D206 – Data Cleaning Performance Assessment: Churn Dataset

Bernard Connelly

College of Information Technology, Western Governors University

Arunima Choudhury

August 23, 2024

D206 – Data Cleaning Performance Assessment: Churn Dataset

Part 1: Research Question

A.1. Research Question

"What are the categories which have a direct causal relationship with customer churn? Additionally, does customer demographics or specific customer service detail have a greater impact on customer churn?"

A.2. Description of Dataset

The data set includes 10,000 customer records from a telecommunications company. The records include 52 distinct variables with a variety of data types and total value counts. The primary dependent variable in the research question is the column "Churn." Overall, these data could be broken down into several major categories:

- Unique IDs and Indexes (Customer ID, Transaction ID, etc.)
- Customer Demographics (per billing information or census data)
- Customer Demographics (Self-reported)
- Service Details per the customer account
- The final 8 items were scaled survey responses

Overall analysis of the variables are listed in the table below. The Data Dictionary suggests there are only 50 distinct variables but there are 52 variables in the csv file, suggesting redundant or inaccurate columns. The very first variable "unnamed :0" appears to be a pure duplicate of the CaseOrder variable which is effectively an index. I used the following code to analyze that column, confirming it was a duplicate:

```
matching = df['Unnamed: 0'] == df['CaseOrder']
print(matching)
```

```
0       True
1       True
2       True
3       True
4       True
        ...
9995    True
9996    True
9997    True
9998    True
9999    True
Length: 10000, dtype: bool
```

```
print(matching.value_counts())
print("\nUnique Values:")
print(matching.unique())
##As all values are True, 'Unnamed: 0' is duplicate column
```

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

Unique Values:
[ True]
```

Additionally, lat/long are stored as two separate columns, but are only described as one in the dictionary. Several variables, such as Age and Children are stored in the dataset as floats with a decimal value, but all represent only integers. Finally, the last 8 items in the dataset describe details on a scale from a survey, so they should be categorized as Qualitative Ordinal, despite being whole number integers. Most other variables can be fairly clearly defined as the example implies without too much nuance into determining the types. The table below provides a more thorough description, example and scale of each variable.

name	environment data type	data type	example	Description / Notes
Unnamed: 0	int64	Qualitative Nominal	2	Duplicate column of CaseOrder- not listed in the data dictionary
CaseOrder	int64	Qualitative Nominal	2	An index to preserve the order of the file
Customer_id	object	Qualitative Nominal	S120509	Unique identifier for each customer

Interaction	object	Qualitative Nominal	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	Unique identifier for each transaction / interaction
City	object	Qualitative Nominal	West Branch	Customer Demographics - billing city
State	object	Qualitative Nominal	MI	Customer Demographics - billing state
County	object	Qualitative Nominal	Ogemaw	Customer Demographics - billing county
Zip	int64	Qualitative Nominal	48661	Customer Demographics - billing zip code
Lat	float64	Quantitative Continuous	44.32893	Customer Demographics - billing latitude
Lng	float64	Quantitative Continuous	-84.2408	Customer Demographics - billing longitude
Population	int64	Quantitative Discrete	10446	Customer Demographics - Census Data - population within a radius of the customer
Area	object	Qualitative Nominal	Urban	Customer Demographics - Census Data - Classification of Area Type (Urban vs. Rural vs. Suburban)
Timezone	object	Qualitative Nominal	America/Detroit	Customer Self-reported Demographics - Time Zone listed when customer signed up
Job	object	Qualitative Nominal	Programmer, multimedia	Customer Self-reported Demographics - Job listed by customer during signup
Children	float64	Quantitative Discrete	1.0	Unclear why this is not an integer Customer Self-reported Demographics - Number of children reported
Age	float64	Quantitative Discrete	27.0	Unclear why this is not an integer Customer Self-reported Demographics - Age of customer
Education	object	Qualitative Ordinal	Regular High School Diploma	Customer Self-reported Demographics - Highest level of education
Employment	object	Qualitative Ordinal	Retired	Customer Self-reported Demographics - Current employment status
Income	float64	Quantitative Continuous	21704.77	Customer Self-reported Demographics - Annual income earned

Marital	object	Qualitative Nominal	Married	Customer Self-reported Demographics - Marital status
Gender	object	Qualitative Nominal	Female	Customer Self-reported Demographics - Gender identification (male, female, non-binary)
Churn	object	Qualitative Nominal	Yes	Service Details - Did customer terminate service within 1 calendar month?
Outage_sec_perweek	float64	Quantitative Continuous	12.01454108	Service Details - Average seconds per week of outage in the customer's neighborhood
Email	int64	Quantitative Discrete	12	Service Details - Frequency of emails sent to customer within 1 calendar year
Contacts	int64	Quantitative Discrete	0	Service Details - Frequency customer contacted the company for technical support
Yearly_equip_failure	int64	Quantitative Discrete	1	Service Details - Frequency of time the customer's equipment failed and needed replacement within one calendar year
Techie	object	Qualitative Nominal	Yes	Customer Self-reported demographics - Is the customer technically inclined?
Contract	object	Qualitative Ordinal	Month-to-month	Service Details - Type of contract the customer has (monthly, annual, bi-annual)
Port_modem	object	Qualitative Nominal	No	Service Details - Does the customer have a portable modem?
Tablet	object	Qualitative Nominal	Yes	Service Details - Does the customer have a tablet?
InternetService	object	Qualitative Nominal	Fiber Optic	Service Details - Type of internet (DSL vs. Fiber Optic vs. none)
Phone	object	Qualitative Nominal	Yes	Service Details - Does the customer have phone line?
Multiple	object	Qualitative Nominal	Yes	Service Details - Does the customer have multiple phone lines?
OnlineSecurity	object	Qualitative Nominal	Yes	Service Details - Does the customer have online security?
OnlineBackup	object	Qualitative Nominal	No	Service Details - Does the customer have online backup?

DeviceProtection	object	Qualitative Nominal	No	Service Details - Does the customer have device protection?
TechSupport	object	Qualitative Nominal	No	Service Details - Does the customer have technical support?
StreamingTV	object	Qualitative Nominal	Yes	Service Details - Does the customer have streaming television?
StreamingMovies	object	Qualitative Nominal	Yes	Service Details - Does the customer have streaming movies?
PaperlessBilling	object	Qualitative Nominal	Yes	Service Details - Does the customer have paperless billing?
PaymentMethod	object	Qualitative Nominal	Bank Transfer(automatic)	Service Details - payment type utilized (electronic check vs. mailed check vs. automatic bank transfer vs. automatic credit card withdrawal
Tenure	float64	Quantitative Continuous	1.156680997	Service Details - Number of months the customer has been with the provider
MonthlyCharge	float64	Quantitative Continuous	242.9480155	Service Details - Average value of the customer's monthly charges
Bandwidth_GB_Year	float64	Quantitative Continuous	800.9827661	Service Details - Average amount of data utilized by the customer on an annual basis in Gigabytes
item1	int64	Qualitative Ordinal	3	Survey response: 1(most)-8(least) importance scale. Inquiry = Timeliness of responses from the company
item2	int64	Qualitative Ordinal	4	Survey response: 1(most)-8(least) importance scale. Inquiry = Timeliness of fixes from the company
item3	int64	Qualitative Ordinal	3	Survey response: 1(most)-8(least) importance scale. Inquiry = Timeliness of replacement items from the company
item4	int64	Qualitative Ordinal	3	Survey response: 1(most)-8(least) importance scale. Inquiry = Overall reliability

item5	int64	Qualitative Ordinal	4	Survey response: 1(most)-8(least) importance scale. Inquiry = Options offered
item6	int64	Qualitative Ordinal	3	Survey response: 1(most)-8(least) importance scale. Inquiry = Respectfulness in the exchange
item7	int64	Qualitative Ordinal	4	Survey response: 1(most)-8(least) importance scale. Inquiry = Degree of courtesy in the exchange
item8	int64	Qualitative Ordinal	4	Survey response: 1(most)-8(least) importance scale. Inquiry = Evidence of active listening

Part 2: Data-Cleaning Plan

B.1. Detection Methods

To identify issues with data quality within the dataset, review of duplicates, finding missing values, identifying outliers, and re-expressing variables is required. Each will require different syntax and techniques to properly identify errors and determine if they need to be treated.

Duplicates

To review duplicates, the duplicated() and value_counts() methods will be utilized to identify any rows which match one another. First, duplicated() was utilized to identify any clear instances, and then value_counts() was utilized to confirm all rows returned false responses for duplicates.

```
#Identifying duplicates in the initial dataset
print(df.duplicated())
```

```
0       False
1       False
2       False
3       False
4       False
        ...
9995    False
9996    False
9997    False
9998    False
9999    False
Length: 10000, dtype: bool
```

```
#Completing full comparison of values
print(df.duplicated().value_counts())
```

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

The code returned no duplicates in the dataset. To further verify, the initial index rows were temporarily removed from the dataset and the same test for value_counts() was performed again to ensure the indexes or transaction ID variability was not preventing duplicates from being identified. This was completed by creating a temporary DataFrame using only columns after the 4th via the iloc (iloc() Function in Python, 2024) then performing the same check with duplicated() and value_counts() to confirm the same results.

```
df_temp = df.iloc[:, 3:]
```

```
print(df_temp.duplicated().value_counts())
```

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

This confirmed there were no duplicates in the dataset.

Outliers

For outliers, the first step in detection was to review the DataFrame and narrow down the analysis to only the quantitative variables as qualitative variables cannot typically have outliers.

To do so the select_dtypes (Select Columns with Specific Data Types, 2024) method was utilized to create a temporary DataFrame housing only quantitative variables categorized as int64 or float64.

```
quantitative_variables = df.select_dtypes(include=['float64', 'int64'])
quantitative_variables
```

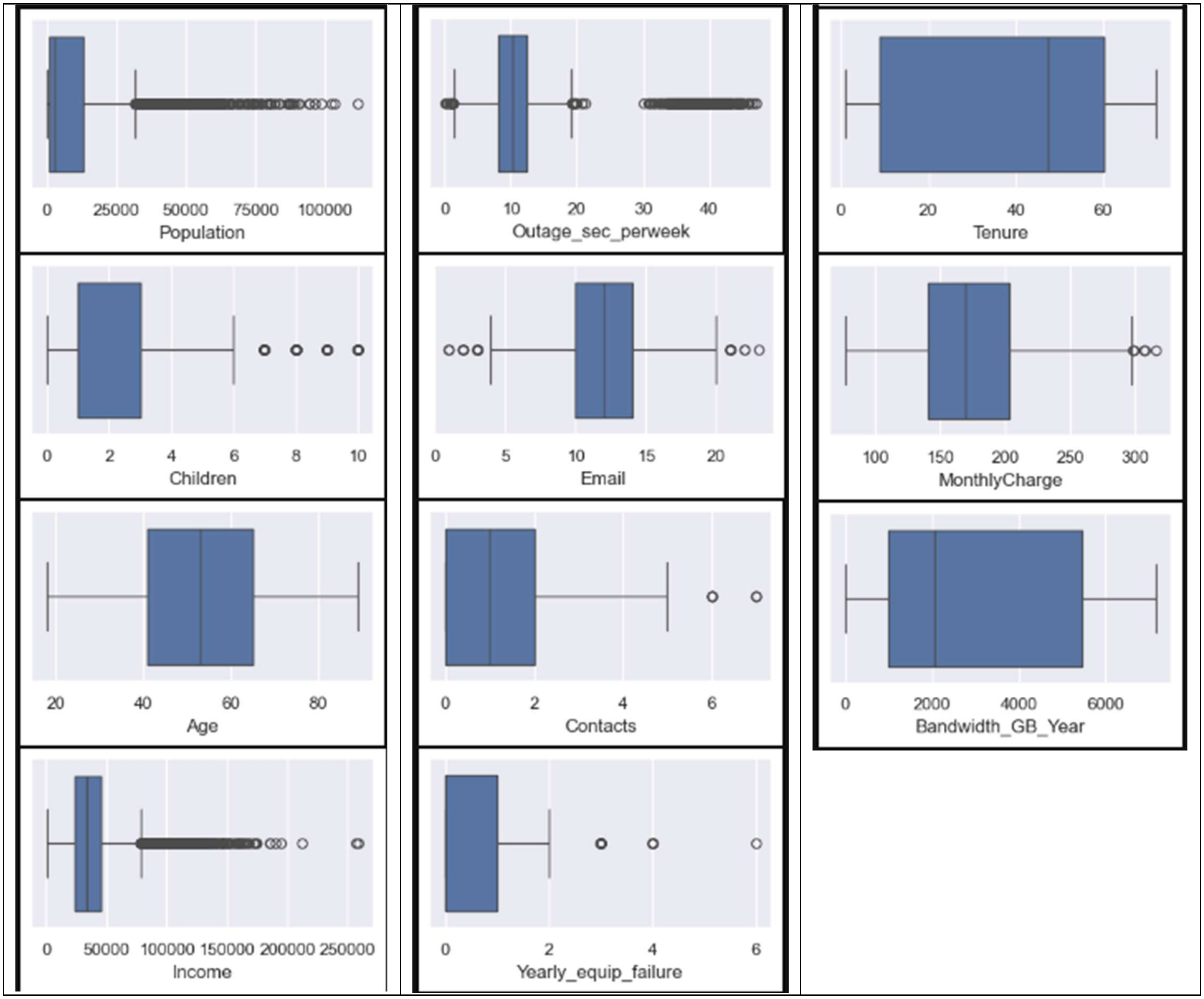
The resultant output included a variety of variables that are not valid for analysis as they were indexes, survey responses, or data which would be retrieved from a HR database such as demographic information or Latitude / Longitude location. These variables were removed from the temporary DataFrame using the drop() function, and the resultant columns were moved to a list using the tolist() method for easy viewing.

```
quantitative_variables = quantitative_variables.drop(columns=['Unnamed: 0', 'CaseOrder', 'Zip', 'Lat',
                                                              'Lng', 'Timely_Response', 'Timely_Fix', 'Timely_Replacement', 'Reliability', 'Options', 'Respectfulness',
                                                              'Courtesy', 'Active_Listening'])
```

```
print(quantitative_variables.columns.tolist())
['Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']
```

With the relevant variables Population, Children, Age, Income, Outage_sec_perweek, Email, Contacts, Yearly_equip_failure, Tenure MonthlyCharge, and Bandwidwth_GB_Year identified, a combination the seaborn package's boxplot function and the describe() function were utilized to review the data and identify outliers.

```
boxplot = sns.boxplot(x='Population', data=df)
plt.show()
boxplot = sns.boxplot(x='Children', data=df)
plt.show()
boxplot = sns.boxplot(x='Age', data=df)
plt.show()
boxplot = sns.boxplot(x='Income', data=df)
plt.show()
boxplot = sns.boxplot(x='Outage_sec_perweek', data=df)
plt.show()
boxplot = sns.boxplot(x='Email', data=df)
plt.show()
boxplot = sns.boxplot(x='Contacts', data=df)
plt.show()
boxplot = sns.boxplot(x='Yearly_equip_failure', data=df)
plt.show()
boxplot = sns.boxplot(x='Tenure', data=df)
plt.show()
boxplot = sns.boxplot(x='MonthlyCharge', data=df)
plt.show()
boxplot = sns.boxplot(x='Bandwidth_GB_Year', data=df)
plt.show()
```




```
print(df[['Population', 'Children', 'Age']].describe())
print(df[['Income', 'Outage_sec_perweek', 'Email']].describe())
print(df[['Contacts', 'Yearly_equip_failure', 'Tenure']].describe())
print(df[['MonthlyCharge', 'Bandwidth_GB_Year']].describe())
```

	Population	Children	Age
count	10000.000000	7505.000000	7525.000000
mean	9756.562400	2.095936	53.275748
std	14432.698671	2.154758	20.753928
min	0.000000	0.000000	18.000000
25%	738.000000	0.000000	35.000000
50%	2910.500000	1.000000	53.000000
75%	13168.000000	3.000000	71.000000
max	111850.000000	10.000000	89.000000

	Income	Outage_sec_perweek	Email
count	7510.000000	10000.000000	10000.000000
mean	39936.762226	11.452955	12.016000
std	28358.469482	7.025921	3.025898
min	740.660000	-1.348571	1.000000
25%	19285.522500	8.054362	10.000000
50%	33186.785000	10.202896	12.000000
75%	53472.395000	12.487644	14.000000
max	258900.700000	47.049280	23.000000

	Contacts	Yearly_equip_failure	Tenure
count	10000.000000	10000.000000	9069.000000
mean	0.994200	0.398000	34.498858
std	0.988466	0.635953	26.438904
min	0.000000	0.000000	1.000259
25%	0.000000	0.000000	7.890442
50%	1.000000	0.000000	36.196030
75%	2.000000	1.000000	61.426670
max	7.000000	6.000000	71.999280

	MonthlyCharge	Bandwidth_GB_Year
count	10000.000000	8979.000000
mean	174.076305	3398.842752
std	43.335473	2187.396807
min	77.505230	155.506715
25%	141.071078	1234.110529
50%	169.915400	3382.424000
75%	203.777441	5587.096500
max	315.878600	7158.982000

Using the above boxplots and details, outliers can be reviewed and identified for treatment.

Missing Values

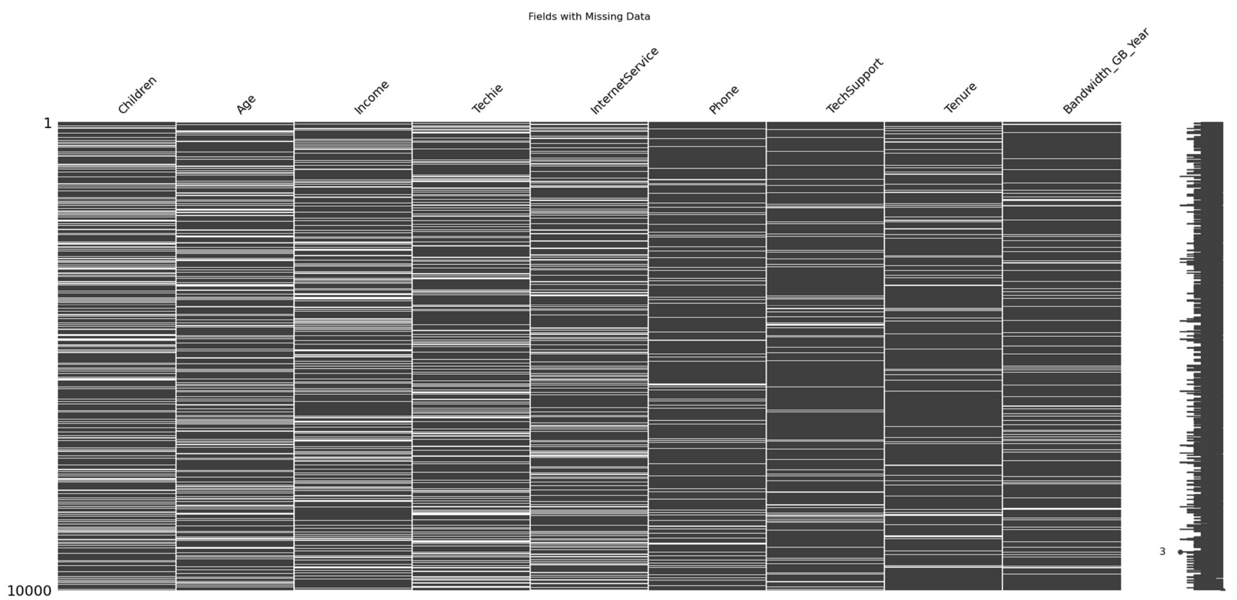
Missing values were primarily identified a combination of the isnull() and sum() methods to identify missing values in the first place, and the msno.matrix() method to visualize where missing data existed. The initial review using df.isnull().sum() produced the following output:

```
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      2129
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
Timely_Response         0
Timely_Fix              0
Timely_Replacement      0
Reliability             0
Options                 0
Respectfulness          0
Courtesy                0
Active_Listening        0
dtype: int64
```

Children, Age, Income, Techie, InternetService, Phone, TechSupport, Tenure, and Bandwidth_GB_Year were identified in scope for review of missing values. The msno.matrix method was utilized to further visualize how much data was missing in each variable.

```
df_columns_missing = df[df.columns[df.isnull().any()]]
msno.matrix(df_columns_missing, fontsize = 14)
plt.title("Fields with Missing Data")
```

The following output was yielded:



Each of the above would need to be assessed and treated individually using different techniques.

Re-Expression of Categorical Variables

To identify variables which would benefit from re-expression, the value_counts() output of each individual variable was reviewed to identify any categorical ordinal variables that had a clear rank or order. Additionally, a loop was utilized for the columns in the DataFrame in conjunction with value_counts and f-strings (f-Strings in Python, 2024), in order to create an output that could easily review each variable. The following code snippit was used:

```
for col in df.columns:
    print(f"Value counts for column '{col}':")
    print(df[col].value_counts())
    print()
```

The above code yielded a long value_counts list for each distinct variable. This allowed for a review of each to identify those which may benefit from being re-expressed.

B.2. Justification of Approach

For duplicated values, duplicated() returns a Boolean value indicating whether or not a row is a duplicate to a previous row. As there are 10,000 records, duplicated() on its own returns too many responses to quickly assess. When used in conjunction with value_counts(), however, a single short output demonstrating 10000 false, or non-duplicated values are returned, which efficiently identifies no duplicates existing. Additional rigor was taken to ensure the identifier rows such as transaction ID or customer ID were not preventing duplicates from occurring. If two customer accounts had all other values identical, they would functionally be duplicates, as no two customers should share all the same address and service information. Temporarily dropping these values using iloc (iloc() Function in Python, 2024) and performing the same check provided additional confirmation of no duplicates to clean.

Identifying missing variables within the dataset was easily discernable with the combination of the isnull() and sum() methods. The isnull() function yields a Boolean response of 'True' if the observation is flagged as None or NaN. The sum() function provides a numerical representation of each instance of 'True' within each variable, demonstrating any instances where cleaning may need to occur. At a glance the specific variables in scope were identified and narrowed down to review. Additionally, msno.matrix() provided an efficient modality to visualize generally how much data was missing from each frame and would need to be effectively cleaned.

Outlier detection can only be performed on Quantitative variables, so the dtypes (Select Columns with Specific Data Types, 2024) method to temporarily narrow down the dataset to variables with float64 or int64 data types. Further individual review was performed to eliminate any variables that are indexes or should not be analyzed due to their true data source being an HR database which would be reviewed by contacting the appropriate department in a real-world scenario. Once the dataset was narrowed, a combination of seaborn's boxplot functionality as well as the describe() method to perform analysis. A boxplot provides an easy to read at a glance modality to see any areas that had significant outliers, and describe() demonstrates individual statistics at a glance for each variable to help identify which variables may need central tendency methods to impute for missing values.

Re-expression identification was performed purely with an f-strings (f-Strings in Python, 2024) loop in conjunction with value_counts(). This is a very manual method, but it allowed for several separate analyses to occur simultaneously. First, by listing out all responses with value_counts(), it was easy to identify any responses or observations which were misspelled, or otherwise mis-identified in comparison with the data dictionary provided. Additionally, comparison of individual responses to ensure continuity in values could occur at a glance. No such instances of 0/1 were used in replace of yes/no within the dataset.

B.3. Justification of Programming Language

For the cleaning of the churn dataset, I opted to utilize the Python programming language. This choice was made primarily because Python was described as having a high degree of "readability and simplicity", and "the consistent syntax of Python makes learning new packages and modules a straightforward task." (Western Governor's University, R or Python). My background in coding is limited to my knowledge of SQL, which does not translate extremely well to object-oriented programming languages. As such, Python was identified as the simpler of the two languages to learn, so I utilized it to review the dataset. Additionally, as Python's syntax is uniform, the packages utilized to perform specific functions would share similar structures and syntax to other details, which made the process of cleaning more effective. For this project, multiple packages needed to be installed and imported to effectively clean the data, as described below:

- Numpy – Provides a large collection of mathematical functions to utilize on arrays and matrices for analysis.
- Pandas – Instrumental in creating DataFrames and series for organizing data, particularly for handling missing values.
- Matplotlib & Seaborn – Each provides different methodologies to review visualizations in Python and identify trends and relationships.
- Missingno – Another visualization method to make it easier to identify missing values in a graphical format.
- SciPy.stats – Useful gathering statistical details in order to identify trends or changes in data, for outliers in particular.

Python provided a consistent and easy to navigate environment whose syntax would remain consistent and repeatable throughout the cleaning process. The alternative, R, had libraries whose syntax varied and would require an increased degree of knowledge and understanding to parse through the steps. Additionally, R was generally described as most effective for research and scientific datasets, whereas the Churn dataset was business oriented.

Part 3: Data Cleaning

C.1. Data Cleaning Findings

Duplicates

Duplicates were not identified in the dataset, despite identifiers being removed to check for any rows which may have been duplicated despite varied indexes or transaction IDs.

Outliers

Initial outlier detection yielded a large volume of quantitative variables to review for outliers. With that said, a number of variables should not be reviewed, explained as follows:

Unnamed: 0, CaseOrder – Indexes

Zip, Lat, Lng – Demographic variables which come from an HR or other database which puts them out of scope for review

The variables at the end of the DataFrame are Likert survey responses, also out of scope for review

After dropping the above from review, the remaining quantitative variables were identified for outlier detection - Population, Children, Age, Income, Outage_sec_perweek, Email, Contacts, Yearly_equip_failure, Tenure MonthlyCharge, and Bandwidwth_GB_Year variables. Analysis of each is as follows:

- Population and Income were both variables with a significant amount of variability between the min and max values. This makes sense as the difference between the amount someone makes as well as how many people live within a mile radius can vary greatly depending on a variety of other details such as their job or whether they live in a rural vs urban area. Additionally, both of these variables have a high frequency of observations on the lower end of the range, which would greatly affect the visibility of the individual boxplots showing high frequency outside 1.5 times the IQR. For both variables, there was not significant variance between the mean and standard deviation, which indicated no treatment was necessary.

- Children flagged anyone with 8, 9 or 10 children as potential outliers. Per the description, the minimum value was 0 and the maximum was 10, both of which are reasonable numbers for the amount of children. Additionally, the mean of 2.1 and standard deviation of 2.15 indicates the vast majority of responses were clustered for this variable. The outliers in this scenario are in an acceptable range and did not need to be treated.
- Outage_sec_perweek had a fairly high frequency of outliers outside the upper IQR, but similar to Income and Population, the variance between mean and standard deviation were not so great to warrant treatment. The critical detail for outlier is it had a number of responses which were negative – which is impossible for the number of seconds the customer was without service within a week. As a result, the low-end values need to be treated.
- Email had a relatively normal distribution with customers receiving anywhere between 1 and 23 emails within 1 calendar year. Potential outliers existed both on the upper and lower ends of the spectrum with a mean of 12 per year. The mean being nearly 4 times the standard deviation implies a high volume of customers received emails clustered around the mean, and those outside of that range could reasonably receive less or more due to service issues or cessation of service. No treatment required.
- Contacts for technical support ranged from 0 to 7 instances annually, with both of these values being within a reasonable range for this to occur. Additionally, the clustered mean at .99 and standard deviation at .98 implies a significant amount of all responses occurred around 1 time per year, and those at the 6 or 7 marks were well within an acceptable

range of occurrence. Before treating it would be relevant to communicate with the company to confirm if the high-end outliers were customers with repeat issues or related concerns to result in an increased frequency.

- Yearly_equip_failure had nearly all responses occur between 0 and 3 times per year. There were 7 instances of a customer having 4 failures, and a single instance of a customer having 6. At first glance, one of these could be considered an outlier, but they all occur within a possible range that issues could arise, and the low frequency would not significantly impact the analysis to warrant cleaning.
- MonthlyCharge returned a boxplot which could at a glance appear to have outliers. With that said, none of the observations were too far outside of the top end whisker to be faulty data, as a smaller percentage of customers could request all services. No treatment required for this variable.
- Age, Tenure and Bandwidth_GB_Year all demonstrated normal boxplots with no visible outliers, so no treatment was required.

Missing Values

The variables with missing values were as follows:

- Children 2495 observations missing
- Age 2475 observations missing
- Income 2490 observations missing
- Techie 2477 observations missing
- Phone 1026 observations missing
- TechSupport 991 observations missing
- Tenure 931 observations missing

- InternetService 2129 observations missing. Of additional note for this variable, one of the valid responses was “None” which Pandas is interpreting as NaN, and thus returning missing values.
- Bandwidth_GB_Year 1021 observations missing. Of additional note for this variable, many responses came back with NaN, which could reasonably be customers who did not have internet through the company, but Pandas was interpreting these as NaN values.

Re-expression:

Review of the detection identified Education as the only variable that had a clear rank or order as a categorical ordinal variable. None of the values came back with mismatched responses for yes/no or 0/1, so no required updates were present.

C.2. Justify Methods for Mitigation

Outliers:

Outlier detection and treatment was performed before missing value was cleaned as any broad updates to missing values would significantly impact the frequency counts of previously missing data. Doing missing data first could create a large discrepancy in outliers for each variable. The only outlier which was treated was Outage_sec_perweek as it had negative values. The first step was to review the exact values that were negative, using the following code:

```
outage_test = df["Outage_sec_perweek"][df["Outage_sec_perweek"] < 0]
print(outage_test)
```

Which yielded:

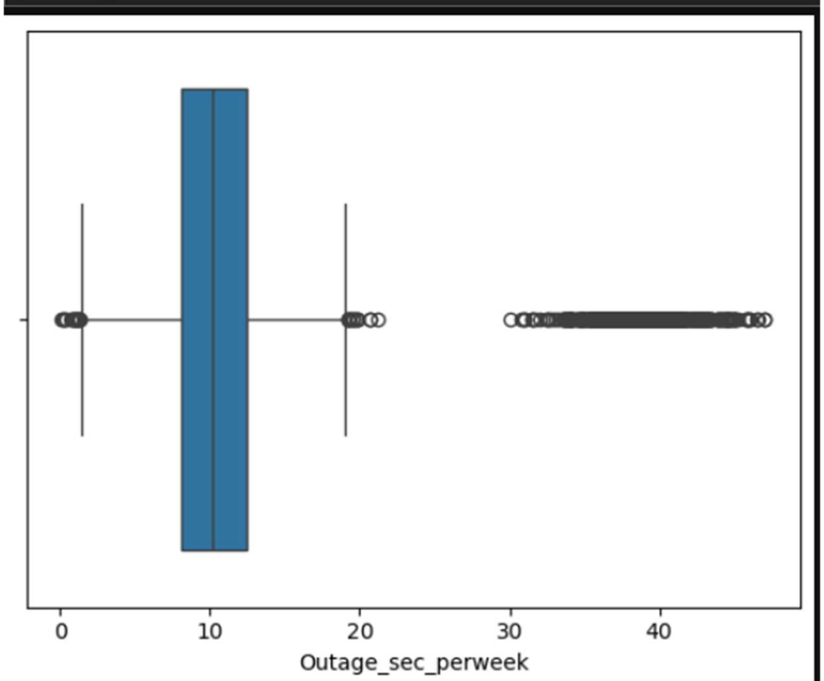
1904	-1.195428
1997	-0.339214
3069	-0.206145
3629	-0.152845
4167	-1.348571
4184	-0.352431
4427	-1.099934
6093	-0.787115
6463	-0.144644
6577	-0.527396
8194	-0.214328

The median was utilized to replace the negative values as this central tendency is not influenced by extreme values and provides a realistic measure to preserve the overall integrity of the dataset. This was completed using the following:

```
df["Outage_sec_perweek"] = np.where(df["Outage_sec_perweek"] < 0, np.nan, df["Outage_sec_perweek"])
df["Outage_sec_perweek"] = df["Outage_sec_perweek"].fillna(df["Outage_sec_perweek"].median())
```

Then validated by checking the boxplot again to confirm negative values were removed:

```
boxplot = sns.boxplot(x='Outage_sec_perweek', data=df)
plt.show()
```



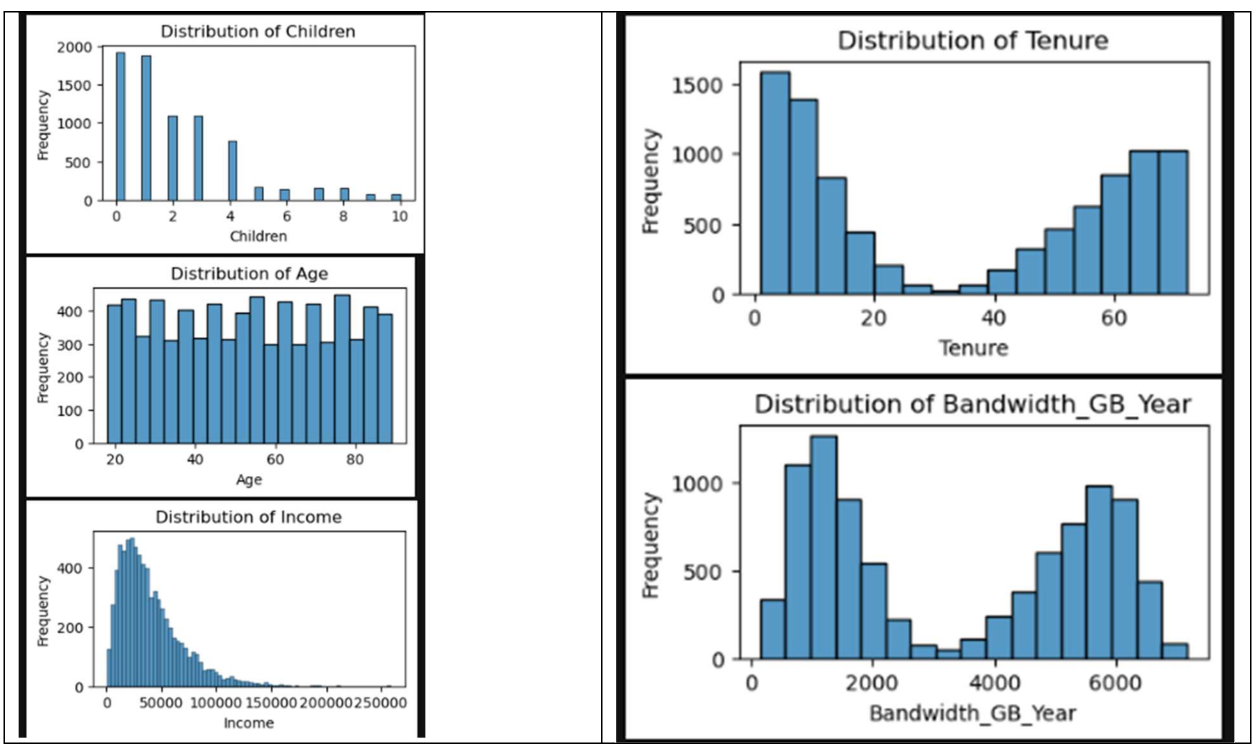
As discussed above, no other outliers were modified as they were determined to either be in a reasonable range of the variable, or the count was low enough to not impact the overall structure of the data significantly.

Missing Values:

Missing values were identified for a variety of variables within the dataset, but there were several solutions for each. After detection, Children, Age, Income, Tenure and Bandwidth_GB_Year were all plotted first using a loop in conjunction with a seaborn histogram to observe the distribution types. The following code was utilized:

```
missing_variables = ["Children", "Age", "Income", "Tenure", "Bandwidth_GB_Year"]
for var in missing_variables:
    plt.figure(figsize=(4, 2))
    sns.histplot(data=df, x=var, kde=False)
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.show()
```

This yielded the following:



Additionally, central tendencies for each variable were calculated for review and to identify the best imputation values.

```
imputation_columns = df[['Children', 'Age', 'Income', 'Tenure', 'Bandwidth_GB_Year']]

print("MEAN")
print(imputation_columns.mean())
print("\nMEDIAN")
print(imputation_columns.median())
print("\nMODE")
print(imputation_columns.mode())
```

```
MEAN
Children                 2.095936
Age                     53.275748
Income               39936.762226
Tenure                  34.498858
Bandwidth_GB_Year     3398.842752
dtype: float64

MEDIAN
Children                 1.00000
Age                     53.00000
Income               33186.78500
Tenure                  36.19603
Bandwidth_GB_Year     3382.42400
dtype: float64

MODE
   Children   Age    Income    Tenure  Bandwidth_GB_Year
0       0.0  55.0  10530.09  55.44991           5228.370
1       NaN   NaN  25598.66  62.86571           5626.094
2       NaN   NaN  36461.20  66.66853           5932.680
3       NaN   NaN  61325.92  69.50480           6081.603
4       NaN   NaN       NaN       NaN           6261.419
5       NaN   NaN       NaN       NaN           6294.845
6       NaN   NaN       NaN       NaN           6417.345
```

Selections were as follows:

- Age was imputed using mean since it was a uniform distribution, and was subsequently rounded to ensure no values came back as non-whole numbers

```
#Age has a uniform distribution. Using Mean to impute
df["Age"] = df["Age"].fillna(df["Age"].mean())
```

```
#Rounding values of Age to ensure all values remain whole numbers
df["Age"] = df["Age"].round()
```

- Children and Income both demonstrated positive distributions, so the median was utilized to impute missing values.

```
#Children has a positively skewed distribution. Using Median to impute
df["Children"] = df["Children"].fillna(df["Children"].median())

#Income has a positvely skewed distribution. Using Median to impute
df["Income"] = df["Income"].fillna(df["Income"].median())
```

- Tenure demonstrated an asymmetric bimodal distribution so the mode was utilized to impute missing variables. Additionally, there were multiple modes in the variable, so the first was selected to impute with.

```
#Tenure has an aysymmetric bimodal disribution. Using first Mode to impute
mode_tenure = df["Tenure"].mode()
mode_to_impute = mode_tenure[0] #Referencing first value in Mode
df["Tenure"] = df["Tenure"].fillna(mode_to_impute)
```

- Before imputing Bandwidth_GB_Year with a central tendency, I wanted to confirm the missing values did not align with customers who did not have internet as a service – as if this was the case the missing values should be imputed with 0. Tested using the following code:

```
testdf = df[['Bandwidth_GB_Year', 'InternetService']]
testdf2 = testdf[testdf['Bandwidth_GB_Year'].isnull()]
print(testdf2)
```

Which yielded the following output

```
      Bandwidth_GB_Year InternetService
14                  NaN             DSL
33                  NaN             DSL
40                  NaN     Fiber Optic
45                  NaN             NaN
58                  NaN             NaN
...                 ...             ...
9896                NaN     Fiber Optic
9914                NaN     Fiber Optic
9939                NaN             NaN
9986                NaN     Fiber Optic
9988                NaN     Fiber Optic

[1021 rows x 2 columns]
```

As this was not the case mode was utilized to impute missing values. Similar to Tenure, there were multiple modes, so the first was utilized.

```
#Bandwidth_GB_Year has an aysmmetric bimodal disribution. Using Mode to impute
mode_bandwidth = df["Bandwidth_GB_Year"].mode()
mode_to_impute = mode_bandwidth[0]
df["Bandwidth_GB_Year"] = df["Bandwidth_GB_Year"].fillna(mode_to_impute)
```

After imputing for the above 5 variables, they were checked to ensure no missing values remained.

```
#Validating missing value changes and ensuring no null values remain
imputation_columns = df[['Children', 'Age', 'Income', 'Tenure', 'Bandwidth_GB_Year']]
imputation_columns.isnull().sum()
## All 5 variables no longer demonstrating missing values
```

```
Children             0
Age                  0
Income               0
Tenure               0
Bandwidth_GB_Year    0
dtype: int64
```

For a final step the central tendencies and the histograms were checked to confirm no significant variance in the data occurred as a result of the imputation. The same code as above was utilized to do so, yielding the following results.

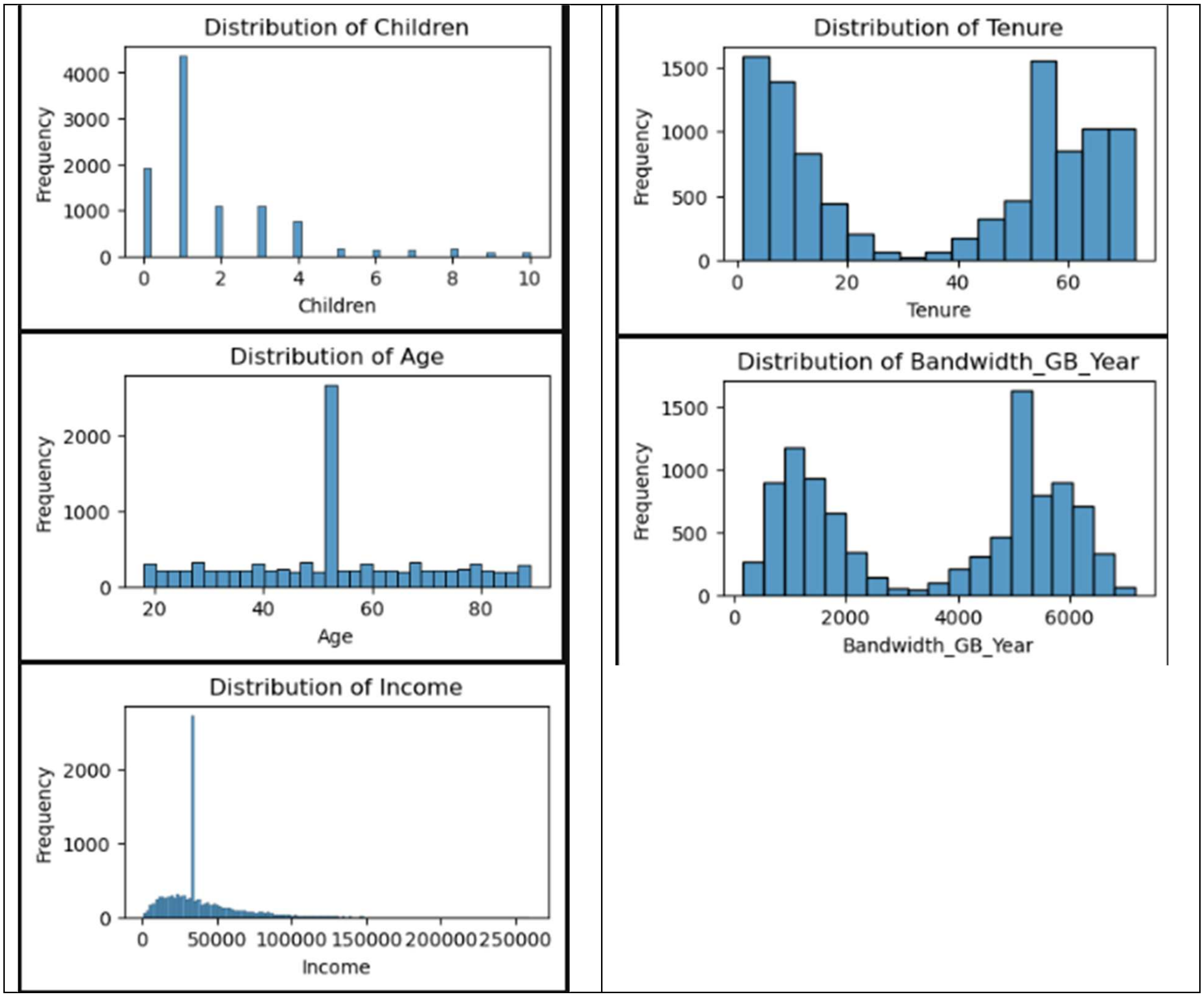
```
MEAN
Children                1.822500
Age                    53.207500
Income              38256.017897
Tenure                 36.449401
Bandwidth_GB_Year    3585.637484
dtype: float64

MEDIAN
Children                1.0000
Age                    53.0000
Income              33186.7850
Tenure                 47.4448
Bandwidth_GB_Year    4472.3030
dtype: float64

MODE
   Children   Age     Income    Tenure  Bandwidth_GB_Year
0       1.0  53.0  33186.785  55.44991            5228.37
```

The mean for some measures varied slightly, but all by not much more than a 5% variance in the original value, well within acceptable changes. Median remained the same for Age, Children and Income, but was reduced by 24% for Tenure and increased by 25% for Bandwidth_GB_Year. These changes can be reviewed as a limitation of the methodology. As expected, each variable only had one mode after imputation due to each selected central tendency being input hundreds or thousands of times.

As a final step of review, the histograms were reviewed to ensure the overall distributions did not change. The same code as above was utilized with the following results.



No unexpected results in the above – a spike was demonstrated in histogram, but the overall distributions remained the same.

The remaining variables with missing data were the qualitative variables Techie, Phone, TechSupport and InternetService. Closer review of InternetService determined the three values present were Fiber Optic, DSL and None, however, Python interpreted the None values as NaN, resulting in the appearance of missing data. To resolve, these values were replaced with "None" using the following code, then confirmed to be updated with all 10,000 observations.

```
df["InternetService"].value_counts()
```

```
InternetService
Fiber Optic    4408
DSL            3463
Name: count, dtype: int64
```

```
#Replacing None with "None"
df["InternetService"] = df["InternetService"].replace([None], "None")
```

```
#Confirming the imputation worked
df["InternetService"].value_counts()
##Values no longer missing
```

```
InternetService
Fiber Optic    4408
DSL            3463
None           2129
Name: count, dtype: int64
```

For Techie, Phone and TechSupport, imputation was not an appropriate method to resolve the missing data. This is due to the fact that since there were only two options to impute, yes or no, imputing either would significantly alter the distribution and quality of the data. Performing the value_counts() method on each of these variables determined 25% of Techie was missing values, and 10% of both Phone and TechSupport were missing values. Instead of risking the integrity of the data, these variables were moved to a separate DataFrame to remove from analysis. Optimally in the real world, there would be a team to contact to review the missing data and provide the relevant gaps as to not affect the overall analysis.

Re-Expression:

This step was optional to the analysis since detection methods did not produce any forms of data which could be considered “Dirty.” None of the observations mismatched or were out of alignment with the other options, nor were any binary options provided as an alternative to yes/no or true/false. Despite this, there was a clear order to the Education variable, so an Education_Numeric variable was created as a part of the cleaning to ensure any statistical analysis could be performed in future data massaging steps. To complete this, first the unique variables were identified with the unique() method. A new variable, Education_Numeric, was created with the same values as the Education variable, and then a dictionary was created assigning the appropriate numerical value to each education level. The code used was:

```
df["Education_Numeric"] = df["Education"]
dict_education = {"Education_Numeric":{
    'No Schooling Completed': 0,
    'Nursery School to 8th Grade': 1,
    '9th Grade to 12th Grade, No Diploma': 2,
    'GED or Alternative Credential': 3,
    'Regular High School Diploma': 4,
    'Professional School Degree': 5,
    'Some College, Less than 1 Year': 6,
    'Some College, 1 or More Years, No Degree': 7,
    "Associate's Degree": 8,
    "Bachelor's Degree": 9,
    "Master's Degree": 10,
    'Doctorate Degree': 11
}}
pd.set_option('future.no_silent_downcasting', True)
df.replace(dict_education, inplace=True)
```

And the update was confirmed via the unique() method on the new variable:

```
#Confirming the replacement worked
df["Education_Numeric"].unique()

array([10, 4, 11, 0, 8, 9, 6, 3, 7, 2, 1, 5], dtype=object)
```

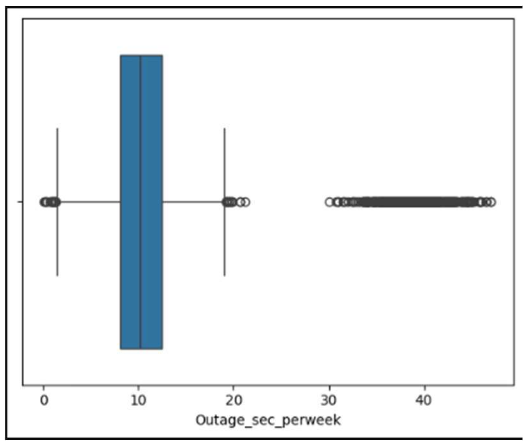
Finally, the survey responses, item1- item8 were renamed to have columns which made more logical sense and were easier to interpret without needing to re-check a data dictionary. This was completed using the rename method, and was done so as one of the first steps in the analysis to make the dataset clearer.

```
df.rename(columns = {
    'item1' : 'Timely_Response',
    'item2' : 'Timely_Fix',
    'item3' : 'Timely_Replacement',
    'item4' : 'Reliability',
    'item5' : 'Options',
    'item6' : 'Respectfulness',
    'item7' : 'Courtesy',
    'item8' : 'Active_Listening'},
    inplace=True)
```

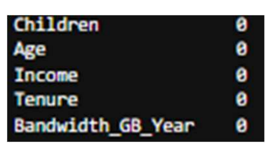
C.3. Summarize Outcomes

After cleaning, the dataset was much more fluid, complete, and more sensical overall. In addition to review for outliers, missing data, re-expression, and missing values, some re-organization in the form of separating out irrelevant columns and renaming unclear columns was performed to create a cleaner dataset overall.

- Replacing outliers for Outage_sec_perweek updated the relevant boxplot to ensure no negative values were included.



- Duplicates were searched for but not identified, including review of the dataset without unique identifiers or indexes which could result in overlooked duplicates.
- Missing values were reviewed across the dataset and rectified via the most relevant central tendency for each data distribution. At the end of the cleaning step, no values were missing for all variables where imputation occurred

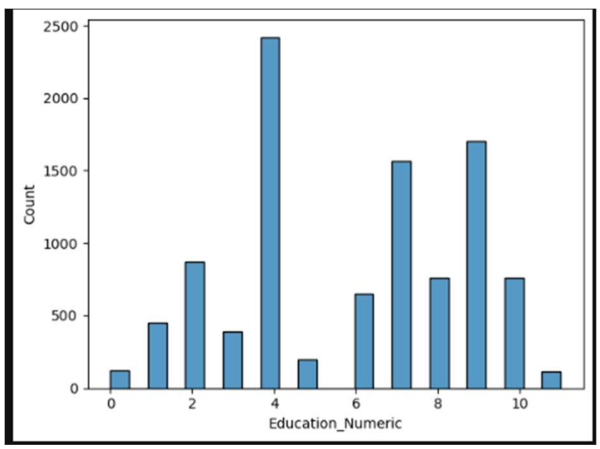


Children	0
Age	0
Income	0
Tenure	0
Bandwidth_GB_Year	0

- Remaining variables with missing data were treated via the exclusion method, but maintained as imputation could cause significant disruption or modification to the integrity of the data. Additionally, the duplicate index column Unnamed: 0 was also separated out to provide a cleaner file for analysis, and also serves as an index for the exclusion file.

```
Index(['Unnamed: 0', 'Techie', 'Phone', 'TechSupport'], dtype='object')
Index(['CaseOrder', 'Customer_id', 'Interaction', 'City', 'State', 'County',
       'Zip', 'Lat', 'Lng', 'Population', 'Area', 'Timezone', 'Job',
       'Children', 'Age', 'Education', 'Employment', 'Income', 'Marital',
       'Gender', 'Churn', 'Outage_sec_perweek', 'Email', 'Contacts',
       'Yearly_equip_failure', 'Contract', 'Port_modem', 'Tablet',
       'InternetService', 'Multiple', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'PaymentMethod', 'Tenure', 'MonthlyCharge',
       'Bandwidth_GB_Year', 'Timely_Response', 'Timely_Fix',
       'Timely_Replacement', 'Reliability', 'Options', 'Respectfulness',
       'Courtesy', 'Active_Listening', 'Education_Numeric'],
      dtype='object')
```

- Re-expression resulted in an additional variable being created to provide a clearer graphical understanding of individual educational level, in addition to creating a format which can be analyzed statistically with less additional rigor.



- Also notable was the re-naming of the survey response columns to names which were easier to identify at a glance without a data dictionary

```
print(df.columns[44:52])

Index(['Timely_Response', 'Timely_Fix', 'Timely_Replacement', 'Reliability',
       'Options', 'Respectfulness', 'Courtesy', 'Active_Listening'],
```

C.4. Discussing Limitations

Outliers in the dataset were treated with both the imputation and retain methodologies. For Outage_sec_perweek, imputation was used via the median, and the outcome could be some bias being introduced into the dataset via guesstimated values. The overall impact was small, only 11 records, but there is some change to the integrity nonetheless. The remainder of the outliers were not modified, and instead retained in their current form as either the frequency was too low to have a major impact, or the outliers were within an acceptable or reasonable range. The downside of this method is the overall reduction in normality during review of statistical analysis.

For missing values, the use of univariate imputation modified and edited some of the central tendencies available in the dataset, and as a result may distort the overall distribution of the data. In reviewing histograms there is also a significant spike in frequency for the value that was imputed, which can make overall review of the distribution more difficult. Additionally, as some variables were removed from the dataset and placed into a separate DataFrame, this can have the result of loss general insight on the original data as whole, and also reduces the overall sample size. The argument could be made to maintain these instead, which would have the downside of creating a skewed representation since there is a smaller sample size to review.

C.5. Limitation Impact on Research Question

Broadly speaking, any instance where modification of the data can lead to bias or impact the central tendencies of a variable has the possibility to yield invalid analysis in future steps of reviewing or presenting the data. With regards to the specific research question asked, the data cleaned had direct impacts to both sides of the “customer demographics or specific customer service detail” portion of the question. As a result, any analyst using the cleaned data may have biases or skewed central tendencies introduced to both specific details about the customer as well as the services they maintained. Additionally, of the variables removed to a separate DataFrame, all of them are bucketed under the category of service details. If there was no method to regain the missing values for these variables to include in the overall analysis with the remainder of the variables, there is some concern with assumptions being made which may not align with a fuller representation of the data.

D.1. Annotated Code

Please see attached code "D206 Assessment.ipynb"

D.2. CSV file(s) of Clean Data

cleaned_churn_data.csv   cleaned_churn_data_cols_removed.csv

E.1. Total Number of Principal Components

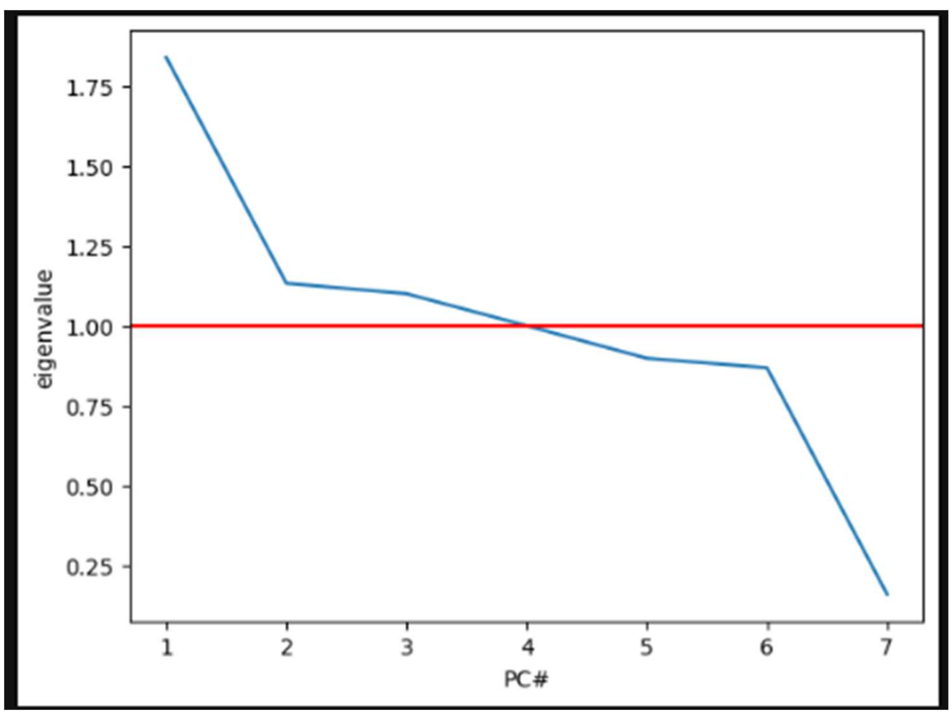
Review of the data demonstrated 7 quantitative continuous variables - Lat, Lng, Income, Outage_sec_perweek, Tenure, MonthlyCharge, Bandwidth_GB_Year. These variables were inserted into a new DataFrame, normalized and the number of principal components was identified as follows:

Using the above value of 7, a loadings matrix was generated with 7 columns for analysis to be completed for the variables identified. Output was the below table:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Lat	-0.021207	0.205989	0.672574	-0.016780	-0.709945	0.021313	-0.002295
Lng	0.004835	-0.132457	-0.690240	0.129016	-0.692559	0.098439	-0.005235
Income	0.003419	0.022593	0.097599	0.990066	0.073546	-0.065656	-0.001092
Outage_sec_perweek	0.022869	0.692372	-0.126679	0.035691	0.100437	0.701909	-0.003578
Tenure	0.705170	-0.053022	0.027246	0.000359	-0.011740	0.039501	0.705325
MonthlyCharge	0.044746	0.676236	-0.211380	-0.039252	-0.025662	-0.700716	0.053101
Bandwidth_GB_Year	0.706913	-0.005334	0.014725	-0.005202	-0.006830	-0.017473	-0.706859

E.2. Justification of Principal Components & Scree Plot

The matrix was reviewed and overall weights were compared to identify variables that had some sort of correlation. Several columns had multiple variables with close correlations both in the positive and inverse. To glean additional insight a scree plot was generated to test and review where the highest eigenvalues occurred. This plot was coded to start with 1 in order to clearly match the various PC values The resultant plot is listed below:



Any PC whose eigenvalue was at or greater than 1 should be maintained for future analysis. The above scree plot identifies PC1, PC2 and PC3 as clearly having eigenvalues greater than one, and thus all should be maintained. Additionally, PC4 was at or near the 1 value, so it should also be maintained since it is close enough to 1 to remain impactful. The remainder can be thrown out or avoided as they drop off with too low of correlation to be statistically significant.

E.3. Benefit of PCA by Organization

Any organization should consider PCA as a part of their data analysis as it can assist with finding covariation amongst variables in a dataset which would not be easily identified with more manual methods. PCA is capable of converting a dataset with a numerous variables including customer demographics, service details and usage patterns, and reduce the number of dimensions while retaining most of the important information. This lends to simplified data that is easier to visualize and analyze. Additionally, by narrowing down details to the most impactful variables, PCA can assist organizations in identifying key areas to reduce customer churn or other attrition problems. In the dataset analyzed above, PC1 identified positive correlations between customer tenure and the amount of bandwidth utilized each year, whose insight could identify business retention strategies such as upselling opportunities to high-bandwidth, long-tenure customers. PC4 was completely dominated by the Income variable, which could yield additional business insights on how to tailor specific service packages based upon different incomes.

Part 4: Supporting Documents

F.1. Panopto Recording

References

Srivastava, Alankrit. (2024). i*loc() Function in Python.*How to use iloc(). Retrieved 08/25/2024. https://www.naukri.com/code360/library/iloc-function-in-python

Select Columns with Specific Data Types in Pandas DataFrame. (2024, May 23). Selecting specific data types. Retrieved 08/25/2024. https://www.geeksforgeeks.org/select-columns-with-specific-data-types-in-pandas-DataFrame/

f-strings in Python. (2024, June 19). Functionality of f-strings. Retrieved 08/26/2024 https://www.geeksforgeeks.org/formatted-string-literals-f-strings-python/

Western Governors University. *R or Python*. https://www.wgu.edu/online-it-degrees/programming-languages/r-or-python.html