Part I: Research Question

A. Question

Can we identify the variables most influential in the readmittance of patients following a visit at our hospital?

B. Variables

CaseOrder: Integer that maintains initial order of raw data Customer_id: String that provides unique key to patient Interaction: String that provides unique key for patient transactions, procedures, and admissions UID: String that provides unique key for patient transactions, procedures, and admissions City: String that indicates patient city of residence on billing statement State: String that indicates patient state of residence on billing statement County: String that indicates patient county of residence on billing statement Zip: Integer that indicates patient zip code of residence on billing statement Lat: Float that indicates Latitude based on billing statement address Lng: Float that indicates Longitude based on billing statement address Population: Integer that indicates population based on census data within one-mile radius of address Area: String that indicates area type based on census Timezone: String that indicates timezone based on patient provided residence Job: String that indicates patient job as provided by admissions information Children: Float that indicates number of children provided by admissions information Age: Float that indicates age of patient provided by admissions information Education: String that indicates highest earned level of education provided by admissions information Employment: String that indicates current employment status provided by admissions information Income: Float that indicates annual income of patient (or primary insurance holder) provided by admissions information Marital: String that indicates current marital status of patient (or primary insurance holder) provided by admissions information Gender: String that indicates self-identified gender ReAdmis: String that indicates whether the patient was readmitted within a month of relevant visit VitD_levels: Float that indicates patient vitamin D levels measured in ng/ml Doc_visits: Integer that indicates the number of times that the PCP (Primary Care Physician) visited the patient during their initial stay Full_meals_eaten: Integer that indicates the number of full meals the patient ate during hospitilization (partial = 0) VitD_supp: Integer that indicates the number of times vitamin D supplements were administered to patient Soft_drink: String that indicates if a patient drinks 3 or more sodas in a day frequently Initial_admin: String that indicates the route in which a patient was admitted into the hospital HighBlood: String that indicates if the patient has high blood pressure Stroke: String that indicates if the patient has had a stroke Complication_risk: String that indicates the level of risk for complication associated with the patient Overweight: Float that indicates if the patient is overweight Arthritis: String that indicates if the patient has arthritis Diabetes: String that indicates if the patient has diabetes Hyperlipidemia: String that indicates if the patient has hyperlipidemia BackPain: String that indicates if the patient has chronic back pain Anxiety: Float that indicates if the patient has an anxiety disorder Allergic_rhinitis: String that indicates if the patient has allergic rhinitis Reflux_esophagitis: String that indicates if the patient has reflux esophagitis Asthma: String that indicates if the patient if the patient has asthma Services: String that indicates the primary service a patient received during their stay Initial_days: Float that indicates the number of days the patient stayed during the initial visit TotalCharge: Float that indicates the average cost per day (Total Cost/# of days) Additional_charges: Float that indicates the average cost of miscellaneous services received during stay Item1: Integer that indicates survey answer about the importance of "Timely Admission" (Scale of 1 most - 8 least) Item2: Integer that indicates survey answer about the importance of "Timely Treatment" (Scale of 1 most - 8 least) Item3: Integer that indicates survey answer about the importance of "Timely Visits" (Scale of 1 most - 8 least) Item4: Integer that indicates survey answer about the importance of "Reliability" (Scale of 1 most - 8 least) Item5: Integer that indicates survey answer about the importance of "Options" (Scale of 1 most - 8 least) Item6: Integer that indicates survey answer about the importance of "Hours of Treatment" (Scale of 1 most - 8 least) Item7: Integer that indicates survey answer about the importance of "Courteous Staff" (Scale of 1 most - 8 least) Item8: Integer that indicates survey answer about the importance of "Evidence of Active Listening from Doctor" (Scale of 1 most - 8 least)

Part II: Data-Cleaning Plan

C. Plan Explanation

C1.

- Orient Myself to the data
- Value Counts
- Missing Values
- Standardized numeric columns
- Categorical variables to Numeric
- Histograms and Boxplots (Outlier Detection)
- PCA

C2.

This data cleaning plan is based up on the Data Preparation Phase explained in the text "Data Science Using Python and R." I have added a few pieces to be more specific about my process that is based on my experience as a full-time Data Analyst.

The first thing is orienting myself to the data. In general - this is just the step where I am going to look at the raw data itself. What is my initial input and what stands out? What are the data types I am working with? etc.

The second step is value counts. In general - this will help me to identify unique identifier columns that may not be beneficial in a model. This will also give me a high level view of distributions for both categorical and numeric values.

Missing values is the step in which I will utilize multiple imputation to address missing values. ML models typically do not account for missing data (there is nuance in this), so you need to assign values in every row of every column. I will use IterativeImputer from sklearn to fill in our missing measures. IterativeImputer is similar to MICE within R. It is going to iterate through the dataset multiple times and establish estimations of the missing value based upon the other variables. (Scikit-learn)

Standardized numeric columns will be two fold - there will be some mapping and then standardizing the columns with sklearns standardscaler. This module will center distributions around 0 as the mean. This will be necessary for PCA and will be easier to evaluate outliers and distribution since the scale will be standardized. (Larose, 2019)

Converting categorical variables to numeric is necessary on a couple levels. We need this for our IterativeImputer so that categorical variables can be evaluated as a part of that model. Also - again this will be necessary in future ML applications. (Larose, 2019)

Histograms and boxplots will allow me to visualize the distributions and outliers in a graphic format. This will help to inform decisions on how to deal with outliers. Do certain observations need to be excluded?

PCA is really the end goal of this project. PCA is a form of dimensionality reduction. "Principla components analysis (PCA) seeks to account for the correlation structure of a set of predictor variables, using a smaller set of uncorrelated linear combinations of these variables, called components," (Larose, 2019). I find that the easiest way to conceptualize this is as the vector transformation along a single line to explain the maximal variance. With this method - we can reduce our necessity for all variables and explain most of the variance in a dataset.

C3.

I am utilizing Python as my programming language of choice in this project. Python is a highly flexible programming language that handles data really well. It is consistently one of the most used programming languages in the world (Eastwood, 2020). In thinking of long-term strategy - it is much simpler to put a Python built model into production within an existing application environment. This allows the data analyst to not just be putting together presentations and visuals, but to truly contribute in production. R is great at statistical programming - it is very often used in research, because it is built for that very implementation. So, there is certainly in argument for its utilization in any project like this. But, I prefer Python for its flexibility. You can create web applications, desktop applications, api's and the list goes on. So, if we ever needed to pivot this analysis into a different domain - it would be far easier to do in Python than R.

As for packages - I am using several. Pandas is one of the most popular packages in Python and it deals with tabular data. It effectively allows you to interact with the data in a format we are traditionally used to - with column headers and row numbers. Numpy is a package that I will use for some calculations and also interacting with the data at certain points as an array. Matplotlib is the package that will be used for visualizations. Sklearn is a scientific package for Python meant to implement the various stages of Machine Learning. So, sklearn will be used in imputation, transformation, and ultimately PCA. Bioinfokit is an additional package that I will be using for some visualization of PCA.

In [108]:

```python
#Package imports
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib import pyplot
from sklearn import preprocessing
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [109]:
```
#Read in dataset and set index to first column that already has indexes.

Med_df = pd.read_csv(r'C:\Users\jacob.colp.UNITY\Downloads\Medical Data I

Med_df = Med_df.reset_index(drop = True)
```

In [110]:
```
#Show relevant statistical information about numeric columns.

Med_df.describe()
```

Out[110]:

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

8 rows × 25 columns

In [111]:

```
#Convert yes/no columns to binary. I also converted Complication Risk to

for x in Med_df.columns:
    if  Med_df[x].dtype == object:
        Med_df[x].replace({'Yes':1,'yes':1,'No':0,'no':0}, inplace = Tru

Med_df.Complication_risk.replace({'High':3, 'Medium':2, 'Low':1}, inplac

Med_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 52 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   CaseOrder           10000 non-null  int64  
 1   Customer_id         10000 non-null  object 
 2   Interaction         10000 non-null  object 
 3   UID                 10000 non-null  object 
 4   City                10000 non-null  object 
 5   State               10000 non-null  object 
 6   County              10000 non-null  object 
 7   Zip                 10000 non-null  int64  
 8   Lat                 10000 non-null  float64
 9   Lng                 10000 non-null  float64
 10  Population          10000 non-null  int64  
 11  Area                10000 non-null  object 
 12  Timezone            10000 non-null  object 
 13  Job                 10000 non-null  object 
 14  Children            7412 non-null   float64
 15  Age                 7586 non-null   float64
 16  Education           10000 non-null  object 
 17  Employment          10000 non-null  object 
 18  Income              7536 non-null   float64
 19  Marital             10000 non-null  object 
 20  Gender              10000 non-null  object 
 21  ReAdmis             10000 non-null  int64  
 22  VitD_levels         10000 non-null  float64
 23  Doc_visits          10000 non-null  int64  
 24  Full_meals_eaten    10000 non-null  int64  
 25  VitD_supp           10000 non-null  int64  
 26  Soft_drink          7533 non-null   float64
 27  Initial_admin       10000 non-null  object 
 28  HighBlood           10000 non-null  int64  
 29  Stroke              10000 non-null  int64  
 30  Complication_risk   10000 non-null  int64  
 31  Overweight          9018 non-null   float64
 32  Arthritis           10000 non-null  int64  
 33  Diabetes            10000 non-null  int64  
 34  Hyperlipidemia      10000 non-null  int64  
 35  BackPain            10000 non-null  int64  
 36  Anxiety             9016 non-null   float64
 37  Allergic_rhinitis   10000 non-null  int64  
 38  Reflux_esophagitis  10000 non-null  int64  
 39  Asthma              10000 non-null  int64  
```

```
 40  Services           10000 non-null  object 
 41  Initial_days       8944 non-null   float64
 42  TotalCharge        10000 non-null  float64
 43  Additional_charges  10000 non-null  float64
 44  Item1              10000 non-null  int64  
 45  Item2              10000 non-null  int64  
 46  Item3              10000 non-null  int64  
 47  Item4              10000 non-null  int64  
 48  Item5              10000 non-null  int64  
 49  Item6              10000 non-null  int64  
 50  Item7              10000 non-null  int64  
 51  Item8              10000 non-null  int64  
dtypes: float64(12), int64(25), object(15)
memory usage: 4.0+ MB
```

In [112]:

```
#This will tell me the number of null values in each column

Med_df.isnull().sum()
```

Out[112]:

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

```
Item8                 0
dtype: int64
```

In [113]:

```
#Find unique columns
for x in Med_df:
    print(x+': '+str(Med_df[x].is_unique))
```

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

In [114]:

```
#I am dropping Job, Income, and Marital Status because those all may not

Med_df.drop(['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'Job', 'I

#Rename all Item survey columns to their question topic.

Med_df.rename({'Item1':'Survey_Timely_Admission', 'Item2':'Survey_Timely_
           'Item7':'Survey_Courteous_Staff', 'Item8':'Survey_Evidence_of_

Med_df.columns
```

Out[114]:

```
Index(['City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Are
a',
       'Timezone', 'Children', 'Age', 'Education', 'Employment', 'Gender',
       'ReAdmis', 'VitD_levels', 'Doc_visits', 'Full_meals_eaten', 'VitD_su
pp',
       'Soft_drink', 'Initial_admin', 'HighBlood', 'Stroke',
       'Complication_risk', 'Overweight', 'Arthritis', 'Diabetes',
       'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic_rhinitis',
       'Reflux_esophagitis', 'Asthma', 'Services', 'Initial_days',
       'TotalCharge', 'Additional_charges', 'Survey_Timely_Admission',
       'Survey_Timely_Treatment', 'Survey_Timely_Visits', 'Survey_Reliabili
ty',
       'Survey_Options', 'Survey_Hours_of_Treatment', 'Survey_Courteous_Sta
ff',
       'Survey_Evidence_of_Active_Listening'],
      dtype='object')
```

In [115]:
```
#I am converting categorical values into the "category" datatype from pa

categorical_variables = ['City', 'State', 'County', 'Zip', 'Area', 'Time

for x in categorical_variables:
    Med_df[x] = Med_df[x].astype('category')

Med_df.dtypes
```

Out[115]:
```
City                                  category
State                                 category
County                                category
Zip                                   category
Lat                                    float64
Lng                                    float64
Population                               int64
Area                                  category
Timezone                              category
Children                               float64
Age                                    float64
Education                             category
Employment                            category
Gender                                category
ReAdmis                                  int64
VitD_levels                            float64
Doc_visits                               int64
Full_meals_eaten                         int64
VitD_supp                                int64
Soft_drink                             float64
Initial_admin                         category
HighBlood                                int64
Stroke                                   int64
Complication_risk                        int64
Overweight                             float64
Arthritis                                int64
Diabetes                                 int64
Hyperlipidemia                           int64
BackPain                                 int64
Anxiety                                float64
Allergic_rhinitis                        int64
Reflux_esophagitis                       int64
Asthma                                   int64
Services                              category
Initial_days                           float64
TotalCharge                            float64
Additional_charges                     float64
Survey_Timely_Admission                  int64
Survey_Timely_Treatment                  int64
Survey_Timely_Visits                     int64
Survey_Reliability                       int64
Survey_Options                           int64
Survey_Hours_of_Treatment                int64
Survey_Courteous_Staff                   int64
Survey_Evidence_of_Active_Listening      int64
dtype: object
```

In [116]:

```
#This code gives me the value counts for each value of each column. I ca

for x in Med_df:
    print(Med_df[x].value_counts())
```

```
Houston           36
San Antonio       26
Springfield       22
Miami             21
New York          21
                  ..
Hollenberg         1
Hollandale         1
Holland Patent     1
Holcombe           1
Zumbro Falls       1
Name: City, Length: 6072, dtype: int64
TX    553
CA    550
PA    547
NY    514
IL    442
OH    383
MO    328
```

In [117]:

```
#Encode categorical variables for multiple imputation in IterativeImputer

label_encode = preprocessing.LabelEncoder()

for x in Med_df[categorical_variables]:
    encode_column = str(x)+'_Encoded'
    Med_df[encode_column] = label_encode.fit_transform(Med_df[x])

Med_df_cat = Med_df[categorical_variables]

Med_df.drop(categorical_variables, axis = 1, inplace = True)
```

In [118]:

```
#Use IterativeImputer from Sklearn to impute missing values in the datase

It_Imp = IterativeImputer(skip_complete = True, min_value = 0)

Med_df_fill = np.round(It_Imp.fit_transform(Med_df))

Med_df_fill = pd.DataFrame(Med_df_fill)

Med_df_fill.isna().sum()
```

Out[118]:

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
18    0
19    0
20    0
21    0
22    0
23    0
24    0
25    0
26    0
27    0
28    0
29    0
30    0
31    0
32    0
33    0
34    0
35    0
36    0
37    0
38    0
39    0
40    0
41    0
42    0
43    0
```

```
44    0
dtype: int64
```

In [119]:

```
#This renames all of my columns in my newly formed DF to match their orig

for x, y in enumerate(It_Imp.feature_names_in_):
    Med_df_fill.rename(columns= {x:y}, inplace = True)

for x in categorical_variables:
    cat_var = str(x)+'_Encoded'
    Med_df_fill[cat_var] = Med_df_fill[cat_var].astype('category')

Med_df_fill.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 45 columns):
 #   Column                              Non-Null Count  Dtype   
---  ------                              --------------  -----   
 0   Lat                                 10000 non-null  float64 
 1   Lng                                 10000 non-null  float64 
 2   Population                          10000 non-null  float64 
 3   Children                            10000 non-null  float64 
 4   Age                                 10000 non-null  float64 
 5   ReAdmis                             10000 non-null  float64 
 6   VitD_levels                         10000 non-null  float64 
 7   Doc_visits                          10000 non-null  float64 
 8   Full_meals_eaten                    10000 non-null  float64 
 9   VitD_supp                           10000 non-null  float64 
 10  Soft_drink                          10000 non-null  float64 
 11  HighBlood                           10000 non-null  float64 
 12  Stroke                              10000 non-null  float64 
 13  Complication_risk                   10000 non-null  float64 
 14  Overweight                          10000 non-null  float64 
 15  Arthritis                           10000 non-null  float64 
 16  Diabetes                            10000 non-null  float64 
 17  Hyperlipidemia                      10000 non-null  float64 
 18  BackPain                            10000 non-null  float64 
 19  Anxiety                             10000 non-null  float64 
 20  Allergic_rhinitis                   10000 non-null  float64 
 21  Reflux_esophagitis                  10000 non-null  float64 
 22  Asthma                              10000 non-null  float64 
 23  Initial_days                        10000 non-null  float64 
 24  TotalCharge                         10000 non-null  float64 
 25  Additional_charges                  10000 non-null  float64 
 26  Survey_Timely_Admission             10000 non-null  float64 
 27  Survey_Timely_Treatment             10000 non-null  float64 
 28  Survey_Timely_Visits                10000 non-null  float64 
 29  Survey_Reliability                  10000 non-null  float64 
 30  Survey_Options                      10000 non-null  float64 
 31  Survey_Hours_of_Treatment           10000 non-null  float64 
 32  Survey_Courteous_Staff              10000 non-null  float64 
 33  Survey_Evidence_of_Active_Listening 10000 non-null  float64 
 34  City_Encoded                        10000 non-null  category
 35  State_Encoded                       10000 non-null  category
 36  County_Encoded                      10000 non-null  category
 37  Zip_Encoded                         10000 non-null  category
 38  Area_Encoded                        10000 non-null  category
 39  Timezone_Encoded                    10000 non-null  category
```

```
 40  Education_Encoded                     10000 non-null  category
 41  Employment_Encoded                    10000 non-null  category
 42  Gender_Encoded                        10000 non-null  category
 43  Initial_admin_Encoded                 10000 non-null  category
 44  Services_Encoded                      10000 non-null  category
dtypes: category(11), float64(34)
memory usage: 3.3 MB
```

In [120]:

```
#Create hold df for our target variable and then drop it from our working

Target = Med_df_fill.ReAdmis

Med_df_fill.drop(columns='ReAdmis', axis = 1, inplace=True)

Target
```

Out[120]:

```
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
       ...
9995    0.0
9996    1.0
9997    1.0
9998    1.0
9999    1.0
Name: ReAdmis, Length: 10000, dtype: float64
```

In [121]:

```
#Temporarily add back in categorical values for cleaned data set

Med_df_fill =  Med_df_fill.merge(Med_df_cat, left_index=True, right_inde
```

In [122]:

```
#Export clean dataset and then drop non-numeric categorical values

Med_df_fill.to_csv(r'C:\Users\jacob.colp.UNITY\Downloads\Medical Data Ra

for x in Med_df_cat:
    Med_df_fill.drop(x, axis=1, inplace=True)
```

In [123]:

```
#This scales all of the numeric values relative to their mean.

Standard_Scaler = StandardScaler()

Scaled_Med_df = Standard_Scaler.fit_transform(Med_df_fill)

Scaled_Med_df = pd.DataFrame(Scaled_Med_df)

for x, y in enumerate(Standard_Scaler.feature_names_in_):
    Scaled_Med_df.rename(columns = {x:y}, inplace = True)

#Getting rid of qualitative variables

Scaled_Med_df = Scaled_Med_df.drop(columns=['City_Encoded', 'County_Enco

Scaled_Med_df.describe()
```

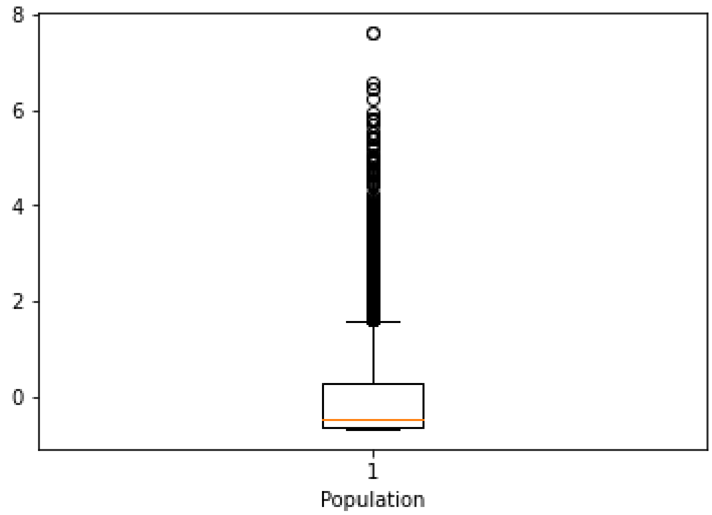
Out[123]:

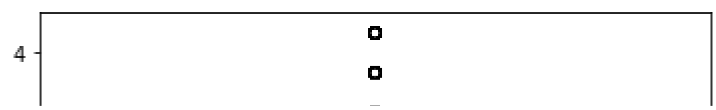
	Population	Children	Age	VitD_levels	Doc_visits	Full_meals_e
count	1.000000e+04	1.000000e+04	1.000000e+04	1.000000e+04	1.000000e+04	1.000000
mean	-1.207923e-17	1.065814e-17	-1.353584e-16	2.405187e-16	3.161915e-17	-7.81597
std	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050
min	-6.722371e-01	-1.116784e+00	-2.276538e+00	-1.397478e+00	-3.836921e+00	-9.93386
25%	-6.253705e-01	-5.780035e-01	-8.081510e-01	-3.576545e-01	-9.679806e-01	-9.93386
50%	-4.854456e-01	-3.922320e-02	-2.501152e-02	-2.091084e-01	-1.166703e-02	-1.38879
75%	2.684661e-01	4.995571e-01	8.193108e-01	8.798388e-02	9.446465e-01	9.90609
max	7.612562e+00	4.271019e+00	2.275461e+00	4.990006e+00	3.813587e+00	5.950600

8 rows × 31 columns

In [124]:

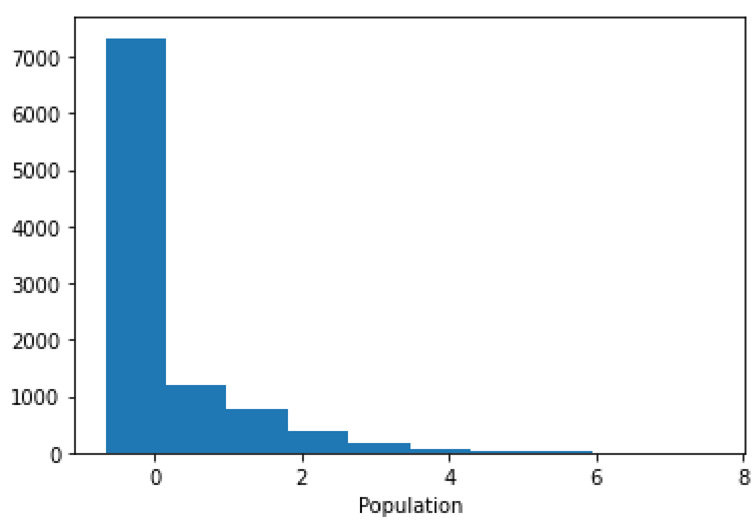
```
#Print boxplots for all non-categorical variables

for x in Scaled_Med_df:
        pyplot.boxplot(Scaled_Med_df[x])
        pyplot.xlabel(x)
        pyplot.show()
```





In [125]:

```
#Print histograms for all non-categorical variables

for x in Scaled_Med_df:
    pyplot.hist(Scaled_Med_df[x])
    pyplot.xlabel(x)
    pyplot.show()
```



Part III: Data-Cleaning

D. Data-Cleaning Process Summary

- Orient Myself to the data
- Value Counts
- Missing Values
- Standardized numeric columns
- Categorical variables to Numeric
- Histograms and Boxplots (Outlier Detection)

D1.

I am choosing not remove any outliers. I believe that all anomilies are a normal part of our studied population. As noted in "Guidelines for Removing and Handling Outliers in Data" - anomalies should be removed when they are either the result of an error, or they cease to represent our studied population. I do not believe any outliers to be indicative of either of those things. Therefore - I am maintaining all values. To address findings - anomalies do exist within the data. To name a few - income, education, and population. As we are evaluating a hospital - it is reasonable to expect that you are going to have broad variations on these factors. Were our research question to be centered around the readmission of low income/low education individuals, or low population areas - we could certainly scrub our data of extremely high anomalies, but as this is an evaluation of an inclusive hospital population - these will be retained.

D2.

My mitigation technique is choosing to leave all values intact. As noted in "Guidelines for Removing and Handling Outliers in Data," - "Outliers can be very informative about the subject-area and data collection process. It’s essential to understand how outliers occur and whether they might happen again as a normal part of the process or study area" (Frost, 2021). Within that article he outlines the scenarios in which the removal of outliers is appropriate. It is only appropriate to remove outliers when there is an error in collection, or the collection is not representative of the studied population. However, these could have been dealt with using some kind of normalization. Perhaps reassigning these outliers to more representative values, or dropping them all together. Again, since the studied population was from a hospital - there is going to be a broad population that will be represented. And, I also do not have the ability to scrutinize collection techniques. Therefore - I have chosen to maintain all values.

D3.

The first data cleaning step was familiarizing myself with the data. The ultimate outcome of this was me being able to identify data types and their general statistical values.

The second step was value counts. This really helps me be able to identify the unique columns that will not aid in the ultimate goal of dimensionality reduction, and gives me a general idea of what exists in each column. To do this I want to iterate through each column and count the number of times is a value is present. This is achieved through this section of code:

for x in Med_df: print(Med_df[x].value_counts())

Third was dealing with missing values. I utilized IterativeImputer from the sklearn package. This iterates over the data multiple times to train a model that will impute missing values based upon that variables relationship to other columns. Ultimately - this helps us to infer missing data points and provide a complete dataset to future models. The columns with null values were Children, Age, Income, Soft drink, Overweight, Anxiety, and Intial_days. The main bulk of this is achieved through this section of code:

It_Imp = IterativeImputer(skip_complete = True, min_value = 0)

Med_df_fill = np.round(It_Imp.fit_transform(Med_df))

Med_df_fill = pd.DataFrame(Med_df_fill)

Med_df_fill.isna().sum()

The output of the above code sample is going to give the analyst the count of null values in each column. In our implementation - this yields no null values in any columns.

Standardized numeric columns was the fourth step that normalized all of our values around a mean of 0. This has multiple functions - it does help in scale for viewing outliers. It is also a necessary step in PCA that will be performed below. It gives the model values to recognize variance across the whole dataset. This is achieved through this section of code:

Standard_Scaler = StandardScaler()

Scaled_Med_df = Standard_Scaler.fit_transform(Med_df_fill)

Scaled_Med_df = pd.DataFrame(Scaled_Med_df)

for x, y in enumerate(Standard_Scaler.feature_names_in_): Scaled_Med_df.rename(columns = {x:y}, inplace = True)

#Getting rid of categorical variables

Scaled_Med_df = Scaled_Med_df.drop(columns=['City_Encoded', 'County_Encoded', 'State_Encoded', 'Lat', 'Lng', 'Timezone_Encoded', 'Zip_Encoded', 'Area_Encoded', 'Education_Encoded', 'Employment_Encoded','Gender_Encoded', 'Initial_admin_Encoded', 'Services_Encoded'], axis=1)

Scaled_Med_df.describe()

There are a few things happening here. I instantiate the StandardScaler model in the first line. I then fit the model and transform the Med_df_fill (the dataframe that was filled using MICE). I convert the scaled array back to a dataframe for easier interaction moving forward. The next for loop is used to rename all of the columns back to their appropriate names. The standardscaler model converts all column headers to an index based upon their location. As noted in "PCA Is Not Feature Selection" - PCA should not be performed with categorical variables - so the next section of code drops all categorical variables from the Scaled_Med_df dataframe. In the final portion I print out the describe function which will give me relevant statistical information about our new data set. This helps to give a high level overview of the outcome of our scaling. These columns all now represent their distribution with their being 0.

Fifth was converting categorical variables to numeric. This is a necessity for our imputation performed above. The iterativeimputer module requires numeric values. It would also help if they were needed for another future model down the line where we wanted to evaluate their impact. Of course we would need to define that there is no ordinal value so that a model would not infer that, but again it was a necessary part of what we were doing now as well. This is done through a feature in the preprocessing package of sklearn. Our outcome is achieved through this section of code:

label_encode = preprocessing.LabelEncoder()

for x in Med_df[categorical_variables]: encode_column = str(x)+'_Encoded' Med_df[encode_column] = label_encode.fit_transform(Med_df[x])

Med_df_cat = Med_df[categorical_variables]

Med_df.drop(categorical_variables, axis = 1, inplace = True)

There are a few steps completed within this code sample - the first is the instantiation of the function itself. I then iterate through all the categorical variables in a for loop and perform conversion on them while also attaching "_Encoded" to the column header, so I can retrieve the categorical values easier down the line. I then place these our old categorical variables into a separate dataframe for holding. Finally, I drop the non-encoded variables from our working Dataframe.

Finally - we have histograms and boxplots. This helps the user to be able to visually identify distribution and outliers. To me - distributions look as to be expected. And, outliers are believed to be a normal part of our studied population. The work for this is done through the following code:

for x in Scaled_Med_df: pyplot.hist(Scaled_Med_df[x]) pyplot.xlabel(x) pyplot.show()

For the box plots - .hist is simply replaced with .boxplot. This for loop iterates through all of our columns and produces a new plot for each one. It labels the x axis and then shows it. This is the most programmatic way to generate these plot for each variable.

In [126]:

```
print(Scaled_Med_df.describe())
print(Scaled_Med_df.isna().sum())
```

```
         Population      Children           Age    VitD_levels     Doc_visits  \
count  1.000000e+04  1.000000e+04  1.000000e+04  1.000000e+04  1.000000e+04
mean  -1.207923e-17  1.065814e-17 -1.353584e-16  2.405187e-16  3.161915e-17
std    1.000050e+00  1.000050e+00  1.000050e+00  1.000050e+00  1.000050e+00
min   -6.722371e-01 -1.116784e+00 -2.276538e+00 -1.397478e+00 -3.836921e+00
25%   -6.253705e-01 -5.780035e-01 -8.081510e-01 -3.576545e-01 -9.679806e-01
50%   -4.854456e-01 -3.922320e-02 -2.501152e-02 -2.091084e-01 -1.166703e-02
75%    2.684661e-01  4.995571e-01  8.193108e-01  8.798388e-02  9.446465e-01
max    7.612562e+00  4.271019e+00  2.275461e+00  4.990006e+00  3.813587e+00

       Full_meals_eaten     VitD_supp    Soft_drink     HighBlood  \
count      1.000000e+04  1.000000e+04  1.000000e+04  1.000000e+04
mean      -7.815970e-17  3.694822e-17 -2.380318e-17 -9.059420e-17
std        1.000050e+00  1.000050e+00  1.000050e+00  1.000050e+00
min       -9.933869e-01 -6.347126e-01 -4.912340e-01 -8.318938e-01
25%       -9.933869e-01 -6.347126e-01 -4.912340e-01 -8.318938e-01
50%       -1.388797e-03 -6.347126e-01 -4.912340e-01 -8.318938e-01
75%        9.906093e-01  9.564446e-01 -4.912340e-01  1.202076e+00
max        5.950600e+00  7.321074e+00  2.035690e+00  1.202076e+00

             Stroke  ...   TotalCharge  Additional_charges  \
count  1.000000e+04  ...  1.000000e+04        1.000000e+04
mean   5.346834e-17  ...  1.364242e-16       -1.111999e-16
std    1.000050e+00  ...  1.000050e+00        1.000050e+00
min   -4.989060e-01  ... -1.372224e+00       -1.499252e+00
25%   -4.989060e-01  ... -7.810871e-01       -7.562773e-01
50%   -4.989060e-01  ... -1.170602e-02       -2.079592e-01
75%   -4.989060e-01  ...  5.103687e-01        4.114351e-01
max    2.004386e+00  ...  4.628563e+00        2.695005e+00

       Survey_Timely_Admission  Survey_Timely_Treatment  Survey_Timely_Visi
ts  \
count             1.000000e+04             1.000000e+04            1.000000e+
04
mean             -1.357137e-16             7.247536e-17            9.521273e-
17
std               1.000050e+00             1.000050e+00            1.000050e+
00
min              -2.440901e+00            -2.422464e+00           -2.431579e+
00
25%              -5.027551e-01            -4.896726e-01           -4.949145e-
01
50%               4.663179e-01            -4.896726e-01            4.734175e-
01
75%               4.663179e-01             4.767229e-01            4.734175e-
01
max               4.342610e+00             3.375910e+00            4.346746e+
00

       Survey_Reliability  Survey_Options  Survey_Hours_of_Treatment  \
count        1.000000e+04    1.000000e+04               1.000000e+04
mean         1.385558e-16   -1.250555e-16               6.501466e-17
```

```
std          1.000050e+00    1.000050e+00                1.000050e+00
min         -2.427165e+00   -2.423843e+00               -2.443515e+00
25%         -4.970906e-01   -4.823612e-01               -5.061393e-01
50%          4.679464e-01   -4.823612e-01                4.625484e-01
75%          4.679464e-01    4.883798e-01                4.625484e-01
max          3.363057e+00    3.400603e+00                3.368612e+00

       Survey_Courteous_Staff  Survey_Evidence_of_Active_Listening
count            1.000000e+04                         1.000000e+04
mean            -1.893596e-16                        -3.268497e-17
std              1.000050e+00                         1.000050e+00
min             -2.441857e+00                        -2.407940e+00
25%             -4.836717e-01                        -4.890334e-01
50%             -4.836717e-01                        -4.890334e-01
75%              4.954208e-01                         4.704200e-01
max              3.432698e+00                         3.348781e+00

[8 rows x 31 columns]
Population                             0
Children                               0
Age                                    0
VitD_levels                            0
Doc_visits                             0
Full_meals_eaten                       0
VitD_supp                              0
Soft_drink                             0
HighBlood                              0
Stroke                                 0
Complication_risk                      0
Overweight                             0
Arthritis                              0
Diabetes                               0
Hyperlipidemia                         0
BackPain                               0
Anxiety                                0
Allergic_rhinitis                      0
Reflux_esophagitis                     0
Asthma                                 0
Initial_days                           0
TotalCharge                            0
Additional_charges                     0
Survey_Timely_Admission                0
Survey_Timely_Treatment                0
Survey_Timely_Visits                   0
Survey_Reliability                     0
Survey_Options                         0
Survey_Hours_of_Treatment              0
Survey_Courteous_Staff                 0
Survey_Evidence_of_Active_Listening    0
dtype: int64
```

D4.

Code can be found above this section of commentary. The main functionality can be found in cells 3-18.

D5.

Cleaned data was exported above this section and will be found in project submission.

D6.

There are certainly limitations within my approach to data cleaning in this project. Ideally - with missing values, I would have preferred to have conversations with the client around potential reasons as to why they were missing in the first place. I had to make the assumption that there was nothing to correct in the process of data gathering and those values needed to be imputed. To impute - I used MICE. MICE stands for "Multiple Imputation by Chained Equations." According to Azur et al - MICE does not have the same "theoretical justification" as other imputation methods, but is widely used and very flexible in comparison. Within the same paper, another disadvantage to MICE is addressed - and that is the failure of the base model to account for clustering. You can account for this, but depending on the analysis that will follow will impact how that is should be addressed in the model. (Azur et al, 2011).

I also chose to leave anomalies as a part of my analysis moving forward. This of course could be misinterpreted by a model down the line, but I again could not have a conversation with the client to determine if these were true values or the result of an error. So, I chose to err on the side of caution and include under the assumption that they were true observations. I do believe that this is ethically correct based upon the intended outcome of the project.

D7.

So, how could the limitations of my approach impact analysis? Imputation could certainly lead to a misrepresentation of the truth. Is there any 100% guarantee that the imputed values are going to be indicative of reality? No. I cannot say with 100% certainty that the imputed values are correct, however that is the nature of imputation and the nature of dealing with missing values. There is no way of getting back to truth unless you go back and confirm that observation. Outside of that more obvious drawback - there are researched impacts on correlation. As noted in the publication "Effects of imputation on correlation: implications for analysis of mass spectrometry data from multiple biological matrices" - "The simulations corroborated these results showing all imputation methods to cause a general reduction in the magnitude of the correlation. The degree of change was greater for strongly correlated compounds, and also increased with increasing levels of missingness," (Taylor et al, 2017). Simply put - imputation of all kinds are going to dull correlation of even highly correlated values, and that impact increases as the number of missing values increases. With further evaluation of correlation - we may find weaker correlations than expected and that could certainly be due to our use of imputation.

As for the non-action on anomalies - this could lead to the outliers impacting our model by providing extremes. However, as they are assumed-to-be-true values - they need to be included as to have accurate information relative to our population. So, while this may lead to some differences in the model compared to their exclusion - it does fall in line with best practices.

In [127]:

```
#This is our correlation matrix to show the relationship between variable

Scaled_Med_df.corr()
```

Out[127]:

	Population	Children	Age	VitD_levels	Doc_visits
Population	1.000000	0.007205	-0.018371	0.002124	0.012646
Children	0.007205	1.000000	0.005393	-0.002391	-0.004734
Age	-0.018371	0.005393	1.000000	0.019033	0.005166
VitD_levels	0.002124	-0.002391	0.019033	1.000000	0.001367
Doc_visits	0.012646	-0.004734	0.005166	0.001367	1.000000
Full_meals_eaten	-0.025608	-0.000856	0.010050	0.009170	-0.002767
VitD_supp	0.009781	-0.000463	0.008860	0.009991	0.005681
Soft_drink	0.004115	0.007961	0.004361	-0.000697	0.017951
HighBlood	0.009764	0.004411	0.010891	0.004970	0.008967
Stroke	-0.001690	0.004573	0.013436	-0.009912	-0.002230
Complication_risk	0.015936	-0.003008	-0.000490	0.005316	0.012306
Overweight	0.000367	-0.021073	-0.007507	-0.007787	0.001087
Arthritis	0.000055	0.004476	0.008995	-0.000469	-0.000719
Diabetes	-0.009975	0.018689	0.005136	-0.023462	0.012781
Hyperlipidemia	-0.006222	-0.009856	0.003964	0.000824	-0.026730
BackPain	0.006437	-0.023047	0.019881	-0.003450	0.008514
Anxiety	-0.012899	0.005274	0.008150	0.014533	-0.002834
Allergic_rhinitis	0.007681	-0.019174	0.014716	-0.002394	0.002920
Reflux_esophagitis	0.014340	0.004483	-0.016896	-0.007717	-0.005330
Asthma	-0.001510	0.005987	0.009301	0.011450	-0.017989
Initial_days	0.019087	0.010640	0.016488	0.008069	-0.007615
TotalCharge	0.013751	0.003058	0.024185	0.727561	-0.004515
Additional_charges	-0.004820	0.009551	0.728847	0.016425	0.008072
Survey_Timely_Admission	0.014312	0.005446	0.004909	-0.004335	0.003680
Survey_Timely_Treatment	0.023612	0.010784	0.002969	-0.017683	0.006024
Survey_Timely_Visits	-0.001248	0.002291	0.004855	-0.012496	-0.002718
Survey_Reliability	-0.004660	0.005824	0.005639	0.012671	-0.006538
Survey_Options	0.008705	0.006133	-0.007860	-0.012255	-0.009434
Survey_Hours_of_Treatment	0.008159	-0.002160	-0.000156	0.007350	0.012530
Survey_Courteous_Staff	0.010034	0.005558	0.010302	0.001992	0.008589
Survey_Evidence_of_Active_Listening	-0.000220	-0.011324	-0.003642	0.004033	0.004571

31 rows × 31 columns

In [128]:

```
#Initiating PCA
#Reference: (Kindsonthegenius, 2020)
Med_PCA = PCA()

Med_PCA_FT = Med_PCA.fit_transform(Scaled_Med_df)

Med_PCA_FT = pd.DataFrame(Med_PCA_FT)

Med_PCA_FT
```

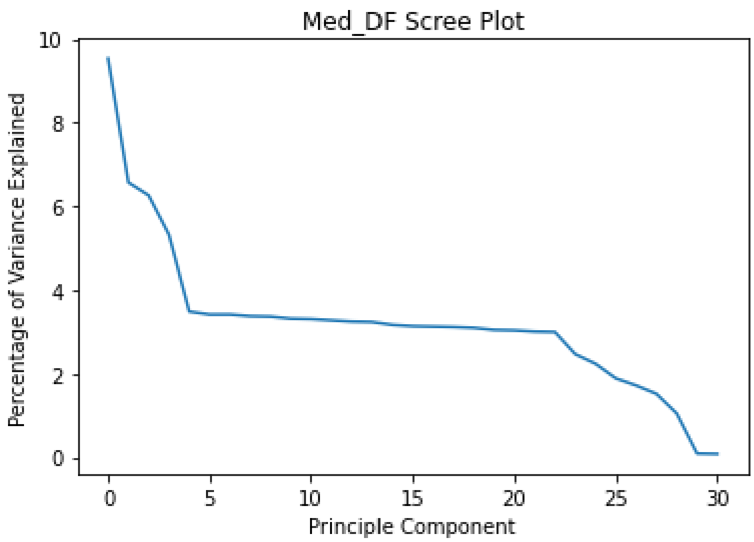
Out[128]:

	0	1	2	3	4	5	6	7	
0	-1.606819	0.087476	-1.603797	0.352726	1.125682	-0.824507	0.640937	-1.515131	0
1	-0.325365	0.219360	-1.222288	-0.104782	-0.962116	-0.164027	-1.638954	0.928217	0
2	-0.201751	-0.355915	-1.831390	-0.660291	0.278594	-0.051720	-1.148161	0.144641	0
3	2.345966	-0.854095	-1.201547	0.391742	1.022480	0.349036	-0.389637	-0.557362	-0
4	-2.372715	-2.752463	0.395789	-0.337430	-0.334034	1.333370	0.521480	1.905066	-1
...	...	...	...	...	...	...	...	...	
9995	-2.107126	-0.304723	0.550835	-0.270944	-0.520516	-0.188865	-1.047018	-0.071344	0
9996	-0.703980	2.668207	-1.576351	1.724087	1.016712	0.329201	-0.810658	-1.916748	0
9997	-1.880049	0.880328	0.037438	0.261843	0.846171	-0.806931	-0.263213	0.499075	0
9998	0.796273	-0.095912	1.684987	0.896192	-0.497747	-0.143362	0.280589	0.528423	-0
9999	0.676988	0.715358	1.253763	0.224172	-0.985252	2.139548	-1.036507	-1.679030	-0

10000 rows × 31 columns

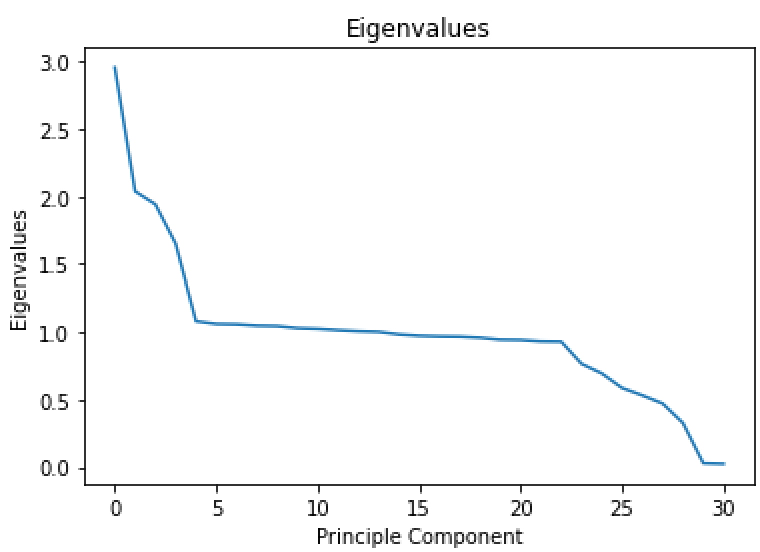
In [129]:

```
#This shows explained variance ratios for each principle component

percent_variance = np.round(Med_PCA.explained_variance_ratio_*100, decima

pyplot.plot(Med_PCA_FT.columns, percent_variance)
pyplot.title('Med_DF Scree Plot')
pyplot.xlabel('Principle Component')
pyplot.ylabel('Percentage of Variance Explained')
plt.show()
```



In [130]:

```
eigenvalues = Med_PCA.explained_variance_

pyplot.plot(Med_PCA_FT.columns, eigenvalues)
pyplot.title('Eigenvalues')
pyplot.xlabel('Principle Component')
pyplot.ylabel('Eigenvalues')
plt.show()
```



In [131]:

```
eigenvalues
```

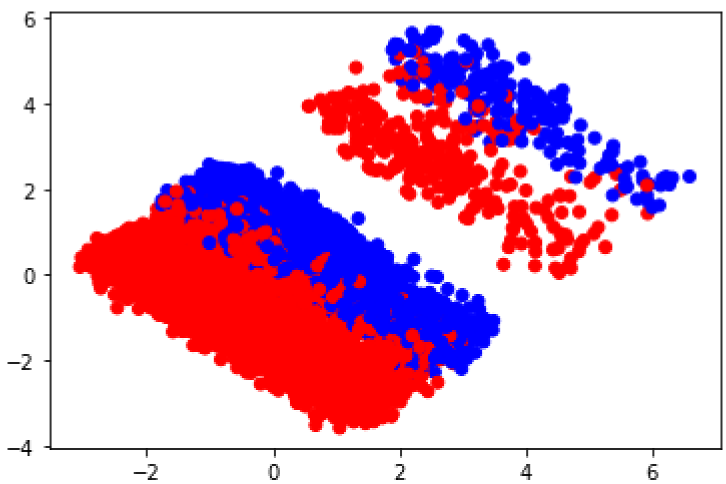
Out[131]:

```
array([2.95685877, 2.03843911, 1.94159811, 1.64979372, 1.08172145,
       1.06155136, 1.05893954, 1.04868604, 1.04592233, 1.03049175,
       1.02554653, 1.01565494, 1.00782696, 1.00312187, 0.98429433,
       0.97400516, 0.96981101, 0.96801084, 0.96002859, 0.94493457,
       0.9428899 , 0.93240619, 0.93052189, 0.76619243, 0.69473346,
       0.58670835, 0.53178077, 0.47118403, 0.32677779, 0.02862751,
       0.02404102])
```

In [132]:

```
#This will show clustering of my observations relative to PC1 and PC2

Med_PCA_FT = Med_PCA_FT.merge(Target, left_index=True, right_index=True)

Med_PCA_FT['Color']= Med_PCA_FT['ReAdmis'].replace({0:'r', 1:'b'})

pyplot.scatter(Med_PCA_FT[1], Med_PCA_FT[2], c = Med_PCA_FT.Color)
```

Out[132]: <matplotlib.collections.PathCollection at 0x2819c6217f0>



In [133]:
```
for x, y in zip(Med_PCA_FT.columns, np.cumsum(Med_PCA.explained_variance_
    print(x, y)
```

```
0 0.09537300264151131
1 0.1611225273178129
2 0.2237484613314441
3 0.2769622917197733
4 0.3118530427783067
5 0.34609321081467875
6 0.38024913482062234
7 0.41407433387307635
8 0.44781038979305654
9 0.4810487349915523
10 0.5141275727512862
11 0.5468873589467472
12 0.5793946548583053
13 0.611750188981412
14 0.6434984437742507
15 0.6749148229595627
16 0.7061959206457349
17 0.7374189541249802
18 0.7683845215516597
19 0.7988632337291206
20 0.8292759952287545
21 0.8593506065006781
22 0.8893644401291758
23 0.9140778533385729
24 0.9364863689582283
25 0.9554105521881042
26 0.9725630552530493
27 0.9877610200006122
28 0.9983011849043533
29 0.9992245607476545
30 1.0000000000000002
```

In [134]:

```
#This gives me all our initial variables and their loadings in each princ

loadings = pd.DataFrame(Med_PCA.components_.T*100, index= Scaled_Med_df.c

loadings
```

Out[134]:

	0	1	2	3	4
Population	1.026974	0.665143	2.180759	2.527812	-38.867162
Children	0.252623	1.025776	-0.351872	1.369559	17.525652
Age	0.625204	38.748640	-35.165643	2.854789	12.690066
VitD_levels	-0.957820	36.082300	38.019565	3.446524	12.190871
Doc_visits	0.706793	0.613241	-1.351132	-0.533841	-13.355231
Full_meals_eaten	-0.051667	1.052641	-2.852055	2.213620	42.384042
VitD_supp	-0.467146	2.892635	1.281249	0.797140	-35.693543
Soft_drink	0.689039	-0.001508	-0.066078	1.389166	25.409101
HighBlood	-0.340725	33.691132	-33.268180	0.445878	-15.031272
Stroke	-0.235984	1.392486	-3.759940	1.544252	-2.891159
Complication_risk	1.285496	4.839999	-0.920201	-0.953717	-15.346986
Overweight	0.426311	-0.103823	-2.925938	1.157815	-18.473379
Arthritis	-1.422671	2.196020	0.959379	-0.741561	9.220582
Diabetes	-0.295194	-1.237835	-1.666642	3.139808	35.350400
Hyperlipidemia	1.707249	0.334629	1.237430	-1.475862	11.077778
BackPain	-1.299412	2.929841	0.178080	-0.655148	-12.771244
Anxiety	-0.075996	3.347648	1.315573	-2.262854	20.985467
Allergic_rhinitis	0.481664	2.343741	-1.145508	1.778240	-2.767455
Reflux_esophagitis	0.632136	-0.241043	2.315297	-1.112417	-11.421476
Asthma	-1.063100	0.931665	-1.676664	2.043914	28.676481
Initial_days	-1.986170	31.075724	34.955306	6.561606	-11.140513
TotalCharge	-1.854153	48.252981	50.801088	6.605548	1.330958
Additional_charges	0.505671	51.079931	-48.241300	2.715812	-0.857136
Survey_Timely_Admission	45.453775	-2.177145	-0.479634	29.500934	-0.139541
Survey_Timely_Treatment	42.822559	-2.234316	-0.597632	29.182524	-1.840905
Survey_Timely_Visits	39.516046	-2.408963	-0.202250	29.369536	0.238986
Survey_Reliability	15.205224	4.398749	2.833105	-55.460947	1.262114
Survey_Options	-18.994090	-5.919344	-2.362698	57.951343	0.370184
Survey_Hours_of_Treatment	41.003413	1.808836	2.106847	-16.146311	1.997520
Survey_Courteous_Staff	35.636388	3.445278	1.404365	-16.893725	2.500745
Survey_Evidence_of_Active_Listening	31.204330	1.960899	2.029133	-16.537068	-1.907401

31 rows × 31 columns

Part III: Data Cleaing (Continued)

E. Principle Component Analysis

E1., E2., E3.

I have printed out the results of PCA in the above code. I will choose to maintain 20 of the principle components as to explain 80% of the total variance in the data moving forward. You can tell this by looking at their cumulative explained variance in combination with the above scree plot. This allows me to have reasonable dimensionality reduction while maintaining the majority of what is being communicated in the data set. As an expansion of this - we can validate this method utilizing eigenvalues. The Kaiser Rule states that eigenvalues above 1 should be kept because this indicates that they explain more than a single variable. However, according to the Variance Extraction Rule - that should be above 70%. By keeping 20 PC's we are able to have all our eigenvalues above 94% which keeps us very close to the Kaiser Criterion. (Factor Analysis, 2021)

PC1 explains about 10% of the total variance and it's variables with highest correlation are:

Timely_Admission Timely_Treatment Hours_of_Treatment Timely_Visits

All of these variables are highly positively correlated to the variance explained in PC1.

So, how can an organization benefit from this analysis? Ultimately - these would be identified as the areas that they should focus research on. Ideally - they would want to reach out to the patients that answered to the extremes within these survey questions. Theoretically - they could define additional areas of concerns and provide that as a part of the dataset for deeper evaluation. Really what this is going to do is give them areas to focus in on for further evaluation. You can see the relationship of the first two principle components to our target in the scatter plot found above.

Works Cited

Chantal D. Larose, and Daniel T. Larose. Data Science Using Python and R. Wiley, 2019. EBSCOhost, search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=nlebk&AN=2091371&site=eds-live&scope=site.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Eastwood, Brian. "The 10 Most Popular Programming Languages to Learn in 2021." Northeastern University Graduate Programs, 18 June 2020, [https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/ (https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/)](https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/).

Frost, Jim. “Guidelines for Removing and Handling Outliers in Data.” Statistics By Jim, 5 Apr. 2021, https://statisticsbyjim.com/basics/remove-outliers/ (https://statisticsbyjim.com/basics/remove-outliers/).

Kindsonthegenius. “Principal Components Analysis(Pca) in Python – Step by Step.” Kindson The Genius, 10 Sept. 2020, https://www.kindsonthegenius.com/principal-components-analysispca-in-python-step-by-step/ (https://www.kindsonthegenius.com/principal-components-analysispca-in-python-step-by-step/).

Bedre, Renesh. “Performing and Visualizing the Principal Component Analysis (PCA) from PCA Function and Scratch in Python.” Renesh Bedre, 18 Apr. 2021, https://www.reneshbedre.com/blog/principal-component-analysis.html (https://www.reneshbedre.com/blog/principal-component-analysis.html).

Walker, Brandon. “PCA Is Not Feature Selection.” Medium, Towards Data Science, 31 Dec. 2019, https://towardsdatascience.com/pca-is-not-feature-selection-3344fb764ae6 (https://towardsdatascience.com/pca-is-not-feature-selection-3344fb764ae6).

Azur, Melissa J, et al. “Multiple Imputation by Chained Equations: What Is It and How Does It Work?” International Journal of Methods in Psychiatric Research, John Wiley & Sons, Ltd, Mar. 2011, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/ (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/).

Sandra L Taylor, L Renee Ruhaak, Karen Kelly, Robert H Weiss, Kyoungmi Kim, Effects of imputation on correlation: implications for analysis of mass spectrometry data from multiple biological matrices, Briefings in Bioinformatics, Volume 18, Issue 2, March 2017, Pages 312–320, https://doi.org/10.1093/bib/bbw010 (https://doi.org/10.1093/bib/bbw010)

“Factor Analysis.” Statistics Solutions, 10 Aug. 2021, https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/factor-analysis/#:~:text=According%20to%20the%20variance%20extraction,not%20consider%20that%20a (https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/factor-analysis/#:~:text=According%20to%20the%20variance%20extraction,not%20consider%20that%20a