D206: Data Cleaning Performance Assessment

Shawn Wheeler

000364855

Table of Contents

A: Question or Decision……………………………………………………………………………………………………………..3

B: Required Variables…………………………………………………………………………………………………………………3

C1: Plan to Assess Quality of Data………………………………………………………………………………………………6

C2: Justification of Approach………………………………………………………………………………………………………6

C3: Justification of Tools…………………………………………………………………………………………………………….6

C4: Provide the Code…………………………………………………………………………………………………………………6

D1: Cleaning Findings………………………………………………………………………………………………………………14

D2: Justification of Mitigation Methods……………………………………………………………………………………14

D3: Summary of the Outcomes……………………………………………………………………………………………..…17

D4: Mitigation Code…………………………………………………………………………………………………………….…..20

D5: Clean Data…………………………………………………………………………………………………………………………22

D6: Limitations…………………………………………………………………………………………………………………………22

D7: Impact of Limitations…………………………………………………………………………………………………………23

E1: Principal Components………………………………………………………………………………………………………..24

E2: Criteria Used……………………………………………………………………………………………………………………...25

E3: Benefits……………………………………………………………………………………………………………………………..26

F: Video…………………………………………………………………………………………………………………………………...27

G: Sources of Third-Party Code…………………………………………………………………………………………………27

H: Sources………………………………………………………………………………………………………………………………..27

Part I – Research Questions and Variables

A: Question or Decision

The research question for this topic will be “what factors affect readmission?” All included data from the data set can be used for this analysis, and the question could be of great importance to a business trying to increase the amount of visits by patients in order to maximize profitability.

B: Required Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name** | **Data Type** | **Description** | **Example** |
| CaseOrder | Qualitative | Unique number for sorting purposes | 2 |
| Customer\_id | Qualitative | ID for each patient | F995323 |
| Interaction | Qualitative | ID for each interaction | 5885f56b-d6da-43a3-8760-83583af94266 |
| UID | Qualitative | Encoded patient information | e4884a42ba809df6a89ded6c97f460d4 |
| City | Qualitative | Patient's city of address | Daleville |
| State | Qualitative | Patient's state of address | WI |
| County | Qualitative | Patient's county of address | Minnehaha |
| Zip | Qualitative | Patient's ZIP Code of address | 56072 |
| Lat | Quantitative | Latitude of patient's address | 39.08062 |
| Lng | Quantitative | Longitude of patient's address | -81.31427 |
| Population | Quantitative | Amount of people living within a one mile radius of patient's address | 17125 |
| Area | Qualitative | Description of patient's neighborhood | Rural |
| TimeZone | Qualitative | Time zone where patient resides | America/Chicago |
| Job | Qualitative | Patient's current job title | Computer games developer |
| Children | Quantitative | Amount of children | 7 |
| Age | Quantitative | Patient's age in years | 51 |
| Education | Qualitative | Patient's education level | Bachelor's Degree |
| Employment | Qualitative | Patient's employment status | Full Time |
| Income | Quantitative | Income per year | 55586.48 |
| Marital | Qualitative | Patient's marital status | Widowed |
| Gender | Qualitative | Patient's gender | Female |
| ReAdmis | Qualitative | Within a month of visit, patient returned | No |
| VitD\_levels | Quantitative | Patient's intake Vitamin D levels | 17.42007928 |
| Doc\_visits | Quantitative | Amount of visits from physician during stay | 5 |
| Full\_meals\_eaten | Quantitative | Amount of meals eaten by patient | 3 |
| VitD\_supp | Quantitative | Amount of times Vitamin D was administered | 1 |
| Soft\_drink | Qualitative | Patient consumes >=3 soft drink per day | No |
| Initial\_admin | Qualitative | Reason for admission | Elective Admission |
| HighBlood | Qualitative | Diagnosed with high blood pressure | Yes |
| Stroke | Qualitative | History of stroke | No |
| Complication\_risk | Qualitative | Risk assessment | Medium |
| Overweight | Qualitative | Patient overweight per BMI | 1 |
| Arthritis | Qualitative | Diagnosed with arthritis | Yes |
| Diabetes | Qualitative | Diagnosed with diabetes | Yes |
| Hyperlipidemia | Qualitative | Diagnosed with hyperlipidemia | No |
| BackPain | Qualitative | Diagnosed with chronic back pain | No |
| Anxiety | Qualitative | Diagnosed with anxiety | 0 |
| Allergic\_rhinitis | Qualitative | Diagnosed with allergic rhinitis | Yes |
| Reflux\_esophagitis | Qualitative | Diagnosed with reflux esophagitis | No |
| Asthma | Qualitative | Diagnosed with asthma | Yes |
| Services | Qualitative | Service received by patient | CT Scan |
| Initial\_days | Quantitative | Length of days for initial visit | 15.12956221 |
| TotalCharge | Quantitative | Patient charge per day | 4060.336615 |
| Additional\_charges | Quantitative | Average charge for patient's treatment | 8363.18729 |
| Item1 | Qualitative | Timely admission | 1 |
| Item2 | Qualitative | Timely treatment | 2 |
| Item3 | Qualitative | Timely visits | 3 |
| Item4 | Qualitative | Reliability | 4 |
| Item5 | Qualitative | Options | 5 |
| Item6 | Qualitative | Hours of treatment | 6 |
| Item7 | Qualitative | Courteous staff | 7 |
| Item8 | Qualitative | Evidence of active listening from doctor | 8 |
| Item8 | Qualitative | Evidence of active listening from doctor | 8 |

Part II - Data Cleaning Plan (Detection)

C1: Plan to Assess Quality of Data

Detecting duplicates within the data set will be achieved by utilizing Python along with the pandas library to determine whether or not duplicates exist in the data set using duplicated(). Columns will be queried using isnull() to determine if they have 10.000 values. Thirdly, the methodology for detecting outliers will be using boxplots, histograms, and zscores for all quantitative variables incorporating the libraries for statistics, seaborn, scipy, numpy, and matplotlib, as well as examining the context of the values themselves. We will be looking for zscores above a three or below a negative three (Chantel, 2019). In order to determine quality with regards to re-expression of categorical variables and to ensure conformity to the data dictionary, each column will be analyzed individually to consider data types and requisites as laid out in the data dictionary by using describe().

C2: Justification of Approach

This plan was decided upon because it focuses on the primary four themes: duplicates, missing data, outliers, and re-expression of categorical variables. It is meticulous and thorough. Using duplicated() is a simple, convenient method for checking for duplicates within a set which is the reason it was chosen in order to detect potentially harmful duplicates. Missing data can cause a data set to be worthless or, at least, much less useful; therefore, isnull() was decided upon to be able to quickly determine the existence of missing rows. Outliers can further cause issues with data analysis as they can contribute to unrealistic results and will be identified via zscores with assistance from visual histograms and boxplots due to their ease of use and reliability. All other columns requiring investigating will be queried using describe() because of its conciseness of data, providing a single source of our needed information.

C3: Justification of Tools

Python was chosen for this project due to its readability, the exceptional ecosystem and libraries available, overall computational speed, and consistent syntax along with being more than capable of handling data cleaning as it relates to this project (Western Governor’s University, n.d.). I chose to use a few libraries in this project including pandas, scipy, numpy, statistics, seaborn, matplotlib, and sklearn. Pandas was used for overall data manipulation and data wrangling (Lustig, n.d.). Scipy, numpy, and statistics are used for zscores, normalization, and other mathematics. Seaborn was used to create boxplots for finding outliers. Matplotlib was used to create visualizations for PCA and other detection. Sklearn was used for PCA.

C4: Provide the Code

*See attached code :* d206completelist.ipynb

*Additional copy of detection code below :*

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

import seaborn

import scipy.stats as stats

import statistics

%matplotlib inline

# Importing and naming packages/libraries to be used

medicaldata = "/Users/Owner/Desktop/Skool/D206 - Data Cleaning/Medical Data/medical\_raw\_data.csv"

df = pd.read\_csv(medicaldata, index\_col=0)

# Opening original .csv data set for manipulation

# Ensuring file loaded correctly

df.info()

# Check data set for missing values

df.isnull().sum()

# Check quantitative variables for outliers. More in depth queries further down. [In-Text Citation : (Prabhakaran, 2023)]

quant\_columns = ['Lat', 'Lng', 'Population', 'Children', 'Age', 'Income', 'VitD\_levels', 'Doc\_visits', 'Full\_meals\_eaten', 'VitD\_supp', 'Initial\_days', 'TotalCharge', 'Additional\_charges']

for column in df:

if column in quant\_columns:

plt.figure()

plt.gca().set\_title(column)

df.boxplot([column])

# Boolean check to see if there are any duplicated rows in the data set

df.duplicated().value\_counts()

df.CaseOrder.duplicated().value\_counts()

# 2 CaseOrder duplicate verification

df.Customer\_id.duplicated().value\_counts()

# 3 Customer\_id duplicate verification

df.Interaction.duplicated().value\_counts()

# 4 Interaction duplicate verification

df.UID.duplicated().value\_counts()

# 5 UID duplicate verification

df.City.value\_counts()

# 6 City checking values

df.State.value\_counts()

# 7 State checking values

df.County.value\_counts()

# 8 County checking values

df.Zip.describe()

# 9 Zip checking for invalid zips, data type mismatch

df.Lat.describe()

# 10 Lat checking to ensure latitude is within the range of 0-90

df.Lng.describe()

# 11 Lng checking to ensure longitude is within the range of -180 - 180

df.Population.describe()

# 12 Population checking values and outliers

df.Area.value\_counts()

# 13 Area checking values, data dictionary mismatch

df.Timezone.value\_counts()

# 14 TimeZone checking values, named TimeZone in data dictionary

df.Job.value\_counts()

# 15 Job checking values

df.Children.value\_counts()

# 16 Children checking for missing values, data type mismatch, and outliers

df.Age.value\_counts()

# 17 Age checking values, outliers

df.Education.value\_counts()

# 18 Education checking values

df.Employment.value\_counts()

# 19 Employment checking values

df.Income.describe()

# 20 Income checking for missing data, outliers

df.Marital.value\_counts()

# 21 Marital checking values

df.Gender.value\_counts()

# 22 Gender checking values, data dictionary mismatch

df.ReAdmis.value\_counts()

# 23 ReAdmis Yes/No checking values, data dictionary mismatch

df.VitD\_levels.describe()

# 24 VitD\_levels checking values, outliers

df.Doc\_visits.value\_counts()

# 25 Doc\_visits checking values, outliers

df.Full\_meals\_eaten.value\_counts()

# 26 Full\_meals\_eaten checking values, outliers. Partial meals count as 0. More than 3 fine.

df.VitD\_supp.value\_counts()

# 27 VitD\_supp checking values, outliers

df.Soft\_drink.value\_counts()

# 28 Soft\_drink checking values, data dictionary mismatch, missing values

df.Initial\_admin.value\_counts()

# 29 Initial\_admin checking values, data dictionary mismatch

df.HighBlood.value\_counts()

# 30 HighBlood checking values, data dictionary mismatch

df.Stroke.value\_counts()

# 31 Stroke checking values, data dictionary mismatch

df.Complication\_risk.value\_counts()

# 32 Complication\_risk checking values, data dictionary mismatch

df.Overweight.value\_counts()

# 33 Overweight checking values, data type mismatch, missing values, data dictionary mismatch

df.Arthritis.value\_counts()

# 34 Arthritis checking values, data dictionary mismatch

df.Diabetes.value\_counts()

# 35 Diabetes checking values, data dictionary mismatch

df.Hyperlipidemia.value\_counts()

# 36 Hyperlipidemia checking values, data dictionary mismatch

df.BackPain.value\_counts()

# 37 BackPain checking values, data dictionary mismatch

df.Anxiety.value\_counts()

# 38 Anxiety checking values, data type mismatch, missing values, data dictionary mismatch

df.Allergic\_rhinitis.value\_counts()

# 39 Allergic\_rhinitis checking values, data dictionary mismatch

df.Reflux\_esophagitis.value\_counts()

# 40 Reflux\_esophagitis checking values, data dictionary mismatch

df.Asthma.value\_counts()

# 41 Asthma checking values, data dictionary mismatch

df.Services.value\_counts()

# 42 Services checking values, data dictionary mismatch

df.Initial\_days.describe()

# 43 Initial\_days checking values, missing data, outliers

df.TotalCharge.describe()

# 44 TotalCharge checking values, outliers

df.Additional\_charges.describe()

# 45 Additional\_charges checking values, outliers

df.Item1.value\_counts()

# 46 Item1 Timely admission, range 1-8

df.Item2.value\_counts()

# 47 Item2 Timely treatment, range 1-8

df.Item3.value\_counts()

# 48 Item3 Timely visits, range 1-8

df.Item4.value\_counts()

# 49 Item4 Reliability 1-8, range 1-8

df.Item5.value\_counts()

# 50 Item5 Options 1-8, range 1-8

df.Item6.value\_counts()

# 51 Item6 Hours of treatment, range 1-8

df.Item7.value\_counts()

# 52 Item7 Courteous staff, range 1-8

df.Item8.value\_counts()

# 53 Item8 Evidence of active listening from doctor, range 1-8

# Going back and checking individual possible outliers more in depth

# Method used is the same for each variable

# initial check of values and count. Quick access to mean and std for zscore below

df.Lat.describe()

# Creating zscore variable to use for creating histograms and boxplots by using statistics zscore function

# [In-Text Citation: (P.S.F., 2024)]

df['Lat\_zscore'] = statistics.NormalDist(mu=38.751099, sigma=5.403085).zscore(df.Lat)

plt.hist(df['Lat\_zscore'])

plt.show()

seaborn.boxplot(data=df.Lat\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Lat\_query = df.query('Lat\_zscore > 3 or Lat\_zscore < -3')

Lat\_query.info()

# Checking for amount of outliers for Lat

# 144 entries

Lat\_query.Lat.describe()

# Looking at values for the 144 outliers for Lat

df.Lng.describe()

# Repeating outlier process as above for Lat

df['Lng\_zscore'] = statistics.NormalDist(mu=-91.243, sigma=15.205998).zscore(df.Lng)

plt.hist(df['Lat\_zscore'])

plt.show()

seaborn.boxplot(data=df.Lat\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Lng\_query = df.query('Lng\_zscore > 3 | Lng\_zscore < -3')

Lng\_query.info()

# 98 entries

Lng\_query.Lng.describe()

# Describe to find values

df.Population.describe()

# Show a picture to see if there are outliers

seaborn.boxplot(data=df.Population.values, width=0.3, fliersize=2, whis=1.5);

# Create zscores

df['Population\_zscore'] = statistics.NormalDist(mu=9965.2538, sigma=14824.758614).zscore(df.Population)

# 4a - Use zscore with histogram

plt.hist(df['Population\_zscore'])

plt.show()

# Use zscore with boxplot

seaborn.boxplot(data=df.Population\_zscore.values, width=0.3, fliersize=2, whis=1.5);

# Create query to find count of outliers

Population\_query = df.query('Population\_zscore > 3')

Population\_query.info()

# 218 entries

Population\_query.Population.describe()

# Step 6 – Find values and additional information about the outliers themselves

df.Children.describe()

# Same process for the rest of the quantitative variables

seaborn.boxplot(data=df.Children.values, width=0.3, fliersize=2, whis=1.5);

df['Children\_zscore'] = statistics.NormalDist(mu=2, sigma=2.155427).zscore(df.Children)

plt.hist(df['Children\_zscore'])

plt.show()

seaborn.boxplot(data=df.Children\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Children\_query = df.query('Children\_zscore > 3')

Children\_query.info()

# 146 count

Children\_query.Children.value\_counts()

df.Age.describe()

df['Age\_zscore'] = statistics.NormalDist(mu=53.295676, sigma=20.659182).zscore(df.Age)

plt.hist(df['Age\_zscore'])

plt.show()

seaborn.boxplot(data=df.Age\_zscore.values, width=0.3, fliersize=2, whis=1.5);

df.Income.describe()

df['Income\_zscore'] = statistics.NormalDist(mu=40484.438268, sigma=28664.86105).zscore(df.Income)

plt.hist(df['Income\_zscore'])

plt.show()

seaborn.boxplot(data=df.Income\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Income\_query = df.query('Income\_zscore > 3')

Income\_query.info()

# 113 entries

Income\_query.Income.describe()

df.VitD\_levels.describe()

df['VitD\_levels\_zscore'] = statistics.NormalDist(mu=19.412675, sigma=6.723277).zscore(df.VitD\_levels)

plt.hist(df['VitD\_levels\_zscore'])

plt.show()

seaborn.boxplot(data=df.VitD\_levels\_zscore.values, width=0.3, fliersize=2, whis=1.5);

VitD\_levels\_query = df.query('VitD\_levels\_zscore > 3')

VitD\_levels\_query.info()

# 500 entries

VitD\_levels\_query.VitD\_levels.sort\_values(ascending=False)

df.Doc\_visits.describe()

df['Doc\_visits\_zscore'] = statistics.NormalDist(mu=5.0122, sigma=1.045734).zscore(df.Doc\_visits)

plt.hist(df['Doc\_visits\_zscore'])

plt.show()

seaborn.boxplot(data=df.Doc\_visits\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Doc\_visits\_query = df.query('Doc\_visits\_zscore > 3 | Doc\_visits\_zscore < -3')

Doc\_visits\_query.info()

# 8 entries

Doc\_visits\_query.Doc\_visits.value\_counts()

df.Full\_meals\_eaten.describe()

df['Full\_meals\_eaten\_zscore'] = statistics.NormalDist(mu=1.0014, sigma=1.008117).zscore(df.Full\_meals\_eaten)

plt.hist(df['Full\_meals\_eaten\_zscore'])

plt.show()

seaborn.boxplot(data=df.Full\_meals\_eaten\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Full\_meals\_eaten\_query = df.query('Full\_meals\_eaten\_zscore > 3')

Full\_meals\_eaten\_query.info()

# 33 entries

Full\_meals\_eaten\_query.Full\_meals\_eaten.describe()

df.VitD\_supp.describe()

df['VitD\_supp\_zscore'] = statistics.NormalDist(mu=0.3989, sigma=0.628505).zscore(df.VitD\_supp)

plt.hist(df['VitD\_supp\_zscore'])

plt.show()

seaborn.boxplot(data=df.VitD\_supp\_zscore.values, width=0.3, fliersize=2, whis=1.5);

VitD\_supp\_query = df.query('VitD\_supp\_zscore > 3')

VitD\_supp\_query.info()

# 70 entries

VitD\_supp\_query.VitD\_supp.describe()

df.Initial\_days.describe()

df['Initial\_days\_zscore'] = statistics.NormalDist(mu=34.432082, sigma=26.28705).zscore(df.Initial\_days)

plt.hist(df['Initial\_days\_zscore'])

plt.show()

seaborn.boxplot(data=df.Initial\_days\_zscore.values, width=0.3, fliersize=2, whis=1.5);

df.TotalCharge.describe()

df['TotalCharge\_zscore'] = statistics.NormalDist(mu=5891.538261, sigma=3377.558136).zscore(df.TotalCharge)

plt.hist(df['TotalCharge\_zscore'])

plt.show()

seaborn.boxplot(data=df.TotalCharge\_zscore.values, width=0.3, fliersize=2, whis=1.5);

TotalCharge\_query = df.query('TotalCharge\_zscore > 3')

TotalCharge\_query.info()

# 276 entries

TotalCharge\_query.TotalCharge.describe()

df.Additional\_charges.describe()

df['Additional\_charges\_zscore'] = statistics.NormalDist(mu=12934.528586, sigma=6542.601544).zscore(df.Additional\_charges)

plt.hist(df['Additional\_charges\_zscore'])

plt.show()

seaborn.boxplot(data=df.Additional\_charges\_zscore.values, width=0.3, fliersize=2, whis=1.5);

Additional\_charges\_query = df.query('Additional\_charges\_zscore > 3')

Additional\_charges\_query.info()

# 0 outliers

Additional\_charges\_query.Additional\_charges.describe()

Part III – Data Cleaning (Treatment)

D1: Cleaning Findings

*Duplicates :*

No duplicates were found in the data set.

*Missing values :*

|  |  |
| --- | --- |
| **Variables** | **# Missing Values** |
| Children | 2588 |
| Age | 2414 |
| Income | 2464 |
| Soft\_drink | 2467 |
| Overweight | 982 |
| Anxiety | 984 |

*Outliers :*

|  |  |  |
| --- | --- | --- |
| **Variables** | **Outliers** | **Values** |
| Population | 218 | Greater than 54,453 |
| Children | 146 | Ages 9-10 |
| Income | 113 | Greater than 127,029 |
| VitD\_levels | 500 | Greater than 40.84 |
| Doc\_visits | 8 | 6 for 1 visit. 2 for 9 visits. |
| Full\_meals\_eaten | 33 | 5-7 meals eaten |
| VitD\_supp | 70 | 3-5 supplements |
| TotalCharge | 276 | Greater than 16,053 |
| Lat | 144 | Greater than 17.96 |
| Lng | 469 | Less than -139.48 |

Re-expression and other data quality issues were detected in Zip, Area, TimeZone, Age, Gender, ReAdmis, Soft\_drink, Initial\_admin, HighBlood, Stroke, Complication\_risk, Overweight, Arthritis, Diabetes, Hyperlipidemia, BackPain, Anxiety, Allergic\_rhinitis, Reflux\_esophagitis, Asthma, and Services.

D2: Justification of Mitigation Methods

*Duplicates :*

There were no duplicates discovered; therefore, no mitigation was required.

*Missing Data :*

|  |  |  |
| --- | --- | --- |
| **Variables** | **Treatment Method** | **Reasoning** |
| Children | Changed data type to Int64 and replaced missing data with '0's | According to data dictionary, number is self reported and the NAs likely mean no children |
| Age | Changed data type to Int64 and filled missing data with ‘0’s which were then converted to the median value | Originally, the thought was to drop null values, but Age required non-null values for PCA, and after some testing, the difference is negligible |
| Income | Changed data type to int64 and filled missing data with '0's which were then converted to the median value | As this variable is quantitative, I've performed univariate imputation utilizing median to increase data quality |
| Soft\_drink | Replaced missing data with 'No' which was then changed to 'no' | Based on the wording in the data dictionary, NAs likely mean 'No' and were therefore simply converted. 'No' became 'no' to fit the data dictionary, as well |
| Overweight | Changed data type to Int64, and replaced missing data with '0's which then became 'no' | These values were altered to fit the data dictionary |
| Anxiety | Changed data type to Int64 and replaced missing data with '0's which were then replaced with 'no' | These values were changed to adhere to the data dictionary |
| Initial\_days | Replaced missing data with median values | In this case, it made sense as a quantitative value that can not be 0 and did not skew outlier data |

*Outliers :*

|  |  |  |
| --- | --- | --- |
| **Variables** | **Treatment Method** | **Reasoning** |
| Population | No action taken / Retain | The maximum of 112,814 is within reason and based on Census data |
| Children | No action taken / Retain | The maximum of 10 children is acceptable |
| Income | No action taken / Retain | The maximum of $207,249.13 is acceptable |
| VitD\_levels | No action taken / Retain | The values range of 9-53 is acceptable |
| Doc\_visits | No action taken / Retain | A maximum of 9 doctor visits is acceptable |
| Full\_meals\_eaten | No action taken / Retain | All values appear reasonable according to the data dictionary |
| VitD\_supp | No action taken / Retain | The range of 0-5 administrations is expected |
| TotalCharge | No action taken / Retain | The maximum of $21,524.22 is within reason |
| Lat | No action taken / Retain | All values within correct range of 0-90 |
| Lng | No action taken / Retain | All values within correct range of -180-180 |

*Re-expression of categorical variables and other data quality issues :*

|  |  |  |
| --- | --- | --- |
| **Variables** | **Treatment Method** | **Reasoning** |
| Zip | Data type changed to str and added 0s to ensure 5 digits for all Zip codes | Fixed invalid zip codes and allowed them to have leading 0s as required |
| Area | Changed values to 'rural, 'urban', and 'suburban' | Match data dictionary |
| TimeZone | Renamed variable to TimeZone | Match data dictionary |
| Children | Data type changed to int64 | Accommodate NA manipulation |
| Age | Data type changed to Int64 | Accommodate NA manipulation |
| Gender | Changed values from 'Male' to 'male', 'Female' to 'female', and 'Prefer not to answer' as 'nonbinary' | Match data dictionary |
| ReAdmis | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Soft\_drink | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Initial\_admin | Renamed 'Emergency Admission', 'Elective Admission', and 'Observation Admission' to 'emergency admission', 'elective admission', and 'observation' | Match data dictionary |
| HighBlood | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Stroke | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Complication\_risk | Renamed 'High', 'Medium', and 'Low to 'high', 'medium', and 'low' | Match data dictionary |
| Overweight | Data type changed to Int64 and values '1' and '0' replaced with 'yes' and 'no' | Match data dictionary |
| Arthritis | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Diabetes | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Hyperlipidemia | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| BackPain | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Anxiety | Data type changed to Int64 and values '1' and '0' replaced with 'yes' and 'no' | Match data dictionary |
| Allergic\_rhinitis | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Reflux\_esophagitis | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Asthma | Changed 'Yes' to 'yes' and 'No' to 'no' | Match data dictionary |
| Services | Renamed 'Blood Work', 'Intravenous', 'CT Scan', and 'MRI' to 'blood work', 'intravenous', 'CT scan', and 'MRI' | Match data dictionary |
| All floats | Rounded all float data types in data set to 2 decimal places | Readability purposes without losing integrity |

D3: Summary of the Outcomes

As there were no detected duplicates, there was no action needed to address potential issues.

In all cases, missing data was replaced with some value. Below illustrates this by comparing the counts before executing the treatment code and afterwards. As can be seen, all values afterwards are all 10000. Data types had to be altered in some circumstances to account for the methods used to alter missing values which can be seen in this example, as well. Also, with regards to Age and consideration of dropping the values, it was tested with dropped values and with median values proving that the difference is incredibly minimal.

*Age data comparison :*

|  |  |
| --- | --- |
| **Age w/ dropped values**  **Mean** | **Age w/ imputation**  **Mean** |
| 53.295676 | 53.2243 |

*Before :*

<class 'pandas.core.frame.DataFrame'>

Index: 10000 entries, 1 to 10000

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 object

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 object

27 Initial\_admin 10000 non-null object

28 HighBlood 10000 non-null object

29 Stroke 10000 non-null object

30 Complication\_risk 10000 non-null object

31 Overweight 9018 non-null float64

32 Arthritis 10000 non-null object

33 Diabetes 10000 non-null object

34 Hyperlipidemia 10000 non-null object

35 BackPain 10000 non-null object

36 Anxiety 9016 non-null float64

37 Allergic\_rhinitis 10000 non-null object

38 Reflux\_esophagitis 10000 non-null object

39 Asthma 10000 non-null object

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(11), int64(14), object(27)

memory usage: 4.0+ MB

*After :*

<class 'pandas.core.frame.DataFrame'>

Index: 10000 entries, 1 to 10000

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 object

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 10000 non-null int64

15 Age 10000 non-null Int64

16 Education 10000 non-null object

17 Employment 10000 non-null object

18 Income 10000 non-null int64

19 Marital 10000 non-null object

20 Gender 10000 non-null object

21 ReAdmis 10000 non-null object

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 10000 non-null object

27 Initial\_admin 10000 non-null object

28 HighBlood 10000 non-null object

29 Stroke 10000 non-null object

30 Complication\_risk 10000 non-null object

31 Overweight 10000 non-null object

32 Arthritis 10000 non-null object

33 Diabetes 10000 non-null object

34 Hyperlipidemia 10000 non-null object

35 BackPain 10000 non-null object

36 Anxiety 10000 non-null object

37 Allergic\_rhinitis 10000 non-null object

38 Reflux\_esophagitis 10000 non-null object

39 Asthma 10000 non-null object

40 Services 10000 non-null object

41 Initial\_days 10000 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: Int64(1), float64(6), int64(15), object(30)

memory usage: 4.1+ MB

All quantitative variables were considered for their potential outliers reducing their usefulness. All of the values for the quantitative variables were at least possible if not expected. With all values seemingly logical and reasonable, it seems prudent to leave them all as is.

There were a large number of changes in lettering or phrasing to align the data with that within the data dictionary. Data types were altered as needed to achieve that goal, also. Zips were converted to a usable state. Lastly, I decided to round all decimals to the hundredth spot for all floats in order to assist with readability and functionality while maintaining data integrity.

*Table showing Zip, decimal truncation, and data dictionary changes*

A screenshot of a graph

Description automatically generated

D4: Mitigation Code

*See code attached :* d206completelist.ipynb

*Additional copy of treatment code below :*

# Setting Zip to str and filling front digits with 0s up to 5 digits to create accurate, functional Zip codes

# [In-Text Citation (Untubu, 2015)]

df['Zip'] = df['Zip'].astype("str").str.zfill(5)

# Renaming to match data dictionary

df['Area'] = df['Area'].map({'Rural': 'rural', 'Urban': 'urban', 'Suburban': 'suburban'})

# Timezone renamed TimeZone per data dictionary by using rename in pandas

df.rename(columns={'Timezone': 'TimeZone'}, inplace=True)

# Children Data mismatch - Had to fill in NAs with 0s before changing float64 to Int64

# Children Missing data - Simply replaced with 0

df['Children'] = df['Children'].fillna(0).astype(np.int64)

# Age data type mismatch - changing to Int64

# Missing values, Int64 allows dealing with NA

df['Age'] = df['Age'].astype("Int64")

df['Age'].dropna(inplace=True)

# Setting outliers to NaN

df['Income'] = np.where(df['Income'] > 127,029, np.nan, df['Income'])

# Setting NaN outliers to Median value

df['Income'].fillna(df['Income'].median(), inplace=True)

# Changed data type to int64

df['Income'] = np.int64(df['Income'])

# Change 'Prefer not to answer' to 'nonbinary' as per data dictionary using map [In-Text Citation: (GeeksforGeeks, 2024)]

# Change Male and Female to lowercase to match data dictionary

df['Gender'] = df['Gender'].map({'Male': 'male', 'Female': 'female', 'Prefer not to answer': 'nonbinary'})

# Remap Yes/No to yes/no per data dictionary

df['ReAdmis'] = df['ReAdmis'].map({'Yes':'yes', 'No':'no'})

# Null values rows filled with No to become no

df.Soft\_drink.fillna('No', inplace=True)

# Soft drink changed from Yes / No to yes/no per data dictionary

df['Soft\_drink'] = df['Soft\_drink'].map({'Yes': 'yes', 'No': 'no'})

# Changing phrases and words to match data dictionary

df['Initial\_admin'] = df['Initial\_admin'].map({'Emergency Admission': 'emergency admission', 'Elective Admission': 'elective admission', 'Observation Admission': 'observation'})

# Yes/No to yes/no data dictionary

df['HighBlood'] = df['HighBlood'].map({'Yes': 'yes', 'No': 'no'})

# Changing Yes/No to yes/no

df['Stroke'] = df['Stroke'].map({'Yes': 'yes', 'No': 'no'})

# Changed to lowercase per data dictionary

df['Complication\_risk'] = df['Complication\_risk'].map({'High': 'high', 'Medium': 'medium', 'Low':'low'})

# Data type mismatch Overweight float64 to Int64

# Replace null values with No

# Changed 1,0s to yes,no

df['Overweight'] = df['Overweight'].fillna(0).astype('Int64')

df['Overweight'] = df['Overweight'].astype(str)

df.Overweight.replace({'1':'yes', '0':'no'}, inplace=True)

# Yes/No replaced with yes/no

df['Arthritis'] = df['Arthritis'].map({'Yes': 'yes', 'No': 'no'})

# Converting Yes/No to yes/no

df['Diabetes'] = df['Diabetes'].map({'Yes': 'yes', 'No': 'no'})

# Yes/No yes/no replacement

df['Hyperlipidemia'] = df['Hyperlipidemia'].map({'Yes': 'yes', 'No': 'no'})

# Yes/No to yes/no

df['BackPain'] = df['BackPain'].map({'Yes': 'yes', 'No': 'no'})

# Data type to Int64

# Replace null values with 0

df['Anxiety'] = df['Anxiety'].fillna(0).astype('Int64')

# Remap values from 1,0 to yes/no

df['Anxiety'] = df['Anxiety'].astype(str)

df.Anxiety.replace({'1':'yes', '0':'no'}, inplace=True)

# Changed Yes/No to yes/no

df['Allergic\_rhinitis'] = df['Allergic\_rhinitis'].map({'Yes': 'yes', 'No': 'no'})

# Changed Yes/No to yes/no

df['Reflux\_esophagitis'] = df['Reflux\_esophagitis'].map({'Yes': 'yes', 'No': 'no'})

# Yes/No to yes/no

df['Asthma'] = df['Asthma'].map({'Yes': 'yes', 'No': 'no'})

# Remapping Blood Work and Intravenous with lower case per data dictionary

df['Services'] = df['Services'].map({'Blood Work': 'blood work', 'Intravenous': 'intravenous', 'CT Scan': 'CT scan', 'MRI': 'MRI'})

# Dropping null values rows

df['Initial\_days'] = np.where(df['Initial\_days'] == '', np.nan, df['Initial\_days'])

# Filling null values with median

df['Initial\_days'].fillna(df['Initial\_days'].median(), inplace=True)

# Rounding all floats in data set to 2 decimal places for readability without losing functionality

# [In-Text Citation: (Saturn Cloud, 2023)]

for col in df.columns:

if isinstance(df[col].iloc[0], float):

df[col] = df[col].apply(lambda x: round(x, 2))

# Exporting clean .csv

df.to\_csv('cleand206data.csv', index=False)

df.head(20)

D5: Clean Data

*See attached .csv file:* cleand206data.csv

D6: Limitations

Some potential limitations were observed over the course of this project. As there were no duplicates to be dealt with, there were no limitations in that regard. However, it could have been useful to be able to access resources or seek answers to questions regarding missing values. Without these, I relied on my personal judgment. For instance, univariate imputation was chosen for the Income variable merely due to wanting to include information related to that rather than merely dropping the missing variables or leaving them as 0s which would follow the logic used in other missing value questions.

Another aspect that required a great deal of subjective judgment was dealing with outliers. To me, and based on my knowledge, the values are all within expected or acceptable levels and should be retained. For instance, the higher value for TotalCharge may be far higher than expected, but based on my available resources, I would say that these values are well within range.

Also, I decided to adhere very strictly to the data dictionary. Performing tasks such as changing capital letters to lowercase ones and vice versa may not have been intended and would be yet another limitation. Lastly, I unilaterally truncated decimals to the hundredths place which may or may not be an issue depending on the intended usage of this data but logically made sense to me. My breadth of knowledge and experience could certainly present unforeseen limitations, as well.

D7: Impact of Limitations

The question posited in section A was “what factors affect readmission?” Now that the data has been cleaned and the limitations have been explored, prognosticating the effects of those limitations in answering the initial question is possible. As far as the manipulation of missing values, a data analyst may have not wanted any imputation performed whatsoever. They may also have a different belief about NA or missing value correlations for certain variables, such as Age or Initial\_days. These changes alone could influence how the data affects readmission rates.

My methods of dealing, or rather not dealing, with outliers in this data set also could pose problematic for the data analyst. There were many instances of zscores being greater than 2 which allows for multiple avenues of manipulating data to better suit one’s needs. This may be an example of my lack of experience coming into place, and another may have chosen a different method that increased the quality of data for understanding readmissions specifically.

Furthermore, my decision to lower the amount of decimals used may create another limitation if the increased significant digits provided significantly higher quality data for determining factors affecting readmission that I overlooked.

Part IV - PCA

E1: Principal Components

The variables used to perform Principal Component Analysis for the medical\_data data set were Lat, Lng, Population, Children, Age, Income, VitD\_levels, Full\_meals\_eaten, VitD\_supp, Initial\_days, TotalCharge, and Additional\_charges.

*See attached code :* d206completelist.ipynb

*Code for PCA :*

# select columns of the dataframe (Following d206 course .pdf)

dfpca = df[['Lat', 'Lng', 'Population', 'Children', 'Age', 'Income', 'VitD\_levels', 'Doc\_visits', 'Full\_meals\_eaten', 'VitD\_supp', 'Initial\_days', 'TotalCharge', 'Additional\_charges']]

# normalize by using Standardization (Mean Normalization)

dfpca\_normalized=(dfpca-dfpca.mean())/dfpca.std()

# Applying PCA cont step A

pca = PCA(n\_components=dfpca.shape[1])

# Step B

pca.fit(dfpca\_normalized)

# Step C

dfpca\_pca = pd.DataFrame(pca.transform(dfpca\_normalized), columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11', 'PC12', 'PC13'])

# Step 4 PCA Loadings

loadings = pd.DataFrame(pca.components\_.T, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11', 'PC12', 'PC13'], index=dfpca\_normalized.columns)

loadings

*PCA Loading Matrix :*

A white background with black numbers

Description automatically generated

E2: Criteria Used

According to the data, we should retain PC1, PC2, PC3, PC4, PC5, and PC6 in accordance with the Kaiser rule. The others fall below the eigenvalue threshold of 1. These are also in order of significance as a higher value indicates a more meaningful clustering of data. Looking at the scree plot, we can use the scree plot test and see that the values indicated by the “elbow” are the same as we obtained via the Kaiser rule (Course Materials, n.d.).

*Scree Plot :*

A graph with a line and a red line

Description automatically generated

*Eigenvalues:*

|  |  |
| --- | --- |
| **Principal Component** | **Eigenvalue** |
| PC1 | 1.9515990463229447 |
| PC2 | 1.6189108284414147 |
| PC3 | 1.2263955306571765 |
| PC4 | 1.0428013708092918 |
| PC5 | 1.023633698955772 |
| PC6 | 1.0163083982153713 |
| PC7 | 0.9960191979449734 |
| PC8 | 0.9930386277885204 |
| PC9 | 0.9792727196479881 |
| PC10 | 0.9723189403658262 |
| PC11 | 0.7469493050263638 |
| PC12 | 0.3778906273357725 |
| PC13 | 0.053561708488375055 |

*Code for eigenvalues :*

# Step 5 Selecting PCs Step A (Continuing to follow course .pdf)

cov\_matrix = np.dot(dfpca\_normalized.T, dfpca\_normalized) / dfpca.shape[0]

# Step 5 Step B

eigenvalues = [np.dot(eigenvector.T, np.dot(cov\_matrix, eigenvector)) for eigenvector in pca.components\_]

# Step C components above 1 eigenvalue are worth using.

plt.plot(np.arange(1, len(eigenvalues)+1), eigenvalues, marker='o')

plt.xlabel('number of components')

plt.ylabel('eigenvalue')

plt.axhline(y=1, color='red')

plt.show()

eigenvalues

E3: Benefits

Principal Component Analysis can provide exceptional benefits to any organization looking to further their data utilization and more efficiently distribute resources. Its primary benefit is in the reduction of dimensions creating more efficient performance. This assists with narrowing down important groupings and consolidation of variables. Furthermore, PCA can then be used to develop easily understandable 2-dimensional physical visualizations and presentations of complex data interactions (Bigabid, 2021).

In my previous example of PCA, there were originally 13 variables input and therefore 13 PCs tested. PC1 through PC6 all had eigenvalues above one showing a high significance. PC7 through PC13 had values lower than one and can therefore be disregarded. This data reveals correlations that may not normally be noticeable. In this example, PC1 has the highest positive relationships with Initial\_days, VitD\_levels, and TotalCharge, but it does also has positive relationships with Additional\_charges, VitD\_supp, Age, Children, and Population. This decreased the focus from 13 groups to only six, a 54% reduction.

F: Video

*Link for Panopto Video :*

*https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=03356439-a59d-4207-bea9-b158013a24f3*

G: Sources of Third-Party Code

GeeksforGeeks. *Pandas Replace Multiple Values in Python*. (2024, March 6). GeeksforGeeks. https://www.geeksforgeeks.org/pandas-replace-multiple-values-in-python/

Prabhakaran, S. (2023, July 30). *How to detect outliers using IQR and boxplots?*. Machine Learning Plus. https://www.machinelearningplus.com/machine-learning/how-to-detect-outliers-using-iqr-and-boxplots/

Python Software Foundation. *statistics — Mathematical statistics functions — Python 3.9.18 documentation*. (n.d.). Docs.python.org. Retrieved April 20, 2024, from https://docs.python.org/3.9/library/statistics.html#module-statistics

Saturn Cloud. (2023, June 19). *How to Set Decimal Precision of a Pandas Dataframe Column with Decimal Datatype | Saturn Cloud Blog*. (2023, June 19). Saturncloud.io. https://saturncloud.io/blog/how-to-set-decimal-precision-of-a-pandas-dataframe-column-with -decimal-datatype/

Unutbu. (2015, Oct 15). *Python - Loading Zip Codes into a DataFrame as Strings?* (n.d.). Stack Overflow. https://stackoverflow.com/questions/33137686/python-loading-zip-codes-into-a-dataframe-as -strings

H: Sources

Bigabid. *What is Principal Component Analysis (PCA) & How to Use It?* (2021, June 6). Bigabid. http://www.bigabid.com/what-is-pca-and-how-can-i-use-it/

Chantal D. Larose, Daniel T. Larose. Data Science Using Python and R. Wiley; 2019. Accessed March 30, 2024. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=2091371&site=eds- live&scope=site

Course Materials

Lustig, Irv. *Data Wrangling Syntax -Creating DataFrames Tidy Data -A foundation for wrangling in pandas*. (n.d.). https://pandas.pydata.org/Pandas\_Cheat\_Sheet.pdf

Western Governors University. (n.d.). *R or Python*. Western Governors University. https://www.wgu.edu/online-it-degrees/programming-languages/r-or-python.html