# Standard imports

import numpy as np

import pandas as pd

from pandas import DataFrame

# Visualization libraries

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from matplotlib import pyplot as plt

plt.style.use('ggplot')

# Scikit-learn

import sklearn

from sklearn import datasets

from sklearn.tree import DecisionTreeRegressor

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

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn import preprocessing

from sklearn.metrics import mean\_absolute\_error as MAE

from sklearn.metrics import mean\_squared\_error as MSE

# Import model, splitting method & metrics from sklearn

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

#load the dataset

df = pd.read\_csv("medical clean 1.1.23.csv")

#examine the first 5 records of data

df.head()

#view describe

df.info

#descriptive stats

df.describe()

#check for null values

df.isnull()

# Getting data types of features

df.dtypes

#change to integers

df['TotalCharge'] = df['TotalCharge'].astype(int)

df['Initial\_days'] = df['Initial\_days'].astype(int)

#Change object to category

df["Gender"] = df["Gender"].astype('category')

df["ReAdmis"] = df["ReAdmis"].astype('category')

df["Soft\_drink"] = df["Soft\_drink"].astype('category')

df["Initial\_admin"] = df["Initial\_admin"].astype('category')

df["HighBlood"] = df["HighBlood"].astype('category')

df["Stroke"] = df["Stroke"].astype('category')

df["Overweight"] = df["Overweight"].astype('category')

df["Arthritis"] = df["Arthritis"].astype('category')

df["Diabetes"] = df["Diabetes"].astype('category')

df["Hyperlipidemia"] = df["Hyperlipidemia"].astype('category')

df["BackPain"] = df["BackPain"].astype('category')

df["Anxiety"] = df["Anxiety"].astype('category')

df["Allergic\_rhinitis"] = df["Allergic\_rhinitis"].astype('category')

df["Reflux\_esophagitis"] = df["Reflux\_esophagitis"].astype('category')

df["Services"] = df["Services"].astype('category')

df["Asthma"] = df["Asthma"].astype('category')

df["Marital"] = df["Marital"].astype('category')

df["Complication\_risk"] = df["Complication\_risk"].astype('category')

#drop columns not being used

to\_drop = ['CaseOrder', 'Customer\_id', 'Marital', 'Age', 'Hyperlipidemia', 'Asthma', 'Complication\_risk', 'vitD\_supp', 'Interaction', 'Gender', 'Reflux\_esophagitis', 'Allergic\_rhinitis', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job', 'Income', 'VitD\_levels', 'Additional\_charges', 'TotalCharge', 'Initial\_days', 'Services', 'Initial\_admin','Children', 'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6','Item7', 'Item8']

df.drop(to\_drop, inplace=True, axis=1)

#check data types

df.dtypes

#Run get dummies on categorical

pd.get\_dummies(df, columns = ['ReAdmis', 'HighBlood', 'Overweight','Soft\_drink', 'Stroke', 'Arthritis', 'Diabetes', 'BackPain', 'Anxiety'])

#check data types

df\_ready.dtypes

#drop multiple columns by name

df\_ready.drop(['ReAdmis\_No', 'HighBlood\_No', 'Overweight\_No', 'Soft\_drink\_No', 'Arthritis\_No', 'Diabetes\_No', 'BackPain\_No', 'Anxiety\_No','Stroke\_No'], axis=1, inplace=True)

#check data types

df\_ready.dtypes

#view histograms to get a feel for the data

plt.style.use('ggplot')

X = df\_ready.drop('ReAdmis\_Yes', 1).values

# drop target variable

y1 = df\_ready['ReAdmis\_Yes'].values

pd.DataFrame.hist(df\_ready, figsize = [10,10]);

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['HighBlood\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Full\_meals\_eaten'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Doc\_visits'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Overweight\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Soft\_drink\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Stroke\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Arthritis\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Diabetes\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['BackPain\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# A scatterplot to get an idea of correlations between potentially related variables

sns.scatterplot(x=df\_ready['Anxiety\_Yes'], y=df\_ready['ReAdmis\_Yes'], color='green')

plt.show()

# set the plot style to ggplot

plt.style.use('ggplot')

# Countplots of categorical variables

plt.figure()

sns.countplot(x='HighBlood\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='Overweight\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='Soft\_drink\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='Stroke\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='Arthritis\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='Diabetes\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='BackPain\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# Countplots of categorical variables

plt.figure()

sns.countplot(x='Anxiety\_Yes', hue='ReAdmis\_Yes', data=df\_ready, palette='RdBu')

plt.xticks([0,1], ['No', 'Yes'])

plt.show()

# A scatter matrix of the discrete variables for high level overview of potential relationships & distributions

df\_discrete = df\_ready[['Doc\_visits', 'Full\_meals\_eaten']]

pd.plotting.scatter\_matrix(df\_discrete, figsize = [20, 20])

#save prepared data

df\_ready.to\_csv('Documents/PreparedData D209 Task2.csv')

# List features for analysis

features = (list(df\_ready.columns[:-1]))

print('Features for analysis include: \n', features)

# Re-read fully numerical prepared dataset

df\_ready = pd.read\_csv("PreparedData D209 Task2.csv")

# Set predictor features & target variable

X = df\_ready.drop('ReAdmis\_Yes', axis=1).values

y = df\_ready['ReAdmis\_Yes'].values

# Create training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state =1)

# Export y\_test dataset

y\_test\_df\_ready = pd.DataFrame(X\_test)

y\_test\_df\_ready.to\_csv('Documents/PreparedData D209 y\_test Task2.csv')

# Export y train dataset

y\_train\_df\_ready = pd.DataFrame(X\_train)

y\_train\_df\_ready.to\_csv('Documents/PreparedData D209 y\_train Task2.csv')

# Export X train dataset

X\_train\_df\_ready = pd.DataFrame(X\_train)

X\_train\_df\_ready.to\_csv('Documents/PreparedData D209 X\_train Task2.csv')

# Export X test dataset

X\_test\_df\_ready = pd.DataFrame(X\_test)

X\_test\_df\_ready.to\_csv('Documents/PreparedData D209 X\_test Task2.csv')

# Instantiate Decision Tree Regressor model

dt = DecisionTreeRegressor(max\_depth = 8, min\_samples\_leaf = 0.1, random\_state = 1)

# Fit dataframe to Decision Tree Regressor model

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

# Predict Outcomes from test set

y\_pred = dt.predict(X\_test)

# Compute test set MSE

mse\_dt = MSE(y\_test, y\_pred)

# Compute test set RMSE

rmse\_dt = mse\_dt\*\*(1/2)

# Print initial RMSE

print('Initial RMSE score Decision Tree Regressor model: {:.3f}'.format(rmse\_dt))

# Compute the coefficient of determination (R-squared)

scores = cross\_val\_score(dt, X, y, scoring='r2')

# Print R-squared value

print('Cross validation R-squared values: ', scores)

# Print Mean Squared Error

print('With a manual calculation, the Mean Squared Error: {:.3f} '.format(sum(abs(y\_test - y\_pred)\*\*2)/len(y\_pred)))

print('Using scikit-lean, the Mean Squared Error: {:.3f}'.format(MSE(y\_test, y\_pred)))

# Calculate & print the Root Mean Squared Error

RMSE = MSE(y\_test, y\_pred)\*\*(1/2)

# Get parameters of Decision Tree Regression model for cross validation

dt.get\_params()

# Define grid of hyperparameters

params\_dt = {'max\_depth': [4, 6, 8],

'min\_samples\_leaf': [0.1, 0.2],

'max\_features': ['log2', 'sqrt']}

# Re-instantiate Decision Tree Regressor for cross validation

dt = DecisionTreeRegressor()

# Instantiate GridSearch cross validation

dt\_cv = GridSearchCV(estimator=dt,

param\_grid=params\_dt,

scoring='neg\_mean\_squared\_error',

cv=5,

verbose=1,

n\_jobs=-1)

# Fit model to

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

# Print best parameters

print('Best parameters for this Decision Tree Regressor model: {}'.format(dt\_cv.best\_params\_))

print('Best score for this Decision Tree Regressor model: {:.3f}'.format(dt\_cv.best\_score\_))