Description

The dataset was created by Max Little of the University of Oxford, in collaboration with the National Centre for Voice and Speech, Denver, Colorado, who recorded the speech signals. The original study published the feature extraction methods for general voice disorders.

```
1. Matrix column entries (attributes):
2. name - ASCII subject name and recording number
3. MDVP:Fo(Hz) - Average vocal fundamental frequency
4. MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
5. MDVP:Flo(Hz) - Minimum vocal fundamental frequency
6. MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several
7. measures of variation in fundamental frequency
8.MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DD
A - Several measures of variation in amplitude
9. NHR,HNR - Two measures of ratio of noise to tonal components in the voice
10. status - Health status of the subject (one) - Parkinson's, (zero) -
healthy
11. RPDE,D2 - Two nonlinear dynamical complexity measures
12. DFA - Signal fractal scaling exponent
13. spread1,spread2,PPE - Three nonlinear measures of fundamental frequency
variation
```

https://www.kaggle.com/datasets/thecansin/parkinsons-data-set

Step 1 : Importing the Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Step 2: Loading the dataset

```
# Loading the csv data to a Pandas DataFrame
parkinsons_data = pd.read_csv('/content/parkinsons.csv')
```

Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

3.1) Understanding Your Variables

3.1.1) Head of the dataset
3.1.2) The shape of the dataset
3.1.3) List types of columns
3.1.4) Info of the dataset
3.1.5) Summary of the dataset

3.1.1) Head of the Dataset

```
# Display first five records
parkinsons_data.head()

# Display last five records
parkinsons_data.tail()
```

3.1.2) The Shape of Dataset

```
parkinsons_data.shape
```

3.1.3)List types of columns

```
parkinsons_data.dtypes
```

3.1.4)Info of Dataset

```
# getting some info about the data
parkinsons_data.info()

# checking for missing values
parkinsons_data.isnull().sum()

# Statistical Summary
parkinsons_data.describe()

# checking the distribution of target Variable
parkinsons_data['status'].value_counts()
```

```
1 --> Parkinson's Positive


2 --> Healthy
```

```
# grouping the data based on the target variable
parkinsons_data.groupby('status').mean()
```

Step 4: Split the data frame in X & Y

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

X.head()

Y.head()
```

Step 5: Splitting the Data into Training data & Test Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=2)

print(X.shape, X_train.shape, X_test.shape)
```

Step 6: Building Classification Algorithm

6.1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,Y_train)
```

6.2) K-Nearest Neighbors Classifier(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,Y_train)
```

6.3) Naive-Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, Y_train)
```

6.4) Support Vector Machine (SVM)

```
from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train,Y_train)
```

6.5) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
```

6.6) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='entropy')
rf.fit(X_train,Y_train)
```

Step 7: Making Prediction

7.1) Making Prediction using Logistic Regression

```
print(f'Initial shape: {X_test.shape}')
lr_pred = lr.predict(X_test)
print(f'{lr_pred.shape}')
```

7.2) Making Prediction using KNN

```
knn_pred = knn.predict(X_test)
knn_pred.shape
```

7.3) Making Prediction using Naive Bayes

```
nb_pred = nb.predict(X_test)
nb_pred.shape
```

7.4) Making Prediction using SVM

```
sv_pred = sv.predict(X_test)
sv_pred.shape
```

7.5) Making Prediction using Decision Tree

```
dt_pred = dt.predict(X_test)
```

7.6) Making Prediciton using Random Forest

```
rf_pred = rf.predict(X_test)
```

Step 8: Model Evaluation

```
from sklearn.metrics import accuracy_score

# Train & Test Scores of Logistic Regression
print("Accuracy (Train) score of Logistic Regression
",lr.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Logistic Regression ",
lr.score(X_test,Y_test)*100)
print("Accuracy score of Logistic Regression ",
accuracy_score(Y_test,lr_pred)*100)

# Train & Test Scores of KNN
print("Accuracy (Train) score of KNN ",knn.score(X_train,Y_train)*100)
print("Accuracy (Test) score of KNN ", knn.score(X_test,Y_test)*100)
print("Accuracy score of KNN ", accuracy_score(Y_test,knn_pred)*100)

# Train & Test Scores of Naive-Bayes
print("Accuracy (Train) score of Naive Bayes ",nb.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Naive Bayes ", nb.score(X_test,Y_test)*100)
print("Accuracy score of Naive Bayes ", accuracy_score(Y_test,nb_pred)*100)

# Train & Test Scores of SVM
print("Accuracy (Train) score of SVM ",sv.score(X_train,Y_train)*100)
print("Accuracy (Test) score of SVM ", sv.score(X_test,Y_test)*100)
print("Accuracy score of SVM ", accuracy_score(Y_test,sv_pred)*100)

# Train & Test Scores of Decision Tree
print("Accuracy (Train) score of Decision Tree
",dt.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Decision Tree ", dt.score(X_test,Y_test)*100)
print("Accuracy score of Decision Tree ", accuracy_score(Y_test,dt_pred)*100)
```

```
# Train & Test Scores of Random Forest
print("Accuracy (Train) score of Random Forest
",rf.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Random Forest ", rf.score(X_test,Y_test)*100)
print("Accuracy score of Random Forest ", accuracy_score(Y_test,rf_pred)*100)
```

Step 9: Building a Predictive System

```
input_data =
(197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.0109
8,0.09700,0.00563,0.00680,0.00802,0.01689,0.00339,26.77500,0.422229,0.741367,
-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)

prediction = sv.predict(input_data_reshaped)
print(prediction)


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

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

Step 10: Saving the trained model

```
import pickle
filename = 'parkinsons_model.sav'
pickle.dump(sv, open(filename, 'wb'))
# loading the saved model
loaded_model = pickle.load(open('parkinsons_model.sav', 'rb'))
for column in X.columns:
  print(column)
```