

```
In [1]: import numpy as np
import pandas as pd
from sklearn import linear_model
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import sklearn
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import import cross_val_score, train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
pd.set_option('display.max_columns', None)
df = pd.read_csv('C:\Users\Fahim\Documents\0_WGUDocuments\d209\medical_clean.csv')
df.head()
df.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   object
 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   object
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   object
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   int64
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

In [2]: #check for missing data
df.isna().any()

Out[2]:
CaseOrder      False
Customer_id    False
Interaction     False
UID            False
City           False
State          False
County         False
Zip            False
Lat            False
Lng            False
Population     False
Area           False
TimeZone      False
Job            False
Children       False
Age            False
Income         False
Marital        False
Gender         False
ReAdmis        False
VitD_levels    False
Doc_visits     False
Full_meals_eaten  False
vitD_supp      False
Soft_drink     False
Initial_admin  False
HighBlood      False
Stroke         False
Complication_risk  False
Overweight     False
Arthritis      False
Diabetes       False
Hyperlipidemia False
BackPain       False
Anxiety        False
Allergic_rhinitis False
Reflux_esophagitis False
Asthma         False
Services       False
Initial_days   False
TotalCharge    False
Additional_charges False
Item1          False
Item2          False
Item3          False
Item4          False
Item5          False
Item6          False
Item7          False
Item8          False
dtype: bool

In [3]: #check for duplicate data in columns
df[df.duplicated()]

Out[3]:
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 Ov

In [4]: # check if any cols are duplicated - looking for False
df.columns.duplicated().any()

Out[4]:
False

In [5]: # check if any rows are duplicated - looking for False
df.duplicated().any()

Out[5]:
False

In [7]: # drop demographic data
df = df.drop(['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job'], axis=1)

In [8]: # verify columns were dropped
df.head()

Out[8]:
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
0         1    53  86575.93  Divorced  Male      No  19.141466         6          0          0          0  No  Emergency Admission      Yes  No  Medium      No  Yes  Yes  No  Yes  Yes  No  Yes
1         3    51  46805.99   Married  Female    No  18.940352         4          2          1          0  No  Emergency Admission      Yes  No  High      Yes  No  No  No  No  No  No  Yes  No
2         3    53  14370.14  Widowed  Female    No  18.057507         4          1          0          0  No  Elective Admission      Yes  No  Medium      Yes  No  Yes  No  No  No  No  No  No
3         0    78  39741.49   Married  Male     No  16.576858         4          1          0          0  No  Elective Admission      No  Yes  Medium      No  Yes  No  No  No  No  No  No  Yes  Yes
4         1    22  1209.56  Widowed  Female    No  17.439069         5          0          2          Yes  Elective Admission      No  No  Low      No  No  No  No  Yes  No  No  Yes  No  No

In [9]: #view descriptive statistics
df.describe()
df.rename(columns={'Item1':'Timely_admis', 'Item2':'Timely_treat', 'Item3':'Timely_visits', 'Item4':'Reliability', 'Item5':'Options', 'Item6':'Hrs_treat', 'Item7':'Courteous', 'Item8':'Active_listen'},inplace=True)

In [10]: df.head()

Out[10]:
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
0         1    53  86575.93  Divorced  Male      No  19.141466         6          0          0          0  No  Emergency Admission      Yes  No  Medium      No  Yes  Yes  No  Yes  Yes  No  Yes
1         3    51  46805.99   Married  Female    No  18.940352         4          2          1          0  No  Emergency Admission      Yes  No  High      Yes  No  No  No  No  No  No  No  Yes  No
2         3    53  14370.14  Widowed  Female    No  18.057507         4          1          0          0  No  Elective Admission      Yes  No  Medium      Yes  No  Yes  No  No  No  No  No  No  No
3         0    78  39741.49   Married  Male     No  16.576858         4          1          0          0  No  Elective Admission      No  Yes  Medium      No  Yes  No  No  No  No  No  No  Yes  Yes
4         1    22  1209.56  Widowed  Female    No  17.439069         5          0          2          Yes  Elective Admission      No  No  Low      No  No  No  No  Yes  No  No  Yes  No  No

In [11]: newdf=df[['Age','Doc_visits','Initial_days','TotalCharge','Additional_charges', 'Initial_admin','Stroke','Complication_risk','ReAdmis']].copy()

In [14]: #change Initial_admin, Stroke, and Complication_risk responses to numeric
newdf['Initial_admin'].replace({'Elective Admission','Observation Admission','Emergency Admission'}, (0,1,2), inplace=True)
newdf['Complication_risk'].replace({'Low','Medium','High'}, (0,1,2), inplace=True)
newdf['Stroke'].replace({'Yes','No'},(1,0),inplace=True)

In [15]: newdf.head()

Out[15]:
Age  Doc_visits  Initial_days  TotalCharge  Additional_charges  Initial_admin  Stroke  Complication_risk  ReAdmis
0    53          6    10.595770    3726.702860         17939.403420          2          0          1          No
1    51          4    15.129562    4193.190458         17612.996120          2          0          2          No
2    53          4     4.772177    2434.234222         17505.192460          0          0          1          No
3    78          4     1.714879    2127.830423         12993.437350          0          1          1          No
4    22          5     1.254807    2113.073274         3716.525796          0          0          0          No

In [17]: newdf.to_csv('C:\Users\Fahim\Documents\0_WGUDocuments\d209\medical_d209TASK2PREPARED.csv', index=False)

In [18]: #scale data
predictors = newdf.columns[newdf.dtypes.apply(lambda c: np.issubdtype(c, np.number))]
scaler=StandardScaler()
newdf[predictors] = scaler.fit_transform(newdf[predictors])

In [19]: #convert target variable to numeric
newdf['ReAdmis']=df.ReAdmis.map(dict(Yes=1, No=0))

In [20]: #show scaled data
newdf.head()

Out[20]:
Age  Doc_visits  Initial_days  TotalCharge  Additional_charges  Initial_admin  Stroke  Complication_risk  ReAdmis
0   -0.024795   0.944647   -0.907310   -0.727185         0.765005   0.895459  -0.498906   -0.168873    0
1  -0.121706  -0.967981  -0.734595  -0.513228         0.715114   0.895459  -0.498906   -1.200737    0
2  -0.024795  -0.967981  -1.128292  -1.319983         0.698635  -1.510396  -0.498906   -0.168873    0
3  1.186592  -0.967981  -1.244503  -1.460517         0.009004  -1.510396  2.004396   -0.168873    0
4  -1.526914  -0.011667  -1.261991  -1.467285        -1.408991  -1.510396  -0.498906   -1.536483    0

In [21]: #split data into test and train datasets
train_test = train_test_split(newdf,test_size=0.20, random_state=42)
x_train=train.drop('ReAdmis',axis=1)
y_train=train['ReAdmis']
x_test=test.drop('ReAdmis',axis=1)
y_test=test['ReAdmis']

In [22]: #export test and train files
x_train.to_csv('C:\Users\Fahim\Documents\0_WGUDocuments\d209\medical_xtrain2.csv', index = False)
x_test.to_csv('C:\Users\Fahim\Documents\0_WGUDocuments\d209\medical_xtest2.csv', index = False)
y_train.to_csv('C:\Users\Fahim\Documents\0_WGUDocuments\d209\medical_ytrain2.csv', index = False)
y_test.to_csv('C:\Users\Fahim\Documents\0_WGUDocuments\d209\medical_ytest2.csv', index = False)

In [23]: from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(bootstrap = True, class_weight = None, criterion = 'gini', max_depth = None, max_features = 'auto', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf = 1, min_samples_split = 2, min

C:\Users\Fahim\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ensemble\_forest.py:427: FutureWarning: 'max_features='auto'' has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, exp
licitly set 'max_features='sqrt' or remove this parameter as it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
warn(

Out[23]:
RandomForestClassifier
RandomForestClassifier(max_features='auto', n_jobs=1)

In [24]: #Saving predictions
y_pred=clf.predict(x_test)

In [25]: #Showing predictions vs actual
pd.DataFrame(data={'Predicted': y_pred, 'Actual': y_test}).head(15)

Out[25]:
Predicted  Actual
6252      0      0
4684      0      0
1731      0      0
4742      0      0
4521      0      0
6340      0      0
576       0      0
5202      1      1
6363      1      1
439       0      0
2750      0      0
7487      0      0
5272      0      0
5653      1      1
3999      0      0

In [26]: #classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

precision    recall  f1-score   support

0           0.98      0.99      0.99      1291
1           0.98      0.97      0.97       789

accuracy          0.98      2680
macro avg         0.98      0.98      0.98      2680
weighted avg         0.98      0.98      0.98      2680

In [27]: #Show accuracy score
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)

0.981

In [29]: #Show recall Score
from sklearn.metrics import recall_score
recall_score(y_test, y_pred, average='weighted')

Out[29]:
0.981

In [30]: #Show precision Score
from sklearn.metrics import precision_score
precision_score(y_test, y_pred, average='weighted')

Out[30]:
0.9809852316991343

In [31]: #Show F1 Score
from sklearn.metrics import f1_score
f1_score(y_test, y_pred, average='weighted')

Out[31]:
0.9809754429261236

In [33]: import scikitplot as skplt
y_probas=clf.predict_proba(x_test)
skplt.metrics.plot_roc(y_test, y_probas, figsize=(10, 8))
plt.show()

ROC Curves

True Positive Rate

False Positive Rate

ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)

In [41]: #Determine AUC
from sklearn import preprocessing
from sklearn.metrics import roc_auc_score
def aucScore(y_test, y_pred, average='weighted'):
    lb = preprocessing.LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return roc_auc_score(y_test, y_pred, average=average)

In [42]: #Area Under Curve
aucScore(y_test, y_pred)

Out[42]:
0.9779765217525256

In [43]: #Mean Squared Error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred)

Out[43]:
0.019

In [ ]:
```