Project Title:Detecting Parkinsons Disease using different models and finding which model is the best to detect early onset of Parkinsons Disease.

1. Project Description (in brief):
   Parkinson's disease is a progressive disorder of the central nervous system affecting movement and inducing tremors and stiffness. It has 5 stages to it and affects more than 1 million individuals every year in India. This is chronic and has no cure yet. It is a neurodegenerative disorder affecting dopamine-producing neurons in the brain. The main objective of this project is to understand what is Parkinson's disease and to detect the early onset of the disease. We will use here XGBoost, KNN Algorithm, Support Vector Machines (SVMs), Random Forest Algorithm,Decision tree and utilize the data-set available on UCL Parkinson Data-set under URL (Index of /ml/machine-learning-databases/Parkinsons (uci.edu)).

2. Type of problem: (supervised\Unsupervised), (Prediction\Classification) Why the problem falls into particular category
   Type of Problem – Supervised learning , Classification
   Since Parkinson's disease detection often requires using labelled data to train a machine learning model, it falls under the category of supervised classification. The objective of supervised classification is to categorise fresh, unobserved data based on the patterns discovered from a labelled training sample.
   When detecting Parkinson's disease, the labelled data may include patient records with confirmed diagnoses of the condition or lists of healthy people. After then, the algorithm would be trained to spot patterns in the data that point to Parkinson's disease, such tremors or trouble moving.
   The model can be used to predict whether new, unlabeled data falls into the Parkinson's disease class or not after being trained on the labelled data. Because it gives the model the chance to learn from examples of both healthy and unwell people, this method is excellent for identifying Parkinson's disease and can increase the precision with which it can categorise new instances.

3. Python Libraries used:
   - Numpy
   - Pandas
   - Sklearn
   - Seaborn
   - XG Boost

4. Data set details: Describe data set in detail. Description must be in terms of number of rows, features, meaning of features, source of the data set and how old data set is

   Source: 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.
   http://archive.ics.uci.edu/ml/datasets/Parkinsons/about.html

Data Set Information: This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD). Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals ("name" column). The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD.
The data is in ASCII CSV format. The rows of the CSV file contain an instance corresponding to one voice recording. There are around six recordings per patient, the name of the patient is identified in the first column.

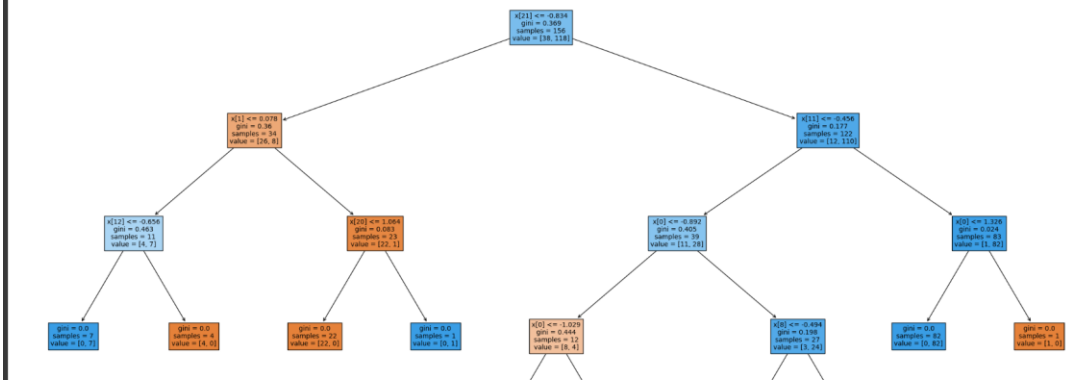
Attribute Information:

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

5. Data set visualization and inference

- Decision tree-A decision tree visualization is used to illustrate how underlying data predicts a chosen target and highlights key insights about the decision tree.

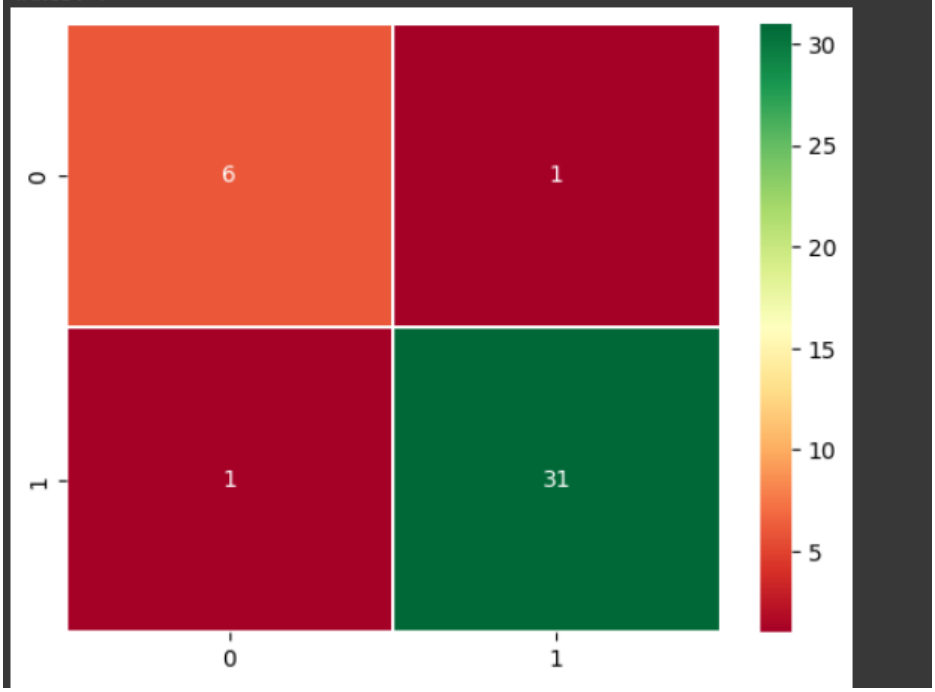
```
from sklearn import tree
fig = plt.subplots(figsize = (30,20))
tree.plot_tree(classifier,fontsize =10,filled = True)
plt.show()
```



- Plotted the Confusion Matrix as a heatmap. The x-axis and y-axis show the predicted and the actual output values, respectively.

```
import seaborn as sns
data=[[TP,FP],[FN,TN]]
sns.heatmap(data, cmap ='RdYlGn', linewidths = 0.30, annot = True)
```

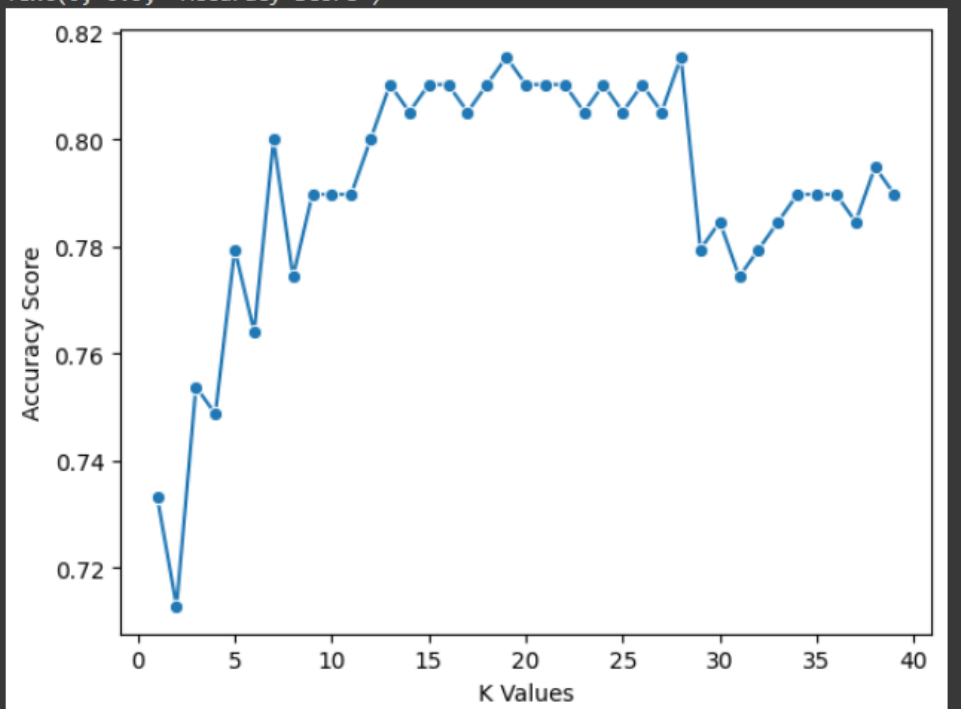
```
<Axes: >
```



- Line plot to visualise Accuracy score vs Kvalues in cross fold validation.

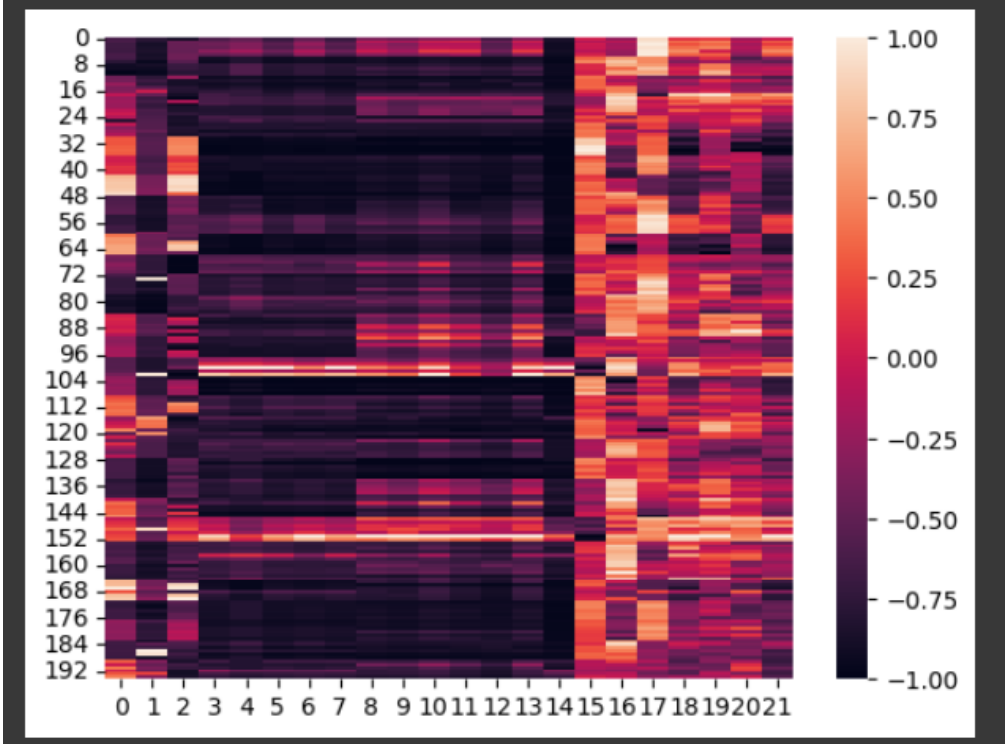
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.lineplot(x = k_values, y = scores, marker = 'o')
plt.xlabel("K Values")
plt.ylabel("Accuracy Score")
```

```
Text(0, 0.5, 'Accuracy Score')
```



- Heat map - Heat map is used as a two-dimensional representation of data in which values are represented by colors. A simple heat map provides an immediate visual summary of information.

```
hm = sns.heatmap(data = x)
```



6. Data Cleaning steps (Describe in detail data cleaning steps under taken. If required along with code)

   Replacing nan value, question marks and all such unwanted values from columns with mean of that column

```
def isNaN(string):
    return string != string
sum = 0
count = 0
for i in df['MDVP:Fo(Hz)']:
  if i == '??' or i == '???' or isNaN(i):
    continue;
  else:
    sum = sum + float(i);
    count = count +1;
mean = sum/count
mean
df['MDVP:Fo(Hz)'].replace(np.nan,round(mean,3),inplace=True)
df['MDVP:Fo(Hz)'].replace('??',round(mean,3),inplace=True)
df['MDVP:Fo(Hz)'].replace('???',round(mean,3),inplace=True)
```

   Replacing String ‘One’ and ‘zero’ with numeric 1 and 0 in the categorical status column.

```
df['status'].replace('zero','0',inplace=True)
df['status'].replace('one','1',inplace=True)

[632] df['status'].unique()

    array(['1', '0'], dtype=object)
```

7. Data Preprocessing steps (Describe in detail data pre-processing steps under taken. If required along with code)
   Dropping unwanted column like name from the data set.
   Feature engineering - Get the features and labels from the DataFrame (dataset). The features are all the columns except 'status', and the labels are those in the 'status' column.

```
[633] features=df.drop(["name","status"],axis=1)
      labels=df["status"]

[ ] #DataFlair - Get the count of each label (0 and 1) in labels
    print(labels[labels=='1'].shape[0], labels[labels=='0'].shape[0])

    147 48
```

8. Feature scaling\Normalization (if applied) – Which technique implemented why?
   MinMax Scalar – to scale the features to between -1 and 1 to normalize them.Used for XGBoost,Decision tree ,Random Forest and Support Vector machine. MinMaxScaler is a technique used for feature scaling in machine learning, which rescales the values of the features to a specified range. In XGBoost, using MinMaxScaler for feature scaling can be beneficial in improving the performance of the model.

```
#Initialize a MinMaxScaler and scale the features to between -1 and 1 to normalize them.
#The MinMaxScaler transforms features by scaling them to a given range.
#The fit_transform() method fits to the data and then transforms it. We don't need to scale the labe
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler((-1,1))
x=scaler.fit_transform(features)
```

   Standard scalar - The need for StandardScaler in machine learning arises from the fact that many algorithms are sensitive to the scale of the input features. StandardScaler rescales the features to have a mean of 0 and a standard deviation of 1, which can prevent features with large values from dominating the learning process, improve the accuracy of the model, and make it more interpretable.
   In K-Nearest Neighbors (KNN) algorithm, feature scaling using StandardScaler is important because the distance metric used by KNN to calculate the similarity between data points is sensitive to the scale of the input features. StandardScaler can help normalize the features and prevent some features from dominating the distance calculation, leading to more accurate predictions.

```
from sklearn.preprocessing import StandardScaler
# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

9. Model building- Describe in detail what all models were build, train and test split size. If required put sample code for model building.

XGboost - XGBoost is an algorithm. That has recently been dominating applied gadget learning. XGBoost set of rules is an implementation of gradient boosted choice timber. That changed into the design for pace and overall performance. In this Python machine learning project, using the Python libraries scikit-learn, numpy, pandas, and xgboost, we will build a model using an XGBClassifier. We'll load the data, get the features and labels, scale the features, then split the dataset, build an XGBClassifier, and then calculate the accuracy of our model.

Test size – 20%(0.2)

```
#DataFlair - Split the dataset
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=7)
```

	True Healthy	True Parkinsons
Predicted Healthy	6	1
Predicted Parkinsons	1	31

Support Vector Machine - Another algorithm for the analysis of classification and regression is the support vector machine. It is a supervised machine algorithm used. Image classification and hand-written recognition are where the support vector machine comes to hand used. It sorts the data in one out of two categories and displays the output with the margin between the two as far as possible.

```
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=7)
from sklearn import svm
clf = svm.SVC()
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print(accuracy_score(y_test, y_pred)*100)
```

	True Healthy	True Parkinsons
Predicted Healthy	6	2
Predicted Parkinsons	5	26

K Nearest Neighbor - K-Nearest Neighbors (KNN) algorithm, is one of the most powerful utilized algorithms of machine learning that is widely used both for regression as well as classification tasks. In order to predict and examine the class in which data points fall, it examines the label of chosen data points surrounded by the target point.

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state=2)
from sklearn.preprocessing import StandardScaler
# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = knn.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8205128205128205
```

	True Healthy	True Parkinsons
Predicted Healthy	2	5
Predicted Parkinsons	0	32

```
x1=df_new_1.drop('status',axis=1)
y1=df_new_1['status']
```

```
X1_train, X1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.2,random_state=3)
from sklearn.preprocessing import StandardScaler
# Scale the features using StandardScaler
scaler = StandardScaler()
X1_train = scaler.fit_transform(X1_train)
X1_test = scaler.transform(X1_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X1_train, y1_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
y1_pred = knn.predict(X1_test)
```

```
accuracy = accuracy_score(y1_test, y1_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9230769230769231
```

	Predicted Healthy	Predicted Parkinsons
True Healthy	6	1
True Parkinsons	2	30

Decision Tree - A decision tree is a flowchart-like tree structure where an internal node represents a feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.

The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision-making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

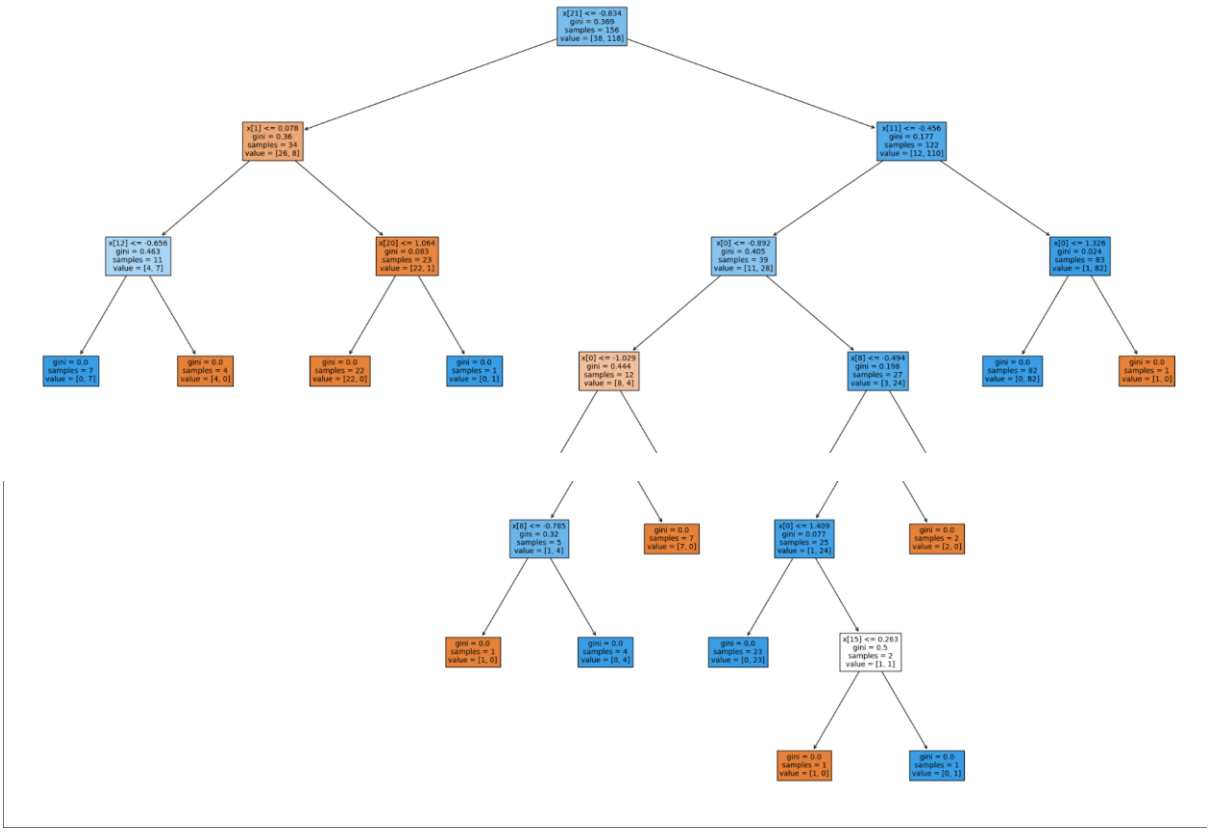
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'gini',random_state=0)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state=1)
from sklearn.preprocessing import StandardScaler
# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
#perform training
classifier = classifier.fit(X_train , y_train)
```

```
y_pred = classifier.predict(X_test)
print(y_pred)
```

```
[1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0
 1 0]
```

```
from sklearn import tree
fig = plt.subplots(figsize = (30,20))
tree.plot_tree(classifier,fontsize =10,filled = True)
plt.show()
```



Random Forest - Random forests are an ensemble version of many choice bushes, wherein each tree will specialize its focus on a specific feature while maintaining a top-level view of all capabilities. Each tree within the random wooded area will do its own random train/check

break up of the information, referred to as bootstrap aggregation and the samples no longer covered are called the ‘out-of-bag samples. Moreover, every tree will do characteristic bagging at every node-branch split to lessen the results of a characteristic mostly correlated with the response. While an individual tree is probably touchy to outliers, the ensemble version will no longer be the same.

```
X = df.drop('status', axis=1)


X = X.drop('name', axis=1)

y = df['status']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=7,test_size=0.2)
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(n_estimators=30, max_depth=10, random_state=1)
random_forest.fit(X_train, y_train)


from sklearn.metrics import accuracy_score

y_predict = random_forest.predict(X_test)

accuracy_score(y_test, y_predict)
```

```
0.9230769230769231
```

	Predicted Healthy	Predicted Parkinsons
True Healthy	6	1
True Parkinsons	2	30

10. Model evaluation – Comparative description regarding training and testing accuracy depending

```
y_pred=model.predict(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred)*100)
```

```
94.87179487179486
```

Figure 1:XGBoost accuracy

```
print(accuracy_score(y_test, y_pred)*100)
```

```
87.17948717948718
```

Figure 2: SVM Accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8205128205128205
```

Figure 3: KNN Accuracy with all attributes

```
accuracy = accuracy_score(y1_test, y1_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9230769230769231
```

Figure 4: KNN Accuracy with only a few attributes

```
y_predict = random_forest.predict(X_test)

accuracy_score(y_test, y_predict)
```

```
0.9230769230769231
```

Figure 5: Random Forest Accuracy

11. Attach collab code file (ipynb file as object)
12. 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 XGBoost,KNN,Suport vector machine and Random Forest. We checked our Parkinson disease data and find out XGBoost is the best

Algorithm to predict the onset of the disease which will enable early treatment and save a life.

13. References

http://archive.ics.uci.edu/ml/datasets/Parkinsons/about.html
https://scikit-learn.org/stable/modules/svm.html
https://scikit-learn.org/stable/modules/neighbors.html
https://scikit-learn.org/stable/modules/tree.html
https://scikit-learn.org/stable/modules/preprocessing.html
https://www.datacamp.com/tutorial/xgboost-in-python