Machine Learning Project

PREDICTION ON

Parkinson's Disease

Guide By :

Deepak Gurav Sir

Created By :

Janhavi Deshmukh

CONTENT

PREDICTION ON PARKINSON'S DISEASE - ML MODEL

IMPORT LIBRARIES

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

LOAD DATASET

```
pd.set_option('display.max_rows',200)
pd.set_option('display.max_columns',25)
pd.set_option('display.width',200)



df = pd.read_csv(r"Parkinsson disease (1).csv")
df
```

DATA PREPROCESSING

```
df.shape
```

```
(195, 24)
```

```
df.isnull().sum()
```

```
name              0
MDVP:Fo(Hz)       0
MDVP:Fhi(Hz)      0
MDVP:Flo(Hz)      0
MDVP:Jitter(%)    0
MDVP:Jitter(Abs)  0
MDVP:RAP          0
MDVP:PPQ          0
Jitter:DDP        0
MDVP:Shimmer      0
MDVP:Shimmer(dB)  0
Shimmer:APQ3      0
Shimmer:APQ5      0
MDVP:APQ          0
Shimmer:DDA       0
NHR               0
HNR               0
status            0
RPDE              0
DFA               0
spread1           0
spread2           0
D2                0
PPE               0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              195 non-null    object 
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64  
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

```
df.dtypes
```

```
name               object
MDVP:Fo(Hz)       float64
MDVP:Fhi(Hz)      float64
MDVP:Flo(Hz)      float64
MDVP:Jitter(%)    float64
MDVP:Jitter(Abs)  float64
MDVP:RAP          float64
MDVP:PPQ          float64
Jitter:DDP        float64
MDVP:Shimmer      float64
MDVP:Shimmer(dB)  float64
Shimmer:APQ3      float64
Shimmer:APQ5      float64
MDVP:APQ          float64
Shimmer:DDA       float64
NHR               float64
HNR               float64
status              int64
RPDE              float64
DFA               float64
spread1           float64
spread2           float64
D2                float64
PPE               float64
dtype: object
```

```
df.describe()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	S
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	0.009920	0.029709	0.282251	
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	0.194877	
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	0.085000	
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	0.148500	
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	0.221000	
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	0.350000	
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	1.302000	

Shimmer:APQ3	Shimmer:APQ5	MDVP:APQ	Shimmer:DDA	NHR	HNR	status	RPDE	DFA	spread1	spread2	D2	PPE
195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000
0.015664	0.017878	0.024081	0.046993	0.024847	21.885974	0.753846	0.498536	0.718099	-5.684397	0.226510	2.381826	0.206552
0.010153	0.012024	0.016947	0.030459	0.040418	4.425764	0.431878	0.103942	0.055336	1.090208	0.083406	0.382799	0.090119
0.004550	0.005700	0.007190	0.013640	0.000650	8.441000	0.000000	0.256570	0.574282	-7.964984	0.006274	1.423287	0.044539
0.008245	0.009580	0.013080	0.024735	0.005925	19.198000	1.000000	0.421306	0.674758	-6.450096	0.174351	2.099125	0.137451
0.012790	0.013470	0.018260	0.038360	0.011660	22.085000	1.000000	0.495954	0.722254	-5.720868	0.218885	2.361532	0.194052
0.020265	0.022380	0.029400	0.060795	0.025640	25.075500	1.000000	0.587562	0.761881	-5.046192	0.279234	2.636456	0.252980
0.056470	0.079400	0.137780	0.169420	0.314820	33.047000	1.000000	0.685151	0.825288	-2.434031	0.450493	3.671155	0.527367

```
df['status'].value_counts()
```

```
1    147
0     48
Name: status, dtype: int64
```

```
df.groupby('status').mean()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_10836\4081209983.py:1: FutureWarning: The default value of numeric_only in DataFram
eGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select onl
y columns which should be valid for the function.
  df.groupby('status').mean()
```

status	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)
0	181.937771	223.636750	145.207292	0.003866	0.000023	0.001925	0.002056	0.005776	0.017615	0.162958
1	145.180762	188.441463	106.893558	0.006989	0.000051	0.003757	0.003900	0.011273	0.033658	0.321204

Shimmer:APQ3	Shimmer:APQ5	MDVP:APQ	Shimmer:DDA	NHR	HNR	RPDE	DFA	spread1	spread2	D2	PPE
0.009504	0.010509	0.013305	0.028511	0.011483	24.678750	0.442552	0.695716	-6.759264	0.160292	2.154491	0.123017
0.017676	0.020285	0.027600	0.053027	0.029211	20.974048	0.516816	0.725408	-5.333420	0.248133	2.456058	0.233828

PLOTTING

```
def distplot(col):
    sns.distplot(df[col])
    plt.show()

for i in list(df.columns)[1:]:
    distplot(i)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_10836\2109848044.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[col])
```

```
plt.figure(figsize=(20,20))
corr=df.corr()
sns.heatmap(corr,annot=True,cmap='rainbow')
```

```
x=df.drop(['status','name'],axis=1)
y=df['status']
```

```
print(x)
```

```
     MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer
MDVP:Shimmer(dB)  Shimmer:APQ3  Shimmer:APQ5  MDVP:APQ  Shimmer:DDA  \
0         119.992       157.302        74.997         0.00784           0.000070   0.00370   0.00554     0.01109       0.04374
0.426       0.02182       0.03130   0.02971     0.06545
1         122.400       148.650       113.819         0.00968           0.000080   0.00465   0.00696     0.01394       0.06134
0.626       0.03134       0.04518   0.04368     0.09403
2         116.682       131.111       111.555         0.01050           0.000090   0.00544   0.00781     0.01633       0.05233
0.482       0.02757       0.03858   0.03590     0.08270
3         116.676       137.871       111.366         0.00997           0.000090   0.00502   0.00698     0.01505       0.05492
0.517       0.02924       0.04005   0.03772     0.08771
4         116.014       141.781       110.655         0.01284           0.000110   0.00655   0.00908     0.01966       0.06425
0.584       0.03490       0.04825   0.04465     0.10470
5         120.552       131.162       113.787         0.00968           0.000080   0.00463   0.00750     0.01388       0.04701
0.456       0.02328       0.03526   0.03243     0.06985
6         120.267       137.244       114.820         0.00333           0.000030   0.00155   0.00202     0.00466       0.01608
0.140       0.00779       0.00937   0.01351     0.02337
7         107.332       113.840       104.315         0.00290           0.000030   0.00144   0.00182     0.00431       0.01567
0.134       0.00829       0.00946   0.01256     0.02487
8          95.730       132.068        91.754         0.00551           0.000060   0.00293   0.00332     0.00880       0.02093
```

```
print(y)
```

```
0     1
1     1
2     1
3     1
4     1
5     1
6     1
7     1
8     1
9     1
10    1
11    1
12    1
13    1
14    1
15    1
16    1
17    1
18    1
```

```
x.head()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	S
0	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	0.426	
1	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	0.626	
2	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	0.482	
3	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	0.517	
4	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	0.584	

Shimmer:APQ3	Shimmer:APQ5	MDVP:APQ	Shimmer:DDA	NHR	HNR	RPDE	DFA	spread1	spread2	D2	PPE
0.02182	0.03130	0.02971	0.06545	0.02211	21.033	0.414783	0.815285	-4.813031	0.266482	2.301442	0.284654
0.03134	0.04518	0.04368	0.09403	0.01929	19.085	0.458359	0.819521	-4.075192	0.335590	2.486855	0.368674
0.02757	0.03858	0.03590	0.08270	0.01309	20.651	0.429895	0.825288	-4.443179	0.311173	2.342259	0.332634
0.02924	0.04005	0.03772	0.08771	0.01353	20.644	0.434969	0.819235	-4.117501	0.334147	2.405554	0.368975
0.03490	0.04825	0.04465	0.10470	0.01767	19.649	0.417356	0.823484	-3.747787	0.234513	2.332180	0.410335

```
y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: status, dtype: int64
```

SPLIT DATA

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print('Shape of X_train=',x_train.shape)
print('Shape of y_train=',y_train.shape)
print('Shape of X_test=',x_test.shape)
print('Shape of y_test=',y_test.shape)
```

```
Shape of X_train= (156, 22)
Shape of y_train= (156,)
Shape of X_test= (39, 22)
Shape of y_test= (39,)
```

STANDARD SCALER

```
scaler = StandardScaler()
```

```
scaler.fit(x_train)
```

```
▾ StandardScaler
StandardScaler()
```

```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
print(x_train)
```

```
[[-0.80172872 -0.70830513 -0.10603303 ...  0.89854281 -0.48294197
   1.64980971]
 [-1.04374224 -0.74950432 -0.29919921 ...  1.17531845  0.65177741
   0.26864313]
 [-0.3790498   0.13122544 -0.82039362 ... -0.7771489  -0.73102398
   0.28542547]
 ...
 [-0.13744368 -0.31583967  0.56292775 ... -1.65126173 -0.08248373
  -0.17764131]
 [-0.14053505 -0.42034011  0.39263744 ...  0.19569946  1.19665768
  -0.11105606]
 [-0.35427092  4.57536567 -1.22354854 ...  0.55494177  0.80279984
   2.57800238]]
```

SUPPORT VECTOR CLASSIFICATION

```
from sklearn import svm

model = svm.SVC(kernel='linear')
```

```
model.fit(x_train, y_train)
```

```
SVC
SVC(kernel='linear')
```

```
x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(y_train, x_train_prediction)
```

```
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.9038461538461539
```

```
x_test_prediction = model.predict(x_test)
test_data_accuracy = accuracy_score(y_test, x_test_prediction)
```

```
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.8717948717948718
```

```
input_data = (197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.09700,0.00563,
              0.00680,0.00802,0.01689,0.00339,26.77500,0.422229,0.741367,-7.348300,0.177551,1.743867,0.085569)

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the data
std_data = scaler.transform(input_data_reshaped)

prediction = model.predict(std_data)
print(prediction)


if (prediction[0] == 0):
    print("The Person does not have Parkinsons Disease")

else:
    print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease
```

RANDOM FOREST CLASSIFICATION

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier=RandomForestClassifier(n_estimators=100,criterion='gini')
classifier.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
classifier.score(x_test,y_test)
```

```
0.9487179487179487
```

```
patient1=(197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.09700,0.00563,0.00680,0.00802,
          0.01689,0.00339,26.77500,0.422229,0.741367,-7.348300,0.177551,1.743867,0.085569)
```

```
patient1=np.array([patient1])
patient1
```

```
array([[ 1.970760e+02,  2.068960e+02,  1.920550e+02,  2.890000e-03,
         1.000000e-05,  1.660000e-03,  1.680000e-03,  4.980000e-03,
         1.098000e-02,  9.700000e-02,  5.630000e-03,  6.800000e-03,
         8.020000e-03,  1.689000e-02,  3.390000e-03,  2.677500e+01,
         4.222290e-01,  7.413670e-01, -7.348300e+00,  1.775510e-01,
         1.743867e+00,  8.556900e-02]])
```

```
classifier.predict(patient1)
```

```
array([0], dtype=int64)
```

```
pred=classifier.predict(patient1)
if pred[0]==0:
     print("The Person does not have Parkinsons Disease")
else:
    print("The Person has Parkinsons")
```

```
The Person does not have Parkinsons Disease
```

DECISION TREE CLASSIFIER

```
]: from sklearn.tree import DecisionTreeClassifier
```

```
]: classifier=DecisionTreeClassifier(criterion='gini')
   classifier.fit(x_train,y_train)
```

```
]: DecisionTreeClassifier
   DecisionTreeClassifier()
```

```
]: classifier.score(x_test,y_test)
```

```
]: 0.8974358974358975
```

```
]: classifier_entropy=DecisionTreeClassifier(criterion='entropy')
   classifier_entropy.fit(x_train,y_train)
```

```
]:          DecisionTreeClassifier
   DecisionTreeClassifier(criterion='entropy')
```

```
]: x_train_scaler=scaler.transform(x_train)
   x_test_scaler=scaler.transform(x_test)
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

```
]: classifier_scaler=DecisionTreeClassifier(criterion='gini')
   classifier_scaler.fit(x_train_scaler,y_train)
```

```
]: DecisionTreeClassifier
   DecisionTreeClassifier()
```

```
classifier_scaler.score(x_test_scaler,y_test)
```

```
0.9230769230769231
```

```
patient1=(197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.09700,0.00563,0.00680,
          0.00802,0.01689,0.00339,26.77500,0.422229,0.741367,-7.348300,0.177551,1.743867,0.085569)
```

```
patient1=np.array([patient1])
patient1
```

```
array([[ 1.970760e+02,  2.068960e+02,  1.920550e+02,  2.890000e-03,
         1.000000e-05,  1.660000e-03,  1.680000e-03,  4.980000e-03,
         1.098000e-02,  9.700000e-02,  5.630000e-03,  6.800000e-03,
         8.020000e-03,  1.689000e-02,  3.390000e-03,  2.677500e+01,
         4.222290e-01,  7.413670e-01, -7.348300e+00,  1.775510e-01,
         1.743867e+00,  8.556900e-02]])
```

```
classifier .predict(patient1)
```

```
array([0], dtype=int64)
```

```
pred=classifier.predict(patient1)
if pred[0]==0:
    print("The Person does not have Parkinsons Disease")
else:
    print("The Person has Parkinsons")
```

```
The Person does not have Parkinsons Disease
```

K-FOLD CROSS VALIDATION

```
rf=RandomForestClassifier(n_estimators=40)
rf.fit(x_train,y_train)
rf.score(x_test,y_test)
```

```
0.9487179487179487
```

```
from sklearn.model_selection import cross_val_score
```

```
score_rf=cross_val_score(RandomForestClassifier(n_estimators=40),
                         x,y,cv=3)
print(score_rf)
print("Avg :",np.average(score_rf))
```

```
[0.81538462 0.87692308 0.78461538]
Avg : 0.8256410256410257
```

PARAMETER TUNING USING K- FOLD

```
scores1=cross_val_score(RandomForestClassifier(n_estimators=5),
                        x,y,cv=10)
print("Avg score for Estimators=5 and cv=10 :",np.average(scores1))
```

```
Avg score for Estimators=5 and cv=10 : 0.8084210526315789
```

```
scores1=cross_val_score(RandomForestClassifier(n_estimators=10),
                        x,y,cv=10)
print("Avg score for Estimators=10 and cv=10 :",np.average(scores1))
```

```
Avg score for Estimators=10 and cv=10 : 0.825
```

```
scores1=cross_val_score(RandomForestClassifier(n_estimators=20),
                        x,y,cv=10)
print("Avg score for Estimators=20 and cv=10 :",np.average(scores1))
```

```
Avg score for Estimators=20 and cv=10 : 0.8389473684210526
```

CONCLUSION

Parkinson's disease affects the CNS of the brain and has yet no treatment unless it's detected early. Late detection leads to no treatment and loss of life. Thus its early detection is significant. For early detection of the disease, we utilized machine learning algorithms such as Support Vector Machine and Random Forest, etc,. We checked our Parkinson disease data and find out Random Forest is the best Algorithm to predict the onset of the disease which will enable early treatment and save a life.

THANK YOU!