Lora Milam

Western Govenors University

D206 Data Cleaning

21 September 2021

D206 Performance Assessment

# 1 Research Question

Lora Milam Masters Data Analytics (9/03/2021) Program Mentor:d206@wgu.edu

### 1.1 Question

Due to the concern for penalties of readmission, which fields can be most associated with readmission? In other words, what makes an individual more likely to be readmitted?

### 1.2 Variables

The data set is 10,000 raw medical patient records. The target variable is whether or not, within a month of release, each customer has been readmitted to the hospital. The title of the column is "ReAdmis".

The predictor variables that are provided in the dataset may have a correlation with the probability of the patient being readmitted due to previous ailments or problems from past admissions. These predictor variables include patient medical conditions (high blood pressure, stroke, obesity, arthritis, diabetes, etc.), patient information (service while hospitalized, days in hospital, type of initial admission, etc.), patient demographics (gender,age, job, education level, etc.). These predictor variables can be seen in the table below:

| **Variable** | **Description** |
|---|---|
| Unnamed: | - integer index |
| CaseOrder: | - integer index<br>- correlated to original order of raw data |
| Customer_id: | - character string object<br>- unique to patient |

| | |
|---|---|
| Interaction: | - character string object<br>- unique to patient transactions, procedures and admissions |
| UID: | - character string object<br>- unique to the transactions, procedures and admissions of a patient |
| City: | - character string object<br>- the city of residence of the patient |
| State: | - character string object<br>- the state of residence of the patient |
| County: | - character string object<br>- the county of residence of the patient |
| Zip: | - integer<br>- the zip code corresponding to the residence of the patient's |
| Lat: | - continuous numeric (floating numeric)<br>- GPS coordinates indicating the latitude corresponding to the patient's residence |
| Lng: | - continuous numeric (floating numeric)<br>- GPS coordinates indicating the longitude corresponding to the patient's residence |
| Population: | - integer<br>- the population that is within a mile radius of patient's resident |
| Area: | - nominal categorical<br>- character string object<br>- the area type corresponding to the patient's residence<br>- based on unofficial census data |

| | |
|---|---|
| | - the unique values are [‘Emergency Admission’, ‘Elective Admission’, ‘Observation Admission’]<br>`In [12]: df['Area'].unique()`<br>`Out[12]: array(['Suburban', 'Urban', 'Rural'], dtype=object)` |
| Timezone: | - nominal categorical<br>- character string object<br>- the time zone corresponding to the patient’s residence<br>- the unique values are ['America/Chicago', 'America/New_York', 'America/Los_Angeles', 'America/Indiana/Indianapolis', 'America/Detroit', 'America/Denver', 'America/Nome', 'America/Anchorage', 'America/Phoenix', 'America/Boise', 'America/Puerto_Rico', 'America/Yakutat', 'Pacific/Honolulu', 'America/Menominee', 'America/Kentucky/Louisville', 'America/Indiana/Vincennes', 'America/Toronto', 'America/Indiana/Marengo', 'America/Indiana/Winamac', 'America/Indiana/Tell_City', 'America/Sitka', 'America/Indiana/Knox', 'America/North_Dakota/New_Salem', 'America/Indiana/Vevay', 'America/Adak', 'America/North_Dakota/Beulah']<br>`In [13]: df['Timezone'].unique()`<br>`Out[13]: array(['America/Chicago', 'America/New_York', 'America/Los_Angeles', 'America/Indiana/Indianapolis', 'America/Detroit', 'America/Denver', 'America/Nome', 'America/Anchorage', 'America/Phoenix', 'America/Boise', 'America/Puerto_Rico', 'America/Yakutat', 'Pacific/Honolulu', 'America/Menominee', 'America/Kentucky/Louisville', 'America/Indiana/Vincennes', 'America/Toronto', 'America/Indiana/Marengo', 'America/Indiana/Winamac', 'America/Indiana/Tell_City', 'America/Sitka', 'America/Indiana/Knox', 'America/North_Dakota/New_Salem', 'America/Indiana/Vevay', 'America/Adak', 'America/North_Dakota/Beulah'], dtype=object)` |
| Job: | - nominal categorical<br>- character string object<br>- the occupation of the patient or insurance holder |
| Children: | - integer<br>- the amount of children within patient’s household |
| Age: | - integer |

| | |
|---|---|
| | - the patient’s age |
| Education: | - nominal categorical<br>- character string object<br>- the highest earned degree held by the patient<br>- the unique values are ['Some College, Less than 1 Year', 'Some College, 1 or More Years, No Degree', 'GED or Alternative Credential', 'Regular High School Diploma', "Bachelor's Degree", "Master's Degree", 'Nursery School to 8th Grade', '9th Grade to 12th Grade, No Diploma', 'Doctorate Degree', "Associate's Degree", 'Professional School Degree', 'No Schooling Completed']<br>In [15]: `df['Education'].unique()`<br>Out[15]: `array(['Some College, Less than 1 Year', 'Some College, 1 or More Years, No Degree', 'GED or Alternative Credential', 'Regular High School Diploma', "Bachelor's Degree", "Master's Degree", 'Nursery School to 8th Grade', '9th Grade to 12th Grade, No Diploma', 'Doctorate Degree', "Associate's Degree", 'Professional School Degree', 'No Schooling Completed'], dtype=object)` |
| Employment: | - categorical<br>- character string object<br>- the employment status currently being held by the patient<br>- the unique values are ['Full Time', 'Retired', 'Unemployed', 'Student', 'Part Time']<br>In [16]: `df['Employment'].unique()`<br>Out[16]: `array(['Full Time', 'Retired', 'Unemployed', 'Student', 'Part Time'], dtype=object)` |
| Income: | - numeric value<br>- the patient’s or insurance holder’s annual income |
| Marital: | - nominal categorical<br>- character string object<br>- the marital status of the patient or insurance holder<br>- the unique values are ['Divorced', 'Married', 'Widowed', 'Never Married', |

| | |
|---|---|
| | 'Separated']<br>In [17]: df['Marital'].unique()<br>Out[17]: array(['Divorced', 'Married', 'Widowed', 'Never Married', 'Separated'],<br>dtype=object) |
| Gender: | - nominal categorical<br>- character string object<br>- the gender of the patient<br>- the unique values are ['Male', 'Female', 'Prefer not to answer']<br>In [18]: df['Gender'].unique()<br>Out[18]: array(['Male', 'Female', 'Prefer not to answer'], dtype=object) |
| ReAdmins | - binary categorical<br>- character string object<br>- whether or not, within a month of release, each customer has been readmitted to the hospital<br>- **target variable** |
| VitD_levels: | - continuous numeric (floating numeric)<br>- value of the vitamin D levels of the patient<br>- measured in ng/mL |
| Doc_visits: | - integer<br>- the number of times during the initial hospitalization that the primary physician visited the patient |
| Full_meals_eaten: | - integer<br>- number of full meals eaten<br>Note: It counts as zero if the patient only eats a partial meal |
| Soft_drink: | - binary categorical<br>- character string object |

| | |
|---|---|
| | - whether or not a patient on a daily basis drinks three or more sodas<br>- the unique values are [Yes, No] |
| VitD_supp: | - integer<br>- the number of times that vitamin D supplements were administered to patient |
| Initial_admin: | - nominal categorical<br>- character string object<br>- the reason why the patient was initially admitted into the hospital<br>In [19]: `df['Initial_admin'].unique()`<br>Out[19]: `array(['Emergency Admission', 'Elective Admission', 'Observation Admission'], dtype=object)` |
| HighBlood: | - binary categorical<br>- character string object<br>- whether or not the patient has high blood pressure<br>- the unique values are [Yes, No] |
| Stroke: | - binary categorical<br>- character string object<br>- whether or not the patient has had a stroke<br>- the unique values are [Yes, No] |
| Complication_risk: | - ordinal categorical<br>- character string object<br>- the level of complication risk<br>- the unique values are [High, Medium, Low] |
| Overweight: | - binary categorical<br>- integer<br>- whether or not the patient is overweight, as determined by their BMI |

| | |
|---|---|
| | elements: age, gender, and height<br>- the unique values are [1,0] |
| Arthritis: | - binary categorical<br>- character string object<br>- whether or not the patient has arthritis<br>- the unique values are [Yes, No] |
| Diabetes: | - binary categorical<br>- character string object<br>- whether or not the patient has diabetes<br>- the unique values are [Yes, No] |
| Hyperlipidemia: | - binary categorical<br>- character string object<br>- whether or not the patient has hyperlipidemia<br>- the unique values are [Yes, No] |
| BackPain: | - binary categorical<br>- character string object<br>- whether or not the patient has chronic backpain<br>- the unique values are [Yes, No] |
| Anxiety: | - binary categorical<br>- integer<br>- whether or not the patient has an anxiety disorder<br>- the unique values are [1,0] |
| Allergic_rhinitis: | - binary categorical |

| | |
|---|---|
| | - character string object<br>- whether or not the patient has allergic rhinitis<br>- the unique values are [Yes, No] |
| Reflux_esophagitis: | - binary categorical<br>- character string object<br>- whether or not the patient has reflux esophagitis<br>- the unique values are [Yes, No] |
| Asthma: | - binary categorical<br>- character string object<br>- whether or not the patient has asthma<br>- the unique values are [Yes, No] |
| Services: | - nominal categorical<br>- character string object<br>- the primary service the patient received while hospitalized<br>- the unique values are ['Blood Work', 'Intravenous', 'CT Scan', 'MRI']<br>In [20]: `df['Services'].unique()`<br>Out[20]: `array(['Blood Work', 'Intravenous', 'CT Scan', 'MRI'], dtype=object)` |
| Initial_days: | - numeric value<br>- number of days,during the initial visit, the patient stayed in the hospital |
| TotalCharge: | - numeric value<br>- patient’s average daily charges,during the initial visit, for typical (not specialized) treatments and services |
| Additional_charges: | - numeric value<br>- patient’s average charges,during the initial visit, for additional treatments and |

| | |
|---|---|
| | services |
| Item1: | - integer value<br>- from most important to least important, the level of importance of timely admission<br>- the unique values are [1:8] |
| Item2: | - integer value<br>- from most important to least important, the level of importance of timely treatment<br>- the unique values are [1:8] |
| Item3: | - integer value<br>- from most important to least important, the level of importance of timely visits<br>- the unique values are [1:8] |
| Item4: | - integer value<br>- from most important to least important, the level of importance of reliability<br>- the unique values are [1:8] |
| Item5: | - integer value<br>- from most important to least important, the level of importance of options |
| Item6: | - integer value<br>- from most important to least important, the level of importance of hours of treatment<br>- the unique values are [1:8] |
| Item7: | - integer value<br>- from most important to least important, the level of importance of courteous staff |

| | |
|---|---|
| | - the unique values are [1:8] |
| Item8: | - integer value<br>- from doctor from most important to least important, the level of importance of evidence of active listening<br>- the unique values are [1:8] |

## 2 Data-Cleaning Plan

### 2.1 Plan to Clean Data

To properly analyze the data given, first I will clean the data by using Python3 and data cleaning methods. Python3 is the latest iteration of the programming language Python, as provided within Jupyter Notebooks. My plan for cleaning the data will be done through the following steps:

1. Reset index
2. Remove any redundant, irrelevant, or misleading fields
    a. 'X','Customer_id','Interaction_id','Job','Marital'
    b. The fields 'Job' and 'Marital' will be removed due to that they are both inconsistent and may lead to false assumptions of the data
    c. The fields associated with the questionnaire will be renamed so as to better reflect the fields' subject
3. Convert character object values to numeric values/separate into separate variables using dummy variables where applicable (Himamsh)
4. Imputation of NULL values
5. Convert NULL values of data provided by the patient to reflect a '0' response (Statsmodels)
6. Standard Deviation imputation for all other NULL values
7. Detection of outliers
8. View summary of univariate stats, search for any red flags

9. Visualize potential outliers with boxplots
10. Run hypothesis test on potential outliers, aka grubbs test
11. Standardize variables, if and when necessary
12. No data will be deleted or changed unless it is an obvious mistake. Note: Adjusting data too much will skew the model towards a conclusion that may not generalize due to the data being overly-manipulated.

### 2.2 Justification of Approach

This approach, though not without its limitations, is a good first iteration of addressing the relationship between readmission rates and other patient variables. This approach provides a more structured layout for the database while still keeping the original data intact. To best align with the course structure, I utilized the practice labs as a basis of the analysis.

### 2.3 Justification of Tools

As mentioned in previous sections, I will be utilizing Python's many capabilities to better analyze the database of medical patient records. Python3 is the latest iteration of the programming language Python, as provided within Jupyter Notebooks. Python is a high level, general purpose language that utilizes a variety of packages to tailor data. Two major packages I will be using to filter the data are Pandas and Numpy. Pandas is a powerful open source data analysis tool built on top of the Python programming language. Numpy provides a multidimensional array object for various math operations. Other packages include Matplotlib.pyplot for plotting data, NumPy to manipulate arrays, Sci-Py for linear algebra transformations, and Missingno for locating missing values.

Even though there are other methods that can be used to address this problem, I find Python3 and Jupyter Notebooks to be a convenient and intuitive way to visualize and draw conclusions from databases.

## 2.4 Code

In [1]:
```
# Import packages
import pandas as pd
import numpy as np
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from scipy.spatial.distance import cosine
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
%matplotlib inline
```

In [2]:
```
# Load data into Pandas dataframe
medical_df= pd.read_csv("/Users/lmilam/Downloads/e9d8sm5uf8df75k650df/medical_raw_data.csv")
```

In [3]:
```
# Display medical dataframe
medical_df
```

Out[3]:

| | Unnamed: 0 | CaseOrder | Customer_id | Interaction | UID | City | State | County | Zip | Lat | ... | TotalCharge |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | C412403 | 8cd49b13-f45a-4b47-a2bd-173ffa932c2f | 3a83ddb66e2ae73798bdf1d705dc0932 | Eva | AL | Morgan | 35621 | 34.34960 | ... | 3191.048774 |
| 1 | 2 | 2 | Z919181 | d2450b70-0337-4406-bdbb-bc1037f1734c | 176354c5eef714957d486009feabf195 | Marianna | FL | Jackson | 32446 | 30.84513 | ... | 4214.905346 |
| 2 | 3 | 3 | F995323 | a2057123-abf5-4a2c-abad-8ffe33512562 | e19a0fa00aeda885b8a436757e889bc9 | Sioux Falls | SD | Minnehaha | 57110 | 43.54321 | ... | 2177.586768 |
| 3 | 4 | 4 | A879973 | 1dec528d-eb34-4079-adce-0d7a40e82205 | cd17d7b6d152cb6f23957346d11c3f07 | New Richland | MN | Waseca | 56072 | 43.89744 | ... | 2465.118965 |

In [4]:
```
# List of Dataframe fields
df = medical_df.columns
print(df)
```

```
Index(['Unnamed: 0', 'CaseOrder', 'Customer_id', 'Interaction', 'UID',
'City',
       'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area',
       'Timezone', 'Job', 'Children', 'Age', 'Education', 'Employment',
       'Income', 'Marital', 'Gender', 'ReAdmis', 'VitD_levels', 'Doc_vis
its',
       'Full_meals_eaten', 'VitD_supp', 'Soft_drink', 'Initial_admin',
       'HighBlood', 'Stroke', 'Complication_risk', 'Overweight', 'Arthri
tis',
       'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety',
       'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services',
       'Initial_days', 'TotalCharge', 'Additional_charges', 'Item1', 'It
em2',
       'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8'],
      dtype='object')
```

### 2.4.1 Remove irrelevant fields

In [5]:
```
# Remove redundant "Unnamed: 0" field in the first column
df = medical_df.drop(columns ='Unnamed: 0', axis =1)
df.head()
```

Out[5]:

| | CaseOrder | Customer_id | Interaction | UID | City | State | |
|---|---|---|---|---|---|---|---|
| **0** | 1 | C412403 | 8cd49b13-f45a-4b47-a2bd-173ffa932c2f | 3a83ddb66e2ae73798bdf1d705dc0932 | Eva | AL | |
| **1** | 2 | Z919181 | d2450b70-0337-4406-bdbb-bc1037f1734c | 176354c5eef714957d486009feabf195 | Marianna | FL | |
| **2** | 3 | F995323 | a2057123-abf5-4a2c-abad-8ffe33512562 | e19a0fa00aeda885b8a436757e889bc9 | Sioux Falls | SD | Mir |
| **3** | 4 | A879973 | 1dec528d-eb34-4079-adce-0d7a40e82205 | cd17d7b6d152cb6f23957346d11c3f07 | New Richland | MN | |
| **4** | 5 | C544523 | 5885f56b-d6da-43a3-8760-83583af94266 | d2f0425877b10ed6bb381f3e2579424a | West Point | VA | |

5 rows × 52 columns

### 2.4.2 Rename misleading field names

In [6]:
```
# Rename survey columns to more identifiable names
df = df.rename(
    {'Item1':'Survey_TimelyAdmin',
     'Item2':'Survey_TimelyTreatment',
     'Item3':'Survey_TimelyVisits',
     'Item4':'Survey_Reliability',
     'Item5':'Survey_Options',
     'Item6':'Survey_HoursTreatment',
     'Item7':'Survey_CourteousStaff',
     'Item8':'Survey_ActiveListening'}, axis=1)
```

In [7]:
```
# Find the number of rows and columns of dataset
df.shape
```

Out[7]:
```
(10000, 52)
```

In [8]:
```
# Describe medical dataset stats
df.describe()
```

Out[8]:

| | CaseOrder | Zip | Lat | Lng | Population | Children | |
|---|---|---|---|---|---|---|---|
| **count** | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 7412.000000 | 7586.00 |
| **mean** | 5000.50000 | 50159.323900 | 38.751099 | -91.243080 | 9965.253800 | 2.098219 | 53.29 |
| **std** | 2886.89568 | 27469.588208 | 5.403085 | 15.205998 | 14824.758614 | 2.155427 | 20.65 |
| **min** | 1.00000 | 610.000000 | 17.967190 | -174.209690 | 0.000000 | 0.000000 | 18.00 |
| **25%** | 2500.75000 | 27592.000000 | 35.255120 | -97.352982 | 694.750000 | 0.000000 | 35.00 |
| **50%** | 5000.50000 | 50207.000000 | 39.419355 | -88.397230 | 2769.000000 | 1.000000 | 53.00 |
| **75%** | 7500.25000 | 72411.750000 | 42.044175 | -80.438050 | 13945.000000 | 3.000000 | 71.00 |
| **max** | 10000.00000 | 99929.000000 | 70.560990 | -65.290170 | 122814.000000 | 10.000000 | 89.00 |

8 rows × 25 columns

### 2.4.4 Set index

In [9]:
```
# Reseting index
df.index = np.arange(1, len(df) + 1)
```

### 2.4.3 Remove less meaningful fields

In [10]:
```
# Remove less relevant fields from stats description
df_stats = df.drop(columns =['CaseOrder','Customer_id','Interaction','UID',
df_stats.head()
```

Out[10]:

| | City | State | County | Zip | Lat | Lng | Population | Area | Timezone |
|---|---|---|---|---|---|---|---|---|---|
| **1** | Eva | AL | Morgan | 35621 | 34.34960 | -86.72508 | 2951 | Suburban | America/Chicago |
| **2** | Marianna | FL | Jackson | 32446 | 30.84513 | -85.22907 | 11303 | Urban | America/Chicago |
| **3** | Sioux Falls | SD | Minnehaha | 57110 | 43.54321 | -96.63772 | 17125 | Suburban | America/Chicago |
| **4** | New Richland | MN | Waseca | 56072 | 43.89744 | -93.51479 | 2162 | Suburban | America/Chicago |
| **5** | West Point | VA | King William | 23181 | 37.59894 | -76.88958 | 5287 | Rural | America/New_York |

5 rows × 46 columns

### 2.4.5 Re-expression of categorical data as numeric data

#### 2.4.5.1 State

```
# Re-express categorical data as numeric data
data = df_stats['State']
state_dict = {
    "AL" : 1, "AK" : 2, "AZ" : 3, "AR" : 4, "CA" : 5, "CO" : 6, "CT" : 7, '
  "GA" : 11, "HI" : 12, "ID" : 13, "IL" : 14, "IN" : 15, "IA" : 16, "KS" :
  "MD" : 21, "MA" : 22, "MI" : 23, "MN" : 24, "MS" : 25, "MO" : 26, "MT" :
  "NJ" : 31, "NM" : 32, "NY" : 33, "NC" : 34, "ND" : 35, "OH" : 36, "OK" :
  "RI" : 41, "SC" : 42, "SD" : 43, "TN" : 44, "TX" : 45, "UT" : 46, "VT" :
  "WI" : 51, "WY" : 52
}
for k,v in state_dict.items():
    data = data.replace(k,v)

df_stats['State'] = data
```

#### 2.4.5.2 Area

```
data = df_stats['Area']
area_dict = {
   "Rural":1, "Suburban":2, "Urban":3
}
for k,v in area_dict.items():
    data = data.replace(k,v)

df_stats['Area'] = data
```

#### 2.4.5.3 Timezone

```
data = df_stats['Timezone']
timezone_dict = {
  "America/Puerto_Rico" : -4,
  "America/Detroit" : -5,
  "America/Indiana/Indianapolis" : -5,
  "America/Indiana/Marengo" : -5,
  "America/Indiana/Vincennes" : -5,
  "America/Indiana/Vevay" : -5,
  "America/Indiana/Winamac" : -5,
  "America/Kentucky/Louisville" : -5,
  "America/New_York" : -5,
  "America/Toronto" : -5,
  "America/Chicago" : -6,
  "America/Indiana/Knox" : -6,
  "America/Indiana/Tell_City" : -6,
  "America/Menominee" : -6,
  "America/North_Dakota/Beulah" : -6,
  "America/North_Dakota/New_Salem" : -6,
  "America/Boise" : -7,
  "America/Denver" : -7,
  "America/Phoenix" : -7,
  "America/Los_Angeles" : -8,
  "America/Anchorage" : -9,
  "America/Nome" : -9,
  "America/Sitka" : -9,
  "America/Yakutat" : -9,
  "America/Adak" : -10,
  "Pacific/Honolulu" : -10
}
for k,v in timezone_dict.items():
    data = data.replace(k,v)

df_stats['Timezone'] = data
```

#### 2.4.5.4 Education

```
data = df_stats['Education']
education_dict = {
 "No Schooling Completed" : 0,
  "Nursery School to 8th Grade" : 8,
  "9th Grade to 12th Grade, No Diploma" : 12,
  "GED or Alternative Credential" : 12,
  "Regular High School Diploma" : 12,
  "Some College, Less than 1 Year" : 13,
  "Some College, 1 or More Years, No Degree" : 14,
  "Associate's Degree" : 15,
  "Bachelor's Degree" : 16,
  "Master's Degree" : 18,
  "Professional School Degree" : 20,
  "Doctorate Degree" : 24}
for k,v in education_dict.items():
    data = data.replace(k,v)

df_stats['Education'] = data
```

#### 2.4.5.5 Employment

```
# Impute qualitive data fields by creating binary dummy columns
# Exclude redundant values: ex. gender is categorized as male or female, ti
dmy = pd.get_dummies(df_stats['Employment'])
df_stats = pd.concat([df_stats,dmy],axis=1)
df_stats = df_stats.drop(columns = "Employment")
```

#### 2.4.5.6 Gender

```
dmy = pd.get_dummies(df_stats['Gender'])
dmy = dmy.iloc[:,:-1]
df_stats = pd.concat([df_stats,dmy],axis=1)
df_stats = df_stats.drop(columns = 'Gender')
```

#### 2.4.5.7 Readmission

```
data = df_stats['ReAdmis']
readmis_dict = {
    "No":0,"Yes":1
}
for k,v in readmis_dict.items():
    data = data.replace(k,v)

df_stats['ReAdmis'] = data
```

#### 2.4.5.8 Soft Drink

```
data = df_stats['Soft_drink']
for k,v in readmis_dict.items():
    data = data.replace(k,v)

df_stats['Soft_drink'] = data
```

#### 2.4.5.9 Initial Admission

```
dmy = pd.get_dummies(df_stats['Initial_admin'])
df_stats = pd.concat([df_stats,dmy],axis=1)
df_stats = df_stats.drop(columns = 'Initial_admin')
```

#### 2.4.5.10 High Blood Pressure

```
data = df_stats['Soft_drink']
for k,v in readmis_dict.items():
    data = data.replace(k,v)

df_stats['Soft_drink'] = data
```

#### 2.4.5.11 Stroke

```
data = df_stats['Stroke']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Stroke'] = data
```

#### 2.4.5.12 Complication Risk

```
data = df_stats['Complication_risk']
CompRisk_dict = {
    "Low":1,"Medium":2,"High":3
}
for k,v in CompRisk_dict.items():
    data = data.replace(k,v)

df_stats['Complication_risk'] = data
```

#### 2.4.5.13 Arthritis

```
data = df_stats['Arthritis']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Arthritis'] = data
```

#### 2.4.5.14 Diabetes

```
data = df_stats['Diabetes']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Diabetes'] = data
```

#### 2.4.5.15 Hyperlipidemia

```
data = df_stats['Hyperlipidemia']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Hyperlipidemia'] = data
```

#### 2.4.5.16 Back Pain

```
data = df_stats['BackPain']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['BackPain'] = data
```

#### 2.4.5.17 Allergic Rhinitis

```
data = df_stats['Allergic_rhinitis']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Allergic_rhinitis'] = data
```

#### 2.4.5.18 Reflux Esophagitis

```
data = df_stats['Reflux_esophagitis']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Reflux_esophagitis'] = data
```

#### 2.4.5.19 Asthma

```
data = df_stats['Asthma']
for k,v in readmis_dict.items():
    data = data.replace(k,v)
df_stats['Asthma'] = data
```

#### 2.4.5.20 Services

```
data = df_stats['Services']
services_dict = {
    "Blood Work":1,"Intravenous":2,"CT Scan":3,"MRI":4
}
for k,v in services_dict.items():
    data = data.replace(k,v)

df_stats['Services'] = data
```

### 2.4.6 Imputation of NULL values

#### 2.4.6.1 Summary of fields

```
df.describe()
```

| | CaseOrder | State | Zip | Lat | Lng | Population | |
|---|---|---|---|---|---|---|---|
| **count** | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.0 |
| **mean** | 5000.50000 | 26.839300 | 50159.323900 | 38.751099 | -91.243080 | 9965.253800 | 1.9 |
| **std** | 2886.89568 | 14.637045 | 27469.588208 | 5.403085 | 15.205998 | 14824.758614 | 0.8 |
| **min** | 1.00000 | 1.000000 | 610.000000 | 17.967190 | -174.209690 | 0.000000 | 1.0 |
| **25%** | 2500.75000 | 14.000000 | 27592.000000 | 35.255120 | -97.352982 | 694.750000 | 1.0 |
| **50%** | 5000.50000 | 26.000000 | 50207.000000 | 39.419355 | -88.397230 | 2769.000000 | 2.0 |
| **75%** | 7500.25000 | 39.000000 | 72411.750000 | 42.044175 | -80.438050 | 13945.000000 | 3.0 |
| **max** | 10000.00000 | 52.000000 | 99929.000000 | 70.560990 | -65.290170 | 122814.000000 | 3.0 |

#### 2.4.6.2 Code

NULL values are found in 7 fields: ‘Children’, ’Age’, ’Income’, ’Soft_drink’, ’Overweight’, ’Anxiety’, ‘Initial_days

NULL values of the fields ‘Children’,’Soft_drink’,’Anxiety’ will be converted to ‘0’. These fields are yes/no data, as reported by the patient. It is rational to conclude that these fields would be left empty by the patient -and thus recorded as null- if they were not applicable to the patient.

```
# Count of missing values by column
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
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
Survey_TimelyAdmin         0
Survey_TimelyTreatment     0
Survey_TimelyVisits        0
Survey_Reliability         0
Survey_Options             0
Survey_HoursTreatment      0
Survey_CourteousStaff      0
Survey_ActiveListening     0
dtype: int64
```

```
# Impute missing values for Soft drink, Children, and Anxiety
# Then print count of null values
df_stats['Soft_drink'] = df_stats['Soft_drink'].fillna('No')
df_stats['Children'] = df_stats['Children'].fillna(0)
df_stats['Anxiety'] = df_stats['Anxiety'].fillna(0)
data = df_stats.isnull().sum()
print(data)
```

```
City                      0
State                     0
County                    0
Zip                       0
Lat                       0
Lng                       0
Population                0
Area                      0
Timezone                  0
Children                  0
Age                       0
Education                 0
Employment                0
Income                    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
Survey_TimelyAdmin        0
Survey_TimelyTreatment    0
Survey_TimelyVisits       0
Survey_Reliability        0
Survey_Options            0
Survey_HoursTreatment     0
Survey_CourteousStaff     0
Survey_ActiveListening    0
dtype: int64
```

#### 2.4.6.2.1 Standard Deviation

All other NULL values will be filled using standard deviation.

```
# Calculate the standard deviation of the numeric fields
data_std = df_stats.std()
print(data_std)
```

```
Zip                      27469.588208
Lat                          5.403085
Lng                         15.205998
Population               14824.758614
Children                     2.155427
Age                         20.659182
Income                   28664.861050
VitD_levels                  6.723277
Doc_visits                   1.045734
Full_meals_eaten             1.008117
VitD_supp                    0.628505
Overweight                   0.454186
Anxiety                      0.467389
Initial_days                26.287050
TotalCharge               3377.558136
Additional_charges        6542.601544
Survey_TimelyAdmin           1.031966
Survey_TimelyTreatment       1.034825
Survey_TimelyVisits          1.032755
Survey_Reliability           1.036282
Survey_Options               1.030192
Survey_HoursTreatment        1.032376
Survey_CourteousStaff        1.021405
Survey_ActiveListening       1.042312
dtype: float64
```

```
# Re-validate column data types and missing values
df.columns.to_series().groupby(df_stats.dtypes).groups
```

```
{int64: ['Zip', 'Population', 'Doc_visits', 'Full_meals_eaten', 'VitD_supp', 'Survey_TimelyAd
min', 'Survey_TimelyTreatment', 'Survey_TimelyVisits', 'Survey_Reliability', 'Survey_Option
s', 'Survey_HoursTreatment', 'Survey_CourteousStaff', 'Survey_ActiveListening'], float64: ['L
at', 'Lng', 'Children', 'Age', 'Income', 'VitD_levels', 'Overweight', 'Anxiety', 'Initial_day
s', 'TotalCharge', 'Additional_charges'], object: ['City', 'State', 'County', 'Area', 'Timezo
ne', 'Education', 'Employment', 'Gender', 'ReAdmis', 'Soft_drink', 'Initial_admin', 'HighBloo
d', 'Stroke', 'Complication_risk', 'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain', 'Al
lergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services']}
```

```
# Find missing values
df.isnull()
```

| | CaseOrder | Customer_id | Interaction | UID | City | State | County | Zip | Lat | Lng | ... | TotalCharge | Additional_charges |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **2** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **3** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **4** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **5** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **9996** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **9997** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **9998** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **9999** | False | False | False | False | False | False | False | False | False | False | ... | False | False |
| **10000** | False | False | False | False | False | False | False | False | False | False | ... | False | False |

10000 rows × 52 columns

```
# Locate rows from dataset containing missing values
df.isnull().any(axis=1)
```

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

```
# Display columns with NAs
df.isna().any()
```

```
CaseOrder                False
Customer_id              False
Interaction              False
UID                      False
City                     False
State                    False
County                   False
Zip                      False
Lat                      False
Lng                      False
Population               False
Area                     False
Timezone                 False
Job                      False
Children                  True
Age                       True
Education                False
Employment               False
Income                    True
Marital                  False
Gender                   False
ReAdmis                  False
VitD_levels              False
Doc_visits               False
Full_meals_eaten         False
VitD_supp                False
Soft_drink                True
Initial_admin            False
HighBlood                False
Stroke                   False
Complication_risk        False
Overweight                True
Arthritis                False
Diabetes                 False
Hyperlipidemia           False
BackPain                 False
Anxiety                   True
Allergic_rhinitis        False
Reflux_esophagitis       False
Asthma                   False
Services                 False
Initial_days              True
TotalCharge              False
Additional_charges       False
Survey_TimelyAdmin       False
Survey_TimelyTreatment   False
Survey_TimelyVisits      False
Survey_Reliability       False
Survey_Options           False
```

```
Survey_HoursTreatment      False
Survey_CourteousStaff      False
Survey_ActiveListening     False
dtype: bool
```

```
# Count of missing values by column
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
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
Survey_TimelyAdmin            0
```

```
Survey_TimelyTreatment        0
Survey_TimelyVisits           0
Survey_Reliability            0
Survey_Options                0
Survey_HoursTreatment         0
Survey_CourteousStaff         0
Survey_ActiveListening        0
dtype: int64
```

```
# Store rows with missing values in new variable
rows_with_missing_values = df.isnull().any(axis=1)
df[rows_with_missing_values]
```

| | CaseOrder | Customer_id | Interaction | UID | City | State | County | Zip | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | C412403 | 8cd49b13-f45a-4b47-a2bd-173ffa932c2f | 3a83ddb66e2ae73798bdf1d705dc0932 | Eva | AL | Morgan | 35621 | 34.3 |
| **2** | 2 | Z919181 | d2450b70-0337-4406-bdbb-bc1037f1734c | 176354c5eef714957d486009feabf195 | Marianna | FL | Jackson | 32446 | 30.8 |
| **3** | 3 | F995323 | a2057123-abf5-4a2c-abad-8ffe33512562 | e19a0fa00aeda885b8a436757e889bc9 | Sioux Falls | SD | Minnehaha | 57110 | 43.5 |
| **4** | 4 | A879973 | 1dec528d-eb34-4079-adce-0d7a40e82205 | cd17d7b6d152cb6f23957346d11c3f07 | New Richland | MN | Waseca | 56072 | 43.8 |
| **5** | 5 | C544523 | 5885f56b-d6da-43a3-8760-83583af94266 | d2f0425877b10ed6bb381f3e2579424a | West Point | VA | King William | 23181 | 37.5 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **9987** | 9987 | Z630066 | 1ed0ed27-4965-4252-85ea-dd7ed73bd51a | f132eca4af3b1c955d89a213096ef88a | Perry | IA | Dallas | 50220 | 41.8 |
| **9991** | 9991 | M07341 | 9b73f4cb-3945-41c1-9a38-129fcecde3a0 | 4f83c32e349fa29482f338ed25896f01 | Crosby | MS | Wilkinson | 39633 | 31.2 |
| **9996** | 9996 | B863060 | a25b594d-0328-486f-a9b9-0567eb0f9723 | 39184dc28cc038871912ccc4500049e5 | Norlina | NC | Warren | 27563 | 36.4 |

```
# Rows that have any duplicates within the dataset
data_duplicates = df.loc[df.duplicated()]
print(data_duplicates)
```

```
Empty DataFrame
Columns: [CaseOrder, Customer_id, Interaction, UID, City, State, County, Zip, Lat, Lng, Popul
ation, Area, Timezone, Job, Children, Age, Education, Employment, Income, Marital, Gender, Re
Admis, VitD_levels, Doc_visits, Full_meals_eaten, VitD_supp, Soft_drink, Initial_admin, HighB
lood, Stroke, Complication_risk, Overweight, Arthritis, Diabetes, Hyperlipidemia, BackPain, A
nxiety, Allergic_rhinitis, Reflux_esophagitis, Asthma, Services, Initial_days, TotalCharge, A
dditional_charges, Survey_TimelyAdmin, Survey_TimelyTreatment, Survey_TimelyVisits, Survey_Re
liability, Survey_Options, Survey_HoursTreatment, Survey_CourteousStaff, Survey_ActiveListeni
ng]
Index: []

[0 rows x 52 columns]
```

```
# Calculate the standard deviation of the numeric fields
data_std = df_stats.std()
print(data_std)
```

```
Zip                       27469.588208
Lat                           5.403085
Lng                          15.205998
Population                14824.758614
Children                      2.155427
Age                          20.659182
Income                    28664.861050
VitD_levels                   6.723277
Doc_visits                    1.045734
Full_meals_eaten              1.008117
VitD_supp                     0.628505
Overweight                    0.454186
Anxiety                       0.467389
Initial_days                 26.287050
TotalCharge                3377.558136
Additional_charges         6542.601544
Survey_TimelyAdmin            1.031966
Survey_TimelyTreatment        1.034825
Survey_TimelyVisits           1.032755
Survey_Reliability            1.036282
Survey_Options                1.030192
Survey_HoursTreatment         1.032376
Survey_CourteousStaff         1.021405
Survey_ActiveListening        1.042312
dtype: float64
```

```
# Then print count of null values now
df_stats['Age'] = df['Age'].fillna(df['Age'].mean())
df_stats['Income'] = df['Income'].fillna(df['Income'].mean())
df_stats['Overweight'] = df['Overweight'].fillna(df['Overweight'].mean())
df_stats['Initial_days'] = df['Initial_days'].fillna(df['Initial_days'].mean())
data = df_stats.isnull().sum()
print(data)
```

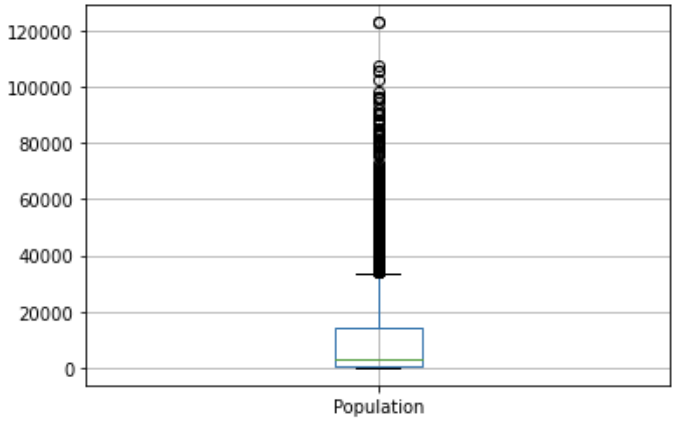
```
City                       0
State                      0
County                     0
Zip                        0
Lat                        0
Lng                        0
Population                 0
Area                       0
Timezone                   0
Children                2588
Age                        0
Education                  0
Employment                 0
Income                     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                 0
Arthritis                  0
Diabetes                   0
Hyperlipidemia             0
BackPain                   0
Anxiety                  984
Allergic_rhinitis          0
Reflux_esophagitis         0
Asthma                     0
Services                   0
Initial_days               0
TotalCharge                0
Additional_charges         0
Survey_TimelyAdmin         0
Survey_TimelyTreatment     0
Survey_TimelyVisits        0
Survey_Reliability         0
Survey_Options             0
Survey_HoursTreatment      0
Survey_CourteousStaff      0
Survey_ActiveListening     0
dtype: int64
```

### 2.4.7 Identifying Outliers

#### 2.4.7.1 Population

Check for outliers using boxplots

```
import matplotlib.pyplot as plt
df.boxplot(['Population'])
plt.savefig('uc_pyplot.jpg')
```



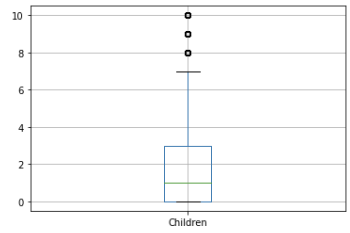

Even though the populations are accurate, the outliers may affect the effectiveness of the model, so this field will be standardized

```
df['Population'] = (df['Population'] - df['Population'].mean()) / df['Population'].std(
print(df['Population'])
```

```
1       -0.473145
2        0.090237
3        0.482959
4       -0.526366
5       -0.315570
           ...
9996    -0.350984
9997    -0.587818
9998    -0.636318
9999    -0.653923
10000    2.128787
Name: Population, Length: 10000, dtype: float64
```

### 2.4.7.2 Children

```
df.boxplot(['Children'])
plt.savefig('uc_pyplot.jpg')
```




Perform zscore on these outliers.

```
data = (df['Children'] - df['Children'].mean()) / np.std(df['Children'])
```

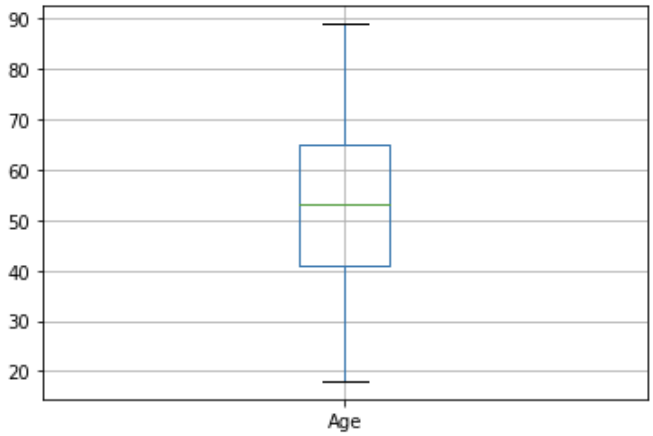
```
data
```

```
1       -0.268130
2        0.697755
3        0.697755
4       -0.751072
5       -0.751072
           ...
9996    -0.751072
9997     1.180697
9998     0.697755
9999     0.697755
10000    3.112467
Name: Children, Length: 10000, dtype: float64
```

The p-value equivalent to the z-score indicates that the field will remain as-is. P-value was above .05

#### 2.4.7.3 Age

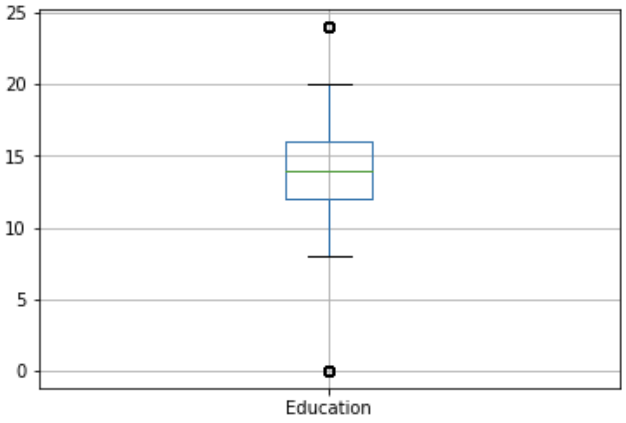
```
df.boxplot(['Age'])
plt.savefig('uc_pyplot.jpg')
```




This field has no outliers

#### 2.4.7.4 Education

```
df.boxplot(['Education'])
plt.savefig('uc_pyplot.jpg')
```



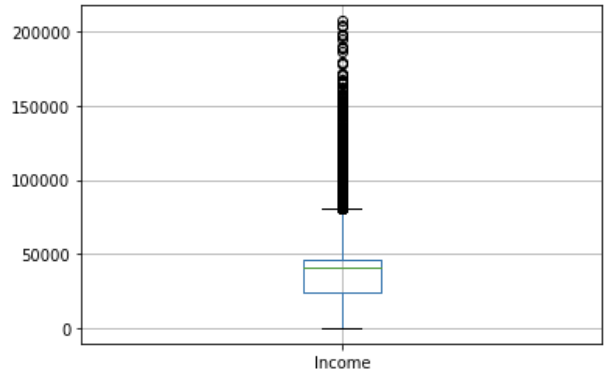

This field will be standardized due to extremely low and extremely high outliers

```
df['Education'] = (df['Education'] - df['Education'].mean()) / df['Education'].std()
print(df['Education'])
```

```
1       -0.197079
2        0.126479
3        0.126479
4       -0.520637
5       -0.520637
           ...
9996     0.773594
9997    -0.520637
9998    -0.520637
9999     0.773594
10000   -0.520637
Name: Education, Length: 10000, dtype: float64
```

### 2.4.7.5 Income

```
df.boxplot(['Income'])
plt.savefig('uc_pyplot.jpg')
```




```
z = (df['Income'] - df['Income'].mean()) / df['Income'].std()
data = df.loc[(z > 3) | (z < -3)]['Income']
data
```

```
31      167105.10
37      122615.82
59      132963.95
64      128855.39
78      135288.23
           ...
9743    153662.04
9863    121276.57
9907    138512.91
9911    119679.13
9956    122291.51
Name: Income, Length: 173, dtype: float64
```

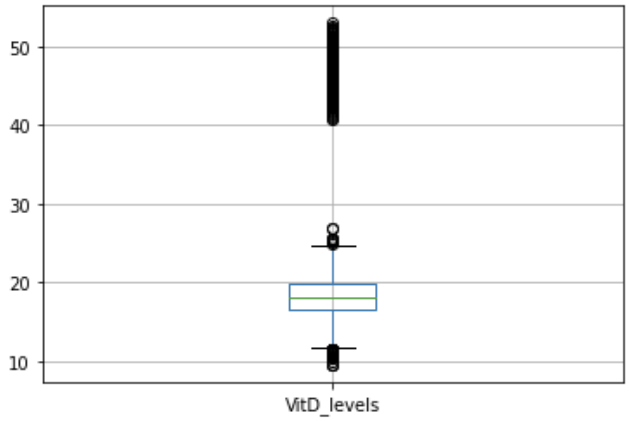
These values vary dramatically from the mean, so they will be standardized

```
df['Income'] = (df['Income'] - df['Income'].mean()) / df['Income'].std()
print(df['Income'])
```

```
1        1.852284
2        0.254045
3       -1.049458
4       -0.029857
5       -1.578344
           ...
9996     0.220353
9997    -1.024828
9998     1.022094
9999    -0.433302
10000    0.892081
Name: Income, Length: 10000, dtype: float64
```

### 2.4.7.6 VitD_levels

```
df.boxplot(['VitD_levels'])
plt.savefig('uc_pyplot.jpg')
```




Check for correlation between VitD_levels and VitD_supp

```
from scipy.spatial.distance import cosine
print(1-cosine(df['VitD_levels'], df['VitD_supp']))
```

```
0.5092760439735684
```

```
high_VitD = np.where((df['VitD_levels']>30) & (df['VitD_supp']>1))
high_VitD
```

```
(array([  94,  837, 1069, 1082, 1379, 1445, 1812, 1972, 2089, 2344, 2372,
        2495, 2669, 3042, 3713, 4002, 4028, 4199, 4596, 4716, 4776, 4890,
        5044, 5128, 5907, 6216, 6560, 6938, 7185, 8088, 8156, 8410, 8744,
        8872, 9095, 9172, 9260, 9263, 9365, 9459, 9483, 9503, 9790, 9933]),)
```

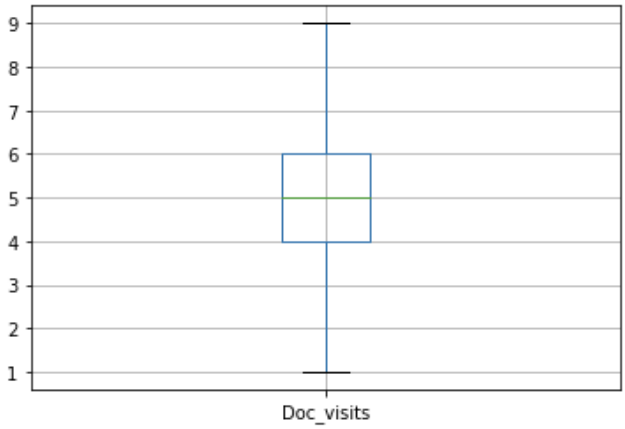
```
print(1-cosine(df['Age'], df['Overweight']))
```

```
0.8086840545032804
```

These outliers are plausible, given that patients over 50 and/or overweight are at higher risk for health issues that would require calcium supplements. This would require more vitamin D to absorb the calcium. These are natural outliers for the dataset and are imperative for analysis. The conclusion can be drawn that patients would have a higher readmittance if proper supplementation is not administered. These fields will remain.

#### 2.4.7.7 Doc_visits

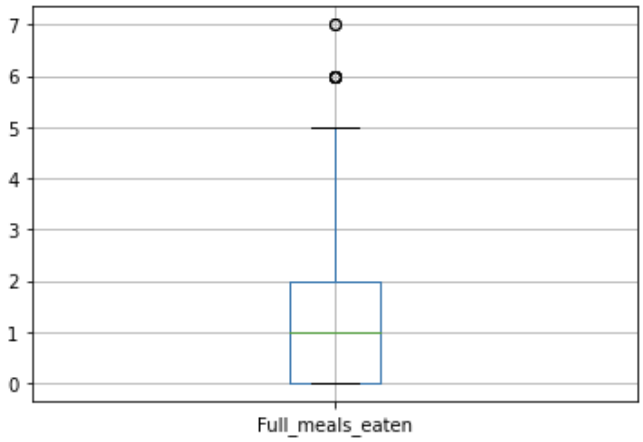
```
df.boxplot(['Doc_visits'])
plt.savefig('uc_pyplot.jpg')
```




There are no outliers within this field

### 2.4.7.8 Full Meals Eaten

```
df.boxplot(['Full_meals_eaten'])
plt.savefig('uc_pyplot.jpg')
```




```
z = (df['Full_meals_eaten'] - df['Full_meals_eaten'].mean()) / df['Full_meals_eaten'].s
data = df.loc[(z > 3) | (z < -3)]['Full_meals_eaten']
data.mean()
```

```
5.303030303030303
```

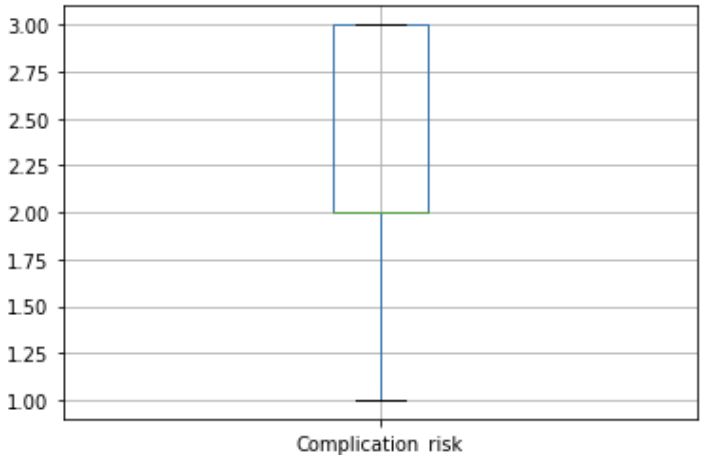
This field will be standardized.

```
df['Full_meals_eaten'] = (df['Full_meals_eaten'] - df['Full_meals_eaten'].mean()) / df[
df['Full_meals_eaten']
```

```
1       -0.993337
2        0.990560
3       -0.001389
4       -0.001389
5       -0.993337
           ...
9996     0.990560
9997    -0.993337
9998     0.990560
9999     0.990560
10000   -0.993337
Name: Full_meals_eaten, Length: 10000, dtype: float64
```

#### 2.4.7.9 Complication_risk

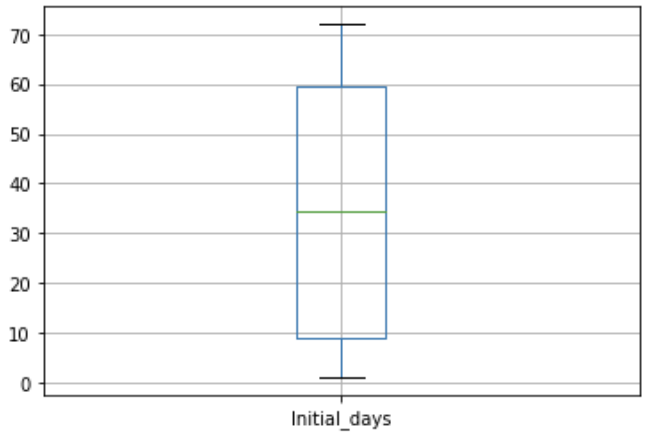
```
df.boxplot(['Complication_risk'])
plt.savefig('uc_pyplot.jpg')
```




This field has no outliers.

#### 2.4.7.10 Initial_days

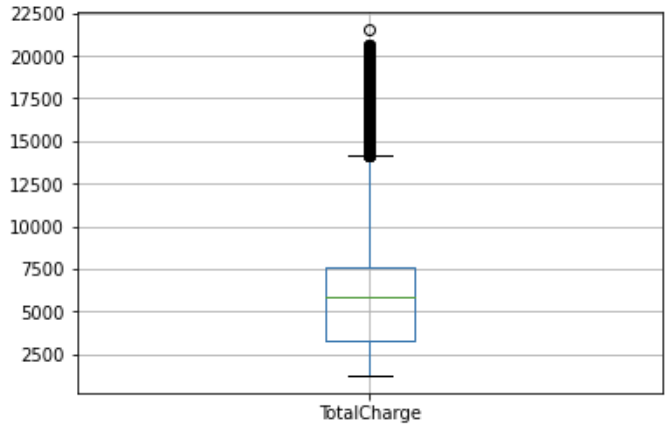
```
df.boxplot(['Initial_days'])
plt.savefig('uc_pyplot.jpg')
```




This field has no outliers.

#### 2.4.7.11 Total Charge

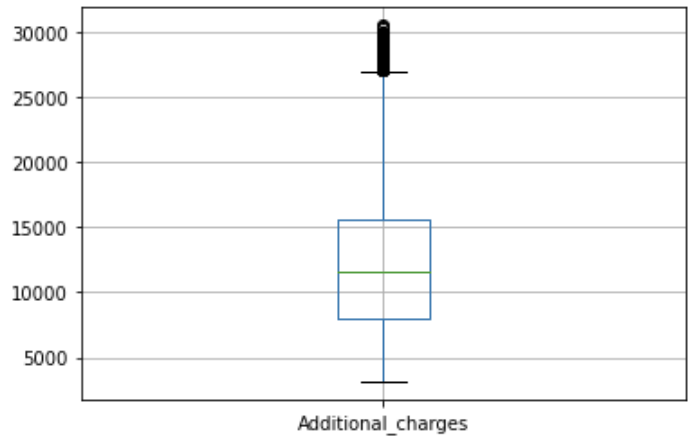
```
df.boxplot(['TotalCharge'])
plt.savefig('uc_pyplot.jpg')
```




There are outliers within the TotalCharge field, however there seems to be a correlation between Readmission and total charge, so this data will stay as-is.

#### 2.4.7.12 Additional Charges

```
df.boxplot(['Additional_charges'])
plt.savefig('uc_pyplot.jpg')
```



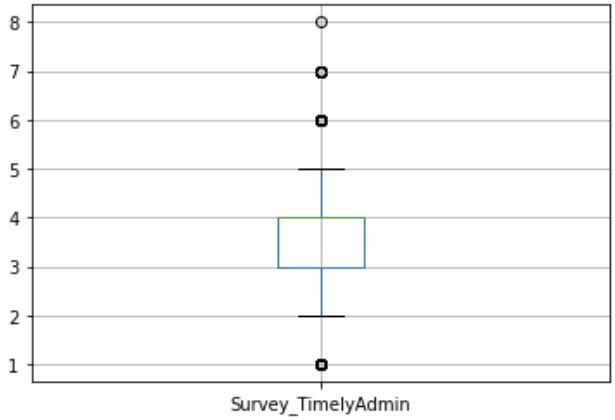

This field will be standardized.

```
df['Additional_charges'] = (df['Additional_charges'] - df['Additional_charges'].mean())
df['Additional_charges']
```

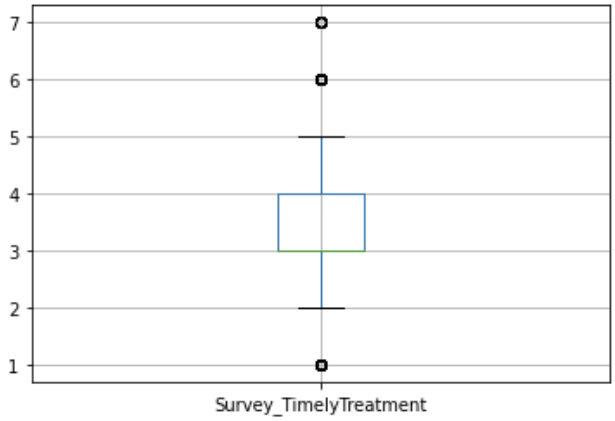
```
1         0.764967
2         0.715078
3         0.698600
4         0.009004
5        -1.408920
            ...
9996     -0.612430
9997      2.380188
9998      0.358678
9999     -0.787584
10000    -0.197374
Name: Additional_charges, Length: 10000, dtype: float64
```

## 2.4.7.13 Survey Results

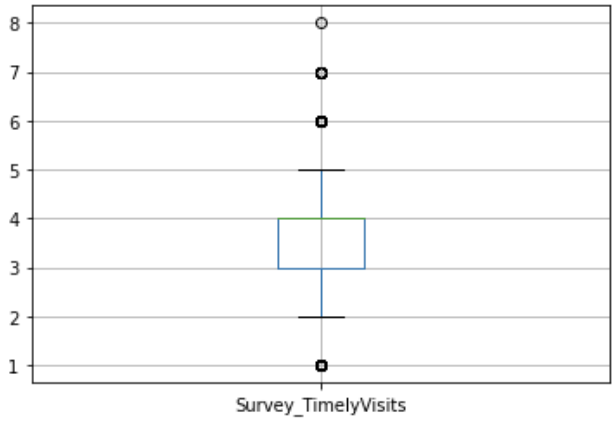
```
df.boxplot(['Survey_TimelyAdmin'])
plt.savefig('uc_pyplot.jpg')
```



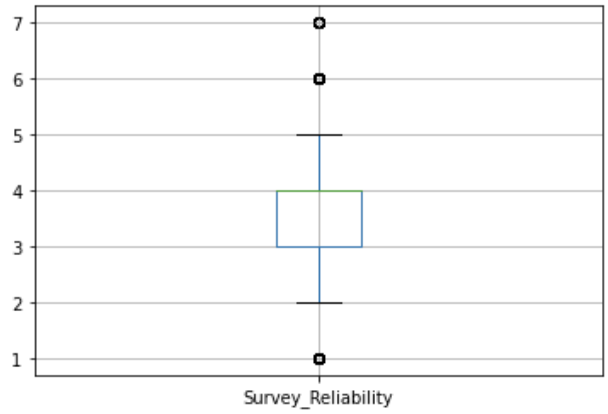

```
df.boxplot(['Survey_TimelyTreatment'])
plt.savefig('uc_pyplot.jpg')
```



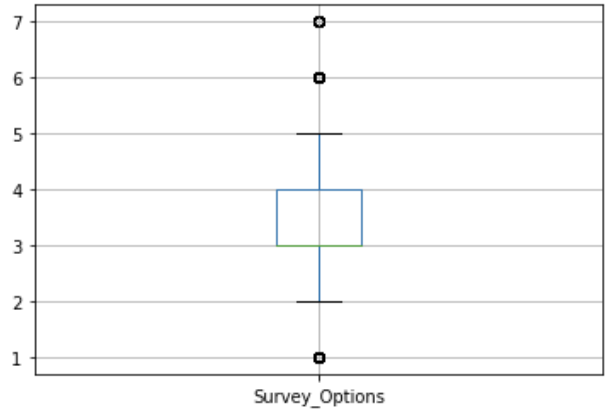

```
df.boxplot(['Survey_TimelyVisits'])
plt.savefig('uc_pyplot.jpg')
```



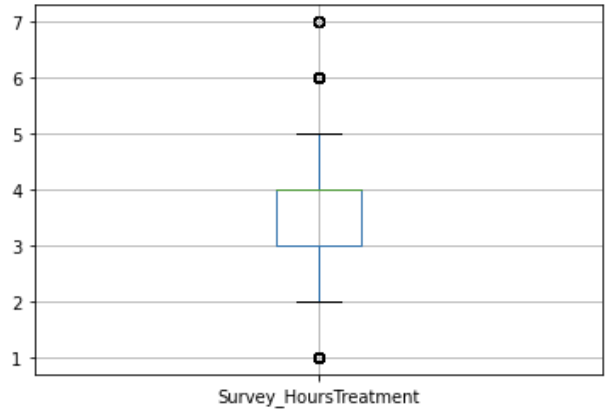
```
df.boxplot(['Survey_Reliability'])
plt.savefig('uc_pyplot.jpg')
```



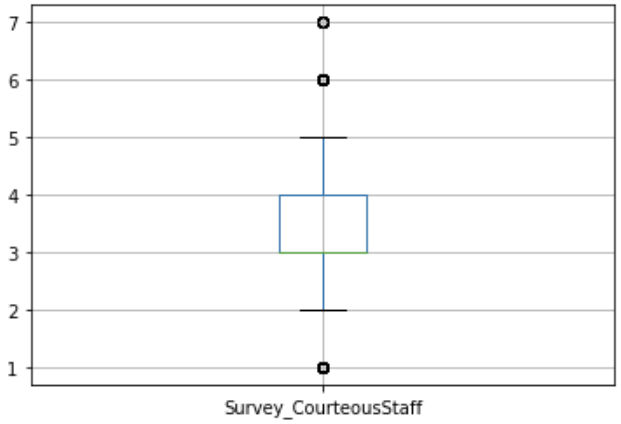

```
df.boxplot(['Survey_Options'])
plt.savefig('uc_pyplot.jpg')
```



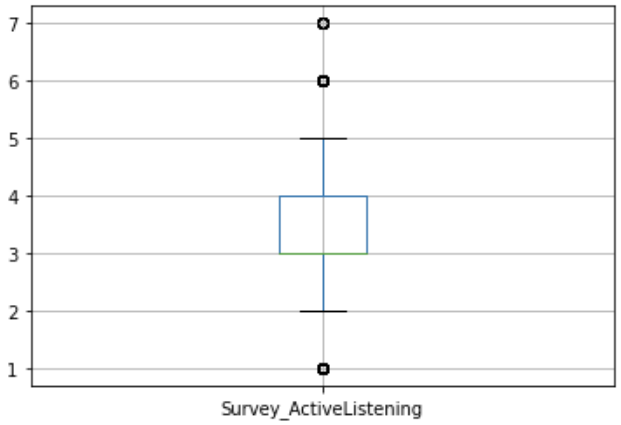

```
df.boxplot(['Survey_HoursTreatment'])
plt.savefig('uc_pyplot.jpg')
```



```
df.boxplot(['Survey_CourteousStaff'])
plt.savefig('uc_pyplot.jpg')
```




```
df.boxplot(['Survey_ActiveListening'])
plt.savefig('uc_pyplot.jpg')
```



```
z = (df['Survey_TimelyAdmin'] - df['Survey_TimelyAdmin'].mean()) / df['Survey_TimelyAdm
z.mean()
```

```
-2.415401212374491e-16
```

```
= (df['Survey_TimelyTreatment'] - df['Survey_TimelyTreatment'].mean()) / df['Survey_Tim
mean()
```

```
-3.064215547965432e-18
```

```
z = (df['Survey_TimelyVisits'] - df['Survey_TimelyVisits'].mean()) / df['Survey_TimelyV
z.mean()
```

```
-1.3541390231353034e-16
```

```
 = (df['Survey_Reliability'] - df['Survey_Reliability'].mean()) / df['Survey_Reliabilit
.mean()
```

```
3.177458296477198e-16
```

```
z = (df['Survey_Options'] - df['Survey_Options'].mean()) / df['Survey_Options'].std()
z.mean()
```

```
-2.879807503575193 7e-16
```

```
z = (df['Survey_CourteousStaff'] - df['Survey_CourteousStaff'].mean()) / df['Survey_Cou
z.mean()
```

```
2.117972464077411e-16
```

```
z = (df['Survey_HoursTreatment'] - df['Survey_HoursTreatment'].mean()) / df['Survey_Hou
z.mean()
```

```
7.278622149442526e-17
```

```
z = (df['Survey_ActiveListening'] - df['Survey_ActiveListening'].mean()) / df['Survey_A
z.mean()
```

```
4.779732165616224e-16
```

The fields will stay as-is.

## 2.5 Principal Component Analysis

```
df.head(5)
```

| | City | State | County | Zip | Lat | Lng | Population | Area | Timezone | Child |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | Eva | 1 | Morgan | 35621 | 34.34960 | -86.72508 | -0.473145 | 2 | America/Chicago | |
| **2** | Marianna | 10 | Jackson | 32446 | 30.84513 | -85.22907 | 0.090237 | 3 | America/Chicago | |
| **3** | Sioux Falls | 43 | Minnehaha | 57110 | 43.54321 | -96.63772 | 0.482959 | 2 | America/Chicago | |
| **4** | New Richland | 24 | Waseca | 56072 | 43.89744 | -93.51479 | -0.526366 | 2 | America/Chicago | |
| **5** | West Point | 48 | King William | 23181 | 37.59894 | -76.88958 | -0.315570 | 1 | America/New_York | |

5 rows × 56 columns

I will be excluding the target field(ReAdmis), the qualitative fields(City and County), and the redundant fields(State,Lat,Lng,Timezone)

```
df_numeric = df.drop(columns = ['ReAdmis','City','County','State','Lat','Lng','Timezone
print(df_numeric.columns)
```

```
Index(['Zip', 'Population', 'Area', 'Children', 'Age', 'Education', 'Income',
       'VitD_levels', 'Doc_visits', 'Full_meals_eaten', 'VitD_supp',
       'Soft_drink', 'HighBlood', 'Stroke', 'Complication_risk', 'Overweight',
       'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety',
       'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services',
       'Initial_days', 'TotalCharge', 'Additional_charges',
       'Survey_TimelyAdmin', 'Survey_TimelyTreatment', 'Survey_TimelyVisits',
       'Survey_Reliability', 'Survey_Options', 'Survey_HoursTreatment',
       'Survey_CourteousStaff', 'Survey_ActiveListening', 'Full Time',
       'Part Time', 'Retired', 'Student', 'Unemployed', 'Female', 'Male',
       'Elective Admission', 'Emergency Admission', 'Observation Admission'],
      dtype='object')
```

```
from sklearn.decomposition import PCA
pcs_names = []
for i, col in enumerate(df.columns):
    pcs_names.append('PC' + str(i+1))

pca = PCA(n_components=df.shape[1])
data_pca=pd.DataFrame(pca.fit_transform(df),columns=pcs_names)
print(data_pca)
```

```
pcs_names = []
for i, col in enumerate(df_numeric.columns):
    pcs_names.append('PC' + str(i+1))


#data_pca=pd.DataFrame(pca.fit_transform(df),columns=pcs_names)
print(pcs_names)
```

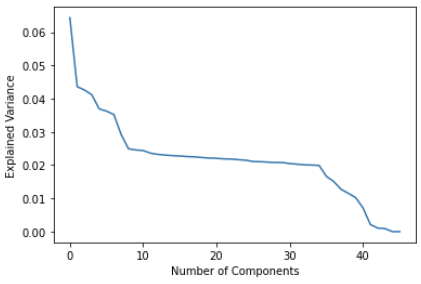
```
['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11', 'PC12'
, 'PC13', 'PC14', 'PC15', 'PC16', 'PC17', 'PC18', 'PC19', 'PC20', 'PC21', 'PC22', 'PC2
3', 'PC24', 'PC25', 'PC26', 'PC27', 'PC28', 'PC29', 'PC30', 'PC31', 'PC32', 'PC33', 'P
C34', 'PC35', 'PC36', 'PC37', 'PC38', 'PC39', 'PC40', 'PC41', 'PC42', 'PC43', 'PC44',
'PC45', 'PC46']
```

```
pca.fit(data_norm)
```

```
PCA(n_components=46)
```

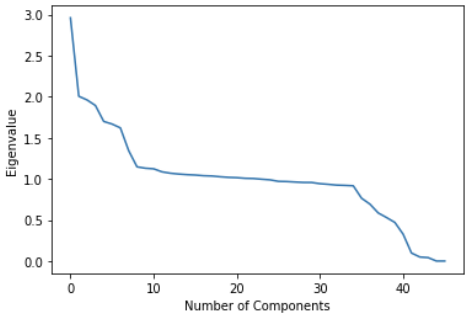
```
medical_pca = pd.DataFrame(pca.transform(data_norm),columns = pcs_names)
```

```
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.show();
```




```
cov_matrix = np.dot(data_norm.T, data_norm) / df_numeric.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for eigenvector i
```

```
plt.plot(eigenvalues)
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalue')
plt.show();
```




```
for pc, var in zip(pcs_names, np.cumsum(pca.explained_variance_ratio_)):
    print(pc, var)
```

```
PC1 0.06434887178807752
PC2 0.10795543420844103
PC3 0.15057259981512067
PC4 0.19173142782368205
PC5 0.22867159792901165
PC6 0.26492591604761184
PC7 0.3001587315951254
PC8 0.3293621624109791
PC9 0.3542953849257158
PC10 0.37885628389223025
PC11 0.40326642153311315
PC12 0.42687706783305573
PC13 0.45011850394953995
PC14 0.4731432717422002
PC15 0.4960087771011487
PC16 0.5187669541246129
PC17 0.5413482646238477
PC18 0.5638532201021936
PC19 0.586154157972328
PC20 0.608299934258612
PC21 0.6303605893179738
PC22 0.652254220520169
PC23 0.6740947478645255
PC24 0.6957444683639841
PC25 0.7172304720740024
PC26 0.738315693686425
PC27 0.7593516589311435
PC28 0.7802645333911438
PC29 0.8010628914680038
PC30 0.8218489006602755
PC31 0.8423188653549483
PC32 0.8626156300749972
PC33 0.8827122948377152
PC34 0.9027322843435649
PC35 0.9226491124451223
PC36 0.9392661957140251
PC37 0.9543370152978192
PC38 0.967066854658399
PC39 0.9785817653704804
PC40 0.9888006798412152
PC41 0.99589747834665
```

```
otation = pd.DataFrame(pca.components_.T, columns = pcs_names, index = df_numeric.colum
rint(rotation)
```

```
Area                0.003011  0.019498  0.025447  0.011031  0.009498
Children           -0.001914 -0.002015  0.009142 -0.007864  0.024458
Age                 0.000208  0.170151  0.204727 -0.398495 -0.010920
Education           0.004284  0.000806 -0.013959  0.014856  0.015679
Income              0.002405  0.002612 -0.016295 -0.005645 -0.042982
VitD_levels         0.008969  0.383699  0.239779  0.278228 -0.039034
Doc_visits         -0.007130 -0.004302  0.013517 -0.012523 -0.009137
Full_meals_eaten    0.000467  0.014302 -0.010680 -0.033090  0.031515
VitD_supp           0.004713  0.036825  0.003260  0.001459 -0.003853
Soft_drink         -0.006803 -0.001713  0.007251  0.002288  0.045423
HighBlood           0.004248  0.176398  0.200299 -0.431834 -0.010382
Stroke              0.002830 -0.001887  0.009310 -0.041139 -0.003876
Complication_risk  -0.012714  0.035628  0.028696 -0.025693 -0.015225
Overweight         -0.000599 -0.009691 -0.004588 -0.032626  0.016389
Arthritis           0.014093  0.026624  0.004207  0.001773 -0.001492
Diabetes            0.003103 -0.014743 -0.011760 -0.015251  0.006919
Hyperlipidemia     -0.017254  0.020090 -0.010039  0.014572  0.024564
BackPain            0.012995  0.010153  0.035804 -0.000182 -0.010470
Anxiety             0.000607  0.018995  0.033483  0.004721 -0.016952
Allergic_rhinitis  -0.004994  0.017191  0.014604 -0.019148  0.010043
Reflux_esophagitis -0.006287 -0.009784  0.019522  0.025609 -0.007431
Asthma              0.010657  0.006046 -0.002607 -0.023206  0.003325
```

```
adings = pd.DataFrame(pca.components_.T, columns = pcs_names,index = df_numeric.columns

adings
```

| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 |
|---|---|---|---|---|---|---|---|
| **Zip** | 0.006944 | -0.001683 | 0.010453 | 0.001572 | -0.003455 | -0.012596 | 0.009333 |
| **Population** | -0.010544 | 0.018319 | -0.000059 | 0.015566 | 0.015860 | 0.006873 | 0.020359 |
| **Area** | 0.003011 | 0.019498 | 0.025447 | 0.011031 | 0.009498 | -0.001183 | -0.004759 |
| **Children** | -0.001914 | -0.002015 | 0.009142 | -0.007864 | 0.024458 | -0.006855 | 0.015713 |
| **Age** | 0.000208 | 0.170151 | 0.204727 | -0.398495 | -0.010920 | -0.050113 | 0.053317 |
| **Education** | 0.004284 | 0.000806 | -0.013959 | 0.014856 | 0.015679 | 0.006920 | 0.007269 |

```
medical_reduced = medical_pca.iloc[ : , 0:3]
print(medical_reduced)
```

```
           PC1       PC2       PC3
0     1.555429  0.600215 -1.015412
1     0.350520 -0.889092  1.343255
2     0.236100 -1.646615  0.653577
3    -2.352985 -0.104929 -1.875978
4     2.431364 -2.912325 -0.953335
...        ...       ...       ...
9995  2.036910  1.033226 -1.080176
9996  0.771279  2.470258  0.012432
9997  1.916401 -0.604135  1.064580
9998 -0.841379  1.366140 -1.105025
9999 -0.612978 -0.148320  1.276939

[10000 rows x 3 columns]
```

# 3 Data Cleaning

## 3.1 Cleaning Findings

Some of the findings from this analysis include:

- There was quite a bit of data missing from the fields ‘Children’, ’Age’, ‘Income’, ‘Soft_drink’, ‘Overweight’, ‘Anxiety’, and ‘Initial_days’. Rationally, the fields ‘Soft_drink’, ‘Children’, and ‘Anxiety’ were defaulted to zero due to the assumption that these fields might not have been applicable to the patient. However, the fields ‘Age’, ‘Income’, ‘Overweight’, and ‘Initial_days’ seemed to be less likely to be non applicable and taking into account the fields’ variance and mean, these fields were defaulted to the fields’ mean.
- Some fields seemed to be either irrelevant or less meaningful, namely ‘Unnamed:0’, 'CaseOrder','Customer_id','Interaction','UID','Job', and ’Marital’, so they were not included in the final analysis so as to dictate the strongest correlating characteristics of readmission.
- The outliers found were either insignificant and hence mitigated or they were within a field that had a correlation to another field or fields and were left alone.

## 3.2 Justification of Mitigation Methods

Following the data cleaning process, almost all fields were converted to numeric, which allows for calculations to be processed easily. There were also no longer any NULL values remaining. Fields were mitigated with imputation using median values where it could be reasonably inferred that the variable would be applicable to the patient. Fields were mitigated with imputation using the number zero where it could be reasonably assumed that the variable may not be applicable to the patient. The ‘TotalCharge’ field has outliers, however there seems to be a correlation between the field ‘Readmision’ and ‘TotalCharge’ so it was left alone. The ‘VitD_levels’ field has outliers, however they seem plausible due to the fact that patients over 50 and/or overweight are at higher risk for health issues that would require calcium supplements. This would require more vitamin D to absorb the calcium. These are natural outliers for the dataset and were imperative for analysis. The conclusion that patients would have a higher readmittance if proper supplementation was not administered.

## 3.3 Summary of Outcomes

After cleaning the data the fields left were the daily total charge of the initial visit, the average charges for additional treatment and services during the initial visit, and the responses to the survey.

## 3.4 Mitigation Code

This can be found in the code provided above and the Panopto recording.

### 3.5 Clean Data

This can be found within the attached file ‘medical_clean.csv’.

### 3.6 Limitations

A limitation of this dataset would be that it was not fully complete. In this scenario, I did not have access to any sources for whom I could have requested assistance in gathering real world data to complete the dataset. Given this project outside of this course, contacting the medical chain associated with the dataset would have been a good first step to acquiring more information and/or reasons behind the missing data. Overall, the data could have fared better, had the data collector been present to explain the variables better.

Although a purely numeric dataset would be easier for a computer to understand and analyze its contents, another limitation would be that the cleansing process creates a new dataset that would require a dictionary for a human to be able to decipher it. So, if the dataset was passed along to a new analyst, without the help of the current analyst, it would be more difficult for the new analyst to manipulate the data without having to repeat the process that the previous analyst went through.

### 3.7 Impact of Limitations

Given that in a real world scenario the staff of the medical chain might be able to assist with the missing data, I find that the impact of the limitations to be very minimal. Even if the staff does not have an exact answer for the missing data, the utilization of follow-ups as a back-up measure to acquire the data. This limitation could also be rectified by implementing stricter data procurement methods, follow-ups and feedback. This could be addressed by utilizing electronic paperwork which can be coded to restrict the patient from proceeding through the paperwork without first filling out the current section.

Another impact would be that a purely numeric dataset might cause confusion and unnecessary waste of time between the transfer of data between teams, if not explained correctly. This could be addressed by having the initial analyst always in correspondence with the data analysis, however this would be very unrealistic due to position changes, different company analysis etc. Another way to address this would be to always have a well documented process, as well as saving the multiple versions of the dataset throughout the process. By doing this, the transfer of data would be less time consuming since new analysts would not need to guess how the data came to be in this format and new analysts would also be able to draw new conclusions from the original dataset as well.

## 4 PCA Application

### 4.1 Principal Components

In order to determine the principal components, I ran PCA using sklearn.decomposition and viewed the Scree plot. The PCA graphs indicated that the following fields were the most important. This was determined by their cos2 being over 0.4.

> TotalCharge, Additional_charges, Survey_HoursTreatment, Survey_TimelyVisits, Survey_TimelyAdmin, Survey_TimelyTreatment

### 4.2 Benefits

Hospitals could benefit from further analysis of these fields and researching further into the correlation between these fields and readmission rates. Future findings could provide valuable information on how to reduce readmission rates.

## 5 Supporting Documents

### 5.1 Video

This can be found within the attached file 'Panopto Recording'.

### 5.2 Sources for Third-Party Code

Himamsh, Viveka. "How to Create Dummy Variables in Python with Pandas?" *GeeksforGeeks*, 8 Oct. 2021, https://www.geeksforgeeks.org/how-to-create-dummy-variables-in-python-with-pandas/.

"Statsmodels.imputation.mice.micedata¶." Statsmodels, https://www.statsmodels.org/stable/generated/statsmodels.imputation.mice.MICEData.html.

### 5.3 Sources

Middleton, K. (2021). D206 Data Cleaning: Course of Study . Salt Lake City ; Western Governors University. blob:https://my.wgu.edu/019fb1b8-04fc-49b8-be85-637e763e69ae

Western Governors University. (n.d.). D206 Data Cleaning_Medical Data Considerations and Dictionary. Salt Lake City.