**D209 Data Mining 1**

**Task 1: Classification Analysis**

Petra I Bier

Western Governors University

D209 Data Mining

April 25, 2024

Dr. Festus Elleh, PhD

**D209 Data Mining 1**

**Task 1: Classification Analysis**

**Part I: Question**

A :

A1: Question

Can the K-Nearest neighbor classification method be used to predict factors that influence the probability that a patient will be readmitted to the hospital within 30 days of their initial hospital stay?

A2: Goal

Medical costs to patients and insurance companies are rising. A thorough data analysis to find patterns and trends to reduce costs and prevent readmissions can proactively benefit all parties. Preventative interventions could be developed based on the results of the analysis, which would reduce costs for insurance companies and improve patient outcomes.

This analysis aims to determine which factors can be used to predict patient readmission. This information can then be used to develop preventative programs to help lower the readmission rate.

**Part II Method**

B :

B1: Prediction Method

The method chosen for this analysis is K- nearest neighbor (KNN). It is a classification system used when predicting a categorical outcome, which is valid for the variable of ReAdmis. This dependent variable only has the two categories of "Yes" and "No." The expected outcome of this model is that there will be a classification accuracy rate of greater than 90% and a sensitivity of 90% or greater.

KNN works on the idea that objects that are close together are similar and attempts to classify new data based on these similarity relationships. The model measures the distance between points to determine similarity. The new data is then classified according to its distance to the nearest neighbors. Different distance metrics can be used between points, including Euclidean and Manhattan. This analysis will use the Euclidian distance for model classification.

B2: Method Assumption

One assumption of the KNN algorithm is that similar points are closer together. The distance between these objects can then be measured, which allows for the classification of new data points based on the measured closeness to the nearest labeled data points (Harrison, 2018).

B3: Packages and Libraries

|  |  |
| --- | --- |
| **Packages and Libraries** | **Usage in analysis** |
| pandas | Import data into DataFrame, One hot encoding |
| numpy | Objects are arrays for calculations |
| matplotlib.pyplot | Visualizations |
| seaborn | Visualizations |
| sklearn.linear\_model import Lasso | Lasso regression for feature selection |
| sklearn.model\_selection import train\_test\_split | Split the data into training and testing sets |
| sklearn.preprocessing import StandardScaler | Variable standardization |
| sklearn.feature\_selection import SelectKBest, f\_classif | Feature selection |
| sklearn.neighbors import KNeighborsClassifier | KNN algorithm |
| sklearn.model\_selection import GridSearchCV | Hyperparameter testing |
| sklearn.model\_selection import KFold | Hyperparameter testing |
| sklearn import metrics | View metric results |
| sklearn.metrics import classification\_report | View results of model |
| from sklearn.metrics import confusion\_matrix | Create confusion matrix from results |
| scipy import stats | Outlier winsorization |
| scipy.stats.mstats import winsorize | Outlier winsorization |

**Part III Preparation**

C :

C1: Preprocessing

When creating a KNN model, it is necessary to preprocess the data. Preprocessing tasks can include cleaning and ensuring no missing or null values. Data wrangling tasks can also be part of the preprocessing. These tasks ensure an accurate, well-fitted model. For this KNN model, two preprocessing tasks took place in the data wrangling phase. These included hot encoding and scaling the data.

1. One hot encoding is used to change categorical nominal data into a numeric value that can be used within the mode. If the categories are not ordered, then a mistake can be made by giving these categories a numeric value. The model will then assume that there is a relationship between the variables. The way to avoid this mistake is to create dummy variables. Each category is given a new column, and when it is present, it is encoded with a one, and when not present, it is given a zero. Unlike regression models, which work under the assumption of no multicollinearity and remove a dummy variable to avoid multicollinearity, KNN is based on relationships, so all variables are retained (Pramoditha, 2021).

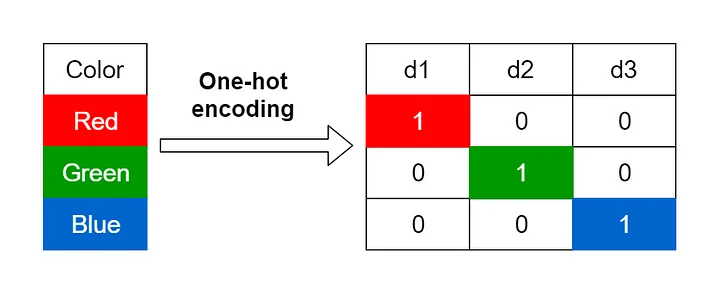


Figure 1: (Encoding Categorical Variables, 2021)

For this analysis, variables that were categorical and had more than two categories, the variable was one hot encoded. This was done using get\_dummies() from the pandas library. The drop\_first attribute has been set to False to retain all new variables.

df = pd.get\_dummies(df, columns = categorical\_cols, dtype = int, drop\_first = False)

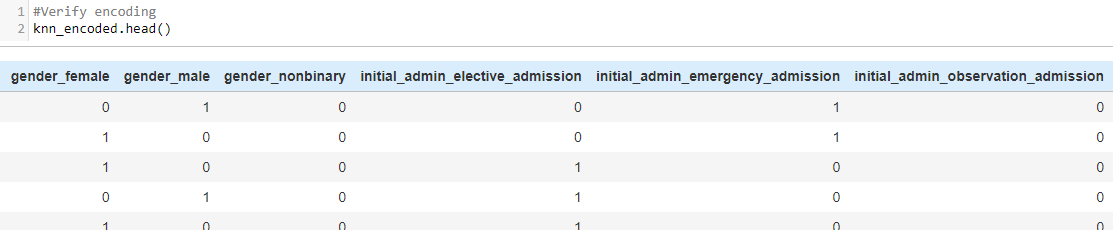


Figure 2: One Hot Encoded Variables

2. Another preprocessing task that was performed on the data was scaling. This allows for variables on different measurement scales to compare to one another using the same comparison scale. The variable Totoalcosts is in dollars, while Initial\_ days is measured in days, making a comparison between the two difficult. An algorithm such as KNN that measures distances between data points will give unequal weight to these variables, unless placed on the same scale for a more equitable measurement between points.

The data was scaled using StandardScaler() from sklearn.preprocessing library. To prevent biases within the data, also known as data leakage, the scaling was performed after the data was split into training and test data sets (Weiran, 2021). The target variable, y, is not scaled, as it must remain in a binary format for the model.

scale = StandardScaler()

X\_train\_scaled = pd.DataFrame(scale.fit\_transform(X\_train), columns=X\_train.columns)

X\_test\_scaled = pd.DataFrame(scale.transform(X\_test), columns=X\_test.columns)

The difference in the numerical values can be seen in a comparison before and after the scaling.

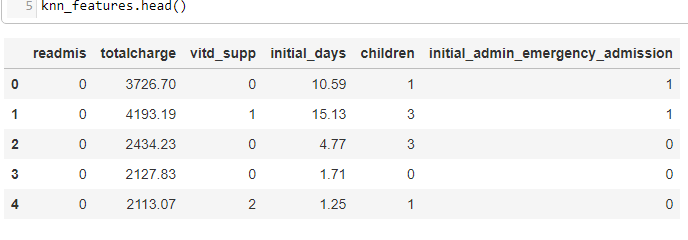


Figure 3: Before Scaling

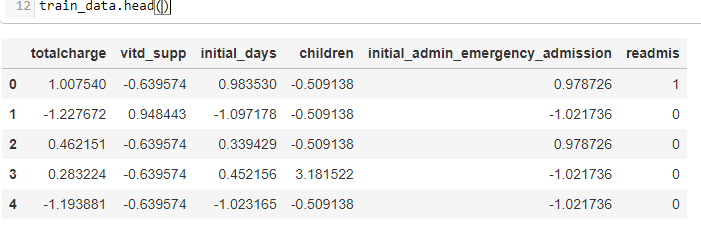


Figure 4: After StandardScaler()

C2: Variable Description

When creating a KNN model, all the data must be transformed into a numeric data type, even if the target variable is bivariate categorical. The data set was examined and passed down through several cleaning steps. The chart below shows the data type of each variable in the initial data to determine if it would be selected for the final model. The final variables chosen for the analysis are highlighted in blue.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Variable Type | Data Type | Selected final analysis |
| Population | Numeric | Discrete | No |
| Area | Categorical | Nominal | No |
| Timezone | Categorical | Nominal | No |
| Job | Categorical | Nominal | No |
| Children | Numeric | Discrete | Yes |
| Age | Numeric | Continuous | No |
| Education | Categorical | Nominal | No |
| Employment | Categorical | Nominal | No |
| Income | Numeric | Continuous | No |
| Marital | Categorical | Nominal | No |
| Gender | Categorical | Nominal | No |
| ReAdmis | Categorical | Nominal / Bivariate | Target/ Dependent Variable |
| VitD\_levels | Numeric | Continuous | No |
| Doc\_visits | Numeric | Discrete | No |
| Full\_meals\_eaten | Numeric | Discrete | No |
| VitD\_supp | Numeric | Discrete | No |
| Soft\_drink | Categorical | Nominal | No |
| Initial\_admin | Categorical | Nominal | Yes – Dummy variable  initial\_admin\_emergency\_admission |
| HighBlood | Categorical | Nominal / Bivariate | No |
| Stroke | Categorical | Nominal / Bivariate | No |
| Complication\_risk | Categorical | Nominal (can be Ordered) | No |
| Overweight | Categorical | Nominal / Bivariate | No |
| Arthritis | Categorical | Nominal / Bivariate | No |
| Diabetes | Categorical | Nominal / Bivariate | No |
| Hyperlipidemia | Categorical | Nominal / Bivariate | No |
| BackPain | Categorical | Nominal / Bivariate | No |
| Anxiety | Categorical | Nominal / Bivariate | No |
| Allergic\_rhinitis | Categorical | Nominal / Bivariate | No |
| Reflux\_esophagitis | Categorical | Nominal / Bivariate | No |
| Asthma | Categorical | Nominal / Bivariate | No |
| Services | Categorical | Nominal | Yes – Dummy variables  services\_ct\_scan  services\_intravenous |
| Initial\_days | Numeric | Continuous | Yes |
| TotalCharge | Numeric | Continuous | Yes |
| Additional\_charges | Numeric | Continuous | No |

C3: Analysis Preparation Steps

Often, the data for an analysis needs to be prepared through cleaning, exploration, and wrangling. These preprocessing steps are vital to ensure clean data that can create an accurate model.

The steps for this analysis included:

* + Duplicate removal
  + finding and treating missing values
  + Determining variables to be removed
  + identify outliers
  + Categorical re-expression
  + One- Hot encoding
  + Scaling of numerical data

1. Import and examine the data set- This step allows for familiarization with the data that will be worked with

med\_df = pd.read\_csv('medical\_clean.csv')

med\_df.info()

1. Find and remove duplicates – This step finds and removes data that could skew the results if left in and counted more than once.

print(med\_df.duplicated().value\_counts())

print(med\_df.duplicated().sum())



1. Find Null values – This step finds values not within the data set. Many functions will not run if missing values are in the data set. Missing values are removed, but they are not the same a zero value, which should remain in the data.

med\_df.isnull().sum()

#Visualization of missing values- Heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(med\_df.isnull(), cbar=False, cmap='viridis', yticklabels=False, xticklabels=med\_df.columns)

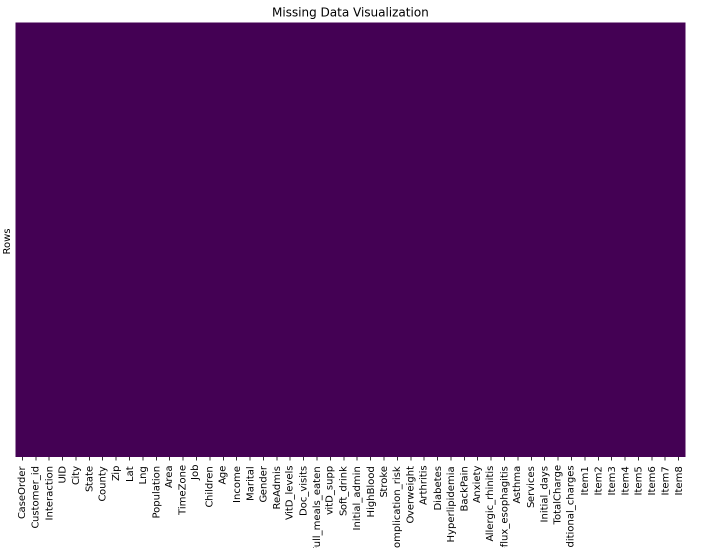
plt.xticks(rotation=90)

plt.title('Missing Data Visualization')

plt.xlabel('Columns')

plt.ylabel('Rows')

plt.show()



1. Remove null values—Null values are removed in the cleaning process as many functions can not be correctly calculated if the information is not available.

No null values were found. The data set was then renamed as a checkpoint in which duplicates and null values were addressed

clean\_df = med\_df.copy()

1. Remove columns with high cardinality – This step removes columns that will not be used due to being an identifier or the volume of variables being unable to be used to build a desired prediction model.

clean\_df.describe()

# Clean the data set columns using created function based on thresholds (Bold Analytics: Mark Keith, 2024)

#missing\_threshold = 0.95, unique threshold = 0.95, only 1 value in a column --> removes

#Function

def clean\_columns(df, columns =[], missing threshold = 0.95, unique threshold = 0.95, messages = True):

if len(columns) == 0:

columns = df.columns #this lets the columns be blank, and every column will be cleaned

for col in columns:

if col in df.columns:

missing = df[col].isna().sum()

unique = df[col].nunique()

rows = df.shape[0]

if missing / rows >= missing\_threshold:

if messages: print(f"To many missing values with ({missing} out of {rows}, {round((missing / rows) \* 100, 2)}%) for {col}, removed")

df.drop(columns =[col], inplace = True)

# For non-numeric columns, check if there are too many unique values

if not pd.api.types.is\_numeric\_dtype(df[col]) and (unique / rows >= unique threshold):

if messages:

print(f"Too many unique values with ({unique} out of {rows}, {round((unique / rows) \* 100, 2)}%) for {col}, removed")

df.drop(columns=[col], inplace=True)

continue

elif unique == 1:

if messages: print(f"Only one value in ({df[col].unique()[0]} for {col}, removed")

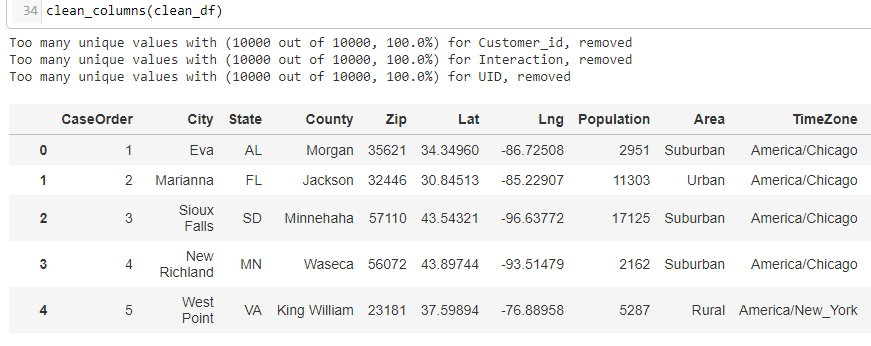
df.drop(columns =[col], inplace = True)

else:

if messages: print(f"The column variable \"{col}\" doesnt exist as spelled in the DataFrame provided")

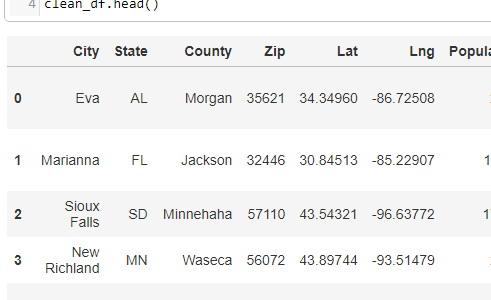
return df

clean\_columns(clean\_df)



# Remove the index column

clean\_df = clean\_df.iloc[:, 1:]



1. Explore summary statistics and univariate visualizations—This step allows for familiarization with the data statistics to determine the distribution and need for dealing with outliers. Many models are sensitive to outliers and must be cleaned to help reduce errors within a model.

#Function for visualizing univariate variables and summary statistics (Bold Analytics: Mark Keith, 2024)

def univariate(df):

stats = []

for col in df.columns:

col\_data = df[col]

dtype = col\_data.dtype

count = col\_data.count()

missing = col\_data.isna().sum()

unique = col\_data.nunique()

mode = col\_data.mode().iloc[0] if not col\_data.mode().empty else None

if pd.api.types.is\_numeric\_dtype(col\_data):

# Compute stats for numeric columns

min\_val = col\_data.min()

q1 = col\_data.quantile(.25)

median = col\_data.median()

q3 = col\_data.quantile(.75)

max\_val = col\_data.max()

mean = col\_data.mean()

std = col\_data.std()

skew = col\_data.skew()

kurt = col\_data.kurt()

stats.append((col, dtype, count, missing, unique, mode, min\_val, q1, median, q3, max\_val, mean, std, skew, kurt))

# Plot histogram for numeric columns

sns.histplot(data=col\_data.dropna(), kde=True)

plt.title(f'Histogram of {col}')

plt.show()

else:

# Stats for non-numeric columns

stats.append((col, dtype, count, missing, unique, mode, "-", "-", "-", "-", "-", "-", "-", "-", "-"))

# Plot countplot for categorical columns

sns.countplot(x=col\_data.dropna(), data=df)

plt.title(f'Count Plot of {col}')

plt.show()

# Create DataFrame from stats list

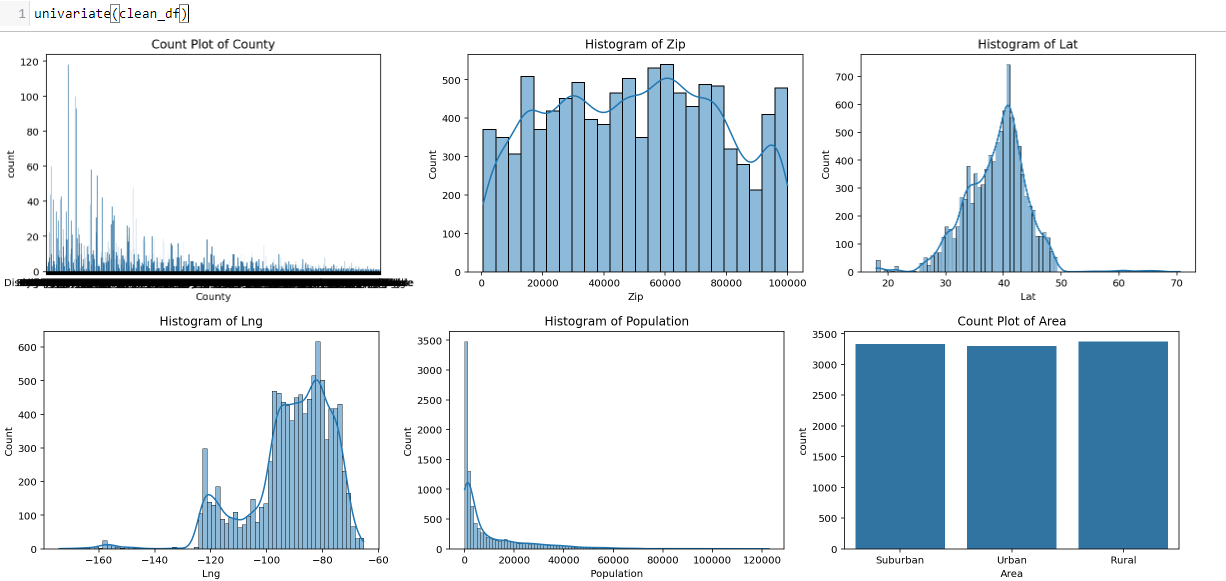
output\_df = pd.DataFrame(stats, columns=["Variable", "Type", "Count", "Missing", "Unique", "Mode",

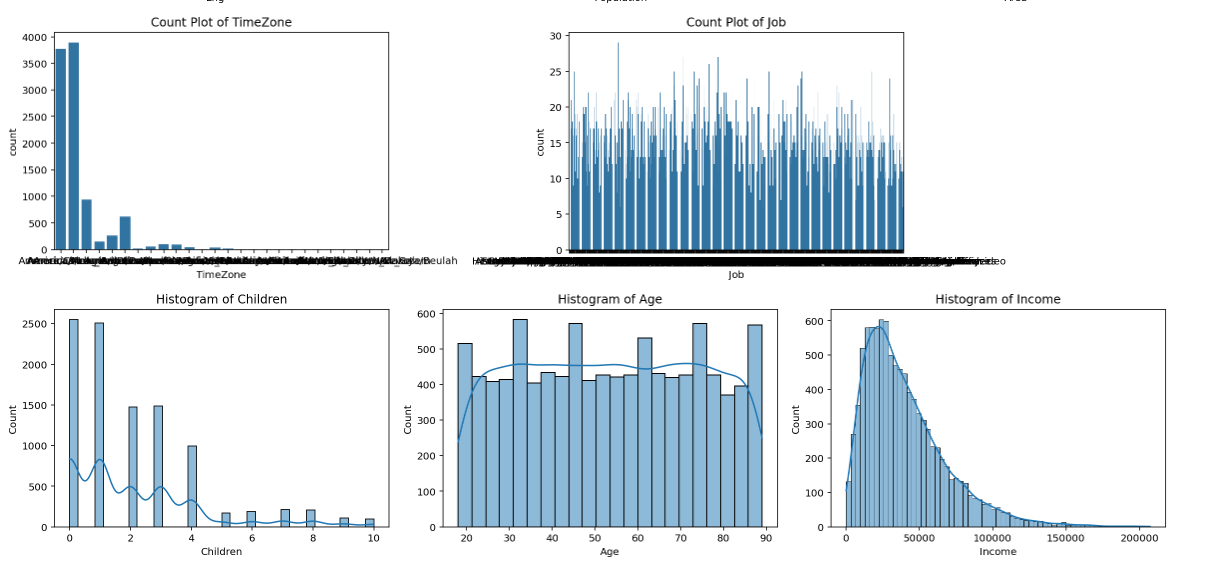
"Min", "Q1", "Median", "Q3", "Max", "Mean", "Std", "Skew", "Kurt"])

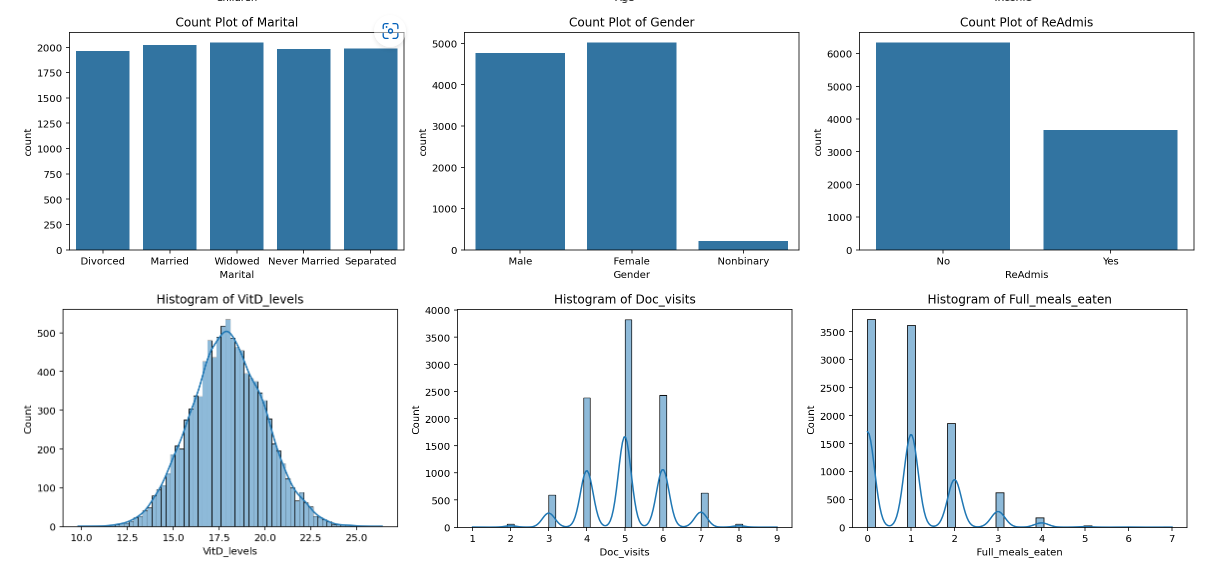
output\_df.set\_index("Variable", inplace=True)

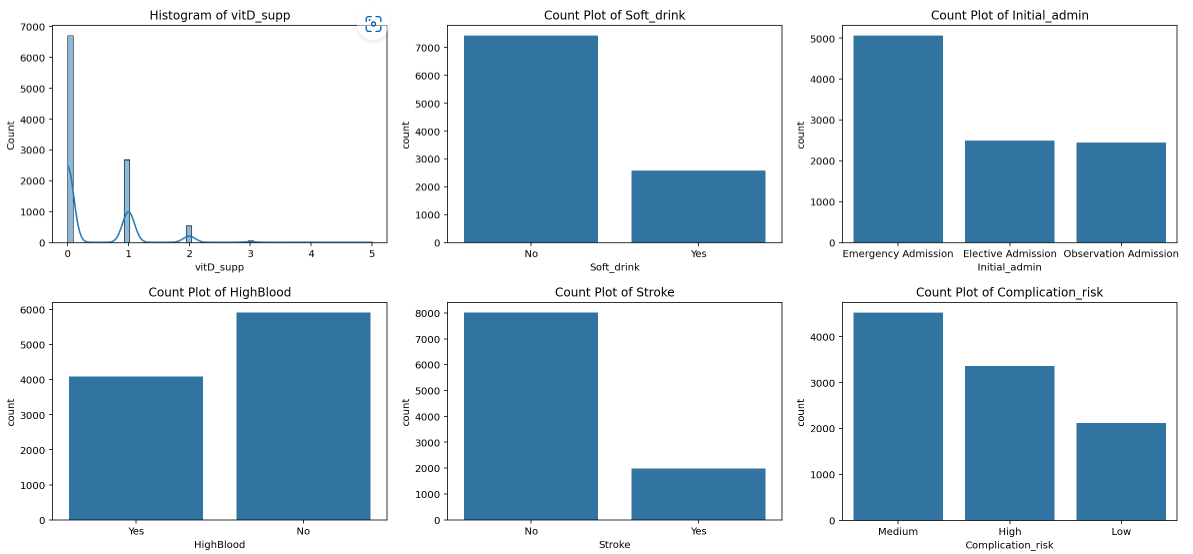
return output\_df

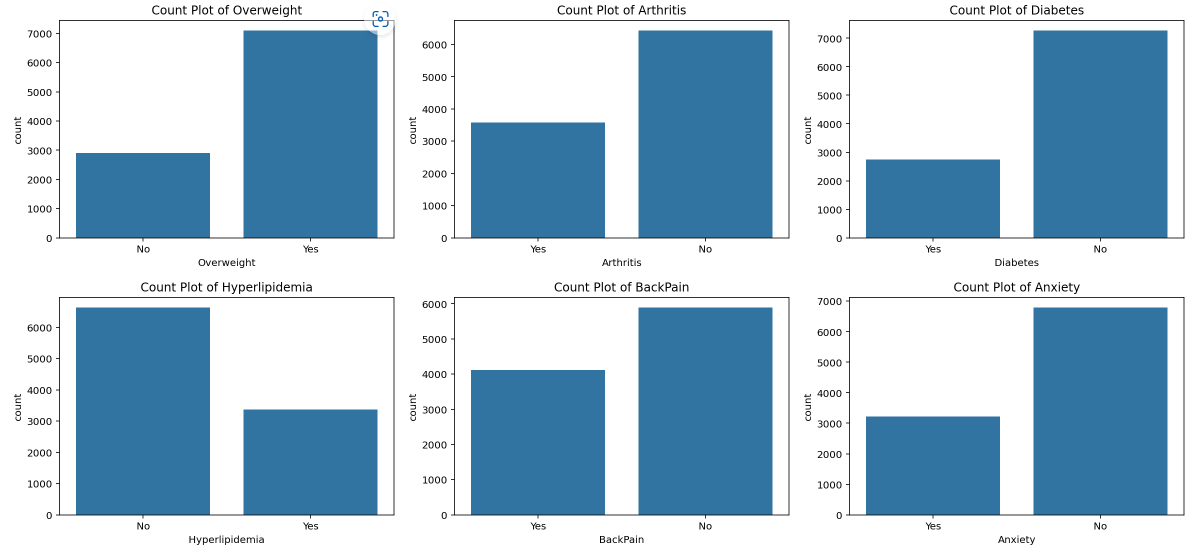
univariate(clean\_df)

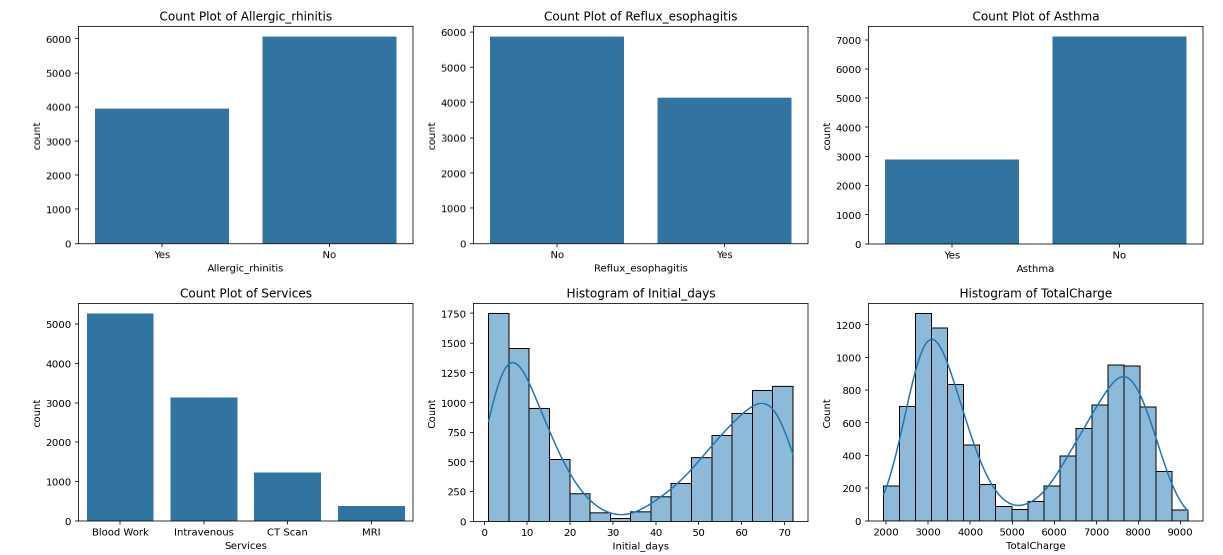


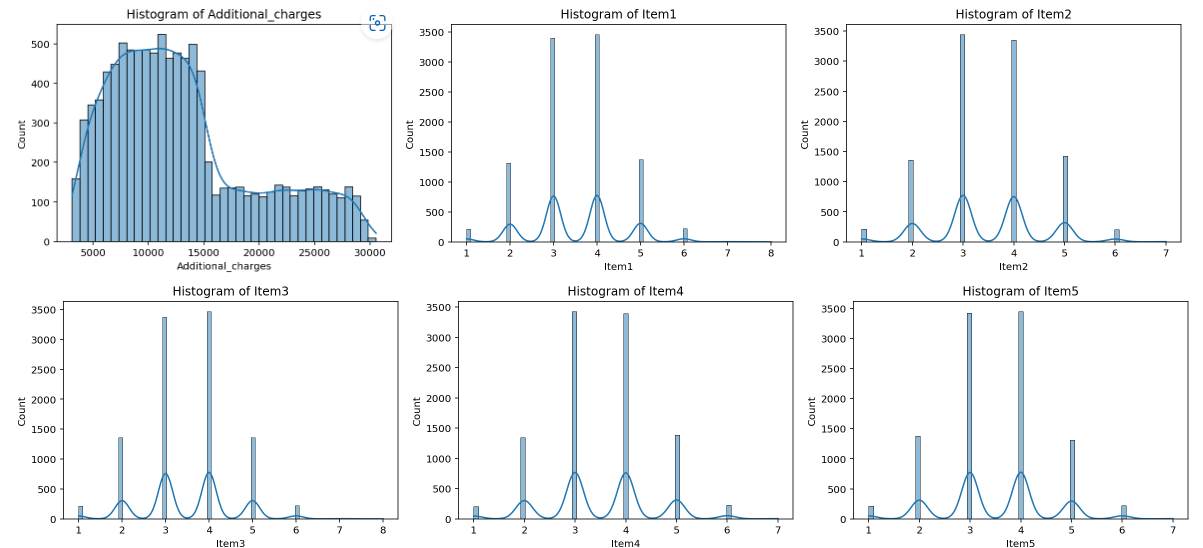


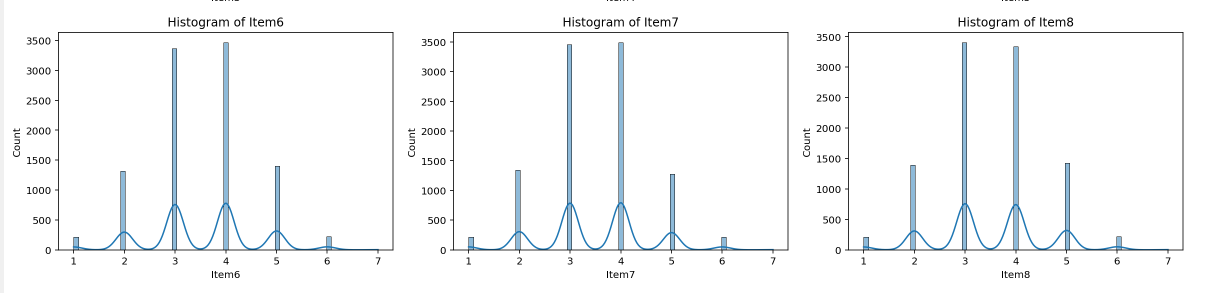


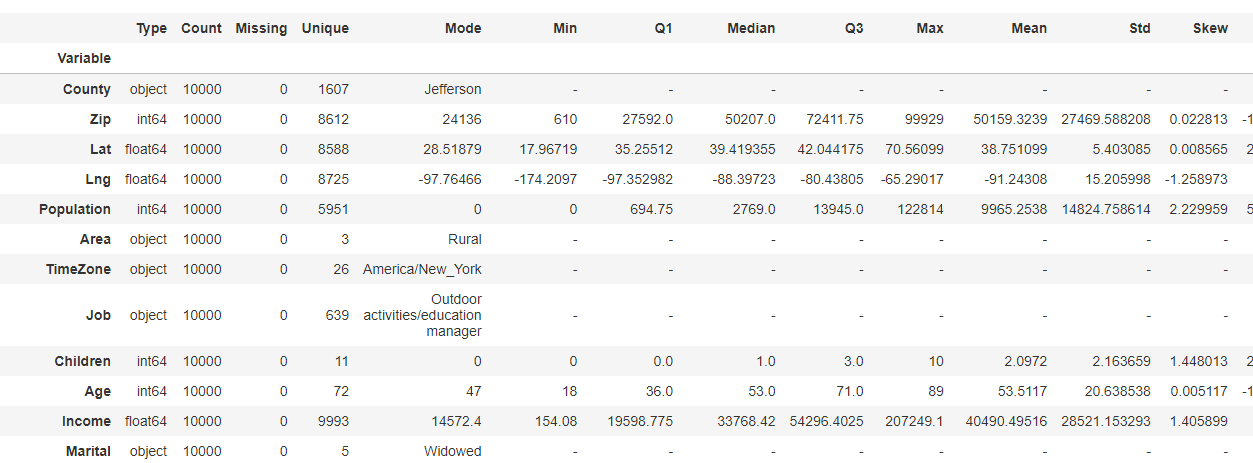












1. Narrow down variables for feature selection—Understanding the subject matter can help reduce the number of features needed for calculations when building models. Not all features will be relevant for every analysis, nor can all features be used. The KNN model is sensitive to having too many variables included within the analysis.

Columns with location data were removed, and all the survey columns were removed because they are subjective measurements.

knn\_data = clean\_df[[

"ReAdmis", "TotalCharge", "Population", "Children", "Age", "Income",

"VitD\_levels", "Doc\_visits", "Full\_meals\_eaten", "vitD\_supp",

"Initial\_days", "Additional\_charges", "Gender", "Initial\_admin",

"Complication\_risk", "HighBlood", "Diabetes", "Hyperlipidemia", "Services"

]].copy()

1. Clean white spaces and round decimals—Cleaning data, such as white spaces and rounding decimals, helps with readability and prevents errors by creating uniformity within the data.

## Remove spaces in category names within columns

# To replace spaces with underscores in specific columns

for col in knn\_data.columns:

if not pd.api.types.is\_numeric\_dtype(knn\_data[col]):

knn\_data[col] = knn\_data[col].str.replace(" ", "\_", regex = False)

##Clean White Space and make all lower case

knn\_data.columns = knn\_data.columns.str.lower().str.strip()

#round values to 2 decimal points

knn\_data = knn\_data.round(2)



1. Detect outliers (Horsch, 2021)—Outliers must be detected and evaluated to determine whether they are expected or bad data. It also needs to be evaluated whether the outliers will be tolerated or need to be treated.

#Function for Winsorization of outliers, using IQR

def winz\_outliers(df):

print("Outlier Analysis Report")

print("=" \* 50) # Print a separator line for visual clarity

for col in df:

if pd.api.types.is\_numeric\_dtype(df[col]):

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

outliers = ((df[col] < (Q1 - 1.5 \* IQR)) | (df[col] > (Q3 + 1.5 \* IQR)))

outlier\_count = outliers.sum()

if outlier\_count > 0:

outer\_fence = 3 \* IQR

outer\_fence\_low = Q1 - outer\_fence

outer\_fence\_up = Q3 + outer\_fence

# Consolidating the print statements for each column

print(f"\nColumn: {col}")

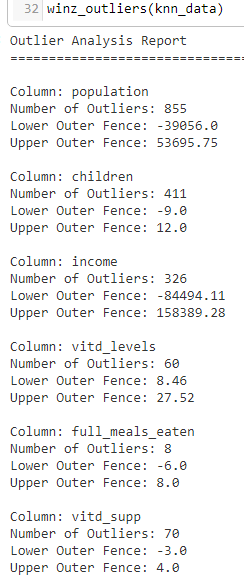
print(f"Number of Outliers: {outlier\_count}")

print(f"Lower Outer Fence: {round(outer\_fence\_low, 2)}")

print(f"Upper Outer Fence: {round(outer\_fence\_up, 2)}")

print("=" \* 50)

winz\_outliers(knn\_data)



# Create a boxplot for each column with outliers

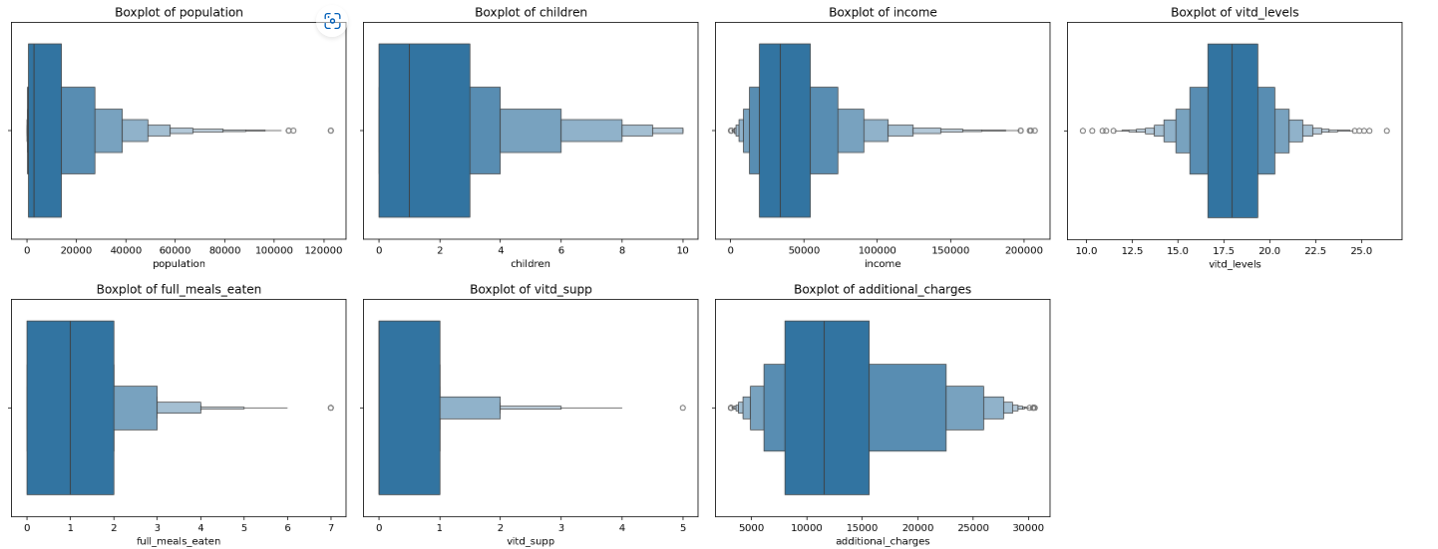
outlier\_shape = knn\_data[["population", "children","income", "vitd\_levels", "full\_meals\_eaten", "vitd\_supp", "additional\_charges"]]

for column in outlier\_shape:

sns.boxenplot(x=knn\_data[column])

plt.title(f'Boxplot of {column}')

plt.show()



1. Treat outliers—Since outliers can change the results in regression and estimates of location, it is necessary to address whether outliers will be treated. The method used for this analysis is winsorization, which works well when a variable's distribution is not a standard bell curve.

Several Outliers were expected and left unchanged. These variables were children, vitd\_supp, and additional\_charges.

## Winsorize the outliers

# Variables to winsorize - Population, income, VitD\_levels, Full\_meals\_eaten

#Note that VitD levels will also need a lower limit calculation

knn\_data['population\_winz'] = winsorize(knn\_data['population'], limits=(0, 0.05))

knn\_data['income\_winz'] = winsorize(knn\_data['income'], limits=(0, 0.05))

knn\_data['vitd\_levels\_winz'] = winsorize(knn\_data['vitd\_levels'], limits=(0.05, 0.01))

knn\_data['full\_meals\_winz'] = winsorize(knn\_data['full\_meals\_eaten'], limits=(0, 0.05))

#Remake data set with the outliers removed

knn\_data = knn\_data.drop(columns=["population", "income", "vitd\_levels", " full\_meals\_eaten"], axis = 1)

1. Variable re-expression and one hot encoding—Bivariate categorical variables, including the target variable of Readmis, were changed to 0/1 values. Categorical variables with more than two categories were hot encoded to dummy variables. No dummy variables were removed since multicollinearity is not a concern when running a KNN analysis.

## Function for changing binary values to 0/1

# and creating dummy var with all variables kept in place

def wrangle\_cat(df):

#handle binary categorical variables

for col in df.columns:

if pd.api.types.is\_string\_dtype(df[col]):

# standardize the text format to lowercase

df[col] = df[col].astype(str).str.strip().str.lower()

# find binary columns with yes/ no or true/false

if df[col].isin(['yes', 'no', 'true', 'false']).all():

mapping = {'yes': 1, 'no': 0, 'true': 1, 'false': 0}

df[col] = df[col].map(mapping)

# one-hot encode the remaining categorical variables

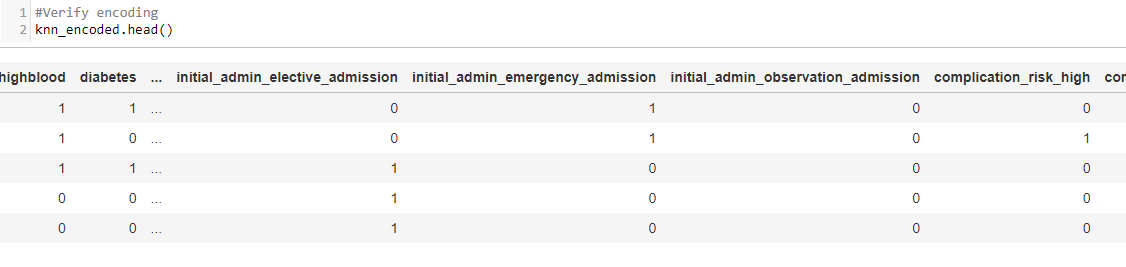
# Ensure to only encode those that have not been converted to numeric in the previous step

categorical\_cols = df.columns[df.dtypes == 'object']

df = pd.get\_dummies(df, columns = categorical\_cols, dtype = int, drop\_first = False)

return df

knn\_encoded = wrangle\_cat(knn\_data)



1. Feature selection -Two methods were used for feature selection: Lasso regression and SelectKBest. The results were combined into a final data set for use in the KNN prediction model.

Lasso feature selection (Agrawal, 2023)

X = knn\_encoded.drop(["readmis"], axis = 1)

y = knn\_encoded["readmis"]

lasso = Lasso(alpha=0.1)

lasso\_coef = np.abs(lasso.fit(X, y).coef\_)

# Use the columns from X for names

lasso\_names = X.columns

# Plot

plt.figure(figsize=(10, 6)) # Optional: Adjusts the figure size for better readability

plt.bar(lasso\_names, lasso\_coef)

plt.xticks(rotation=90)

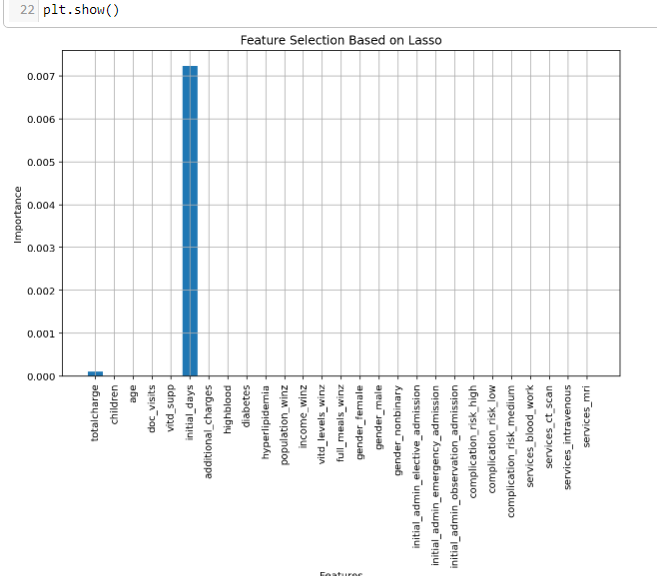
plt.grid()

plt.title("Feature Selection Based on Lasso")

plt.xlabel("Features")

plt.ylabel("Importance")

plt.show()



SelectKBest feature selection (D, 2023) – The p-value assigned for significance was 0.05.

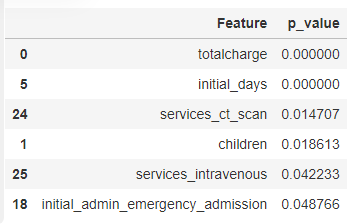
skbest = SelectKBest(score\_func = f\_classif, k ='all') # Adjust 'k' based on feature selection strategy

X\_new = skbest.fit\_transform(X, y)

X\_new.shape

p\_values = pd.DataFrame({"Feature": X.columns, "p\_value":skbest.pvalues\_}).sort\_values("p\_value")

p\_values[p\_values["p\_value"] < .05]



1. Data set for KNN prediction model—The features selected for the KNN model were readmis, totalcharge, initial\_days, children, and initial\_admin\_emergency\_admission. The lasso model did not add features not already found using SelectKBest.

knn\_features = knn\_encoded[["readmis", "totalcharge", "initial\_days","services\_ct\_scan", "services\_intravenous", "children", "initial\_admin\_emergency\_admission"]].copy()

C4: Clean Data

A file of the cleaned data set is attached as a .csv file labeled KNN\_features\_209.csv.

**Part IV**

D :

D1: Split Data

The data was split into training and test data, which were scaled after the split to prevent data leakage. The training data was split and contained 80% of the data, and the test data contained 20% (test\_size = 0.2). The data was also stratified (stratify = y ) to ensure that each category was represented in the training and test data in the same proportion.

X = knn\_features.drop(["readmis"], axis=1)

y = knn\_features["readmis"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y,

test\_size=0.2, # Specifies 20% of the data for testing

random\_state=42, # Ensures reproducibility

stratify=y) # Stratifies the split based on the labels in 'y'

Copies of the split and scaled data set are attached as .csv files labeled:

* Xscale\_train\_209.csv
* Xscale\_test\_209.csv
* Y\_train\_209
* Y\_test\_209

#Export testing and training files

# file path is within jupyter Lab project

X\_train\_scaled.to\_csv("Xscale\_train\_209.csv", index=False)

X\_test\_scaled.to\_csv("Xscale\_test\_209.csv", index = False)

y\_train.to\_csv("Y\_train\_209.csv", index = False)

y\_test.to\_csv("Y\_test\_209.csv", index = False)

D2: Analysis and Calculations

The method used to analyze the data is the K—Nearest Neighbor, which uses supervised learning to classify labeled data based on how similar it is to other records. KNN is frequently used when there is a binary decision output, which makes it a strong choice for the response variable of ReAdmis, which is a Yes or No variable.

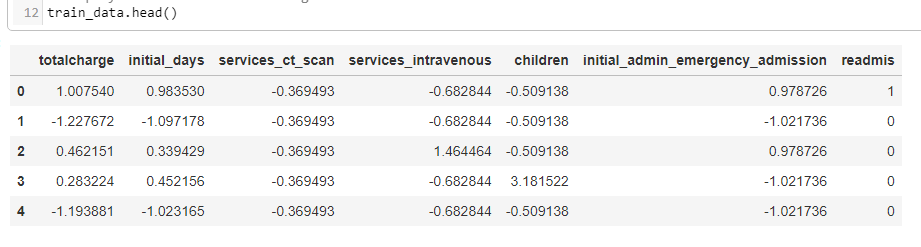
The similarity between the data points can be detected using a distance metric. Records with closer distances to one another are said to be neighbors. Two standard distance metrics are Euclidean and Manhattan. The Euclidean distance measures a straight line directly between two points, like the idea of “as the crow flies”. Manhattan distance is measured along points traveled in a straight horizontal or vertical distance, like traveling on city blocks (Bruce, et al., 2019). The metric used for this analysis was the Euclidean distance.

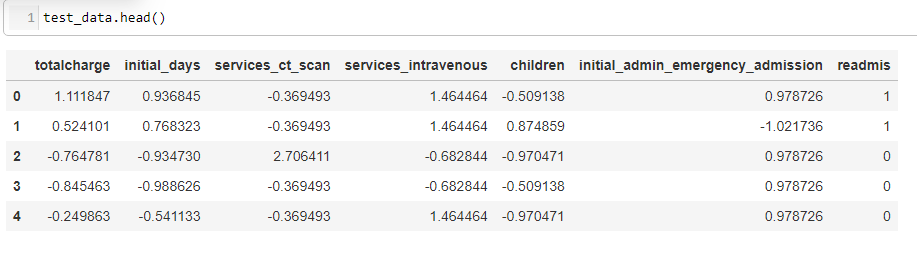
The *k* in the model is how many neighbors are in proximity when making a classification. When there are few neighbors, the model risks being overfitted, and when there are a larger number of *k*, the model may be too generalized. Both result in poor prediction accuracy. The *k* value can be adjusted with hyperparameter tuning and cross-validation methods.

This analysis entailed several steps, including scaling the numeric data, running the K nearest neighbor analysis, and then hyperparameter tuning. No intermediate calculations were performed during the analysis. All code for the following analysis descriptions can be found in a subsequent section of D3.

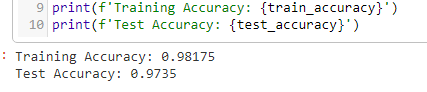
1. The numeric data was standardized after the split to prevent data leakage, which occurs when the model is exposed to information outside the designated training data (Brownlee, 2020). The standardization of the features is solely based on the training data.

When there are large distances between variables due to being on a larger scale, those measurements will dominate. The data was standardized using the scikit-learn method StandardScaler(). This allows numeric data on different measurement scales to be compared with one another (Hale, 2020) without the weight of large distances.





2. The KNN analysis model was trained using the scaled training data set, and the model performance was analyzed using the test data. The initial k value was set at 5, an industry standard for a starting value (Band, 2020), which would be tuned in the following steps.



The results show an accuracy of 98.2% for the training data and 97.4% for test data. Accuracy can be defined as the ratio of correct predictions to the total number of predictions.



It is one measure of how well a model performs, and a confusion matrix addresses precision and recall as model performance metrics were measured after the hyperparameter testing was completed.

3. The model was evaluated to improve the hyperparameters in an attempt to improve accuracy. The first way the k value was assessed was to run the model with a k value of 1 up to 20. These were then compared to one another, and the best k value was taken as that which had the highest accuracy for test and training data. This can be plotted in a graph for easy visualization.



This method of hyperparameter testing showed that the optimal k value is 12



The one problem with this method is that it can be memory intensive, as the model is run for each value of k. It also just uses the same training data, which does not consider the possibility that there may be differences in the data depending on how the data was split.

This can be resolved by folding the training data. This entails splitting the training data into "folds" and then retesting that fold data.

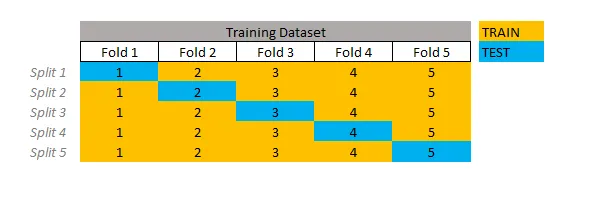
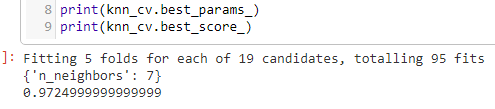
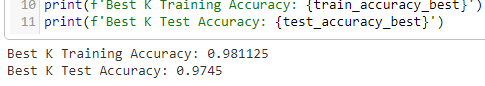


Figure 5: (Santos, 2020)

The scikit\_learn library has a module GridSearchCV() that allows for this and adjusting the hyperparameters. For this tuning, accuracy was used as the metric to achieve the optimal value of the number of k to be used in the KNN analysis. The best k value was determined to be 7.



Once this k value was tuned, the KNN model was re-run with the new value for the final model.



The tuned KNN model has a training accuracy of 98.1% and a test accuracy of 97.5%.

D3: Code

A jupyter notebook file of all the code used in the analysis is attached as PBier\_209\_pt1.ipynb .

1. Standardization of the data set after spit 80% training and 20% test.

# Standardize the values using scale

scale = StandardScaler()

X\_train\_scaled = pd.DataFrame(scale.fit\_transform(X\_train), columns=X\_train.columns)

X\_test\_scaled = pd.DataFrame(scale.transform(X\_test), columns=X\_test.columns)

# Concatenate scaled features with the target variable

train\_data = pd.concat([X\_train\_scaled, y\_train.reset\_index(drop=True)], axis=1)

test\_data = pd.concat([X\_test\_scaled, y\_test.reset\_index(drop=True)], axis=1)

2. KNN Model training and testing. The initial k value was set to 5.

# Initialize and train the KNN classifier on the training data

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train\_scaled, y\_train)

# Evaluate the model on both the training and test datasets

train\_accuracy = knn.score(X\_train\_scaled, y\_train)

test\_accuracy = knn.score(X\_test\_scaled, y\_test)



3. Hyperparameter testing

a. Re-running of same training data using k values 1 through 20 –

##run hyperparameter testing

train\_accuracies = {}

test\_accuracies = {}

neighbors = np.arange(1, 20)

for neighbor in neighbors:

knn = KNeighborsClassifier(n\_neighbors=neighbor)

knn.fit(X\_train\_scaled, y\_train)

train\_accuracies[neighbor] = knn.score(X\_train\_scaled, y\_train)

test\_accuracies[neighbor] = knn.score(X\_test\_scaled, y\_test)

plt.figure(figsize=(8, 6))

plt.title("KNN: Varying Number of Neighbors")

plt.plot(neighbors, train\_accuracies.values(), label="Training Accuracy")

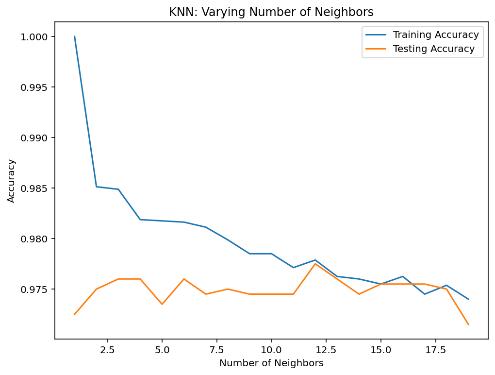
plt.plot(neighbors, test\_accuracies.values(), label="Testing Accuracy")

plt.legend()

plt.xlabel("Number of Neighbors")

plt.ylabel("Accuracy")

plt.show()



# Find the key (neighbor value) with the highest test accuracy

best\_k = max(test\_accuracies, key=test\_accuracies.get)

best\_test\_accuracy = test\_accuracies[best\_k]

best\_train\_accuracy = train\_accuracies[best\_k]

print(f"The best value of k is {best\_k}, with a test accuracy of {best\_test\_accuracy:.2f} and a training accuracy of {best\_train\_accuracy:.2f}")

b. GridSearchCV() hyperparameter testing with k = 7

# Re-run KNN with the best found k value on training data only

knn\_best = KNeighborsClassifier(n\_neighbors= 7)

knn\_best.fit(X\_train\_scaled, y\_train)

y\_pred = knn\_best.predict(X\_test\_scaled)

# Evaluate the model again on both the training and test datasets with the best k

train\_accuracy\_best = knn\_best.score(X\_train\_scaled, y\_train)

test\_accuracy\_best = knn\_best.score(X\_test\_scaled, y\_test)

**Part V**

E :

E1: Accuracy Analysis

The model was then evaluated using several different methods. A confusion matrix was created to determine accuracy, precision, and recall. The ROC curve and the area under the curve (AUC) were also calculated and visualized.

Accuracy is the ratio of correct prediction to the total number of predictions. It was also used as the metric to determine the optimal k value when performing the hyperparameter testing.

A confusion matrix was created to calculate the precision and recall metrics. Precision measures the ability to measure the correctness of a predicted outcome, in this case, a Yes or No value.

Precision = True Positives / (True Positives + False Positives)

Recall is similar, but it concerns how well the model can detect the actual Yes or No cases. This is also known as sensitivity.

Recall = True Positives / (True Positives + False Negatives)

ROC visualizes the tradeoff between sensitivity and specificity since they are inversely related. When the values are plotted against one another, it shows the effectiveness of the model in making a prediction. A graph with a curve higher into the upper left corner is very effective. The AUC is a numeric calculator of the area beneath this curve. The higher the AUC value, the more effective the model.

E2: Results

1. Accuracy—The tuned model's training data accuracy was 98.2%, and the test accuracy was 97.4%. The final tuned model had almost identical accuracy for Training data, at 98.1%, and test data, at 97.5%. The GridSearchCV() method means that the chosen k = 7 is robust since it has gone through cross-validation and is less inclined to over- or underfit the data.

2. Precision and Recall - A confusion matrix allowed for calculating the precision and recall. These metrics allow for a closer view of how well a model performs regarding its ability to detect the desired outcomes, which in this analysis was who would be readmitted within 30 days of admission. Several visualizations were done to increase the comprehensibility of the results.

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

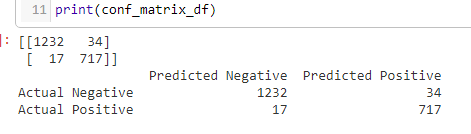
print(conf\_matrix)

# Creating a DataFrame from the confusion matrix with labels

conf\_matrix\_df = pd.DataFrame(conf\_matrix,

index=['Actual Negative', 'Actual Positive'],

columns=['Predicted Negative', 'Predicted Positive'])



# Normalize by the number of instances in each class

conf\_matrix\_norm = conf\_matrix.astype('float') / conf\_matrix.sum(axis=1)[:, np.newaxis]

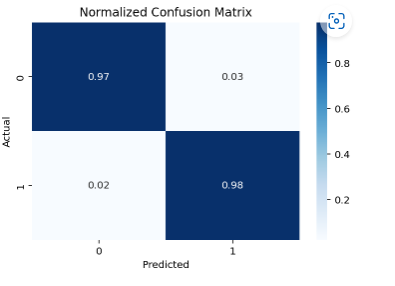
sns.heatmap(conf\_matrix\_norm, annot=True, fmt='.2f', cmap='Blues')

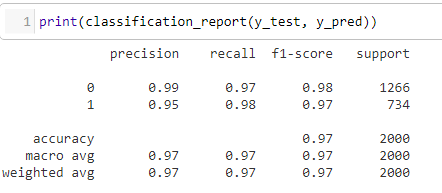
plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Normalized Confusion Matrix')

plt.show()





These results show that the precision rate for the zero class (No) is 99% or that the model correctly classifies the No predictions 99% of the time. The one class (Yes) is predicted correctly 95% of the time.

The recall metric tells us how well the model can detect the actual Yes or No cases. For this model, the Yes group was found 97% of the time, and the No group was found 97% of the time.

The AUC value is 0.99, indicating the strength of this model in making accurate predictions.

## ROC and AUC

# Generate probability scores

y\_pred\_prob = knn.predict\_proba(X\_test\_scaled)[ :, 1]

# Calculate the ROC curve

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_pred\_prob)

# Calculate AUC

auc = roc\_auc\_score(y\_test, y\_pred\_prob)

# Plotting the ROC curve

plt.figure(figsize=(8, 6))

plt.plot(false\_positive\_rate, true\_positive\_rate, color='darkorange', lw=2, label=f'ROC curve (area = {auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

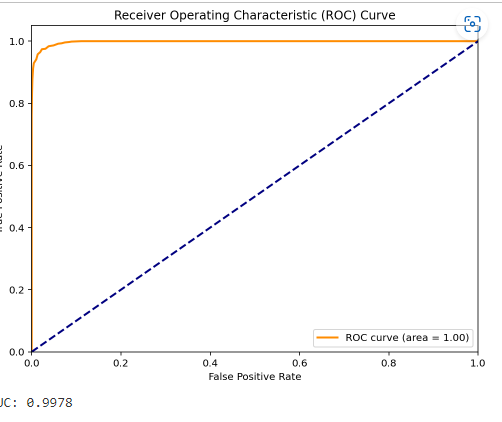
plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

print(f"AUC: {auc:.4f}")



The results of this analysis have high values for all the metrics used in evaluating the model. This allows for solid confidence in predicting which patient will be readmitted within thirty days based on the selected features being present. The implications of a model that has high evaluation metrics mean that when imputing new data for evaluation, the results can also be used with a high level of confidence, especially if actionable steps are being taken based on the results.

E3: Limits

While the metrics imply high performance of this KNN classification model, some limitations should be considered. This data set has a class imbalance of the ReAdmis variable, with the non-readmission rate at approximately 63%. When the data has a more prevalent response than other categories, it is possible to have a model with a high accuracy rate but poor prediction of the variable that is important to predict. This model shows a higher prediction rate for readmission patients than for readmission patients. It can be more detrimental to be incorrect in this category than simple miss-classification of a patient who did not return to the hospital. It may be prudent to adjust the model to have a lower accuracy rate to catch more readmissions than a high overall accuracy rate (Bruce et al., 2019).

E4: Recommendations –

Based on the results of this model, the recommendation is to create a dashboard to track patients who have been admitted with these specific criteria. As the patients are tracked, resources can be allotted to address extra needs these patients may have. With the penalties that hospitals receive due to one-month readmission rates, the return on investment of increasing patient care may be offset by possible decreased readmissions (Allen, 2023). A dashboard or specific tracking system would allow for further insights into the possible causation of readmissions among these patient groups. The one-month buffer would allow for thoughtful delegation of increased resources for patients while being mindful of costs to the hospital.

**Part VI**

F : Panopto

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=3f7487eb-1148-4a41-b76d-b15d01744df0>

G: Web sources

Allen, J., MD (2023, January 4). *Understanding The 2023 Medicare Hospital Readmission Penalty*. Hospital Medical Director. Retrieved February 29, 2024, from <https://hospitalmedicaldirector.com/understanding-the-2023-medicare-hospital-readmission-penalty/>

Agrawal, S. (2023, June 25). *Feature Selection Using Lasso Regression*. Retrieved April 15, 2024, from <https://medium.com/@agrawalsam1997/feature-selection-using-lasso-regression-10f49c973f08>

Band, A. (2020, May 23). *How to find the optimal value of K in KNN?* Toward Data Science. Retrieved April 15, 2024, from <https://medium.com/towards-data-science/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>

[Bold Analytics : Mark Keith]. (2024, January 29). *Python Data Science: Automating EDA: Univariate Statistics and Visualizations* [Video]. YouTube. <https://www.youtube.com/watch?v=eloR5Li0Huo&list=PLe9UEU4oeAuX3GZgRKzUlLcHd5SpYzZGW&index=1>

[Bold Analytics : Mark Keith]. (2024, February 5). *Python Data Science: Automating Cleaning: Data Wrangling: Empty, Single Value, Primary Key columns* [Video]. YouTube. https://www.youtube.com/watch?v=8IsxTImOhF0&list=PLe9UEU4oeAuX3GZgRKzUlLcHd5SpYzZGW&index=8

Brownlee, J. (2020, August 15). *Data Leakage in Machine Learning*. Machine Learning Mastery. Retrieved April 15, 2024, from <https://machinelearningmastery.com/data-leakage-machine-learning/>

D, K. (2023, February 15). *Optimizing Performance: SelectKBest for Efficient Feature Selection in Machine Learning*. Retrieved April 15, 2024, from <https://medium.com/@Kavya2099/optimizing-performance-selectkbest-for-efficient-feature-selection-in-machine-learning-3b635905ed48>

Harrison, O. (2018, September 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Toward Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>

Hale, J. (2019, March 4). *Scale, Standardize, or Normalize with Scikit-Learn*. Toward Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02>

Haymond, S., PhD, DABCC, FADLM, & Master, S., MD, PhD, FADLM (2022, April 1). Why Clinical Laboratorians Should Embrace the R Programming Language A Case for Learning R as a Gateway to Laboratory Medicine’s Digital Future. AACC.org. [https://www.aacc.org/cln/articles/2020/april/why-clinical-laboratorians-should-embrace-the-r-programming-language#](https://www.aacc.org/cln/articles/2020/april/why-clinical-laboratorians-should-embrace-the-r-programming-language)

Horsch, A. (2021, February 14). *Detecting and Treating Outliers In Python — Part 3 Hands-On Tutorial On Treating Outliers — Winsorizing and Imputation*. Towards Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/detecting-and-treating-outliers-in-python-part-3-dcb54abaf7b0>

Kumar, D. (2018, December 25). *Introduction to Data Preprocessing in Machine Learning*. Towards Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d>

Middleton, K. (2023, July 13). D208 - Webinar: Getting Started with D208 Part II [Lecture]. <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=39bbe2db-de7d-4bf5-913b-af5c0003da9d>

Pramoditha, R. (2021). *Encoding Categorical Variables* [Photograph]. Medium : Towards Data Science. <https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>

Pramoditha, R. (2021, December 16). *Encoding Categorical Variables: One-hot vs Dummy Encoding Implementation with Pandas and Scikit-learn*. Towards Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>

Santos, G. (2021, August 18). *How to do Cross-Validation, KFold and Grid Search in Python*. Toward Data Science. Retrieved April 15, 2024, from <https://medium.com/towarhttps://medium.com/gustavorsantos/how-to-do-cross-validation-kfold-and-grid-search-in-python-e570cdb20a28ds-data-science/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>

Soetewey, A. (2019, December 30). Variable types and examples. Stats and R. Retrieved August 26, 2023, from <https://statsandr.com/blog/variable-types-and-examples/>

Weiran, S. (2019, October 22). *Avoid Data Leakage — Split Your Data Before Processing*. Towards Data Science. Retrieved April 15, 2024, from <https://towardshttps://medium.com/towards-data-science/avoid-data-leakage-split-your-data-before-processing-a7f172632b00datascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>

[WGU - D206 Data Cleaning]. Middleton, K. (2023, August 1). Getting Started with D206 | Missing Values [Video]. WGU Panopto. <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=767749d2-ba19-4f94-bec8-b058017b2f5e>

[WGU - D206 Data Cleaning]. Middleton, K. (2023, August 1). Getting Started with D206 | Principal Component Analysis (PCA) [Video]. WGU Panopto. <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=3bcc452f-fa35-43be-b69f-b05901356f95>

I: Sources

Anscombe, F. J. (1973). Graphs in Statistical Analysis. The American Statistician, 27(1), 17–21. <https://doi.org/10.2307/2682899>

Bishop PA, Herron RL. Use and Misuse of the Likert Item Responses and Other Ordinal Measures. Int J Exerc Sci. 2015 Jul 1;8(3):297-302. PMID: 27182418; PMCID: PMC4833473

Bruce, P., Bruce, A., & Gedeck, P. (2019). Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python (2nd ed.). O'Reilly Media.

Larouse, D. T. (2015). Data Mining and Predictive Analytics (p. 20). John Wiley & Sons, Incorporated. <http://ebookcentral.proquest.com/lib/westerngovernors-ebooks/detail.action?docID=7104155>

Monaghan, T. F., Rahmen, S. N., Agudelo, C. W., Wein, A. J., Lazar, J. M., Everaert, K., & Dmochowski, R. R. (2021). Foundational Statistical Principles in Medical Research: Sensitivity, Specificity, Positive Predictive Value, And Negaive Predictive Value. *Medicina*, *57*(5), 503. <https://doi.org/10.3390>

Stigler, S. Fisher and the 5% level. CHANCE 21, 12 (2008). https://doi.org/10.1007/s00144-008-0033-3

References

Allen, J., MD (2023, January 4). *Understanding The 2023 Medicare Hospital Readmission Penalty*. Hospital Medical Director. Retrieved February 29, 2024, from <https://hospitalmedicaldirector.com/understanding-the-2023-medicare-hospital-readmission-penalty/>

Anscombe, F. J. (1973). Graphs in Statistical Analysis. The American Statistician, 27(1), 17–21. <https://doi.org/10.2307/2682899>

Agrawal, S. (2023, June 25). *Feature Selection Using Lasso Regression*. Retrieved April 15, 2024, from <https://medium.com/@agrawalsam1997/feature-selection-using-lasso-regression-10f49c973f08>

Bishop, P. A., & Herron, R. L. (2015). Use and Misuse of the Likert Item Responses and Other Ordinal Measures. International journal of exercise science, 8(3), 297–302.

Band, A. (2020, May 23). *How to find the optimal value of K in KNN?* Toward Data Science. Retrieved April 15, 2024, from <https://medium.com/towards-data-science/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>

[Bold Analytics : Mark Keith]. (2024, January 29). *Python Data Science: Automating EDA: Univariate Statistics and Visualizations* [Video]. YouTube. <https://www.youtube.com/watch?v=eloR5Li0Huo&list=PLe9UEU4oeAuX3GZgRKzUlLcHd5SpYzZGW&index=1>

[Bold Analytics : Mark Keith]. (2024, February 5). *Python Data Science: Automating Cleaning: Data Wrangling: Empty, Single Value, Primary Key columns* [Video]. YouTube. https://www.youtube.com/watch?v=8IsxTImOhF0&list=PLe9UEU4oeAuX3GZgRKzUlLcHd5SpYzZGW&index=8

Brownlee, J. (2020, August 15). *Data Leakage in Machine Learning*. Machine Learning Mastery. Retrieved April 15, 2024, from <https://machinelearningmastery.com/data-leakage-machine-learning/>

Bruce, P., Bruce, A., & Gedeck, P. (2019). Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python (2nd ed.). O'Reilly Media.

D, K. (2023, February 15). *Optimizing Performance: SelectKBest for Efficient Feature Selection in Machine Learning*. Retrieved April 15, 2024, from <https://medium.com/@Kavya2099/optimizing-performance-selectkbest-for-efficient-feature-selection-in-machine-learning-3b635905ed48>

Hale, J. (2019, March 4). *Scale, Standardize, or Normalize with Scikit-Learn*. Toward Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02>

Harrison, O. (2018, September 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Toward Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>

Horsch, A. (2021, February 14). *Detecting and Treating Outliers In Python — Part 3 Hands-On Tutorial On Treating Outliers — Winsorizing and Imputation*. Towards Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/detecting-and-treating-outliers-in-python-part-3-dcb54abaf7b0>

Kumar, D. (2018, December 25). *Introduction to Data Preprocessing in Machine Learning*. Towards Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d>

Larouse, D. T. (2015). Data Mining and Predictive Analytics (p. 20). John Wiley & Sons, Incorporated. <http://ebookcentral.proquest.com/lib/westerngovernors-ebooks/detail.action?docID=7104155>.

Middleton, K. (2023, July 13). D208 - Webinar: Getting Started with D208 Part II [Lecture]. <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=39bbe2db-de7d-4bf5-913b-af5c0003da9d>

Monaghan, T. F., Rahmen, S. N., Agudelo, C. W., Wein, A. J., Lazar, J. M., Everaert, K., & Dmochowski, R. R. (2021). Foundational Statistical Principles in Medical Research: Sensitivity, Specificity, Positive Predictive Value, And Negaive Predictive Value. *Medicina*, *57*(5), 503. <https://doi.org/10.3390>

Pramoditha, R. (2021). *Encoding Categorical Variables* [Photograph]. Medium : Towards Data Science. <https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>

Pramoditha, R. (2021, December 16). *Encoding Categorical Variables: One-hot vs Dummy Encoding Implementation with Pandas and Scikit-learn*. Towards Data Science. Retrieved April 15, 2024, from <https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>

Santos, G. (2021, August 18). *How to do Cross-Validation, KFold and Grid Search in Python*. Toward Data Science. Retrieved April 15, 2024, from <https://medium.com/towarhttps://medium.com/gustavorsantos/how-to-do-cross-validation-kfold-and-grid-search-in-python-e570cdb20a28ds-data-science/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>

Soetewey, A. (2019, December 30). Variable types and examples. Stats and R. Retrieved August 26, 2023, from <https://statsandr.com/blog/variable-types-and-examples/>

Stigler, S. Fisher and the 5% level. CHANCE 21, 12 (2008). <https://doi.org/10.1007/s00144-008-0033-3>

Weiran, S. (2019, October 22). *Avoid Data Leakage — Split Your Data Before Processing*. Towards Data Science. Retrieved April 15, 2024, from <https://towardshttps://medium.com/towards-data-science/avoid-data-leakage-split-your-data-before-processing-a7f172632b00datascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>

[WGU - D206 Data Cleaning]. Middleton, K. (2023, August 1). Getting Started with D206 | Missing Values [Video]. WGU Panopto. <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=767749d2-ba19-4f94-bec8-b058017b2f5e>