Data Cleaning
Performance Evaluation

D206

TRESA (TESSIE) AUSTIN

WGU

April 23, 2024

Part I: Research Question

A. Research Question

"What medical conditions influence hospital readmission?"

The above research question will be derived from medical data files, medical_raw_data.csv. This data has been complied to look at hospital readmission numbers and ways to prevent (when appropriate) which can cause an increase in costs to facilities if not addressed, as well as disruption to patients and family's' lives (Centers for Medicare & Medicaid Services, n.d.).

B. Description of Columns

The data set being used contains 10,000 records regarding patients who have been readmitted to the hospital within a month of their initial admission. It not only holds patient demographics, but also contains information about patient medical conditions, information about the initial admission and survey information around patient feelings about initial hospital stay.

Below is a table listing all columns in the data set (medical_raw_data.csv). Each column has been listed with the type, description, and an example from the data set.

Column Name	**Type**	**Description**	**Examples**
Index	Int64	Index of data	1
CaseOrder	Int64	Placeholder column to preserve order of raw data file	1
Customer_id	object	Unique Identifier for patient	C123456
Interaction	object	Identifier unique to patient transactions	8cd49b13-f45a-4b47-a2bd-173ffa932c2f
UID	object	Identifier unique to patient transactions	3a83ddb66e2ae73798bdf1d705dc0932
City	object	Patient City of residence	Detroit
State	object	Patient State of residence	MI
County	object	Patient County of residence	Kent
Zip	Int64	Patient Zip of residence	11111
Lat	Float64	Patient latitude of residence	11.1111
Lng	Float64	Patient longitude of residence	-11.11111
Population	Int64	Population within a mile radius of patient residence, based on census data	1000
Area	object	Residence type (rural, urban, suburban) based on census data	Urban
TimeZone	object	Time zone of patient's residence	America/Detroit
Job	object	Job of patient (or primary insurance holder)	Contractor
Children	Float64	Number of children in patient household	2
Age	Float64	Age of patient	54
Income	Float64	Annual income of patient	42000.00
Marital	object	Marital status of patient (or primary insurance holder)	Never Married
Gender	object	Patient gender (male, female, nonbinary)	Nonbinary

ReAdmis	object	Whether patient was readmitted within a month of release	No
VitD_levels	Float64	Patient Vitamin D levels as measured in ng/mL	18.111111
Doc_visits	Int64	Number of times the patient was visited by primary physician during initial hospitalization	4
Full_meals_eaten	Int64	Number of meals eaten while hospitalized (partial meals count as 0, some patients had more than 3 meals a day if requested)	2
VitD_supp	Int64	Number of time patient received Vitamin D supplements	1
Soft_drink	object	Whether or not patient habitually drinks three or more sodas a day	Yes
Initial_admin	object	How the patient was initially admitted to hospital(emergency, elective, observation)	Emergency Admission
HighBlood	object	Whether or not patient has high blood pressure	Yes
Stroke	object	Whether or not patient has had a stroke	No
Complication_risk	object	Level of complication risk as assessed by primary patient assessment (high, med, low)	Medium
Overweight	Float64	Whether or not the patient is overweight based on age, gender, and height	Yes
Arthritis	object	Whether or not patient has arthritis	Yes
Diabetes	object	Whether or not patient has diabetes	No
Hyperlipidemia	object	Whether or not patient has hyperlipidemia	Yes
BackPain	object	Whether or not patient has chronic back pain	Yes
Anxiety	Float64	Whether or not patient has anxiety disorder	No
Allergic_rhinitis	object	Whether or not patient has allergies	Yes
Reflux_esophagitis	object	Whether or not patient has acid reflux	Yes
Asthma	object	Whether or not patient has asthma	No
Services	object	Primary service patient received (blood work, intravenous, CT scan, MRI)	Blood Work
Initial_days	Float64	Number of days patient stayed in hospital on initial visit	4.8988889
TotalCharge	Float64	Amount charged to patient daily	2434.234222
Additional_charges	Float64	Average amount charged to patient for miscellaneous procedures, treatments, meds, anesthesiology, etc.	17505.19246
Items 1- 8		Based on customer survey, each question was asked, and patients rated each on a scale of 1 to 8 (1 = most important, 8 = least important)	
Item1	Int64	Timely admission	2
Item2	Int64	Timely treatment	3

Item3	Int64	Timely visits	5
Item4	Int64	Reliability	4
Item5	Int64	Options	1
Item6	Int64	Hours of treatment	2
Item7	Int64	Courteous staff	6
Item8	Int64	Evidence of active listening from doctor	4

From Python, a check data types and number of values, as well as overall size of dataframe using `PatientReadmit.info()` was ran. Below are the results.

```
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 1 to 10000
Data columns (total 52 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   UID                 10000 non-null  object
 4   City                10000 non-null  object
 5   State               10000 non-null  object
 6   County              10000 non-null  object
 7   Zip                 10000 non-null  int64
 8   Lat                 10000 non-null  float64
 9   Lng                 10000 non-null  float64
 10  Population          10000 non-null  int64
 11  Area                10000 non-null  object
 12  Timezone            10000 non-null  object
 13  Job                 10000 non-null  object
 14  Children            7412 non-null   float64
 15  Age                 7586 non-null   float64
 16  Education           10000 non-null  object
 17  Employment          10000 non-null  object
 18  Income              7536 non-null   float64
 19  Marital             10000 non-null  object
 20  Gender              10000 non-null  object
 21  ReAdmis             10000 non-null  object
 22  VitD_levels         10000 non-null  float64
 23  Doc_visits          10000 non-null  int64
 24  Full_meals_eaten    10000 non-null  int64
 25  VitD_supp           10000 non-null  int64
 26  Soft_drink          7533 non-null   object
 27  Initial_admin       10000 non-null  object
 28  HighBlood           10000 non-null  object
 29  Stroke              10000 non-null  object
 30  Complication_risk   10000 non-null  object
 31  Overweight          9018 non-null   float64
 32  Arthritis           10000 non-null  object
 33  Diabetes            10000 non-null  object
 34  Hyperlipidemia      10000 non-null  object
 35  BackPain            10000 non-null  object
 36  Anxiety             9016 non-null   float64
 37  Allergic_rhinitis   10000 non-null  object
 38  Reflux_esophagitis  10000 non-null  object
 39  Asthma              10000 non-null  object
 40  Services            10000 non-null  object
 41  Initial_days        8944 non-null   float64
 42  TotalCharge         10000 non-null  float64
 43  Additional_charges  10000 non-null  float64
 44  Item1               10000 non-null  int64
 45  Item2               10000 non-null  int64
 46  Item3               10000 non-null  int64
 47  Item4               10000 non-null  int64
 48  Item5               10000 non-null  int64
 49  Item6               10000 non-null  int64
 50  Item7               10000 non-null  int64
 51  Item8               10000 non-null   int64
dtypes: float64(11), int64(14), object(27)
```

From this we can see that there are 11 columns of float64 type, 14 columns of int64 type, and 27 columns of object type. The 27 columns of object type will need to be converted to use the data in a meaningful way.

Part II: Data-Cleaning Plan

C. **Explanations for data cleaning plan**
   1. **Approach to data cleaning plan.**

The approach for cleaning medical_raw_data.csv will consist of the following and will use Python via Jupyter Lab where applicable.

Several steps will be needed to clean the data, starting with looking at the data and assessing the columns needed to answer the research question and how to fold into the entire data set.

Step 1: Describe and inspect the data to determine what you are working with. First, Take into consideration the full nature of the dataset. There are 52 columns of data that contain 10,000 records that range from identifiers for a patient by number to survey questions about a range of factors of a patient's overall experience. Next assess information around the data dictionary. This provides insight into if/how the data is sectioned off. These sections of data will help drive how the data is used based on the research question. Lastly in the first step, an inspection of the data using the set_option to display max columns can be used to show each column. With this step, the current picture of the raw data starts to come into focus.

Step 2: Check for duplicates in the data. To check for duplicates in the data, an isnull() with sum() check will be used. This allows for any duplicates in the data to be found so they can be addressed to ensure the data is able to effectively be used to answer the research question.

Step 3: Check for missing values in the data. Find if there are any missing values in the data. Figure out if there is any missing data, which might skew any of the results of data analysis. If the code used signifies there is missing data, next we will need to figure out what to do about the missing values.

Step 4: Determine measures of central tendency by first using describe() to get summary statistics of the data will be helpful when seeing what standard correlations need to be made and what the minimum and maximum values are of each column in table form. It is important to consider the scale and range of values in each column while interpreting the data.

Then, to visualize the mean, median and mode of each column, create a charts of each column as series objects. Seaborn works well with matplotlib and is compatible with pandas structure. According to Seaborn tutorial (n.d.) it's dataset aligned coding allows for better plotting on charts.

Step 5: Correct missing values after deciding the best central tendency measure to use. Using the information they describe() function, we can see what each columns count, mean, standard deviation, minimum, maximum, and quartile values are in table form. We can also use the histograms created to determine how the data is distributed as well as being used as a check for measures of central tendency.

Step 6: Find outliers in the data and assess if these outliers represent natural variations in the dataset or not. If not due to natural variations, then outliers are problematic and can skew the mean and/or impact the measures of centra tendency ((Geeks for Geeks, 2024).

Before being able to check for outliers, we first need to change some of the datatypes to numeric values. Those identified columns are HighBlood , Stroke , Complication_risk , Arthritis , Diabetes , Hyperlipidemia, BackPain , Allergic_rhinitis , Reflux_esophagitis , Asthma.
Using code to convert needed columns to Boolean types allows us to then address outliers and missing values. According to Zadka (2020), Booleans are a numeric data type in Python, so converting the above columns to 'bool' will help address future research needs. The above listed columns will be converted using Pandas package, however, the column Soft_drink must be changed to a numeric type value for the same above reasons, but Pandas is unable to accommodate that change. Sklern's LabelEncoder will be used to transmute Soft_drink values to numeric values. According to GeeksforGeeks (Geeks for Geeks, 2023) using the sklern LabelEncoder, helps preserve the meaning of the elements values and is used to convert categorical columns to numeric columns.

Now that the data types have been addressed, the actual check for outliers can be done. Creating box plots or box whisker plots shows us visually which columns have outliers and what these outliers are. To create box plots, a combination of Pandas, Matplotlib, and Seaborn can be used in the coding. This combination allows us to select the numerical columns by data type via Pandas, create then display graphs as box plots via Matplotlib, and have custom outlier markings via Seaborn.

This data cleaning plan ends with generating a cleaned data set of PatientReadmit_Clean as a csv file.

2. Justification for data cleaning plan

Columns in the data set pertain not only to the patient's admission to the hospital but also to the patients' demographics. As this is a medical dataset, there can be additional complexity with the values in the columns. Some of these demographic columns are insignificant to the research question. Looking at patterns in each column can provide some information but it is not relevant to answering the research question.

A check for duplicate values revealed no columns with duplicates, this ensured that the data aligned with needs for the research question.

Missing values were found in a few of the columns, this was the area where other means to determine the best path were employed. Determining the mean, median and mode of each column above during the data assessment plan provides needed information to replace missing values. These missing values may be due to patients not feeling comfortable providing the piece of information, the information being unavailable to the person giving information upon the patients admission to the hospital, or the columns value depends on several other columns values and one of those is missing.

Several methods were needed to ensure imputation of missing values is aligned with research needs. Typically measure of central tendency values can be found using histograms (Solomon, 2018). A histogram for each column type yielded useful results to back up what was gathered from other coding. Other graphs to show just each columns mean, median and mode were generated.   As well as using the describe() function from Pandas to create a table of mean, standard deviation, minimum, maximum and quartiles for each column. All of this allowed the visualization from the histogram to be checked and balanced with data from this table and other graphs.

Outliers posed another hurdle for columns in the data set. The first step in ensuring we can evaluate the data objectively is to ensure that needed columns are transformed to numeric type values so that further investigation can be done around how to address treatment of outliers as well as imputing missing values.

Using box plots to "see" outliers in needed data fields helps address the question of which columns have outliers. According to Newcastle University (n.d.), the body of the box shows three lines, the initial one describes the lower interquartile value, the middle one shows the median, and the last one shows the upper quartile value. The graph is sometimes called a 'box and whisker' diagram, where the 'whiskers' are the straight lines that come from the end of the box to the max and min values, excluding outliers.

3. Justification for choice of tools

The main language chosen to use for this task is Python. This choice was made based on two main factors: One is that the library available makes string options quicker and easier to use (Sisense Team, 2018). Pandas library, associated with Python, is one of the most powerful libraries you can use when working with structured data and finding missing values, removing duplicates and transforming data are just a few of the options that are easily done (divine_inner_voice, 2023). The web-based platform of Jupyter Notebook was used to create the code for the task. Using Jupyter Notebook via a web browser allows for use on different computers and ability to download the code(Jupyter Notebook, n.d).

Coding for this task primarily utilizes Pandas and NumPy for data manipulation as well as Seaborn and Matplotlib for visualization. For the PCA section, PCA was used from Sklearn. Information on how and why each of these packages were chosen is listed below:

- Pandas is by far the most widely used and flexible of all Python packages. Focusing on real world data analysis per Python(2024) as stated in the package overview, Pandas uses series and dataframes which oversee most use cases of analysis. The list of functions that Pandas has is longer than this paper by many times over. A data analyst would be reinventing the wheel if they tried to clean data without Pandas.
- From NumPy(2022) the numpy package is one of the fundamental Python packages used in computing scientific and mathematical data. One of the biggest features is the ndarray object which allows for varying sizes of arrays to be worked during compiling. Another feature is that the data elements associated are all required to be the same data type and size, not including arrays. For its standardized data types and array functions, numpy was a desirable choice this data set.
- Matplotlib is another library for creating visualizations of statistical data, integrates well with Seaborn, and can combine with Pandas coding. Again, it's primary purpose is to create visually useful representations of data so it's easier to understand and analyze (Geeks for Geeks, 2023). It was chosen for use in creating histograms as well since it integrates easily with the other packages used.
- Seaborn is a library for creating statistical graphics, adds on top of matplotlib and combines well with Pandas structure. Creating histograms to help determine mean and median are crucial functions for this task and Seaborn (sns) code is used to create graphs for those two measures of central tendency. Features such as ability to choose the interval when categorizing your data via "bin" allow you to easier visualize and interpret the graphs (Seaborn, n.d.)

- Scikit-Learn’s decomposition component for Principal Component Analysis is helpful when taking a dataset and explaining variance. The fit() is a transformer object that allows the machine learning to do the heavy lifting of calculations (Pedregosa, 2011). For PCA components, it’s a must use.

The use of all these tools together in Python helps makes the data easier to visualize, makes the data aligned so it can be compared and makes the data valuable to the company.

4. **Annotated code**

See code attached, file name TAustin_PatientReadmit_Code_2.ipnyb

Part III: Data Cleaning

D. **Summarize data cleaning plan**

Based on the above work, the medical_raw_data has been cleaned and the data frame named PatientReadmit for ease of use. This data has been assessed for missing values, duplicate values, misleading column headers and outliers as listed above in Part II. Findings during the data cleaning are outlined below.

1. **Findings around data quality issues:**

The data quality is average. Medical data overall can have many issues based on how it’s collected for starters. The medical_raw_data file provided for this research starts off with the first column being unnamed which had to be addressed.

- Duplicates:
    - There are no duplicates in the columns.
- Missing values:
    - There are seven of the fifty-two columns with missing data. Below the columns and number of missing values out of 10,000 are listed:

Children	2588
Age	2414
Income	2464
Soft_drink	2467
Overweight	982
Anxiety	984
Initial_days	1056

- Data types:
    - Twenty-seven of the fifty-two columns also are data type ‘object’ which is not useful in many of the functions needed to address missing data and outliers. Those twenty-seven must be converted to a data type that allows for Python packages to be used. To address the missing values, measures of central tendency need to be gathered from the data.
- Outliers:

- Nineteen of the fifty-two columns have outliers based on the box plots and the columns are listed below:
    - Lat – outliers expected
    - Lng– outliers expected
    - Population – outliers expected
    - Children – outliers expected
    - Income – outliers expected
    - VitD_levels (mean 19.412675, median 18.08, mode 9.52)
        - Expected outliers as some patients may suffer from issues with high or low Vitamin D and others may not have issues with Vitamin D at all.
    - Full_meals_eaten (mean 1.001400, median 1, mode 0)
        - Expected outliers as answer could be dependent on time of day admitted/released and/or issue patient was in for
    - VitD_supp (mean 0.398900, median 0.0, mode 0)
        - Expected outliers as some patients could be more deficient in Vitamin D and need more supplements
    - TotalCharge (mean 5891.538261, median 5852.25, mode 1256.75)
        - Expected outliers as total charges by day is highly dependent on what the patient is hospitalized for.
    - Additional_charges (mean 12934.528586, median 11573.98, mode 8013.79)
        - Expected outliers again as the average amount of additional charges is highly dependent on what the patient is hospitalized for.
    - Stroke (mean 0.1993, median 0, mode 0)

      Patents can either have had a stroke (1/Yes/True) or not (0/No/False), so this is not an outlier.

  Items 1-8 are survey questions based on each patients feelings on a scale of 1 (most important) to 8 (least important). Patients value distinct factors in their hospitalization and while this information can be useful to the hospital these values are based on a person's perception of importance so outliers would be expected.

    - Item1 (mean 3.518800, median 4, mode 4)
    - Item2 (mean 3.506700, median 3, mode 3)
    - Item3 (mean 3.511100, median 4, mode 4)
    - Item4 (mean 3.515100, median 4, mode 3)
    - Item5 (mean 3.496900, median 3, mode 4)
    - Item6 (mean 3.522500, median 4, mode 4)
    - Item7 (mean 3.494000, median 3, mode 4)
    - Item8 (mean 3.509700, median 3, mode 3)

- Based on the above list of outliers and information, there are no outliers that need to be treated for this research.

2. Justify methods of mitigation

To mitigate issues within the data, the following occurred:

- Import Data

After reading the file into Python, getting information about number of missing values in each variable was pulled via `PatientReadmit.info().` There are 25 variable that are numerical type and 27 that are not. The 27 that are not of numeric type will be addressed later in the summary.

- Duplicates

Checking for duplicates was a short step, but an important one. Code was used to check and determine there are no duplicates.

- Missing Data

Figuring out missing data was next on the list. Missing data in your dataset that isn't addressed can undermine any conclusion you reach on your research question, finding and replacing (if needed) is imperative in the data cleaning process. Also calculating the PCA requires that there be no missing or null values. There are seven variables that have missing data. Determining which measure of central tendency will be used is imperative. After that missing values were replaced based on data type with mean or median using `fillna` after identification.

Variable Missing
Children 2588
Description: Number of children in the patient's household as reported in the admissions information (#)
Age 2414
Description: Age of the patient as reported in admissions information(#)
Income 2464
Description: Annual income of the patient (or primary insurance holder) as reported at time of admission(#)
Soft_drink 2467
Description: Whether the patient habitually drinks three or more sodas in a day (yes, no)
Overweight 982
Description: Whether the patient is considered overweight based on age, gender, and height (yes, no)
Anxiety 984
Description: Whether the patient has an anxiety disorder (yes, no)
Initial_days 1056
Description: The number of days the patient stayed in the hospital during the initial visit(#)

First, Children, Age, and Income were self-reported by the patient at the time of admission. Patients may not feel comfortable answering these questions, making the missing data in these variables MNR (Missing Not at Random). Addressing missing variables and assessing their usefulness towards the research question requires further examination of quartiles and measures of central tendency.

- Outliers

Finding outliers was done with histograms. This provided a visual way to identify outliers and using the data from the data dictionary around what each column measures and how it was measured gave valuable information on determining if an outlier was expected or not.

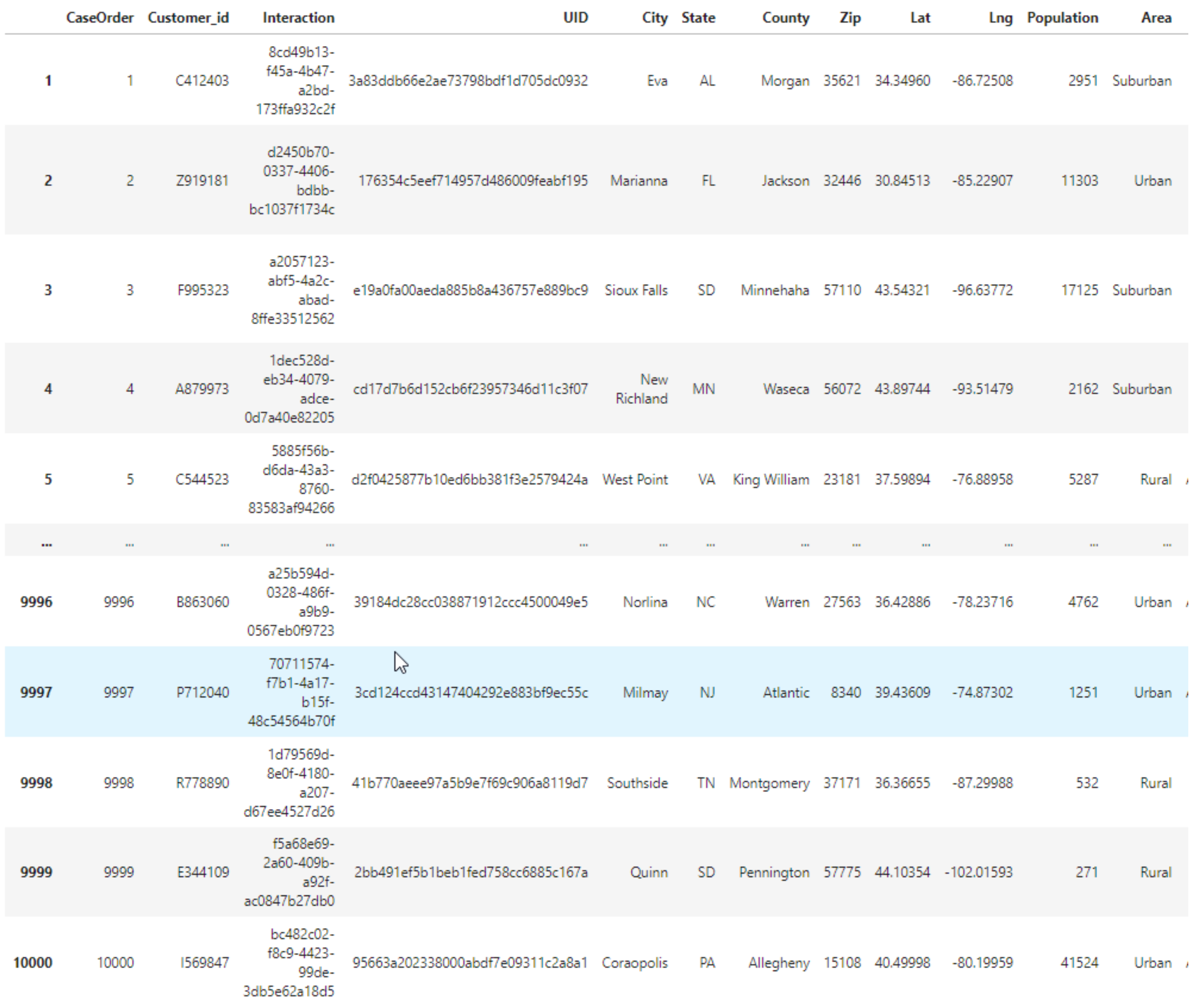
3. Summarize outcome of each step in data cleaning

Importing needed packages in Python started the process of getting all data from the raw format to a readable dataframe.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load raw data and name dataframe
# Data first column is index, solve for potential duplicate column use
'index_col=0'
PatientReadmit = pd.read_csv('./medical_raw_data.csv', index_col=0)
```

The above code accomplished the getting started step. The first step in cleaning the data revolved around inspecting the data using two checks.

```
# Check data types, number of values, & size of dataframe
PatientReadmit.info()
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 1 to 10000
Data columns (total 52 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   UID                 10000 non-null  object 
 4   City                10000 non-null  object 
 5   State               10000 non-null  object 
 6   County              10000 non-null  object 
 7   Zip                 10000 non-null  int64  
 8   Lat                 10000 non-null  float64
 9   Lng                 10000 non-null  float64
 10  Population          10000 non-null  int64  
 11  Area                10000 non-null  object 
 12  Timezone            10000 non-null  object 
 13  Job                 10000 non-null  object 
 14  Children            7412 non-null   float64
 15  Age                 7586 non-null   float64
 16  Education           10000 non-null  object 
 17  Employment          10000 non-null  object 
 18  Income              7536 non-null   float64
 19  Marital             10000 non-null  object 
 20  Gender              10000 non-null  object 
 21  ReAdmis             10000 non-null  object 
 22  VitD_levels         10000 non-null  float64
 23  Doc_visits          10000 non-null  int64  
 24  Full_meals_eaten    10000 non-null  int64  
 25  VitD_supp           10000 non-null  int64  
 26  Soft_drink          7533 non-null   object 
 27  Initial_admin       10000 non-null  object 
 28  HighBlood           10000 non-null  object 
 29  Stroke              10000 non-null  object 
 30  Complication_risk   10000 non-null  object 
 31  Overweight          9018 non-null   float64
 32  Arthritis           10000 non-null  object 
 33  Diabetes            10000 non-null  object 
 34  Hyperlipidemia      10000 non-null  object 
 35  BackPain            10000 non-null  object 
 36  Anxiety             9016 non-null   float64
 37  Allergic_rhinitis   10000 non-null  object 
 38  Reflux_esophagitis  10000 non-null  object 
 39  Asthma              10000 non-null  object 
 40  Services            10000 non-null  object 
 41  Initial_days        8944 non-null   float64
 42  TotalCharge         10000 non-null  float64
 43  Additional_charges  10000 non-null  float64
 44  Item1               10000 non-null  int64  
 45  Item2               10000 non-null  int64  
 46  Item3               10000 non-null  int64  
 47  Item4               10000 non-null  int64  
 48  Item5               10000 non-null  int64  
 49  Item6               10000 non-null  int64  
 50  Item7               10000 non-null  int64  
 51  Item8               10000 non-null  int64  
dtypes: float64(11), int64(14), object(27)
memory usage: 4.0+ MB

# Inspect PatientReadmit data
pd.set_option("display.max_columns", None)
PatientReadmit
```



	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	Population	Area
1	1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL	Morgan	35621	34.34960	-86.72508	2951	Suburban
2	2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL	Jackson	32446	30.84513	-85.22907	11303	Urban
3	3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD	Minnehaha	57110	43.54321	-96.63772	17125	Suburban
4	4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN	Waseca	56072	43.89744	-93.51479	2162	Suburban
5	5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	VA	King William	23181	37.59894	-76.88958	5287	Rural
...	...	...	...	...	...	...	...	...	...	...	...	...
9996	9996	B863060	a25b594d-0328-486f-a9b9-0567eb0f9723	39184dc28cc038871912ccc4500049e5	Norlina	NC	Warren	27563	36.42886	-78.23716	4762	Urban
9997	9997	P712040	70711574-f7b1-4a17-b15f-48c54564b70f	3cd124ccd43147404292e883bf9ec55c	Milmay	NJ	Atlantic	8340	39.43609	-74.87302	1251	Urban
9998	9998	R778890	1d79569d-8e0f-4180-a207-d67ee4527d26	41b770aeee97a5b9e7f69c906a8119d7	Southside	TN	Montgomery	37171	36.36655	-87.29988	532	Rural
9999	9999	E344109	f5a68e69-2a60-409b-a92f-ac0847b27db0	2bb491ef5b1beb1fed758cc6885c167a	Quinn	SD	Pennington	57775	44.10354	-102.01593	271	Rural
10000	10000	I569847	bc482c02-f8c9-4423-99de-3db5e62a18d5	95663a202338000abdf7e09311c2a8a1	Coraopolis	PA	Allegheny	15108	40.49998	-80.19959	41524	Urban

The first allowed information to show in Phyton on if the data expected was there in terms of columns and rows. The second allowed all the columns and the first five/last five rows to be inspected.

Next, we move to checking for duplicates via the below code.

```
# Check for duplicate data
duplicate_data = PatientReadmit.duplicated().any()
print("Duplicates present:", duplicate_data)
Duplicates present: False
```

Missing data was checked and counts for null values were displayed. With this listing we can start formulating a plan for how to deal with missing values.

```
# Check for missing data & display counts
missing_data = PatientReadmit.isnull().sum()
print("Missing data counts:")
print(missing_data)
Missing data counts:
CaseOrder              0
Customer_id            0
Interaction            0
UID                    0
City                   0
State                  0
County                 0
Zip                    0
```

```
Lat                      0
Lng                      0
Population               0
Area                     0
Timezone                 0
Job                      0
Children              2588
Age                   2414
Education                0
Employment               0
Income                2464
Marital                  0
Gender                   0
ReAdmis                  0
VitD_levels              0
Doc_visits               0
Full_meals_eaten         0
VitD_supp                0
Soft_drink            2467
Initial_admin            0
HighBlood                0
Stroke                   0
Complication_risk        0
Overweight             982
Arthritis                0
Diabetes                 0
Hyperlipidemia           0
BackPain                 0
Anxiety                984
Allergic_rhinitis        0
Reflux_esophagitis       0
Asthma                   0
Services                 0
Initial_days          1056
TotalCharge              0
Additional_charges       0
Item1                    0
Item2                    0
Item3                    0
Item4                    0
Item5                    0
Item6                    0
Item7                    0
Item8                    0
dtype: int64
```

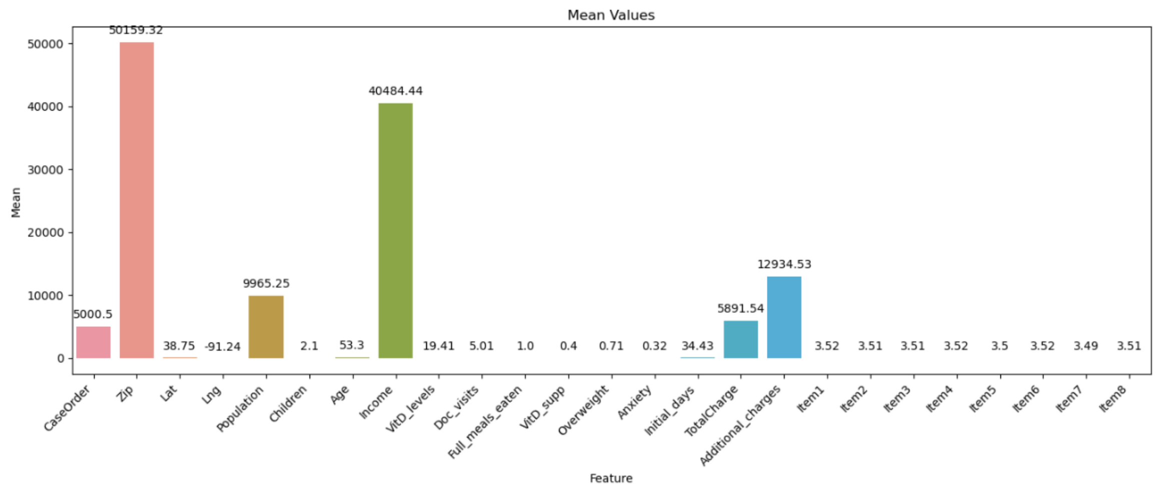
To be able to determine what measure of central tendency will need to be used to replace the missing values above, several pieces of information were gathered starting with a listing of summary statistics around the data for any column not of object type. This summary includes mean, standard deviation, minimum, maximum, and quartile values.

```
CaseOrder           Zip           Lat           Lng     Population  \
count  10000.00000  10000.000000  10000.000000  10000.000000   10000.000000
mean    5000.50000  50159.323900     38.751099    -91.243080    9965.253800
std     2886.89568  27469.588208      5.403085     15.205998   14824.758614
min        1.00000    610.000000     17.967190   -174.209690       0.000000
25%     2500.75000  27592.000000     35.255120    -97.352982     694.750000
50%     5000.50000  50207.000000     39.419355    -88.397230    2769.000000
75%     7500.25000  72411.750000     42.044175    -80.438050   13945.000000
max    10000.00000  99929.000000     70.560990    -65.290170  122814.000000

          Children          Age         Income   VitD_levels    Doc_visits  \
count  7412.000000  7586.000000    7536.000000  10000.000000  10000.000000
mean      2.098219    53.295676   40484.438268     19.412675      5.012200
std       2.155427    20.659182   28664.861050      6.723277      1.045734
min       0.000000    18.000000     154.080000      9.519012      1.000000
25%       0.000000    35.000000   19450.792500     16.513171      4.000000
50%       1.000000    53.000000   33942.280000     18.080560      5.000000
75%       3.000000    71.000000   54075.235000     19.789740      6.000000
max      10.000000    89.000000  207249.130000     53.019124      9.000000
```

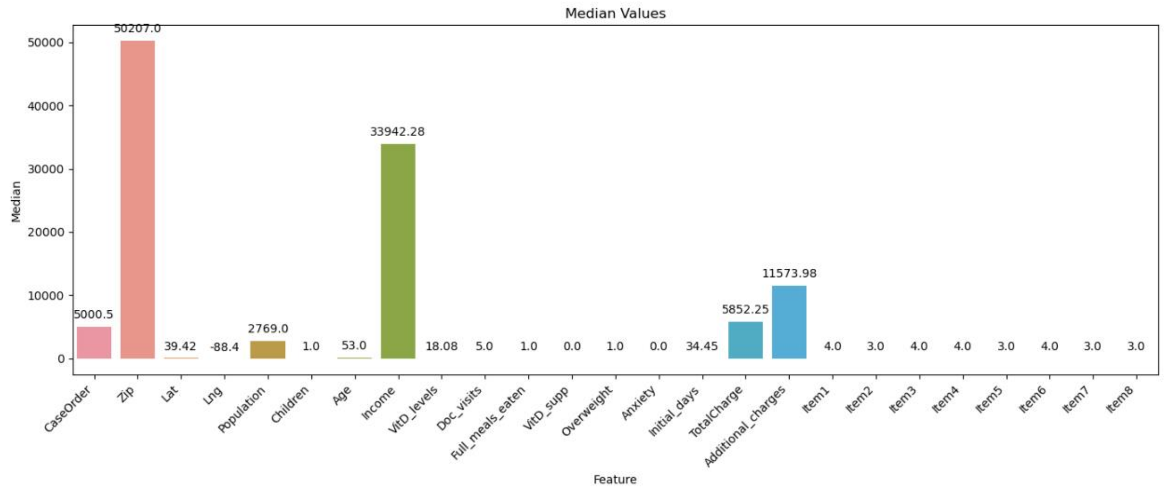
```
       Full_meals_eaten     VitD_supp   Overweight      Anxiety  Initial_days  \
count      10000.000000  10000.000000  9018.000000  9016.000000   8944.000000
mean           1.001400      0.398900     0.709137     0.322316     34.432082
std            1.008117      0.628505     0.454186     0.467389     26.287050
min            0.000000      0.000000     0.000000     0.000000      1.001981
25%            0.000000      0.000000     0.000000     0.000000      7.911709
50%            1.000000      0.000000     1.000000     0.000000     34.446941
75%            2.000000      1.000000     1.000000     1.000000     61.124654
max            7.000000      5.000000     1.000000     1.000000     71.981486

        TotalCharge  Additional_charges         Item1         Item2  \
count  10000.000000        10000.000000  10000.000000  10000.000000
mean    5891.538261        12934.528586      3.518800      3.506700
std     3377.558136         6542.601544      1.031966      1.034825
min     1256.751699         3125.702716      1.000000      1.000000
25%     3253.239465         7986.487642      3.000000      3.000000
50%     5852.250564        11573.979365      4.000000      3.000000
75%     7614.989701        15626.491033      4.000000      4.000000
max    21524.224210        30566.073130      8.000000      7.000000

              Item3         Item4         Item5         Item6         Item7  \
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000
mean       3.511100      3.515100      3.496900      3.522500      3.494000
std        1.032755      1.036282      1.030192      1.032376      1.021405
min        1.000000      1.000000      1.000000      1.000000      1.000000
25%        3.000000      3.000000      3.000000      3.000000      3.000000
50%        4.000000      4.000000      3.000000      4.000000      3.000000
75%        4.000000      4.000000      4.000000      4.000000      4.000000
max        8.000000      7.000000      7.000000      7.000000      7.000000

              Item8
count  10000.000000
mean       3.509700
std        1.042312
min        1.000000
25%        3.000000
50%        3.000000
75%        4.000000
max        7.000000
```

Another piece of information to help determine how to address missing values was gathered using the following:

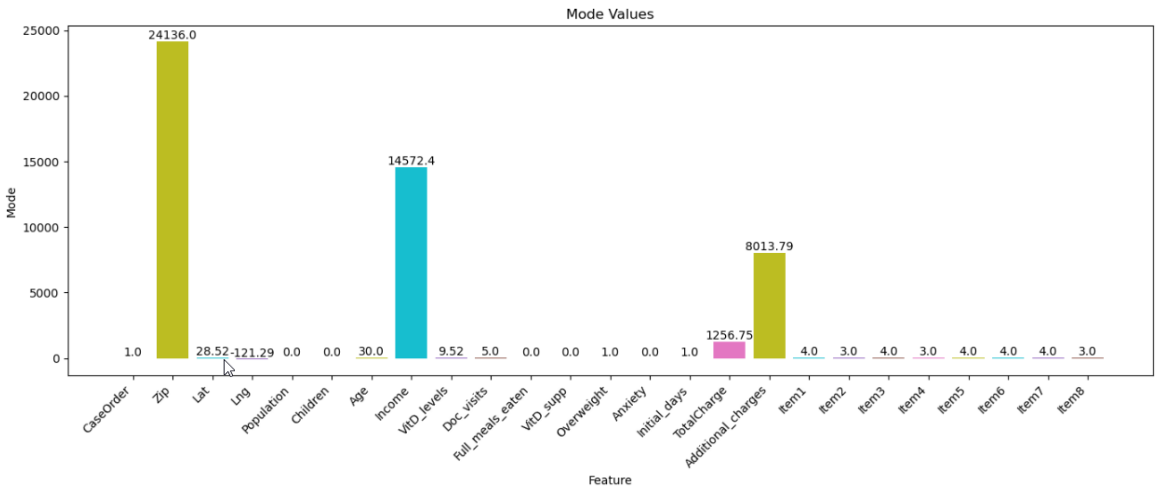
```
# Barplot to find mean
mean_value = PatientReadmit.describe()

plt.figure(figsize=(14, 6))

ax = sns.barplot(x=mean_value.columns, y=mean_value.loc['mean'])
plt.title('Mean Values')
plt.ylabel('Mean')
plt.xlabel('Feature')

for p in ax.patches:
    ax.annotate(str(round(p.get_height(), 2)), (p.get_x() + p.get_width() /
2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset
points')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



```
# Barplot to find median
median_value = PatientReadmit.describe()

plt.figure(figsize=(14, 6))

ax = sns.barplot(x=median_value.columns, y=median_value.loc['50%'])
plt.title('Median Values')
plt.ylabel('Median')
plt.xlabel('Feature')

for p in ax.patches:
    ax.annotate(str(round(p.get_height(), 2)), (p.get_x() + p.get_width() /
2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset
points')

plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



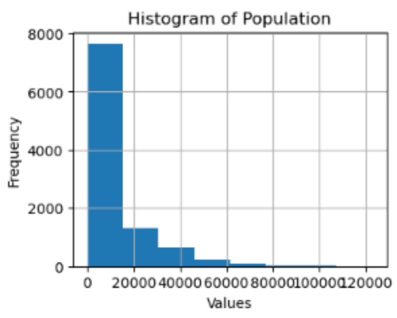
```
# Barplot to find Mode (convert values to numeric)
mode_values = PatientReadmit.mode().iloc[0]
mode_values_numeric = mode_values.apply(pd.to_numeric, errors='coerce')
mode_values_numeric = mode_values_numeric.dropna()
plt.figure(figsize=(14, 6))

# Define a list of colors for each bar
colors = [ '#e377c2','#bcbd22','#17becf','#9467bd','#8c564b']  # Add more
colors as needed
plt.bar(mode_values_numeric.index, mode_values_numeric, color=colors)

plt.title('Mode Values')
plt.ylabel('Mode')
plt.xlabel('Feature')
plt.xticks(rotation=45, ha='right')

for i, v in enumerate(mode_values_numeric):
    plt.text(i, v, str(round(v, 2)), ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

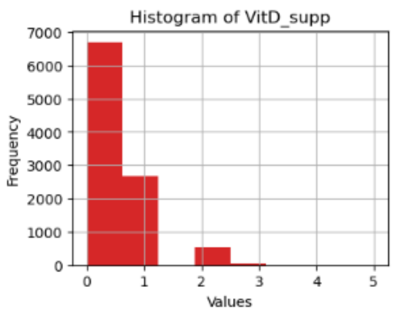


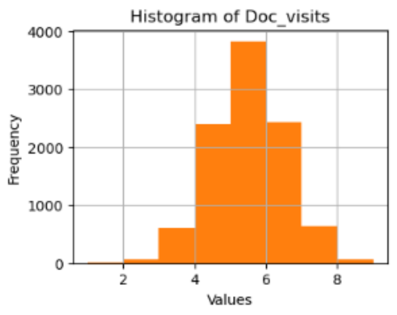
And the last piece to help with determining how to correct missing value is histograms.

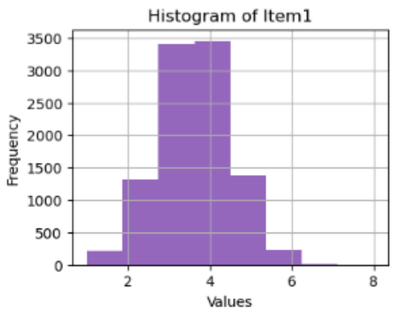
The histograms were created by doing the following:

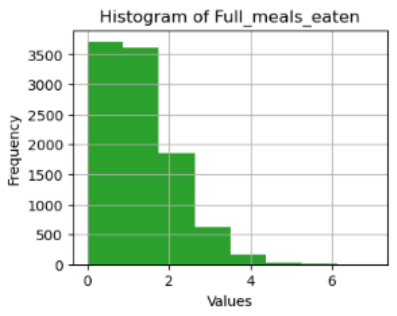
```
#Create histograms for listed columns
columns_to_plot_1 = ['Population', 'Doc_visits', 'Full_meals_eaten',
'VitD_supp', 'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7',
'Item8']

# Define a list of colors for each bar(Blue: #1f77b4,Orange: #ff7f0e,Green:
#2ca02c,Red: #d62728,Purple: #9467bd,Brown: #8c564b,Pink: #e377c2,Gray:
#7f7f7f,Yellow: #bcbd22,Cyan: #17becf)
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd','#8c564b',
'#e377c2', '#7f7f7f', '#bcbd22', '#17becf','#2ca02c','#7f7f7f' ]

for i, column in enumerate(columns_to_plot_1):
    plt.figure(figsize=(4, 3))
    plt.hist(PatientReadmit[column], bins=8, color=colors[i])
    plt.title(f'Histogram of {column}')
    plt.xlabel('Values')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

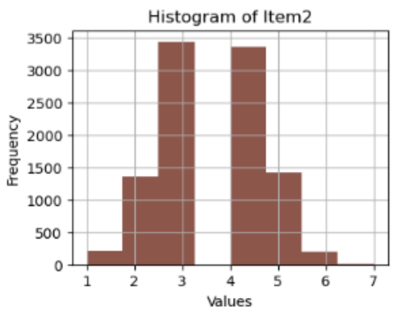

Histogram of Population
Frequency
Values
0
2000
4000
6000
8000
20000
40000
60000
80000
100000
120000


Histogram of VitD_supp
Frequency
Values
0
1000
2000
3000
4000
5000
6000
7000
1
2
3
4
5

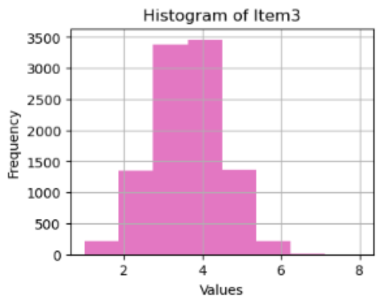

Histogram of Doc_visits
Frequency
Values
0
1000
2000
3000
4000
2
4
6
8

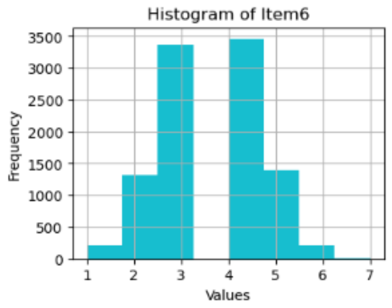

Histogram of Item1
Frequency
Values
0
500
1000
1500
2000
2500
3000
3500
2
4
6
8

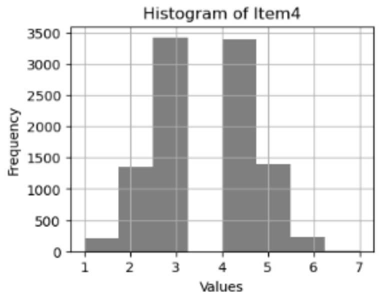

Histogram of Full_meals_eaten
Frequency
Values
0
500
1000
1500
2000
2500
3000
3500
2
4
6

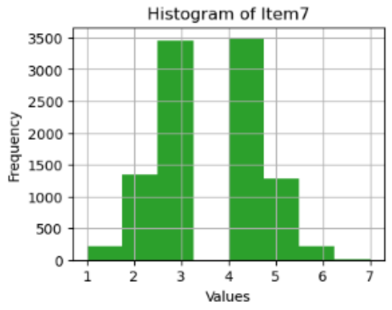

Histogram of Item2
Frequency
Values
0
500
1000
1500
2000
2500
3000
3500
1
2
3
4
5
6
7

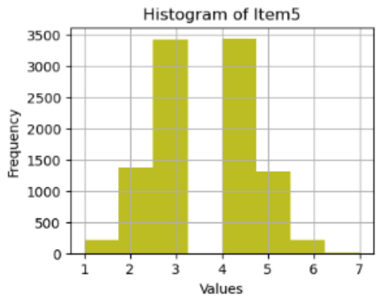
Values

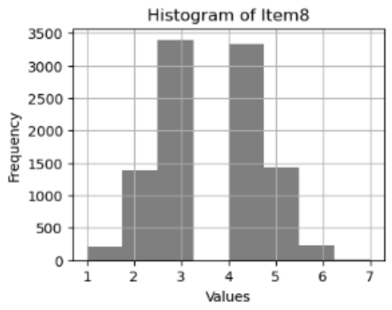




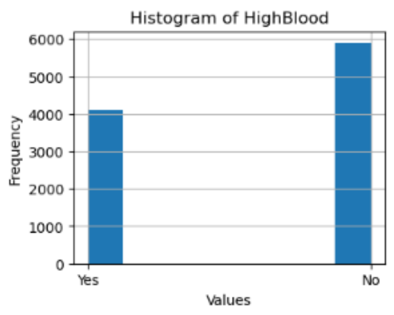


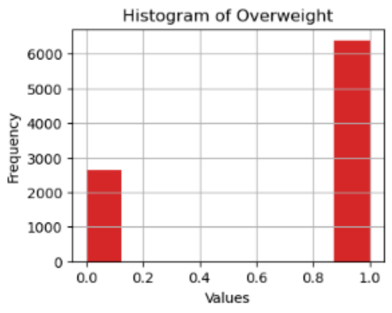


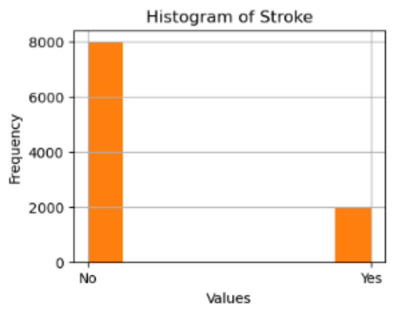


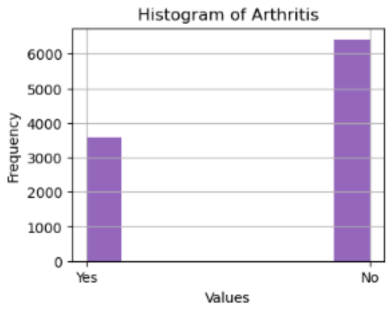


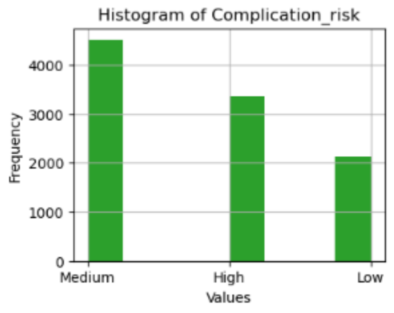
```
# Create histograms for listed columns
columns_to_plot_2 = ['HighBlood', 'Stroke', 'Complication_risk',
'Overweight', 'Arthritis', 'Diabetes',  'Hyperlipidemia', 'BackPain',
'Anxiety', 'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma']

# Define list of colors for bars(Blue: #1f77b4,Orange: #ff7f0e,Green:
#2ca02c,Red: #d62728,Purple: #9467bd,Brown: #8c564b,Pink: #e377c2,Gray:
#7f7f7f,Yellow: #bcbd22,Cyan: #17becf)
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
'#e377c2', '#7f7f7f', '#bcbd22', '#17becf','#2ca02c','#7f7f7f']

for i, column in enumerate(columns_to_plot_2):
    plt.figure(figsize=(4, 3))
    plt.hist(PatientReadmit[column], bins=8, color=colors[i])
    plt.title(f'Histogram of {column}')
    plt.xlabel('Values')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

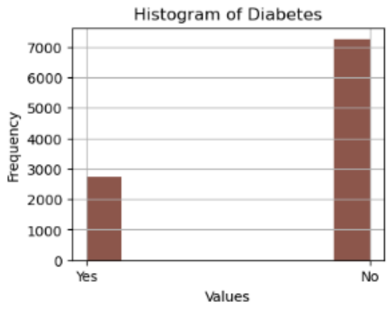

Histogram of HighBlood
Frequency
Values
Yes
No
0
1000
2000
3000
4000
5000
6000

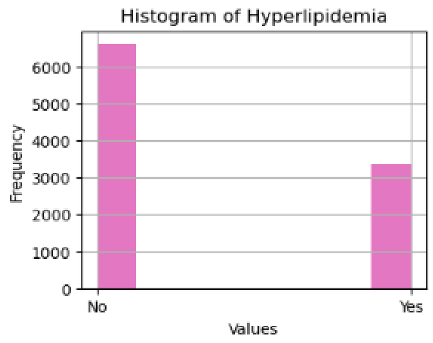

Histogram of Overweight
Frequency
Values
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000
6000

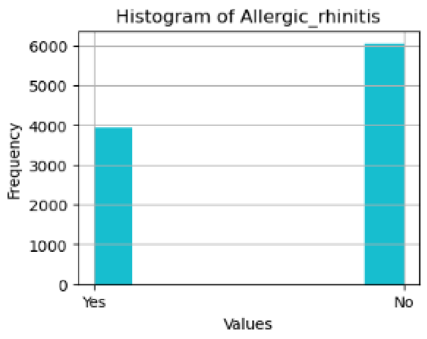

Histogram of Stroke
Frequency
Values
No
Yes
0
2000
4000
6000
8000

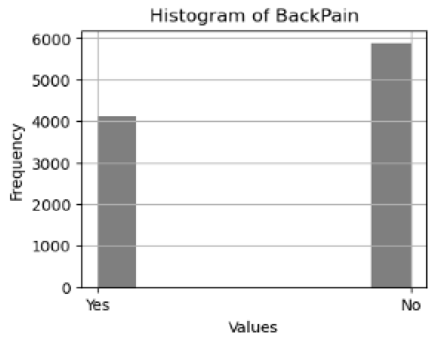

Histogram of Arthritis
Frequency
Values
Yes
No
0
1000
2000
3000
4000
5000
6000

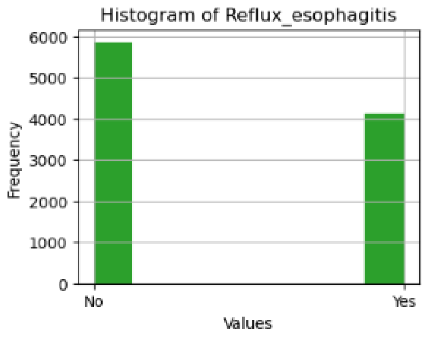

Histogram of Complication_risk
Frequency
Values
Medium
High
Low
0
1000
2000
3000
4000

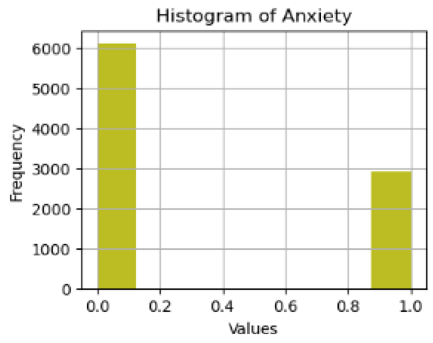

Histogram of Diabetes
Frequency
Values
Yes
No
0
1000
2000
3000
4000
5000
6000
7000

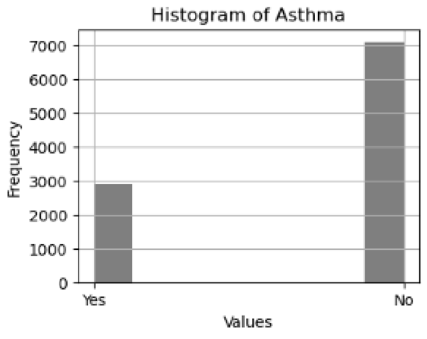




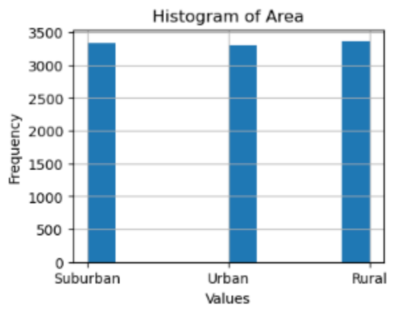


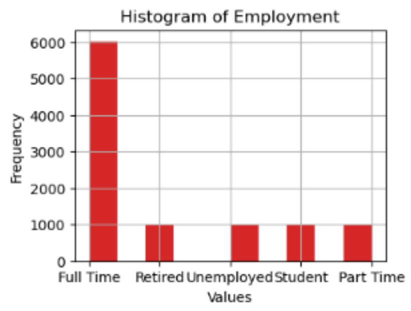


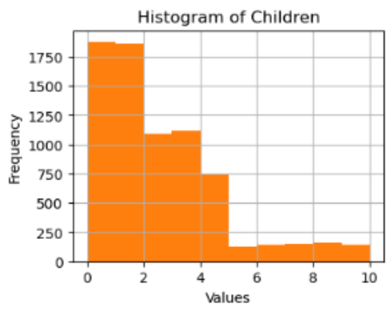


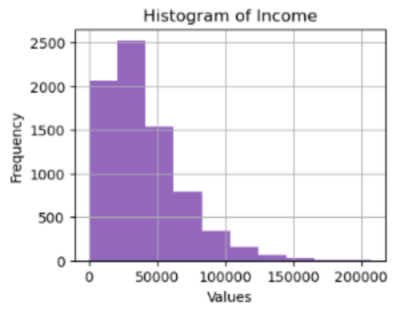


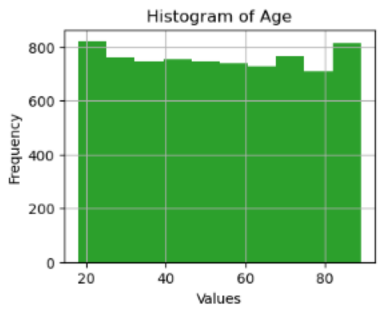
```
# Create histograms for listed columns
columns_to_plot_3 =
['Area','Children','Age','Employment','Income','Marital','Gender','ReAdmis',
                     'VitD_levels','Soft-
drink','Inital_admin','TotalCharge','Additional_charges']

# Define list of colors for bars(Blue: #1f77b4,Orange: #ff7f0e,Green:
#2ca02c,Red: #d62728,Purple: #9467bd,Brown: #8c564b,Pink: #e377c2,Gray:
#7f7f7f,Yellow: #bcbd22,Cyan: #17becf)
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
'#e377c2', '#7f7f7f', '#bcbd22', '#17becf','#2ca02c','#7f7f7f']

for i, column in enumerate(columns_to_plot_2):
    plt.figure(figsize=(4, 3))
    plt.hist(PatientReadmit[column], bins=10, color=colors[i])
    plt.title(f'Histogram of {column}')
    plt.xlabel('Values')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

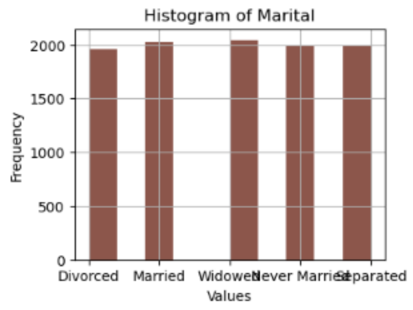

Histogram of Area
Frequency
Values
Suburban
Urban
Rural

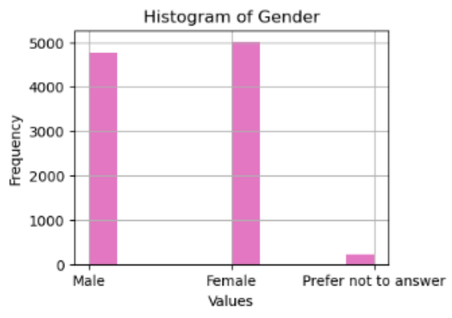

Histogram of Employment
Frequency
Values
Full Time
Retired
Unemployed
Student
Part Time

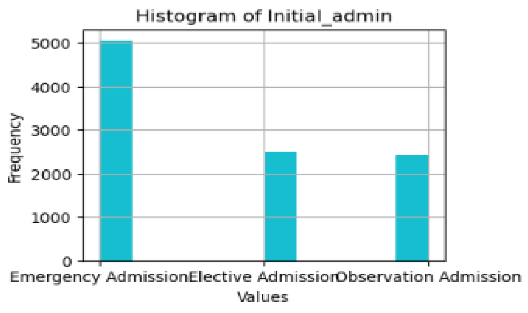

Histogram of Children
Frequency
Values

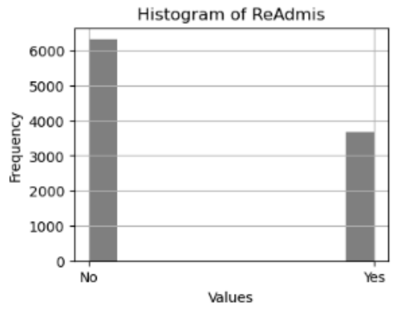

Histogram of Income
Frequency
Values

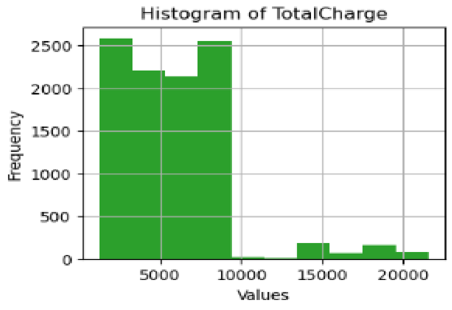

Histogram of Age
Frequency
Values

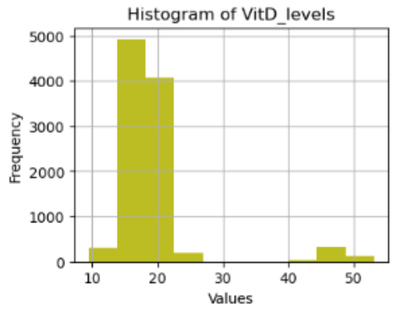

Histogram of Marital
Frequency
Values
Divorced
Married
Widowed
Never Married
Separated

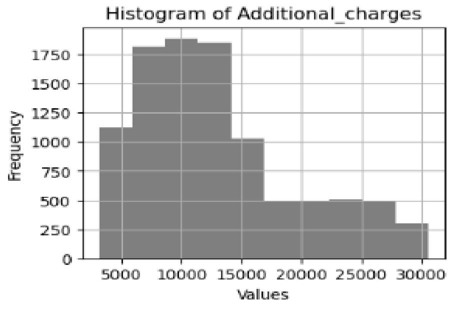












Correcting missing values is now able to be completed after gathering the above information. Based on the variables and the graphs above, the missing values in Income, Age, Income and Children are left skewed in the above histograms so the median will be the used value. For Soft_drink, Overweight, Anxiety, and Initial_admin they are all non-numeric so it’s appropriate to use the mode.
All missing values were replaced based on the measure of central tendency that was appropriate by using `fillna()`

```
# Replace missing values with median
median_age = PatientReadmit['Age'].median()
PatientReadmit['Age'].fillna(median_age, inplace=True)

median_income = PatientReadmit['Income'].median()
PatientReadmit['Income'].fillna(median_income, inplace=True)

median_children = PatientReadmit['Children'].median()
PatientReadmit['Children'].fillna(median_children, inplace=True)

# Replace missing values with mode
mode_soft_drink = PatientReadmit['Soft_drink'].mode()[0]
PatientReadmit['Soft_drink'].fillna(mode_soft_drink, inplace=True)

mode_overweight = PatientReadmit['Overweight'].mode()[0]
PatientReadmit['Overweight'].fillna(mode_overweight, inplace=True)

mode_anxiety = PatientReadmit['Anxiety'].mode()[0]
PatientReadmit['Anxiety'].fillna(mode_anxiety, inplace=True)

mode_initial_days = PatientReadmit['Initial_days'].mode()[0]
PatientReadmit['Initial_days'].fillna(mode_initial_days, inplace=True)
```

Another check to see if there are any missing values was performed.

```
# Check for missing data & display counts
missing_data = PatientReadmit.isnull().sum()
print("Missing data counts:")
print(missing_data)

Missing data counts:
CaseOrder             0
Customer_id           0
Interaction           0
UID                   0
City                  0
State                 0
County                0
Zip                   0
Lat                   0
Lng                   0
Population            0
Area                  0
Timezone              0
Job                   0
Children              0
Age                   0
Education             0
Employment            0
Income                0
Marital               0
Gender                0
ReAdmis               0
VitD_levels           0
Doc_visits            0
Full_meals_eaten      0
VitD_supp             0
Soft_drink            0
Initial_admin         0
HighBlood             0
Stroke                0
Complication_risk     0
Overweight            0
Arthritis             0
Diabetes              0
Hyperlipidemia        0
BackPain              0
Anxiety               0
Allergic_rhinitis     0
Reflux_esophagitis    0
Asthma                0
Services              0
Initial_days          0
TotalCharge           0
Additional_charges    0
Item1                 0
Item2                 0
Item3                 0
Item4                 0
Item5                 0
Item6                 0
Item7                 0
Item8                 0
dtype: int64
```

Now that all values have been replaced, checking for outliers is the next step. That starts with changing some of the columns to boolean data types since Python treats boolean as a numeric data type (Zadka, 2020).

```
# Select columns to convert
convert_columns = [
    'HighBlood', 'Stroke', 'Complication_risk', 'Overweight', 'Arthritis',
    'Diabetes', 'Hyperlipidemia','BackPain','Anxiety', 'Allergic_rhinitis',
    'Reflux_esophagitis', 'Asthma', 'Soft_drink']

# Convert columns to bool
for col in convert_columns:
    # Convert 'Yes' to True and 'No' to False
    try:
        PatientReadmit[col] = PatientReadmit[col].str.startswith('Yes')
    except AttributeError:
        print(f"{col}' is not a string column. Next column-")

    PatientReadmit[col] = PatientReadmit[col].astype(bool)
# Replace null values with median
    PatientReadmit[col] =
PatientReadmit[col].fillna(PatientReadmit[col].median())
```

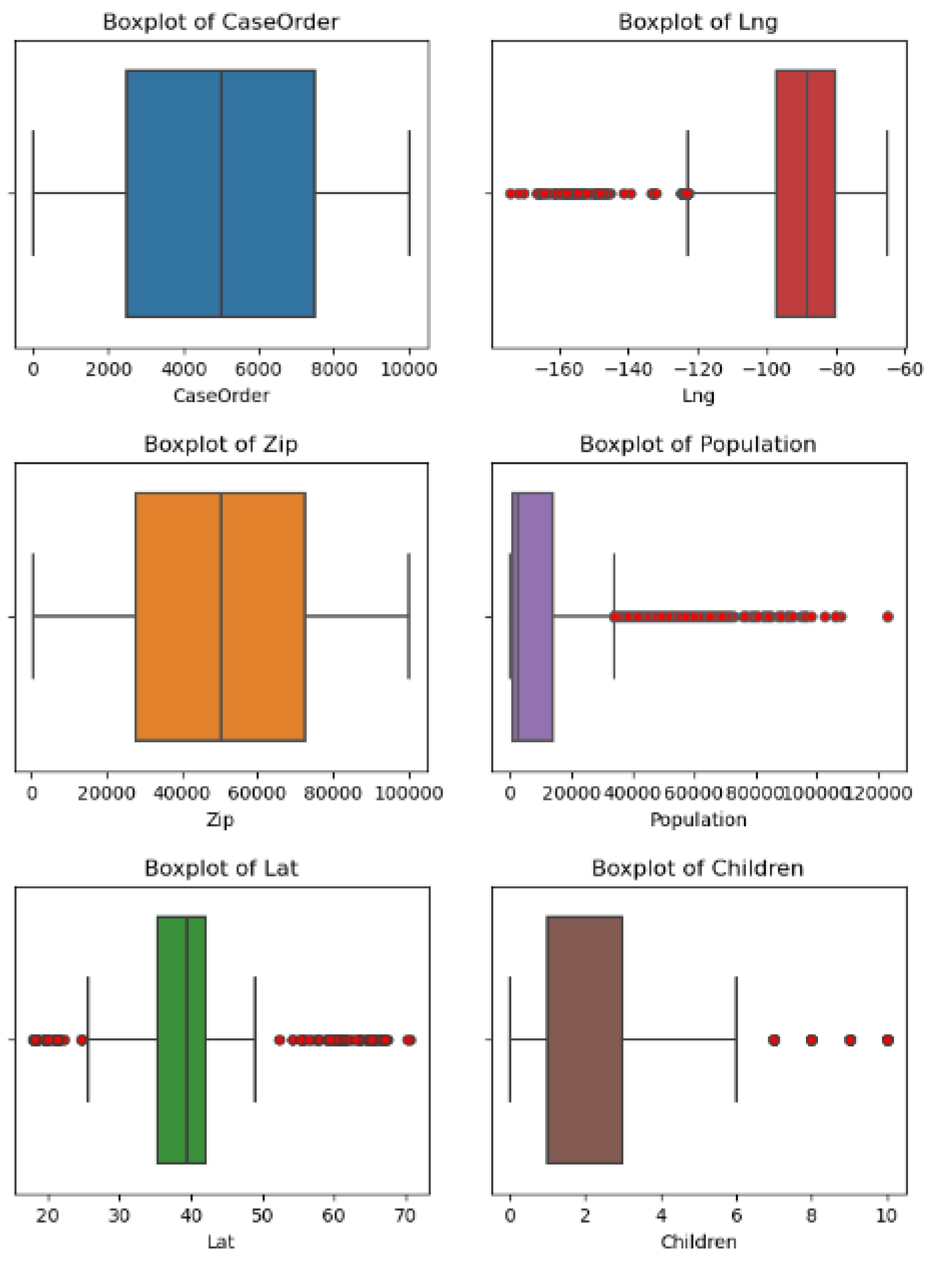
This changes the listed columns to False or True (0, 1) and iterates through the columns, listing out if any in the above list cannot be converted.

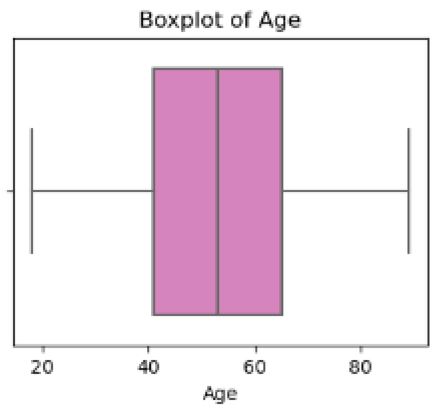
Another check of the data types to ensure they are ready for creating box plots is next.

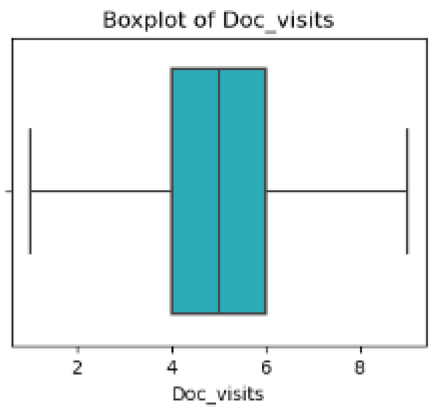
```
# Check the data types of each column in the DataFrame
print(PatientReadmit.dtypes)
```

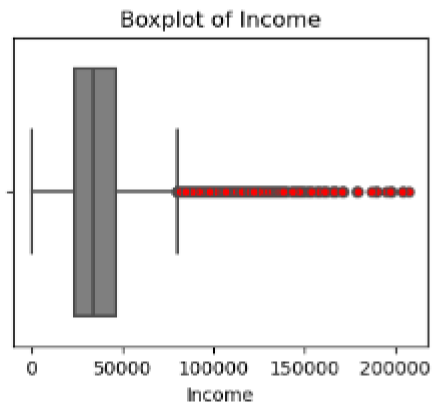
Now create box plots to check for outliers so that we can address any values that could skew our data. This code changes the boolean columns to numeric so that Matlibplot can create the needed box plots showing outliers.

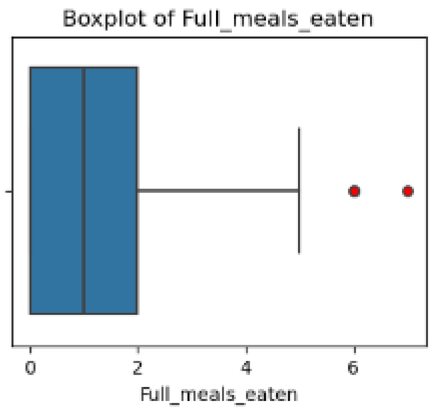
```
# Create boxplots
# Define list of colors for bars
custom_palette = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd",
                  "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22", "#17becf"]

# Change boolean to numeric (0 for False, 1 for True)
PatientReadmit_box = PatientReadmit.select_dtypes(include=['float64',
'int64', 'int32']).copy()
bool_col = PatientReadmit.select_dtypes(include=['bool']).astype(int)
PatientReadmit_box[bool_col.columns] = bool_col

# Create boxplot for columns
for i, column in enumerate(PatientReadmit_box.columns):
    plt.figure(figsize=(4 ,3))
    sns.boxplot(x=PatientReadmit_box[column], color=custom_palette[i %
len(custom_palette)], flierprops=dict(markerfacecolor='r', marker='o'))
    plt.title(f'Boxplot of {column}')
    plt.show()
```

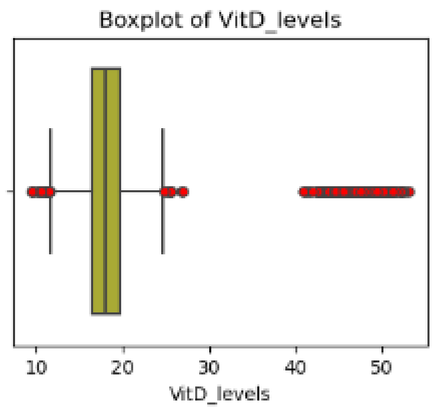

Boxplot of CaseOrder
0 2000 4000 6000 8000 10000
CaseOrder
Boxplot of Lng
−160 −140 −120 −100 −80 −60
Lng
Boxplot of Zip
0 20000 40000 60000 80000 100000
Zip
Boxplot of Population
0 20000 40000 60000 80000 100000 120000
Population
Boxplot of Lat
20 30 40 50 60 70
Lat
Boxplot of Children
0 2 4 6 8 10
Children

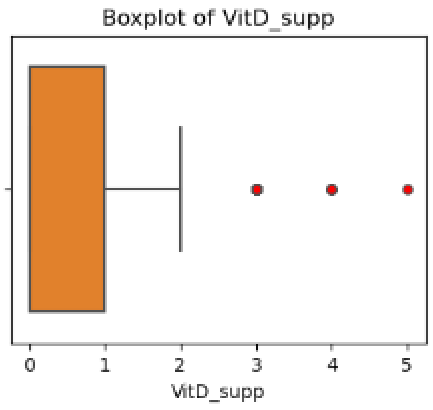

Boxplot of Age
20
40
60
80
Age

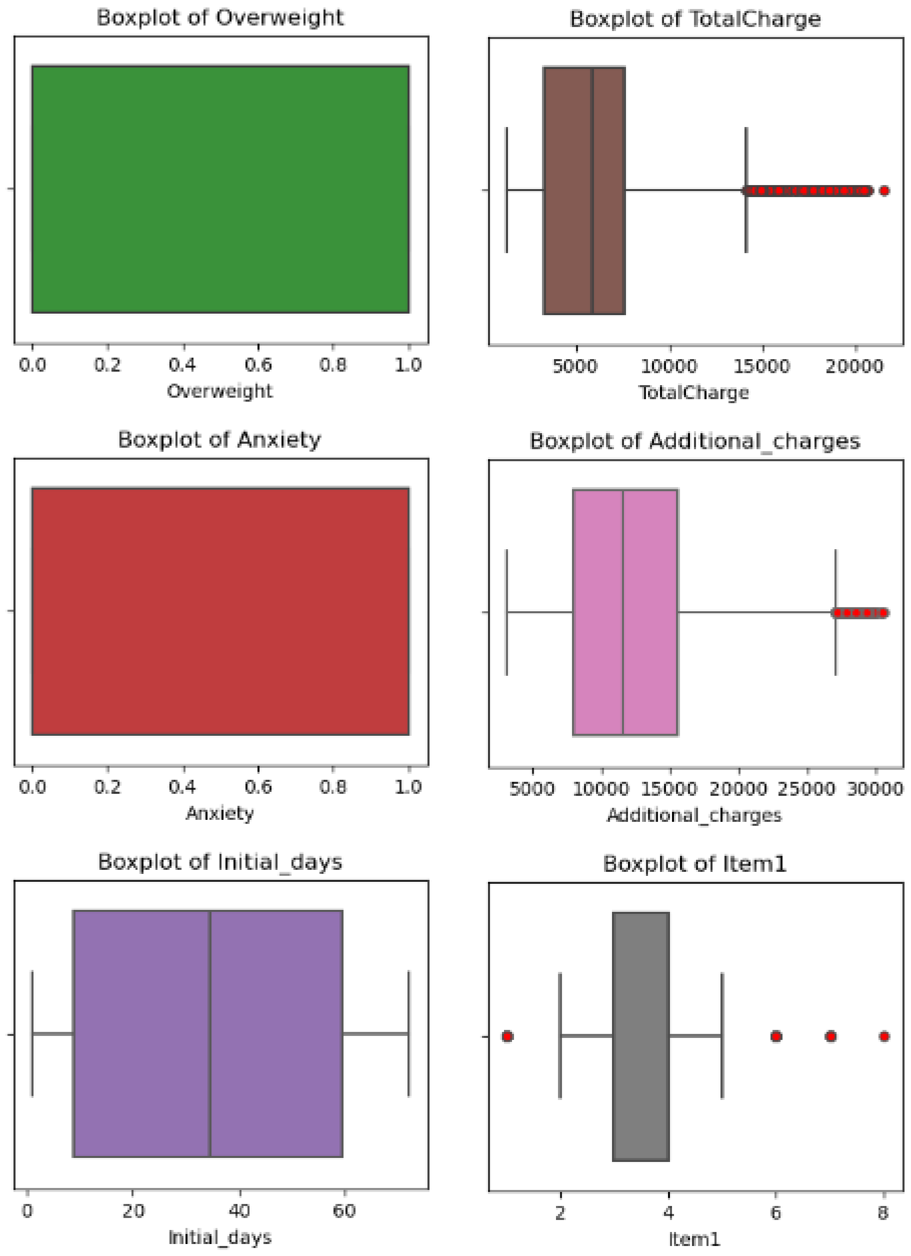

Boxplot of Doc_visits
2
4
6
8
Doc_visits

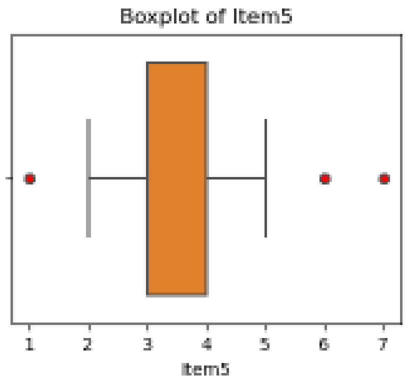

Boxplot of Income
0
50000
100000
150000
200000
Income

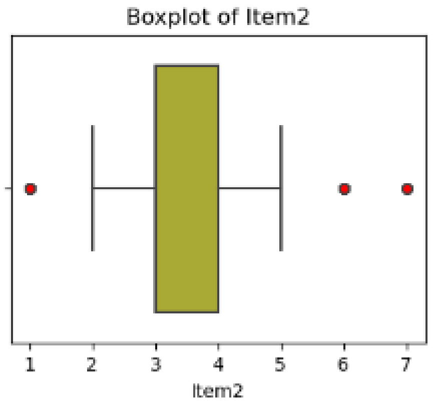

Boxplot of Full_meals_eaten
0
2
4
6
Full_meals_eaten

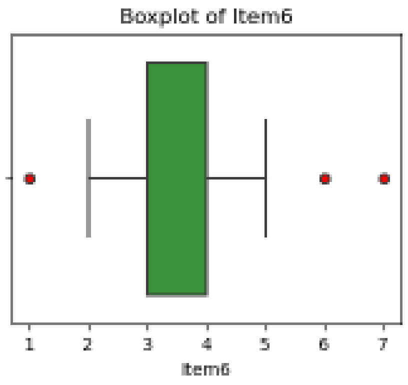

Boxplot of VitD_levels
10
20
30
40
50
VitD_levels

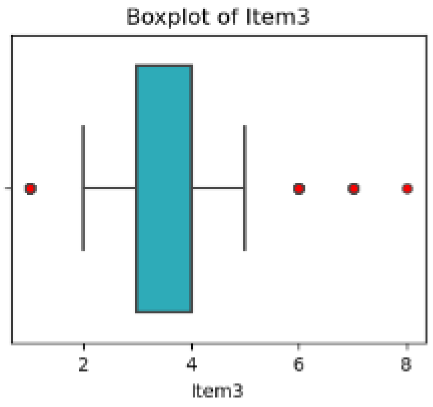

Boxplot of VitD_supp
0
1
2
3
4
5
VitD_supp

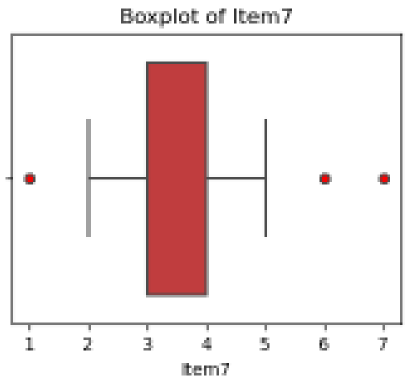

Boxplot of Overweight
0.0
0.2
0.4
0.6
0.8
1.0
Overweight
Boxplot of TotalCharge
5000
10000
15000
20000
TotalCharge
Boxplot of Anxiety
0.0
0.2
0.4
0.6
0.8
1.0
Anxiety
Boxplot of Additional_charges
5000
10000
15000
20000
25000
30000
Additional_charges
Boxplot of Initial_days
0
20
40
60
Initial_days
Boxplot of Item1
2
4
6
8
Item1

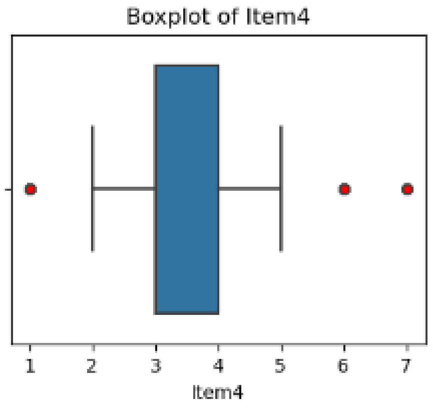

Boxplot of Item5
1
2
3
4
5
6
7
Item5

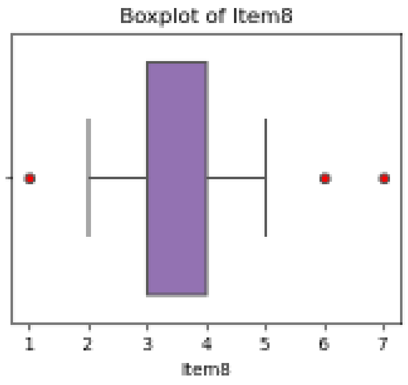

Boxplot of Item2
1
2
3
4
5
6
7
Item2

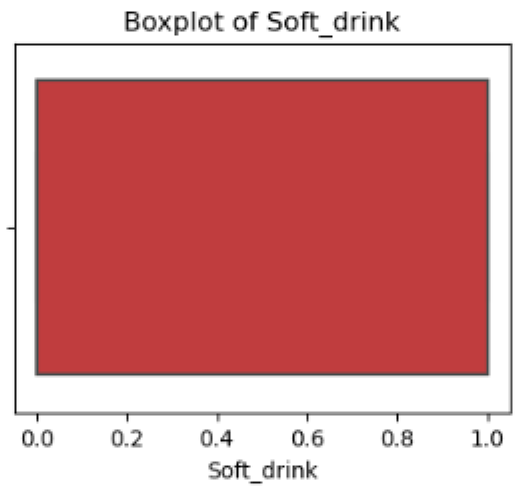

Boxplot of Item6
1
2
3
4
5
6
7
Item6

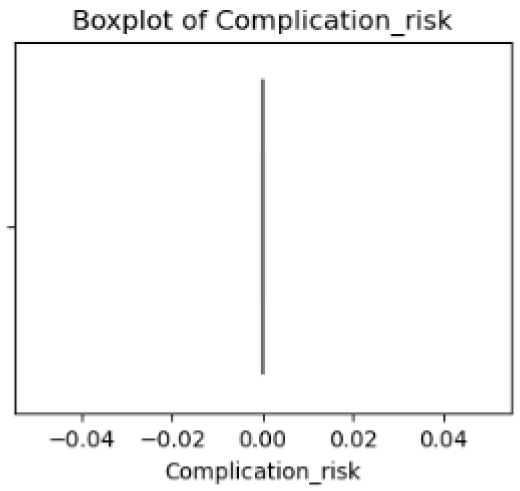

Boxplot of Item3
2
4
6
8
Item3

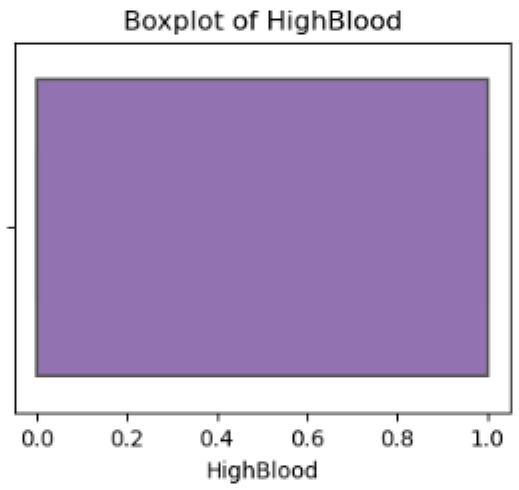

Boxplot of Item7
1
2
3
4
5
6
7
Item7

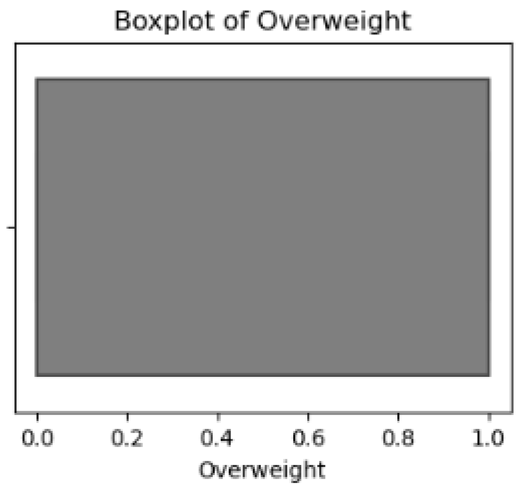

Boxplot of Item4
1
2
3
4
5
6
7
Item4

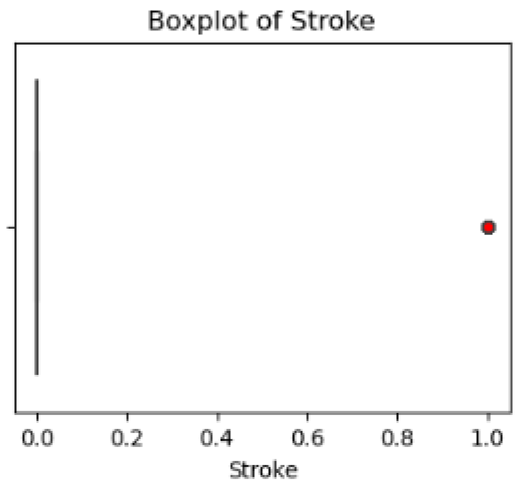

Boxplot of Item8
1
2
3
4
5
6
7
Item8

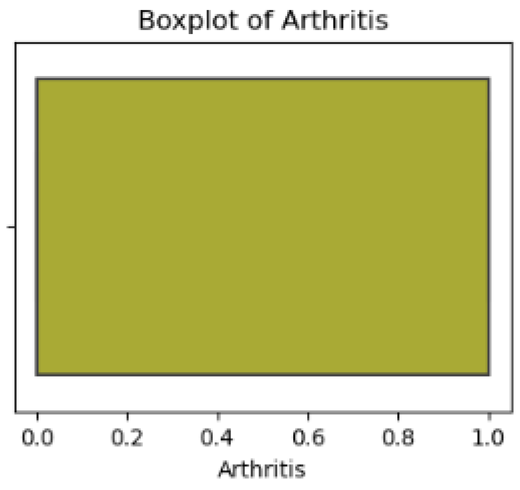

Boxplot of Soft_drink
0.0
0.2
0.4
0.6
0.8
1.0
Soft_drink

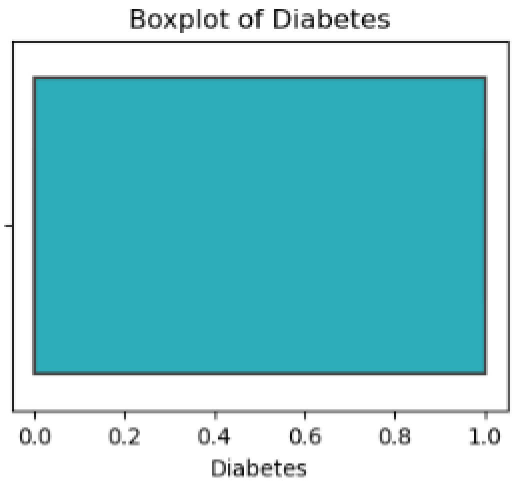

Boxplot of Complication_risk
−0.04
−0.02
0.00
0.02
0.04
Complication_risk

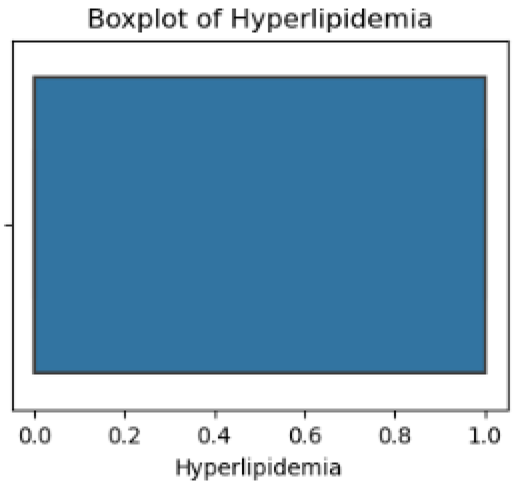

Boxplot of HighBlood
0.0
0.2
0.4
0.6
0.8
1.0
HighBlood

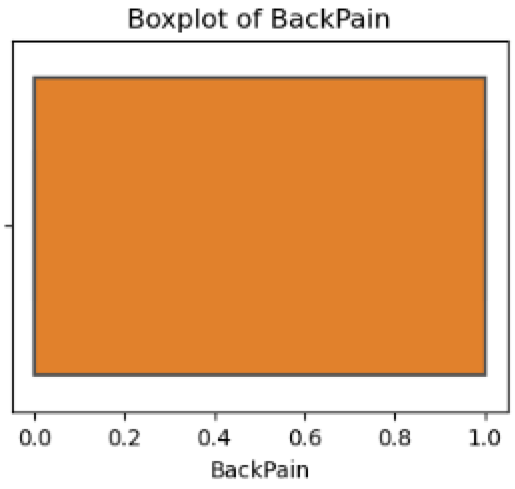

Boxplot of Overweight
0.0
0.2
0.4
0.6
0.8
1.0
Overweight

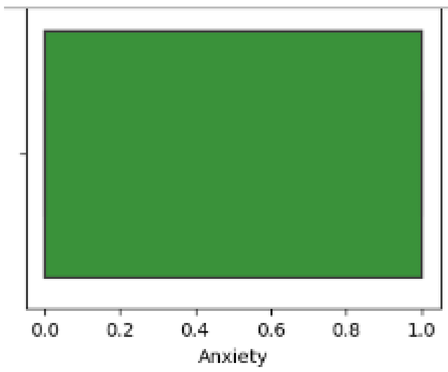

Boxplot of Stroke
0.0
0.2
0.4
0.6
0.8
1.0
Stroke

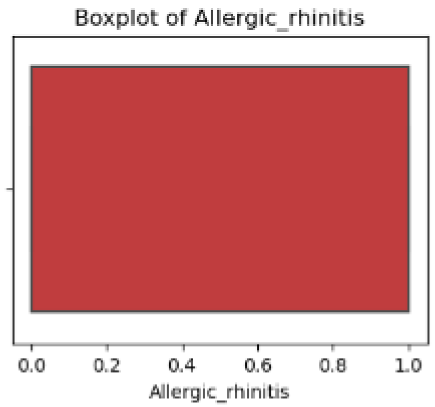

Boxplot of Arthritis
0.0
0.2
0.4
0.6
0.8
1.0
Arthritis

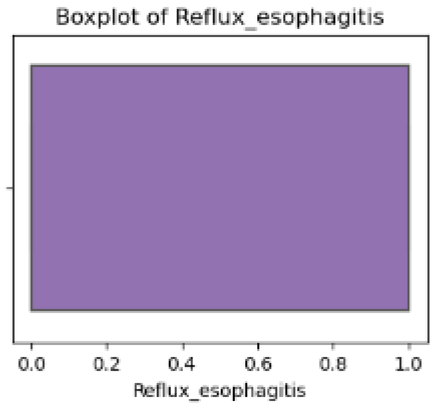

Boxplot of Diabetes
0.0
0.2
0.4
0.6
0.8
1.0
Diabetes

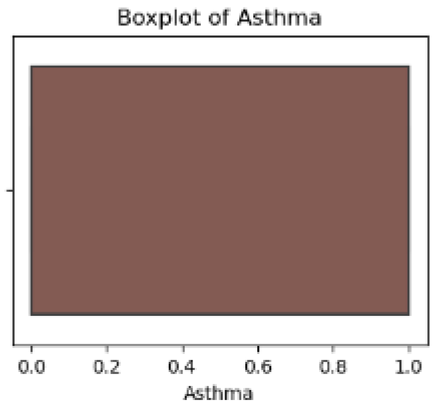

Boxplot of Hyperlipidemia
0.0
0.2
0.4
0.6
0.8
1.0
Hyperlipidemia


Boxplot of BackPain
0.0
0.2
0.4
0.6
0.8
1.0
BackPain


0.0
0.2
0.4
0.6
0.8
1.0
Anxiety


Boxplot of Allergic_rhinitis
0.0
0.2
0.4
0.6
0.8
1.0
Allergic_rhinitis


Boxplot of Reflux_esophagitis
0.0
0.2
0.4
0.6
0.8
1.0
Reflux_esophagitis


Boxplot of Asthma
0.0
0.2
0.4
0.6
0.8
1.0
Asthma

The columns that had outliers identified were Lat, Lng, Population, Children, Income, VitD_levels, Full_meals_eaten, VitD_supp, TotalCharge, Additional_charges, Items 1-8 and Stroke. Additional checking on Stroke was needed to gather the mean, median and mode to assess the outlier. It was determined that since the values are 0 or 1 that it is not an outlier, as patients can either have had a stroke (1) or not had a stroke (0). All the ones listed with values outside the "box" are expected based on the variable and it's definition.

```
# Get mean, med, mode for Stroke
PatientReadmit["Stroke_m"] = PatientReadmit["Stroke"].astype(int)

mean_stroke = PatientReadmit["Stroke_m"].mean()
median_stroke = PatientReadmit["Stroke_m"].median()
mode_stroke = PatientReadmit["Stroke_m"].mode()

print("Mean:", mean_stroke)
print("Median:", median_stroke)
print("Mode:", mode_stroke)
 OUTPUT:
Mean: 0.1993
Median: 0.0
Mode: 0    0
Name: Stroke_m, dtype: int32
```

After addressing the outlier, Stroke_m was dropped from the table. This was done to ensure that there was no duplication of values for the variable.

4. **Annotated code attached, file name D206_Austin_T_Code.ipynb dropped seperately in PA submission**
5. **Copy of the cleaned data attached, file name D206_Austin_T_Clean.csv dropped seperately in PA submission.**
6. **Limitations of the cleaning process**

There were a few limitations found during the cleaning process. First, much of the data in the medical_data_raw file was around patient demographics or answers to survey questions. This type of data doesn't lend itself to answering questions around patient readmission or why patients' initial hospital stay wasn't sufficient to resolve/help their health issues. (Agarwal, 2018).

A significant amount of missing data was encountered, and the columns required imputations based on measures of central tendency, type of variable and the normal distribution curve. Replacing missing values with the right information is key to ensuring quality data for any research.

There we no outliers that would affect the research based on the values collected and the visual inspections of graphing that was done. There are several columns, such as income and some of the survey questions that could skew other research questions, so those will need to be more thorough addressed in the future using this data (Saturn Cloud, 2023).

The descriptions of the columns were also taken into consideration for the outliers and the missing values imputation. There were several columns that were vague in their description, or the values were not well explained, such as the survey questions.

7. **Limitations effect on research question**

The current research question, "What medical conditions influence hospital readmission?", isn't highly affected by the limitations on the data cleaning process since the variables used are yes/no questions. There are enough columns in the data set around patients medical conditions that can be evaluated and compared against other data in the set such as the type of admission, meals eaten, and readmission information just to name a few. There could be some issues with NA responses and using chosen measures to address those not answered, however, overall, the data cleaning isn't an issue.

E. Principal Component Analysis (PCA)

1. Total number of principal component and output matrix

The first step was loading the cleaned data, import needed packages below and checking the data types and nulls.

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

# Load cleaned data
PatientPCA = pd.read_csv('D206_Austin_T_Clean.csv', index_col=0)
```

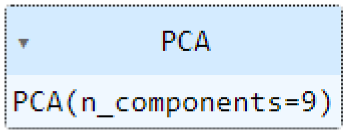
Since PCA's can only use numeric variables, a new data frame was created to store the subset of columns to be used to calculate the PCA

```
# Create new data frame with only continuous columns)
PatientPCA_Select =
PatientPCA[['Lat','Lng','Children','Age','Income','VitD_levels',
'Initial_days','TotalCharge','Additional_charges']]

print(PatientPCA_Select)

Lat                      float64
Lng                      float64
Children                 float64
Age                      float64
Income                   float64
VitD_levels              float64
Initial_days             float64
TotalCharge              float64
Additional_charges       float64
dtype: object
```

To ensure that the selected columns are aligned with needs of the PCA, the following was called via the print(). This solidifies that all the data in the data frame is of numeric type.

Next normalization of the data and generating PCA will allow for the next steps in the process. Once the code is ran, the output on how many components are in the PCA is generated.

```
# Normalize data
PatientPCA_Normalized = (PatientPCA_Select-
PatientPCA_Select.mean())/PatientPCA_Select.std()
print(PatientPCA_Normalized)

# Apply PCA
pca = PCA(n_components = PatientPCA_Normalized.shape[1])

# Fit PCA to the data
pca.fit(PatientPCA_Normalized)
```



```
PCA
PCA(n_components=9)
```

Continuing down the path to completing the finds, component loadings or loadings need to be surveyed. According to Holland (n.d.) loadings describe how much each variable contributes to a particular principal component. If the loadings are positive that means a variable and a principal component are positively correlated so an increase in one increases the other. Negative loadings then mean there is a negative correlation. Large loadings no matter if positive or negative correlates to a variable strongly affecting that principal component.

```
# Create DataFrame with principal component loadings
loadings = pd.DataFrame(pca.components_.T,
                        columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6',
'PC7', 'PC8', 'PC9'],
                        index=PatientPCA_Normalized.columns)
loadings
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
Lat	-0.014818	-0.017106	0.704040	-0.096144	-0.058160	0.004931	-0.700760	-0.009581	0.002759
Lng	-0.003314	0.018036	-0.704133	-0.023509	-0.080351	-0.092307	-0.698711	0.011585	0.000410
Children	0.003221	0.011683	0.072267	0.609904	0.397154	-0.680020	-0.049046	-0.009012	0.001387
Age	0.094835	0.700340	0.006861	0.005642	-0.015913	0.018509	-0.001800	-0.706786	0.017500
Income	-0.008110	-0.005165	-0.046336	0.136618	0.787006	0.586312	-0.126086	-0.007601	0.000329
VitD_levels	0.558516	-0.068361	-0.007355	-0.468903	0.281456	-0.257744	0.019736	-0.023686	-0.562906
Initial_days	0.420309	-0.074935	0.015911	0.616150	-0.365454	0.343990	-0.044463	-0.006053	-0.428872
TotalCharge	0.702085	-0.091269	0.002220	-0.000453	0.002093	0.004542	-0.007980	0.021354	0.705831
Additional_charges	0.095796	0.700089	0.028206	0.003548	0.004385	0.011901	-0.001306	0.706424	-0.026324

As you can see, there are no significant correlations between a PC and a variable. Taking into account the Kaiser Rule (Kaiser Criterion) as researched in an article on statpower.net(Steigler, 2015) this rule states that components based on eigenvalues greater than 1 are to be kept. The above data indicates there are no variables and components that have a correlation greater than .78 so these are all below average relationships.

2. Justify reduced number and scree plot

PCA's are a particular type of regression analysis designed to look at linear combinations with maximum variance. There are limitations on the type of data that can be analyzed according to Steiger(2015). In the medical_raw_data file, there were columns of categorical and numerical data. PCA's require all data to be numerical therefore the data to be used had to be limited.

Calculating the covariance matrix and eigenvalues led us to define how many components are significant in the data set.

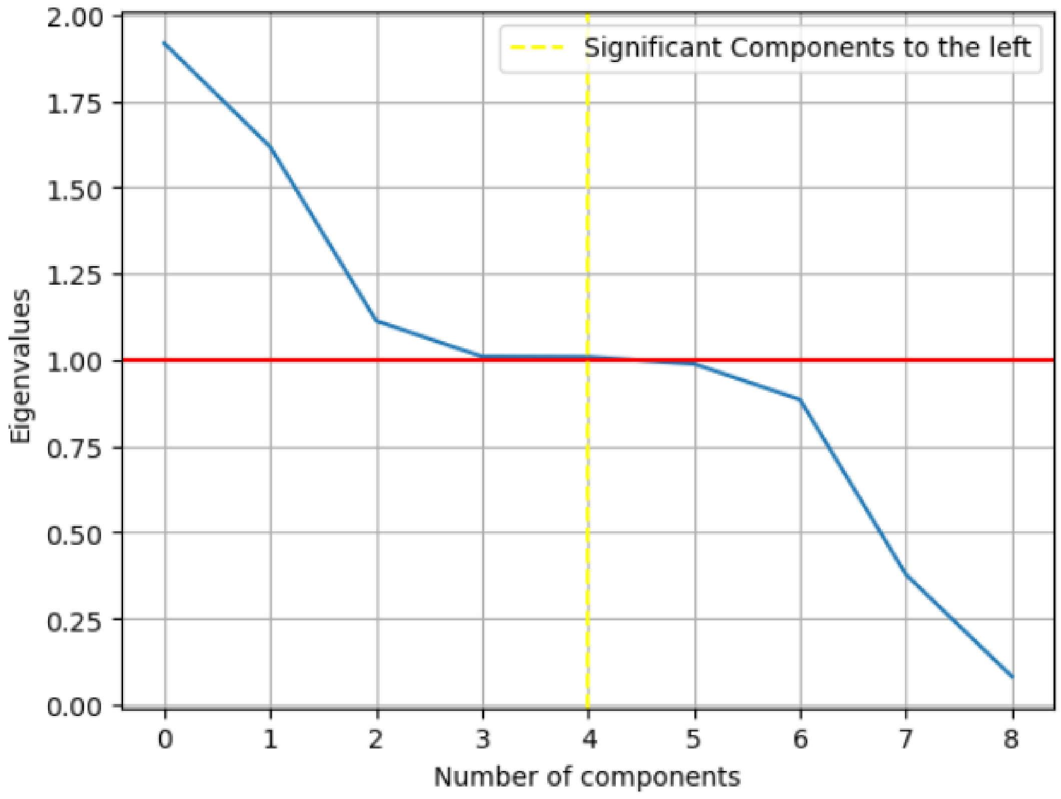
```
# Calculate covariance and vectors to define eigenvalues
PatientPCA_covariance = np.dot(PatientPCA_Normalized.T,
PatientPCA_Normalized)/PatientPCA_Select.shape[0]
PatientPCA_eigenvalues = [np.dot(eigenvector.T, np.dot(PatientPCA_covariance,
eigenvector)) for eigenvector in pca.components_]

signif_components = len(list(filter(lambda x: x >= 1, eigenvalues)))
```

```
print(f"The total number of significant components is {signif_components}.")
```

```
OUTPUT:
The total number of significant components is 5.
```

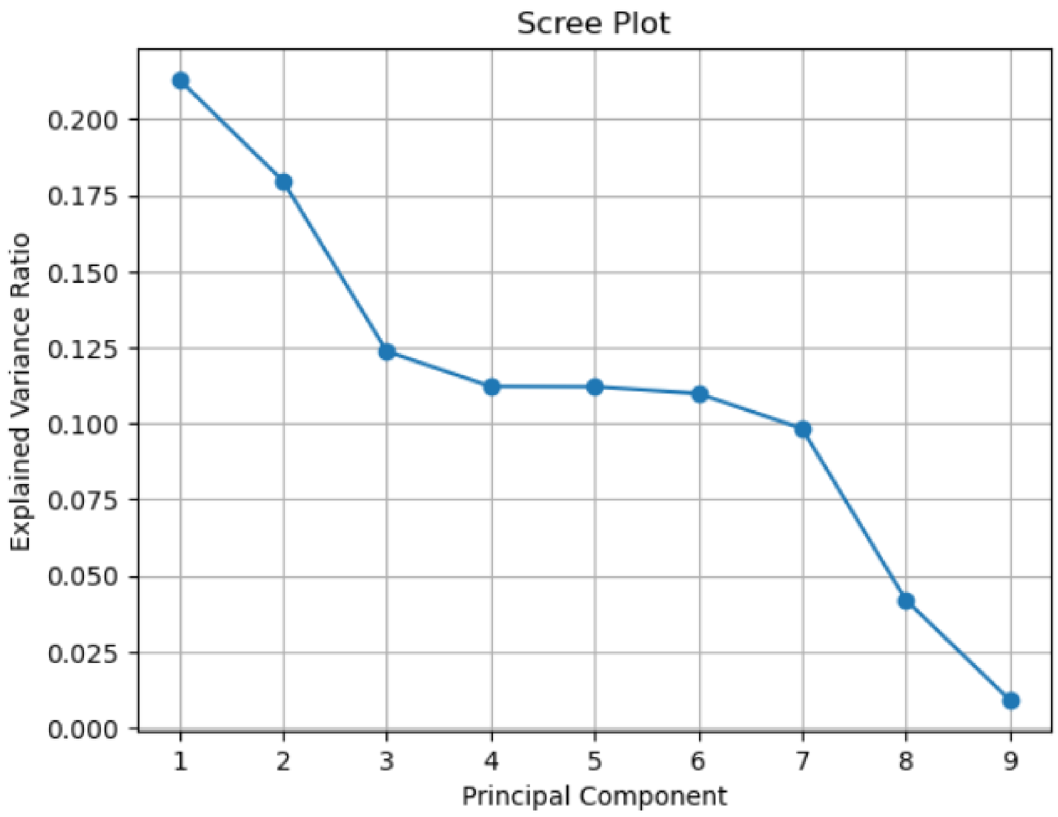
Based on the above, out of 9 components there are 5 that explain most of the data. A graph of eigenvalues will help visualize which components these are. Below shows the number of significant components indicated to the left of the yellow line and a red line to mark the Kaiser rule (>=1).

```
plt.plot(PatientPCA_eigenvalues)
plt.xlabel('Number of components')
plt.ylabel('Eigenvalues')
plt.axhline(y=1, color="red")
plt.axvline(x=signif_components-1, color = "yellow", linestyle="--",
label='Significant Components to the left')
plt.grid(True)
plt.legend()
plt.show()
```



A scree plot by contrast provides information around the data and how much variance is reflected by a given component. As you can see the highest value is just above 20% so the most significant component only explains about 1/4 of the data which isn't a good indicator that the PCA is helpful for the company in research with this data.

```
# Plot scree plot
plt.plot(np.arange(1, len(explained_variance_ratio) + 1),
explained_variance_ratio, marker='o', linestyle='-')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.xticks(np.arange(1, len(explained_variance_ratio) + 1))
plt.grid(True)
plt.show()
```



3. **How the organization will benefit from PCA**

As you can see above the highest value is just above 20% so the most significant component only explains about 1/4 of the data which isn't a good indicator that the PCA is helpful for the company in research with this data. Out of 52 columns of data, there were only 9 that were "right" for use with a PCA, again this is a low percentage of the gathered data to begin with. At this time, based on how the data was gathered, there isn't much value in the PCA for the organization.

Part IV. Panopto Video

Below is the link to a video demonstration of the functionality of the code used for analysis of the programming environment.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=566598d4-609a-4e0c-9d78-b159015e106e

F. Web Sources:

(To ensure that web sources and other sources were in APA citation style, OpenAI was consulted to help with formatting. OpenAI is cited as a source below based on that action)

Agarwal, M. (2018, March 26). Python Data Cleaning With NumPy and Pandas. Real Python. https://realpython.com/python-data-cleaning-numpy-pandas/

Banerjee, C. (2023, March 7). LabelEncoder. Medium. https://medium.com/@chandradip93/labelencoder-a31a27763a9f

Seaborn. (n.d.). Introduction to Seaborn. Retrieved from https://seaborn.pydata.org/tutorial/introduction.html

GeeksforGeeks. (2023, April 18). ML | Label Encoding of Datasets in Python. GeeksforGeeks. https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

"Numeracy, Maths and Statistics - Academic Skills Kit." (n.d.). Retrieved from https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/data-presentation/box-and-whisker-plots.html

Seaborn. (n.d.). Distributions. Retrieved from https://seaborn.pydata.org/tutorial/distributions.html

G. Sources:

Devireddy, S. K. (2021, June 20). Principal Component Analysis. *Medium*. Retrieved from https://medium.com/analytics-vidhya/principal-component-analysis-754781cfb30f

divine_inner_voice. (2023, April 3). Why is Python Used for Data Cleaning in Data Science. Medium. https://medium.com/@divine_inner_voice/why-is-python-used-for-data-cleaning-in-data-science-740c669432ec

GeeksforGeeks. (2024, January 24). Detect and Remove the Outliers using Python. Retrieved from https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/

Holland, S. (n.d.). Data analysis in the geosciences. Retrieved from http://stratigrafia.org/8370/lecturenotes/principalComponents.html

Jupyter Notebook. (n.d.). Jupyter Notebook documentation. Retrieved from https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

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

Ninja, N. (2023, Jun 23) Handling Missing Values: Filling the Gaps in Data. https://letsdatascience.com/handling-missing-values/

OpenAI. (2024, April 19). Chat with ChatGPT [Chat transcript]. Retrieved from OpenAI platform.

Pandas. (2024). Package Overview. Retrieved from https://pandas.pydata.org/docs/getting_started/overview.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from https://scikit-learn.org/stable/modules/decomposition.html#pca

Saturn Cloud. (2023, June 19) How to Detect and Exclude Outliers in a Pandas DataFrame. https://saturncloud.io/blog/how-to-detect-and-exclude-outliers-in-a-pandas-dataframe

Sisense Team. (2018, May 2). R & Python 101 – Data Cleaning. Sisense. Retrieved from https://www.sisense.com/blog/r-python-101-data-cleaning/

Solomon, B. (2018, July 2). Python Histogram Plotting: NumPy, Matplotlib, pandas & Seaborn. https://realpython.com/python-histograms/

Steiger, J. H. (2015, February 16). Principal components analysis. Retrieved from https://www.statpower.net/Content/312/R%20Stuff/PCA.html

Verma, Y. (2022, May 18) How to detect and treat outliers in categorical data? https://analyticsindiamag.com/how-to-detect-and-treat-outliers-in-categorical-data/

Zadka, M. (2020, October 19). Python Booleans: Use Truth Values in Your Code. Real Python. Retrieved from https://realpython.com/python-boolean/