Numerical dataset

General Project Details :

Dataset Name: House Sales in King County

Total number of samples: 21613

Sample used: 18372 Training Set, 3241 Test Set

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
[72] df1.describe().T
```

	count	mean	std	min	25%	50%	75%	max
price	21613.0	540088.141767	367127.196483	75000.0000	321950.000	450000.0000	645000.000	7.700000e+06
bedrooms	21613.0	3.370842	0.930062	0.0000	3.000	3.0000	4.000	3.300000e+01
bathrooms	21613.0	2.114757	0.770163	0.0000	1.750	2.2500	2.500	8.000000e+00
sqft_living	21613.0	2079.899736	918.440897	290.0000	1427.000	1910.0000	2550.000	1.354000e+04
sqft_lot	21613.0	15106.967566	41420.511515	520.0000	5040.000	7618.0000	10688.000	1.651359e+06
floors	21613.0	1.494309	0.539989	1.0000	1.000	1.5000	2.000	3.500000e+00
waterfront	21613.0	0.007542	0.086517	0.0000	0.000	0.0000	0.000	1.000000e+00
view	21613.0	0.234303	0.766318	0.0000	0.000	0.0000	0.000	4.000000e+00
condition	21613.0	3.409430	0.650743	1.0000	3.000	3.0000	4.000	5.000000e+00
grade	21613.0	7.656873	1.175459	1.0000	7.000	7.0000	8.000	1.300000e+01
sqft_above	21613.0	1788.390691	828.090978	290.0000	1190.000	1560.0000	2210.000	9.410000e+03
sqft_basement	21613.0	291.509045	442.575043	0.0000	0.000	0.0000	560.000	4.820000e+03
zipcode	21613.0	98077.939805	53.505026	98001.0000	98033.000	98065.0000	98118.000	9.819900e+04
lat	21613.0	47.560053	0.138564	47.1559	47.471	47.5718	47.678	4.777760e+01
long	21613.0	-122.213896	0.140828	-122.5190	-122.328	-122.2300	-122.125	-1.213150e+02
sqft_living15	21613.0	1986.552492	685.391304	399.0000	1490.000	1840.0000	2360.000	6.210000e+03
sqft_lot15	21613.0	12768.455652	27304.179631	651.0000	5100.000	7620.0000	10083.000	8.712000e+05
date_year	21613.0	2014.322954	0.467616	2014.0000	2014.000	2014.0000	2015.000	2.015000e+03
date_month	21613.0	6.574423	3.115308	1.0000	4.000	6.0000	9.000	1.200000e+01
date_day	21613.0	15.688197	8.635063	1.0000	8.000	16.0000	23.000	3.100000e+01

Feature Extraction:

Preprocessing: Removing NaN values, outliers, checking correlations, splitting data into train and test

Checking Correlation :

```python
# Find & Visualize Correlation with Target Feature!
correlation_matrix = df.corr()
correlation_with_target = correlation_matrix['price'].sort_values(ascending=False)

print(correlation_with_target)

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```

```
<ipython-input-73-52df4839d024>:4: FutureWarning: The default value of numeric_only in DataFra
  correlation_matrix = df.corr()
price            1.000000
sqft_living      0.702035
grade            0.667434
sqft_above       0.605567
sqft_living15    0.585379
bathrooms        0.525138
view             0.397293
sqft_basement    0.323816
bedrooms         0.308350
lat              0.307003
waterfront       0.266369
floors           0.256794
sqft_lot         0.089661
sqft_lot15       0.082447
condition        0.036362
long             0.021626
date_year        0.003576
date_month      -0.010081
date_day        -0.014670
id              -0.016762
zipcode         -0.053203
Name: price, dtype: float64
```

Cross Validation : [0.54507704, 0.53800088, 0.52531746, 0.53129612, 0.53125455, 0.53505903, 0.54944398, 0.56031074 0.55883981, 0.57868293]

10 Folds

Ratio 9 : 1

Model : Linear Regression

- **Hyperparameters : default**

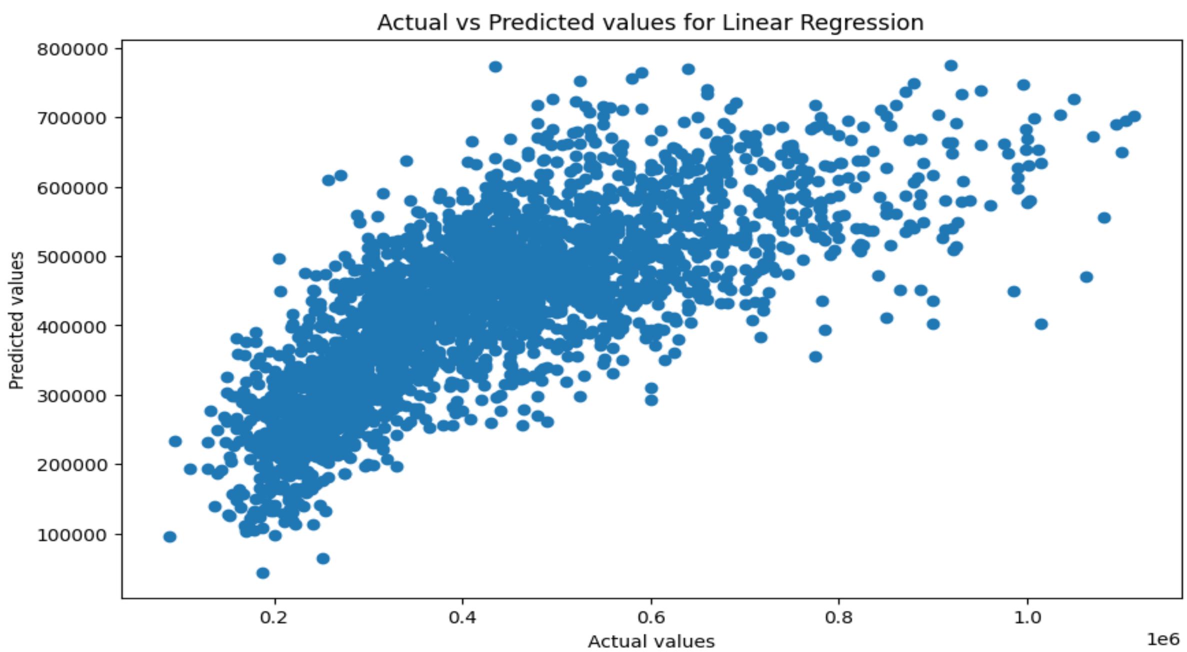
Results :

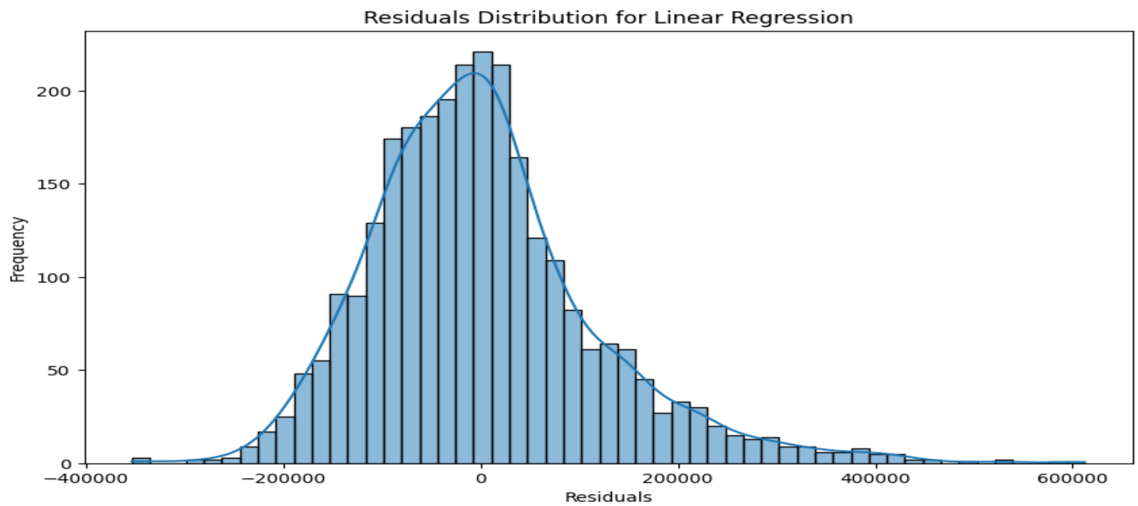
Mean Squared : 13314480617.695955

Mean : 115388.39030723998

Accuracy : 0.5640643958290714

Plotting :





Optimization using Standard Scaler for normalizing the dataset

Results :

Mean Squared : 13314480617.695396

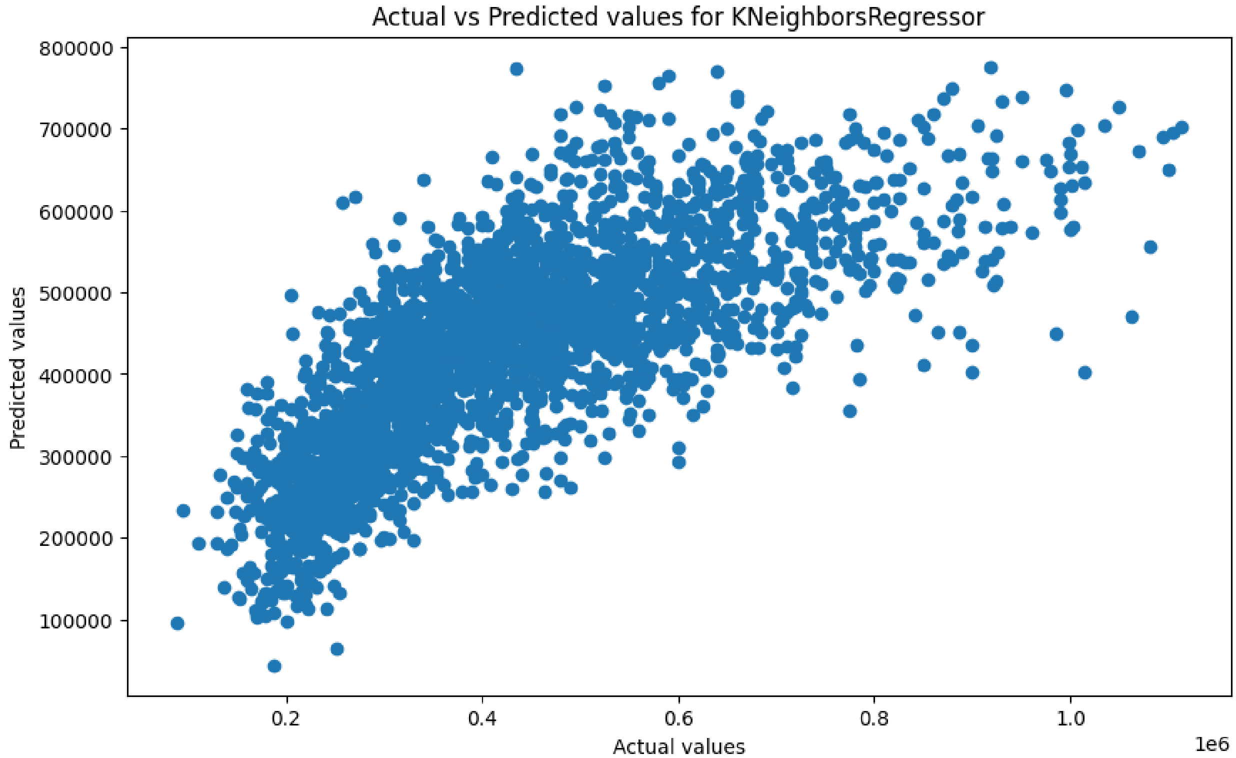
Mean : 115388.39030723757

Accuracy : 0.5640643958290896

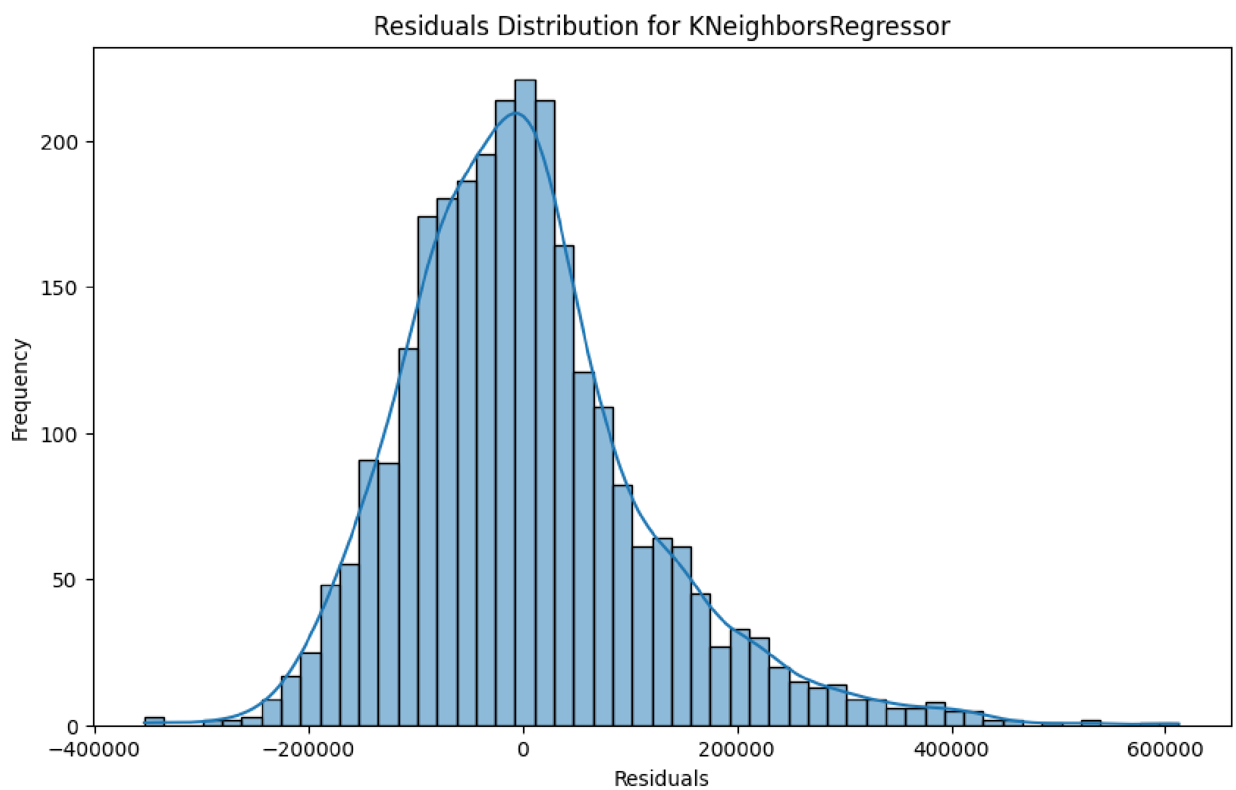
Model : KNeighborRegressor

- **hyperparameters : {n_neighbors : 9} determining number of points to get their mean and assign it to value**

Plotting :



Actual vs Predicted values for KNeighborsRegressor
Predicted values
800000
700000
600000
500000
400000
300000
200000
100000
0.2
0.4
0.6
0.8
1.0
1e6
Actual values



Residuals Distribution for KNeighborsRegressor
Frequency
200
150
100
50
0
−400000
−200000
0
200000
400000
600000
Residuals

Results :

Accuracy : 0.2539521073384233

Mean Squared : 13314480617.695955

Mean : 115388.39030723998

Optimization using Standard Scaler for normalizing the dataset

Results :

Accuracy : 0.731737208194416

Mean Squared : 13314480617.695396

Mean : 115388.39030723757

The Oxford-IIIT Pet
Image dataset

General Information on dataset:

About data set: -

This dataset contains 37 category pet dataset with roughly 200 images for each class. The images have a large variations in scale, pose and lighting. All images have an associated ground truth annotation of breed, head ROI, and pixel level trimap segmentation. It's species 1:Cat and 2:Dog, breed id 1-25:Cat 1-12:Dog, all images with 1st letter as captial are cat images and images with small first letter are dog images .

The population of the dataset is 7349 images, 2371 of images are cat type, 4978 of images are dog type.

Classes:

37 classes in dataset:

Labels : Abyssinian, Bengal, Birman, Bombay, British_Shorthair, Egyptian_Mau, Maine_Coon, Persian, Ragdoll, Russian_Blue, Siamese, Sphynx, american_bulldog, american_pit_bull_terrier, basset_hound, beagle, boxer, chihuahua, english_cocker_spaniel, english_setter, german_shorthaired, great_pyrenees, havanese, japanese_chin, keeshond, leonberger, miniature_pinscher, newfoundland, pomeranian, pug, saint_bernard, samoyed, scottish_terrier, shiba_inu, staffordshire_bull_terrier, wheaten_terrier, yorkshire_terrier,

5 classes used in this model:

0 => cats ['Bengal', 'Birman', 'British_Shorthair']

1 => dogs ['american_bulldog', 'american_pit_bull_terrier']

Labels:

Classes are encoded from string into 1 to 5.

Samples:

***Total number of samples:* 983**

***Samples used in training:* 688**

***Samples used in testing:* 295**

Implementation details:

Preprocessing: -

Feature Extraction: -

The number of features are extracted are 29520.

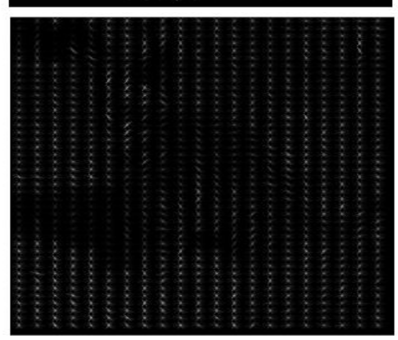
1. **Image resizing:**
   Images in dataset are resized into 600 X 300 px.

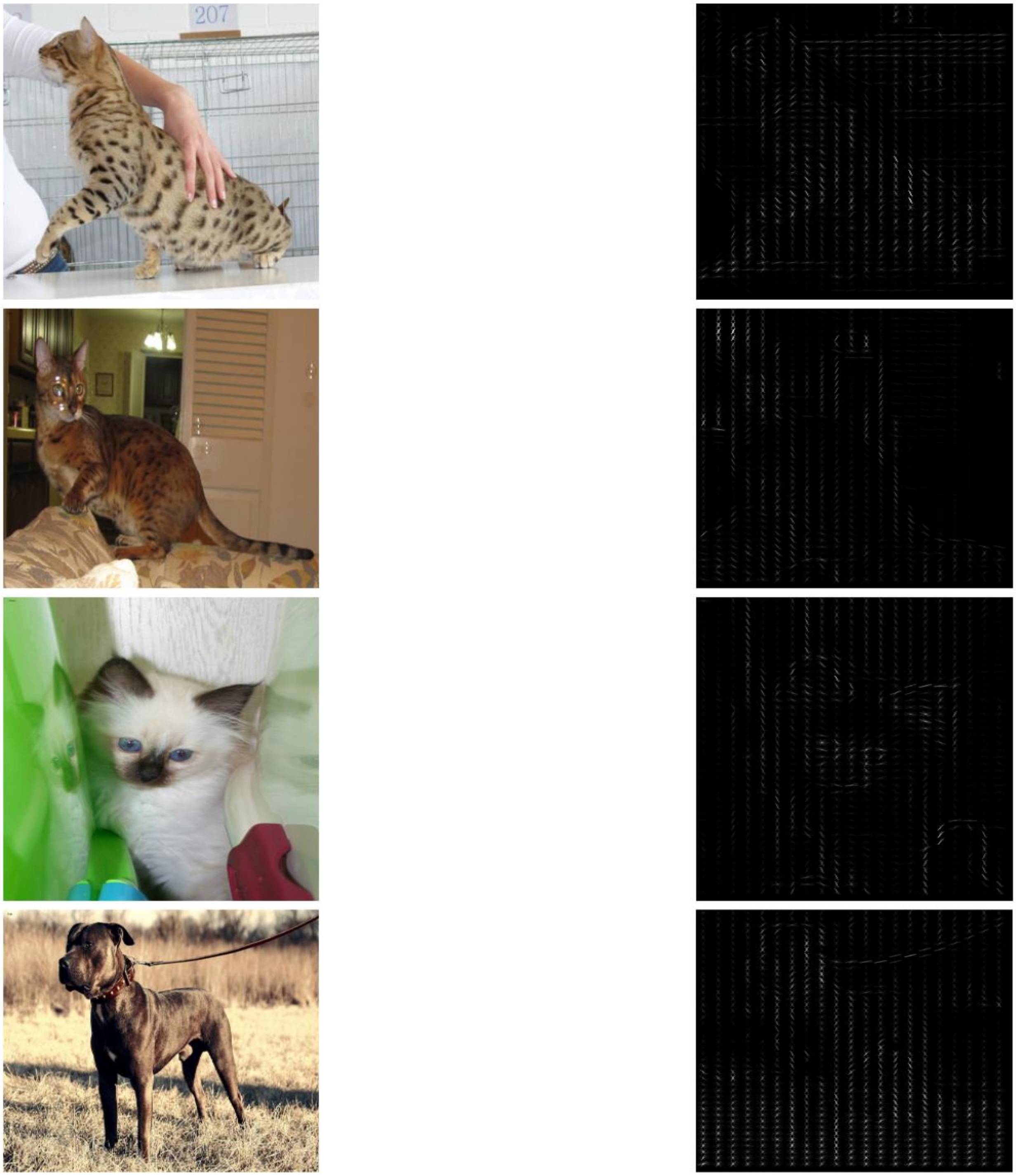
2. **Gray image:**

```
gray_image = color.rgb2gray(image)
```

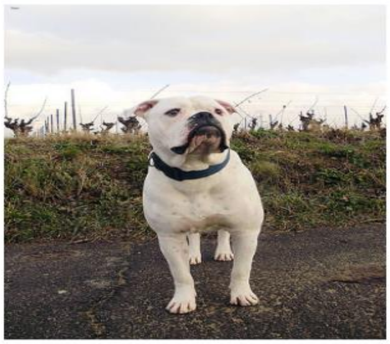
3. **HOG:**
   We use Histogram of Oriented Gradient (HOG) features to extract features from the image is a feature descriptor used in computer vision and image processing to detect objects or shapes within images. It works by capturing the distribution of local gradient orientations in an image.

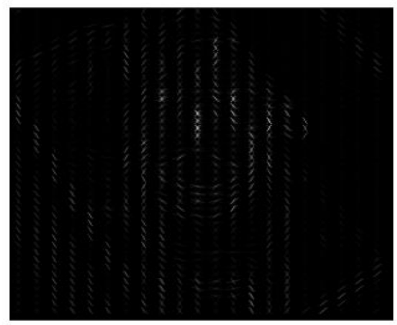
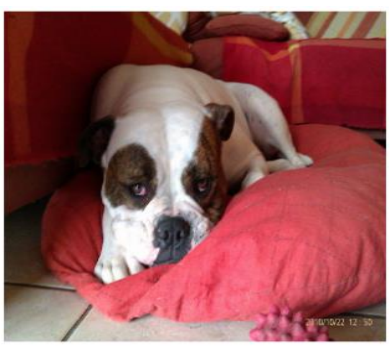
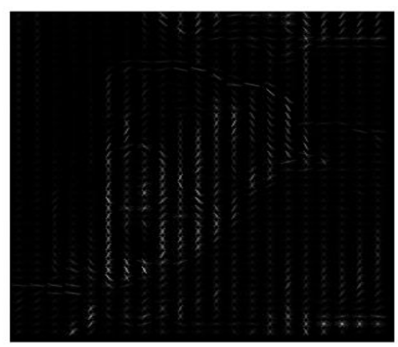
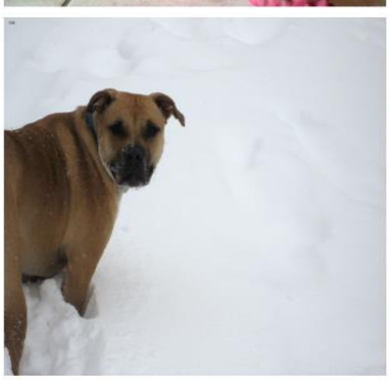
```
hog_features, hog_img = feature.hog(
        gray_image, pixels_per_cell=(14,14),
        cells_per_block=(2, 2),
        orientations=9,
        visualize=True,
        block_norm='L2-Hys')  # Specify block_norm
```




207






4. Convert lists into NumPy arrays:

We use NumPy arrays because it offer efficient data structures and operations for numerical computations compared to Python lists and to enable us to deal with machine learning frameworks and libraries like scikit-learn as it our main library in this model.

```
features_array = np.array(features_list)
labels_array = np.array(labels_list)
binary_labels_array = np.array(binary_labels_list)
```

```
[[0.2154987  0.14039081 0.15123429 ... 0.21621496 0.2571532  0.13358338]
 [0.02019346 0.04039182 0.02190496 ... 0.05417006 0.13913538 0.19306663]
 [0.14049355 0.19571326 0.04703247 ... 0.01730118 0.04609517 0.10060392]
 ...
 [0.3191164  0.04827603 0.02326313 ... 0.05616948 0.09562944 0.27668561]
 [0.11932652 0.07191888 0.09668749 ... 0.0781744  0.10543532 0.10495157]
 [0.3094895  0.12653894 0.03318428 ... 0.11372894 0.24982091 0.26087529]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 ...
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

5. Reshape features:

In this step we reshape the array from 1D to 2D to prepare it to train the model.

```
features_np = features_array.reshape(len(features_array), -1)
```

`(983, 29520)`

❖ Logistic regression: -

Logistic Regression is a fundamental and widely used statistical technique for binary classification. Despite its name containing "regression," it's actually a classification algorithm.

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000) # solver='lbfgs'
model_liblinear = LogisticRegression(solver='liblinear', max_iter=1000)

# Train 5 classes using 2 models
model_liblinear.fit(X_train, y_train)
model.fit(X_train, y_train)
```

Hyperparameters:

```
max_iter=1000
```

```
solver='liblinear'
```

❖ Kmeans:-

K-means is an unsupervised machine learning algorithm used for clustering. Its primary goal is to partition a dataset into 'K' distinct, non-overlapping clusters.

```
# Apply KMeans clustering on scaled HOG features
kmeans = KMeans(n_clusters=len(np.unique(y_train)), random_state=42)  # 
Adjust parameters as needed
kmeans.fit(hog_features_train_scaled)
```

Hyperparameters:

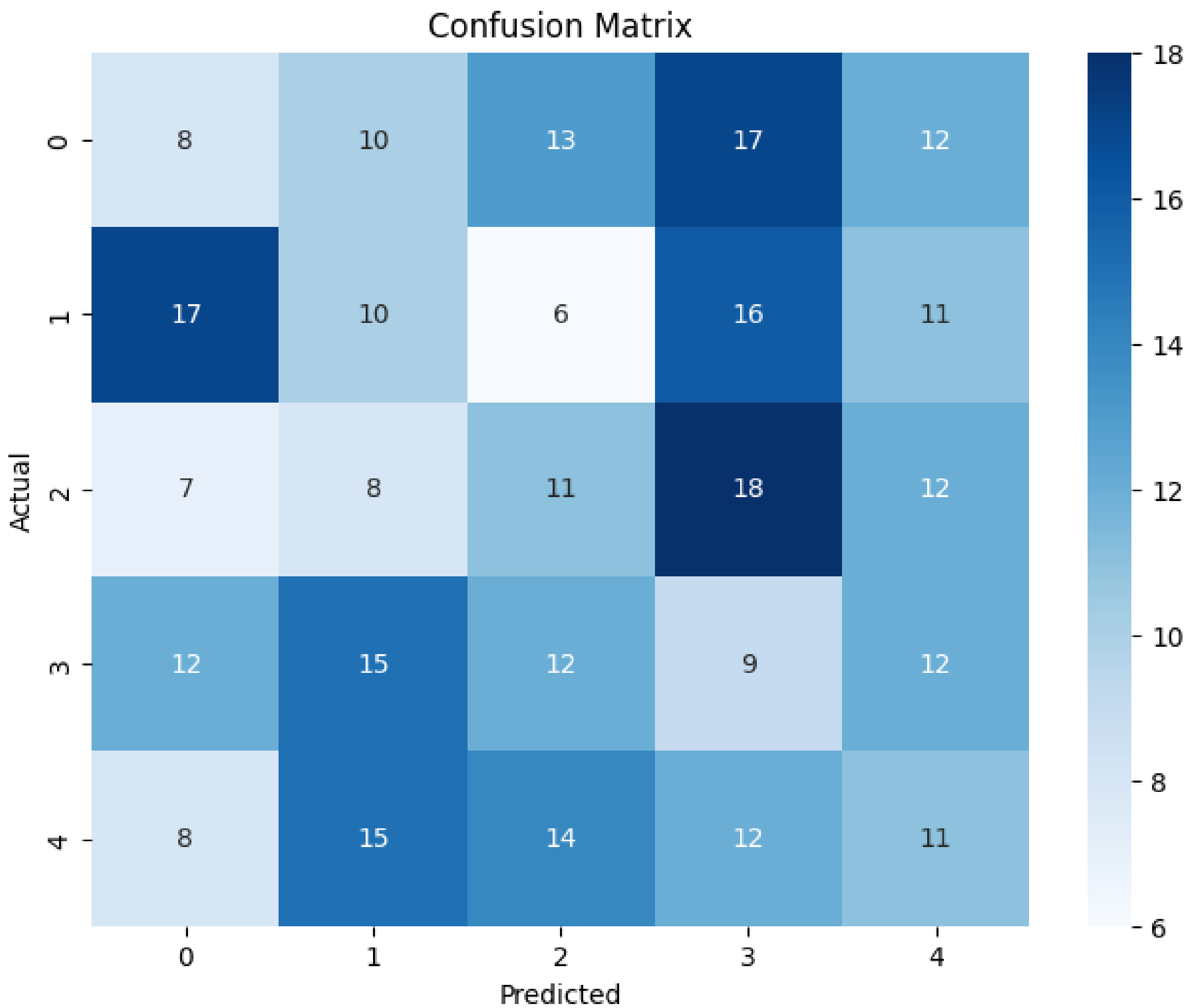
```
n_clusters=len(np.unique(y_train))
```

```
random_state=42
```

Results details:

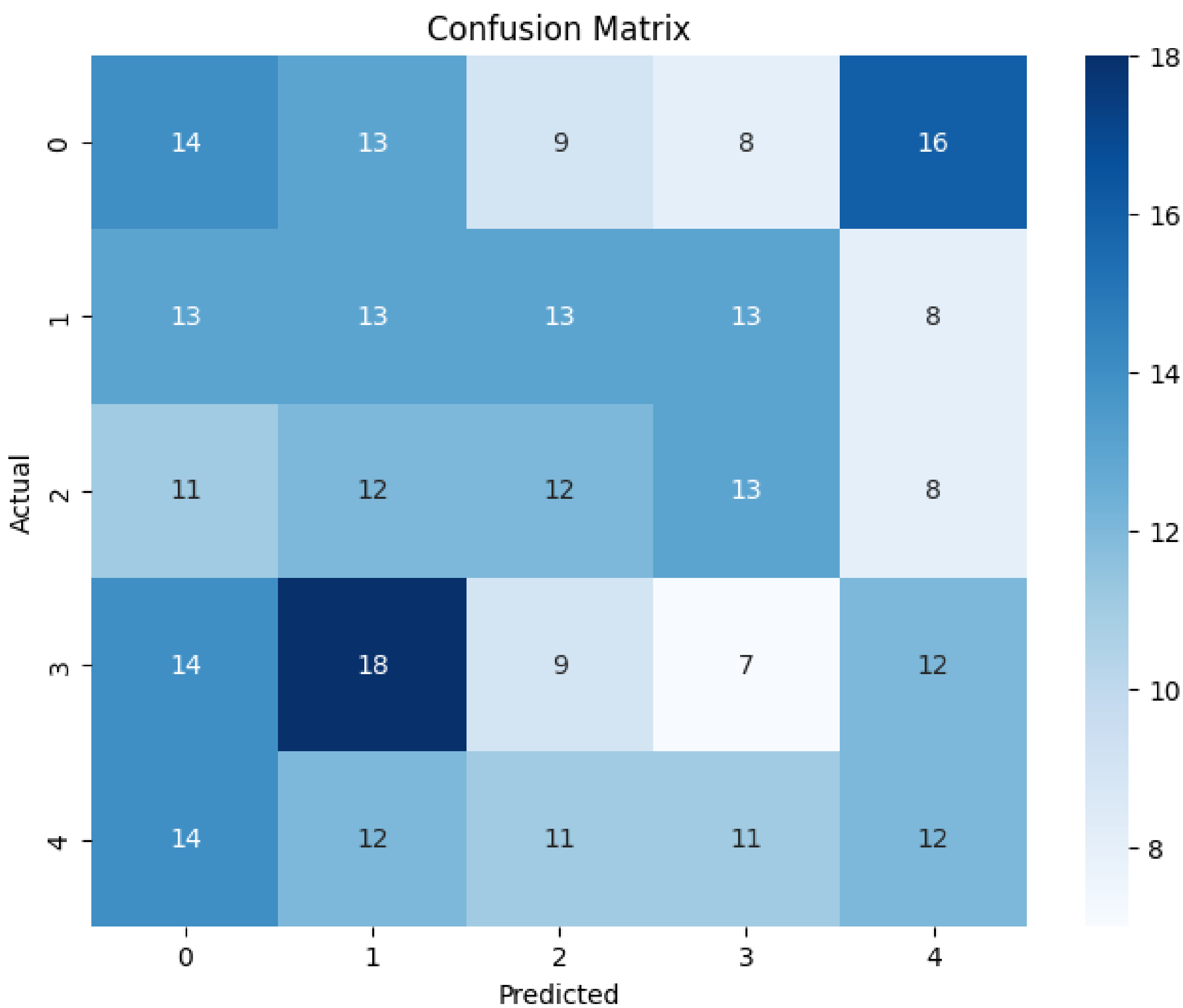
1 - Confusion matrix :

LogisticRegression before feature scaling




```
Accuracy Before Feature Scaling :  0.19594594594594594
```

LogisticRegression after feature scaling




```
Accuracy Before Feature Scaling :  0.6756756756756757
```

2- Classification report:

LogisticRegression before feature scaling

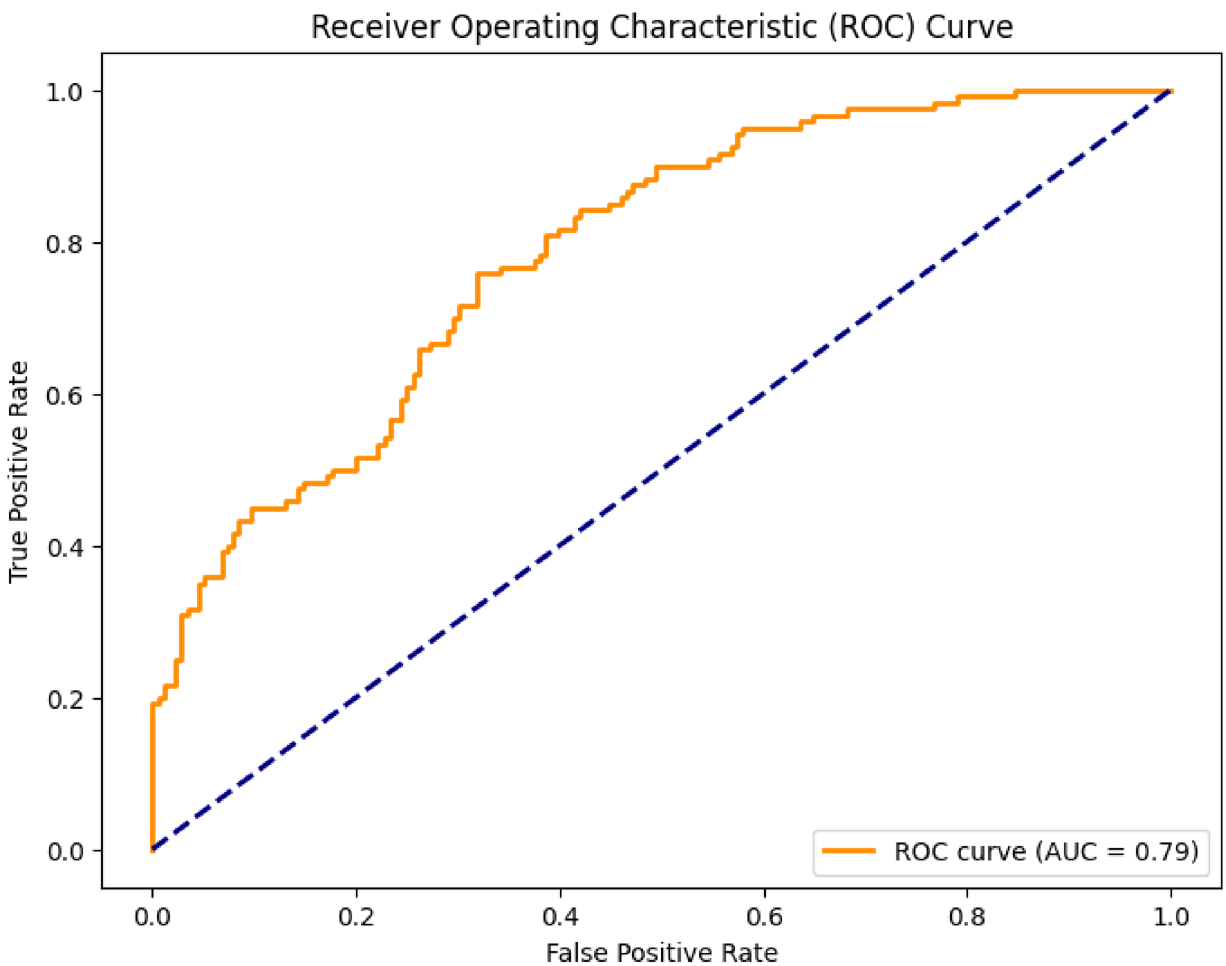
```
Classification Report:
              precision    recall  f1-score   support

           0       0.21      0.23      0.22        60
           1       0.19      0.22      0.20        60
           2       0.22      0.21      0.22        56
           3       0.13      0.12      0.12        60
           4       0.21      0.20      0.21        60

    accuracy                           0.20       296
   macro avg       0.19      0.20      0.20       296
weighted avg       0.19      0.20      0.19       296

Confusion Matrix Before Feature Scaling :
[[14 13  9  8 16]
 [13 13 13 13  8]
 [11 12 12 13  8]
 [14 18  9  7 12]
 [14 12 11 11 12]]
```

LogisticRegression after feature scaling

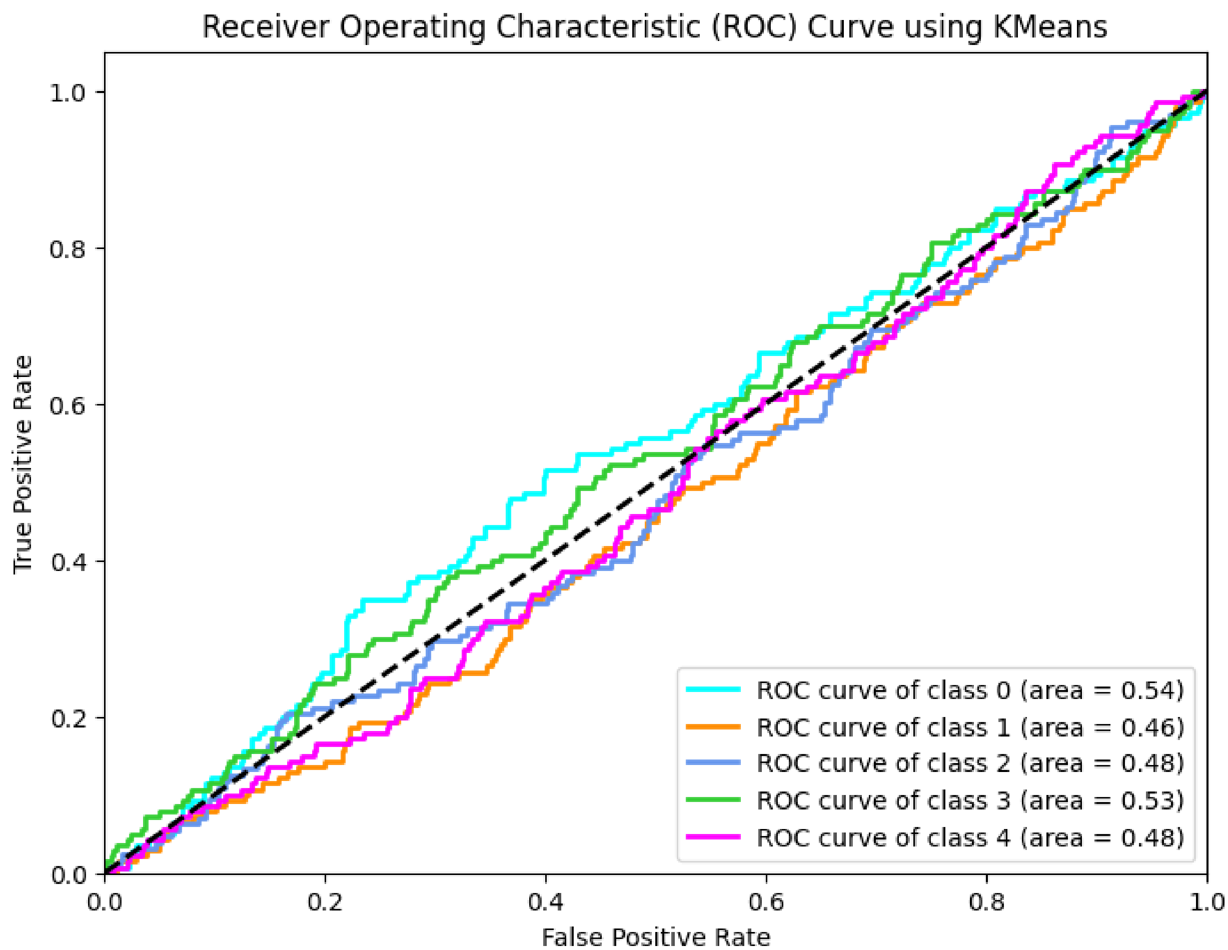
```
Classification Report:
              precision    recall  f1-score   support

           0       0.24      0.20      0.22        60
           1       0.20      0.23      0.22        60
           2       0.19      0.25      0.22        56
           3       0.17      0.13      0.15        60
           4       0.20      0.18      0.19        60

    accuracy                           0.20       296
   macro avg       0.20      0.20      0.20       296
weighted avg       0.20      0.20      0.20       296

Confusion Matrix liblinear Before Feature Scaling :
[[12 14 12  7 15]
 [10 14 18 12  6]
 [ 7 14 14 11 10]
 [10 15 13  8 14]
 [12 12 15 10 11]]
```

3- ROC:

Logistic Regression



Kmeans




```
Accuracy using KMeans as a classifier with feature scaling: 22.53%
```