D206 Data Cleaning Performance Assessment

Author: Eric Colwell

Western Governors University

This report deals with the rubric for Western Governors University course D206 Data Cleaning and answers all the items in rubric order.

# **Part I: Research Question and Variables**

**A:** **Research Question**

The question posed is whether education level, employment status, and income show a statistical impact on the tenure, or number of months the customer has stayed with the provider, of the customer.

**B: Description of Dataset**

The description of the variables in the data set will be in the order of the columns in the Churn database.

**CaseOrder:** Datatype is int64 and has no null values. This is a placeholder variable to preserve the original order of the raw data file.

**Customer\_id:** Datatype is object and has no null values. This variable is a unique customer identifier.

**Interaction:** Datatype is object and has no null values. This variable has unique identifiers related to customer transactions, technical support, and sign-ups.

**City:** Datatype is object, qualitative, and has no null values. This variable shows the customers city of residence as listed on the billing statement.

**State:** Datatype is object, qualitative, and has no null values. This variable shows the customers city of residence as listed on the billing statement.

**County:** Datatype is object, qualitative, and has no null values. This variable shows the customers county of residence as listed on the billing statement.

**Zip:** Datatype is int64, qualitative, and has no null values. This variable shows the customers zip code of residence as listed on the billing statement.

**Lat:** Datatype is float64, qualitative, and has no null values. This variable shows the customers GPS coordinates for latitude of residence as listed on the billing statement.

**Lng:** Datatype is float64, qualitative, and has no null values. This variable shows the customers GPS coordinates for longitude of residence as listed on the billing statement.

**Population:** Datatype is int64, quantitative, and has no null values. This variable shows the population within a mile radius of the customer based on census data.

**Area:** Datatype is object, qualitative, and has no null values. This variable shows the area type (rural, urban, suburban) based on census data.

**Timezone:** Datatype is object, qualitative, and has no null values. This variable shows the time zone of the customer residence based on customer’s sign-up information.

**Job:** Datatype is object, qualitative, and has no null values. This variable shows the job of the customer (or invoiced person) based on customer’s sign-up information.

**Children:** Datatype is float64, quantitative, and has 2495 null values. This variable shows the number of children in customer’s household as reported in sign-up information.

**Age:** Datatype is float64, quantitative, and has 2475 null values. This variable shows the age of the customer as reported in sign-up information.

**Education:** Datatype is object, qualitative, and has no null values. This variable shows the highest degree earned by a customer as reported in sign-up information.

**Employment:** Datatype is object, qualitative, and has no null values. This variable shows the employment status of a customer as reported in sign-up information.

**Income:** Datatype is float64, quantitative, and has 2490 null values. This variable shows the annual income of the customer as reported in sign-up information.

**Marital:** Datatype is object, qualitative, and has no null values. This variable shows the marital status of a customer as reported in sign-up information.

**Gender:** Datatype is object, qualitative, and has no null values. This variable shows the customers self-identification as male, female, or nonbinary.

**Churn:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer discontinued service with the last month (Yes, No).

**Outage\_sec\_perweek:** Datatype is float64, quantitative, and has no null values. This variable shows the average number of seconds per week of system outages in the customer’s neighborhood.

**Email:** Datatype is int64, quantitative, and has no null values. This variable shows the number of emails sent to the customer in the last year (range from 1 to 23).

**Contacts:** Datatype is intt64, quantitative, and has no null values. This variable shows the number of times the customer contacted technical support (range from 0 to 7).

**Yearly\_equip\_failure:** Datatype is intt64, quantitative, and has no null values. This variable shows the number of times the customer’s equipment failed and had to be reset or replaced in the past year (range from 0 to 6).

**Techie:** Datatype is object, qualitative, and has 2477 null values. This variable shows whether the customer considers themselves technically inclined based on customer questionnaire when they signed up for services (Yes, No).

**Contract:** Datatype is object, qualitative, and has no null values. This variable shows the contract term of the customer (month-to-month, one year, two year).

**Port\_modem:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has a portable modem (Yes, No).

**Tablet:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer owns a tablet such as iPad, Surface, etc. (Yes, No).

**InternetService:** Datatype is object, qualitative, and has no null values. This variable shows the customer’s internet provider (DSL, fiber optic, None).

**Phone:** Datatype is object, qualitative, and has 1026 null values. This variable shows whether the customer has a phone service (Yes, No).

**Multiple:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has multiple lines (Yes, No).

**OnlineSecurity:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has the online security add-on (Yes, No).

**OnlineBackup:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has the online backup add-on (Yes, No).

**DeviceProtection:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has the device protection add-on (Yes, No).

**TechSupport:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has the technical support add-on (Yes, No).

**StreamingTV:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has streaming TV (Yes, No).

**StreamingMovies:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has streaming movies (Yes, No).

**PaperlessBilling:** Datatype is object, qualitative, and has no null values. This variable shows whether the customer has paperless billing (Yes, No).

**PaymentMethod:** Datatype is object, qualitative, and has no null values. This variable shows the customer’s payment method (electronic check, mailed check, bank (automatic bank transfer), credit card (automatic)).

**Tenure:** Datatype is float64, quantitative, and has 931 null values. This variable shows the number of months a customer has stayed with the provider (range from roughly 1 to 72).

**MonthlyCharge:** Datatype is float64, quantitative, and has no null values. This variable shows the amount charged to the customer monthly. This value reflects an average per customer (range from roughly 77 to 316).

**Bandwidth\_GB\_Year:** Datatype is float64, quantitative, and has 1021 null values. This variable shows the average amount of data used, in GB, in a year by the customer (amount charged to the customer monthly) (range from roughly 155 to 7159).

**Item1:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating timely response (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item2:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating timely fixes (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item3:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating timely replacements (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item4:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating reliability (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item5:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating options (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item6:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating respectful response (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item7:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating courteous exchange (scale from 1 to 8 where 1 = most important and 8 = least important).

**Item8:** Datatype is intt64, quantitative, and has no null values. This variable represents a response to a survey rating evidence of active listening (scale from 1 to 8 where 1 = most important and 8 = least important).

**Part II: Detection**

**C1. Detection Methods**

To begin I used the head(), shape(), and info() methods to gain an initial understanding of the dataset. Next, I checked for exact duplicated instances or rows using the duplicated() method combined with the sum() method to get an output or count of how many duplicates there are in the dataset. Next, I checked for null values the isnull() method combined with the sum() method on the whole dataset to get a total of null values for every variable. Also, I used msno.matrix() method and plt.show() to visualize the null values in a matrix. Then I used the sort\_values() method combined with the unique() method to see the unique values sorted for each variable separately. I then visualized correlation between null values using the msno.heatmap() method.

For detection of outliers, I calculated z-scores using the st.zscore() method; then used plt.hist(). plt.show(), and sns.boxplot() methods to visualize the possible outliers. (Larose & Larose, 2019)

I re-expressed the variables ‘Education’, ‘Employment’, ‘Churn’, and ‘InternetService’. ‘Education’ has been re-expressed into int64 datatype categories for each unique value and given a value correlating to the years of education completed. ‘Employment’ has been re-expressed into int64 datatype categories for each unique value and given a value correlating to employment status. ‘Churn’ has been re-expressed into int64 datatype with categories where ‘No’ = 0 and ‘Yes’ = 1. ‘InternetService’ has been re-expressed into int64 datatype with categories where ‘None’ = 0, ‘DSL’ = 1, and ‘Fiber Optic’ = 2.

**C2: Justification for Detection Methods**

The detection of duplicated instances, missing or null values, and outliers is a “step-by-step” process (Larose & Larose, 2019). By following this process using the above-described methods I have confidently detected data that may need to be treated.

**C3: Justification for Program Language**

I chose to work with the programming language Python. I have more experience using this language than R. I used Pandas for importing the csv file and manipulation of numerical data. I used Numpy for numerical calculations. I used matplotlib.pyplot for plotting data. I used stats from scipy for statistical analysis and outlier detection. I used missingno for visualization of missing data. I used IterativeImputer from fancyimpute for the iterative imputations. I used Counter to count instances for an example. I used seaborn for data visualizations, and I used PCA from sklearn.decomposition for the principal component analysis.

**C4: Detection Code**

The code below was used to identify anomalous values in the dataset. To detect outliers in the data I used Z\_scores. This step of the data detection process requires there to be no null values. Null value mitigation code must be included to be error free.

# Import pandas for manipulating numerical tables

import pandas as pd

# Import numpy for numerical calculations

import numpy as np

# Import matplotlib for plotting data

import matplotlib.pyplot as plt

%matplotlib inline

# Import scipy-stats for statistical analysis and outlier detection

from scipy import stats as st

# Import missingno for visualizations

import missingno as msno

# Import IterativeImputer for imputations

from fancyimpute import IterativeImputer

# Import seaborn for visualizations

import seaborn as sns

# Import PCA for principal component analysis

from sklearn.decomposition import PCA

# Import Counter for counting instances

from collections import Counter

# Importing the churn dataset

df = pd.read\_csv('C:/users/eric7/D206/churn\_database/churn\_raw\_data.csv')

# Just checking to see if the data is there

df.head()

# Check shape of df

print(df.shape)

# 10000 rows or instances and 52 columns or variables

# Check df info

print(df.info())

# Check for duplicates

df.duplicated().sum()

# Check for null values

print(df.isnull().sum())

# Visualize df for missingness

msno.matrix(df, labels = True)

plt.show()

# Create a copy of df

df1 = df.copy()

# Checking variables unique values for validity

print(df1['Zip'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Area'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Children'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Age'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Education'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Employment'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Income'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Marital'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Gender'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Churn'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Outage\_sec\_perweek'].sort\_values().unique())

# Find and count negative time values in Outage\_sec\_perweek

neg\_outage = df1.Outage\_sec\_perweek[df1.Outage\_sec\_perweek < 0]

print(neg\_outage)

neg\_outage.count()

# Make the bad values NaN

df1.Outage\_sec\_perweek[df1.Outage\_sec\_perweek < 0] = np.nan

# Drop the bad values from df

df1 = df1.dropna(subset = ['Outage\_sec\_perweek'])

# Check to see if they are gone. We have 11 less instances now

df1.info()

# Checking variables unique values for validity

print(df1['Email'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Contacts'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Yearly\_equip\_failure'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Techie'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Contract'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Port\_modem'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Tablet'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['InternetService'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Phone'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Multiple'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['OnlineSecurity'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['OnlineBackup'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['DeviceProtection'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['TechSupport'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['StreamingTV'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['StreamingMovies'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['PaperlessBilling'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['PaymentMethod'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Tenure'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['MonthlyCharge'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['Bandwidth\_GB\_Year'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item1'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item2'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item3'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item4'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item5'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item6'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item7'].sort\_values().unique())

# Checking variables unique values for validity

print(df1['item8'].sort\_values().unique())

# Check for correlation of missingness

msno.heatmap(df1)

# Check the null values--Outage\_sec\_perweek should be 0

df1.isnull().sum()

# Make a copy of df1

df\_it\_imp = df1.copy()

# Separate the non\_numeric type columns

non\_numeric\_cols = df\_it\_imp.select\_dtypes(exclude='number')

# Drop the non\_numeric columns from the copied df

df\_it\_imp.drop(non\_numeric\_cols, axis=1, inplace=True)

# Create Imputer object with minimum value set to zero for no negative values

iterative\_imputer = IterativeImputer(min\_value = 0)

# Impute on numeric columns

df\_it\_imp.iloc[:, :] = iterative\_imputer.fit\_transform(df\_it\_imp)

# Check for null values in numeric columns--Should all be 0

df\_it\_imp.isna().sum()

# Place the cleaned numeric columns back into df1

df1['Children'] = df\_it\_imp['Children']

df1['Age'] = df\_it\_imp['Age']

df1['Income'] = df\_it\_imp['Income']

df1['Tenure'] = df\_it\_imp['Tenure']

df1['Bandwidth\_GB\_Year'] = df\_it\_imp['Bandwidth\_GB\_Year']

# Check full df for nullity--Should just be categorical variable nullity

df1.isna().sum()

# Mode imputation

df1['Techie'] = df1['Techie'].fillna(df1['Techie'].mode()[0])

# Mode imputation

df1['Phone'] = df1['Phone'].fillna(df1['Phone'].mode()[0])

# Mode imputation

df1['TechSupport'] = df1['TechSupport'].fillna(df1['TechSupport'].mode()[0])

# Check nullity--df should have 0 null values now

df1.isna().sum()

# Visualize the clean dataset

msno.matrix(df1, labels = True)

plt.show()

# Checking for outliers using Z-scores--stats

df1['Z\_children'] = st.zscore(df1['Children'])

df1[['Children', 'Z\_children']].head()

# Check it on the histogram

plt.hist(df1['Z\_children'])

plt.show()

# Check the boxplot for outliers

boxplot = sns.boxplot(x='Children', data = df1)

# Checking for outliers using Z-scores--stats

df1['Z\_age'] = st.zscore(df1['Age'])

df1[['Age', 'Z\_age']].head()

# Check it on the histogram

plt.hist(df1['Z\_age'])

plt.show()

# Check the boxplot for outliers

boxplot = sns.boxplot(x='Age', data = df1)

# Checking for outliers using Z-scores--stats

df1['Z\_income'] = st.zscore(df1['Income'])

df1[['Income', 'Z\_income']].head()

# Check it on the histogram

plt.hist(df1['Z\_income'])

plt.show()

# Check the boxplot for outliers

boxplot = sns.boxplot(x='Income', data = df1)

# Checking for outliers using Z-scores--stats

df1['Z\_outage'] = st.zscore(df1['Outage\_sec\_perweek'])

df1[['Outage\_sec\_perweek', 'Z\_outage']].head()

# Check it on the histogram

plt.hist(df1['Z\_outage'])

plt.show()

# Check the boxplot for outliers

boxplot = sns.boxplot(x='Outage\_sec\_perweek', data = df1)

**Part III: Treatment**

**D1. Discussion of Findings**

I found that there were 0 duplicated instances in the dataset. In the variable ‘Outage\_sec\_perweek’ I found 11 instances where the average amount of seconds was less than zero. I found that 8 out of the 52 variables did have missing or NaN data. ‘Children’ = 2495 null values, ‘Age’ = 2475 null values, ‘Income’ = 2490 null values, ‘Techie’ = 2477 null values, ‘Phone’ = 1026 null values, ‘TechSupport’ = 991 null values, ‘Tenure’ = 931 null values, and ‘Bandwidth\_GB\_Year’ = 1021 null values. Using msno.heatmap(), I found no exact correlation of the missingness.

Z-scores “rule of thumb” is that values above 3 or below -3 could be considered an outlier. Using z-scores with visualizations on histograms and boxplots I detected values that were outside of that scale. These possible outlier values were found in variables ‘Children’, ‘Income’, and ‘Outage\_sec\_perweek’. ‘Children’ has 302 instances where z-scores are greater than 3 and none less than -3. The highest value in the ‘Children’ variable is 10. ‘Income’ has 181 instances where z-scores are greater than 3 and none less than -3. The highest value in the ‘Income’ variable is 258900.7. ‘Outage\_sec\_perweek’ has 491 instances where z-scores are greater than 3 and none less than -3. The highest value in the ‘Outage\_sec\_perweek’ is 47.04928.

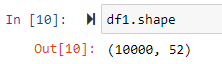
**D2. Treatment Methods and Justification**

The image below represents the missing values in the dataset with white lines in the column where null values are present.

A picture containing text, computer, screenshot

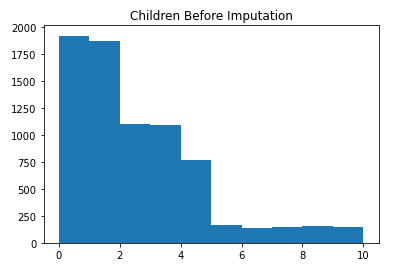
Description automatically generated

Since there were 0 duplicated rows in the dataset there is no treatment needed. Finding there were 11 instances of negative time values in the variable ‘Outage\_sec\_perweek’ shows that these values are invalid. There are 10,000 rows in the dataset. A basic “rule of thumb” for dropping instances from a dataset is if the instances total less than 5% of the total dataset then it is okay to drop or delete them completely, as opposed to some form of imputation. In this case our erroneous values equal 0.11% of the dataset so I dropped them (Larose & Larose, 2019). First, I changed the negative values to NaN using np.nan. Then I dropped the instances using dropna(). I checked the dataset using info() and shape() to make sure the values had been dropped. Now total rows equal 9989 proving that the 11 instances have been dropped. The images below show the dataset shape before and after dropping the 11 instances.

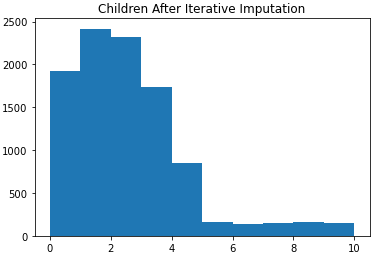
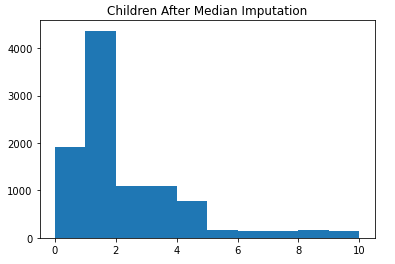


There are 5 numeric variables and 3 categorical variables containing null values that needed to be treated. Because all 8 variables have a total of null values above 5% of the total, I shouldn’t drop them from the dataset. That would have caused too much reduction of the dataset. Imputation of data is the process of inserting data in the place of the null values. The two different types of imputation I used include univariate (mean, median, mode) imputation and iterative imputation. For the 5 numeric variables I chose to test both types of imputation.

When the ‘Children’ variable is viewed in a histogram I observed it’s distribution is skewed right.



For this type of data distribution, it is recommended to use univariate median imputation (Larose & Larose, 2019). Below are histograms showing the data distribution after median imputation and iterative imputation.



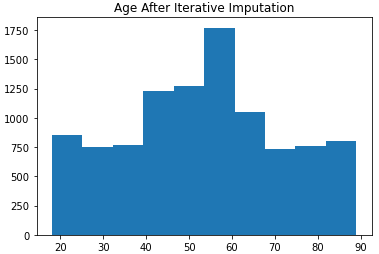
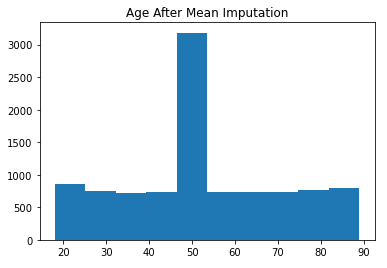
I observed with the median imputation there was a larger spike than with iterative imputation. The iterative imputation method is the better choice, because the data distribution is closer to the original distribution than with median imputation.

Below is a histogram showing the original data distribution for the ‘Age’ variable. I observed the distribution is uniform.

Chart, histogram

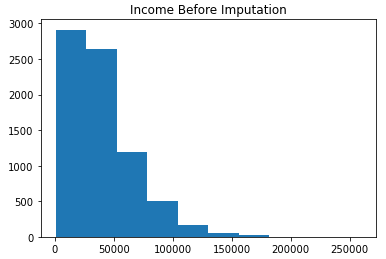
Description automatically generated

For this type of distribution, a univariate mean imputation is recommended. Below are histograms showing the data distribution after mean and iterative imputation.



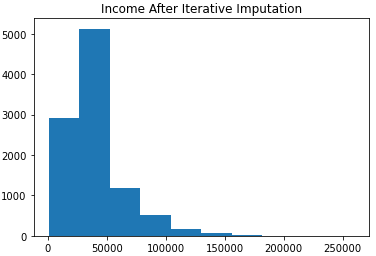
Again, I observed with iterative imputation the data distribution more closely resembles the original distribution than with univariate mean imputation.

Below is a histogram showing the original data distribution for the ‘Income’ variable. It is skewed right, like the ‘Children’ variable.



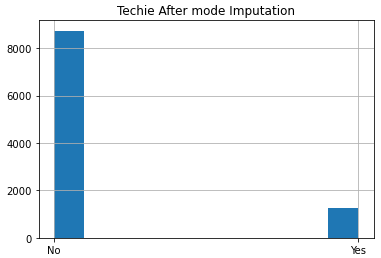
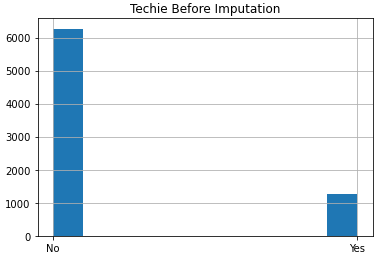
For this type of data distribution, it is recommended to use univariate median imputation. Below are histograms showing the data distribution after median imputation and iterative imputation.

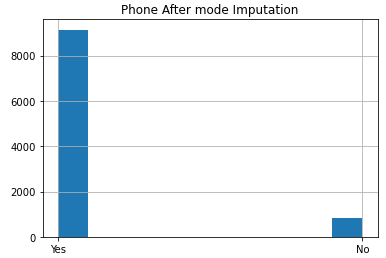
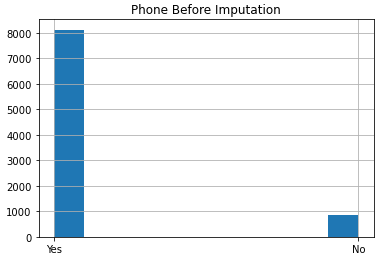
Chart, histogram

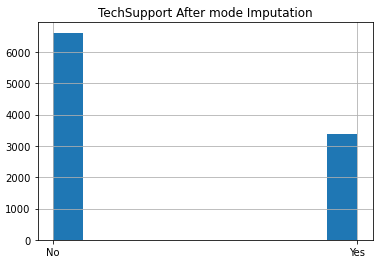
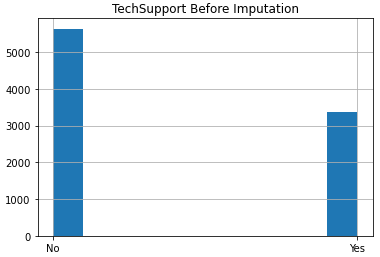
Description automatically generated

These distributions appear very similar. Iterative imputation is a more mathematically advanced form of imputation; I chose to use it over the univariate median imputation.

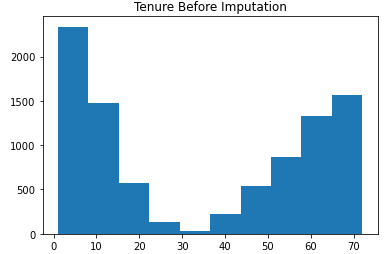
The next three variables (‘Techie’, ‘Phone’, ‘TechSupport’) are all categorical variables with only two values, Yes and No. I used univariate mode imputation as recommended in the PowerPoint presentation provided by instructor, Keonia Middleton. (Larose & Larose, 2019) Below are the histograms before and after mode imputation for each of the three categorical variables.



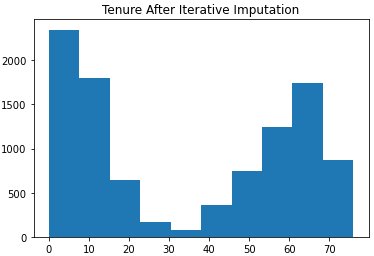
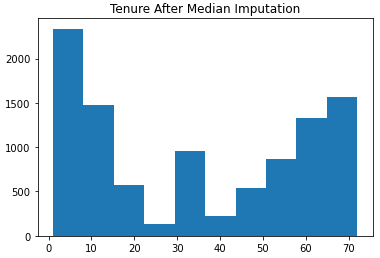




Below is a histogram showing the original data distribution of the numerical ‘Tenure’ variable.



This data distribution is referred to as bimodal. I performed a test with univariate median imputation and iterative imputation. Below are the histograms for both methods.



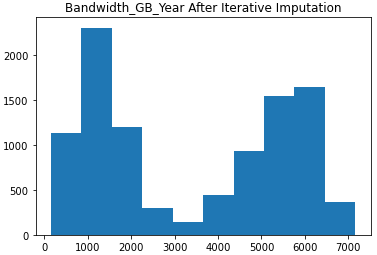
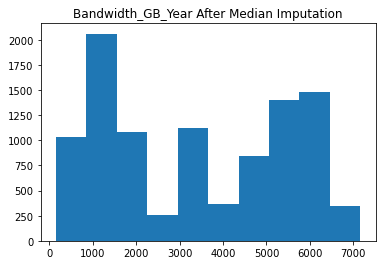
I clearly observed a spike in the center of the histogram after median imputation. The data distribution in the histogram after iterative imputation shows a much better resemblance to the original distribution.

Below is a histogram showing the original data distribution of the numerical ‘Bandwidth\_GB\_Year’ variable.

Chart, histogram

Description automatically generated

This data distribution is bimodal like ‘Tenure’. I again tested with univariate median imputation and iterative imputation. Below are the histograms for both methods.



With median imputation I observed a clear spike in the center. The iterative imputation retains a close resemblance to the original data distribution.

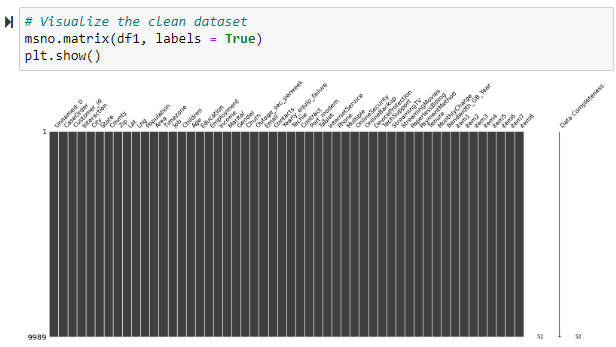
For the variables ‘Children’, ‘Income’, and ‘Outage\_sec\_perweek where Z-scores showed outliers, I decided not to remove those instances. Future analysis may require these values for predicting outcomes (Sequitin, 2021).

**D3. Summary of Treatment**

Through this step of the data cleaning process, I removed anomalies and missing values in the dataset. I dropped 11 instances from the ‘Outage\_sec\_perweek’ variable, because instances having negative time values were impossible and therefore false. The rest of the anomalies that I found were null values. Each of the variables containing null values had a percentage well above the 5% threshold. Simply dropping this number of instances would cause too much reduction to the size of the given dataset. For the numerical variables that had missing or null values, using the iterative method provided a better outcome than came from univariate mean or median methods. For the categorical variables, using univariate mode imputation allowed me to replace the missing values with the mode value (Larose & Larose, 2019).

The first screenshot below verifies that every variable that had null values now has zero null values. The output of nine 0’s proves this. The second screenshot also verifies that there are zero null values present in the dataset because there are no white lines in the matrix.





**D4. Treatment Code**

The code provided below shows how I treated null values and false values in the dataset. To see the outcomes please use the full code provided in my submission named eric\_colwell\_full\_code.

# Checking variables unique values for validity

print(df1['Outage\_sec\_perweek'].sort\_values().unique())

# Find and count negative time values in Outage\_sec\_perweek

neg\_outage = df1.Outage\_sec\_perweek[df1.Outage\_sec\_perweek < 0]

print(neg\_outage)

neg\_outage.count()

# Make the bad values NaN

df1.Outage\_sec\_perweek[df1.Outage\_sec\_perweek < 0] = np.nan

# Drop the bad values from df

df1 = df1.dropna(subset = ['Outage\_sec\_perweek'])

# Check to see if they are gone. We have 11 less instances now

df1.info()

# Make a copy of df1

df\_it\_imp = df1.copy()

# Separate the non\_numeric type columns

non\_numeric\_cols = df\_it\_imp.select\_dtypes(exclude='number')

# Drop the non\_numeric columns from the copied df

df\_it\_imp.drop(non\_numeric\_cols, axis=1, inplace=True)

# Create Imputer object with minimum value set to zero for no negative values

iterative\_imputer = IterativeImputer(min\_value = 0)

# Impute on numeric columns

df\_it\_imp.iloc[:, :] = iterative\_imputer.fit\_transform(df\_it\_imp)

# Check for null values in numeric columns--Should all be 0

df\_it\_imp.isna().sum()

# Place the cleaned numeric columns back into df1

df1['Children'] = df\_it\_imp['Children']

df1['Age'] = df\_it\_imp['Age']

df1['Income'] = df\_it\_imp['Income']

df1['Tenure'] = df\_it\_imp['Tenure']

df1['Bandwidth\_GB\_Year'] = df\_it\_imp['Bandwidth\_GB\_Year']

# Check full df for nullity--Should just be categorical variable nullity

df1.isna().sum()

# Make a copy for mean and median imputation visualizations

df\_imp = df.copy()

# Checking variables unique values for validity

print(df1['Children'].sort\_values().unique())

# round the children variable to whole numbers

# because it is a discrete variable

df1['Children'] = round(df1['Children'])

# Check to make sure it worked

print(df1['Children'].sort\_values().unique())

# Comparing histograms before and after imputation

# Original with null values

plt.hist(df['Children'])

plt.title('Children Before Imputation')

# Median imputation

df\_imp['Children'].fillna(df\_imp['Children'].median(), inplace=True)

# Comparing histograms before and after imputation

# Using median because the original histogram distribution is skewed

plt.hist(df\_imp['Children'])

plt.title('Children After Median Imputation')

# Comparing histograms before and after imputation

plt.hist(df1['Children'])

plt.title('Children After Iterative Imputation')

# Comparing histograms before and after imputation

# Original with null values

plt.hist(df['Age'])

plt.title('Age Before Imputation')

# Mean imputation

df\_imp['Age'].fillna(df\_imp['Age'].mean(), inplace=True)

# Comparing histograms before and after imputation

# Using mean because the original histogram distribution is uniform

plt.hist(df\_imp['Age'])

plt.title('Age After Mean Imputation')

# Comparing histograms before and after imputation

plt.hist(df1['Age'])

plt.title('Age After Iterative Imputation')

# Comparing histograms before and after imputation

# Original with null values

plt.hist(df['Income'])

plt.title('Income Before Imputation')

# Median imputation

df\_imp['Income'].fillna(df\_imp['Income'].median(), inplace=True)

# Comparing histograms before and after imputation

# Using median because the original histogram distribution is skewed

plt.hist(df\_imp['Income'])

plt.title('Income After Median Imputation')

# Comparing histograms before and after imputation

plt.hist(df1['Income'])

plt.title('Income After Iterative Imputation')

# Comparing histograms before and after imputation

# Original with null values

df['Techie'].hist()

plt.title('Techie Before Imputation')

# Mode imputation

df1['Techie'] = df1['Techie'].fillna(df1['Techie'].mode()[0])

# Comparing histograms before and after imputation

df1['Techie'].hist()

plt.title('Techie After mode Imputation')

# Comparing histograms before and after imputation

# Original with null values

df['Phone'].hist()

plt.title('Phone Before Imputation')

# Mode imputation

df1['Phone'] = df1['Phone'].fillna(df1['Phone'].mode()[0])

# Comparing histograms before and after imputation

df1['Phone'].hist()

plt.title('Phone After mode Imputation')

# Comparing histograms before and after imputation

# Original with null values

df['TechSupport'].hist()

plt.title('TechSupport Before Imputation')

# Mode imputation

df1['TechSupport'] = df1['TechSupport'].fillna(df1['TechSupport'].mode()[0])

# Comparing histograms before and after imputation

df1['TechSupport'].hist()

plt.title('TechSupport After mode Imputation')

# Comparing histograms before and after imputation

# Original with null values

plt.hist(df['Tenure'])

plt.title('Tenure Before Imputation')

# Median imputation

df\_imp['Tenure'].fillna(df\_imp['Tenure'].median(), inplace=True)

# Comparing histograms before and after imputation

# Using median because the original histogram distribution is skewed

# Mean imputation would be similar result

plt.hist(df\_imp['Tenure'])

plt.title('Tenure After Median Imputation')

# Comparing histograms before and after imputation

plt.hist(df1['Tenure'])

plt.title('Tenure After Iterative Imputation')

# Comparing histograms before and after imputation

# Original with null values

plt.hist(df['Bandwidth\_GB\_Year'])

plt.title('Bandwidth\_GB\_Year Before Imputation')

# Median imputation

df\_imp['Bandwidth\_GB\_Year'].fillna(df\_imp['Bandwidth\_GB\_Year'].median(), inplace=True)

# Comparing histograms before and after imputation

# Using median because the original histogram distribution is skewed

# Mean imputation would be similar result

plt.hist(df\_imp['Bandwidth\_GB\_Year'])

plt.title('Bandwidth\_GB\_Year After Median Imputation')

# Comparing histograms before and after imputation

plt.hist(df1['Bandwidth\_GB\_Year'])

plt.title('Bandwidth\_GB\_Year After Iterative Imputation')

# Check nullity--df should have 0 null values now

df1.isna().sum()

# Show that each variable now has Zero null values

print(df1['Outage\_sec\_perweek'].isna().sum())

print(df1['Children'].isna().sum())

print(df1['Age'].isna().sum())

print(df1['Income'].isna().sum())

print(df1['Techie'].isna().sum())

print(df1['Phone'].isna().sum())

print(df1['TechSupport'].isna().sum())

print(df1['Tenure'].isna().sum())

print(df1['Bandwidth\_GB\_Year'].isna().sum())

# Visualize the clean dataset

msno.matrix(df1, labels = True)

plt.show()

**D5. Clean Dataset**

CSV file of the cleaned dataset is uploaded with submission.

**D6. Limitations**

There are disadvantages of using the methods employed to clean the dataset. Imputation of data changes the dataset to some degree. The imputed values are mathematically derived and therefore, not factual. I showed the difference of data distribution before and after using histograms. The numerical variables had some noticeable change in the distributions, but using iterative imputation is theoretically more accurate than with mean or median imputation. The categorical variables showed the most change because I used univariate mode imputation. These variables have only two choices, ‘Yes’ or ‘No’. With mode imputation, whichever of the choices is the most frequent, receives 100% of the imputed data. For example, the variable ‘Phone’ had 1026 null values, 8128 ‘Yes’ values and 846 ‘No’ values totaling 10,000. The mode of this variable is ‘Yes’. After mode imputation the ‘Yes’ values equal 9143 and the ‘No’ values remain unchanged at 846. This example demonstrates the disadvantage of using univariate imputations.

**D7. Implications**

To answer the research question from part A, a data analyst must understand how the imputation methods have changed the data within the dataset. The categorical variables ‘Education’ and ‘Employment’ had zero null values and were unaffected by imputation, but these variables were changed manually through ordinal encoding. I changed these variables to numerical by assigning a value to each unique string as shown below. I used ordinal encoding for the variables ‘Education’, ‘Employment’, ‘Churn’, and ‘InternetService. This process created new numeric variables named ‘education\_numeric’, ‘employment\_numeric’, ‘churn\_numeric’, and ‘int\_serv\_numeric’.

Text

Description automatically generated

Chart, scatter chart

Description automatically generated with medium confidence

The other variables used to answer the research question, ‘Income’ and ‘Tenure’, were affected by imputation. Variables ‘Outage\_sec\_perweek’, ‘Children’, ‘Age’, ‘Techie’, ‘Phone’, ‘TechSupport’, and ‘Bandwidth\_GB\_Year’ have also been affected to some degree by imputation. These changes must be considered and accounted for by the analyst to answer the question.

**Part IV: PCA**

**E1: Variables and PCA Loadings**

For the PCA I included all the continuous quantitative variables in the dataset. The variables included are as follows: ‘Age’, ‘Income’, ‘Outage\_sec\_perweek’, ‘Tenure’, ‘MonthlyCharge’, and ‘Bandwidth\_GB\_Year’.

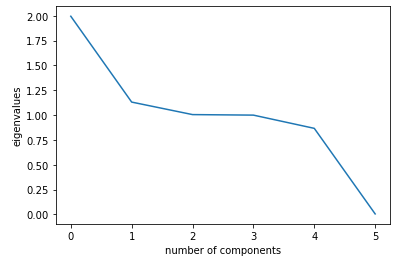
PCA loadings:

Table

Description automatically generated

**E2: PCs Selection**

When determining which PC’s to keep the Kaiser rule should be considered. This rule says I should keep the principal components having variance or eigenvalue larger than 1. In the scree plot below, it looks like the 3rd and 4th PC’s have similar eigenvalues very close to 1. Another rule that can be considered is the Elbow rule. In the scree plot there is an ‘elbow’ at the point corresponding to 1 number of components. Going by this rule I will choose to keep the first two PC’s because they contain most of the variance.



**E3. Benefits**

There are benefits to using PCA. One is dimensionality reduction of the dataset. PCA basically forms a new dataset of principle components needing less storage space while retaining most of the information. This newly created dataset with less dimensions will reduce the chance of overfitting. Overfitting happens when models learn too much from training data, then cannot accurately make predictions on the test data.

References

Larose, C.D., & Larose, D.T.(2019). *Data science using Python and R*. ISBN-13: 978-1-119-52684-1.

Sequitin, K.(2021, October 5). *What is an outlier.* https://careerfoundry.com/en/blog/data-analytics/what-is-an-outlier