**Redefining Complication Risk Assessments in the US**

David Shao

College of Information Technology, Western Governors University

D206: Data Cleaning

Dr. Eric Straw

**A. Research Question**

Hospital readmissions reflect the quality of care from hospitals to their patients and community. Many hospital readmissions portray a bad image and message about the healthcare quality to the community, thus influencing where people go for their healthcare needs (Krumholz et al, 2017). With the high cost of healthcare services in the US, it is safe to say that people prefer not to go to the hospital unless necessary. Sadly, there are certain individuals with problematic complications that require them to receive more medical help or counsel. It is up to physicians and primary health care providers to assess individuals’ complication risks and treat them accordingly to decrease the risk. This may consist of prescribing a health care plan or aiding to increase the overall patient’s safety. The Institute of Medicine reported that 98,000 people die in any given year from medical errors while in the hospital (2000). This statistic prompted Congress to enact the Patient Safety and Quality Improvement Act of 2005 which articulated several key principles and responsibilities that healthcare providers should abide by for patient safety (Levy et al, 2010). An important duty that healthcare providers have is to establish a patient safety database. Assessing patient risk cannot be determined by just one individual and when only one person is used, the assessment may not be entirely accurate. “The healthcare system is made up of individual players, but its ultimate goals of patient care and safety are accomplished through teamwork” (McGowan et al, 2023, para.11).

Keeping these things in mind, I believe it is important to analyze and investigate the decisions that healthcare providers make, especially those that may have been made without peer review or second opinions. The data provided from the CSV file has a variable known as ‘Complication\_risk’. I believe that data provided from this variable can be analyzed to see if it truly accurately represents the risk that patients have as they may experience hospital readmission. The ‘Initial\_admin’ variable can be used to look at the patients who were readmitted to the hospital based on emergency admissions, elective admission, or observation admission. All in all, cleaning and analyzing the data with these important variables I mind will help find patients who have been readmitted within a month of their release and evaluating the complication risk assessment that they received. So, are complication risk assessments given by primary health providers accurate? Hopefully, the data provided can provide insights into the legitimacy of complication risk assessments and if they are to be revised or redefined to help accurately measure patient health and attention for the future.

**B. Data Variables**

The data set from the file “medical\_raw\_data.csv” contains useful data on patients’ medical histories and demographics. This data set has 10,000 rows and 52 variables. Each row contains data for a ‘customer’. Columns consist of patient residential information such as city, state, county, zip code, latitude, longitude, and time zone. These variables represent the patient’s residential address, the GPS coordinates of the patient, and the time zone that they are in. The data type for most of these variables is qualitative nominal data except for latitude, and longitude. The latitude and longitude values have a quantitative continuous data type since they can take any value within an available range. Another instance of qualitative data is found in the “Population” variable which describes the population within a mile of the patient.

The variables that differentiate each patient and case in the data set are the “Customer\_id”, “CaseOrder”, “Interaction” and “UID” variables. The “Customer\_id” column serves as the primary key for the data set and is used to identify customers based on a qualitative nominal value. The “CaseOrder” variable is a unique variable that orders the raw data in numerical order and the “UID”/ “Interaction” variables give IDs to help locate specific patient information with regards to transactions, procedures, and admissions. The data type for the “CaseOrder” is qualitative discrete data while the “UID” / “Interaction” variables have quantitative nominal data. The following is an example of the values found from these variables with the first CaseOrder from the CSV file.



The next set of variables ranges into patient personal information. The personal information listed in the patient data includes their job, number of children in the household, age, highest earned educational degree, employment status, annual income, marital status, and self-identified gender. All the variables represented in the next following columns use qualitative nominal data: “Job”, “Education”, “Employment”, “Marital”, and “Gender”. An example of the following data is a continuation of the data from the last example.



The “Children”, “Age”, and “Income” columns are also shown in the example and use numerical values and are categorized as quantitative data. It is important to note that the “Income” variable uses quantitative continuous data whilst the other two are discrete data types.

The next set of variables includes data on the patient’s personal medical history. The column “ReAdmis” has qualitative nominal data that describes if the patient was readmitted within a month of their release. It is important to note that this variable will be very important for the data-cleaning process. There are only three columns that have values with quantitative discrete data. The variables “Doc\_visits”, “Full\_meals\_eaten”, and “VitD\_supp” describe how many doctor visits, meals, and vitamin D supplements were administered to the patient. The variable “VitD\_levels” contains the numeric level of vitamin D the patient has measured as ng/ml, thus making the values quantitative continuous data. Examples from this data are the following:

A picture containing text, font, screenshot, line

Description automatically generated

The variables “Overweight”, and “Anxiety” should have qualitative nominal values of ‘Yes’ or ‘No’; however, based on the CSV file, the columns have numeric (1 and 0) and missing values (NA). A plan to change these values will be carried out during data cleaning. The rest of the variables related to the patient’s medical history are the following: “Soft\_drink”, “Initial\_admin”, “HighBlood”, “Stroke”, “Complication\_risk”, “Arthritis”, “Diabetes”, “Hyperlipidemia”, “BackPain”, “Allergic\_rhinitis”, “Reflux\_esophagitis”, and “Asthma”. These values from these variables are mostly ‘Yes’ or ‘No’, and “NA” for missing values. This indicates that the variables consist of qualitative nominal data. The exception to this analysis is the variable, “Complication\_risk”. This variable has the values ‘high’, ‘medium’, and ‘low’, thus considering the data type to be qualitative ordinal data. The following list explains the types of questions the variables answer in terms of patient medical history:

“Soft\_drink” – Does the patient usually drink more than three sodas in a day?

“Initial\_admin” - How was the patient admitted into the hospital with either of the following choices: emergency admission, elective admission, or observation?

“HighBlood” - Does the patient have high blood pressure?

“Stroke” – Has the patient had a stroke?

“Complication\_risk” – What is the level of complication risk for the patient after being assessed by a primary patient assessment?

“Arthritis” – Does the patient have arthritis?

“Diabetes” – Does the patient have diabetes?

“Hyperlipidemia” – Does the patient have hyperlipidemia?

“BackPain” – Does the patient have chronic back pain?

“Allergic\_rhinitis” – Does the patient have hyperlipidemia?

“Reflux\_esophagitis” – Does the patient have reflux esophagitis?

“Asthma” – Does the patient have asthma?

Below is an example of the values found in the data file. It is important to keep in mind that the “Overweight” and “Anxiety” variable values will be changed to “Yes” or “No” values during data cleaning.



The next set of variables pertains to patient billing with the following: “Services”, “Initial\_days”, “TotalCharge”, and “Additional\_charges”. The “Services” variable describes the primary service that the patient received while hospitalized as qualitative nominal data. “Initial\_days” is the number of days the patient stayed in the hospital during the initial visit with quantitative discrete data. “TotalCharge” is the amount of money charged to the patient daily, not including specialized treatments. “Additional\_charges” is the average amount charged to the patient for miscellaneous procedures, treatments, medicines, and others. These two variables contain quantitative continuous data values. An example of this data can be seen below as a continuation of the first case order:

A picture containing text, font, screenshot, line

Description automatically generated

Finally, the last eight variables, “Item1”, “Item2”, “Item3”, “Item4”, “Item5”, “Item6”, “Item7”, and “Item8” represent responses from an eight-question survey that asked patients to rate the importance of certain factors using a scale of 1 to 8 (1 representing the most important, and 8 representing the least important). Each variable has qualitative ordinal data due to the representation of the number given by the patient. The items represented are the following in sequential order: timely admission, timely treatment, timely visits, reliability, options, hours of treatment, courteous staff, and if there is evidence of active listening from a doctor. An example of how these are represented for case order 1 can be found below:

****

**C. Data Cleaning Plan**

To begin cleaning the data, the pandas function *read\_csv()* will be used to load the CSV file into Python and create a data frame for us (Larose, 2019, p. 12). Once done, *df.info()* will be used to give information about the dataset. This function will provide the column names and the number of null values in each column along with the data type. To check if there are any duplicate rows that share the same data with other rows, the function *df.duplicated()* is used to return a Pandas series of Boolean values for duplicate rows. Rows that are duplicates are listed as “True” next to the row number. A limitation of this function is that the return is only based on the settings within the respective environment to return a limited number of rows (nkmk, 2021). In this case, the first 5 and last 5 results from the dataset were the only rows returned. In order to find the sum of total duplicates in our dataset the function *value\_counts()* will be used to count how many duplicates there are in the data frame. This function alone can be used to count the unique values in ascending or descending order from variables in the data frame; however, when used with the *duplicated()* function we can count the total number of duplicates found. The duplicates will be labelled as “True” and the unique rows as “False” (Chen, 2021, para. 1). The combination of functions will be used as follows: *print(df.duplicated().value\_counts()).* If this results in any number of “True” values, the *drop\_duplicates()* function will then be used to remove the duplicates from the dataset (nkmk, 2021). To check if the duplicates were dropped, the function used last time, *print(df.duplicated().value\_counts())* will be used again.

We will then detect any missing values that must be addressed in our data frame. The *isnull()* function will be used with the *.sum()* function to identify which variables contain any missing values and how many. A visualization of the missing data can also be performed by using the missingno package and using the *msno.matrix(df, labels = True)* function to look at the sparsity of missing values within our data frame. We will give the matrix a title by using *plt.title(‘Missing Data Matrix’)* and display it by using *plt.show()*.

Using these tools will allow us to look at the variables that need to be further examined. It will then be up to us to determine how the variables with missing values should be treated to ultimately clean our data set. I believe that it is imperative that we do not resort to deletion and instead use univariate imputation to deal with the missing values that are apparent in the various variables found in the data frame. This method will allow us to examine the distribution of data, replace the missing value with an estimated value, verify that the missing values have been resolved, and make sure that the distribution of values is in alignment with the distribution before the imputation. Variables that have a normal distribution or uniform distribution will allow us to replace missing values with the mean value. If the variables have a skewed distribution (left or right) the median value will be used to safely replace the missing values in the variable. A bimodal distribution will require us to use a median or mode value to replace the missing values. For categorical variables, we will always use the mode for univariate imputation.

Looking at the variables that contain missing values, we will examine the distribution of values by using histograms found in the matplotlib.pyplot package. By using the function *plt.hist(df[])* on a variable, we are given a histogram that can visually provide the distribution of values within a variable. This will be incredibly useful to use for variables that have numerical values (Hunter et al, 2023). After analyzing the value distribution of the variable, we will replace the missing values with either the mean, median, or mode. The *fillna()* function will be used to fill all “NA” values with a mean or median for quantitative values and mode for categorical values. It is important to note that within the *fillna()* function *inplace=True* should be used in order to have the imputation remain permanently within the variable. For categorical variables in which we replace missing values with the mode, we use a locator to ensure that the mode is being imputed on all values that contain a null value within the variable.

Once the imputations have been performed, we will verify that the distribution has not been significantly altered after the changes have been made to the missing values. To verify that all the missing values have been replaced, the *isnull().sum()* function is used to detect any null values found in the data frame. Once verified, we will use the histogram function from matplotlib.pyplot once again to create the histograms of the variables that received treatment. The distribution of values within these variables should remain the same as they were before the changes. Depending on the number of treated missing values within certain variables, the distribution graph may have a large spike on the histogram. This may be due to the large number of imputations performed on many missing values within the variable. Even though there is a perceived change in appearance, there should still be a relatively normal distribution. To verify that nothing has significantly changed, the mean and median values before and after the imputation should be calculated and compared to ensure that values are still aligned. The mean and median should be the same. In Python we can use the *describe()* function to make a statistical summary of variables. The function can be used to specifically look at the specific variables that we treated. A drawback from this function in Python is that we will not see the median value from the variables; however, the statistics package will allow us to use the *median()* function to get the median values before and after treatment of missing values.

To find any outliers in the data I will be looking into the range of values of outliers within the dataset to determine if the values are within or outside an acceptable range. I will also determine the number of outliers within the dataset. To do this, I will be calculating the z-scores for the variables and using a box plot to have a visual representation of the values. The z-score is a numerical measurement that can describe the value’s relationship to the mean. Values with a z-score of less than or equal to -3 and greater than or equal to 3 are classified as outliers. To calculate the z-score the function *zscore()* will be used from the scipy.stats package. We will also create a new column for the z-score calculation and use the *plt.hist()* function to visualize the outliers by the z-score. We will also use boxplots that will provide us with mean values, the dispersion of the dataset, and any outliers that are below the minimum or above the maximum fence. This will be done by using the *boxplot()* function from the seaborn package. Once we create the boxplot for a variable, I will use the *query()* function to determine the number of outliers based on the values found outside of the boxplot. To treat any outliers that we find, we have the choice to impute, retain, exclude, or remove the outliers. In the imputation method we recreate the imputation method earlier described when treating missing values. Retaining the outliers is used if the values are acceptable or justified. Excluding the outliers is used if it is too difficult to confirm if the values are legitimate or not. Removing the outliers should only be used when no other treatment options are viable. This type of treatment can greatly reduce the sample size and greatly affect the end analysis.

Depending on the variable, we will replace the outliers with the mean, median, or mode when using the imputation method. This will consist of us using the *where()* function from the numpy package to select the outliers in the variable and transforming them into missing values with *np.nan*. This will allow us to use the function *fillna()* to replace the missing values we created with the mean, median, or mode. After this is done, we will create a new boxplot with the transformed variable and see if there are still any outliers within the variable. This process may have to be done again until we find that the count and the values of the outliers are acceptable and reasonable, and/or expected.

If we exclude any outliers from our dataset, we will need to store the outliers within a new data frame. We will create the new data frame by placing the outliers into the outliers data frame first, and then dropping the outliers from our working data frame by using the *drop()* function. The *index()* function will be used in conjunction to index what is being dropped from our data set. Again, a new boxplot will be created to view the transformed variable and detect any other outliers. We will use these series of repetitions until there are no outliers that need treatment. It is important to note that the variables being treated into the dataset will differ and there may be a combination of different outlier treatments for a variable. I will also be using the *hist()* function to verify that the distribution of values before and after treatment. Looking at the distribution of values is a necessary to detect outliers that other methods can’t detect (Kleppen, 2023).

Finally, we will finish cleaning the data by observing and re-expressing categorical variables if needed. The categorical variables in our data set will be either ordinal or nominal. Earlier, I defined the data types of each variable and found which categorical variables they were. I will be re-expressing categorical variables that are inconsistent in their expressions that are found within the data set. As previously discussed, I found that the ‘Anxiety’ and ‘Overweight’ variables are supposed to have “Yes” or “No” values; however, the values present in these variables consist of the numerical values of “1” and “0”. These variables have ordinal categorical data as there is some inherent order within the values of these variables. Looking at the data we can determine that the ordinal encoding technique should be used to reassign the numerical values as “Yes” or “No”. We will run the *value\_counts()* function to confirm the number of unique values for these two variables. We will then use the *replace()* function to replace the values of “0” with “No”, and those with the value of “1” with “Yes”. To check that the script worked, the previous function, *value\_counts()*, will be used to confirm that the change was successful.

In the end, we will use the *info()* function to look at the size of our final data frame and verify if there are any null values within any of the variables. We will export our work by using the *to\_csv()* function to create a CSV file with all the changes that were made to the original data. To see all the code that was used for cleaning, please see the code attached to this submission.

The language I chose to clean the “medical\_raw\_data.csv” data frame was Python. Python uses powerful libraries and packages that ease the data-cleaning process. A huge reason why I wanted to use Python is because of the intuitive and simple syntax that the language uses. Python and R both have their advantages and are incredible tools for data science. I understand that there will be times when one language will be used over the other but the “true competitive skill is knowing how to break down and solve the given business problem, regardless of the language” (WGU, 2022, para. 4).

The important packages that will be imported are pandas, statistics, numpy, scipy.stats, matplotlib.pyplot, missingno, sklearn.decomposition, and seaborn. These packages will allow us to treat any missing data, duplicates, outliers, and re-express categorical variables throughout the data frame. With the pandas package, we will be able to read the CSV file, filter, drop, replace, and make overall changes to the data. This package will allow us to use data analysis tools to count values and identify specific or missing data. This will be the main package used to allow us to place the data into a specific data frame for analysis. A good reason to use pandas for data analysis is that it plans to become the most powerful and flexible open-source data analysis or manipulation tool that is available in any language (The Pandas Development Team, 2023).

The statistic package will allow us to calculate mathematical statistics for numeric data. This package will give us functions to find the mean, median, or mode of quantitative variables (Python Software Foundation, 2023). We are going to use the numpy package to aid us in manipulating values within variables during our treatment. NumPy will be used when we treat our outliers by using the *where()* function to specify the range of values needed to be imputed or excluded. NumPy is a powerful tool that works with numerical data in Python and can perform mathematical operations on arrays (NumPy Developers, 2022). During the treatment of outliers we will be using the *zscore()* function from the package scipy.stats to find the z-score of our quantitative variables. This package contains a wide range of probability functions that can be used for probabilistic distributions and statistical operations (The SciPy Community, 2023).

The package matplotlib.pyplot will allow us to visualize the data that we will be focusing on treating. Using this package will allow us to create important visuals, such as histograms, to demonstrate important data patterns or to justify the treatment of missing data (Hunter et al, 2023). Another important package to aid us in visualizing the number of missing values in our data is missingno. This package will help create a matrix to find any patterns of “missingness”, or sparsity, in the dataset (SujanDutta, 2019).

To conduct principal component analysis (PCA), we will import the package PCA from sklearn.decomposition. Performing PCA on our quantitative data will allow us to identify and reduce any noise we may have in our data set. This will allow us to obtain results faster on real-world datasets (Pramoditha, 2023). We will also use the package seaborn by importing Seaborn. Seaborn will allow us to create the visual for our PCA in our data. Seaborn is a data visualization library based on matplotlib and it can provide informative statistical graphs (Waskom, 2022).

**D. Data Cleaning (Treatment)**

During our data cleaning we did not find any duplicate values; however, we did find a considerable number of missing values from the original dataset. There were only 7 variables that contained missing values. The variables with missing values were ‘Children’, ‘Age’, ‘Income’, ‘Soft\_drink’, ‘Overweight’, ‘Anxiety’, and ‘Initial\_days’. The ‘Children’ variable contained 2588 missing values. The ‘Age’ variable contained 2414 missing values. The ‘Income’ variable contained 2464 missing values. The ‘Soft\_drink’ variable contained 2467 missing values. The ‘Overweight’ variable contained 982 missing values. The ‘Anxiety’ variable contained 984 missing values. The ‘Initial\_days’ variable contained 1056 missing values.

I was also able to find several outliers from certain quantitative variables from our dataset. The variables that I treated with outliers were the ‘Children’, ‘Income’, ‘VitD\_levels’, ‘Full\_meals\_eaten’, ‘VitD\_supp’, and ‘TotalCharge’ variables. The number of outliers found from the variables are the following: ‘Children’ = 457 outliers, ‘Income’ = 714 outliers, ‘VitD\_levels’ = 694 outliers, ‘Full\_meals\_eaten’ = 8 outliers, and ‘VitD\_supp’ = 65 outliers. The other quantitative variables that were not mentioned did not have any outliers that needed treatment.

To check for any duplicates in the data, I used the code: *print(df.duplicated().value\_counts())*. The output of this code resulted in 10000 ‘False’ values. This meant that all the rows were unique; therefore, there were no duplicate rows to drop. I used this method because it was one of the easiest and fastest ways to find any duplicate rows that needed attention. Since there were no duplicate values to drop, we then moved on to finding the missing values in the dataset. To find any missing values, I specifically used *df.isnull().sum()* to create a list of the sum of missing values for each of the variables from the data set. This code helped clearly demonstrate how many missing values there were for each variable in the original dataset. To treat the missing values, I used the univariate imputation method to replace the missing values from the ‘Children’, ‘Age’, ‘Income’, ‘Soft\_drink’, ‘Overweight’, ‘Anxiety’, and ‘Initial\_days’ variables. This imputation method is the treatment of choice as opposed to dropping the missing values, which could greatly affect the sample size and statistical analysis of our original data.

Using this treatment method required the use of histograms to plot the distribution of values from these variables. Depending on the distribution, I used univariate imputation to replace the missing values with either the mean, median, or mode. The main goal of our univariate imputation treatment was to replace the missing values with an estimated value that would not change the original distribution of the variable. I made a histogram for all the variables that contained missing values. The ‘Children’ and ‘Income’ variables had a skewed distribution with most of the values located on the right side of the graph. This led to replacing the missing values from these variables with the median value from that variable. The ‘Age’ variable contained had uniform value distribution on the histogram; therefore, I replaced the missing values with the mean value from the variable. The ‘Initial\_days’ variable had a bimodal distribution which required the use of the median to replace the missing values.

For the three remaining variables, ‘Soft\_drink’, ‘Overweight’, and ‘Anxiety’, the mode was used to replace their missing values. These three variables are defined as categorical variables and therefore, their missing values should only be replaced by the variable’s mode. The *hist()* function was used to verify the value distribution from the ‘Overweight’ and ‘Anxiety’ variables. The histograms resulted in a skewed value distribution opposite from one another. To visually represent the value distribution of the variable ‘Soft\_drinks’ I was not able to use the *hist()* function that was used for all the previous variables. This is because the values from this variable are represented as strings (“Yes” or “No”). Instead, I used pandas to plot the *value\_counts()* of the variable. The code I made for this plot was *df['Soft\_drink'].value\_counts().plot(kind = 'bar')*. This code allowed me to create a bar graph like the histograms that I’ve been creating with matplotlib.Once done, I was able to find that the value distribution was similar to that of the other two categorical variables. This investigation helped clarify the need of re-expressing the categorical values found in the variables ‘Overweight’ and ‘Anxiety’. Treatment of this issue will be explained further along.

After I verified what the missing values had to be replaced with, I created a table using the *describe()* function to indicate statistical information, specifically belonging to the variables with missing values. This table was made before the imputation to verify that the changes made to the missing values do not significantly alter or change the statistical values or distribution of our variables. The *median()* function was also used before and after the changes since the *describe()* function doesn’t list the median value from the variables. The *fillna()* function was used to replace the missing values with the aforementioned value depending on the variable. A final look at the distribution of values was then made by using the *hist()* function to verify that the changes did not affect the distribution of values before the changes. The tables created using the *describe()* function and the *median()* function were also used to verify that the changes made did not affect the overall distribution.

To treat any outliers, I once again used the univariate imputation method to replace any outliers with estimated statistical data. This method allowed me to keep my sample size and analyze outliers that were acceptable and reasonable, and/or expected. To start, I created a boxplot of every quantitative variable by using the *boxplot()* function from the seaborn package. The boxplots made a visual representation of any outliers found in the quantitative variables. I then created new columns for each of the quantitative variables which contained the z-score. To calculate the z-scores, the *zscore()* function was used from the scipy.stats package. The column was then plotted using the *hist()* function to find out if the z-score was less than or equal to -3, or greater than or equal to 3. Z-Score values that fall in those numbers were considered as outliers. The quantitative variables in our dataset were composed of the following variables: ‘Children’, ‘Age’, ‘Income’, ‘VitD\_levels’, ‘Doc\_visits’, ‘Full\_meals\_eaten’, ‘VitD\_supp’, ‘Initial\_days’, ‘TotalCharge’, and ‘Additional\_charges’.

There was one instance where I had to use the exclusion method to treat the outliers in one quantitative variable. This required me to take the outliers and transfer them to another data frame of interest. I called the new data frame ‘outliers’ and gave a condition to isolate the outliers from the variable. The *info()* function was used to confirm the number of outliers that we would be excluding from the current data frame. The outliers were then dropped from the main data frame by using the *drop()* function and indexing the condition. The method was specifically done to the first treatment of the ‘VitD\_levels’ variable.

The ’Children’ variable showed that values greater than 6 were outliers. The *query()* function was used to count the number of outliers within that range and then converted into null values using the *where()* function. The *fillna()* function was then used to replace the null values with the median. The median was used to replace the outliers because of the distribution investigated previously. Once the outliers were replaced, I created a new boxplot and calculated updated z-scores to verify that all the outliers were taken care of. Evidently, there were still several values that had a z-score greater than 3; however, believed that a further imputation treatment would affect the distribution of values present within the variable. I decided to retain the outliers since we could not affectively prove or deny the accuracy of the data within the variable itself.

The ’Age’ variable resulted in a boxplot without any significant outliers as the value distribution was uniform throughout. Calculating the z-score also indicated that there were no outliers present within the variable. There were other variables with similar and varying observations that also did not need treatment. Some of the variables, upon first inspection, did contain outliers. However, due to further investigation and judgement, no treatment was necessary for the variable. The variables that did not receive any treatment for outliers were the following: ‘Initial\_days’, ‘Doc\_visits’, ‘VitD\_supp’, ‘TotalCharge’, and ‘Additional\_charges’. The bimodal distribution from the ‘Initial\_days’ z-score histogram indicated that there was an even distribution of the values and, thus, did not have noticeable outliers within the variable. There were also no visual outliers on the boxplot. The ‘Doc\_visits’ variable had a boxplot with no visual outliers. The z-score histogram demonstrated a normal distribution of values with only a few outliers greater than 3 and less than -3. I decided to keep these outliers since the normal distribution of values was reasonable and the recorded data may play a big role in explaining the need, or lack of, doctor visits. The ‘TotalCharge’ variable gave a boxplot with no visual outliers and the z-score calculated showed a bimodal distribution of values within the z-score range. There was no need for any treatment for this variable.

For the ‘VitD\_supp’ variable, the boxplot created represented outliers above the maximum fence. Looking at the z-score showed that there were values that were significantly skewed. I used the *query()* function to find that there were 70 outliers with values greater than 2. This variable consisted of the number of times vitamin D supplements were administered to patients. I used the *value\_counts()* function to look at the number of unique values from this variable and found that there was an acceptable distribution of values with only one patient receiving the most vitamin D supplements, 5. I believe that these outliers may be valuable for the analysis of this data since they may be subjects of investigation or follow-up due to the unusual patient data.

The ‘Additional\_charges’ variable also contained outliers that were greater than the maximum fence on the variable’s boxplot. However, when looking at the option of using the imputation method, a look at the z-score values indicated that the values were less than 3 and greater than -3. I used a histogram and the *describe()* function to confirm the z-score range. It was interesting to see that there was a reasonable distribution of values despite there being outliers on the variable’s boxplot. I decided to leave the potential outlier values unchanged since the distribution of the values and the z-score verify that there is no significant skew within the data. Another reason for this decision was that the additional charges accrued from a patient may be explained due to certain procedures or treatments that we may not have the data for.

The ’Income’ variable contained the greatest number of outliers from all quantitative variables. When looking at the boxplot, the maximum fence was located at around the value of 80,000. I recreated the steps taken for the treatment of ‘Children’ variable outliers and used the *query(), where(),* and *fillna()* functions to replace the values that are greater than 80000 with the median. This in turn changed the z-score range to less than 3 and greater than -3. The boxplot still showcased a few outliers; however, I believe that changing the remaining outliers would significantly alter the integrity of the data. Inspecting the distribution of the values using a histogram helped verify that this change was able to treat the outliers for this variable and did not distort the original distribution of the data. The remaining outliers seemed to be acceptable for our working data since income provided by the patients cannot be definitively confirmed or denied.

The “VitD\_levels’ variable contained many outliers with an amount of 694. Creating a boxplot of this variable helped visualize that there were many outliers greater than 40. This was a significant anomaly as there was a gap in the data between these outliers and the remaining lower values. I used the *query()* function and found that there were 500 outliers with their values greater than 40. This prompted me to use the exclusion treatment and create a new data frame called ‘outliers’. Once done, I then used the *drop()* function to delete the rows from the working data frame. Once the step was completed, there were still visual indications of the remaining outliers on the updated boxplot. This insight required me to look at the values and determine whether these outliers were still acceptable or reasonable. According to the National Institutes of Health (2022), “most people have adequate blood levels of vitamin D. However, almost one out of four people have vitamin D blood levels that are too low or inadequate for bone and overall health” (para.12). The boxplot indicated that there were 121 outliers greater than the maximum fence at around 23 and 73 outliers less than the minimum fence at 12.5. I decided to retain these outliers since these outliers hold significant value for further follow up or discussion. A histogram also confirmed a normal distribution of values after the initial exclusion of outliers.

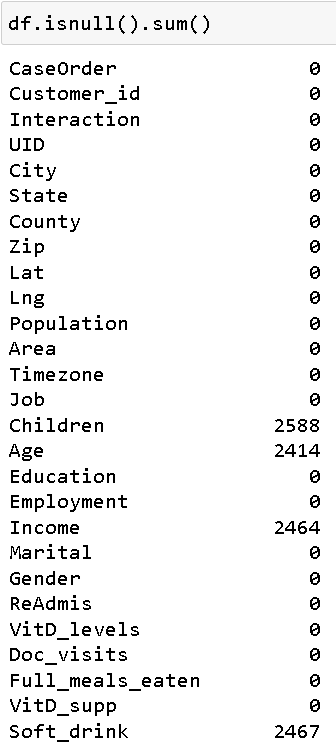
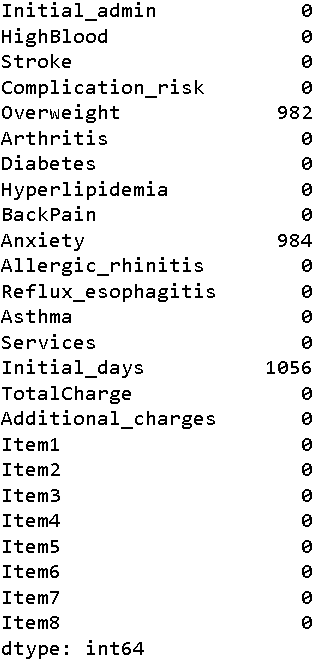
The ‘Full\_meals\_eaten’ variable appeared to have a small number of outliers according to the boxplot and z-score. I used the *value\_counts()* function to inspect the number of unique values and found that 7 meals was the highest number. I used the *query()* function to find the number of outliers greater than 5, the maximum fence value. This resulted in a total of 8 outliers that were treated using the imputation method. I chose to treat these outliers as since the volume of patients who received more than 5 meals against those who received more was discerning. To treat these outliers, I decided to replace them with the value of 5, maximum fence value on the boxplot. Replacing these outliers with the maximum fence value resulted in a boxplot that did not visualize any outliers; however, when looking at the z-score, there were still a few z-score values that were greater than 3. I made the decision to hold onto these values because of the overall distribution of values verifying the consistency of the number of meals eaten by patients. I also decided to keep these outliers because the distribution of values showcase that these high values are irregular when compared to most of the patients on the data set. These values may raise concern and investigate why patients were receiving more meals than the average number of meals given.

Once we confirmed that our treatment methods were reasonable and used the *info()* function to confirm that there were no null values in the dataset, I finished the data cleaning process with re-expressing categorical variables. As it has been mentioned countless times, the ‘Overweight’ and ‘Anxiety’ categorical variables were given numerical values instead of “Yes” or “No” values as mentioned in the dataset’s data dictionary. In order to change and re-express the values in these variables, I first used the *value\_counts()* function to verify the different values found within the variables. This gave me the count of the values “1” and “0”. This function was used for both variables, verifying that both only contained those two unique values. To re-express the values, I used the *replace()* function to replace the values of “0” with “No”, and those with the value of “1” with “Yes”. To verify that the treatment was successful, I used the *value\_counts()* function once again. The output of this function verified that the changes were successfully made. I also used the *info()* function to confirm that the data type was changed from a float to an object.

To summarize the data cleaning process I undertook, I used the *duplicated()* function to find any duplicate rows that may have been overlooked in the original dataset. This in turn gave me the length or size of the dataset which was 10000. This function alone was able to give me two important pieces of information, the confirmation of the size of the dataset I was working with, and the verification that there were no duplicate rows that had to be dropped due to redundancy. I also used the *value\_counts()* function to verify that the total number of “False” duplicates was the same number as the size of the A screenshot of a computer

Description automatically generateddata. The verification of no duplicate rows is demonstrated with the following lines of code:

I was able to then work on finding all the missing values that existed within the original dataset and went through the process of replacing all the missing, or null, values using the univariate imputation method. It was imperative that I treated all the missing data without dropping or deleting any values to maintain the integrity of the original dataset. To find what variables had missing values, I used the *isnull()* function. This gave us a total of 12,955 missing values in our data.



A black and white image of a bar code

Description automatically generatedI created visual representations of the data by using the *matrix* function from the missingno package and *hist()* function from the matplotlib.pyplot package.

A graph with a number of bars

Description automatically generated with medium confidenceA blue graph with numbers

Description automatically generatedA graph with blue and white bars

Description automatically generatedOnce I inspected the distribution of data using the histograms, I was able to decide what I needed to replace the missing values with. Histograms that have a normal distribution or uniform distributions allowed us to replace missing values with the mean value. If the variables had a skewed distribution (left or right) the median value was used to replace the missing values. Bimodal distributions required us to use the median to replace the missing values. For categorical variables, we only used the mode for univariate imputation.

‘Children’ - Skewed Distribution

‘Age’ - Uniform Distribution

‘Initial\_days’ - Bimodal Distribution

A screenshot of a computer screen

Description automatically generatedTo verify that this was done correctly, I compared the histograms of the untreated variables with the treated variables and used *describe()* and *median()* to make sure that no significant statistical figures were changed after the treatment.

**Before**

A white screen with red text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**After**

A screenshot of a computer

Description automatically generated

A computer code with text

Description automatically generatedThe missing values from the ‘Children’, ‘Income’, and ‘Initial\_days’ variables were replaced by the median value and the missing values from the ‘Age’ variable were replaced by the mean. As for the categorical variables, ‘Overweight’, ‘Anxiety’, and ‘Soft\_drink’, the missing values were replaced by the mode.

A graph with numbers and lines

Description automatically generatedOnce all the missing values were treated and verified, I looked at all the quantitative variables to find any outliers that needed to be treated. I used the package scipy.stats to create boxplots for visual representation of outliers and calculated the z-score from the quantitative variables. All this aided in my approach to treat any outliers that were within the dataset. Creating the boxplots allowed me to find the minimum and maximum fences and if there were any values beyond or below these fences. Calculating the z-scores required me to create new columns that contained these values. These new columns were given the prefix ‘Z\_Score\_”. The following example is the boxplot and z-score table for the ‘Children’ variable.

A screenshot of a computer

Description automatically generated

I then created histograms from these columns to find the distribution of z-score values for the variables. Values below -3 or above 3 led to the confirmation of outliers present within the variable. The example below is the histogram for the ‘Z\_Score\_Children’ column.

A blue and white graph

Description automatically generated

I used the univariate imputation to treat these outliers, since deletion of the outliers would have greatly impacted the integrity of the original data and any future analysis. This method was done on all quantitative variables. I used the *query()* function to find how many outliers were greater than the maximum fence, or less than the minimum fence on the box plot.

A screenshot of a computer code

Description automatically generatedA graph with numbers and lines

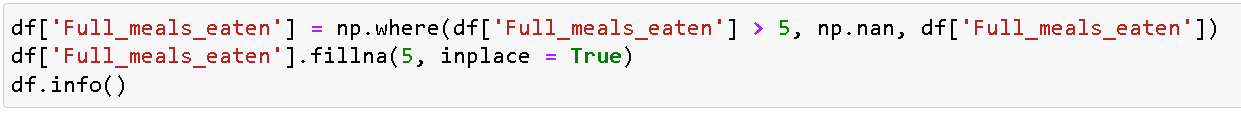
Description automatically generated

**Maximum Fence Value:** 6

A screenshot of a computer code

Description automatically generatedOnce found, I treated the outliers by replacing them with null values and then replacing them with the value best suited for treatment. To replace the outliers with missing values I used the *where()* function to define what variables had to be replaced. I then used the *fillna()* function to replace the null values with the value needed to treat the outliers. The function *info()* was then used to confirm that all the null values we made were replaced.

Once this was done, I created a new box plot for the treated variable. If any more outliers were evident, then the process was repeated until the data was accepted and reasonable, and/or expected. The updated z-score was calculated and plotted to verify that the values were within the acceptable range. There were certain quantitative variables that did not have any outliers before and after treatment, but provided z-scores that weren’t within the range of acceptance. To maintain the integrity of the original data I made decisions to keep the data unchanged and keep the outliers since replacing the values would have erased important clues in the data which may require further analysis or inspection. This decision was also met with variables that still had outliers on the boxplot and z-scores. It was of the upmost importance to justify the reasoning by retaining the outliers and explaining why they should be accepted.

There was an instance where I found that replacing the outliers with the median would skew the integrity of the original data; therefore, I used my judgement to replace the values with the maximum fence value. This was done on the variable ‘Full\_meals\_eaten’. The variable ‘Full\_meals\_eaten’ indicated that values greater than 5 were outliers; therefore, I replaced the outliers with the maximum fence value. I believe that the outlier values provided reasons to investigate why patients were receiving more meals than the average number of meals given and replacing them with the median would have greatly changed the interpretation of the data.

A screenshot of a computer code

Description automatically generatedI did not use the univariate imputation method on the ‘VitD\_levels’ variable and instead used the exclusion method to treat the outliers. There was a significant gap data between outliers greater than 40 and the remaining lower values. This prompted me to use the exclusion treatment and create a new data frame called ‘outliers’. Once done, I then used the *drop()* function to delete the rows from the working data frame.

A screenshot of a computer code

Description automatically generated

I then verified that there were still a few remaining outliers on the updated boxplot and z-score calculations. However, I decided to keep these values since these outliers still held significant value for further patient follow up or discussion. The distribution of values from the z-score histogram did confirm that the values seemed to be normally distributed after treatment. The overall treatment of this variable decreased our dataset size to 9500, a 5% decrease.

A screen shot of a graph

Description automatically generatedA screenshot of a computer

Description automatically generatedThere were also certain variables that did not have any outliers when creating the boxplot and calculating the z-scores. Variables that did not show signs of outliers such as ‘Age’ did not receive any outlier treatment.

The decision to retain outliers was made on the variables ‘Children’, ‘Income’, ‘Full\_meals\_eaten’, ‘VitD\_supp’, and ‘Additional\_charges’. This was due to different reasons pertaining to data integrity and reasonability. Retaining these outliers aided in maintaining the sample size of the data and preserving the diversity of the dataset. Being able to analyze the boxplot and z-score from these variables were important tools that allowed me to discern what data seemed still to be reasonable or not. The *info()* function was then used to verify the sample size of our data frame and the confirmation of no missing values.

The final step to clean the data was to re-express any categorical values that needed to be treated. I was able to find that the two categorical variables ‘Overweight’ and ‘Anxiety’ had numerical values instead of the string values “Yes” or “No”. In order to treat these values, I first confirmed the unique values A screenshot of a computer code

Description automatically generatedpresent within these variables using the ­*value\_counts()* function.

A close-up of a white card

Description automatically generatedI then used the *replace()* function to replace the values of “0” with “No” and the values of “1” with “Yes”.

To verify that the change was made, I again used the *value\_counts()* function to look at the unique values in the variables.

A screenshot of a computer code

Description automatically generated

After implementing the data cleaning treatment, we were left with a dataset that did not contain any null values and significant, or unreasonable outliers. I created and attached a Jupyter notebook file to the submission containing all the code used named ‘Performance Assessment.ipynb’. The final cleaned working and outlier data frames were exported to CSV files. These CSV files are attached to the submission as “performance\_assessment\_DS.csv” and “performance\_assessment\_outliers\_DS.csv”. To do this the *to\_csv()* function was used.

Some disadvantages to the method I used when searching for duplicates in the dataset include the inability to discern if there are duplicates that may have similar submissions or slight differences in values between two different patients. Because I worked with a large data set, some of these duplicates may have slipped through unnoticed. Using the *isnull()* function is an easy way to find missing values, but I think a big disadvantage is that there may be values that represent “empty” values within the data that can be categorized as missing. These “empty” values may consist of incoherent or illogical responses. Since the function can’t detect those, I believe that there is a chance that missing values may remain present unless the data itself is inspected.

I think that the univariate imputation method’s weakness is that it can generalize data that may skew the overall range of data if not used properly. I think that replacing these values with either the median, mean, or mode may hide certain possible anomalies and make the data generalized or too homogenous. This can also be said when treating outliers with the univariate imputation method as generalizing outliers may hide important data. By using the exclusion or removal method on outliers we also reduce the sample size and data diversity that the data frame may contain. I believe that removal of outliers does more harm than good for data, while excluding the outliers can help create a new data frame that can be further studied and corrected if possible.

If my final work was analyzed by a data analyst, I believe that they would want to know where 5% of the original data went. I think that they would be challenged to find out why there were several outliers that had higher than average vitamin D levels. By dropping the 500 rows of data, the analysis may be affected. I think that another challenge from my data is the possibility that trends may be hidden or lost after we used univariate imputation to treat the missing values. By assigning the missing values to the median, mean, or mode, we could have changed any statistical trends when comparing certain variables. I will say that creating the histograms and verifying the distribution of values will hopefully counter act most of the possible errors in statistical analysis of the data. Overall, in my previously stated research question, I believe that using the ‘Complication\_risk’ and ‘Initial\_days’ variables will be important to use when comparing the complication risks assigned to patients and the reason for their readmissions. Because the data was cleaned and most of the original data was retained, a data analyst will have a large sample size to work with.

**E. PCA**

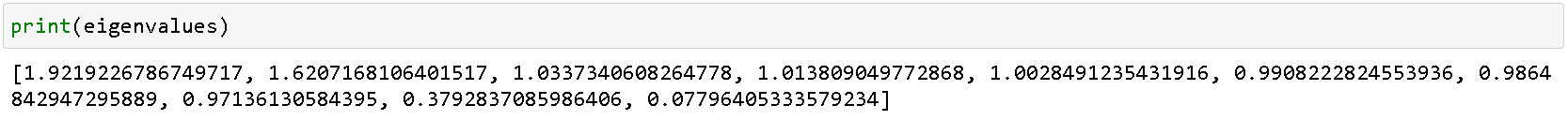
When conducting the PCA on the cleaned data frame, I used the variables ‘Children’, ‘Age’, ‘Income’, ‘VitD\_levels’, ‘Doc\_visits’, ‘Full\_meals\_eaten’, ‘VitD\_supp’, ‘Initial\_days’, ‘TotalCharge’, and ‘Additional\_charges’ as my components. PCA uses an orthogonal mathematical concept that helps find linear combinations of variables (“*What is Principal Component Analysis (PCA) & How to use it?”,* 2023). Performing PCA on these quantitative variables provided us with 10 components. The matrix below is the output of the PCA loadings matrix. A table with numbers and letters

Description automatically generated

The loadings matrix demonstrates the correlation coefficients between the variables that we chose for PCA and the factors. The correlation coefficient is the statistical measure of the relation between two variables, and it can help us identify what strength each variable has when contributing to the principal components analysis.

A graph with a red line

Description automatically generatedTo visualize the PCA, a scree plot was created that represented the number of components (in our case 10) and the eigenvalues that those components have. I will be utilizing the Kaiser Rule to find what components should be kept for statistical inference. According to the rule, components with eigenvalues equal to or greater than 1 should be kept (Villasante, 2023). A horizontal line was added at the 1 value on the y-axis to help consider what components should be kept according to the rule. On the x-axis the principal components are represented starting with the value 0; therefore, 0 represents PC1, 1 represents PC2, and so on. The principal components that will be retained are PC1, PC2, PC3, PC4, and PC5.

To make sure that we selected the correct principal components to retain, I simply used the *print()* function to print all the eigenvalues.

This verified that PC6, despite having a value close to 1, should not be retained as stated by the Kaiser Rule. It is important to know why some principal components are retained and others dropped. Looking back at the loading matrix I was able to find a correlation between the ‘Initial\_days’ and ‘TotalCharge’ variables on PC1. This can simply demonstrate that patients who stay at the hospital more days during their initial visit end up paying more money. On PC2 there is a correlation between the ‘Age’ and ‘Additional\_charges’ variables. This correlation may explain how patients who are older end up receiving additional treatments or services during their hospitalization. On PC3 there seems to be a slight correlation between the variables ‘Full\_meals\_eaten’ and ‘VitD\_supp’. It appears that patients who have less meals are taking more vitamin D supplements during their hospitalization. This may be due to patients who aren’t able to eat more meals, needing to take more vitamin supplements. As we go down the list, the correlation coefficient seems to decrease between the variables. PC4 may demonstrate a correlation between the ‘VitD\_levels’ and ‘Doc\_visits’ variables. An explanation may be that more doctor visits has helped patients monitor and increase their vitamin D levels. On PC5 the ‘Children’ and ‘Doc\_visits’ variables seem to have the most correlation in the principal component. It may seem that those with more children visit the doctor more often.

I believe that PCA is very important when looking at big data. Interpreting the meaning behind values and what trends lie in the data can be deciphered to help organizations make data-driven decisions. PCA can most certainly help hospitals learn certain trends that procedures or treatments may incur on patients. This may also explain and help predict what kind of help certain patients need according to their medical data or demographics. When working with a large sample size such as this data, PCA can reduce noise from the data and make it easier to find trends that would otherwise take a longer time to find. This allows us to understand the main variance in the data and understand the dimensionality by plotting variance vector (“*What is Principal Component Analysis (PCA) & How to use it?”,* 2023).

Cleaning the data may only be the first step when analyzing and presenting a whole data set. PCA such as the one I performed can be used to find trends as to what kind of patient demographics have a higher chance to be readmitted to hospitals. For example, as patients grow older, the chances of them being readmitted to hospitals grow. I believe that the data I have cleaned can help find trends as to how primary healthcare providers can more accurately assess the complication risk found in patients. It is known that health care providers have a difficult time accepting or acknowledging mistakes; however, I know that investigating and analyzing data such as this can help improve the patient care process overall (Helo and Moulton, 2017). It is important to help patients understand their risks and aid them in avoiding the need for hospital readmissions in the long term, especially emergency readmissions.

**References**

Chen, B. (2021, September 6). *9 Pandas value\_counts() tricks to improve your data analysis*. Medium. https://towardsdatascience.com/9-pandas-value-counts-tricks-to-improve-your-data-analysis-7980a2b46536

Helo S, Moulton CE (2017). *Complications: acknowledging, managing, and coping with human error*. Transl Androl Urol. 6(4):773-782. https://doi.org/10.21037/tau.2017.06.28.

Hunter, J., Dale, D., Droettboom, M., & the Matplotlib development team. (2023). *matplotlib.pyplot.hist#*. Matplotlib 3.7.2 documentation. https://matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.hist.html

Kleppen, E. (2023, May 11). *How To Find Outliers in Data Using Python (and How To Handle Them)*. CareerFoundry. https://careerfoundry.com/en/blog/data-analytics/how-to-find-outliers/

Krumholz HM, Wang K, Lin Z, et al (2017)*. Hospital-readmission risk—isolating hospital effects from patient effects.* New England Journal of Medicine 2017; 377:1055–64. https://doi.org/10.1056/NEJMsa1702321

Larose, C. D., & Larose, D. T. (2019). *Data Science Using Python and R*. Wiley Blackwell.

Levy F, Mareiniss D, Iacovelli C, Howard J (2010). *The Patient Safety and Quality Improvement Act of 2005: Preventing Error and Promoting Patient Safety*. Journal of Legal Medicine. 31(4):397-422. https://doi.org/10.1080/01947648.2010.535424

McGowan J, Wojahn A, Nicolini JR (2023). *Risk Management Event Evaluation and Responsibilities*. StatPearls. https://www.ncbi.nlm.nih.gov/books/NBK559326/#\_article-34645\_s2\_

Waskom M. (2022). *seaborn: statistical data visualization*. Seaborn. https://seaborn.pydata.org/

nkmk. (2021, January 7). pandas: Find and remove duplicate rows of DataFrame, Series. https://note.nkmk.me/en/python-pandas-duplicated-drop-duplicates/

National Institutes of Health. (2022, November 8). *Vitamin D*. NIH Office of Dietary Supplements. https://ods.od.nih.gov/factsheets/VitaminD-Consumer/

NumPy Developers. (2022). *The absolute basics for beginners#*. NumPy. https://numpy.org/doc/stable/user/absolute\_beginners.html

Pramoditha, R. (2023, February 2). *Principal Component Analysis (PCA) with Scikit-Learn*. Medium. https://medium.com/data-science-365/principal-component-analysis-pca-with-scikit-learn-1e84a0c731b0

Python Software Foundation. (2023, August 24). *Statistics - Mathematical statistics functions*. Python Documentation. https://docs.python.org/3/library/statistics.html

SujaDutta. (2019, July 4). *Python: Visualize missing values (nan) values using Missingno Library*. GeeksforGeeks. https://www.geeksforgeeks.org/python-visualize-missing-values-nan-values-using-missingno-library/

The Pandas Development Team. (2023, May 29). *Pandas*. PyPI The Python Package Index. https://pypi.org/project/pandas/

The SciPy Community. (2023). *Statistical Functions (scipy.stats)#*. SciPy Documentation. https://docs.scipy.org/doc/scipy/reference/stats.html

WGU. (2022, November 16). *R or Python*. Western Governors University. https://www.wgu.edu/online-it-degrees/programming-languages/r-or-python.html

*What is Principal Component Analysis (PCA) & How to use it?*. Bigabid. (2023, February 8). https://www.bigabid.com/what-is-pca-and-how-can-i-use-it/

Villasante, P. (2023, May 30). *Choose optimal number of components for PCA (example)*. Statistics Globe. https://statisticsglobe.com/choose-optimal-number-components-pca