D206 Task 1 Performance Assessment

Hillary Osei (Student ID #011039266)

Western Governors University, College of Information Technology

Program Mentor: Dan Estes

August 22, 2024

Table of Contents

Part I: Research Question and Variables

Section A1. Research Question

The research question addressed using the 'churn' dataset is: Which types of customers are more likely to terminate their services? This question is crucial for telecommunications companies aiming to improve customer retention. Investigating this question will help these companies refine their customer service strategies to boost satisfaction and allocate resources more efficiently, strengthening customer relationships.

Section A2. Variables

The churn dataset has the following 50 columns:

Column Name	Data Type	Description	Example
Case Order	Qualitative	Placeholder variable to keep the original order of the raw data file	7
Customer_id	N/A	Unique customer identifier	U335188
Interaction	Qualitative	Unique identifier for customer's transactions, technical support, and sign-ups	6630d501-838c-4be4-a59c-6f58c814ed6a
City	Qualitative	City customer resides in	Pioneer
State	Qualitative	State customer resides in	TN
County	Qualitative	County customer resides in	Scott
Zip	Qualitative	Zip code customer resides in	37847
Lat	Quantitative	GPS coordinates of customer's residence	36.4342
Lng	Quantitative	GPS coordinates of customer's residence	-84.27892
Population	Quantitative	Number of people living within a mile radius of a customer	2535

Area	Qualitative	The area in which a customer resides	Suburban
TimeZone	Qualitative	Time zone customer resides in	America/New_York
Job	Qualitative	Customer's occupation	Surveyor, hydrographic
Children	Quantitative	Number of children in customer's household	0
Age	Quantitative	Customer's age	49
Education	Qualitative	The highest educational level a customer has attained	Some College, Less than 1 Year
Employment	Qualitative	Customer's employment status	Full Time
Income	Quantitative	Customer's annual income	58634.51
Martial	Qualitative	Customer's marital status	Separated
Gender	Qualitative	Customer's gender	Prefer not to answer
Churn	Qualitative	Whether customer cut off their service within the last month	No
Outage_sec_perweek	Quantitative	Average number of seconds per week system outages happen in customer's neighborhood	6.637258801
Email	Quantitative	Number of emails sent to customer in the last year	20
Contacts	Quantitative	Number of times customer has contacted technical support	2
Yearly_equip_failure	Quantitative	Number of times customer's equipment failed and had to be reset or replaced in the past year	3
Techie	Qualitative	Whether customer considers themselves to be technically inclined	No
Contract	Qualitative	Customer's contract term	Month-to-month

Port_modem	Qualitative	Whether customer has a port modem	Yes
Tablet	Qualitative	Whether customer has a tablet	No
InternetService	Qualitative	Customer's internet service provider	DSL
Phone	Qualitative	If customer has phone service	Yes
Multiple	Qualitative	Whether customer has multiple lines	No
OnlineSecurity	Qualitative	Whether customer has online security add-on	Yes
OnlineBackup	Qualitative	Whether customer has online backup add-on	Yes
DeviceProtection	Qualitative	Whether customer has device protection add-on	No
TechSupport	Qualitative	Whether customer has technical support add-on	No
StreamingTV	Qualitative	Whether customer has streaming TV	No
StreamingMovies	Qualitative	Whether customer has streaming movies	No
PaperlessBilling	Qualitative	Whether customer is enrolled in paperless billing	Yes
PaymentMethod	Qualitative	Customer's payment method	Bank Transfer(automatic)
Tenure	Quantitative	Number of months customer has been/had been with provider	8.220686373
MonthlyCharge	Quantitative	Average monthly amount charged to customer	118.3668439
Bandwidth_GB_Year	Quantitative	Average amount of data used by customer in a year	1312.874964
		The following represents responses to an 8-question	

		survey on the importance of multiple factors on a scale of 1-8 (1 = most important, 8 = least important)	
Item1	Qualitative	Timely response	5
Item2	Qualitative	Timely fixes	4
Item3	Qualitative	Timely replacements	4
Item4	Qualitative	Reliability	3
Item5	Qualitative	Options	4
Item6	Qualitative	Respectful response	3
Item7	Qualitative	Courteous exchange	4
Item8	Qualitative	Evidence of active listening	4

Part II: Data-Cleaning Plan (Detection)

Section B1. Detection Methods

The df.duplicated() function detected duplicate entries in the churn dataset. This function returns a Boolean Series in which each value indicates whether a row is a duplicate (True) or not (False).

For identifying missing values across all columns, the df.isnull().sum() function was used, revealing missing values in the columns 'Children,' 'Age,' 'Income,' 'Techie,' 'Phone,' 'TechSupport,' 'Tenure,' and 'Bandwidth_GB_Year.' The missingno.matrix(df) function was then used to provide a visual representation of the missing data points. Additionally, histograms were created for the quantitative variables using plt.hist(df[['Children', 'Age', 'Income', 'Tenure', 'Bandwidth_GB_Year']]), which provided a visual distribution of these variables and their missing values.

Outliers in quantitative variables were detected using the seaborn.boxplot() function, which helped to visually identify outliers for the following variables: ‘Lat,’ ‘Lng,’ 'Population,' 'Children,' 'Age,' 'Income,' 'Outage_sec_perweek,' 'Email,' 'Contacts,' 'Yearly_equip_failure,' 'Bandwidth_GB_Year,' 'MonthlyCharge,' and 'Tenure.' The Interquartile Range (IQR) method was then applied to calculate the IQR and identify values outside the range.

Categorical variables with missing values, 'Techie,' 'Phone,' and 'TechSupport,' were identified for unique values using df[[‘Techie’, ‘Phone,’ ‘TechSupport’]].unique(). Ordinal encoding was applied to convert these categories into numerical form using dictionary mappings, such as yes_no_dict = {‘No’: 0, ‘Yes’:1} for ‘Techie,’ ‘Phone,’ and ‘TechSupport.’

The “None” values present in the column ‘InternetService’ was retained using the df["InternetService"].fillna("None", inplace=True)

The first ten rows of the columns 'Age,' 'Children,' 'Contacts,' 'Techie,' 'Phone,' 'Tenure,' 'TechSupport,' 'MonthlyCharge,' 'Bandwidth_GB_Year,' 'Income,' 'Email,' 'Yearly_equip_failure,' and 'Outage_sec_perweek’ were examined using df[‘column_name’.head(n=10) o ensure correct data type assignments. The variables ‘Age,’ ‘Tenure,’ and ‘Outage_sec_perweek’ were rounded to the nearest whole number using the .round() function before being converted to integers using the .astype(int) function. Similarly, ‘TechSupport,’ ‘Phone,’ ‘Techie,’ ‘Contacts,’ ‘Children,’ ‘Email,’ ‘Yearly_equip_failure’ variables were also converted to integers using the .astype(int) function. Moreover, the ‘Income,’ ‘Bandwidth_GB_Year,’ and ‘MonthlyCharge’ variables were rounded to two decimal places using the .round(2) function.

Section B2. Detection Methods Justification

The df.duplicated() function detected duplicate rows in the dataset, identifying redundancy that could cause skewed results if not addressed. Removing duplicates ensures that each row in the dataset is unique, leading to more reliable insights.

The “None” values in the ‘InternetService’ variable was retained using df["InternetService"].fillna("None", inplace=True) so that it is not counted as a missing value in the detection process.

To detect missing values, the df.isnull().sum() function counted how many missing rows were in each column, pinpointing where the data was incomplete. The missingno library visualized this missing data, providing imagery of the gaps. Additionally, the plt.hist() function was used to visualize the distribution of columns with missing values, helping to understand how these missing values affect the data.

The seaborn.boxplot() function detected outliers by providing a clear visual summary of the data, displaying the median, quartiles, signs of skewness, data distribution, and outliers in an easily interpretable manner. The lower and upper bounds of the boxplot, known as the "whiskers," are typically set 1.5 times the interquartile range (IQR) from the first quartile (Q1) and the third quartile (Q3), with outliers appearing outside these bounds.

Categorical values were re-expressed using ordinal encoding to facilitate the detection of missing values by assigning numerical values to categories. Without this step, detecting missing values for categorical data would be challenging. Dictionary mapping was used to ensure that numerical values assigned to categorical values (e.g., Yes/No) were accurately represented as 0s and 1s, preventing any errors.

Section B3. Programming Language, Libraries, and Packages Justification

Python was used for data cleaning because of its simple and straightforward syntax, similar to English (Western Governors University, n.d.). It makes it accessible for both beginners and experienced data professionals. The pandas library is used for data manipulation. The programming language R has similar functionality with packages like dplyr. However, pandas is more versatile, allowing programmers to perform a wide range of tasks, like merging and filtering data. On the other hand, dplyr frequently requires chaining with other R packages to do complex tasks, while with pandas, it can be done within one library. (RDocumentation, n.d.) The NumPy package was used for working with arrays and calculating advanced mathematical operations on large data sets (What Is NumPy? — NumPy V2.1 Manual, n.d.). One of its advantages is that it uses less memory and storage space (DataScientest, 2023) Matplotlib.pylot library was used for creating histograms to see visual representations of the data's distribution. This library is specifically helpful in providing multiple ways to customize visualizations, such as adding markers, legends, and titles (Jindal, 2023). Seaborn generates boxplots to visualize outliers in a dataset. Seaborn's ability to create plots with minimal code makes it more appealing than other tools. The missingno library was used to visualize the distribution of missing data within the dataframe. This library is favorable over different tools, such as Python's df.isnull.sum(), because it is helpful for quickly identifying gaps in the dataset. Seaborn was used to generate boxplots to detect outliers. The SciPy package was used to calculate the z-score as well as detect outliers. klearn.decomposition import PCA was used for analyzing PCA and to simplify data by reducing its dimensions.

Part III: Data Cleaning Plan (Treatment)

Section C1. Cleaning Findings

When detecting duplicates, using the df.duplicated() function, it was found that there were 0 duplicates in the dataset. All 10,000 rows returned False for duplicated rows.

In detecting missing values, using df.isnull().sum(), it was discovered that the 'Children,' 'Age,' 'Income,' 'Techie,' 'Phone,' 'TechSupport,' 'Tenure,' and 'Bandwidth_GB_Year' contained missing values. The 'Children' column contained 2,495 missing values. The 'Age' column contained 2,475 missing values. The 'Income' column contained 2,490 missing values. The 'Techie' column

contained 2,477 missing values. The 'Phone' column contained 1,026 missing values. The 'TechSupport' column contained 991 missing values. The 'Tenure' column contained 931 missing values. The 'Bandwidth_GB_Year' column contained 1,021 missing values.

It was found that there were outliers in the following columns: ‘Lat,’ ‘Lng,’ 'Population,' 'Children,' 'Income,' 'Outage_sec_perweek,' 'Email,' 'Contacts,' 'Yearly_equip_failure,' and 'MonthlyCharge.' The ‘Lat’ variable contained 158 outliers, ranging from 17.96612 to 70.64066. The ‘Lng’ variable contained 273 outliers, ranging from -171.68815 to -122.5884. The 'Population' variable contained 937 outliers, ranging from 31,816 to 111,850. The 'Children' variable contained 451 outliers, ranging from 7.0 to 10.0. The 'Income' variable contained 759 outliers, ranging from $78,272.96 to $25,8900.70. The 'Outage_sec_perweek' variable contained 491 outliers, ranging from 32.58126 to 47.04928. The 'Email' variable contained 38 outliers, ranging from 1 to 23. The 'Contacts' variable contained eight outliers, ranging from 6 to 7. The 'Yearly_equip_failure' variable contained 94 outliers, ranging from 3 to 6. The 'MonthlyCharge' variable contained five outliers, ranging from $298.1730235 to $315.8786.

Section C2. Justification of Mitigation Methods

There was no need to treat duplicates in the dataset since none were found. Univariate imputation was performed on all missing values by replacing the value with either the variable's mean, median, or mode to treat the missing values. This method was preferred over treatment methods such as backward/forward fill and deletion/elimination because it preserves the integrity of the data. Because univariate imputation fills in missing data based on the distribution of the data, it helps keep its overall structure. Backward/Forward fill may not be feasible because it requires there to be neighboring values. If there is a long sequence of missing data, this method will not have its intended effect. Deletion/elimination can also introduce bias if the missing data is not distributed randomly. For example, if customers making less than $40K tend to have more missing data, the dataset will be skewed towards wealthier customers.

The distribution of the variables determines whether values will be imputed using the mean, median, or mode. Using plt.hist() to create a histogram, the 'Age' column was found to have a uniform distribution, meaning it was equally spread. The missing values in the 'Age' column were imputed with the mean of the column for the data to have a normal, symmetric distribution. The 'Income' column had a positively skewed distribution. Therefore, the missing values were imputed with the median. The 'Tenure' column had a bimodal distribution, so it could either be imputed with the mode or median. Imputing with the median was ultimately decided. The 'Bandwidth_GB_Year' column had a bimodal distribution and was imputed with the median. The categorical variables 'Techie,' 'Phone,' and 'TechSupport,' which were re-expressed into numerical values, were imputed using the mode, a standard option for categorical variables.

The outliers found in the 'Population,' 'Children,' 'Income,' 'Email,' 'Contacts,' 'Yearly_equip_failure,' and 'MonthlyCharge' were first found using the IQR method. The IQR method is a technique to detect outliers by focusing on the middle 50% of the data. It calculates the difference between the first and third quartiles to find the interquartile range. Data points below Q1 by 1.5 * IQR or above Q3 by 1.5 * IQR are considered outliers. This method was chosen instead of Z-scores because Z-score assumes that the data distribution is normal, whereas, with the IQR method, distribution does not matter.

For the ‘Outage_sec_perweek’ variable, it was initially detected and treated with the IQR method. However, this resulted in a box plot with a straight line meaning that the data was compressed. Instead, the Z-score method was selected to identify outliers more effectively and then these outliers were replaced with the median value. While this approach still resulted in some remaining outliers, the data now better reflects the true variation without the extreme compression associated with the IQR method.

Furthermore, outliers were treated using quartile flooring and capping for the 'Population,' 'Children,' and 'Income,' variables. This technique involves capping the outlier at a specific value above the 90th percentile or floored below the 10th percentile. Values lesser than the 10th percentile are replaced with the 10th percentile value and values greater than the 90th percentile are replaced with the 90th percentile value (Bonthu, 2024). Then, the outliers are deleted, and the rest of the data is copied onto another array. This technique was used because it only partially distorts the data. It simply adjusts outlier values to reasonable values within the IQR range.

For the 'Email,' 'Yearly_equip_failure,' 'Contacts,' and 'MonthlyCharge' variables, there were fewer outliers compared to the other variables, so a more precise technique needed to be applied. Outliers were replaced with NaN and then imputed with the median of each respective variable. This method was used because the median is not affected by outliers, and it also accurately reflects the central tendency of the data, ensuring that the imputed values reflect typical data points.

For the ‘Lat’ and ‘Lng,’ the detected outliers were retained because these are geographical locations which naturally vary. Removing these outliers could lead to a loss of important geographic information that would be important in understanding customer’s locations. Therefore, retaining these outliers ensures that the analysis accurately reflects the full range of customer locations.

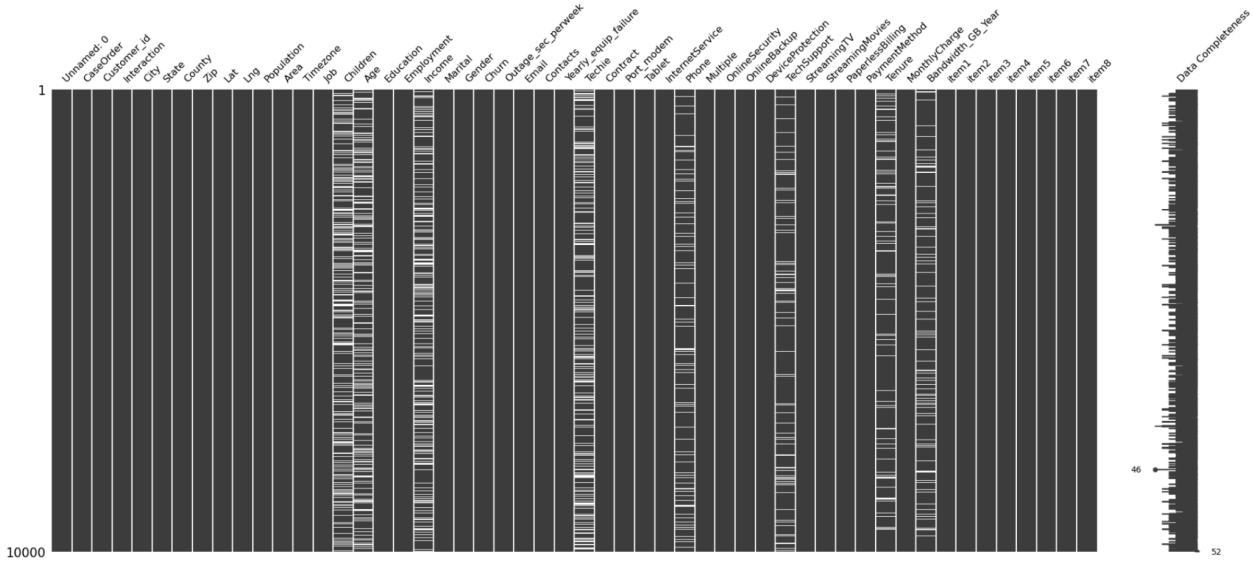
The first step in re-expressing categorical variables involved identifying the unique values using the .unique() function, such as df.Techie.unique(). This was also applied to 'Phone' and 'TechSupport' to determine the appropriate mapping for each category. After identifying that

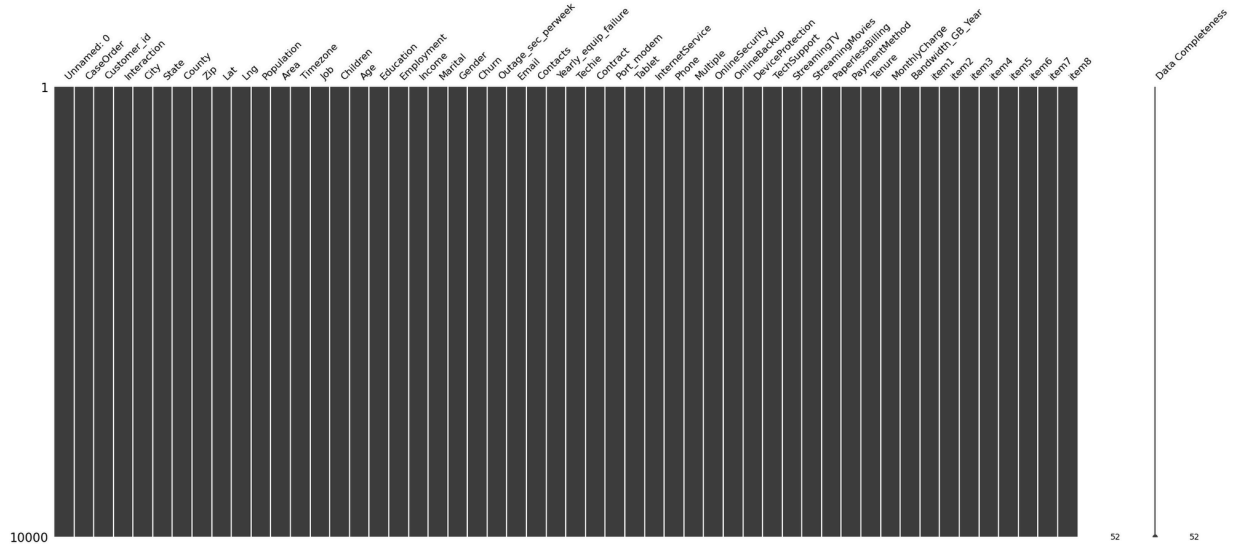
the 'Techie,' 'Phone,' and 'TechSupport' variables contained 'Yes' and 'No' values, the variables were re-expressed using a dictionary mapping. A dictionary, 'yes_no_dict,' was made with the mappings: {'No': 0, 'Yes':1}. The 'Yes' corresponds to a specific characteristic being present, depending on the column. The 'No' corresponds to that specific characteristic being absent. The function infer_object(copy=False) was used to circumvent an error message stating that the replace() function in Pandas will not be available in the upcoming version. All of this was done to make the treatment process for missing values easier.

As part of the final data cleaning process, the values 'Age,' 'Tenure,' and 'Outage_sec_perweek' were rounded to the nearest whole number and converted from float to integer. This was necessary because age, tenure (number of months a customer has been with the provider), and seconds are typically expressed in whole numbers, reflecting real-world conditions more accurately. Similarly, 'TechSupport,' 'Phone,' 'Techie,' 'Contacts,' 'Children,' 'Email,' and 'Yearly_equip_failure' variables were converted to integers to ensure they were in the appropriate format. The 'Children' and 'Contacts' variables represent count data, so expressing them as whole numbers was necessary. The 'MonthlyCharge' and 'Bandwidth_GB_Year' variables were rounded to two decimal points because this is standard practice for financial and measurement data. This ensures consistency in the data and makes it easier to interpret.

Section C3. Summary of Outcomes

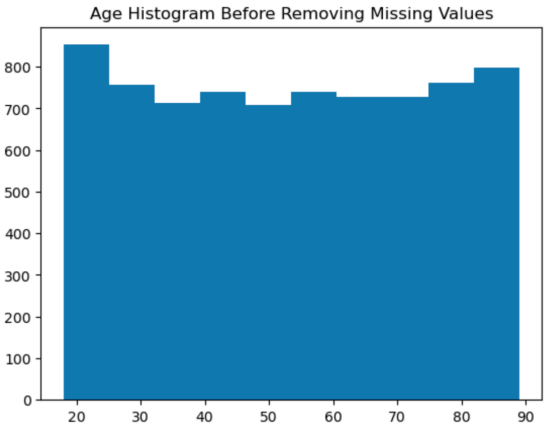
Duplicates were detected using the df.duplicated() function, where it was determined that there were none. Next, missing values were detected using df.isnull().sum() function and then visualized using msno.matrix(df) before and after treatment to ensure no missing data remained in the dataset.

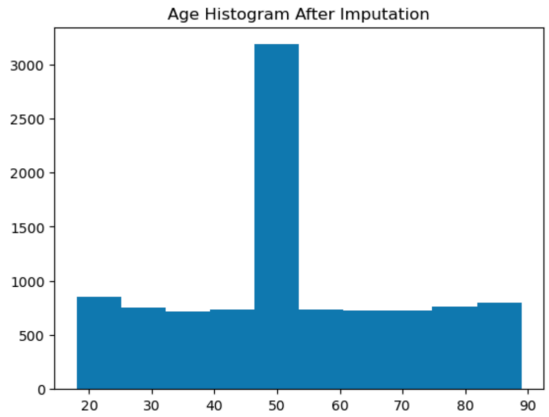




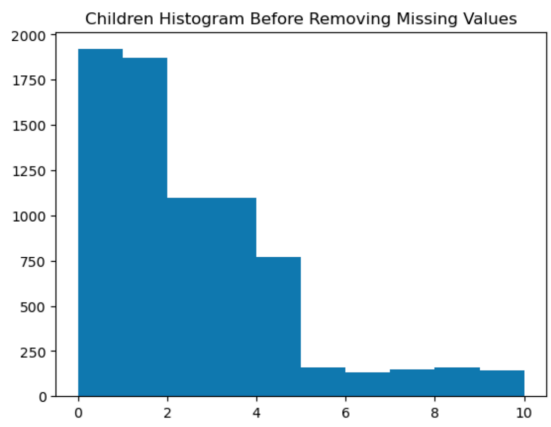
Before the missing values were treated, histograms were created for the 'Age,' 'Children,' 'Tenure,' and 'Bandwidth_GB_Year' variables to view the data distribution. Categorical variables with missing values, 'Techie,' 'TechSupport,' and 'Phone,' were re-expressed using ordinal encoding. Then, missing data was replaced with the mean and median (for quantitative variables) and the mode (for re-expressed categorical variables). Histograms were generated again after treatment to see a new visualization of the data distribution.

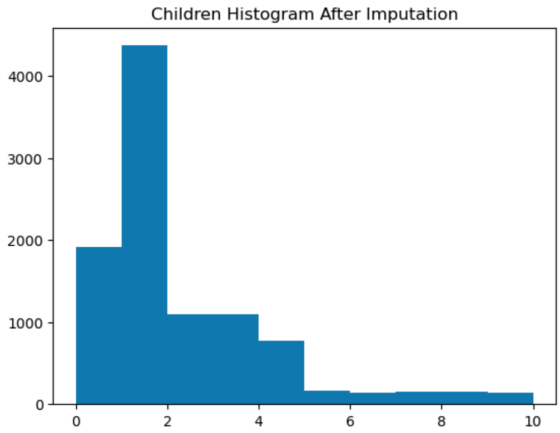
'Age':



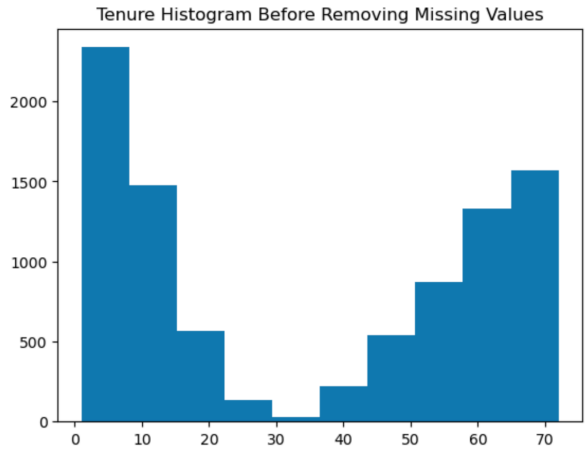


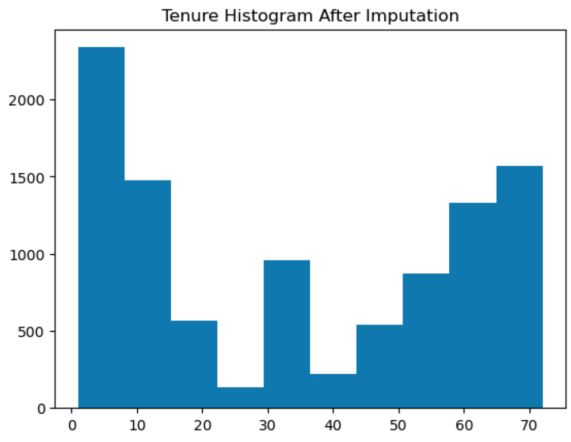
'Children':



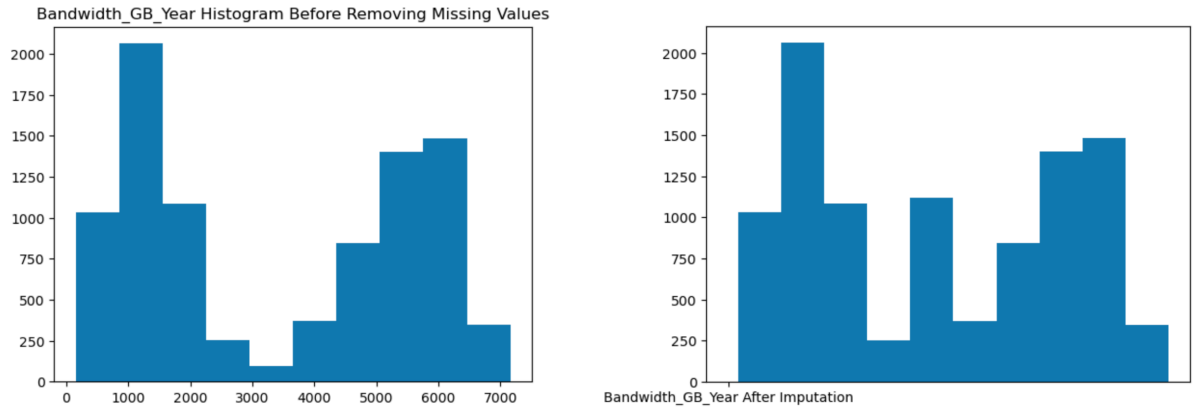


'Tenure':



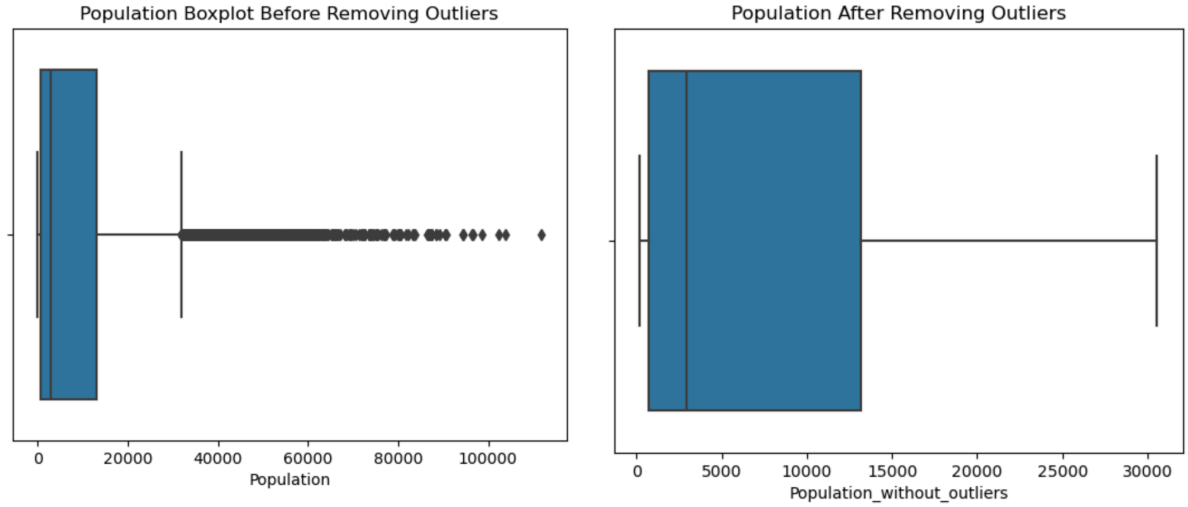


'Bandwidth_GB_Year':

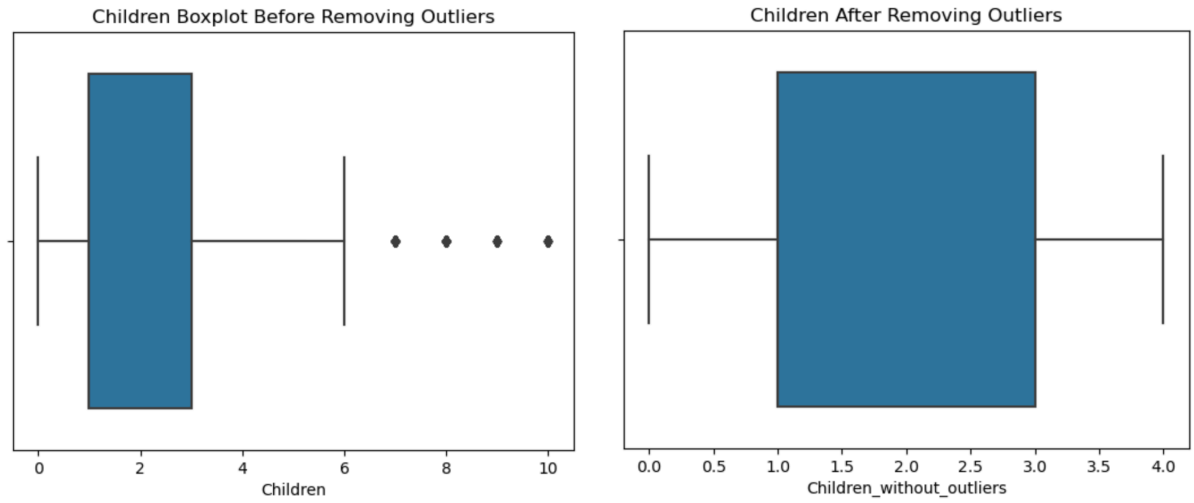


Outliers were detected using boxplots, where it was determined that there were outliers present in ‘Lat,’ ‘Lng,’ 'Population,' 'Children,' 'Age,' 'Income,' 'Outage_sec_perweek,' 'Email,' 'Contacts,' 'Yearly_equip_failure,' 'Bandwidth_GB_Year,' 'MonthlyCharge,' and 'Tenure.' Next, the IQR method was used to find the number of outliers and their values. Then, the quartile flooring and capping technique were used to cap outliers at a value above the 90th percentile value or floored below the 10th percentile value. Below are 'Population,' 'Children,' and 'Income.'

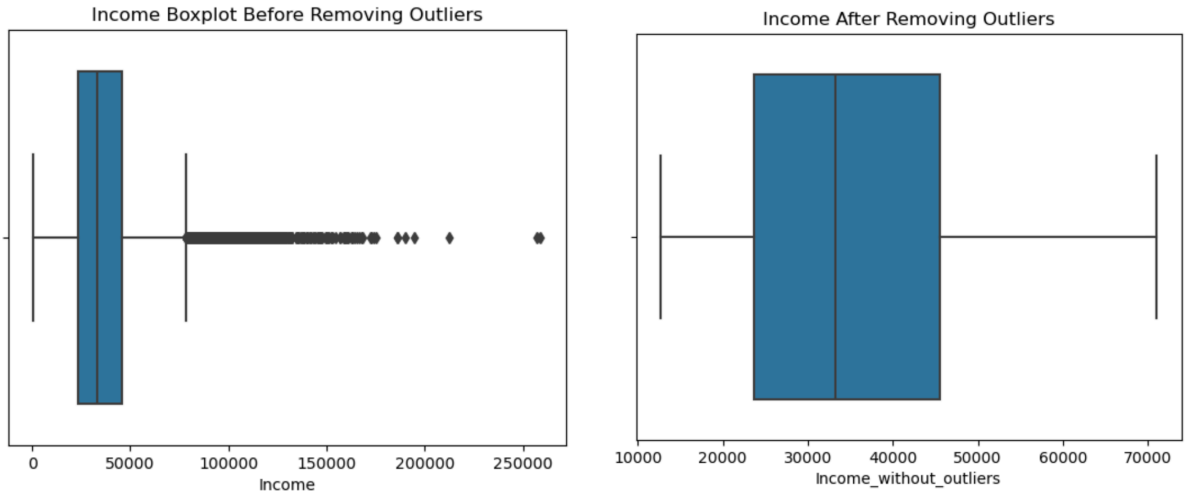
'Population':



'Children':



'Income':



Finally, to ensure the data reflected real-world situations and were easily understood, values in 'Age' and 'Tenure' were rounded and converted from float to integer data type. Here is a preview of the 'Age' column before (left) and after (right) the conversion:

```
0    68.000000
1    27.000000
2    50.000000
3    48.000000
4    83.000000
5    83.000000
6    53.275748
7    53.275748
8    49.000000
9    86.000000
Name: Age, dtype: float64
```

```
0    68
1    27
2    50
3    48
4    83
5    83
6    53
7    53
8    49
9    86
Name: Age, dtype: int64
```

Next, the data types for the re-expressed categorical variables were converted from float to integer to make it consistent with the mapping dictionary created when converting their values to numerical ones. For example, here is the 'Techie' column before and after the conversion:

```
0    0.0
1    1.0
2    1.0
3    1.0
4    0.0
5    0.0
6    1.0
7    1.0
8    0.0
9    0.0
Name: Techie, dtype: float64
```

```
0    0
1    1
2    1
3    1
4    0
5    0
6    1
7    1
8    0
9    0
Name: Techie, dtype: int64
```

Then, 'MonthlyCharge' and 'Bandwidth_GB_Year' were rounded to 2 decimal points to reflect real-world situations and make them easier to understand. Here is 'MonthlyCharge' before and after as an example:

```
0    171.449762
1    242.948015
2    159.440398
3    120.249493
4    150.761216
5    184.401558
6    200.064886
7    114.754111
8    118.366844
9    163.005280
Name: MonthlyCharge, dtype: float64
```

```
0    171.45
1    242.95
2    159.44
3    120.25
4    150.76
5    184.40
6    200.06
7    114.75
8    118.37
9    163.01
Name: MonthlyCharge, dtype: float64
```

Section C4. Limitations

Univariate imputation was selected to treat missing values by replacing them with the mean, median, or mode, depending on the distribution of the variable. While this technique helps maintain the overall structure of the data and is simple to implement, it can also introduce bias by reducing the dataset's variability. It distorts the mean and standard deviation of the dataset and does not consider relationships with other variables (Zhang, 2016), potentially leading to a misrepresentation of the data.

The IQR method uses a fixed threshold to detect outliers, which may only be appropriate for some datasets because it assumes a symmetric distribution around the media. This could lead to an overlook of outliers or a mislabeling of normal values as outliers. Furthermore, the quartile flooring and capping technique for treating outliers adjust outliers to the nearest quartile, which can alter the data distribution, leading to inaccurate representations.

Replacing outliers with NaN and imputing with the median assumes that outliers are invalid data points. This approach risks losing valuable insights that the outliers might offer but also needs to consider the critical variations essential in understanding the data.

Section C5. Impact of Limitations

Data analysts may need help with analyzing the recently cleaned churn dataset, which could lead to inaccurate or inconsistent findings. For example, there is a potential for human

error during cleaning. Because there is no standardized process, data cleaning can be subjective, leading to variations in how data is handled and making it harder to draw reliable conclusions.

Using univariate imputation to treat missing values helps maintain the dataset's structure but can introduce biases by reducing variability and distorting critical metrics like the mean and standard deviation. This distortion can lead to a skewed understanding of customer behavior, potentially causing analysts to overlook important patterns that indicate why specific customers are more likely to churn. Since univariate imputation does not account for relationships between variables, it may fail to capture the crucial components for accurately predicting customer churn.

The IQR method for detecting and handling outliers assumes a symmetrical distribution around the median, which may not be accurate for all variables. This could result in either misidentifying outliers or removing valid data points. Techniques like quartile flooring and capping, which adjust outliers to the nearest quartile, can also alter the data's natural distribution, potentially hiding essential insights.

Replacing outliers with NaN and imputing with the median could lead to losing valuable information about customer groups. These extreme values represent customers who are more likely to churn, and their removal or alteration can prevent analysts from fully understanding these key groups.

Section D1. Mitigation Code

The following code was used for the detection and treatment of duplicates, missing data, outliers, and other data quality issues:

Detection of duplicates:

```
# Check for duplicates in data
df.duplicated()
```

Detection of missing data:

```
# Check for missing values
df.isnull().sum()

# Visualize nullity using missingno
import missingno as msno
import matplotlib.pyplot as plt
msno.matrix(df, fontsize=12, labels=True)
plt.show()
```

```
# Check distribution of 'Children' column
plt.hist(df['Children'])
plt.title('Children Histogram Before Removing Missing Values')

# Check distribution of 'Age' column
plt.hist(df['Age'])
plt.title('Age Histogram Before Removing Missing Values')

# Check distribution of 'Income' column
plt.hist(df['Income'])
plt.title('Income Histogram Before Removing Missing Values')

# Check distribution of 'Tenure' column
plt.hist(df['Tenure'])
plt.title('Tenure Histogram Before Removing Missing Values')

# Check distribution of 'Bandwidth_GB_Year' column
plt.hist(df['Bandwidth_GB_Year'])
plt.title('Bandwidth_GB_Year Histogram Before Removing Missing Values')
```

Re-expression of categorical variables:

```
# Re-express categorical variables for 'Techie', 'Phone', 'TechSupport' columns
yes_no_dict = {'No': 0, 'Yes': 1}
df['Techie'] = df['Techie'].replace(yes_no_dict).infer_objects(copy=False)
df['Phone'] = df['Phone'].replace(yes_no_dict).infer_objects(copy=False)
df['TechSupport'] = df['TechSupport'].replace(yes_no_dict).infer_objects(copy=False)
```

Treatment of missing data:

```
# Perform univariate imputation on 'Children' column
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Children'].fillna(df['Children'].median(), inplace=True)

# Perform univariate imputation on 'Age' column
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Perform univariate imputation on 'Income' column
```

```
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Income'].fillna(df['Income'].median(), inplace=True)

# Perform univariate imputation on 'Tenure' column
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Tenure'].fillna(df['Tenure'].median(), inplace=True)

# Perform univariate imputation on 'Bandwidth_GB_Year' column
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Bandwidth_GB_Year'].fillna(df['Bandwidth_GB_Year'].median(), inplace=True)

# Perform univariate imputation on 'Techie'
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Techie'].fillna(df['Techie'].mode()[0], inplace=True)

# Perform univariate imputation on 'Phone'
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Phone'].fillna(df['Phone'].mode()[0], inplace=True)

# Perform univariate imputation on 'TechSupport'
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['TechSupport'].fillna(df['TechSupport'].mode()[0], inplace=True)
```

Detection of outliers for all quantitative variables:

```
# Create boxplot for 'Lat' to check for outliers
boxplot = seaborn.boxplot(x='Lat', data=df).set(title='Lat Boxplot Before Removing
Outliers')

# Create boxplot for 'Lng' to check for outliers
boxplot = seaborn.boxplot(x='Lng', data=df).set(title='Lng Boxplot Before Removing
Outliers')
```

```
# Create boxplot for 'Population' to check for outliers
import seaborn
boxplot = seaborn.boxplot(x='Population', data=df).set(title='Population Boxplot Before
Removing Outliers')

# Create boxplot for 'Children' to find outliers
boxplot = seaborn.boxplot(x='Children', data=df).set(title='Children Boxplot Before
Removing Outliers')

# Create boxplot for 'Age' to check for outliers
boxplot = seaborn.boxplot(x='Age', data=df).set(title='Age Boxplot')

# Create boxplot for 'Income' to check for outliers
boxplot = seaborn.boxplot(x='Income', data=df).set(title='Income Boxplot Before
Removing Outliers')

# Create boxplot for 'Outage_sec_perweek' to check for outliers
boxplot = seaborn.boxplot(x='Outage_sec_perweek',
data=df).set(title='Outage_sec_perweek Boxplot Before Removing Outliers')

# Use Z-Score to find outlier values in 'Outage_sec_perweek' column
# Based on code from Middleton (n.d., Slide 6, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Z_Score_Outage_sec_perweek'] = stats.zscore(df['Outage_sec_perweek'])
df[['Children','Z_Score_Outage_sec_perweek']].head()
df_outliers = df.query('(Z_Score_Outage_sec_perweek > 3) |
(Z_Score_Outage_sec_perweek < -3)')
print("Number of outliers:", df_outliers.shape[0])
print("Outlier values:", df_outliers['Outage_sec_perweek'].to_dict().values())

# Create boxplot for 'Email' to check for outliers
boxplot = seaborn.boxplot(x='Email', data=df).set(title='Email Boxplot Before Removing
Outliers')

# Create boxplot for 'Contacts' to check for outliers
boxplot = seaborn.boxplot(x='Contacts', data=df).set(title='Contacts Boxplot Before
Removing Outliers')

# Create boxplot for 'Yearly_equip_failure' to check for outliers
```

```
boxplot = seaborn.boxplot(x='Yearly_equip_failure',
data=df).set(title='Yearly_equip_failure Boxplot Before Removing Outliers')

# Create boxplot for 'Tenure' to check for outliers
boxplot = seaborn.boxplot(x='Tenure', data=df).set(title='Tenure Boxplot')

# Create boxplot for 'MonthlyCharge'
boxplot = seaborn.boxplot(x='MonthlyCharge', data=df).set(title='MonthlyCharge
Boxplot Before Removing Outliers')

# Create boxplot for 'Bandwidth_GB_Year'
boxplot = seaborn.boxplot(x='Bandwidth_GB_Year',
data=df).set(title='Bandwidth_GB_Year Boxplot')
```

Treatment of outliers (**Note: The following is for the 'Population' column. Application of this code for other variables can be found in the code attached, titled "d206-churn-pa.ipynb"**)*:*

```
# Use IQR method to find outlier values in 'Population' column
# Based on code from Bonthu, 2024
import numpy as np
outliers = []
def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    # print(q1, q3)
    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    # print(lwr_bound, upr_bound)
    for i in data:
    if (i<lwr_bound or i>upr_bound):
    outliers.append(i)
    return outliers
Population_column = df['Population']
Population_outliers = detect_outliers_iqr(Population_column)
print("Number of Outliers: ", len(Population_outliers))
print("Outliers from IQR method: ", Population_outliers)
```

```
# Treat outliers in Population using quartile capping and flooring
# Based on code from Bonthu, 2024
# Calculate Q1 and Q3
tenth_percentile = np.percentile(df['Population'], 10)
ninetieth_percentile = np.percentile(df['Population'], 90)
# Apply flooring and capping
b = np.where(df['Population']<tenth_percentile, tenth_percentile, df['Population'])
b = np.where(b>ninetieth_percentile, ninetieth_percentile, b)
print("New array:",b)


# Use IQR method to find outlier values in 'Email' column
# Based on code from Bonthu, 2024
email_outliers = []
def detect_email_outliers_iqr(data):
        data = sorted(data)
        q1 = np.percentile(data, 25)
        q3 = np.percentile(data, 75)
        # print(q1, q3)
        IQR = q3-q1
        lwr_bound = q1-(1.5*IQR)
        upr_bound = q3+(1.5*IQR)
        # print(lwr_bound, upr_bound)
        for i in data:
        if (i<lwr_bound or i>upr_bound):
        email_outliers.append(i)
        return email_outliers
Email_column = df['Email']
Email_outliers = detect_email_outliers_iqr(Email_column)
print("Number of Email Outliers: ", len(Email_outliers))
print("Outliers from IQR method: ", Email_outliers)

# Replace outliers in 'Email' with NaN
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Email'] = np.where((df['Email'] < 4) | (df['Email'] > 20), np.nan, df['Email'])

# Impute null values in 'Email' with the median
```

```
# Based on code from Middleton (n.d., Slide 12, "Welcome To Getting Started With
Detecting and Treating Outliers")
df['Email'].fillna(df['Email'].median(), inplace=True)
```

Data Type Verification and Rounding:

```
# Round values in Age and change data type to integer
df['Age'] = df['Age'].round().astype(int)
# Recheck first 10 values in Age
df['Age'].head(n=10)

# Round values in Outage_sec_perweek and convert to int
df['Outage_sec_perweek'] = df['Outage_sec_perweek'].round().astype(int)
# Check values
df['Outage_sec_perweek'].head(n=10)

# Round values in Tenure to nearest whole number and convert to int
df['Tenure'] = df['Tenure'].round().astype(int)
# Recheck first 10 values in Tenure
df['Tenure'].head(n=10)

# Convert values in Children to int
df['Children'] = df['Children'].astype(int)
# Recheck first 10 values in Children
df['Children'].head(n=10)

# Convert values in Contacts to int
df['Contacts'] = df['Contacts'].astype(int)
# Recheck first 10 values in Contacts
df['Contacts'].head(n=10)

# Convert values in Techie to int
df['Techie'] = df['Techie'].astype(int)
# Recheck first 10 values in Techie
df['Techie'].head(n=10)

# Convert values in Phone to int
df['Phone'] = df['Phone'].astype(int)
# Recheck first 10 values in Phone
df['Phone'].head(n=10)
```

```
# Convert values in TechSupport to int
df['TechSupport'] = df['TechSupport'].astype(int)
# Recheck first 10 values in TechSupport
df['TechSupport'].head(n=10)

# Convert values in Email to int
df['Email'] = df['Email'].astype(int)
# Recheck first 10 values in Email
df['Email'].head(n=10)

# Convert values in Yearly_equip_failure to int
df['Yearly_equip_failure'] = df['Yearly_equip_failure'].astype(int)
#Check values
df['Yearly_equip_failure'].head(n=10)

# Round values in MonthlyCharge to 2 decimal points
df['MonthlyCharge'] = df['MonthlyCharge'].round(2)
# Recheck first 10 values in MonthlyCharge
df['MonthlyCharge'].head(n=10)

# Round values in Bandwidth_GB_Year to 2 decimal points
df['Bandwidth_GB_Year'] = df['Bandwidth_GB_Year'].round(2)
# Recheck first 10 values in Bandwidth_GB_Year
df['Bandwidth_GB_Year'].head(n=10)

# Round values in Income to 2 decimal points
df['Income'] = df['Income'].round(2)
# Recheck first 10 values values in Income
df['Income'].head(n=10)
```

Section D2. Clean Data

See the cleaned data attached, titled “clean_churn_data1.csv”

Section E1. Principal Components

Principal Component Analysis (PCA) is a dimensionality reduction technique that simplifies complex datasets by identifying and focusing on the most important features or principal components. These components capture most of the data's variance, allowing analysts to work with a smaller, more manageable set of variables without losing significant information. PCA is widely used to enhance data analysis, visualization, and the performance of machine learning models by reducing noise and highlighting key patterns (*What Is Principal Component Analysis (PCA) & How to Use It?*, n.d.)

Code used for PCA:

```
from sklearn.decomposition import PCA
# Define variables for PCA
# Based on code from Middleton (n.d., slide 11, "Welcome To Getting Started With
Principal Component Analysis (PCA)")
churn_features =  df[['Age', 'Children','Income', 'Population', 'Outage_sec_perweek',
'Email','Contacts', 'Tenure', 'Bandwidth_GB_Year', 'Yearly_equip_failure']]
# Normalize data using standardization (mean normalization)
# Based on code from Middleton (n.d., slide 12, "Welcome To Getting Started With
Principal Component Analysis (PCA)")
normalized_churn_features =
(churn_features-churn_features.mean())/churn_features.std()
# Based on code from Middleton (n.d., slide 13, "Welcome To Getting Started With
Principal Component Analysis (PCA)")
pca = PCA(n_components=churn_features.shape[1])
pca.fit(normalized_churn_features)

# Based on code from Middleton (n.d., slide 13, "Welcome To Getting Started With
Principal Component Analysis (PCA)")
churn_features_pca =
pd.DataFrame(pca.transform(normalized_churn_features),columns=
['PC1','PC2','PC3','P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10'])
# Get PCA loadings
# Based on code from Middleton (n.d., slide 15, "Welcome To Getting Started With
Principal Component Analysis (PCA)")
loadings = pd.DataFrame(pca.components_.T, columns=['PC1','PC2','PC3','PC4', 'PC5',
'PC6', 'PC7', 'PC8', 'PC9', 'PC10'], index=normalized_churn_features.columns)
print(loadings)
```

PCA loadings matrix:

```
                          PC1       PC2       PC3       PC4       PC5  \
Age                 -0.012711 -0.417290 -0.331789  0.328212 -0.203042
Children            -0.002011  0.532367  0.153402 -0.045656  0.357935
Income               0.006126  0.205151 -0.018431  0.658577  0.534572
Population           0.000081 -0.318153  0.459834 -0.092060  0.238434
Outage_sec_perweek   0.019241 -0.057707 -0.403278 -0.498605  0.527074
Email               -0.018090 -0.321521  0.432913 -0.304008  0.189228
Contacts             0.003599 -0.431899 -0.395599  0.057673  0.385535
Tenure               0.706570 -0.020863  0.010204  0.013630 -0.019663
Bandwidth_GB_Year    0.706962  0.005385  0.012329 -0.004802  0.002503
Yearly_equip_failure 0.007139  0.324714 -0.384441 -0.321608 -0.159355

                          PC6       PC7       PC8       PC9      PC10
Age                 -0.310074  0.263905 -0.587319 -0.235574  0.021052
Children            -0.347831  0.261611 -0.070221 -0.606753 -0.018403
Income               0.097393  0.105231 -0.135343  0.446281  0.001353
Population           0.644476  0.191084 -0.315378 -0.265855 -0.000690
Outage_sec_perweek  -0.040323 -0.385599 -0.383858  0.097393 -0.010393
Email               -0.510955  0.381522  0.023283  0.416479  0.004997
Contacts             0.056548  0.251741  0.612872 -0.250590 -0.003234
Tenure              -0.013924  0.011243 -0.006475  0.004318 -0.706588
Bandwidth_GB_Year   -0.008891  0.005471  0.005789 -0.007320  0.706965
Yearly_equip_failure 0.303085  0.675686 -0.093636  0.246379 -0.002628
```

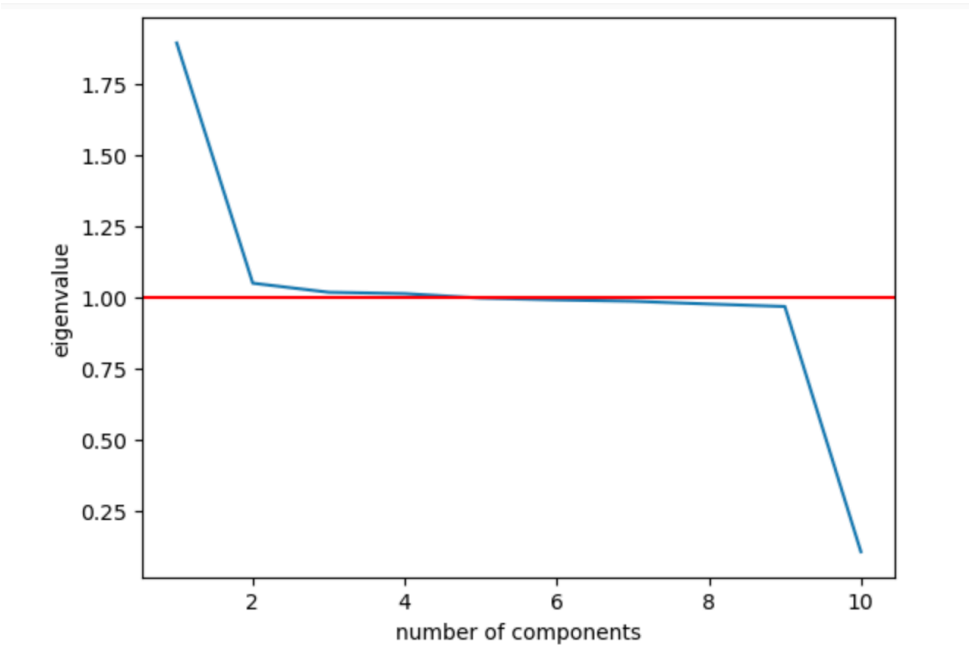
Section E2. Criteria Used

According to the Kaiser Criteria, the number of principal components corresponding to eigenvalues greater than 1 should be retained (Middleton, n.d., Slide 16, “Welcome to Getting Started With Principal Component Analysis"). An eigenvalue greater than 1 indicates that the associated principal component captures significant information from the data. The larger the eigenvalue, the more meaningful the component is in explaining the variation in the dataset. The scree plot, shown below, visually represents these eigenvalues. PC 1-4 should be retained based on this criterion.

Code for eigenvalues:

```
# Select PCs
# Calculate covariance
# Based on code from Middleton (n.d., slide 17, "Welcome To Getting Started With Principal Component Analysis (PCA)")
cov_matrix = np.dot(normalized_churn_features.T, normalized_churn_features) / churn_features.shape[0]
# Calculate Vectors
# Based on code from Middleton (n.d., slide 17, "Welcome To Getting Started With Principal Component Analysis (PCA)")
```

```
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for eigenvector in
pca.components_]
# Create scree plot
# Based on code from Middleton (n.d., slide 17, "Welcome To Getting Started With
Principal Component Analysis (PCA)")
plt.plot(range(1, len(eigenvalues) + 1), eigenvalues)
plt.xlabel('number of components')
plt.ylabel('eigenvalue')
plt.axhline(y=1, color="red")
plt.show()
```



Section E3. Benefits

Using Principal Component Analysis (PCA) can significantly benefit the organization in understanding which types of customers are more likely to terminate their services. By reducing the dimensionality of the data, PCA helps create more efficient and faster models, improving their performance and generalization. (What Is Principal Component Analysis (PCA) & How to Use It?, n.d.) This technique also reduces noise in the data, making it easier to identify the key factors driving customer churn. Additionally, PCA provides uncorrelated features and enhances data visualization, enabling better segmentation and insight into customer behavior. PCA allows the organization to develop more accurate, targeted strategies to retain at-risk customers.

Part IV: Supporting Documents

Section F. Panopto Video

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=514224e7-1180-490e-85f0-b1d401758a25

Section G. Third-Party Code References

Section H. References

Bondu, H. (2024, August 20). *Detecting and Treating Outliers | Treating the odd one out!* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/

DataScientest. (2023, March 13). *NumPy : the most used Python library in Data Science*. DataScientest. https://datascientest.com/en/numpy-the-python-library-in-data-science

Jindal, K. (2023, March 7). *Customization using Matplotlib in Python.* DEV Community. Retrieved August 20, 2024, from https://dev.to/jindalkeshav82/customization-using-matplotlib-in-python-2ji5

Middleton, K. (n.d.). *Welcome to getting started with principal component analysis (PCA) [Slide 11]*. Western Governors University.

Middleton, K. (n.d.). *Welcome to getting started with principal component analysis (PCA) [Slide 12]*. Western Governors University.

Middleton, K. (n.d.). *Welcome to getting started with principal component analysis (PCA) [Slide 16]*. Western Governors University.

Middleton, K. (n.d.). *Welcome to getting started with principal component analysis (PCA) [Slide 17]*. Western Governors University.

Middleton, K. (n.d.). *Welcome to getting started with principal component analysis (PCA) [Slide 13]*. Western Governors University.

Middleton, K. (n.d.). *Welcome to getting started with principal component analysis (PCA) [Slide 15]*. Western Governors University.

Middleton, K. (n.d.). *Welcome to getting started with detecting and treating outliers [Slide 12]*. Western Governors University.

RDocumentation. (n.d.). *Chain: dplyr (version 0.4.3)*. https://www.rdocumentation.org/packages/dplyr/versions/0.4.3/topics/chain

Western Governors University. (n.d.). *R vs. Python: Which programming language should I learn?* R or Python. https://www.wgu.edu/online-it-degrees/programming-languages/r-or-python.html

What is NumPy? — NumPy v2.1 Manual. (n.d.). NumPy -. Retrieved August 20, 2024, from https://numpy.org/doc/stable/user/whatisnumpy.html

What is Principal Component Analysis (PCA) & How to Use It? (n.d.). Bigabid. Retrieved August 20, 2024, from https://www.bigabid.com/what-is-pca-and-how-can-i-use-it/

Zhang, Z. (2016). Missing data imputation: focusing on single imputation. *Annals of translational medicine*, *4(1)*, 9. https://doi.org/10.3978/j.issn.2305-5839.2015.12.38