Andreas Dunn

WGU, D206 PA

8/6/2021

Part 1

A.

In this analysis, I will be using a summation of the following variables: HighBlood, Stroke, Overweight, Arthritis, Diabetes, Hyperlipidemia, BackPain, Anxiety, Allergic_rhinitis, Reflux_esophagitis, and Asthma into a new variable called health_score. I will also be using the variable Doc_visits. My goal is to answer the following question: is there a positive correlation between Doc_visits + health_score and readmission rates for the hospital. This way I can answer an important business question: does an increase in the number of Doc_visits plus a high number of the health_score column lead to a greater chance of readmission rates. In other words, is a high number for both Doc_visits and health_score positively correlated with readmission rates?

B.

I will be running analysis on the following variables: Doc_visits, HighBlood, Stroke, Overweight, Arthritis, Diabetes, Hyperlipidemia, BackPain, Anxiety, Allergic_rhinitis, Reflux_esophagitis, and Asthma. Doc_visits is a number while all the other data are yes or no answers except for Overweight and Anxiety, which are either 1 or 0. Since some of the data is numeric, and some is categorical, I will be re-coding all the yes or no answers into numbers 1 or 0 (1 for yes, 0 for no). This way all our independent variables will be numeric and our correlation will be consistent across all columns. We will need to create a new column that stores the "health score" from the other columns which will be called health_score. Our analysis will therefore be looking at explaining the ReAdmis data by using Doc_visits and health_score.

Part 2

C.

I will first be re-coding all the categorical data in the columns HighBlood, Stroke, Overweight, Arthritis, Diabetes, Hyperlipidemia, BackPain, Anxiety, Allergic_rhinitis, Reflux_esophagitis, and Asthma to numeric data with yes answers being a 1 and no answers being a 0. There are a few Python functions needed to re-code this data into the new column called health_score, which will then have a box-plot and whisker plot to show the distribution of the data. Then I will repeat the process on Doc_visits to show how the data is spread out. I will also be calculating Z-scores on both of the variables mentioned to identify anomalies/outliers.

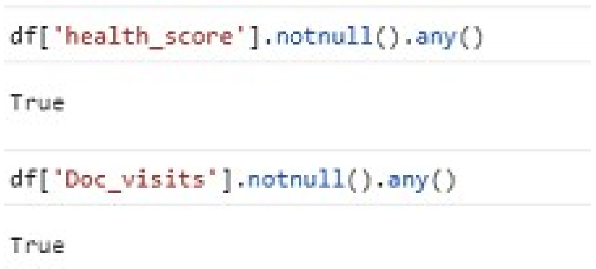
Since I will be re-coding all the data from categorical to numeric and then making a new column, it would make sense to drop all rows that have a null value. I will be using Python methods to detect and drop any rows that contain null values, missing data, and altering inconsistent datatypes. Judging from the data at hand, since the data dictionary does NOT explain whether 1 or 0 represents either a yes or no answer to both the Overweight and Anxiety columns, both of these columns need to be excluded from the analysis; they will not be included in the health_score column. Next I will be calculating Z-scores on Doc_visits and health_score. I will be analyzing all the data's Z-scores and dropping any individual rows that have a Z-score higher than 3 or less than -3.

Using Z-scores to determine outliers is a very common and strong tool when identifying data that doesn't meet the *general* trend. Since our goal is to find general trends within the data, using Z-scores in combination with visual plots should give us a powerful understanding of how to lower readmission rates using these two variables.

Since all the data are numeric and have no null values, I will assume that the data was recorded without error and truly represents the population in question. Having no null values makes our analysis strong since it suggests that the hospital did a good job recording the information and that the data reflects reality. The quality of the data is therefore strong and will provide us with an insightful analysis to answer the research question.

I will be using Python with the libraries of pandas, numpy, matplotlib, and scipy. These libraries give the Python language a way to interpret the data and show trends, as well as clean the data so that our analysis is consistent and without error. I will not being using R for any of the analysis since Python has all the necessary libraries included in the Anaconda package. All code is written and verified in Jupyter notebook.

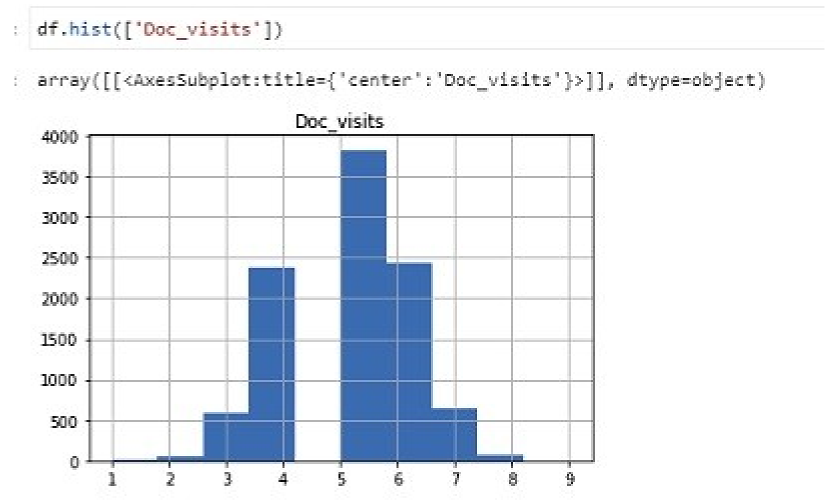
Here is the code for detecting our outliers where I checked for null values:



```
df['health_score'].notnull().any()

True

df['Doc_visits'].notnull().any()

True
```

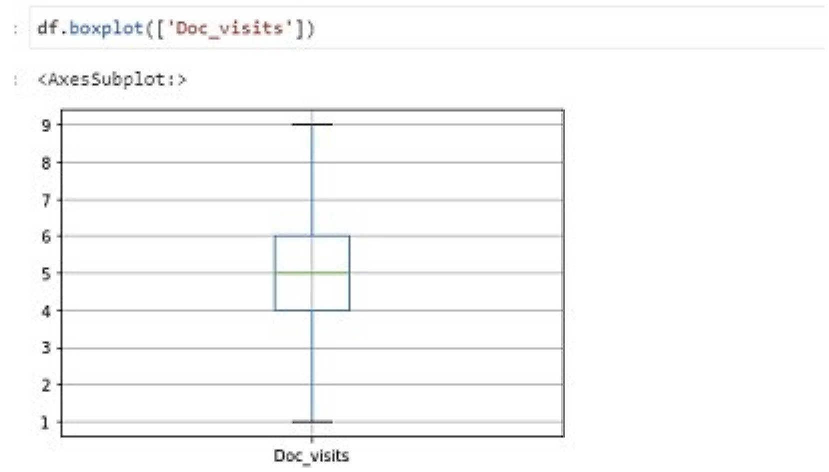
The notnull().any() function will check to see if there are any not null values (which means we have all data that is not null) and returns true if it is true.

Here are box plots and histograms for the Doc_visits column so we can graphically represent the distribution of the data:



```
df.hist(['Doc_visits'])
```

```
array([[<AxesSubplot:title={'center':'Doc_visits'}>]], dtype=object)
```



```
df.boxplot(['Doc_visits'])
```

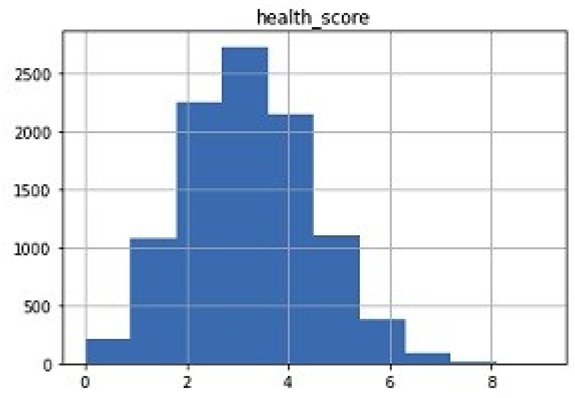
```
<AxesSubplot:>
```

We can see from the code that our data is mostly a normal distribution with some gaps from 4-5 Doc_visits.

Below we have a box plot and histogram for our newly made column health_score:

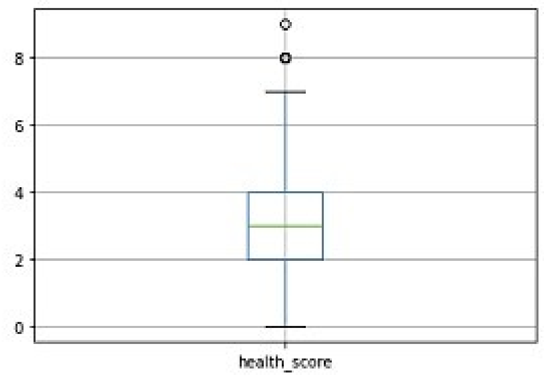
```
df.hist(['health_score'])
```

```
array([[<AxesSubplot:title={'center':'health_score'}>]], dtype=object)
```



```
df.boxplot(['health_score'])
```

```
<AxesSubplot:>
```



Again the histogram shows a basic normal distribution implying that most of our sample population is less than one standard deviation from the mean.

Part 3

D.

Firstly, I dropped both the Anxiety column and the Overweight column from the analysis; they weren't included at all. Then I check for null values using the isnull().any() function which returns true if the columns have no null values; there were no null values or missing data found in the data set.

Next, I re-code all categorical data into numeric data and then sum them to create our new column called health_score. I then show how this data is distributed with a box plot and histogram and move onto the next stage which is calculating Z-scores and dropping outliers.

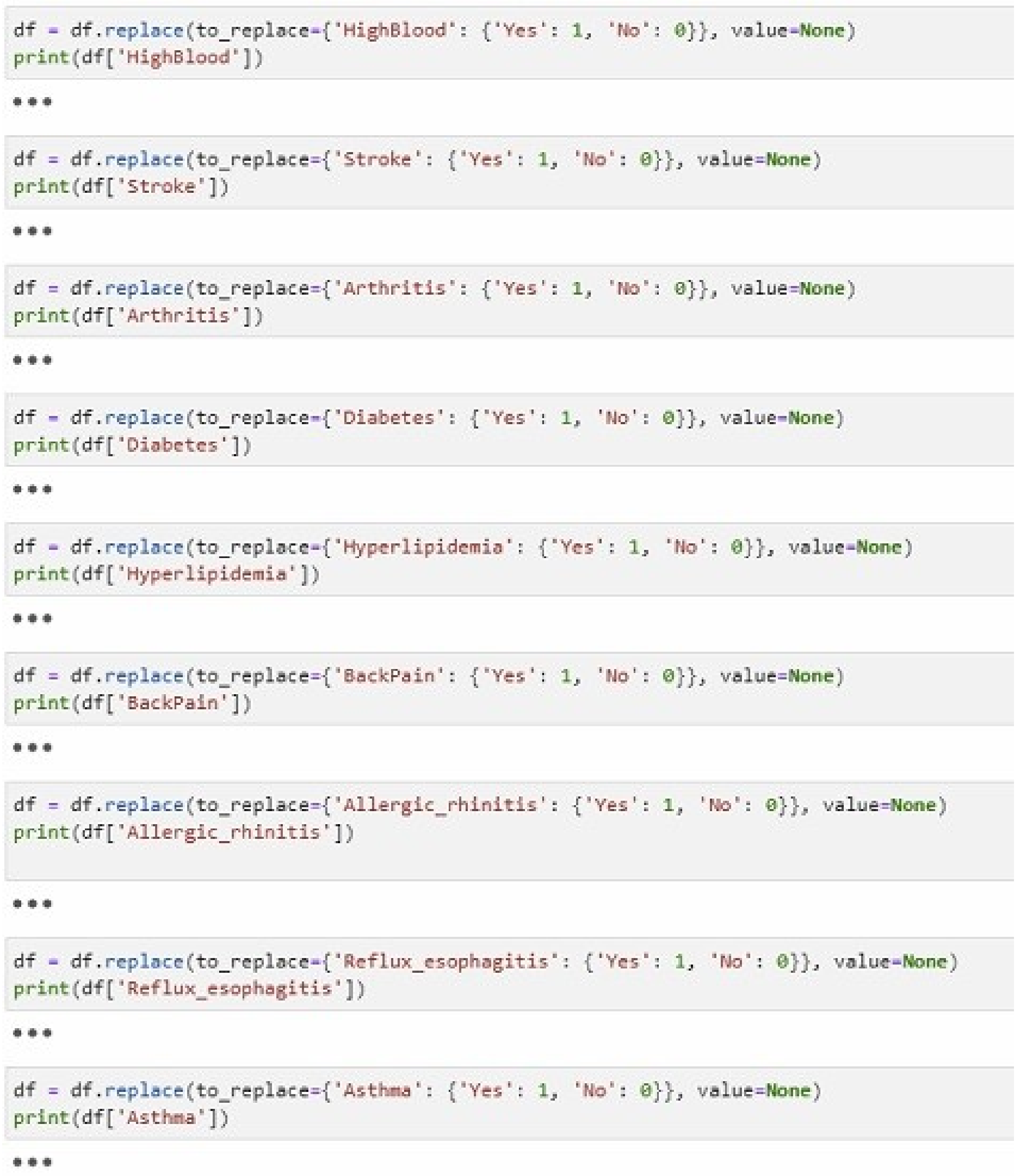
In order to calculate Z-scores I had to make a new column that stores Z-scores. To get a new Z-score column for health_score, we have to create an array out of the health_score column, save it to a new variable, and then run the stats.zscore method on the new array. We then save the output to a new column and add it to our data frame.

We repeat the same process that we did for health_score but now for Doc_visits and get a new column. The columns are titled zscore_hs (stores Z-scores for health_score) and zscore_dv (stores Z-Scores for Doc_visits).

One problem I encountered was that, after we had created our new column to store Z-scores, the datatype for the new columns were floats and not ints. I changed the datatype back to int using the astype() method and now all our columns are ready for the next stage.

Next I run the drop() method and remove any rows that have a Z-score higher than the absolute value of 3, which resulted in only one row being removed and this is verified by running the info() method which showed that we still have 9,999 rows of data remaining. This is good news since our data has now been fully cleaned and all data outliers/anomalies have been removed.

Below is where I re-code all categorical data to numeric data and create our new health_score column:



```
df = df.replace(to_replace={'HighBlood': {'Yes': 1, 'No': 0}}, value=None)
print(df['HighBlood'])
```

•••

```
df = df.replace(to_replace={'Stroke': {'Yes': 1, 'No': 0}}, value=None)
print(df['Stroke'])
```

•••

```
df = df.replace(to_replace={'Arthritis': {'Yes': 1, 'No': 0}}, value=None)
print(df['Arthritis'])
```

•••

```
df = df.replace(to_replace={'Diabetes': {'Yes': 1, 'No': 0}}, value=None)
print(df['Diabetes'])
```

•••

```
df = df.replace(to_replace={'Hyperlipidemia': {'Yes': 1, 'No': 0}}, value=None)
print(df['Hyperlipidemia'])
```

•••

```
df = df.replace(to_replace={'BackPain': {'Yes': 1, 'No': 0}}, value=None)
print(df['BackPain'])
```

•••

```
df = df.replace(to_replace={'Allergic_rhinitis': {'Yes': 1, 'No': 0}}, value=None)
print(df['Allergic_rhinitis'])
```

•••

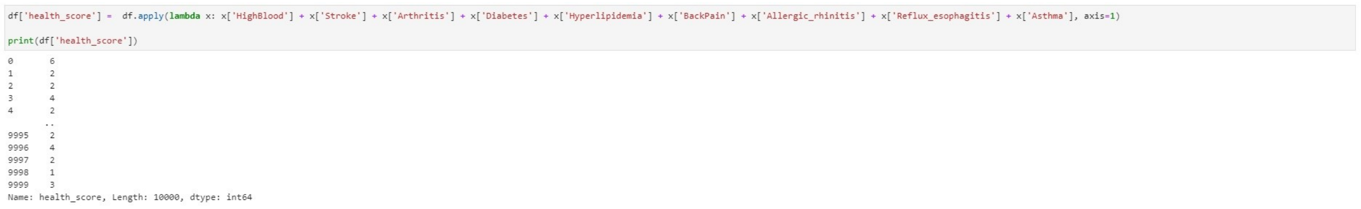
```
df = df.replace(to_replace={'Reflux_esophagitis': {'Yes': 1, 'No': 0}}, value=None)
print(df['Reflux_esophagitis'])
```

•••

```
df = df.replace(to_replace={'Asthma': {'Yes': 1, 'No': 0}}, value=None)
print(df['Asthma'])
```

•••

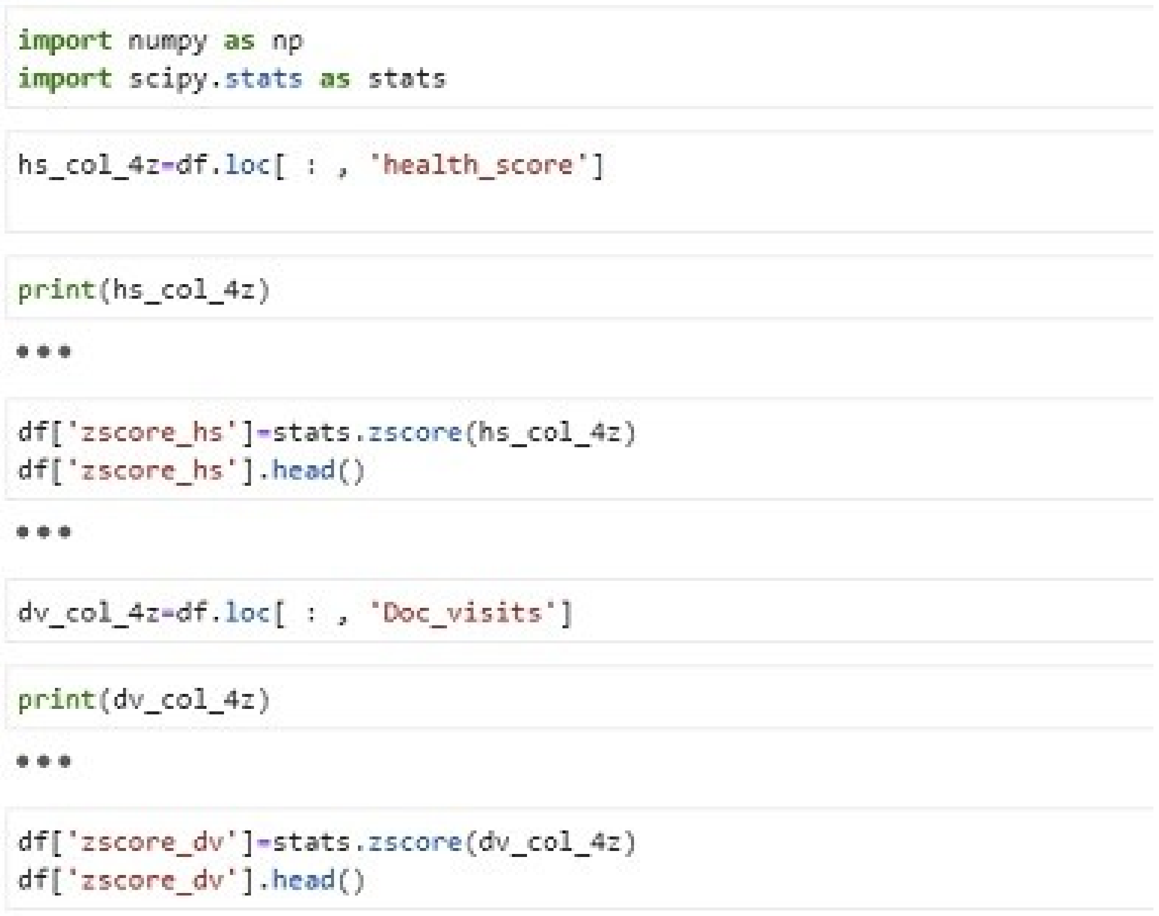
Code for creating the health_score column:



```
df['health_score'] =  df.apply(lambda x: x['HighBlood'] + x['Stroke'] + x['Arthritis'] + x['Diabetes'] + x['Hyperlipidemia'] + x['BackPain'] + x['Allergic_rhinitis'] + x['Reflux_esophagitis'] + x['Asthma'], axis=1)

print(df['health_score'])
```

```
0       6
1       2
2       2
3       4
4       2
       ..
9995    2
9996    4
9997    2
9998    1
9999    3
Name: health_score, Length: 10000, dtype: int64
```

Next I create a Z-score column for both Doc_visits and health_score:



```
import numpy as np
import scipy.stats as stats
```

```
hs_col_4z=df.loc[ : , 'health_score']
```

```
print(hs_col_4z)
```

•••

```
df['zscore_hs']=stats.zscore(hs_col_4z)
df['zscore_hs'].head()
```

•••

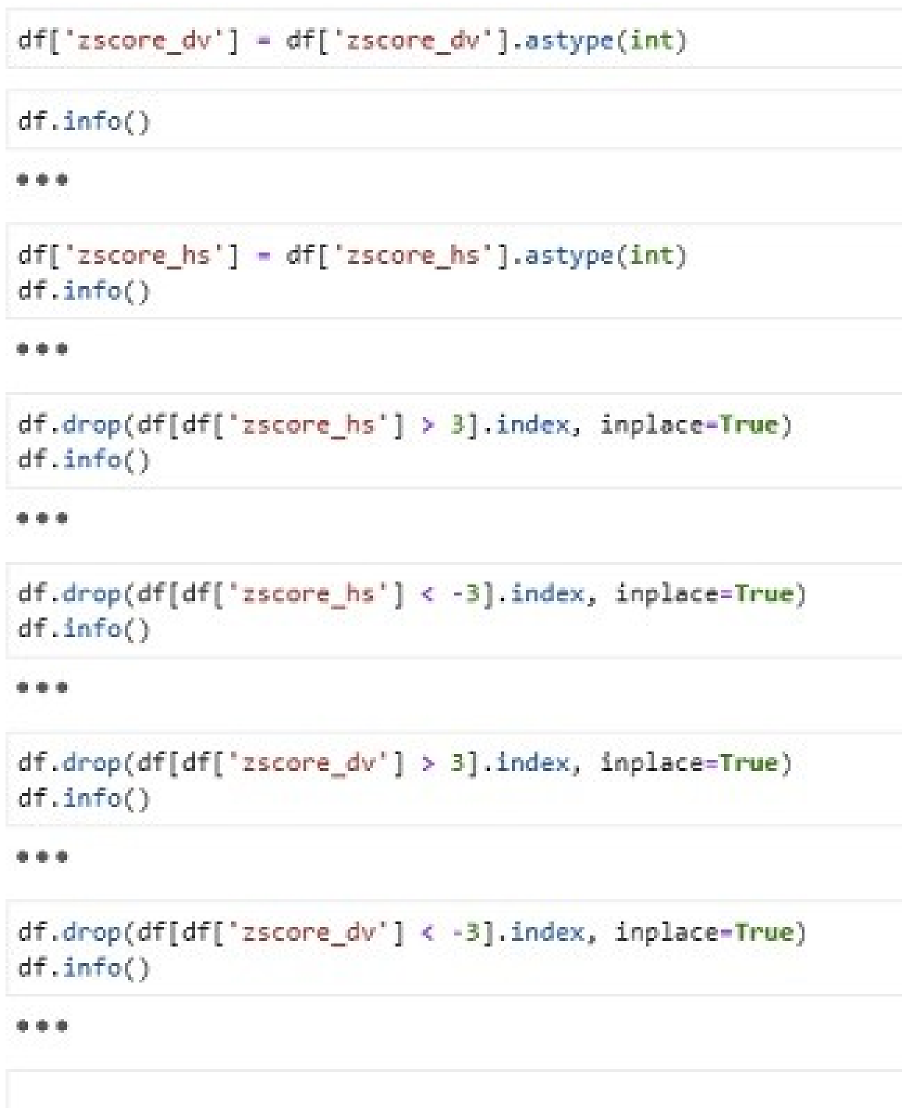
```
dv_col_4z=df.loc[ : , 'Doc_visits']
```

```
print(dv_col_4z)
```

•••

```
df['zscore_dv']=stats.zscore(dv_col_4z)
df['zscore_dv'].head()
```

Next I had to convert both the new Z-score columns from a float to an int to make our datatypes consistent and then filter for any int values that were more than the absolute value of 3 and drop those rows:



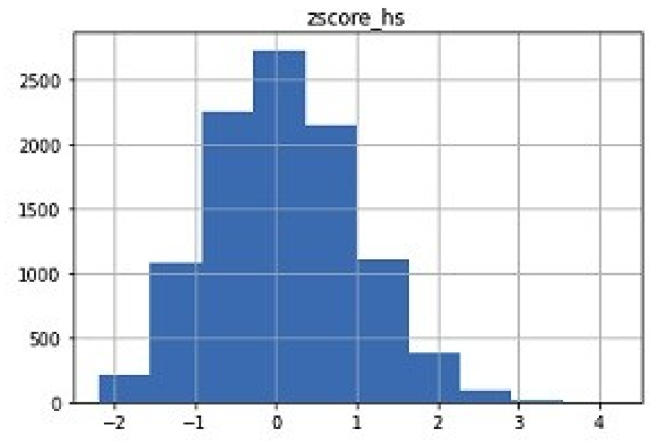
```
df['zscore_dv'] = df['zscore_dv'].astype(int)

df.info()

•••

df['zscore_hs'] = df['zscore_hs'].astype(int)
df.info()

•••

df.drop(df[df['zscore_hs'] > 3].index, inplace=True)
df.info()

•••

df.drop(df[df['zscore_hs'] < -3].index, inplace=True)
df.info()

•••

df.drop(df[df['zscore_dv'] > 3].index, inplace=True)
df.info()

•••

df.drop(df[df['zscore_dv'] < -3].index, inplace=True)
df.info()

•••
```

I run ds.info() after each change so that I can verify that the values were changed and dropped correctly.

Below are the histogram charts for our Z-score columns so that we can better understand how the data is distributed:

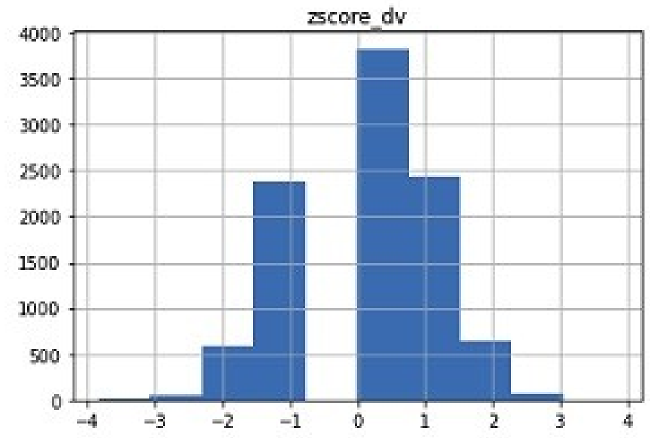
```
df.hist(['zscore_hs'])
```

```
array([[<AxesSubplot:title={'center':'zscore_hs'}>]], dtype=object)
```



```
df.hist(['zscore_dv'])
```

```
array([[<AxesSubplot:title={'center':'zscore_dv'}>]], dtype=object)
```



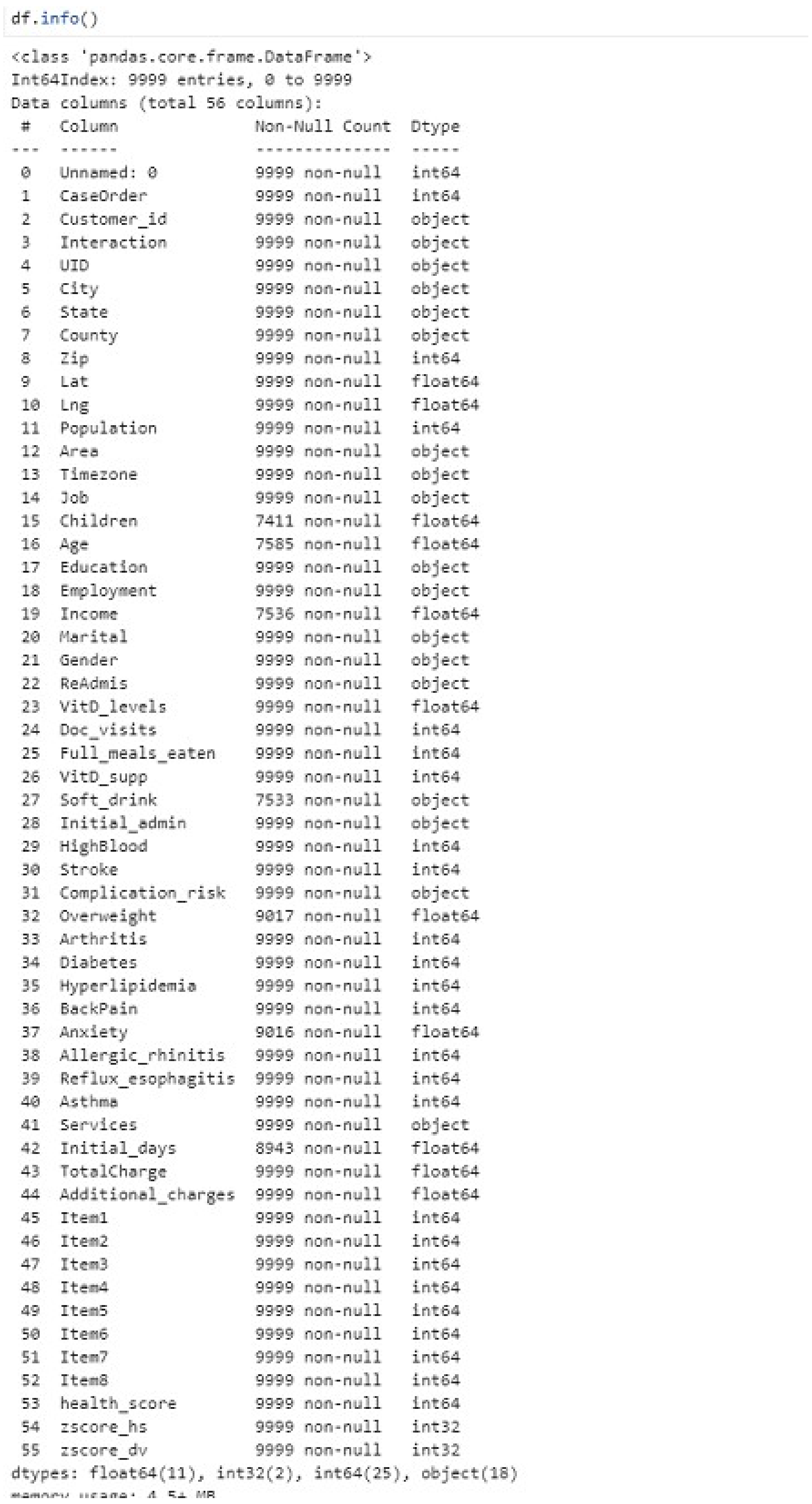
In summary, we first took our raw data and excluded two columns completely (Anxiety and Overweight) because the data dictionary did not indicate whether or not a 1 or 0 represented a yes or a no answer. Then we filtered for any missing data or null values for the remaining columns and found none.

We then took our remaining columns (HighBlood, Stroke, Arthritis, Diabetes, Hyperlipidemia, BackPain, Reflux_esophagitis, and Asthma) and transformed all 'Yes' rows to be the number 1 and all "No' rows to be number 0. We then summed all those columns into a new column labeled health_score so that we could run a Z-score function on the data and identify outliers.

Next we created a new column to represent the Z-scores for all the data from health_score and Doc_visits then dropped all rows for both those columns in which the Z-score was higher than the absolute value of 3, resulting in only 1 row out of 10,000 being removed.

Some of the limitations to this method of cleaning data is that using, or relying entirely on, Z-scores can be problematic when we run further analysis on the data. Z-scores assume a normal distribution of our variables which may not always be the case. The limitations of using a Z-score approach to removing outliers is that it removes some of the extreme cases from the analysis which may be helpful in discovering the causal relationship between health_score + Doc_visits and readmission rates. Although Z-scores aren't perfect, we only had to remove 1 row of data in this analysis which I think gives strength to our summary.

Below is the cleaned data set: it has no null values, it has consistent datatypes, and all outliers removed according to Z-score (1 row in total).



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9999 entries, 0 to 9999
Data columns (total 56 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          9999 non-null   int64  
 1   CaseOrder           9999 non-null   int64  
 2   Customer_id         9999 non-null   object 
 3   Interaction         9999 non-null   object 
 4   UID                 9999 non-null   object 
 5   City                9999 non-null   object 
 6   State               9999 non-null   object 
 7   County              9999 non-null   object 
 8   Zip                 9999 non-null   int64  
 9   Lat                 9999 non-null   float64
 10  Lng                 9999 non-null   float64
 11  Population          9999 non-null   int64  
 12  Area                9999 non-null   object 
 13  Timezone            9999 non-null   object 
 14  Job                 9999 non-null   object 
 15  Children            7411 non-null   float64
 16  Age                 7585 non-null   float64
 17  Education           9999 non-null   object 
 18  Employment          9999 non-null   object 
 19  Income              7536 non-null   float64
 20  Marital             9999 non-null   object 
 21  Gender              9999 non-null   object 
 22  ReAdmis             9999 non-null   object 
 23  VitD_levels         9999 non-null   float64
 24  Doc_visits          9999 non-null   int64  
 25  Full_meals_eaten    9999 non-null   int64  
 26  VitD_supp           9999 non-null   int64  
 27  Soft_drink          7533 non-null   object 
 28  Initial_admin       9999 non-null   object 
 29  HighBlood           9999 non-null   int64  
 30  Stroke              9999 non-null   int64  
 31  Complication_risk   9999 non-null   object 
 32  Overweight          9017 non-null   float64
 33  Arthritis           9999 non-null   int64  
 34  Diabetes            9999 non-null   int64  
 35  Hyperlipidemia      9999 non-null   int64  
 36  BackPain            9999 non-null   int64  
 37  Anxiety             9016 non-null   float64
 38  Allergic_rhinitis   9999 non-null   int64  
 39  Reflux_esophagitis  9999 non-null   int64  
 40  Asthma              9999 non-null   int64  
 41  Services            9999 non-null   object 
 42  Initial_days        8943 non-null   float64
 43  TotalCharge         9999 non-null   float64
 44  Additional_charges  9999 non-null   float64
 45  Item1               9999 non-null   int64  
 46  Item2               9999 non-null   int64  
 47  Item3               9999 non-null   int64  
 48  Item4               9999 non-null   int64  
 49  Item5               9999 non-null   int64  
 50  Item6               9999 non-null   int64  
 51  Item7               9999 non-null   int64  
 52  Item8               9999 non-null   int64  
 53  health_score        9999 non-null   int64  
 54  zscore_hs           9999 non-null   int32  
 55  zscore_dv           9999 non-null   int32  
dtypes: float64(11), int32(2), int64(25), object(18)
memory usage: 4.5+ MB
```

In conclusion, the result of this cleaning has led to two columns: health_score and Doc_visits which have been cleaned of any missing values, null values, and outliers/anomalies. Our data is now cleaned and ready for the next step in the data analytics life cycle.

E.

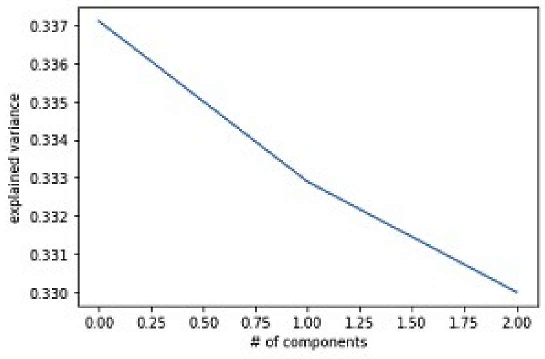
In this section I am going to be doing something similar to what I did before by grouping data but I will be doing the grouping according to PCA. The components in the data set are going to be the column I created labeled health_score, as well as Doc_visits, and VitD_levels. Using 3 components total we can then analyze the data using these variables and see if we can learn something about readmissions using PCA.

PCA has to use integers or floating ints, so we are somewhat limited in which variables we can choose, I chose these variables because I thought they would be the best in explaining and answering our research question. The PCA shows us that PC1 has the most weight on explaining the variance on Doc_visits while PC3 has the most weight on explaining our other two columns (health_score and VitD_levels). The reason we should use PCA is the it can sometimes better explain the trends we see in data and help simplify our analysis. We can see from our scree plot that we really only need 1 component to explain most of the variance in the data because our other principal components have eigenvalues under 1.

This analysis may not benefit a ton from using PCA because a lot of our data is categorical and not numeric. While we can estimate some numerical values out of our categorical data (like I did for health_score) it may not always be the most comprehensive approach for this data set. PCA works great when you have a lot of numeric variables that can be logically grouped together which makes it a strong tool for performing data analytics.

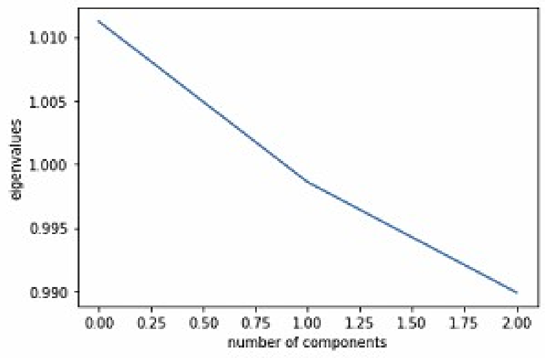
Below is the code for and the graphs showing the PCA:

```
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('# of components')
plt.ylabel('explained variance')
plt.show()
```



```
cov_matrix = np.dot(newdf_normalized.T, newdf_normalized) / newdf.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for eigenvector in pca.components_]
```

```
plt.plot(eigenvalues)
plt.xlabel('number of components')
plt.ylabel('eigenvalues')
plt.show()
```



```
loadings = pd.DataFrame(pca.components_.T,
columns=['pc1', 'pc2', 'pc3'],
index=newdf.columns)
loadings
```

	pc1	pc2	pc3
Doc_visits	0.412976	-0.856269	0.310248
health_score	-0.683552	-0.066306	0.726884
VitD_levels	0.601837	0.512256	0.612687

Part 4.

G. No third party code was used other than that contained within the WGU lessons or methods found on Python panda docs and stack overflow:

https://pandas.pydata.org/docs/user_guide/index.html

https://stackoverflow.com/

H. No other works cited.