Western Governors University

D209 Data Mining I – Task 2 Predictive Analysis

Krista Moik

Table of Contents

Part I: Research Question
A1: Proposal of Question

Using the medical_clean data set, my research question is: **Can I use Random Forest Regression to determine which variables have the greatest impact on how many Initial_days a patient is admitted for?** Initial_days is the continuous variable I will be using as my target variable.

A2: Defined Goal

The goal of my data analysis is to create a machine learning algorithm using random forest regression to identify key variables that predict how many days a patient is initially admitted for. Once I develop this model, I would be able to offer real-world recommendations regarding admission duration. Knowing admission duration would provide insights to hospitals such as staffing needs and hospital bed needs and availability.

Part II: Method Justification
B1: Explanation of Prediction Method

I chose to use random forest regression as it can be used with both categorical and numerical data. Random Forest Regression isa a tree-based model that combines predictions from multiple decision trees to produce more accurate and stable predictions. Decision trees are a tree-like flow chart where leaf nodes are created and denotes a feature or attribute, and each branch is a test outcome. This model splits the data into subsets based on the values of the attributes until the best attribute is selected (Geeks for Geeks, n.d.). Ideally, using this method will allow me to make predictions using my independent variables to predict the length of a patient's admission.

B2: Summary of Method Assumption

One assumption of Random Forest Regression is that it assumes the sampling is representative, which could result in misclassification (Western Governors University, n.d.).

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 of the data including matrices. Additionally, plot_tree will be used to visualize one of the decision trees from the Random Forest Regression.

I will import preprocessing and OneHotEncoder from sklearn to scale and re-express certain categorical variables. I will import variance_inflation_factor from statsmodels to check the VIF of my variables.

Finally, I will use several packages from sklearn to complete the analysis as well. SelectKBest, f_regression, and f_classif from sklearn.feature_selection will be imported to help select the best variables for the model. Train_test_split will be imported to split the data into the training and testing datasets. GridSearchCV and make_scorer will be used for hyperparameter tuning. RandomForestRegressor will be used to instantiate random forest regression. Mean_squared_error as MSE, r2_score, mean_absolute_error, and accuracy_score will be used to test the accuracy of the model.

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 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. As random forests are non-parametric I will not need to drop the first column after using onehot_encoder.

C2: Data Set Variables

As my model in task 1 proved to be very accurate, I will be using the same variables. Please see below:

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, cross_val_score
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import classification_report, accuracy_score
```

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())
```

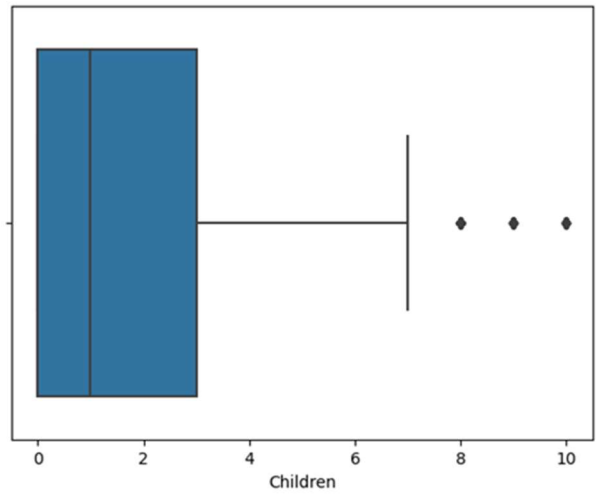
```
False    10000
Name: count, dtype: int64
```

```
#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 the variable with apparent outliers:

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

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

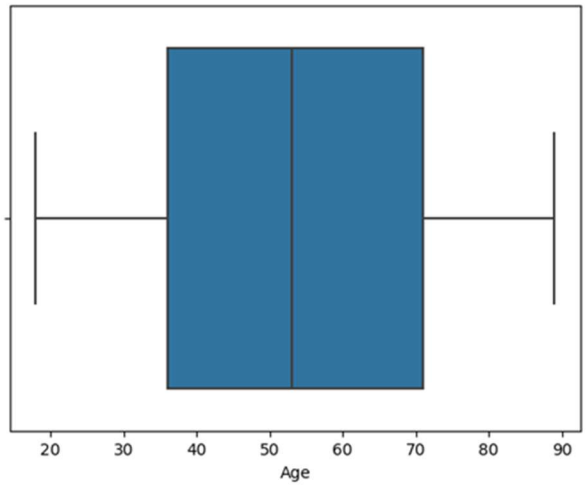



```
df_209.Children.describe()
```

```
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
```

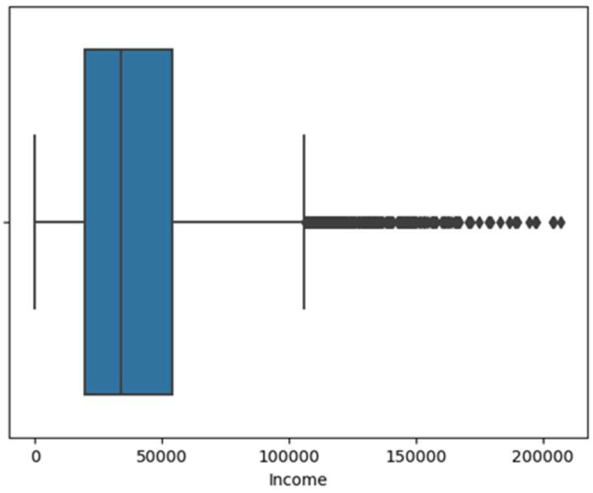
```
sns.boxplot(df_209, x='Age')
```

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




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

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

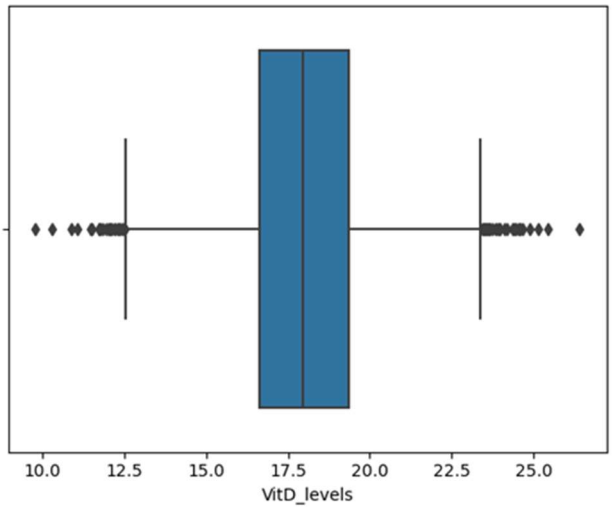



```
df_209.Income.describe()
```

```
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'>
```

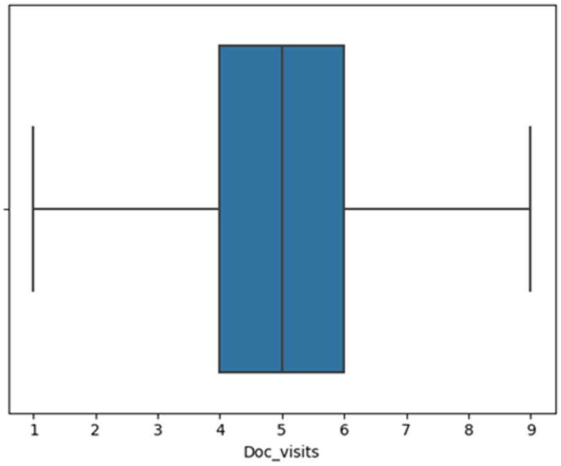



```
df_209.VitD_levels.describe()
```

```
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
```

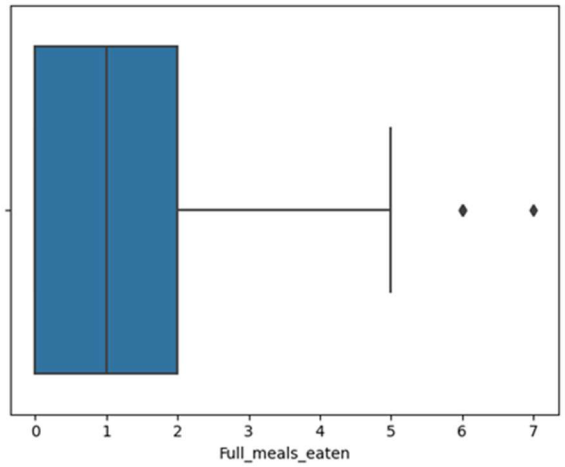
```
sns.boxplot(df_209, x='Doc_visits')
```

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



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

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

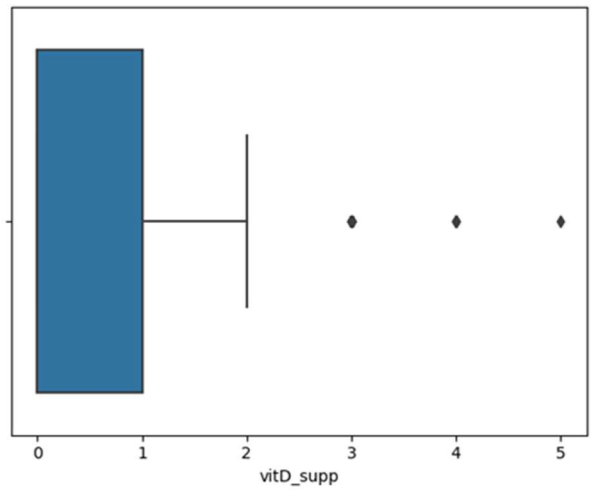



```
df_209.Full_meals_eaten.describe()
```

```
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'>
```

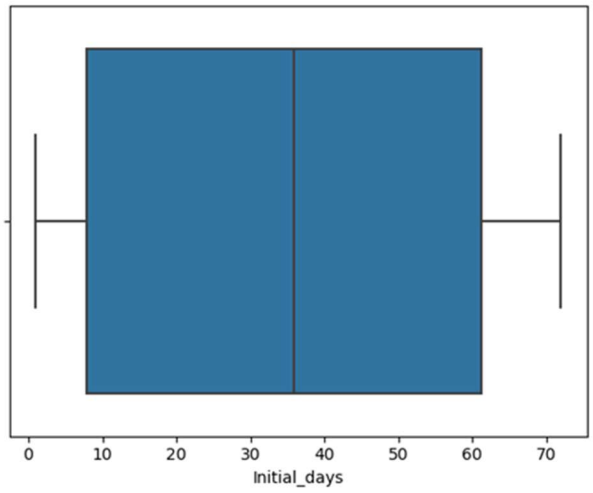



```
df_209.vitD_supp.describe()
```

```
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
```

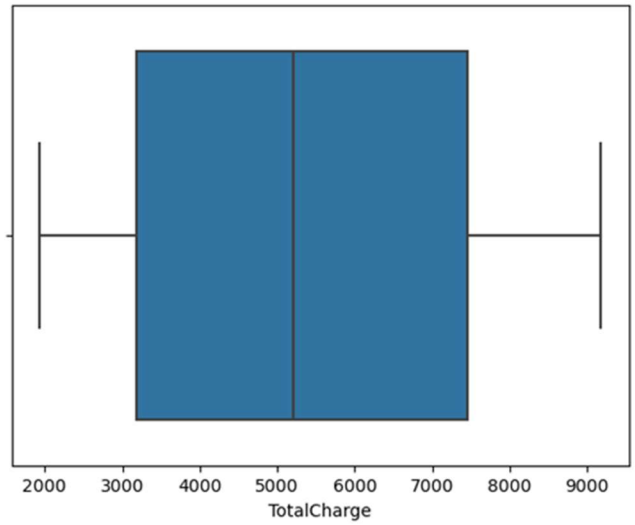
```
sns.boxplot(df_209, x='Initial_days')
```

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



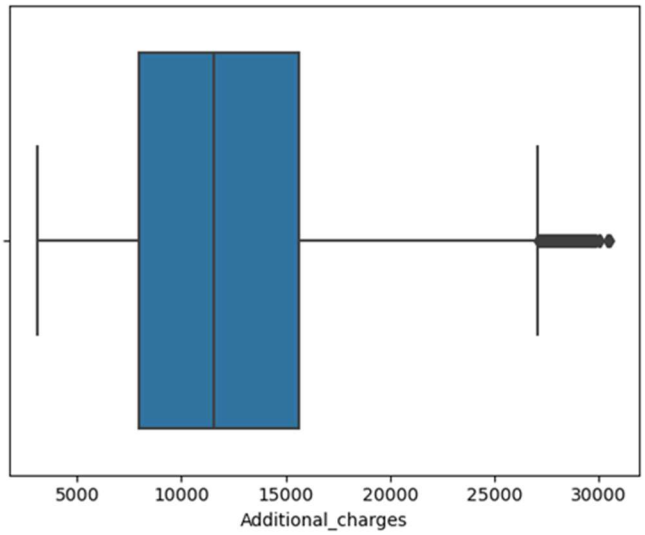

```
sns.boxplot(df_209, x='TotalCharge')
```

```
<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 determined 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.

I then dropped the columns I knew I would not be using in my model using the drop() function:

```
#drop columsn that will not be used
df_209.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'TimeZone', 'Lat', 'Lng', 'Population', 'Item1',
```

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 and dropping the original columns:

```
#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)

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

I then used onehot_encoder to re-express the variables that had more than 2 response types. As decision trees are non-parametric, I dropped the original columns, but I did not drop the first column after re-expression.

```
#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_output` 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

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
```

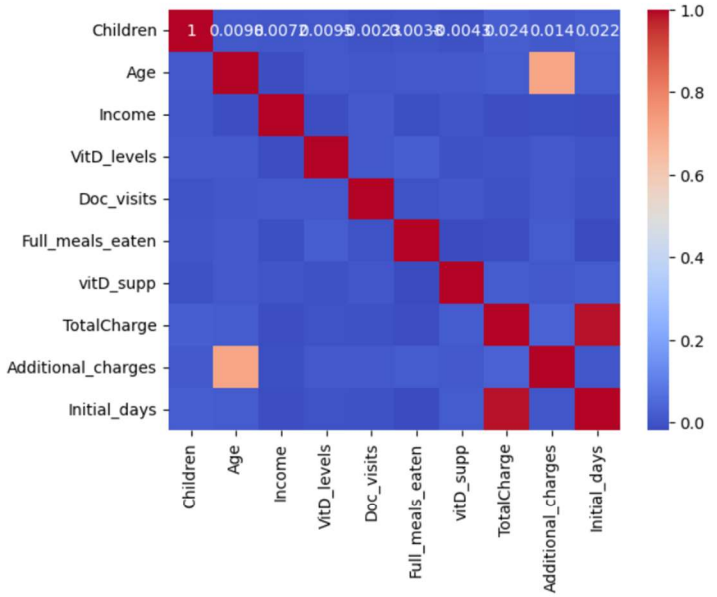
```
 1  Age                                 10000 non-null  int64
 2  Income                              10000 non-null  float64
 3  VitD_levels                         10000 non-null  float64
 4  Doc_visits                          10000 non-null  int64
 5  Full_meals_eaten                    10000 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 created a heatmap showing to review for correlation. Due to the number of variables, I split the heatmap into 2 sections – the first compared the quantitative variable with Initial_days, and the second

compared Initial_days with all of the variables that had to be re-expressed. Based on the quantitative heatmap, there appears to be correlation between Age and AdditionalCharges, and perhaps more obviously, Initial_days and TotalCharge.

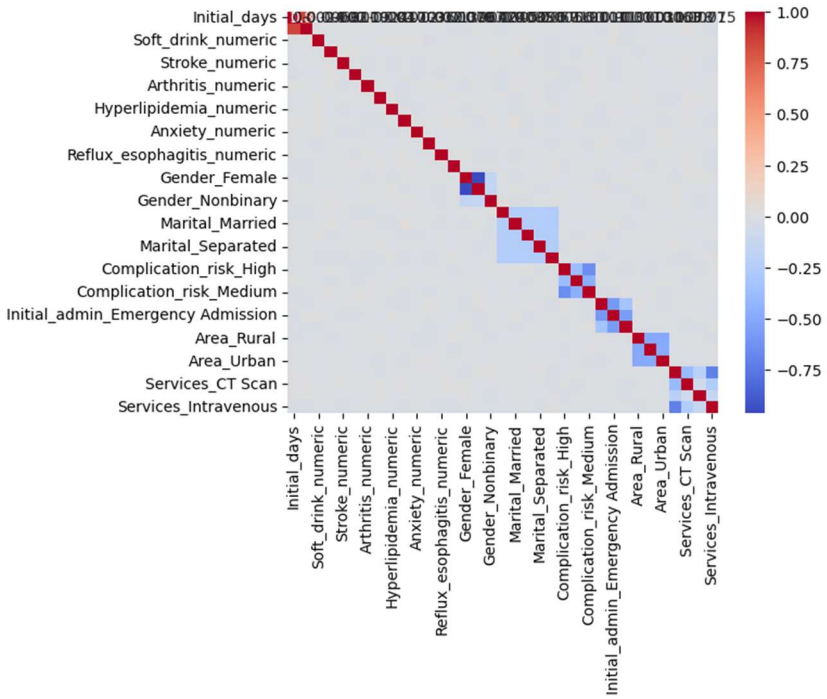
```
#heat map with quantitative data
sns.heatmap(data=df_209[['Children', 'Age', 'Income', 'VitD_levels', 'Doc_visits', 'Full_meals_eaten', 'vitD_supp', 'TotalCharge', 'Additional_charges',
```

```
<Axes: >
```




```
#heat map with qualitative variables
sns.heatmap(data=df_209[['Initial_days', 'ReAdmis_numeric', 'Soft_drink_numeric', 'HighBlood_numeric', 'Stroke_numeric', 'Overweight_numeric', 'Arthritis
```

```
<Axes: >
```



Next, I separated my predictor variables from my dependent variable and then scaled the predictor variables (Western Governors University, n.d.):

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

```
#print features - confirm all other columns present
feature_names = X.columns
print(feature_names)
```

```
Index(['Children', 'Age', 'Income', 'VitD_levels', 'Doc_visits',
       'Full_meals_eaten', 'vitD_supp', 'TotalCharge', 'Additional_charges',
       'ReAdmis_numeric', 'Soft_drink_numeric', 'HighBlood_numeric',
       'Stroke_numeric', 'Overweight_numeric', 'Arthritis_numeric',
       'Diabetes_numeric', 'Hyperlipidemia_numeric', 'BackPain_numeric',
       'Anxiety_numeric', 'Allergic_rhinitis_numeric',
       'Reflux_esophagitis_numeric', 'Asthma_numeric', 'Gender_Female',
       'Gender_Male', 'Gender_Nonbinary', 'Marital_Divorced',
       'Marital_Married', 'Marital_Never Married', 'Marital_Separated',
       'Marital_Widowed', 'Complication_risk_High', 'Complication_risk_Low',
       'Complication_risk_Medium', 'Initial_admin_Elective Admission',
       'Initial_admin_Emergency Admission',
       'Initial_admin_Observation Admission', 'Area_Rural', 'Area_Suburban',
       'Area_Urban', 'Services_Blood Work', 'Services_CT Scan',
       'Services_Intravenous', 'Services_MRI'],
      dtype='object')
```

I then used SelectKBest to identify the most significant variables to my predictor variable and print the p-values, 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
42                          Services_MRI  0.000000
40                     Services_CT Scan  0.000000
35  Initial_admin_Observation Admission  0.000000
34   Initial_admin_Emergency Admission  0.000000
```

33	Initial_admin_Elective Admission	0.000000
29	Marital_Widowed	0.000000
25	Marital_Divorced	0.000000
24	Gender_Nonbinary	0.000000
9	ReAdmis_numeric	0.000000
10	Soft_drink_numeric	0.000000
23	Gender_Male	0.000000
12	Stroke_numeric	0.000000
22	Gender_Female	0.000000
18	Anxiety_numeric	0.000000
15	Diabetes_numeric	0.000000
7	TotalCharge	0.000895
0	Children	0.215158
4	Doc_visits	0.287752
8	Additional_charges	0.348687
39	Services_Blood Work	0.428658
20	Reflux_esophagitis_numeric	0.440197
11	HighBlood_numeric	0.441588
14	Arthritis_numeric	0.463419
16	Hyperlipidemia_numeric	0.475233
30	Complication_risk_High	0.476128
41	Services_Intravenous	0.492195
6	vitD_supp	0.530324
5	Full_meals_eaten	0.600849
26	Marital_Married	0.623205
27	Marital_Never Married	0.629947
2	Income	0.725870
3	VitD_levels	0.729458
17	BackPain_numeric	0.752141
19	Allergic_rhinitis_numeric	0.758061
1	Age	0.771269
36	Area_Rural	0.785536
37	Area_Suburban	0.787981
38	Area_Urban	0.789504
13	Overweight_numeric	0.816905
21	Asthma_numeric	0.817904
31	Complication_risk_Low	0.887188
32	Complication_risk_Medium	0.891094
28	Marital_Separated	0.901265

I obtained the 16 most significant variables with p values less than 0.05: Services_MRI, Services_CT Scan, Initial_admin_Observation Admission, Initial_admin_Emergency Admission, Initial_admin_Elective Admission, Marital_Widowed, Marital_Divorced, Gender_Nonbinary, ReAdmis_numeric, Soft_drink_numeric, Gender_Male, Stroke_numeric, Gender_Female, Anxiety_numeric, Diabetes_numeric, TotalCharge.

Next, I checked for the VIF values of each variable to determine if there was multicollinearity:

```
#check for multicollinearity using VIF
#set independent variable for VIF
X=df_209[['TotalCharge', 'Marital_Divorced', 'Initial_admin_Emergency Admission', 'Services_CT Scan', 'Services_MRI', 'Initial_admin_Elective Admission',
```

```
#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))]
```

```
C:\Users\Kmoik WGU\anaconda3\Lib\site-packages\statsmodels\stats\outliers_influence.py:198: RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
print(vif_df_209)
```

	feature	VIF
0	TotalCharge	3.588237
1	Marital_Divorced	1.068627
2	Initial_admin_Emergency Admission	inf
3	Services_CT Scan	1.007837
4	Services_MRI	1.007156
5	Initial_admin_Elective Admission	inf
6	Marital_Widowed	1.068000
7	Gender_Nonbinary	inf
8	ReAdmis_numeric	3.546527
9	Soft_drink_numeric	1.001768
10	Gender_Male	inf
11	Stroke_numeric	1.000806
12	Gender_Female	inf
13	Anxiety_numeric	1.003595
14	Diabetes_numeric	1.001992
15	Initial_admin_Observation Admission	inf

There are several variables with infinite multicollinearity. I will remove the first from the data set: Initial_admin_Emergency Admission:

```
#will remove the first variable with INF multicollinearity - Initial_admin_Emergency Admission
#set independent variable for VIF
X=df_209[['TotalCharge', 'Marital_Divorced', 'Services_CT Scan', 'Services_MRI', 'Initial_admin_Elective Admission', 'Marital_Widowed', 'Gender_Nonbinary
```

```
#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	3.588237
1	Marital_Divorced	1.068627
2	Services_CT Scan	1.007837
3	Services_MRI	1.007156
4	Initial_admin_Elective Admission	1.142038
5	Marital_Widowed	1.068000
6	Gender_Nonbinary	1.363318
7	ReAdmis_numeric	3.546527
8	Soft_drink_numeric	1.001768
9	Gender_Male	8.818122
10	Stroke_numeric	1.000806
11	Gender_Female	9.266724
12	Anxiety_numeric	1.003595
13	Diabetes_numeric	1.001992
14	Initial_admin_Observation Admission	1.146025

Remaining VIF are all under 10.

No remaining variables have a VIF greater than or equal to 10. My initial model will use the remaining variables: TotalCharge, Marital_Divorced, Services_CT Scan, Services_MRI, Initial_admin_Elective admission, Marital_Widowed, Gender_Nonbinary, ReAdmis_Numeric, Soft_drink_numeric, Gender_Male, Stroke_numeric, Gender_Female, Anxiety_numeric, Diabetes_numeric, Initial_admin_Observation Admission.

I then renamed several of the variables to names without spaces using the rename() function:

```
#renaming statistically significant columns without spaces
df_209.rename(columns={'Services_CT Scan' : 'Services_CTscan', 'Initial_admin_Elective Admission' : 'Initial_admin_Elective', 'Initial_admin_Observation
```

I saved the cleaned data set to a new CSV file:

```
#save cleaned data set
clean_df_209=df_209.copy(deep=True)
```

```
clean_df_209.to_csv('C:/Users/Kmoik WGU/Desktop/KMoikD209_medical.csv')
```

I created a new data set with only the variables I would be using:

```
#create dataset with only statistically significant variables
columns_to_keep=['Services_MRI', 'Services_CTscan', 'Initial_admin_Observe', 'Initial_admin_Elective', 'Marital_Widowed', 'Marital_Divorced', 'Gender_Nor
reduced_209=df_209[columns_to_keep]
```

```
reduced_209.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Services_MRI           10000 non-null  int64
 1   Services_CTscan        10000 non-null  int64
 2   Initial_admin_Observe  10000 non-null  int64
 3   Initial_admin_Elective 10000 non-null  int64
 4   Marital_Widowed        10000 non-null  int64
 5   Marital_Divorced       10000 non-null  int64
 6   Gender_Nonbinary       10000 non-null  int64
 7   ReAdmis_numeric        10000 non-null  int64
 8   Soft_drink_numeric     10000 non-null  int64
 9   Gender_Male            10000 non-null  int64
 10  Gender_Female          10000 non-null  int64
 11  Stroke_numeric         10000 non-null  int64
 12  Anxiety_numeric        10000 non-null  int64
 13  Diabetes_numeric       10000 non-null  int64
 14  TotalCharge            10000 non-null  float64
 15  Initial_days           10000 non-null  float64
dtypes: float64(2), int64(14)
memory usage: 1.2 MB
```

Finally, I updated my X and y variables with the reduced data set:

```
#update X and y with reduced data set
X = reduced_209.drop('Initial_days', axis=1)
# Assign values to y for the dependent variable
y = reduced_209['Initial_days']
```

My next steps for splitting the data will be in section D.

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, I split the data into training and testing sets where 80% of the data was split into a training set, and the remaining 20% was split into the testing set (Western Governors University, n.d.):

```
#update X and y with reduced data set
X = reduced_209.drop('Initial_days', axis=1)
# Assign values to y for the dependent variable
y = reduced_209['Initial_days']
```

```
#Split the data set with an 70/30 split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2, 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 first instantiated and fit the RandomForestRegressor (Western Governors University, n.d.):

```
#Instantiate Random Forest Regressor
rfr=RandomForestRegressor(random_state=15)
```

```
#fit regressor
rfr.fit(X,y)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=15)
```

```
rfr
```

```
RandomForestRegressor
RandomForestRegressor(random_state=15)
```

Then I performed hyperparameter tuning to find the best parameters for the model. I used GridSearchCV and best_params to fit and tune the model and determine the best values for max_depth, n_estimators, and max_features (Western Governors University, n.d.):

```
#Hyperparameter Tuning for RFR
#obtain optimal values for parameters
param_grid = {'n_estimators': [10, 50, 100],
              'max_depth': [8, None],
              'max_features': [2,3,4]}
```

```
#perform grid search with 5 fold cross validation
rfr_cv=GridSearchCV(rfr, param_grid, cv=5)
```

```
#fit the model
rfr_cv.fit(X_train, y_train)
```


```
#check best Parameters used
print('Best Parameters: ',rfr_cv.best_params_)
```

```
Best Parameters:  {'max_depth': None, 'max_features': 4, 'n_estimators': 100}
```

The Grid Search used k-fold cross validation to compare different models to find the best one (Geeks for Geeks, n.d.). This method determined that the best parameters for max depth is none, the best parameter for max_features is 4, and the best parameter for n_estimators is 100.

Next, I scored the training and testing data set models:

```
#Check best score for top performing model
print('Training Score (MSE): ',rfr_cv.best_score_)
print('Training Score (RMSE): ',(rfr_cv.best_score_)**(1/2))
y_train_pred=rfr_cv.predict(X_train)
print('Training - R-squared Score for Model: ', r2_score(y_train, y_train_pred))
```

```
Training Score (MSE):  0.9888686990909278
Training Score (RMSE):  0.994418774506459
Training - R-squared Score for Model:  0.9984536848655955
```

```
#Check Prediction accuracy
y_pred=rfr_cv.predict(X_test)
print('Testing - Mean Squared Error for Model: ', MSE(y_test, y_pred))
print('Testing - Root Mean Squared Error for Model: ', MSE(y_test, y_pred)**(1/2))
print('Testing - R Squared Score for Model: ',r2_score(y_test, y_pred))
```

```
Testing - Mean Squared Error for Model:  7.588416353876204
Testing - Root Mean Squared Error for Model:  2.7547080342345183
Testing - R Squared Score for Model:  0.9890115558335932
```

```
#Accuracy score
print('The Accuracy Score of the Random Forest Regressor:')
print(rfr_cv.score(X_test, y_test))
```

```
The Accuracy Score of the Random Forest Regressor:
0.9890115558335932
```

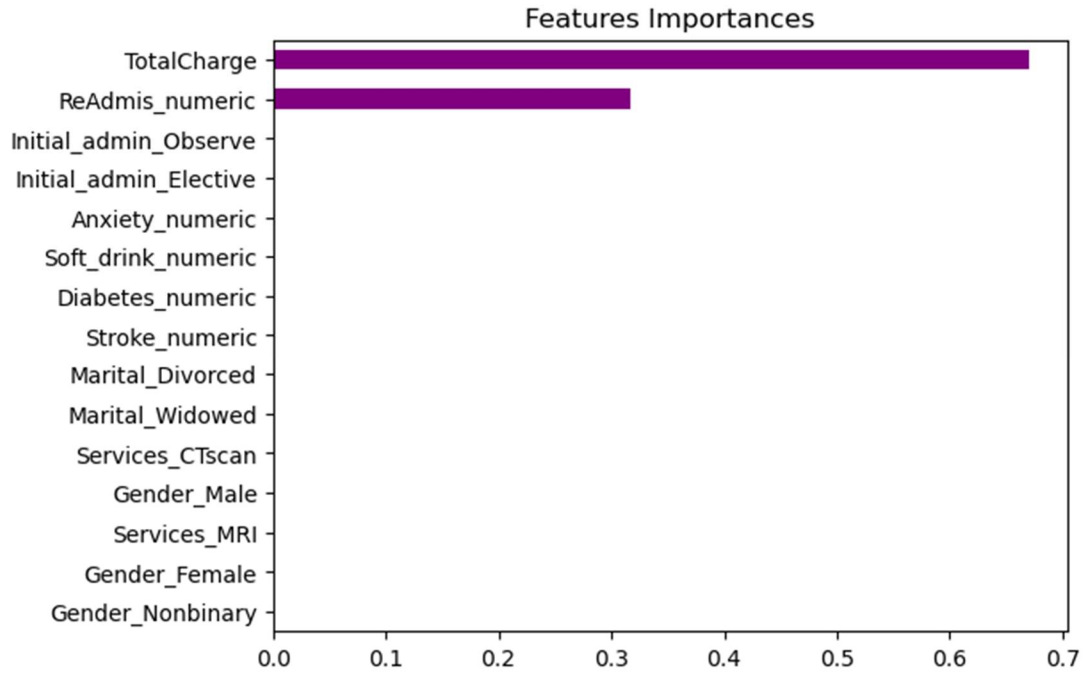
These scores will be discussed further in section E1.

Next, I found and plotted the most important variables (Western Governors University, n.d.):

```
#create pd.Series of features importances
importances=pd.Series(data=rfr_cv.best_estimator_.feature_importances_, index=X_train.columns)
```

```
#sort importances
importances_sorted=importances.sort_values()
```

```
#draw barplot of importances_sorted
importances_sorted.plot(kind='barh', color='purple')
plt.title('Features Importances')
plt.show()
```

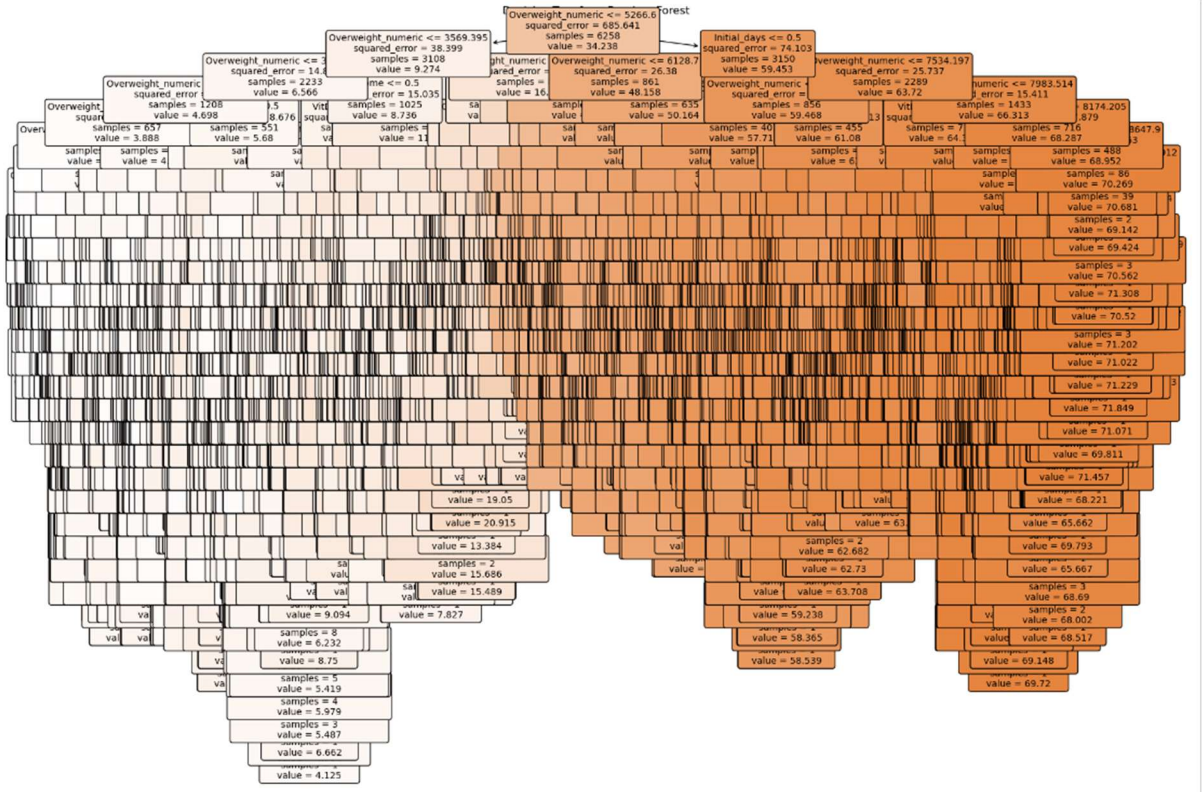


This shows that TotalCharge and ReAdmis_numeric were found to be the most important variables of the data set.

After obtaining the best parameters, I visualized the RandomForestRegressor (Geeks for Geeks.):

Initial RFR:

```
#plot a Decision Tree from Random Forest Regressor
tree_to_plot=rfr.estimators_[0]
plt.figure(figsize=(20,15))
plot_tree(tree_to_plot, feature_names=df_209.columns.tolist(), filled=True, rounded=True, fontsize=10)
plt.title('Decision Tree from Random Forest')
plt.show()
```



This visualization is clearly a forest rather than a single tree and is hard to read. However, I felt it was important to show as it visualizes the forest it creates versus the single tree from decision tree classifier.

D3: Code Execution

A full copy of the code used can be seen in the attached PDF and ipynb files titled KMoikD209Code2.

All code was obtained from WGU course materials or Geeks for Geeks (WGU, n.d., Geeks for Geeks, n.d.).

Code to re-express variables:

```
#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)
```

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

```
#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_output` 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
```

```
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 created heatmaps of the quantitative and qualitative

Next, I used SelectKBest and p-values to obtain the most relevant features:

```
#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)
```

I checked for multicollinearity with VIF:

```
#check for multicollinearity using VIF
#set independent variable for VIF
X=df_209[['TotalCharge', 'Marital_Divorced', 'Initial_admin_Emergency Admission', 'Services_CT Scan', 'Services_MRI', 'Initial_admin_Elective Admission',
```

```
#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))]
```

```
C:\Users\Kmoik WGU\anaconda3\Lib\site-packages\statsmodels\stats\outliers_influence.py:198: RuntimeWarning: divide by zero encountered in scalar di
```

```
print(vif_df_209)
```

```
feature VIF
```

```
#will remove the first variable with INF multicollinearity - Initial_admin_Emergency Admission
#set independent variable for VIF
X=df_209[['TotalCharge', 'Marital_Divorced', 'Services_CT Scan', 'Services_MRI', 'Initial_admin_Elective Admission', 'Marital_Widowed', 'Gender_Nonbinary
```

```
#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)
```

I renamed the columns that needed to be renamed and then reduced the data set to the relevant variables:

```
#renaming statistically significant columns without spaces
df_209.rename(columns={'Services_CT Scan' : 'Services_CTscan', 'Initial_admin_Elective Admission' : 'Initial_admin_Elective', 'Initial_admin_Observation
```

```
#save cleaned data set
clean_df_209=df_209.copy(deep=True)
```

```
clean_df_209.to_csv('C:/Users/Kmoik WGU/Desktop/KMoikD209_medical.csv')
```

```
#create dataset with only statistically significant variables
columns_to_keep=['Services_MRI', 'Services_CTscan', 'Initial_admin_Observe', 'Initial_admin_Elective', 'Marital_Widowed', 'Marital_Divorced', 'Gender_Nor
reduced_209=df_209[columns_to_keep]
```

```
reduced_209.info()
```

The data was split 80/20 where 80% was the training data set and 20% was the testing data set. The split data sets were then saved as separate CSV files:

```
#update X and y with reduced data set
X = reduced_209.drop('Initial_days', axis=1)
# Assign values to y for the dependent variable
y = reduced_209['Initial_days']
```

```
#Split the data set with an 70/30 split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2, 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')
```

I then began creating, fitting, and tuning the model (Western Governors University, n.d.):

```
#Instantiate Random Forest Regressor
rfr=RandomForestRegressor(random_state=15)
```

```
#fit regressor
rfr.fit(X,y)
```

```
<style>#sk-container-id-1 {color: black;background-color: white;}#sk-container-id-1 pre{padding: 0;}#sk-container-id-1 div.sk-toggleable {backgroun ...
```

```
rfr
```

```
<style>#sk-container-id-2 {color: black;background-color: white;}#sk-container-id-2 pre{padding: 0;}#sk-container-id-2 div.sk-toggleable {backgroun ...
```

```
#Hyperparameter Tuning for RFR
#obtain optimal values for parameters
param_grid = {'n_estimators': [10, 50, 100],
              'max_depth': [8, None],
              'max_features': [2,3,4]}
```

```
#perform grid search with 5 fold cross validation
rfr_cv=GridSearchCV(rfr, param_grid, cv=5)
```

```
#fit the model
rfr_cv.fit(X_train, y_train)
```

```
<style>#sk-container-id-3 {color: black;background-color: white;}#sk-container-id-3 pre{padding: 0;}#sk-container-id-3 div.sk-toggleable {backgroun ...
```

```
#check best Parameters used
print('Best Parameters: ',rfr_cv.best_params_)
```

```
Best Parameters: {'max_depth': None, 'max_features': 4, 'n_estimators': 100} ...
```

Then I scored the models (Western Governors University, n.d.):

```
#Check best score for top performing model
print('Training Score (MSE): ',rfr_cv.best_score_)
print('Training Score (RMSE): ',(rfr_cv.best_score_)**(1/2))
y_train_pred=rfr_cv.predict(X_train)
print('Training - R-squared Score for Model: ', r2_score(y_train, y_train_pred))
```

```
Training Score (MSE): 0.9888686990909278 ...
```

```
#Check Prediction accuracy
y_pred=rfr_cv.predict(X_test)
print('Testing - Mean Squared Error for Model: ', MSE(y_test, y_pred))
print('Testing - Root Mean Squared Error for Model: ', MSE(y_test, y_pred)**(1/2))
print('Testing - R Squared Score for Model: ',r2_score(y_test, y_pred))
```

```
Testing - Mean Squared Error for Model: 7.588416353876204 ...
```

```
#Accuracy score
print('The Accuracy Score of the Random Forest Regressor:')
print(rfr_cv.score(X_test, y_test))
```

```
The Accuracy Score of the Random Forest Regressor: ...
```

```
#create pd.Series of features importances
importances=pd.Series(data=rfr_cv.best_estimator_.feature_importances_, index=X_train.columns)
```

I obtained and plotted the most important features:

```
#create pd.Series of features importances
importances=pd.Series(data=rfr_cv.best_estimator_.feature_importances_, index=X_train.columns)
```

•••

```
#sort importances
importances_sorted=importances.sort_values()
```

Finally, I created a visualization of the RandomForestRegression (Geels for Geeks, n.d.):

```
#plot a Decision Tree from Random Forest Regressor
tree_to_plot=rfr.estimators_[0]
plt.figure(figsize=(20,15))
plot_tree(tree_to_plot, feature_names=df_209.columns.tolist(), filled=True, rounded=True, fontsize=10)
plt.title('Decision Tree from Random Forest')
plt.show()
```

Part V: Data Summary and Implications
E1: Accuracy and MSE

I printed the accuracy score, Mean Square Error (MSE), Root Mean Squared Error (RMSE), and R-Squared (R2) scores (Western Governors University, n.d.):

```
#Check best score for top performing model
print('Training Score (MSE): ',rfr_cv.best_score_)
print('Training Score (RMSE): ',(rfr_cv.best_score_)**(1/2))
y_train_pred=rfr_cv.predict(X_train)
print('Training - R-squared Score for Model: ', r2_score(y_train, y_train_pred))
```

```
Training Score (MSE):  0.9888686990909278
Training Score (RMSE):  0.994418774506459
Training - R-squared Score for Model:  0.9984536848655955
```

```
#Check Prediction accuracy
y_pred=rfr_cv.predict(X_test)
print('Testing - Mean Squared Error for Model: ', MSE(y_test, y_pred))
print('Testing - Root Mean Squared Error for Model: ', MSE(y_test, y_pred)**(1/2))
print('Testing - R Squared Score for Model: ',r2_score(y_test, y_pred))
```

```
Testing - Mean Squared Error for Model:  7.588416353876204
Testing - Root Mean Squared Error for Model:  2.7547080342345183
Testing - R Squared Score for Model:  0.9890115558335932
```

```
#Accuracy score
print('The Accuracy Score of the Random Forest Regressor:')
print(rfr_cv.score(X_test, y_test))
```

```
The Accuracy Score of the Random Forest Regressor:
0.9890115558335932
```

The accuracy score is 0.98 indicating the model is highly accurate.

The MSE of the Training model is 0.98 whereas the MSE of the Testing model is 7.58 As it is the average squared difference between predicted and actual values, the Training model appears to be very accurate, especially when compared with the Testing model (Western Governors University, n.d.).

The RMSE of the Training model is 0.99, whereas the RMSE of the Testing model is 2.75 . Similar to the MSE, this indicates that the predicted values are closer to the actual values, and thus the Training model is relatively accurate, especially when compared with the Testing model (Western Governors University, n.d.).

The R-squared score of the Training model is 0.99, and the R2 of the Testing model is 0.98. This indicates that both models are a good fit for the data.

E2: Results and Implications

After reducing my model to the most significant variables as determined by SelectKBest, p-values, and removing variables that created high multicollinearity, the RandomForestRegression supervised machine learning model appeared to be accurate for the Training data set, but less so for the Testing data set. This could indicate that my model is too complex or overfitting is occurring. The accuracy of my model is 0.98, indicating that the model is accurate approximately 98% of the time. As always, my model could always benefit from different and additional data, as well as further tuning. The most important features in the data set are TotalCharge and ReAdmis. As these variables are actually affected by the number of initial days and are not predictive of the number of initial days, it appears that perhaps the variables that affect how many days a patient is initially admitted for may not be present in the current data.

E3: Limitation

RandomForestRegression is supposed to be more accurate than decision trees, however, they are still prone to overfitting and having errors the more complex they become (Geeks for Geeks, n.d.). Additionally, it seems like the data that could predict the number of days a patient is initially admitted for is not in our current data set, which most likely limits the effectiveness of the model.

E4: Course of Action

Based on the accuracy score of my model, on the surface this model would be useful. However, I believe that the variables that actually effect and could help predict the number of days a patient is initially admitted for are not currently present in our data. As such, I would recommend obtaining more data points such as comorbidities and diagnoses, as well as additional observations. Also, there could be concerns of overfitting so I would try additional tuning of the model and perhaps using another method like Lasso Regression.

Part VI: Demonstration
F: Panopto Recording

My Panopto Video can be viewed at this URL:

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=29dca913-9508-4f62-ac2e-b12000f47cad

G: Sources for Third-Party Code

For C3, D1, D2, D3: Western Governors University. (n.d.). *D209 Webinar Splitting Data and Creating Model* [Video]. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=77e354f7-ed96-4886-9a44-b07a00d44a31

For D2, D3: Geeks for Geeks. (n.d.). *Random Forest Regression in Python*. Geeks for Geeks. https://www.geeksforgeeks.org/random-forest-regression-in-python/

For D2: Geeks for Geeks. (n.d.). *Cross Validation in Machine Learning*. Geeks for Geeks. https://www.geeksforgeeks.org/cross-validation-machine-learning/

H: Sources

For B1, D2, E3: Geeks for Geeks. (n.d.). *Random Forest Regression in Python*. Geeks for Geeks. https://www.geeksforgeeks.org/random-forest-regression-in-python/

For B2, B3, C3, E1: Western Governors University. (n.d.). *D209 Webinar Splitting Data and Creating Model* [Video]. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=77e354f7-ed96-4886-9a44-b07a00d44a31

For D2: Geeks for Geeks. (n.d.). *Cross Validation in Machine Learning*. Geeks for Geeks. https://www.geeksforgeeks.org/cross-validation-machine-learning/

I: Professional Communication