NAME:P.LAKSHMI NARASIMHA SHATHAMANYU

COLLEGE:G.PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY

YEAR:3rd YEAR

ACADEMIC YEAR:2021-2023

MAJOR-1

In [28]:

```
# Load libraries import numpy as np import
pandas as pd from matplotlib import pyplot from
pandas import read_csv, set_option from
pandas.plotting import scatter_matrix from
sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression from sklearn.tree import
DecisionTreeClassifier from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB from sklearn.svm import
SVC from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomForestCl

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score import

os In [29]:
```

```
os.getcwd()
```

Out[29]:

```
'D:\\Bharath\\Internship\\Projects'
```

In [30]:

```
os.chdir ('D:\\Bharath\\Internship\\Projects')
os.getcwd()
```

Out[30]:

```
'D:\\Bharath\\Internship\\Projects'
```

In [31]:

```
dataset=pd.read_csv('sonar.all-data.csv')
display(dataset)
```

0	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072
5	5.230e-02	0.084	0.069	0.118	0.258	0.216	0.348	0.334	0.287	...	8.400e-03	8.900e-03	0.005	9.400e-03
6	5.820e-02	0.110	0.108	0.097	0.228	0.243	0.377	0.560	0.619	...	2.320e-02	1.660e-02	0.009	1.800e-02
0	1.710e-02	0.062	0.021	0.021	0.037	0.110	0.128	0.060	0.126	...	1.210e-02	3.600e-03	0.015	8.500e-03
6	6.660e-02	0.048	0.039	0.059	0.065	0.121	0.247	0.356	0.446	...	3.100e-03	5.400e-03	0.011	1.100e-02
9	4.530e-02	0.028	0.017	0.038	0.099	0.120	0.183	0.210	0.304	...	4.500e-03	1.400e-03	0.004	1.300e-03
2	9.560e-02	0.132	0.141	0.167	0.171	0.073	0.140	0.208	0.351	...	2.010e-02	2.480e-02	0.013	7.000e-03
2	5.480e-02	0.084	0.032	0.116	0.092	0.103	0.061	0.146	0.284	...	8.100e-03	1.200e-02	0.004	1.210e-02

In [32]:

```
dataset.shape
```

In
Out[32]:

```
(207, 61)
```

[33]:

```
set_option('display.max_rows', 500)
dataset.dtypes
```

Out[33]:

```
0.0200    float64
0.0371    float64
0.0428    float64
0.0207    float64
0.0954    float64
0.0986    float64
0.1539    float64
0.1601    float64
0.3109    float64
0.2111    float64
0.1609    float64
0.1582    float64
0.2238    float64
0.0645    float64
0.0660    float64
0.2273    float64
0.3100    float64
0.2999    float64
0.5078    float64
0.4797    float64
0.5783    float64
0.5071    float64
0.4328    float64
0.5550    float64
0.6711    float64
0.6415    float64
0.7104    float64
0.8080    float64
0.6791    float64
0.3857    float64
0.1307    float64
0.2604    float64
0.5121    float64
0.7547    float64
0.8537    float64
0.8507    float64
0.6692    float64
0.6097    float64
0.4943    float64
0.2744    float64
0.0510    float64
0.2834    float64
0.2825    float64
0.4256    float64
0.2641    float64
0.1386    float64
0.1051    float64
0.1343    float64
0.0383    float64
0.0324    float64
0.0232    float64
0.0027    float64
0.0065    float64
0.0159    float64
```

In

```
0.0072    float64 0.0167    float64 0.0180    float64 0.0084    float64
0.0090    float64 0.0032    float64 R          object dtype: object
```

[34]:

```
# peek at data
set_option('display.width', 100)
dataset.head(20)
```

Out[34]:

	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0
0	0.045	0.052	0.084	0.069	0.118	0.258	0.216	0.348	0.334	0.287	...	0.008	
1	0.026	0.058	0.110	0.108	0.097	0.228	0.243	0.377	0.560	0.619	...	0.023	
2	0.010	0.017	0.062	0.021	0.021	0.037	0.110	0.128	0.060	0.126	...	0.012	
3	0.076	0.067	0.048	0.039	0.059	0.065	0.121	0.247	0.356	0.446	...	0.003	
4	0.029	0.045	0.028	0.017	0.038	0.099	0.120	0.183	0.210	0.304	...	0.004	
5	0.032	0.096	0.132	0.141	0.167	0.171	0.073	0.140	0.208	0.351	...	0.020	
6	0.052	0.055	0.084	0.032	0.116	0.092	0.103	0.061	0.146	0.284	...	0.008	
7	0.022	0.037	0.048	0.048	0.065	0.059	0.075	0.010	0.068	0.149	...	0.015	
8	0.016	0.017	0.035	0.007	0.019	0.067	0.106	0.070	0.096	0.025	...	0.009	
9	0.004	0.006	0.015	0.034	0.031	0.028	0.040	0.027	0.032	0.045	...	0.006	
10	0.012	0.031	0.017	0.031	0.036	0.010	0.018	0.058	0.112	0.084	...	0.013	
11	0.008	0.009	0.005	0.025	0.034	0.055	0.053	0.096	0.101	0.124	...	0.018	
12	0.009	0.006	0.025	0.049	0.120	0.159	0.139	0.099	0.096	0.190	...	0.006	
13	0.012	0.043	0.060	0.045	0.060	0.035	0.053	0.034	0.105	0.212	...	0.008	
14	0.030	0.061	0.065	0.092	0.162	0.229	0.218	0.203	0.146	0.085	...	0.003	
15	0.035	0.012	0.019	0.047	0.074	0.118	0.168	0.154	0.147	0.291	...	0.035	
16	0.019	0.061	0.038	0.077	0.139	0.081	0.057	0.022	0.104	0.119	...	0.033	
17	0.027	0.009	0.015	0.028	0.041	0.076	0.103	0.114	0.079	0.152	...	0.008	
18	0.013	0.015	0.064	0.173	0.257	0.256	0.295	0.411	0.498	0.592	...	0.009	
19	0.047	0.051	0.082	0.125	0.178	0.307	0.301	0.236	0.383	0.376	...	0.019	

20 rows × 61 columns

In
In [35]:

```
# describe data
set_option('precision', 3)
dataset.describe()
```

Out[35]:

	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.21
count	207.000	2.070e+02	207.000	207.000	207.000	207.000	207.000	207.000	207.000	207.0
mean	0.029	3.844e-02	0.044	0.054	0.075	0.105	0.122	0.135	0.177	0.2
std	0.023	3.304e-02	0.039	0.047	0.056	0.059	0.062	0.085	0.118	0.1
min	0.002	6.000e-04	0.002	0.006	0.007	0.010	0.003	0.005	0.007	0.0
25%	0.013	1.640e-02	0.019	0.024	0.038	0.067	0.081	0.080	0.097	0.1
50%	0.023	3.080e-02	0.034	0.044	0.062	0.092	0.106	0.112	0.152	0.1
75%	0.036	4.810e-02	0.058	0.066	0.101	0.134	0.153	0.170	0.231	0.2
max	0.137	2.339e-01	0.306	0.426	0.401	0.382	0.373	0.459	0.683	0.7

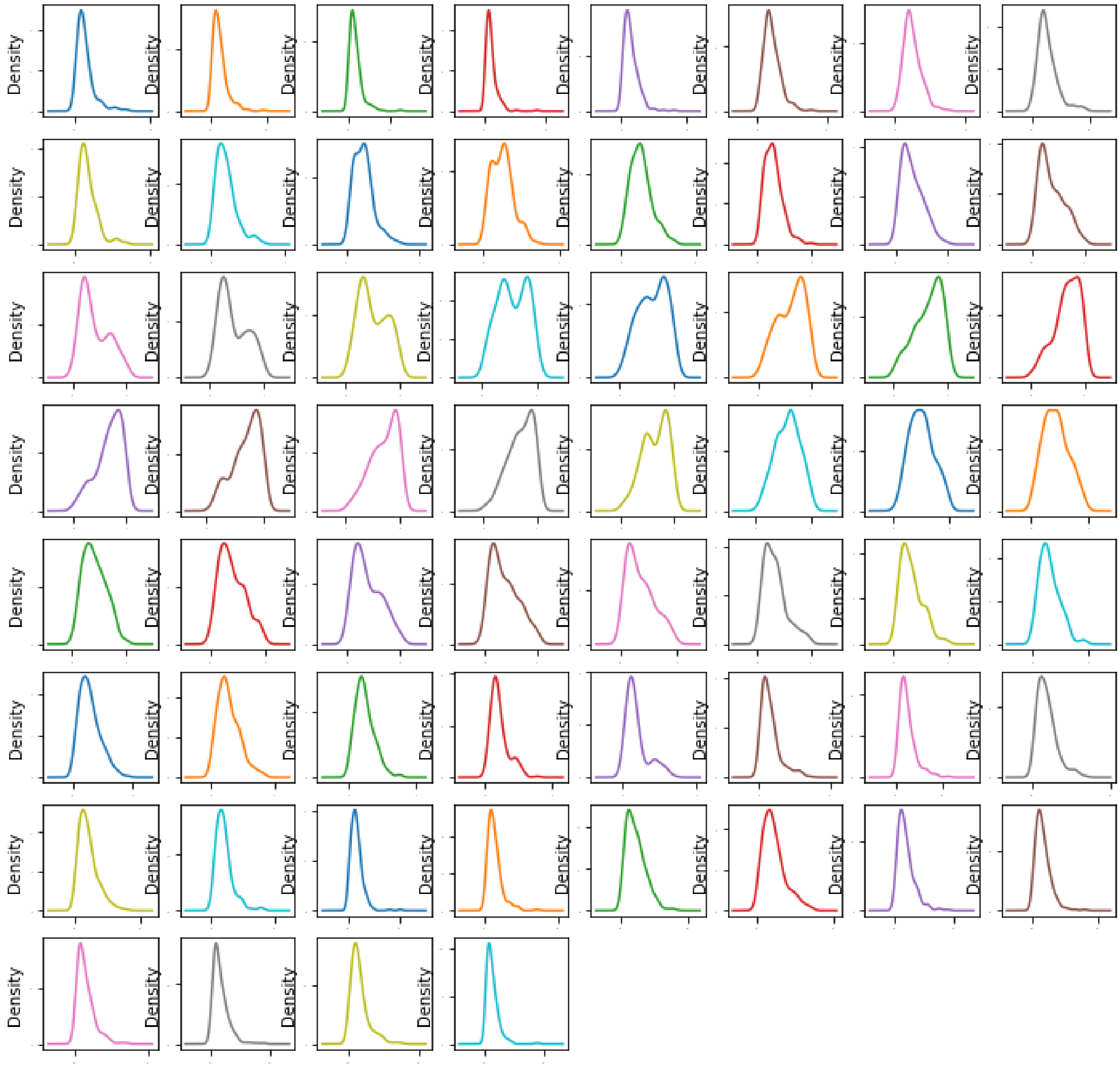
8 rows × 60 columns

[37]:

```
# histograms
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1, figsize=(12,12))
pyplot.show()
```

In
In [38]:

```
# density
dataset.plot(kind='density', subplots=True, layout=(8,8), sharex=False, legend=False, fonts
pyplot.show()
```

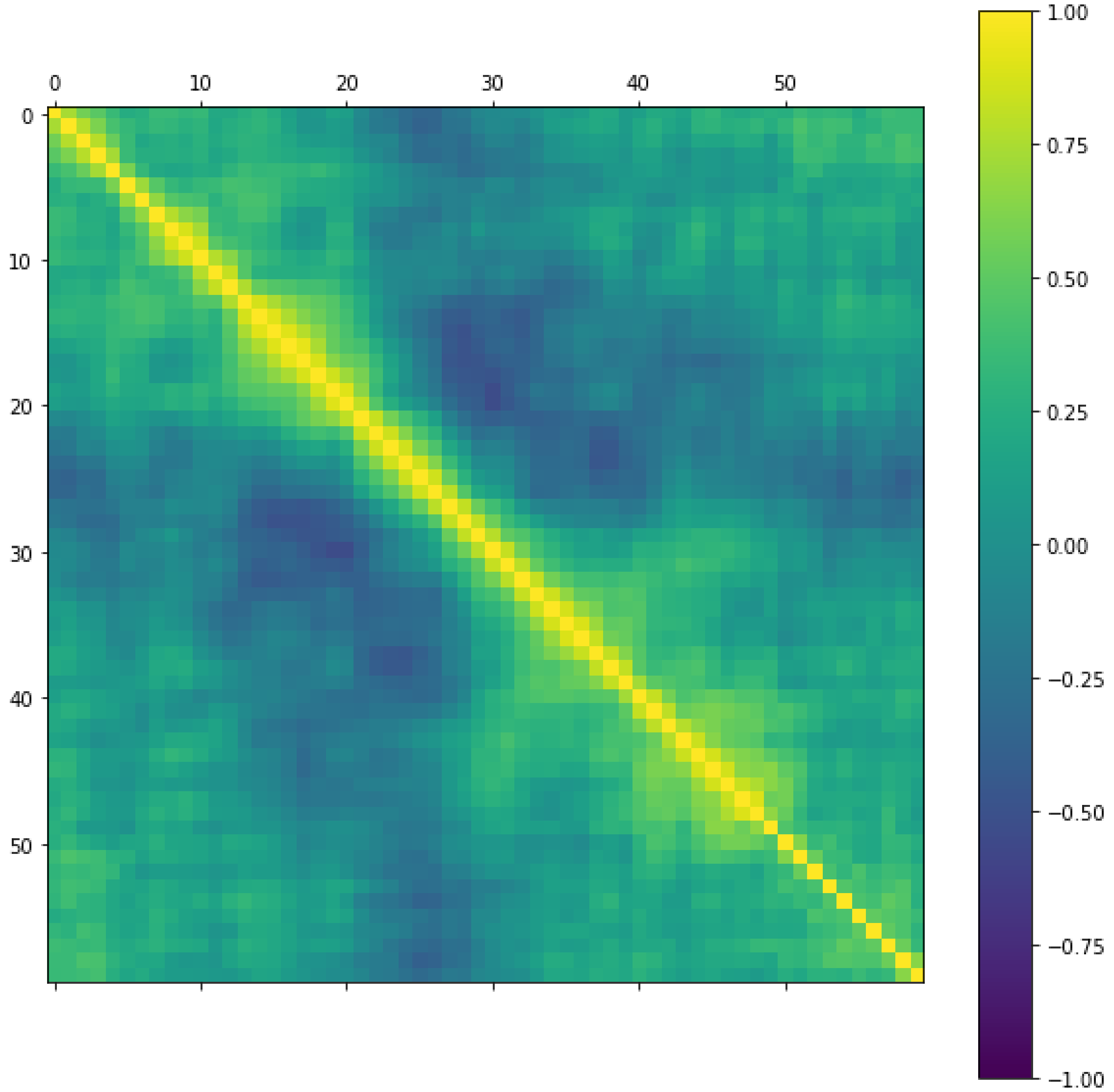


In [39]:

```
# box and whisker
#dataset.plot(kind='box', subplots=True, layout=(8,8), sharex=False, sharey=False, fontsize
#pyplot.show()
```

[40]:

```
# correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(dataset.corr(), vmin=-1, vmax=1, interpolation='none')
fig.colorbar(cax)
fig.set_size_inches(10,10)
pyplot.show()
```



In [45]:

```
# split out validation dataset for the end
array = dataset.values
X = array[:,0:-1].astype(float)
Y = array[:,-1]
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=validation_
```

In [46]:

```
# test options
num_folds = 10
seed = 7
scoring = 'accuracy'
```

In [47]:

```
# spot check some algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

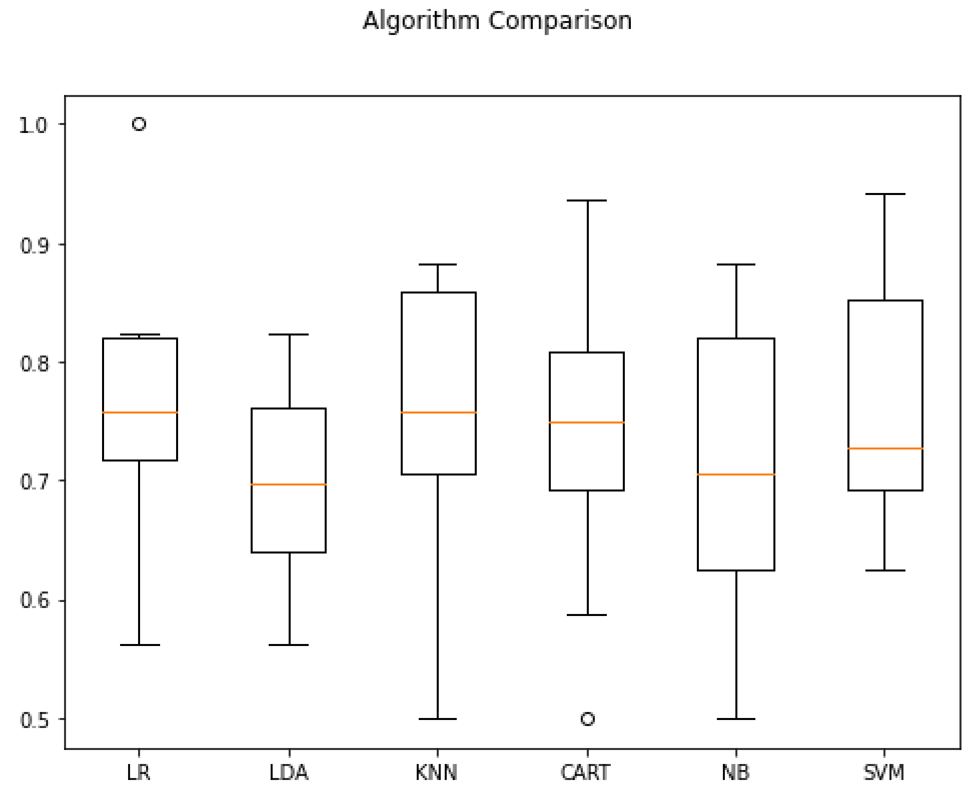
In [52]:

```
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds, random_state=None)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.769853 (0.106086)
LDA: 0.701838 (0.086338)
KNN: 0.756618 (0.111644)
CART: 0.733088 (0.117408)
NB: 0.700368 (0.132161)
SVM: 0.768750 (0.106420)
```

In [53]:

```
# compare algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
fig.set_size_inches(8,6)
pyplot.show()
```



[54]:

```
# standardized the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()), ('LR', LogisticRegres
pipelines.append(('ScaledLDA', Pipeline([('Scaler', StandardScaler()), ('LDA',
LinearDiscri pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),
('KNN', KNeighborsCl pipelines.append(('ScaledCART', Pipeline([('Scaler',
StandardScaler()), ('CART', DecisionTr pipelines.append(('ScaledNB', Pipeline([('Scaler',
StandardScaler()), ('NB', GaussianNB())] pipelines.append(('ScaledSVM',
Pipeline([('Scaler', StandardScaler()), ('SVM', SVC())]))) In [56]:
```

In

```
results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=num_folds, random_state=None)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

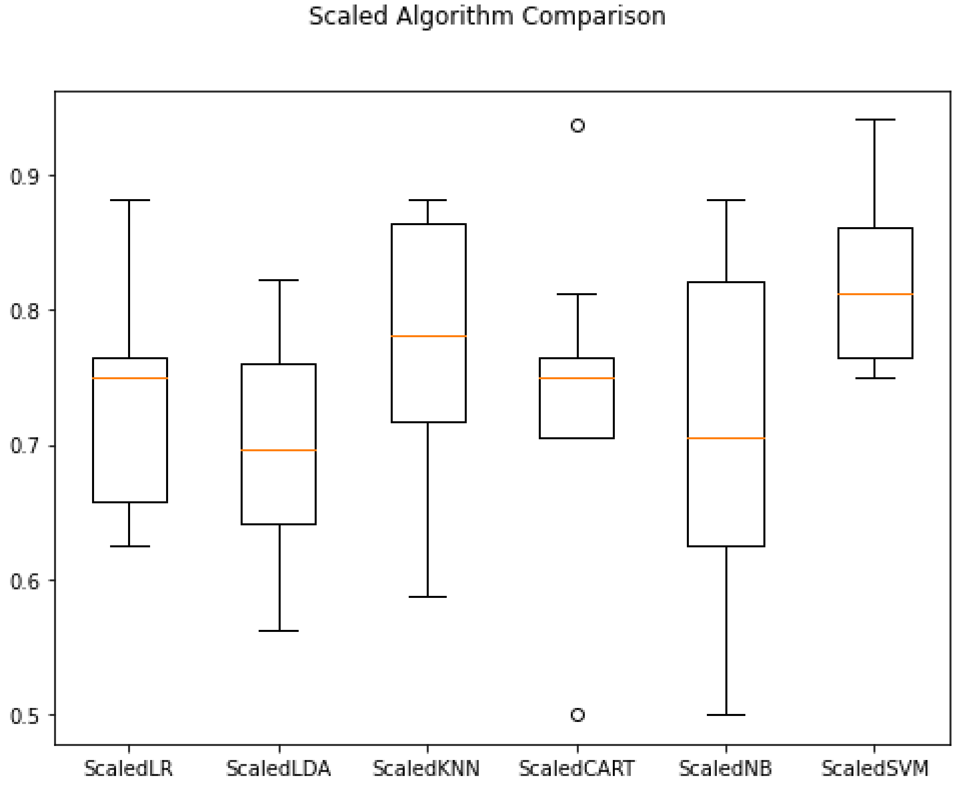
```
ScaledLR: 0.733088 (0.077279)
ScaledLDA: 0.701838 (0.086338)
ScaledKNN: 0.775368 (0.092709)
ScaledCART: 0.739706 (0.103332)
ScaledNB: 0.700368 (0.132161) ScaledSVM:
0.824632 (0.067452)
```

In [ ]:

```

```

In [57]:

```
# compare scaled algorithms
fig = pyplot.figure()
fig.suptitle('Scaled Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
fig.set_size_inches(8,6)
pyplot.show()
```



In [60]:

```
# KNN algorithm tuning
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
neighbors = [1,3,5,7,9,11,13,15,17,19,21]
param_grid = dict(n_neighbors=neighbors)
model = KNeighborsClassifier()
kfold = KFold(n_splits=num_folds, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
```

[61]:

In

```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score'] stds =
grid_result.cv_results_['std_test_score'] params =
grid_result.cv_results_['params'] ranks =
grid_result.cv_results_['rank_test_score'] for mean, stdev, param, rank in
zip(means, stds, params, ranks):     print("#%d %f (%f) with: %r" % (rank,
mean, stdev, param))
```

```
Best: 0.830147 using {'n_neighbors': 1}
#1 0.830147 (0.066580) with: {'n_neighbors': 1}
#2 0.818015 (0.065360) with: {'n_neighbors': 3}
#3 0.781618 (0.088590) with: {'n_neighbors': 5}
#4 0.751838 (0.054834) with: {'n_neighbors': 7}
#5 0.720956 (0.073923) with: {'n_neighbors': 9}
#6 0.697794 (0.049045) with: {'n_neighbors': 11}
#8 0.684926 (0.076167) with: {'n_neighbors': 13}
#7 0.685294 (0.068351) with: {'n_neighbors': 15}
#9 0.679412 (0.078831) with: {'n_neighbors': 17}
#11 0.672426 (0.068632) with: {'n_neighbors': 19}
#10 0.678676 (0.060727) with: {'n_neighbors': 21}
```

In [63]:

```
# SVM algorithm tuning
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
param_grid = dict(C=c_values, kernel=kernel_values) model =
SVC()
kfold = KFold(n_splits=num_folds, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
```

[64]:

```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score'] stds =
grid_result.cv_results_['std_test_score'] params =
grid_result.cv_results_['params'] ranks =
grid_result.cv_results_['rank_test_score'] for mean, stdev, param, rank in
zip(means, stds, params, ranks):     print("#%d %f (%f) with: %r" % (rank,
mean, stdev, param))
```

```
Best: 0.831250 using {'C': 1.5, 'kernel': 'rbf'}
#10 0.757721 (0.045667) with: {'C': 0.1, 'kernel': 'linear'}
#40 0.525368 (0.175329) with: {'C': 0.1, 'kernel': 'poly'}
#39 0.534926 (0.097883) with: {'C': 0.1, 'kernel': 'rbf'}
#37 0.659191 (0.101074) with: {'C': 0.1, 'kernel': 'sigmoid'}
```

In

```
#16 0.738603 (0.087951) with: {'C': 0.3, 'kernel': 'linear'}
#38 0.634191 (0.155287) with: {'C': 0.3, 'kernel': 'poly'}
#13 0.744853 (0.077872) with: {'C': 0.3, 'kernel': 'rbf'}
#31 0.708824 (0.102741) with: {'C': 0.3, 'kernel': 'sigmoid'}
#35 0.701471 (0.104166) with: {'C': 0.5, 'kernel': 'linear'}
#36 0.670956 (0.164123) with: {'C': 0.5, 'kernel': 'poly'}
#8 0.781618 (0.063661) with: {'C': 0.5, 'kernel': 'rbf'}
#26 0.715441 (0.109188) with: {'C': 0.5, 'kernel': 'sigmoid'}
#25 0.720588 (0.099060) with: {'C': 0.7, 'kernel': 'linear'}
#29 0.709191 (0.141384) with: {'C': 0.7, 'kernel': 'poly'}
#7 0.794485 (0.060309) with: {'C': 0.7, 'kernel': 'rbf'}
#27 0.715074 (0.113950) with: {'C': 0.7, 'kernel': 'sigmoid'}
#34 0.708088 (0.095859) with: {'C': 0.9, 'kernel': 'linear'}
#30 0.708824 (0.154033) with: {'C': 0.9, 'kernel': 'poly'}
#6 0.794853 (0.069713) with: {'C': 0.9, 'kernel': 'rbf'}
#31 0.708824 (0.123464) with: {'C': 0.9, 'kernel': 'sigmoid'}
#28 0.714338 (0.092206) with: {'C': 1.0, 'kernel': 'linear'}
#22 0.720956 (0.158314) with: {'C': 1.0, 'kernel': 'poly'}
#5 0.800735 (0.069403) with: {'C': 1.0, 'kernel': 'rbf'}
#31 0.708824 (0.123464) with: {'C': 1.0, 'kernel': 'sigmoid'}
#20 0.726838 (0.096111) with: {'C': 1.3, 'kernel': 'linear'}
#12 0.745221 (0.125098) with: {'C': 1.3, 'kernel': 'poly'}
#3 0.819118 (0.078539) with: {'C': 1.3, 'kernel': 'rbf'}
#22 0.720956 (0.115226) with: {'C': 1.3, 'kernel': 'sigmoid'}
#20 0.726838 (0.096111) with: {'C': 1.5, 'kernel': 'linear'}
#13 0.744853 (0.137799) with: {'C': 1.5, 'kernel': 'poly'}
#1 0.831250 (0.073095) with: {'C': 1.5, 'kernel': 'rbf'}
#18 0.732721 (0.147050) with: {'C': 1.5, 'kernel': 'sigmoid'}
#22 0.720956 (0.102407) with: {'C': 1.7, 'kernel': 'linear'}
#15 0.738971 (0.130346) with: {'C': 1.7, 'kernel': 'poly'}
#4 0.818750 (0.069815) with: {'C': 1.7, 'kernel': 'rbf'}
#17 0.738235 (0.149473) with: {'C': 1.7, 'kernel': 'sigmoid'}
#19 0.727206 (0.086453) with: {'C': 2.0, 'kernel': 'linear'}
#11 0.751103 (0.112827) with: {'C': 2.0, 'kernel': 'poly'}
#2 0.830882 (0.073768) with: {'C': 2.0, 'kernel': 'rbf'}
#9 0.769485 (0.117889) with: {'C': 2.0, 'kernel': 'sigmoid'}
```

In [65]:

```
# ensembles
ensembles = []
# Boosting methods
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
# Bagging methods
ensembles.append(('RF', RandomForestClassifier()))
ensembles.append(('ET', ExtraTreesClassifier()))
```

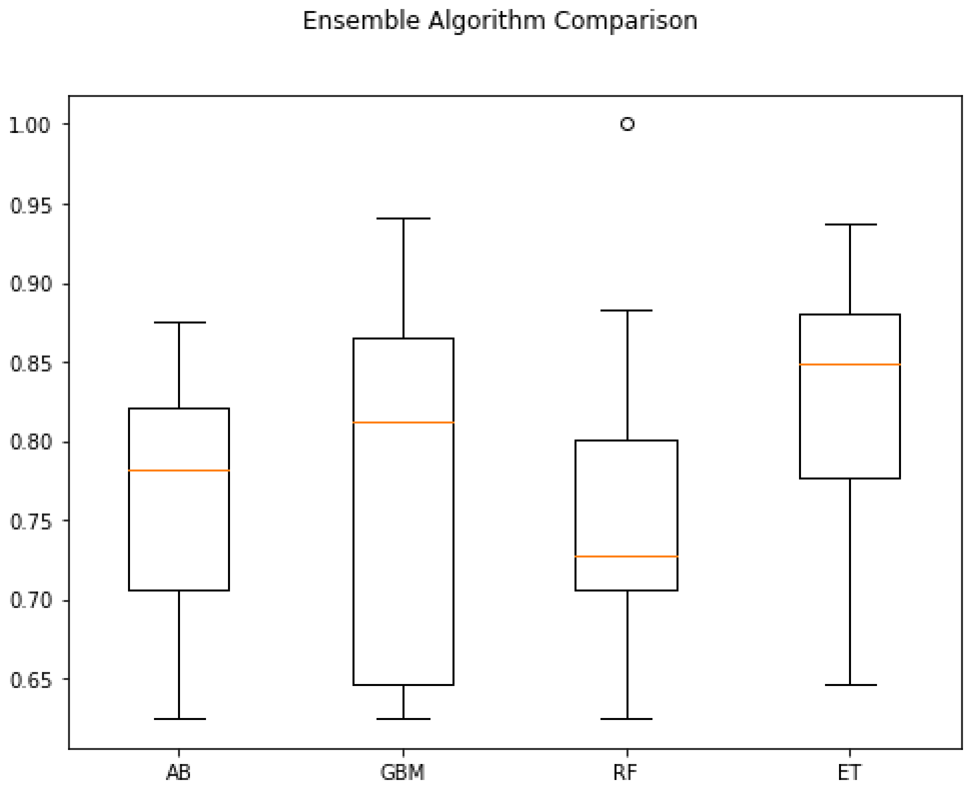
In [67]:

```
results = []
names = []
for name, model in ensembles:
    kfold = KFold(n_splits=num_folds, random_state=None)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
AB: 0.758088 (0.079819)
GBM: 0.776471 (0.118890)
RF: 0.763971 (0.103710)
ET: 0.825000 (0.080819)
```

In [68]:

```
# compare ensemble algorithms
fig = pyplot.figure()
fig.suptitle('Ensemble Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
fig.set_size_inches(8,6)
pyplot.show()
```



In [69]:

```
# prepare model
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = SVC(C=1.5) # rbf is default kernel
model.fit(rescaledX, Y_train)
```

Out[69]:

SVC(C=1.5) [70]:

```
# estimate accuracy on validation set
rescaledValidationX = scaler.transform(X_validation)
```

In

```
predictions = model.predict(rescaledValidationX)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9285714285714286
[[24  2] [ 1 15]]              precision    recall
 f1-score   support

           M       0.96      0.92      0.94        26
 R       0.88      0.94      0.91        16

    accuracy                           0.93        42

macro avg       0.92       0.93       0.93         42

weighted avg       0.93      0.93      0.93        42
```

In [71]:

```
predictions
```

Out[71]:

```
array(['M', 'M', 'R', 'R', 'M', 'R', 'M', 'R', 'R', 'R', 'M', 'R', 'M',
        'R', 'M', 'M', 'R', 'M', 'M', 'R', 'M', 'R', 'M', 'M', 'M', 'R',
     'R', 'M', 'M', 'M', 'R', 'M', 'M', 'R', 'M', 'M', 'R', 'M', 'M',
       'M', 'R', 'M'], dtype=object)
```

In [72]:

```
Y_validation
```

Out[72]:

```
array(['M', 'M', 'R', 'R', 'M', 'R', 'M', 'M', 'R', 'R', 'R', 'M', 'M',
       'R', 'M', 'M', 'R', 'M', 'M', 'R', 'M', 'R', 'M', 'M', 'M', 'R',
 'R', 'M', 'M', 'M', 'R', 'M', 'M', 'R', 'M', 'M', 'R', 'M', 'M',
 'M', 'R', 'M'], dtype=object)
```

In [ ]:

NAME:P.LAKSHMI NARASIMHA SHATHAMANYU

COLLEGE:G.PULLAIAH COLLEGE OF TECHNOLOGY AND TECHNOLOGY

YEAR:3rd YEAR

ACADEMIC YEAR:2021-2023

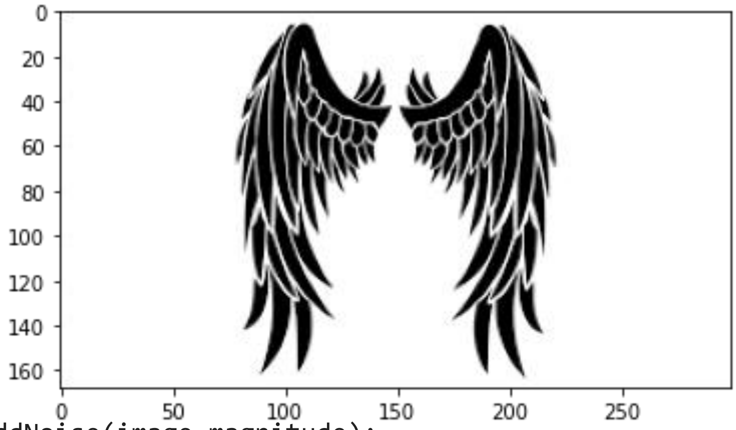
MAJOR-2

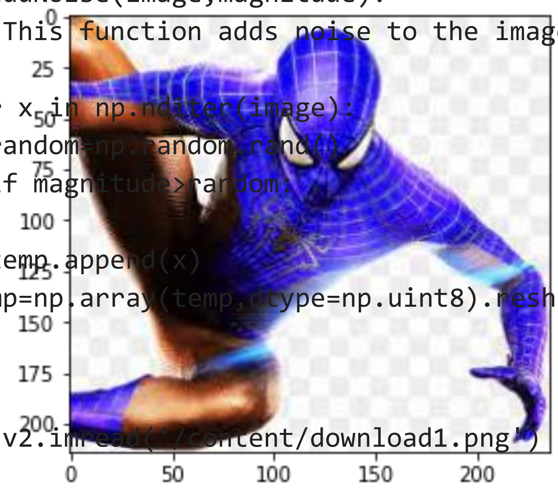
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import cv2 import matplotlib.pyplot as plt
#from PIL import Image
# Input data files are available in the "../input/" directory.
```

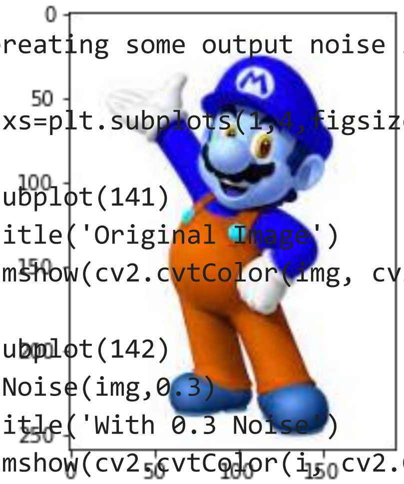
```
pip install opencv-python
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-
package
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-package
```

```
img1 = cv2.imread('/content/download1.png')
plt.imshow(img1) plt.show() img2 =
cv2.imread('/content/download2.jpg')
plt.imshow(img2) plt.show() img3 =
cv2.imread('/content/download 3.jpg')
plt.imshow(img3) plt.show()
```

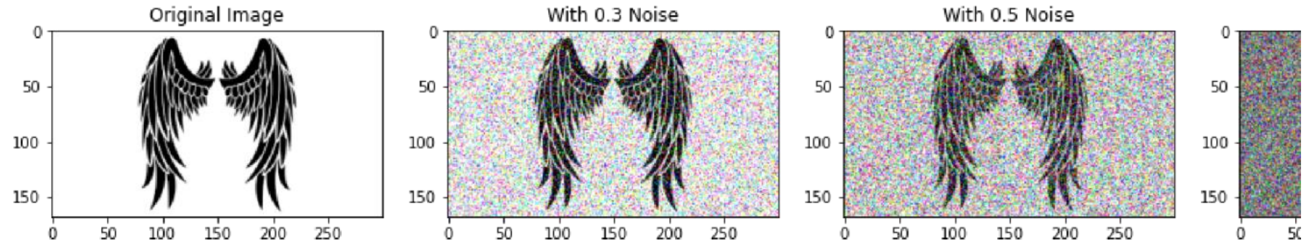






```
def addNoise(image,magnitude):
  '''This function adds noise to the image'''

  for x in np.nditer(image):
    random=np.random.rand()
    if magnitude>random:

    temp.append(x)
  temp=np.array(temp,dtype=np.uint8).reshape(image.shape[0],image.shape[1],image.shape[2])


img=cv2.imread('/content/download1.png')


#Genereating some output noise images

fig,axs=plt.subplots(1,4,figsize=(18,15))

plt.subplot(141)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

plt.subplot(142)
i=addNoise(img,0.3)
plt.title('With 0.3 Noise')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))
  temp=[]



      x=np.random.randint(255) # If magnitude is greater than random numeber then add nois


  return temp
```

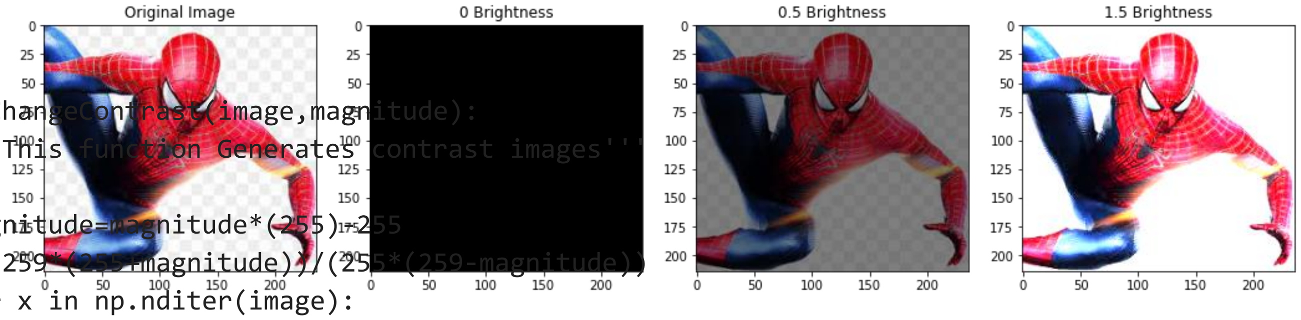
```
plt.subplot(143) i=addNoise(img,0.5)
plt.title('With 0.5 Noise')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))

plt.subplot(144) i=addNoise(img,1)
plt.title('With 1.0 Noise')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))
plt.show()
```



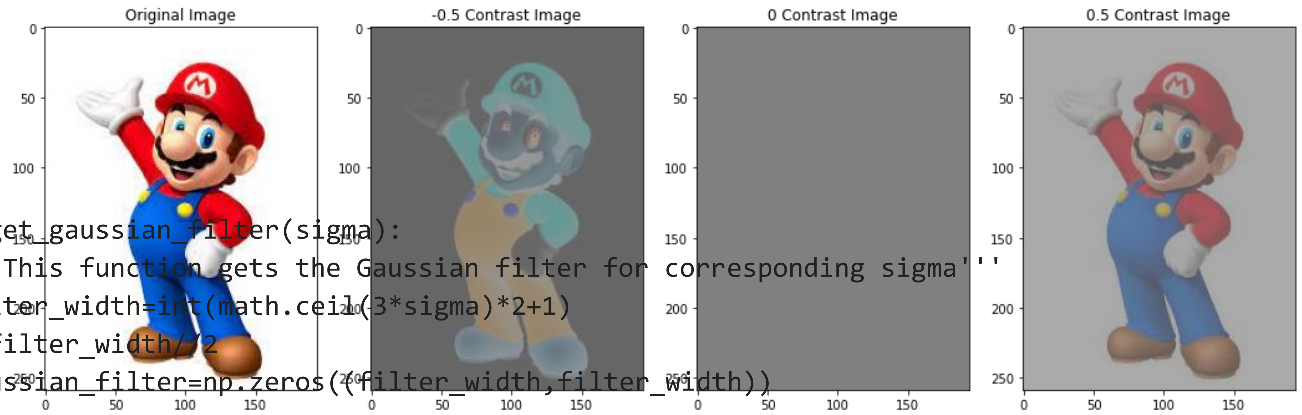

```
def Brighten(image,magnitude):
  '''This funtion Brightesns the given input images'''
temp=[]   for x in np.nditer(image):
    x=x*magnitude
if x>255:
x=255
temp.append(x)
  temp=np.array(temp,dtype=np.uint8).reshape(image.shape[0],image.shape[1],image.shape[2])
return temp


img=cv2.imread('/content/download2.jpg')
```

```
fig,axs=plt.subplots(1,4,figsize=(18,15))

plt.subplot(141) plt.title('Original Image')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

plt.subplot(142) i=Brighten(img,0)
plt.title('0 Brightness')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))

plt.subplot(143) i=Brighten(img,0.5)
plt.title('0.5 Brightness')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))

plt.subplot(144) i=Brighten(img,1.5)
plt.title('1.5 Brightness')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))
plt.show()
```

```
  temp=
```

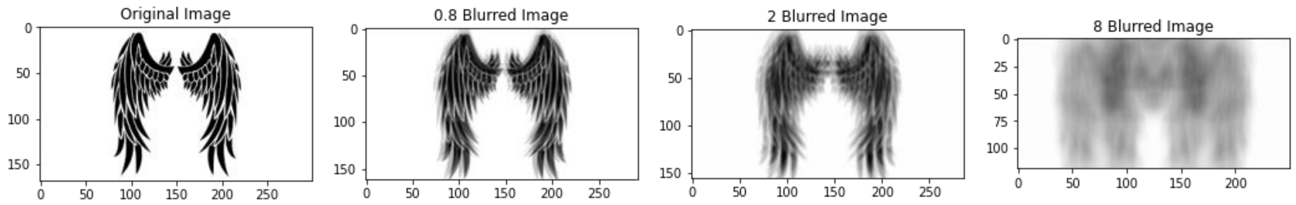



```
def ChangeContrast(image,magnitude):
  '''This function Generates contrast images'''

  magnitude=magnitude*(255)-255
  f=(259*(255+magnitude))/(255*(259-magnitude))
  for x in np.nditer(image):
[]
    x=f*(x-128)+128     temp.append(x)
temp=np.array(temp,dtype=np.uint8).reshape(image.shape[0],image.shape[1],image.shape[2])
return temp


img=cv2.imread('/content/download 3.jpg')

fig,axs=plt.subplots(1,4,figsize=(18,15))

plt.subplot(141) plt.title('Original Image')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))


i=ChangeContrast(img,-0.5) plt.subplot(142)
plt.title('-0.5 Contrast Image')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))

i=ChangeContrast(img,0) plt.subplot(143)
plt.title('0 Contrast Image')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))

i=ChangeContrast(img,0.5) plt.subplot(144)
plt.title('0.5 Contrast Image')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))


plt.show()
```




```
def get_gaussian_filter(sigma):
  '''This function gets the Gaussian filter for corresponding sigma'''
  filter_width=int(math.ceil(3*sigma)*2+1)
  x=filter_width//2
  gaussian_filter=np.zeros((filter_width,filter_width))
for i in range(-x,x+1):      for j in range(-x,x+1):
      numerator=math.exp(-1*(i**2+j**2)/(2*(sigma**2)))
denominator=2*math.pi*(sigma**2)
gaussian_filter[i][j]=numerator/denominator   return
gaussian_filter
```

```
import math def
Blur(image,magnitude):
  '''This function blurs the images'''   sigma=magnitude
gaussian_filter=get_gaussian_filter(sigma)   blurred_image= np.zeros((image.shape[0]-
gaussian_filter.shape[0], image.shape[1]-gaussia   for x in range(image.shape[0]-
gaussian_filter.shape[0]):     for y in range(image.shape[1]-gaussian_filter.shape[1]):
for z in range(image.shape[2]):
        blurred_image[x,y,z]=(gaussian_filter * image[x: x+gaussian_filter.shape[0], y: y+
blurred_image=np.array(blurred_image,dtype=np.uint8)   return blurred_image


img=cv2.imread('/content/download1.png')

fig,axs=plt.subplots(1,4,figsize=(18,15))

plt.subplot(141) plt.title('Original Image')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

i=Blur(img,0.8) plt.subplot(142)
plt.title('0.8 Blurred Image')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))

i=Blur(img,2) plt.subplot(143) plt.title('2
Blurred Image') plt.imshow(cv2.cvtColor(i,
cv2.COLOR_BGR2RGB))

i=Blur(img,8) plt.subplot(144) plt.title('8
Blurred Image') plt.imshow(cv2.cvtColor(i,
cv2.COLOR_BGR2RGB)) plt.show()
```




```
def Sharpen(image):
  '''This Function Sharpens the Images'''   #sigma=magnitude
sharpen_filter=np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])   image_padded =
np.zeros((image.shape[0] + 2, image.shape[1] + 2,image.shape[2]))
image_padded[1:-1, 1:-1,:] = image
  sharpened_image= np.zeros((image.shape[0], image.shape[1],image.shape[2]))
for x in range(image.shape[0]):     for y in range(image.shape[1]):
for z in range(image.shape[2]):
        sharpened_image[x,y,z]=(sharpen_filter * image_padded[x: x+3, y:y+3,z]).sum()
sharpened_image=np.array(sharpened_image,dtype=np.uint8)   return sharpened_image


img=cv2.imread('/content/download2.jpg')

fig,axs=plt.subplots(1,2,figsize=(8,6))
```

```
plt.subplot(121) plt.title('Original
Image')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))


i=Sharpen(img) plt.subplot(122)
plt.title('Sharpened Image')
plt.imshow(cv2.cvtColor(i, cv2.COLOR_BGR2RGB))
plt.show()
```