Importing important libraries

In [13]:

```
#working with data
import pandas as pd
import numpy as np
#visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

## Scikit-learn features various classification, regression and clustering algorithms
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import (average_precision_score, confusion_matrix, accuracy_score,
classification_report,f1_score)

## Scaling
from sklearn.preprocessing import StandardScaler

## Algorithm
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
import warnings
warnings.filterwarnings('ignore')
```

Reading the text file

In [14]:

```
fileObject = open(r"C:\Users\Anas Khanooni\Documents\ANAS KHANOONI\ANAS POST-GRD IN AI\Assi
data = fileObject.read()
print(data)
```

```
Title: Parkinsons Disease Data Set

Abstract: Oxford Parkinson's Disease Detection Dataset

-----------------------------------------------------

Data Set Characteristics: Multivariate
Number of Instances: 197
Area: Life
Attribute Characteristics: Real
Number of Attributes: 23
Date Donated: 2008-06-26
Associated Tasks: Classification
Missing Values? N/A

-----------------------------------------------------

Source:

The dataset was created by Max Little of the University of Oxford, in
collaboration with the National Centre for Voice and Speech, Denver,
Colorado, who recorded the speech signals. The original study published the
feature extraction methods for general voice disorders.

-----------------------------------------------------

Data Set Information:

This 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 PD, according to "status"
column which is set to 0 for healthy and 1 for PD.

The data is in ASCII CSV format. The rows of the CSV file contain an
instance corresponding to one voice recording. There are around six
recordings per patient, the name of the patient is identified in the first
column.For further information or to pass on comments, please contact Max
Little (littlem '@' robots.ox.ac.uk).

Further details are contained in the following reference -- if you use this
dataset, please cite:
Max A. Little, Patrick E. McSharry, Eric J. Hunter, Lorraine O. Ramig (200
8),
'Suitability of dysphonia measurements for telemonitoring of Parkinson's dis
ease',
IEEE Transactions on Biomedical Engineering (to appear).

-----------------------------------------------------

Attribute Information:

Matrix column entries (attributes):
```

```
name - ASCII subject name and recording number
MDVP:Fo(Hz) - Average vocal fundamental frequency
MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
MDVP:Flo(Hz) - Minimum vocal fundamental frequency
MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several
measures of variation in fundamental frequency
MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA
- Several measures of variation in amplitude
NHR,HNR - Two measures of ratio of noise to tonal components in the voice
status - Health status of the subject (one) - Parkinson's, (zero) - healthy
RPDE,D2 - Two nonlinear dynamical complexity measures
DFA - Signal fractal scaling exponent
spread1,spread2,PPE - Three nonlinear measures of fundamental frequency vari
ation

-----------------------------------------------------

Citation Request:

If you use this dataset, please cite the following paper:
'Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Di
sorder Detection',
Little MA, McSharry PE, Roberts SJ, Costello DAE, Moroz IM.
BioMedical Engineering OnLine 2007, 6:23 (26 June 2007)
```

In [15]:

```
data = pd.read_csv(r"C:\Users\Anas Khanooni\Documents\ANAS KHANOONI\ANAS POST-GRD IN AI\Ass
```

In [16]:

```
data.head()
```

Out[16]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.0000
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.0000
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.0000
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.0000
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.0001

5 rows × 24 columns

2. Eye-ball raw data to get a feel of the data

In [19]:

```
#fetch all columns
data.columns
```

Out[19]:

```
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 [20]:

```
#shape
data.shape
```

Out[20]:

```
(195, 24)
```

In [22]:

```
#info
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
```

The data has 195 instances and 24 attributes. 1 integer type, 1 object and 22 float type

In [23]:

```
data.describe()
```

Out[23]:

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

8 rows × 23 columns

In [24]:

```
data.isnull().sum()
```

Out[24]:

```
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 [25]:

```
#Overview of data
data.describe().T
```

Out[25]:

	count	mean	std	min	25%	50%	7
MDVP:Fo(Hz)	195.0	154.228641	41.390065	88.333000	117.572000	148.790000	182.769
MDVP:Fhi(Hz)	195.0	197.104918	91.491548	102.145000	134.862500	175.829000	224.205
MDVP:Flo(Hz)	195.0	116.324631	43.521413	65.476000	84.291000	104.315000	140.018
MDVP:Jitter(%)	195.0	0.006220	0.004848	0.001680	0.003460	0.004940	0.007
MDVP:Jitter(Abs)	195.0	0.000044	0.000035	0.000007	0.000020	0.000030	0.000
MDVP:RAP	195.0	0.003306	0.002968	0.000680	0.001660	0.002500	0.003
MDVP:PPQ	195.0	0.003446	0.002759	0.000920	0.001860	0.002690	0.003
Jitter:DDP	195.0	0.009920	0.008903	0.002040	0.004985	0.007490	0.011
MDVP:Shimmer	195.0	0.029709	0.018857	0.009540	0.016505	0.022970	0.037
MDVP:Shimmer(dB)	195.0	0.282251	0.194877	0.085000	0.148500	0.221000	0.350
Shimmer:APQ3	195.0	0.015664	0.010153	0.004550	0.008245	0.012790	0.020
Shimmer:APQ5	195.0	0.017878	0.012024	0.005700	0.009580	0.013470	0.022
MDVP:APQ	195.0	0.024081	0.016947	0.007190	0.013080	0.018260	0.029
Shimmer:DDA	195.0	0.046993	0.030459	0.013640	0.024735	0.038360	0.060
NHR	195.0	0.024847	0.040418	0.000650	0.005925	0.011660	0.025
HNR	195.0	21.885974	4.425764	8.441000	19.198000	22.085000	25.075
status	195.0	0.753846	0.431878	0.000000	1.000000	1.000000	1.000
RPDE	195.0	0.498536	0.103942	0.256570	0.421306	0.495954	0.587
DFA	195.0	0.718099	0.055336	0.574282	0.674758	0.722254	0.761
spread1	195.0	-5.684397	1.090208	-7.964984	-6.450096	-5.720868	-5.046
spread2	195.0	0.226510	0.083406	0.006274	0.174351	0.218885	0.279
D2	195.0	2.381826	0.382799	1.423287	2.099125	2.361532	2.636
PPE	195.0	0.206552	0.090119	0.044539	0.137451	0.194052	0.252

In [26]:

```
# A skewness value of 0 in the output denotes a symmetrical distribution
# A negative Skewness value in the output denotes tail is larger towards left hand side of
# A positive Skewness value in the output denotes tail is larger towards Right hand side of
data.skew()
```

Out[26]:

```
MDVP:Fo(Hz)         0.591737
MDVP:Fhi(Hz)        2.542146
MDVP:Flo(Hz)        1.217350
MDVP:Jitter(%)      3.084946
MDVP:Jitter(Abs)    2.649071
MDVP:RAP            3.360708
MDVP:PPQ            3.073892
Jitter:DDP          3.362058
MDVP:Shimmer        1.666480
MDVP:Shimmer(dB)    1.999389
Shimmer:APQ3        1.580576
Shimmer:APQ5        1.798697
MDVP:APQ            2.618047
Shimmer:DDA         1.580618
NHR                 4.220709
HNR                -0.514317
status             -1.187727
RPDE               -0.143402
DFA                -0.033214
spread1             0.432139
spread2             0.144430
D2                  0.430384
PPE                 0.797491
dtype: float64
```

Data Understanding

1. There is NO missing data
2. The feature 'name' does not add any information, there is no relationship between 'name' and 'status', We can remove this row.
3. All features are numerical
4. There is lots of variation in units of data, gap between features values is very high, need to scale it.
5. Most of the features are Skewed

3. Using univariate & bivariate analysis to check the individual attributes for their basic statistics.

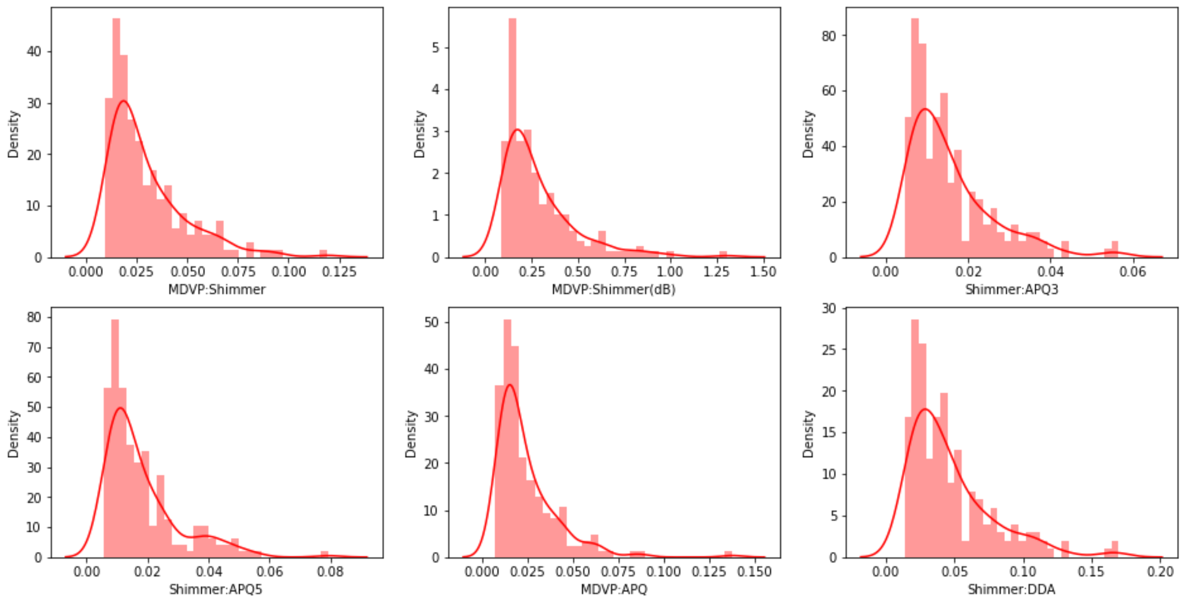
Univariate analysis

In [28]:

```
#Univariate analysis of Fundamental Frequency
ig, axes = plt.subplots(2, 3, figsize=(16, 8))
sns.distplot(data['MDVP:Shimmer'],bins=30,ax=axes[0,0],color='red')
sns.distplot(data['MDVP:Shimmer(dB)'],bins=30,ax=axes[0,1],color='red')
sns.distplot(data['Shimmer:APQ3'],bins=30,ax=axes[0,2],color='red')
sns.distplot(data['Shimmer:APQ5'],bins=30,ax=axes[1,0],color='red')
sns.distplot(data['MDVP:APQ'],bins=30,ax=axes[1,1],color='red')
sns.distplot(data['Shimmer:DDA'],bins=30,ax=axes[1,2],color='red')
```

Out[28]:

```
<AxesSubplot:xlabel='Shimmer:DDA', ylabel='Density'>
```



Observations

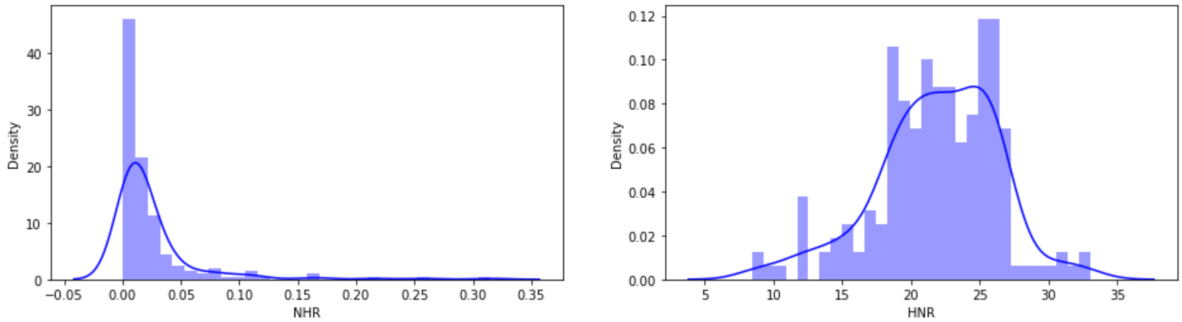
From the above graphs, we can observe that all graphs have almost same distribution, and each graph is positively skewed

In [30]:

```
#analysis for measures of ratio of noise to tonal components in the voice
fig, axes = plt.subplots(1, 2, figsize=(16, 4))
sns.distplot(data['NHR'],bins=30,ax=axes[0],color='blue')
sns.distplot(data['HNR'],bins=30,ax=axes[1],color='blue')
```

Out[30]:

<AxesSubplot:xlabel='HNR', ylabel='Density'>



Observations

NHR is right skewed, most of the values lies around 0.00 to 0.04, so all values are very small.

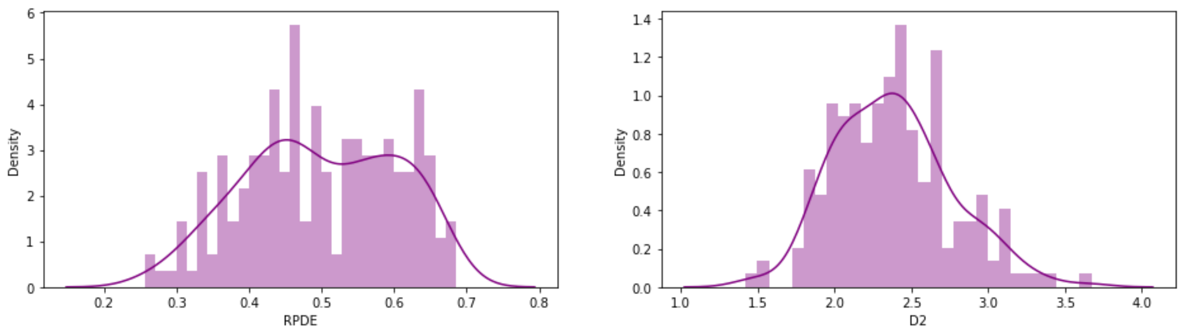
The value HNR seems to be a slight negativeness skewness

In [32]:

```
#Analysis for two non-linear dynamical complexity measures
fig, axes = plt.subplots(1, 2, figsize=(16, 4))
sns.distplot(data['RPDE'],bins=30,ax=axes[0],color='purple')
sns.distplot(data['D2'],bins=30,ax=axes[1],color='purple')
```

Out[32]:

<AxesSubplot:xlabel='D2', ylabel='Density'>



Observations

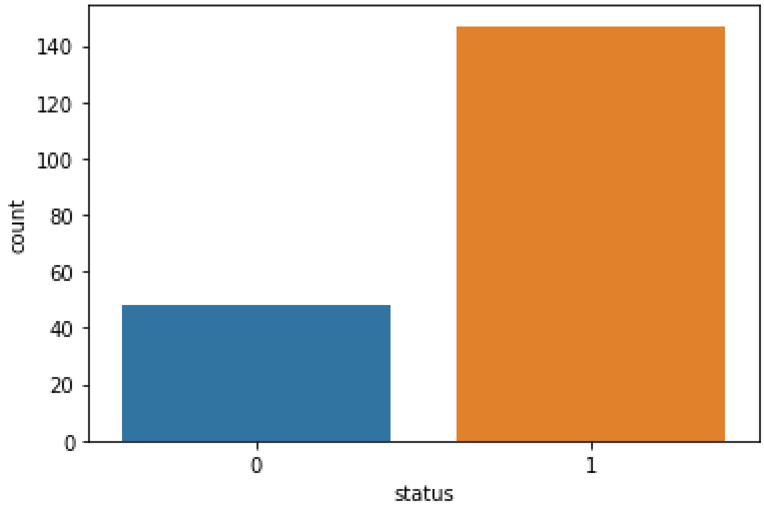
All graphs tends to be normalised graph with few outliers

In [33]:

```
# Variation of target variables
sns.countplot(x=data['status'])
```

Out[33]:

```
<AxesSubplot:xlabel='status', ylabel='count'>
```



Observations

More people (75%) have Parkinson then people not having Parkinson as per given capital

Bi-variate analysis

It would take lots of graphs and time to plot each graph with relation to target variable, so let's first find the most co-related columns and then do bi-vaiate analysis of those columns

In [35]:

```
corr = data.corr()
corr
```

Out[35]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs
MDVP:Fo(Hz)	1.000000	0.400985	0.596546	-0.118003	-0.38202
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.02919
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.27781
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.93571
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.00000
MDVP:RAP	-0.076194	0.097177	-0.100519	0.990276	0.92291
MDVP:PPQ	-0.112165	0.091126	-0.095828	0.974256	0.89777
Jitter:DDP	-0.076213	0.097150	-0.100488	0.990276	0.92291
MDVP:Shimmer	-0.098374	0.002281	-0.144543	0.769063	0.70332
MDVP:Shimmer(dB)	-0.073742	0.043465	-0.119089	0.804289	0.71660
Shimmer:APQ3	-0.094717	-0.003743	-0.150747	0.746625	0.69715
Shimmer:APQ5	-0.070682	-0.009997	-0.101095	0.725561	0.64896
MDVP:APQ	-0.077774	0.004937	-0.107293	0.758255	0.64879
Shimmer:DDA	-0.094732	-0.003733	-0.150737	0.746635	0.69717
NHR	-0.021981	0.163766	-0.108670	0.906959	0.83497
HNR	0.059144	-0.024893	0.210851	-0.728165	-0.65681
status	-0.383535	-0.166136	-0.380200	0.278220	0.33865
RPDE	-0.383894	-0.112404	-0.400143	0.360673	0.44183
DFA	-0.446013	-0.343097	-0.050406	0.098572	0.17503
spread1	-0.413738	-0.076658	-0.394857	0.693577	0.73577
spread2	-0.249450	-0.002954	-0.243829	0.385123	0.38854
D2	0.177980	0.176323	-0.100629	0.433434	0.31069
PPE	-0.372356	-0.069543	-0.340071	0.721543	0.74816

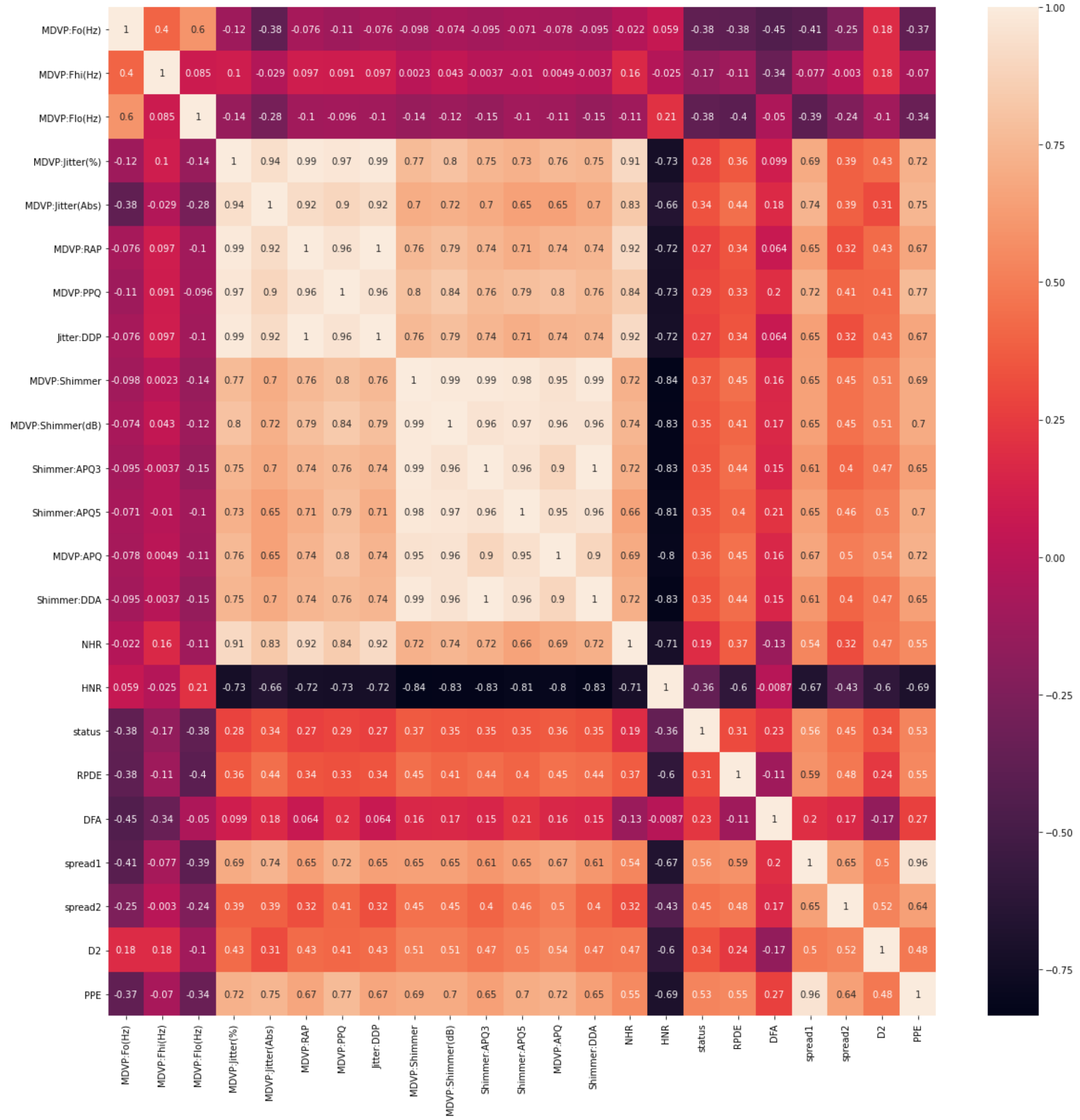
23 rows × 23 columns

In [36]:

```
plt.subplots(figsize=(20,20))
sns.heatmap(corr,annot=True)
```

Out[36]:

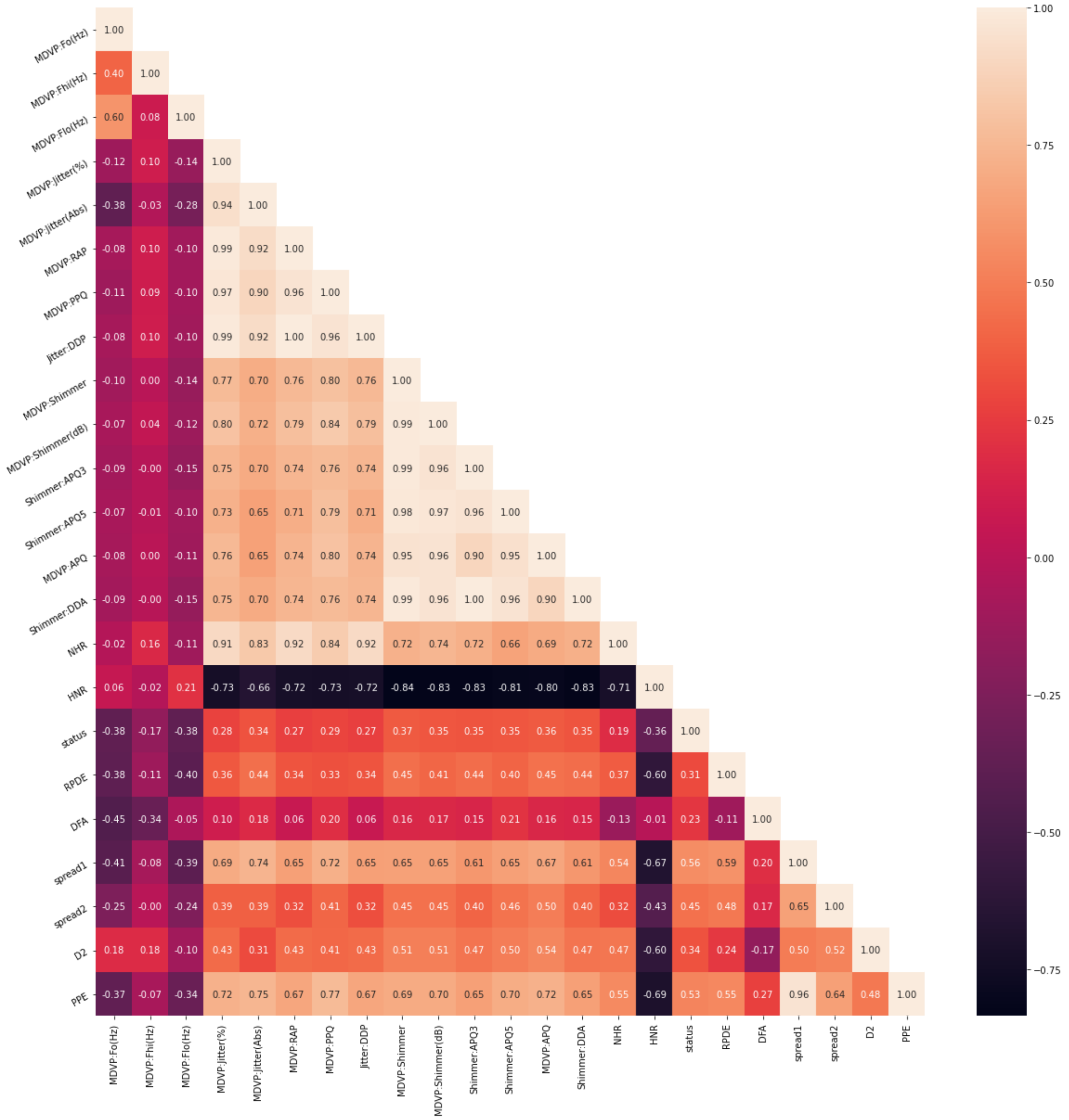
<AxesSubplot:>



create a mask so we only see the correlation values once

In [37]:

```
plt.subplots(figsize=(20,20))
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, 1)] = True
a = sns.heatmap(corr,mask=mask, annot=True, fmt='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```



Observation

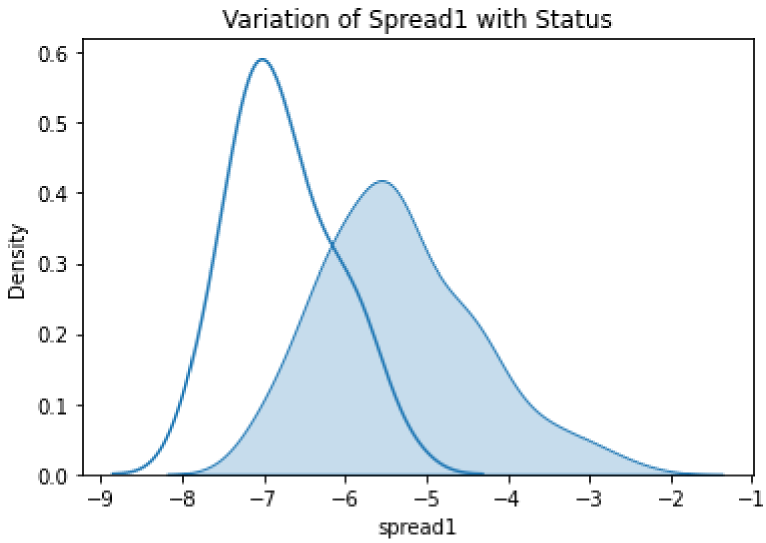
MDVP:Jitter(%) have high correlation i.e. above +90% value with Jitter(Abs),MDVP:RAP,MDVP:PPQ,NHR HNR have High correlation i.e. above -80% with MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA MDVP:Shimmer has a very correlation with MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA These Realtion may be coz they are derived from each other The target variable status has a weak positive corelation with spread1

In [38]:

```
#variation of Spread1 with Target Variable
sns.kdeplot(data[data.status == 0]['spread1'], shade=False,)
sns.kdeplot(data[data.status == 1]['spread1'], shade=True)
plt.title("Variation of Spread1 with Status")
```

Out[38]:

```
Text(0.5, 1.0, 'Variation of Spread1 with Status')
```



Observation

People whose spread1 is between -6.5 and -5 have more chances of having Parkinson

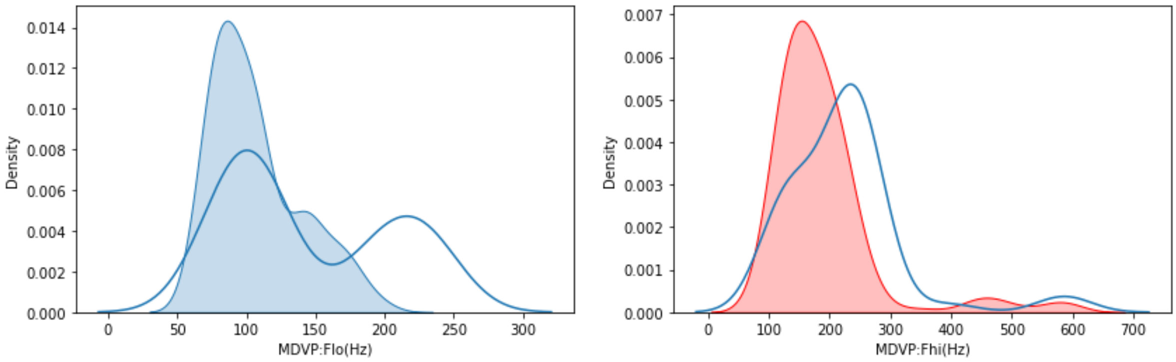
People whose spread1 is between -9.5 and -7.5 have more chances of not having Parkinson

In [54]:

```
#variation of Maximum, Minimum vocal fundamental frequency
fig, ax = plt.subplots(1,2,figsize=(14,4))
sns.kdeplot(data[data.status == 0]['MDVP:Flo(Hz)'], shade=False,ax=ax[0])
sns.kdeplot(data[data.status == 1]['MDVP:Flo(Hz)'], shade=True,ax=ax[0])
sns.kdeplot(data[data.status == 0]['MDVP:Fhi(Hz)'], shade=False,ax=ax[1])
sns.kdeplot(data[data.status == 1]['MDVP:Fhi(Hz)'], shade=True,color='r',ax=ax[1])
```

Out[54]:

```
<AxesSubplot:xlabel='MDVP:Fhi(Hz)', ylabel='Density'>
```



Observations

People with Minimum vocal fundamental frequency above 250 give more evidence of not having Parkinson

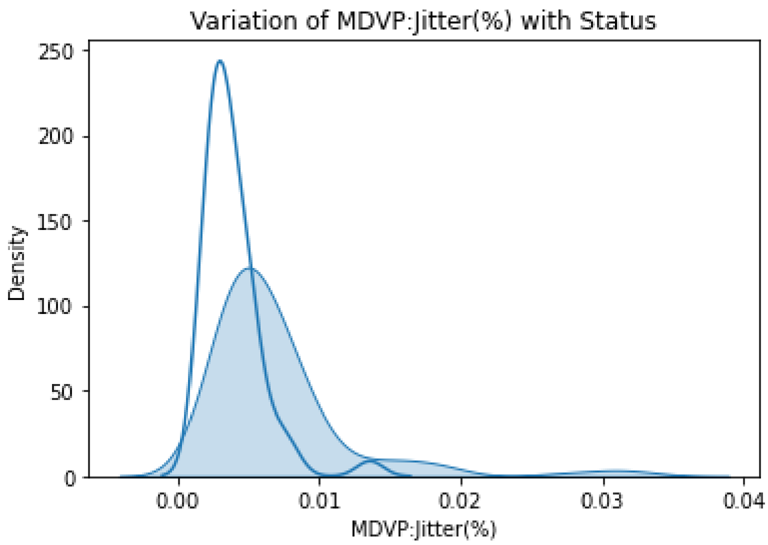
People with Maximum vocal fundamental frequency from 100-200 give more evidence of having Parkinson

In [59]:

```
#variation of Spread1 with Target Variable
sns.kdeplot(data[data.status == 0]['MDVP:Jitter(%)'], shade=False,)
sns.kdeplot(data[data.status == 1]['MDVP:Jitter(%)'], shade=True)
plt.title("Variation of MDVP:Jitter(%) with Status")
```

Out[59]:

```
Text(0.5, 1.0, 'Variation of MDVP:Jitter(%) with Status')
```



Observations

If MDVP:jitter(%) value is >0.005 more likely are the chances of having Parkinson.

4. Split the dataset into training and test set in the ratio of 70:30 (Training:Test)

In [61]:

```
#Split the data into training and test set in the ratio of 70:30 respectively
X = data.drop(['status','name'],axis=1)
y = data['status']
# split data into train subset and test subset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100)
# checking the dimensions of the train & test subset
# to print dimension of train set
print(X_train.shape)
# to print dimension of test set
print(X_test.shape)
```

```
(136, 22)
(59, 22)
```

5. Prepare the Data for Training

In [62]:

```
#checking the variance
#high variance means fearure does not affect the target variable
X_train.var()
```

Out[62]:

```
MDVP:Fo(Hz)          1.744818e+03
MDVP:Fhi(Hz)         7.726066e+03
MDVP:Flo(Hz)         2.018009e+03
MDVP:Jitter(%)       2.117029e-05
MDVP:Jitter(Abs)     1.233427e-09
MDVP:RAP             8.129439e-06
MDVP:PPQ             6.477595e-06
Jitter:DDP           7.317514e-05
MDVP:Shimmer         3.075943e-04
MDVP:Shimmer(dB)     3.001363e-02
Shimmer:APQ3         9.606286e-05
Shimmer:APQ5         1.202009e-04
MDVP:APQ             1.953454e-04
Shimmer:DDA          8.645573e-04
NHR                  1.602961e-03
HNR                  1.737357e+01
RPDE                 1.052263e-02
DFA                  3.032206e-03
spread1              1.090529e+00
spread2              5.837355e-03
D2                   1.197875e-01
PPE                  7.620135e-03
dtype: float64
```

In [63]:

```
#dropping correlated values which are have either more then 80% or less then -80%
X_train.drop(['MDVP:Shimmer','MDVP:Jitter(%)','HNR'],axis=1,inplace=True)
X_test.drop(['MDVP:Shimmer','MDVP:Jitter(%)','HNR'],axis=1,inplace=True)
```

In [64]:

```
#since there is lots of variety in the units of features let's scale it
scaler=StandardScaler().fit(X_train)
scaler_x_train=scaler.transform(X_train)
scaler=StandardScaler().fit(X_test)
scaler_x_test=scaler.transform(X_test)
```

6. Training with Standard Classification Algorithms

Logistic Regression

In [65]:

```
# Train and Fit model
model = LogisticRegression(random_state=0)
model.fit(scaler_x_train, y_train)
```

Out[65]:

```
LogisticRegression(random_state=0)
```

In [66]:

```
#predict the Personal Loan Values
y_pred = model.predict(scaler_x_test)
y_pred
```

Out[66]:

```
array([1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1], dtype=int64)
```

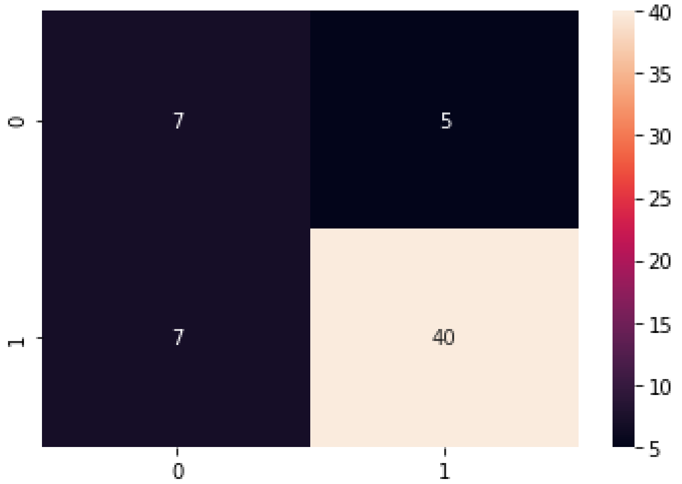
In [67]:

```
# Let's measure the accuracy of this model's prediction
print("confusion_matrix")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot= True)
```

```
confusion_matrix
```

Out[67]:

```
<AxesSubplot:>
```



In [68]:

```
# And some other metrics for Test
cr=classification_report(y_test, y_pred, digits=2)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.50      0.58      0.54        12
           1       0.89      0.85      0.87        47

    accuracy                           0.80        59
   macro avg       0.69      0.72      0.70        59
weighted avg       0.81      0.80      0.80        59
```

Model Scoring

1. Accuracy :: 80%
2. Re-call :: 85%
3. Precision :: 89%
4. F1-Score :: 87%

Ratio in target variable is 75% to 25% so we will take f1-score i.e. 87% as our scoring method

SVM Algorithm

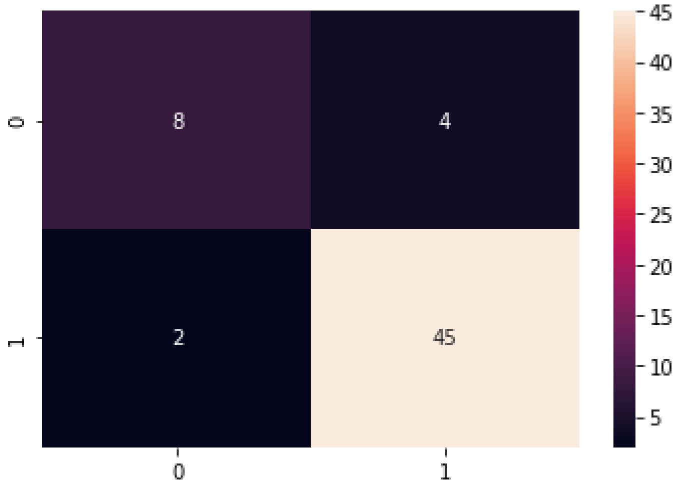
In [69]:

```
clf = SVC(gamma=0.05, C=3,random_state=0)
clf.fit(scaler_x_train , y_train)
# predict the response
prediction_SVC = clf.predict(scaler_x_test)
# Let's measure the accuracy of this model's prediction
print("confusion_matrix")
cm = confusion_matrix(y_test,prediction_SVC)
sns.heatmap(cm, annot= True)
```

```
confusion_matrix
```

Out[69]:

```
<AxesSubplot:>
```



In [70]:

```
# evaluate Model Score
print(classification_report(y_test, prediction_SVC, digits=2))
```

```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73        12
           1       0.92      0.96      0.94        47

    accuracy                           0.90        59
   macro avg       0.86      0.81      0.83        59
weighted avg       0.89      0.90      0.89        59
```

Model Scoring

1. Accuracy :: 90%
2. Re-call :: 96%
3. Precision :: 92%
4. F1-Score :: 94%

Ratio in target variable is 75% to 25% so we will take f1-score i.e. 94% as our scoring method

KNN

In [79]:

```
# creating odd list of K for KNN
myList = list(range(3,40,2))

# empty list that will hold accuracy scores
ac_scores = []

# perform accuracy metrics for values from 3,5....29
for k in myList:
 knn = KNeighborsClassifier(n_neighbors=k)
 knn.fit(scaler_x_train, y_train)

 # predict the response
 y_pred = knn.predict(scaler_x_test)

 # evaluate F1 Score
 scores = f1_score(y_test, y_pred)
 ac_scores.append(scores)

# changing to misclassification error
MSE = [1 - x for x in ac_scores]

# determining best k
optimal_k = myList[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d" % optimal_k)
```

```
The optimal number of neighbors is 29
```

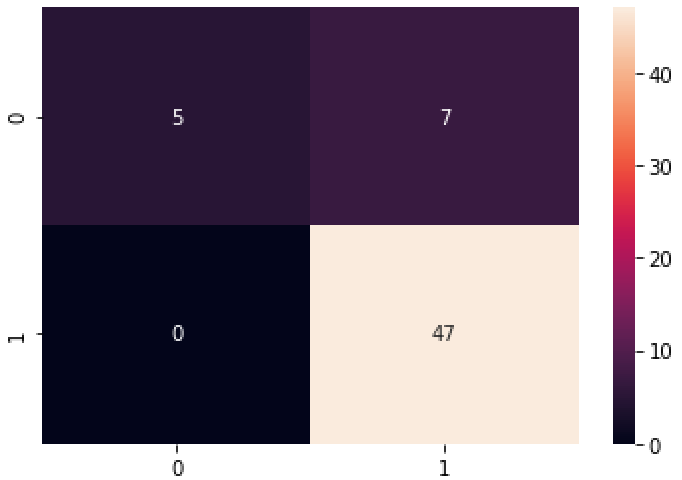
In [80]:

```
# instantiate learning model (k = 29)
knn = KNeighborsClassifier(n_neighbors = 29, weights = 'uniform', metric='euclidean')
# fitting the model
knn.fit(scaler_x_train, y_train)
# predict the response
y_Knn_pred = knn.predict(scaler_x_test)
# Let's measure the accuracy of this model's prediction
print("confusion_matrix")
cm = confusion_matrix(y_test,y_Knn_pred)
sns.heatmap(cm, annot=True)
```

```
confusion_matrix
```

Out[80]:

```
<AxesSubplot:>
```



In [81]:

```
# evaluate Model Score
print(classification_report(y_test, y_Knn_pred, digits=2))
```

```
              precision    recall  f1-score   support

           0       1.00      0.42      0.59        12
           1       0.87      1.00      0.93        47

    accuracy                           0.88        59
   macro avg       0.94      0.71      0.76        59
weighted avg       0.90      0.88      0.86        59
```

Model Scoring

1. Accuracy :: 88%
2. Re-call :: 100%
3. Precision :: 87%
4. F1-Score :: 93%

Ratio in target variable is 75% to 25% so we will take f1-score i.e. 93% as our scoring method

Determining which standard model performed better

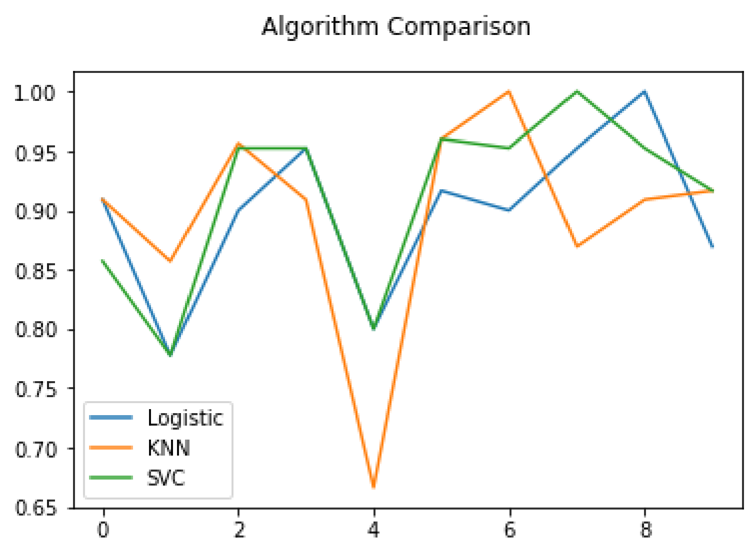
In [88]:

```
#Using K fold to check how my algorighm varies throughout my data if we split it in 10 equa
models = []
models.append(('Logistic Regression', LogisticRegression()))
models.append(('K-NN', KNeighborsClassifier(n_neighbors = 29, weights = 'uniform', metric='
models.append(('SVM', SVC(gamma=0.05, C=3)))
# evaluate each model
results = []
names = []
scoring = 'f1'
for name, model in models:
 kfold = model_selection.KFold(n_splits=10, random_state=101)
 cv_results = model_selection.cross_val_score(model, scaler_x_train, y_train, cv=kfold, sco
 results.append(cv_results)
 names.append(name)
 print("Name = %s , Mean F1-Score = %f, SD F1-Score = %f" % (name, cv_results.mean
(), cv_results.std()))
```

```
Name = Logistic Regression , Mean F1-Score = 0.897786, SD F1-Score = 0.06454
7
Name = K-NN , Mean F1-Score = 0.895384, SD F1-Score = 0.086206
Name = SVM , Mean F1-Score = 0.912111, SD F1-Score = 0.070824
```

In [89]:

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.plot(results[0],label='Logistic')
plt.plot(results[1],label='KNN')
plt.plot(results[2],label='SVC')
plt.legend()
plt.show()
```



Conclusion

After comparing Logistic, KNN, SVC also we can conclude that SVC performed slightly better

7. Train a meta-classifier

Stacking

In [91]:

```
#Stacking the idea of stacking is to learn several different weak learners

# and combine them by training a meta-model to output predictions based on the multiple pre

# returned by these weak models. So, we need to define two things in order to build our sta

# the L learners we want to fit and the meta-model that combines them.

# defining level hetrogenious model

level0 = list()
level0.append(('lr', LogisticRegression()))
level0.append(('knn', KNeighborsClassifier(n_neighbors = 29, weights = 'uniform', metric='e
level0.append(('cart', DecisionTreeClassifier()))
level0.append(('svm', SVC(gamma=0.05, C=3)))
level0.append(('bayes', GaussianNB()))

# define meta learner model
level1 = SVC(gamma=0.05, C=3)

# define the stacking ensemble with cross validation of 5
Stack_model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
```

In [92]:

```
# predict the response
Stack_model.fit(scaler_x_train, y_train)
prediction_Stack = Stack_model.predict(scaler_x_test)
```

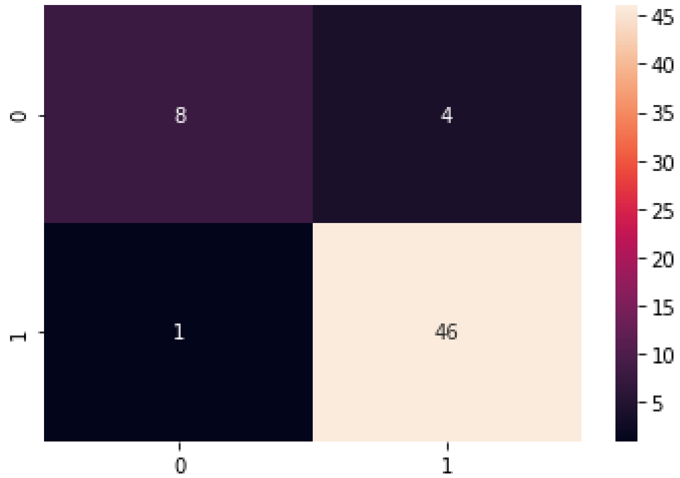
In [93]:

```
# Let's measure the accuracy of this model's prediction
print("confusion_matrix")
cm = confusion_matrix(y_test,prediction_Stack)
sns.heatmap(cm, annot=True)
```

```
confusion_matrix
```

Out[93]:

```
<AxesSubplot:>
```



In [94]:

```
# evaluate Model Score
print(classification_report(y_test, prediction_Stack, digits=2))
```

```
              precision    recall  f1-score   support

           0       0.89      0.67      0.76        12
           1       0.92      0.98      0.95        47

    accuracy                           0.92        59
   macro avg       0.90      0.82      0.86        59
weighted avg       0.91      0.92      0.91        59
```

Model Scoring

1. Accuracy :: 90%
2. Re-call :: 96%
3. Precision :: 92%
4. F1-Score :: 94%

Ratio in target variable is 75% to 25% so we will take f1-score i.e. 94% as our scoring method

AUC-ROC for stacking

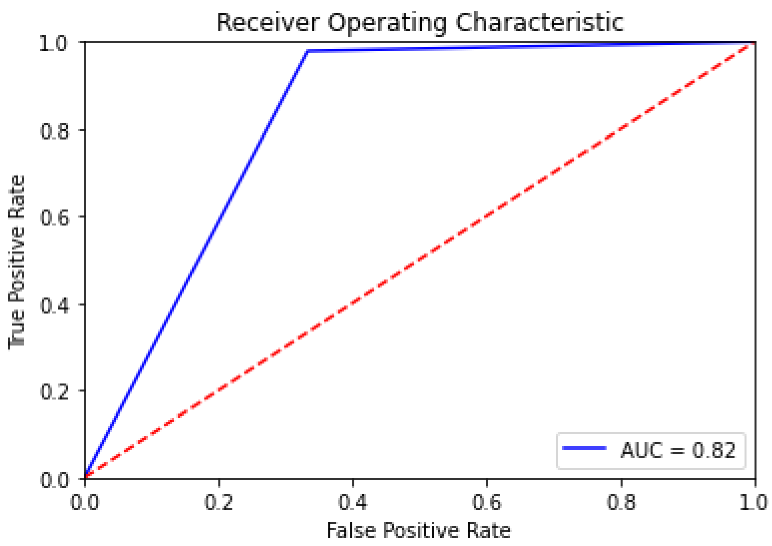
In [95]:

```
#determining false positive rate and True positive rate, threshold
fpr, tpr, threshold = metrics.roc_curve(y_test, prediction_Stack)
roc_auc_stack = metrics.auc(fpr, tpr)
# print AUC
print("AUC : % 1.4f" %(roc_auc_stack))
```

```
AUC :  0.8227
```

In [96]:

```
#plotting ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc_stack)
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.show()
```



8.Standard Ensemble Model

In [98]:

```
#creating model of Random Forest
RandomForest = RandomForestClassifier(n_estimators = 100,criterion='entropy',max_features=1
RandomForest = RandomForest.fit(scaler_x_train, y_train)

# predict the response
RandomForest_pred = RandomForest.predict(scaler_x_test)
```

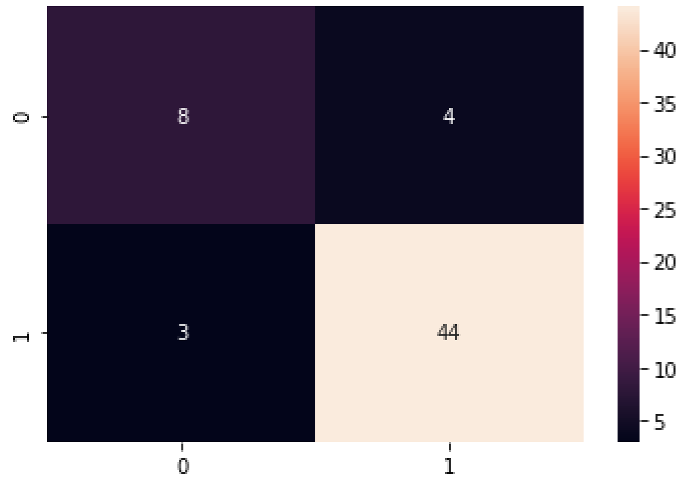
In [99]:

```
# Let's measure the accuracy of this model's prediction
print("confusion_matrix")
cm = confusion_matrix(y_test,RandomForest_pred)
sns.heatmap(cm, annot=True)
```

```
confusion_matrix
```

Out[99]:

```
<AxesSubplot:>
```



In [100]:

```
# evaluate Model Score
print(classification_report(y_test, RandomForest_pred, digits=2))
```

```
              precision    recall  f1-score   support

           0       0.73      0.67      0.70        12
           1       0.92      0.94      0.93        47

    accuracy                           0.88        59
   macro avg       0.82      0.80      0.81        59
weighted avg       0.88      0.88      0.88        59
```

Model Scoring

1. Accuracy :: 88%
2. Re-call :: 94%
3. Precision :: 92%
4. F1-Score :: 93%

Ratio in target variable is 75% to 25% so we will take f1-score i.e. 94% as our scoring method

AOC-ROC for stacking

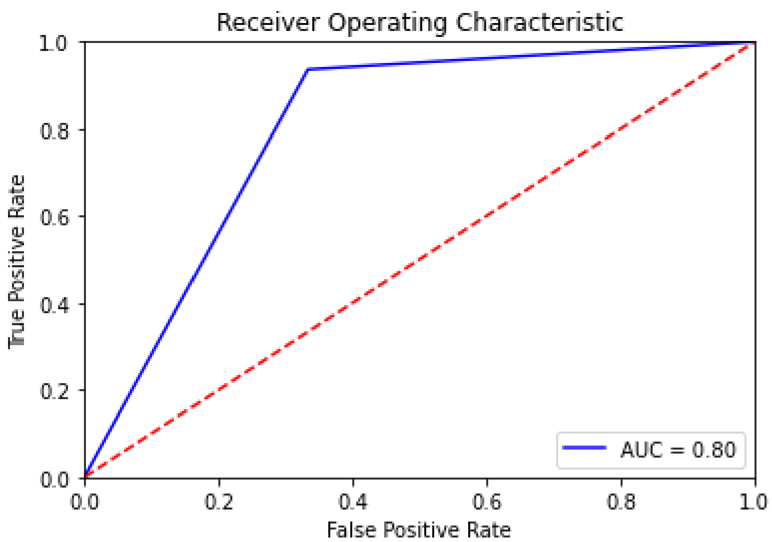
In [101]:

```
#determining false positive rate and True positive rate, threshold
fpr, tpr, threshold = metrics.roc_curve(y_test, RandomForest_pred)
roc_auc_rf = metrics.auc(fpr, tpr)
# print AUC
print("AUC : % 1.4f" %(roc_auc_rf))
```

```
AUC :  0.8014
```

In [102]:

```
#plotting ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc_rf)
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.show()
```



In [105]:

```
# Lets check features importance
feature_imp = pd.Series(RandomForest.feature_importances_,index=X_train.columns).sort_value
feature_imp
```

Out[105]:

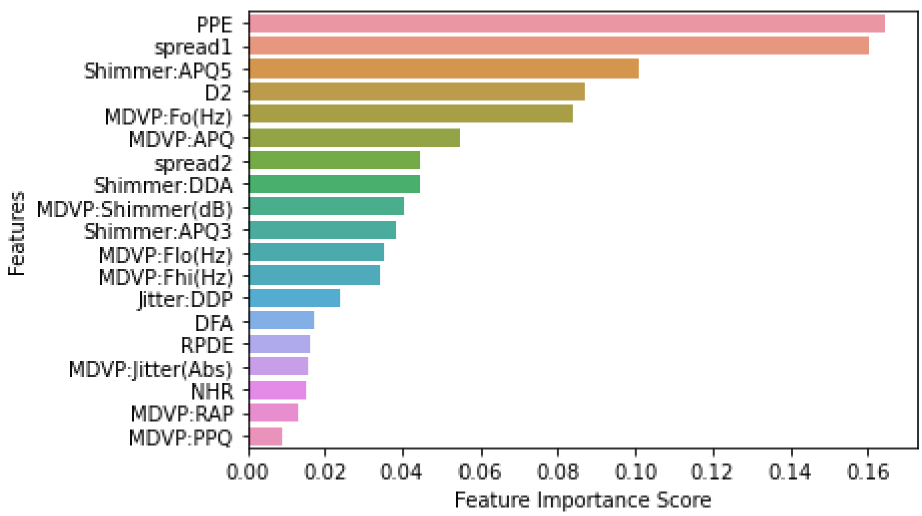
```
PPE                  0.164688
spread1              0.160459
Shimmer:APQ5         0.100701
D2                   0.086762
MDVP:Fo(Hz)          0.084116
MDVP:APQ             0.055122
spread2              0.044662
Shimmer:DDA          0.044605
MDVP:Shimmer(dB)     0.040475
Shimmer:APQ3         0.038352
MDVP:Flo(Hz)         0.035452
MDVP:Fhi(Hz)         0.034257
Jitter:DDP           0.024163
DFA                  0.017156
RPDE                 0.016049
MDVP:Jitter(Abs)     0.015653
NHR                  0.015023
MDVP:RAP             0.013220
MDVP:PPQ             0.009085
dtype: float64
```

In [106]:

```
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
```

Out[106]:

```
Text(0, 0.5, 'Features')
```



Adaptive Boosting

In [107]:

```
#create and fit the model
AdBs = AdaBoostClassifier( n_estimators= 50)
AdBs = AdBs.fit(scaler_x_train, y_train)
# predict the response
AdBs_pred = AdBs.predict(scaler_x_test)
```

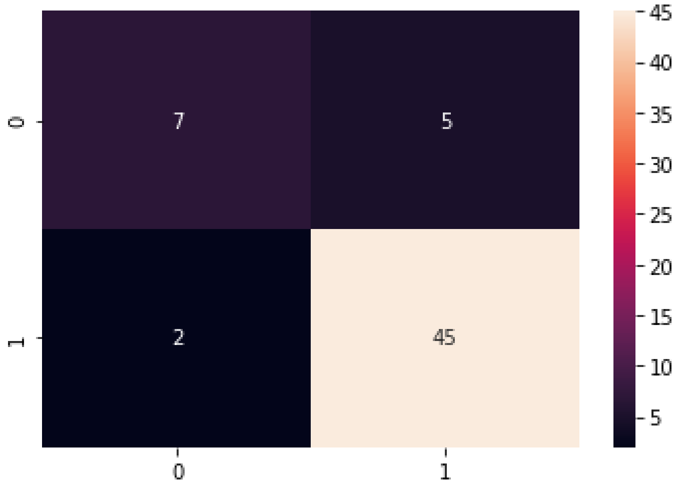
In [108]:

```
# Let's measure the accuracy of this model's prediction
print("confusion_matrix")
cm = confusion_matrix(y_test,AdBs_pred)
sns.heatmap(cm, annot=True)
```

```
confusion_matrix
```

Out[108]:

```
<AxesSubplot:>
```



In [109]:

```
# evaluate Model Score
print(classification_report(y_test, AdBs_pred, digits=2))
```

```
              precision    recall  f1-score   support

           0       0.78      0.58      0.67        12
           1       0.90      0.96      0.93        47

    accuracy                           0.88        59
   macro avg       0.84      0.77      0.80        59
weighted avg       0.88      0.88      0.87        59
```

Model Scoring

1. Accuracy :: 88%
2. Re-call :: 96%
3. Precision :: 90%
4. F1-Score :: 93%

Ratio in target variable is 75% to 25% so we will take f1-score i.e. 93% as our scoring method

AUC-ROC for AdaBoost

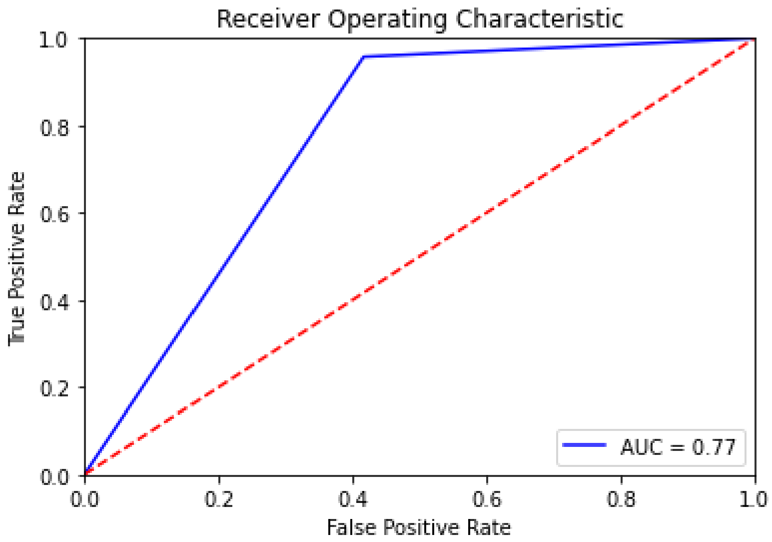
In [111]:

```
#determining false positive rate and True positive rate, threshold
fpr, tpr, threshold = metrics.roc_curve(y_test, AdBs_pred)
roc_auc_ada = metrics.auc(fpr, tpr)
# print AUC
print("AUC : % 1.4f" %(roc_auc_ada))
```

```
AUC :  0.7704
```

In [112]:

```
#plotting ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc_ada)
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.show()
```



9. Compare all models

In [117]:

```
#Using K fold to check how my algorighm varies throughout my data if we split it in 10 equa
models = []
models.append(('Logistic Regression', LogisticRegression()))
models.append(('K-NN', KNeighborsClassifier(n_neighbors = 29, weights = 'uniform', metric='
models.append(('SVM', SVC(gamma=0.05, C=3)))
models.append(('Stacking', StackingClassifier(estimators=level0, final_estimator = level1,
models.append(('Random Forest', RandomForestClassifier(n_estimators = 100,criterion='entrop
models.append(('Adaptive Boosting', AdaBoostClassifier( n_estimators= 50)))

# evaluate each model

results = []
names = []
scoring = 'accuracy'
for name, model in models:
 kfold = model_selection.KFold(n_splits=10, random_state=101)
 cv_results = model_selection.cross_val_score(model, scaler_x_train, y_train, cv=kfold, sco
 results.append(cv_results)
 names.append(name)
 print("Name = %s , Mean Accuracy = %f, Max Accuracy = %f, SD Accuracy = %f" % (name, cv_re
```

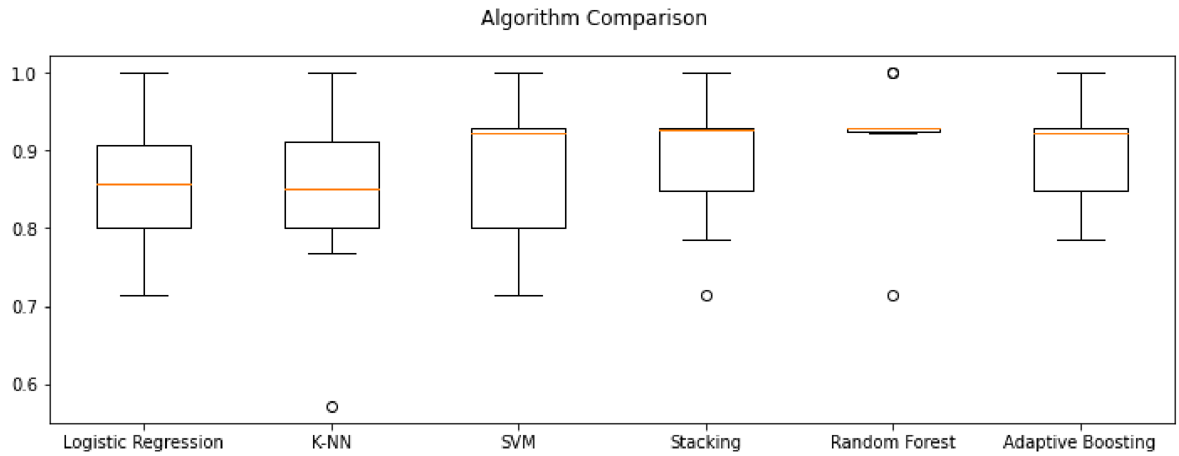
```
Name = Logistic Regression , Mean Accuracy = 0.853846, Max Accuracy = 0.8538
46, SD Accuracy = 0.079441
Name = K-NN , Mean Accuracy = 0.839011, Max Accuracy = 0.839011, SD Accuracy
= 0.110467
Name = SVM , Mean Accuracy = 0.876374, Max Accuracy = 0.876374, SD Accuracy
= 0.084524
Name = Stacking , Mean Accuracy = 0.891209, Max Accuracy = 0.891209, SD Accu
racy = 0.086311
Name = Random Forest , Mean Accuracy = 0.920330, Max Accuracy = 0.920330, SD
Accuracy = 0.074501
Name = Adaptive Boosting , Mean Accuracy = 0.890659, Max Accuracy = 0.89065
9, SD Accuracy = 0.065955
```

In [118]:

```
# boxplot algorithm comparison
fig = plt.figure(figsize=(12,4))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot()
plt.boxplot(results)
ax.set_xticklabels(names)
```

Out[118]:

```
[Text(1, 0, 'Logistic Regression'),
 Text(2, 0, 'K-NN'),
 Text(3, 0, 'SVM'),
 Text(4, 0, 'Stacking'),
 Text(5, 0, 'Random Forest'),
 Text(6, 0, 'Adaptive Boosting')]
```



Conclusion

Random Forest Algrorithm performs better in terms of overall performace

In [ ]: