Q2-parkinsons

April 17, 2021

0.0.1 PROBLEM STATEMENT for K-NN:

The given 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 Parkinson's Disease, according to "status" column which is set to 0 for healthy and 1 for PD. Dataset can be downloaded from below link.

https://archive.ics.uci.edu/ml/datasets/parkinsons

Create classification model using KNN. Identify the optimum no of neighbors and dimensions for your model.

Justify if KNN model should be considered or not for the problem statement.

[3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import random
import warnings
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import␣
↪train_test_split,GridSearchCV,cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,classification_report,roc_curve,auc
```

[4]:
```
ds=pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
↪parkinsons/parkinsons.data')
```

About Parkinson Disease:

- Parkinson's disease is a chronicle disorder of central nervous system which causes the death of the nervous cell in the brain.
- Parkinson's disease more often appeared after the age of 60.
- Parkinson's disease is progressive and the number of people suffering from the disease is expected to rise. The disease usually happens slowly and persists over a long period of time.

0.0.2 1) Handling of missing values, outliers, if any .

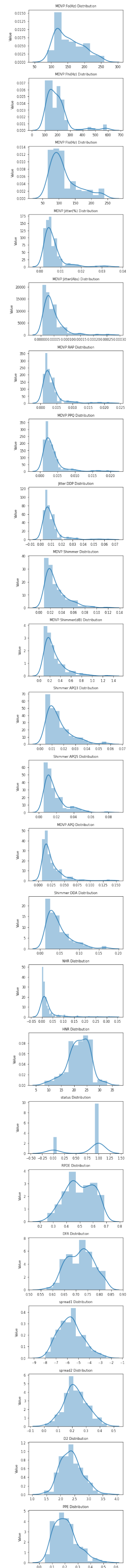
[5]:
```
print(ds[ds.duplicated()])
ds.info()
```

```
Empty DataFrame
Columns: [name, 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]
Index: []

[0 rows x 24 columns]
<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
```

From above analysis, Understood that is no null or duplicate values identified.

0.0.3 Outlier Treatment:

[6]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
fig, axes = plt.subplots(nrows=23, ncols=3, figsize=(15, 70))
for idx,cat_col in enumerate(ds.describe().columns):
    cl_idx=0
    sns.distplot(ds[cat_col],ax=axes[idx,cl_idx])
    plt.title(cat_col)
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=.
 ↪3, hspace=.2)
    axes[idx][cl_idx].set_title(cat_col+" Distribution",fontsize=8)
    axes[idx][cl_idx].xaxis.set_tick_params(labelsize=8,rotation=0)
    axes[idx][cl_idx].yaxis.set_tick_params(labelsize=8)
    axes[idx][cl_idx].set_ylabel('Value',fontsize=8)
    axes[idx][cl_idx].set(xlabel=None)
    df = pd.DataFrame(ds[cat_col], columns=[cat_col])
    des_lb=df.describe()
    axes[idx][cl_idx+1].axis('off')
    axes[idx][cl_idx+1].axis('tight')
    table = axes[idx][cl_idx+1].table(cellText=des_lb.values, colLabels=des_lb.
 ↪columns, rowLabels=des_lb.index, loc='center')
    table.set_fontsize(12)
    table.scale(1,1.5)
    sns.boxplot(y=ds[cat_col],ax=axes[idx,cl_idx+2])
    plt.title(cat_col)
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=.
 ↪3, hspace=.30)
    axes[idx][cl_idx+2].set_title(cat_col,fontsize=12)
    axes[idx][cl_idx+2].yaxis.set_tick_params(labelsize=10)
```



	MDVP:Fo(Hz)
count	195.0
mean	154.22864102564102
std	41.39006474907147
min	88.333
25%	117.572
50%	148.79
75%	182.769
max	260.105

	MDVP:Fhi(Hz)
count	195.0
mean	197.10491794871797
std	91.49154763503036
min	102.145
25%	134.8625
50%	175.829
75%	224.2055
max	592.03

	MDVP:Flo(Hz)
count	195.0
mean	116.32463076923077
std	43.52141318199365
min	65.476
25%	84.291
50%	104.315
75%	140.01850000000002
max	239.17

	MDVP:Jitter(%)
count	195.0
mean	0.006220461538461538
std	0.004848133692602560
min	0.00168
25%	0.00346
50%	0.00494
75%	0.007365
max	0.03316

	MDVP:Jitter(Abs)
count	195.0
mean	4.395897435897436e-05
std	3.482190859976326e-05
min	7e-06
25%	2e-05
50%	3e-05
75%	6e-05
max	0.00026

	MDVP:RAP
count	195.0
mean	0.003306410256410257
std	0.002967774416201684
min	0.00068
25%	0.00166
50%	0.0025
75%	0.003835
max	0.02144

	MDVP:PPQ
count	195.0
mean	0.003446358974358974
std	0.0027589766469679313
min	0.00092
25%	0.00186
50%	0.00269
75%	0.003955
max	0.01958

	Jitter:DDP
count	195.0
mean	0.009919948717948717
std	0.008903344355858987
min	0.00204
25%	0.004985
50%	0.00749
75%	0.011505000000000001
max	0.06433

	MDVP:Shimmer
count	195.0
mean	0.0297091282051282
std	0.01885693185894681
min	0.00954
25%	0.016505
50%	0.02297
75%	0.037885
max	0.11908

	MDVP:Shimmer(dB)
count	195.0
mean	0.2822512820512821
std	0.19487729006053414
min	0.085
25%	0.1485
50%	0.221
75%	0.35
max	1.302

	Shimmer:APQ3
count	195.0
mean	0.015664153846153845
std	0.010153161595709018
min	0.00455
25%	0.008245
50%	0.01279
75%	0.020265
max	0.05647

	Shimmer:APQ5
count	195.0
mean	0.017878256410256407
std	0.012023705538741727
min	0.0057
25%	0.00958
50%	0.01347
75%	0.02238
max	0.0794

	MDVP:APQ
count	195.0
mean	0.02408148717948718
std	0.016946736247029432
min	0.00719
25%	0.01308
50%	0.01826
75%	0.0294
max	0.13778

	Shimmer:DDA
count	195.0
mean	0.04699261538461539
std	0.030459119431240397
min	0.01364
25%	0.024735
50%	0.03836
75%	0.060795
max	0.16942

	NHR
count	195.0
mean	0.02484707692307692
std	0.04041844855606928
min	0.00065
25%	0.005925
50%	0.01166
75%	0.02564
max	0.31482

	HNR
count	195.0
mean	21.885974358974355
std	4.425764269063427
min	8.441
25%	19.198
50%	22.085
75%	25.075499999999998
max	33.047

	status
count	195.0
mean	0.7538461538461538
std	0.43187803371226474
min	0.0
25%	1.0
50%	1.0
75%	1.0
max	1.0

	RPDE
count	195.0
mean	0.4985355384615385
std	0.10394171413073468
min	0.25657
25%	0.421306
50%	0.495954
75%	0.5875625
max	0.685151

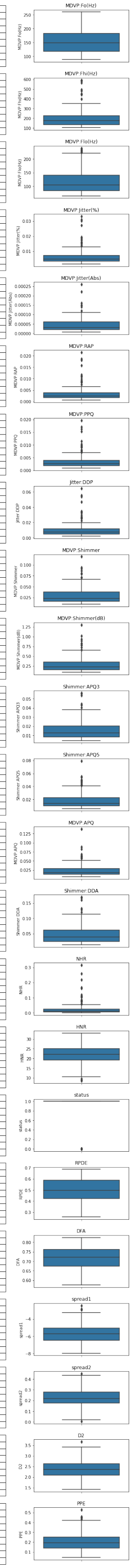
	DFA
count	195.0
mean	0.7180990461538461
std	0.0553358303465968
min	0.574282
25%	0.6747575
50%	0.722254
75%	0.7618815
max	0.825288

	spread1
count	195.0
mean	-5.684396743589745
std	1.090207763740309
min	-7.964984
25%	-6.450096
50%	-5.720868
75%	-5.046192
max	-2.434031

	spread2
count	195.0
mean	0.22651034871794873
std	0.08340576262039776
min	0.006274
25%	0.17435050000000002
50%	0.218885
75%	0.279234
max	0.450493

	D2
count	195.0
mean	2.3818260871794874
std	0.3827990465461168
min	1.423287
25%	2.0991255
50%	2.361532
75%	2.636456
max	3.671155

	PPE
count	195.0
mean	0.20655164102564103
std	0.09011932248227507
min	0.044539
25%	0.137451
50%	0.194052
75%	0.25298
max	0.527367



- Most of the attributes are either positively skewed or negatively skewed, In those places there are chances of few outliers. Data Understanding
- 23 Interval Attributes
- 1 Nominal Attribute

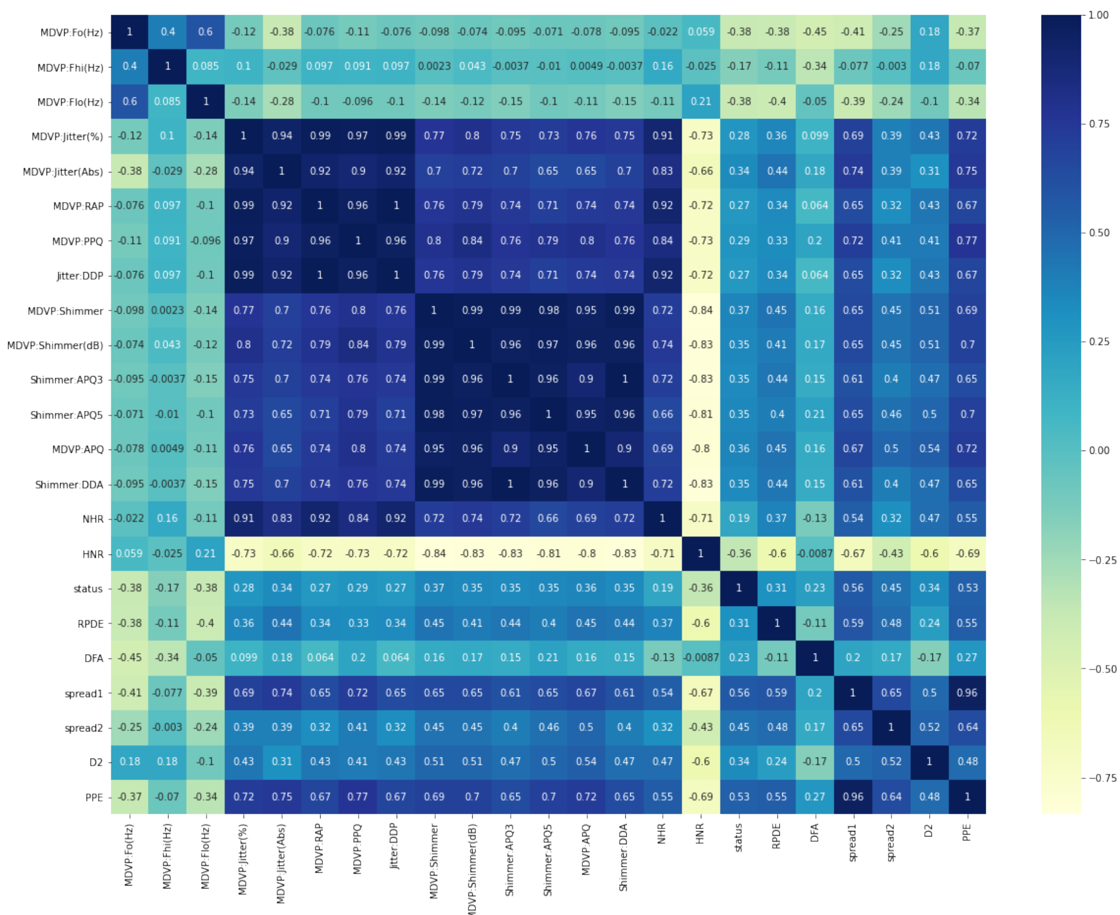
[7]:
```
ds.describe().transpose()
```

[7]:
```
                  count        mean        std         min         25%  \
MDVP:Fo(Hz)       195.0  154.228641  41.390065   88.333000  117.572000
MDVP:Fhi(Hz)      195.0  197.104918  91.491548  102.145000  134.862500
MDVP:Flo(Hz)      195.0  116.324631  43.521413   65.476000   84.291000
MDVP:Jitter(%)    195.0    0.006220   0.004848    0.001680    0.003460
MDVP:Jitter(Abs)  195.0    0.000044   0.000035    0.000007    0.000020
MDVP:RAP          195.0    0.003306   0.002968    0.000680    0.001660
MDVP:PPQ          195.0    0.003446   0.002759    0.000920    0.001860
Jitter:DDP        195.0    0.009920   0.008903    0.002040    0.004985
MDVP:Shimmer      195.0    0.029709   0.018857    0.009540    0.016505
MDVP:Shimmer(dB)  195.0    0.282251   0.194877    0.085000    0.148500
Shimmer:APQ3      195.0    0.015664   0.010153    0.004550    0.008245
Shimmer:APQ5      195.0    0.017878   0.012024    0.005700    0.009580
MDVP:APQ          195.0    0.024081   0.016947    0.007190    0.013080
Shimmer:DDA       195.0    0.046993   0.030459    0.013640    0.024735
NHR               195.0    0.024847   0.040418    0.000650    0.005925
HNR               195.0   21.885974   4.425764    8.441000   19.198000
status            195.0    0.753846   0.431878    0.000000    1.000000
RPDE              195.0    0.498536   0.103942    0.256570    0.421306
DFA               195.0    0.718099   0.055336    0.574282    0.674758
spread1           195.0   -5.684397   1.090208   -7.964984   -6.450096
spread2           195.0    0.226510   0.083406    0.006274    0.174351
D2                195.0    2.381826   0.382799    1.423287    2.099125
PPE               195.0    0.206552   0.090119    0.044539    0.137451

                         50%         75%         max
MDVP:Fo(Hz)       148.790000  182.769000  260.105000
MDVP:Fhi(Hz)      175.829000  224.205500  592.030000
MDVP:Flo(Hz)      104.315000  140.018500  239.170000
MDVP:Jitter(%)      0.004940    0.007365    0.033160
MDVP:Jitter(Abs)    0.000030    0.000060    0.000260
MDVP:RAP            0.002500    0.003835    0.021440
MDVP:PPQ            0.002690    0.003955    0.019580
Jitter:DDP          0.007490    0.011505    0.064330
MDVP:Shimmer        0.022970    0.037885    0.119080
MDVP:Shimmer(dB)    0.221000    0.350000    1.302000
Shimmer:APQ3        0.012790    0.020265    0.056470
Shimmer:APQ5        0.013470    0.022380    0.079400
```

```
MDVP:APQ           0.018260    0.029400    0.137780
Shimmer:DDA        0.038360    0.060795    0.169420
NHR                0.011660    0.025640    0.314820
HNR               22.085000   25.075500   33.047000
status             1.000000    1.000000    1.000000
RPDE               0.495954    0.587562    0.685151
DFA                0.722254    0.761881    0.825288
spread1           -5.720868   -5.046192   -2.434031
spread2            0.218885    0.279234    0.450493
D2                 2.361532    2.636456    3.671155
PPE                0.194052    0.252980    0.527367
```

[8]:
```python
a4_dims = (20, 15)
fig, ax = plt.subplots(figsize=a4_dims)
sns.heatmap(ds.corr(), annot=True, cmap="YlGnBu")
```

[8]: <AxesSubplot:>



Univariate Analysis

- In order to generalize the model well, it is crucial that the training data be an accurate representation of the population.
- In other words, each time a new sample is derived from the population, it is crucial that the sample must accurately paint a picture of the population.
- A training set of data must be representative of the cases you want to generalize to. To analyse this we need to see how the data is distributed and findout the if the data is positively skewed or negatively skewed.
- Analysis of this is important an the samples might tend to favor a select portion of the population, and thus might not accurately represent the true population. This is also popularly known as the presence of Skewness in data, and the data can be either right-skewed or left-skewed. Pearson's Kurtosis also defines how the data is distributed and where the data lies.

Kurtosis with positive values indicates that those attributes have more data points around the tail.

Skewness with positive values indicates data is skewed towards right.

Skewness with negative values indicates data is skewed towards left

[9]:
```
kur=ds.kurtosis(numeric_only  = True)
kur
```

[9]:
```
MDVP:Fo(Hz)         -0.627898
MDVP:Fhi(Hz)         7.627241
MDVP:Flo(Hz)         0.654615
MDVP:Jitter(%)      12.030939
MDVP:Jitter(Abs)    10.869043
MDVP:RAP            14.213798
MDVP:PPQ            11.963922
Jitter:DDP          14.224762
MDVP:Shimmer         3.238308
MDVP:Shimmer(dB)     5.128193
Shimmer:APQ3         2.720152
Shimmer:APQ5         3.874210
MDVP:APQ            11.163288
Shimmer:DDA          2.720661
NHR                 21.994974
HNR                  0.616036
status              -0.595518
RPDE                -0.921781
DFA                 -0.686152
spread1             -0.050199
spread2             -0.083023
D2                   0.220334
PPE                  0.528335
```

dtype: float64

[10]:
```
prc_row=[]
def fnd_skew_kurt(col_val):
    prc_row.append([col_val,ds[col_val].skew(),ds[col_val].kurtosis()])
for i in ds.describe().columns[:22]:
    fnd_skew_kurt(i)
before_out_treatment=pd.
↪DataFrame(prc_row,columns=['Feature_Before_Treatment','Skew_Before_Treatment','Kurtosis_Bef
```

0.0.4 Outlier Treatment

[11]:
```
ds_clone=ds.copy()
out_trt_row=[]
def outlier_treatment(col_val):
    q3 = ds_clone[col_val].quantile(0.75)
    q1 = ds_clone[col_val].quantile(0.25)
    t = q3-q1
    outliers_above = q3+t
    outliers_below = q1-t
    max_val = ds_clone[col_val].loc[ds_clone[col_val]<=outliers_above].mean()
    ds_clone[col_val] = ds_clone[col_val].
↪mask(ds_clone[col_val]>outliers_above,max_val)
    out_trt_row.append([col_val,ds_clone[col_val].skew(),ds_clone[col_val].
↪kurtosis()])

for i in ds.describe().columns[:22]:
    outlier_treatment(i)
after_out_treatment=pd.
↪DataFrame(out_trt_row,columns=['Feature_After_Treatment','Skew_After_Treatment','Kurtosis_A
sk_kur_comp=after_out_treatment.
↪merge(before_out_treatment,left_on='Feature_After_Treatment',right_on='Feature_Before_Treat
sk_kur_comp[['Feature_After_Treatment','Skew_Before_Treatment','Skew_After_Treatment','Kurtosi
```

[11]:
```
   Feature_After_Treatment  Skew_Before_Treatment  Skew_After_Treatment  \
0              MDVP:Fo(Hz)               0.591737              0.559907
1             MDVP:Fhi(Hz)               2.542146              0.298456
2             MDVP:Flo(Hz)               1.217350              1.060780
3           MDVP:Jitter(%)               3.084946              0.658673
4         MDVP:Jitter(Abs)               2.649071              0.730093
5                 MDVP:RAP               3.360708              0.676447
6                 MDVP:PPQ               3.073892              0.658232
7               Jitter:DDP               3.362058              0.675812
8             MDVP:Shimmer               1.666480              0.981248
9         MDVP:Shimmer(dB)               1.999389              0.913218
10            Shimmer:APQ3               1.580576              0.925069
```

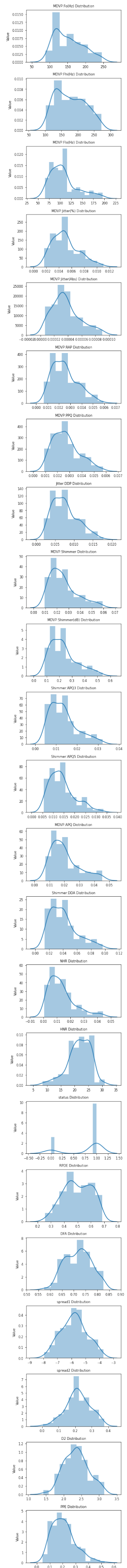
```
11            Shimmer:APQ5              1.798697            0.939986
12                MDVP:APQ              2.618047            1.003840
13             Shimmer:DDA              1.580618            0.925074
14                     NHR              4.220709            1.235100
15                     HNR             -0.514317           -0.723377
16                  status             -1.187727           -1.187727
17                    RPDE             -0.143402           -0.143402
18                     DFA             -0.033214           -0.033214
19                 spread1              0.432139            0.068635
20                 spread2              0.144430           -0.147932
21                      D2              0.430384            0.146502

    Kurtosis_Before_Treatment  Kurtosis_After_Treatment
0                   -0.627898                 -0.684961
1                    7.627241                 -1.046282
2                    0.654615                  0.546257
3                   12.030939                 -0.088996
4                   10.869043                  0.008361
5                   14.213798                 -0.195339
6                   11.963922                 -0.159571
7                   14.224762                 -0.196970
8                    3.238308                  0.302098
9                    5.128193                  0.181029
10                   2.720152                  0.253735
11                   3.874210                  0.445137
12                  11.163288                  0.250042
13                   2.720661                  0.253705
14                  21.994974                  1.190064
15                   0.616036                  0.609559
16                  -0.595518                 -0.595518
17                  -0.921781                 -0.921781
18                  -0.686152                 -0.686152
19                  -0.050199                 -0.522764
20                  -0.083023                 -0.113678
21                   0.220334                 -0.268670
```

[12]:

```
print('After outlier treatment')
warnings.simplefilter(action='ignore', category=FutureWarning)
fig, axes = plt.subplots(nrows=23, ncols=3, figsize=(15, 70))
for idx,cat_col in enumerate(ds_clone.describe().columns):
    cl_idx=0
    sns.distplot(ds_clone[cat_col],ax=axes[idx,cl_idx])
    plt.title(cat_col)
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=.
↪3, hspace=.2)
    axes[idx][cl_idx].set_title(cat_col+" Distribution",fontsize=8)
    axes[idx][cl_idx].xaxis.set_tick_params(labelsize=8,rotation=0)
```

```
    axes[idx][cl_idx].yaxis.set_tick_params(labelsize=8)
    axes[idx][cl_idx].set_ylabel('Value',fontsize=8)
    axes[idx][cl_idx].set(xlabel=None)
    df = pd.DataFrame(ds_clone[cat_col], columns=[cat_col])
    des_lb=df.describe()
    axes[idx][cl_idx+1].axis('off')
    axes[idx][cl_idx+1].axis('tight')
    table = axes[idx][cl_idx+1].table(cellText=des_lb.values, colLabels=des_lb.
↪columns, rowLabels=des_lb.index, loc='center')
    table.set_fontsize(12)
    table.scale(1,1.5)
    sns.boxplot(y=ds_clone[cat_col],ax=axes[idx,cl_idx+2])
    plt.title(cat_col)
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=.
↪3, hspace=.30)
    axes[idx][cl_idx+2].set_title(cat_col,fontsize=12)
    axes[idx][cl_idx+2].yaxis.set_tick_params(labelsize=10)
```

After outlier treatment



	MDVP:Fo(Hz)
count	195.0
mean	153.1711139896373
std	40.05631358767923
min	88.333
25%	117.572
50%	148.79
75%	180.58800000000002
max	245.51

	MDVP:Fhi(Hz)
count	195.0
mean	177.4146338797814
std	45.036006830644354
min	102.145
25%	134.8625
50%	175.829
75%	211.78250000000003
max	272.21

	MDVP:Flo(Hz)
count	195.0
mean	107.5852388888889
std	30.939352044802185
min	65.476
25%	84.291
50%	104.315
75%	115.29249999999999
max	195.708

	MDVP:Jitter(%)
count	195.0
mean	0.005044525139664804
std	0.002059661961350799
min	0.00168
25%	0.00346
50%	0.00494
75%	0.006200000000000001
max	0.01101

	MDVP:Jitter(Abs)
count	195.0
mean	3.7442622950819684e-05
std	2.03952121353593e-05
min	7e-06
25%	2e-05
50%	3e-05
75%	5e-05
max	9e-05

	MDVP:RAP
count	195.0
mean	0.0025485310734463273
std	0.0011238185504912838
min	0.00068
25%	0.00166
50%	0.0025
75%	0.00326
max	0.00593

	MDVP:PPQ
count	195.0
mean	0.0027136363636363634
std	0.0010872842189699036
min	0.00092
25%	0.00186
50%	0.00269
75%	0.00332
max	0.00576

	Jitter:DDP
count	195.0
mean	0.00764632768361582
std	0.0033701588516771557
min	0.00204
25%	0.004985
50%	0.00749
75%	0.00979
max	0.01778

	MDVP:Shimmer
count	195.0
mean	0.025372640449438204
std	0.01173753790964504
min	0.00954
25%	0.016505
50%	0.02297
75%	0.03177
max	0.05925

	MDVP:Shimmer(dB)
count	195.0
mean	0.23194886363636363
std	0.10553072889493624
min	0.085
25%	0.1485
50%	0.221
75%	0.2785
max	0.542

	Shimmer:APQ3
count	195.0
mean	0.013434245810055867
std	0.006511856978241614
min	0.00455
25%	0.008245
50%	0.01279
75%	0.01663
max	0.03223

	Shimmer:APQ5
count	195.0
mean	0.014171052631578945
std	0.005963060175127994
min	0.0057
25%	0.00958
50%	0.01347
75%	0.017235
max	0.03347

	MDVP:APQ
count	195.0
mean	0.020310111731843574
std	0.009455403570948845
min	0.00719
25%	0.01308
50%	0.01826
75%	0.02487
max	0.04525

	Shimmer:DDA
count	195.0
mean	0.040303128491620106
std	0.01953565276906898
min	0.01364
25%	0.024735
50%	0.03836
75%	0.049885
max	0.09669

	NHR
count	195.0
mean	0.013435930232558814
std	0.009717097470614179
min	0.00065
25%	0.005925
50%	0.01166
75%	0.01806
max	0.04441

	HNR
count	195.0
mean	21.720322916666667
std	4.22106579347945
min	8.441
25%	19.198
50%	21.875
75%	25.0215
max	30.94

	status
count	195.0
mean	0.7538461538461538
std	0.43187803371226474
min	0.0
25%	1.0
50%	1.0
75%	1.0
max	1.0

	RPDE
count	195.0
mean	0.4985355384615385
std	0.10394171413073468
min	0.25657
25%	0.421306
50%	0.495954
75%	0.5875625
max	0.685151

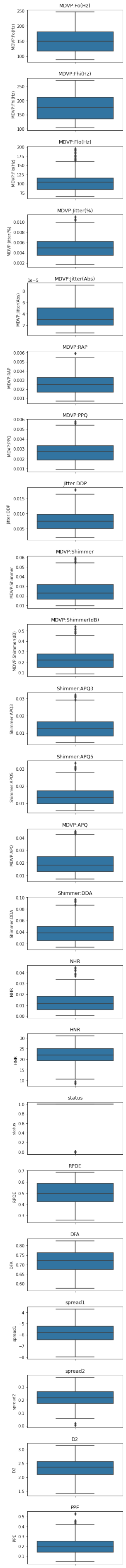
	DFA
count	195.0
mean	0.7180990461538461
std	0.0553358303465968
min	0.574282
25%	0.6747575
50%	0.722254
75%	0.7618815
max	0.825288

	spread1
count	195.0
mean	-5.808335747311828
std	0.9295089321409171
min	-7.964984
25%	-6.450096
50%	-5.808335747311828
75%	-5.2492730000000005
max	-3.700544

	spread2
count	195.0
mean	0.21781108064516128
std	0.07323952404873695
min	0.006274
25%	0.17435050000000002
50%	0.2178110806451613
75%	0.265141
max	0.375531

	D2
count	195.0
mean	2.3557621315789476
std	0.34622425618666414
min	1.423287
25%	2.0991255
50%	2.3557621315789476
75%	2.5684925
max	3.142364

	PPE
count	195.0
mean	0.20655164102564103
std	0.09011932248227507
min	0.044539
25%	0.137451
50%	0.194052
75%	0.25298
max	0.527367



Based on above visualiztion, its evident that We are able to treat outlier for the below features - MDVP:Fhi(Hz) - MDVP:Jitter(Abs) - spread1
- D2

There are few other features, That the outlier are still exist even after treatment. So we retained without outlier treatment. This can be handle with higher data distribution.

[13]:
```
ds.columns
```

[13]:
```
Index(['name', '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'],
      dtype='object')
```

0.0.5 2) Identifying data and model issues if any.

Based on above visualiztion, its evident that We are able to treat outlier for the below features

- MDVP:Fhi(Hz)
- MDVP:Jitter(Abs)
- spread1
- D2

There are few other features, That the outlier are still exist even after treatment. So we igonre without outlier treatment for those. This can be handle with higher data distribution.

[14]:
```
ds['MDVP:Fhi(Hz)']=ds_clone['MDVP:Fhi(Hz)']
ds['MDVP:Jitter(Abs)']=ds_clone['MDVP:Jitter(Abs)']
ds['spread1']=ds_clone['spread1']
ds['D2']=ds_clone['D2']
ds=ds.drop(['MDVP:Flo(Hz)', 'MDVP:Shimmer(dB)','NHR', 'PPE','spread2','HNR'],␣
 ↪axis=1)
print(ds.shape)
ds.head(5)
```

```
(195, 18)
```

[14]:
```
             name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Jitter(%)  \
0  phon_R01_S01_1      119.992       157.302         0.00784
1  phon_R01_S01_2      122.400       148.650         0.00968
2  phon_R01_S01_3      116.682       131.111         0.01050
3  phon_R01_S01_4      116.676       137.871         0.00997
4  phon_R01_S01_5      116.014       141.781         0.01284

   MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
```

```
0          0.000070   0.00370   0.00554     0.01109       0.04374
1          0.000080   0.00465   0.00696     0.01394       0.06134
2          0.000090   0.00544   0.00781     0.01633       0.05233
3          0.000090   0.00502   0.00698     0.01505       0.05492
4          0.000037   0.00655   0.00908     0.01966       0.06425

   Shimmer:APQ3  Shimmer:APQ5  MDVP:APQ  Shimmer:DDA  status      RPDE  \
0       0.02182       0.03130   0.02971      0.06545       1  0.414783
1       0.03134       0.04518   0.04368      0.09403       1  0.458359
2       0.02757       0.03858   0.03590      0.08270       1  0.429895
3       0.02924       0.04005   0.03772      0.08771       1  0.434969
4       0.03490       0.04825   0.04465      0.10470       1  0.417356

        DFA   spread1        D2
0  0.815285 -4.813031  2.301442
1  0.819521 -4.075192  2.486855
2  0.825288 -4.443179  2.342259
3  0.819235 -4.117501  2.405554
4  0.823484 -3.747787  2.332180
```

0.0.6 3) Choice of packages and distance measure used. justify your answer.

Package Choice: - sklearn for metrics preprocessing(LabelEncoder,StandardScaler),model_selection,train_test_sp import KNeighborsClassifier,metrics(accuracy_score,confusion_matrix,classification_report,roc_curve,auc) - numpy - pandas - seaborn - matplotlib

The following lists the string metric identifiers and the associated distance metric classes, Metrics intended for real-valued vector spaces:

identifier

class name

args

distance function

“euclidean”

EuclideanDistance

sqrt(sum((x - y)^2))

“manhattan”

ManhattanDistance

sum(|x - y|)

“chebyshev”

ChebyshevDistance

max(|x - y|)

“minkowski”

MinkowskiDistance

p

sum(|x - y|^p)^(1/p)

“wminkowski”

WMinkowskiDistance

p, w

sum(|w * (x - y)|^p)^(1/p)

“seuclidean”

SEuclideanDistance

V

sqrt(sum((x - y)^2 / V))

“mahalanobis”

MahalanobisDistance

V or VI

sqrt((x - y)’ V^-1 (x - y))

[15]:
```
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 18 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:Jitter(%)    195 non-null    float64
 4   MDVP:Jitter(Abs)  195 non-null    float64
 5   MDVP:RAP          195 non-null    float64
 6   MDVP:PPQ          195 non-null    float64
 7   Jitter:DDP        195 non-null    float64
 8   MDVP:Shimmer      195 non-null    float64
 9   Shimmer:APQ3      195 non-null    float64
 10  Shimmer:APQ5      195 non-null    float64
 11  MDVP:APQ          195 non-null    float64
 12  Shimmer:DDA       195 non-null    float64
 13  status            195 non-null    int64
 14  RPDE              195 non-null    float64
 15  DFA               195 non-null    float64
```

```
 16   spread1           195 non-null    float64
 17   D2                195 non-null    float64
dtypes: float64(16), int64(1), object(1)
memory usage: 27.5+ KB
```

[16]:
```
col = ds.pop("status")
ds.insert(len(ds.columns), col.name, col)
df=ds.drop(columns='name')
df.head(10)
```

[16]:
```
   MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Jitter(%)  MDVP:Jitter(Abs)  MDVP:RAP  \
0      119.992       157.302         0.00784          0.000070   0.00370
1      122.400       148.650         0.00968          0.000080   0.00465
2      116.682       131.111         0.01050          0.000090   0.00544
3      116.676       137.871         0.00997          0.000090   0.00502
4      116.014       141.781         0.01284          0.000037   0.00655
5      120.552       131.162         0.00968          0.000080   0.00463
6      120.267       137.244         0.00333          0.000030   0.00155
7      107.332       113.840         0.00290          0.000030   0.00144
8       95.730       132.068         0.00551          0.000060   0.00293
9       95.056       120.103         0.00532          0.000060   0.00268

   MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  Shimmer:APQ3  Shimmer:APQ5  MDVP:APQ  \
0   0.00554     0.01109       0.04374       0.02182       0.03130   0.02971
1   0.00696     0.01394       0.06134       0.03134       0.04518   0.04368
2   0.00781     0.01633       0.05233       0.02757       0.03858   0.03590
3   0.00698     0.01505       0.05492       0.02924       0.04005   0.03772
4   0.00908     0.01966       0.06425       0.03490       0.04825   0.04465
5   0.00750     0.01388       0.04701       0.02328       0.03526   0.03243
6   0.00202     0.00466       0.01608       0.00779       0.00937   0.01351
7   0.00182     0.00431       0.01567       0.00829       0.00946   0.01256
8   0.00332     0.00880       0.02093       0.01073       0.01277   0.01717
9   0.00332     0.00803       0.02838       0.01441       0.01725   0.02444

   Shimmer:DDA      RPDE       DFA   spread1        D2  status
0      0.06545  0.414783  0.815285 -4.813031  2.301442       1
1      0.09403  0.458359  0.819521 -4.075192  2.486855       1
2      0.08270  0.429895  0.825288 -4.443179  2.342259       1
3      0.08771  0.434969  0.819235 -4.117501  2.405554       1
4      0.10470  0.417356  0.823484 -3.747787  2.332180       1
5      0.06985  0.415564  0.825069 -4.242867  2.187560       1
6      0.02337  0.596040  0.764112 -5.634322  1.854785       1
7      0.02487  0.637420  0.763262 -6.167603  2.064693       1
8      0.03218  0.615551  0.773587 -5.498678  2.322511       1
9      0.04324  0.547037  0.798463 -5.011879  2.432792       1
```

[17]:
```
X = df.iloc[:,:(len(df.columns)-1)].values
y = df.iloc[:,-1].values
sc = StandardScaler()
X = sc.fit_transform(X)
sc = StandardScaler()
X = sc.fit_transform(X)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.
↪3,random_state=5)
```

[18]:
```
def dist_meas_on_classif(met):
    knn = KNeighborsClassifier(n_neighbors=6,metric=met)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print(met.upper()," Accuracy:",metrics.accuracy_score(y_test, y_pred))
    return knn.get_params()
```

[19]:
```
distance_measures=['euclidean','manhattan','chebyshev','hamming','minkowski']
for i in distance_measures:
    print("\n",dist_meas_on_classif(i),"\n")
```

```
EUCLIDEAN  Accuracy: 0.9491525423728814

 {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'euclidean', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 6, 'p': 2, 'weights': 'uniform'}

MANHATTAN  Accuracy: 0.9491525423728814

 {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'manhattan', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 6, 'p': 2, 'weights': 'uniform'}

CHEBYSHEV  Accuracy: 0.847457627118644

 {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'chebyshev', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 6, 'p': 2, 'weights': 'uniform'}

HAMMING  Accuracy: 0.7457627118644068

 {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'hamming', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 6, 'p': 2, 'weights': 'uniform'}

MINKOWSKI  Accuracy: 0.9491525423728814

 {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 6, 'p': 2, 'weights': 'uniform'}
```

from sklearn.model_selection import train_test_split X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3,random_state=5)

[20]:
```
def nn_with_diff_k(kval):
    knn = KNeighborsClassifier(n_neighbors=kval)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
```

0.0.7 4) Selection of train, test split.

- 70% of my total data to train my model and rest 30% to test it.
- Next, I train my model with different values of "K" and capture its accuracy on my test data.

[21]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.
 ↪3,random_state=5)
```

k-Fold Cross-Validation

- Cross-validation is when the dataset is randomly split up into 'k' groups.
- One of the groups is used as the test set and the rest are used as the training set.
- The model is trained on the training set and scored on the test set.
- Then the process is repeated until each unique group as been used as the test set.

[22]:
```
knn_cv = KNeighborsClassifier(n_neighbors=3)
cv_scores = cross_val_score(knn_cv, X_train, y_train, cv=5)
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
[0.85714286 0.96296296 0.88888889 0.77777778 0.96296296]
cv_scores mean:0.8899470899470898
```

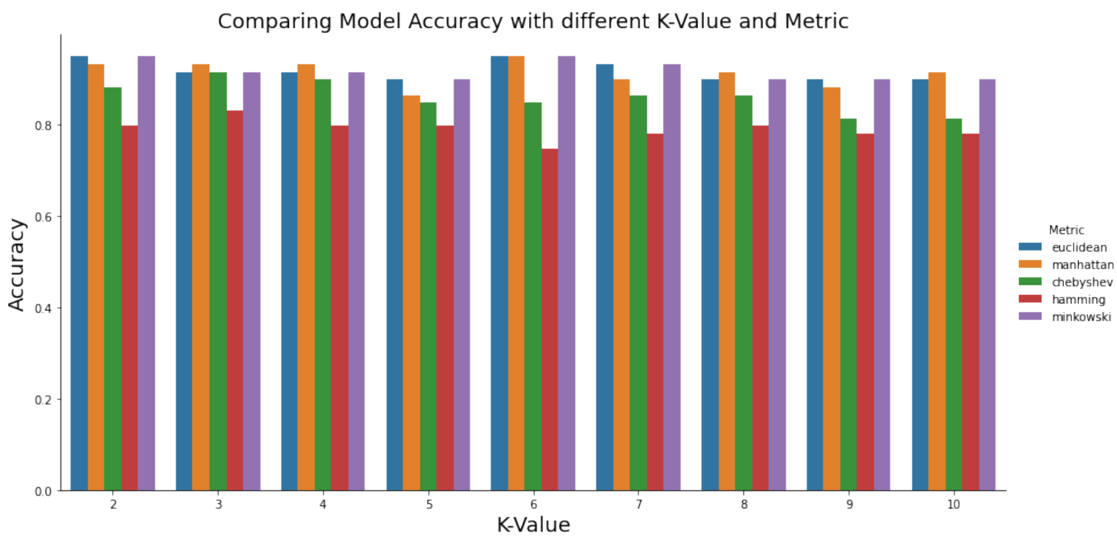
0.0.8 5) Final model creation and accuracy matrix selected for the model.

[23]:
```
def getKNNClassifierPerformance(k,dmeas):
    knn = KNeighborsClassifier(n_neighbors=k,metric=dmeas)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    model_accuracy=metrics.accuracy_score(y_test, y_pred)
    return model_accuracy
karray=[]
for i in range(1,10):
    karray.append(i+1)
karray
```

[23]:
```
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[24]:
```
perf_row=[]
for kValue in karray:
    for dis_meas in distance_measures:
        model_accuracy=getKNNClassifierPerformance(kValue,dis_meas)
```

```
        perf_row.append([kValue,dis_meas,dis_meas.upper()+'␣
↪KValue-'+str(kValue),model_accuracy])

model_performance_df = pd.DataFrame(perf_row, columns=["KValue",␣
↪"Metric","KMet","Accuracy"])
sns.catplot(data=model_performance_df, kind="bar", x="KValue", y="Accuracy",␣
↪hue="Metric", height=6, aspect=2,)
plt.xlabel('K-Value', fontsize=18)
plt.ylabel('Accuracy', fontsize=18)
plt.title("Comparing Model Accuracy with different K-Value and Metric",␣
↪fontsize=18)
plt.show()
```



0.0.9 Hypertuning model parameters using GridSearchCV

- GridSearchCV works by training our model multiple times on a range of parameters that we specify.
- That way, we can test our model with each parameter and figure out the optimal values to get the best accuracy results.

[25]:
```
knn_cls = KNeighborsClassifier()
param_grid = {'n_neighbors': np.arange(2, 25)}
knn_gscv_inst = GridSearchCV(knn_cls, param_grid, cv=9)
knn_gscv_inst.fit(X_train, y_train)
print(knn_gscv_inst.best_params_)
print(knn_gscv_inst.best_score_)
```

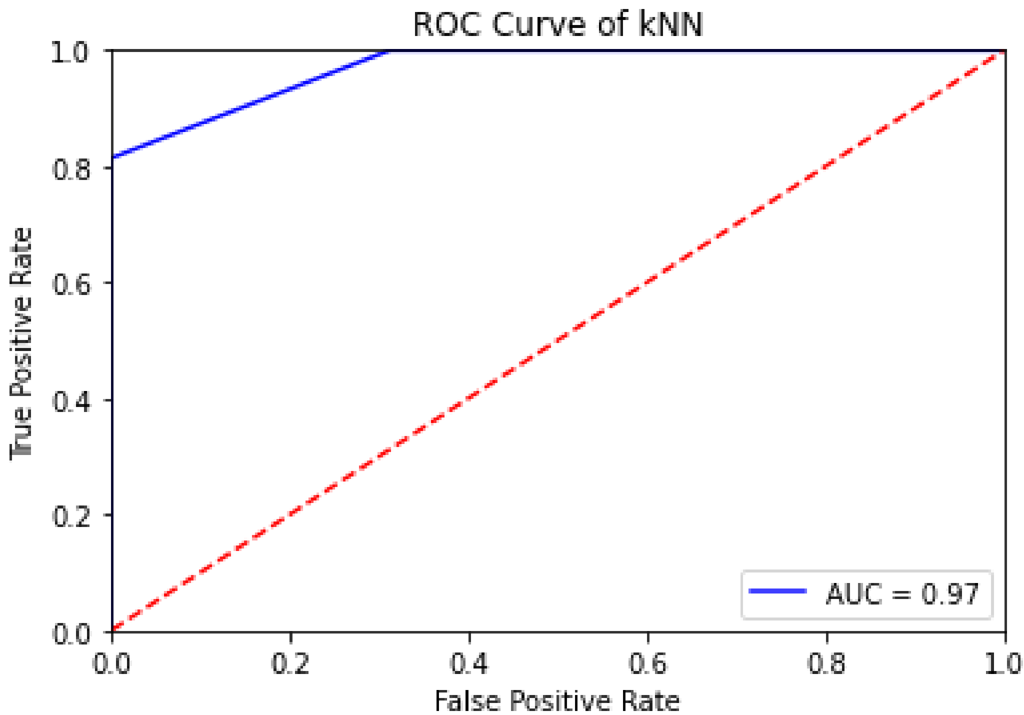
```
{'n_neighbors': 3}
0.9125000000000001
```

0.0.10 Based on k-Fold Cross-Validation & GridSearchCV, choosing K-3 & Distance EUCLIDEAN

[26]:
```
k_gscv=3
dist_gscv='euclidean'
model = KNeighborsClassifier(n_neighbors=k_gscv,metric=dist_gscv)
model.fit(X_train, y_train)
naive_pre= model.predict(X_test)
cnf_mat=confusion_matrix(y_test,naive_pre)
print(classification_report(y_test,naive_pre))
MNB = accuracy_score(y_test, naive_pre)
print("The Accuracy for KNN With K-Value:{0} with {1} Metric is {2}".
↪format(k_gscv,dist_gscv.upper(),MNB))
MultinomialNBScore = model.score(X_test,y_test)
```

```
              precision    recall  f1-score   support

           0       1.00      0.69      0.81        16
           1       0.90      1.00      0.95        43

    accuracy                           0.92        59
   macro avg       0.95      0.84      0.88        59
weighted avg       0.92      0.92      0.91        59

The Accuracy for KNN With K-Value:3 with EUCLIDEAN Metric is 0.9152542372881356
```

[27]:
```
knn = KNeighborsClassifier(n_neighbors = k_gscv,metric=dist_gscv)
knn.fit(X_train,y_train)
y_scores = knn.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of kNN')
plt.show()
```



6) Future scope of the work.

- The proposed method will be implemented in a distributed environment to further improve Parkinson's Disease diagnostic efficiency.
- Yet another future scope of work is Feature selection, Given dataset size is small. Feature section with high volume of data

0.0.11 Identify the optimum no of neighbors and dimensions for your model.

- Optimum k-Value is : 3
- Number of dimensions considered are 17.

0.0.12 Justify if KNN model should be considered or not for the problem statement.

- K Nearest Neighbors (KNN) algorithms can be considered for this kind of smaller dataset. When dataset size is huge then distance calculation will become more costly, which leads to performance impact.

- The purpose of this above implementation is to distinguishing between Parkinson's Diseased patient and healthy individual. Experimental results show that the KNN giving considerable accuracy.

- Where as there are other algorithms also can also be used for this kind of classification. For instance, ANN classifier will also gives higher average performance than the KNN classifier in term of accuracy.