Template Descriptive Statistics

1. Printing the shape of the data

```
shape of data

[ ] print(dataset.shape)

(58509, 49)
```

2. Printing the first 20 rows of the data

```
first 20 rows of data

print(dataset.head(20))

               0             1             2  ...      46      47  48
0  -3.014600e-07  8.260300e-06 -1.151700e-05  ... -1.4996 -1.4996   1
1   2.913200e-06 -5.247700e-06  3.342100e-06  ... -1.5005 -1.5005   1
2  -2.951700e-06 -3.184000e-06 -1.592000e-05  ... -1.4985 -1.4985   1
3  -1.322600e-06  8.820100e-06 -1.587900e-05  ... -1.4975 -1.4976   1
4  -6.836600e-08  5.666300e-07 -2.590600e-05  ... -1.4959 -1.4959   1
5  -9.584900e-07  5.214300e-08 -4.735900e-05  ... -1.4972 -1.4973   1
```

3. Printing the descriptive statistics

```
[ ] print(dataset.describe())

                  0             1  ...            47            48
count  58509.000000  5.850900e+04  ...  58509.000000  58509.000000
mean      -0.000003  1.439648e-06  ...     -1.497686      6.000000
std        0.000072  5.555429e-05  ...      0.003175      3.162305
min       -0.013721 -5.414400e-03  ...     -1.521300      1.000000
25%       -0.000007 -1.444400e-05  ...     -1.499500      3.000000
50%       -0.000003  8.804600e-07  ...     -1.498000      6.000000
75%        0.000002  1.877700e-05  ...     -1.496200      9.000000
max        0.005784  4.525300e-03  ...     -1.337100     11.000000

[8 rows x 49 columns]
```

4. Printing the class distribution

```
class distribution

[ ] print(dataset.groupby(48).size())

48
1     5319
2     5319
3     5319
4     5319
5     5319
6     5319
7     5319
8     5319
9     5319
10    5319
11    5319
dtype: int64
```

5. Printing the data types of the features. All features are of type float. However, the class is of type int.

```
print(dataset.dtypes)

0     float64
1     float64
2     float64
3     float64
4     float64
5     float64
6     float64
7     float64
```

6. Printing the Pearson Correlation..

data correlation

```
[ ] print(dataset.corr(method='pearson'))
```

```
           0         1         2  ...        46        47        48
0   1.000000  0.574463  0.268946  ... -0.203724 -0.205733 -0.017853
1   0.574463  1.000000  0.188993  ... -0.089415 -0.089916 -0.098850
2   0.268946  0.188993  1.000000  ... -0.061003 -0.061243 -0.030237
3   0.272271  0.261633  0.069828  ... -0.075473 -0.077045  0.005692
4   0.098113  0.198154  0.076619  ... -0.071136 -0.070909 -0.110669
5   0.099363  0.112018  0.040039  ... -0.036910 -0.036684 -0.035783
6   0.051068  0.038799 -0.014772  ... -0.000942 -0.001107 -0.407439
7   0.050196  0.037278 -0.015060  ... -0.000805 -0.000970 -0.407312
8   0.048455  0.036054 -0.021511  ... -0.000412 -0.000575 -0.407069
9   0.024870  0.039779 -0.006834  ... -0.024485 -0.023500 -0.340275
```

7. Printing the skewness of the dataset.

```
[ ] print(dataset.skew())
```

```
0      -139.597413
1       -12.209919
2        -3.231163
3      -175.778935
4       -62.548594
5        -2.436558
6        -1.468722
7        -1.468106
8        -1.466851
9        -1.017130
10       -1.016869
11       -1.016463
12       46.450351
13       17.391509
14        4.643801
15       32.396501
16       21.652524
17        2.261594
```

8. Histograms of all features

histograms for all features

```
[ ] pylab.rcParams['figure.figsize'] = (10, 20)
    dataset.hist(color='cyan', layout=(16, 5))
    plt.tight_layout()
    plt.show()
```

9. Scatter Matrix

```
# We choose only four features and the class for visualization purpose
# (The  features chosen are 0, 1, 2, 3)
pylab.rcParams['figure.figsize'] = (12, 12)

scatter_matrix(dataset[[0, 1, 2, 3, 48]])
plt.tight_layout()
plt.show()
```

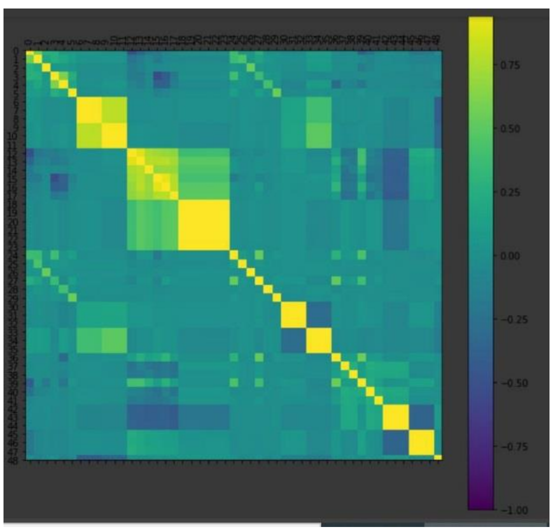
10. Density plot for each feature.

```
pylab.rcParams['figure.figsize'] = (10, 25)
dataset.plot(kind='density', subplots=True, layout=(16,4), sharex=False, sharey=False)
plt.tight_layout()
plt.show()
```

11. Correlation Matrix Plot

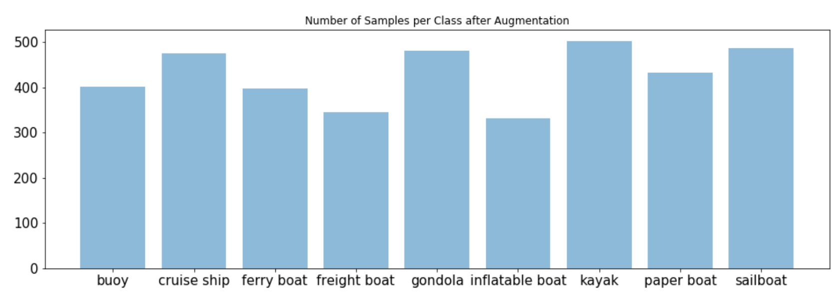
```
correlation matrix plot

[ ] pylab.rcParams['figure.figsize'] = (10, 10)
    correlations = dataset.corr()
    # plot correlation matrix
    fig = plt.figure()
    ax = fig.add_subplot(111)
    cax = ax.matshow(correlations, vmin=-1, vmax=1)
    fig.colorbar(cax)
    ticks = np.arange(0,len(dataset.columns),1)
    ax.set_xticks(ticks)
    ax.set_yticks(ticks)
    ax.set_xticklabels(dataset.columns, rotation=90)
    ax.set_yticklabels(dataset.columns)
    plt.show()
    correlations = dataset.corr()
```



12. Classification: split the dataset into train, validate and test (70%-20%-10%). Show the number of three sets in the way shown below. I need the total number of dataset patterns to be presented in the way shown below. The same should be done for the training, validation and test sets. **The same should go into the code of 3b) and 3c).**

```
Total Number of Samples:  3852
Number of Samples per Class after Augmentation:
buoy : 402
cruise ship : 475
ferry boat : 398
freight boat : 345
gondola : 480
inflatable boat : 331
kayak : 502
paper boat : 432
sailboat : 487
```



13. Spot Check Algorithms

spot-check algorithms

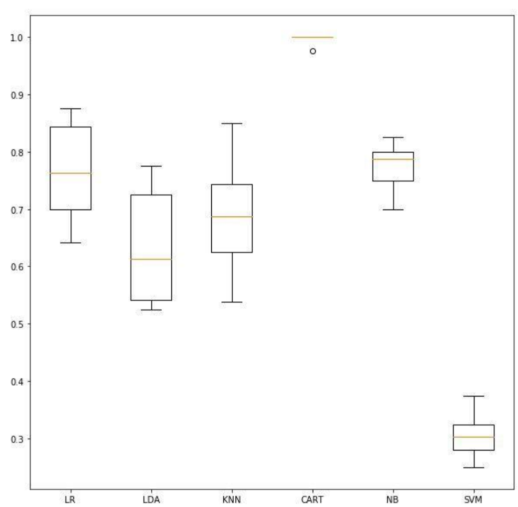
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.547353 (0.032326)
LDA: 0.857073 (0.005235)
KNN: 0.115944 (0.002750)
CART: 0.982951 (0.002077)
NB: 0.765185 (0.012104)
SVM: 0.257098 (0.003425)
```

14. Compare Algorithms

compare algorithms

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



clearly, the best performing model is CART

15.

make predictions on validation dataset using CART model

```
cart = DecisionTreeClassifier()
cart.fit(X_train, Y_train)
predictions = cart.predict(X_validation)
print('accuracy:', accuracy_score(Y_validation, predictions))
print('confusion matrix:\n', confusion_matrix(Y_validation, predictions))
print('classification report:\n', classification_report(Y_validation, predictions))
```

```
accuracy: 0.99
confusion matrix:
 [[34  0  0  0  0  0  0]
 [ 0 13  0  0  0  0  0]
 [ 0  0 19  0  0  0  0]
 [ 0  0  0 19  0  0  0]
 [ 0  0  0  0  8  0  0]
 [ 0  0  0  0  0  5  0]
 [ 0  0  0  0  0  1  1]]
classification report:
```

In the above confusion matrix, the x axis shows the true labels and the y axis shows the predicted class.

16.

```
classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        34
         1.0       1.00      1.00      1.00        13
         2.0       1.00      1.00      1.00        19
         3.0       1.00      1.00      1.00        19
         4.0       1.00      1.00      1.00         8
         5.0       0.83      1.00      0.91         5
         6.0       1.00      0.50      0.67         2

    accuracy                           0.99       100
   macro avg       0.98      0.93      0.94       100
weighted avg       0.99      0.99      0.99       100
```