Darian Gurrola

D206 Performance Assessment

Course Instructor: Keiona Middleton

Part 1:

A.

What are the most relevant factors in predicting customer churn?

B.

The table below provides the name, data type, description, and a value example of each variable in the churn dataset. Data from the "churn_raw_data.csv" file was imported into the "df_churn" dataframe using the "read_csv()" function from the pandas library.

Variable Name	Data Type	Description	Value Example
CaseOrder	Qualitative	Index number used to order customer data	5555
Customer_id	Qualitative	Customer identification number	L387243
Interaction	Qualitative	Unique identification number for customer interaction	1836388d-6afd-4c7d-9e81-e5e2fca0169f
City	Qualitative	City listed on billing address	Killeen
State	Qualitative	State listed on billing address	TX
County	Qualitative	County listed on billing address	Bell
Zip	Qualitative	ZIP code listed on billing address	76541
Lat	Qualitative	Latitude of customer billing address	31.11409
Lng	Qualitative	Longitude of customer billing address	-97.72892

Variable Name	Data Type	Description	Value Example
Population	Quantitative	Population within a mile radius of billing address	18198
Area	Qualitative	Type of community the customer lives in	Rural
TimeZone	Qualitative	Time zone of customer address	America/Chicago
Job	Qualitative	Occupation of customer	Arts administrator
Children	Quantitative	Number of children reported by customer during sign-up	NA
Age	Quantitative	Age of customer	49
Education	Qualitative	Formal education level of customer	Regular High School Diploma
Employment	Qualitative	Employment status of customer	Full Time
Income	Quantitative	Yearly income of customer	20541.47
Marital	Qualitative	Marital status of customer	Divorced
Gender	Qualitative	Self-reported gender of customer	Female
Churn	Qualitative	Indicates that customer cancelled service within the last month (Y/N)	No
Outage_sec_perweek	Quantitative	Average duration of service outages in customer vicinity. Measured in seconds per week.	7.418861
Email	Quantitative	Quantity of emails sent to the customer over the past year	12
Contacts	Quantitative	Number of times customer has requested assistance	1
Yearly_equip_failure	Quantitative	Number of equipment failures reported by customer over the last year	0
Techie	Qualitative	Indicates that the customer considers themself to be tech-savvy	No

Variable Name	Data Type	Description	Value Example
		(Y/N)	
Contract	Qualitative	Duration of customer contract (Monthly, One-year, Two-Year)	Month-to-month
Port_modem	Qualitative	Indicates that the customer has a portable modem (Y/N)	Yes
Tablet	Qualitative	Indicates that the customer owns a tablet (Y/N)	No
InternetService	Qualitative	Type of internet service used by customer	DSL
Phone	Qualitative	Indicates that the customer has phone service (Y/N)	Yes
Multiple	Qualitative	Indicates that the customer has more than one phone line (Y/N)	No
OnlineSecurity	Qualitative	Indicates that the customer has supplemental online security (Y/N)	Yes
OnlineBackup	Qualitative	Indicates that the customer has supplemental online backup (Y/N)	No
DeviceProtection	Qualitative	Indicates that the customer has supplemental device protection (Y/N)	No
TechSupport	Qualitative	Indicates that the customer has technical support coverage (Y/N)	Yes
StreamingTV	Qualitative	Indicates that the customer has television streaming (Y/N)	Yes
StreamingMovies	Qualitative	Indicates that the customer has movie streaming (Y/N)	No
PaperlessBilling	Qualitative	Indicates that the customer is signed up for paperless billing (Y/N)	Yes
PaymentMethod	Qualitative	Payment method used by customer	Electronic Check
Tenure	Quantitative	Length of customer relationship in months	48.35393
MonthlyCharge	Quantitative	Average monthly bill for the customer	130.1858
Bandwidth_GB_Year	Quantitative	Yearly amount of data used in gigabytes	4711.281
Item1	Qualitative	Measures customer rating of timely responses on a scale of 1 to 8.	4

Variable Name	Data Type	Description	Value Example
Item2	Qualitative	Measures customer rating of timely fixes on a scale of 1 to 8.	4
Item3	Qualitative	Measures customer rating of timely replacements on a scale of 1 to 8.	2
Item4	Qualitative	Measures customer rating of reliability on a scale of 1 to 8.	3
Item5	Qualitative	Measures customer rating of service options on a scale of 1 to 8.	5
Item6	Qualitative	Measures customer rating of respectful responses on a scale of 1 to 8.	2
Item7	Qualitative	Measures customer rating of courteous exchanges on a scale of 1 to 8.	4
Item8	Qualitative	Measures customer rating of active listening on a scale of 1 to 8.	3

C1.

The first step in detecting data quality issues in the dataset is to identify duplicates. For this step, I used will use the command, “df_churn.duplicated.value_counts()”. The functions "duplicated()" and "value_counts()" will be imported from the pandas library. These functions will be used together to find the total number of duplicate rows in the df_churn dataframe by counting the number of rows that are "TRUE"(duplicates) or "FALSE"(non-duplicates).

The next step is to identify missing values in df_churn. I will use the command “df_churn.isnull().sum()” to calculate the number of missing values in each variable. The function "isnull()" will be imported from the pandas library.

The next step in assessing data quality issues is to look for outliers in the dataset. I plan to do this by generating a boxplot and a matching summary for each quantitative variable. To generate a boxplot I will use the “sns.boxplot()” function from the seaborn library.

To generate a summary for the boxplot values, I plan to build a function called “boxplot_info()”. This function will accept a column from df_churn as an input. I plan to avoid NaN calculation errors by using the "dropna()" pandas function on the input. I will use the “quantile()” function from pandas to find the values of the first(Q1) and third(Q3) quartiles of the boxplot. I will then use these values to calculate the interquartile range(IQR) by calculating the difference between the first and third quartiles. To calculate the values of the whiskers, I will subtract 1.5 * IQR from Q1, and add 1.5 * IQR to Q3. The final step will be to calculate the number of outliers. I will do this by calculating the number of values greater than the upper whisker and less than the lower whisker. Generating a boxplot with information will make it easier for me to determine how to treat outliers.

Lastly, I plan to re-express the “Education” column by first creating a dictionary, "dict_edu". This dictionary will store each category in "Education" and its corresponding value, which is the

number of years of schooling. I will then use the “replace()” function to swap the original categories in “Education” with the values stored in “dict_edu”.

C2.

I will use the “df_churn.duplicated().value_counts()” to detect duplicate rows in df_churn because it provides a clear output of the number of duplicates in the dataset. The “duplicated()” function denotes if a value is a duplicate using a boolean output. The “value_counts()” function then counts the number of rows that are "TRUE" and "FALSE".

To detect missing values, I will use the command “df_churn.isnull().sum()” because it identifies the number of missing values and the variables to which they belong. The "isnull()" function determines if a value is missing using "TRUE" or "FALSE", and the sum() function groups the number of null values by column. The output will clearly identify which variables need to be properly treated for missing values.

To detect outliers in df_churn, I will use the seaborn “boxplot()” function because it provides a clear visual overview of outliers for each variable. The output should show the normal range of values for a given variable and indicate the values that fall outside of that range. I plan to create and use a “boxplot_info()” function because I want to calculate the number of outliers and the their range of values. The purpose of the function will be to supplement the boxplot and provide more detailed information on outliers.

The last step in the process is to re-express categorical variables in the dataset. I plan to create the dictionary, "dict_edu", and use the "replace()" function to quickly and effectively encode the various categories. “Education” is the only value I plan to re-express because of its ordinal nature. Each category in “Education” can be re-expressed using the number of years of schooling.

C3.

I chose to use python to clean the data because of my prior experience and familiarity with the language. One of the advantages of Python is its usability and relatively simple syntax. It is an extremely versatile language due to the wide variety of libraries that can be imported.

I used the numpy package's "np.where()" function to identify outliers in a quantitative variable and impute them with the median. I used pandas to import the original csv file and manipulate the data. I used matplotlib to generate histograms and analyze the distributions of the quantitative variables. I used seaborn to generate boxplots and visualize outliers within quantitative variables. Lastly, I used the sklearn package to conduct principal component analysis.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.decomposition import PCA
```

```
#Read and import 'churn_raw_data.csv' into to df_churn variable
df_churn = pd.read_csv('churn_raw_data.csv')

#Obtain general infomation about the df_churn dataframe
df_churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 52 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            10000 non-null  int64
 1   CaseOrder             10000 non-null  int64
 2   Customer_id           10000 non-null  object
 3   Interaction           10000 non-null  object
 4   City                  10000 non-null  object
 5   State                 10000 non-null  object
 6   County                10000 non-null  object
 7   Zip                   10000 non-null  int64
 8   Lat                   10000 non-null  float64
 9   Lng                   10000 non-null  float64
 10  Population            10000 non-null  int64
 11  Area                  10000 non-null  object
 12  Timezone              10000 non-null  object
 13  Job                   10000 non-null  object
 14  Children              7505 non-null   float64
 15  Age                   7525 non-null   float64
 16  Education             10000 non-null  object
 17  Employment            10000 non-null  object
 18  Income                7510 non-null   float64
 19  Marital               10000 non-null  object
 20  Gender                10000 non-null  object
 21  Churn                 10000 non-null  object
 22  Outage_sec_perweek    10000 non-null  float64
 23  Email                 10000 non-null  int64
 24  Contacts              10000 non-null  int64
 25  Yearly_equip_failure  10000 non-null  int64
 26  Techie                7523 non-null   object
 27  Contract              10000 non-null  object
 28  Port_modem            10000 non-null  object
 29  Tablet                10000 non-null  object
 30  InternetService       10000 non-null  object
 31  Phone                 8974 non-null   object
 32  Multiple              10000 non-null  object
 33  OnlineSecurity        10000 non-null  object
 34  OnlineBackup          10000 non-null  object
 35  DeviceProtection      10000 non-null  object
 36  TechSupport           9009 non-null   object
 37  StreamingTV           10000 non-null  object
 38  StreamingMovies       10000 non-null  object
```

```
 39  PaperlessBilling      10000 non-null  object
 40  PaymentMethod         10000 non-null  object
 41  Tenure                9069 non-null   float64
 42  MonthlyCharge         10000 non-null  float64
 43  Bandwidth_GB_Year     8979 non-null   float64
 44  item1                 10000 non-null  int64
 45  item2                 10000 non-null  int64
 46  item3                 10000 non-null  int64
 47  item4                 10000 non-null  int64
 48  item5                 10000 non-null  int64
 49  item6                 10000 non-null  int64
 50  item7                 10000 non-null  int64
 51  item8                 10000 non-null  int64
dtypes: float64(9), int64(15), object(28)
memory usage: 4.0+ MB
```

C4.

See detection input code below. A copy of the executable code has been attached for review. Please note that the "%%capture" command was used to prevent the Jupyter Notebook from automatically generating an output. The output will be provided in section D1.

```
%%capture

#detect duplicate rows in df_churn
print(df_churn.duplicated().value_counts())

%%capture

#detect missing values in each variable of df_churn
df_churn.isnull().sum();

#Detect outliers in the following Quantitative variables
#Population
#Children
#Age
#Income
#Outage_sec_perweek
#Email
#Contacts
#Yearly_equip_failure
#Tenure
#MonthlyCharge
#Bandwidth_GB_Year

%%capture

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

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

    #generate q1 and q3 using pandas.DataFrame.quantile.  [In-text
citation: (Pandas Documentation)]
    q1 = input.quantile(0.25)
    print("Q1: " + str(q1))
    q3 = input.quantile(0.75)
    print("Q3: " + str(q3))

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

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

     #Find number of outliers outside of Q1 and Q3.  Print total
number of outliers in column.
    outliers_min = (input < whisker_lower).sum()
    print("Number of outliers lower than boxplot minimum: " +
str(outliers_min))
    outliers_max = (input > whisker_upper).sum()
    print("Number of outliers greater than boxplot maximum: " +
str(outliers_max))
    outliers_total = outliers_min + outliers_max
    print("Total number of Outliers: " + str(outliers_total))
    max_outlier = max(data)
    print("Highest Outlier: " + str(max_outlier))
    min_outlier = min(data)
    print("Lowest Outlier: " + str(min_outlier))

%%capture

#Generate boxplot for Population variable
population_boxplot = sns.boxplot(x="Population", data =
df_churn).set_title("Population")

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

%%capture

#Generate distribution for Population variable
plt.hist(df_churn['Population'])
plt.title("Population")
```

```
%%capture

#Generate boxplot for Children variable
children_boxplot = sns.boxplot(x="Children", data =
df_churn).set_title("Children")

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

%%capture

#Generate distribution for Children variable
plt.hist(df_churn['Children'])
plt.title("Children")

%%capture

#Generate boxplot for Age variable
age_boxplot = sns.boxplot(x="Age", data = df_churn).set_title("Age")

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

%%capture

#Generate distribution for Age variable
plt.hist(df_churn['Age'])
plt.title("Age")

%%capture

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

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

%%capture

#Generate distribution for Income variable
plt.hist(df_churn['Income'])
plt.title("Income")

%%capture

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

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

```
%%capture

#Generate distribution for Outage_sec_perweek variable
plt.hist(df_churn['Outage_sec_perweek'])
plt.title("Outage_sec_perweek")

%%capture

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

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

%%capture

#Generate distribution for Email variable
plt.hist(df_churn['Email'])
plt.title("Email")

%%capture

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

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

%%capture

#Generate distribution for Contacts variable
plt.hist(df_churn['Contacts'])
plt.title("Contacts")

%%capture

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

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

%%capture

#Generate distribution for Yearly_equip_failure variable
plt.hist(df_churn['Yearly_equip_failure'])
plt.title("Yearly_equip_failure")
```

```
%%capture

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

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

%%capture

#Generate distribution for Tenure variable
plt.hist(df_churn['Tenure'])
plt.title("Tenure")

%%capture

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

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

%%capture

#Generate distribution for MonthlyCharge variable
plt.hist(df_churn['MonthlyCharge'])
plt.title("MonthlyCharge")

%%capture

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

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

%%capture

#Generate distribution for Bandwidth_GB_Year variable
plt.hist(df_churn['Bandwidth_GB_Year'])
plt.title("Bandwidth_GB_Year")
```

PART 2:

D1.

As mentioned in C2, I used the “df_churn.duplicated().sum()” command to identify all duplicate rows in the churn dataset. The output of this command shows that all 10,000 rows in the dataset are listed as “FALSE”. This indicates that there are no duplicate rows in the dataset.

To look for missing values in df_churn, I used the function, “df_churn.isnull().sum()”. The output was a list of all the variables and the number of missing values associated with each one. There were several variables with missing values. According to the results, there were 2495 missing values in “Children”, 2475 missing values in “Age”, 2490 missing values in “Income”, 2477 missing values in “Techie”, 1026 missing values in “Phone”, 991 missing values in “TechSupport”, 931 missing values in “Tenure”, and 1021 missing values in “Bandwidth_GB_Year”.

After using the “sns.boxlot()” and “boxplot_info()” function on the quantitative variables, I was able to find the range and number of outliers for each one. Upon review, the variables “Population”, “Children”, “Income”, “Outage_sec_perweek”, “Email”, “Contacts”, “Yearly_equip_failure”, and “MonthlyCharge” variables all contained outliers. The variables “Age”, “Tenure”, and “Bandwidth_GB_Year” did not contain any outliers. I also used the "plt.hist()" function from the matplotlib on each of the variables in order to see how the distributions appear before any values are treated.

“Population” had 937 outliers between the range of 31813 and 111850. "Children" had 302 outliers between 8 and 10. “Outage_sec_perweek” had 539 outliers. 26 of those were between the range of -1.35 and 1.40, and 513 were between the range of 19.14 and 47.05. “Email” had 38 outliers. 23 of those outliers were between the range of 1 and 4 and 15 were between the range of 20 and 23. “Contacts” had 8 outliers between the range of 5 and 7. “Yearly_equip_failure” had 94 outliers between the range of 3 and 6. Lastly, “MonthlyCharge” had 5 outliers between the range of 297.83 and 315.88.

Please see the output of the detection code below.

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

```
False    10000
dtype: int64
```

```
#detect missing values in each variable of df_churn
df_churn.isnull().sum()
```

```
Unnamed: 0                0
CaseOrder                 0
Customer_id               0
Interaction               0
City                      0
State                     0
```

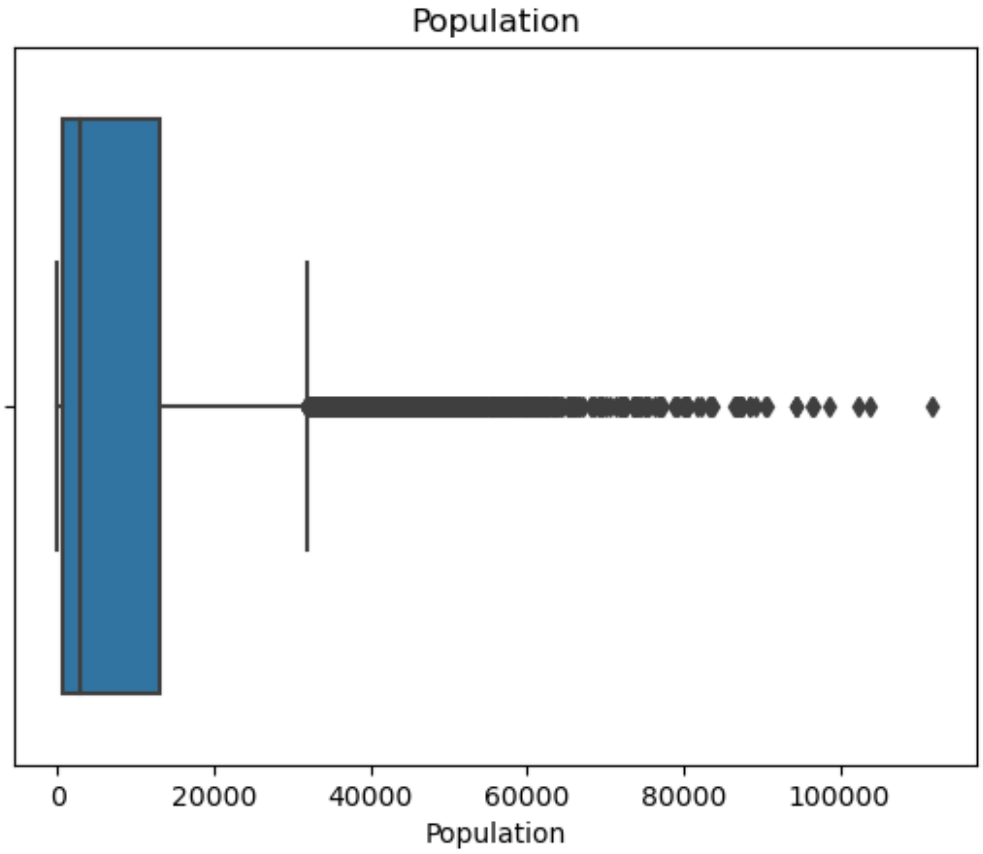
```
County                    0
Zip                       0
Lat                       0
Lng                       0
Population                0
Area                      0
Timezone                  0
Job                       0
Children               2495
Age                    2475
Education                 0
Employment                0
Income                 2490
Marital                   0
Gender                    0
Churn                     0
Outage_sec_perweek        0
Email                     0
Contacts                  0
Yearly_equip_failure      0
Techie                 2477
Contract                  0
Port_modem                0
Tablet                    0
InternetService           0
Phone                  1026
Multiple                  0
OnlineSecurity            0
OnlineBackup              0
DeviceProtection          0
TechSupport             991
StreamingTV               0
StreamingMovies           0
PaperlessBilling          0
PaymentMethod             0
Tenure                  931
MonthlyCharge             0
Bandwidth_GB_Year      1021
item1                     0
item2                     0
item3                     0
item4                     0
item5                     0
item6                     0
item7                     0
item8                     0
dtype: int64
```

```
#Detect outliers in the following Quantitative variables using
Boxplot() and Boxplot_info() functions
```

```
#Generate distrbutions for the following Quantitative variables using
plt.hist()

#Population
#Children
#Age
#Income
#Outage_sec_perweek
#Email
#Contacts
#Yearly_equip_failure
#Tenure
#MonthlyCharge
#Bandwidth_GB_Year

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

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

    #generate q1 and q3 using pandas.DataFrame.quantile.  [In-text
citation: (Pandas documentation)]
    q1 = input.quantile(0.25)
    print("Q1: " + str(q1))
    q3 = input.quantile(0.75)
    print("Q3: " + str(q3))

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

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

     #Find number of outliers outside of Q1 and Q3.  Print total
number of outliers in column.
    outliers_min = (input < whisker_lower).sum()
    print("Number of outliers lower than boxplot minimum: " +
str(outliers_min))
    outliers_max = (input > whisker_upper).sum()
    print("Number of outliers greater than boxplot maximum: " +
str(outliers_max))
    outliers_total = outliers_min + outliers_max
    print("Total number of Outliers: " + str(outliers_total))
    max_outlier = max(data)
```

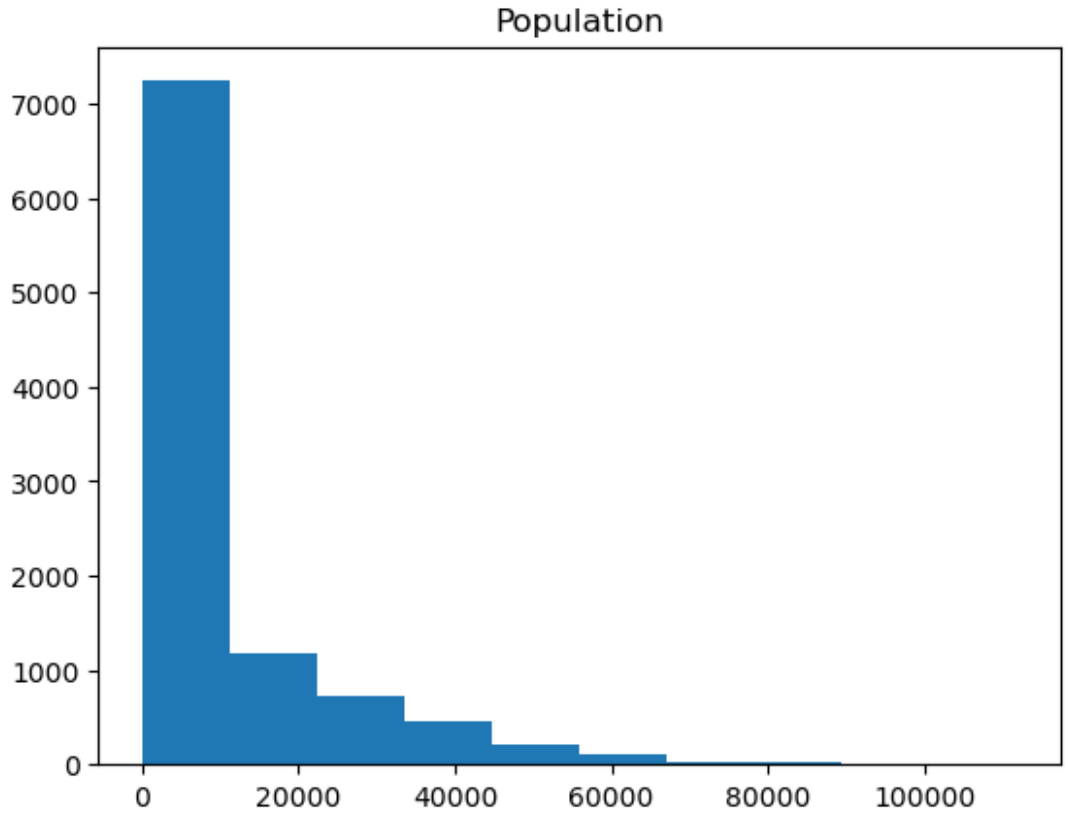
```
    print("Highest Outlier: " + str(max_outlier))
    min_outlier = min(data)
    print("Lowest Outlier: " + str(min_outlier))

#Generate boxplot for Population variable
population_boxplot = sns.boxplot(x="Population", data =
df_churn).set_title("Population")

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

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



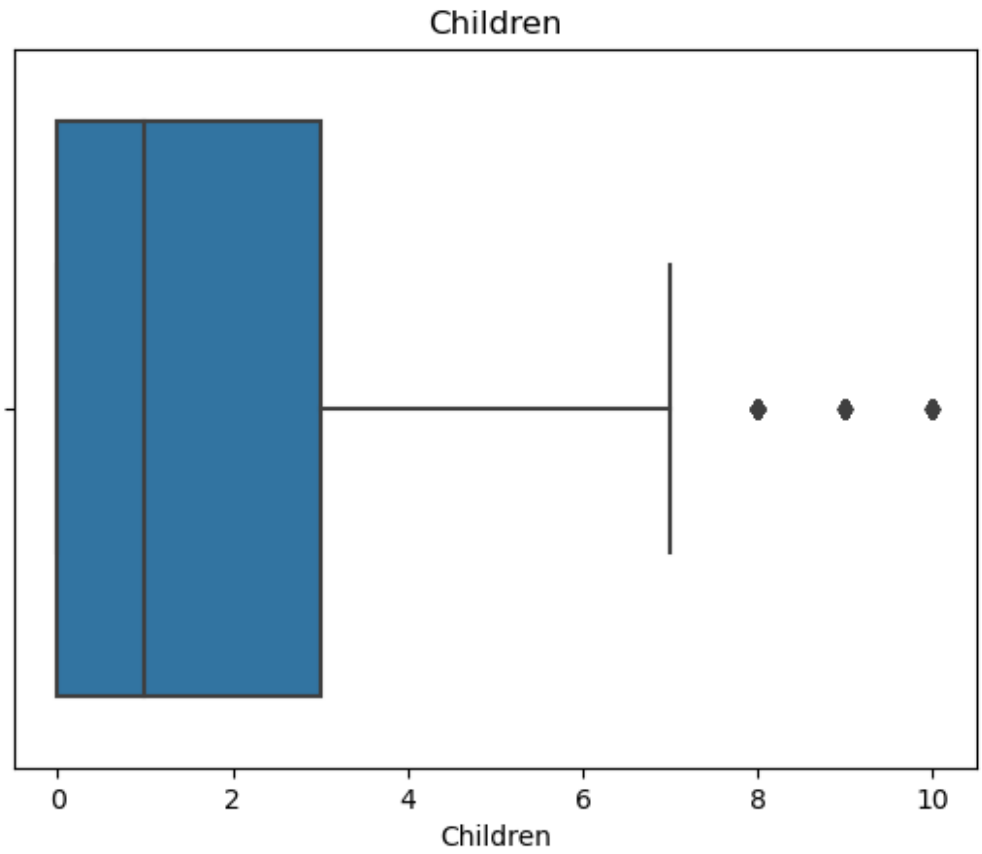
```
#Generate distribution for Population variable
plt.hist(df_churn['Population'])
plt.title("Population")
```

```
Text(0.5, 1.0, 'Population')
```



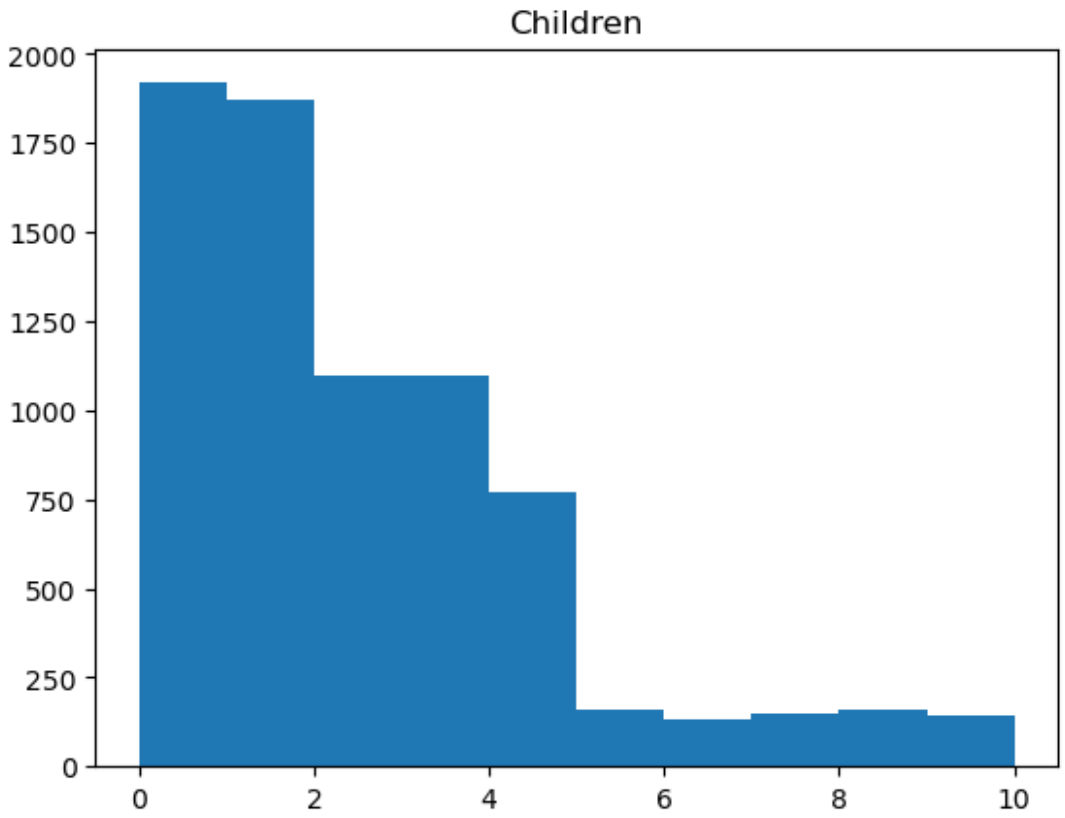

```
#Generate boxplot for Children variable
children_boxplot = sns.boxplot(x="Children", data =
df_churn).set_title("Children")

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

```
Q1: 0.0
Q3: 3.0
IQR: 3.0
Lower Whisker: -4.5
Upper Whisker: 7.5
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 302
Total number of Outliers: 302
Highest Outlier: 10.0
Lowest Outlier: 0.0
```



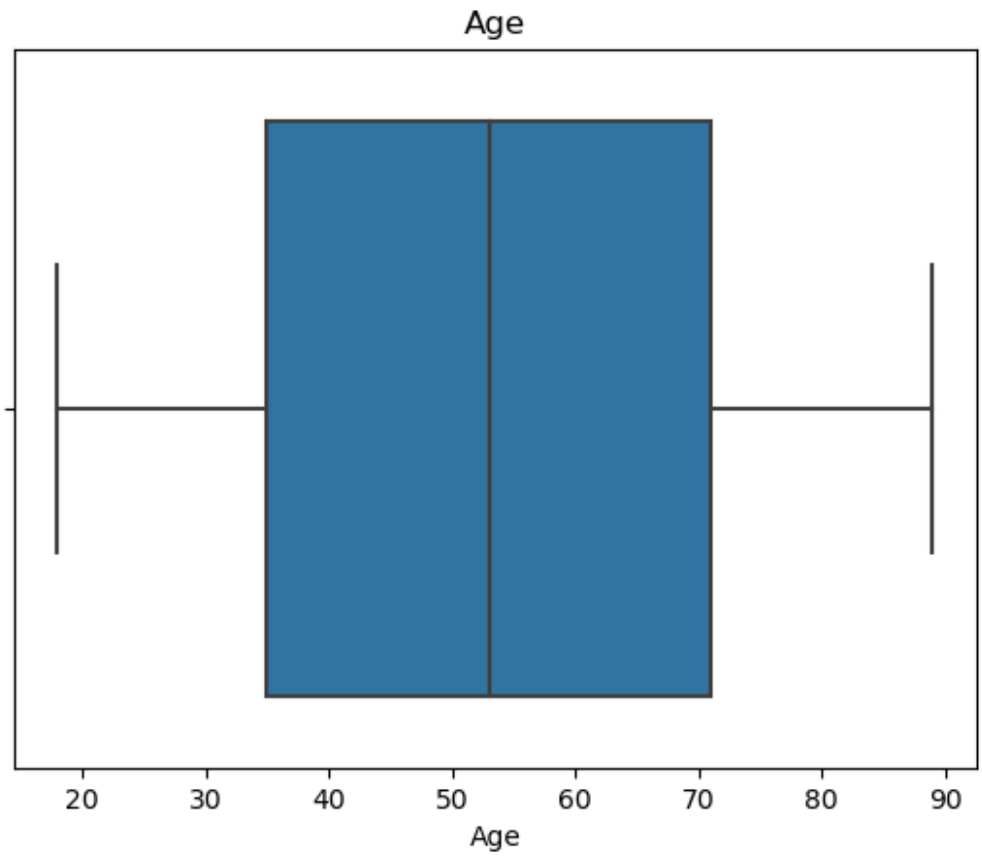

```
#Generate distribution for Children variable
plt.hist(df_churn['Children'])
plt.title("Children")
```

```
Text(0.5, 1.0, 'Children')
```



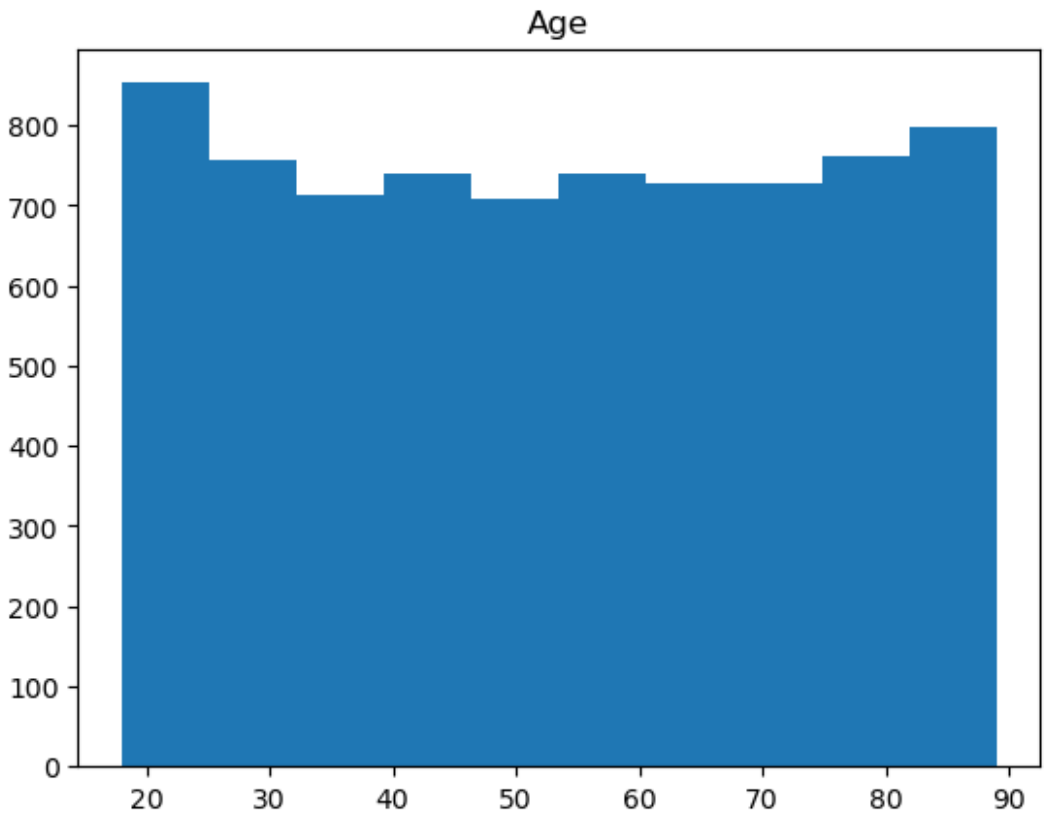

```
#Generate boxplot for Age variable
age_boxplot = sns.boxplot(x="Age", data = df_churn).set_title("Age")

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

```
Q1: 35.0
Q3: 71.0
IQR: 36.0
Lower Whisker: -19.0
Upper Whisker: 125.0
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 89.0
Lowest Outlier: 18.0
```



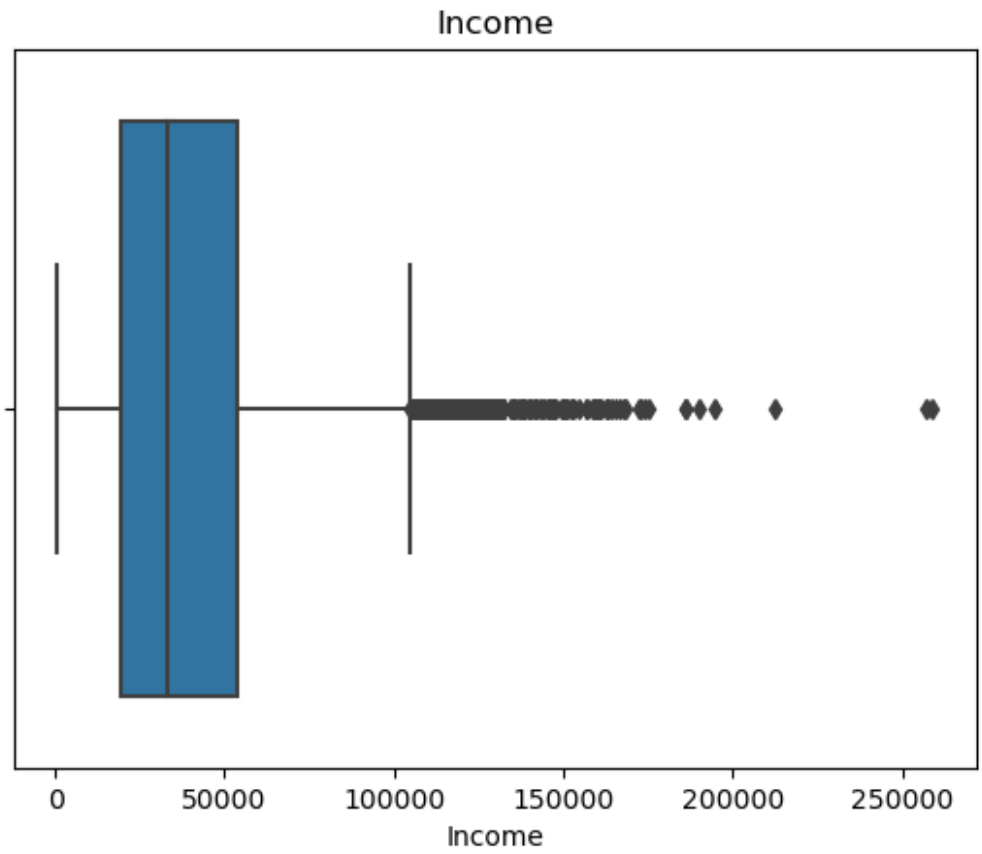

```
#Generate distribution for Age variable
plt.hist(df_churn['Age'])
plt.title("Age")
```

```
Text(0.5, 1.0, 'Age')
```



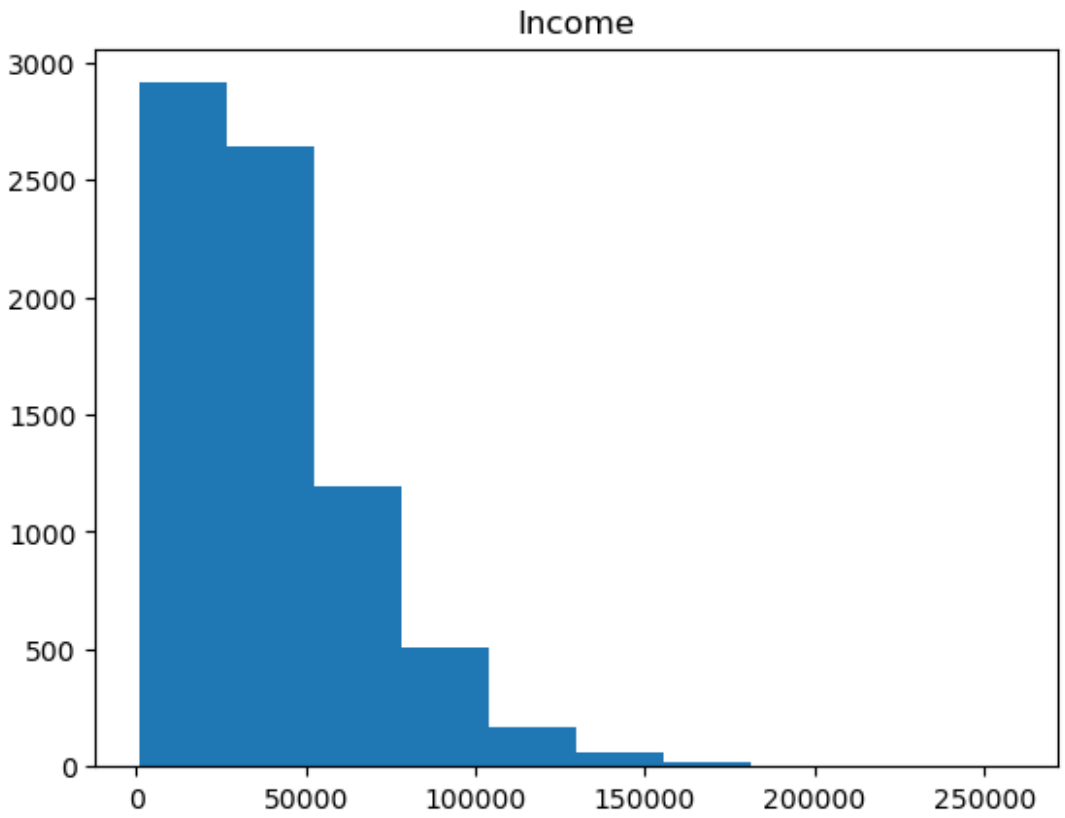

```
#Generate boxplot for Income variable
income_boxplot = sns.boxplot(x="Income", data =
df_churn).set_title("Income")

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

```
Q1: 19285.5225
Q3: 53472.395000000004
IQR: 34186.872500000005
Lower Whisker: -31994.786250000012
Upper Whisker: 104752.70375000002
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 249
Total number of Outliers: 249
Highest Outlier: 258900.7
Lowest Outlier: 740.66
```



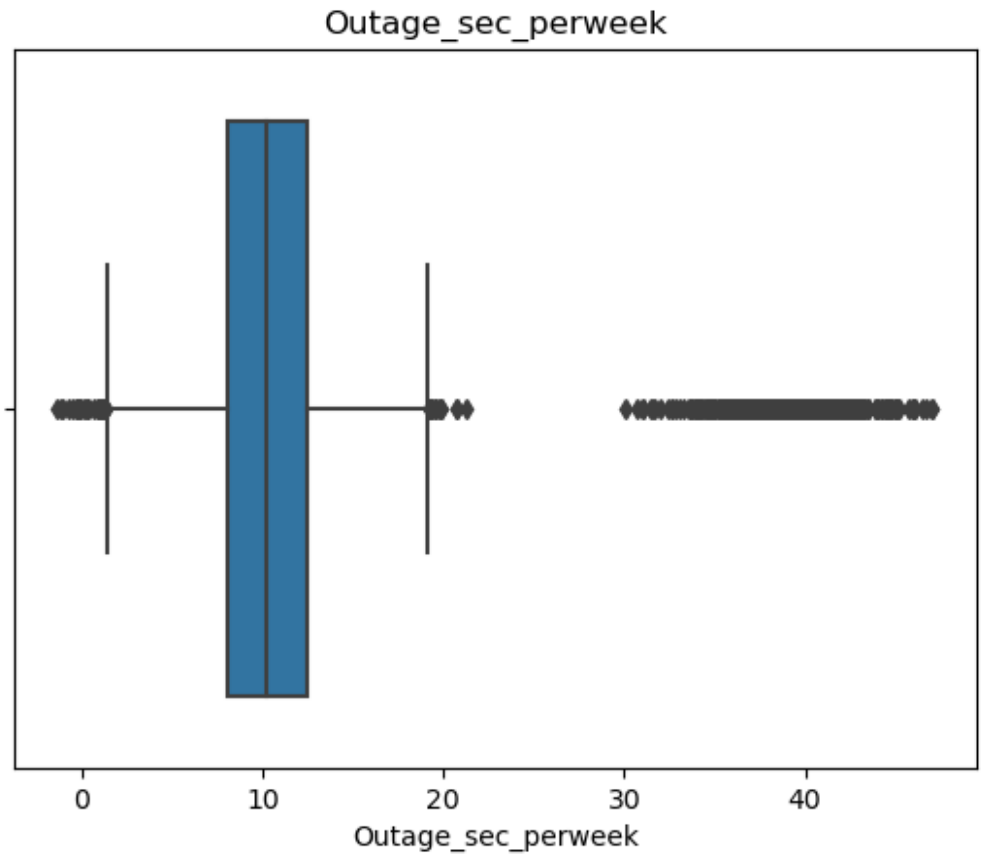

```
#Generate distribution for Income variable
plt.hist(df_churn['Income'])
plt.title("Income")
```

```
Text(0.5, 1.0, 'Income')
```



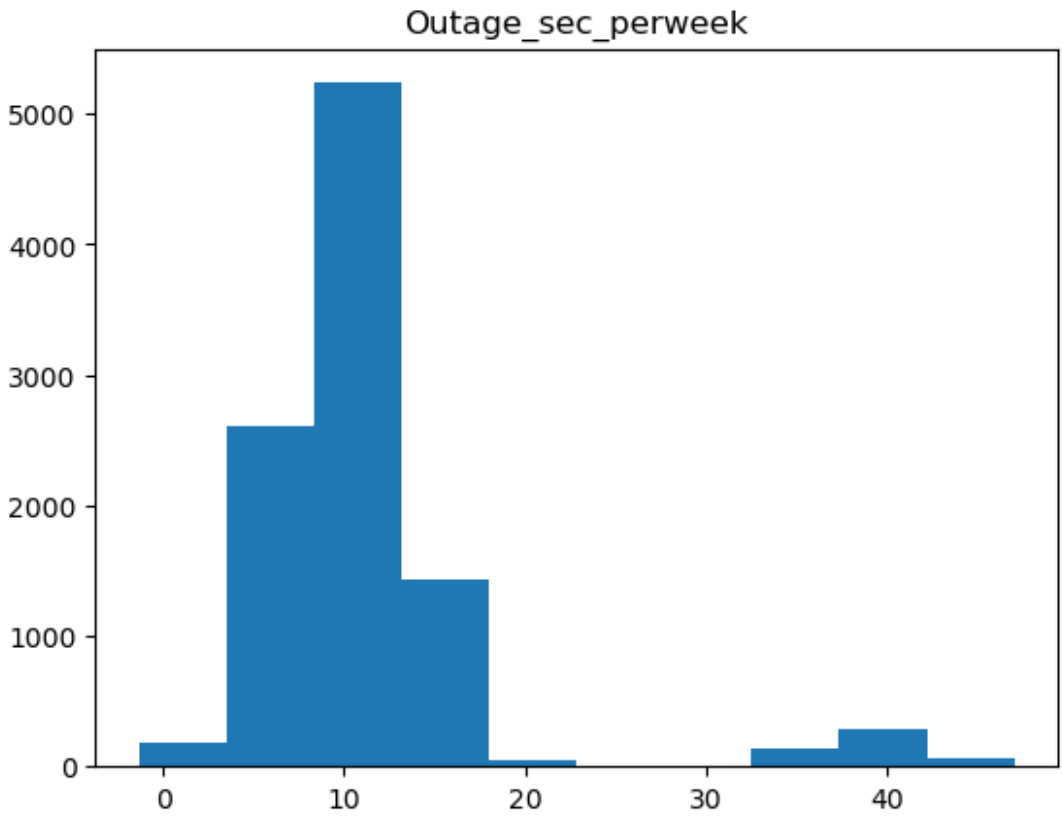

```
#Generate boxplot for Outage_sec_perweek variable
outage_boxplot = sns.boxplot(x="Outage_sec_perweek", data =
df_churn).set_title("Outage_sec_perweek")

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

```
Q1: 8.054362000000001
Q3: 12.487643622499998
IQR: 4.433281622499997
Lower Whisker: 1.404439566250005
Upper Whisker: 19.137566056249995
Number of outliers lower than boxplot minimum: 26
Number of outliers greater than boxplot maximum: 513
Total number of Outliers: 539
Highest Outlier: 47.04928
Lowest Outlier: -1.348571
```



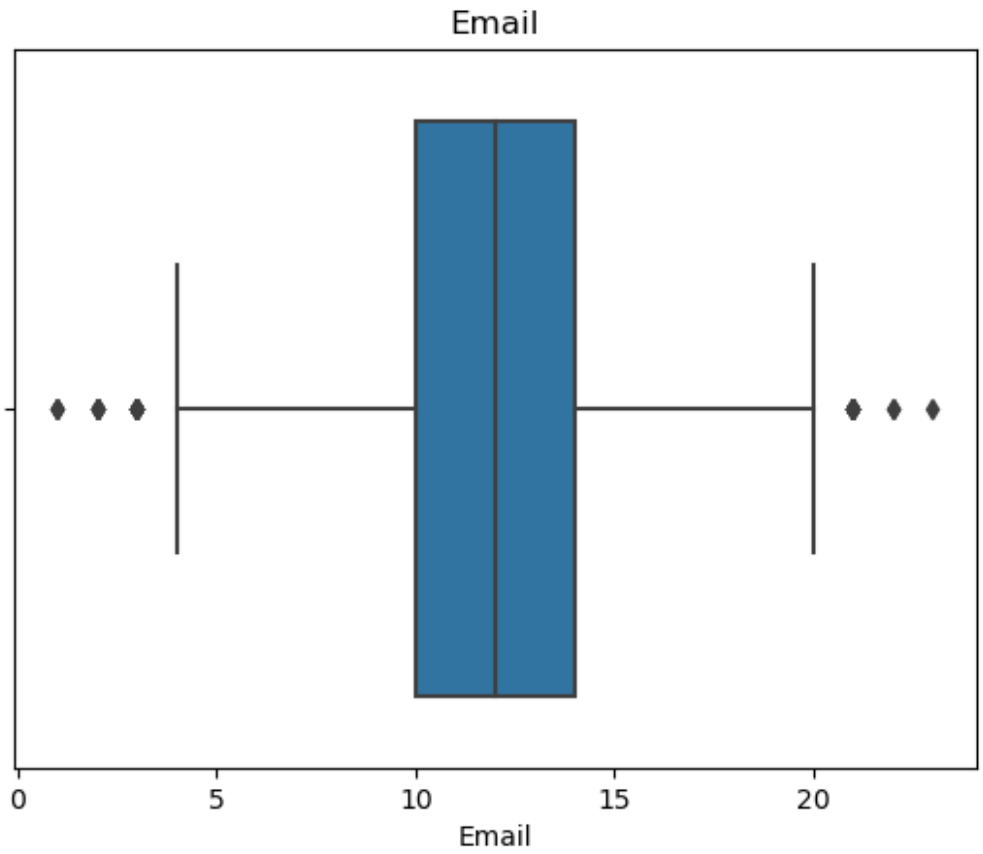

```
#Generate distribution for Outage_sec_perweek variable
plt.hist(df_churn['Outage_sec_perweek'])
plt.title("Outage_sec_perweek")
```

```
Text(0.5, 1.0, 'Outage_sec_perweek')
```



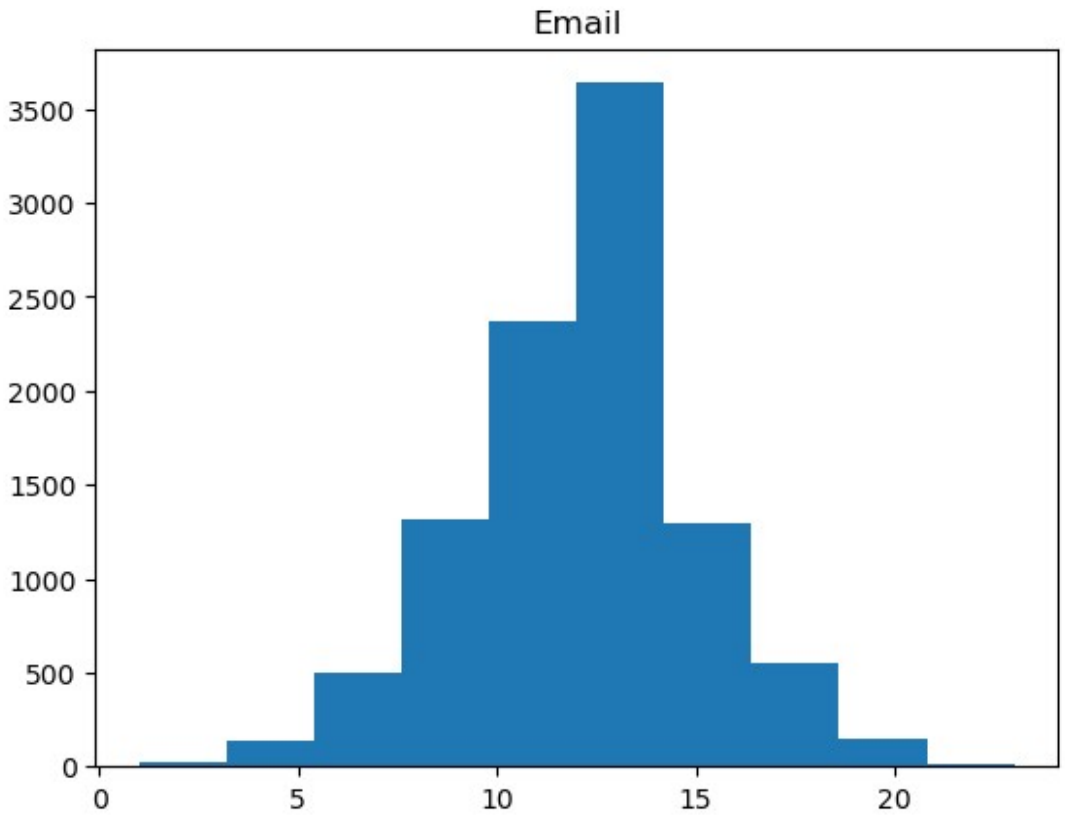

```
#Generate boxplot for Email variable
email_boxplot = sns.boxplot(x="Email", data =
df_churn).set_title("Email")

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

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



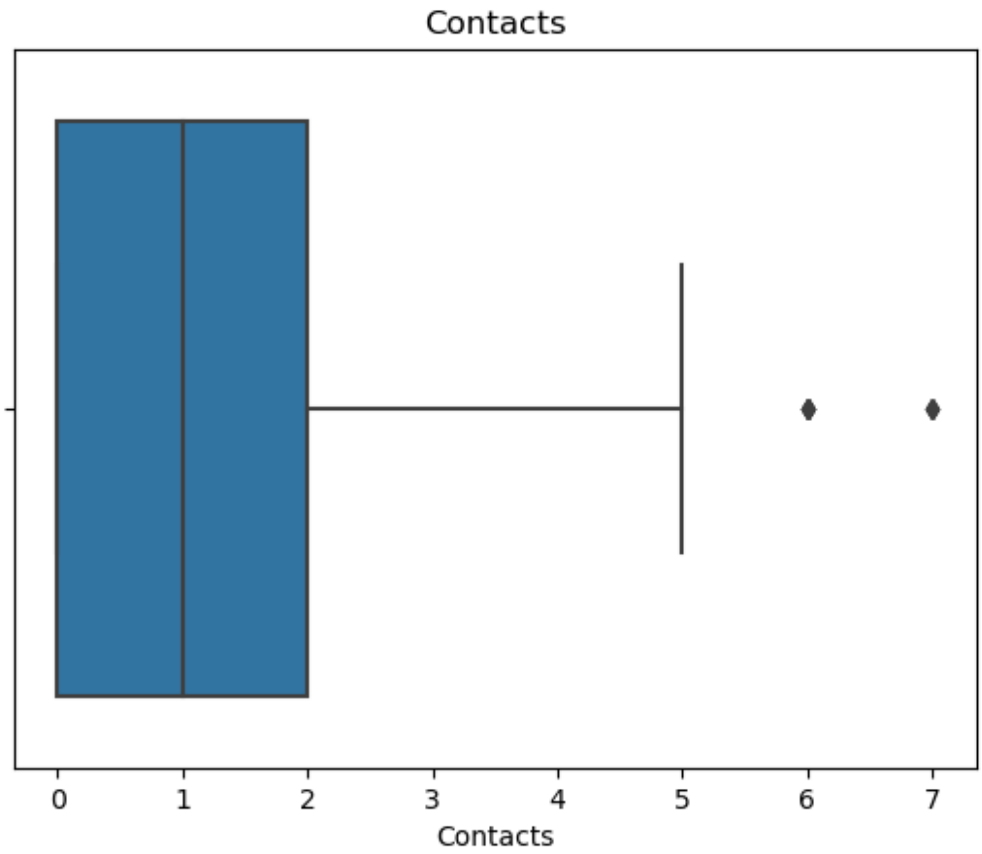

```
#Generate distribution for Email variable
plt.hist(df_churn['Email'])
plt.title("Email")
```

```
Text(0.5, 1.0, 'Email')
```



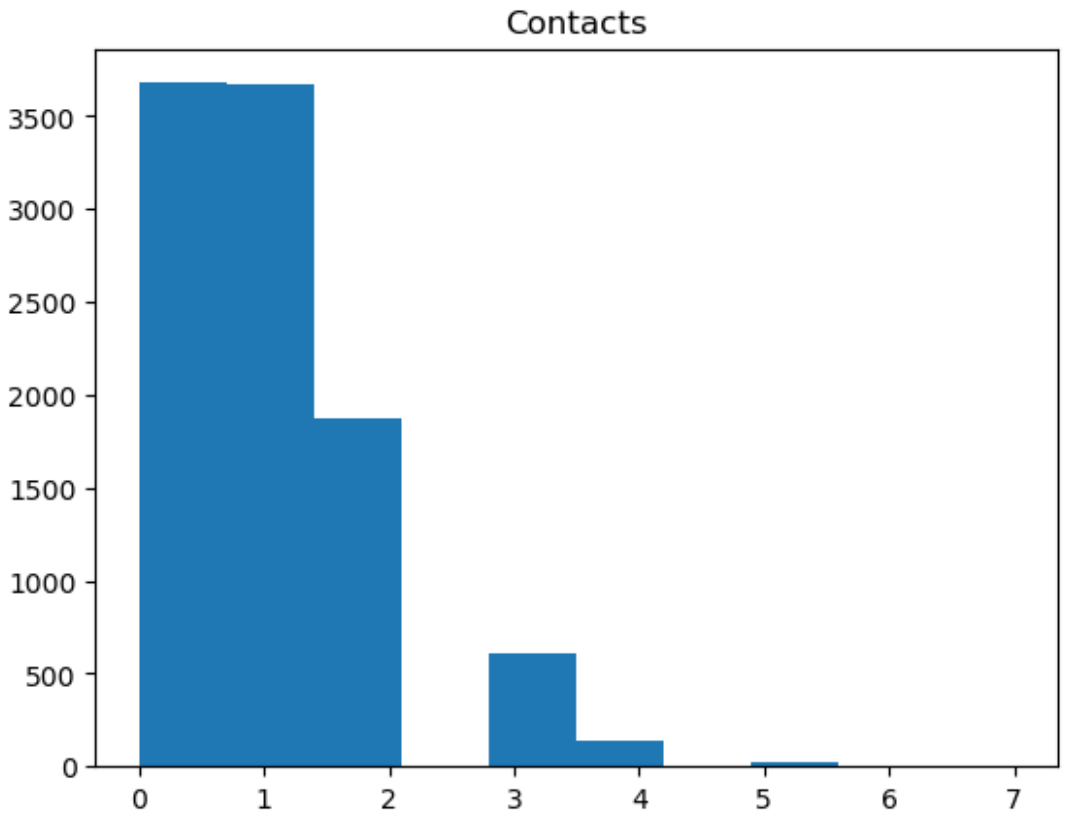

```
#Generate boxplot for Contacts variable
Contacts_boxplot = sns.boxplot(x="Contacts", data =
df_churn).set_title("Contacts")

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

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



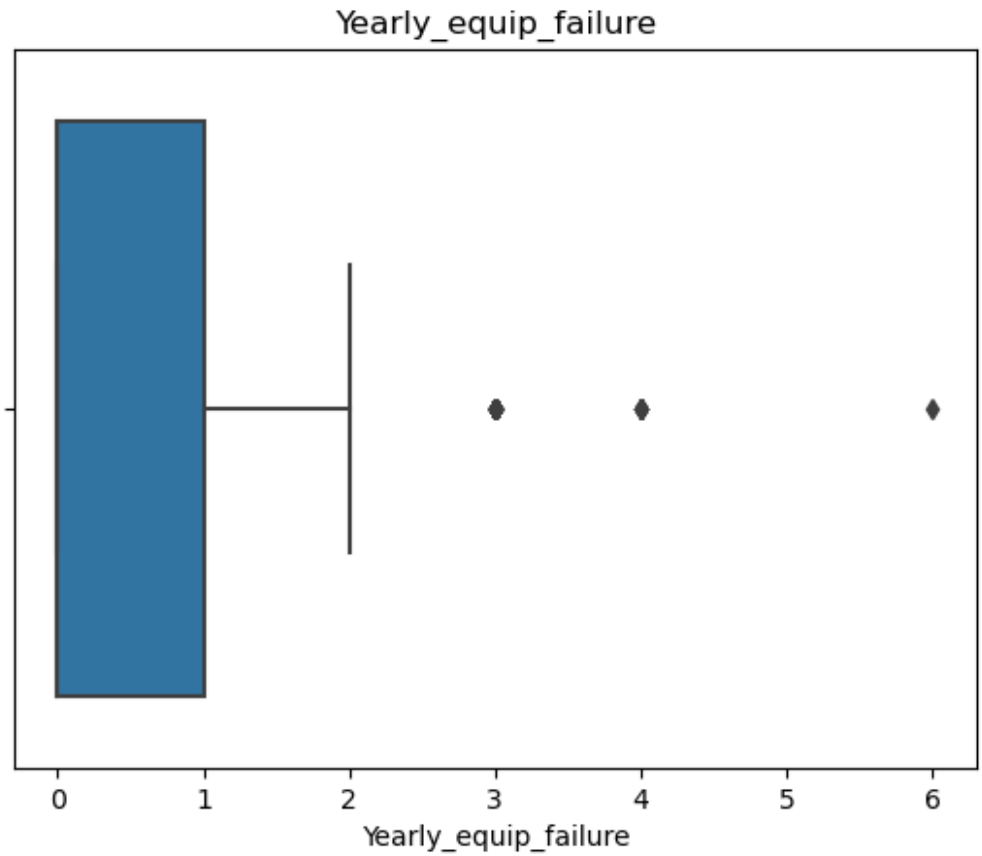

```
#Generate distribution for Contacts variable
plt.hist(df_churn['Contacts'])
plt.title("Contacts")
```

```
Text(0.5, 1.0, 'Contacts')
```



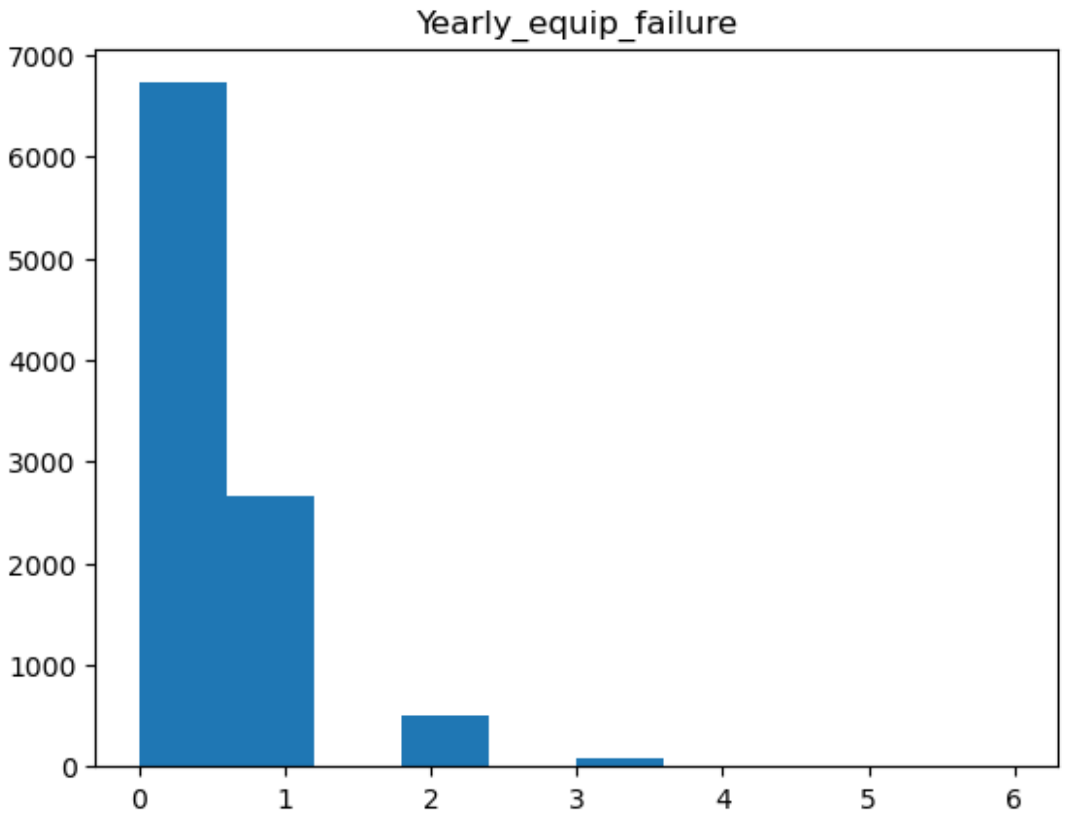

```
#Generate boxplot for Yearly_equip_failure variable
failure_boxplot = sns.boxplot(x="Yearly_equip_failure", data =
df_churn).set_title("Yearly_equip_failure")

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

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



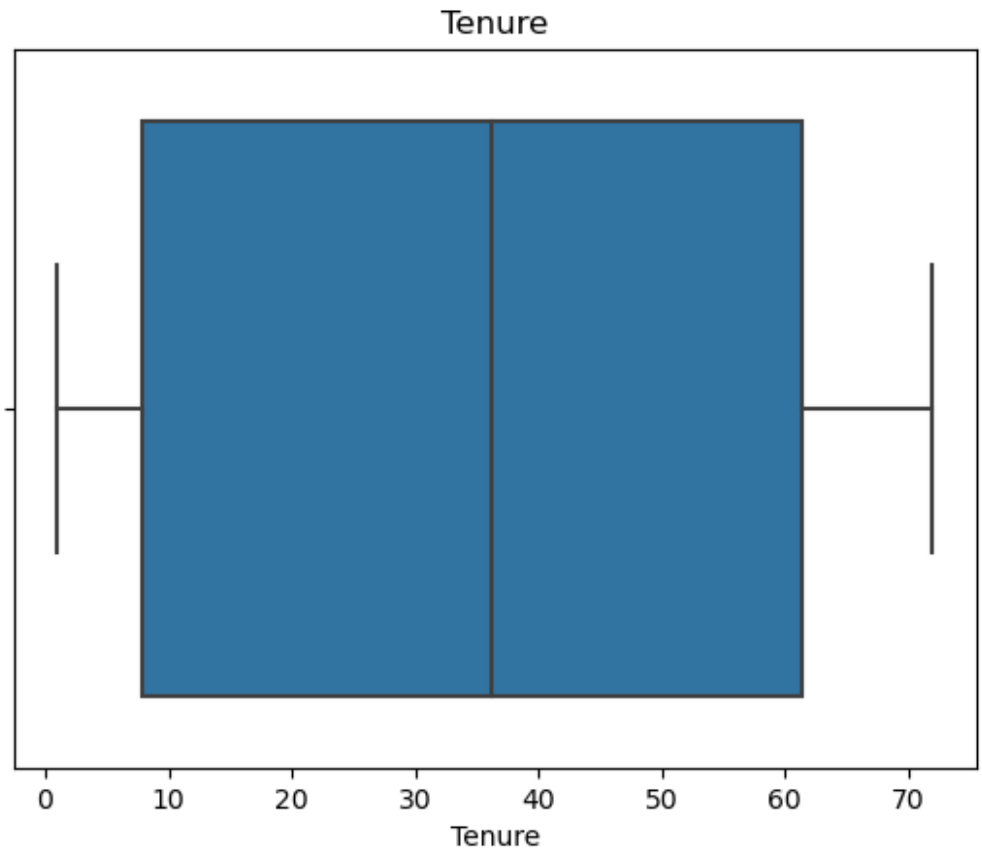

```
#Generate distribution for Yearly_equip_failure variable
plt.hist(df_churn['Yearly_equip_failure'])
plt.title("Yearly_equip_failure")
```

```
Text(0.5, 1.0, 'Yearly_equip_failure')
```



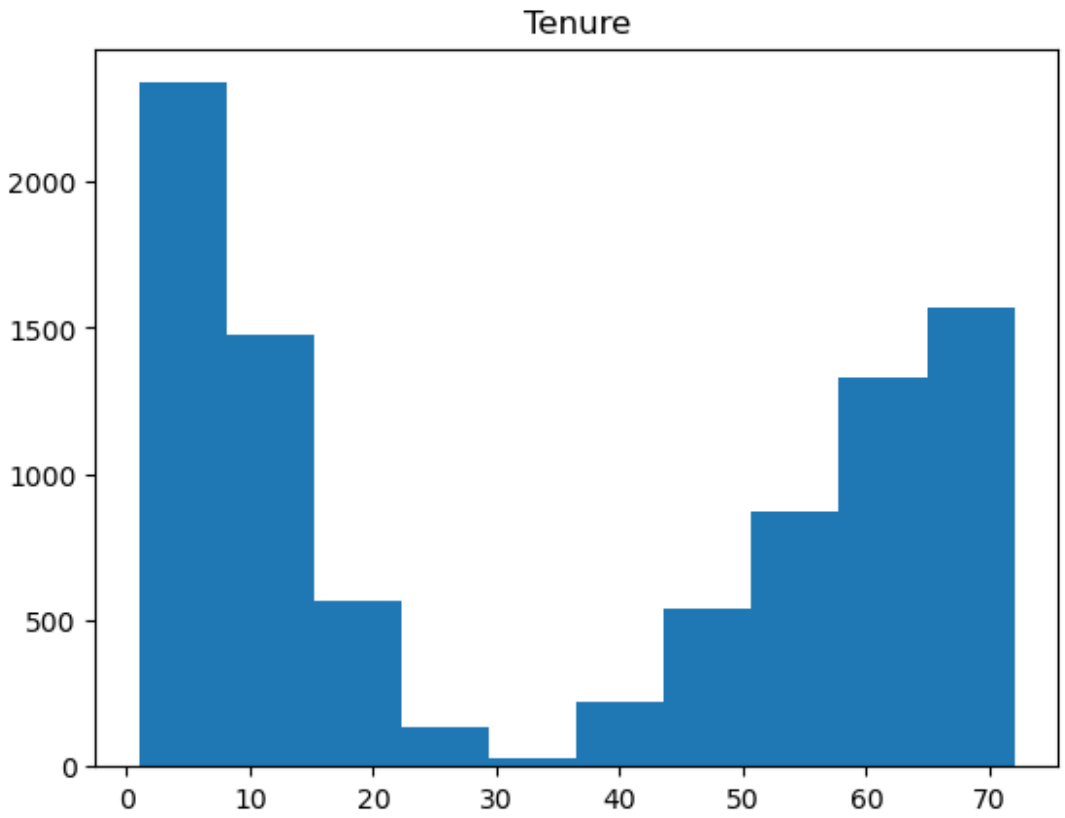

```
#Generate boxplot for Tenure variable
tenure_boxplot = sns.boxplot(x="Tenure", data =
df_churn).set_title("Tenure")

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

```
Q1: 7.890442
Q3: 61.42667
IQR: 53.536228
Lower Whisker: -72.41390000000001
Upper Whisker: 141.73101200000002
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 71.99928
Lowest Outlier: 1.00025934
```



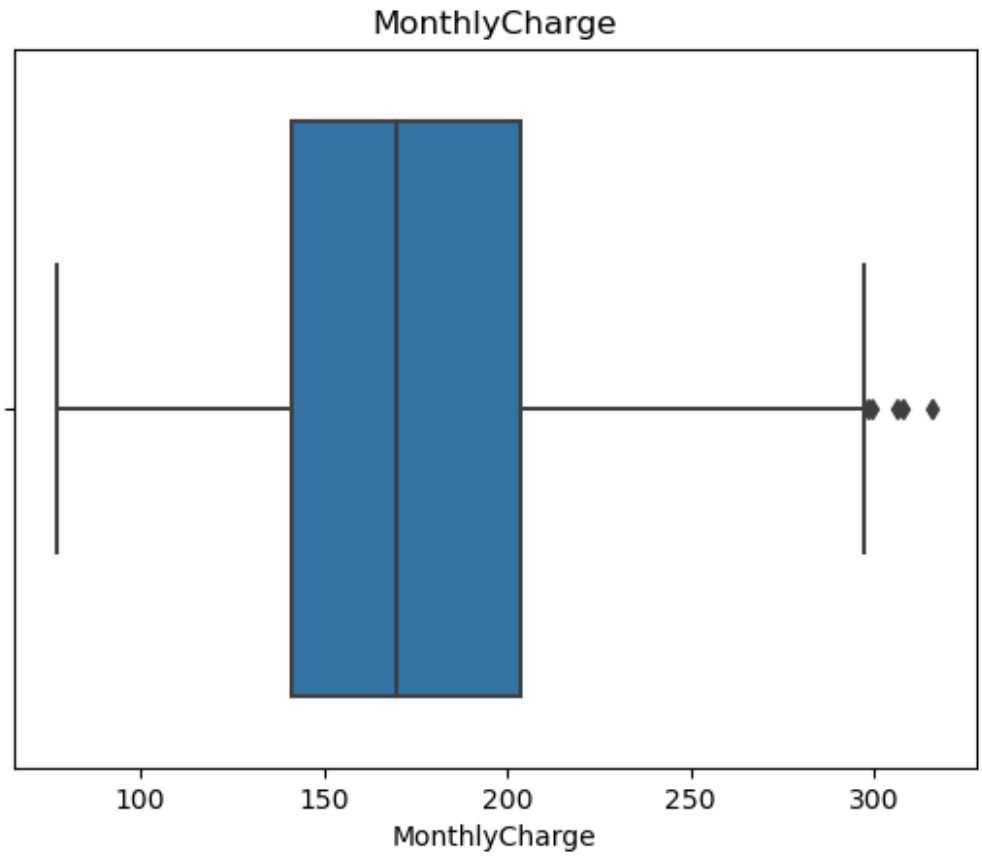

```
#Generate distribution for Tenure variable
plt.hist(df_churn['Tenure'])
plt.title("Tenure")
```

```
Text(0.5, 1.0, 'Tenure')
```



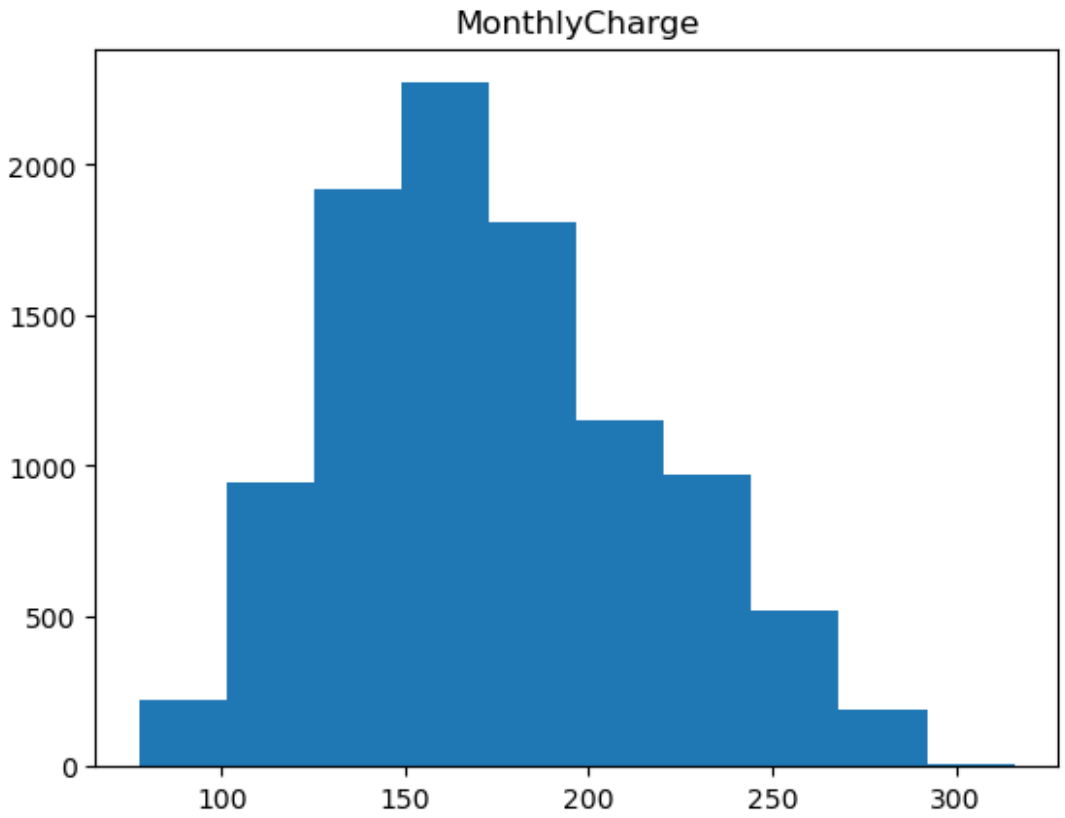

```
#Generate boxplot for MonthlyCharge variable
MonthlyCharge_boxplot = sns.boxplot(x="MonthlyCharge", data =
df_churn).set_title("MonthlyCharge")

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

```
Q1: 141.07107845
Q3: 203.777441275
IQR: 62.706362825000014
Lower Whisker: 47.01153421249997
Upper Whisker: 297.8369855125
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 5
Total number of Outliers: 5
Highest Outlier: 315.8786
Lowest Outlier: 77.50523
```



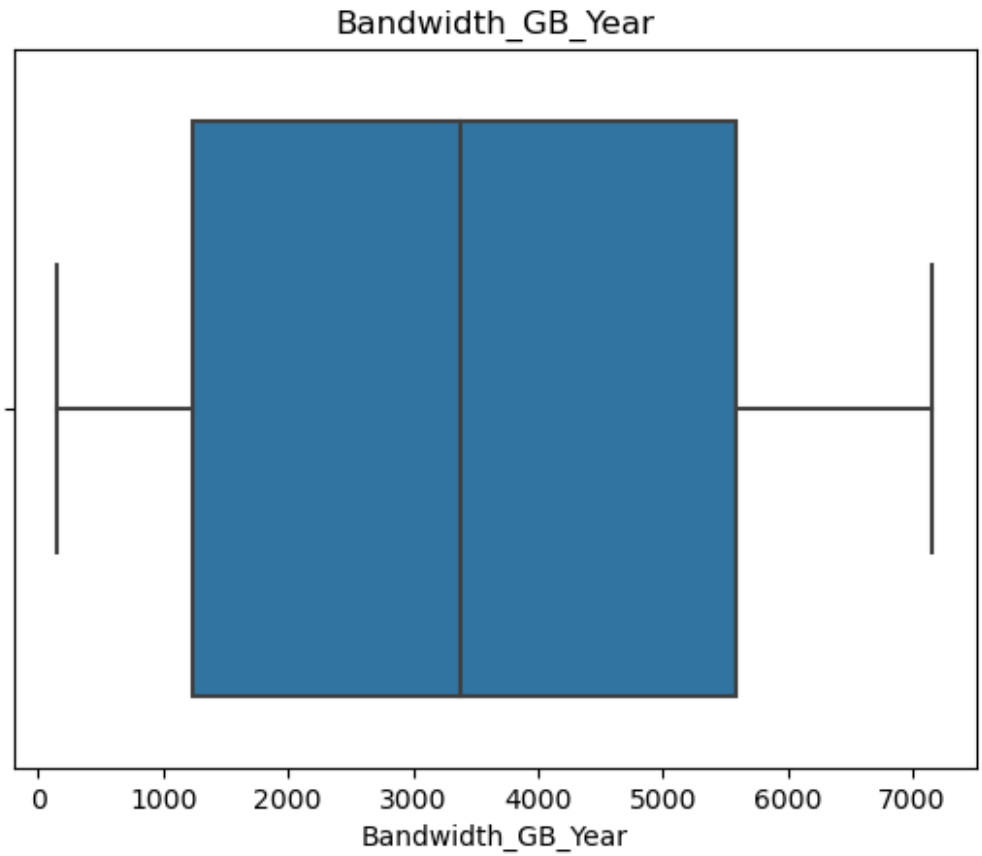

```
#Generate distribution for MonthlyCharge variable
plt.hist(df_churn['MonthlyCharge'])
plt.title("MonthlyCharge")
```

```
Text(0.5, 1.0, 'MonthlyCharge')
```



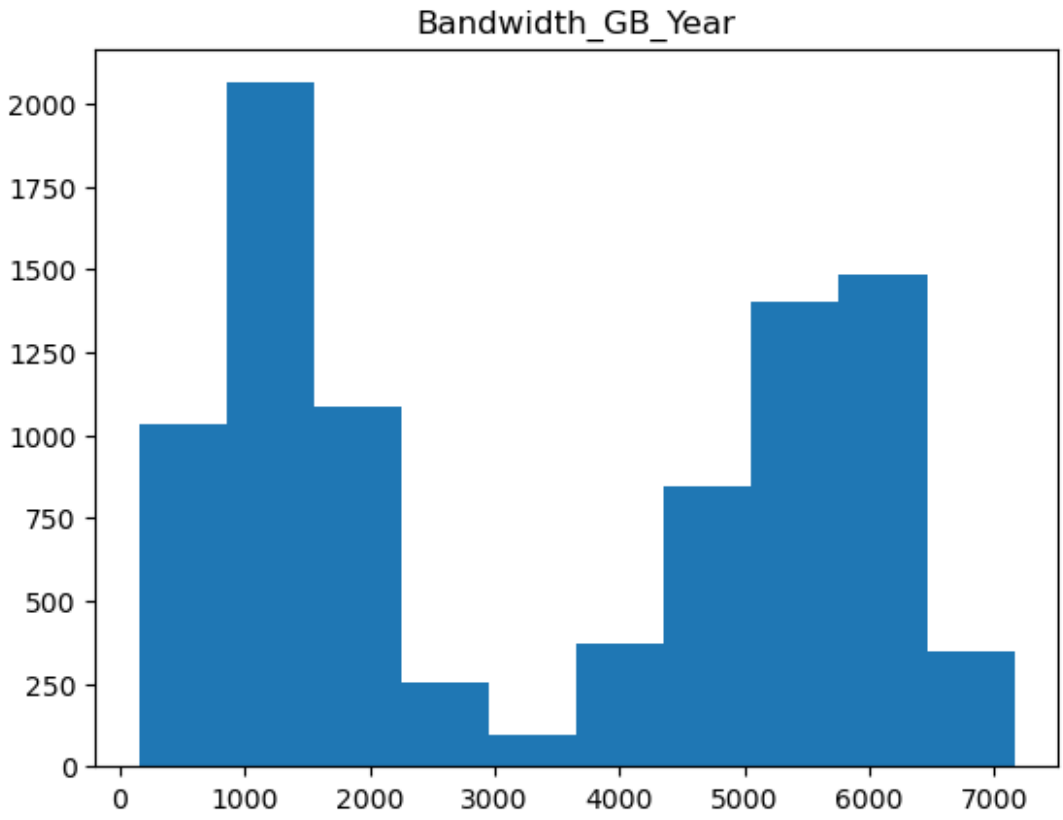

```
#Generate boxplot for Bandwidth_GB_Year variable
bandwidth_boxplot = sns.boxplot(x="Bandwidth_GB_Year", data =
df_churn).set_title("Bandwidth_GB_Year")

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

```
Q1: 1234.110529
Q3: 5587.0965
IQR: 4352.985971
Lower Whisker: -5295.3684275
Upper Whisker: 12116.575456499999
Number of outliers lower than boxplot minimum: 0
Number of outliers greater than boxplot maximum: 0
Total number of Outliers: 0
Highest Outlier: 7158.982
Lowest Outlier: 155.5067148
```




```
#Generate distribution for Bandwidth_GB_Year variable
plt.hist(df_churn['Bandwidth_GB_Year'])
plt.title("Bandwidth_GB_Year")
```

```
Text(0.5, 1.0, 'Bandwidth_GB_Year')
```



D2.

The next phase of the data cleaning process is treatment. The goal of this phase is to treat data quality issues by handling duplicates, missing values, outliers, and other factors that could impede an analysis.

Duplicate rows were not treated because there were none found in the dataset. However, there were a significant number of missing values that needed to be dealt with. To identify the best course of action, I decided to generate histograms for each of the variables with missing values using the “plt.hist()” function from the matplotlib library. I did this because viewing the distribution for each variable would allow me to select the best method of imputation.

The variables “Children” and “Income” both resulted in skewed distributions, while “Tenure” and “Bandwidth_GB_Year” both resulted in bimodal distributions. Because of this, I chose to impute their missing values using the median. I chose to impute the missing values in “Age” with the mean because it had a uniform distribution. Rather than using distributions for “Techie”, “Phone”, and “TechSupport”, I chose to simply impute their missing values with the mode because they are all categorical. Ultimately, I treated the missing values using univariate imputation because it is a simple solution that did not require me to reduce the sample size of the data.

Upon reviewing the columns the boxplots and outlier infomation shown in D2, I chose to retain outliers for almost all of the quantitative variables. Even though there was a significant amount of outliers in "Population", "Children", and "Income" I chose to retain them because the

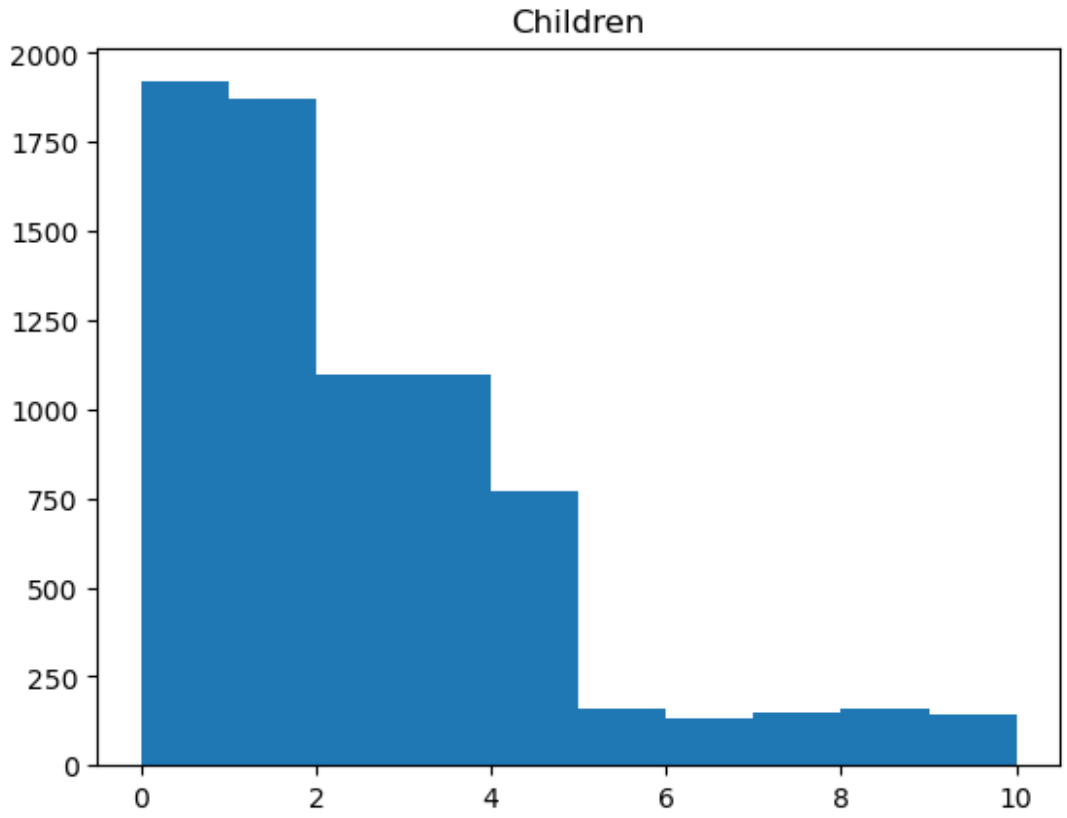
values were plausible and not egregious enough to exclude. I also did not want to reduce the sample size or potentially introduce bias into the dataset.

I chose to impute the outliers for "Outage_sec_perweek" because I noticed that the lowest value was -1.348571 which is clearly an error. There is also a noticeable gap and extreme range in the upper outliers. To fix this issue, I imputed the outliers using the median. This is because the values and count of outliers were not acceptable or expected.

Apart from these data quality issues, I also chose to re-express the “Education” column because of its ordinal nature. To do this I created a dictionary named “dict_edu”. This dictionary contains all the unique categorical values in “Education” and assigns a corresponding value which represents the years of education. I then used the “replace()” function on df_churn to update all the values in the “Education” column.

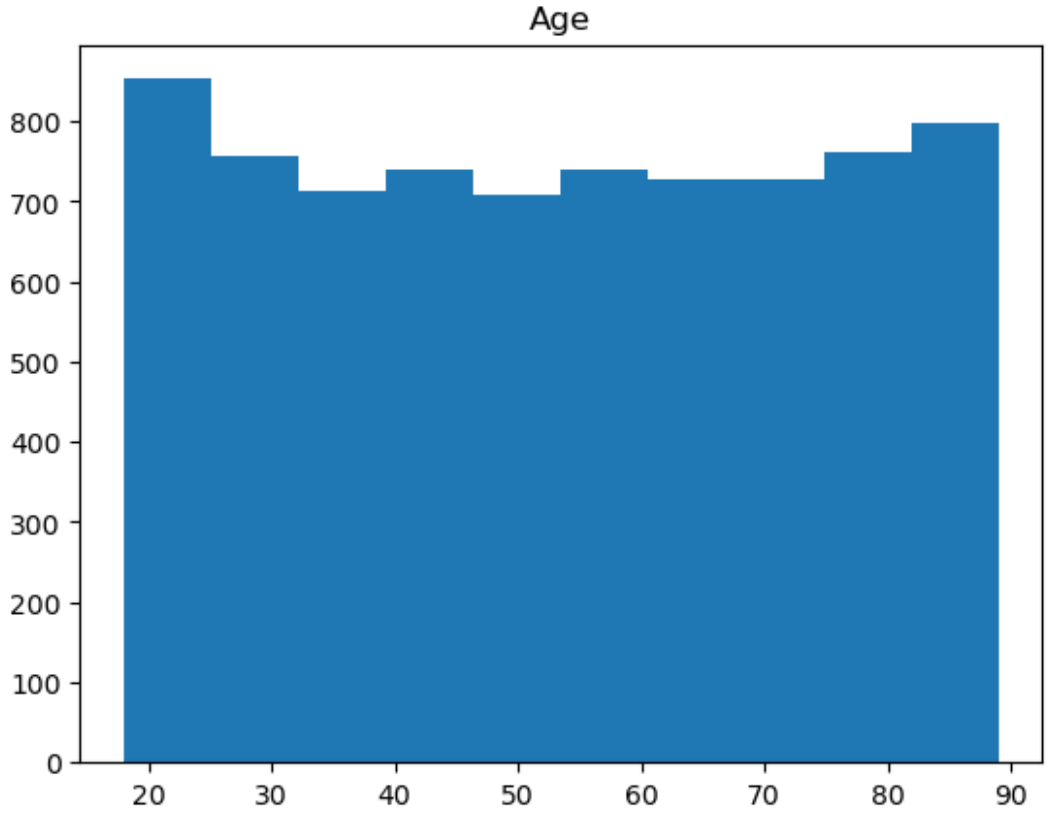
```
#Generate histograms for variables with missing values and determine
best method of treatment

#Generate distribution for Children variable
plt.hist(df_churn['Children'])
plt.title("Children")
```

```
Text(0.5, 1.0, 'Children')
```



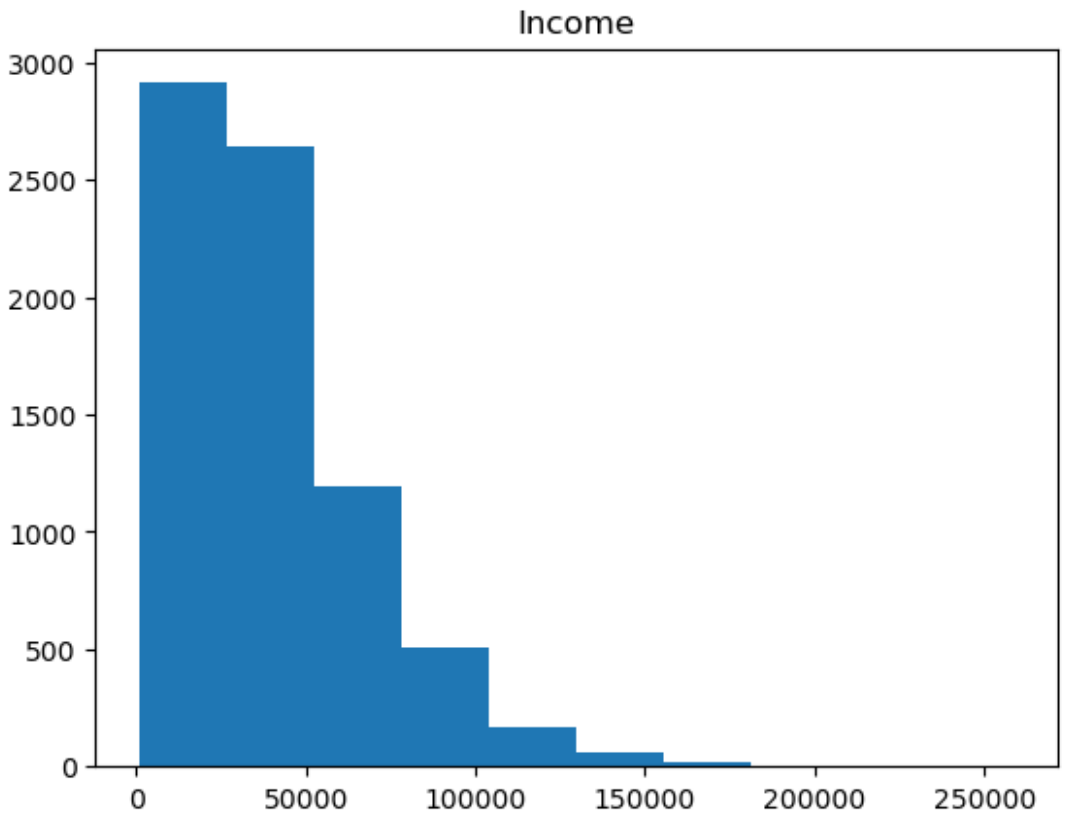
```
#Based on the skew of the distribution, the missing values will be
imputed with the median of the Children variable
df_churn['Children'].fillna(df_churn['Children'].median(), inplace =
True)

#Generate distribution for Age variable
plt.hist(df_churn['Age'])
plt.title("Age")
```

```
Text(0.5, 1.0, 'Age')
```

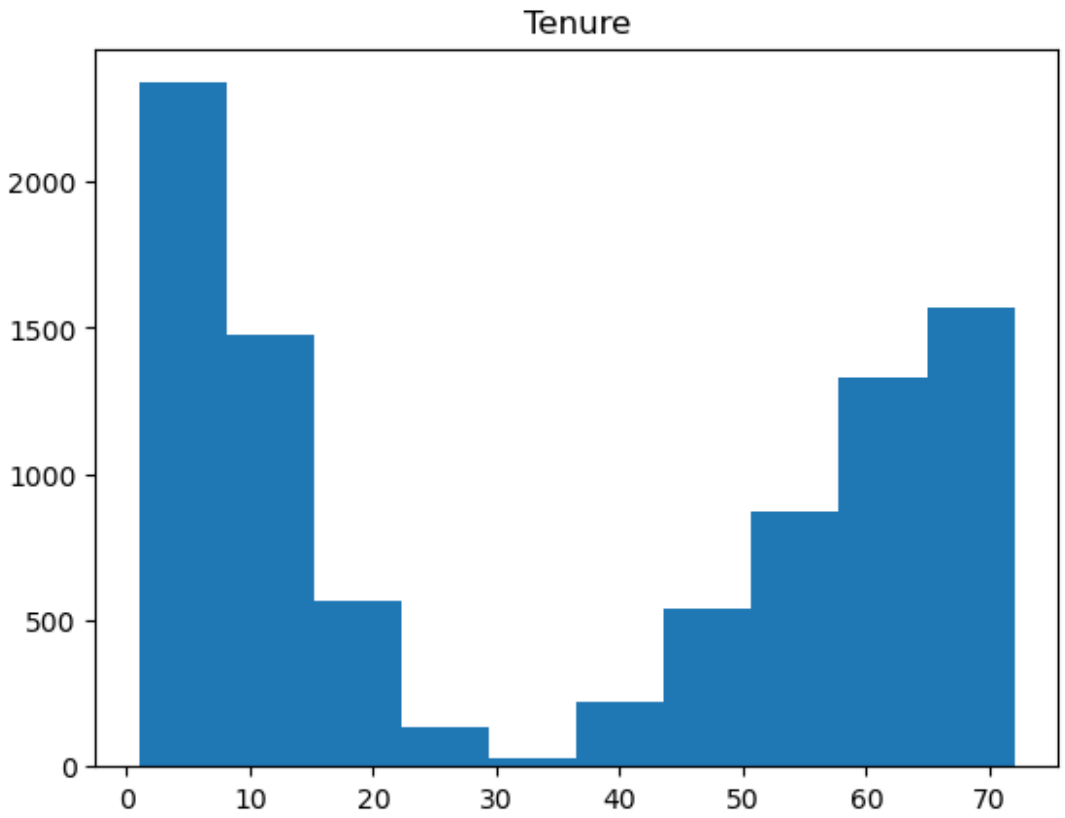



```
#Based on the skew of the distribution, the missing values will be
imputed with the rounded mean of the Age variable
df_churn['Age'].fillna(round(df_churn['Age'].mean()), inplace = True)

#Generate distribution for Income variable
plt.hist(df_churn['Income'])
plt.title("Income")
```

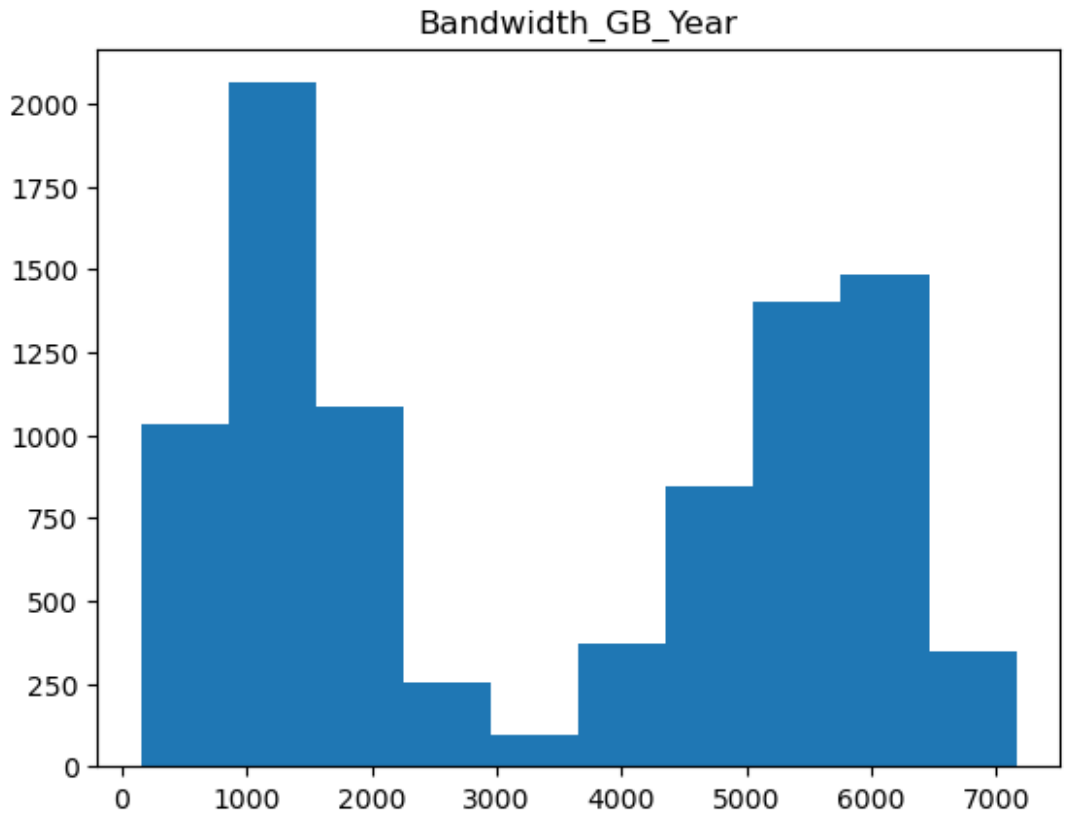
```
Text(0.5, 1.0, 'Income')
```




```
#Based on the skew of the distribution, the missing values will be
imputed with the median of the Income variable
df_churn['Income'].fillna(df_churn['Income'].median(), inplace = True)

#Generate distribution for Tenure variable
plt.hist(df_churn['Tenure'])
plt.title("Tenure")

Text(0.5, 1.0, 'Tenure')
```




```
#Based on the skew of the distribution, the missing values will be
imputed with the median of the Income variable
df_churn['Tenure'].fillna(df_churn['Tenure'].median(), inplace = True)

#Generate distribution for Bandwidth_GB_Year variable
plt.hist(df_churn['Bandwidth_GB_Year'])
plt.title("Bandwidth_GB_Year")

Text(0.5, 1.0, 'Bandwidth_GB_Year')
```




```
#Based on the skew of the distribution, the missing values will be
imputed with the median of the Income variable
df_churn['Bandwidth_GB_Year'].fillna(df_churn['Bandwidth_GB_Year'].med
ian(), inplace = True)

#For categorical variables "Techie", "Phone" and "Income", impute with
mode

df_churn['Techie'] =
df_churn['Techie'].fillna(df_churn['Techie'].mode()[0])

df_churn['Phone'] = df_churn['Phone'].fillna(df_churn['Phone'].mode()
[0])

df_churn['TechSupport'] =
df_churn['TechSupport'].fillna(df_churn['TechSupport'].mode()[0])

#Impute outliers in "Outage_sec_perweek" with the median
df_churn["Outage_sec_perweek"] =
np.where(df_churn["Outage_sec_perweek"] < 4, np.nan,
df_churn["Outage_sec_perweek"])
df_churn["Outage_sec_perweek"] =
np.where(df_churn["Outage_sec_perweek"] > 20, np.nan,
df_churn["Outage_sec_perweek"])
```

```
#Impute NaN values with median
df_churn["Outage_sec_perweek"].fillna(df_churn["Outage_sec_perweek"].m
edian(), inplace=True)

#Re-express categorical variables

#Step 1: Find unique values in Education column
print(df_churn['Education'].unique())

#Step 2: Create dictionary to ordinally encode Education values

dict_edu = {"Education":
                {"Master's Degree": 18,
                        "Regular High School Diploma": 12,
                        "Doctorate Degree":24,
                        "No Schooling Completed": 0,
                        "Associate's Degree": 15,
                        "Bachelor's Degree": 16,
                        "Some College, Less than 1 Year": 13,
                        "GED or Alternative Credential": 12,
                        "Some College, 1 or More Years, No Degree":
14,
                        "9th Grade to 12th Grade, No Diploma": 12,
                        "Nursery School to 8th Grade": 8,
                        "Professional School Degree": 20,
                }
           }

#Step 3: Reexpress Education column as numeric
df_churn.replace(dict_edu, inplace = True)

#Step 4: Confirm values have been re-expressed
print(df_churn['Education'].unique())
```

```
["Master's Degree" 'Regular High School Diploma' 'Doctorate Degree'
 'No Schooling Completed' "Associate's Degree" "Bachelor's Degree"
 'Some College, Less than 1 Year' 'GED or Alternative Credential'
 'Some College, 1 or More Years, No Degree'
 '9th Grade to 12th Grade, No Diploma' 'Nursery School to 8th Grade'
 'Professional School Degree']
[18 12 24  0 15 16 13 14  8 20]
```

D3.

The data has now been transformed in several ways. The columns "Children", "Age", "Income", "Tenure", "Bandwidth_GB_Year" have had all of their missing values imputed with the median or mean based on their initial distribution. "Techie", "Phone", and "TechSupport" had their missing values imputed with the mode because they are categorical variables. The output of the "df_churn.isnull().sum() function below confirms that none of the variables have missing values anymore.

To visualize the changes resulting from treatment, I generated new histograms. "Children" and "Income" still have a left-leaning skew, however the distributions are slightly more centered than before. "Age" continues to have a uniform distribution with a spike in the middle due to the imputation of missing values with the mean. "Tenure" and "Bandwidth_GB_"Year" still have bimodal distributions but they now have spikes in the middle due to the imputation of missing values with the mean. This was also expected when altering the data.

The variables "Phone", "Techie", and "TechSupport" had their missing values imputed with the mode. Because a distribution could not be generated, I chose to review the changes using the "value_counts()" function. All three variables now have a large ratio of "Yes" values to "No" values.

As mentioned previously, "Outage_sec_perweek" was the only variable whose outliers I chose to impute. The new distribution for this variable appears much more normal than before, with most of the values appearing in the middle. Before imputation, the distribution had noticeable outliers and a strong left-leaning skew.

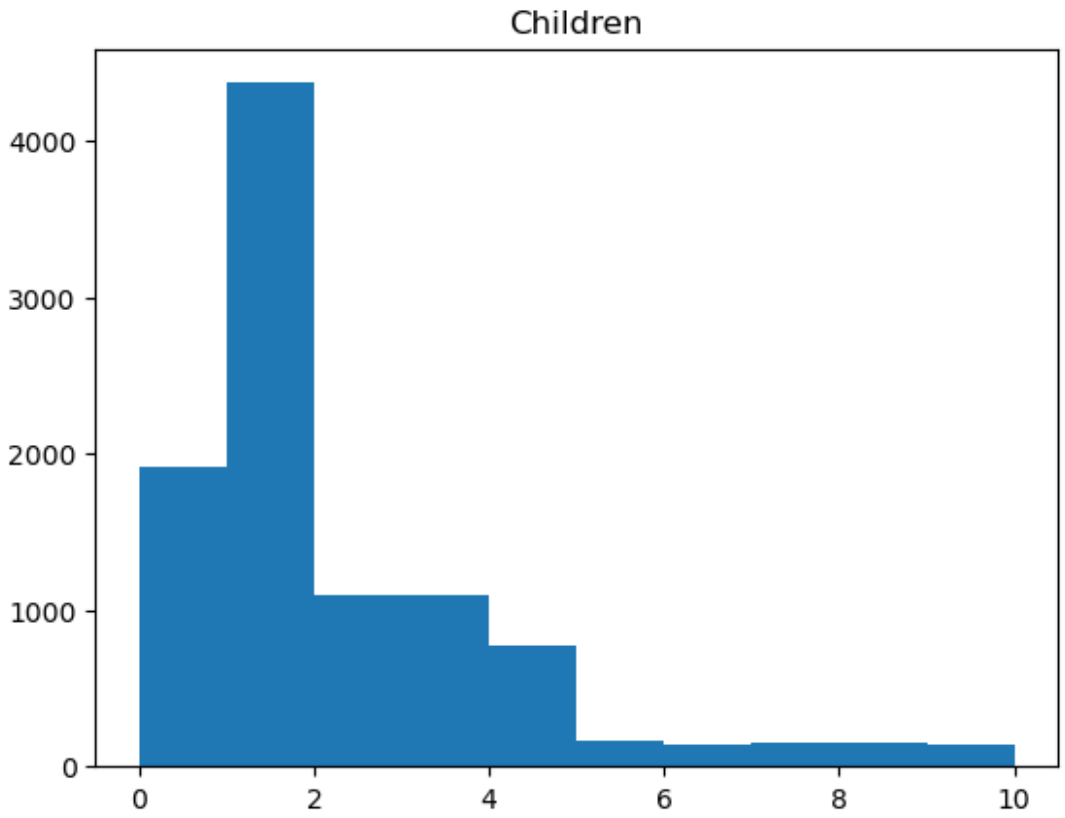
```
#Confirm missing values have been imputed
df_churn.isnull().sum()
```

```
Unnamed: 0              0
CaseOrder               0
Customer_id             0
Interaction             0
City                    0
State                   0
County                  0
Zip                     0
Lat                     0
Lng                     0
Population              0
Area                    0
Timezone                0
Job                     0
Children                0
Age                     0
Education               0
Employment              0
Income                  0
Marital                 0
Gender                  0
Churn                   0
Outage_sec_perweek      0
Email                   0
Contacts                0
Yearly_equip_failure    0
Techie                  0
Contract                0
Port_modem              0
Tablet                  0
```

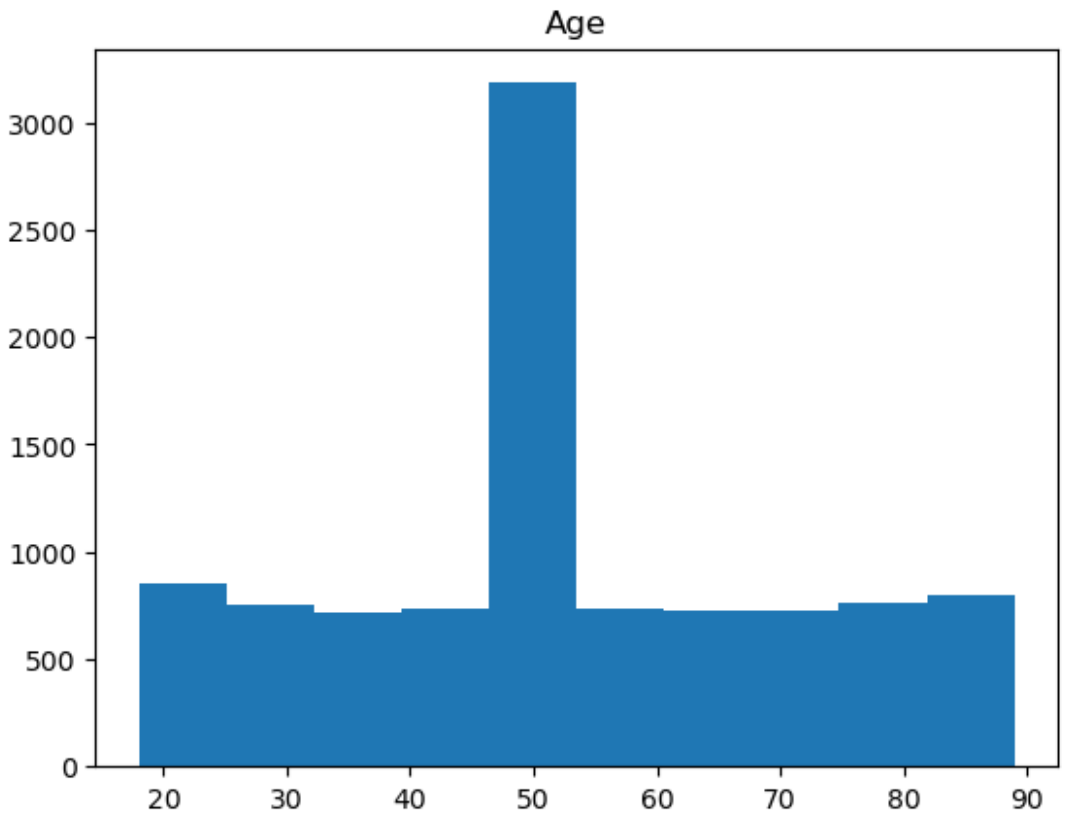
```
InternetService          0
Phone                    0
Multiple                 0
OnlineSecurity           0
OnlineBackup             0
DeviceProtection         0
TechSupport              0
StreamingTV              0
StreamingMovies          0
PaperlessBilling         0
PaymentMethod            0
Tenure                   0
MonthlyCharge            0
Bandwidth_GB_Year        0
item1                    0
item2                    0
item3                    0
item4                    0
item5                    0
item6                    0
item7                    0
item8                    0
dtype: int64
```

```
#Generate new distribution for Children variable
plt.hist(df_churn['Children'])
plt.title("Children")
```

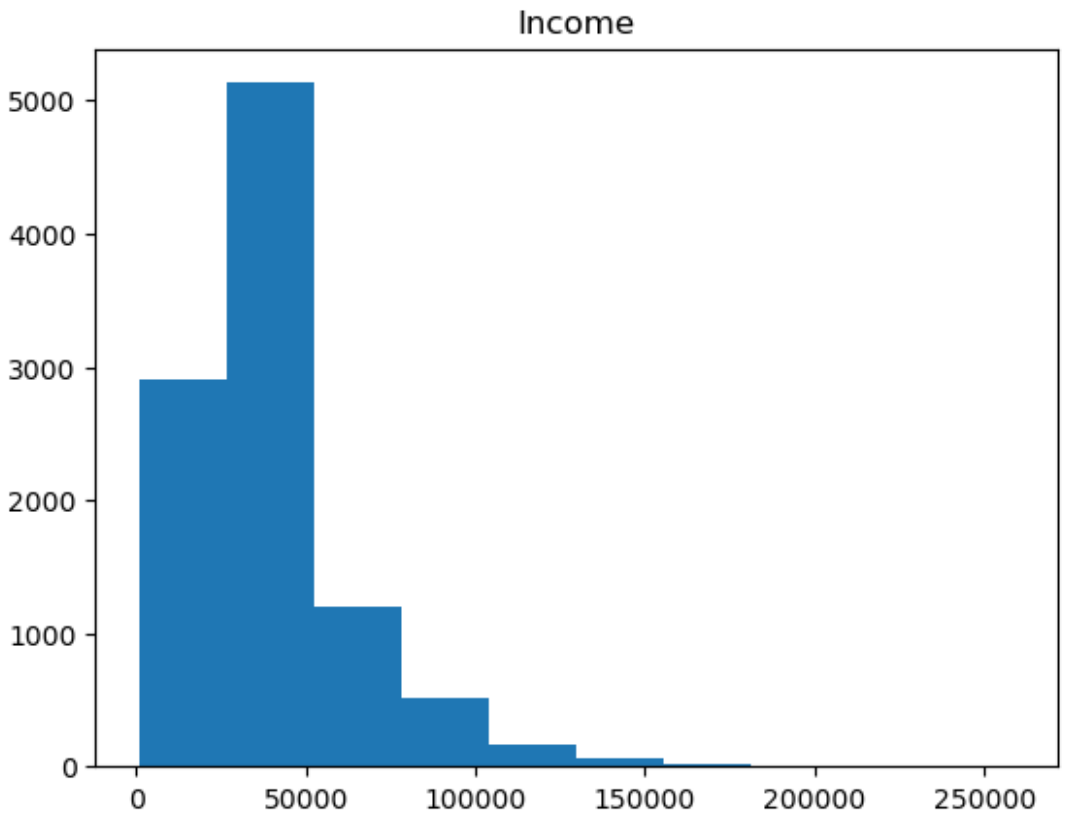
```
Text(0.5, 1.0, 'Children')
```



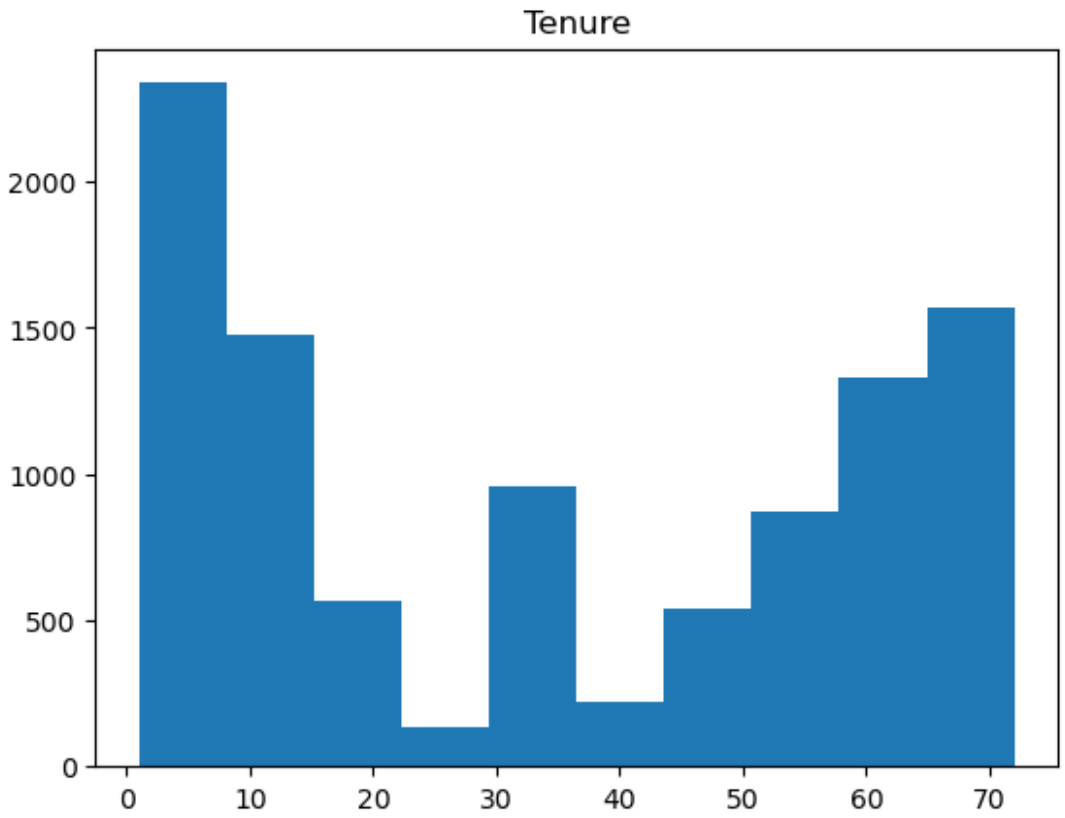

```
#Generate new distribution for Age variable
plt.hist(df_churn['Age'])
plt.title("Age")

Text(0.5, 1.0, 'Age')
```



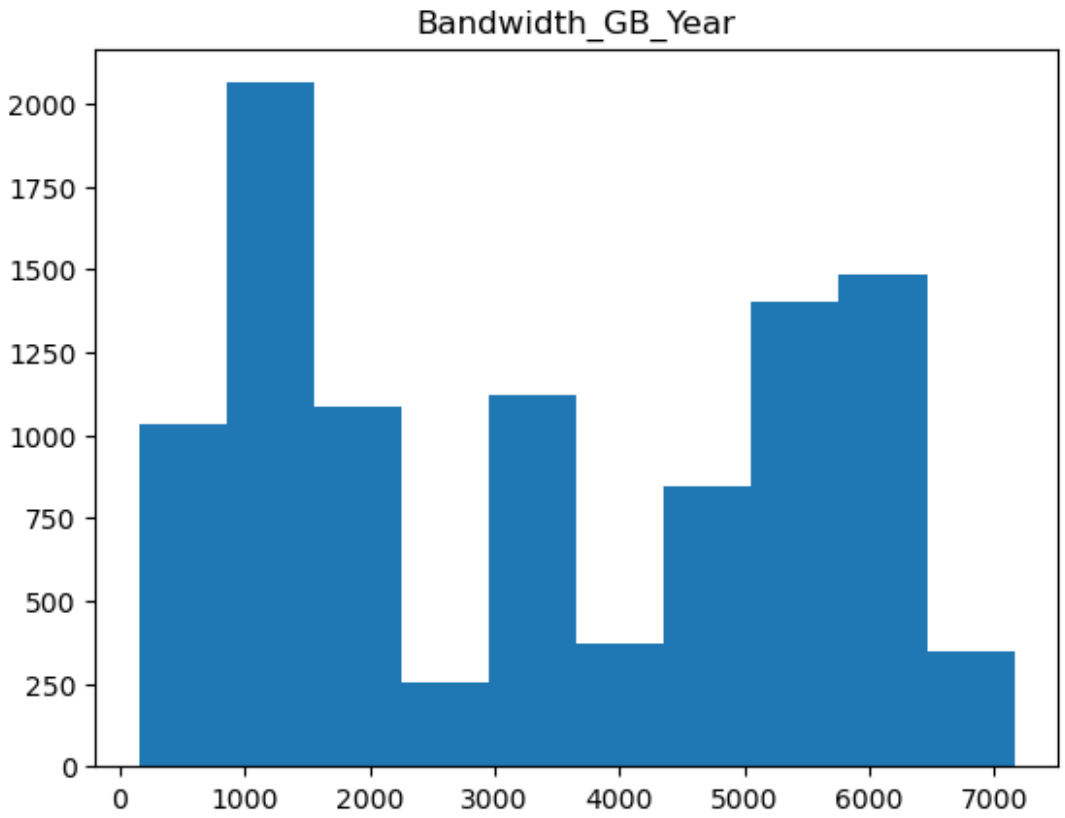

```
#Generate distribution for Income variable
plt.hist(df_churn['Income'])
plt.title("Income")

Text(0.5, 1.0, 'Income')
```




```
#Generate distribution for Tenure variable
plt.hist(df_churn['Tenure'])
plt.title("Tenure")

Text(0.5, 1.0, 'Tenure')
```

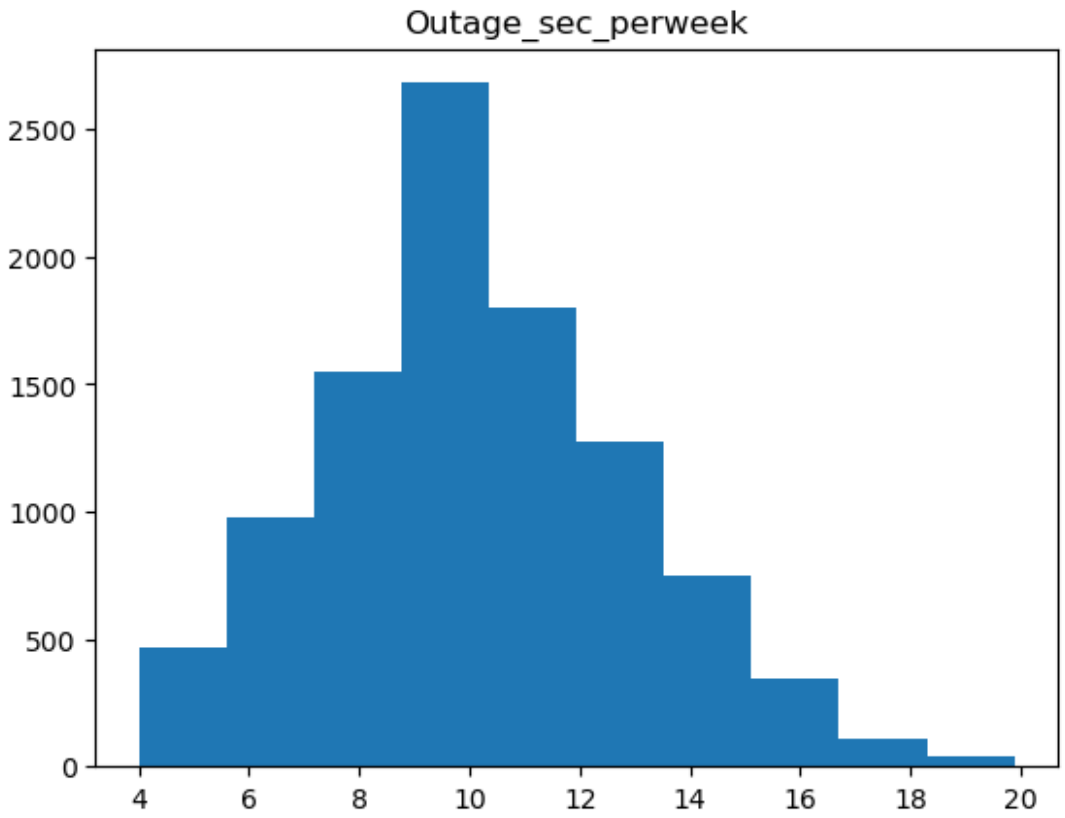



```
#Generate distribution for Bandwidth_GB_Year variable
plt.hist(df_churn['Bandwidth_GB_Year'])
plt.title("Bandwidth_GB_Year")

Text(0.5, 1.0, 'Bandwidth_GB_Year')
```




```
#Generate distribution for Outage_sec_perweek variable
plt.hist(df_churn['Outage_sec_perweek'])
plt.title("Outage_sec_perweek")
```

```
Text(0.5, 1.0, 'Outage_sec_perweek')
```




```
print(df_churn['Phone'].value_counts())
```

```
Yes    9154
No      846
Name: Phone, dtype: int64
```

```
print(df_churn['Techie'].value_counts())
```

```
No     8743
Yes    1257
Name: Techie, dtype: int64
```

```
print(df_churn['TechSupport'].value_counts())
```

```
No     6626
Yes    3374
Name: TechSupport, dtype: int64
```

D4.

Please see the treatment input code below. An executable copy has been attached for review. Please note that the magic command "%%capture" was used to prevent Jupyter Notebook from automatically generating an output.

```
%%capture

#Based on the skew of the distribution, the missing values will be
imputed with the median of the Children variable
df_churn['Children'].fillna(df_churn['Children'].median(), inplace =
True)

%%capture

#Based on the skew of the distribution, the missing values will be
imputed with the mean of the Age variable
df_churn['Age'].fillna(df_churn['Age'].mean(), inplace = True)

%%capture

#Based on the skew of the distribution, the missing values will be
imputed with the median of the Income variable
df_churn['Income'].fillna(df_churn['Income'].median(), inplace = True)

%%capture

#Based on the skew of the distribution, the missing values will be
imputed with the median of the Income variable
df_churn['Tenure'].fillna(df_churn['Tenure'].median(), inplace = True)

%%capture

#Based on the skew of the distribution, the missing values will be
imputed with the median of the Income variable
df_churn['Bandwidth_GB_Year'].fillna(df_churn['Bandwidth_GB_Year'].med
ian(), inplace = True)

%%capture

#For categorical variables "Techie", "Phone" and "Income", impute with
mode

df_churn['Techie'] =
df_churn['Techie'].fillna(df_churn['Techie'].mode()[0])

df_churn['Phone'] = df_churn['Phone'].fillna(df_churn['Phone'].mode()
[0])

df_churn['TechSupport'] =
df_churn['TechSupport'].fillna(df_churn['TechSupport'].mode()[0])

%%capture

#Impute outliers in "Outage_sec_perweek" with the median
df_churn["Outage_sec_perweek"] =
np.where(df_churn["Outage_sec_perweek"] < 4, np.nan,
df_churn["Outage_sec_perweek"])
```

```
df_churn["Outage_sec_perweek"] =
np.where(df_churn["Outage_sec_perweek"] > 20, np.nan,
df_churn["Outage_sec_perweek"])

#Impute NaN values with median
df_churn["Outage_sec_perweek"].fillna(df_churn["Outage_sec_perweek"].m
edian(), inplace=True)

%%capture

#Re-express categorical variables

#Step 1: Find unique values in Education column
print(df_churn['Education'].unique())

#Step 2: Create dictionary to ordinally encode Education values

dict_edu = {"Education":
                {"Master's Degree": 18,
                        "Regular High School Diploma": 12,
                        "Doctorate Degree":24,
                        "No Schooling Completed": 0,
                        "Associate's Degree": 15,
                        "Bachelor's Degree": 16,
                        "Some College, Less than 1 Year": 13,
                        "GED or Alternative Credential": 12,
                        "Some College, 1 or More Years, No Degree":
14,
                        "9th Grade to 12th Grade, No Diploma": 12,
                        "Nursery School to 8th Grade": 8,
                        "Professional School Degree": 20,
                }
           }

#Step 3: Reexpress Education column as numeric
df_churn.replace(dict_edu, inplace = True)

#Step 4: Confirm values have been re-expressed
print(df_churn['Education'].unique())
```

D5.

Please see the attached CSV file containing the now cleaned data.

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

D6.

The data is now clean, however, there might be a few disadvantages with the methods I used. I chose mean and median imputation as my method of dealing with missing values in "Children", "Age", "Income", "Tenure", and "Bandwidth_GB_Year". Although this was a quick solution to the problem, the disadvantages are that the original distributions were impacted and there were likely changes in correlation and covariance between the variables (Gowtham, 2022).

For the categorical variables "Phone", "Techie", and "TechSupport" I used the mode to impute missing values. A disadvantage of this is that the "Yes" category may now be overrepresented (Singhal, 2023). This is reflected in the high ratio of "Yes" to "No" responses for each of these variables.

My decision to retain outliers in “Population”, “Children”, “Income”, “Email”, “Contacts”, “Yearly_equip_failure”, and “MonthlyCharge” could also have some drawbacks. Retaining outliers could influence regression results and possible reduce the statistical significance of relationships between variables (Sullivan et al, 2021). Additionally, the decision to impute the outliers in "Outage_sec_perweek" might have introduced bias by distorting the variable's distribution.

D7.

There are a few challenges a data analyst might face by using the cleaned data. Because I used univariate imputation to treat missing data and outliers, it's possible that the relationships between the variables have been distorted (Singhal, 2023). This could influence the results of Principal Component Analysis because it relies on these relationships to determine the best groupings. Additionally, the use of imputation to treat data quality issues might lead to misleading assumptions about variability. This could present a challenge if the analyst tries to perform a statistical analysis and verify which factors have the greatest influence on customer churn.

E1.

The variables used in principal component analysis are "Income", "Outage_sec_perweek", "Yearly_equip_failure", "Tenure", "MonthlyCharge", and "Bandwidth_GB_Year". These are all the continuous quantitative variables in the churn dataset. The code used to generate the PCA loadings matrix has been provided below.

```
quant = df_churn[
    ["Income",
     "Outage_sec_perweek",
     "Yearly_equip_failure",
     "Tenure",
     "MonthlyCharge",
     "Bandwidth_GB_Year"]
]

quant_normalized = (quant - quant.mean())/quant.std()
```

```
pca = PCA(n_components=quant.shape[1])

pca.fit(quant_normalized)
```

```
PCA(n_components=6)
```

```
quant_pca = pd.DataFrame(pca.transform(quant_normalized),
columns = ['PC1','PC2','PC3','PC4','PC5','PC6'])

loadings = pd.DataFrame(pca.components_.T,
columns = ['PC1','PC2','PC3','PC4','PC5','PC6'],
index = quant.columns)
loadings
```

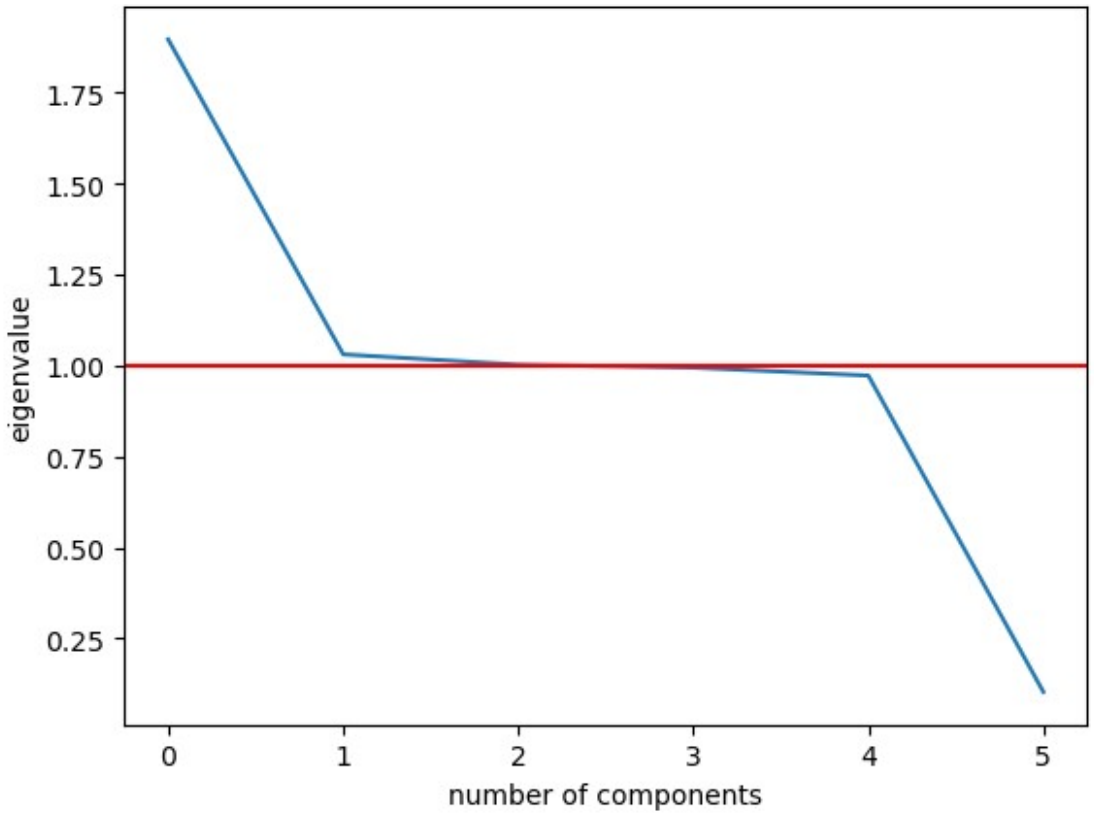
```
                            PC1       PC2       PC3       PC4       PC5
\
Income                 0.005748 -0.287677 -0.565963  0.772022  0.029600

Outage_sec_perweek     0.007073  0.671294  0.014840  0.234023  0.703024

Yearly_equip_failure   0.015202 -0.096773  0.812385  0.563697 -0.112574

Tenure                 0.705603 -0.048564  0.000467 -0.023768  0.037973

MonthlyCharge          0.042359  0.674425 -0.139536  0.175608 -0.700546

Bandwidth_GB_Year      0.707118  0.005763 -0.005120 -0.007537 -0.000779


                            PC6
Income                 0.000761
Outage_sec_perweek    -0.009169
Yearly_equip_failure  -0.002647
Tenure                -0.705520
MonthlyCharge         -0.047773
Bandwidth_GB_Year      0.707013
```

E2.

The code below was used to generate a scree plot, which measures eigenvalues for each principal component. I came to the conclusion that PC1, PC2, and PC3 are the most important principal components and should be retained. The scree plot below shows that these principal components have eigenvalues greater than or equal to 1. This means that they conform to the Kaiser Rule and should be retained is principal components.

```
#Generate covariance matrix
cov_matrix = np.dot(quant_normalized.T, quant_normalized) /
quant.shape[0]

#Generate eigenvalues for each component
```

```
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector))
for eigenvector in pca.components_]

#Generate scree plot with eigenvalues plotted
plt.plot(eigenvalues)
plt.xlabel('number of components')
plt.ylabel('eigenvalue')
plt.axhline(y=1, color="red")
plt.show()
```



E3.

Principal Component Analysis can provide numerous benefits to the organization. One of the biggest advantages of PCA is dimensionality reduction. This helps to simplify large amounts of data and identify only the most important features. This is very useful to the organization because of the large amounts of variables being reviewed. After performing PCA using the churn dataset, we can gain strong insight using only 3 principal components rather than relying on all of the individual variables.

PCA would also be helpful if the organization wishes to conduct more in-depth analyses. By reducing the amount of features in the data, the amount of resources necessary to perform complex calculations is also reduced (Sullivan, 2023). This might make it easier, faster, and more cost-effective to perform an analysis in the future.

F.

Please see the link below for the Panopto recording.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=ad24302b-8f6a-43f2-a582-b0e20001a84c

G. Third Party Code Citations

pandas.DataFrame.quantile. (n.d.). pandas. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.quantile.html

H. Content References

Gowtham, S. R. (2022, November 23). Handling Missing Values With Univariate Imputation. Medium. https://medium.com@/@gowthamsr37/handling-missing-values-with-univariate-imputation-2dfd25e86c17#:~:text=Disadvantages%3A%201%20The%20original%20data%20distribution%20gets%20impacted.,changes%20in%20co

Sullivan, J. H., Warkentin, M., Wallace, L. (2022, November 21). So many ways for assessing outliers: What really works and does it matter? Journal of Business Research, 132, 530-543. https://doi.org/10.1016/j.jbusres.2021.03.066

Sullivan, J. (n.d.). What is PCA and how can I use it?. Statistics By Jim. https://statisticsbyjim.com/basics/principal-component-analysis/

Singhal, S. (2023, August 29). Defining, Analysing, and Implementing Imputation Techniques. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/defining-analysing-and-implementing-imputation-techniques/#:~:text=The%20production%20model%20will%20not%20know%20what%20to,not%20know%20what%20to%20do%20w