OCTOBER 22, 2023

D206 DATA CLEANING

PERFORMANCE ASSESSMENT

ROBERT PATTON
WESTERN GOVERNORS' UNIVERSITY
Rpatt33@wgu.edu

Table of Contents

Part I.

A: Research Question

For this performance assessment, I decided to use the medical data for cleaning. The data provides information on variables related to patient admission along with other key elements that help with patient identification. Healthcare is an ever-growing industry as illness, disease, and new medical findings are frequently presented in today's modern medicine. I currently work in healthcare and based on the information in the medical data file, I have decided to work with this dataset for my project. After reviewing the information in the CSV file, I have developed my research question as such:

Can we assume there is a correlation between high blood pressure and strokes? Due to this correlation, do these patients tend to be more frequently admitted on an emergent basis?

B: Variables

The medical dataset contains both qualitative and quantitative data types from a total of 53 columns/variables. A list of all the variables, their data types, and a description of the variables is as follows:

Unnamed: 0: There is no name for this variable, but it is numeric because it has numbers from 1-10000.
Case order: This variable is numerical or quantitative. It is numbered from 1-10000 and keeps the data organized by row.
Customer_id: This variable is qualitative because it includes a letter before a series of numbers, such as C412403, and gives a unique identifier to each patient in the dataset.
Interaction/UID: These two variables represent qualitative data since they contain both text and numbers. Their purpose is to also give unique patient identifiers based on services received by the hospital. For example, 3a83ddb66e2ae73798bdf1d705dc0932.
City/State/County: Each of these is its own variable in the dataset and a type of qualitative data. The data in these columns give information about the city, state, or county, for billing, that each patient lives in and therefore can be categorized nominally. Examples of these data are states as AL, FL, SD or cities like Eva, West Point, and Counties like Morgan, Waseca, and Jackson.

Zip: The zip code variable is a qualitative data type as it contains numeric values, such as 35621, that tell which location a patient lives in within a specific area. The zip code variable is another identifier for the location of patient residency for billing purposes.

Lat/Lng: Again, we see numerical data within these two separate variables in the dataset. Therefore, they would be considered continuous quantitative data types that also give the location of a patient's residence, via GPS, for billing purposes. For example, latitude data (Lat) would be positive numbers such as 34.3496, and longitude data (Lng) are represented by negative numbers such as -86.72508.

Population: Gathered via census data, this variable gives population size within a mile radius of the patient. This data type is considered a quantitative variable. For example, 2951, 11303, 17125.

Area: This data type is an example of qualitative data that can be categorized. Area type is based on whether the patient lives in a suburban, urban, or rural area.

Timezone: This is a qualitative nominal data type that can be categorized based on the region the patient lives within the United States. The data was derived from initial sign-up by the patient themselves. For example, America/Chicago or America/Denver.

Job: This data type is categorical/qualitative, showing the job type of the patient or the insurance holder at the time admission information was filled out.

Children: This represents discrete quantitative data and provides information on how many children reside in the patient's residence. For example, 1, 3, or 7 children.

Age: This data is also an example of quantitative data and is reported by the patient at the time of admission. The data is listed as 53, 51, or 22, as an example.

Education: This data describes the highest level of education as reported by the patient. It is a qualitative data type that is ordinal. Examples of such data would be: Some college, less than 1 year or some college, 1 or more years, no degree, or Doctorate.

Employment: This is a qualitative data type, reported by the patient, that tells if the patient has full-time, part-time, retired, or unemployed employment status.

Income: Reported at the time of admission, this data type is quantitative and represents the annual income of the patient or the insurance holder. An example from the dataset would be values such as: 86575.93, 14370.14, or 46805.99.

Marital: The marital status is also reported at the time admission information is taken. It represents the marital status of the patient or the insurance holder and is an ordinal qualitative data type. Examples of this data type would be married, widowed, never married, divorced, or separated.

Gender: Qualitative data that is reported by the patient and describes how they self-identify, such as male, female, or prefer not to answer.

ReAdmis: An ordinal, qualitative variable in the dataset, that provides information on whether the patient had been readmitted within a month after discharge. Examples for the data are yes or no.

VitD_levels: This is a quantitative data type of patient's vitamin d levels such as, 17.80233049, that is interpreted through measured levels of ng/ml.

Doc_visits: A discrete quantitative data type that provides information on how many times a patient's primary physician visited them during initial admission. Examples are 6, 4, 5, 7, etc.

Full_meals_eaten: Another set of quantitative data that describes how many full meals a patient had during their hospital admission. The data analyst should note that partial meals eaten count as 0, for example, and other examples for full meals would be whole numbers such as 2 or 3 meals.

VitD_supp: Quantitative data that provides information on how many vitamin D supplements the patient was given during hospital admission, expressed in whole numbers such as 0, 1, 3, and so forth.

Soft_drink: Ordinal, qualitative data that provides information on whether a patient drinks three or more soft drink beverages daily. Examples of this variable are either yes, no, or NA.

Initial_admin: Categorical data, that is nominal, and expresses the primary route of admission to the hospital. Examples of this variable are emergency admission, elective admission, and observation admission.

High blood: Qualitative data that expresses whether the patient has high blood pressure or not. Values are either yes or no.

Stroke: This variable is also qualitative ordinal data that expresses whether the patient has had a stroke or not. Values are either yes or no.

Complication_risk: This variable is ordinal qualitative data. It describes the level of complication risk for the patient during the primary assessment. Risk is assessed as high, medium, or low.

Overweight: Ordinal qualitative data that has been re-expressed into values such as 0, 1, or NA. The data is determined based on the patient's age, height, and gender and answered as either yes or no. Yes and no values are re-expressed into 0 or 1.

Arthritis, Diabetes, Hyperlipidemia, Backpain, Anxiety, Allergic_rhinitis, Reflux_esophagitis and Asthma: Each of these individual variables is of the ordinal, qualitative data type. Patient assessment determines whether they have any of these medical conditions based on a yes or no answer from the patient.

Services: The services variable is a nominal, qualitative data type. The information for this variable describes the primary medical service each patient received during their hospital admission. Value examples are blood work, CT scan, MRI, or intravenous.

Initial_days: Data variable that describes the length of initial admission to the hospital in days. For example, 10.58576971. This data type is quantitative.

Total Charge: A data variable that provides details on how much each patient was charged daily and is an average divided by the total cost of hospital admission. The data does not include any specialized treatments. This is a quantitative data type expressed in values such as 3191.048774.

Additional_charges: Charges billed to the patient for miscellaneous services such as medicines or anesthesiology. This is also quantitative data and is expressed in values such as 17939.40342.

Item 1-8: Items 1-8 are ordinal categorical variables. These variables in the dataset are based on a survey, presented to patients, that rate the importance of various factors related to their hospital experience. The survey asks patients to rate each variable from 1 (most important) to 8 (least important). Each variable is re-expressed into numerical values 1-8 for the dataset. The examples of survey questions are as follows, beginning with item 1: Timely admission, timely treatment, timely visits, reliability, options, hours of treatment, courteous staff, and evidence of active listening from a doctor.

Part II: Data-Cleaning Plan

C1: Methods for detecting anomalies in the medical dataset

Step 1. Download all the required packages for Python into the Jupyter integrated development environment. This step is going to allow me to visualize and analyze the data using packages such as NumPy and pandas.

Step 2. Using the read_csv() function, I can import the medical data csv file into my IDE. In doing so, I can examine the data features, such as the number of rows, columns, data types, and any potential abnormalities.

Step 3. Detect and visualize any duplicates using the .duplicated() command to look for any duplicates in the dataset.

Step 4. After determining there are no duplicates, I then use the .info() command, I can see where null values may be present and proceed to deal with those missing values. I then use the isnull().sum() function to gather information on how many null functions are present in each variable of the dataset.

Step 5. Detect outliers in the dataset using boxplots and z-score calculations. I used the seaborn package and sns.boxplot() command to create boxplots that allowed me to visualize which variables had outliers present and the scipy package to calculate z-scores and identify how many outliers were present.

C2: Justify Approach

When cleaning a dataset, it is important to detect and treat any anomalies so that the integrity of the data is reliable. A dataset with duplicate variables, missing values, and over-abundant outliers, can produce results that compromise data quality. When looking for duplicate values, the .duplicated() function returns true statements that allow me to see how many duplicated values there are, if any, needing correction. Using the .info() and isnull().sum() functions return information on the variables that include null values and the exact number of missing values. With this information, I can then deal with the missing values and create visualizes that give me further insight on how to treat them. Visualization tools are helpful in determining the extent of dealing with outliers in the dataset. The seaborn package allows me to use the sns.boxplot() function to create boxplots with minimum and maximum values, where everything outside of those values is considered an outlier. The Scipy package allows me to calculate z-scores, which I can then visualize in histograms, and get an exact count on outliers in the data set needing attention. Each of these steps and tools is an essential component to organizing and cleaning datasets for improving data integrity.

C3: Language Used

For this performance assessment, I decided to use the Python programming language. I chose Python because it is the programming language I intend to work with most often when I finish the MSDA program. It is also a great program for handling large datasets and is one of the most popular coding languages. Because Python has a lot of pre-loaded packages, it is an easy task to import those packages and use the pre-programmed functions within them for managing data. The packages I used for this performance assessment were pandas for loading in data, NumPy for mathematical and array management, matplotlib, seaborn, scipy for calculating z-scores, and sklearn for principal component analysis.

C4: Detection Code

(Please see copy of code attached)

```
# Import all libraries and packages
        import pandas as pd
        import numpy as np
        %matplotlib inline
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.decomposition import PCA
#Import csv file
        mdf = pd.read_csv('/Users/robertpatton/Desktop/D206 /medical_raw_data.csv')
#Examine data types
        mdf.info()
#Determine if there are any duplicate values
        print(mdf.duplicated())
#Get the count of total duplicate values
        print(mdf.duplicated().value_counts())
#Look for null/missing values in dataset
        mdf.isnull().sum()
#Examine distribution of missing values for numerical quantitative variables
        mdf[['Children','Age','Income','Initial_days']].hist()
#Find all numerical values
        mdf_num_data=mdf.select_dtypes(include=[np.number])
        mdf_num_data.info()
#Use boxplot to find outliers in qualitative data types
        plt.subplots(figsize=(30,10))
        boxplot=sns.boxplot(data=mdf_num_data, width=0.4)
#Import scipy package for calculation of z-scores
        from pandas import DataFrame
        import scipy.stats as stats
#Calculate z-score for population
        mdf['Population_z_score']=stats.zscore(mdf['Population'])
        mdf[['Population', 'Population_z_score']].head()
#Get total number of outliers for population
        population_z_score=mdf.query('Population_z_score > 3')
        population_z_score.info()
#Calculate z-score for income
        mdf['Income_z_score']=stats.zscore(mdf['Income'])
        mdf[['Income', 'Income_z_score']].head()
#Get total number of outliers for income
        income_z_score=mdf.query('Income_z_score > 3')
        income_z_score.info()
#Calculate z-score for TotalCharge
        mdf['TotalCharge_z_score']=stats.zscore(mdf['TotalCharge'])
        mdf[['TotalCharge', 'TotalCharge_z_score']].head()
#Get total number of outliers for total charge
        totalcharge_z_score=mdf.query('TotalCharge_z_score > 3')
        totalcharge_z_score.info()
```

Part III: Data Cleaning (Treatment)

D1: Findings

1. During the detection phase of this data-cleaning process, I was able to determine duplicated values, missing/null values, and any outliers in the medical data frame. After implementing the .duplicate() function, I was able to determine there were no duplicate values in this data set. Of the 10,000 entries in each column, the duplicated code returned all false values, indicating there were indeed no duplicates.
2. After using the isnull().sum() function, I found there were 2,588 missing values for children, 2,414 missing values for age, 2,464 missing values for income, 2,467 missing values for soft_drink, 982 missing values for overweight, 984 missing values for anxiety, and 1,056 missing values for initial_days.

3. Outliers were also discovered during my cleaning process, using boxplots and z-scores, in variables population, income, and total charge. In the population column, 218 outliers were found with values from 0-122,814. In the income column, 180 outliers were found with values from $154.08-$207,249.13. In the total charge column, 276 outliers were found with values from $1256.751699-$21524.22421.

D2: Treatment Methods

1. The first step in my cleaning process was looking for any duplicates in the dataset. Using the .duplicated() function, a returned query showed no duplicate values existed. If duplicates were present in the data, the query would have produced True results. Of the 10,000 entries in this dataset, all results returned false and indicated no duplicates existed. To confirm there were no duplicates, I used the .duplicated().value_counts() function to give me the total number of false values, which was 10000. However, I did find that there was an unknown column with the same values as the case order column, which I assume is a placeholder value for each row in the data set, and decided to remove it using the .drop() function as it shows no significant value or use to the data frame.
2. To look for missing values, I used the isnull().sum() function to query which variables had missing values and the exact counts of missing values for those columns. In determining this information, I could then get to work on fixing those missing values. I first begin with visualizing the distribution of the quantitative variables by using histograms. Using histograms allows me to determine how I am going to replace these missing values, depending on the shape or form of each individual histogram, by using either a mean or median to fill in those missing values. For example, the children and income histograms produced skewed results which indicate that I needed to use the .fillna() and .median() function for imputation, while the age column produced a uniform histogram and required the use of the .fillna() and .mean() function for imputation. Lastly, the initial_days column produced a bi-modal histogram that also requires the use of the .fillna() and .median() functions for imputation. After executing my imputation code for each variable, I then confirmed that imputation fixed the missing values in these columns using the isnull().sum() function, returning a query where missing values were no longer present in these variables.

After addressing the missing values in the quantitative variables, I then needed to address the qualitative data types. For qualitative data types, I needed to use the .fillna() and .mode() functions for imputation. After imputing those missing values, I then needed to fix the overweight and anxiety variables so they would match the other qualitative variable types in a yes or no value. By doing so, this improves the quality of the data by making each column uniform and organized. I used the .replace() function to change the 0 values to no, and the 1 values to yes. I then confirmed the values were changed using the .unique() function and confirmed the missing values for the qualitative data types were imputed using the .isnull().sum() function once again.

3. The third step to my data-cleaning process was to look for outliers in the data. I first looked for all the numerical data types using the select_dtypes() and np.number functions. I was then able to code a boxplot using the .subplots() and sns.boxplot() functions to visualize which numerical variables had outliers present. After reviewing my boxplot, I then imported the scipy.stats package so that I could proceed with calculating z-scores for each of these variables containing outliers. I used z-scores because I could not tell exactly how many outliers I had by just looking at the boxplots. After calculating z-scores for each variable, I then used the .query() and .info() functions to produce results that gave me the exact number of outliers beyond the z-scores of negative 3 or positive 3.

   The next step was to deal with these outliers and because there were so many in each variable, I did not want to remove them from the data set and compromise the integrity of the data. To address the outliers, I used the interquartile range method to determine the median and maximum values. I first obtained the median value by using the .quantile(0.5) function. To calculate the IQR, I subtracted the 25th percentile from the 75th percentile. Now that I have my IQR, I could then calculate maximum values for each variable by adding 1.5 multiplied by the IQR to the 75th percentile. In doing this, I could then replace those outliers beyond the maximum value of the boxplots with the median values.

D3: Summary of Work

The treatment for duplicates was not needed because none existed, however. I did notice a column that was unknown and mimicked the data for the case order column. I deemed it an unnecessary column and decided to drop it from the data set as a duplicated column with no significance. The missing data in numerical variables were imputed by using either the mean or median, depending on the distribution as visualized using histograms. Categorical missing values were imputed using the mode. After dealing with missing values, I moved to dealing with any outliers in the medical dataset. I used a combination of boxplots, z-scores, and interquartile range calculations to find where outliers existed, the exact count of outliers, and median values to replace outliers with beyond maximum values of each boxplot. Because the columns with outliers had so many, I decided it would not be logical to remove them, rather imputing them with median values to maintain some integrity in the data frame. The data has now been verified to have no duplicates, missing values, or extreme outliers and is ready for the next step in the data-cleaning process.

D4: Treatment code

```
#Remove unknown column
        mdf=mdf.drop(mdf.columns[[0]], axis=1)
#We have missing values in Children, Age, Income, Soft_drink, Overweight, Anxiety, and Initial_days
#Perform imputation for missing values on quantitative data types using mean and median
        mdf['Children'].fillna(mdf['Children'].median(), inplace = True)
        mdf['Age'].fillna(mdf['Age'].mean(), inplace = True)
        mdf['Income'].fillna(mdf['Income'].median(), inplace = True)
        mdf['Initial_days'].fillna(mdf['Initial_days'].median(), inplace = True)
#Verify missing values for quantitative values are filled after imputation
        mdf.isnull().sum()
#Perform imputation on missing values for qualitative variables using mode
        mdf['Soft_drink']= mdf['Soft_drink'].fillna(mdf['Soft_drink'].mode()[0])
        mdf['Overweight']= mdf['Overweight'].fillna(mdf['Overweight'].mode()[0])
        mdf['Anxiety']=mdf['Anxiety'].fillna(mdf['Anxiety'].mode()[0])
#Replace 'Overweight' column values to match other yes/no types
        mdf['Overweight'].replace({0: "No", 1: "Yes"}, inplace= True)
```

```
#Confirm replace values for 'Overweight'
        mdf.Overweight.unique()
#Replace 'Anxiety' column values to match other yes/no types
        mdf['Anxiety'].replace({0:"No", 1: "Yes"}, inplace= True)
#Confirm replace values for 'Anxiety'
        mdf.Anxiety.unique()
#Verify missing values for qualitative variables were imputed
        mdf.isnull().sum()
#Get median value for population
        med=mdf['Population'].quantile(0.5)
        print(med)
#Calculate IQR and max value for population
        q1=mdf['Population'].quantile(0.25)
        q3=mdf['Population'].quantile(0.75)
        iqr=q3-q1
        upper_limit=q3+(iqr * 1.5)
        print(upper_limit)
#Replace outliers with median value in population
        mdf['Population']=np.where(mdf['Population'] > 33820.375, 2769.0, mdf['Population'])
#Get median value for income
        med=mdf['Income'].quantile(0.5)
        print(med)
#Calculate IQR and max value for income
        q1=mdf['Income'].quantile(0.25)
        q3=mdf['Income'].quantile(0.75)
        iqr=q3-q1
        upper_limit=q3+(iqr * 1.5)
        print(upper_limit)
#Replace outliers with median value in income
        mdf['Income']=np.where(mdf['Income']>80232.75, 33942.28,mdf['Income'])
#Get median value for total charge
```

```
    med=mdf['TotalCharge'].quantile(0.5)
    print(med)
#Calculate IQR and max value for total charge
    q1=mdf['TotalCharge'].quantile(0.25)
    q3=mdf['TotalCharge'].quantile(0.75)
    iqr=q3-q1
    upper_limit=q3+(iqr * 1.5)
    print(upper_limit)
#Replace outliers with median value in total charge
    mdf['TotalCharge']=np.where(mdf['TotalCharge']>14157.615055625001,
    5852.250564,mdf['TotalCharge'])
```

D5: Attach CSV File

An attached CSV file has been provided.

D6: Disadvantages

For this data-cleaning process, we used imputation methods to fix missing values in the dataset. This means that we used either the mean, median, or mode to fill in those missing values. Because of using this method, we never knew the actual values of missing variables in each column, therefore we assumed the values using the mean, median, or mode. This could limit the actual integrity of the data because of these imputation methods. Outliers were also imputed with estimated values and posed the same concerns. Reasons for missing values could be simply not following up with certain patients on surveys, accidentally skipping over a value, or accidentally adding an extra number in a numerical value.

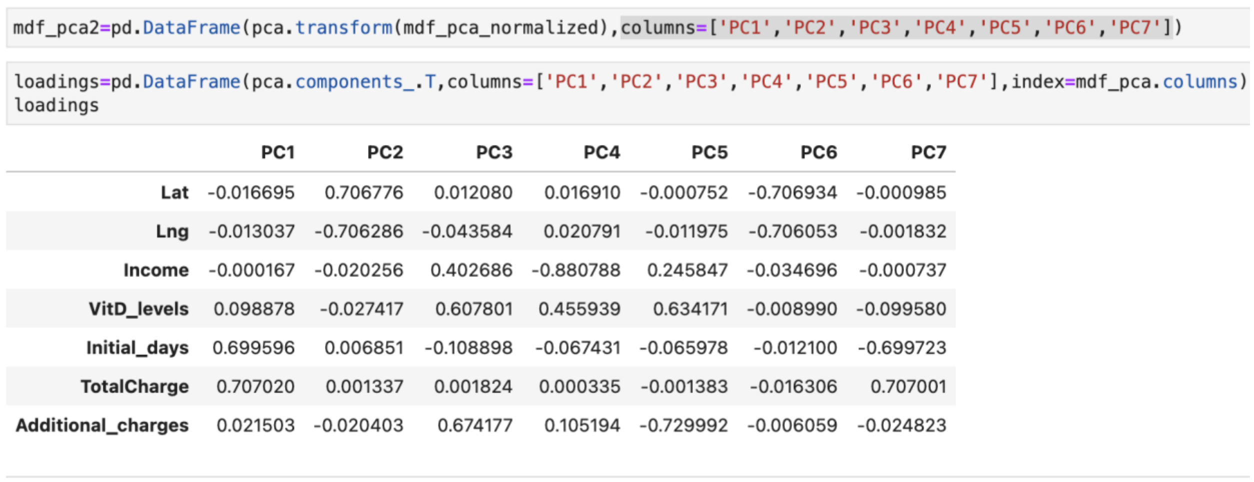
D7: Challenges

I think that if all of the actual values or if the data frame was fully complete, we could get a completely different-looking analysis in each column that had missing values and outliers. Distributions may not look as drastic, and outliers may not be as extensive. If an analyst were to use my now-cleaned data to answer my research question, there could be limitations in getting an accurate population of patients for the study. This is in part due to the imputation method I used to deal with outliers in the population variable, where I assumed outlying values by imputing them with a median. This could skew data for attaining accurate population numbers and for getting accurate data on how correlated high blood pressure is to having strokes among a study group.

Part IV: PCA

E1: Principal Components

Variables used for the principal component analysis were Lat, Lng, Income, VitD_levels, Initial_days, Total Charge, and Additional_charges. The goal of PCA is to select continuous quantitative variables only for analysis in this step. After performing PCA, I produce a loadings matrix with the values for each principal component that is attached below.



```
mdf_pca2=pd.DataFrame(pca.transform(mdf_pca_normalized),columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7'])
```

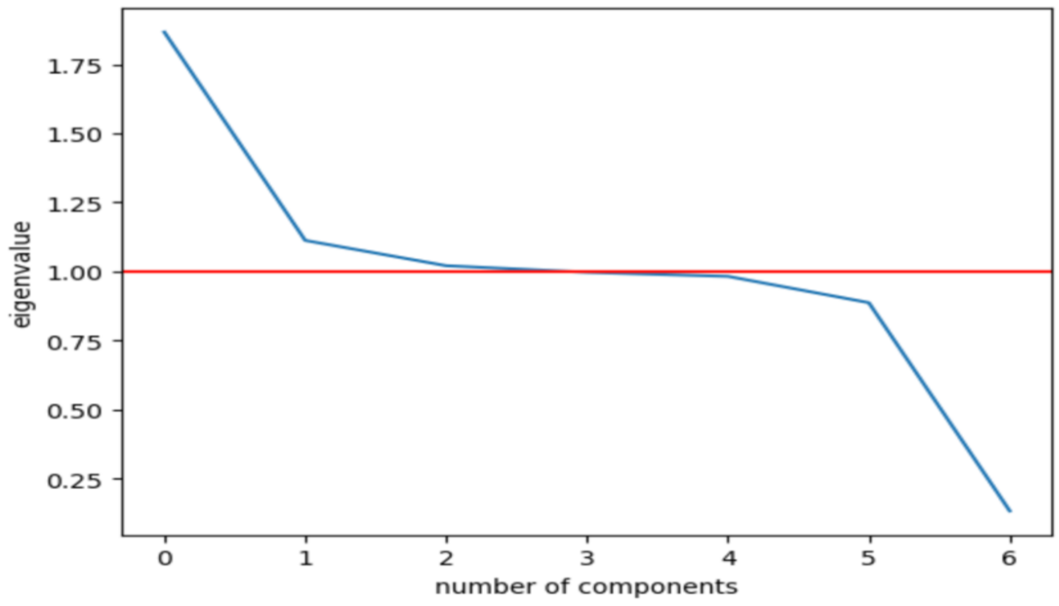
```
loadings=pd.DataFrame(pca.components_.T,columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7'],index=mdf_pca.columns)
loadings
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Lat	-0.016695	0.706776	0.012080	0.016910	-0.000752	-0.706934	-0.000985
Lng	-0.013037	-0.706286	-0.043584	0.020791	-0.011975	-0.706053	-0.001832
Income	-0.000167	-0.020256	0.402686	-0.880788	0.245847	-0.034696	-0.000737
VitD_levels	0.098878	-0.027417	0.607801	0.455939	0.634171	-0.008990	-0.099580
Initial_days	0.699596	0.006851	-0.108898	-0.067431	-0.065978	-0.012100	-0.699723
TotalCharge	0.707020	0.001337	0.001824	0.000335	-0.001383	-0.016306	0.707001
Additional_charges	0.021503	-0.020403	0.674177	0.105194	-0.729992	-0.006059	-0.024823

E2: Retained Principal Components

After obtaining the loadings matrix using the values discussed above, I was then able to use a scree plot to determine which components I should retain. A scree plot gives an analyst a visual representation of which components should be kept during analysis by plotting eigenvalues for each component. Components with eigenvalues greater than or equal to one should be retained, as they represent how meaningful a PC is in the data being analyzed. By using a scree plot, I was able to determine that PC 0, PC 1, PC 2, and PC 3 should be retained because they had eigenvalues greater than or equal to 1. A screenshot is provided for reference.

```
plt.plot(eigenvalues)
plt.xlabel('number of components')
plt.ylabel('eigenvalue')
plt.axhline(y=1, color="red")
plt.show()
```



E3: Benefits

In relation to the medical dataset I have been working with for this performance assessment, principal component analysis can be beneficial for any healthcare organization. PCA can help an analyst find relationships between variables, like that of a patient's income and their vitamin D levels or total days admitted, and the total charge of their hospital stay. I think PCA is beneficial for correlative analysis in settings like healthcare. Furthermore, I was able to see how valuable specific variables were by using PCA. PCA may also be useful in helping healthcare admissions departments determine which treatments could reduce the length of stay for hospital admissions. This process helps reduce the data and highlight variables that are most significant to the analysis.

F: Panopto

A link to my Panopto video was provided for discussing this project.

G: Third-Party References

No third-party references were used in this project.

H: References

No text or web sources were used for this project.