|  |
| --- |
| Data Cleaning: medical\_data |
| Western Governors University |
| Heather Rauch |

Table of Contents

[Part I: Research Question and Variables 2](#_Toc114007087)

[A: Question or Decision 2](#_Toc114007088)

[B: Required Variables 2](#_Toc114007089)

[Part II: Data-Cleaning Plan 10](#_Toc114007139)

[C: Data-Cleaning Plan (Detection) 10](#_Toc114007140)

[C1: Plan to Find Anomalies 10](#_Toc114007141)

[C2: Justification of Approach 10](#_Toc114007142)

[C3: Justification of Tools 10](#_Toc114007143)

[C4: Provide the Code 11](#_Toc114007144)

[Part III: Data Cleaning 11](#_Toc114007145)

[D: Data Cleaning 15](#_Toc114007146)

[D1: Cleaning Findings 15](#_Toc114007147)

[D2: Justification of Mitigation Methods 18](#_Toc114007148)

[D3: Summary of the Outcomes 19](#_Toc114007149)

[D4: Mitigation Code 21](#_Toc114007150)

[D5: Clean Data 21](#_Toc114007151)

[D6: Limitations 25](#_Toc114007152)

[D7:Impact of the Limitations 25](#_Toc114007153)

[E: Principal Component Analysis (PCA) 25](#_Toc114007154)

[E1: Principal Components 25](#_Toc114007155)

[E2: Criteria Used 25](#_Toc114007156)

[E3: Benefits 26](#_Toc114007157)

[Part IV. Supporting Documents 27](#_Toc114007158)

[F: Video 27](#_Toc114007159)

[G: Sources for Third-Party Code 27](#_Toc114007160)

[H: Sources 27](#_Toc114007161)

# ****Part I: Research Question and Variables****

## A: Question or Decision

Which medical conditions and factors predict whether a patient will be readmitted to a hospital?

## B: Required Variables

### CaseOrder

* Data type: CaseOrder is quantitative because it specifies the numeric order in which the source records were originally provided.
* Description: The CaseOrder variable is a sequential integer, ranging from 1 to 10,000, which is assigned to each row to maintain the original dataset order.
* Example: The first row contains a value of 1, the second row contains a value of 2, and the nth row contains a value of n.

### Customer\_id

* Data type: Customer\_id is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Customer\_id variable is a unique alphanumeric code used to identify each patient.
* Example: Some examples include “C412403”, “Z919181”, and “F995323”.

### Interaction

* Data type: Interaction is qualitative because it is non-numeric and does not measure or quantify anything
* Description: The Interaction variable is a unique alphanumeric code used to identify each patient interaction.
* Example: Some examples include “8cd49b13-f45a-4b47-a2bd-173ffa932c2f”, “d2450b70-0337-4406-bdbb-bc1037f1734c”, and “a2057123-abf5-4a2c-abad-8ffe33512562”.

### UID

* Data type: UID is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The UID variable is a unique alphanumeric code used to identify each patient transaction or procedure.
* Example: Some examples include “3a83ddb66e2ae73798bdf1d705dc0932”, “176354c5eef714957d486009feabf195”, and “e19a0fa00aeda885b8a436757e889bc9”.

### City

* Data type: City is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The City variable represents the city portion of each patient’s mailing address.
* Example: Some examples include “Eva”, “Marianna”, and “Sioux Falls”.

### State

* Data type: Stat is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The State variable represents the two-letter abbreviation for the state portion of each patient’s mailing address.
* Example: Some examples include “AL”, “FL”, and “SD”.

### County

* Data type: County is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The County variable represents the municipal County in which each patient’s mailing address resides.
* Example: Some examples include “Morgan”, “Jackson”, and “Minnehaha”.

### Zip

* Data type: Zip must be read in as a string variable to ensure leading zeroes are not dropped from the beginning of each number. Because of this and the fact that zip codes are labels that do not quantify anything, Zip is intentionally being classified as a quantitative variable.
* Description: The Zip variable represents the five-digit postal code from each patient’s mailing address.
* Example: Some examples include “35621”, “32446”, and “32446”.

### Lat

* Data type: Lat is quantitative because it measures the degrees of each horizontal line used to quantify individual locations on Earth.
* Description: The Lat variable represents the latitude portion of the GPS coordinates of each patient’s mailing address.
* Example: Some examples include 34.34960, 30.84513, and 43.54321.

### Lng

* Data type: Lng is quantitative because it measures the degrees of each vertical line used to quantify individual locations on Earth.
* Description: The Lng variable represents the longitude position of the GPS coordinates of each patient’s mailing address.
* Example: Some examples include -86.72508, -85.22907, -96.63772

### Population

* Data type: Population is quantitative because it measures the count of people.
* Description: The Population variable represents the number of people who live within a one-mile radius of the patient according to the United States Census.
* Example: Some examples include 2951, 11303, and 17125.

### Area

* Data type: Area is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Area variable represents the type of land on which each patient’s address resides.
* Example: Some examples include “Suburban”, “Urban”, and “Rural”.

### Timezone

* Data type: Timezone is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Timezone variable represents the time zone in which the patient’s address resides.
* Example: Some examples include “America/Chicago”, “America/New\_York”, and “America/Denver”.

### Job

* Data type: Job is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Job variable represents the professional title of the holder of the patient’s insurance policy. This could be the job of the patient or a family member depending on who the primary insurance policyholder is.
* Example: Some examples include “Psychologist, sport and exercise”, “Community development worker”, and “Chief Executive Officer”.

### Children

* Data type: Children is quantitative because it measures the count of children.
* Description: The Children variable represents the number of children each patient reports as living in the same household.
* Example: Some examples include 1, 3, and 0.

### Age

* Data type: Age is quantitative because it measures the years a person has been alive.
* Description: The Age variable represents the patient’s age in years as of their last birthdate.
* Example: Some examples include 53, 51, and 78.

### Education

* Data type: Education is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Education variable represents the patient’s highest level of education completed.
* Example: Some examples include “Some College, Less than 1 Year”, Come College, 1 or More Years, No Degree”, and “GED or Alternative Credential”.

### Employment

* Data type: Employment is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Employment variable represents the patient’s employment status.
* Example: Some examples include “Full Time”, “Retired”, and “Unemployed”.

### Income

* Data type: Income is quantitative because it measures the amount of money earned annually.
* Description: The Income variable represents the annual income of the holder of the patient’s insurance policy. This could be the income of the patient or a family member depending on who the primary insurance policyholder is.
* Example: Some examples include 86575.93, 46805.99, and 14370.14.

### Marital

* Data type: Marital is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Marital variable represents the marital status of the holder of the patient’s insurance policy. This could be the marital status of the patient or a family member depending on who the primary insurance policyholder is.
* Example: Some examples include “Divorced”, “Married”, and “Widowed”.

### Gender

* Data type: Gender is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Gender variable represents the patient’s self-identified gender.
* Example: Some examples include “Male”, “Female”, and “Prefer not to answer”.

### ReAdmis

* Data type: ReAdmis is a qualitative variable because it is non-numeric and does not measure or quantify anything.
* Description: The ReAdmis variable represents whether the patient has been re-admitted within the previous month.
* Example: Some examples include “No” and “Yes”.

### VitD\_levels

* Data type: VitD\_levels is quantitative because it measures the level of vitamin D in a patient.
* Description: The VitD\_levels variable represents the vitamin D level of the patient in nanograms per milliliter (ng/mL).
* Example: Some examples include 17.802330, 18.994640, and 17.415889.

### Doc\_visits

* Data type: Doc\_visits is quantitative because it measures the count of times the doctor was seen.
* Description: The Doc\_visits variable represents the number of visits the patient had with a primary physician the first time they were hospitalized.
* Example: Some examples include 6, 4, and 5.

### Full\_meals\_eaten

* Data type: Full\_meals\_eaten is quantitative because it measures the count of meals.
* Description: The Full\_meals\_eaten variable represents a count of full (not partial) meals consumed by the patient. Partial meals are not counted.
* Example: Some examples include 0, 2, and 1.

### VitD\_supp

* Data type: VitD\_supp is quantitative because it measures the count of supplements received.
* Description: The VitD\_supp variable represents the count of vitamin D supplements administered to the patient.
* Example: Some examples include 0, 1, and 2.

### Soft\_drink

* Data type: Soft\_drink is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Soft\_drink variable represents whether or not the patient regularly consumes three or more soft drinks daily.
* Example: Some examples include “Yes” and “No”.

### Initial\_admin

* Data type: Initial admin is qualitative because it is non-numeric and does not measure or quantify anything
* Description: The Initial\_admin variable represents the urgency or severity of the initial admission.
* Example: Some examples include “Emergency Admission”, “Elective Admission”, and “Observation Admission”.

### HighBlood

* Data type: HighBlood is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The HighBlood variable represents whether or not the patient has a diagnosis of high blood pressure.
* Example: Some examples include “Yes” and “No”.

### Stroke

* Data type: Stroke is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Stroke variable represents whether or not the patient has had a diagnosis of stroke.
* Example: Some examples include “Yes” and “No”.

### Complication\_risk

* Data type: Complication\_risk is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Complication\_risk variable represents the patient’s level of risk for possible complications.
* Example: Some examples include “High”, “Medium”, and “Low”.

### Overweight

* Data type: Overweight is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Overweight variable represents whether the patient’s height, weight, and age indicate they are overweight.
* Example: Some examples include “Yes” and “No”.

### Arthritis

* Data type: Arthritis is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Arthritis variable represents whether the patient has a diagnosis of arthritis.
* Example: Some examples include “Yes” and “No”.

### Diabetes

* Data type: Diabetes is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Diabetes variable represents whether the patient has a diagnosis of diabetes.
* Example: Some examples include “Yes” and “No”.

### Hyperlipidemia

* Data type: Hyperlipidemia is qualitative because it is non-numeric and does not measure anything.
* Description: The Hyperlipidemia variable represents whether the patient has a diagnosis of hyperlipidemia.
* Example: Some examples include “Yes” and “No”.

### BackPain

* Data type: BackPain is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The BackPain variable represents whether the patient has a diagnosis of chronic back pain.
* Example: Some examples include “Yes” and “No”.

### Anxiety

* Data type: Anxiety is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Anxiety variable represents whether the patient has a diagnosis of anxiety.
* Example: Some examples include “Yes” and “No”.

### Allergic\_rhinitis

* Data type: Allergic\_rhinitis is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Allergic\_rhinitis variable represents whether the patient has a diagnosis of allergic rhinitis.
* Example: Some examples include “Yes” and “No”.

### Reflux\_esophagitis

* Data type: Reflux\_esophagitis is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Reflux\_esophagitis variable represents whether the patient has a diagnosis of reflux esophagitis.
* Example: Some examples include “Yes” and “No”.

### Asthma

* Data type: Asthma is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Asthma variable represents whether the patient has a diagnosis of asthma.
* Example: Some examples include “Yes” and “No”.

### Services

* Data type: Services is qualitative because it is non-numeric and does not measure or quantify anything.
* Description: The Services variable represents the type of services the patient received during their hospital stay.
* Example: Some examples include “Blood Work”, “Intravenous”, and “CT Scan”.

### Initial\_days

* Data type: Initial\_days is quantitative because it measures a period of time.
* Description: The Initial\_days variable represents the total number of days the patient stayed during their initial hospitalization.
* Example: Some examples include 10.585770, 15.129562, and 4.772177.

### TotalCharge

* Data type: TotalCharge is quantitative because it measures an amount of money.
* Description: The TotalCharge variable represents the average daily charges received by the patient, excluding those for specialized treatments.
* Example: Some examples include 3191.048774, 4214.905346, and 2177.586768.

### Additional\_charges

* Data type: Additional\_charges is quantitative because it measures an amount of money.
* Description: The Additional\_charges variable represents the average of miscellaneous charges received by the patient.
* Example: Some examples include 17939.403420, 17612.998120, and 17505.192460.

### Item1

* Data type: Item1 is quantitative because it is numeric and measures a rating.
* Description: The Item1 variable represents the patient’s rating of timely admission on a scale of most important (1) to least important (8)
* Example: Some examples include 3, 2, and 1.

### Item2

* Data type: Item2 is quantitative because it is numeric and measures a rating.
* Description: The Item2 variable represents the patient’s rating of timely treatment on a scale of most important (1) to least important (8).
* Example: Some examples include 3, 4, and 5.

### Item3

* Data type: Item3 is quantitative because it is numeric and measures a rating.
* Description: The Item3 variable represents the patient’s rating of timely visits on a scale of most important (1) to least important (8).
* Example: Some examples include 2, 3, and 4.

### Item4

* Data type: Item4 is quantitative because it is numeric and measures a rating.
* Description: The Item4 variable represents the patient’s rating of reliability on a scale of most important (1) to least important(8).
* Example: Some examples include 2, 4, and 3.

### Item5

* Data type: Item5 is quantitative because it is numeric and measures a rating.
* Description: The Item5 variable represents the patient’s rating of options on a scale of most important (1) to least important (8).
* Example: Some examples include 3, 4, and 5.

Item6

* Data type: Item5 is quantitative because it is numeric and measures a rating.
* Description: The Item6 variable represents the patient’s rating of hours of treatment on a scale of most important (1) to least important (8).
* Example: Some examples include 3, 4, and 5.

Item7

* Data type: Item7 is quantitative because it is numeric and measures a rating.
* Description: The Item7 variable represents the patient’s rating of courteous staff on a scale of most important (1) to least important (8).
* Example: Some examples include 3, 5, and 4.

Item8

* Data type: Item8 is quantitative because it is numeric and measures a rating.
* Description: The Item8 variable represents the patient’s rating of evidence of active listening from the doctor on a scale of most important (1) to least important (8).
* Example: Some examples include 4, 3, and 5.

# ****Part II: Data-Cleaning Plan****

## C: Data-Cleaning Plan (Detection)

### C1: Plan to Find Anomalies

To find anomalies in the dataset, three primary approaches were used: duplicate row detection, missing value detection, and outlier detection. To detect duplicate rows, the pandas DataFrame duplicated method was used. The pandas Series isnull method was used to detect missing values. The results were summed for each column to retrieve missing counts by column. Outlier detection was accomplished through a few steps. First, matplotlib was used to create boxplots of each numeric variable. The boxplots were visually inspected to determine on which side of the distribution the search for outliers should be performed for each column. Second, the z-score for each column with outliers was calculated using the scipy stats.zscore function. Third, Boolean outlier columns were created for each zscore, searching for values above 3 and/or below -3, based on the results of the boxplot review. In addition to the three primary approaches, data type correction was performed to align the data types with the data dictionary.

### C2: Justification of Approach

According to Webinar 2: “Getting Started with Missing Values and Outliers” from the Course Webinar video series and section 3.3, Data Preparation Phase, of the course text, Data Science Using Python and R, three common data cleaning tasks are deduplication, handling missing fields, and handling outliers. Identifying duplicate rows and missing fields is fairly straightforward, but outlier detection is not. This process is generally only applicable to numeric data and should be adjusted based on the shape of the data according to Webinar 2. In the webinar, z-scores and boxplots are most highly recommended and used in this project (Larose & Larose, 2019).

### C3: Justification of Tools

To detect anomalies in the medical\_data dataset the Python language was used with the pandas, matplotlib, and scipy packages. Python is a well-known programming and scripting language, often used for data science. The pandas package provides the ability to work with data in much the way one would with SQL, including functions for extracting, transforming, and rearranging data. matplotlib is a well-known plotting package that provides various statistical plotting methods, such as the boxplots and histograms produced in this project. SciPy is a package built for math and statistical functions like the z-scores calculated during outlier detection.

There are tools other than Python which could be used to accomplish the same or similar outcomes such as writing analytic SQL queries or utilizing spreadsheet software. However, SQL is not a language that lends itself well to statistical analyses and can’t produce the visualizations needed to assess distributions without using additional software. Most spreadsheet software can accomplish everything that is required but will take far longer to process than Python when working with larger datasets.

### C4: Provide the Code

Note: To ensure the code below runs free of error, portions are included which are not relevant to anomaly detection. Those sections are grayed out for clarity.

For the full data cleaning script, see the attached file, “task1.py”.

**import** os  
**import** sys  
**import** logging  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
**from** datetime **import** datetime  
**from** scipy **import** stats  
  
FILE\_PATH = **r'C:\Users\user\OneDrive\Documents\Education\Western Govenors University\MS - Data Analytics\Data '** \  
 **r'Cleaning\medical\_raw\_data.csv'**OUTPUT\_PATH = **r'..\output'**DTYPES = {  
 **'CaseOrder'**: int, **'Customer\_id'**: str, **'Interaction'**: str, **'UID'**: str, **'City'**: str, **'State'**: str, **'County'**: str,  
 **'Zip'**: str, **'Lat'**: float, **'Lng'**: float, **'Population'**: int, **'Area'**: str, **'Timezone'**: str, **'Job'**: str,  
 **'Children'**: float, **'Age'**: float, **'Education'**: str, **'Employment'**: str, **'Income'**: float, **'Marital'**: str,  
 **'Gender'**: str, **'ReAdmis'**: str, **'VitD\_levels'**: float, **'Doc\_visits'**: int, **'Full\_meals\_eaten'**: int, **'VitD\_supp'**: int,  
 **'Soft\_drink'**: str, **'Initial\_admin'**: str, **'HighBlood'**: str, **'Stroke'**: str, **'Complication\_risk'**: str,  
 **'Overweight'**: float, **'Arthritis'**: str, **'Diabetes'**: str, **'Hyperlipidemia'**: str, **'BackPain'**: str, **'Anxiety'**: float,  
 **'Allergic\_rhinitis'**: str, **'Asthma'**: str, **'Services'**: str, **'Initial\_days'**: float, **'TotalCharge'**: float,  
 **'Additional\_charges'**: float, **'Item1'**: int, **'Item2'**: int, **'Item3'**: int, **'Item4'**: int, **'Item5'**: int, **'Item6'**: int,  
 **'Item7'**: int, **'Item8'**: int  
}  
  
  
**def** configure\_logging(output\_path: str):  
 *"""Configures logging for the program. Output is directed to both the console and a log file.* **:param** *output\_path: The path to which the log file should be stored.* **:return***: None  
 """* logging.basicConfig(  
 level=logging.INFO,  
 format=**"%(asctime)s [%(levelname)s] %(message)s"**,  
 handlers=[  
 logging.FileHandler(output\_path),  
 logging.StreamHandler(sys.stdout)  
 ]  
 )  
  
  
**def** load\_data(file\_path: str) -> pd.DataFrame:  
 *"""Loads a CSV file into a pandas DataFrame* **:param** *file\_path: a str representing the path at which the CSV file to be read can be located* **:return***: a pandas DataFrame object containing the contents of the CSV located at the given file path  
 """* **return** pd.read\_csv(file\_path, index\_col=**'Unnamed: 0'**, dtype=DTYPES, na\_values=pd.NA)  
  
  
**def** deduplicate(df: pd.DataFrame) -> pd.DataFrame:  
 *"""Inspects a pandas DataFrame for duplicate rows and removes them if necessary* **:param** *df: a pandas DataFrame which should be inspected for duplicate rows* **:return***: df with all duplicate rows removed, if any existed  
 """* n\_dups = df[df.duplicated()].shape[0]  
 **if** n\_dups > 0:  
 logging.info(**f'{**n\_dups**} duplicates found.\nDropping duplicate records.'**)  
 df.drop\_duplicates(inplace=**True**)  
 deduplicate(df)  
 **else**:  
 logging.info(**'No duplicates found.'**)  
 **return** df  
  
  
**def** detect\_missing(df: pd.DataFrame):  
 *"""Inspects each column of a pandas DataFrame for missing values* **:param** *df: a pandas DataFrame which should be inspected for missing values* **:return***: None  
 """* **for** col **in** df.columns:  
 n\_null = df[col].isnull().sum()  
 **if** n\_null > 0 **and 'zscore' not in** col:  
 logging.info(**f'{**col**} missing {**n\_null**} ({**n\_null / df.shape[0] \* 100**}%) values'**)  
  
  
**def** impute\_missing(df: pd.DataFrame, methods: dict) -> pd.DataFrame:  
 *"""Imputes missing values in a pandas DataFrame according to the provided map* **:param** *df: a pandas DataFrame with values to be imputed* **:param** *methods: a dict with the structure {column name: imputation method}. Acceptable imputation methods include  
 'mean', 'median', and 'mode'.* **:return***: pandas DataFrame with imputed values  
 """* **for** col, method **in** methods.items():  
 logging.info(**f'Imputing {**col**} with the {**method**}'**)  
 **if** method == **'mean'**:  
 df[**f'{**col**}\_imputed'**] = df[col].isna()  
 df[col].fillna(df[col].mean(), inplace=**True**)  
 **elif** method == **'median'**:  
 df[**f'{**col**}\_imputed'**] = df[col].isna()  
 df[col].fillna(df[col].median(), inplace=**True**)  
 **elif** method == **'mode'**:  
 df[**f'{**col**}\_imputed'**] = df[col].isna()  
 df[col].fillna(df[col].mode()[0], inplace=**True**)  
 **else**:  
 **raise** NotImplementedError(**f'{**method**} is not an implemented imputation method.'**)  
 **return** df  
  
  
**def** detect\_outliers(df: pd.DataFrame, query: dict) -> pd.DataFrame:  
 *"""Inspects each numeric column of a pandas DataFrame for outliers* **:param** *df: a pandas DataFrame which should be inspected for outliers* **:param** *query: a dict of search strings to be used to find outliers* **:return***: df with z-score and outlier columns added for each numeric column  
 """  
 # Add z-score and outlier columns* **for** col, query\_string **in** query.items():  
 df[**f'{**col**}\_zscore'**] = stats.zscore(df[col])  
 df[**f'{**col**}\_outlier'**] = df.index.isin(df.query(query\_string).index)  
 outliers = df[**f'{**col**}\_outlier'**].sum()  
 **if** outliers > 0:  
 logging.info(**f'{**col**} contains {**outliers**} outliers'**)  
 **return** df  
  
  
**def** float\_to\_yn(x: float) -> str:  
 *"""Used to transform yes/no fields read in as floats* **:param** *x: 1. or 0.* **:return***: 'Yes' or 'No'  
 """* **if** x == 1.:  
 **return 'Yes'  
 elif** x == 0.:  
 **return 'No'  
  
  
def** has\_duplicate\_digits(number: int) -> bool:  
 *"""Used to check integers for sequentially duplicative digits* **:param** *number: an integer to be checked* **:return***: True or False depending on whether duplicate digits are found  
 """* num\_str = str(number)  
 **for** i, num **in** enumerate(num\_str):  
 **if** i + 1 == len(num\_str):  
 **return False  
 if** num == num\_str[i + 1]:  
 **return True  
  
  
def** main():  
 date\_time = **f'{**datetime.now().strftime(**"%Y%m%d%H%M%S%f"**)**}'** log\_dir = os.path.join(OUTPUT\_PATH, date\_time)  
 log\_path = os.path.join(log\_dir, **f'{**date\_time**}.log'**)  
 os.makedirs(OUTPUT\_PATH, exist\_ok=**True**)  
 os.makedirs(log\_dir)  
 configure\_logging(log\_path)  
  
 filename = \_\_file\_\_.split(**'\\'**)[-1]  
 logging.info(**f'{**5 \* **"\*"} Running {**filename**} {**5 \* **"\*"}'**)  
  
 logging.info(**f'Loading medical\_data from {**FILE\_PATH**}'**)  
 medical\_data = load\_data(FILE\_PATH)  
  
 logging.info(**f'{**5 \* **"\*"} Detecting duplicate rows {**5 \* **"\*"}'**)  
 deduplicate(medical\_data)  
  
 logging.info(**f'{**5 \* **"\*"} Detecting missing values {**5 \* **"\*"}'**)  
 detect\_missing(medical\_data)  
  
 logging.info(**f'{**5 \* **"\*"} Imputing missing values {**5 \* **"\*"}'**)  
 methods = {  
 **'Children'**: **'median'**,  
 **'Age'**: **'mean'**,  
 **'Income'**: **'median'**,  
 **'Soft\_drink'**: **'mode'**,  
 **'Overweight'**: **'mode'**,  
 **'Anxiety'**: **'mode'**,  
 **'Initial\_days'**: **'median'** }  
 medical\_data\_imputed = impute\_missing(medical\_data.copy(), methods)  
  
 *# Align dtypes with data dictionary  
 # performing prior to imputation threw errors* int\_cols = [**'Children'**, **'Age'**]  
 **for** col **in** int\_cols:  
 medical\_data\_imputed[col] = medical\_data\_imputed[col].astype(int)  
 yn\_cols = [**'Overweight'**, **'Anxiety'**]  
 **for** col **in** yn\_cols:  
 medical\_data\_imputed[col] = medical\_data\_imputed[col].apply(float\_to\_yn, convert\_dtype=**True**)  
  
 logging.info(**'Inspecting distributions before outlier detection...'**)  
 fields = [[**'Lat'**, **'Lng'**, **'Population'**, **'Children'**, **'Age'**, **'Income'**],  
 [**'VitD\_levels'**, **'Doc\_visits'**, **'Full\_meals\_eaten'**, **'VitD\_supp'**, **'Initial\_days'**, **'TotalCharge'**],  
 [**'Additional\_charges'**]]  
 plt.close()  
 fig, ax = plt.subplots(3, 6, figsize=(8, 5))  
 fig.set\_tight\_layout(**True**)  
 fig.suptitle(**'medical\_data distributions'**)  
 **for** r, row **in** enumerate(fields):  
 **for** c, col **in** enumerate(row):  
 ax[r, c].boxplot(medical\_data\_imputed[col], sym=**'.'**, labels=[col])  
 plt.savefig(os.path.join(log\_dir, **'pre\_outlier\_dist.png'**))  
  
 logging.info(**f'{**5 \* **"\*"} Detecting outliers {**5 \* **"\*"}'**)  
 outlier\_query = {  
 **'Lat'**: **'Lat\_zscore > 3 | Lat\_zscore < -3'**,  
 **'Lng'**: **'Lng\_zscore < -3'**,  
 **'Population'**: **'Population\_zscore > 3'**,  
 **'Children'**: **'Children\_zscore > 3'**,  
 **'Income'**: **'Income\_zscore > 3'**,  
 **'VitD\_levels'**: **'VitD\_levels\_zscore > 3 | VitD\_levels\_zscore < -3'**,  
 **'Full\_meals\_eaten'**: **'Full\_meals\_eaten\_zscore > 3'**,  
 **'VitD\_supp'**: **'VitD\_supp\_zscore > 3'**,  
 **'TotalCharge'**: **'TotalCharge\_zscore > 3'**,  
 **'Additional\_charges'**: **'Additional\_charges\_zscore > 3'** }  
 medical\_data\_outliers = detect\_outliers(medical\_data\_imputed.copy(), outlier\_query)  
  
 logging.info(**f'{**5 \* **"\*"} Investigating Outliers {**5 \* **"\*"}'**)  
  
 location\_outliers = medical\_data\_outliers.query(**'Lat\_outlier | Lng\_outlier'**)[  
 [**'State'**, **'Lat'**, **'Lng'**, **'Lat\_outlier'**, **'Lng\_outlier'**]].sort\_values([**'Lat'**, **'Lng'**])  
 location\_outliers\_grouped = location\_outliers.groupby(**'State'**, as\_index=**False**).mean()  
 logging.info(**f'Lat and Lng outliers grouped by State...\n{**location\_outliers\_grouped.to\_string()**}'**)  
  
 population\_outliers = medical\_data\_outliers.query(**'Population\_outlier'**)[[**'State'**, **'City'**, **'Area'**, **'Population'**]]. \  
 sort\_values(**'Population'**, ascending=**False**)  
 logging.info(**f'Population outliers...\n{**population\_outliers.head().to\_string()**}'**)  
 first\_two = medical\_data\_outliers.query(**'Population == 122814'**)[  
 [**'Lat'**, **'Lng'**, **'Job'**, **'Children'**, **'Education'**, **'Marital'**, **'Gender'**, **'Age\_imputed'**, **'Income\_imputed'**]]  
 logging.info(**f'Inspecting first two records...\n{**first\_two.to\_string()**}'**)  
 others = medical\_data\_outliers.query(**'not Population\_outlier & State == "TX" & Area == "Rural"'**).\  
 Population.describe()  
 logging.info(**f'Looking at population for other rural Texas residents...\n{**others.to\_string()**}'**)  
 logging.info(**f'Checking for other entries with duplicate digits...'**)  
 population\_outliers[**'duplicate\_digits'**] = population\_outliers.Population.apply(has\_duplicate\_digits)  
 num = population\_outliers.duplicate\_digits.sum()  
 den = population\_outliers.shape[0]  
 logging.info(**f'{**num**}/{**den**} ({**num / den \* 100**}%) values contain duplicate digits'**)  
  
 children\_outliers = medical\_data\_outliers.query(**'Children\_outlier'**)[[**'Marital'**, **'Children'**]].\  
 groupby(**'Marital'**).count().sort\_values(**'Children'**, ascending=**False**)  
 logging.info(**f'Children outliers...\n{**children\_outliers.to\_string()**}'**)  
  
 income\_outliers = medical\_data\_outliers.query(**'Income\_outlier'**)[[**'Education'**, **'Employment'**, **'Income'**]].\  
 groupby([**'Education'**, **'Employment'**]).count().sort\_values(**'Income'**, ascending=**False**)  
 logging.info(**f'Income outliers...\n{**income\_outliers.head().to\_string()**}'**)  
  
 nutrition\_outliers = medical\_data\_outliers.\  
 query(**'VitD\_levels\_outlier | Full\_meals\_eaten\_outlier | VitD\_supp\_outlier'**)[  
 [**'VitD\_levels'**, **'Full\_meals\_eaten'**, **'VitD\_supp'**, **'VitD\_levels\_outlier'**,  
 **'Full\_meals\_eaten\_outlier'**, **'VitD\_supp\_outlier'**]  
 ].groupby([**'VitD\_levels\_outlier'**, **'Full\_meals\_eaten\_outlier'**, **'VitD\_supp\_outlier'**]).mean().\  
 sort\_values([**'VitD\_levels'**, **'Full\_meals\_eaten'**, **'VitD\_supp'**], ascending=**False**)  
 logging.info(**f'Nutrition outliers...\n{**nutrition\_outliers.to\_string()**}'**)  
  
 totalcharge\_outliers = medical\_data\_outliers.query(**'TotalCharge\_outlier'**).TotalCharge.describe()  
 totalcharge\_non\_outliers = medical\_data\_outliers.query(**'not TotalCharge\_outlier'**).TotalCharge.describe()  
 logging.info(**f'Comparing...\nTotalCharge outliers...\n{**totalcharge\_outliers**}'  
 f'\nto TotalCharge non-outliers\n{**totalcharge\_non\_outliers**}'**)  
  
 logging.info(**f'{**5 \* **"\*"} Run Complete {**5 \* **"\*"}'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

# ****Part III: Data Cleaning****

## D: Data Cleaning

### D1: Cleaning Findings

When the medical\_data dataset was reviewed for duplicate rows, none were found. However, several fields were missing data:

|  |  |  |
| --- | --- | --- |
| **Column** | **Number Missing** | **Percent Missing** |
| Children | 2588 | 25.88% |
| Age | 2414 | 24.14% |
| Income | 2464 | 24.64% |
| Soft\_drink | 2467 | 24.67% |
| Overweight | 982 | 9.82% |
| Anxiety | 984 | 9.84% |
| Initial\_days | 1056 | 10.56% |

Before attempting to detect outliers, boxplots were reviewed for each numeric variable to determine on which side of the distribution the search should be performed.

Diagram

Description automatically generated

Based on the results of the boxplots, the following outlier definitions were created:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Outlier Definition** | **Outliers Detected** |
| Lat | z-score > 3 or z-score < -3 | 144 |
| Lng | z-score < -3 | 98 |
| Population | z-score > 3 | 218 |
| Children | z-score > 3 | 303 |
| Income | z-score > 3 | 180 |
| VitD\_levels | z-score > 3 or z-score < -3 | 500 |
| Full\_meals\_eaten | z-score > 3 | 33 |
| VitD\_supp | z-score > 3 | 70 |
| TotalCharge | z-score > 3 | 276 |
| Additional\_charges | z-score > 3 | 0 |

The Lat and Lng outliers were reviewed together because they are almost always reported as pairs. For context, the State variable was also included. Once these outliers were grouped by State, it became clear what caused them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **State** | **Lat (mean)** | **Lng (mean)** | **Lat\_outlier (mean)** | **Lng\_outlier (mean)** |
| AK | 61.879795 | -153.809046 | 0.957143 | 0.914286 |
| HI | 20.859143 | -157.006051 | 1.000000 | 1.000000 |
| PR | 18.267618 | -66.381990 | 1.000000 | 0.000000 |

Alaska, Hawaii, and Puerto Rico are the three State values that account for 100% of the Lat and Lng variables.

When reviewing the Population outliers, there was not as obvious of a pattern. These outliers were reviewed alongside the State, City, and Area variables. These were the first five results.

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **City** | **Area** | **Population** |
| TX | Katy | Rural | 122814 |
| TX | Katy | Rural | 122814 |
| TX | Houston | Suburban | 107700 |
| CA | Pacoima | Urban | 105799 |
| CA | Pacoima | Urban | 105799 |

Two patterns appeared in the results. First, when viewing only these four variables, there were duplicates present. Second, all of the population values returned in the top five rows contained sequential, duplicate digits (e.g., 22 in the first two entries, 77 and 00 in the third entry, and 99 in the last two entries), suggesting they could have been keyed incorrectly. When the first two outliers were inspected further, it seemed that they were most likely a father and daughter living in the same home. This would explain why they appeared as duplicates when only reviewing variables related to location. Digging further into the Population value for these two records, specifically, the non-outlier, rural Texan Population descriptive statistics were reviewed to determine how plausible the values were.

|  |  |
| --- | --- |
| **Statistic** | **Value** |
| Count | 156 |
| Mean | 12233.615385 |
| Standard Deviation | 14789.789818 |
| Minimum | 0 |
| 25th percentile | 731.25 |
| Median | 5243 |
| 75th percentile | 17920.75 |
| Maximum | 53357 |

When comparing these statistics with the value, 122814, it was clear that this was most likely a data entry error. To determine how likely it was that the remaining outliers were data entry errors, each entry was checked for duplicate digits like the first five entries. 88 or approximately 40 percent of the 218 Population outliers also contained at least one set of sequential duplicate digits.

The Children outlier inspection did not reveal anything suspicious. There were more Married and Divorced patients with Children outliers, but it is not inconceivable that a person could have as many as 10 children.

|  |  |
| --- | --- |
| **Marital** | **Children** |
| Married | 76 |
| Divorced | 64 |
| Widowed | 57 |
| Never Married | 54 |
| Separated | 52 |

The Income outliers were reviewed along with the Education and Employment variables for context. The majority of the outliers have at least a High School Diploma and many have had some form of higher education. The five most frequent are shown below.

|  |  |  |
| --- | --- | --- |
| **Education** | **Employment** | **Income Count** |
| Some College, 1 or More Years, No Degree | Full Time | 21 |
| Regular High School Diploma | Full Time | 20 |
| Bachelor’s Degree | Full Time | 16 |
| Associate’s Degree | Full Time | 11 |
| Master’s Degree | Full Time | 9 |

The VitD\_levels, Full\_meals\_eaten, and VitD\_supp outliers were all reviewed together since they are all related to nutrition. When grouped by the outlier indicator values to get the mean of each field, it made more sense.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **VitD\_levels**  **\_outlier** | **Full\_meals\_**  **eaten\_outlier** | **VitD\_supp**  **\_outlier** | **VitD\_levels** | **Full\_meals\_**  **eaten** | **VitD\_Supp** |
| True | True | False | 47.81923 | 5 | 0 |
| True | False | False | 47.02260 | 0.98371 | 0.40937 |
| True | False | True | 47.02253 | 1 | 3 |
| False | False | True | 18.45823 | 0.89062 | 3.10938 |
| False | True | False | 17.48078 | 5.32143 | 0.35714 |
| False | True | True | 17.39101 | 6 | 3 |

In the table above, values related to outliers are coded orange and values related to non-outliers are coded green. Full\_meals\_eaten and VitD\_supp appear to work together to balance out VitD\_levels. This makes sense since both eating and taking Vitamin D supplements should increase Vitamin D levels.

The TotalCharge outliers did not seem to follow much of a pattern. However, they were quite distant from the rest of the values with the median TotalCharge outlier exceeding the Maximum TotalCharge non-outlier by over $3,000.00.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Non-Outliers** |  | **Outliers** |
| Count | 9724 |  | 276 |
| Mean | 5523.364684 |  | 18862.987055 |
| Standard Deviation | 2605.498644 |  | 1052.013576 |
| Minimum | 1256.751699 |  | 16053.462880 |
| 25th Percentile | 3224.649199 |  | 18315.537830 |
| Median | 5526.536036 |  | **18983.908835** |
| 75th Percentile | 7525.450166 |  | 19607.727452 |
| Maximum | **15969.605660** |  | 21524.224210 |

### D2: Justification of Mitigation Methods

To account for missing values in the medical\_data dataset, Univariate Imputation was performed. The distributions for each numeric variable with missing values were first reviewed to determine whether the mean or median would be most appropriate.

Chart, histogram

Description automatically generated

The median was used for both Children and Income, the mean was used for Age and Initial\_days, and the mode was used for Soft\_drink, Overweight, and Anxiety.

To detect outliers, each variable’s z-score was calculated and checked to see if it was above 3 and/or below -3 based on their distributions as recommended by the course text (Larose & Larose, 2019). Then each set of outliers was reviewed as was shown in section D1 to come to the following decisions:

|  |  |  |
| --- | --- | --- |
| **Field(s)** | **Decision** | **Reasoning** |
| Lat, Lng | Exclude records where State is ‘PR’ | The Scenario indicates the data are about readmissions to a hospital chain with patients in almost every state in the United States. Puerto Rico is not a US State, but an unincorporated US territory (Reichard, 2022). |
| Population | Exclude all outliers | A large number of the outliers are likely typographical errors as was indicated by the frequency of duplicate sequential digits. |
| Income | Keep all outliers | While it is less common to have an income over $200K, it does happen (DQYDJ, 2022). |
| VitD\_levels, Full\_meals\_eaten, VitD\_supp | Keep all outliers | VitD\_levels are within the normal range (Icahn School of Medicine at Mount Sinai, 2022). Full\_meals\_eaten measures meals during the entire stay and patients can have more than three meals per day. |
| TotalCharge | Keep all outliers | The TotalCharge outliers are high, but it is not uncommon to have surgery during an inpatient hospital stay and the costs can be astronomical (Fay, 2022). |
| Item1 – Item8 | Keep all outliers | Because these are Likert-scales, outliers will not be excluded (Re: How do I identify outliers in Likert-scale?, 2016). |

All excluded records were saved to a separate dataset titled ‘medical\_data\_excluded.csv’.

### D3: Summary of the Outcomes

When duplicate row detection was performed, no duplicates were found. However, if any had existed, they would have been removed.

After the columns with missing values were imputed, the pre and post-imputation distributions were compared for the numeric variables to ensure their symmetry remained generally the same.

Chart

Description automatically generated

There are noticeable spikes where the imputations took place. However, the symmetry of each distribution has remained unchanged. The Children and Income distributions remained positively skewed, the Age distribution is no longer uniform, but it is symmetrical, and the Initial\_days distribution no longer appears bimodal, but the only change occurred in the middle of the data, maintaining the symmetry. The data was checked again for missing values, and none were detected.

Based on the decisions that were listed in D2, records where State equaled “PR” or where the population value was an outlier were excluded.

Chart, histogram

Description automatically generated

The Lat variable no longer contains values in the 20 bin and the Lng variable contains fewer values in the -80 bin as a result of the removal of records where State was “PR”. The Population variable no longer contains values above the 50000 bin as a result of removing the Population outliers. The differences are very minor, but that is not necessarily a bad thing. As was shown during the outlier investigation, the outliers that still exist in the data do not appear to be errors but features of the dataset.

### D4: Mitigation Code

Note: To ensure the code below runs free of error, portions are included which are not relevant to anomaly detection. Those sections are grayed out for clarity.

For the full data cleaning script, see the attached file, “task1.py”.

**import** os  
**import** sys  
**import** logging  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
**from** datetime **import** datetime  
**from** scipy **import** stats  
  
FILE\_PATH = **r'C:\Users\user\OneDrive\Documents\Education\Western Govenors University\MS - Data Analytics\Data '** \  
 **r'Cleaning\medical\_raw\_data.csv'**OUTPUT\_PATH = **r'..\output'**DTYPES = {  
 **'CaseOrder'**: int, **'Customer\_id'**: str, **'Interaction'**: str, **'UID'**: str, **'City'**: str, **'State'**: str, **'County'**: str,  
 **'Zip'**: str, **'Lat'**: float, **'Lng'**: float, **'Population'**: int, **'Area'**: str, **'Timezone'**: str, **'Job'**: str,  
 **'Children'**: float, **'Age'**: float, **'Education'**: str, **'Employment'**: str, **'Income'**: float, **'Marital'**: str,  
 **'Gender'**: str, **'ReAdmis'**: str, **'VitD\_levels'**: float, **'Doc\_visits'**: int, **'Full\_meals\_eaten'**: int, **'VitD\_supp'**: int,  
 **'Soft\_drink'**: str, **'Initial\_admin'**: str, **'HighBlood'**: str, **'Stroke'**: str, **'Complication\_risk'**: str,  
 **'Overweight'**: float, **'Arthritis'**: str, **'Diabetes'**: str, **'Hyperlipidemia'**: str, **'BackPain'**: str, **'Anxiety'**: float,  
 **'Allergic\_rhinitis'**: str, **'Asthma'**: str, **'Services'**: str, **'Initial\_days'**: float, **'TotalCharge'**: float,  
 **'Additional\_charges'**: float, **'Item1'**: int, **'Item2'**: int, **'Item3'**: int, **'Item4'**: int, **'Item5'**: int, **'Item6'**: int,  
 **'Item7'**: int, **'Item8'**: int  
}  
  
  
**def** configure\_logging(output\_path: str):  
 *"""Configures logging for the program. Output is directed to both the console and a log file.* **:param** *output\_path: The path to which the log file should be stored.* **:return***: None  
 """* logging.basicConfig(  
 level=logging.INFO,  
 format=**"%(asctime)s [%(levelname)s] %(message)s"**,  
 handlers=[  
 logging.FileHandler(output\_path),  
 logging.StreamHandler(sys.stdout)  
 ]  
 )  
  
  
**def** load\_data(file\_path: str) -> pd.DataFrame:  
 *"""Loads a CSV file into a pandas DataFrame* **:param** *file\_path: a str representing the path at which the CSV file to be read can be located* **:return***: a pandas DataFrame object containing the contents of the CSV located at the given file path  
 """* **return** pd.read\_csv(file\_path, index\_col=**'Unnamed: 0'**, dtype=DTYPES, na\_values=pd.NA)  
  
  
**def** deduplicate(df: pd.DataFrame) -> pd.DataFrame:  
 *"""Inspects a pandas DataFrame for duplicate rows and removes them if necessary* **:param** *df: a pandas DataFrame which should be inspected for duplicate rows* **:return***: df with all duplicate rows removed, if any existed  
 """* n\_dups = df[df.duplicated()].shape[0]  
 **if** n\_dups > 0:  
 logging.info(**f'{**n\_dups**} duplicates found.\nDropping duplicate records.'**)  
 df.drop\_duplicates(inplace=**True**)  
 deduplicate(df)  
 **else**:  
 logging.info(**'No duplicates found.'**)  
 **return** df  
  
  
**def** impute\_missing(df: pd.DataFrame, methods: dict) -> pd.DataFrame:  
 *"""Imputes missing values in a pandas DataFrame according to the provided map* **:param** *df: a pandas DataFrame with values to be imputed* **:param** *methods: a dict with the structure {column name: imputation method}. Acceptable imputation methods include  
 'mean', 'median', and 'mode'.* **:return***: pandas DataFrame with imputed values  
 """* **for** col, method **in** methods.items():  
 logging.info(**f'Imputing {**col**} with the {**method**}'**)  
 **if** method == **'mean'**:  
 df[**f'{**col**}\_imputed'**] = df[col].isna()  
 df[col].fillna(df[col].mean(), inplace=**True**)  
 **elif** method == **'median'**:  
 df[**f'{**col**}\_imputed'**] = df[col].isna()  
 df[col].fillna(df[col].median(), inplace=**True**)  
 **elif** method == **'mode'**:  
 df[**f'{**col**}\_imputed'**] = df[col].isna()  
 df[col].fillna(df[col].mode()[0], inplace=**True**)  
 **else**:  
 **raise** NotImplementedError(**f'{**method**} is not an implemented imputation method.'**)  
 **return** df  
  
  
**def** detect\_outliers(df: pd.DataFrame, query: dict) -> pd.DataFrame:  
 *"""Inspects each numeric column of a pandas DataFrame for outliers* **:param** *df: a pandas DataFrame which should be inspected for outliers* **:param** *query: a dict of search strings to be used to find outliers* **:return***: df with z-score and outlier columns added for each numeric column  
 """  
 # Add z-score and outlier columns* **for** col, query\_string **in** query.items():  
 df[**f'{**col**}\_zscore'**] = stats.zscore(df[col])  
 df[**f'{**col**}\_outlier'**] = df.index.isin(df.query(query\_string).index)  
 outliers = df[**f'{**col**}\_outlier'**].sum()  
 **if** outliers > 0:  
 logging.info(**f'{**col**} contains {**outliers**} outliers'**)  
 **return** df  
  
  
**def** float\_to\_yn(x: float) -> str:  
 *"""Used to transform yes/no fields read in as floats* **:param** *x: 1. or 0.* **:return***: 'Yes' or 'No'  
 """* **if** x == 1.:  
 **return 'Yes'  
 elif** x == 0.:  
 **return 'No'  
  
  
def** main():  
 date\_time = **f'{**datetime.now().strftime(**"%Y%m%d%H%M%S%f"**)**}'** log\_dir = os.path.join(OUTPUT\_PATH, date\_time)  
 log\_path = os.path.join(log\_dir, **f'{**date\_time**}.log'**)  
 os.makedirs(OUTPUT\_PATH, exist\_ok=**True**)  
 os.makedirs(log\_dir)  
 configure\_logging(log\_path)  
  
 filename = \_\_file\_\_.split(**'\\'**)[-1]  
 logging.info(**f'{**5 \* **"\*"} Running {**filename**} {**5 \* **"\*"}'**)  
  
 logging.info(**f'Loading medical\_data from {**FILE\_PATH**}'**)  
 medical\_data = load\_data(FILE\_PATH)  
  
 logging.info(**f'{**5 \* **"\*"} Detecting duplicate rows {**5 \* **"\*"}'**)  
 deduplicate(medical\_data)  
  
 logging.info(**'Plotting quantitative variables with missing values...'**)  
 fields = [[**'Children'**, **'Age'**], [**'Income'**, **'Initial\_days'**]]  
 plt.close()  
 fig, ax = plt.subplots(2, 2)  
 fig.set\_tight\_layout(**True**)  
 fig.suptitle(**'Numeric variables with missing values'**)  
 **for** r, row **in** enumerate(fields):  
 **for** c, col **in** enumerate(row):  
 ax[r, c].hist(medical\_data[col])  
 ax[r, c].set\_title(col)  
 plt.savefig(os.path.join(log\_dir, **'initial\_dist.png'**))  
  
 logging.info(**f'{**5 \* **"\*"} Imputing missing values {**5 \* **"\*"}'**)  
 methods = {  
 **'Children'**: **'median'**,  
 **'Age'**: **'mean'**,  
 **'Income'**: **'median'**,  
 **'Soft\_drink'**: **'mode'**,  
 **'Overweight'**: **'mode'**,  
 **'Anxiety'**: **'mode'**,  
 **'Initial\_days'**: **'median'** }  
 medical\_data\_imputed = impute\_missing(medical\_data.copy(), methods)  
  
 logging.info(**'Verifying distributions pre and post imputation...'**)  
 fields = [**'Children'**, **'Age'**, **'Income'**, **'Initial\_days'**]  
 plt.close()  
 fig, ax = plt.subplots(4, 2, figsize=(6.4, 9))  
 fig.set\_tight\_layout(**True**)  
 fig.suptitle(**'Pre/Post imputation distibutions'**)  
 **for** f, field **in** enumerate(fields):  
 ax[f, 0].hist(medical\_data[field])  
 ax[f, 0].set\_title(**f'{**field**} pre'**)  
 ax[f, 1].hist(medical\_data\_imputed[field])  
 ax[f, 1].set\_title(**f'{**field**} post'**)  
 plt.savefig(os.path.join(log\_dir, **'dist\_compare.png'**))  
  
 *# Align dtypes with data dictionary  
 # performing prior to imputation threw errors* int\_cols = [**'Children'**, **'Age'**]  
 **for** col **in** int\_cols:  
 medical\_data\_imputed[col] = medical\_data\_imputed[col].astype(int)  
 yn\_cols = [**'Overweight'**, **'Anxiety'**]  
 **for** col **in** yn\_cols:  
 medical\_data\_imputed[col] = medical\_data\_imputed[col].apply(float\_to\_yn, convert\_dtype=**True**)  
  
 logging.info(**f'{**5 \* **"\*"} Detecting outliers {**5 \* **"\*"}'**)  
 outlier\_query = {  
 **'Lat'**: **'Lat\_zscore > 3 | Lat\_zscore < -3'**,  
 **'Lng'**: **'Lng\_zscore < -3'**,  
 **'Population'**: **'Population\_zscore > 3'**,  
 **'Children'**: **'Children\_zscore > 3'**,  
 **'Income'**: **'Income\_zscore > 3'**,  
 **'VitD\_levels'**: **'VitD\_levels\_zscore > 3 | VitD\_levels\_zscore < -3'**,  
 **'Full\_meals\_eaten'**: **'Full\_meals\_eaten\_zscore > 3'**,  
 **'VitD\_supp'**: **'VitD\_supp\_zscore > 3'**,  
 **'TotalCharge'**: **'TotalCharge\_zscore > 3'**,  
 **'Additional\_charges'**: **'Additional\_charges\_zscore > 3'** }  
 medical\_data\_outliers = detect\_outliers(medical\_data\_imputed.copy(), outlier\_query)  
  
 logging.info(**f'{**5 \* **"\*"} Handling outliers {**5 \* **"\*"}'**)  
 logging.info(**'Excluding records where State = "PR"'**)  
 logging.info(**'Excluding Population outliers'**)  
 logging.info(**'Keeping all other outliers'**)  
 medical\_data\_excluded = medical\_data\_outliers.query(**'State == "PR" | Population\_outlier'**)  
 medical\_data\_clean = medical\_data\_outliers.query(**'State != "PR" & not Population\_outlier'**)  
  
 logging.info(**'Reviewing distributions before and after handling outliers...'**)  
 fields = [**'Lat'**, **'Lng'**, **'Population'**]  
 plt.close()  
 fig, ax = plt.subplots(3, 2, figsize=(8, 9))  
 fig.suptitle(**'medical\_data distributions'**)  
 fig.set\_tight\_layout(**True**)  
 **for** r, field **in** enumerate(fields):  
 ax[r, 0].hist(medical\_data\_outliers[field])  
 ax[r, 0].set\_title(field + **' before handling outliers'**)  
 ax[r, 1].hist(medical\_data\_clean[field])  
 ax[r, 1].set\_title(field + **' after handling outliers'**)  
 plt.savefig(os.path.join(log\_dir, **'post\_outlier\_dist.png'**))  
  
 logging.info(**f'{**5 \* **"\*"} Run Complete {**5 \* **"\*"}'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

### D5: Clean Data

The cleaned data can be found in the attachments as “medical\_data\_clean.csv”.

### D6: Limitations

During the data cleaning process, decisions had to be made about how to handle specific situations which could introduce limitations to the dataset. First, during the duplicate record detection process, only exact matches were inspected. Because of this, any inconsistencies in how free-text fields are typed could cause otherwise duplicative records to be ignored. The second primary limitation was introduced during univariate imputation. As a result of imputing a single value for each variable with missing values, there was some distortion to their distributions. This was visible in the pre/post imputation plots. The distribution symmetries were unchanged, but the distributions themselves were distorted. The third primary limitation was introduced during the handling of outliers. As a result of excluding some records from the dataset, the sample size has been reduced by 256 records.

### D7:Impact of the Limitations

Several challenges could complicate analysis as a result of the limitations that were introduced. First, if there were any duplicate records in the dataset that were not technically exact matches, they have not been removed. The existence of duplicate records could distort analytic findings. The second challenge is with the distribution distortion that was introduced as a result of the univariate imputation. The third challenge is the presence of outliers. Because so many of the outliers in the original dataset could have represented plausible entries, they were not excluded. This will add a layer of complexity to analysis because outliers often prevent data from being normally distributed. The distributions will need to be accounted for when selecting statistical methods to ensure no assumptions are violated.

## E: Principal Component Analysis (PCA)

### E1: Principal Components

The variables used in the Principal Component Analysis include Lat, Lng, Population, Children, Age, Income, VitD\_levels, Doc\_visits, Full\_meals\_eaten, VitD\_supp, Initial\_days, TotalCharge, Additional\_charges, Item1, Item2, Item3, Item4, Item5, Item6, Item7, and Item8.

Table

Description automatically generated with medium confidence

### E2: Criteria Used

The principal components which should be retained include PC1 through PC16. This is because, when referencing the cumulative explained variance ratio, at least the first 16 components are needed to maintain a minimum of 90% of the explained variance.

Chart, line chart

Description automatically generated

### E3: Benefits

Principal Component Analysis is a very powerful tool. By transforming the axes to better align with the data, more complete features, called Principal Components, are produced. These components take the portions of any duplicative information in the dataset and lump them into components that better explain why the data varies. Because the new components more compactly represent the variation in the data, fewer components are needed than variables to properly model those variations. This leads to using fewer features in predictive models, which can reduce the chances of overfitting the model to the training data. For the national hospital chain working to reduce their readmission rate, this means more accurately identifying what predicts readmissions. It also means the decisions they make based on that information will better translate to their patient population because the model will be less likely to overfit the sample population.

# ****Part IV. Supporting Documents****

## F: Video

The Panopto recording of the python code being executed can be found at this address: <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=26f68185-01b8-4c20-bae6-af11000029a2>

## G: Sources for Third-Party Code

No third-party code was utilized.

## H: Sources

DQYDJ. (2022). *Income Percentile Calculator for the United States.* Retrieved from dqydj.com: https://dqydj.com/income-percentile-calculator/

Fay, B. (2022, October 12). *Hospital and Surgery Costs*. Retrieved from Paying for Medical Treatment: https://www.debt.org/medical/hospital-surgery-costs/

Icahn School of Medicine at Mount Sinai. (2022). *25-hydroxy vitamin D test Information*. Retrieved from Mount Sinai - New York: https://www.mountsinai.org/health-library/tests/25-hydroxy-vitamin-d-test

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

*Re: How do I identify outliers in Likert-scale?* (2016). Retrieved from researchgate.com: https://www.researchgate.net/post/How\_do\_I\_identify\_outliers\_in\_Likert-scale/5852aeffed99e1d9613d5a91/citation/download.

Reichard, R. (2022). *Why Isn't Puerto Rico a State?* Retrieved from HISTORY: https://www.history.com/news/puerto-rico-statehood