Performance Assessment for
D206 Data Cleaning

Drew Mendez
MSDA Western Governors University
D206: Data Cleaning
Dr. Keiona Middleton
April 15, 2024

D206_PA_MendezD

April 15, 2024

1 Part I: Research Question

1.0.1 A. Research Question:

The data set selected for this performance assessment is the **churn** data set. The research question for this assessment is:

What customer attributes (all variables from ***demographic data****,* ***personal information data****,* ***account information and services data****, defined below) contribute to whether a customer will discontinue their service?*

1.0.2 B. Description of Variables:

In the churn data set, there are a total of 10000 rows representing 10000 customers. According to the data dictionary, there are 50 variables. However, although the data dictionary defines 'Lat, Lng' as one variable, the data frame appears to have two distinct variables, 'Lat' and 'Lng'. Furthermore, there appears to be an unnamed first column which indexes the data set, which is not referenced in the data dictionary.

The following variables represent various **customer IDs**:

- unnamed column: An index column without a title (**nominal categorical data**)
- CaseOrder: A placeholder variable to preserve the original order of the raw data (**nominal categorical data**)
- Customer_id: Unique customer ID (**nominal categorical data**)
- Interaction, UID: Unique IDs related to customer interactions (**nominal categorical data**)

The following variables represent customer **demographic data**:

- City: Customer city of residence (**nominal categorical data**)
- State: Customer state of residence (**nominal categorical data**)
- County: Customer county of residence (**nominal categorical data**)
- Zip: Customer zip code of residence (**nominal categorical data**)
- Lat: GPS coordinates of the latitude of the customer residence (**continuous numeric data**)
- Lng: GPS coordinates of the longitude of the customer residence (**continuous numeric data**)
- Population: Population within a mile radius of customer (**discrete numeric data**)
- Area: Area type (rural, urban, suburban) (**nominal categorical data**)
- TimeZone: Time zone of customer residence based on customer sign-up information (**nominal categorical data**)

The following variables represent customer **personal information data**:

- Job: Job of the customer/invoiced person (**nominal categorical data**)
- Children: Number of children in customer's household (**discrete numeric data**)
- Age: Age of customer (**continuous numeric data**)
- Education: Highest degree earned by customer (**ordinal categorical data**)
- Employment: Employment status of customer (**nominal categorical data**)
- Income: Annual income of customer (**continuous numeric data**)
- Marital: Marital status of customer (**nominal categorical data**)
- Gender: Customer self-identification as male, female, or nonbinary (**nominal categorical data**)

The following variables represent customer **account information and services data**:

- Churn: Whether the customer discontinued service within the last month (yes, no) (**binary nominal categorical data**)
- Outage_sec_perweek: Average number of seconds per week of system outages in the customer's neighborhood (**continuous numeric data**)
- Email: Number of emails sent to the customer in the last year (marketing or correspondence) (**discrete numeric data**)
- Contacts: Number of times customer contacted technical support (**discrete numeric data**)
- Yearly_equip_failure: The number of times customer's equipment failed and had to be reset/replaced in the past year (**discrete numeric data**)
- Techie: Whether the customer considers themselves technically inclined (yes, no) (**binary nominal categorical data**)
- Contract: The contract term of the customer (month-to-month, one year, two year) (**nominal categorical data**)
- Port_modem: Whether the customer has a portable modem (yes, no) (**binary nominal categorical data**)
- Tablet: Whether the customer owns a tablet such as iPad, Surface, etc. (yes, no) (**binary nominal categorical data**)
- InternetService: Customer's internet service provider (DSL, fiber optic, None) (**nominal categorical data**)
- Phone: Whether the customer has a phone service (yes, no) (**binary nominal categorical data**)
- Multiple: Whether the customer has multiple lines (yes, no) (**binary nominal categorical data**)
- OnlineSecurity: Whether the customer has an online security add-on (yes, no) (**binary nominal categorical data**)
- OnlineBackup: Whether the customer has an online backup add-on (yes, no) (**binary nominal categorical data**)
- DeviceProtection: Whether the customer has device protection add-on (yes, no) (**binary nominal categorical data**)
- TechSupport: Whether the customer has a technical support add-on (yes, no) (**binary nominal categorical data**)
- StreamingTV: Whether the customer has streaming TV (yes, no) (**binary nominal categorical data**)
- StreamingMovies: Whether the customer has streaming movies (yes, no) (**binary nominal categorical data**)
- PaperlessBilling: Whether the customer has paperless billing (yes, no) (**binary nominal categorical data**)

- PaymentMethod: The customer's payment method (electronic check, mailed check, bank (automatic bank transfer), credit card (automatic)) (**nominal categorical data**)
- Tenure: Number of months the customer has stayed with the provider (**continuous numeric data**)
- MonthlyCharge: The amount charged, on average, per customer monthly (**continuous numeric data**)
- Bandwidth_GB_Year: The average amount of data used, in GB, in a year by the customer (**continuous numeric data**)

The following variables represent customers' responses to an eight-question survey, with responses on a scale of 1 to 8 (1 = most important, 8 = least important)

- Item1: Timely response (**ordinal categorical data**)
- Item2: Timely fixes (**ordinal categorical data**)
- Item3: Timely replacements (**ordinal categorical data**)
- Item4: Reliability (**ordinal categorical data**)
- Item5: Options (**ordinal categorical data**)
- Item6: Respectful response (**ordinal categorical data**)
- Item7: Courteous exchange (**ordinal categorical data**)
- Item8: Evidence of active listening (**ordinal categorical data**)

1.0.3 B1. Example from the Data Set

As an example of the data set, consider the 40th observation in the data set:

At the time of registration, the 40th observation is a divorced male from Bucks County in Dublin, PA, 18917, which has a population of 2123. He holds a regular high school diploma, and he is employed full-time as an editorial assistant, making $50,336.50 annually. He had not discontinued his service within the last month.

Variable	Value of 40th Observation
CaseOrder	40
Customer_id	Z666770
Interaction	e8e28d51-c371-4a39-a81c-0c08d2f1f6ba
City	Dublin
State	PA
County	Bucks
Zip	18917
Lat	40.37305
Lng	-75.2041
Population	2123
Area	Suburban
Timezone	America/New_York
Job	Editorial assistant
Children	1.0
Age	72.0
Education	Regular High School Diploma
Employment	Full Time
Income	50336.5

Variable	Value of 40th Observation
Marital	Divorced
Gender	Male
Churn	No
Outage_sec_perweek	7.790281
Email	10
Contacts	1
Yearly_equip_failure	1
Techie	No
Contract	Two Year
Port_modem	Yes
Tablet	No
InternetService	Fiber Optic
Phone	Yes
Multiple	No
OnlineSecurity	No
OnlineBackup	Yes
DeviceProtection	Yes
TechSupport	No
StreamingTV	No
StreamingMovies	No
PaperlessBilling	No
PaymentMethod	Electronic Check
Tenure	16.042022
MonthlyCharge	147.291188
Bandwidth_GB_Year	1530.10769
item1	3
item2	4
item3	3
item4	6
item5	4
item6	4
item7	6
item8	2

2 Part II: Data-Cleaning Plan

2.0.1 C1. Plan to Clean the Data

The plan to assess the quality of the data in the data set involves detecting duplicates, missing values, and outliers, and the re-expression of a categorical variable using ordinal encoding. The steps and techniques necessary to assess the quality of the data is given:

- Import: the data will first be imported into a data frame using the `read_csv()` function from the Pandas library.
- Duplicates:
  - Duplicates will be detected by chaining the `.duplicated()` and `.sum()` methods from

the Pandas library and calling them on the data frame, returning the total count of duplicate observations. Duplicate columns will be detected using Pandas' `.equals()` method, accessing the unnamed column using Pandas' `.iloc` indexer.

- Missing Values:
  - Missing values will be detected by chaining the `.isnull()` and `.sum()` methods from the Pandas library and calling them on the data frame, returning the total count of missing values for each variable.
- Outliers:
  - Outliers of quantitative variables will be identified using the `boxplot()` function from the Seaborn library.
- Re-expression of Categorical Variables:
  - The churn data dictionary was used to identify which variables required re-expression. By identifying the possible options for each categorical variable, it can be determined which variables are considered ordinal and could therefore be re-expressed using ordinal encoding.

2.0.2 C2. Justification of the Approach for Assessing Data Quality

- The data set contains 10000 observations, so it was necessary to check for duplicates. An efficient way to do so is with Pandas' `.duplicated()` and `.sum()` methods, which when chained, return the sum of the binary series produced by `.duplicated()`, thus counting the total number of duplicates.
- To verify that the first unnamed column and the **CaseOrder** variable are identical, Pandas' `.equals()` method will be used, as this method efficiently checks for element-wise equality between the two objects being compared, and will only return `True` if all are equal. It was necessary to access the unnamed column with Pandas' `.iloc` indexer because the unnamed column could not be accessed via its column name.
- Since it is necessary to check all variables for missing values, an effective way to do so utilized `.isnull()` and `.sum()`. When chained and called on the data frame, this will return a series with the sum of all missing values for each variable in the data frame.
- The boxplot is a quick way to visualize and examine outliers of a data set, and Seaborn's `boxplot()` function provides an equally quick way to generate these plots for ease of visualization of the outliers.
- It is only necessary to consult the data dictionary in order to ascertain which categorical variables require re-expression.

2.0.3 C3. Justification of Programming Language and Libraries/Packages

- Python was selected for this PA for ease of use, readability, and due to its widespread use in data analytics. It was also selected for its ability to handle large data sets efficiently.
- In order to clean the data set, the following libraries/packages will be utilized:
  - **Pandas**: Providing many of the tools used here, Pandas is an essential library because it provides the methods `.isnull()`, `.duplicated()`, `.equals()`, and `.sum()`, which provide important basic functionality that can be used effectively here.
  - **Seaborn**: Seaborn will be used to generate boxplots, which enables observation of the outliers of quantitative variables.
  - `pyplot` from **Matplotlib**: Matplotlib's `pyplot` will be used to generate histograms of the variables, which is a simple and efficient method of observing the distribution of each

variable, to aid in univariate imputation.

- **Numpy**: Numpy will be used in the treatment of a particular outlier anomaly, in which the `where()` function is used to find observations that satisfy a particular condition, which will then be replaced with null values using `numpy.nan`.
- **missingno**: The `matrix()` function from the **missingno** library will be used to generate a matrix of missing values to confirm that all missing values have been treated, as it provides an efficient means of visualizing the missing data.
- `decomposition` from **sklearn**: sklearn's `decomposition` will be used for Principal Component Analysis.

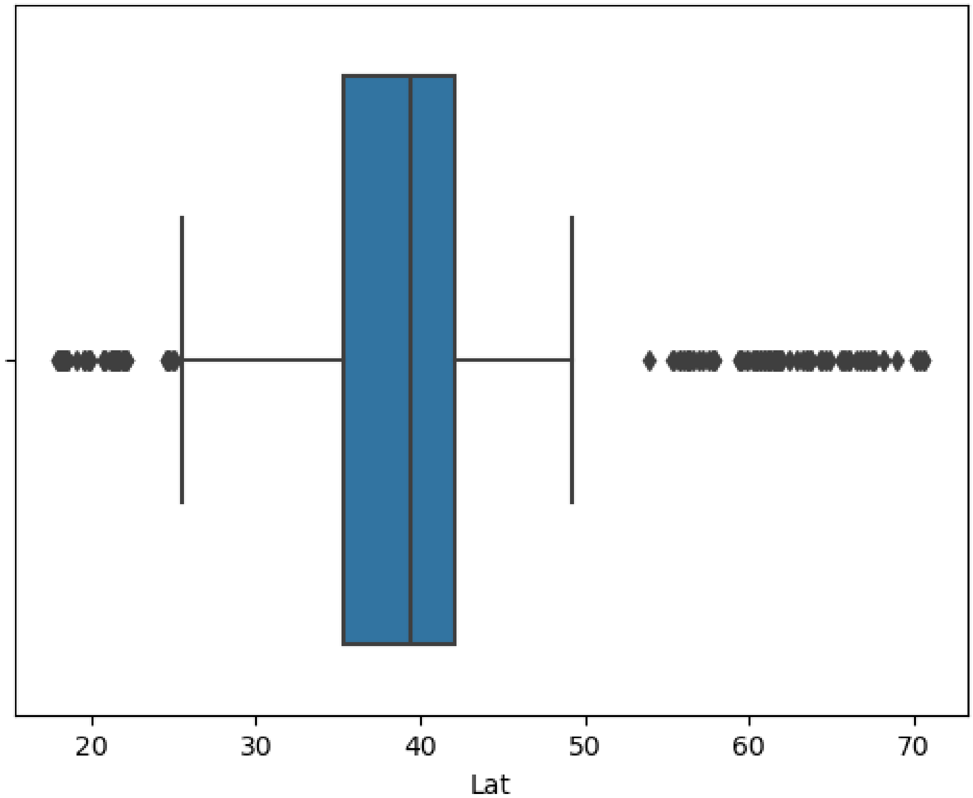
2.0.4 C4. Annotated Code used to Assess Data Quality

The following cells include the annotated code used to assess the quality of the data. See code attached, `D206_PA_MendezD.ipynb`, for the executable script.

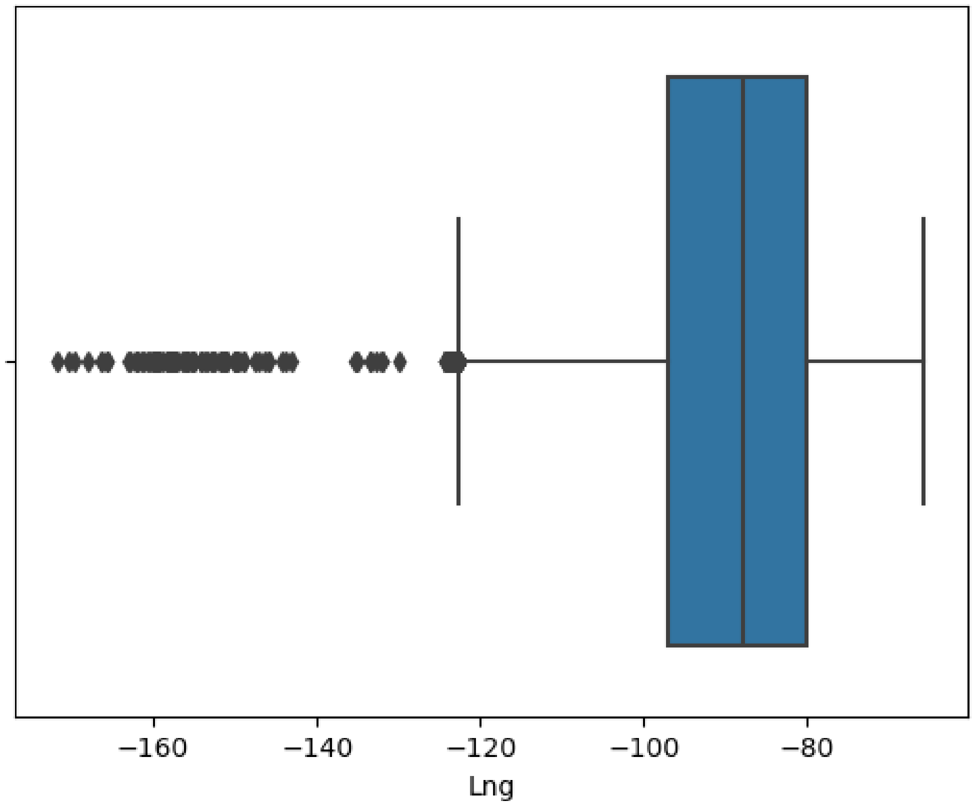
[1]:
```
## C4. The following cells include the annotated code used to assess the␣
↪quality of the data.
# See code attached, in D206_PA_MendezD.ipynb



# C1 Import the Pandas library, then load the data into a data frame with␣
↪Pandas' .read_csv() function
import pandas as pd
df = pd.read_csv('/Users/drewmendez/Documents/WGU/D206/churndict/churn_raw_data.
↪csv')

# C1 Verify that the first unnamed column is identical
# to the second column, the 'CaseOrder' variable

isFirstEqualtoSecond = df.iloc[:, 0].equals(df['CaseOrder'])

# C1 Detect duplicates in the data frame with Pandas' .duplicated() method,
# then sum the resulting series with the .sum() method
duplicate_count = df.duplicated().sum()

# C1 Identify missing values in the data frame with Pandas' .isnull() method,
# then sum the resulting series with the .sum() method
missing_values_count = df.isnull().sum()
```

[2]:
```
# All quantitative variables assessed for outliers using .boxplot() from␣
↪Seaborn.
# The quantitative variables being assessed:
# Lat, Lng, Population, Children, Age, Income, Outage_sec_perweek, Email,␣
↪Contacts,
# Yearly_equip_failure, Tenure, MonthlyCharge, Bandwidth_GB_Year
```

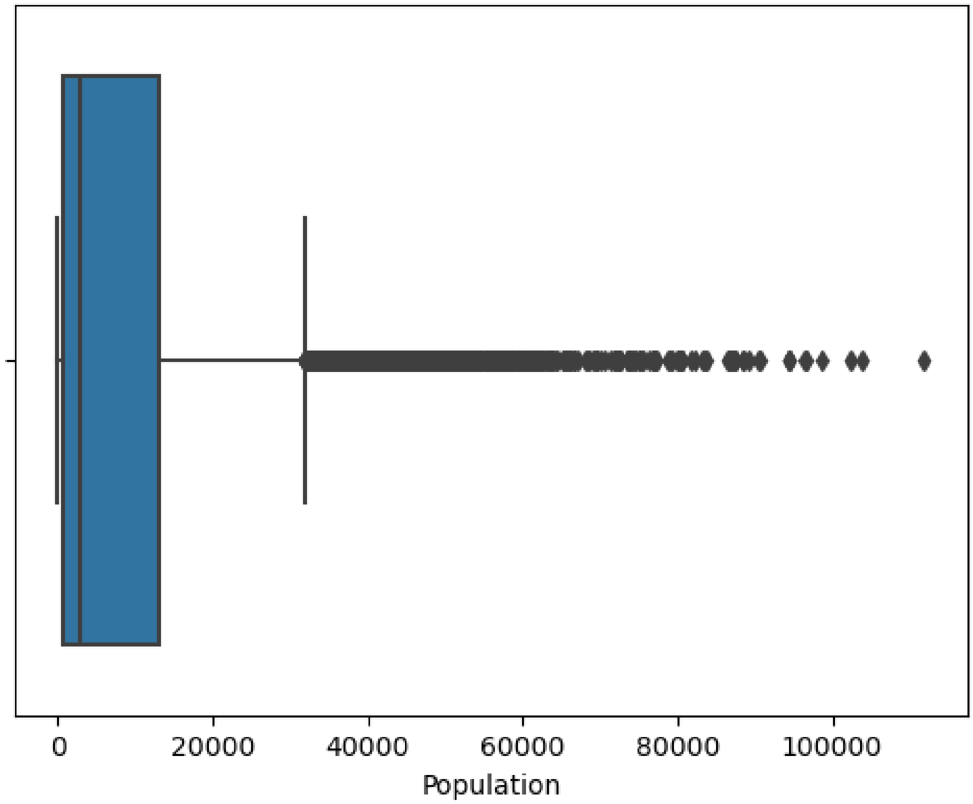
```python
## C1 Detection of 'Lat' Outliers

import seaborn as sb

# Boxplot to visualize 'Lat' outliers
boxplot = sb.boxplot(x = 'Lat', data=df)
```



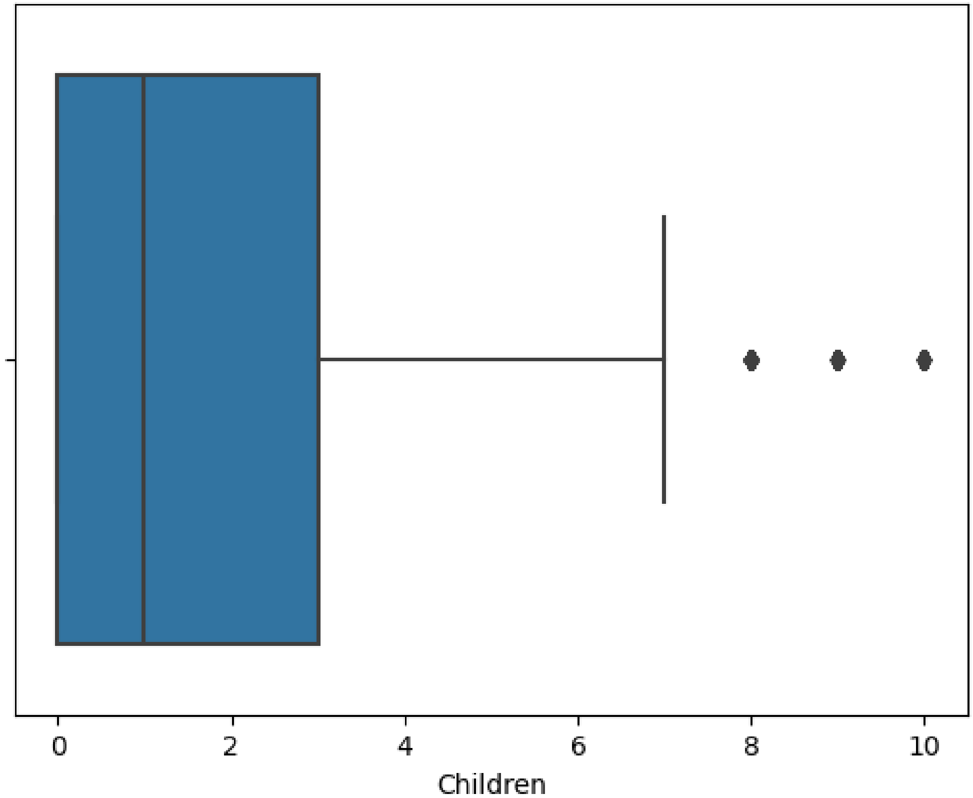
[3]:
```python
## C1 Detection of 'Lng' Outliers

# Boxplot to visualize 'Lng' outliers
boxplot = sb.boxplot(x = 'Lng', data=df)
```



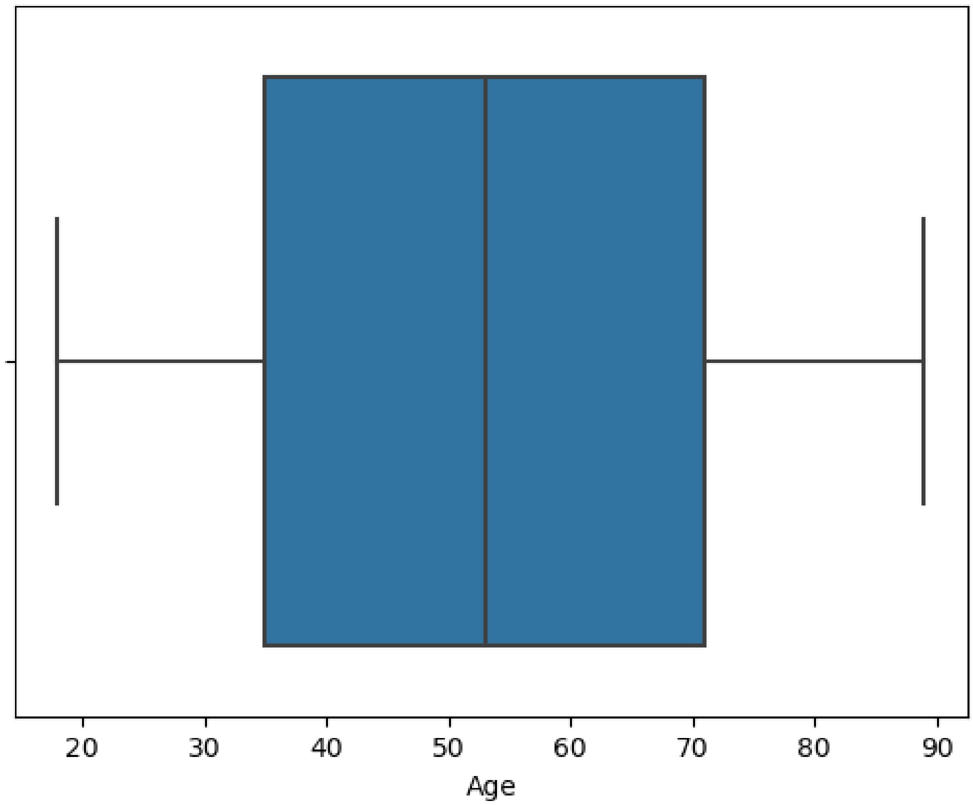
[4]:
```
## C1 Detection of 'Population' Outliers

# Boxplot to visualize 'Population' outliers
boxplot = sb.boxplot(x = 'Population', data=df)
```



[5]:
```
## C1 Detection of 'Children' Outliers

# Boxplot to visualize 'Children' outliers
boxplot = sb.boxplot(x = 'Children', data=df)
```

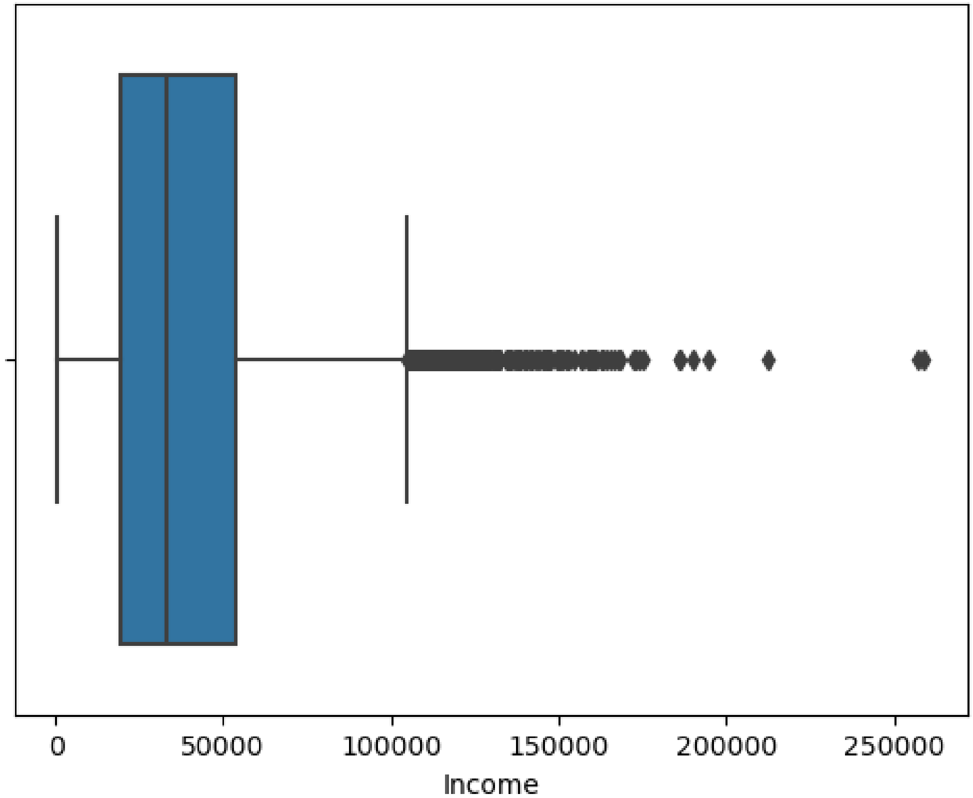


[6]:
```
## C1 Detection of 'Age' Outliers

# Boxplot to visualize 'Age' outliers
boxplot = sb.boxplot(x = 'Age', data=df)
```

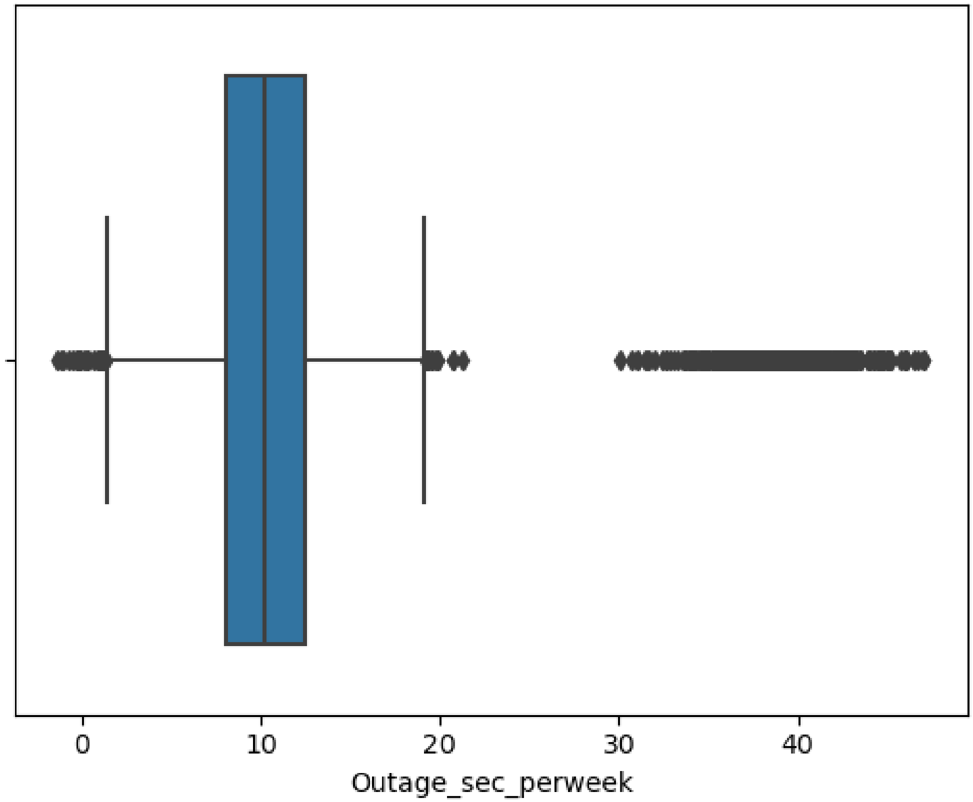


[7]:

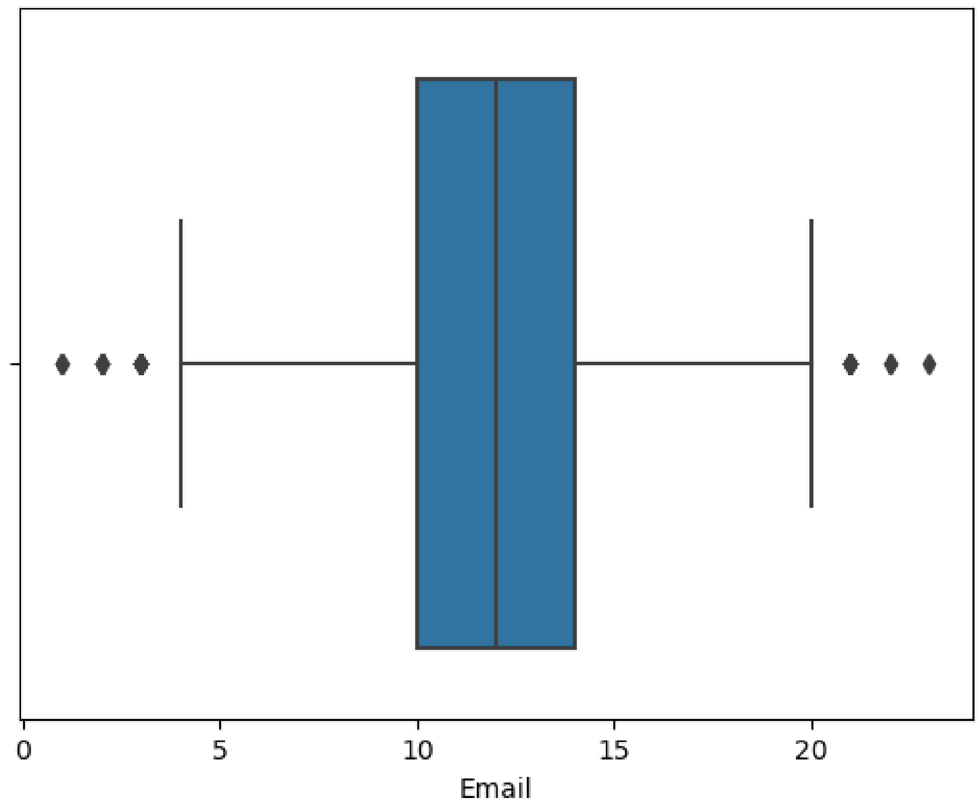
```
## C1 Detection of 'Income' Outliers

# Boxplot to visualize 'Income' outliers
boxplot = sb.boxplot(x = 'Income', data=df)
```



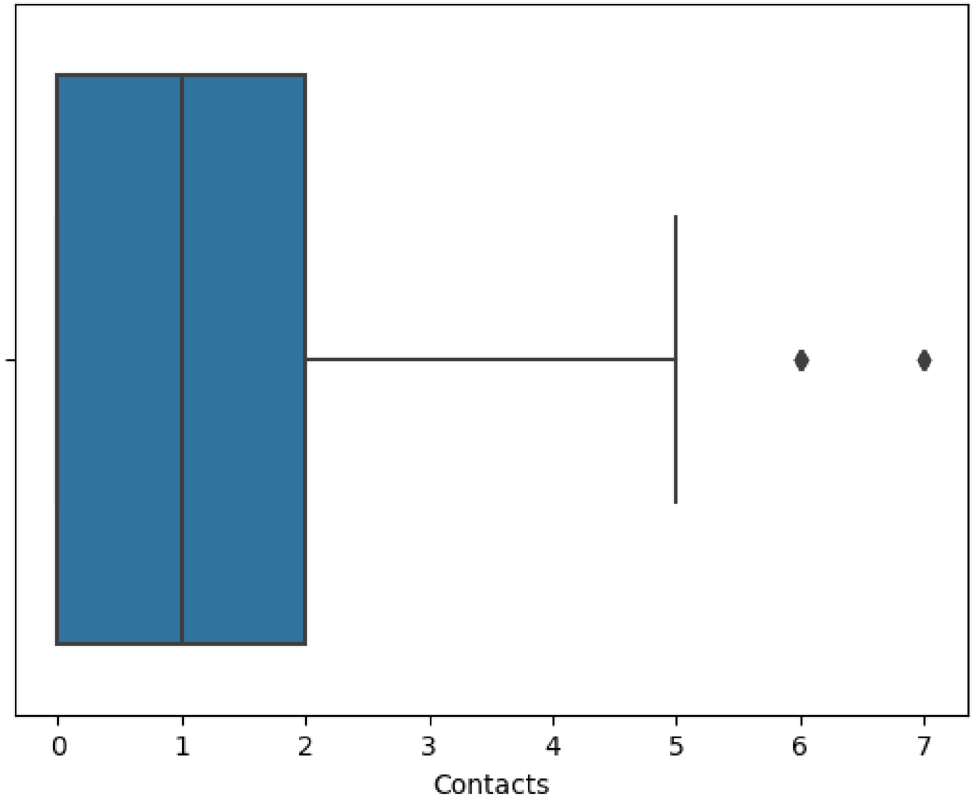
[8]:
```
## C1 Detection of 'Outage_sec_perweek' Outliers

# Boxplot to visualize 'Outage_sec_perweek' outliers
boxplot = sb.boxplot(x = 'Outage_sec_perweek', data=df)
```



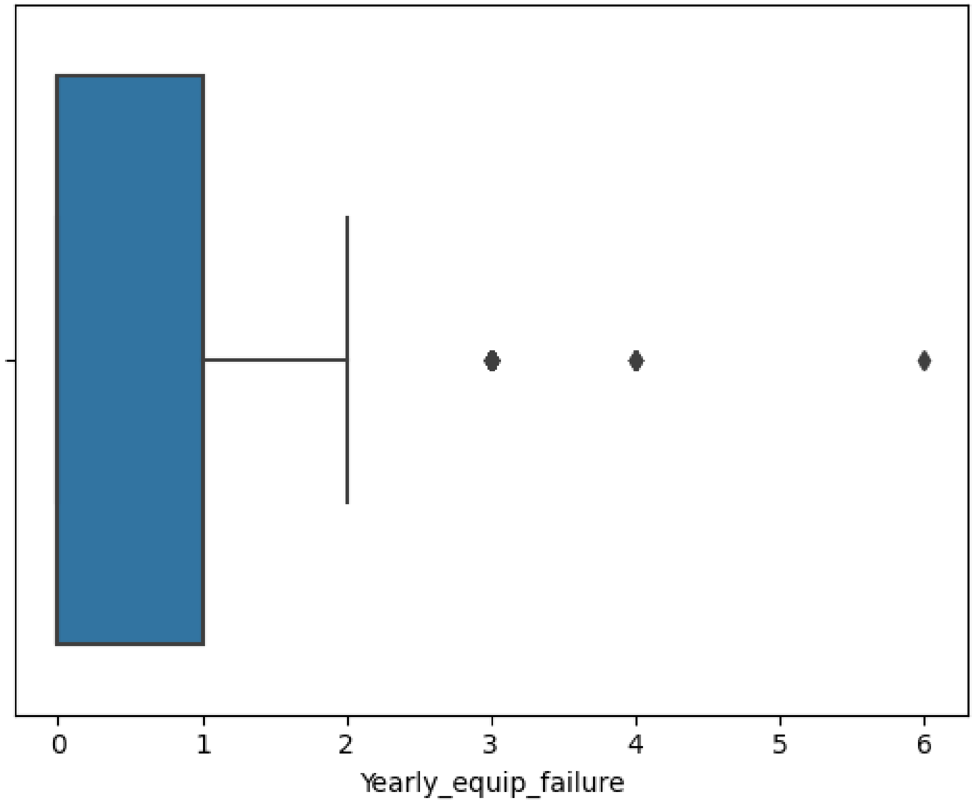
[9]:
```
## C1 Detection of 'Email' Outliers

# Boxplot to visualize 'Email' outliers
boxplot = sb.boxplot(x = 'Email', data=df)
```



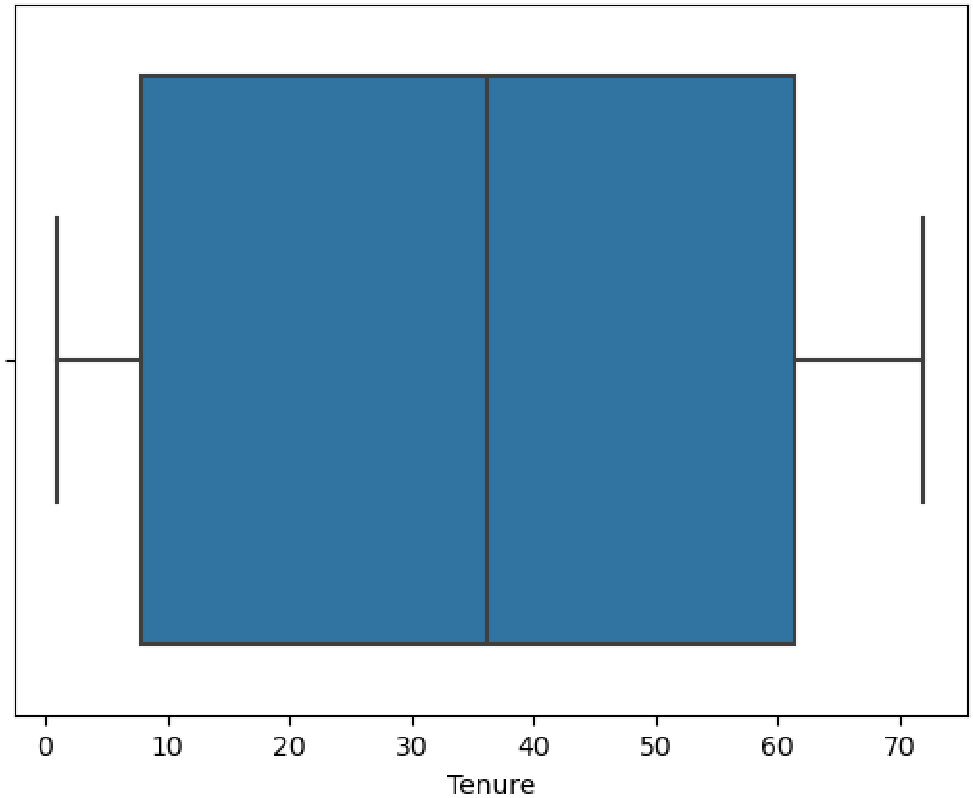
[10]:
```
## C1 Detection of 'Contacts' Outliers

# Boxplot to visualize 'Contacts' outliers
boxplot = sb.boxplot(x = 'Contacts', data=df)
```



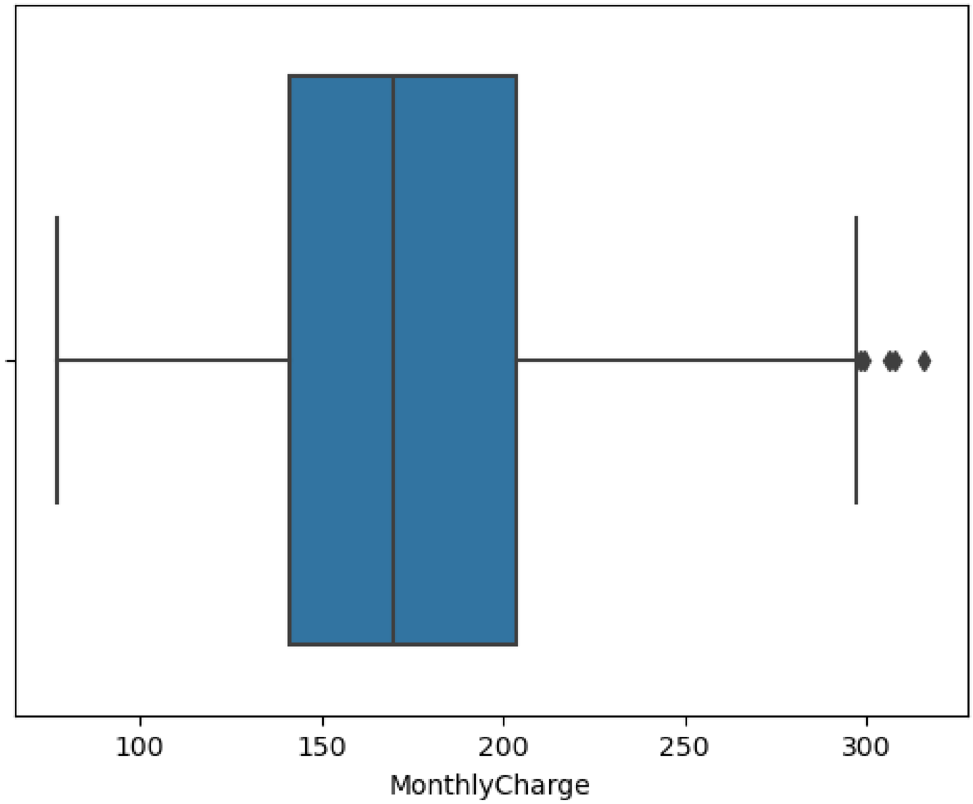
[11]:
```
## C1 Detection of 'Yearly_equip_failure' Outliers

# Boxplot to visualize 'Yearly_equip_failure' outliers
boxplot = sb.boxplot(x = 'Yearly_equip_failure', data=df)
```



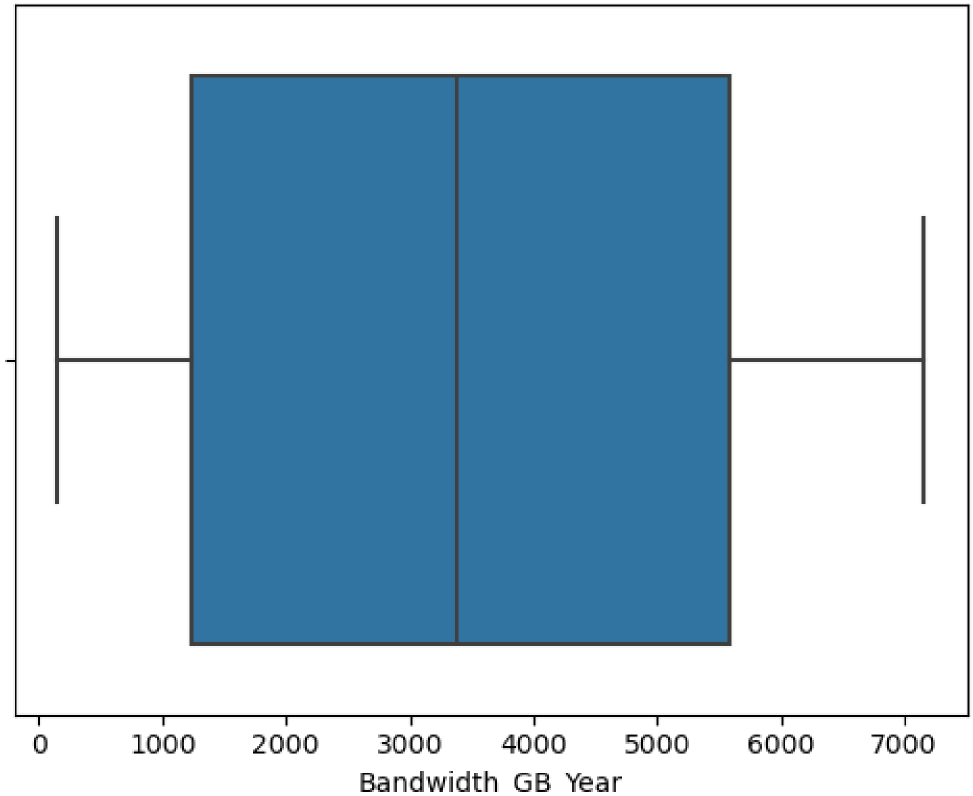
[12]:
```
## C1 Detection of 'Tenure' Outliers

# Boxplot to visualize 'Tenure' outliers
boxplot = sb.boxplot(x = 'Tenure', data=df)
```



[13]:
```
## C1 Detection of 'MonthlyCharge' Outliers

# Boxplot to visualize 'MonthlyCharge' outliers
boxplot = sb.boxplot(x = 'MonthlyCharge', data=df)
```



[14]:
```
## C1 Detection of 'Bandwidth_GB_Year' Outliers

# Boxplot to visualize 'Bandwidth_GB_Year' outliers
boxplot = sb.boxplot(x = 'Bandwidth_GB_Year', data=df)
```



[15]:
```
## The following cells include the code used to count the outliers to respond␣
↪to D1.

# Please consider this the annotation for the cells in which the following code␣
↪is utilized more than once,
# in order to avoid redundantly repeating the same annotation in each of the␣
↪cells below.


# Within these cells, the following is utilized:

# The variables' first and third quartiles, Q1 and Q3 are found using .
↪quantile() from Pandas,
# then the interquartile range, or the difference between Q3 and Q1, is found␣
↪using IQR = Q3 - Q1.

# The upper whisker of the boxplot is found using max = Q3 + 1.5 * IQR.
# The lower whisker of the boxplot is found using min = Q1 - 1.5 * IQR.
```

```
# The .sum() method returns the count of observations greater than the max or␣
↪less than the min.
# The .round() method rounds the outlier count to two decimals.
```

[16]:
```
## D1 Counting 'Lat' Outliers

Q1 = df['Lat'].quantile(0.25)
Q3 = df['Lat'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
min = round(Q1 - 1.5 * IQR, 2)
outlier_count_up = (df['Lat'] > max).sum()
outlier_count_low = (df['Lat'] < min).sum()

outliers_lat = outlier_count_up + outlier_count_low

print('For the Lat variable, all observations greater than', max, 'or less␣
↪than', min, 'are considered outliers.')
print('The count of observations greater than', max,'is', outlier_count_up)
print('The count of observations less than', min,'is', outlier_count_low)
```

```
For the Lat variable, all observations greater than 52.25 or less than 25.19 are
considered outliers.
The count of observations greater than 52.25 is 77
The count of observations less than 25.19 is 81
```

[17]:
```
## D1 Counting 'Lng' Outliers

Q1 = df['Lng'].quantile(0.25)
Q3 = df['Lng'].quantile(0.75)
IQR = Q3 - Q1
min = round(Q1 - 1.5 * IQR, 2)
outliers_lng = (df['Lng'] < min).sum()

print('For the Lng variable, all observations less than', min, 'are considered␣
↪outliers.')
print('The count of observations less than', min,'is', outliers_lng)
```

```
For the Lng variable, all observations less than -122.57 are considered
outliers.
The count of observations less than -122.57 is 273
```

[18]:
```
## D1 Counting 'Population' Outliers

Q1 = df['Population'].quantile(0.25)
Q3 = df['Population'].quantile(0.75)
IQR = Q3 - Q1
max = Q3 + 1.5 * IQR
```

```
outliers_pop = (df['Population'] > max).sum()

print('For the Population variable, all observations greater than', max, 'are␣
↪considered outliers.')
print('The count of observations greater than', max,'is', outliers_pop)
```

```
For the Population variable, all observations greater than 31813.0 are
considered outliers.
The count of observations greater than 31813.0 is 937
```

[19]:
```
## D1 Counting 'Children' Outliers

Q1 = df['Children'].quantile(0.25)
Q3 = df['Children'].quantile(0.75)
IQR = Q3 - Q1
max = Q3 + 1.5 * IQR
outliers_child = (df['Children'] > max).sum()

print('For the Children variable, all observations greater than', max, 'are␣
↪considered outliers.')
print('The count of observations greater than', max,'is', outliers_child)
```

```
For the Children variable, all observations greater than 7.5 are considered
outliers.
The count of observations greater than 7.5 is 302
```

[20]:
```
## D1 Counting 'Income' Outliers

Q1 = df['Income'].quantile(0.25)
Q3 = df['Income'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
outliers_income = (df['Income'] > max).sum()

print('For the Income variable, all observations greater than', max, 'are␣
↪considered outliers.')
print('The count of observations greater than', max,'is', outliers_income)
```

```
For the Income variable, all observations greater than 104752.7 are considered
outliers.
The count of observations greater than 104752.7 is 249
```

[21]:
```
## D1 Counting 'Outage_sec_perweek' Outliers

Q1 = df['Outage_sec_perweek'].quantile(0.25)
Q3 = df['Outage_sec_perweek'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
```

```
min = round(Q1 - 1.5 * IQR, 2)
outlier_count_up = (df['Outage_sec_perweek'] > max).sum()

# Since this variable has negative observations, first find the count of␣
↪observations
# both greater or equal to 0 and less than the min, or the acceptable lower␣
↪outliers
outlier_count_low = ((df['Outage_sec_perweek'] >= 0) &␣
↪(df['Outage_sec_perweek'] < min)).sum()

# Then find the count of observations less than 0
outlier_count_neg = (df['Outage_sec_perweek'] < 0).sum()

outliers_outage = outlier_count_up + outlier_count_low

print('For the Outage_sec_perweek variable, all observations greater than',␣
↪max, 'or less than', min, 'are considered outliers.')
print('The count of observations greater than', max,'is', outlier_count_up)
print('The count of observations less than', min,'and greater than 0 is',␣
↪outlier_count_low)
print('The count of observations less than 0 is', outlier_count_neg)
```

```
For the Outage_sec_perweek variable, all observations greater than 19.14 or less
than 1.4 are considered outliers.
The count of observations greater than 19.14 is 513
The count of observations less than 1.4 and greater than 0 is 15
The count of observations less than 0 is 11
```

[22]:
```
## D1 Counting 'Email' Outliers

Q1 = df['Email'].quantile(0.25)
Q3 = df['Email'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
min = round(Q1 - 1.5 * IQR, 2)
outlier_count_up = (df['Email'] > max).sum()
outlier_count_low = (df['Email'] < min).sum()

outliers_email = outlier_count_up + outlier_count_low

print('For the Email variable, all observations greater than', max, 'or less␣
↪than', min, 'are considered outliers.')
print('The count of observations greater than', max,'is', outlier_count_up)
print('The count of observations less than', min,'is', outlier_count_low)
```

```
For the Email variable, all observations greater than 20.0 or less than 4.0 are
considered outliers.
The count of observations greater than 20.0 is 15
```

```
The count of observations less than 4.0 is 23
```

[23]:
```python
## D1 Counting 'Contacts' Outliers

Q1 = df['Contacts'].quantile(0.25)
Q3 = df['Contacts'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
outliers_contacts = (df['Contacts'] > max).sum()

print('For the Contacts variable, all observations greater than', max, 'are␣
↪considered outliers.')
print('The count of observations greater than', max,'is', outliers_contacts)
```

```
For the Contacts variable, all observations greater than 5.0 are considered
outliers.
The count of observations greater than 5.0 is 8
```

[24]:
```python
## D1 Counting 'Yearly_equip_failure' Outliers

Q1 = df['Yearly_equip_failure'].quantile(0.25)
Q3 = df['Yearly_equip_failure'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
outliers_failure = (df['Yearly_equip_failure'] > max).sum()

print('For the Yearly_equip_failure variable, all observations greater than',␣
↪max, 'are considered outliers.')
print('The count of observations greater than', max,'is', outliers_failure)
```

```
For the Yearly_equip_failure variable, all observations greater than 2.5 are
considered outliers.
The count of observations greater than 2.5 is 94
```

[25]:
```python
## D1 Counting 'MonthlyCharge' Outliers

Q1 = df['MonthlyCharge'].quantile(0.25)
Q3 = df['MonthlyCharge'].quantile(0.75)
IQR = Q3 - Q1
max = round(Q3 + 1.5 * IQR, 2)
outliers_monthly = (df['MonthlyCharge'] > max).sum()

print('For the MonthlyCharge variable, all observations greater than', max,␣
↪'are considered outliers.')
print('The count of observations greater than', max,'is', outliers_monthly)
```

```
For the MonthlyCharge variable, all observations greater than 297.84 are
considered outliers.
The count of observations greater than 297.84 is 5
```

3 Part III: Data Cleaning

3.1 D. Summary of the Data-Cleaning Process

3.1.1 D1. Description of Data Quality Issues

- Duplicates: according to the output of the code below, as determined in C4, it is true that the first unnamed column is identical to the second column, the 'CaseOrder' variable.

[26]:
```
## Duplicate columns, as found in C4

print('Is the first column equal to the second column?', isFirstEqualtoSecond)
```

```
Is the first column equal to the second column? True
```

- Duplicates: according to the output of the code below, as determined in C4, there were no duplicate observations detected in the data set.

[27]:
```
## D1. Duplicate observations, as found in C4

print('Number of duplicate rows:', duplicate_count)
```

```
Number of duplicate rows: 0
```

- Missing values: as found in C4, according to the output of the code below, the following variables were shown to have missing values:
  - Children: 2495 missing values
  - Age: 2475 missing values
  - Income: 2490 missing values
  - Techie: 2477 missing values
  - InternetService: 2129 missing values
    - According to the data dictionary, the 'InternetService' variable has "None" as an option, so these are in fact not missing values.
  - Phone: 1026 missing values
  - TechSupport: 991 missing values
  - Tenure: 931 missing values
  - Bandwidth_GB_Year: 1021 missing values

[28]:
```
# D1. Missing Values, as found in C4

print("Number of missing values per variable:")
print(missing_values_count)
```

```
Number of missing values per variable:
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
item1                      0
item2                      0
item3                      0
item4                      0
item5                      0
item6                      0
item7                      0
item8                      0
dtype: int64
```

- Outliers:

- The number of outliers in the 'Lat' variable: 158
- The number of outliers in the 'Lng' variable: 273
- The number of outliers in the 'Population' variable: 937
- The number of outliers in the 'Children' variable: 302
- The number of outliers in the 'Income' variable: 249
- The number of outliers in the 'Outage_sec_perweek' variable: 539, 11 of which are negative
- The number of outliers in the 'Email' variable: 38
- The number of outliers in the 'Contacts' variable: 8
- The number of outliers in the 'Yearly_equip_failure' variable: 94
- The number of outliers in the 'MonthlyCharge' variable: 5

[29]:
```
## D1. Outliers

print('The number of outliers in the 'Lat' variable:', outliers_lat)
print('The number of outliers in the 'Lng' variable:', outliers_lng)
print('The number of outliers in the 'Population' variable:', outliers_pop)
print('The number of outliers in the 'Children' variable:', outliers_child)
print('The number of outliers in the 'Income' variable:', outliers_income)
print('The number of outliers in the 'Outage_sec_perweek' variable:',␣
↪outliers_outage)
print('The number of negative values in the 'Outage_sec_perweek' variable is',␣
↪outlier_count_neg)
print('The number of outliers in the 'Email' variable:', outliers_email)
print('The number of outliers in the 'Contacts' variable:', outliers_contacts)
print('The number of outliers in the 'Yearly_equip_failure' variable:',␣
↪outliers_failure)
print('The number of outliers in the 'MonthlyCharge' variable:',␣
↪outliers_monthly)
```

```
The number of outliers in the 'Lat' variable: 158
The number of outliers in the 'Lng' variable: 273
The number of outliers in the 'Population' variable: 937
The number of outliers in the 'Children' variable: 302
The number of outliers in the 'Income' variable: 249
The number of outliers in the 'Outage_sec_perweek' variable: 528
The number of negative values in the 'Outage_sec_perweek' variable is 11
The number of outliers in the 'Email' variable: 38
The number of outliers in the 'Contacts' variable: 8
The number of outliers in the 'Yearly_equip_failure' variable: 94
The number of outliers in the 'MonthlyCharge' variable: 5
```

3.1.2 D2. Justification for Methods Used to Mitigate Data Quality Issues

- Duplicates:
  - Rows: Since there were no duplicate rows, treatment was concluded upon verification that there were no duplicates.
  - Variables: According to the data dictionary, there should be one index variable, the

    ‘CaseOrder’ variable. Upon inspection, it was observed that there were actually two index variables, one of which was unnamed. It was shown that these two variables were identical, then the unnamed variable was dropped from the data frame.

- Missing values: Missing values were imputed with the median if the distribution of the variable was skewed or bimodal, imputed with the mean if the distribution was normal or uniform, or imputed with the mode if the variable was categorical. (Middleton, K.)
  - ‘InternetService’: Since the ‘InternetService’ variable has ‘None’ as one of its options, these erroneously identified null values were imputed with ‘None’ to avoid being interpreted as nulls.
  - ‘Children’ missing values: Since the distribution of the ‘Children’ variable is skewed right, missing values were imputed with the median.
  - ‘Age’ missing values: Since the ‘Age’ variable has a uniform distribution, missing values were imputed with the mean.
  - ‘Income’ missing values: Since the distribution of the ‘Income’ variable is skewed right, missing values were imputed with the median.
  - ‘Techie’ missing values: Since the ‘Techie’ variable is categorical, missing values were imputed with the mode.
  - ‘Phone’ missing values: Since the ‘Phone’ variable is categorical, missing values were imputed with the mode.
  - ‘TechSupport’ missing values: Since the ‘TechSupport’ variable is categorical, missing values were imputed with the mode.
  - ‘Tenure’ missing values: Since the ‘Tenure’ variable has a bimodal distribution, missing values were imputed with the median.
  - ‘Bandwidth_GB_Year’ missing values: Since the ‘Bandwidth_GB_Year’ variable has a bimodal distribution, missing values were imputed with the median.
- Outliers:
  - ‘Lat’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Lng’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Population’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Children’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Income’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Outage_sec_perweek’ outliers: 539 of these outliers were observed.
    * Only 528 were acceptable and retained, while 11 of them were negative, which was unreasonable, as these should be positive values. These were imputed with with NANs using `numpy.nan`, and since the distribution was skewed right, these were then imputed with the median.
  - ‘Email’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Contacts’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘Yearly_equip_failure’ outliers: Since these outliers were observed to be in the acceptable range for the variable, they were retained.
  - ‘MonthlyCharge’ outliers: Since these outliers were observed to be in the acceptable

range for the variable, they were retained.

- Re-expression:
  - 'Education': Since the 'Education' variable is an ordinal categorical variable, it was re-expressed using numeric values in a new column, 'Education_numeric', using the following mapping:
    - Master's Degree: 18
    - Regular High School Diploma: 12
    - Doctorate Degree: 20
    - No Schooling Completed: 0
    - Associate's Degree: 14
    - Bachelor's Degree: 16
    - Some College, Less than 1 year: 13
    - GED or Alternative Credential: 12
    - Some College, 1 or More Years, No Degree: 14
    - 9th Grade to 12th Grade, No Diploma: 11
    - Nursery School to 8th Grade: 8
    - Professional School Degree: 18

3.1.3 D3. Summary of the Outcome of Implementing Each Data Cleaning Step

Detection

- Duplicate columns and observations were detected. The potential duplicate column was verified to be identical to the 'CaseOrder' variable using Pandas' `.equals()` method. The data set was then checked for duplicate observations by calling Pandas' `.duplicated()` method on the data frame. The resulting series was then summed using the `.sum()` method to determine the count of all duplicate observations. - Missing values of each variable were detected by calling Pandas' `.isnull()` method on the data frame. The resulting series was then summed using the `.sum()` method to determine the count of all missing values. - Outliers were visually detected using Seaborn's `boxplot()` function. If a variable was found to have outliers, the number of outliers was determined. In order to count the outliers, it was necessary to calculate the values of the boxplot's upper and lower whiskers. Then the upper outliers were found to be greater than the value of the upper whisker, and lower outliers were found to be less than the value of the lower whisker. The total counts of outliers were then stored in appropriately named variables.

Treatment

- The duplicate column, unnamed column, was dropped by calling Pandas' `.drop()` method on the data frame. This resulted in a data frame of reduced size, with one less column. - No duplicate observations were detected, so the data frame remained unchanged after performing this step. - The distributions of variables containing missing values were observed using Matplotlib's `plot()` function to generate histograms of each variable. Based on each distribution, the missing values that were detected were then imputed using univariate imputation, utilizing Pandas' `.fillna()` method chained with the appropriate statistical measure. A histogram of each newly treated variable was then generated in order to verify that the distribution had not been drastically changed. - Outliers were inspected and gauged against the data dictionary and range of expected values to determine the course of action. All but one variable had outliers that were determined to be reasonable, and as such, were retained. - One variable, 'Outage_sec_perweek', was found to have negative values, which were outside the range of what was considered reasonable for that variable. These negative values were isolated, converted to nulls, and then imputed with the median.

3.2 D3. Visual Evidence Confirming Data is Cleaned

This section has been moved to the end of section D4 in order to execute it in the proper order.

3.3 D4. Annotated Code used to Mitigate Data Quality Issues

The following cells include the annotated code used to mitigate data quality issues, which includes treating duplicates, missing values, outliers, as well as addressing anomalies. See code attached, `D206_PA_MendezD.ipynb`, for the executable script.

[30]:
```
## D4. The following cells include the annotated code used to mitigate the data␣
↪quality issues.
# See code attached, in D206_PA_MendezD.ipynb

# Before dropping the first unnamed column, first verify that it is identical
# to the second column, the 'CaseOrder' variable

print('Is the first column equal to the second column?', isFirstEqualtoSecond)
```

```
Is the first column equal to the second column? True
```

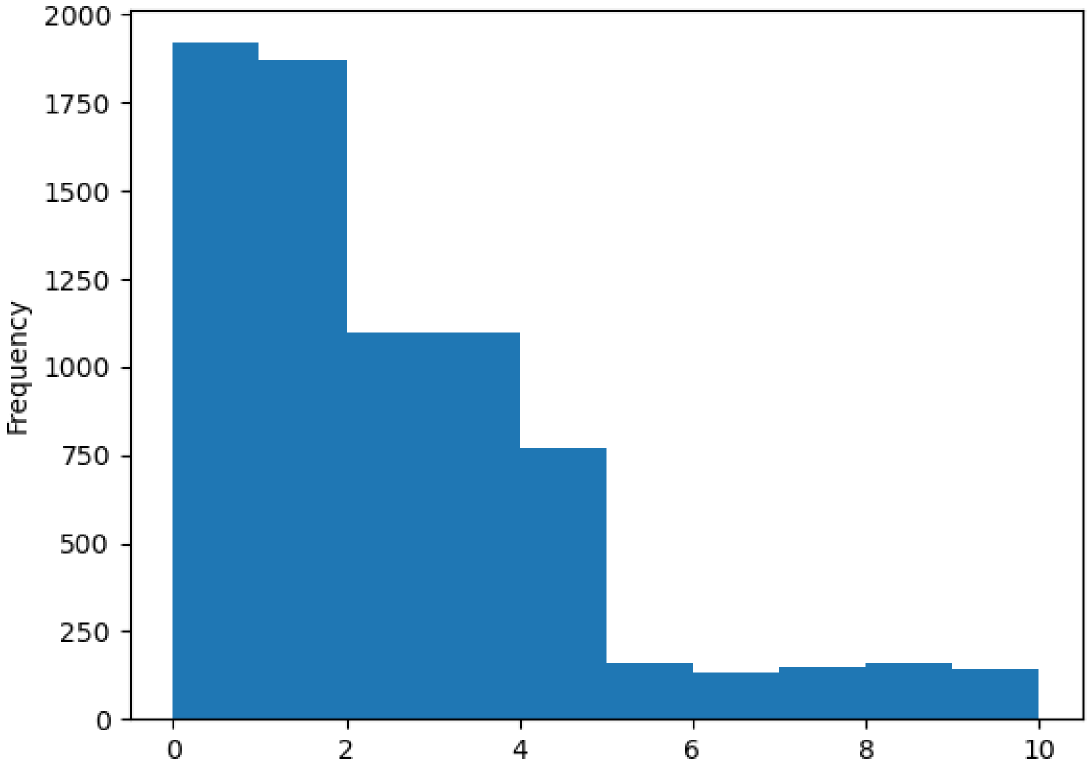
[31]:
```
## D4. The following cells include the annotated code used to mitigate the data␣
↪quality issues.
# See code attached, in D206_PA_MendezD.ipynb


# Since the first unnamed column and the second column, the 'CaseOrder'␣
↪variable are identical,
# proceed with dropping the unnamed column

df = df.drop(df.columns[0], axis=1)

# Verify the unnamed column was dropped
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 51 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   CaseOrder              10000 non-null  int64
 1   Customer_id            10000 non-null  object
 2   Interaction            10000 non-null  object
 3   City                   10000 non-null  object
 4   State                  10000 non-null  object
 5   County                 10000 non-null  object
 6   Zip                    10000 non-null  int64
 7   Lat                    10000 non-null  float64
 8   Lng                    10000 non-null  float64
```

```
 9   Population            10000 non-null  int64
 10  Area                  10000 non-null  object
 11  Timezone              10000 non-null  object
 12  Job                   10000 non-null  object
 13  Children              7505 non-null   float64
 14  Age                   7525 non-null   float64
 15  Education             10000 non-null  object
 16  Employment            10000 non-null  object
 17  Income                7510 non-null   float64
 18  Marital               10000 non-null  object
 19  Gender                10000 non-null  object
 20  Churn                 10000 non-null  object
 21  Outage_sec_perweek    10000 non-null  float64
 22  Email                 10000 non-null  int64
 23  Contacts              10000 non-null  int64
 24  Yearly_equip_failure  10000 non-null  int64
 25  Techie                7523 non-null   object
 26  Contract              10000 non-null  object
 27  Port_modem            10000 non-null  object
 28  Tablet                10000 non-null  object
 29  InternetService       7871 non-null   object
 30  Phone                 8974 non-null   object
 31  Multiple              10000 non-null  object
 32  OnlineSecurity        10000 non-null  object
 33  OnlineBackup          10000 non-null  object
 34  DeviceProtection      10000 non-null  object
 35  TechSupport           9009 non-null   object
 36  StreamingTV           10000 non-null  object
 37  StreamingMovies       10000 non-null  object
 38  PaperlessBilling      10000 non-null  object
 39  PaymentMethod         10000 non-null  object
 40  Tenure                9069 non-null   float64
 41  MonthlyCharge         10000 non-null  float64
 42  Bandwidth_GB_Year     8979 non-null   float64
 43  item1                 10000 non-null  int64
 44  item2                 10000 non-null  int64
 45  item3                 10000 non-null  int64
 46  item4                 10000 non-null  int64
 47  item5                 10000 non-null  int64
 48  item6                 10000 non-null  int64
 49  item7                 10000 non-null  int64
 50  item8                 10000 non-null  int64
dtypes: float64(9), int64(14), object(28)
memory usage: 3.9+ MB
```

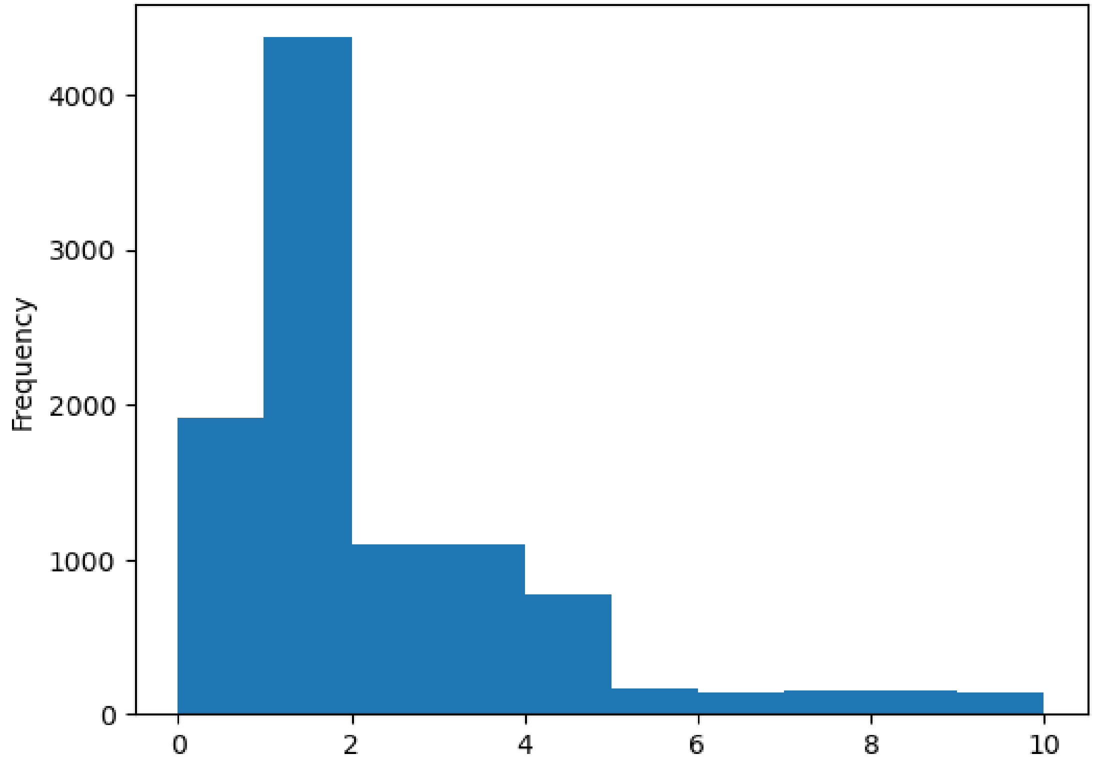
[32]:
```
import matplotlib.pyplot as plt
```

```
# Generate a histogram of the 'Children' variable to observe distribution
df['Children'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```

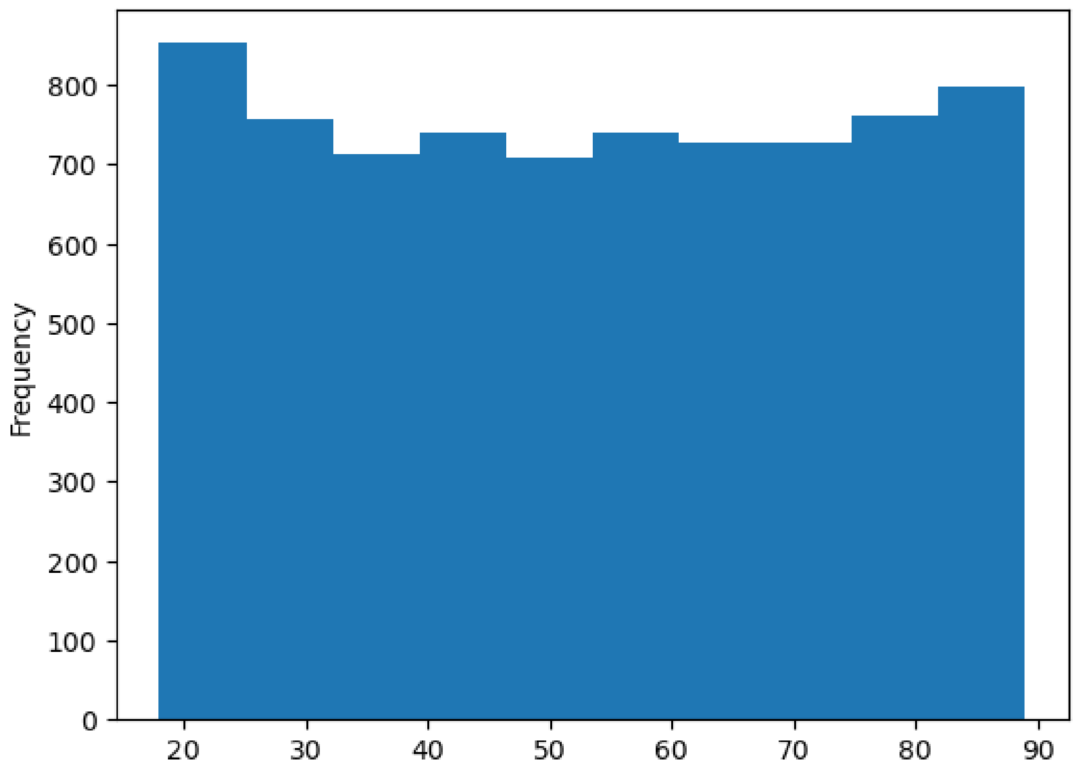



```
[33]: # Since the 'Children' variable is skewed right, impute with the median

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪median
df['Children'].fillna(df['Children'].median(), inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['Children'].isnull().sum())

# Generate a new histogram of the 'Children' variable to observe unchanged␣
↪distribution
df['Children'].plot(kind='hist')

# Display the histogram
plt.show()
```

```
Number of nulls: 0
```

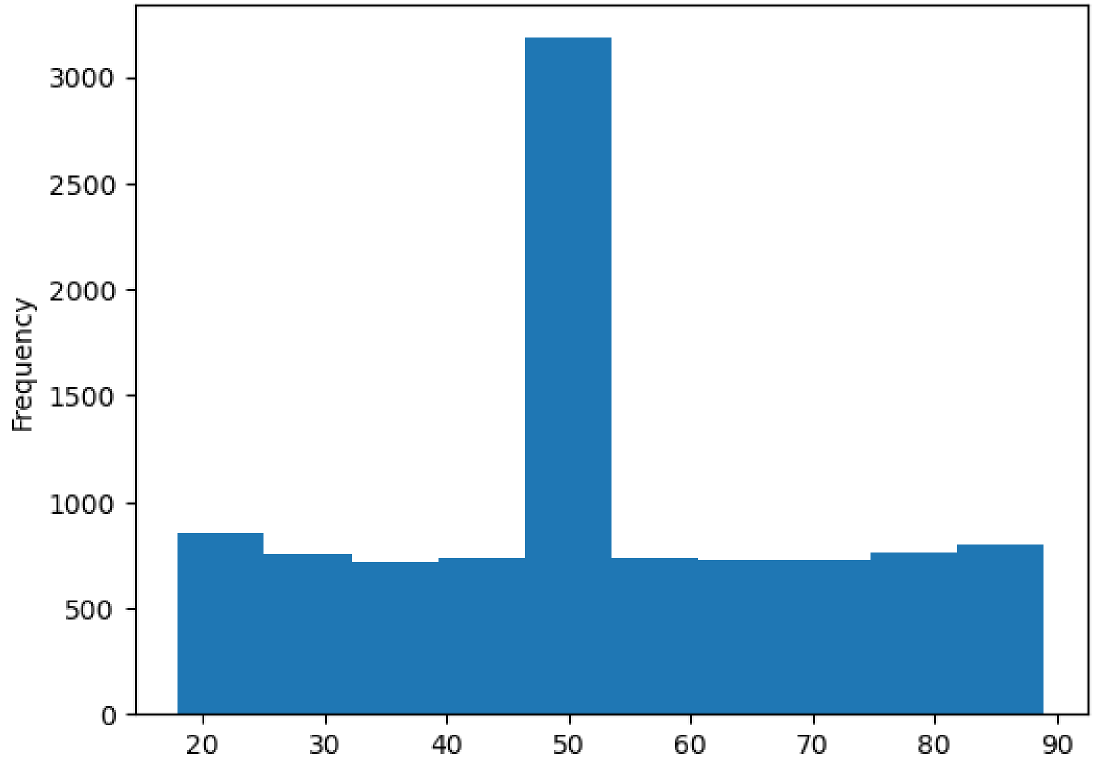


[34]:
```
# Generate a histogram of the 'Age' variable to observe distribution
df['Age'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```

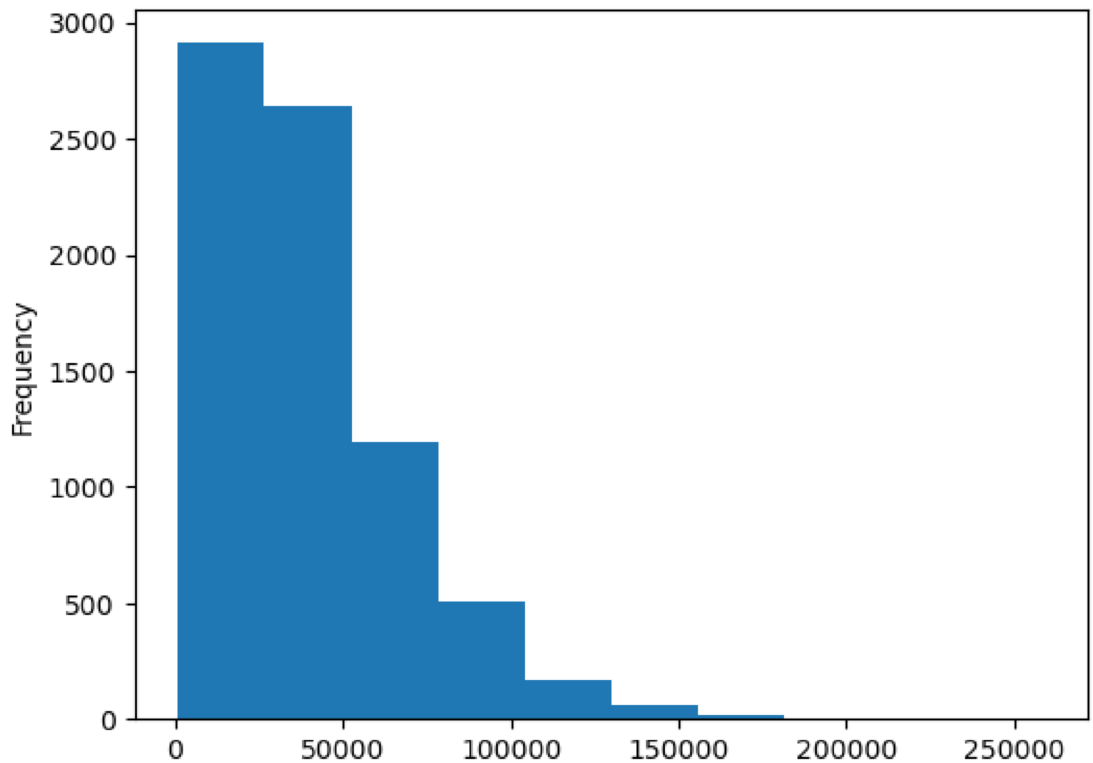


[35]:
```
# Since the 'Age' variable has a uniform distribution, impute with the mean.

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪mean
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['Age'].isnull().sum())

# Generate a new histogram of the 'Age' variable to observe unchanged␣
↪distribution
df['Age'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```

```
Number of nulls: 0
```

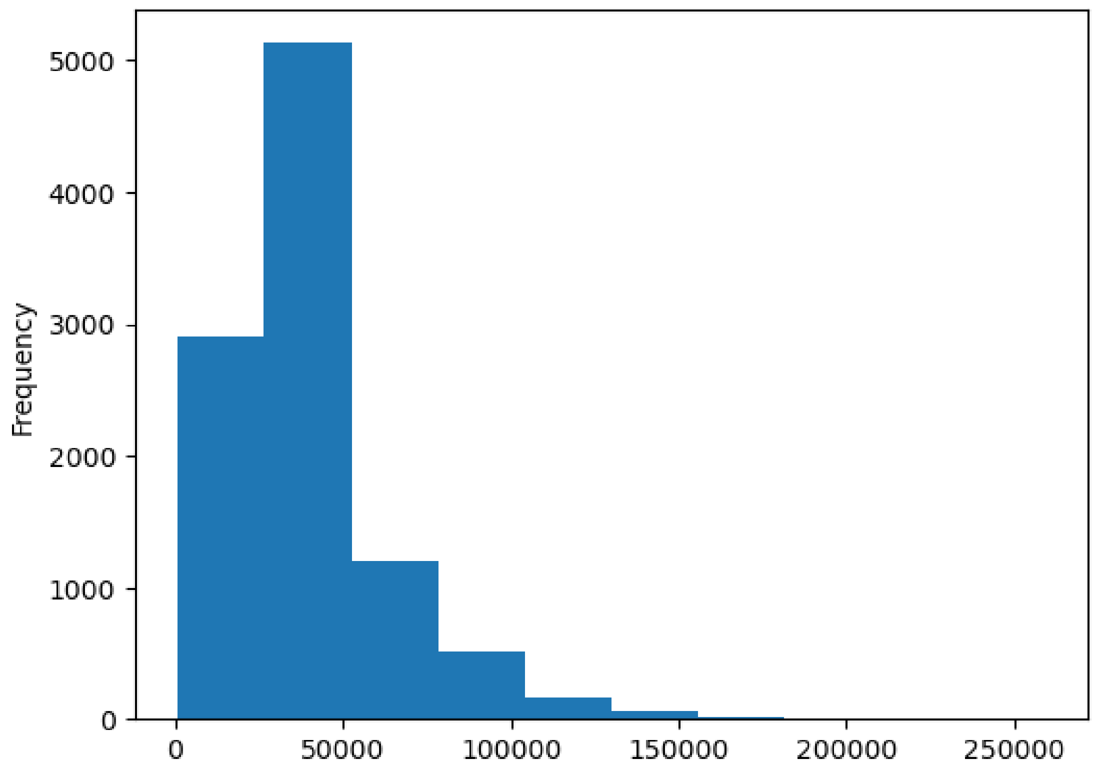


[36]:
```
# Generate a histogram of the 'Income' variable to observe distribution
df['Income'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```



[37]:
```
# Since the 'Income' variable is skewed right, impute with the median

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪median
df['Income'].fillna(df['Income'].median(), inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['Income'].isnull().sum())

# Generate a new histogram of the 'Income' variable to observe unchanged␣
↪distribution
df['Income'].plot(kind='hist')

# Display the histogram
plt.show()
```

```
Number of nulls: 0
```



[38]:
```
# Since the 'Techie' variable is categorical, impute with the mode

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪mode
df['Techie'].fillna(df['Techie'].mode()[0], inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['Techie'].isnull().sum())
```

```
Number of nulls: 0
```

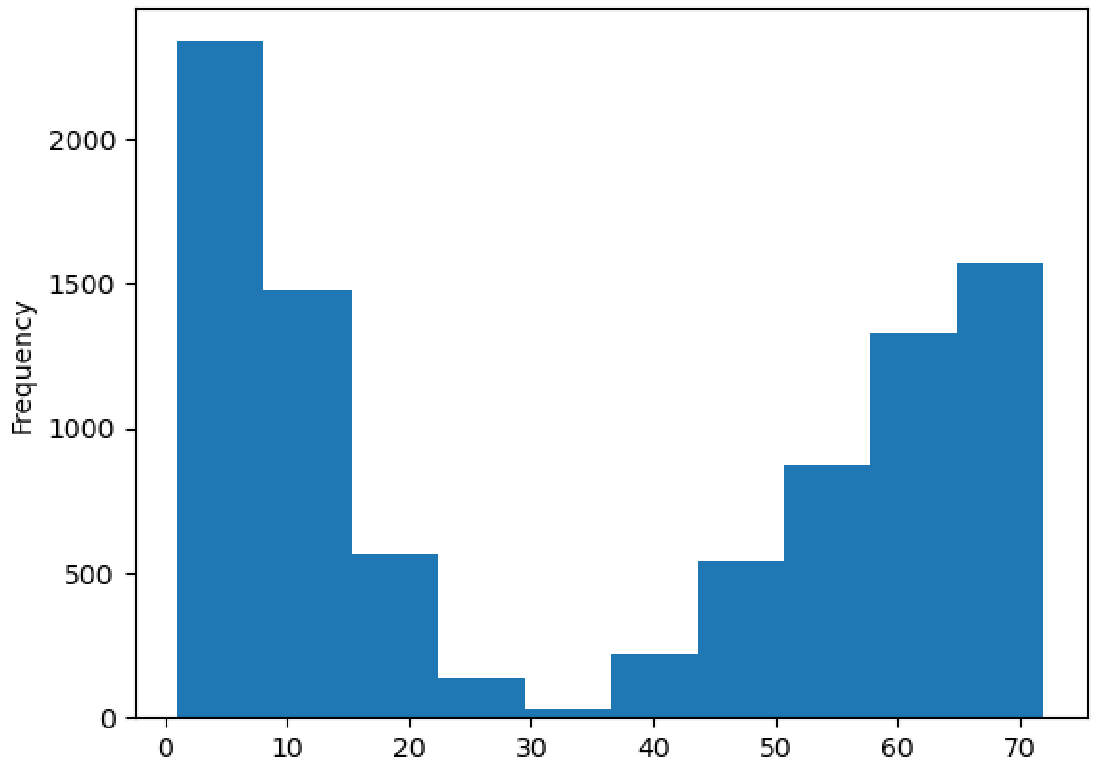
[39]:
```
# Since the 'Phone' variable is categorical, impute with the mode

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪mode
df['Phone'].fillna(df['Phone'].mode()[0], inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['Phone'].isnull().sum())
```

```
Number of nulls: 0
```

[40]:
```
# Since the 'TechSupport' variable is categorical, impute with the mode

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪mode
df['TechSupport'].fillna(df['TechSupport'].mode()[0], inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['TechSupport'].isnull().sum())
```

```
Number of nulls: 0
```

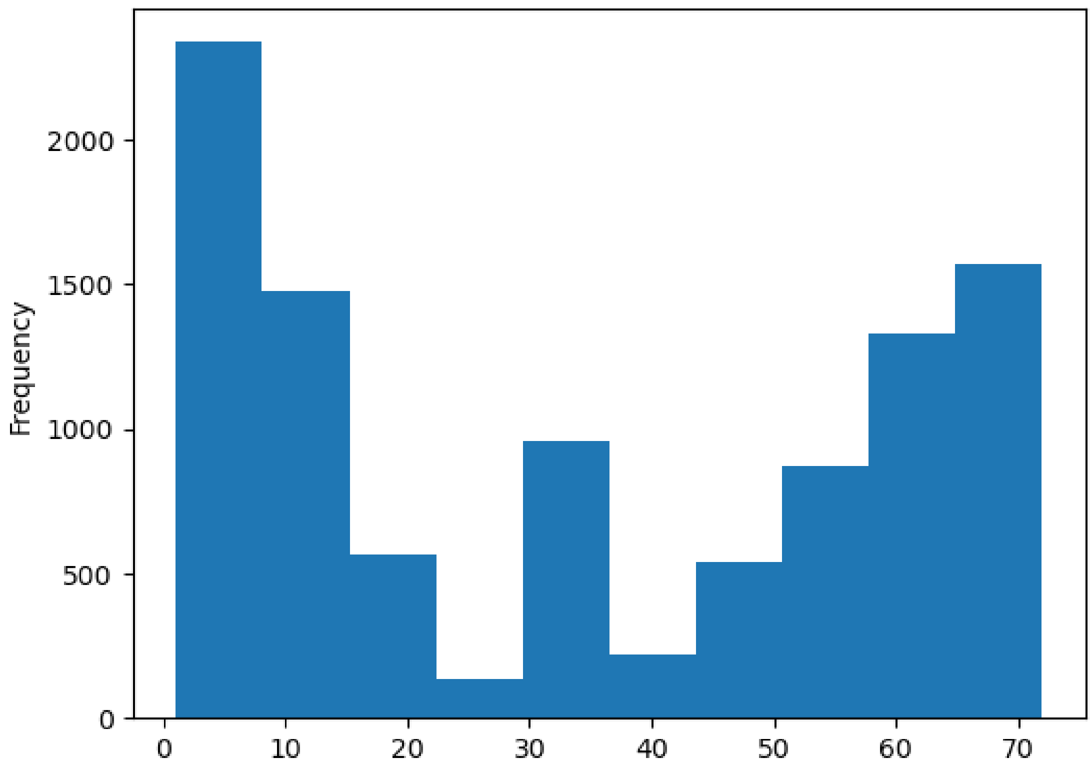
[41]:
```
# Generate a histogram of the 'Tenure' variable
df['Tenure'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```



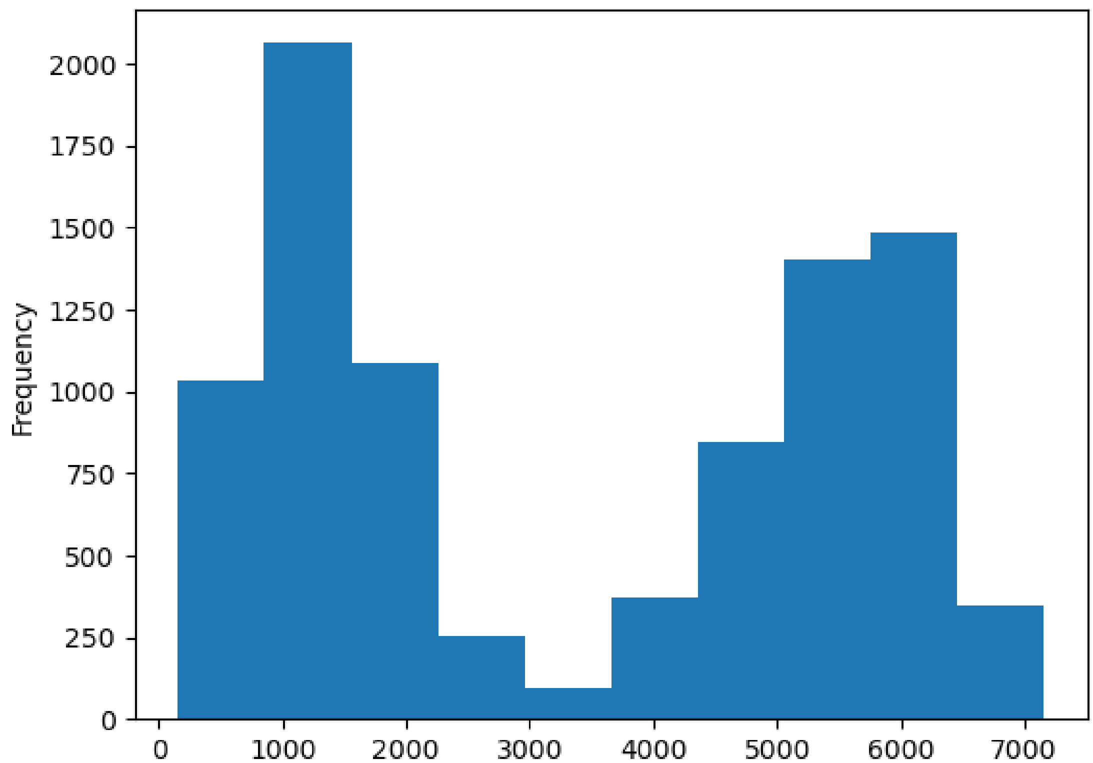
[42]:
```
# Since the 'Tenure' variable is bimodal, impute with the median

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪median
df['Tenure'].fillna(df['Tenure'].median(), inplace=True)
```

```
# Verify that all nulls are treated
print('Number of nulls:', df['Tenure'].isnull().sum())

# Generate a new histogram of the 'Tenure' variable to observe unchanged␣
↪distribution
df['Tenure'].plot(kind='hist')

# Display the histogram to observe unchanged distribution
plt.show()
```

```
Number of nulls: 0
```

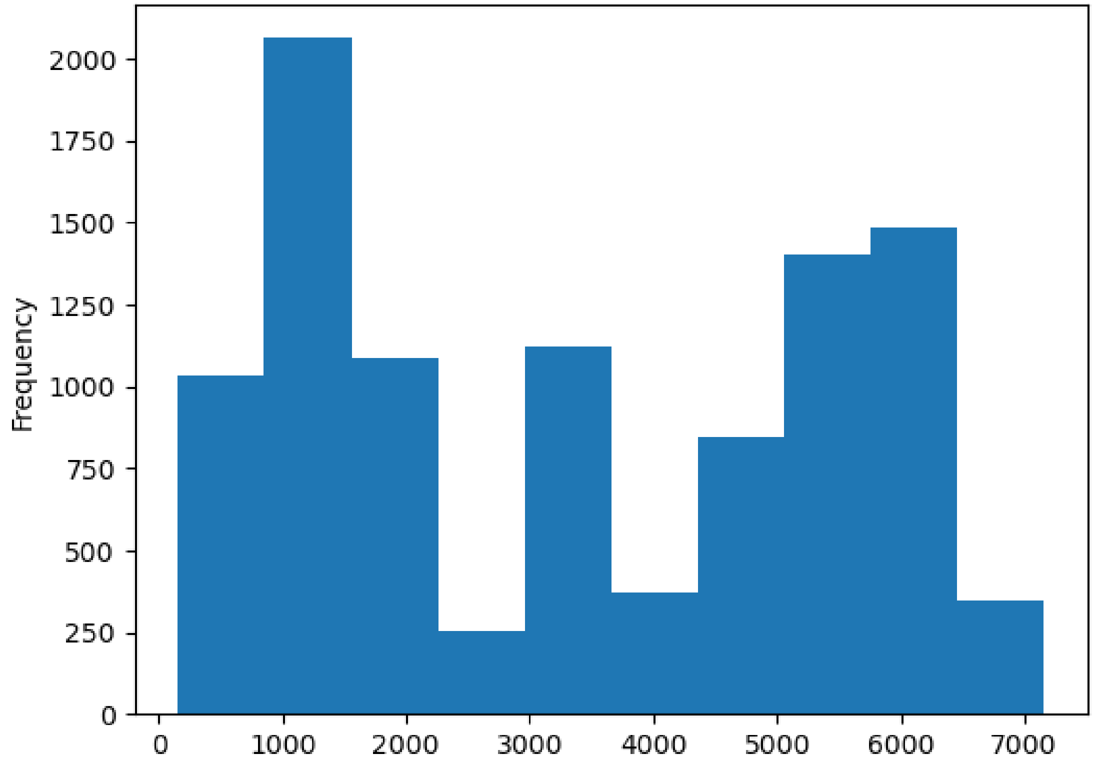


[43]:
```
# Generate a histogram of the 'Bandwidth_GB_Year' variable
df['Bandwidth_GB_Year'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```



[44]:
```
# Since the 'Bandwidth_GB_Year' variable is bimodal, impute with the median

# Call the data frame and variable, then fill all NAs of that variable with its␣
↪median
df['Bandwidth_GB_Year'].fillna(df['Bandwidth_GB_Year'].median(), inplace=True)

# Verify that all nulls are treated
print('Number of nulls:', df['Bandwidth_GB_Year'].isnull().sum())

# Generate a new histogram of the 'Bandwidth_GB_Year' variable to observe␣
↪unchanged distribution
df['Bandwidth_GB_Year'].plot(kind='hist')

# Display the histogram
plt.show()
```

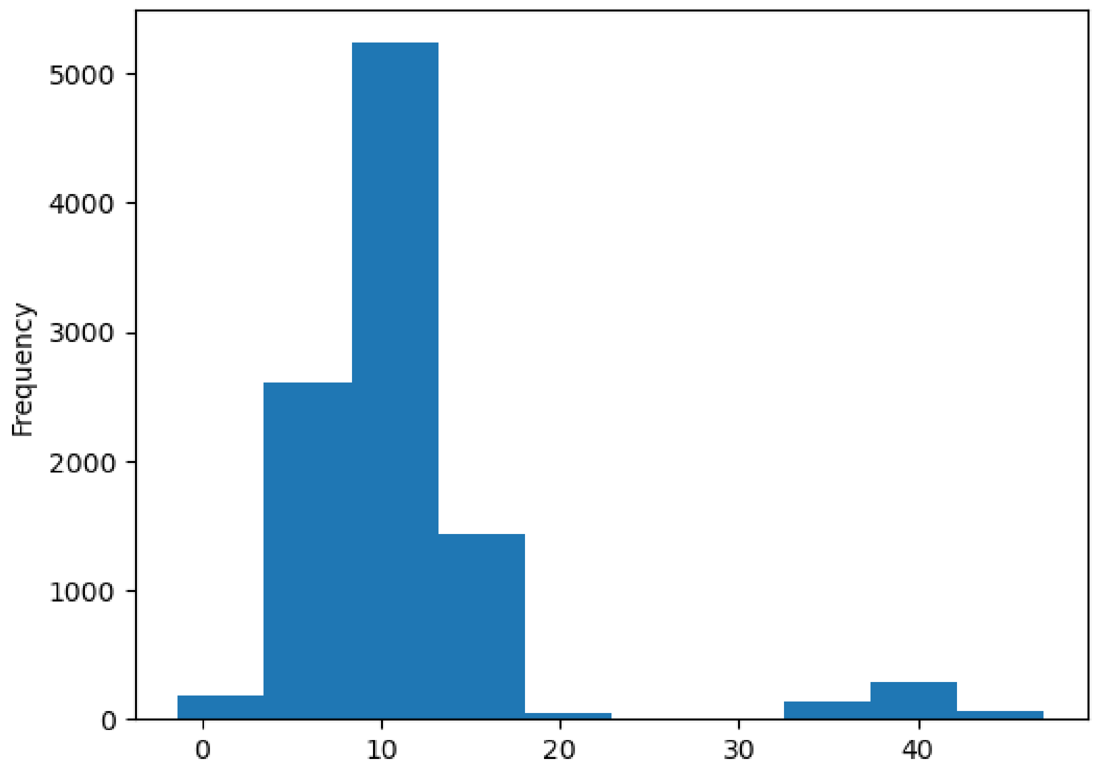
```
Number of nulls: 0
```



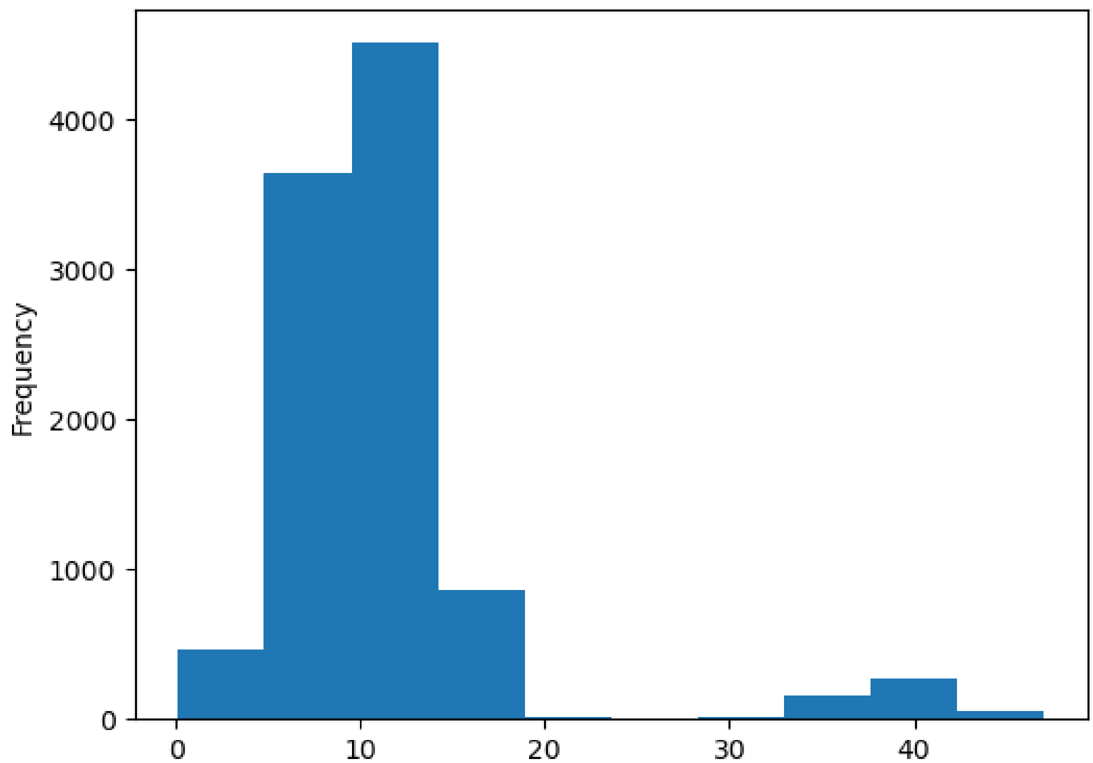
[45]:
```
# Since the 'InternetService' variable has 'None' as one of its options,
# it is necessary to impute 'None'

df['InternetService'].fillna('None', inplace=True)

# Verify that 'None' no longer appears as 'Null'
print('Number of nulls:', df['Tenure'].isnull().sum())
```

```
Number of nulls: 0
```

[46]:
```
# Since the 'Outage_sec_perweek' variable had negative outliers,
# these must be converted to nulls and then treated accordingly.

# Generate a histogram of the 'Outage_sec_perweek' variable
df['Outage_sec_perweek'].plot(kind='hist')

# Display the histogram to observe the distribution of the variable
plt.show()
```



[47]:
```
# Since the 'Outage_sec_perweek' distribution is skewed right,
# impute negative outliers with the median
import numpy as np

# Replace negative outliers with nulls using np.nan
df['Outage_sec_perweek'] = np.where(df['Outage_sec_perweek'] < 0, np.nan,␣
↪df['Outage_sec_perweek'])

# Use fillna() to impute with the median
df['Outage_sec_perweek'].fillna(df['Outage_sec_perweek'].median(), inplace=True)

# Generate a new histogram of the 'Outage_sec_perweek' variable
df['Outage_sec_perweek'].plot(kind='hist')

# Display the histogram to observe the unchanged distribution of the variable
plt.show()
```

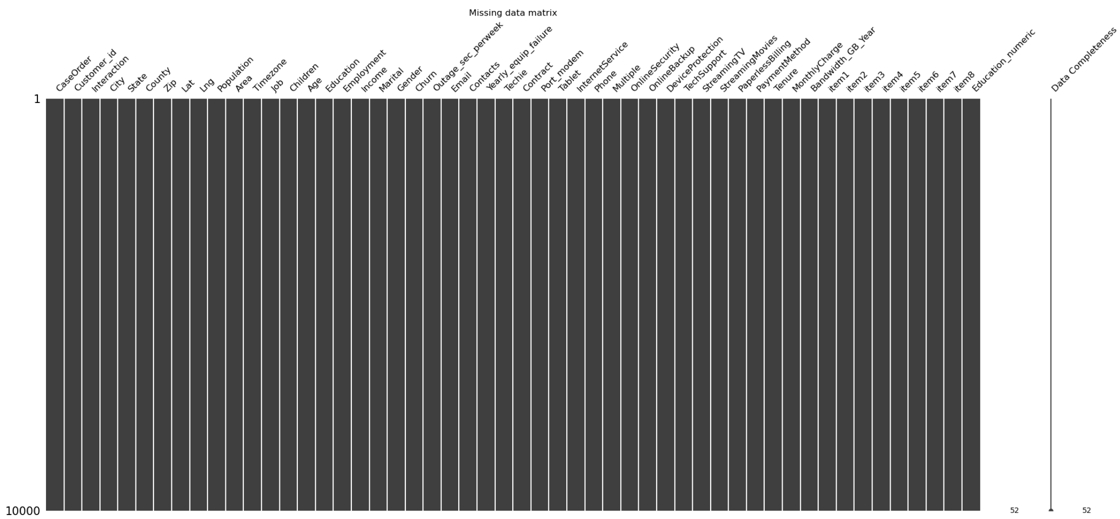


[48]:
```
# Determine all unique values for 'Education' variable
df.Education.unique()

# Make a copy of the variable in the data frame
df['Education_numeric'] = df['Education']

# Create a dictionary assigning numeric values to each categorical value
dict_edu = {"Education_numeric": {"Master's Degree":18, "Regular High School␣
↪Diploma":12,
    "Doctorate Degree":20, "No Schooling Completed":0, "Associate's Degree":14,
    "Bachelor's Degree":16, "Some College, Less than 1 Year":13,
    "GED or Alternative Credential":12, "Some College, 1 or More Years, No␣
↪Degree":14,
    "9th Grade to 12th Grade, No Diploma":11, "Nursery School to 8th Grade":8,␣
↪"Professional School Degree":18}}

# Use the dictionary to replace values of the the variable
df.replace(dict_edu, inplace=True)
```

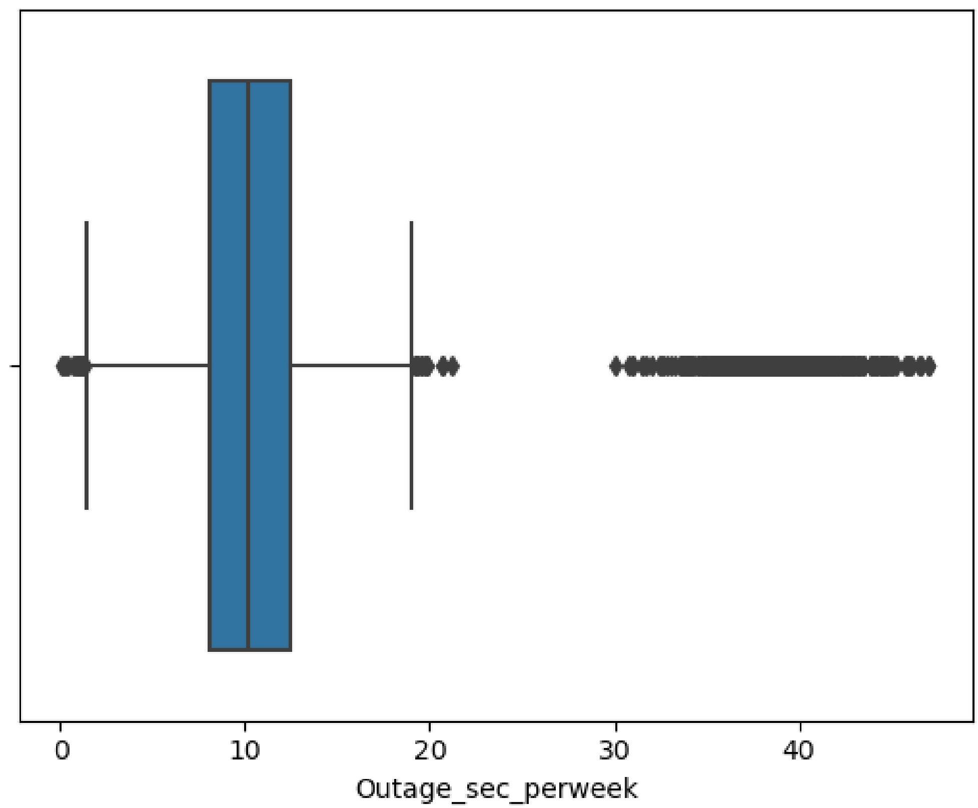
3.4 D3. Visual Evidence Confirming Data is Cleaned

- Using the `matrix()` function from the **missingno** library is used here to generate a matrix of missing values. It is clear that all missing values have been treated.
- By creating a `boxplot()` of the 'Outage_sec_perweek' variable, it is apparent that the negative outliers have been treated.
- By calling the `.unique()` method on the newly created 'Education_numeric' variable, it is clear that the categorical options have been re-mapped to numeric values using ordinal encoding.

[49]:
```python
## D3. Visual Evidence Confirming Data is Cleaned: Missing Values

import missingno as msno
import matplotlib.pyplot as plt

msno.matrix(df, fontsize = 12, labels=True)
plt.title('Missing data matrix')
plt.show()
```



[50]:
```python
## D3. Visual Evidence Confirming Data is Cleaned: Unreasonable Outliers

# Boxplot to visualize 'Outage_sec_perweek' outliers
boxplot = sb.boxplot(x = 'Outage_sec_perweek', data=df)
```



[51]:
```
## D3. Visual Evidence Confirming Data is Cleaned: Re-Expression of Categorical␣
↪Variables

ed_cats = df.Education_numeric.unique()
print('The numerically coded levels of education are:', ed_cats)
```

```
The numerically coded levels of education are: [18 12 20  0 14 16 13 11  8]
```

3.4.1 D5. Copy of Cleaned Data File

[52]:
```
# Code used to export .csv of cleaned data

df.to_csv('D206_PA_MendezD_cleaned.csv', sep=',', encoding='utf-8', index=False)
```

3.4.2 D6. Summarize the Limitations of the Data-Cleaning Process

Since the imputed values may not accurately represent the true values, data imputation, like used in this data-cleaning process, can introduce uncertainty into the data set. The choice of statistical measure used for imputation can also impact the analysis. The retention of outliers could potentially distort summary statistics like the mean and standard deviation. These retained outliers may also skew the interpretation of relationships within the data.

3.4.3 D7. How the Above Limitations May Affect the Analysis

Retention of outliers and imputed values could potentially mislead the decision-making process, as well as misinform the insights provided by the model. The outliers may also obscure the interpretation of results and make it difficult it draw meaningful conclusions from the analysis.

3.5 E. Applying Principal Component Analysis

3.5.1 E1. Total Number of Principal Components and the Output of the Principal Components Loading Matrix

The variables used to perform PCA were the eight continuous variables from the data set:

- Lat
- Lng
- Age
- Income
- Outage_sec_perweek
- Tenure
- MonthlyCharge
- Bandwidth_GB_Year

[53]:
```
# code from K. Middleton's PCA Webinar


# Create data frame with the continuous variables to be used for PCA
pca_frame = df[['Lat', 'Lng', 'Age', 'Income', 'Outage_sec_perweek', 'Tenure',␣
↪'MonthlyCharge', 'Bandwidth_GB_Year']]

# Apply z-score normalization for each column of the data frame
pca_frame_norm = (pca_frame - pca_frame.mean()) / pca_frame.std()
```

[54]:
```
# code from K. Middleton's PCA Webinar


## Apply PCA
from sklearn.decomposition import PCA

# call the PCA function, shape the data based upon the number of PCs created
pca = PCA(n_components = pca_frame.shape[1])

# fit the PCA on the normalized data set
pca.fit(pca_frame_norm)
```

[54]:
```
PCA(n_components=8)
```

[55]:
```
# code from K. Middleton's PCA Webinar
```

```
# Transform into data frame
df_pca = pd.DataFrame(pca.transform(pca_frame_norm), columns=['PC1', 'PC2',␣
 ↪'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8'])

loadings = pd.DataFrame(pca.components_.T, columns=['PC1', 'PC2', 'PC3', 'PC4',␣
 ↪'PC5', 'PC6', 'PC7', 'PC8'], index=pca_frame.columns)
loadings
```

[55]:

```
                         PC1       PC2       PC3       PC4       PC5  \
Lat                -0.022641  0.212141  0.669318  0.041926 -0.014681
Lng                 0.008450 -0.138408 -0.687924 -0.049746  0.126937
Age                -0.012315 -0.057228 -0.039973  0.988936  0.058813
Income              0.005639  0.022758  0.098045 -0.046491  0.988954
Outage_sec_perweek  0.021446  0.692868 -0.133030 -0.048226  0.033216
Tenure              0.705031 -0.051911  0.030280  0.018309 -0.001266
MonthlyCharge       0.045211  0.670271 -0.220657  0.113709 -0.032179
Bandwidth_GB_Year   0.706866 -0.004845  0.016132 -0.004533 -0.006539

                         PC6       PC7       PC8
Lat                -0.709891  0.024054  0.000840
Lng                -0.692522  0.096739  0.000591
Age                 0.006681  0.114006  0.021900
Income              0.073664 -0.064958  0.000821
Outage_sec_perweek  0.101081  0.698664  0.000425
Tenure             -0.007873  0.038496 -0.705293
MonthlyCharge      -0.025413 -0.695028 -0.048254
Bandwidth_GB_Year  -0.008519 -0.013021  0.706931
```

3.5.2 E2. Justification for Reduced Number of Principal Components with Scree Plot

According to the Kaiser Rule, we retain PCs with eigenvalues greater than or equal to 1. (Middleton, K.)

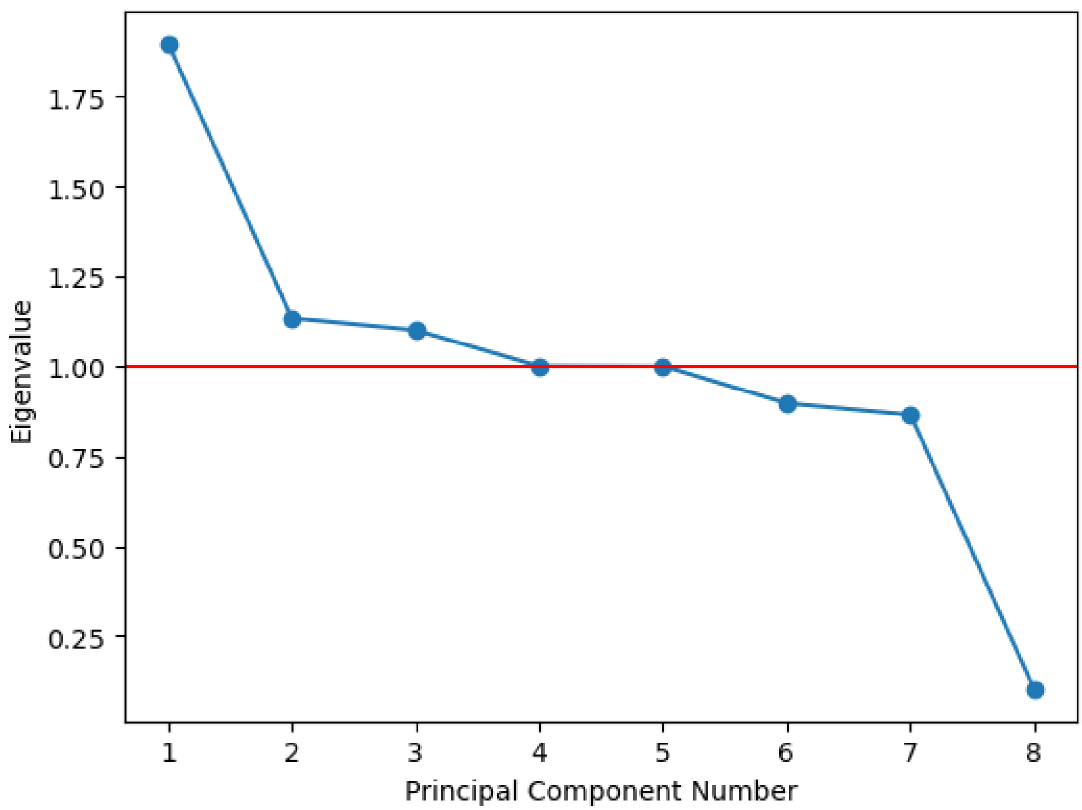
The scree plot below shows that five PCs have eigenvalues greater than or equal to 1:

- PC1
- PC2
- PC3
- PC4
- PC5

[56]:

```
# code from K. Middleton's PCA Webinar


# Calculate covariance and vectors then define eigenvalues before creating␣
 ↪scree plot

```

```
# Create a covariance matrix
cov_matrix = np.dot(pca_frame_norm.T, pca_frame_norm) / pca_frame.shape[0]

# Create and store the eigenvectors in a new data frame
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for␣
 ↪eigenvector in pca.components_]

# Plot eigenvalues
plt.plot(np.arange(1, len(eigenvalues)+1), eigenvalues, marker = 'o')
plt.xlabel('Principal Component Number')
plt.ylabel('Eigenvalue')
plt.axhline(y=1, color="red")
plt.show()
```



3.5.3 E3. Benefits to the Organization from PCA

Principal Component Analysis can help to reduce the dimensionality of the data set by transforming the original continuous variables into a smaller set of variables called principal components. This can simplify the analysis and visualization of data, making it easier for the organization to interpret and understand the findings (Bigabid, 2023).

4 Part IV. Supporting Documents

4.0.1 F. Panopto Video Demonstrating the Code Functionality

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c71d2f83-8164-44fd-8ab9-b14f012326f1

4.0.2 G. Acknowledgement of Web Sources

Middleton, K. *D206 - Getting Started with D206 | PCA* [Webinar]. Western Governors University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=3bcc452f-fa35-43be-b69f-b05901356f95

4.0.3 H. Acknowledgement of Sources

Middleton, K. *D206 - Getting Started with D206 | Missing Values* [Webinar]. Western Governors University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=767749d2-ba19-4f94-bec8-b058017b2f5e

Middleton, K. *D206 - Getting Started with D206 | PCA* [Webinar]. Western Governors University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=3bcc452f-fa35-43be-b69f-b05901356f95

What is Principal Component Analysis and How Can I Use It?. Bigabid. (2023, February 8). https://www.bigabid.com/what-is-pca-and-how-can-i-use-it/