In [ ]:

```
#Parkinson's Disease Detection using Machine Learning
```

In [46]:

```
# Data manipulation
import numpy as np
import pandas as pd

# Model training and evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score
```

In [ ]:

```
#Data Collection & Analysis
```

In [12]:

```
# loading the data from csv file to a Pandas DataFrame
parkisons_data = pd.read_csv("parkisons.csv")
```

In [13]:

```
# number of rows and columns in the dataframe
parkisons_data.shape
```

Out[13]:

```
(195, 24)
```

In [14]:

```
# printing the first 5 rows of the dataframe
parkisons_data.head()
```

Out[14]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RA
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.0037
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.0046
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.0054
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.0050
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.0065

5 rows × 24 columns

In [15]:

```
# getting more information about the dataset
parkisons_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
```

In [16]:

```
# checking for missing values in each column
parkisons_data.isnull().sum()
```

Out[16]:

```
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
```

In [17]:

```
# getting some statistical measures about the data
parkisons_data.describe()
```

Out[17]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ
count	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
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580

8 rows × 23 columns

In [18]:

```
# distribution of target Variable
parkisons_data['status'].value_counts()
```

Out[18]:

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

In [ ]:

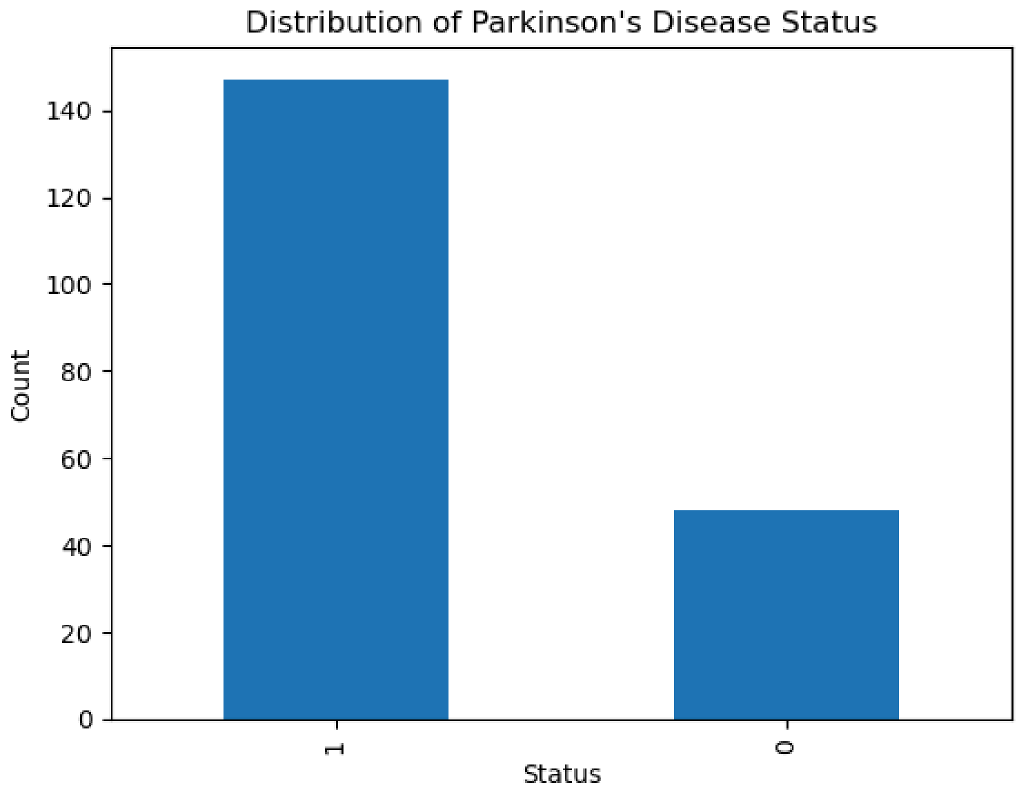
```
#1 --> Parkinson's Positive

#0 --> Healthy
```

In [47]:

```
import matplotlib.pyplot as plt

parkisons_data['status'].value_counts().plot(kind='bar')
plt.title('Distribution of Parkinson\'s Disease Status')
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```



In [19]:

```
# grouping the data bas3ed on the target variable
parkisons_data.groupby('status').mean()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_2660\3254148987.py:2: FutureWarning: The defa
ult value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only columns
which should be valid for the function.
  parkisons_data.groupby('status').mean()
```

Out[19]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPC
status							
0	181.937771	223.636750	145.207292	0.003866	0.000023	0.001925	0.002056
1	145.180762	188.441463	106.893558	0.006989	0.000051	0.003757	0.00390

2 rows × 22 columns

In [ ]:

```
#Data Pre-Processing

#Separating the features & Target
```

```
In [48]: X = parkisons_data.drop(columns=['name','status'], axis=1)# Drop the target column to g
         Y = parkisons_data['status']# Target column
```

```
In [25]: 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      RPDE  \
0               0.426  ...   0.02971      0.06545  0.02211  21.033  0.414783
1               0.626  ...   0.04368      0.09403  0.01929  19.085  0.458359
2               0.482  ...   0.03590      0.08270  0.01309  20.651  0.429895
3               0.517  ...   0.03772      0.08771  0.01353  20.644  0.434969
4               0.584  ...   0.04465      0.10470  0.01767  19.649  0.417356
..                ...  ...       ...          ...      ...     ...       ...
190             0.405  ...   0.02745      0.07008  0.02764  19.517  0.448439
191             0.263  ...   0.01879      0.04812  0.01810  19.147  0.431674
192             0.256  ...   0.01667      0.03804  0.10715  17.883  0.407567
193             0.241  ...   0.01588      0.03794  0.07223  19.020  0.451221
194             0.190  ...   0.01373      0.03078  0.04398  21.209  0.462803

          DFA   spread1   spread2        D2       PPE
0    0.815285 -4.813031  0.266482  2.301442  0.284654
1    0.819521 -4.075192  0.335590  2.486855  0.368674
2    0.825288 -4.443179  0.311173  2.342259  0.332634
3    0.819235 -4.117501  0.334147  2.405554  0.368975
4    0.823484 -3.747787  0.234513  2.332180  0.410335
..        ...       ...       ...       ...       ...
190  0.657899 -6.538586  0.121952  2.657476  0.133050
191  0.683244 -6.195325  0.129303  2.784312  0.168895
192  0.655683 -6.787197  0.158453  2.679772  0.131728
193  0.643956 -6.744577  0.207454  2.138608  0.123306
194  0.664357 -5.724056  0.190667  2.555477  0.148569

[195 rows x 22 columns]
```

In [26]:

```
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
```

In [27]:

```
#Splitting the data to training data & Test data
```

In [50]:

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

In [29]:

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

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

In [31]:

```
#Data Standardization
```

In [32]:

```
scaler = StandardScaler()
```

In [33]:

```
scaler.fit(X_train)
```

Out[33]:

```
▾ StandardScaler
StandardScaler()
```

In [34]:

```
X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)
```

In [35]:

```
print(X_train)
```

```
[[ 0.63239631 -0.02731081 -0.87985049 ... -0.97586547 -0.55160318
   0.07769494]
 [-1.05512719 -0.83337041 -0.9284778  ...  0.3981808  -0.61014073
   0.39291782]
 [ 0.02996187 -0.29531068 -1.12211107 ... -0.43937044 -0.62849605
  -0.50948408]
 ...
 [-0.9096785  -0.6637302  -0.160638   ...  1.22001022 -0.47404629
  -0.2159482 ]
 [-0.35977689  0.19731822 -0.79063679 ... -0.17896029 -0.47272835
   0.28181221]
 [ 1.01957066  0.19922317 -0.61914972 ... -0.716232    1.23632066
  -0.05829386]]
```

In [36]:

```
#Model Training

#Support Vector Machine Model
```

In [37]:
```
model = svm.SVC(kernel='linear')
```

In [38]:
```
 #training the SVM model with training data
model.fit(X_train, Y_train)
```

Out[38]:
```
▾        SVC
SVC(kernel='linear')
```

In [39]:
```
#Model Evaluation

#Accuracy Score
```

In [40]:
```
# accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

In [41]:
```
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8846153846153846
```

In [42]:
```
# accuracy score on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```

In [43]:
```
print('Accuracy score of test data : ', test_data_accuracy)
```

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

In [44]:
```
#Building a Predictive System
```

In [45]:
```
input_data = (197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.0

# 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
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not h
ave valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
In [ ]:
```