**Nefisa Hassen**

**D209 task1**

**Part I: Research Question**

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

Is it possible to forecast patient readmission using chronic conditions, lifestyle factors, and healthcare services?

The k-nearest neighbor (KNN) classification method will be used to answer the research question.

**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 looking at chronic conditions, lifestyle factors, and healthcare service types.

**Part II: Method Justification**

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

The Nearest Neighbor rule consistently exhibits robust performance, avoiding the need for predefined assumptions about the distributions of training examples. It operates on a training set that encompasses both positive and negative cases. When classifying a new sample, the method calculates its distance to the nearest training case, and the sign of that point dictates the sample's classification. Extending this idea, the k-NN classifier incorporates the nearest k points, determining the classification based on the majority's sign.

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

The closer the two given points are to each other, the more related and similar they are.

**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.
* **KNeighborsClassifier** it provides an implementation of the k-Nearest Neighbors algorithm for classification.
* **metrics** Various metrics for evaluating machine learning models, such as accuracy, precision, recall, and F1-score.
* **roc\_curve** Used for Receiver Operating Characteristic (ROC) curve analysis, a tool for evaluating the performance of classification algorithms.
* **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.

**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 .

**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 [2]:

1

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

In [3]:

1

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

Out[3]:

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 [4]:

1

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

In [5]:

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 [6]:

1

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

Out[6]:

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 [7]:

1

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

Out[7]:

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 [8]:

1

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

Out[8]:

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 [9]:

1

*#creating dummy variables*

In [10]:

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 [11]:

1

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

In [12]:

1

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

In [13]:

1

*# checking for outliers.*

2

med\_data.std()

Out[13]:

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 [14]:

1

*#treating outliers with zscore method.*

In [15]:

1

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

In [16]:

1

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

In [17]:

1

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

In [18]:

1

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

In [19]:

1

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

In [20]:

1

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

In [21]:

1

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

In [22]:

1

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

In [23]:

1

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

Out[23]:

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 [24]:

1

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

In [25]:

1

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

Out[25]:

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 [26]:

1

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

Out[26]:

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 [27]:

1

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

Out[27]:

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 [28]:

1

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

Out[28]:

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

In [29]:

1

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

Out[29]:

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

In [30]:

1

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

Out[30]:

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

In [31]:

1

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

Out[31]:

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

In [32]:

1

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

Out[32]:

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

In [33]:

1

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

Out[33]:

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

In [34]:

1

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

Out[34]:

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

In [35]:

1

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

Out[35]:

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

In [36]:

1

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

Out[36]:

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

In [37]:

1

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

Out[37]:

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

In [38]:

1

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

Out[38]:

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

In [39]:

1

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

Out[39]:

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

In [40]:

1

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

Out[40]:

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

In [41]:

1

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

Out[41]:

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

In [42]:

1

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

Out[42]:

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

In [43]:

1

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

Out[43]:

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

In [44]:

1

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

Out[44]:

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

In [45]:

1

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

Out[45]:

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 [46]:

1

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

Out[46]:

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 [47]:

1

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

Out[47]:

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

In [48]:

1

*#Describtibve ananlysis*

2

med\_data.describe()

Out[48]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **ReAdmis** | **VitD\_levels** | **Doc\_visits** | **Full\_meals\_eaten** | **vitD\_supp** | **Soft\_drink** | **HighBlood** | **Stroke** | **Overweight** | **Arthritis** | **...** | **Services\_Intravenous** | **Services\_MRI** | **Complication\_risk\_Low** | **Complication\_risk\_Medium** | **Initial\_admin\_Emergency Admission** | **Initial\_admin\_Observation Admission** | **TotalCharge\_z** | **Additional\_charges\_z** | **Initial\_days\_z** | **VitD\_levels\_z** |
| **count** | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | ... | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 0.366900 | 4985.993300 | 4.012200 | 1.001400 | 0.398900 | 0.257500 | 0.409000 | 0.199300 | 0.709400 | 0.357400 | ... | 0.313000 | 0.038000 | 0.212500 | 0.451700 | 0.506000 | 0.243600 | 0.010100 | 0.098600 | -0.000800 | 0.000100 |
| **std** | 0.481983 | 2878.750357 | 1.045734 | 1.008117 | 0.628505 | 0.437279 | 0.491674 | 0.399494 | 0.454062 | 0.479258 | ... | 0.463738 | 0.191206 | 0.409097 | 0.497687 | 0.499989 | 0.429276 | 0.690974 | 0.704079 | 0.717113 | 0.650032 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 |
| **25%** | 0.000000 | 2493.750000 | 3.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -1.000000 | 0.000000 |
| **50%** | 0.000000 | 4984.500000 | 4.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 1.000000 | 7478.250000 | 5.000000 | 2.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | ... | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 |
| **max** | 1.000000 | 9975.000000 | 8.000000 | 7.000000 | 5.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | ... | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 2.000000 | 1.000000 | 1.000000 |

8 rows × 31 columns

In [49]

1

**#Histograms of all Variables**

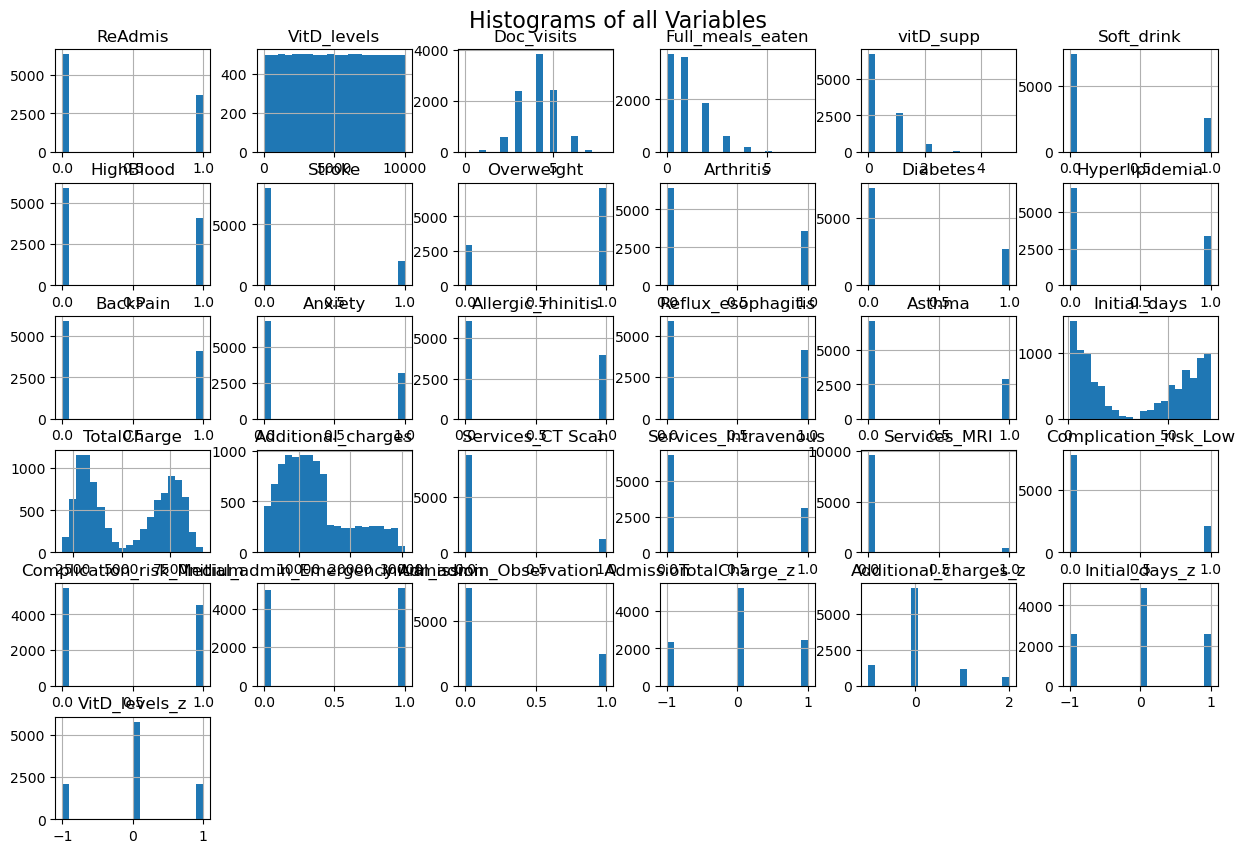
med\_data.hist(bins**=**20, figsize**=**(15, 10))

2

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

3

plt.show()



In [50]:

1

**# Heatmap of all the variables**

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

In [51]:

1

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

2

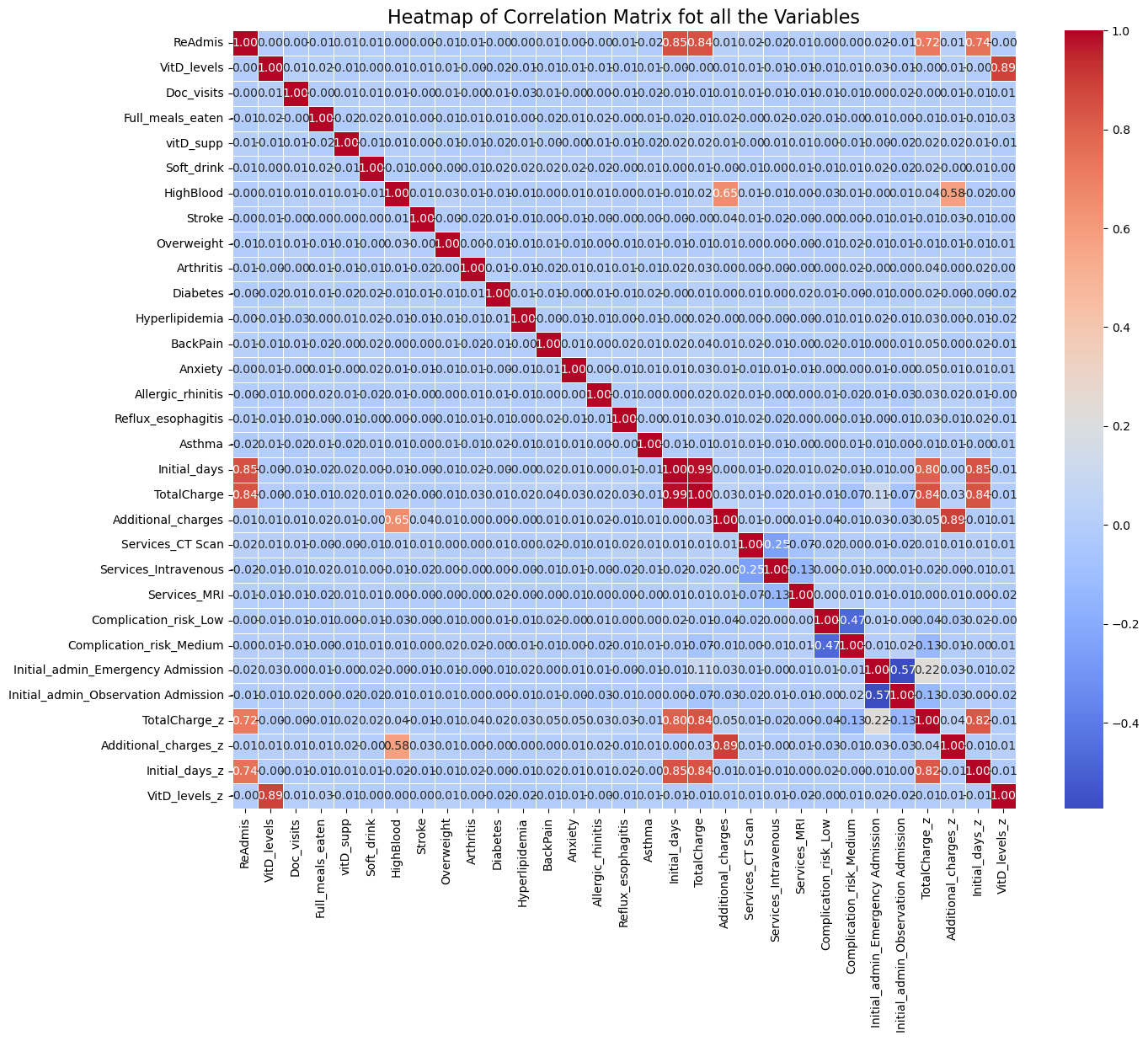
sns.heatmap(correlation\_matrix, cmap**=**'coolwarm', annot**=True**, fmt**=**".2f", linewidths**=**.5)

3

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

4

plt.show()



In [52]:

1

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

2

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

In [53]:

1

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

In [54]:

1

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

2

3

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

4

5

X\_new.shape

Out[54]:

(10000, 30)

In [55]:

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 [56]:

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 [57]:

1

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

In [58]:

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 [59]:

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 [60]:

1

scale **=** StandardScaler()

In [61]:

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 [62]:

1

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

In [63]:

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 [64]:

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 [65]:

1

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

In [66]:

1

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

2

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

3

In [67]:

1

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

2

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

In [68]:

1

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

2

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

# **Part IV: Analysis**

In [69]:

1

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

In [70]:

1

*#the above code shows the data split into test and train sets*

In [71]:

1

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

First, define a parameter grid for n\_neighbors ranging from 1 to 50 and create an instance of the KNeighborsClassifier. Secondly, create a GridSearchCV object using the KNN classifier. Fit the GridSearchCV model to the training data, which involves training the KNN classifier with different values of n\_neighbors and using cross-validation to find the best hyperparameter. By using the best\_params function, it is determined that the optimal number of neighbors to consider for classification is 5. The best\_score indicates that the highest cross-validated accuracy is approximately 94.83%.

In [72]:

1

param\_grid **=** {'n\_neighbors': np.arange(1, 51)}

In [73]:

1

knn **=** KNeighborsClassifier()

In [74]:

1

knn\_cv **=** GridSearchCV(knn, param\_grid, cv**=**5)

2

In [75]:

1

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

Out[75]:

**GridSearchCV**

**estimator: KNeighborsClassifier**

KNeighborsClassifier

In [76]:

1

knn\_cv.best\_params\_

Out[76]:

{'n\_neighbors': 5}

In [77]:

1

knn\_cv.best\_score\_

Out[77]:

0.94825

In [78]:

1

knn **=** KNeighborsClassifier(n\_neighbors **=**5)

In [79]:

1

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

Out[79]:

KNeighborsClassifier

KNeighborsClassifier()

In [ ]:

1

In [80]:

1

y\_pred **=** knn.predict(X\_test)

2

y\_pred

Out[80]:

array([0, 0, 1, ..., 1, 1, 1])

In [81]:

1

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

2

final\_matrix

Out[81]:

array([[1184, 82],  
 [ 38, 696]], dtype=int64)

In [82]:

1

sns.heatmap(final\_matrix, annot**=True**, fmt**=**"d", cmap**=**"Blues")

2

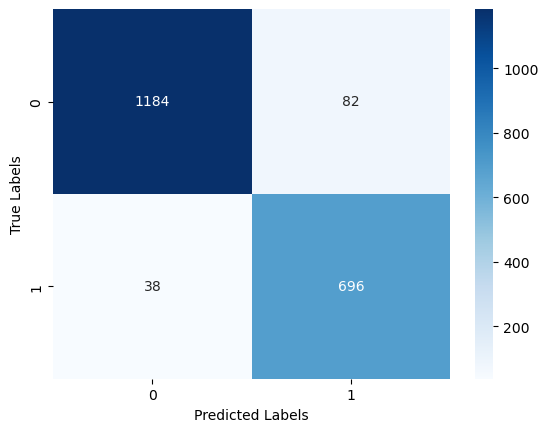
plt.xlabel("Predicted Labels")

3

plt.ylabel("True Labels")

4

plt.show()



In [ ]:

1

In [83]:

1

(TP) 696 - The model correctly predicted that 696 instances belong to class 1 (positive).

2

(TN) 1184 - The model correctly predicted that 1184 instances belong to class 0 (negative).

3

(FP)82 - The model incorrectly predicted that 82 instances belong to class 1.

4

(FN) 38 - The model incorrectly predicted that 38 instances belong to class 0.

In [83]:

1

*#D3. Provide the code used to perform the classification analysis from part D2.*

In [ ]:

1

med\_data.to\_csv('MSDA209\_PA\_task1\_D2.cvs')

# **Part V: Data Summary and Implications**

In [ ]:

1

*#E1.Explain the accuracy and the area under the curve (AUC) of your classification model.*

Based on the provided code, the model demonstrates an accuracy of 94% on the test set. Additionally, the Area Under the Curve (AUC) is calculated to be 0.9417, a value near 1. This suggests strong overall performance across various threshold values, affirming the model's ability to effectively distinguish between positive and negative instances.

In [84]:

1

*# Calculating accuracy*

2

accuracy **=** accuracy\_score(y\_test, y\_pred)

3

print(f'Accuracy: {accuracy:.4f}')

4

Accuracy: 0.9400

In [85]:

1

2

*# Classification Report*

3

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

4

print('\nClassification Report:')

5

print(classification\_rep)

Classification Report:  
 precision recall f1-score support  
  
 0 0.97 0.94 0.95 1266  
 1 0.89 0.95 0.92 734  
  
 accuracy 0.94 2000  
 macro avg 0.93 0.94 0.94 2000  
weighted avg 0.94 0.94 0.94 2000

In [86]:

1

2

*# Calculating AUC*

3

accuracy\_new **=** metrics.roc\_auc\_score(y\_test, y\_pred)

4

print(f'AUC: {accuracy\_new:.4f}')

AUC: 0.9417

In [88]:

1

fpr, tpr, \_ **=** roc\_curve(y\_test, y\_pred)

2

roc\_auc\_new **=** auc(fpr, tpr)

In [89]:

1

*# Ploting ROC curve*

2

plt.figure()

3

plt.plot(fpr, tpr, label**=**'ROC curve (area = %0.2f)' **%** roc\_auc\_new)

4

plt.plot([0, 1], [0, 1], 'k--')

5

plt.xlabel('False Positive Rate')

6

plt.ylabel('True Positive Rate')

7

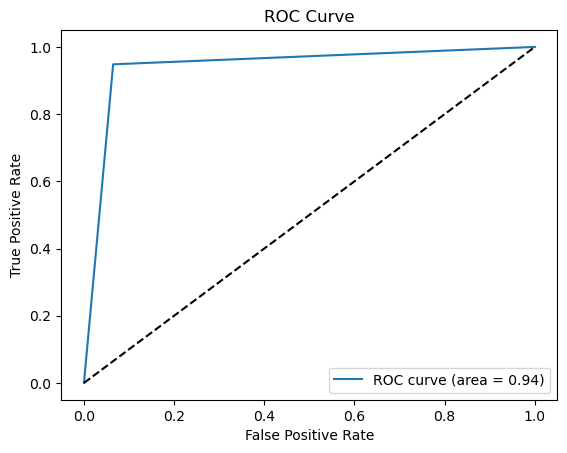
plt.title('ROC Curve')

8

plt.legend(loc**=**"lower right")

9

plt.show()



In [ ]:

1

*#E2. Discuss the results and implications of your classification analysis.*

The model demonstrates high accuracy (94.825%) in a cross-validated setting for predicting patient readmission based on chronic conditions, lifestyle factors, and healthcare services a patient receives. The examination of the classification report further explains the model's notable achievements in terms of high precision, recall, and F1-score. Additionally, the analysis includes the computation of the Area Under the Curve (AUC), a metric ranging from 0 to 1. The calculated AUC of 0.9417, nearing the upper limit, signifies superior model performance. This proximity to 1 implies that the model demonstrates a robust true positive rate and maintains a low false positive rate across diverse threshold settings, highlighting its effectiveness in distinguishing between positive and negative instances.

In [ ]:

1

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

One possible limitation for the model is data quality and bias. The model's performance heavily relies on the quality

and representativeness of the training data. If the data used for training is biased or incomplete, the model may

inherit these biases and make inaccurate predictions, especially in underrepresented groups. Another limitation is

that the analysis relies on a dataset of 10,000 patients, which might be inadequate for making precise outcome

predictions.

In [ ]:

1

*#E4.Recommend a course of action for the real-world organizational situation from part A1 based on your results and implications discussed in part E2.*

2

3

First, I recommend that the health organization implement the KNN classification model into their system to predict patient readmission based on chronic conditions, lifestyle factors, and healthcare services. This model, with a best-performing parameter of 5 neighbors, has demonstrated high accuracy (94.825%) in a cross-validated setting.

Second, establish a system for continuous monitoring and evaluation of the model's performance. Regularly assess its accuracy, precision, recall, and other relevant metrics to ensure its effectiveness over time.

lastly, collaborate with healthcare professionals to integrate the predictive model into the existing healthcare workflow. This could involve incorporating the model's predictions into decision-making processes related to patient care and resource allocation.

1

*#F* [*https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d4858191-1934-4991-89ea-b0f8013dfd9e*](https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d4858191-1934-4991-89ea-b0f8013dfd9e)

#G Statology. (n.d.). How to Plot an ROC Curve in Python. Retrieved from <https://www.statology.org/plot-roc-curve-python/>

Statology. (n.d.). How to Calculate AUC in Python. Retrieved from <https://www.statology.org/auc-in-python/>

W3Schools. (n.d.). Python - Machine Learning - Standardize Data. Retrieved from <https://www.w3schools.com/python/python_ml_scale.asp>

Pulagam, S. (2020, June 6). How to detect and deal with Multicollinearity. Medium. <https://towardsdatascience.com/how-to-detect-and-deal-with-multicollinearity-9e02b18695f1>

Raj, R. (2022, March 16). Classification Algorithms in Python. Thrive in AI. <https://medium.com/thrive-in-ai/classification-algorithms-in-python-5f58a7a27b88>