#Import the libraries needed

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import sklearn as sl

from statsmodels.formula.api import ols

#load the dataset

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

#Check total rows/columns

rows = df.shape[0]

cols = df.shape[1]

# display the number of rows and columns

print("Number of Rows: " + str(rows))

print("Number of Columns: " + str(cols))

#check for nan and null values

df.isna().any()

df.isnull().any()

# using isnull() function

df.isnull()

# using isnull() function

df.notnull()

# count of unique values in each column

print(df.nunique())

#check data types

df.dtypes

#Change object to category for ReAdmis

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

#Change object to category for Anxiety

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

#Change object to category for Services

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

#Change object to category for Complication Risk

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

#Change object to category for Overweight

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

#Change object to category for Initial\_admin

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

#Change object to category for Arthritis

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

#Change object to category for HighBlood

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

#Change object to category for Soft\_drink

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

#Change object to category for Stroke

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

#Change object to category for Gender

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

#Change object to category for Diabetes

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

df.dtypes

#change floats to integers

df['Initial\_days'] = df['Initial\_days'].apply(np.int64)

df.dtypes

#drop all columns and rows not being used

to\_drop = ['CaseOrder', 'Customer\_id','Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job', 'Children', 'Income', 'Marital', 'VitD\_levels', 'Full\_meals\_eaten', 'vitD\_supp','Hyperlipidemia', 'BackPain', 'Allergic\_rhinitis', 'Reflux\_esophagitis', 'Asthma', 'Additional\_charges', 'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8', 'TotalCharge']

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

#Run get dummies on categorical

pd.get\_dummies(df, columns = ['ReAdmis', 'Anxiety', 'Services', 'Complication\_risk', 'Overweight', 'Initial\_admin', 'HighBlood','Arthritis', 'Gender', 'Soft\_drink', 'Stroke', 'Diabetes' ])

#create new df with the get dummies responses

dfupdated = pd.get\_dummies(df, columns = ['ReAdmis', 'Anxiety', 'Services', 'Overweight', 'Arthritis','HighBlood', 'Initial\_admin', 'Complication\_risk', 'Gender', 'Soft\_drink', 'Stroke', 'Diabetes'])

#Drop columns to avoid dummy variable trap

to\_drop = ['ReAdmis\_No', 'Anxiety\_No', 'Services\_CT Scan', 'Services\_Blood Work', 'Services\_Intravenous', 'Complication\_risk\_Low', 'Complication\_risk\_Medium', 'Overweight\_No', 'HighBlood\_No', 'Initial\_admin\_Elective Admission', 'Initial\_admin\_Observation Admission','Arthritis\_No', 'Gender\_Male', 'Gender\_Nonbinary', 'Soft\_drink\_No', 'Diabetes\_No', 'Stroke\_No' ]

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

#RUN VIF

from patsy import dmatrices

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

#find design matrix for linear regression model using 'Initial\_days' as response variable

y, X = dmatrices('Initial\_days ~ Age+Doc\_visits', data=dfupdated, return\_type='dataframe')

#calculate VIF for each explanatory variable

vif = pd.DataFrame()

vif['VIF'] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

vif['variable'] = X.columns

#view VIF for each explanatory variable

vif

#export data and save file

dfupdated.to\_csv('Documents/PreparedDataSubmit.csv')

#Create Histograms

plt.hist(dfupdated["Initial\_days"])

plt.title("Initial\_days");

plt.hist(dfupdated["ReAdmis\_Yes"])

plt.title("ReAdmis\_Yes");

plt.hist(dfupdated["Age"])

plt.title("Age")

plt.hist(dfupdated["Doc\_visits"])

plt.title("Doc\_visits")

plt.hist(dfupdated["Anxiety\_Yes"])

plt.title("Anxiety\_Yes")

plt.hist(dfupdated["Services\_MRI"])

plt.title("Services\_MRI")

plt.hist(dfupdated["Overweight\_Yes"])

plt.title("Overweight\_Yes")

plt.hist(dfupdated["Arthritis\_Yes"])

plt.title("Arthritis\_Yes")

plt.hist(dfupdated["HighBlood\_Yes"])

plt.title("HighBlood\_Yes")

plt.hist(dfupdated["Initial\_admin\_Emergency Admission"])

plt.title("Initial\_admin\_Emergency Admission")

plt.hist(dfupdated["Complication\_risk\_High"])

plt.title("Complication\_risk\_High")

plt.hist(dfupdated["Gender\_Female"])

plt.title("Gender\_Female")

plt.hist(dfupdated["Soft\_drink\_Yes"])

plt.title("Soft\_drink\_Yes")

plt.hist(dfupdated["Stroke\_Yes"])

plt.title("Stroke\_Yes")

plt.hist(dfupdated["Diabetes\_Yes"])

plt.title("Diabetes\_Yes")

#Scatterplot for Initial Days

plt.scatter(dfupdated.index,dfupdated["Initial\_days"])

plt.title("Initial\_days")

plt.show()

#Boxplot for initial days

plt.boxplot (dfupdated["Initial\_days"])

plt.title("Initial\_days")

#scatterplot for initial days and age with trend line

sns.regplot(x="Initial\_days", y="Age", data=dfupdated, ci=None)

plt.show()

#scatterplot for initial days and doctor visits with trend line

sns.regplot(x="Initial\_days", y="Doc\_visits", data=dfupdated, ci=None)

plt.show()

#visualizing numerical and categorical data with a box plot

sns.boxplot(data=dfupdated, x="ReAdmis\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Services\_MRI", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Anxiety\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Overweight\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Arthritis\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="HighBlood\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Initial\_admin\_Emergency Admission", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Complication\_risk\_High", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Gender\_Female", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Soft\_drink\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Stroke\_Yes", y="Initial\_days")

sns.boxplot(data=dfupdated, x="Diabetes\_Yes", y="Initial\_days")

#Pearson’s correlation coefficient test on doc visits

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Doc\_visits']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on soft drink

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Soft\_drink\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on age

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Age’]

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on ReAdmis\_Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['ReAdmis\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Anxiety\_Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated[‘Initial\_days’]

list2 = dfupdated[‘Anxiety\_Yes’]

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Services MRI

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Services\_MRI']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Overweight\_Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Overweight\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Arthritis\_Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Arthritis\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on HighBlood\_Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['HighBlood\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

dfupdated.rename(columns = {'Initial\_admin\_Emergency Admission':'EmergencyAdmission'}, inplace = True)

#Pearson’s correlation coefficient test on Initial\_admin\_Emergency Admission

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated[‘EmergencyAdmission’]

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Complication\_risk\_High

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Complication\_risk\_High']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Gender\_Female

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Gender\_Female']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Stroke Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Stroke\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Pearson’s correlation coefficient test on Diabetes Yes

from scipy.stats import pearsonr

# Convert dataframe into series

list1 = dfupdated['Initial\_days']

list2 = dfupdated['Diabetes\_Yes']

# Apply the pearsonr()

corr, \_ = pearsonr(list1, list2)

print('Pearsons correlation: %.3f' % corr)

#Multiple Regression Model

# (First regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

dfupdated\_train, dfupdated\_test = train\_test\_split(dfupdated, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Age', 'Doc\_visits', 'Initial\_days']

dfupdated\_train[num\_vars] = scaler.fit\_transform(dfupdated\_train[num\_vars])

dfupdated\_train

#Divide the data into X and Y

y\_train = dfupdated\_train.pop('Initial\_days')

X\_train = dfupdated\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

dfupdated2 = dfupdated[['Initial\_days', 'Age', 'Doc\_visits', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Services\_MRI', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female', 'Soft\_drink\_Yes', 'Diabetes\_Yes', 'Stroke\_Yes']]

# (Second regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

dfupdated2\_train, df\_test = train\_test\_split(dfupdated2, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Age', 'Doc\_visits', 'Initial\_days']

dfupdated2\_train[num\_vars] = scaler.fit\_transform(dfupdated2\_train[num\_vars])

dfupdated2\_train

#Divide the data into X and Y

y\_train = dfupdated2\_train.pop('Initial\_days')

X\_train = dfupdated2\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated3 = dfupdated2[['Initial\_days', 'Age', 'Doc\_visits', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Services\_MRI', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female', 'Soft\_drink\_Yes', 'Stroke\_Yes']]

# Third regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated3\_train, df\_test = train\_test\_split(Dfupdated3, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Age','Doc\_visits', 'Initial\_days']

Dfupdated3\_train[num\_vars] = scaler.fit\_transform(Dfupdated3\_train[num\_vars])

Dfupdated3\_train

#Divide the data into X and Y

y\_train = Dfupdated3\_train.pop('Initial\_days')

X\_train = Dfupdated3\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated4 = Dfupdated3[['Initial\_days', 'Doc\_visits', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Services\_MRI', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female', 'Soft\_drink\_Yes', 'Stroke\_Yes']]

# 4th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated4\_train, df\_test = train\_test\_split(Dfupdated4, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Doc\_visits', 'Initial\_days']

Dfupdated4\_train[num\_vars] = scaler.fit\_transform(Dfupdated4\_train[num\_vars])

Dfupdated4\_train

#Divide the data into X and Y

y\_train = Dfupdated4\_train.pop('Initial\_days')

X\_train = Dfupdated4\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated5 = Dfupdated4[['Initial\_days', 'Doc\_visits', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Services\_MRI', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female', 'Soft\_drink\_Yes']]

# Fifth regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated5\_train, Dfupdated5\_test = train\_test\_split(Dfupdated5, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Doc\_visits', 'Initial\_days']

Dfupdated5\_train[num\_vars] = scaler.fit\_transform(Dfupdated5\_train[num\_vars])

Dfupdated5\_train

#Divide the data into X and Y

y\_train = Dfupdated5\_train.pop('Initial\_days')

X\_train = Dfupdated5\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated6 = Dfupdated5[['Initial\_days', 'Doc\_visits', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female', 'Soft\_drink\_Yes']]

# 6th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated6\_train, Dfupdated6\_test = train\_test\_split(Dfupdated6, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Doc\_visits', 'Initial\_days']

Dfupdated6\_train[num\_vars] = scaler.fit\_transform(Dfupdated6\_train[num\_vars])

Dfupdated6\_train

#Divide the data into X and Y

y\_train = Dfupdated6\_train.pop('Initial\_days')

X\_train = Dfupdated6\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated7 = Dfupdated6[['Initial\_days', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female', 'Soft\_drink\_Yes']]

# 7th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated7\_train, Dfupdated7\_test = train\_test\_split(Dfupdated7, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Initial\_days']

Dfupdated7\_train[num\_vars] = scaler.fit\_transform(Dfupdated7\_train[num\_vars])

Dfupdated7\_train

#Divide the data into X and Y

y\_train = Dfupdated7\_train.pop('Initial\_days')

X\_train = Dfupdated7\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated8 = Dfupdated7[['Initial\_days', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Overweight\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female']]

# 8th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated8\_train, Dfupdated8\_test = train\_test\_split(Dfupdated8, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Initial\_days']

Dfupdated8\_train[num\_vars] = scaler.fit\_transform(Dfupdated8\_train[num\_vars])

Dfupdated8\_train

#Divide the data into X and Y

y\_train = Dfupdated8\_train.pop('Initial\_days')

X\_train = Dfupdated8\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated9 = Dfupdated8[['Initial\_days', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'Arthritis\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female']]

# 9th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated9\_train, Dfupdated9\_test = train\_test\_split(Dfupdated9, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Initial\_days']

Dfupdated9\_train[num\_vars] = scaler.fit\_transform(Dfupdated9\_train[num\_vars])

Dfupdated9\_train

#Divide the data into X and Y

y\_train = Dfupdated9\_train.pop('Initial\_days')

X\_train = Dfupdated9\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#create a new dataframe with only the variables kept

Dfupdated10 = Dfupdated9[['Initial\_days', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission', 'Gender\_Female']]

# 10th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated10\_train, Dfupdated10\_test = train\_test\_split(Dfupdated10, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Initial\_days']

Dfupdated10\_train[num\_vars] = scaler.fit\_transform(Dfupdated10\_train[num\_vars])

Dfupdated10\_train

#Divide the data into X and Y

y\_train = Dfupdated10\_train.pop('Initial\_days')

X\_train = Dfupdated10\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#Create new dataframe removing the highest p-value

Dfupdated11 = Dfupdated10[['Initial\_days', 'ReAdmis\_Yes', 'Anxiety\_Yes', 'HighBlood\_Yes', 'EmergencyAdmission']]

# 11th regression model. Split the data into two different data sets with a 7:3 ratio. Since some columns have smaller integer ratios, this is important.

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

# We specify random seed so that the train and test data set always have the same rows, respectively

np.random.seed(0)

Dfupdated11\_train, Dfupdated11\_test = train\_test\_split(Dfupdated11, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

#Next – rescale the features. This does not include the dummy variables. Rescaling is very important as the features all need comparable scale. If not, that could set the coefficients results will not be accurate.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables

num\_vars = ['Initial\_days']

Dfupdated11\_train[num\_vars] = scaler.fit\_transform(Dfupdated11\_train[num\_vars])

Dfupdated11\_train

#Divide the data into X and Y

y\_train = Dfupdated11\_train.pop('Initial\_days')

X\_train = Dfupdated11\_train

#Build a linear model and add all variables

import statsmodels.api as sm

X\_train\_lm = sm.add\_constant(X\_train)

lr\_1 = sm.OLS(y\_train, X\_train\_lm).fit()

lr\_1.summary()

#Review the means of the remaining variables

Print(Dfupdated11.describe())

#Create residual plot

#fit multiple linear regression model

model = ols('Initial\_days ~ Anxiety\_Yes + HighBlood\_Yes + ReAdmis\_Yes + EmergencyAdmission', data=Dfupdated11).fit()

#view model summary

print(model.summary())

#create residual vs. predictor plot for ‘HighBlood Yes’

fig = plt.figure(figsize=(12,8))

fig = sm.graphics.plot\_regress\_exog(model, 'HighBlood\_Yes', fig=fig)

#create residual vs. predictor plot for 'Anxiety\_Yes'

fig = plt.figure(figsize=(12,8))

fig = sm.graphics.plot\_regress\_exog(model, 'Anxiety\_Yes', fig=fig)

import scipy.stats as st

# create 95% confidence interval

st.t.interval(alpha=0.95,

df=len(Dfupdated11)-1,

loc=np.mean(Dfupdated11),

scale=st.sem(Dfupdated11))