PARKINSON'S DISEASE DETECTION

Objective:

To identify whether a person is suffering from Parkinson.s disease or not.

Importing Dependencies

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score
```

The Data

In [2]:
```
parkinsons_data=pd.read_csv(r'C:\Users\HP\Downloads\archive\parkinsons.csv')
```

In [3]:
```
parkinsons_data.head()
```

Out[3]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shimmer:DDA	NHR	HNR	status	RPDE	DFA	spread1	sp
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...	0.06545	0.02211	21.033	1	0.414783	0.815285	-4.813031	0.2
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...	0.09403	0.01929	19.085	1	0.458359	0.819521	-4.075192	0.3
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...	0.08270	0.01309	20.651	1	0.429895	0.825288	-4.443179	0.3
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...	0.08771	0.01353	20.644	1	0.434969	0.819235	-4.117501	0.3
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...	0.10470	0.01767	19.649	1	0.417356	0.823464	-3.747787	0.2

5 rows × 24 columns

In [5]:
```
parkinsons_data.shape
```

Out[5]:
```
(195, 24)
```

This dataframe has 195 rows and 24 columns.

In [6]:
```
parkinsons_data.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
```

This dataframe has no null values.

Getting some statistical insights from the dataset

In [7]:
```
parkinsons_data.describe()
```

Out[7]:

	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:DDA	NHR	HNR	status	RPDE	
count	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	195.000000	195.000000	19
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	0.009920	0.029709	0.282251	...	0.046993	0.024847	21.885974	0.753846	0.498536	
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	0.194877	...	0.030459	0.040418	4.425764	0.431878	0.103942	
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	0.085000	...	0.013640	0.000500	8.441000	0.000000	0.256570	
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	0.148500	...	0.024735	0.005925	19.198000	1.000000	0.421306	
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	0.221000	...	0.038360	0.011660	22.085000	1.000000	0.495954	
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	0.350000	...	0.060795	0.025640	25.075500	1.000000	0.587562	
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	1.302000	...	0.169420	0.314820	33.047000	1.000000	0.685151	

8 rows × 23 columns

Distribution of target variable i.e. status in this case

In [8]:
```
parkinsons_data['status'].value_counts()
```

Out[8]:
```
1    147
0     48
Name: status, dtype: int64
```

1 --> Parkinson's positive 0 --> Healthy

Hence it can be concluded that in this dataset 147 people are suffering from Parkinson's disease while the rest i.e. 48 people are free from this disease.

Grouping the data based on the target variable

In [9]:
```
parkinsons_data.groupby('status').mean()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_10264\3507134583.py:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeri
c_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  parkinsons_data.groupby('status').mean()
```

Out[9]:

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)	...	MDVP:APQ	Shimmer:DDA	NHR	HNR	RPDE	DFA
0	181.937771	223.636750	145.207292	0.003866	0.000023	0.001925	0.002056	0.005776	0.017615	0.162958	...	0.013305	0.028511	0.011483	24.678750	0.442552	0.69571
1	145.180762	188.441463	106.893558	0.006989	0.000051	0.003757	0.003900	0.011273	0.033658	0.321204	...	0.027600	0.053027	0.029211	20.974048	0.516816	0.72540

2 rows × 22 columns

It can be clearly observed from these mean values that those who are healthy people have voice of higher frequency than those who are affected by Parkinson's disease.

Data Preprocessing

Seperating the features and target

In [10]:
```
X= parkinsons_data.drop(columns=['name','status'],axis=1)
Y= parkinsons_data['status']
```

In [11]:
```
print(X)
```

```
     MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
0        119.992       157.302        74.997         0.00784   
1        122.400       148.650       113.819         0.00968   
2        116.682       131.111       111.555         0.01050   
3        116.676       137.871       111.366         0.00997   
4        116.014       141.781       110.655         0.01284   
..           ...           ...           ...             ...   
190      174.188       230.978        94.261         0.00459   
191      209.516       253.017        89.488         0.00564   
192      174.688       240.005        74.287         0.01360   
193      198.764       396.961        74.904         0.00740   
194      214.289       260.277        77.973         0.00567   

     MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
0             0.00007   0.00370   0.00554     0.01109       0.04374   
1             0.00008   0.00465   0.00696     0.01394       0.06134   
2             0.00009   0.00544   0.00781     0.01633       0.05233   
3             0.00009   0.00502   0.00698     0.01505       0.05492   
4             0.00011   0.00655   0.00908     0.01966       0.06425   
..                ...       ...       ...         ...           ...   
190           0.00003   0.00263   0.00259     0.00790       0.04087   
191           0.00003   0.00331   0.00292     0.00994       0.02751   
192           0.00008   0.00624   0.00564     0.01873       0.02308   
193           0.00004   0.00370   0.00390     0.01109       0.02296   
194           0.00003   0.00295   0.00317     0.00885       0.01884   

     MDVP:Shimmer(dB)  ...  MDVP:APQ  Shimmer:DDA      NHR      HNR  \
0               0.426  ...   0.02971       0.06545  0.02211  21.033   
1               0.626  ...   0.04368       0.09403  0.01929  19.085   
2               0.482  ...   0.03590       0.08270  0.01309  20.651   
3               0.517  ...   0.03772       0.08771  0.01353  20.644   
4               0.584  ...   0.04465       0.10470  0.01767  19.649   
..                ...  ...       ...           ...      ...     ...   
190             0.405  ...   0.02745       0.07008  0.02764  19.517   
191             0.263  ...   0.01879       0.04812  0.01810  19.147   
192             0.256  ...   0.01667       0.03804  0.10715  17.883   
193             0.241  ...   0.01588       0.03794  0.07223  19.020   
194             0.190  ...   0.01373       0.03078  0.04398  21.209   

          RPDE       DFA   spread1   spread2        D2       PPE  
0     0.415785  0.815285 -4.813031  0.266482  2.301442  0.284654  
1     0.458359  0.819521 -4.075192  0.335590  2.486855  0.368674  
2     0.429895  0.825288 -4.443179  0.311173  2.342259  0.332634  
3     0.434969  0.819235 -4.117501  0.334147  2.405554  0.368975  
4     0.417356  0.823464 -3.747787  0.234513  2.332180  0.410335  
..         ...       ...       ...       ...       ...       ...  
190   0.657899  0.538586  0.121952  2.657476  0.133050  
191   0.683244 -0.195325  0.129303  2.784312  0.168895  
192   0.655683  0.787197  0.158453  2.679772  0.131728  
193   0.643956 -0.744577  0.207454  2.138608  0.123306  
194   0.664357 -5.724056  0.190667  2.555477  0.148569  

[195 rows x 22 columns]
```

In [12]:
```
print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
190    0
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64
```

Splitting the data to training data & test data

In [13]:
```
X_train , X_test , Y_train ,Y_test = train_test_split(X , Y, test_size=0.2 , random_state=2)
```

In [15]:
```
print(X.shape,X_train.shape,X_test.shape)
```

```
(195, 22) (156, 22) (39, 22)
```

Data Standardization

In [16]:
```
scaler= StandardScaler()
```

In [18]:
```
scaler.fit(X_train)
```

Out[18]:
```
StandardScaler()
```

In [19]:
```
X_train = scaler.transform(X_train)
```

In [20]:
```
X_test = scaler.transform(X_test)
```

In [21]:
```
print(X_train)
```

```
[[ 0.63239631 -0.02731081 -0.87085049 ... -0.97586547 -0.55160318
   0.07769494]
 [-1.05512719 -0.83337041 -0.9284778  ...  0.3981808  -0.61014073
   0.39291782]
 [ 0.02990187 -0.29531068 -1.12211107 ... -0.43937044 -0.62849685
  -0.50948408]
 ...
 [-0.9096785  -0.6637302  -0.160638   ...  1.22001022 -0.47404629
  -0.2159482 ]
 [-0.35977689  0.19731822 -0.70063679 ... -0.17896029 -0.47272835
   0.28181221]
 [ 1.01957066  0.19922317 -0.61914972 ... -0.716232    1.23632066
  -0.05829386]]
```

Model Training

Support Vector Machine Model

In [23]:
```
model=svm.SVC(kernel='linear')
```

Training the svm model with training data

In [24]:
```
model.fit(X_train,Y_train)
```

Out[24]:
```
SVC(kernel='linear')
```

Model Evaluation

Accuracy Score

Accuracy Score of training data

In [25]:
```
X_train_prediction=model.predict(X_train)
```

In [26]:
```
training_data_accuracy = accuracy_score(Y_train , X_train_prediction)
```

In [28]:
```
print('Accuracy Score of training data:',training_data_accuracy)
```

```
Accuracy Score of training data: 0.8846153846153846
```

Accuracy Score of test data

In [29]:
```
X_test_prediction=model.predict(X_test)
testing_data_accuracy = accuracy_score(Y_test , X_test_prediction)
print('Accuracy score of testing data:',testing_data_accuracy)
```

```
Accuracy score of testing data: 0.8717948717948718
```

Building a Predictive System

In [32]:
```
input_data=(197.076,206.896,192.055,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.097,0.00563,0.0068,0.00802,0.01689,0.00339,26.775,0.422229,0.741367,-7.3483,0.177551,1.743867,
)

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

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

# standardising 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 Parkinson's disease.")

else:
    print("The person is suffering from Parkinson's disease.")
```

```
[0]
The person does not have Parkinson's disease.
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]:
```

```