```
#importing the libraries
import pandas as pd
import numpy as np
import os, sys
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
import warnings
warnings.filterwarnings("ignore")

#Loading the data to pandas dataframe
parkinsons_data=pd.read_csv('/content/parkinsons.data')

parkinsons_data.head()
```

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

5 rows × 24 columns

```
#number of rows and columns
parkinsons_data.shape
```

```
(195, 24)
```

```
#Data Information
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
```

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

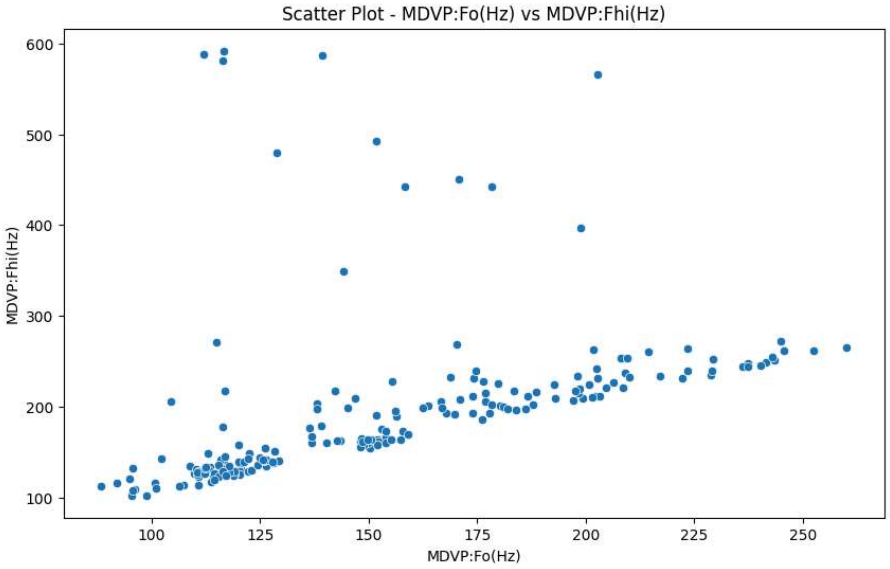
```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

```
#getting the statistical measures from the data
parkinsons_data.describe()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.00
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.00
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.00
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.00
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.00
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.00
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.00
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.02

8 rows × 23 columns

```
#plotting the features
plt.figure(figsize=(10, 6))
sns.scatterplot(x='MDVP:Fo(Hz)', y='MDVP:Fhi(Hz)', data=parkinsons_data)
plt.title('Scatter Plot - MDVP:Fo(Hz) vs MDVP:Fhi(Hz)')
plt.show()
```



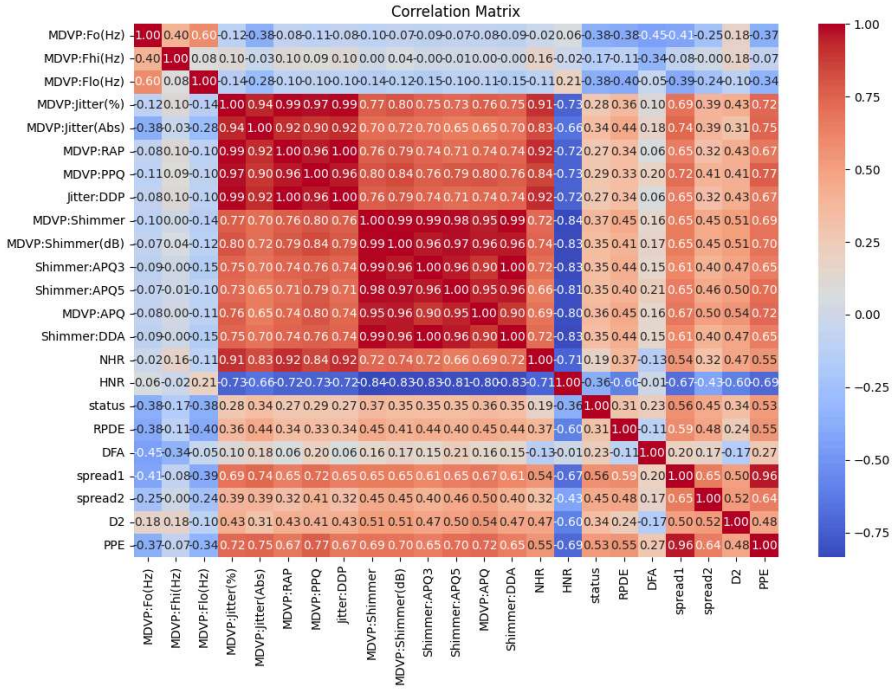

```
#Display histograms
parkinsons_data.hist(figsize=(15, 10))
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-096c0104e0be> in <cell line: 2>()
      1 #Display histograms
----> 2 parkinsons_data.hist(figsize=(15, 10))
      3 plt.tight_layout()
      4 plt.show()

NameError: name 'parkinsons_data' is not defined
```

Next steps: Explain error

```
#Creating a heatmap of the correlation matrix
correlation_matrix = parkinsons_data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

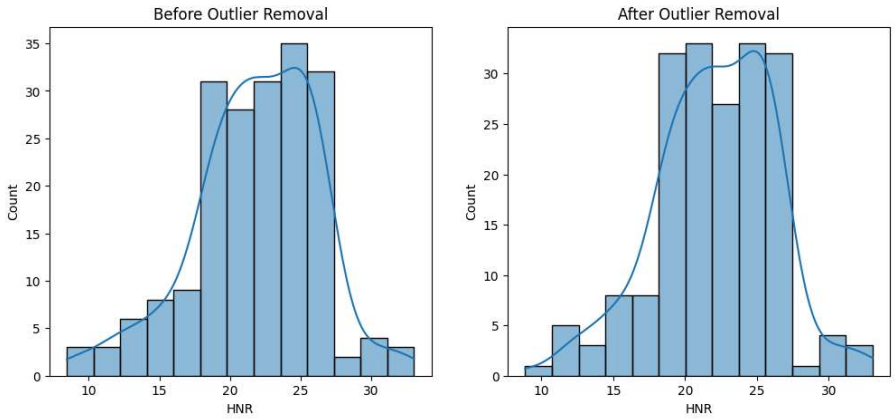


```
#outlier removal
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers_removed = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return outliers_removed

columns_to_check = ['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:APQ3', 'Shimmer:APQ5',
       'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
       'spread1', 'spread2', 'D2', 'PPE']

for col in columns_to_check:
    data_after = remove_outliers_iqr(parkinsons_data, col)


#before and after outlier removal
def plot_histograms(data_before, data_after, column):
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    sns.histplot(data_before[column], ax=axes[0], kde=True)
    axes[0].set_title('Before Outlier Removal')
    sns.histplot(data_after[column], ax=axes[1], kde=True)
    axes[1].set_title('After Outlier Removal')
    plt.show()
plot_histograms(parkinsons_data, data_after, "HNR")
```




```
#statistics before and after outlier removal
def compare_summary_statistics(data_before, data_after, column):
    summary_stats_before = data_before[column].describe()
    summary_stats_after = data_after[column].describe()
    print("Summary Statistics Before Outlier Removal:\n", summary_stats_before)
    print("\nSummary Statistics After Outlier Removal:\n", summary_stats_after)
compare_summary_statistics(parkinsons_data, data_after,"NHR")
```

```
Summary Statistics Before Outlier Removal:
 count    195.000000
mean       0.024847
std        0.040418
```

```
min        0.000650
25%        0.005925
50%        0.011660
75%        0.025640
max        0.314820
Name: NHR, dtype: float64

Summary Statistics After Outlier Removal:
 count    190.000000
mean       0.021587
std        0.032645
min        0.000650
25%        0.005870
50%        0.011420
75%        0.023320
max        0.314820
Name: NHR, dtype: float64
```

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

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

parkinsons_data['status']: This selects the column named 'status'

.value_counts(): This method counts the occurrences of each unique value in the selected column ('status')

```
parkinsons_data.groupby('status').mean(numeric_only=True)
```

status	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:
0	181.937771	223.636750	145.207292	0.003866	0.000023	0.001
1	145.180762	188.441463	106.893558	0.006989	0.000051	0.003

2 rows × 22 columns

parkinsons_data.groupby('status'): This groups the DataFrame parkinsons_data by the unique values in the 'status' column. This means that it separates the data into groups based on whether the individuals have Parkinson's disease or not.

.mean(numeric_only=True): This calculates the mean (average) of the numeric columns within each group. The numeric_only=True parameter ensures that only numeric columns are included in the calculation of the mean.

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

X = parkinsons_data.drop(columns=['name', 'status'], axis=1): This line creates a new DataFrame X by removing the columns named 'name' and 'status' from the original DataFrame parkinsons_data.(axis=1 indicates columns). The resulting DataFrame X will contain all the features except 'name' and 'status'.

Y = parkinsons_data['status']: This line creates a new Series Y by selecting only the 'status' column from the original DataFrame parkinsons_data. This column represents the target variable, indicating whether a person has Parkinson's disease (1) or not (0).

X

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAF
0	119.992	157.302	74.997	0.00784	0.00007	0.0037(
1	122.400	148.650	113.819	0.00968	0.00008	0.0046!
2	116.682	131.111	111.555	0.01050	0.00009	0.0054-
3	116.676	137.871	111.366	0.00997	0.00009	0.0050;
4	116.014	141.781	110.655	0.01284	0.00011	0.0065!
...	...	...	...	...	...	..
190	174.188	230.978	94.261	0.00459	0.00003	0.0026;
191	209.516	253.017	89.488	0.00564	0.00003	0.0033:
192	174.688	240.005	74.287	0.01360	0.00008	0.0062-
193	198.764	396.961	74.904	0.00740	0.00004	0.0037(
194	214.289	260.277	77.973	0.00567	0.00003	0.0029!

195 rows × 22 columns

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
```

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

X:independent(features) Y:dependent(target variable) test_size=20% testing 80% training

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

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

```
print(Y.shape,Y_train.shape,Y_test.shape)
```

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

```
scaler=StandardScaler()
```

to scale and standardize the features (independent variables)

Standardization: It standardizes the features by removing the mean and scaling to unit variance. This means it transforms the data such that it has a mean of 0 and a standard deviation of 1.

Scaling: It scales the features to have a mean of 0 and a standard deviation of 1. This ensures that all features have the same scale, which can be important for certain machine learning algorithms that are sensitive to the scale of the features (e.g., gradient descent-based algorithms like linear regression or neural networks).

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

```
▾ StandardScaler
StandardScaler()
```

it calculates the mean and standard deviation for each feature (column) in the training data X_train. These statistics are then stored within the

```
X_train=scaler.transform(X_train)

X_test=scaler.transform(X_test)
```

are used to transform (scale) the features (independent variables) in both the training set (X_train) and the testing set (X_test) using the scaling parameters (mean and standard deviation) computed earlier with scaler.fit(X_train).

```
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]]
```

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

model.fit(X_train,Y_train)
```

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

fit(): This is a method provided by scikit-learn's machine learning models that trains the model on the given training data.

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

to make predictions on the training data (X_train) using the trained model (model)

This line computes the accuracy of the predictions (X_train_prediction) compared to the actual labels (Y_train).

```
print("Accuracy of training data :",training_data_accuracy)
```

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

```
X_test_prediction=model.predict(X_test)
testing_data_accuracy=accuracy_score(Y_test,X_test_prediction)
```

they are used to make predictions on the testing data (X_test) and then compute the accuracy of the predictions compared to the actual labels (Y_test).

```
print("Accuracy of testing data :",testing_data_accuracy)
```

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

```
input_data=(119.99200,157.30200,74.99700,0.00784,0.00007,0.00370,0.00554,0.01109,0.04374,0.42600,0.0218

input_data_array=np.asarray(input_data)

input_data_reshaped=input_data_array.reshape(1,-1)

std_data=scaler.transform(input_data_reshaped)
```