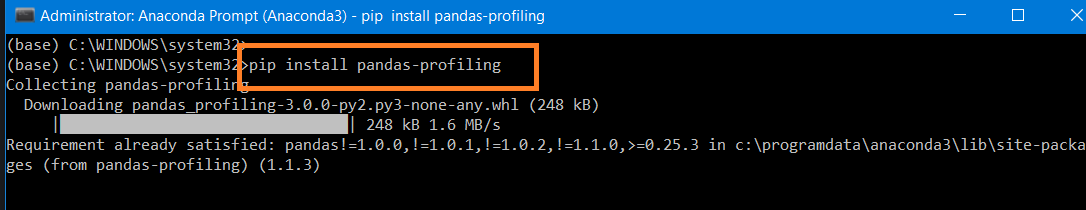
Parkinsons Disease Prediction

**Install the following Libraries**

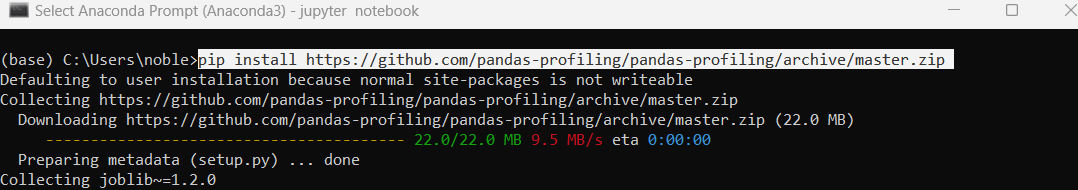
pip install pandas-profiling



If Profiling is not working and getting an error use the following statement to install pandas profiling

# TypeError: concat() got an unexpected keyword argument 'join\_axes' | Pandas Profiling

pip install <https://github.com/pandas-profiling/pandas-profiling/archive/master.zip>



**Import Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn import metrics

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

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import os

**Check Current Directory**

print (os.getcwd())

**Change the directory**

os.chdir ('C:\\Noble\\Training\\Acmegrade\\Data Science\\Projects\\Detection of Parkinsons Disease\\')

print (os.getcwd())

**Read Data, display records**

df=pd.read\_csv('parkinsons.data')

display (df)

**Attribute Information: Target column - Status**

Matrix column entries (attributes):

name - ASCII subject name and recording number

MDVP:Fo(Hz) - Average vocal fundamental frequency

MDVP:Fhi(Hz) - Maximum vocal fundamental frequency

MDVP:Flo(Hz) - Minimum vocal fundamental frequency

MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several

measures of variation in fundamental frequency

MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude

NHR,HNR - Two measures of ratio of noise to tonal components in the voice

status - Health status of the subject (one) - Parkinson's, (zero) - healthy

RPDE,D2 - Two nonlinear dynamical complexity measures

DFA - Signal fractal scaling exponent

spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

**Pandas Profiling Report**

import ydata\_profiling as pf

display(pf.ProfileReport(df))

**Display the shape**

display (df.shape)

**Number of rows**

print (len(df))

**Display the data type of all columns**

display (df.dtypes )

**Display Details**

print (df.info())

**Describe the details**

display (df.describe())

**Check for Null Values**

display (df.isna().sum() )

**Display column details**

print (df.columns)

**Display the dependent variable**

# status - health status of the subject (one) - Parkinson's, (zero) – healthy

print (df['status'])

**Create Histogram with Status column**

# The dataset has high number of patients effected with Parkinson's disease.

plt.figure(figsize=(10, 6))

df.status.hist()

plt.xlabel('Status')

plt.ylabel('Frequencies')

plt.plot()

plt.show()

**Create Bar graph- X-Axis Status, Y- Axis NHR**

'''

The patients affected with Parkinson's disease have high NHR which is the measure of the ratio of noise to tonal components in the voice.

'''

plt.figure(figsize=(10, 6))

sns.barplot(x="status",y="NHR",data=df);

plt.show()

**Create Bar graph- X-Axis Status, Y- Axis HNR**

'''

The patients affected with Parkinson's disease have high HNR

that is the measure of the ratio of noise to tonal components in the voice.

'''

plt.figure(figsize=(10, 6))

sns.barplot(x="status",y="HNR",data=df);

plt.show()

**Create Bar graph- X-Axis Status, Y- Axis RPDE**

'''

The nonlinear dynamical complexity measure RPDE is high in the patients affected with Parkinson's disease.

'''

plt.figure(figsize=(10, 6))

sns.barplot(x="status",y="RPDE",data=df);

plt.show()

**Create Distribution plot – This used to check skewness in data**

import warnings

warnings.filterwarnings('ignore')

rows=3

cols=7

fig, ax=plt.subplots(nrows=rows,ncols=cols,figsize=(16,4))

col=df.columns

index=1

for i in range(rows):

for j in range(cols):

sns.distplot(df[col[index]],ax=ax[i][j])

index=index+1

plt.tight\_layout()

plt.show()

**Display the top 3 records**

display (df.head(3))

**Display Co relation Matrix**

dfc=df.iloc[:,1:]

corr = dfc.corr()

display (corr)

**Display Heat Map**

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 20,10

sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='cubehelix',annot = True)

plt.show()

**Heatmap with Default Parameters**

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 20,10

sns.heatmap(corr)

plt.show()

**Drop the name column**

# Removing name column for machine learning algorithms.

df.drop(['name'],axis=1,inplace=True)

display (df)

**Spitting the dataset into x and y**

**Create X**

X=df.drop(labels=['status'],axis=1)

display (X.head())

**Create – Y**

Y=df['status']

display (Y.head())

**Splitting the data into x\_train, y\_train, x\_test, y\_test**

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=40)

print (X.shape,Y.shape)

print(X\_train.shape,X\_test.shape,Y\_train.shape,Y\_test.shape)

**Create a Logistic Regression Model**

log\_reg = LogisticRegression().fit(X\_train, Y\_train)

#predict on train

train\_preds = log\_reg.predict(X\_train)

#accuracy on train

print("Model accuracy on train is: ", accuracy\_score(Y\_train, train\_preds))

#predict on test

test\_preds = log\_reg.predict(X\_test)

#accuracy on test

print("Model accuracy on test is: ", accuracy\_score(Y\_test, test\_preds))

print('-'\*50)

#Confusion matrix

print("confusion\_matrix train is:\n ", confusion\_matrix(Y\_train, train\_preds))

print("confusion\_matrix test is:\n ", confusion\_matrix(Y\_test, test\_preds))

print('\nClassification Report Train is ')

print(classification\_report (Y\_train, train\_preds))

print('\nClassification Report Test is ')

print(classification\_report (Y\_test, test\_preds))

**Create Random Forest Model**

RF=RandomForestClassifier().fit(X\_train,Y\_train)

#predict on train

train\_preds2 = RF.predict(X\_train)

#accuracy on train

print("Model accuracy on train is: ", accuracy\_score(Y\_train, train\_preds2))

#predict on test

test\_preds2 = RF.predict(X\_test)

#accuracy on test

print("Model accuracy on test is: ", accuracy\_score(Y\_test, test\_preds2))

#Confusion matrix

print("confusion\_matrix train is:\n ", confusion\_matrix(Y\_train, train\_preds2))

print("confusion\_matrix test is:\n ", confusion\_matrix(Y\_test, test\_preds2))

print('\nClassification Report Train is ')

print(classification\_report (Y\_train, train\_preds2))

print('\nClassification Report Test is ')

print(classification\_report (Y\_test, test\_preds2))

**Wrong Predictions made**

print((Y\_test !=test\_preds2).sum(),'/',((Y\_test == test\_preds2).sum()+(Y\_test != test\_preds2).sum()))

**Kappa Score**

print('KappaScore is: ', metrics.cohen\_kappa\_score(Y\_test,test\_preds2))

**What is the meaning of kappa value?**

**The higher the kappa value, the stronger the degree of agreement**. When:

Kappa = 1, perfect agreement exists. Kappa < 0, agreement is weaker than expected by chance; this rarely happens. Kappa close to 0, the degree of agreement is the same as would be expected by chance

**Display the test and Predicted Values**

ddf=pd.DataFrame(data=[test\_preds2,Y\_test])

display (ddf)

**Transpose and display**

display (ddf.T)

**Decision Tree Classifier**

from sklearn.tree import DecisionTreeClassifier

#fit the model on train data

DT = DecisionTreeClassifier().fit(X,Y)

#predict on train

train\_preds3 = DT.predict(X\_train)

#accuracy on train

print("Model accuracy on train is: ", accuracy\_score(Y\_train, train\_preds3))

#predict on test

test\_preds3 = DT.predict(X\_test)

#accuracy on test

print("Model accuracy on test is: ", accuracy\_score(Y\_test, test\_preds3))

print('-'\*50)

#Confusion matrix

print("confusion\_matrix train is:\n ", confusion\_matrix(Y\_train, train\_preds3))

print("confusion\_matrix test is: \n", confusion\_matrix(Y\_test, test\_preds3))

print('Wrong predictions out of total')

print('-'\*50)

print('\nClassification Report Train is ')

print(classification\_report (Y\_train, train\_preds3))

print('\nClassification Report Test is ')

print(classification\_report (Y\_test, test\_preds3))

**Wrong Prediction and Kappa Score**

# Wrong Predictions made.

print((Y\_test !=test\_preds3).sum(),'/',((Y\_test == test\_preds3).sum()+(Y\_test != test\_preds3).sum()))

print('-'\*50)

# Kappa Score

print('KappaScore is: ', metrics.cohen\_kappa\_score(Y\_test,test\_preds3))

**Naïve Bayce algorithm**

from sklearn.naive\_bayes import GaussianNB

#fit the model on train data

NB=GaussianNB()

NB.fit(X\_train,Y\_train)

#predict on train

train\_preds4 = NB.predict(X\_train)

#accuracy on train

print("Model accuracy on train is: ", accuracy\_score(Y\_train, train\_preds4))

#predict on test

test\_preds4 = NB.predict(X\_test)

#accuracy on test

print("Model accuracy on test is: ", accuracy\_score(Y\_test, test\_preds4))

print('-'\*50)

#Confusion matrix

print("confusion\_matrix train is: \n", confusion\_matrix(Y\_train, train\_preds4))

print("confusion\_matrix test is:\n ", confusion\_matrix(Y\_test, test\_preds4))

print('Wrong predictions out of total')

print('-'\*50)

print('\nClassification Report Train is ')

print(classification\_report (Y\_train, train\_preds4))

print('\nClassification Report Test is ')

print(classification\_report (Y\_test, test\_preds4))

**Wrong Prediction and Kappa Score**

# Wrong Predictions made.

print((Y\_test !=test\_preds4).sum(),'/',((Y\_test == test\_preds4).sum()+(Y\_test != test\_preds4).sum()))

print('-'\*50)

# Kappa Score

print('KappaScore is: ', metrics.cohen\_kappa\_score(Y\_test,test\_preds4))

**K Neighbours Classifier**

from sklearn.neighbors import KNeighborsClassifier

#fit the model on train data

# Using the parameter weights='distance' to fix the error 'Flags' object has no attribute 'c\_contiguous'

KNN = KNeighborsClassifier(weights='distance').fit(X\_train,Y\_train)

#predict on train

train\_preds5 = KNN.predict(X\_train)

#accuracy on train

print("Model accuracy on train is: ", accuracy\_score(Y\_train, train\_preds5))

#predict on test

test\_preds5 = KNN.predict(X\_test)

#accuracy on test

print("Model accuracy on test is: ", accuracy\_score(Y\_test, test\_preds5))

print('-'\*50)

#Confusion matrix

print("confusion\_matrix train is:\n ", confusion\_matrix(Y\_train, train\_preds5))

print("confusion\_matrix test is:\n ", confusion\_matrix(Y\_test, test\_preds5))

print('Wrong predictions out of total')

print('-'\*50)

print('\nClassification Report Train is ')

print(classification\_report (Y\_train, train\_preds5))

print('\nClassification Report Test is ')

print(classification\_report (Y\_test, test\_preds5))

**Wrong Prediction and Kappa Score**

# Wrong Predictions made.

print((Y\_test !=test\_preds5).sum(),'/',((Y\_test == test\_preds5).sum()+(Y\_test != test\_preds5).sum()))

print('-'\*50)

# Kappa Score

print('KappaScore is: ', metrics.cohen\_kappa\_score(Y\_test,test\_preds5))

**Support Vector Machine**

from sklearn.svm import SVC

#fit the model on train data

SVM = SVC(kernel='linear')

SVM.fit(X\_train, Y\_train)

#predict on train

train\_preds6 = SVM.predict(X\_train)

#accuracy on train

print("Model accuracy on train is: ", accuracy\_score(Y\_train, train\_preds6))

#predict on test

test\_preds6 = SVM.predict(X\_test)

#accuracy on test

print("Model accuracy on test is: ", accuracy\_score(Y\_test, test\_preds6))

print('-'\*50)

#Confusion matrix

print("confusion\_matrix train is: \n", confusion\_matrix(Y\_train, train\_preds6))

print("confusion\_matrix test is:\n ", confusion\_matrix(Y\_test, test\_preds6))

print('Wrong predictions out of total')

print('-'\*50)

print("recall", metrics.recall\_score(Y\_test, test\_preds6))

print('-'\*50)

print('\nClassification Report Train is ')

print(classification\_report (Y\_train, train\_preds6))

print('\nClassification Report Test is ')

print(classification\_report (Y\_test, test\_preds6))

**Wrong Prediction and Kappa Score**

# Wrong Predictions made.

print((Y\_test !=test\_preds6).sum(),'/',((Y\_test == test\_preds6).sum()+(Y\_test != test\_preds6).sum()))

print('-'\*50)

# Kappa Score

print('KappaScore is: ', metrics.cohen\_kappa\_score(Y\_test,test\_preds6))

**Create Pickle File**

import pickle

# Saving model to disk

pickle.dump(SVM,open('deploy\_SVM.pkl','wb'))

# Open the Pickle File

model=pickle.load(open('deploy\_SVM.pkl','rb'))

# Prediction

print (model.predict (X\_train))