Section C

1).EDA-

The data was read using the pandas library.

```
The number of images per class are as follows:
label
sitting               840
using_laptop          840
hugging               840
sleeping              840
drinking              840
clapping              840
dancing               840
cycling               840
calling               840
laughing              840
eating                840
fighting              840
listening_to_music    840
running               840
texting               840
Name: count, dtype: int64
```

```
#Average Image Size
average_width = sum(w for w, h in image_sizes) / len(image_sizes)
average_height = sum(h for w, h in image_sizes) / len(image_sizes)
print(f"Average Image Size:{round(average_height,2)} X {round(average_width,2)}")
```

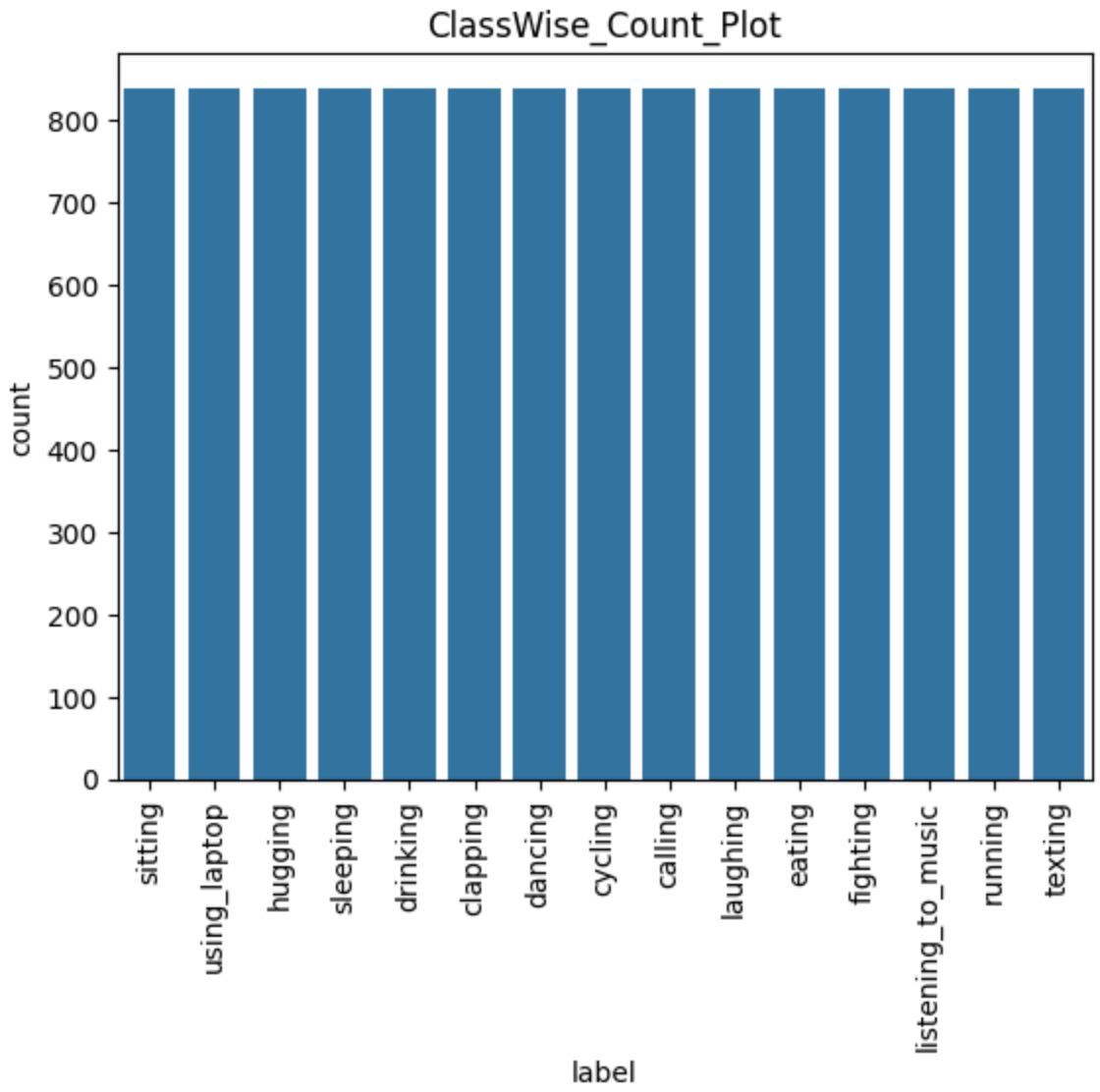
✓ 0.0s Python

```
Average Image Size:260.38 X 196.57
```

```
#relevent statistics
#EDA
#number of unique classes
unique_classes = []
for i in Y_df['label']:
    if i not in unique_classes:
        unique_classes.append(i)
print(f"The dataset contains {len(unique_classes)} unique classes")
```

✓ 0.0s Python

```
The dataset contains 15 unique classes
```



The plots have been made using seaborn and matplotlib direct implementations.

Class: sitting

Class: hugging

Class: sleeping

Class: drinking

Class: drinking

Class: clapping

.....

From the images it is clear that the sample images correctly represent the labels given in the csv file, each of these being clear representations of the said actions.

1).The Average Image Size is - 260.38 X 196.57

2).From the above it is clear that the data is well separated across classes with 840 samples per class(total - 12600 samples).This is also indicated in the count plot above.There is no bias towards a specific class.

3).The dataset contains a total of 15 unique classes indicating that the problem is a multi-class-classification problem.

2).Feature Extraction-

```
row_data = {
    'Corners': corners,
    'Edges': edges,
    'Contours': contours[0],  # We only need contour areas here
    'Hierarchy': hierarchy.flatten() if hierarchy is not None else np.array([np.nan]),
    'hogFeatures': hogfeatures,
    'Keypoints': keypoints,  # Include keypoints
    'ORBfeatures': des,  # Include ORB descriptors
    'ColorHistogram': color_histogram,
    'Label': label_dict[file]
}
```

The following features were first extracted and models were trained on each of these features to find out the 3 - best performing features.

```
row_data = {
    'Edges': edges,
    'hogFeatures': hogfeatures,
    'ColorHistogram': color_histogram,
    'LBP': lbp_features,
    'Gabor': gabor_features,
    'Label': label_dict[file]
}
```

Out of this LBP and Gabor features performed well as well,thus these were chosen for the other 2 - features.

All in all the extraction was narrowed down to 5 features namely - Edges , Hog , color histogram , LBP and Gabor.

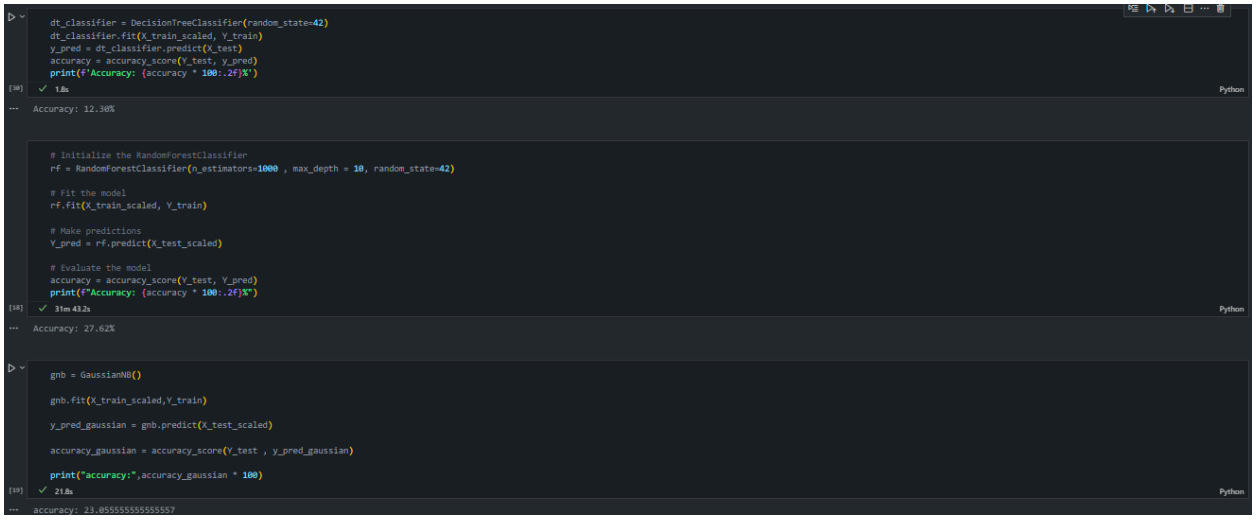
Algorithms used -
1).Edges - Canny Edge Detection , mainly for finding defining edges.
2).Hog - CV2 hog , for finding color gradients in the images.
3).Color Histogram - CV2 Color Histogram , for finding statistical summary of colors present in the image.
4).LBP - skimage LBP, For capturing local texture information.
5).GABOR - skimage , for edge detection and texture analysis enhancing the features detected by Canny Edge Detection.

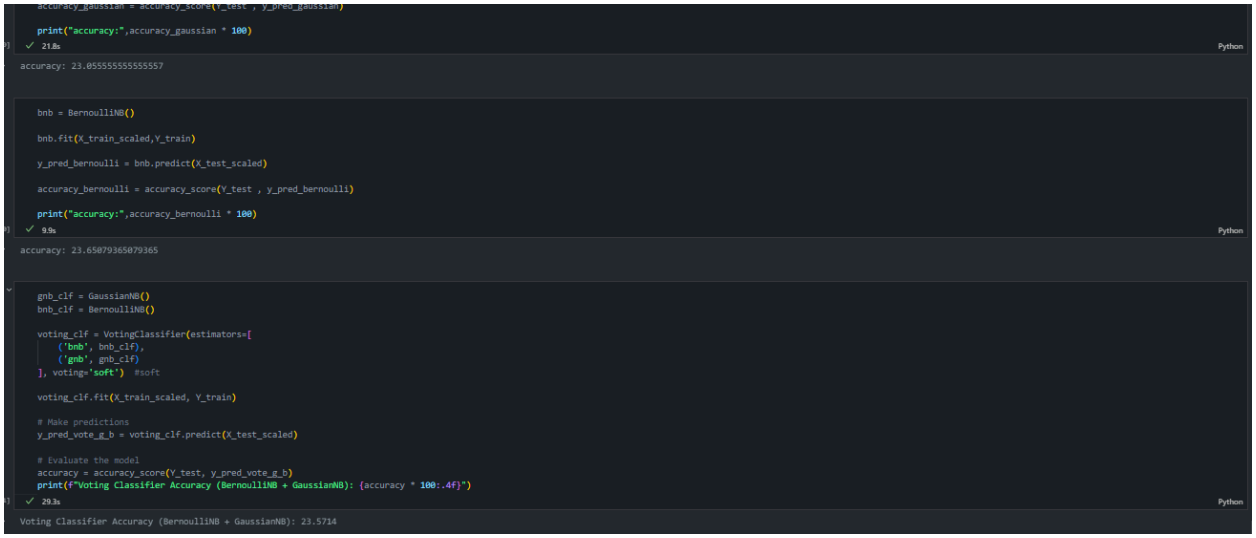
For the purpose of feature extraction I have used batch processing based on the sliding window algorithm in order to reduce load on the ram as well as optimize run time.

3).Model Selection and Implementation-

Several Models and their ensembles were trained Finally achieving an accuracy of 32 percent(31.98 percent) using an ensemble of decision trees(using the XGBOOST classifier).
The Models were trained using readily available library implementations of each of Decision trees , Random forests , perceptron , Bernoulli and Guassian Naive Bayes , as well the XGBoost algorithm , as well as a Voting Classifier.

Accuracy Readings are as below -





```
accuracy_gaussian = accuracy_score(Y_test , y_pred_gaussian)

print("accuracy:",accuracy_gaussian * 100)
```

21.8s Python

```
accuracy: 23.055555555555557
```

```
bnb = BernoulliNB()

bnb.fit(X_train_scaled,Y_train)

y_pred_bernoulli = bnb.predict(X_test_scaled)

accuracy_bernoulli = accuracy_score(Y_test , y_pred_bernoulli)

print("accuracy:",accuracy_bernoulli * 100)
```

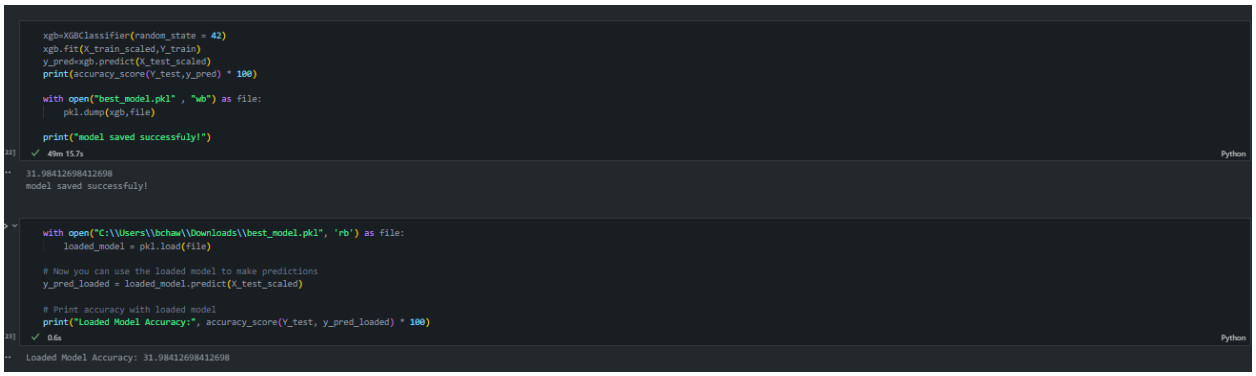
9.9s Python

```
accuracy: 23.65079365079365
```

```
gnb_clf = GaussianNB()
bnb_clf = BernoulliNB()

voting_clf = VotingClassifier(estimators=[
    ('bnb', bnb_clf),
    ('gnb', gnb_clf)
], voting='soft')  #soft

voting_clf.fit(X_train_scaled, Y_train)

# Make predictions
y_pred_vote_g_b = voting_clf.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(Y_test, y_pred_vote_g_b)
print(f"Voting Classifier Accuracy (BernoulliNB + GaussianNB): {accuracy * 100:.4f}")
```

29.3s Python

```
Voting Classifier Accuracy (BernoulliNB + GaussianNB): 23.5714
```



```
xgb=XGBClassifier(random_state = 42)
xgb.fit(X_train_scaled,Y_train)
y_pred=xgb.predict(X_test_scaled)
print(accuracy_score(Y_test,y_pred) * 100)

with open("best_model.pkl" , "wb") as file:
    pkl.dump(xgb,file)

print("model saved successfuly!")
```

49m 15.7s Python

```
31.98412698412698
model saved successfuly!
```

```
with open("C:\\Users\\bchaw\\Downloads\\best_model.pkl", 'rb') as file:
    loaded_model = pkl.load(file)

# Now you can use the loaded model to make predictions
y_pred_loaded = loaded_model.predict(X_test_scaled)

# Print accuracy with loaded model
print("Loaded Model Accuracy:", accuracy_score(Y_test, y_pred_loaded) * 100)
```

0.6s Python

```
Loaded Model Accuracy: 31.98412698412698
```

A random-state value of 42 has been used throughout the code to keep outputs fixed upon every fresh execution of the code.