group and categories
   x_labels = df_clean['Churn'].value_counts().keys().tolist()
   student = [no_churn['Student'], yes_churn['Student']]
   full_time = [no_churn['Full Time'], yes_churn['Full Time']]
   part_time = [no_churn['Part Time'], yes_churn['Part Time']]
   retired = [no_churn['Retired'], yes_churn['Retired']]
   unemployed = [no_churn['Unemployed'], yes_churn['Unemployed']]

   # Plotting bars
   barWidth = 0.8
   plt.figure(figsize=(7,7))
   ax1 = plt.bar(x_labels, student, color='#00BFFF', label=('Student'),
   edgecolor='white', width=barWidth)
   ax2 = plt.bar(x_labels, full_time, bottom=student, color='lightgreen',
   label=('Full Time'), edgecolor='white', width=barWidth)
   ax3 = plt.bar(x_labels, part_time, bottom=np.array(full_time) +
   np.array(student), color='#FF9999', label=('Part Time'),
   edgecolor='white', width=barWidth)
   ax4 = plt.bar(x_labels, retired, bottom=np.array(student) +
   np.array(full_time) + np.array(part_time), color='#FFA07A',
```

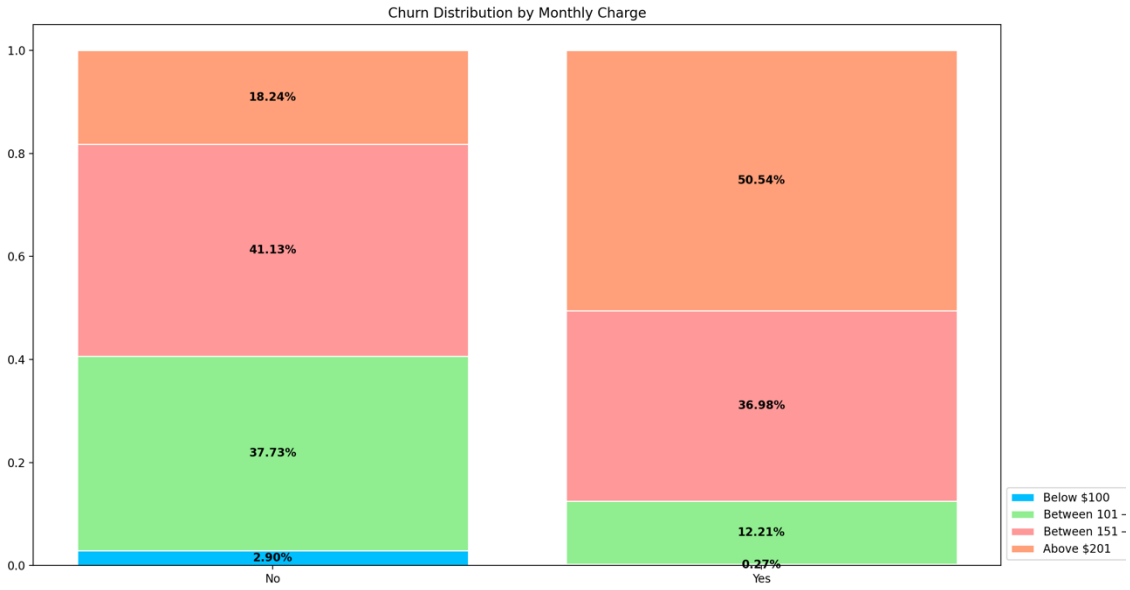
```
    label=('Retired'), edgecolor='white', width=barWidth)
    ax5 = plt.bar(x_labels, unemployed, bottom=np.array(student) +
    np.array(full_time) + np.array(part_time) + np.array(retired),
    color='#F0E68C', label=('Unemployed'), edgecolor='white',
    width=barWidth)

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

    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 43:** Employment Against Churn Rate

16. Monthly Charge: As expected, the higher the monthly charge, the higher the customer churn rate.

```
#Monthly Charge and Churn Rate
no_churn =
((df_clean_6[df_clean['Churn']=='No']['Monthly_Charge'].value_counts())
/(df_clean_6[df_clean_6['Churn']=='No']['Monthly_Charge'].value_counts().sum(
)))
yes_churn =
((df_clean_6[df_clean['Churn']=='Yes']['Monthly_Charge'].value_counts())
/(df_clean_6[df_clean_6['Churn']=='Yes']['Monthly_Charge'].value_counts().sum
()))

# Getting values from the group and categories
x_labels = df_clean_6['Churn'].value_counts().keys().tolist()
level1 = [no_churn['Charge Level 1'], yes_churn['Charge Level 1']]
level2 = [no_churn['Charge Level 2'], yes_churn['Charge Level 2']]
level3 = [no_churn['Charge Level 3'], yes_churn['Charge Level 3']]
level4 = [no_churn['Charge Level 4'], yes_churn['Charge Level 4']]

# Plotting bars
barWidth = 0.8
plt.figure(figsize=(7,7))
ax1 = plt.bar(x_labels, level1, color='#00BFFF', label=('Below $100'),
edgecolor='white', width=barWidth)
ax2 = plt.bar(x_labels, level2, bottom=level1, color='lightgreen',
label=('Between $101 - $150'), edgecolor='white', width=barWidth)
ax3 = plt.bar(x_labels, level3, bottom=np.array(level2) + np.array(level1),
color='#FF9999', label=('Between $151 - $200'), edgecolor='white',
width=barWidth)
ax4 = plt.bar(x_labels, level4, bottom=np.array(level1) + np.array(level2) +
np.array(level3), color='#FFA07A', label=('Above $201'), 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()
```

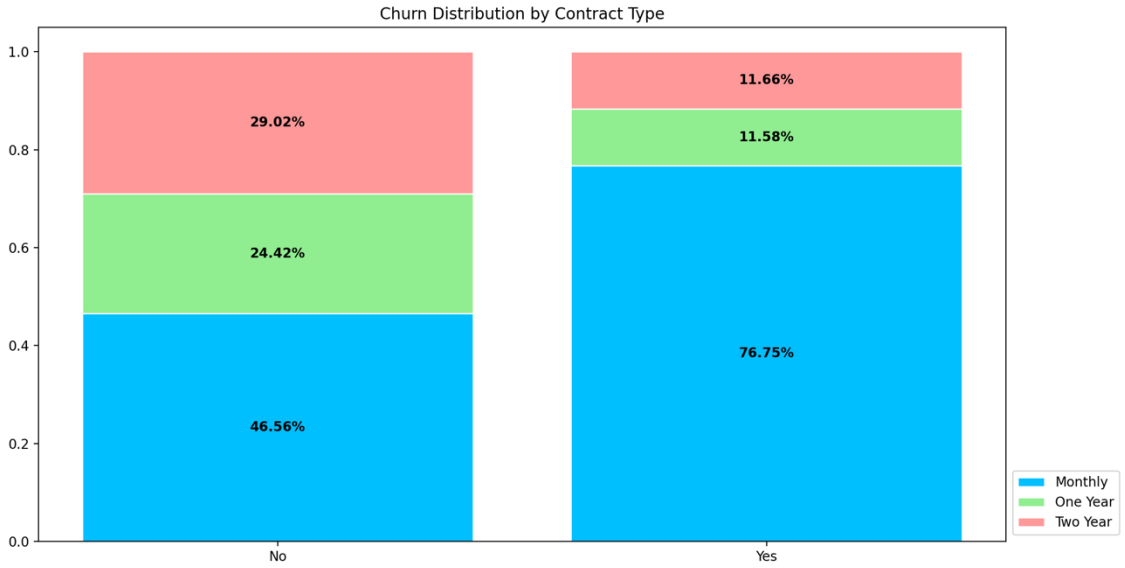



**Picture 44:** Monthly Charge by Churn Rate

17. Contract Type: As expected, “Month to Month” contract customers tend to churn more than customers with longer contracts.

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

# Getting values from the group and categories
x_labels = df_clean['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()
```

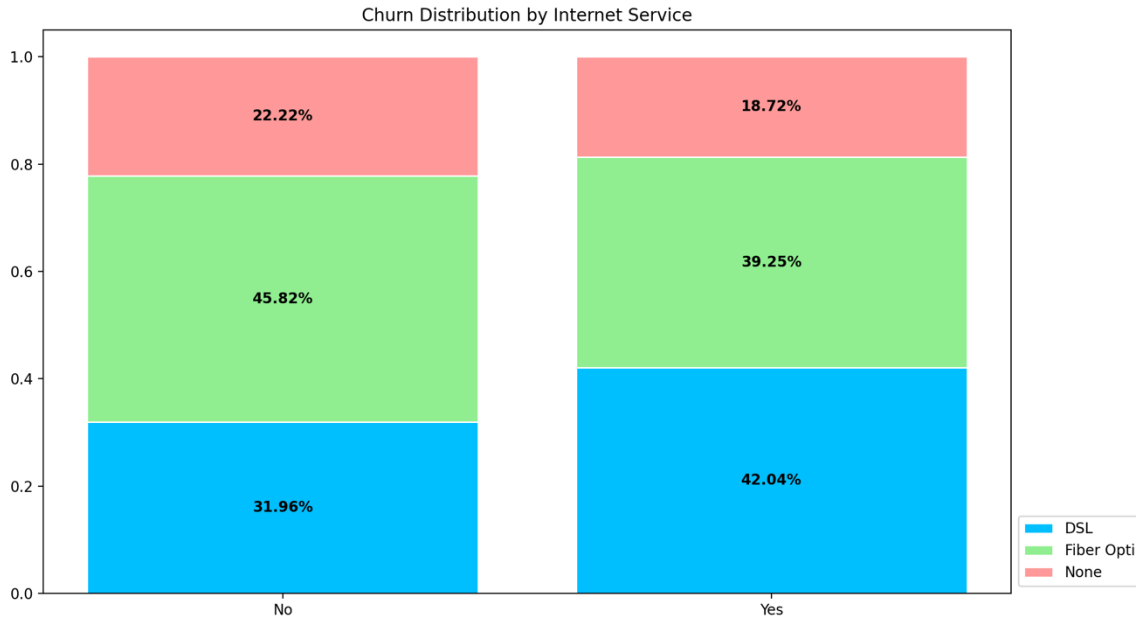



**Picture 45:** Monthly Charge by Contract Type

18. Internet Service: We can notice that customers with DSL internet service tend to churn more than customers with other types of service.

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

```
# Getting values from the group and categories
x_labels = df_clean['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()
```

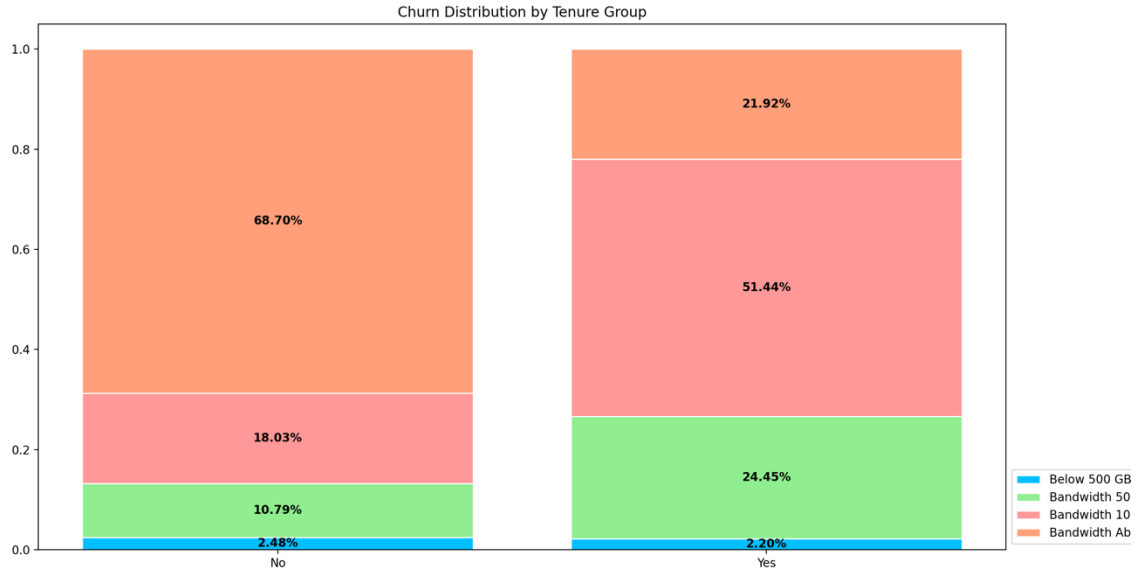



**Picture 46:** Churn Rate and Internet Service

19. Bandwidth_GB_Year: we can clearly see that customers with bandwidth usage from 1000 to 2000GB per year tend to churn more than other customers

```
#Bandwidth
no_churn =
((df_clean_10[df_clean_10['Churn']=='No']['Bandwidth_GB_Year'].value_counts()
)
/(df_clean_10[df_clean_10['Churn']=='No']['Bandwidth_GB_Year'].value_counts()
.sum()))
yes_churn =
((df_clean_10[df_clean_10['Churn']=='Yes']['Bandwidth_GB_Year'].value_counts(
))
/(df_clean_10[df_clean_10['Churn']=='Yes']['Bandwidth_GB_Year'].value_counts(
).sum()))

# Getting values from the group and categories
x_labels = df_clean['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=('Bandwidth 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=('Bandwidth 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=('Bandwidth Above 2000'), 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 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 47:** Churn Rate and Bandwidth per year

**PART D1 – CLEANING FINDINGS**

**DATA CLEANING:**
Lets start verifying the types of data I still have in my dataset:

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

**Data columns (total 46 columns):**

| # | Column | Non-Null. | Count | Dtype |
|---|---|---|---|---|
| 0 | Customer_id | non-null | 10000 | Object |
| 1 | Area | non-null | 10000 | Object |
| 2 | TimeZone | non-null | 10000 | Object |
| 3 | Job | non-null | 10000 | Object |
| 4 | Children | non-null | 7505 | float64 |
| 5 | Age | non-null | 7525 | float64 |
| 6 | Education | non-null | 10000 | Object |
| 7 | Employment | non-null | 10000 | Object |
| 8 | Income | non-null | 7510 | Float64 |
| 9 | Marital | non-null | 10000 | Object |
| 10 | Gender | non-null | 10000 | Object |
| 11 | Churn | non-null | 10000 | Object |
| 12 | Outage_sec_perweek | non-null | 10000 | Float64 |
| 13 | Email | non-null | 10000 | Int64 |
| 14 | Contacts | non-null | 10000 | Int64 |
| 15 | Yearly_equip_failure | non-null | 10000 | Int64 |
| 16 | Techie | non-null | 7523 | Object |
| 17 | Contract | non-null | 10000 | Object |
| 18 | Port_modem | non-null | 10000 | Object |
| 19 | Tablet | non-null | 10000 | Object |
| 20 | InternetService | non-null | 10000 | Object |
| 21 | Phone | non-null | 8974 | Object |
| 22 | Multiple | non-null | 10000 | Object |
| 23 | OnlineSecurity | non-null | 10000 | Object |
| 24 | OnlineBackup | non-null | 10000 | Object |
| 25 | DeviceProtection | non-null | 10000 | Object |
| 26 | TechSupport | non-null | 9009 | Object |
| 27 | StreamingTV | non-null | 10000 | Object |
| 28 | StreamingMovies | non-null | 10000 | Object |
| 29 | PaperlessBilling | non-null | 10000 | Object |
| 30 | PaymentMethod | non-null | 10000 | Object |
| 31 | Tenure | non-null | 9069 | Float64 |
| 32 | MonthlyCharge | non-null | 10000 | Float64 |

| | | | | |
|---|---|---|---|---|
| 33 | Bandwidth_GB_Year | non-null | 8979 | Float64 |
| 34 | Responses | non-null | 10000 | Int64 |
| 35 | Fixes | non-null | 10000 | Int64 |
| 36 | Replacements | non-null | 10000 | Int64 |
| 37 | Reliability | non-null | 10000 | Int64 |
| 38 | Options | non-null | 10000 | Int64 |
| 39 | Respectfulness | non-null | 10000 | Int64 |
| 40 | Courteous | non-null | 10000 | Int64 |
| 41 | Listening | non-null | 10000 | Int64 |

dtypes: float64(7), int64(11), object(24)
memory usage: 3.2+ MB

We notice that lots of columns display values below 10,000 meaning that we have missing data. Unfortunately most predictive modeling techniques cannot handle any missing values [6]. Let's concentrate in analyzing the missing data and how to proceed from here.

Next step is to check which columns present null values: True for columns lacking data and False if the column has no missing data.

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

| Columns | Null Data? | Columns | Null Data? | Columns | Null Data? |
|---|---|---|---|---|---|
| Customer_id | False | Outage_sec_perweek | False | StreamingTV | False |
| State | False | Email | False | StreamingMovies | False |
| County | False | Contacts | False | PaperlessBilling | False |
| Population | False | Yearly_equip_failure | False | PaymentMethod | False |
| Area | False | Techie | True | Tenure | True |
| Timezone | False | Contract | False | MonthlyCharge | False |
| Job | False | Port_modem | False | Bandwidth_GB_Year | True |
| Children | True | Tablet | False | Responses | False |
| Age | True | InternetService | False | Fixes | False |
| Education | False | Phone | True | Replacements | False |
| Employment | False | Multiple | False | Reliability | False |
| Income | True | OnlineSecurity | False | Options | False |
| Marital | False | OnlineBackup | False | Respectfulness | False |
| Gender | False | DeviceProtection | False | Courteous | False |
| Churn | False | TechSupport | True | Listening | False |

dtype: bool

We can see that some data is missing, lets see how much data is truly missing:

```
data_null_sum = df.isnull().sum()
print(data_null_sum)
```

| Columns | # of missing entries | Columns | # of missing entries | Columns | # of missing entries |
|---|---|---|---|---|---|
| Customer_id | 0 | Outage_sec_perweek | 0 | StreamingTV | False |
| State | 0 | Email | 0 | StreamingMovies | 0 |
| County | 0 | Contacts | 0 | PaperlessBilling | 0 |
| Population | 0 | Yearly_equip_failure | 0 | PaymentMethod | 0 |
| Area | 0 | Techie | 2477 | Tenure | 931 |
| Timezone | 0 | Contract | 0 | MonthlyCharge | 0 |
| Job | 0 | Port_modem | 0 | Bandwidth_GB_Year | 1021 |
| Children | 2495 | Tablet | 0 | Responses | 0 |
| Age | 2475 | InternetService | 0 | Fixes | 0 |
| Education | 0 | Phone | 1026 | Replacements | 0 |
| Employment | 0 | Multiple | 0 | Reliability | 0 |
| Income | 2490 | OnlineSecurity | 0 | Options | 0 |
| Marital | 0 | OnlineBackup | 0 | Respectfulness | 0 |
| Gender | 0 | DeviceProtection | 0 | Courteous | 0 |
| Churn | 0 | TechSupport | 991 | Listening | 0 |

I divided the dataset in two groups: columns with numerical attributes (Numerical Att) and columns with categorical attributes (Categorial Att):

```
#Created a list for each attribute

all_att = list(df_clean.columns)
num_att = list(df_clean._get_numeric_data().columns)
cat_att = list(df_clean.select_dtypes(include=['object']).columns)
print('Numerical Att:' + '\n {}'.format(num_att))
print('Categorical Att:' + '\n {}'.format(cat_att))
#Removing Churn and CustomerID
id_col = ['customerID']
target = ['Churn']
cat_att = [x for x in cat_att if x not in id_col + target]

print('Numerical Attributes:' + '\n {}'.format(num_att))
print('Categorical Attributes:' + '\n {}'.format(cat_att))
```

**Numerical Att:**
Numerical Attributes:
['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Responses', 'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness', 'Courteous', 'Listening']

**Categorical Att:**
Categorical Attributes:
['Customer_id', 'Area', 'Timezone', 'Job', 'Education', 'Employment', 'Marital', 'Gender', 'Churn', 'Techie', 'Contract', 'Port_modem', 'Tablet', 'InternetService', 'Phone', 'Multiple', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'PaperlessBilling', 'PaymentMethod']

```
#Standard deviation of numeric columns in my dataset
data_std = df_clean.std()
print(data_std)
```

Standard Deviation for the numerical type:

| | |
|---|---|
| Children | 1.925971 |
| Age | 18.003457 |
| Income | 24747.863148 |
| Churn | 0.441355 |
| Outage_sec_perweek | 7.025921 |
| Email | 3.025898 |
| Contacts | 0.988466 |
| Yearly_equip_failure | 0.635953 |
| Port_modem | 0.499749 |
| Tablet | 0.457887 |
| Multiple | 0.498486 |
| OnlineSecurity | 0.479317 |
| OnlineBackup | 0.497579 |
| DeviceProtection | 0.496241 |
| StreamingTV | 0.499975 |
| StreamingMovies | 0.499904 |
| PaperlessBilling | 0.492184 |
| Tenure | 25.188194 |
| MonthlyCharge | 43.335473 |
| Bandwidth_GB_Year | 2072.726654 |
| Responses | 1.037797 |
| Fixes | 1.034641 |
| Replacements | 1.027977 |
| Reliability | 1.025816 |
| Options | 1.024819 |

```
Respectfulness      1.033586
Courteous           1.028502
Listening           1.028633
dtype: float64
```

**PART D2 – JUSTIFICATION OF MITIGATION METHODS**

In any data analytics project, missing data will always exist. Since we don't have the chance to talk to the person who collected the data and ask why some is missing, I am going to fill it in with a value that would make sense, like median or mean. Since the amount of missing data is not that significant that wouldn't allow us to go further with our analysis, I am going to replace it with the median of each column:

```
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)
```

**PART D3 – SUMMARY OF THE OUTCOMES**

Dataset is clean at this point, I only left variables with stronger relationship with CHURN like tenure, monthly charge, bandwidth usage and 8 responses for survey.

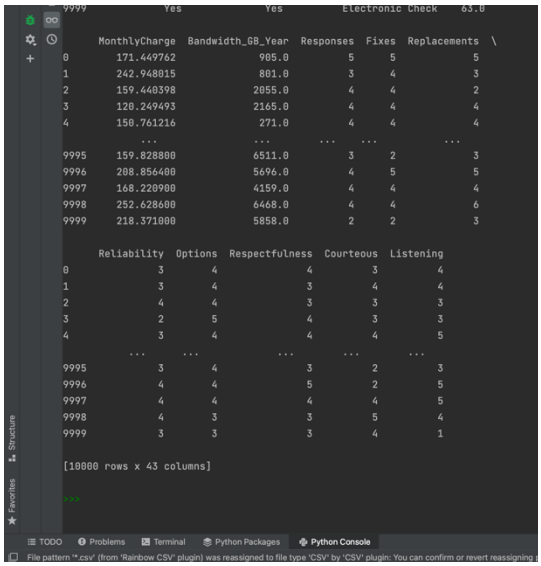
**PART D4 – MITIGATION CODE**

The code is throughout the document

**PART D5 – CLEAN DATASET**

Extract clean dataset:

```
#Extracting the clean dataset
df_clean.to_csv('churn_clean.csv')
churn_user = pd.read_csv('churn_clean.csv')
print(churn_user)
result = churn_user.isna().any()
```




**Picture 48:** Clean dataset extraction

All numeric columns now bring the FALSE value for missing data:

```
print(result)
```


**Picture 49:** No more missing values for the numeric columns

Every numerical column had the missing data replaced by its median. Machine learning algorithms can usually have only numerical data as independent variables. Because of this fact we will have to encode the categorical data with appropriate numerical values. Columns with **two** categorical values will be treated.

**PART D6 - LIMITATIONS**

One clear limitation here is not being able to talk to the person/department who collected the data. We can't ask them why some data is missing.

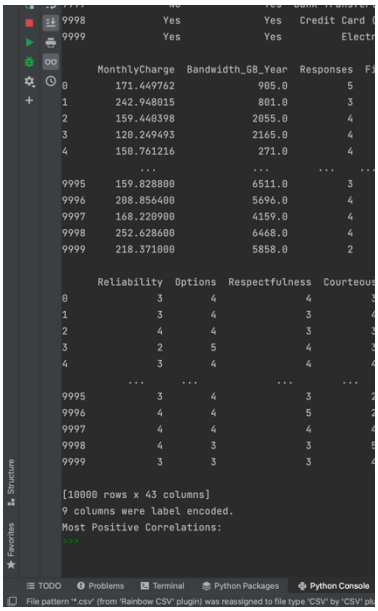
**PART D7 – IMPACT OF THE LIMITATIONS**

In this project I had to input some missing data on tenure, for example. I used the median to help the company to create a model to predict which customers are more likely to churn. It doesn't mean that this "new" median value is correct, and actually it could be very different from the real value. There is no reason for the tenure input for customers to be missing on our dataset. So the process of collecting data should change to a better collection method to avoid missing data that shouldn't be empty.

**PART E – PRINCIPAL COMPONENTS**

Encoder for columns with 2 or less categorical attributes

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

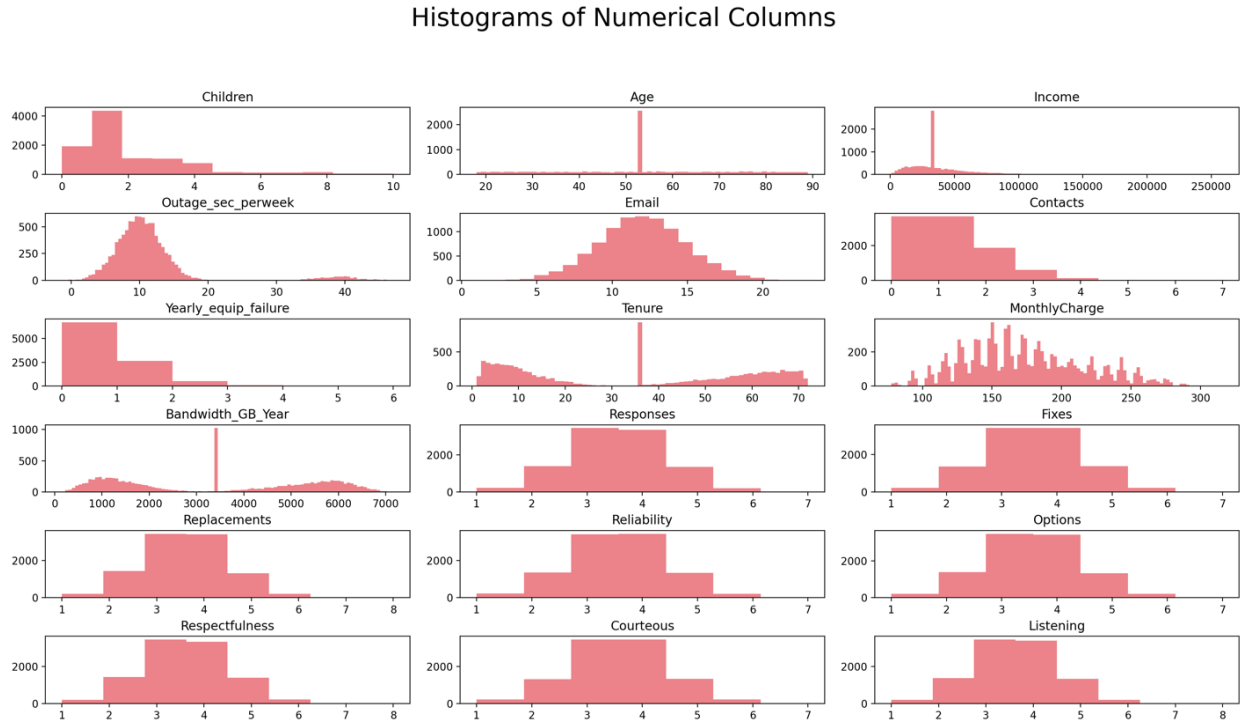



**Picture 50:** 9 columns were label encoded

Next step is to plot histograms for the numerical data: ['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Responses', 'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness', 'Courteous', 'Listening']

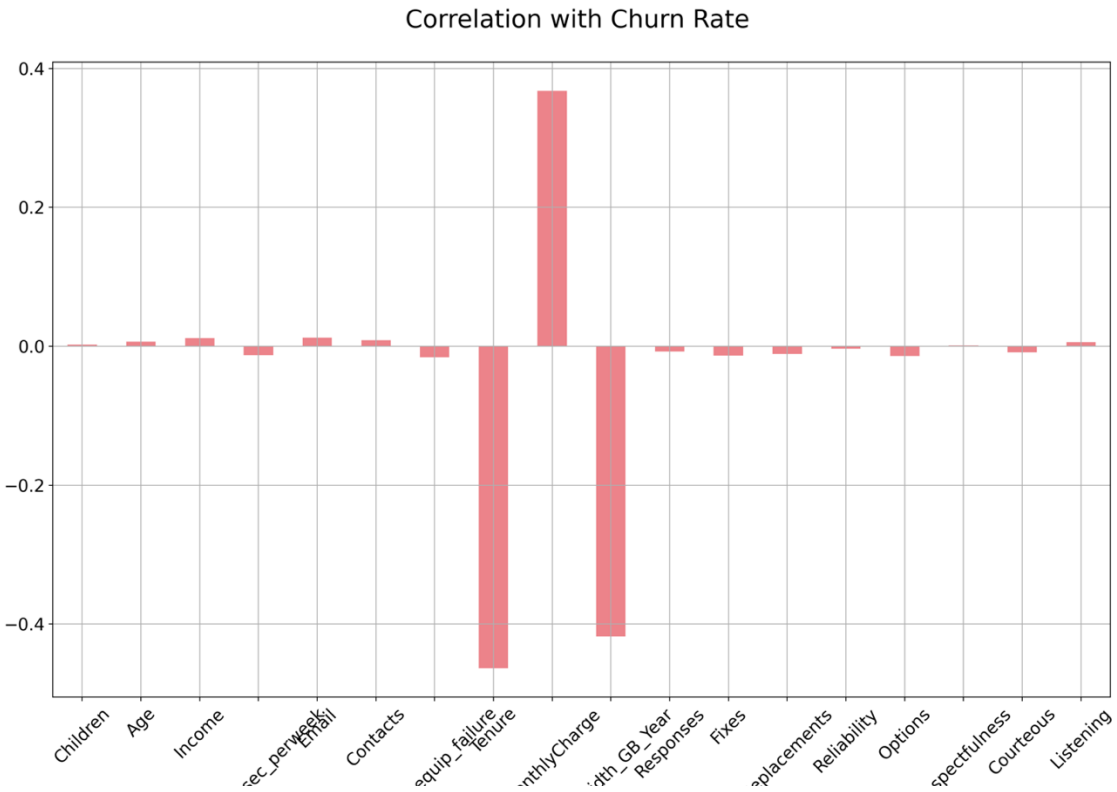
```
# Creating histograms of Numerical Data
dataset = df_clean[['Children', 'Age', 'Income', 'Outage_sec_perweek',
'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge',
'Bandwidth_GB_Year', 'Responses', 'Fixes', 'Replacements', 'Reliability',
'Options', 'Respectfulness', 'Courteous', 'Listening']]
# Histogram:
fig = plt.figure(figsize=(15, 12))
plt.suptitle('Histograms of Numerical Columns\n',
horizontalalignment="center", fontstyle="normal", fontsize=24,
fontfamily="sans-serif")
for i in range(dataset.shape[1]):
    plt.subplot(6, 3, i + 1)
    f = plt.gca()
    f.set_title(dataset.columns.values[i])
    vals = np.size(dataset.iloc[:, i].unique())
    if vals >= 100:
        vals = 100
    plt.hist(dataset.iloc[:, i], bins=vals, color='#ec838a')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

From the histograms we can see that the majority of variables don’t have a normal distribution.




**Picture 51:** Histograms of Numerical Column

```
#Independent Variables and Churn Rate Positive and Negative Correlations
correlations = dataset.corrwith(df_clean.Churn)
correlations = correlations[correlations!=1]
positive_correlations = correlations[correlations >0].sort_values(ascending =
False)
negative_correlations =correlations[correlations<0].sort_values(ascending =
False)
print('Most Positive Correlations: \n', positive_correlations)
print('\nMost Negative Correlations: \n', negative_correlations)

#Creating Correlations
correlations = dataset.corrwith(df_clean.Churn)
correlations = correlations[correlations!=1]
correlations.plot.bar(figsize = (18, 10), fontsize = 15, color = '#ec838a',
rot = 45, grid = True)
plt.title('Correlation with Churn Rate \n', horizontalalignment="center",
fontstyle = "normal", fontsize = "22", fontfamily = "sans-serif")
```




**Picture 52:** Independent Variables and Churn Rate

From this plot above we can clearly see that Tenure and Bandwidth_GB_Year have a negative correlation with Churn: meaning higher the tenure and amount of data usage lower the churn. Monthly charge has a positive correlation with churn: higher the monthly charge, higher the churn.

```
print(corr['Churn'].sort_values(ascending=False))
```

| | |
|---|---|
| Churn | 1.000000 |
| MonthlyCharge | 0.367495 |
| StreamingMovies | 0.289262 |
| StreamingTV | 0.230151 |
| Multiple | 0.131771 |
| DeviceProtection | 0.056489 |
| OnlineBackup | 0.050508 |
| Email | 0.012326 |
| Income | 0.011542 |
| Contacts | 0.008567 |
| Port_modem | 0.008157 |
| PaperlessBilling | 0.007030 |
| Age | 0.006131 |

| | |
|---|---|
| Listening | 0.005653 |
| Children | 0.002397 |
| Respectfulness | 0.001130 |
| Tablet | -0.002779 |
| Reliability | -0.003396 |
| Responses | -0.007341 |
| Courteous | -0.008851 |
| Replacements | -0.011143 |
| Outage_sec_perweek | -0.012813 |
| Fixes | -0.013253 |
| OnlineSecurity | -0.013540 |
| Options | -0.013971 |
| Yearly_equip_failure | -0.015927 |
| Bandwidth_GB_Year | -0.418047 |
| Tenure | -0.463720 |

Name: Churn, dtype: float64

Most Positive Correlations:

| | |
|---|---|
| **MonthlyCharge** | **0.367495** |
| Email | 0.012326 |
| Income | 0.011542 |
| Contacts | 0.008567 |
| Age | 0.006131 |
| Listening | 0.005653 |
| Children | 0.002397 |
| Respectfulness | 0.001130 |

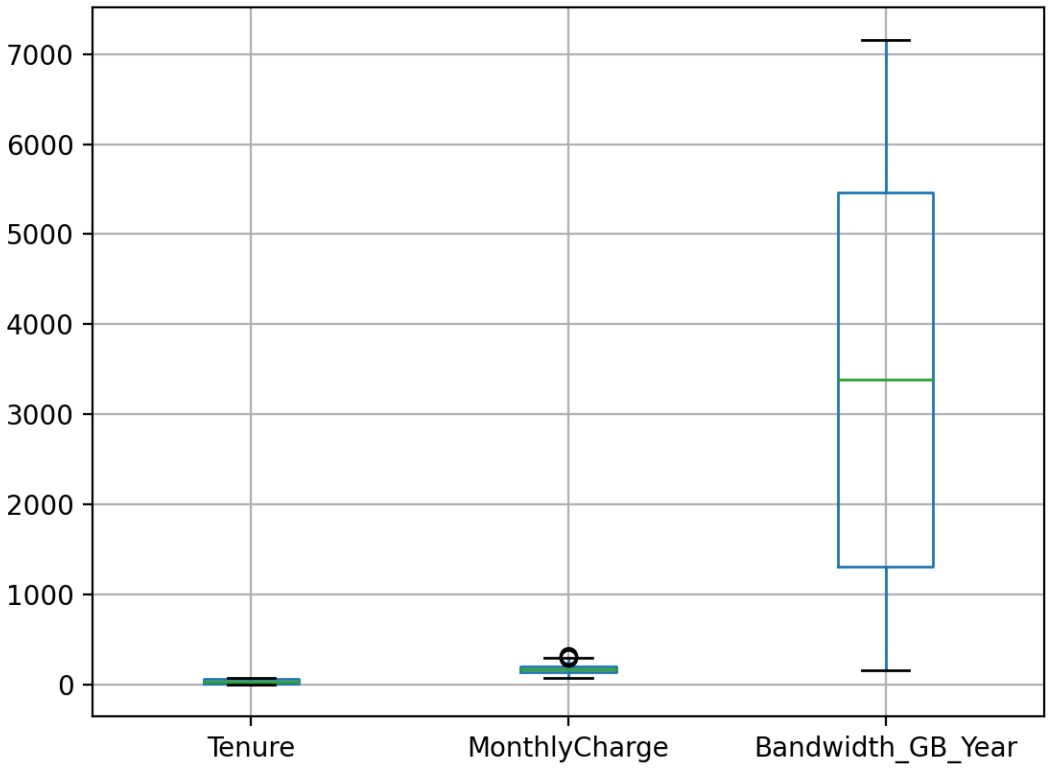
dtype: float64

Most Negative Correlations:

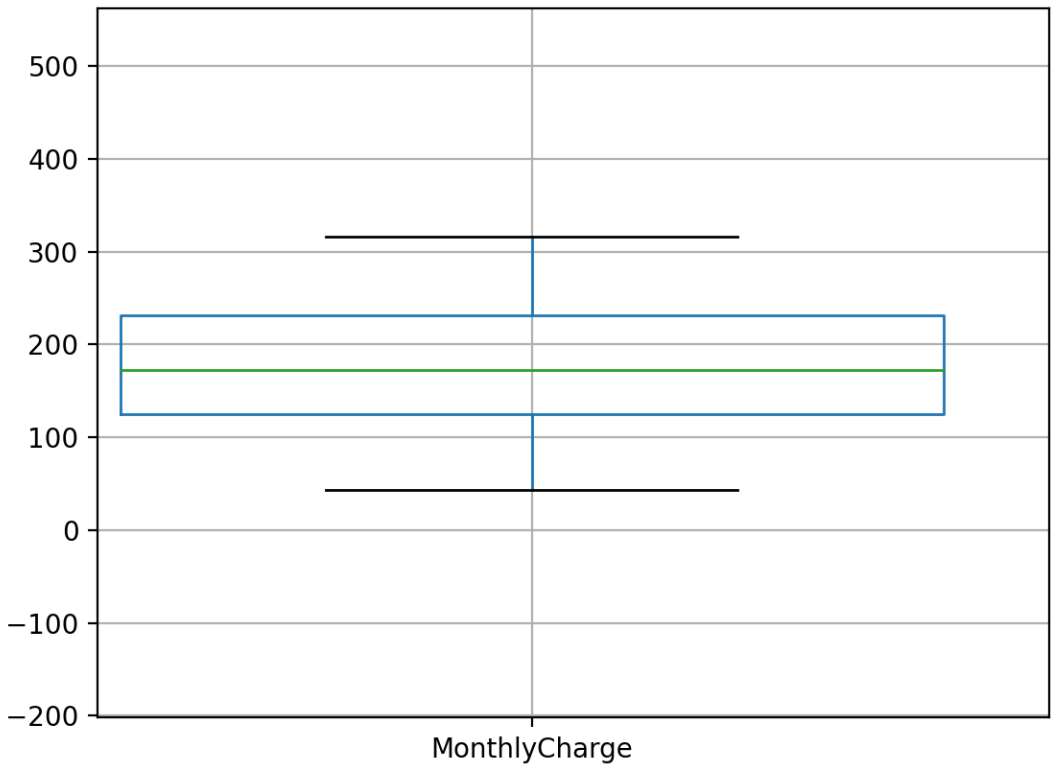
| | |
|---|---|
| Reliability | -0.003396 |
| Responses | -0.007341 |
| Courteous | -0.008851 |
| Replacements | -0.011143 |
| Outage_sec_perweek | -0.012813 |
| Fixes | -0.013253 |
| Options | -0.013971 |
| Yearly_equip_failure | -0.015927 |
| **Bandwidth_GB_Year** | **-0.418047** |
| **Tenure** | **-0.463720** |

dtype: float64

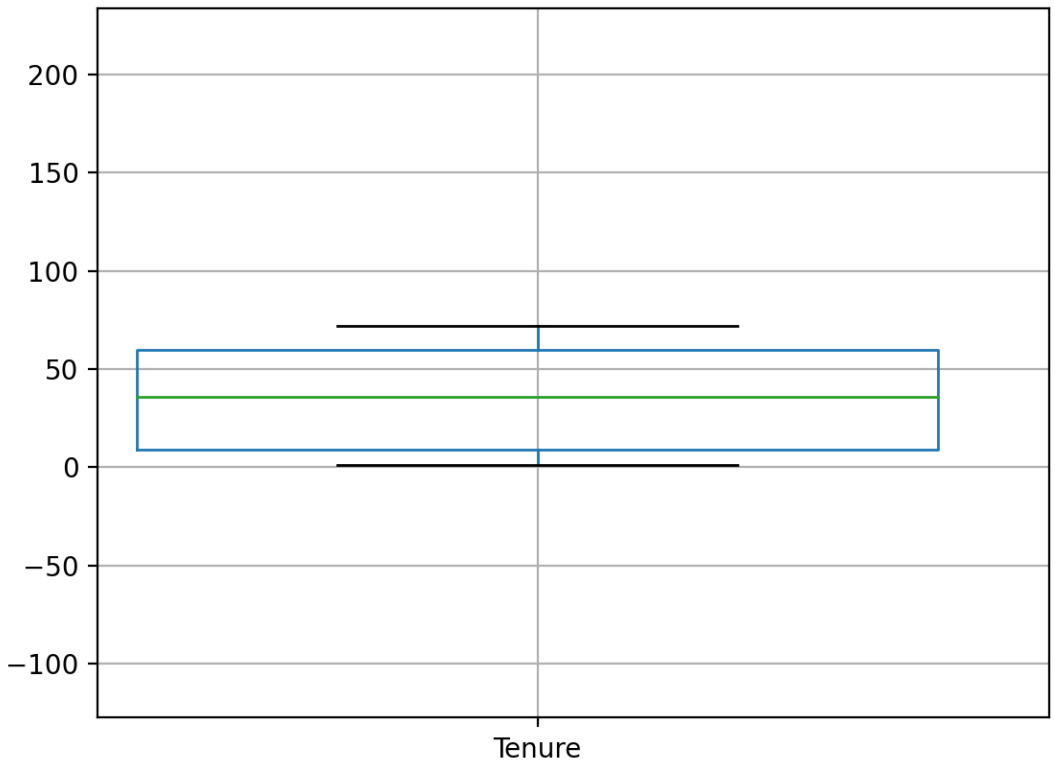
```
df_clean.boxplot(['Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year'])
plt.savefig('churn_boxplots.png')
plt.show()
```




**Picture 53:** Boxplot of “Tenure”, “MonthlyCharge” and “Bandwidth”




**Picture 54:** MonthlyCharge Boxplot

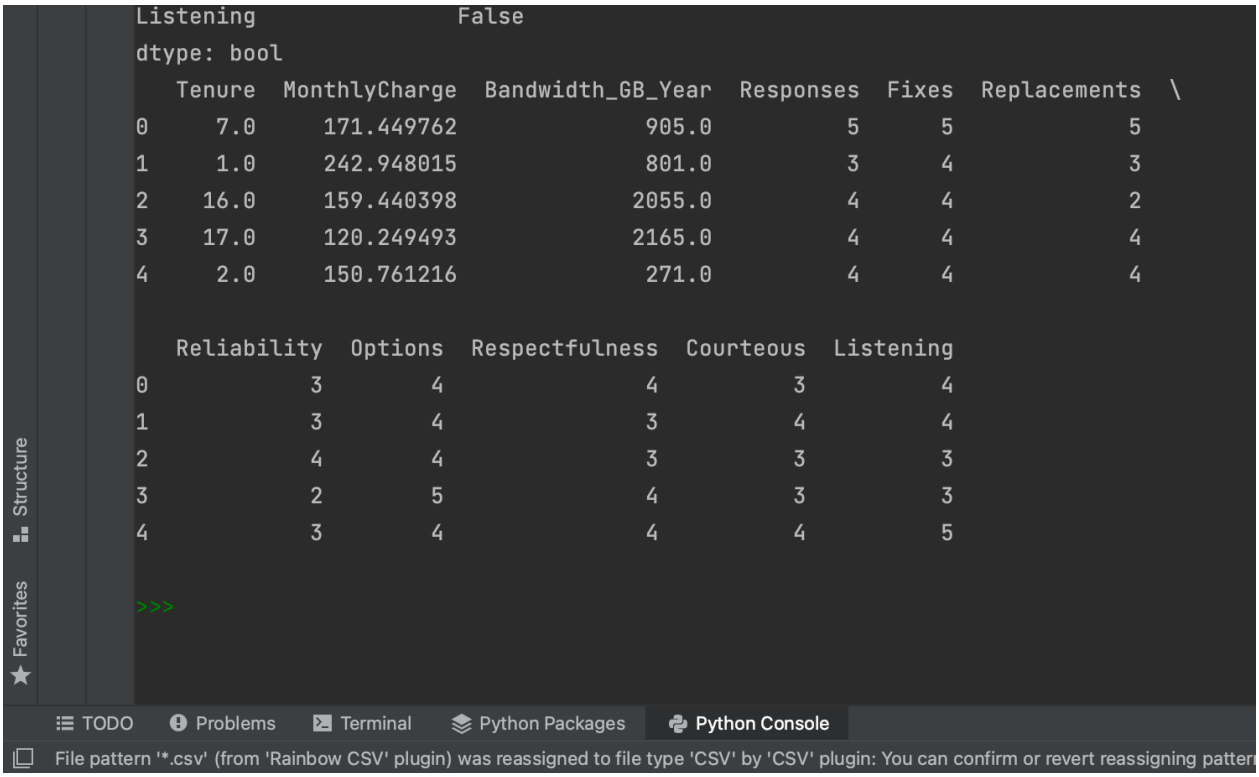



**Picture 55:** Boxplot Tenure

I am going to analyze all rows from “Tenure” and after.

```
# Extract Clean dataset
df_stats.to_csv('churn_clean.csv')

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

# Slice off all but last eleven service realted variables
data = churn_user.loc[:, 'Tenure':'Listening']
print(data.head())
```

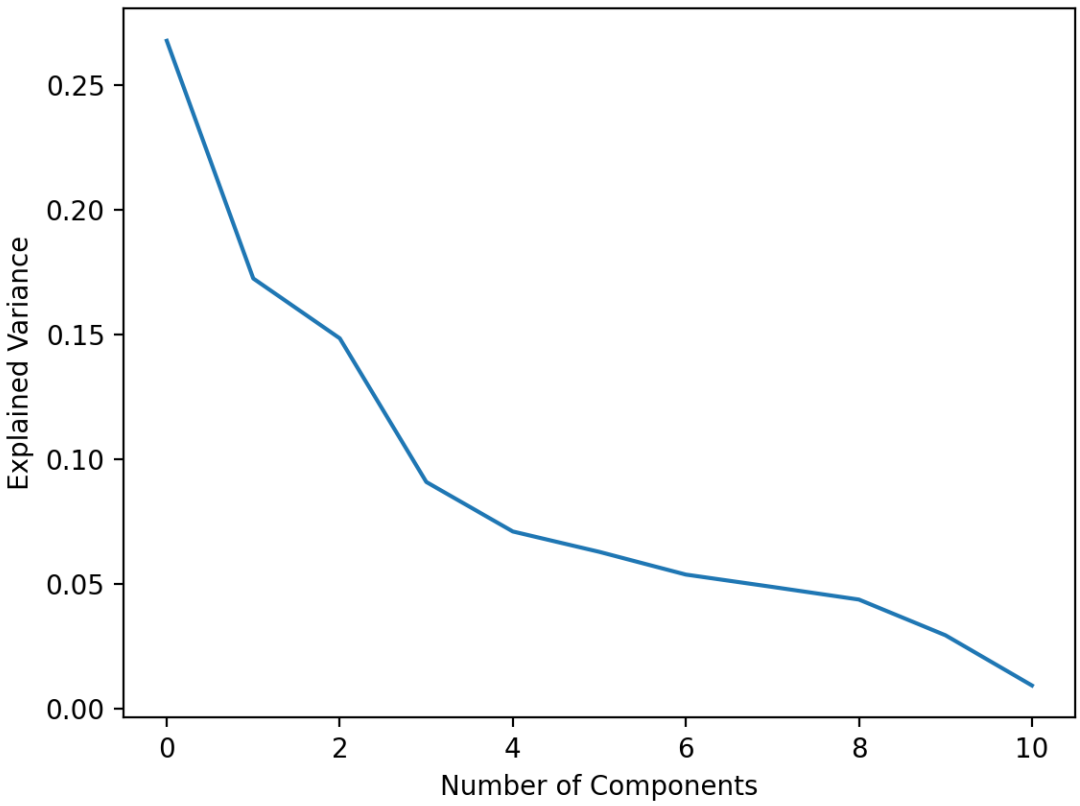
**PCA:** Principal Component Analysis is a widely used technique to reduce the dimension of your feature space. When we have a system with 10 independent variables, for example, we create 10 “new” variables that are a combination of each of the 10 old ones. We create the new variables in an specific way and order by how well they predict our dependent variable. These new 10 variables carry the most valuable parts of our old variables since the new ones are a combination of the old ones. And for that reason, it’s not a problem when we drop some of the new created variables. A huge benefit of using PCA is that the new variables are independent of one another. [4]




**Picture 56:** Dataset with only last few columns (the ones that presented the bigger relation with Churn Rate)

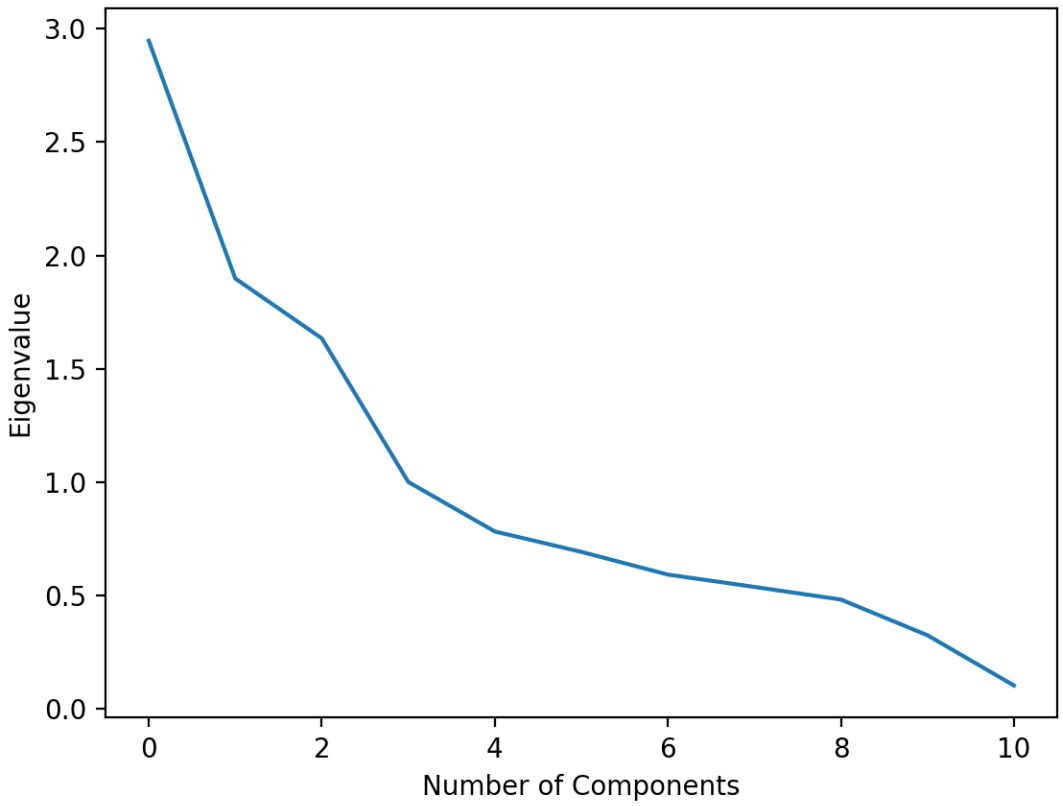
Since PCA seeks to maximize the variance of each component, the first step of PCA is to normalize the data. Our dataset has different kinds of variables with different units. So normalization is a very important first step of this process.

```
######PCA ANALYSIS#########
# Normalize the data
churn_normalized = (data - data.mean()) / data.std()
pca = PCA(n_components = data.shape[1])
churn_numeric = data[['Tenure', 'MonthlyCharge',
'Bandwidth_GB_Year','Responses','Fixes', 'Replacements', 'Reliability',
'Options', 'Respectfulness', 'Courteous', 'Listening']]
pcs_names = []
for i, col in enumerate(churn_numeric.columns):
    pcs_names.append('PC' + str(i + 1))
print(pcs_names)

pca.fit(churn_normalized)
churn_pca = pd.DataFrame(pca.transform(churn_normalized), columns =
pcs_names)

plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.show()
```




**Picture 57:** Number of Components vs Variance

```
#Extract the eigenvalues
cov_matrix = np.dot(churn_normalized.T, churn_normalized) / data.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for
eigenvector in pca.components_]

# Plot the eigenvalues
plt.plot(eigenvalues)
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalue')
plt.show()
```




**Picture 58:** Number of Components vs Eigenvalues

**PART E2 – CRITERIA USED**

```
print(eigenvalues)
```

[2.9468883332915676, 1.8985243408639927, 1.634492017682998, 1.0010368713053779, 0.7827244979058301, 0.6928225371388447, 0.5926768068225855, 0.5383031903870839, 0.48256121993367873, 0.3251427179121127, 0.10372746675593421]

Eigenvalues are always positive numbers. Therefore, the importance of each value decreases as it approaches 0. An eigenvalue greater than 1, its considered better than average at explaining variance in the dataset. The larger its value, the better the given principal component is at explaining variance [3]

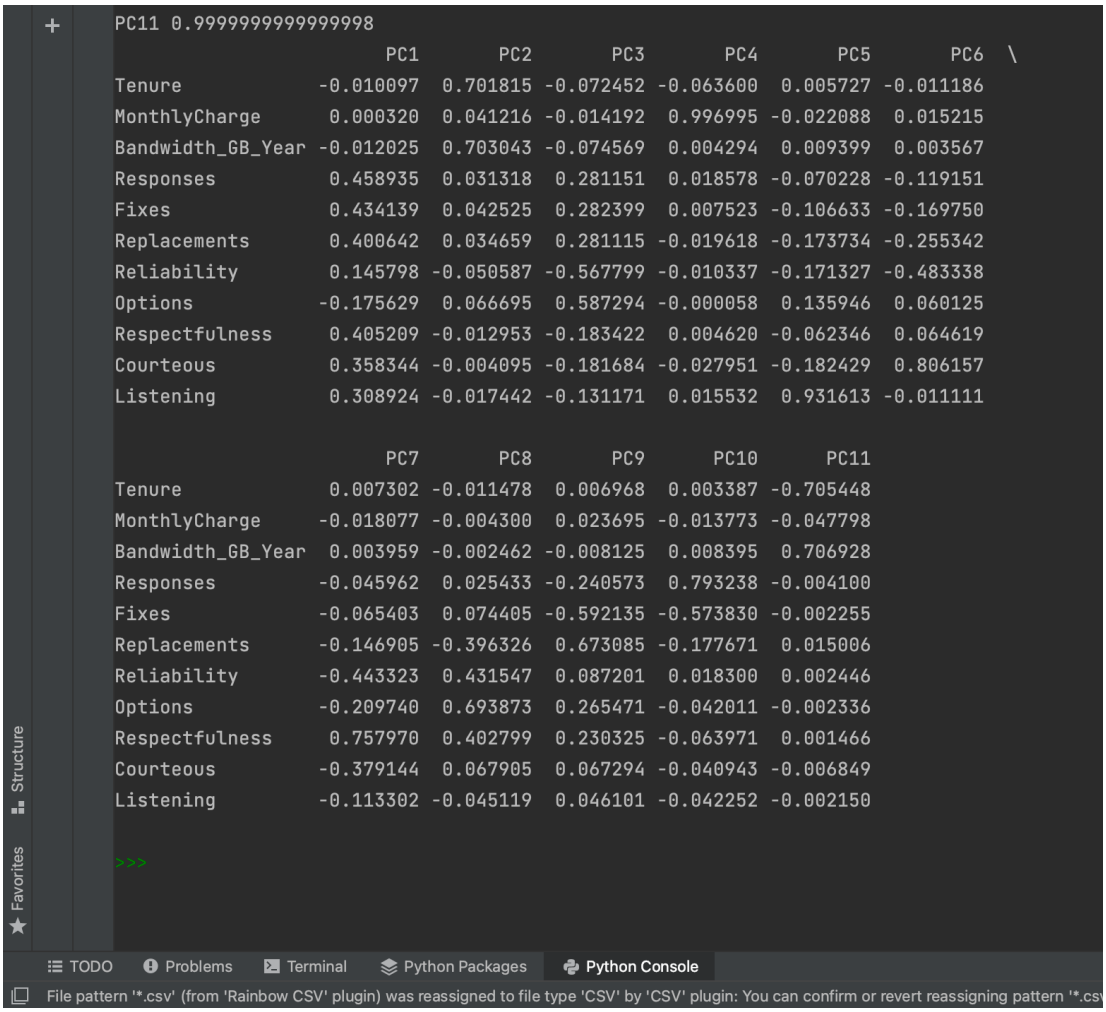
To select the number of principal component to extract from the analysis, we must see how many eigenvalues are greater than 1, I used the KAISER CRITERION. In this case we have 4 above 1. So, 4 components could be used to reduce the original variables and still maintain the maximum level of information from the original data.

```
#Select the fewest components
for pc, var in zip(pcs_names, np.cumsum(pca.explained_variance_ratio_)):
    print(pc, var)
```

PC1 0.2679257319633391
PC2 0.4405361148983589
PC3 0.5891411588284788
PC4 0.6801536120106493
PC5 0.7513175009364353
PC6 0.8143076669656604
PC7 0.8681927651866267
PC8 0.9171343130129624
PC9 0.9610079021840322
PC10 0.9905692872236372
PC11 0.9999999999999998

```
#Creating Rotation
rotation = pd.DataFrame(pca.components_.T, columns = pcs_names, index =
churn_numeric.columns)
print(rotation)
```

Tenure, Monthly charge and Bandwidth have the most important components when predicting churn rate. I used a scree plot eigenvalue vs number of components to analyze the curve. I think an explanation of **86%** variance at 7 components is a good stop. Since that happens at PC7, the 7 first principal components will be used to describe the data.




**Picture 59:** Rotation Matrix

```
churn_reduced = churn_pca.iloc[ : , 0:4]
print(churn_reduced)
```

```
Respectfulness     0.757970  0.402799  0.230325 -0.063971  0.001466
Courteous         -0.379144  0.067905  0.067294 -0.040943 -0.006849
Listening         -0.113302 -0.045119  0.046101 -0.042252 -0.002150
               PC1       PC2       PC3       PC4
0         1.923291 -1.415229  1.903106  0.041909
1        -0.200322 -1.710019  0.539808  1.665210
2        -0.668355 -0.978071  0.227002 -0.252295
3         0.046202 -0.731563  2.282558 -1.169836
4         1.325996 -1.915198  0.825584 -0.440915
            ...       ...       ...       ...
9995     -2.097281  1.957219  0.103812 -0.379701
9996      1.917986  1.644993  0.610413  0.794847
9997      1.432067  0.318193  0.028682 -0.127101
9998      2.012166  2.184716 -0.080515  1.646372
9999     -2.265721  1.582787 -0.819727  0.876021

[10000 rows x 4 columns]
```

**Picture 60:** Extract reduced dataset and print 4 components

**PART E3 – BENEFITS**

The analysis suggests that the tenure, monthly charge and bandwidth have the biggest relationships with churn. In this case, the company will be able to create a strategy plan to retain more customers and decrease churn to a much smaller number than 26.5%. Also to accelerate the machine learning algorithm, instead of using the entire dataset, the company will be able to use only the 7 first principal components to train the algorithm. That will be a much faster process.

**PART F – VIDEO**

Video is uploaded in separate:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0104977b-5431-4045-b786-ad720148734b

**PART G – SOURCES FOR THIRD PARTY CODE**

Moreira Marques, Igor Jose. (202, Sept 29), Using Machine Learning for Customer Churn Prediction, https://www.linkedin.com/pulse/using-machine-learning-customer-churn-prediction-moreira-marques

Sree. (2020, October 26). Predict Customer Churn in Python.
https://towardsdatascience.com/predict-customer-churn-in-python-e8cd6d3aaa7

**PART H – SOURCES**

[1] Pycharn Tutorial https://www.jetbrains.com/help/pycharm/quick-start-guide.html

[2] Panda Tutorial https://mode.com/python-tutorial/libraries/pandas

[3] Larose, C. D. & Larose, D. T. (2019). Data Science: Using Python and R. John Wiley & Sons, Inc.

[4] Brems, Matt. (2017, April 17) A One-Stop Shop for Principal Component Analysis https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis 5582fb7e0a9c

[5] Zaric, Drazen. (2019, April 15) Better Heatmaps and Correlation Matrix Plots in Python https://towardsdatascience.com/better-heatmaps-and-correlation-matrix-plots-in-python-41445d0f2bec

[6] Brownlee, Jason. (2017, March 20) How to Handle Missing Data with Python https://machinelearningmastery.com/handle-missing-data-python/