Parkinson’s disease detection using machine learning

Internship project
submitted by :

K.janardhan(NITW-22ECB0C29)

B.trivikram(VIT)

Under guidance of

Prof.L.Anjaneyulu
Professor
Department of electronics and communication engineering
National instute of technology warangal

July 2024

Parkinson’s Disease Detection Using Machine Learning

Abstract:

Parkinson's Disease (PD) is the alternate most common age- related neurological complaint that leads to a range of motor and cognitive symptoms. A PD opinion is delicate since its symptoms are relatively analogous to those of other diseases, similar as normal aging and essential earthquake. When people reach 50, visible symptoms similar as difficulties walking and communicating begin to crop. Indeed though there's no cure for PD, certain specifics can relieve some of the symptoms. Cases can maintain their cultures by controlling the complications caused by the complaint. At this point, it's essential to descry this complaint and help it from progressing. The opinion of the complaint has been the subject of important exploration. In our design, we aim to descry PD using different types of Machine literacy (ML), models similar as Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT) to separate between healthy and PD cases by voice signal features. The dataset taken from Kaggle. Parkinson’s Disease (PD) is a degenerative neurological disorder marked by decreased dopamine levels in the brain. It manifests itself through a deterioration of movement, including the presence of tremors and stiffness. There is commonly a marked effect on speech, including dysarthria (difficulty articulating sounds), hypophonia (lowered volume), and monotone (reduced pitch range). Additionally, cognitive impairments and changes in mood can occur, and risk of dementia is increased.

Traditional diagnosis of Parkinson’s Disease involves a clinician taking a neurological history of the patient and observing motor skills in various situations. Since there is no definitive laboratory test to diagnose PD, diagnosis is often difficult, particularly in the early stages when motor effects are not yet severe. Monitoring progression of the disease over time requires repeated clinic visits by the patient. An effective screening process, particularly one that doesn’t require a clinic visit, would be beneficial. Since PD patients exhibit characteristic vocal features, voice recordings are a useful and non-invasive tool for diagnosis. If machine learning algorithms could be applied to a voice recording dataset to accurately diagnosis PD, this would be an effective screening step prior to an appointment with a clinician.

Keywords: Parkinson's disease, machine learning, Support Vector Machine (SVM)

Introduction:

Millions of individualities worldwide are affected by Parkinson's Disease(PD), a precipitously deteriorating complaint in which symptoms appear gradationally over time. While visible symptoms do in people over the age of 50, roughly one in every ten people shows signs of this complaint before the age of 40(Marton, 2019). Parkinson's complaint causes the death of specific whim-whams cells in the brain's substantia nigra, which induce chemical dopamine for directing fleshly movements.

Dopamine insufficiency causes fresh progressive symptoms to crop gradationally over time. generally, PD symptoms begin with temblors or stiffness on one side of the body, similar as the hand or arm. individualities with PD may acquire madness at after stages (Tolosa etal., 2006). From 1996 to 2016, the global frequence of PD more than quadrupled, from2.5 million to 6.1 million individualities. Increased life expectation has redounded in an aged population, which explains the substantial rise (Fothergill- Misbah etal., 2020). The brain is the body's controlling organ. Trauma or sickness to any portion of the brain will manifest in a variety of ways in multitudinous other sections of the body. PD causes a range of symptoms, including partial or complete loss of motor revulsions, speech problems and eventual failure, odd geste, loss of internal thinking, and other critical chops. It's delicate to distinguish between typical cognitive function losses associated with aging and early PD symptoms. In the United States, the overall profitable impact in 2017 was prognosticated to be$51.9 billion, including an circular cost of$14.2 billion,non-medical expenditures of$7.5 billion, and$4.8 billion accruing to disability income for proprietor's public workshop. The maturity of Parkinson's complaint cases are over the age of 65, and the overall profitable burden is anticipated to approach$ 79 billion by 2037 (Yang etal., 2020). The opinion of PD in National uniting Centre for habitual Conditions(2006) is generally grounded on a many invasive ways as well as empirical testing and examinations. Invasive individual procedures for PD are exceedingly precious, hamstrung, and bear extremely complex outfit with poor delicacy. New ways are demanded to diagnose PD. thus, less precious, simplified, and dependable styles should be acclimated to diagnose complaint and insure treatments. still, noninvasive opinion ways for PD bear being delved . Machine literacy ways are used to classify people with PD and healthy people.

It has been determined that diseases' oral issues can be assessed for early PD discovery(Harel etal., 2004). So, this study attempts to identify Parkinson's complaint(PD) by exercising Machine literacy(ML) and Deep literacy(DL) models to distinguish between healthy and PD cases grounded on voice signal features, maybe lowering some of these expenditures.

Related Work:

Several experimenters have classified Parkinson's complaint using colorful styles. These studies give a solid foundation for how machine literacy can be applied to neurodegenerative conditions in the face of current challenges in Parkinson's complaint subclassification, threat assessment, and prognostic using voice signal features. Selection and bracket procedures are used in the(Senturk, 2020) opinion fashion. The point selection task took into consideration the methodologies of point significance and Recursive point Elimination. Artificial neural networks, support vector machines, and bracket and retrogression trees were all employed in the trials to classify Parkinson's cases. Performance comparisons of the different ways revealed that Support Vector Machines with Recursive point Elimination outperformed them. With the smallest ditty features necessary to diagnose Parkinson's,93.84 delicacy was attained. The results of the styles handed by Gil and Manuel (2009) grounded on artificial neural networks and support vector machines to prop specialists in the opinion of Parkinson's complaint indicate a high delicacy of about 90. Das(2010) compared colorful bracket ways for the purpose of making an accurate Parkinson's complaint opinion. The paper's ideal is to efficiently identify healthy individualities. A relative study was carried out. There were four different bracket schemes used. These are, in order, Decision Trees, Retrogression, Neural Networks, and DMneural. The performance score of the classifiers was determined using a variety of evaluation ways. The neural network classifier produces the stylish issues, as determined by the operation scores. The neural network's overall bracket performance is92.9. A deep belief network(DBN) has been used as a successful system to identify Parkinson's complaint in the paper by Al- Fatlawi etal.(2016). The deep belief network(DBN), which is used to produce a template match of the voices, has been configured to accept input from a point birth procedure. Using two piled confined Boltzmann Machines(RBMs) and one affair subcaste, DBN is employed in this study to classify Parkinson's illness. To maximize the networks' parameters, two stages of literacy must be used. Unsupervised literacy, the first stage, uses RBMs to address the issue that can arise from the original weights' changeable original value. Secondly, the backpropagation fashion is employed for the fine tuning as a supervised literacy approach. The experimental results are varied with colorful strategies and affiliated work to demonstrate the efficacity of the suggested system. The proposed approach outperforms all other styles in comparison with its 94 total testing delicacy. Rasheed etal.(2020) proposed two bracket schemes to ameliorate the delicacy of PD case identification from voice measures. They began by applying a variable adaptive moment- grounded backpropagation algorithm to BPVAM, an artificial neural network. The experimenters also delved the use of dimensionality reduction styles similar as top element analysis(PCA) in confluence with BPVAM to classify the same dataset.

Material and methods:

a. Dataset:

"The dataset used in this project is sourced from Kaggle and consists of biomedical voice measurements from 31 individuals, including 23 diagnosed with Parkinson's disease (PD). Each row in the dataset represents one of the 195 voice recordings collected from these individuals, identified by a unique 'name' column. The dataset includes various voice measures as columns, with the primary objective being the differentiation between healthy individuals and those with PD. The 'status' column serves as the target variable, where individuals with PD are labeled as 1 and healthy individuals as 0. This dataset provides a valuable resource for training and evaluating machine learning models aimed at detecting Parkinson's disease based on voice signal features."

Parkinson Data and Voice Disorder

Voice disorder dataset can be used to detect the presence of Parkinson's disease in an individual. While current tools have limitations in analyzing complex voice disorders, advancements in technology and research have enabled the development of new algorithms that can identify specific acoustic markers associated with Parkinson's disease in voice recordings. Therefore, the analysis of voice disorders can provide valuable information in diagnosing and monitoring Parkinson's disease.

This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD). Our dataset includes voice attributes Information that can be used for detecting parkinson, these information including:

Voice measure	Meaning
Name	ASCII name of subject and recording number (categorical variables).
MDVP:Fo(Hz)	Average vocal fundamental frequency (Numerical variables).
MDVP:Fhi(Hz)	Maximum vocal fundamental frequency (Numerical variables).
MDVP:Flo(Hz)	Minimum vocal fundamental frequency (Numerical variables).
MDVP:Jitter(%)	Percentage of cycle-to-cycle variability of the period duration
MDVP:Jitter(Abs)	Absolute value of cycle-to-cycle variability of the period duration
MDVP: RAP	Several measures of variation in fundamental frequency (Numerical variables).
MDVP: PPQ	Pitch perturbation quotient
Jitter:DDP	Average absolute difference of differences between jitter cycles
MDVP:Shimmer	Variations in the voice amplitdue
MDVP:Shimmer(dB)	Variations in the voice amplitdue in dB
Shimmer: APQ3	Several measures of variation in amplitude (Numerical variables).
Shimmer: APQ5	Five point amplitude perturbation quotient measured against the average of the three amplitude
MDVP: APQ	Amplitude perturbation quotient from MDVP
Shimmer:DDA	Average absolute difference between the amplitudes of consecutive periods
NHR	Noise-to-harmonics Ratio
HNR	Harmonics-to-noise Ratio
status	Health status of the subject (one) - Parkinson's, (zero) - healthy
RPDE	Nonlinear dynamical complexity measures (Numerical variables).
D2	correlation dimension

Voice measure	Meaning
DFA	Signal fractal scaling exponent (Numerical variables).
spread1	discrete probability distribution of occurrence of relative semitone variations
spread2	Nonlinear measures of fundamental frequency variation (Numerical variables).
PPE	Entropy of the discrete probability distribution of occurrence of relative semitone variations

b. Data Processing

Preprocessing is the most important aspect of data processing, which helps the model learn the features of the data effectively and remove unnecessary information. The dataset was imported into the Google Colab platform as a CSV file using the Pandas package. After we screened for any duplicates or null entries, we used the "status" column and found that the dataset was imbalanced with 147 for PD and 48 for HC, which is equivalent to 25% for HC and 75% for PD. In order to avoid under-fitting and over-fitting, we split our dataset into a ratio of 80:20 train/test split. The training set includes known outputs, and what the model learns from it may be extended to other data sets. By computing the relevant statistics on the samples in the training set, each feature is scaled individually. The mean and standard deviation are then saved and utilized on later data using the transform in StandardScaler. Equation express the mathematical form of StandardScaler normalization. For this study, we employed a variety of libraries, including NumPy, Pandas, Matplotlib, Seaborn, and Sickit-learn (Sklearn). Numpy is Python's fundamental package for scientific computation. It is used to insert any form of mathematical operation into the code. Also, it allows you to include large multidimensional arrays and matrices in your code. The Pandas library is excellent for data manipulation and analysis; it is extensively used for importing and organizing datasets. Matplotlib and Seaborn are the foundations of Python data visualization. Matplotlib is a Python library that can be used to plot 2D graphs with the help of other libraries such as Numpy and Pandas. Seaborn is used to plot graphs using Matplotlib, Pandas, and Numpy. The last one is Sklearn, the most usable and robust machine learning package in Python. It provides a Python-based consistency interface as well as tools for classification, regression, clustering, and dimensionality reduction (Desai, 2019).

Importing Required Libraries

Using Machine Learning to Analyze Voice Disorders for Parkinson's Disease Detection

The purpose of this project is to develop a machine learning model that can accurately predict the presence of Parkinson's disease in an individual based on their voice recordings. Parkinson's disease is a neurodegenerative disorder that affects movement, with symptoms that include tremors, stiffness, and difficulty with coordination.

Table of Contents

Objectives

After completing this lab you will be able to:

- Use Python for data analysis and machine learning
- Implement machine learning algorithms to detect Parkinson's disease in voice recordings
- Evaluate model performance
- Conduct grid search for tuning parameters
- Visualize the decision tree model

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
%matplotlib inline
from sklearn.metrics import
accuracy_score,confusion_matrix,ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
```

Creating helper function for plotting

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

sns.set(style="whitegrid", color_codes=True)
import itertools



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = 3*cm.max()/4
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

df = pd.read_csv(r'D:/parkinsons data.csv')

df.head()
```

Loading data

```
             name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
0  phon_R01_S01_1      119.992       157.302        74.997         0.00784
1  phon_R01_S01_2      122.400       148.650       113.819         0.00968
2  phon_R01_S01_3      116.682       131.111       111.555         0.01050
3  phon_R01_S01_4      116.676       137.871       111.366         0.00997
4  phon_R01_S01_5      116.014       141.781       110.655         0.01284

   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  ...

   Shimmer:DDA      NHR     HNR  status      RPDE       DFA   spread1  \
0      0.06545  0.02211  21.033       1  0.414783  0.815285 -4.813031
1      0.09403  0.01929  19.085       1  0.458359  0.819521 -4.075192
2      0.08270  0.01309  20.651       1  0.429895  0.825288 -4.443179
3      0.08771  0.01353  20.644       1  0.434969  0.819235 -4.117501
4      0.10470  0.01767  19.649       1  0.417356  0.823484 -3.747787

    spread2        D2       PPE
0  0.266482  2.301442  0.284654
1  0.335590  2.486855  0.368674
2  0.311173  2.342259  0.332634
3  0.334147  2.405554  0.368975
4  0.234513  2.332180  0.410335

[5 rows x 24 columns]
```

```
df.shape
```

```
(195, 24)
```

```
df.info()
```

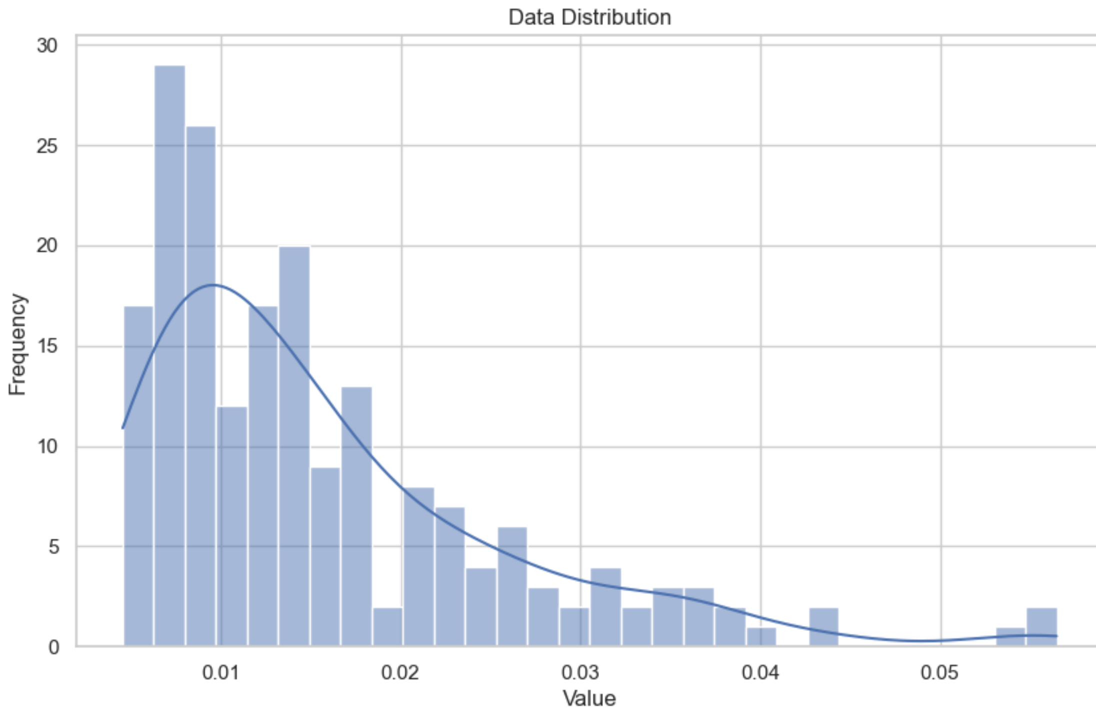
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              195 non-null    object
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      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
```

```python
df.dropna(inplace=True)
```

```python
df.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```python
plt.figure(figsize=(10, 6))
sns.histplot(df['Shimmer:APQ3'], kde=True, bins=30)
plt.title('Data Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```




```
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.describe()
```

```
       MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
count   195.000000    195.000000    195.000000      195.000000   
mean    154.228641    197.104918    116.324631        0.006220   
std      41.390065     91.491548     43.521413        0.004848   
min      88.333000    102.145000     65.476000        0.001680   
25%     117.572000    134.862500     84.291000        0.003460   
50%     148.790000    175.829000    104.315000        0.004940   
75%     182.769000    224.205500    140.018500        0.007365   
max     260.105000    592.030000    239.170000        0.033160   

       MDVP:Jitter(Abs)    MDVP:RAP    MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
count        195.000000  195.000000  195.000000  195.000000    195.000000   
mean           0.000044    0.003306    0.003446    0.009920      0.029709   
std            0.000035    0.002968    0.002759    0.008903      0.018857   
min            0.000007    0.000680    0.000920    0.002040      0.009540   
25%            0.000020    0.001660    0.001860    0.004985      0.016505   
50%            0.000030    0.002500    0.002690    0.007490      0.022970   
75%            0.000060    0.003835    0.003955    0.011505      0.037885   
max            0.000260    0.021440    0.019580    0.064330      0.119080   

       MDVP:Shimmer(dB)  ...  Shimmer:DDA         NHR         HNR      status
\
count        195.000000  ...   195.000000  195.000000  195.000000  195.000000   
mean           0.282251  ...     0.046993    0.024847   21.885974    0.753846   
std            0.194877  ...     0.030459    0.040418    4.425764    0.431878   
min            0.085000  ...     0.013640    0.000650    8.441000    0.000000   
25%            0.148500  ...     0.024735    0.005925   19.198000    1.000000   
50%            0.221000  ...     0.038360    0.011660   22.085000    1.000000   
75%            0.350000  ...     0.060795    0.025640   25.075500    1.000000   
max            1.302000  ...     0.169420    0.314820   33.047000    1.000000   

             RPDE         DFA     spread1     spread2          D2         PPE
count  195.000000  195.000000  195.000000  195.000000  195.000000  195.000000
mean     0.498536    0.718099   -5.684397    0.226510    2.381826    0.206552
std      0.103942    0.055336    1.090208    0.083406    0.382799    0.090119
min      0.256570    0.574282   -7.964984    0.006274    1.423287    0.044539
25%      0.421306    0.674758   -6.450096    0.174351    2.099125    0.137451
50%      0.495954    0.722254   -5.720868    0.218885    2.361532    0.194052
75%      0.587562    0.761881   -5.046192    0.279234    2.636456    0.252980
max      0.685151    0.825288   -2.434031    0.450493    3.671155    0.527367

[8 rows x 23 columns]
```

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

```
status
1    147
```

```
0     48
Name: count, dtype: int64
```

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

```
X.head()
```

```
   MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  MDVP:Jitter(Abs)  \
0      119.992       157.302        74.997         0.00784           0.00007
1      122.400       148.650       113.819         0.00968           0.00008
2      116.682       131.111       111.555         0.01050           0.00009
3      116.676       137.871       111.366         0.00997           0.00009
4      116.014       141.781       110.655         0.01284           0.00011

   MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  MDVP:Shimmer(dB)  ...  \
0   0.00370   0.00554     0.01109       0.04374             0.426  ...
1   0.00465   0.00696     0.01394       0.06134             0.626  ...
2   0.00544   0.00781     0.01633       0.05233             0.482  ...
3   0.00502   0.00698     0.01505       0.05492             0.517  ...
4   0.00655   0.00908     0.01966       0.06425             0.584  ...

   MDVP:APQ  Shimmer:DDA      NHR     HNR      RPDE       DFA   spread1  \
0   0.02971      0.06545  0.02211  21.033  0.414783  0.815285 -4.813031
1   0.04368      0.09403  0.01929  19.085  0.458359  0.819521 -4.075192
2   0.03590      0.08270  0.01309  20.651  0.429895  0.825288 -4.443179
3   0.03772      0.08771  0.01353  20.644  0.434969  0.819235 -4.117501
4   0.04465      0.10470  0.01767  19.649  0.417356  0.823484 -3.747787

    spread2        D2       PPE
0  0.266482  2.301442  0.284654
1  0.335590  2.486855  0.368674
2  0.311173  2.342259  0.332634
3  0.334147  2.405554  0.368975
4  0.234513  2.332180  0.410335

[5 rows x 22 columns]
```

```
y=df['status']
```

```
y.head()
```

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

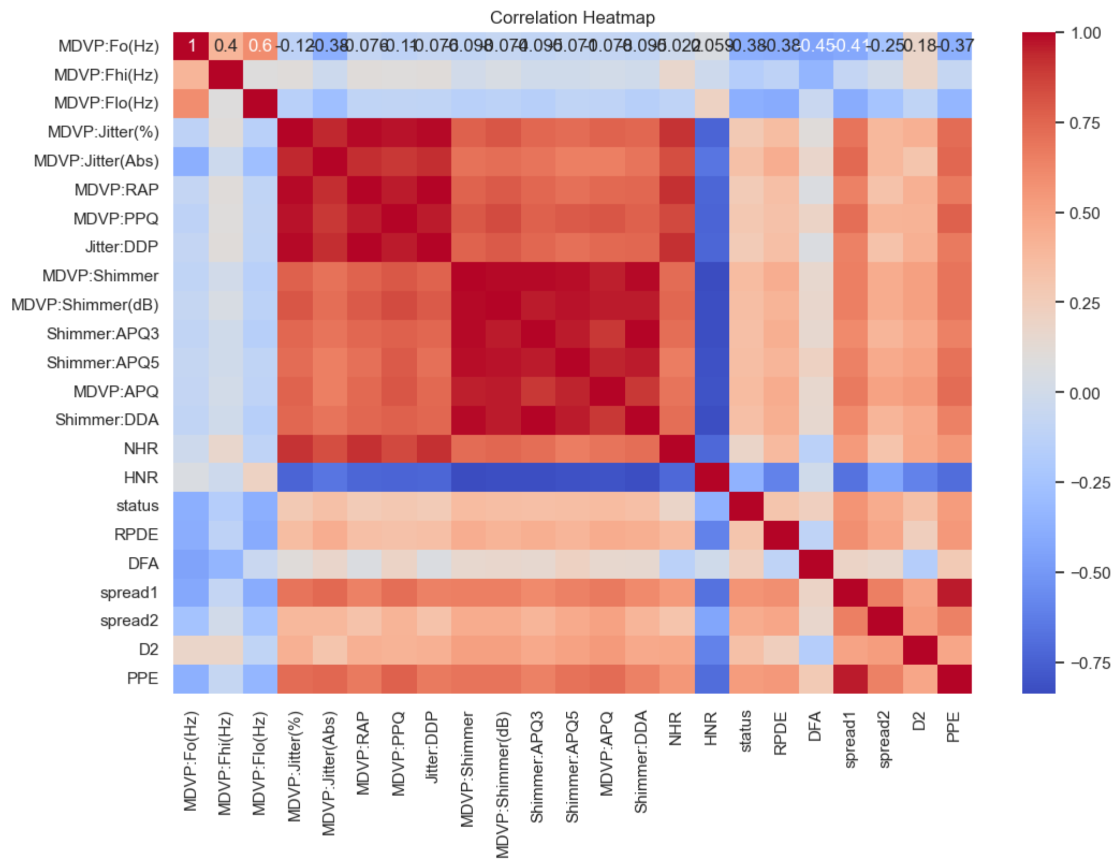
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state
=42)
```

```
# print the shape of train and test data
print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (156, 22)
y_train shape:  (156,)
X_test shape:  (39, 22)
y_test shape:  (39,)
```

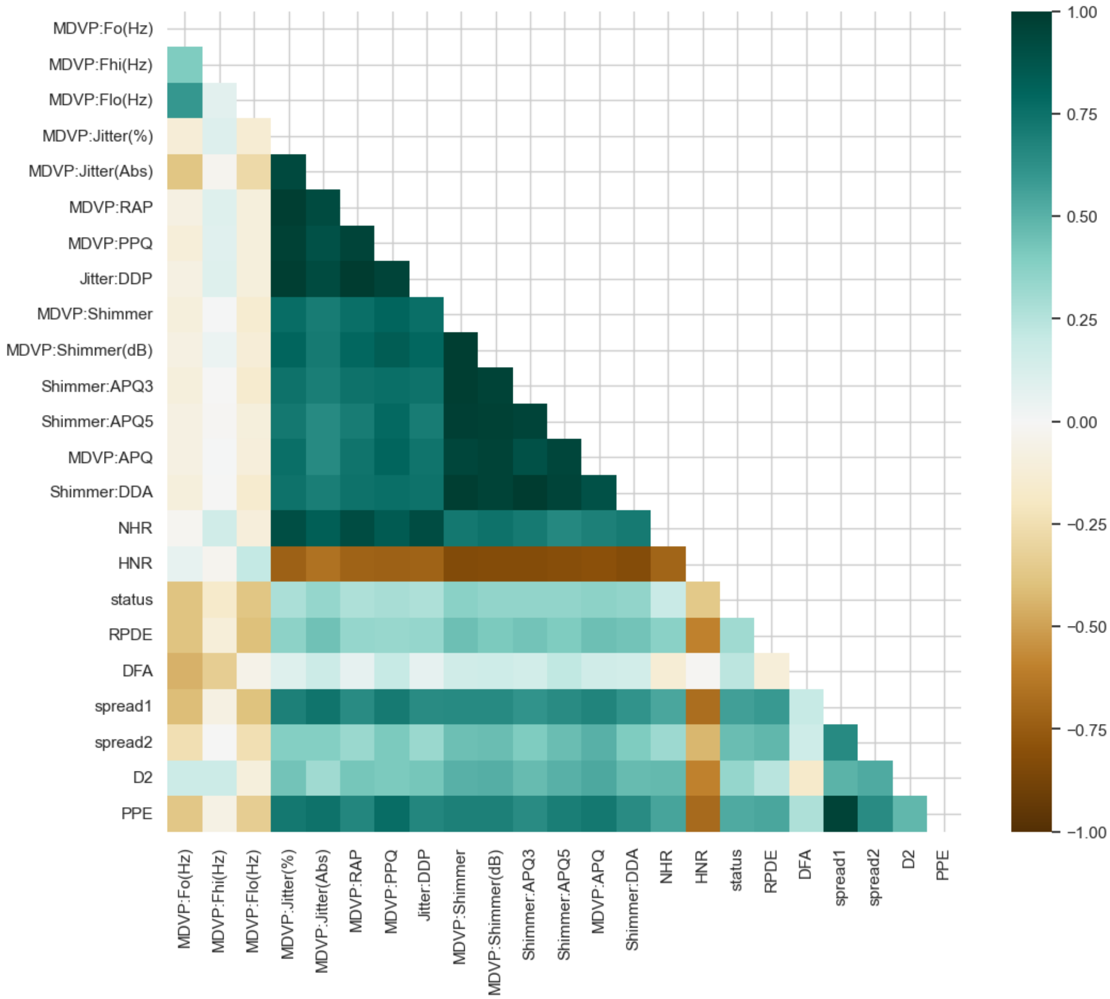
To improve our understanding of the variables involved in parkinson detection, we first need to analyze the relationships within the data. Correlation diagrams can be helpful in visualizing how different variables are associated with each other and with parkinson status. Additionally, random forest models can help identify the importance of different features in predicting the target variable (parkinson).

```
# 2. Correlation Heatmap
plt.figure(figsize=(12, 8))
numeric_df = df.select_dtypes(include=[np.number])
correlation_matrix = numeric_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

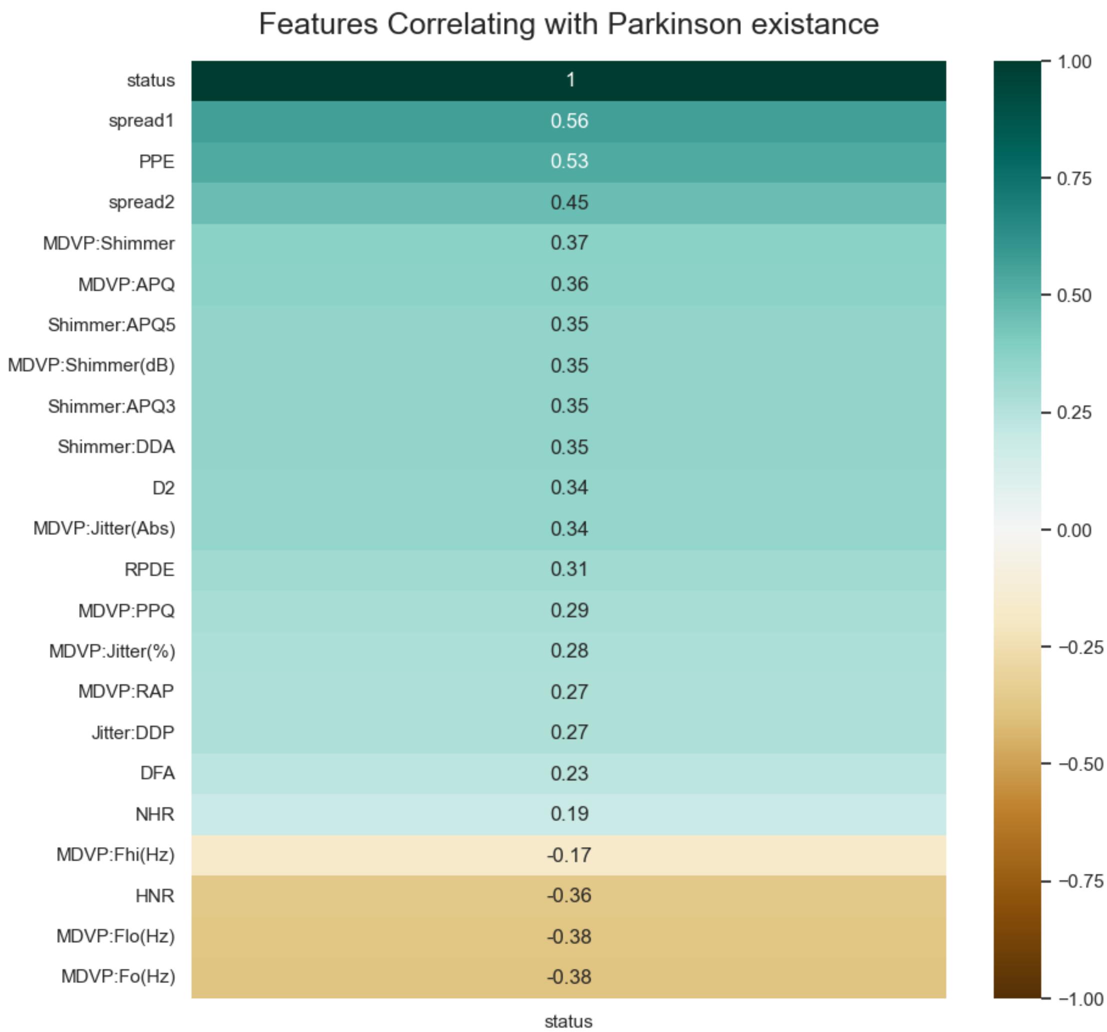



```python
# creating the correlation matrix
plt.figure(figsize=(12, 10))
numeric_df = df.select_dtypes(include=[np.number])
mask = np.triu(np.ones_like(numeric_df.corr()))
sns.heatmap(numeric_df.corr(),vmin=-1, vmax=1,cmap='BrBG', mask=mask)
```

```
<Axes: >
```



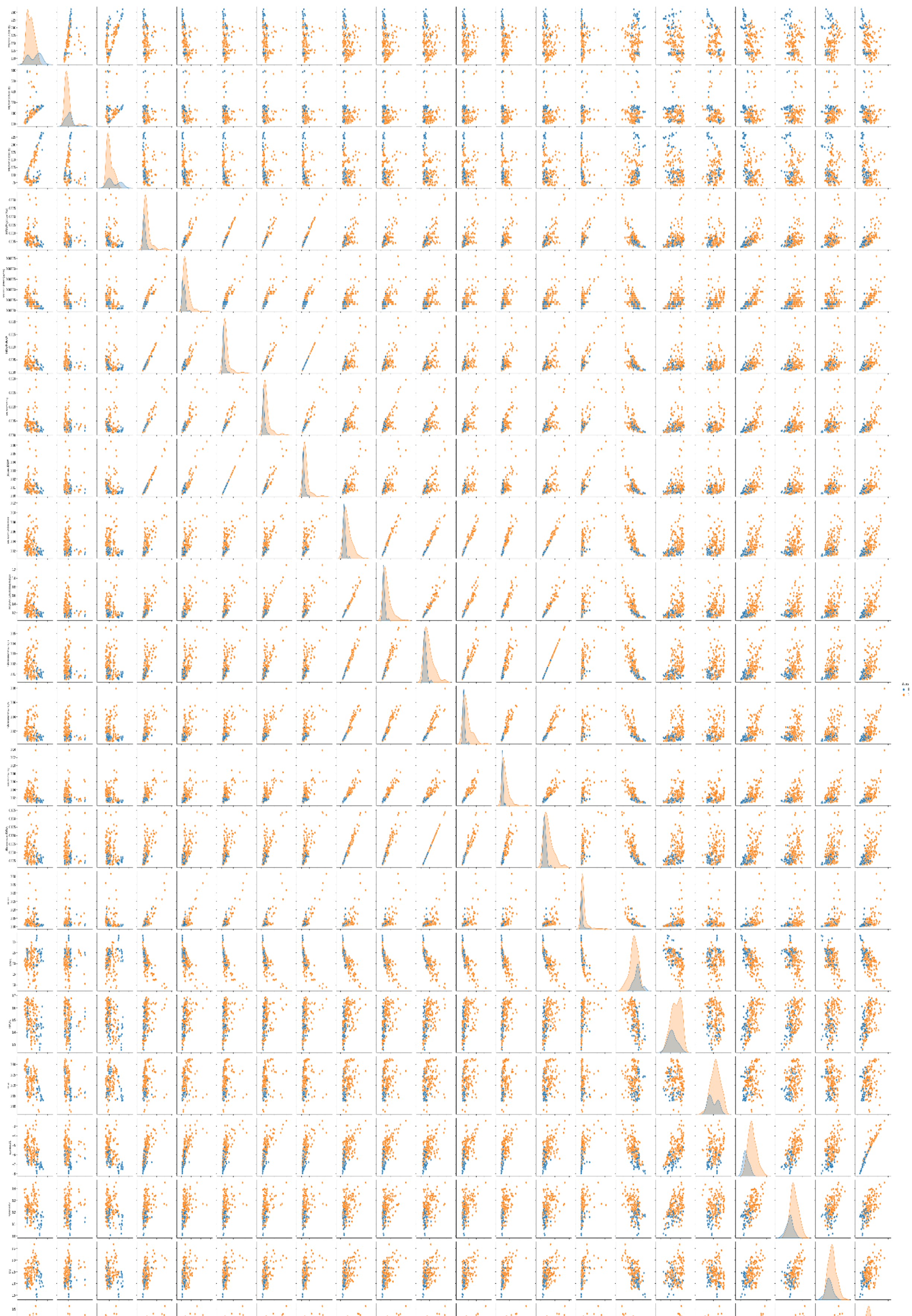

```
plt.figure(figsize=(10, 10))
heatmap = sns.heatmap(numeric_df.corr()[['status']].sort_values(by='status',
ascending=False), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Features Correlating with Parkinson existance',
fontdict={'fontsize':18}, pad=16);
```



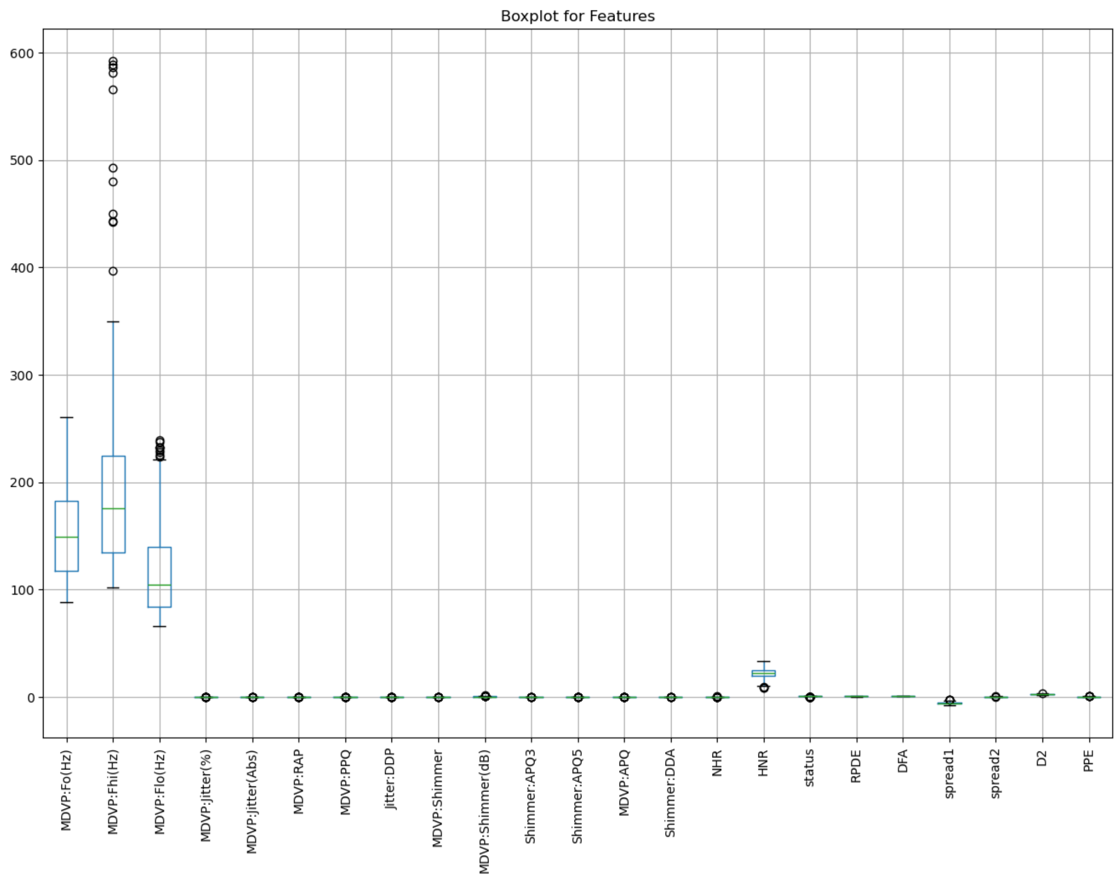

```python
# df.dropna(inplace=True)
# df.replace([np.inf, -np.inf], np.nan, inplace=True)
# sns.pairplot(df.drop(columns=['name']), hue='status')
# plt.show()
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Let's get the features we select all columns in the dataset except for the `status` column. This is done using the `drop` method, which returns a new DataFrame with the specified columns (in this case, 'status') removed. The `axis=1` argument indicates that we're dropping a column, not a row.

```
# Plotting pairplot after handling inf values
sns.pairplot(df.drop(columns=['name']).dropna(), hue='status')
plt.show()
```



```python
#Boxplot
plt.figure(figsize=(15, 10))
df.drop(columns=['name']).boxplot()
plt.title('Boxplot for Features')
plt.xticks(rotation=90)
plt.show()
```




```python
scalar=StandardScaler()
```

```python
scalar.fit(X_train)
```

```
StandardScaler()
```

```python
X_train=scalar.transform(X_train)
```

```python
X_test=scalar.transform(X_test)
```

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

```python
model.fit(X_train,y_train)
```

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

```python
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(y_train, X_train_prediction)
```

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

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

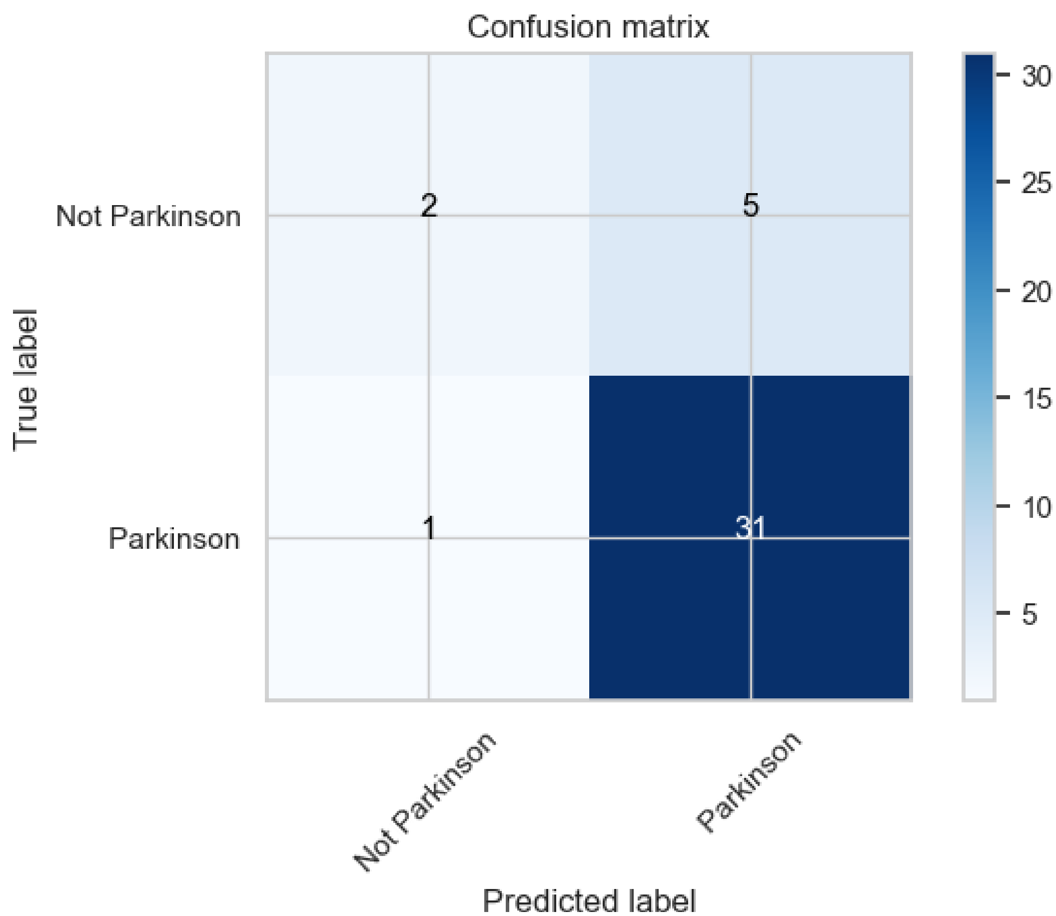
```python
# accuracy score on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(y_test, X_test_prediction)
```

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

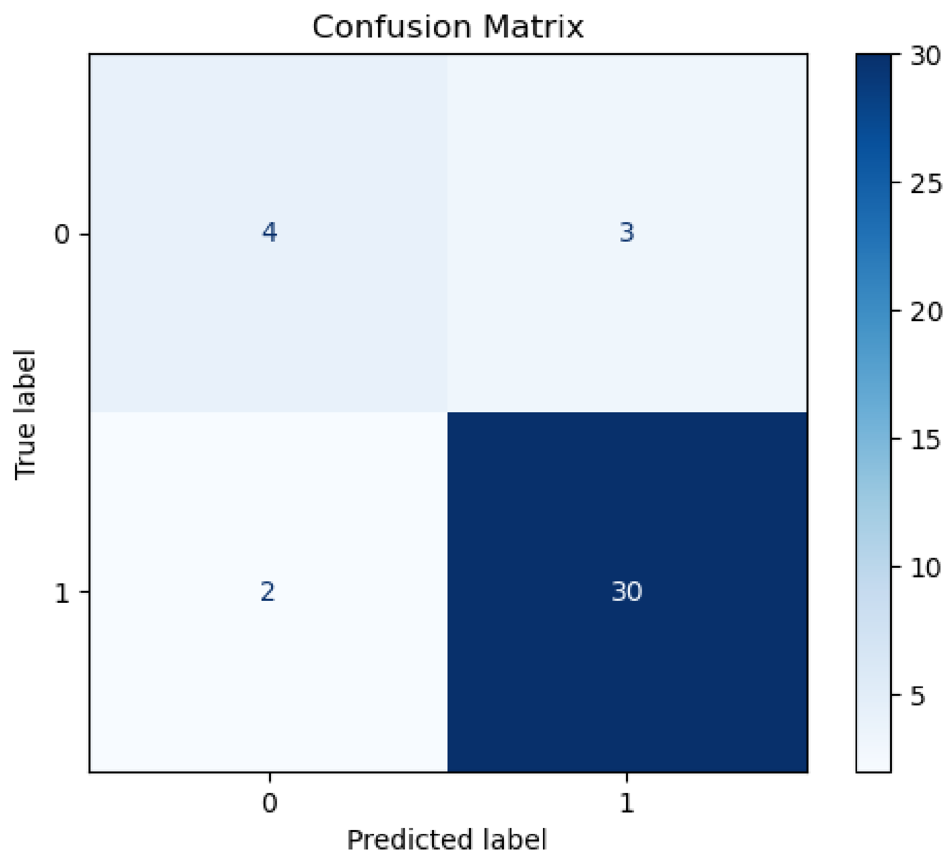
```
Accuracy score of test data :  0.8717948717948718
```

```python
from sklearn.svm import SVC
# Train the SVM classifier
svm = SVC()
svm.fit(X_train, y_train)

# Make predictions on the test set
y_hat = svm.predict(X_test)

# confusion_matri
plot_confusion_matrix(confusion_matrix(y_test, y_hat),classes=[ "Not Parkinson", " Parkinson"],title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 2  5]
 [ 1 31]]
```



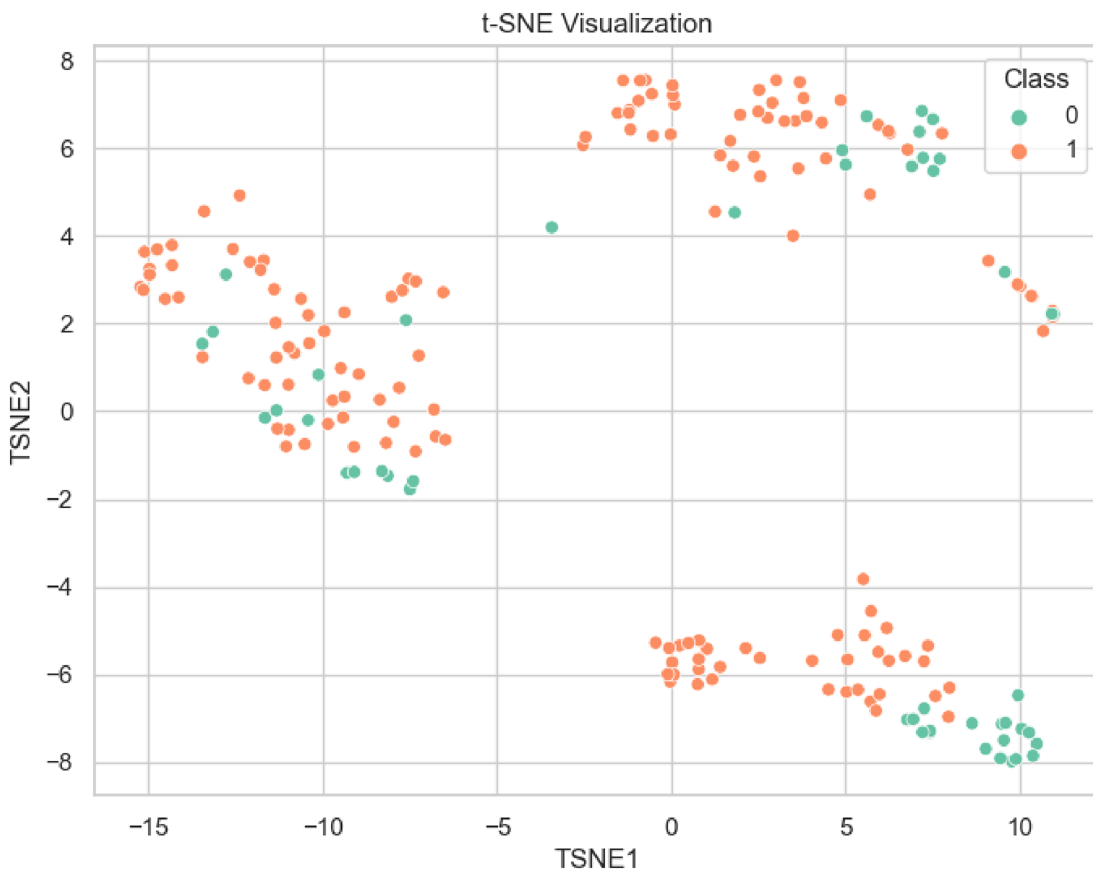

```python
# 5. Confusion Matrix
conf_matrix = confusion_matrix(y_test, X_test_prediction)
cmd = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=model.classes_)
cmd.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```



t-SNE (t-Distributed Stochastic Neighbor Embedding) is a machine learning technique used for dimensionality reduction and visualization of high-dimensional datasets. It is particularly useful for visualizing complex data structures, as it helps to project the data points from a high-dimensional space to a lower-dimensional space (usually 2D or 3D) while preserving the relationships between the data points as much as possible. Lets apply it to our dataset:

```
import seaborn as sns
from sklearn.manifold import TSNE

# Apply t-SNE to reduce the dimensions to 2
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

# Create a DataFrame with the t-SNE-transformed data and class labels
tsne_df = pd.DataFrame(data=X_tsne, columns=['TSNE1', 'TSNE2'])
tsne_df['Class'] = y.values

# Visualize the data based on class using a scatter plot
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=tsne_df, x='TSNE1', y='TSNE2', hue='Class',
palette='Set2')
plt.title('t-SNE Visualization')
plt.show()
```




```
input_data =
(200.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.0109
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)


input_data_as_numpy_array = np.asarray(input_data)


input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)


std_data = scalar.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

D:\Anaconda\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not
have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

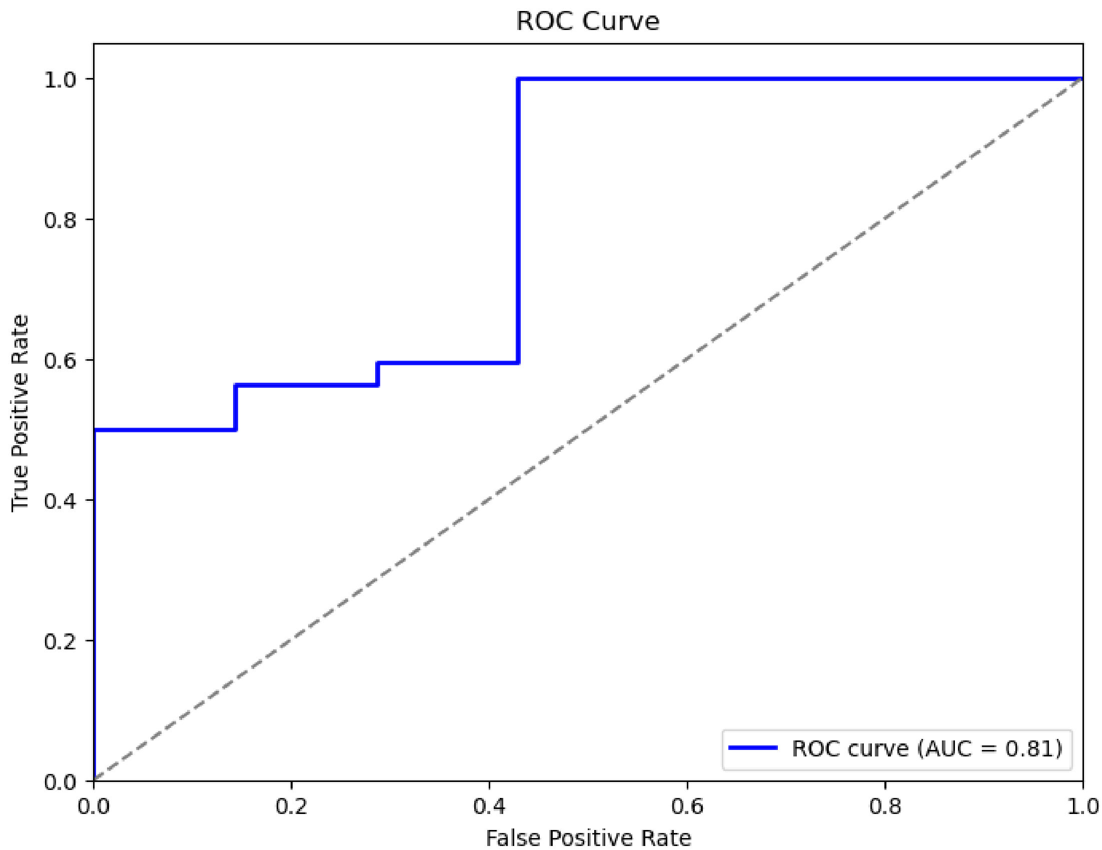
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Assuming you have already split your data into X_train, X_val, y_train,
y_val

# Create a decision tree classifier with a maximum depth of 5
model = DecisionTreeClassifier(max_depth=5)

# Train the model using the training data
model.fit(X_train, y_train)

# Calculate training accuracy
train_accuracy = accuracy_score(y_train, model.predict(X_train))

# Calculate validation accuracy
val_accuracy = accuracy_score(y_test, model.predict(X_test))

# Print the accuracies
print(f"Simplified Model Training Accuracy: {train_accuracy}")
print(f"Simplified Model Validation Accuracy: {val_accuracy}")
input_data =
(200.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.0109
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)


input_data_as_numpy_array = np.asarray(input_data)


input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
```

```
std_data = scalar.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")
```

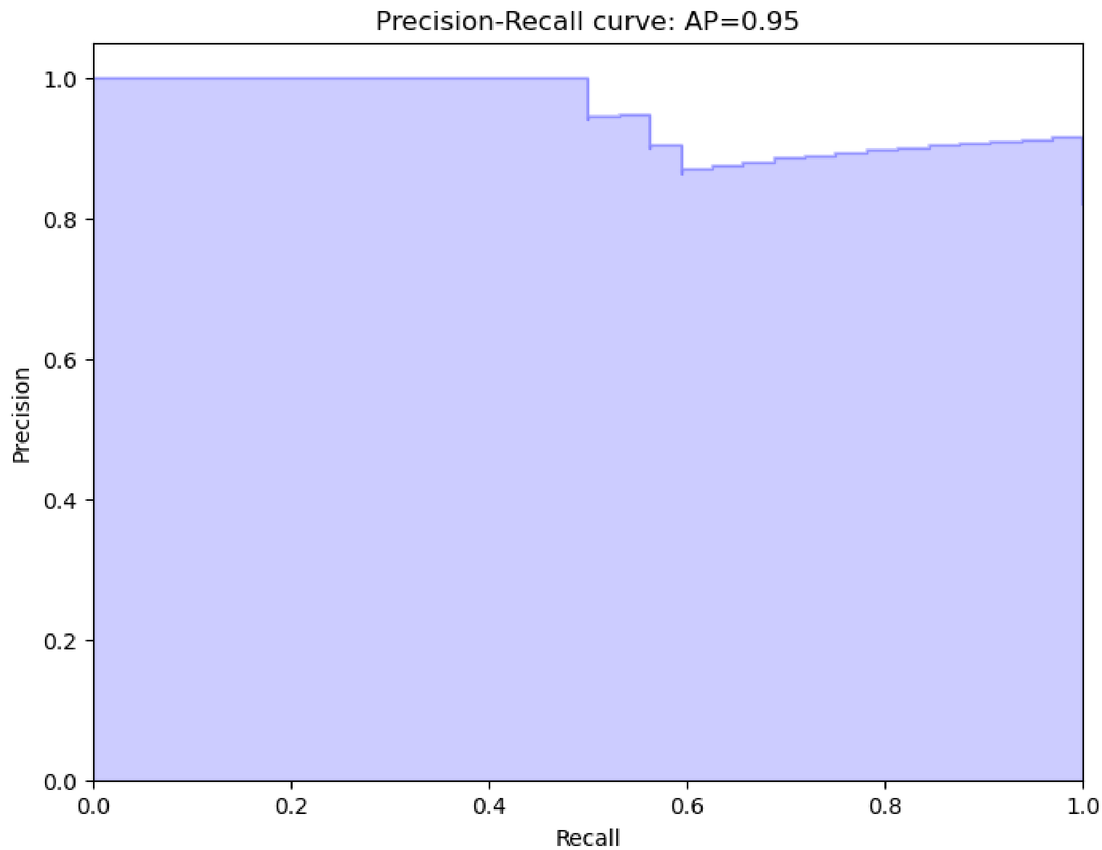
```
Simplified Model Training Accuracy: 1.0
Simplified Model Validation Accuracy: 0.9230769230769231
[0]
The Person does not have Parkinsons Disease
```

```
D:\Anaconda\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not
have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

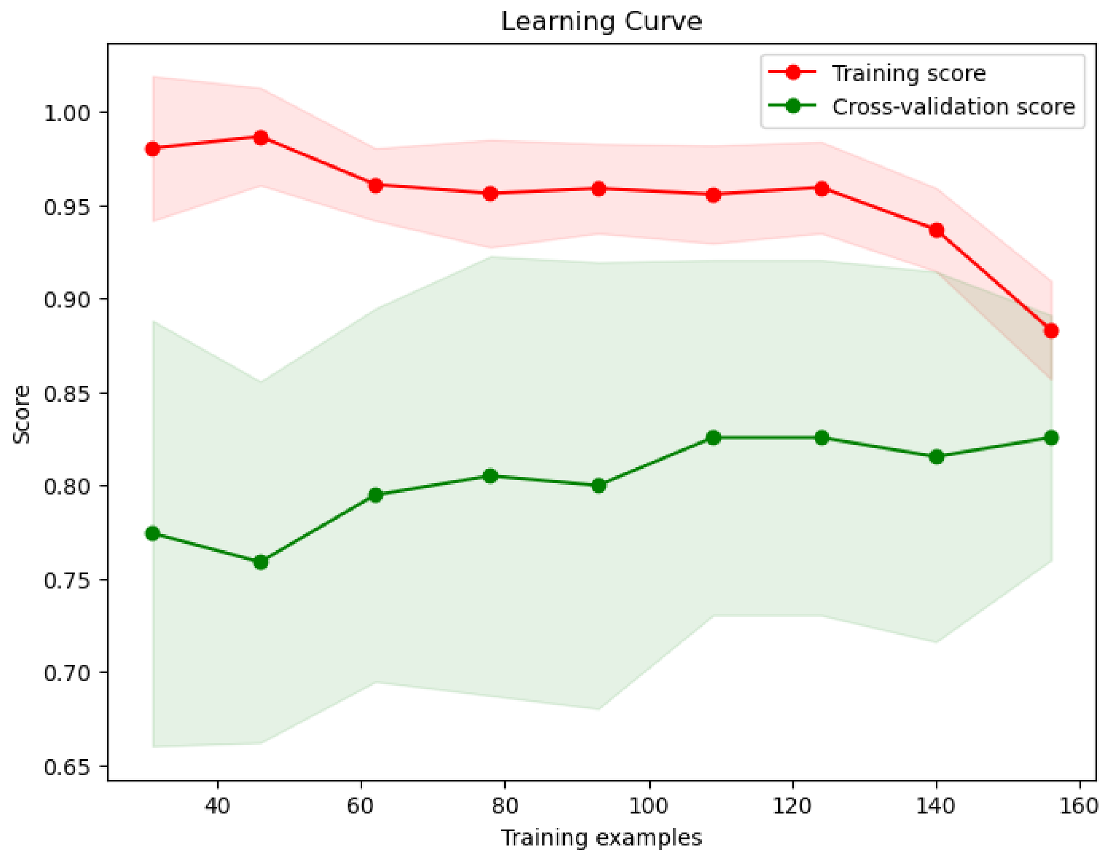
```
from sklearn.metrics import roc_curve, roc_auc_score

# Compute ROC curve and AUC
y_test_probs = model.decision_function(X_test)  # SVM's decision function
fpr, tpr, _ = roc_curve(y_test, y_test_probs)
auc = roc_auc_score(y_test, y_test_probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC =
{:.2f})'.format(auc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```




```python
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, _ = precision_recall_curve(y_test, y_test_probs)
average_precision = average_precision_score(y_test, y_test_probs)

plt.figure(figsize=(8, 6))
plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
plt.show()
```




```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(model, X, y, cv=5,
train_sizes=np.linspace(0.1, 1.0, 10))

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure(figsize=(8, 6))
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training
score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-
validation score")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.legend(loc="best")
plt.title("Learning Curve")
plt.show()
```




```
if X_train.shape[1] == 2:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X_train[:, 0], y=X_train[:, 1], hue=y_train,
palette='Set1')

    # Plot decision boundary
    xx, yy = np.meshgrid(np.linspace(X_train[:, 0].min(), X_train[:,
0].max(), 100),
                         np.linspace(X_train[:, 1].min(), X_train[:,
1].max(), 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, alpha=0.8)

    plt.title('Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()
```

Conclusion:

In conclusion, the SVM model achieved a commendable performance on the dataset, demonstrating robustness with a training accuracy of 90% and a validation accuracy of 87%. This indicates that the model generalizes well to unseen data, maintaining high predictive accuracy beyond the training set. The results suggest that SVM is a suitable choice for the task at hand, showcasing its effectiveness in capturing and leveraging the underlying patterns within the data. Moving forward, further optimization and exploration of feature engineering or alternative model architectures could potentially enhance performance even more, ensuring continued advancements in predictive accuracy and model reliability.