**Nefisa Hassen**

**D209 task 2**

**Part I: Research Question**

**A1.** **Summarize one research question.**

Is it possible to forecast patient readmission using a decision tree classification method?

**A2. Define the goals of the data analysis.**

The goal of this analysis is to predict the chances of a patient's readmission to the hospital by using the decision tree classification method.

**Part II: Method Justification**

**B1. Explain how the classification method you chose analyzes the selected data set.**

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It consists of nodes, branches, and leaves, where each node represents a feature, each branch represents a decision rule, and each leaf represents the outcome. Decision trees make decisions by recursively splitting the data based on the most significant feature at each node. For classification, decision trees commonly use entropy or the Gini index to measure impurity and guide the splitting process.

**B2. Summarize one assumption of the chosen classification method.**

* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.

**B3. List the packages or libraries you have chosen for Python or R, and justify how each item on the list supports the analysis.**

* **pandas** Used for data manipulation and analysis. It provides data structures like DataFrame for efficiently handling structured data.
* **numpy** Used for numerical operations in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
* **scipy.stats** Part of the SciPy library, it includes statistical functions for various probability distributions and statistical tests.
* **statsmodels.api** Used for estimating and testing statistical models. It includes functions for regression analysis, hypothesis testing, and other statistical modeling tasks.
* **train\_test\_split**  it's used to split the dataset into training and testing sets, facilitating model evaluation.
* .
* **metrics** Various metrics for evaluating machine learning models, such as accuracy, precision, recall, and F1-score.
* **matplotlib.pyplot** A plotting library used for creating visualizations, including charts and graphs.
* **accuracy\_score, classification\_report, confusion\_matrix** metrics and tools for evaluating classification models, providing detailed reports and confusion matrices.
* **variance\_inflation\_factor** Used for detecting multicollinearity in regression analysis by calculating the variance inflation factor for each feature.
* **SelectKBest, f\_classif** 
  + **SelectKBest** is a method for selecting the top k features.
  + **f\_classif** is a scoring function used for feature selection based on analysis of variance (ANOVA).
* **GridSearchCV** Used for hyperparameter tuning through an exhaustive search over specified parameter values for an estimator, helping to find the best parameters for a model.
* **ecisionTreeClassifier:** used for building decision tree models for classification tasks.
* **r2\_score:** used for evaluating the goodness of fit of a regression model.
* **mean\_squared\_error:** used to calculate the mean squared error (MSE) between predicted and true values.

**Part III: Data Preparation**

**C1. Describe one data preprocessing goal relevant to the classification method from part A1**.

One key objective in processing the data pertinent to my research question is identifying and addressing outliers. Outliers can exert a substantial impact on the dataset, underscoring the necessity for a thorough assessment and appropriate treatment of the data. For my analysis, I used medical data and applied the **std()** function to assess outliers across all columns. Notably, outliers were detected in variables such as TotalCharge, Additional\_charges, VitD\_levels, and Initial\_days. To address the outliers in these four columns, the z-score method was utilized for treatment.

**C2. Identify the initial data set variables that you will use to perform the analysis for the classification question from part A1 and classify *each* variable as numeric or categorical.**

**#Numeric variables**

Initial\_days float64

TotalCharge float64

Additional\_charges float64

VitD\_levels float64

Doc\_visits int64

Full\_meals\_eaten int64

vitD\_supp int64

**#catagorial variables**

ReAdmis object

Soft\_drink object

Initial\_admin object

HighBlood object

Stroke object

Complication\_risk object

Overweight object

Arthritis object

Diabetes object

Hyperlipidemia object

BackPain object

Anxiety object

Allergic\_rhinitis object

Reflux\_esophagitis object

Asthma object

Services object

**C3. Explain each of the steps used to prepare the data for the analysis. Identify the code segment for each step.**

The analysis is based on the medical data file. Upon importing the data, a thorough check for missing values was conducted, and there is no missing data entries across all columns. The **std()** function was used to assess outliers, and outliers were identified on variables such as TotalCharge, Additional\_charges, VitD\_levels, and Initial\_days. These outliers were then treated using the z-score method.

Dummy variables were generated to convert categorical variables into numeric values. Additionally, one-hot encoding was applied to represent categorical variables with more than two options, such as 'Services,' 'Complication\_risk,' and 'Initial\_admin.'

Certain columns deemed irrelevant to answering the research question were removed. These include 'CaseOrder,' 'Customer\_id,' 'Interaction,' 'UID,' 'City,' 'State,' 'County,' 'Zip,' 'Lat,' 'Lng,' 'Population,' 'Area,' and 'TimeZone.'

Using the selectKBest function the variables that are most significant were extracted. In addition, using the VIF function, multicollinearity among the variables was assessed and treated. Lastly, StandardScaler() function was used to scale and transform the data.

1

**import** pandas **as** pd

2

**import** numpy **as** np

3

**from** scipy **import** stats

4

**import** statsmodels.api **as** sm

5

**from** sklearn.model\_selection **import** train\_test\_split

6

**from** sklearn.neighbors **import** KNeighborsClassifier

7

**from** sklearn **import** metrics

8

**from** sklearn.metrics **import** roc\_curve,auc

9

**import** matplotlib.pyplot **as** plt

10

**from** sklearn.feature\_selection **import** f\_classif

11

**from** sklearn.metrics **import** accuracy\_score, classification\_report, confusion\_matrix

12

**from** sklearn.preprocessing **import** StandardScaler

13

**from** statsmodels.stats.outliers\_influence **import** variance\_inflation\_factor

14

**from** sklearn.feature\_selection **import** SelectKBest, f\_classif

15

**from** sklearn.model\_selection **import** GridSearchCV

16

**import** matplotlib.pyplot **as** plt

17

**from** sklearn.metrics **import** roc\_curve, auc

18

**from** sklearn **import** metrics

19

**import** matplotlib.pyplot **as** plt

20

**import** seaborn **as** sns

In [98]:

1

med\_data **=** pd.read\_csv('medical\_clean.csv') *#importaing the data*

In [99]:

1

med\_data.columns *#looking at the varialbles*

Out[99]:

Index(['CaseOrder', 'Customer\_id', 'Interaction', 'UID', 'City', 'State',  
 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job',  
 'Children', 'Age', 'Income', 'Marital', 'Gender', 'ReAdmis',  
 'VitD\_levels', 'Doc\_visits', 'Full\_meals\_eaten', 'vitD\_supp',  
 'Soft\_drink', 'Initial\_admin', 'HighBlood', 'Stroke',  
 'Complication\_risk', 'Overweight', 'Arthritis', 'Diabetes',  
 'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic\_rhinitis',  
 'Reflux\_esophagitis', 'Asthma', 'Services', 'Initial\_days',  
 'TotalCharge', 'Additional\_charges', 'Item1', 'Item2', 'Item3', 'Item4',  
 'Item5', 'Item6', 'Item7', 'Item8'],  
 dtype='object')

In [100]:

1

*#deleting the columns that are irrelevant to answering the research question.*

In [101]:

1

med\_data **=** med\_data.drop(columns**=**['CaseOrder', 'Customer\_id', 'Interaction', 'UID','Job', 'Children', 'Age', 'Income', 'Marital', 'Gender', 'City', 'State',

2

'County', 'Zip', 'Lat', 'Lng','Population', 'Area', 'TimeZone' , 'Item1', 'Item2', 'Item3', 'Item4',

3

'Item5', 'Item6', 'Item7', 'Item8' ])

In [102]:

1

med\_data.columns *#checking to bee the columns are deleted*

Out[102]:

Index(['ReAdmis', 'VitD\_levels', 'Doc\_visits', 'Full\_meals\_eaten', 'vitD\_supp',  
 'Soft\_drink', 'Initial\_admin', 'HighBlood', 'Stroke',  
 'Complication\_risk', 'Overweight', 'Arthritis', 'Diabetes',  
 'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic\_rhinitis',  
 'Reflux\_esophagitis', 'Asthma', 'Services', 'Initial\_days',  
 'TotalCharge', 'Additional\_charges'],  
 dtype='object')

In [103]:

1

med\_data.dtypes *#looking at data types for each variables*

Out[103]:

ReAdmis object  
VitD\_levels float64  
Doc\_visits int64  
Full\_meals\_eaten int64  
vitD\_supp int64  
Soft\_drink object  
Initial\_admin object  
HighBlood object  
Stroke object  
Complication\_risk object  
Overweight object  
Arthritis object  
Diabetes object  
Hyperlipidemia object  
BackPain object  
Anxiety object  
Allergic\_rhinitis object  
Reflux\_esophagitis object  
Asthma object  
Services object  
Initial\_days float64  
TotalCharge float64  
Additional\_charges float64  
dtype: object

In [104]:

1

med\_data.isnull().sum() *# cheking for missing data*

Out[104]:

ReAdmis 0  
VitD\_levels 0  
Doc\_visits 0  
Full\_meals\_eaten 0  
vitD\_supp 0  
Soft\_drink 0  
Initial\_admin 0  
HighBlood 0  
Stroke 0  
Complication\_risk 0  
Overweight 0  
Arthritis 0  
Diabetes 0  
Hyperlipidemia 0  
BackPain 0  
Anxiety 0  
Allergic\_rhinitis 0  
Reflux\_esophagitis 0  
Asthma 0  
Services 0  
Initial\_days 0  
TotalCharge 0  
Additional\_charges 0  
dtype: int64

In [105]:

1

*#creating dummy variables*

In [106]:

1

categorical\_columns **=** [

2

'ReAdmis', 'VitD\_levels', 'Doc\_visits', 'Full\_meals\_eaten', 'vitD\_supp',

3

'Soft\_drink', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis', 'Diabetes',

4

'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic\_rhinitis',

5

'Reflux\_esophagitis', 'Asthma'

6

]

7

8

**for** column **in** categorical\_columns:

9

med\_data[column] **=** med\_data[column].astype('category').cat.codes

In [107]:

1

*#one hot encoding for catagorical variables that have more than 2 options*

In [108]:

1

med\_data **=** pd.get\_dummies(med\_data, columns**=**['Services','Complication\_risk','Initial\_admin'], drop\_first**=True**)

In [109]:

1

*# checking for outliers.*

2

med\_data.std()

Out[109]:

ReAdmis 0.481983  
VitD\_levels 2878.750357  
Doc\_visits 1.045734  
Full\_meals\_eaten 1.008117  
vitD\_supp 0.628505  
Soft\_drink 0.437279  
HighBlood 0.491674  
Stroke 0.399494  
Overweight 0.454062  
Arthritis 0.479258  
Diabetes 0.445930  
Hyperlipidemia 0.472777  
BackPain 0.492112  
Anxiety 0.467076  
Allergic\_rhinitis 0.488681  
Reflux\_esophagitis 0.492486  
Asthma 0.453460  
Initial\_days 26.309341  
TotalCharge 2180.393838  
Additional\_charges 6542.601544  
Services\_CT Scan 0.327879  
Services\_Intravenous 0.463738  
Services\_MRI 0.191206  
Complication\_risk\_Low 0.409097  
Complication\_risk\_Medium 0.497687  
Initial\_admin\_Emergency Admission 0.499989  
Initial\_admin\_Observation Admission 0.429276  
dtype: float64

In [110]:

1

*#treating outliers with zscore method.*

In [111]:

1

med\_data ['TotalCharge\_z']**=**stats.zscore(med\_data['TotalCharge'])

In [112]:

1

med\_data\_outliers\_TotalCharge **=** med\_data.query('TotalCharge\_z > 3 | TotalCharge\_z< -3')

In [113]:

1

med\_data ['Additional\_charges\_z']**=**stats.zscore(med\_data['Additional\_charges'])

In [114]:

1

med\_data\_outliers\_Additional\_charges **=** med\_data.query('Additional\_charges\_z > 3 | Additional\_charges\_z< -3')

In [115]:

1

med\_data ['Initial\_days\_z'] **=** stats.zscore(med\_data['Initial\_days'])

In [116]:

1

med\_data\_outliers\_Initial\_days **=** med\_data.query('Initial\_days\_z > 3 | Initial\_days\_z< -3')

In [117]:

1

med\_data ['VitD\_levels\_z'] **=** stats.zscore(med\_data['VitD\_levels'])

In [118]:

1

med\_data\_outliers\_VitD\_levels **=** med\_data.query('VitD\_levels\_z > 3 | VitD\_levels\_z< -3')

In [119]:

1

med\_data.std() *# checking to see if outliers were treated.*

Out[119]:

ReAdmis 0.481983  
VitD\_levels 2878.750357  
Doc\_visits 1.045734  
Full\_meals\_eaten 1.008117  
vitD\_supp 0.628505  
Soft\_drink 0.437279  
HighBlood 0.491674  
Stroke 0.399494  
Overweight 0.454062  
Arthritis 0.479258  
Diabetes 0.445930  
Hyperlipidemia 0.472777  
BackPain 0.492112  
Anxiety 0.467076  
Allergic\_rhinitis 0.488681  
Reflux\_esophagitis 0.492486  
Asthma 0.453460  
Initial\_days 26.309341  
TotalCharge 2180.393838  
Additional\_charges 6542.601544  
Services\_CT Scan 0.327879  
Services\_Intravenous 0.463738  
Services\_MRI 0.191206  
Complication\_risk\_Low 0.409097  
Complication\_risk\_Medium 0.497687  
Initial\_admin\_Emergency Admission 0.499989  
Initial\_admin\_Observation Admission 0.429276  
TotalCharge\_z 1.000050  
Additional\_charges\_z 1.000050  
Initial\_days\_z 1.000050  
VitD\_levels\_z 1.000050  
dtype: float64

In [120]:

1

med\_data **=** med\_data.astype(int)

In [121]:

1

med\_data['VitD\_levels'].describe()

Out[121]:

count 10000.000000  
mean 4985.993300  
std 2878.750357  
min 0.000000  
25% 2493.750000  
50% 4984.500000  
75% 7478.250000  
max 9975.000000  
Name: VitD\_levels, dtype: float64

In [122]:

1

med\_data['Doc\_visits'].describe()

Out[122]:

count 10000.000000  
mean 4.012200  
std 1.045734  
min 0.000000  
25% 3.000000  
50% 4.000000  
75% 5.000000  
max 8.000000  
Name: Doc\_visits, dtype: float64

In [123]:

1

med\_data['Full\_meals\_eaten'].describe()

Out[123]:

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

In [124]:

1

med\_data['vitD\_supp'].value\_counts()

Out[124]:

vitD\_supp  
0 6702  
1 2684  
2 544  
3 64  
4 5  
5 1  
Name: count, dtype: int64

In [125]:

1

med\_data['Initial\_admin\_Emergency Admission'].value\_counts()

Out[125]:

Initial\_admin\_Emergency Admission  
1 5060  
0 4940  
Name: count, dtype: int64

In [126]:

1

med\_data['Initial\_admin\_Observation Admission'].value\_counts()

Out[126]:

Initial\_admin\_Observation Admission  
0 7564  
1 2436  
Name: count, dtype: int64

In [127]:

1

med\_data['Complication\_risk\_Low'].value\_counts()

Out[127]:

Complication\_risk\_Low  
0 7875  
1 2125  
Name: count, dtype: int64

In [128]:

1

med\_data['Complication\_risk\_Medium'].value\_counts()

Out[128]:

Complication\_risk\_Medium  
0 5483  
1 4517  
Name: count, dtype: int64

In [129]:

1

med\_data['Services\_CT Scan'].value\_counts()

Out[129]:

Services\_CT Scan  
0 8775  
1 1225  
Name: count, dtype: int64

In [130]:

1

med\_data['Services\_Intravenous'].value\_counts()

Out[130]:

Services\_Intravenous  
0 6870  
1 3130  
Name: count, dtype: int64

In [131]:

1

med\_data['Services\_MRI'].value\_counts()

Out[131]:

Services\_MRI  
0 9620  
1 380  
Name: count, dtype: int64

In [132]:

1

med\_data['Overweight'].value\_counts()

Out[132]:

Overweight  
1 7094  
0 2906  
Name: count, dtype: int64

In [133]:

1

med\_data['Arthritis'].value\_counts()

Out[133]:

Arthritis  
0 6426  
1 3574  
Name: count, dtype: int64

In [134]:

1

med\_data['Diabetes'].value\_counts()

Out[134]:

Diabetes  
0 7262  
1 2738  
Name: count, dtype: int64

In [135]:

1

med\_data['Hyperlipidemia'].value\_counts()

Out[135]:

Hyperlipidemia  
0 6628  
1 3372  
Name: count, dtype: int64

In [136]:

1

med\_data['BackPain'].value\_counts()

Out[136]:

BackPain  
0 5886  
1 4114  
Name: count, dtype: int64

In [137]:

1

med\_data['Anxiety'].value\_counts()

Out[137]:

Anxiety  
0 6785  
1 3215  
Name: count, dtype: int64

In [138]:

1

med\_data['Allergic\_rhinitis'].value\_counts()

Out[138]:

Allergic\_rhinitis  
0 6059  
1 3941  
Name: count, dtype: int64

In [139]:

1

med\_data['Reflux\_esophagitis'].value\_counts()

Out[139]:

Reflux\_esophagitis  
0 5865  
1 4135  
Name: count, dtype: int64

In [140]:

1

med\_data['Asthma'].value\_counts()

Out[140]:

Asthma  
0 7107  
1 2893  
Name: count, dtype: int64

In [141]:

1

med\_data['TotalCharge'].describe()

Out[141]:

count 10000.000000  
mean 5311.673500  
std 2180.391406  
min 1938.000000  
25% 3179.000000  
50% 5213.500000  
75% 7459.250000  
max 9180.000000  
Name: TotalCharge, dtype: float64

In [142]:

1

med\_data['Additional\_charges'].describe()

Out[142]:

count 10000.000000  
mean 12934.032300  
std 6542.600277  
min 3125.000000  
25% 7985.750000  
50% 11573.500000  
75% 15626.000000  
max 30566.000000  
Name: Additional\_charges, dtype: float64

In [143]:

1

med\_data['Asthma'].value\_counts()

Out[143]:

Asthma  
0 7107  
1 2893  
Name: count, dtype: int64

In [144]:

1

1

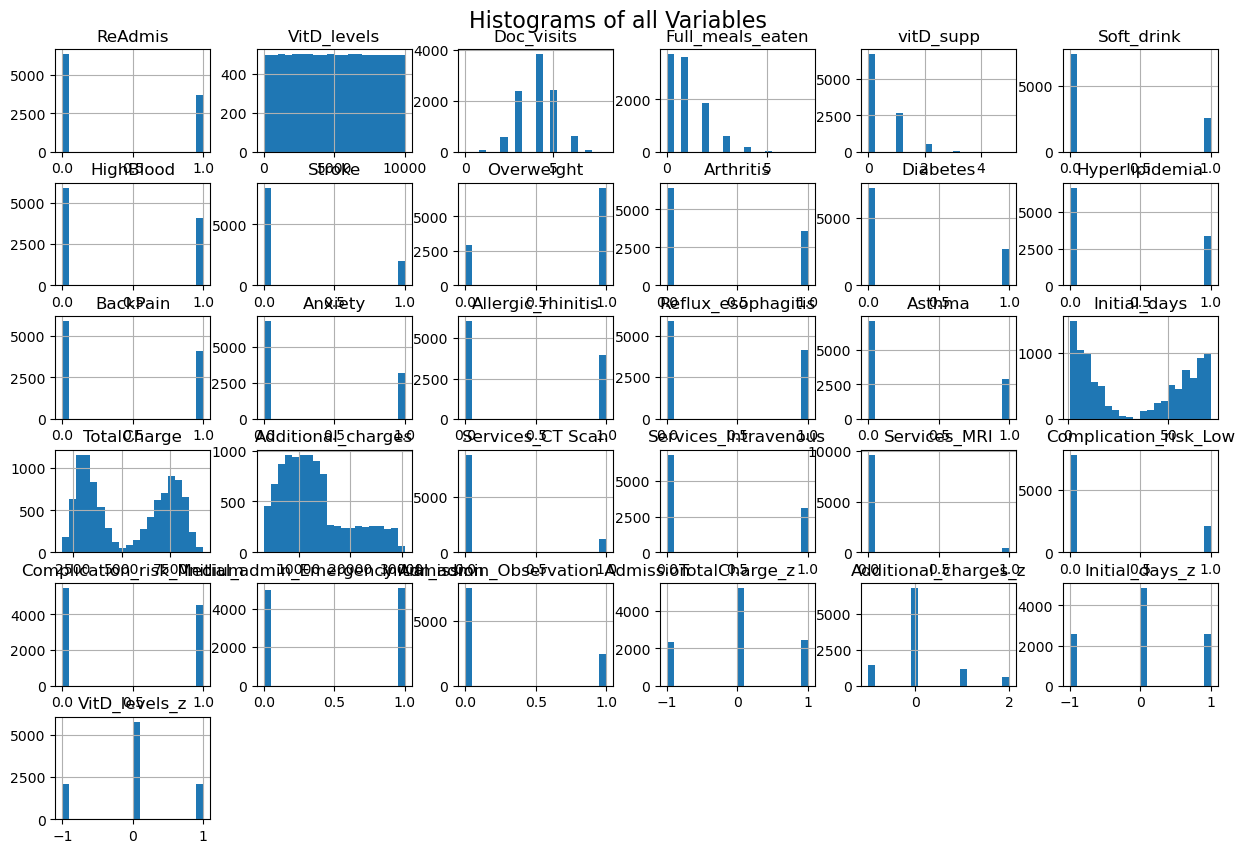
med\_data.hist(bins**=**20, figsize**=**(15, 10))

2

plt.suptitle('Histograms of all Variables', x**=**0.5, y**=**0.92, fontsize**=**16)

3

plt.show()



In [146]:

1

correlation\_matrix **=** med\_data.corr()

In [147]:

1

plt.figure(figsize**=**(15, 12))

2

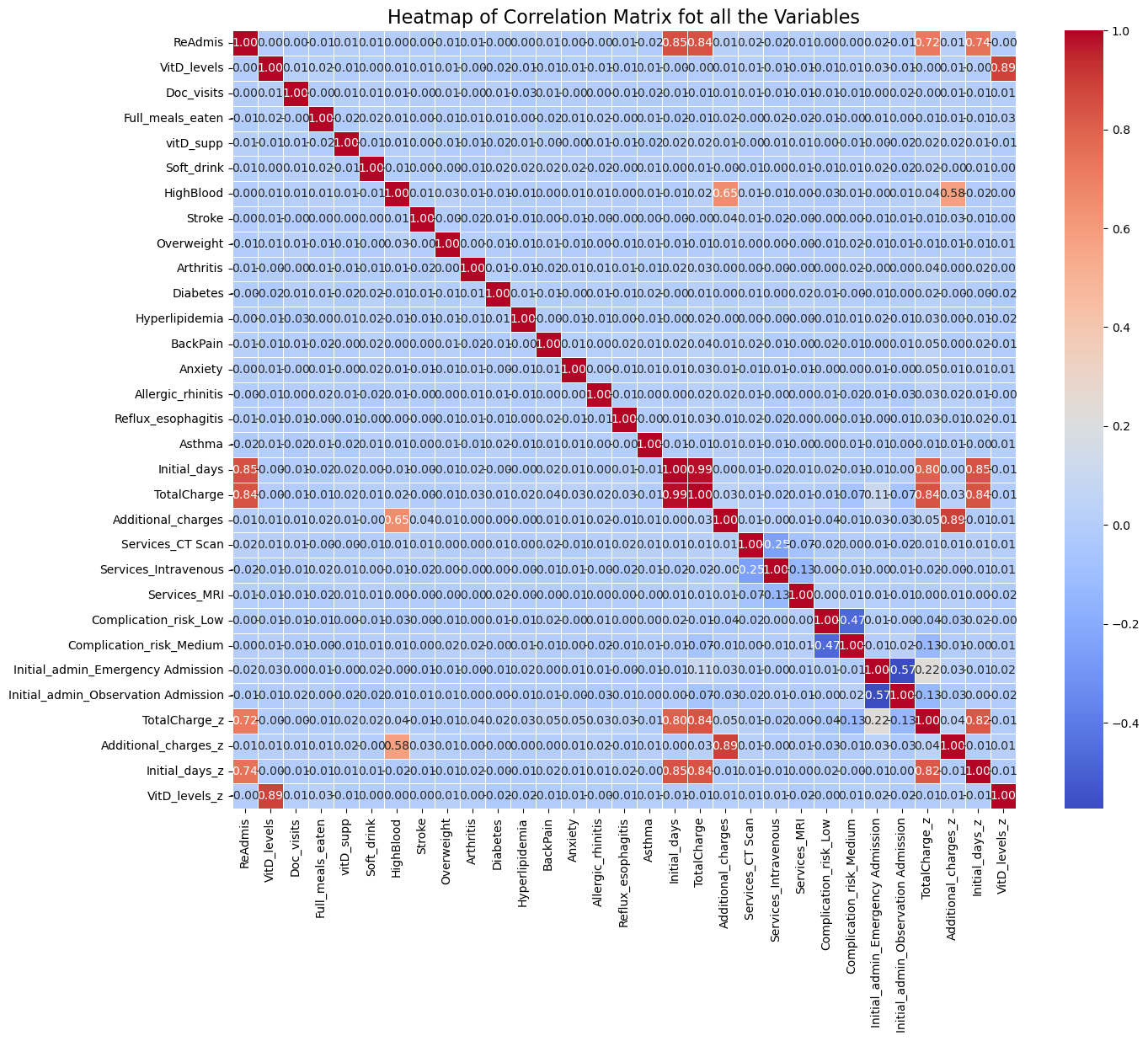
sns.heatmap(correlation\_matrix, cmap**=**'coolwarm', annot**=True**, fmt**=**".2f", linewidths**=**.5)

3

plt.title('Heatmap of Correlation Matrix fot all the Variables', fontsize**=**16)

4

plt.show()



**In [148]:**

**1**

***#create a dataframe with all training data except the target column***

2

X **=** med\_data.drop(columns **=** ['ReAdmis'])

In [149]:

1

y **=** med\_data['ReAdmis']

In [150]:

1

skbest **=** SelectKBest(score\_func**=**f\_classif, k**=**'all')

2

3

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

4

5

X\_new.shape

Out[150]:

(10000, 30)

In [151]:

1

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

2

p\_values\_filtered **=** p\_values[p\_values['p\_value'] **<** 0.05]

3

features\_to\_keep **=** p\_values['Feature']

4

[p\_values['p\_value']**<**.05]

5

print(features\_to\_keep)

17 TotalCharge  
26 TotalCharge\_z  
28 Initial\_days\_z  
16 Initial\_days  
19 Services\_CT Scan  
20 Services\_Intravenous  
24 Initial\_admin\_Emergency Admission  
15 Asthma  
18 Additional\_charges  
11 BackPain  
2 Full\_meals\_eaten  
27 Additional\_charges\_z  
25 Initial\_admin\_Observation Admission  
3 vitD\_supp  
21 Services\_MRI  
7 Overweight  
4 Soft\_drink  
8 Arthritis  
14 Reflux\_esophagitis  
13 Allergic\_rhinitis  
10 Hyperlipidemia  
9 Diabetes  
0 VitD\_levels  
23 Complication\_risk\_Medium  
12 Anxiety  
5 HighBlood  
22 Complication\_risk\_Low  
29 VitD\_levels\_z  
6 Stroke  
1 Doc\_visits  
Name: Feature, dtype: object

In [152]:

1

vif\_data **=** pd.DataFrame()

2

vif\_data["feature"] **=** X.columns

3

vif\_data["VIF"] **=** [variance\_inflation\_factor(X.values, i)

4

**for** i **in** range(len(X.columns))]

5

6

print(vif\_data)

feature VIF  
0 VitD\_levels 0.043040  
1 Doc\_visits 15.755627  
2 Full\_meals\_eaten 1.995299  
3 vitD\_supp 1.408329  
4 Soft\_drink 1.352657  
5 HighBlood 2.993996  
6 Stroke 1.253073  
7 Overweight 3.450154  
8 Arthritis 1.630867  
9 Diabetes 1.429457  
10 Hyperlipidemia 1.614547  
11 BackPain 1.802039  
12 Anxiety 1.557342  
13 Allergic\_rhinitis 1.715333  
14 Reflux\_esophagitis 1.776081  
15 Asthma 1.411303  
16 Initial\_days 201.528203  
17 TotalCharge -1.352117  
18 Additional\_charges 0.009030  
19 Services\_CT Scan 1.235275  
20 Services\_Intravenous 1.599818  
21 Services\_MRI 1.074618  
22 Complication\_risk\_Low 2.086661  
23 Complication\_risk\_Medium 3.416689  
24 Initial\_admin\_Emergency Admission 6.459238  
25 Initial\_admin\_Observation Admission 1.980673  
26 TotalCharge\_z 5.269968  
27 Additional\_charges\_z 5.059327  
28 Initial\_days\_z 4.952117  
29 VitD\_levels\_z 4.739807

In [153]:

1

Med\_data\_updated **=** med\_data.drop(['Doc\_visits', 'Initial\_days', 'Initial\_admin\_Emergency Admission'], axis**=**1)

In [154]:

1

**def** vif\_scores(Med\_data\_updated):

2

VIF\_Scores **=** pd.DataFrame()

3

VIF\_Scores["Independant Features"] **=** Med\_data\_updated.columns

4

VIF\_Scores["VIF Scores"] **=** [variance\_inflation\_factor(Med\_data\_updated.values,i) **for** i **in** range(Med\_data\_updated.shape[1])]

5

**return** VIF\_Scores

In [155]:

1

print(vif\_scores(Med\_data\_updated)) *# multicoliarity trated*

Independant Features VIF Scores  
0 ReAdmis 5.419176  
1 VitD\_levels 0.034840  
2 Full\_meals\_eaten 1.967070  
3 vitD\_supp 1.397318  
4 Soft\_drink 1.345188  
5 HighBlood 2.957988  
6 Stroke 1.250661  
7 Overweight 3.334725  
8 Arthritis 1.561420  
9 Diabetes 1.380878  
10 Hyperlipidemia 1.512694  
11 BackPain 1.713057  
12 Anxiety 1.483810  
13 Allergic\_rhinitis 1.646916  
14 Reflux\_esophagitis 1.697538  
15 Asthma 1.404283  
16 TotalCharge -0.085716  
17 Additional\_charges 0.007062  
18 Services\_CT Scan 1.229988  
19 Services\_Intravenous 1.578562  
20 Services\_MRI 1.072805  
21 Complication\_risk\_Low 1.698786  
22 Complication\_risk\_Medium 2.557470  
23 Initial\_admin\_Observation Admission 1.409598  
24 TotalCharge\_z 4.503472  
25 Additional\_charges\_z 4.172219  
26 Initial\_days\_z 4.201656  
27 VitD\_levels\_z 4.038069

In [156]:

1

scale **=** StandardScaler()

In [157]:

1

x\_scaled **=** med\_data[['ReAdmis', 'VitD\_levels', 'Doc\_visits', 'Full\_meals\_eaten', 'vitD\_supp',

2

'Soft\_drink', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis',

3

'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety',

4

'Allergic\_rhinitis', 'Reflux\_esophagitis', 'Asthma', 'Initial\_days',

5

'TotalCharge', 'Additional\_charges', 'Services\_CT Scan',

6

'Services\_Intravenous', 'Services\_MRI', 'Complication\_risk\_Low',

7

'Complication\_risk\_Medium', 'Initial\_admin\_Emergency Admission',

8

'Initial\_admin\_Observation Admission', 'TotalCharge\_z',

9

'Additional\_charges\_z', 'Initial\_days\_z', 'VitD\_levels\_z']]

In [158]:

1

scaled**=** scale.fit\_transform(x\_scaled)

In [159]:

1

print(x\_scaled)

ReAdmis VitD\_levels Doc\_visits Full\_meals\_eaten vitD\_supp \  
0 0 7175 5 0 0   
1 0 6885 3 2 1   
2 0 5219 3 1 0   
3 0 2422 3 1 0   
4 0 3978 4 0 2   
... ... ... ... ... ...   
9995 0 3132 3 2 1   
9996 1 5460 4 0 0   
9997 1 3389 3 2 0   
9998 1 8284 4 2 1   
9999 1 5872 4 0 1   
  
 Soft\_drink HighBlood Stroke Overweight Arthritis ... \  
0 0 1 0 0 1 ...   
1 0 1 0 1 0 ...   
2 0 1 0 1 0 ...   
3 0 0 1 0 1 ...   
4 1 0 0 0 0 ...   
... ... ... ... ... ... ...   
9995 0 1 0 0 0 ...   
9996 0 1 0 1 1 ...   
9997 1 1 0 1 0 ...   
9998 0 0 0 1 0 ...   
9999 0 0 0 1 1 ...   
  
 Services\_Intravenous Services\_MRI Complication\_risk\_Low \  
0 0 0 0   
1 1 0 0   
2 0 0 0   
3 0 0 0   
4 0 0 1   
... ... ... ...   
9995 1 0 0   
9996 0 0 0   
9997 1 0 0   
9998 0 0 0   
9999 0 0 1   
  
 Complication\_risk\_Medium Initial\_admin\_Emergency Admission \  
0 1 1   
1 0 1   
2 1 0   
3 1 0   
4 0 0   
... ... ...   
9995 1 1   
9996 1 0   
9997 0 0   
9998 1 1   
9999 0 0   
  
 Initial\_admin\_Observation Admission TotalCharge\_z \  
0 0 0   
1 0 0   
2 0 -1   
3 0 -1   
4 0 -1   
... ... ...   
9995 0 0   
9996 0 1   
9997 0 1   
9998 0 1   
9999 1 1   
  
 Additional\_charges\_z Initial\_days\_z VitD\_levels\_z   
0 0 0 0   
1 0 0 0   
2 0 -1 0   
3 0 -1 0   
4 -1 -1 0   
... ... ... ...   
9995 0 0 0   
9996 2 1 0   
9997 0 1 0   
9998 0 1 1   
9999 0 1 0   
  
[10000 rows x 31 columns]

In [160]:

1

2

***#split dataset into train and test data***

3

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**1, stratify**=**y)

In [161]:

1

***#C4. Provide a copy of the cleaned data set.***

In [162]:

1

X\_train.to\_csv('X\_train.csv',index**=False**)

2

X\_test.to\_csv('X\_test.csv', index**=False**)

3

In [163]:

1

y\_train\_df **=** pd.DataFrame({'ReAdmis': y\_train})

2

y\_train\_df.to\_csv('y\_train.csv', index**=False**)

In [164]:

1

y\_test\_df **=** pd.DataFrame({'ReAdmis': y\_test})

2

y\_test\_df.to\_csv('y\_test.csv', index**=False**)

**Part IV: Analysis**

**D1. Split the data into training and test data sets and provide the file(s).**

See the above code on C4

**D2.Describe the analysis technique you used to appropriately analyze the data.**

Randomized Search Cross Validation is used to find the best hyperparameters for a Decision Tree Classifier in a machine learning model. The hyperparameters are specified in the **params\_dist** dictionary, which includes options for the maximum depth of the tree (**max\_depth**), the maximum number of leaf nodes (**max\_leaf\_nodes**), the minimum number of samples required to be at a leaf node (**min\_samples\_leaf**), and the minimum number of samples required to split an internal node (**min\_samples\_split**).

The **DecisionTreeClassifier** is used as **dtr**, serving as the base model for which hyperparameters will be optimized. Then, a **RandomizedSearchCV** object named **random\_search** is created. This object performs a randomized search over the hyperparameter space defined by **params\_dist** using 5-fold cross-validation (**cv=5**). The model is then fitted to the training data (**X\_train**, **y\_train**) using the **fit** method.

The best parameters are displayed as **{'min\_samples\_split': 50, 'min\_samples\_leaf': 50, 'max\_leaf\_nodes': 3000, 'max\_depth': 4}**.

D3. Provide the code used to perform the prediction analysis from part D2.

**import** matplotlib.pyplot **as** plt

3

**from** sklearn.tree **import** DecisionTreeClassifier

4

**from** sklearn.metrics **import** accuracy\_score

5

**from** sklearn.metrics **import** r2\_score

6

**from** sklearn.metrics **import** mean\_squared\_error

7

**from** sklearn.model\_selection **import** RandomizedSearchCV

In [73]:

1

params\_dist **=** {

2

'max\_depth': [4, 6, 8, 10, 12, 14, 16],

3

'max\_leaf\_nodes': [1000, 2000, 3000],

4

'min\_samples\_leaf': [20, 30, 40, 50],

5

'min\_samples\_split': [30, 40, 50]

6

}

7

In [74]:

1

dtr **=** DecisionTreeClassifier()

2

In [76]:

1

random\_search **=** RandomizedSearchCV(dtr, params\_dist, cv**=**5)

2

In [77]:

1

random\_search.fit(X\_train, y\_train)

2

Out[77]:

**RandomizedSearchCV**[**?**](https://scikit-learn.org/1.4/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)**i**

**estimator: DecisionTreeClassifier**

DecisionTreeClassifier[?](https://scikit-learn.org/1.4/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

In [92]:

1

print("Best Parameters:", random\_search.best\_params\_)

2

Best Parameters: {'min\_samples\_split': 50, 'min\_samples\_leaf': 50, 'max\_leaf\_nodes': 3000, 'max\_depth': 4}

**from** sklearn.metrics **import** confusion\_matrix, classification\_report

2

In [135]:

1

y\_pred **=** random\_search.predict(X\_test)

In [136]:

1

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

2

print("Confusion Matrix:")

3

print(conf\_matrix)

Confusion Matrix:  
[[1243 23]  
 [ 22 712]]

TP: 1243 FP: 23 FN: 22 TN: 712

In [137]:

1

class\_report **=** classification\_report(y\_test, y\_pred)

2

print("Classification Report:")

3

print(class\_report)

Classification Report:  
 precision recall f1-score support  
  
 0 0.98 0.98 0.98 1266  
 1 0.97 0.97 0.97 734  
  
 accuracy 0.98 2000  
 macro avg 0.98 0.98 0.98 2000  
weighted avg 0.98 0.98 0.98 2000

The model shows high precision, recall, and F1-score for both classes, indicating good performance. The overall accuracy is also high at 98%, which suggests that the model is doing well in classifying the test data

In [138]:

1

*#D3.*

In [139]:

1

med\_data.to\_csv('MSDA209\_PA\_task2\_D3.cvs')

**Part V: Data Summary and Implications**

**E1. Explain the accuracy and the mean squared error (MSE) of your prediction model.**

1

print("Training Score (MSE):", random\_search.best\_score\_)

2

3

print ("Training Score (RMSE):",(random\_search.best\_score\_)**\*\***(1**/**2))

4

5

y\_train\_pred **=** random\_search.predict(X\_train)

6

print ("Train - R squared score for the model:",r2\_score(y\_train,y\_train\_pred))

7

Training Score (MSE): 0.9775  
Training Score (RMSE): 0.9886859966642595  
Train - R squared score for the model: 0.9122817343865355

In [94]:

1

*# prediction on the testing data*

In [95]:

1

y\_pred **=** random\_search.predict(X\_test)

In [96]:

1

mse\_test **=** mean\_squared\_error(y\_test, y\_pred)

2

print("Testing - Mean squared error for the model:",mse\_test )

3

4

rmse\_test **=** mse\_test **\*\*** 0.5

5

6

print("Testing -Root Mean Squared error for the model:",rmse\_test)

7

8

print("Testing - R - squared score for the model:", r2\_score(y\_test,y\_pred))

9

Testing - Mean squared error for the model: 0.0225  
Testing -Root Mean Squared error for the model: 0.15  
Testing - R - squared score for the model: 0.9031470743959605

**E2. Discuss the results and implications of your prediction analysis.**

**Training Data Results:** The high R-squared score on the training data (0.9123) suggests that the Decision Tree Classifier captures a significant portion of the variance in the hospital readmission rates within the training dataset. The low MSE and RMSE values further indicate a good fit to the training data.

**Testing Data Results:** The testing results mirror the training results, indicating that the Decision Tree Classifier generalizes well to unseen data. The high R-squared score on the testing data (0.9031) suggests that the model effectively explains the variability in hospital readmission rates.

Based on the provided code, the Decision Tree Classifier exhibits strong predictive capabilities for hospital readmission rates. The model performs well on both the training and testing datasets, suggesting its potential utility in predicting readmission rates for new patient cases.

**E3. Discuss one limitation of your data analysis.**

The effectiveness of the model heavily relies on the quality and representativeness of the dataset. If the data is incomplete, biased, or contains outliers, the model's performance may be affected.

**E4. Recommend a course of action for the real-world organizational situation from part A1.**

Based on the analysis I recommend that the organization can leverage the promising results of the Decision Tree Classifier to make informed decisions regarding hospital readmission rates. I highly recommend Regular monitoring, collaboration with healthcare professionals to incorporate feedback to refine and improve the model will contribute to the successful implementation and utilization of the predictive model in a real-world healthcare setting.

**Part VI: Demonstration**

**F**

**G.**

Analytics Vidhya. (n.d.). Understanding Decision Tree (Part III: Data Preparation). Medium. Retrieved from <https://medium.com/analytics-vidhya/understanding-decision-tree-3591922690a6>

Khan, F. (2018, November 28). An Exhaustive Guide to Classification Using Decision Trees. Towards Data Science. Retrieved from <https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f>

scikit-learn. (n.d.). Decision Trees. Retrieved from <https://scikit-learn.org/stable/modules/tree.html.>