Western Governors University

D209 Data Mining – Task 1

By Krista Moik

Table of Contents

A1: Proposal of Question **2**

A2: Defined Goal **2**

B1: Explanation of Classification Method **2**

B2: Summary of Method Assumption **2**

B3: Packages or Libraries List **2**

C1: Data Preprocessing **3**

C2: Data Set Variables **3**

C3: Steps for Analysis **4**

C4: Cleaned Data Set **18**

D1: Splitting the Data **18**

D2: Output and Intermediate Calculations **19**

D3: Code Execution **21**

E1: Accuracy and AUC **23**

E2: Results and Implications **24**

E3: Limitation **25**

E4: Course of Action **25**

F: Panopto Recording **25**

G: Sources for Third-Party Code **25**

H: Sources **25**

I: Professional Communication **26**

Part I: Research Question

A1. Proposal Question

Using the medical_clean dataset, my research question is: **Which variables predict whether a patient is readmitted?** To answer this question, I will use k-nearest neighbor (KNN) to create a supervised machine learning model.

A2. Defined Goal

The goal of my data analysis is to create a supervised machine learning model using KNN to identify key variables that predict the probability of patients being readmitted. Once I develop this model, I would be able to offer real-world recommendations regarding readmission.

Part II: Method Justification

B1. Explanation of Classification Method

The classification method I have chosen is KNN. KNN is useful as it is non-parametric and can be used with both numerical and categorical data. This method makes predictions based on the similarity of data points using a distance metric, where the class or value of the data point is determined by the average of the K neighbors (Geeks for Geeks, n.d.).

B2. Summary of Method Assumption

One assumption of KNN is that it assumes that things that are in close proximity with each other are also similar to each other. Thus, this model assumes that information about one data point can help us draw conclusions about neighboring data points (Grant, 2019).

B3. Packages or Libraries List

I am using Python for this analysis and will import the following libraries, some of which are suggested in WGU Course Materials (Western Governors University, n.d.):

First, I will import pandas and numpy as usual for their abilities in data manipulation and simple math and arrays.

I will also import matplotlib and seaborn to perform visualizations like the ROC curve required for this analysis and matrices.

I will use several packages from sklearn to complete the analysis as well. SelectKBest and f_classif sklearn.feature_selection import SelectKBest,f_regression will be imported to help select the best variables for the model. Scale will be imported to scale the variables, as KNN requires. Train_test_split will be imported to split the data in the train and test datasets. GridSearchCV will be used for

hyperparameter tuning. Confusion_matrix, roc_auc_score, and roc_curve will be used in evaluating my model. KNeighborsClassifier and classification_report will be used to identify K and run the KNN algorithm. Logistic Regression and variance_inflation_factor from statsmodel will be used to locate p-values and multicollinearity. I will also import preprocessing and OneHotEncoder from sklearn to scale and re-express certain categorical variables.

Part III: Data Preparation

C1. Data Preprocessing

One of the most important steps in the preprocessing process is re-expressing all of the categorical data points into numeric form so they can be used in the KNN model with the other numeric data points. For this data set, I will be using both ordinal encoding to change Yes and No responses to 1s and 0s, and onehot_encoder for variables that have more than 2 response types. It is important to note that when using KNN, all variables should be kept, whereas in regression, we dropped the first column (Shmueli, 2015). Once all variables have been properly re-expressed, I will use the preprocessing package to scale my independent variables. ReAdmis is my dependent variable as that is what I am trying to predict using KNN.

C2. Data Set Variables

Below are my initial variables I chose to test in my model. I did choose to remove such variables like CaseOrder, UID, and all survey responses as I did not feel they would be relevant to predicting readmission.

Variable	Type
Children	Numeric – Discrete
Age	Numeric – Continuous
Income	Numeric – Continuous
VitD_levels	Numeric – Continuous
Doc_visits	Numeric – Discrete
Full_meals_eaten	Numeric – Discrete
vitD_supp	Numeric – Discrete
Initial_days	Numeric – Continuous
TotalCharge	Numeric – Continuous
Additional_charges	Numeric – Continuous
ReAdmis	Categorical – yes or no
Soft_drink	Categorical – yes or no
HighBlood	Categorical – yes or no
Stroke	Categorical – yes or no
Overweight	Categorical – yes or no
Arthritis	Categorical – yes or no
Diabetes	Categorical – yes or no
Hyperlipidemia	Categorical – yes or no
BackPain	Categorical – yes or no
Anxiety	Categorical – yes or no

Allergic_rhinitis	Categorical – yes or no
Reflux_esophagitis	Categorical – yes or no
Asthma	Categorical – yes or no
Gender	Categorical – Male, Female, Nonbinary
Marital	Categorical – Married, Separated, Divorced, Widowed, Never Married
Complication_risk	Categorical – High, Medium, Low
Initial_admin	Categorical – Emergency, Elective, Observation
Area	Categorical – Urban, Suburban, Rural
Services	Categorical – Blood work, Intravenous, CT Scan, MRI

In the next section, I will describe my treatment of the variables to prepare for my model.

C3. Steps for Analysis

First, I imported the packages I planned on using in my analysis, many of which were suggested in the WGU course materials (Western Governors University, n.d.):

```
#import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, f_regression, f_classif
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import classification_report
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

I then loaded the data set medical_clean.csv from my desktop:

```
#Load medical_clean CSV
df_209=pd.read_csv('C:/Users/Kmoik WGU/Desktop/D209/medical_clean.csv')
```

I then viewed the dataset using the supplied data dictionary and code df_209.info():

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column            Non-Null Count  Dtype
```

```
---  ------              --------------  -----
 0   CaseOrder          10000 non-null  int64
 1   Customer_id        10000 non-null  object
 2   Interaction        10000 non-null  object
 3   UID                10000 non-null  object
 4   City               10000 non-null  object
 5   State              10000 non-null  object
 6   County             10000 non-null  object
 7   Zip                10000 non-null  int64
 8   Lat                10000 non-null  float64
 9   Lng                10000 non-null  float64
 10  Population         10000 non-null  int64
 11  Area               10000 non-null  object
 12  TimeZone           10000 non-null  object
 13  Job                10000 non-null  object
 14  Children           10000 non-null  int64
 15  Age                10000 non-null  int64
 16  Income             10000 non-null  float64
 17  Marital            10000 non-null  object
 18  Gender             10000 non-null  object
 19  ReAdmis            10000 non-null  object
 20  VitD_levels        10000 non-null  float64
 21  Doc_visits         10000 non-null  int64
 22  Full_meals_eaten   10000 non-null  int64
 23  vitD_supp          10000 non-null  int64
 24  Soft_drink         10000 non-null  object
 25  Initial_admin      10000 non-null  object
 26  HighBlood          10000 non-null  object
 27  Stroke             10000 non-null  object
 28  Complication_risk  10000 non-null  object
 29  Overweight         10000 non-null  object
 30  Arthritis          10000 non-null  object
 31  Diabetes           10000 non-null  object
 32  Hyperlipidemia     10000 non-null  object
 33  BackPain           10000 non-null  object
 34  Anxiety            10000 non-null  object
 35  Allergic_rhinitis  10000 non-null  object
 36  Reflux_esophagitis 10000 non-null  object
 37  Asthma             10000 non-null  object
 38  Services           10000 non-null  object
 39  Initial_days       10000 non-null  float64
 40  TotalCharge        10000 non-null  float64
 41  Additional_charges 10000 non-null  float64
 42  Item1              10000 non-null  int64
 43  Item2              10000 non-null  int64
 44  Item3              10000 non-null  int64
 45  Item4              10000 non-null  int64
 46  Item5              10000 non-null  int64
```

```
 47  Item6       10000 non-null  int64
 48  Item7       10000 non-null  int64
 49  Item8       10000 non-null  int64
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

Next, I confirmed there were no duplicate or null values using duplicated().value_counts() and isnull().sum().

```
#check for duplicates
print(df_209.duplicated().value_counts())
```

```
False    10000
Name: count, dtype: int64
```

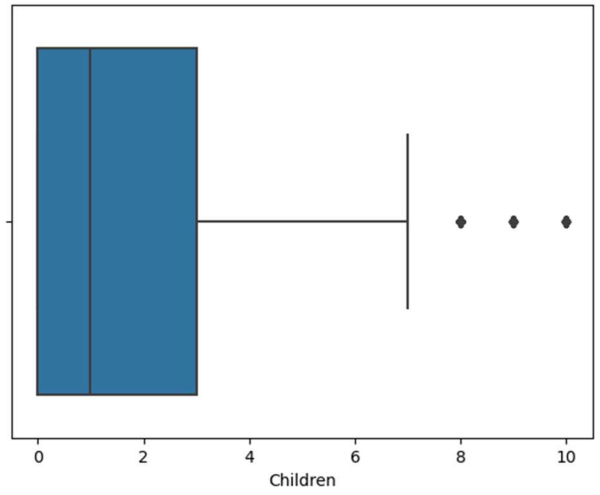
Confirmed no duplicates

```
#check for null values - even though view of data indicates no nulls
df_209.isnull().sum()
```

I used the function boxplot() to visualize all of the numeric variables and describe() to obtain the statistics of them:

```
sns.boxplot(df_209, x='Children')
```

```
<Axes: xlabel='Children'>
```

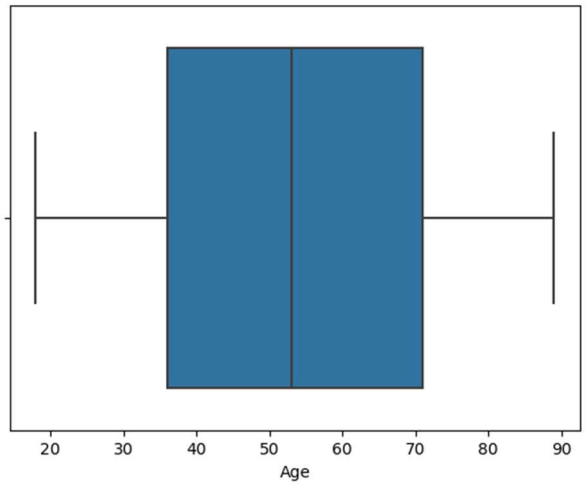



```
df_209.Children.describe()
```

```
count    10000.000000
mean         2.097200
std          2.163659
min          0.000000
25%          0.000000
50%          1.000000
75%          3.000000
max         10.000000
Name: Children, dtype: float64
```

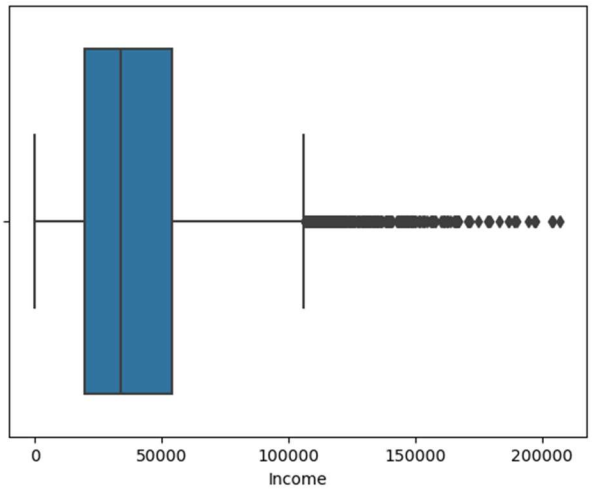
```
sns.boxplot(df_209, x='Age')
```

```
<Axes: xlabel='Age'>
```




```
sns.boxplot(df_209, x='Income')
```

```
<Axes: xlabel='Income'>
```

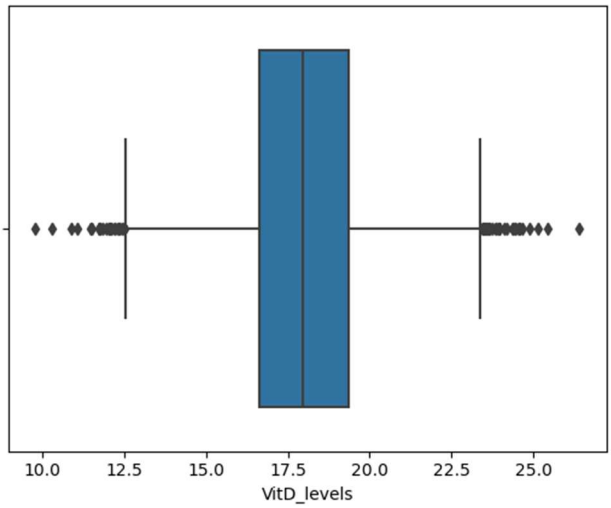



```
df_209.Income.describe()
```

```
count     10000.000000
mean      40490.495160
std       28521.153293
min         154.080000
25%       19598.775000
50%       33768.420000
75%       54296.402500
max      207249.100000
Name: Income, dtype: float64
```

```
sns.boxplot(df_209, x='VitD_levels')
```

```
<Axes: xlabel='VitD_levels'>
```

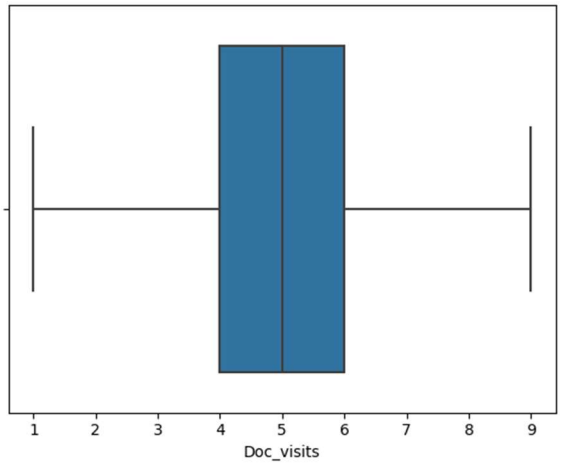



```
df_209.VitD_levels.describe()
```

```
count    10000.000000
mean        17.964262
std          2.017231
min          9.806483
25%         16.626439
50%         17.951122
75%         19.347963
max         26.394449
Name: VitD_levels, dtype: float64
```

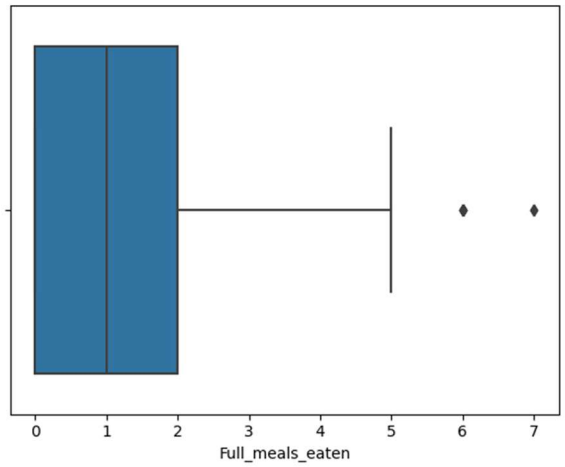
```
sns.boxplot(df_209, x='Doc_visits')
```

```
<Axes: xlabel='Doc_visits'>
```



```
sns.boxplot(df_209, x='Full_meals_eaten')
```

```
<Axes: xlabel='Full_meals_eaten'>
```

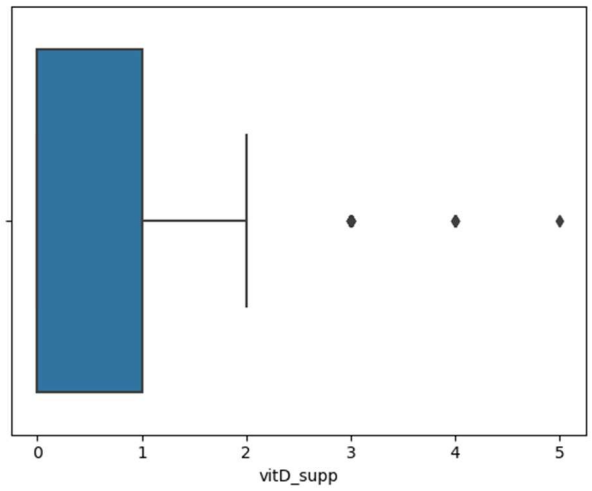



```
df_209.Full_meals_eaten.describe()
```

```
count    10000.000000
mean         1.001400
std          1.008117
min          0.000000
25%          0.000000
50%          1.000000
75%          2.000000
max          7.000000
Name: Full_meals_eaten, dtype: float64
```

```
sns.boxplot(df_209, x='vitD_supp')
```

```
<Axes: xlabel='vitD_supp'>
```

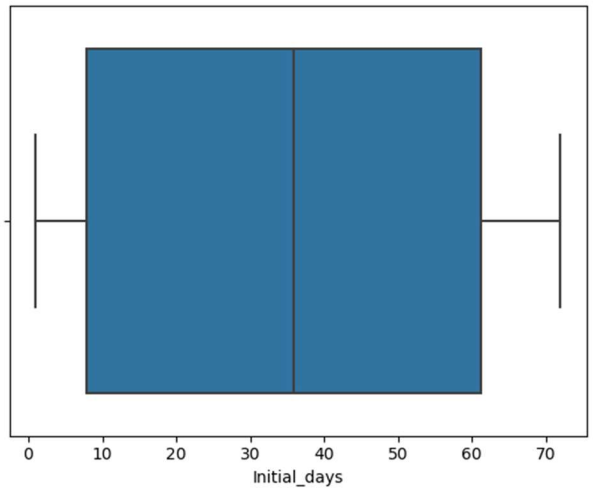



```
df_209.vitD_supp.describe()
```

```
count    10000.000000
mean         0.398900
std          0.628505
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          5.000000
Name: vitD_supp, dtype: float64
```

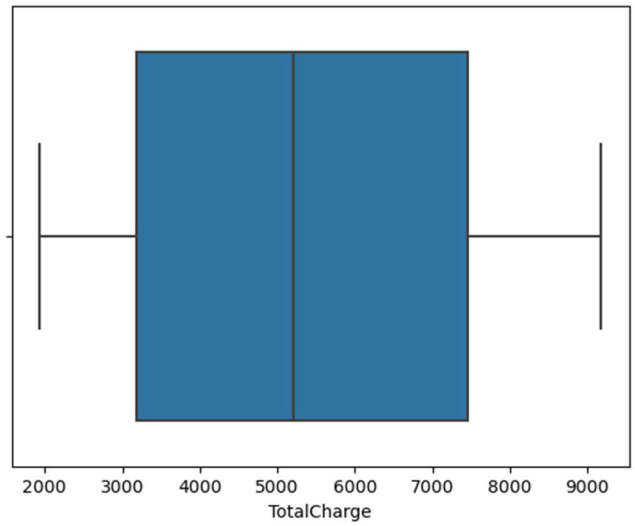
```
sns.boxplot(df_209, x='Initial_days')
```

```
<Axes: xlabel='Initial_days'>
```



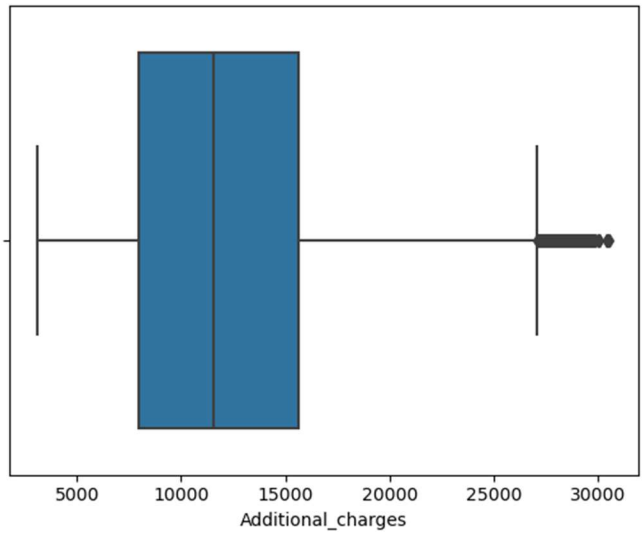

```
sns.boxplot(df_209, x='TotalCharge')
```

```
<Axes: xlabel='TotalCharge'>
```



```
sns.boxplot(df_209, x='Additional_charges')
```

```
<Axes: xlabel='Additional_charges'>
```




```
df_209.Additional_charges.describe()
```

```
count    10000.000000
mean     12934.528587
std       6542.601544
min       3125.703000
25%       7986.487755
50%      11573.977735
75%      15626.490000
max      30566.070000
Name: Additional_charges, dtype: float64
```

After reviewing the outliers, I once again chose to retain all outliers. I felt this was the best option to maintain the integrity and diversity of the dataset as all of the outliers appeared to be reasonable and justified.

Once the data was clean, I moved on to re-expressing my categorical variables. For the variables that were yes or no responses, I replaced the yes responses with 1 and the no responses with 0, renaming those columns as ColumnName_numeric:

```
#re-expressing ReAdmis
df_209['ReAdmis_numeric']=df_209['ReAdmis']
```

```
#set up dictionary
dict_readmis={'ReAdmis_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_readmis, inplace=True)
```

```
#re-expressing Soft_drink
df_209['Soft_drink_numeric']=df_209['Soft_drink']
```

```
#set up dictionary
dict_soft={'Soft_drink_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_soft, inplace=True)
```

```
#re-expressing HighBlood
df_209['HighBlood_numeric']=df_209['HighBlood']
```

```
#set up dictionary
dict_blood={'HighBlood_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_blood, inplace=True)
```

```
#re-expressing Stroke
df_209['Stroke_numeric']=df_209['Stroke']
```

```
#set up dictionary
dict_stroke={'Stroke_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_stroke, inplace=True)
```

```
#re-expressing Overweight
df_209['Overweight_numeric']=df_209['Overweight']
```

```
#set up dictionary
dict_weight={'Overweight_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_weight, inplace=True)
```

```
#re-expressing Arthritis
df_209['Arthritis_numeric']=df_209['Arthritis']
```

```
#set up dictionary
dict_arthritis={'Arthritis_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_arthritis, inplace=True)
```

```
#re-expressing Diabetes
df_209['Diabetes_numeric']=df_209['Diabetes']
```

```
#set up dictionary
dict_diab={'Diabetes_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_diab, inplace=True)
```

```
#re-expressing Hyperlipidemia
df_209['Hyperlipidemia_numeric']=df_209['Hyperlipidemia']
```

```
#set up dictionary
dict_lip={'Hyperlipidemia_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_lip, inplace=True)
```

```
#re-expressing BackPain
df_209['BackPain_numeric']=df_209['BackPain']
```

```
#set up dictionary
dict_back={'BackPain_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_back, inplace=True)
```

```
#re-expressing Anxiety
df_209['Anxiety_numeric']=df_209['Anxiety']
```

```
#set up dictionary
dict_anx={'Anxiety_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_anx, inplace=True)
```

```
#re-expressing Allergic_rhinitis
df_209['Allergic_rhinitis_numeric']=df_209['Allergic_rhinitis']
```

```
#set up dictionary
dict_alle={'Allergic_rhinitis_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_alle, inplace=True)
```

```
#re-expressing Reflux_esophagitis
df_209['Reflux_esophagitis_numeric']=df_209['Reflux_esophagitis']
```

```
#set up dictionary
dict_refl={'Reflux_esophagitis_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_refl, inplace=True)
```

```
#re-expressing Asthma
df_209['Asthma_numeric']=df_209['Asthma']
```

```
#set up dictionary
dict_asth={'Asthma_numeric' : {'No':0, 'Yes':1}}
```

```
#replace variable's values
df_209.replace(dict_asth, inplace=True)
```

I dropped the original columns from the dataset:

```
#drop original columns
df_209.drop(['Asthma', 'Reflux_esophagitis', 'Allergic_rhinitis', 'ReAdmis', 'Soft_drink', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis', 'Diabetes', 'BackPain', 'Anxiety', 'Hyperlipidemia'], a
```

I then used onehot_encoder to re-express the variables that had more than 2 response types. Unlike regression modeling, for KNN, I did not drop the first column after re-expression. I did dop the original columns.

```
#re-express variables with more than 2 response types
onehot_encoder = OneHotEncoder(sparse=False)
```

```
onehot_encoded=onehot_encoder.fit_transform(df_209[['Gender', 'Marital', 'Complication_risk', 'Initial_admin', 'Area', 'Services']])
```

```
C:\Users\Kmoik WGU\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_out
put` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

```
df_209encoded=pd.DataFrame(onehot_encoded, columns=onehot_encoder.get_feature_names_out(['Gender', 'Marital', 'Complication_risk', 'Initial_admin', 'Area', 'Services']))
```

```
df_209encoded=df_209encoded.astype('int64')
```

```
#merging encoded columns to df
df_209=pd.concat([df_209, df_209encoded], axis=1)
```

```
#drop original columns
df_209.drop(['Gender', 'Marital', 'Complication_risk', 'Initial_admin', 'Area', 'Services'], axis=1, inplace=True)
```

I used code df_209.info() to confirm the updated variables in the dataset were all numeric:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 44 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Children                    10000 non-null  int64
 1   Age                         10000 non-null  int64
 2   Income                      10000 non-null  float64
 3   VitD_levels                 10000 non-null  float64
 4   Doc_visits                  10000 non-null  int64
 5   Full_meals_eaten            10000 non-null  int64
 6   vitD_supp                   10000 non-null  int64
 7   Initial_days                10000 non-null  float64
 8   TotalCharge                 10000 non-null  float64
 9   Additional_charges          10000 non-null  float64
 10  ReAdmis_numeric             10000 non-null  int64
 11  Soft_drink_numeric          10000 non-null  int64
 12  HighBlood_numeric           10000 non-null  int64
 13  Stroke_numeric              10000 non-null  int64
 14  Overweight_numeric          10000 non-null  int64
 15  Arthritis_numeric           10000 non-null  int64
 16  Diabetes_numeric            10000 non-null  int64
 17  Hyperlipidemia_numeric      10000 non-null  int64
 18  BackPain_numeric            10000 non-null  int64
 19  Anxiety_numeric             10000 non-null  int64
 20  Allergic_rhinitis_numeric   10000 non-null  int64
 21  Reflux_esophagitis_numeric  10000 non-null  int64
 22  Asthma_numeric              10000 non-null  int64
 23  Gender_Female               10000 non-null  int64
 24  Gender_Male                 10000 non-null  int64
 25  Gender_Nonbinary            10000 non-null  int64
 26  Marital_Divorced            10000 non-null  int64
 27  Marital_Married             10000 non-null  int64
 28  Marital_Never Married       10000 non-null  int64
 29  Marital_Separated           10000 non-null  int64
```

```
 30  Marital_Widowed                  10000 non-null  int64
 31  Complication_risk_High           10000 non-null  int64
 32  Complication_risk_Low            10000 non-null  int64
 33  Complication_risk_Medium         10000 non-null  int64
 34  Initial_admin_Elective Admission    10000 non-null  int64
 35  Initial_admin_Emergency Admission   10000 non-null  int64
 36  Initial_admin_Observation Admission  10000 non-null  int64
 37  Area_Rural                       10000 non-null  int64
 38  Area_Suburban                    10000 non-null  int64
 39  Area_Urban                       10000 non-null  int64
 40  Services_Blood Work              10000 non-null  int64
 41  Services_CT Scan                 10000 non-null  int64
 42  Services_Intravenous             10000 non-null  int64
 43  Services_MRI                     10000 non-null  int64
dtypes: float64(5), int64(39)
memory usage: 3.4 MB
```

Next, I separated my predictor variables from my dependent variable and then scaled the predictor variables since by using KNN we were using distance, using code from WGU Course Materials (Western Governors University, n.d.):

```
# Assign values to X for all predictor features
X = df_209.drop('ReAdmis_numeric', axis=1)
# Assign values to y for the dependent variable
y = df_209['ReAdmis_numeric']
```

```
# Standardize the X variables (explanatory) to put all on the same scale for KNN to measure distance
df_209 = pd.DataFrame(preprocessing.MinMaxScaler().fit_transform(df_209), columns=df_209.columns)
df_209
```

Next, I used SelectKBest to identify the most significant variables to my predictor variable, using code from WGU Course Materials (Western Governors University, n.d.):

```
#k=features
k_best = SelectKBest(score_func = f_classif, k='all')
```

```
X_new = k_best.fit_transform(X, y)
```

```
selected_features_indices=k_best.get_support(indices=True)
```

```
selected_features_names=X.columns[selected_features_indices]
```

```
#find p values
p_values = pd.DataFrame({'Feature': X.columns, 'p_value':k_best.pvalues_}).sort_values('p_value')
features_to_keep = p_values['Feature'][p_values['p_value'] < .05]
```

```
print(p_values)
```

```
                Feature   p_value
8           TotalCharge  0.000000
7          Initial_days  0.000000
```

40	Services_CT Scan	0.014707
0	Children	0.018613
25	Marital_Divorced	0.030143
41	Services_Intravenous	0.042233
34	Initial_admin_Emergency Admission	0.048766
21	Asthma_numeric	0.086677
1	Age	0.113891
9	Additional_charges	0.173237
17	BackPain_numeric	0.183133
5	Full_meals_eaten	0.223574
35	Initial_admin_Observation Admission	0.231263
22	Gender_Female	0.243554
2	Income	0.250029
6	vitD_supp	0.269697
33	Initial_admin_Elective Admission	0.276650
23	Gender_Male	0.326515
42	Services_MRI	0.351962
13	Overweight_numeric	0.390612
29	Marital_Widowed	0.390638
10	Soft_drink_numeric	0.441195
14	Arthritis_numeric	0.443546
36	Area_Rural	0.453272
38	Area_Urban	0.476787
27	Marital_Never Married	0.496533
26	Marital_Married	0.509864
24	Gender_Nonbinary	0.520390
20	Reflux_esophagitis_numeric	0.587736
19	Allergic_rhinitis_numeric	0.641923
16	Hyperlipidemia_numeric	0.666731
3	VitD_levels	0.683120
30	Complication_risk_High	0.690952
15	Diabetes_numeric	0.759776
32	Complication_risk_Medium	0.779605
18	Anxiety_numeric	0.809868
11	HighBlood_numeric	0.820441
31	Complication_risk_Low	0.905636
12	Stroke_numeric	0.926830
39	Services_Blood Work	0.942745
28	Marital_Separated	0.957280
37	Area_Suburban	0.966399
4	Doc_visits	0.980401

The top 7 variables all had p-values less than 0.05, indicating they were statistically significant. I then checked the variance inflation factors to identify any multicollinearity among those 7 most significant variables, using code from WGU Course Materials (Western Governors University, n.d.):

```
#check for multicollinearity using VIF
#set independent variable for VIF
X=df_209[['TotalCharge', 'Initial_days', 'Children', 'Marital_Divorced', 'Initial_admin_Emergency Admission', 'Services_CT Scan', 'Services_Intravenous']]
```

```
#VIF dataframe
vif_df_209=pd.DataFrame()
vif_df_209["feature"]=X.columns
```

```
#calculate VIF for all independent variables
vif_df_209["VIF"]=[variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
```

```
print(vif_df_209)
```

```
                          feature         VIF
0                     TotalCharge  215.896697
1                    Initial_days  177.701826
2                        Children    1.751354
3                  Marital_Divorced    1.204073
4  Initial_admin_Emergency Admission    4.510178
5                  Services_CT Scan    1.179536
6              Services_Intravenous    1.433406
```

The VIF values show high multicollinearity between TotalCharge and Initial_days. I removed the variable with the highest VIF, TotalCharge, and rechecked the VIF:

```
#removing totalcharge due to high multicollinearity
#set independent variable for VIF
X=df_209[['Initial_days', 'Children', 'Marital_Divorced', 'Initial_admin_Emergency Admission', 'Services_CT Scan', 'Services_Intravenous']]
```

```
#VIF dataframe
vif_df_209=pd.DataFrame()
vif_df_209["feature"]=X.columns
```

```
#calculate VIF for all independent variables
vif_df_209["VIF"]=[variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
```

```
print(vif_df_209)
```

```
                          feature       VIF
0                    Initial_days  1.847057
1                        Children  1.650881
2                  Marital_Divorced  1.186747
3  Initial_admin_Emergency Admission  1.647724
4                  Services_CT Scan  1.153524
5              Services_Intravenous  1.374259
```

All remaining variables had a VIF value under 10, indicating there was most likely no strong multicollinearity among the remaining variables. These 6 variables are what I will use in my KNN model to predict readmission: Initial_days, Children, Marital_Divorced, Initial_admin_Emergency Admission, Services_CT Scan, and Services_Intravenous.

I did rename Initial_admin_Emergency Admission to Initial_admin_Emergency and Services_CT Scan to Services_CTscan to remove the spaces.

```
#renaming statistically significant columns without spaces
df_209.rename(columns={'Initial_admin_Emergency Admission' : 'Initial_admin_Emergency', 'Services_CT Scan' : 'Services_CTscan'}, inplace=True)
```

C4. Cleaned Data Set

Please see the attached CSV titled KMoikD209_cleanmedical with my cleaned data set.

Part IV: Analysis

D1. Splitting the Data

Using code from WGU Course Materials and Dr. Straw's tips for success, I split the data into training and testing sets where 70% of the data was split into a training set, and the remaining 30% was split into the testing set (Western Governors University, n.d.):

```
#Split the data set with an 70/30 split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 25)
```

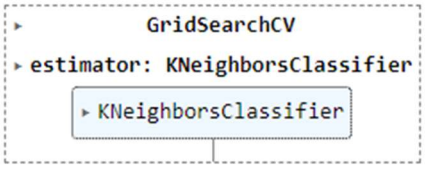
```
#Save the training and testing sets as csv files
pd.DataFrame(X_train).to_csv('C:/Users/Kmoik WGU/Desktop/X_train.csv')
pd.DataFrame(X_test).to_csv('C:/Users/Kmoik WGU/Desktop/X_test.csv')
pd.DataFrame(y_train).to_csv('C:/Users/Kmoik WGU/Desktop/y_train.csv')
pd.DataFrame(y_test).to_csv('C:/Users/Kmoik WGU/Desktop/y_test.csv')
```

Please see the attached CSV files titled X_train, X_test, y_train, and y_test with the split data.

D2. Output and Intermediate Calculations

I used GridSearchCV and best_params to determine the best value for n_neighbors (Western Governors University, n.d.). In this model, using a range of 1 to 50, that value was determined to be 7, which means that the algorithm will look at the 7 nearest neighbors (scikit-learn, n.d.).

```
#hyperparameter tuning
param_grid={'n_neighbors': np.arange(1,50)}
knn=KNeighborsClassifier()
knn_cv=GridSearchCV(knn, param_grid, cv=5)
knn_cv.fit(X_train, y_train)
```




```
# Find best parameter from GridSearch
knn_cv.best_params_
```

```
{'n_neighbors': 7}
```

```
# Find score of best parameter from GridSearchCV
knn_cv.best_score_
```

```
0.9728571428571428
```

This model is using Euclidean distance, which is the actual straight-line distance between 2 points, and per the best_params, I will be looking at the 7 nearest neighbors to make predictions. The best_score of 0.97 indicates this model has a high rate of accuracy. No weights were used.

Using the n_neighbors=7 and KNeighborsClassifier, I fit the model and printed the classification report. This report shows the Accuracy is 0.97, which is the ratio of total correct predictions and total number of test points. The Precision Score is the percentage of correct classifications per label, the Recall Score is

the number of correctly predicted positive observations to all actual positive observations, and the F1-Score is the average of the Precision and Recall Scores (Medium, 2019).

```
# Perform KNN using the value of k=7 from the above grid search
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)
y_pred_prob=knn.predict_proba(X_test)[:,1]
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1900
           1       0.95      0.98      0.97      1100

    accuracy                           0.97      3000
   macro avg       0.97      0.98      0.97      3000
weighted avg       0.97      0.97      0.97      3000
```

Next, I used code to obtain the Accuracy Score, Area Under the Curve (AUC), and create a Confusion Matrix (Western Governors University, n.d.):

```
#Accuracy Score, Confusion Matrix, AUC
# Fit to the training data
knn.fit(X_train, y_train)
print("The Accuracy Score of the KNN model:")
print(knn.score(X_test, y_test))
y_predicted=knn.predict(X_test)
print("The Confusion Matrix for the KNN Model:")
print(confusion_matrix(y_test, y_predicted))
y_predicted_probability = knn.predict_proba(X_test)[:,1]
print("The Area Under the Curve (AUC) for the KNN Model:")
print(roc_auc_score(y_test, y_predicted_probability))
```

```
The Accuracy Score of the KNN model:
0.9743333333333334
The Confusion Matrix for the KNN Model:
[[1840   60]
 [  17 1083]]
The Area Under the Curve (AUC) for the KNN Model:
0.9959787081339713
```

The accuracy score for the model is still high at 0.97. The AUC of 0.99 indicates the model is a very high performer.

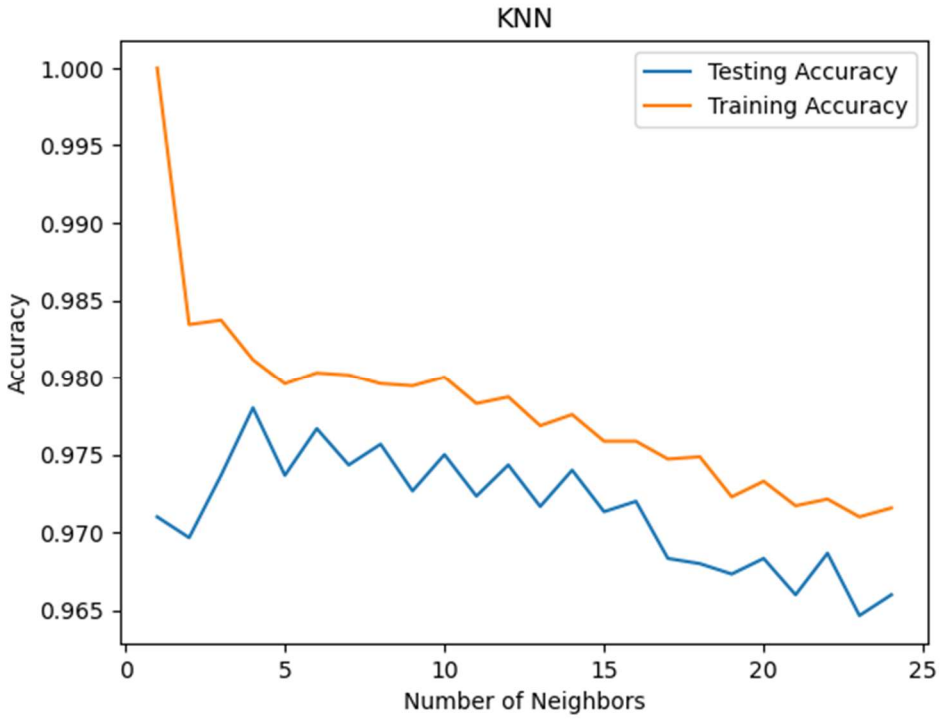
Finally, I created a complexity curve to show the various values of K on the training and testing data set to check for accuracy and to see how accuracy and errors are affected by different k values (Western Governors University, n.d.):

```
#Model Complexity Curve
neighbors = np.arange(1,25)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
# Loop over different values of k
for i, k in enumerate (neighbors):
    knn = KNeighborsClassifier (n_neighbors=k)
    knn.fit (X_train,y_train)
    train_accuracy[i] = knn.score (X_train, y_train)
    test_accuracy[i] = knn.score (X_test, y_test)
print("K: ", i, "Train Score:", train_accuracy[i], "Test Score:", test_accuracy[i])
```

```
K:  23 Train Score: 0.9715714285714285 Test Score: 0.966
```

```
#Generate plot
plt.title('KNN')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```



The accuracy score for the training data set is 0.97 and the accuracy score of the testing data set is 0.96. The complexity curve visualizes the relationship between the testing accuracy and training accuracy across multiple numbers of neighbors. As visualized by the graph, the training data set has a slightly higher accuracy score across all number of neighbors.

D3. Code Execution

A full copy of the code used can be seen in the attached PDF and ipynb files titled KMoikD209Code1. All of the code used below was obtained from WGU Course Materials (Western Governors University, n.d.):

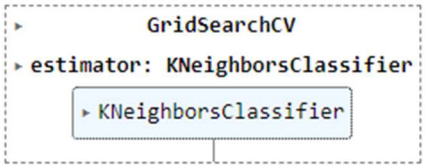
To split the data 70/30 into training and testing data sets and then save them to separate CSV files:

```
#Split the data set with an 70/30 split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 25)
```

```
#Save the training and testing sets as csv files
pd.DataFrame(X_train).to_csv('C:/Users/Kmoik WGU/Desktop/X_train.csv')
pd.DataFrame(X_test).to_csv('C:/Users/Kmoik WGU/Desktop/X_test.csv')
pd.DataFrame(y_train).to_csv('C:/Users/Kmoik WGU/Desktop/y_train.csv')
pd.DataFrame(y_test).to_csv('C:/Users/Kmoik WGU/Desktop/y_test.csv')
```

To find and score the best value for n_neighbors:

```
#hyperparameter tuning
param_grid={'n_neighbors': np.arange(1,50)}
knn=KNeighborsClassifier()
knn_cv=GridSearchCV(knn, param_grid, cv=5)
knn_cv.fit(X_train, y_train)
```




```
# Find best parameter from GridSearch
knn_cv.best_params_
```

```
{'n_neighbors': 7}
```

```
# Find score of best parameter from GridSearchCV
knn_cv.best_score_
```

```
0.9728571428571428
```

To print the classification report:

```
# Perform KNN using the value of k=7 from the above grid search and create classification report
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)
y_pred_prob=knn.predict_proba(X_test)[:,1]
print(classification_report(y_test, y_pred))
```

To find the accuracy score, AUC, and create a confusion matrix:

```
#Accuracy Score, Confusion Matrix, AUC
# Fit to the training data
knn.fit(X_train, y_train)
print("The Accuracy Score of the KNN model:")
print(knn.score(X_test, y_test))
y_predicted=knn.predict(X_test)
print("The Confusion Matrix for the KNN Model:")
print(confusion_matrix(y_test, y_predicted))
y_predicted_probability = knn.predict_proba(X_test)[:,1]
print("The Area Under the Curve (AUC) for the KNN Model:")
print(roc_auc_score(y_test, y_predicted_probability))
```

To model and plot the complexity curve and obtain accuracy scores for the training and testing data sets:

```
#Model Complexity Curve
neighbors = np.arange(1,25)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
# Loop over different values of k
for i, k in enumerate (neighbors):
    knn = KNeighborsClassifier (n_neighbors=k)
    knn.fit (X_train,y_train)
    train_accuracy[i] = knn.score (X_train, y_train)
    test_accuracy[i] = knn.score (X_test, y_test)
print("K: ", i, "Train Score:", train_accuracy[i], "Test Score:", test_accuracy[i])
```

```
K:  23 Train Score: 0.9715714285714285 Test Score: 0.966
```

```
#Generate plot
plt.title('KNN')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Part V: Data Summary and Implications

E1. Accuracy and AUC

My model had an accuracy score of 0.97, which is the number of correct classifications divided by the total samples. The high value indicates a very accurate model.

My model had an AUC (area under the curve) value of 0.99. This is the model's ability to correctly determine positive and negative classifications. The high number means that the model was almost completely accurate in its predictions.

```
#Accuracy Score, Confusion Matrix, AUC
# Fit to the training data
knn.fit(X_train, y_train)
print("The Accuracy Score of the KNN model:")
print(knn.score(X_test, y_test))
y_predicted=knn.predict(X_test)
print("The Confusion Matrix for the KNN Model:")
print(confusion_matrix(y_test, y_predicted))
y_predicted_probability = knn.predict_proba(X_test)[:,1]
print("The Area Under the Curve (AUC) for the KNN Model:")
print(roc_auc_score(y_test, y_predicted_probability))
```

```
The Accuracy Score of the KNN model:
0.9743333333333334
The Confusion Matrix for the KNN Model:
[[1840   60]
 [  17 1083]]
The Area Under the Curve (AUC) for the KNN Model:
0.9959787081339713
```

My accuracy scores for the train and test data sets were also high and similar. The accuracy score for the training data set was 0.971 and the accuracy score for the testing data set was 0.966. These values are very similar, and both reflect high accuracy. The training data set could be showing as more accurate due to possible overfitting though.

```
#Model Complexity Curve
neighbors = np.arange(1,25)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
# Loop over different values of k
for i, k in enumerate (neighbors):
    knn = KNeighborsClassifier (n_neighbors=k)
    knn.fit (X_train,y_train)
    train_accuracy[i] = knn.score (X_train, y_train)
    test_accuracy[i] = knn.score (X_test, y_test)
print("K: ", i, "Train Score:", train_accuracy[i], "Test Score:", test_accuracy[i])
```

```
K:  23 Train Score: 0.9715714285714285 Test Score: 0.966
```

```
#Generate plot
plt.title('KNN')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

E2. Results and Implications

After reducing my model to the most significant variables as determined by SelectKBest and p-values, and removing a variable that created high multicollinearity, my k-NN model was able to predict fairly accurately whether a patient would be readmitted. The accuracy of my model is 0.97 and (AUC) is 0.99. This means my model is accurate 97% of the time. The AUC is very close to 1, which indicates that it

performs almost perfectly. While this model appears sufficient to predict readmission rates among patients, as always, the model could benefit from additional data.

E3. Limitation

While my model turned out be very close to completely accurate, there is always room for improvement. As over 63% of patients were NOT readmitted in this data set, additional data and more observations would be beneficial as it may allow for better distribution and perhaps a more improved model. Additionally, the type of model itself, KNN, does have disadvantages, one of which is that it is prone to overfitting.

E4. Course of Action

Based on the accuracy scores of my model, this would be relevant and useful to a health care provider looking to predict readmission for patients. However, I believe this model could be improved with additional data points, such as comorbidities and diagnoses, as well as additional observations. Thus, I would recommend additional data be obtained and then the model be reevaluated for accuracy and consideration be given to using a weighted KNN model or using a different distance metric such as Manhattan or Hamming.

Part VI: Demonstration

F. Panopto Recording

The link to my Panopto video recording is:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6054c185-b978-49e6-9d33-b118013cb473

G. Sources for Third-Party Code

For C3: Western Governors University. (n.d.). *D208 Predictive Modeling Episode 5* [Video]. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=15852089-76a2-4828-8a42-ad3100e8460c

For C3, D1, D2, D3, E1: Western Governors University. (n.d.). *D209 Webinar: Task 1 Expectations - Python* [Video]. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=7329050b-3de4-412a-9ff4-b0790009d02b

H. Sources

For B1: Geeks for Geeks. (N.d.). *K-Nearest Neighbor(KNN) Algorithm*. Geeks for Geeks. https://www.geeksforgeeks.org/k-nearest-neighbours/

For B2: Grant, Peter. (2019). *Introducing k-Nearest Neighbors*. Towards Data Science. https://towardsdatascience.com/introducing-k-nearest-neighbors-7bcd10f938c5

For B3 & D1: Western Governors University. (n.d.). *D209 Webinar: Task 1 Expectations - Python* [Video]. WGU. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=7329050b-3de4-412a-9ff4-b0790009d02b

For C1: Shmueli, Galit. (2015). *Categorical predictors: how many dummies to use in regression vs. k-nearest neighbors*. Business Analytics, Statistics, Teaching. https://www.bzst.com/2015/08/categorical-predictors-how-many-dummies.html

For D1: Western Governors University. (2023). *Dr. Straw's Tips for Success in D209*. Western Governors University. https://westerngovernorsuniversity.sharepoint.com/sites/DataScienceTeam/Shared%20Documents/Forms/AllItems.aspx?isAscending=true&sortField=LinkFilename&id=%2Fsites%2FDataScienceTeam%2FShared%20Documents%2FGraduate%20Team%2FD209%2FStudent%20Facing%20Resources%2FDr%2E%20Straw%20tips%20for%20success%20in%20D209%2Epdf&viewid=417f25b8%2D97d1%2D4fc0%2D8aed%2D5025eade468a&parent=%2Fsites%2FDataScienceTeam%2FShared%20Documents%2FGraduate%20Team%2FD209%2FStudent%20Facing%20Resources

For D2: scikit-learn. (n.d.). *sklearn.neighbors.KNeighborsClassifier.* scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

For D2: Western Governors University. (n.d.). *D209 Webinar: Task 1 Expectations - Python* [Video]. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=7329050b-3de4-412a-9ff4-b0790009d02b

For D2: Medium. (2019). *Precision, Recall*. Medium. https://medium.com/@itisgobiviswa/precision-recall-5c3529a013bb

I. Professional Communication