#DATA PREPROCESSING

import pandas as pd

import numpy as np

import seaborn as sns

pd.set\_option('display.max\_rows',None)

pd.set\_option('display.max\_columns',None)

pd.set\_option('display.width',None)

data=pd.read\_csv("E:\screen time analysis project/STS data cp\_2.csv")

data.isnull().sum()

data.columns

#data - original data

Data

#data1 - dropped name col, trimmed sleep hours, converted screen time; pysical activities and time to sleep

data1=data.drop(data.columns[0],axis=1)

data1

data1["Physical\_Activities"].unique()

pa\_map={

'No physical activity':1,

'Less than 30 min':2,

'30 min - 1 hour':3,

'1 - 2 hours':4,

'more than 2 hours':5

}

data1["Physical\_Activities"]=data1["Physical\_Activities"].map(pa\_map)

data1

data1["Time\_to\_sleep"].unique()

tts\_map={

'Less than 30 min':1,

'30 min - 1 hour':2,

'1-2 hours':3,

'more than 2 hours':4

}

data1["Time\_to\_sleep"]=data1["Time\_to\_sleep"].map(tts\_map)

data1

#data2 - process of converting gender, created gender dummies

data2=pd.DataFrame(data1)

data2.head()

gen\_dummy=pd.get\_dummies(data2["Gender"])

gen\_dummy.head()

data2=pd.concat((data2,gen\_dummy),axis=1)

data2.head()

#data3- converted gender to 0's and 1's (Female -1 , Male-0)

data3=data2.drop(data2.columns[1],axis=1)

data3.head()

data3=data3.drop(data3.columns[21],axis=1)

data3.head()

data3.rename(columns={"Female":"Gender"},inplace=True)

data3.head()

#data5 - converted screen affcet(1- Yes,0 - No)

data5=pd.DataFrame(data3)

sa\_dummy=pd.get\_dummies(data5["Screen\_affect"])

sa\_dummy.head()

data5=pd.concat((data5,sa\_dummy),axis=1)

data5.head(2)

data5=data5.drop(data5.columns[21],axis=1)

data5=data5.drop(data5.columns[3],axis=1)

data5.head(2)

data5.rename(columns={"Yes":"Screen\_Affect"},inplace=True)

data5.head(1)

#data6 - convert screen befor bed to 0's and 1's (1 - Yes, 0 - No)

data6=pd.DataFrame(data5)

data6.head(2)

sbb\_dummy=pd.get\_dummies(data6["Screeb\_before\_bed"])

sbb\_dummy.head(2)

data6=pd.concat((data6,sbb\_dummy),axis=1)

data6.head(2)

data6=data6.drop(data6.columns[21],axis=1)

data6.head(2)

data6.rename(columns={"Yes":"Screen\_Before\_bed"},inplace=True)

data6.head(2)

data6=data6.drop(data6.columns[3],axis=1)

data6.head(1)

#data7 convert drowsiness to 0's and 1's (yes - 1 No - 0)

data7=pd.DataFrame(data6)

data7.head()

drowsy\_dummy=pd.get\_dummies(data7["Drowsiness"])

drowsy\_dummy.head()

data7=pd.concat((data7,drowsy\_dummy),axis=1)

data7.head(2)

data7=data7.drop(data7.columns[21],axis=1)

data7.head(1)

data7=data7.drop(data7.columns[6],axis=1)

data7.rename(columns={"Yes":"Drowsiness"},inplace=True)

data7.head(1)

#data 8 converted low\_interest to 0's and 1's Yes - 1, No – 0

data8=pd.DataFrame(data7)

low\_dummy=pd.get\_dummies(data8["Lowr\_interest\_in\_other\_activities"])

low\_dummy.head()

data8=pd.concat((data8,low\_dummy),axis=1)

data8.head()

data8=data8.drop(data8.columns[10],axis=1)

data8.head(2)

data8=data8.drop(data8.columns[20],axis=1)

data8.rename(columns={"Yes":"Low\_intr\_otherAct"},inplace=True)

data8

#data9 - cleared Age

data9=pd.DataFrame(data8)

data9

data9["Age"].str.strip()

data9["Age"]

data9["Age"]=data9["Age"].str.extract('([0-9]+)')

data9["Age"]

data9["Age"].fillna("21",inplace=True)

data9["Age"].isnull().sum()

data9.isnull().sum()

#data10 - filling null values

data10=pd.DataFrame(data9)

data10.head()

data10["Asthenopia"].fillna("0",inplace=True)

data10["Asthenopia"].isnull().sum()

data10["Headache"].fillna("0",inplace=True)

data10["Sleepdepr"].fillna("0",inplace=True)

data10["B/Npain"].fillna("0",inplace=True)

data10["Obesity"].fillna("0",inplace=True)

data10["None"].fillna("0",inplace=True)

data10.isnull().sum()

data10=data10.drop(data10.columns[2],axis=1)

data10.head()

data10["Screen\_time"]

data11=pd.DataFrame(data10)

St\_less4=pd.DataFrame(data11,columns=['Screen\_time'])

St\_less4.loc[St\_less4['Screen\_time']<=3,'<= 4hours']='1'

St\_less4.loc[St\_less4['Screen\_time']>3,'<= 4hours']='0'

St\_less4

extracted\_col=St\_less4["<= 4hours"]

data11=data11.join(extracted\_col)

data11.head()

#RANDOM FOREST

import pandas as pd

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import confusion\_matrix

df=pd.read\_csv("E:\screen time analysis project/updated\_data2.csv")

X=df.drop("<= 4hours",axis=1)

X.head(1)

print(X.dtypes)

Y=df["<= 4hours"]

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

chi2\_features = SelectKBest(chi2, k=3)

X\_kbest\_features = chi2\_features.fit\_transform(X,Y)

print("org:",X.shape[1])

print("Reduced:",X\_kbest\_features.shape[1])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_kbest\_features, Y, test\_size=0.3, random\_state=44)

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier(n\_estimators=50, max\_features="auto", random\_state=44)

rf\_model.fit(X\_train, y\_train)

predictions = rf\_model.predict(X\_test)

predictions

y\_test

#Each array contains two probabilities in this case because we have two categories to predict: yes or no. The left value shows the predicted probability of belonging to the category of no, the second shows the same for belonging to the category of yes.

rf\_model.classes\_

rf\_model.predict\_proba(X\_test)

importances = rf\_model.feature\_importances\_

columns = X.columns

i=0

while i<len(columns):

print(f" The importance of feature '{columns[i]}' is {round(importances[i] \* 100,2)}%.")

i+=1

from sklearn import metrics

confusion\_matrix = metrics.confusion\_matrix(y\_test, predictions)

print(confusion\_matrix)

import matplotlib.pyplot as plt

import seaborn as sns

sns.set(font\_scale=1.4)

sns.heatmap(confusion\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted',fontsize = 12)

plt.ylabel('True',fontsize = 12)

plt.show()

predictions\_train = rf\_model.predict(X\_train)

confusion\_matrix\_train = metrics.confusion\_matrix(y\_train, predictions\_train)

print(confusion\_matrix\_train)

Accuracy=metrics.accuracy\_score(y\_train,predictions\_train)

CR=metrics.classification\_report(y\_train,predictions\_train)

print("Accuracy - ",Accuracy)

print("Classification Report:")

print(CR)

Accuracy=metrics.accuracy\_score(y\_test,predictions)

CR=metrics.classification\_report(y\_test,predictions)

print("Accuracy - ",Accuracy)

print("Classification Report:")

print(CR)

#K Fold Cross Validation

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

import numpy as np

scores = cross\_val\_score(rf\_model, X\_kbest\_features, Y,

scoring="accuracy",

cv=10, n\_jobs=-1)

print("Accuracy scores: ", np.round(scores, 2))

print("Average accuracy: ", np.round(np.mean(scores), 2))

#ADABOOST

import pandas as pd

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import confusion\_matrix

df=pd.read\_csv("E:\screen time analysis project/updated\_data2.csv")

X=df.drop("<= 4hours",axis=1)

X.head(1)

Y=df["<= 4hours"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=44)

from sklearn.ensemble import AdaBoostClassifier

clf = AdaBoostClassifier(random\_state=96)

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

sc\_train=clf.score(X\_train,y\_train)

sc\_test=clf.score(X\_test,y\_test)

# evaluate adaboost algorithm for classification

from numpy import mean

from numpy import std

print('Accuracy: %.3f (%.3f)' % (mean(sc\_train), std(sc\_test)))

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

y\_pred

from sklearn import metrics

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

# Load libraries

from sklearn.ensemble import AdaBoostClassifier

# Import Support Vector Classifier

from sklearn.svm import SVC

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

svc=SVC(probability=True, kernel='linear')

# Create adaboost classifer object

abc =AdaBoostClassifier(n\_estimators=50, base\_estimator=svc,learning\_rate=1)

# Train Adaboost Classifer

model = abc.fit(X\_train, y\_train)

#Predict the response for test dataset

y\_pred1 = model.predict(X\_test)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred1))

from sklearn.ensemble import RandomForestClassifier

clf1=AdaBoostClassifier(random\_state=96,base\_estimator=RandomForestClassifier(random\_state=101),n\_estimators=100,learning\_rate=0.01)

clf1.fit(X\_train,y\_train)

rf\_test=clf1.score(X\_test,y\_test)

rf\_test

rf\_train=clf1.score(X\_train,y\_train)

rf\_train

print('Accuracy: %.3f (%.3f)' % (mean(rf\_train), std(rf\_test)))