SAMPLE OUTPUT

```
In [42]: import pandas as pd
         #List of useful imports that  I will use
         %matplotlib inline
         import os

         import matplotlib.pyplot as plt
         import pandas as pd
         import cv2
         import numpy as np
         from glob import glob
         import seaborn as sns
         import random
         import pickle

         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import roc_curve
```

```
In [43]: data = pd.read_csv(r'C:\Users\RAJA KANNAN\Music\PARKINSON\DATASET\parkinsons.csv')
```

```
In [44]: data
```

Out[44]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shi
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...	
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...	
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...	
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...	
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...	
...	...	...	...	...	...	...	...	...	...	...	...	
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.00003	0.00263	0.00259	0.00790	0.04087	...	
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.00994	0.02751	...	
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.00008	0.00624	0.00564	0.01873	0.02308	...	
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.00004	0.00370	0.00390	0.01109	0.02296	...	
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00003	0.00295	0.00317	0.00885	0.01884	...	

195 rows × 24 columns

```
In [45]: data.columns
```

```
Out[45]: 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')
```

In [46]:
```
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
```

In [47]:
```
data.describe()
```

Out[47]:

	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)	...
count	195.000000	195.000000	195.000000	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	0.009920	0.029709	0.282251	...
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	0.194877	...
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	0.085000	...
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	0.148500	...
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	0.221000	...
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	0.350000	...
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	1.302000	...

8 rows × 23 columns

In [48]:
```
data.isnull().sum()
```

Out[48]:

```
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
```

In [49]:

```
data.isnull().any()
```

Out[49]:

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

In [50]:

```
data.drop('name',axis=1,inplace=True)
```

In [51]:

```
data.corr()
```

Out[51]:

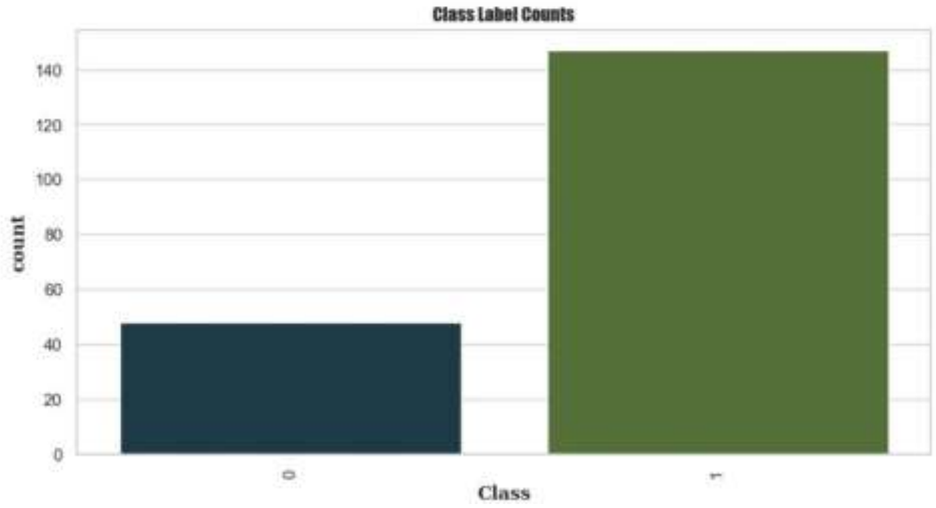
	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Sh
MDVP:Fo(Hz)	1.000000	0.400985	0.596546	-0.118003	-0.382027	-0.076194	-0.112165	-0.076213	-0.098374	
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.029198	0.097177	0.091126	0.097150	0.002281	
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.277815	-0.100519	-0.095828	-0.100488	-0.144543	
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.935714	0.990276	0.974256	0.990276	0.769063	
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.000000	0.922911	0.897778	0.922913	0.703322	
MDVP:RAP	-0.076194	0.097177	-0.100519	0.990276	0.922911	1.000000	0.957317	1.000000	0.759581	
MDVP:PPQ	-0.112165	0.091126	-0.095828	0.974256	0.897778	0.957317	1.000000	0.957319	0.797826	
Jitter:DDP	-0.076213	0.097150	-0.100488	0.990276	0.922913	1.000000	0.957319	1.000000	0.759555	
MDVP:Shimmer	-0.098374	0.002281	-0.144543	0.769063	0.703322	0.759581	0.797826	0.759555	1.000000	
MDVP:Shimmer(dB)	-0.073742	0.043465	-0.119089	0.804289	0.716601	0.790652	0.839239	0.790621	0.987258	
Shimmer:APQ3	-0.094717	-0.003743	-0.150747	0.746625	0.697153	0.744912	0.763580	0.744894	0.987625	

In [52]:

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

Out[52]:

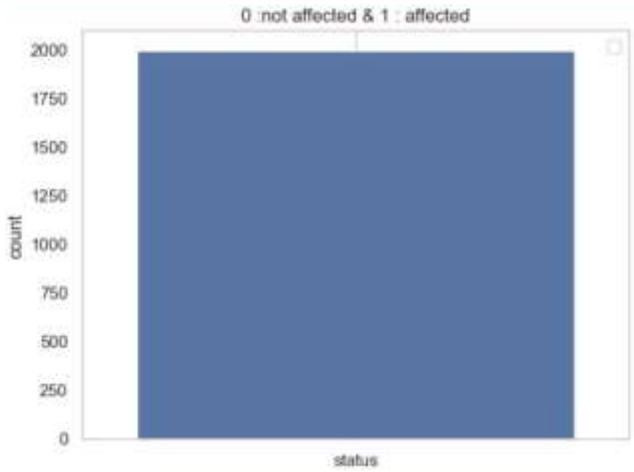
```
status
1    147
0     48
Name: count, dtype: int64
```

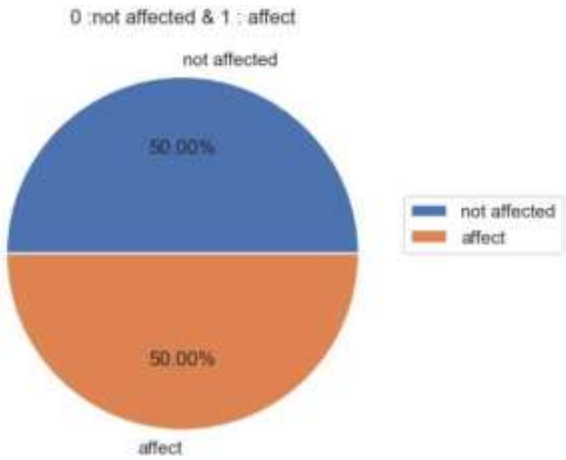
In [53]:
```
#counts of top 10 drugs
sns.set(style="whitegrid")
plt.figure(figsize=(10, 5))
ax = sns.countplot(x="status", data=data, palette=sns.color_palette("cubehelix", 4))
plt.xticks(rotation=90)
plt.title("Class Label Counts", {"fontname":"fantasy", "fontweight":"bold", "fontsize":"medium"})
plt.ylabel("count", {"fontname": "serif", "fontweight":"bold"})
plt.xlabel("Class", {"fontname": "serif", "fontweight":"bold"})
```
Out[53]:
```
Text(0.5, 0, 'Class')
```


In [54]:
```
from sklearn.utils import resample
# Separate majority and minority classes
df_majority = data[data['status']== 1]
df_minority = data[data['status']== 0]

# Downsample majority class and upsample the minority class
df_minority_upsampled = resample(df_minority, replace=True,n_samples=1000,random_state=100)
df_majority_downsampled = resample(df_majority, replace=True,n_samples=1000,random_state=100)

# Combine minority class with downsampled majority class
df_balanced = pd.concat([df_minority_upsampled, df_majority_downsampled])

# Display new class counts
df_balanced['status'].value_counts()
```
Out[54]:
```
status
0    1000
1    1000
Name: count, dtype: int64
```

In [55]:
```
sns.countplot(df_balanced[['status']])
plt.grid()
plt.legend()
plt.title(' 0 :not affected & 1 : affected ')
plt.show()
print(' ')
plt.pie([1000,1000],labels=['not affected','affect'],autopct='%.2f%%')
plt.legend(loc=(1,0.5))
plt.title(' 0 :not affected & 1 : affect ')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend
() is called with no arguement.
```





In [56]:
```
# shuffle the DataFrame rows
data= df_balanced.sample(frac = 1)
```

In [57]:
```
data
```

Out[57]:

	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)	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
172	110.739	113.597	100.139	0.00356	0.00003	0.00170	0.00200	0.00510	0.01484	0.133	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
30	197.076	206.896	192.055	0.00289	0.00001	0.00166	0.00168	0.00498	0.01098	0.097	...
...	...	...	...	...	...	...	...	...	...	...	...
174	117.004	144.466	99.923	0.00353	0.00003	0.00176	0.00218	0.00528	0.01657	0.145	...
106	155.078	163.736	144.148	0.00168	0.00001	0.00068	0.00092	0.00204	0.01064	0.097	...
28	155.358	227.383	80.055	0.00310	0.00002	0.00159	0.00176	0.00476	0.01718	0.161	...
51	126.344	134.231	112.773	0.00448	0.00004	0.00131	0.00169	0.00393	0.02033	0.185	...
86	180.978	200.125	155.495	0.00406	0.00002	0.00220	0.00244	0.00659	0.03852	0.331	...

2000 rows × 23 columns

In [58]:
```
data.isnull().sum()
```

Out[58]:
```
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 [59]:
```
data.dropna(inplace=True)
```

In [60]:
```
data
```

Out[60]:

	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)	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
172	110.739	113.597	100.139	0.00356	0.00003	0.00170	0.00200	0.00510	0.01484	0.133	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
30	197.076	206.896	192.055	0.00289	0.00001	0.00166	0.00168	0.00498	0.01098	0.097	...
...	...	...	...	...	...	...	...	...	...	...	...
174	117.004	144.466	99.923	0.00353	0.00003	0.00176	0.00218	0.00528	0.01657	0.145	...
106	155.078	163.736	144.148	0.00168	0.00001	0.00068	0.00092	0.00204	0.01064	0.097	...
28	155.358	227.383	80.055	0.00310	0.00002	0.00159	0.00176	0.00476	0.01718	0.161	...
51	126.344	134.231	112.773	0.00448	0.00004	0.00131	0.00169	0.00393	0.02033	0.185	...
86	180.978	200.125	155.495	0.00406	0.00002	0.00220	0.00244	0.00659	0.03852	0.331	...

2000 rows × 23 columns

In [61]:
```
#  get the all features except "status"
x = data.loc[:, data.columns != 'status']
```

In [62]:
```
x
```

Out[62]:

	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)	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
172	110.739	113.597	100.139	0.00356	0.00003	0.00170	0.00200	0.00510	0.01484	0.133	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
30	197.076	206.896	192.055	0.00289	0.00001	0.00166	0.00168	0.00498	0.01098	0.097	...
...	...	...	...	...	...	...	...	...	...	...	...
174	117.004	144.466	99.923	0.00353	0.00003	0.00176	0.00218	0.00528	0.01657	0.145	...
106	155.078	163.736	144.148	0.00168	0.00001	0.00068	0.00092	0.00204	0.01064	0.097	...
28	155.358	227.383	80.055	0.00310	0.00002	0.00159	0.00176	0.00476	0.01718	0.161	...
51	126.344	134.231	112.773	0.00448	0.00004	0.00131	0.00169	0.00393	0.02033	0.185	...
85	180.978	200.125	155.495	0.00406	0.00002	0.00220	0.00244	0.00659	0.03852	0.331	...

2000 rows × 22 columns

In [63]:
```
y = data.iloc[:,-7]
```

In [64]:
```
y
```

Out[64]:
```
126    1
172    0
165    0
165    0
30     0
      ..
174    0
106    1
28     1
51     0
85     1
Name: status, Length: 2000, dtype: int64
```

In [65]:
```
x.head()
```

Out[65]:

	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)	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
172	110.739	113.597	100.139	0.00356	0.00003	0.00170	0.00200	0.00510	0.01484	0.133	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
165	236.200	244.663	102.137	0.00277	0.00001	0.00154	0.00153	0.00462	0.02448	0.217	...
30	197.076	206.896	192.055	0.00289	0.00001	0.00166	0.00168	0.00498	0.01098	0.097	...

5 rows × 22 columns

In [66]:
```
y.tail()
```

Out[66]:
```
174    0
106    1
28     1
51     0
85     1
Name: status, dtype: int64
```

In [67]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,stratify=y ,random_state=40)
```

In [68]:
```
x_train
```

Out[68]:

	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)	...
77	110.568	125.394	106.821	0.00462	0.00004	0.00226	0.00280	0.00677	0.02199	0.197	...
42	237.226	247.326	225.227	0.00298	0.00001	0.00169	0.00182	0.00507	0.01752	0.164	...
171	112.547	133.374	105.715	0.00355	0.00003	0.00166	0.00190	0.00499	0.01358	0.129	...
188	114.563	119.167	86.647	0.00327	0.00003	0.00146	0.00184	0.00439	0.01185	0.106	...
43	241.404	248.834	232.483	0.00281	0.00001	0.00157	0.00173	0.00470	0.01760	0.154	...
...	...	...	...	...	...	...	...	...	...	...	...
172	110.739	113.597	100.139	0.00356	0.00003	0.00170	0.00200	0.00510	0.01484	0.133	...
193	198.764	396.961	74.904	0.00740	0.00004	0.00370	0.00390	0.01109	0.02296	0.241	...
46	245.510	262.090	231.848	0.00235	0.00001	0.00127	0.00148	0.00380	0.01608	0.141	...
171	112.547	133.374	105.715	0.00355	0.00003	0.00166	0.00190	0.00499	0.01358	0.129	...
15	142.167	217.455	83.159	0.00369	0.00003	0.00157	0.00203	0.00471	0.01503	0.126	...

1400 rows × 22 columns

In [69]:
```
y_test
```

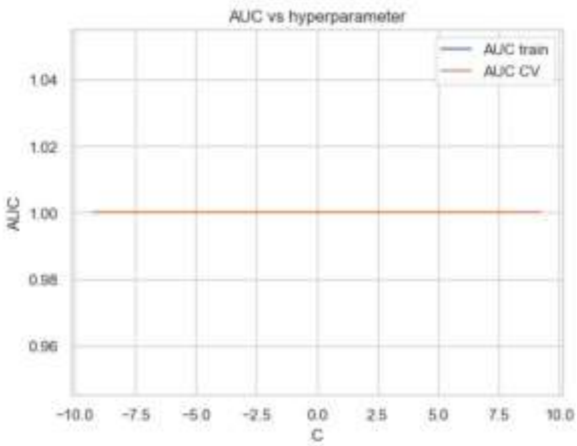
Out[69]:
```
46     0
152    1
133    1
155    1
85     1
      ..
35     0
186    0
34     0
32     0
190    0
Name: status, Length: 600, dtype: int64
```

In [70]:
```
x_test.to_csv('Parkinsons_test.csv',index = False)
```

SUPPORT VECTOR MACHINE

In [31]:

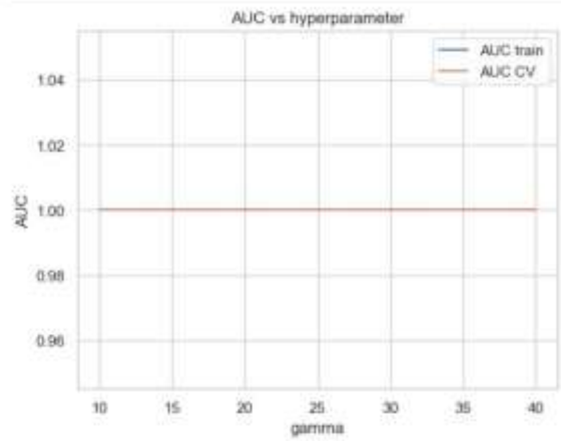
```
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
import math
from sklearn.metrics import accuracy_score

C = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

train_auc = []
cv_auc = []

for i in C:
    model = SVC(C=i,gamma=50)
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(x_train,y_train)
    prob_cv = clf.predict(x_test)
    cv_auc.append(accuracy_score(y_test,prob_cv))
    prob_train = clf.predict(x_train)
    train_auc.append(accuracy_score(y_train,prob_train))
optimal_C= C[cv_auc.index(max(cv_auc))]
C=[math.log(x) for x in C]

#plot auc vs alpha
x = plt.subplot( )
x.plot(C, train_auc, label='AUC train')
x.plot(C, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('C')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal C for which auc is maximum : ',optimal_C)
```




```
optimal C for which auc is maximum :  10000
```

```
In [33]: #Testing AUC on Test data
svc = SVC(C=optimal_C,gamma=optimal_gamma)

svc.fit(x_train,y_train)
filename = r'C:\Users\RAJA KANNAN\Music\PARKINSON\CODING\frontend\svc_park.pkl'
pickle.dump(svc, open(filename, 'wb'))
#predict on test data and train data

y_predtests = svc.predict(x_test)
y_predtrains = svc.predict(x_train)

print('*'*35)

#accuracy on training and testing data

print('the accuracy on testing data',accuracy_score(y_test,y_predtests))
print('the accuracy on training data',accuracy_score(y_train,y_predtrains))
train = accuracy_score(y_train,y_predtrains)
test = accuracy_score(y_test,y_predtests)

print('*'*35)


# Code for drawing seaborn heatmaps
class_names = ['not affcted','affect']
cm = pd.DataFrame(confusion_matrix(y_test, y_predtests.round()), index=class_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(cm, annot=True, fmt="d")
```

```
***********************************
the accuracy on testing data 1.0
the accuracy on training data 1.0
***********************************
```




```
optimal gamma for which auc is maximum :  10
```

In [34]:

```
original =  ['affected' if x==1 else 'not affected' for x in y_test[:20]]
predicted = svc.predict(x_test[:20])
pred = []

for i in predicted:
  if i == 1:
    k = 'affected'
    pred.append(k)
  else:
    k = 'not affected'
    pred.append(k)
# Creating a data frame
dfr = pd.DataFrame(list(zip(original, pred,)),
               columns =['original_Classlabel', 'predicted_classlebel'])
dfr
```

Out[34]:

	original_Classlabel	predicted_classlebel
0	not affected	not affected
1	affected	affected
2	affected	affected
3	affected	affected
4	affected	affected
5	affected	affected
6	not affected	not affected
7	not affected	not affected
8	not affected	not affected
9	affected	affected
10	affected	affected
11	not affected	not affected
12	affected	affected
13	not affected	not affected
14	not affected	not affected
15	not affected	not affected
16	not affected	not affected
17	affected	affected
18	affected	affected
19	not affected	not affected

```
optimal n_estimators 20
optimal max_depth 5
```

```
In [39]: #testing AUC on test data


xgb.fit(x_train,y_train)
filename = r'C:\Users\RAJA KANNAN\Music\PARKINSON\CODING\frontend\xgb_park.pkl'
pickle.dump(xgb, open(filename, 'wb'))
#predict on test data and train data

y_predtest = xgb.predict(x_test)
y_predtrain = xgb.predict(x_train)

print('*'*35)

#accuracy on training and testing data

print('the accuracy on testing data',accuracy_score(y_test,y_predtest))
print('the accuracy on training data',accuracy_score(y_train,y_predtrain))
train2 = accuracy_score(y_train,y_predtrain)
test2 = accuracy_score(y_test,y_predtest)

print('*'*35)


# Code for drawing seaborn heatmaps
class_names = ['not affected','affected']
cm = pd.DataFrame(confusion_matrix(y_test, y_predtest.round()), index=class_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(cm, annot=True, fmt="d")
```

```
***********************************
the accuracy on testing data 1.0
the accuracy on training data 1.0
***********************************
```

In [40]:
```python
original =  ['affected' if x==1 else 'not affected' for x in y_test[:20]]
predicted = xgb.predict(x_test[:20])
pred = []

for i in predicted:
  if i == 1:
    k = 'affected'
    pred.append(k)
  else:
    k = 'not affected'
    pred.append(k)
# Creating a data frame
dfr = pd.DataFrame(list(zip(original, pred,)),
               columns =['original_Classlabel', 'predicted_classlebel'])
dfr
```

7	not affected	not affected
8	not affected	not affected
9	affected	affected
10	affected	affected
11	not affected	not affected
12	affected	affected
13	not affected	not affected
14	not affected	not affected
15	not affected	not affected
16	not affected	not affected
17	affected	affected
18	affected	affected
19	not affected	not affected

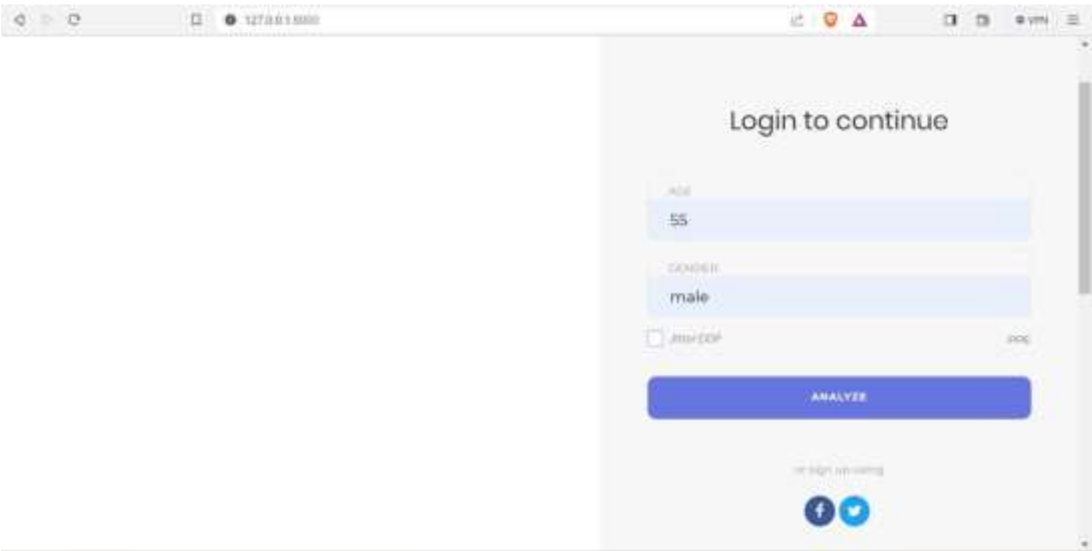
In [41]:
```python
new = ['XGB-Classifier',train2, test2]
all_model_result.loc[1] = new
```

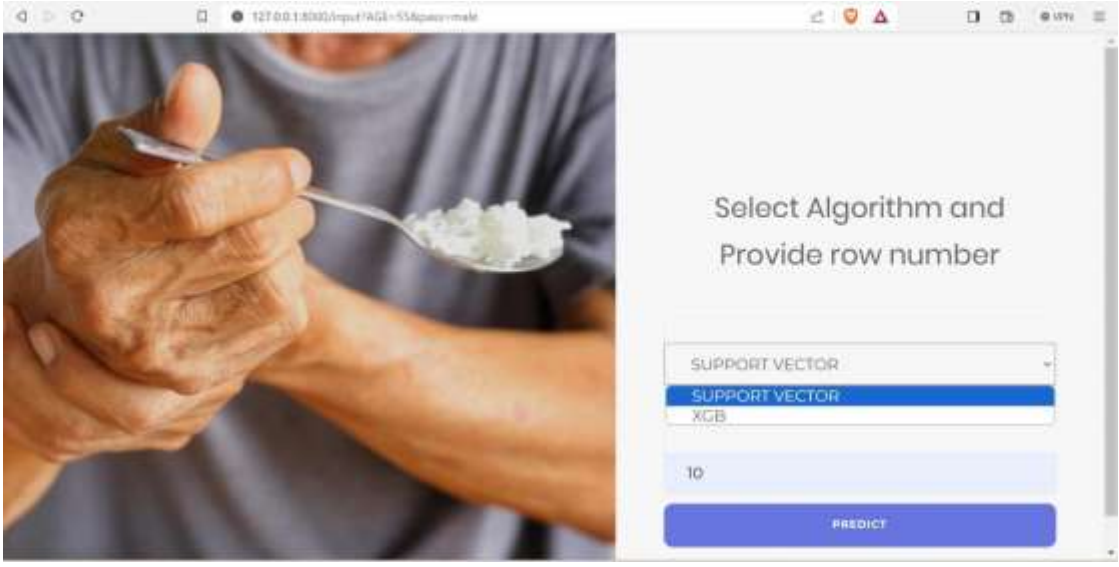
In [42]:
```python
all_model_result
```

Out[42]:

	Classifier	Train-Accuracy	Test-Accuracy
0	SUPPORT VECTOR-Classifier	1.0	1.0
1	XGB-Classifier	1.0	1.0

IMPLEMENTATION




Select Algorithm and
Provide row number
SUPPORT VECTOR
SUPPORT VECTOR
XGB
10
PREDICT


Stages of Parkinson's Disease
Stage 2: Symptoms such as tremors and stiffness begin to worsen, may develop poor posture or have trouble walking
PARKINSONS
DISEASE detection
in process...
affected