```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the data
data = pd.read_csv('/content/segmentation.csv')
data.head()
```

	REGION-CENTROID-COL	REGION-CENTROID-ROW	REGION-PIXEL-COUNT	SHORT-LINE-DENSITY-5	SHORT-LINE-DENSITY-2	VEDGE-MEAN	VEDGE-SD	HEDGE-MEAN	HE
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	BRICKFACE	140.0	125.0	9.0	0.0	0.0	0.277778	0.062963	0.66
2	BRICKFACE	188.0	133.0	9.0	0.0	0.0	0.333333	0.266667	0.50
3	BRICKFACE	105.0	139.0	9.0	0.0	0.0	0.277778	0.107407	0.83
4	BRICKFACE	34.0	137.0	9.0	0.0	0.0	0.500000	0.166667	1.11

```
data = data.dropna()
```

```
data.head()
```

	REGION-CENTROID-COL	REGION-CENTROID-ROW	REGION-PIXEL-COUNT	SHORT-LINE-DENSITY-5	SHORT-LINE-DENSITY-2	VEDGE-MEAN	VEDGE-SD	HEDGE-MEAN	HE
1	BRICKFACE	140.0	125.0	9.0	0.0	0.0	0.277778	0.062963	0.66
2	BRICKFACE	188.0	133.0	9.0	0.0	0.0	0.333333	0.266667	0.50
3	BRICKFACE	105.0	139.0	9.0	0.0	0.0	0.277778	0.107407	0.83
4	BRICKFACE	34.0	137.0	9.0	0.0	0.0	0.500000	0.166667	1.11
5	BRICKFACE	39.0	111.0	9.0	0.0	0.0	0.722222	0.374074	0.88

```
# Get the dimensions of the dataset
rows, columns = data.shape
print("Number of rows:", rows)
print("Number of columns:", columns)
```

```
Number of rows: 210
Number of columns: 19
```

```
data.isna().sum()
```

```
REGION-CENTROID-COL     0
REGION-CENTROID-ROW     0
REGION-PIXEL-COUNT      0
SHORT-LINE-DENSITY-5    0
SHORT-LINE-DENSITY-2    0
VEDGE-MEAN              0
VEDGE-SD                0
HEDGE-MEAN              0
HEDGE-SD                0
INTENSITY-MEAN          0
RAWRED-MEAN             0
RAWBLUE-MEAN            0
RAWGREEN-MEAN           0
EXRED-MEAN              0
EXBLUE-MEAN             0
EXGREEN-MEAN            0
VALUE-MEAN              0
SATURATION-MEAN         0
HUE-MEAN                0
dtype: int64
```

```
# Check data types of columns
print(data.dtypes)
```

```
REGION-CENTROID-COL      object
REGION-CENTROID-ROW     float64
REGION-PIXEL-COUNT      float64
SHORT-LINE-DENSITY-5    float64
SHORT-LINE-DENSITY-2    float64
VEDGE-MEAN              float64
VEDGE-SD                float64
HEDGE-MEAN              float64
HEDGE-SD                float64
INTENSITY-MEAN          float64
RAWRED-MEAN             float64
RAWBLUE-MEAN            float64
RAWGREEN-MEAN           float64
EXRED-MEAN              float64
EXBLUE-MEAN             float64
EXGREEN-MEAN            float64
VALUE-MEAN              float64
SATURATION-MEAN         float64
HUE-MEAN                float64
dtype: object
```

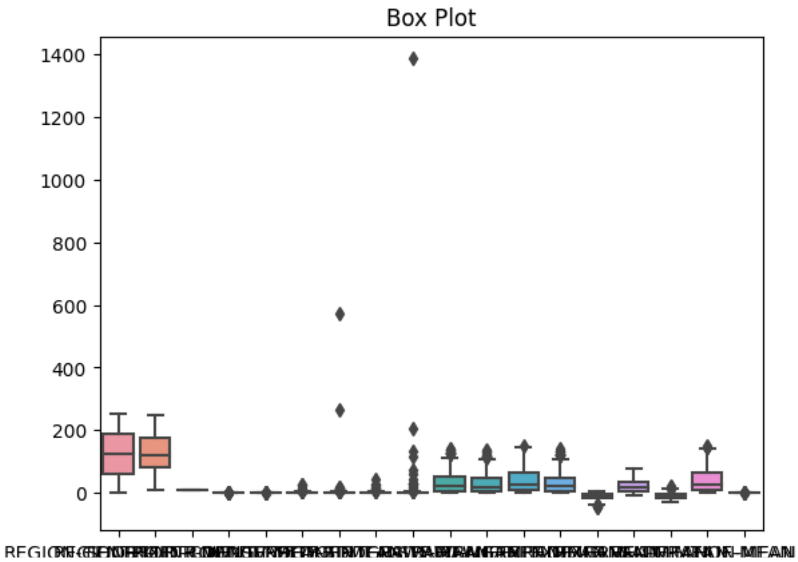
```
data['REGION-CENTROID-COL'].value_counts()
```

```
BRICKFACE    30
SKY          30
FOLIAGE      30
CEMENT       30
WINDOW       30
PATH         30
GRASS        30
Name: REGION-CENTROID-COL, dtype: int64
```

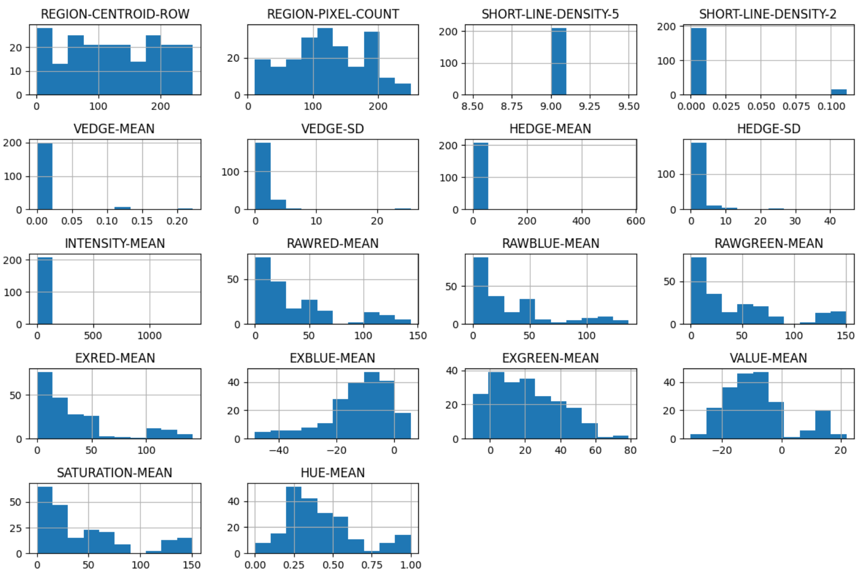
```
# Display summary statistics of numerical features
print(data.describe())
```

```
       REGION-CENTROID-ROW  REGION-PIXEL-COUNT  SHORT-LINE-DENSITY-5  \
count           210.000000          210.000000                 210.0
mean            124.647619          122.757143                   9.0
std              74.104024           58.139686                   0.0
min               1.000000           11.000000                   9.0
25%              60.500000           81.500000                   9.0
50%             123.500000          121.500000                   9.0
75%             189.750000          174.500000                   9.0
max             252.000000          250.000000                   9.0

       SHORT-LINE-DENSITY-2  VEDGE-MEAN    VEDGE-SD  HEDGE-MEAN    HEDGE-SD  \
count            210.000000  210.000000  210.000000  210.000000  210.000000
mean               0.008466    0.006349    1.925132    5.719529    2.604233
std                0.029549    0.030077    3.158211   43.495942    4.798268
min                0.000000    0.000000    0.000000    0.000000    0.000000
25%                0.000000    0.000000    0.666667    0.400921    0.777779
50%                0.000000    0.000000    1.222222    0.828695    1.388889
75%                0.000000    0.000000    1.888890    1.676634    2.597221
max                0.111111    0.222222   25.500000  572.996400   44.722225

       INTENSITY-MEAN  RAWRED-MEAN  RAWBLUE-MEAN  RAWGREEN-MEAN  EXRED-MEAN  \
count      210.000000   210.000000    210.000000     210.000000  210.000000
mean        11.638377    37.091005     32.967725      44.011112   34.294180
std         97.390023    38.677168     35.540563      43.804447   37.057003
min          0.000000     0.000000      0.000000       0.000000    0.000000
25%          0.410816     6.453704      7.000000       8.277778    3.805555
50%          0.913176    21.314816     18.611112      26.833334   20.000000
75%          1.980485    52.629629     46.750000      64.194447   46.472223
max       1386.329200   143.444440    136.888890     150.888890  142.555560

       EXBLUE-MEAN  EXGREEN-MEAN  VALUE-MEAN  SATURATION-MEAN    HUE-MEAN
count   210.000000    210.000000  210.000000       210.000000  210.000000
mean    -12.369841     20.760317   -8.390476        44.888360    0.423230
std      11.559599     18.761842   11.003746        43.235182    0.227333
min     -48.222220     -9.666667  -30.555555         0.000000    0.000000
25%     -18.111110      4.111111  -15.750000        10.527778    0.275722
50%     -10.333333     19.555556   -9.888889        28.388890    0.365455
75%      -4.666666     34.333332   -3.722222        64.194447    0.539738
max       5.777778     78.777780   21.888890       150.888890    1.000000
```

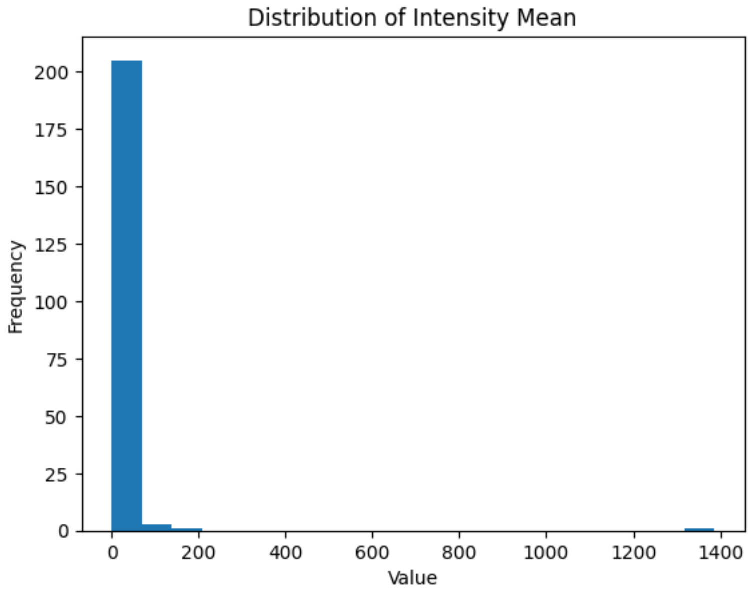
```
import seaborn as sns

# Box plot to identify outliers
sns.boxplot(data=data)
plt.title('Box Plot')
plt.show()
```




```
import matplotlib.pyplot as plt

# Plot histograms of numerical features
data.hist(figsize=(12, 8))
plt.tight_layout()
plt.show()
```



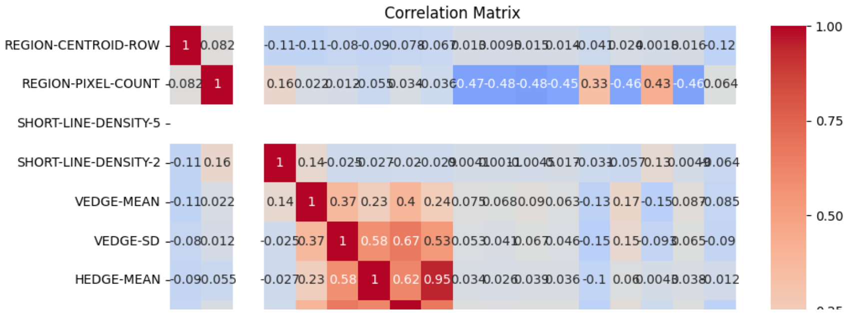

```
import matplotlib.pyplot as plt

# Visualize the distribution of a numerical feature
plt.hist(data['INTENSITY-MEAN'], bins=20)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Distribution of Intensity Mean')
plt.show()
```



Coorelation Analysis

```
import seaborn as sns

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
<ipython-input-23-d0ac522ad2a5>:4: FutureWarning: The default value of numeric_only i
  correlation_matrix = data.corr()
```



Feature Selection

```
import pandas as pd
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression


# Separate features and target variable
X = data.drop('REGION-CENTROID-COL',axis=1)
y = data['REGION-CENTROID-COL']

# Perform feature selection using RFE with cross-validation
estimator = LogisticRegression()
selector = RFECV(estimator, cv=5)
X_new = selector.fit_transform(X, y)

# Get the selected feature indices
selected_indices = selector.get_support(indices=True)
selected_features = X.columns[selected_indices]
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Feature Extraction - PCA

```python
import pandas as pd
from sklearn.decomposition import PCA


# Perform feature extraction using PCA
n_components = 10  # Number of principal components to extract
pca = PCA(n_components=n_components)
X_new = pca.fit_transform(X)

# Convert the extracted components back to a DataFrame
columns = [f'PC{i}' for i in range(1, n_components+1)]
X_new = pd.DataFrame(X_new, columns=columns)

# Concatenate the extracted components with the target variable
data_extracted = pd.concat([X_new, y], axis=1)

# Check the updated data
print(data_extracted.head())
```

```
         PC1        PC2        PC3        PC4       PC5       PC6        PC7  \
0 -29.274454  65.331223  10.816026  28.905154  3.670107 -1.775233  -9.163849
1 -34.553339  66.570403  59.104599  26.750537  4.514670 -0.895009 -11.648498
2 -27.242806  71.904926 -22.348493  12.331846  3.861868 -2.124479 -10.340216
3 -20.557326  73.414263 -92.807196   6.380089  5.007876 -2.953277 -11.233634
4 -18.192801  62.360860 -90.475917  30.265299  3.411658 -2.906218  -7.425729

        PC8       PC9      PC10 REGION-CENTROID-COL
0 -0.937699 -0.421450 -0.142443                 NaN
1 -0.782419 -0.298807  0.352214           BRICKFACE
2 -0.968352 -0.493137 -0.116605           BRICKFACE
3 -0.752698 -0.353380  0.766267           BRICKFACE
4 -0.867824 -0.109318  0.032178           BRICKFACE
```

Feature Scaling - Standardization

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler


# Perform standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert the scaled data back to a DataFrame
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Check the scaled data
print(X_scaled.head())
```

```
   REGION-CENTROID-ROW  REGION-PIXEL-COUNT  SHORT-LINE-DENSITY-5  \
0             0.207668            0.038669                   0.0
1             0.856954            0.176598                   0.0
2            -0.265769            0.280044                   0.0
3            -1.226171            0.245562                   0.0
4            -1.158537           -0.202706                   0.0

   SHORT-LINE-DENSITY-2  VEDGE-MEAN  VEDGE-SD  HEDGE-MEAN  HEDGE-SD  \
0             -0.287183   -0.211604 -0.522856   -0.130359 -0.404770
1             -0.287183   -0.211604 -0.505223   -0.125664 -0.439588
2             -0.287183   -0.211604 -0.522856   -0.129335 -0.369952
3             -0.287183   -0.211604 -0.452325   -0.127969 -0.311923
4             -0.287183   -0.211604 -0.381793   -0.123189 -0.358347

   INTENSITY-MEAN  RAWRED-MEAN  RAWBLUE-MEAN  RAWGREEN-MEAN  EXRED-MEAN  \
0       -0.116586    -0.800981     -0.722995      -0.831680   -0.831478
1       -0.118988    -0.788502     -0.694791      -0.829138   -0.822461
2       -0.114413    -0.802901     -0.716727      -0.841851   -0.831478
```

```
3       -0.114909    -0.809620     -0.710460      -0.859649   -0.837489
4       -0.115366    -0.804820     -0.732396      -0.831680   -0.834483

   EXBLUE-MEAN  EXGREEN-MEAN  VALUE-MEAN  SATURATION-MEAN  HUE-MEAN
0     1.371334     -0.871709    0.045692        -0.860393  0.539724
1     1.506224     -0.931072    0.005205        -0.844937  0.508617
2     1.448414     -0.931072    0.065936        -0.865545  0.482371
3     1.573669     -1.014181    0.076057        -0.860393  0.663177
4     1.323159     -0.847963    0.055814        -0.857817  0.615934
```

```
from sklearn.model_selection import train_test_split

# Load the dataset
X = data.drop('REGION-CENTROID-COL',axis=1)
y = data['REGION-CENTROID-COL']

# Split the data into training and testing sets
test_size = 0.2
random_state = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(168, 18)
(42, 18)
(168,)
(42,)
```

1. KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Create an instance of the KNeighborsClassifier
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)

# Fit the model to the training data
knn.fit(X_train, y_train)

# Predict the target variable for the test features
y_pred1 = knn.predict(X_test)

# Calculate the accuracy of the model
accuracy_knn = accuracy_score(y_test, y_pred1)
accuracy_knn
```

```
0.6904761904761905
```

```
print('Misclassifiaction samples=',(y_test!=y_pred1).sum())
```

```
Misclassifiaction samples= 13
```

```
from sklearn.metrics import classification_report
cr_knn = classification_report(y_test,y_pred1)
print(cr_knn)
```

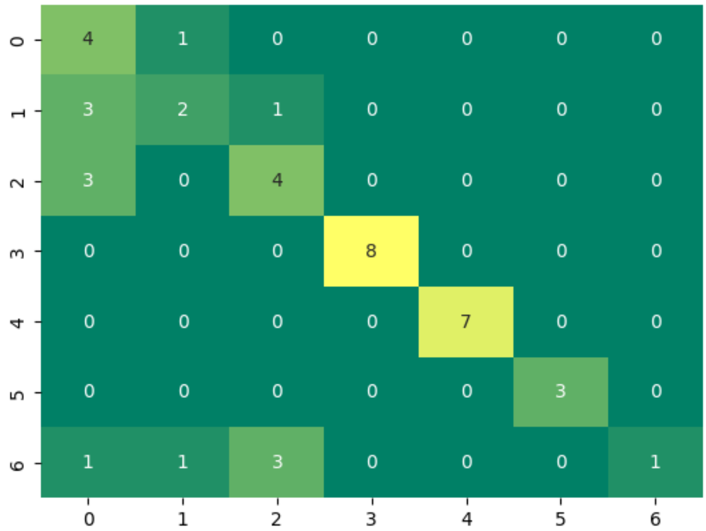
```
              precision    recall  f1-score   support

   BRICKFACE       0.36      0.80      0.50         5
      CEMENT       0.50      0.33      0.40         6
     FOLIAGE       0.50      0.57      0.53         7
       GRASS       1.00      1.00      1.00         8
        PATH       1.00      1.00      1.00         7
         SKY       1.00      1.00      1.00         3
      WINDOW       1.00      0.17      0.29         6

    accuracy                           0.69        42
   macro avg       0.77      0.70      0.67        42
weighted avg       0.77      0.69      0.67        42
```

```
from sklearn.metrics import confusion_matrix
cm_knn = confusion_matrix(y_test,y_pred1)
print(cm_knn)
```

```
[[4 1 0 0 0 0 0]
 [3 2 1 0 0 0 0]
 [3 0 4 0 0 0 0]
 [0 0 0 8 0 0 0]
 [0 0 0 0 7 0 0]
 [0 0 0 0 0 3 0]
 [1 1 3 0 0 0 1]]
```

```
sns.heatmap(cm_knn,annot=True,cbar=False,cmap='summer')
```

```
<Axes: >
```



2. SVM

```
from sklearn import svm
from sklearn.metrics import accuracy_score

# Create an instance of the SVM classifier
svm_classifier = svm.SVC(kernel='linear')

# Fit the model to the training data
svm_classifier.fit(X_train, y_train)

# Predict the target variable for the test features
y_pred2 = svm_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy_svm = accuracy_score(y_test, y_pred2)
accuracy_svm
```

```
0.8571428571428571
```

```
print('Misclassifiaction samples=',(y_test!=y_pred2).sum())
```

```
Misclassifiaction samples= 6
```

```
from sklearn.metrics import classification_report
cr_svm = classification_report(y_test,y_pred2)
print(cr_svm)
```

```
              precision    recall  f1-score   support

   BRICKFACE       1.00      1.00      1.00         5
      CEMENT       0.67      0.67      0.67         6
     FOLIAGE       0.86      0.86      0.86         7
       GRASS       1.00      1.00      1.00         8
        PATH       0.78      1.00      0.88         7
         SKY       1.00      1.00      1.00         3
      WINDOW       0.75      0.50      0.60         6

    accuracy                           0.86        42
   macro avg       0.86      0.86      0.86        42
```

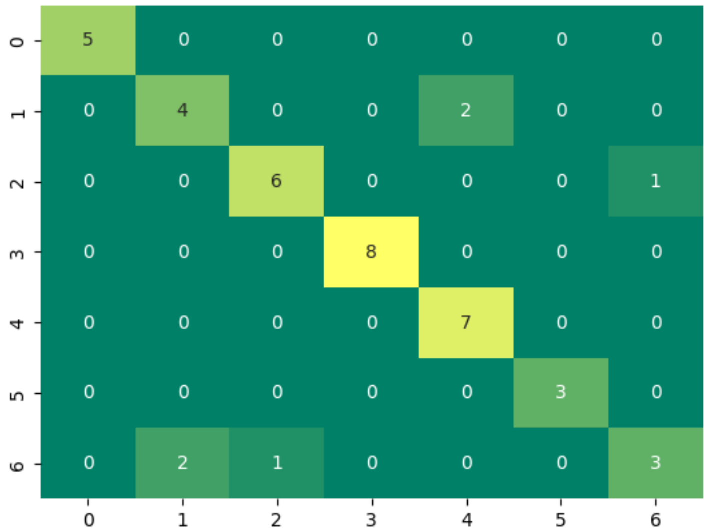
```
    weighted avg       0.86      0.86      0.85        42
```

```
from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_test,y_pred2)
print(cm_svm)
```

```
    [[5 0 0 0 0 0 0]
     [0 4 0 0 2 0 0]
     [0 0 6 0 0 0 1]
     [0 0 0 8 0 0 0]
     [0 0 0 0 7 0 0]
     [0 0 0 0 0 3 0]
     [0 2 1 0 0 0 3]]
```

```
sns.heatmap(cm_svm,annot=True,cbar=False,cmap='summer')
```

```
    <Axes: >
```



3. Naive Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Create an instance of the Gaussian Naive Bayes classifier
naive_bayes_classifier = GaussianNB()

# Fit the model to the training data
naive_bayes_classifier.fit(X_train, y_train)

# Predict the target variable for the test features
y_pred3 = naive_bayes_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy_naive = accuracy_score(y_test, y_pred3)
accuracy_naive
```

```
    0.6428571428571429
```

```
print('Misclassifiaction samples=',(y_test!=y_pred3).sum())
```

```
    Misclassifiaction samples= 15
```

```
from sklearn.metrics import classification_report
cr_naive = classification_report(y_test,y_pred3)
print(cr_naive)
```

```
                  precision    recall  f1-score   support

       BRICKFACE       0.50      0.40      0.44         5
          CEMENT       0.75      0.50      0.60         6
         FOLIAGE       0.33      0.14      0.20         7
           GRASS       1.00      1.00      1.00         8
            PATH       1.00      1.00      1.00         7
             SKY       1.00      1.00      1.00         3
          WINDOW       0.23      0.50      0.32         6
```

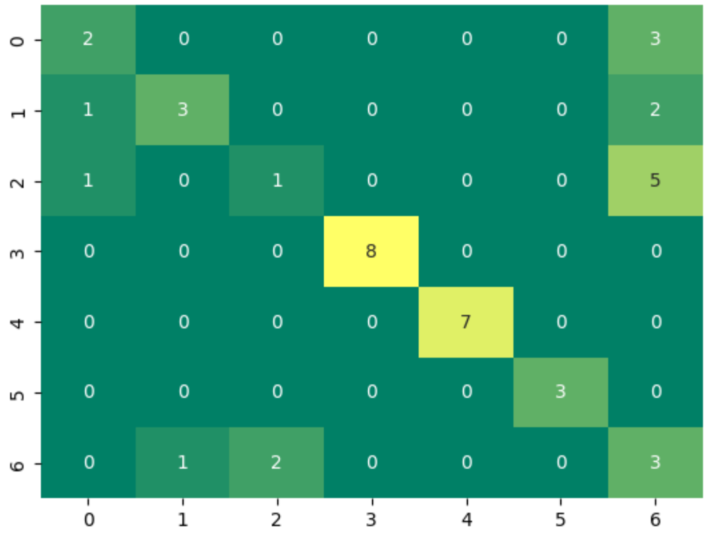
```
    accuracy                           0.64        42
   macro avg       0.69      0.65      0.65        42
weighted avg       0.68      0.64      0.65        42
```

```
from sklearn.metrics import confusion_matrix
cm_naive = confusion_matrix(y_test,y_pred3)
print(cm_naive)
```

```
[[2 0 0 0 0 0 3]
 [1 3 0 0 0 0 2]
 [1 0 1 0 0 0 5]
 [0 0 0 8 0 0 0]
 [0 0 0 0 7 0 0]
 [0 0 0 0 0 3 0]
 [0 1 2 0 0 0 3]]
```

```
sns.heatmap(cm_naive,annot=True,cbar=False,cmap='summer')
```

```
<Axes: >
```



4. Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Create an instance of the Random Forest classifier
random_forest_classifier = RandomForestClassifier(n_estimators=100, random_state=random_state)

# Fit the model to the training data
random_forest_classifier.fit(X_train, y_train)

# Predict the target variable for the test features
y_pred4 = random_forest_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy_random = accuracy_score(y_test, y_pred4)
accuracy_random
```

```
0.8571428571428571
```

```
print('Misclassifiaction samples=',(y_test!=y_pred4).sum())
```

```
Misclassifiaction samples= 6
```

```
from sklearn.metrics import classification_report
cr_random = classification_report(y_test,y_pred4)
print(cr_random)
```

```
              precision    recall  f1-score   support

   BRICKFACE       1.00      1.00      1.00         5
      CEMENT       0.75      0.50      0.60         6
     FOLIAGE       0.86      0.86      0.86         7
       GRASS       1.00      1.00      1.00         8
        PATH       0.78      1.00      0.88         7
         SKY       1.00      1.00      1.00         3
```

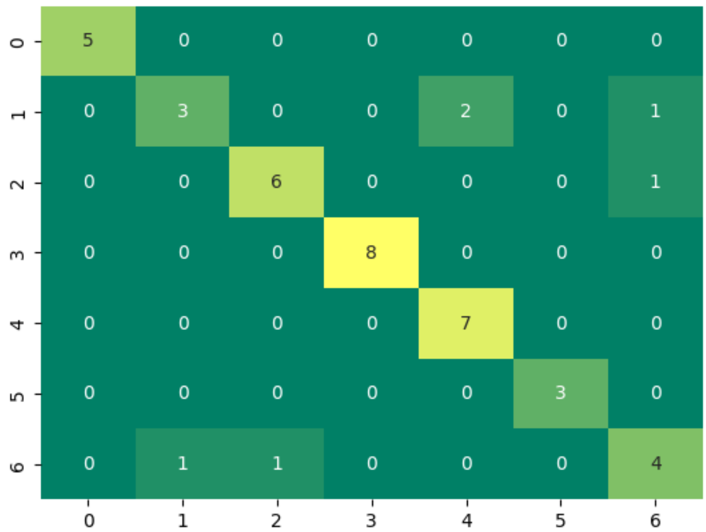
```
      WINDOW       0.67      0.67      0.67         6

    accuracy                           0.86        42
   macro avg       0.86      0.86      0.86        42
weighted avg       0.86      0.86      0.85        42
```

```
from sklearn.metrics import confusion_matrix
cm_random = confusion_matrix(y_test,y_pred4)
print(cm_random)
```

```
[[5 0 0 0 0 0 0]
 [0 3 0 0 2 0 1]
 [0 0 6 0 0 0 1]
 [0 0 0 8 0 0 0]
 [0 0 0 0 7 0 0]
 [0 0 0 0 0 3 0]
 [0 1 1 0 0 0 4]]
```

```
sns.heatmap(cm_random,annot=True,cbar=False,cmap='summer')
```

```
<Axes: >
```



5. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create an instance of the Logistic Regression classifier
logistic_regression_classifier = LogisticRegression(random_state=random_state)

# Fit the model to the training data
logistic_regression_classifier.fit(X_train, y_train)

# Predict the target variable for the test features
y_pred5 = logistic_regression_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy_lr = accuracy_score(y_test, y_pred5)
accuracy_lr
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.7857142857142857
```

```
print('Misclassifiaction samples=',(y_test!=y_pred5).sum())
```

```
Misclassifiaction samples= 9
```

```
from sklearn.metrics import classification_report
cr_lr = classification_report(y_test,y_pred5)
print(cr_lr)
```

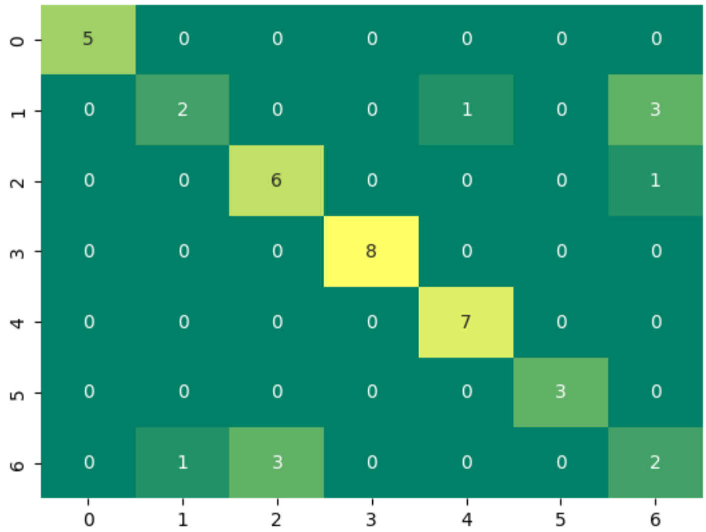
```
              precision    recall  f1-score   support

   BRICKFACE       1.00      1.00      1.00         5
      CEMENT       0.67      0.33      0.44         6
     FOLIAGE       0.67      0.86      0.75         7
       GRASS       1.00      1.00      1.00         8
        PATH       0.88      1.00      0.93         7
         SKY       1.00      1.00      1.00         3
      WINDOW       0.33      0.33      0.33         6

    accuracy                           0.79        42
   macro avg       0.79      0.79      0.78        42
weighted avg       0.78      0.79      0.77        42
```

```
from sklearn.metrics import confusion_matrix
cm_lr = confusion_matrix(y_test,y_pred5)
print(cm_lr)
```

```
[[5 0 0 0 0 0 0]
 [0 2 0 0 1 0 3]
 [0 0 6 0 0 0 1]
 [0 0 0 8 0 0 0]
 [0 0 0 0 7 0 0]
 [0 0 0 0 0 3 0]
 [0 1 3 0 0 0 2]]
```

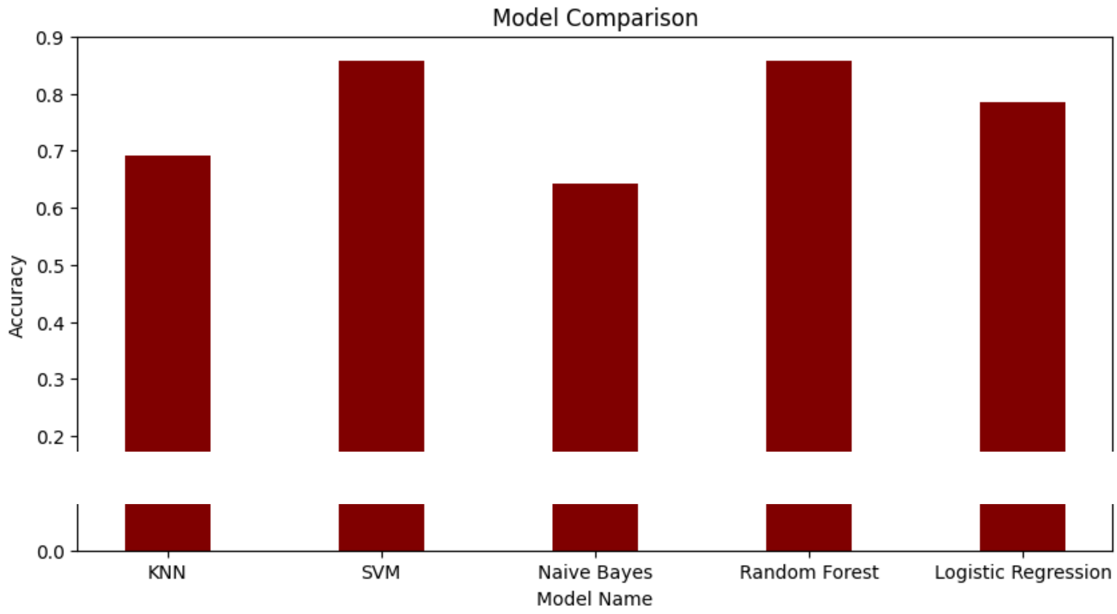
```
sns.heatmap(cm_lr,annot=True,cbar=False,cmap='summer')
```

<Axes: >



Model Comparison

```
acc = [accuracy_knn,accuracy_svm,accuracy_naive,accuracy_random,accuracy_lr]
model_name = ["KNN","SVM","Naive Bayes","Random Forest","Logistic Regression"]
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(model_name,acc, color ='maroon',
        width = 0.4)

plt.xlabel("Model Name")
plt.ylabel("Accuracy")
plt.title("Model Comparison")
plt.show()
```



✓ 0s completed at 17:05