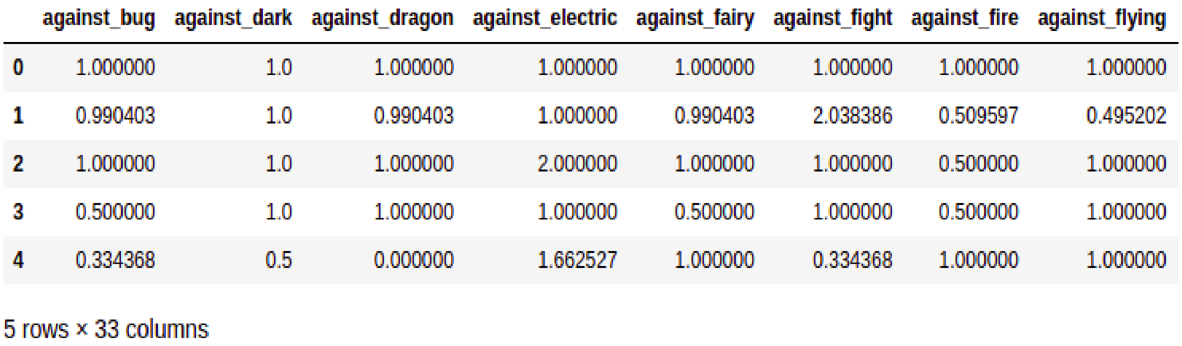
Assignment 2
(Dimensionality Reduction,
Feature Selection and Unsupervised Learning)

Ahmed Shehata Mahmoud **Sarah Hossam Elmowafy**

1. Loading Dataset

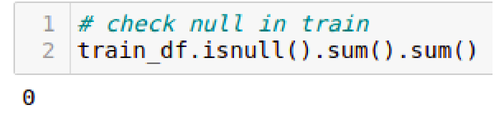
- Pokémon data set, numerical data with 32 features and one target.



	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	against_fire	against_flying
0	1.000000	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	0.990403	1.0	0.990403	1.000000	0.990403	2.038386	0.509597	0.495202
2	1.000000	1.0	1.000000	2.000000	1.000000	1.000000	0.500000	1.000000
3	0.500000	1.0	1.000000	1.000000	0.500000	1.000000	0.500000	1.000000
4	0.334368	0.5	0.000000	1.662527	1.000000	0.334368	1.000000	1.000000

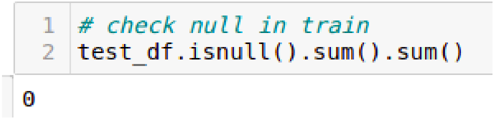
5 rows × 33 columns

- The data has no null values.



```
# check null in train
train_df.isnull().sum().sum()
```

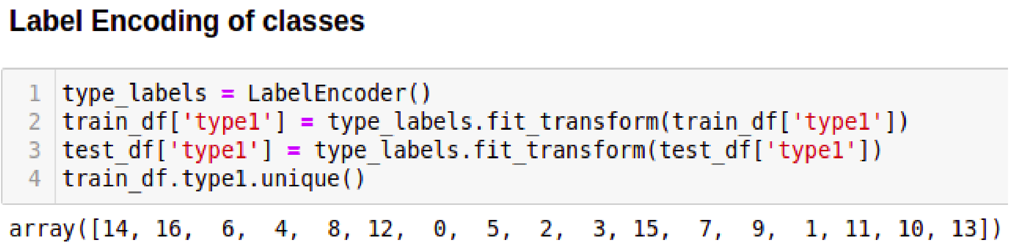
```
0
```



```
# check null in train
test_df.isnull().sum().sum()
```

```
0
```

- Encoding the target feature using label encoder.

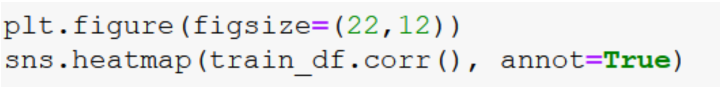


Label Encoding of classes

```
type_labels = LabelEncoder()
train_df['type1'] = type_labels.fit_transform(train_df['type1'])
test_df['type1'] = type_labels.fit_transform(test_df['type1'])
train_df.type1.unique()
```

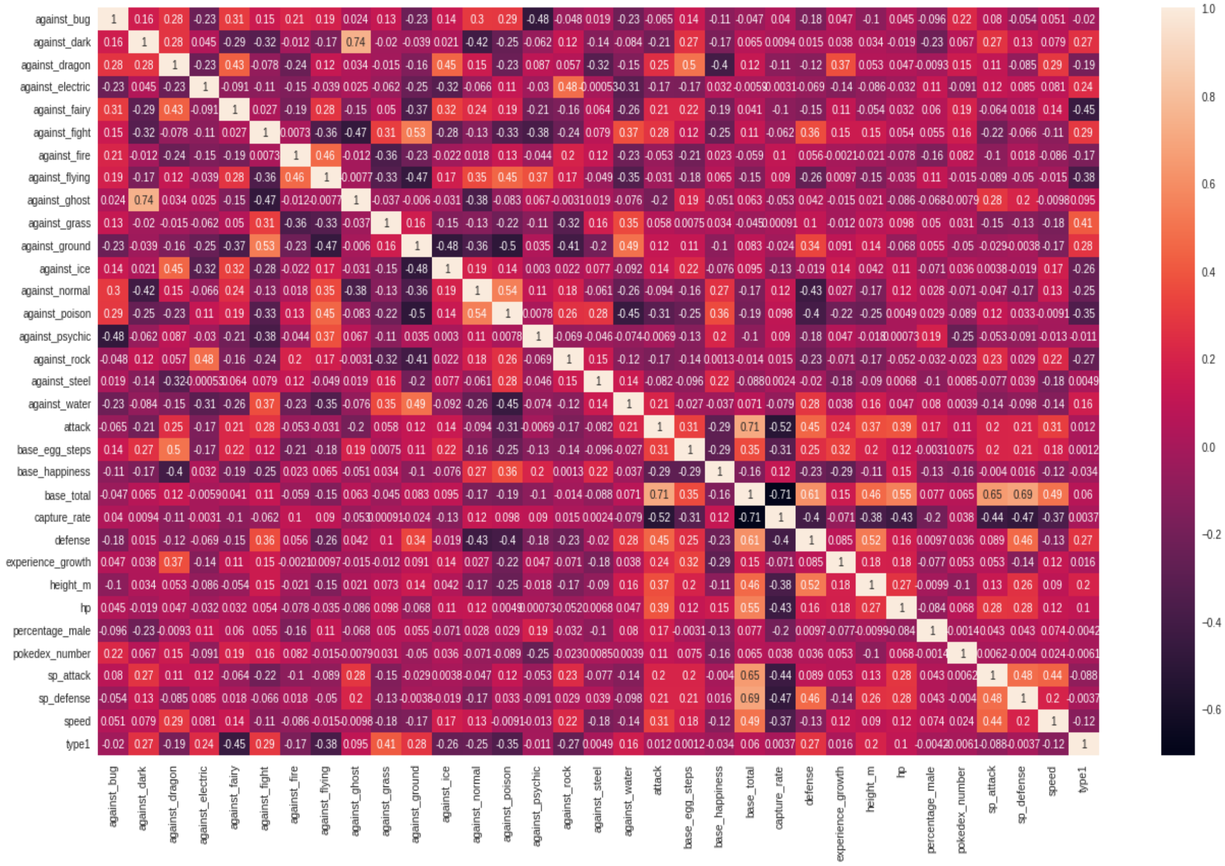
```
array([14, 16,  6,  4,  8, 12,  0,  5,  2,  3, 15,  7,  9,  1, 11, 10, 13])
```

- Heatmap of Features



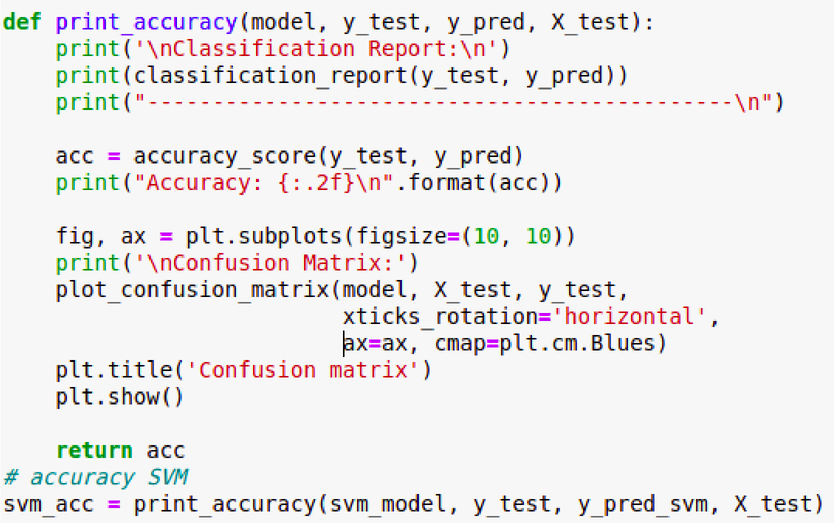
```
plt.figure(figsize=(22,12))
sns.heatmap(train_df.corr(), annot=True)
```

- Heatmap of Features.



2. Building SVM and GNB Classifiers

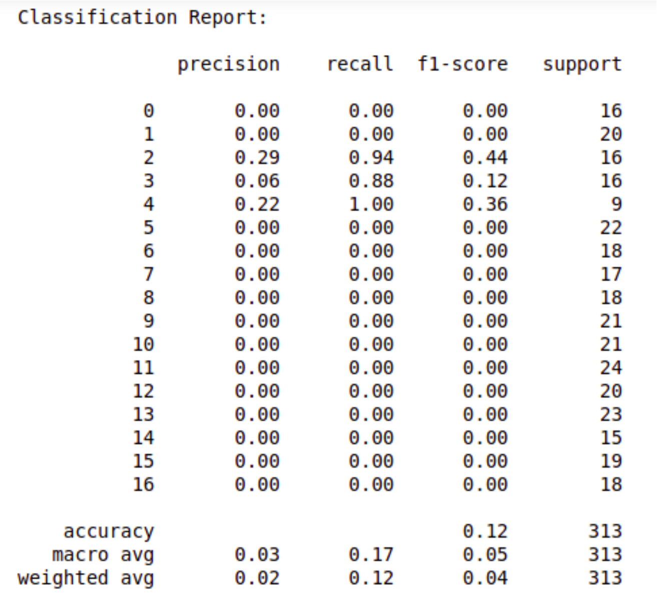
a. Create Method to print model accuracies.



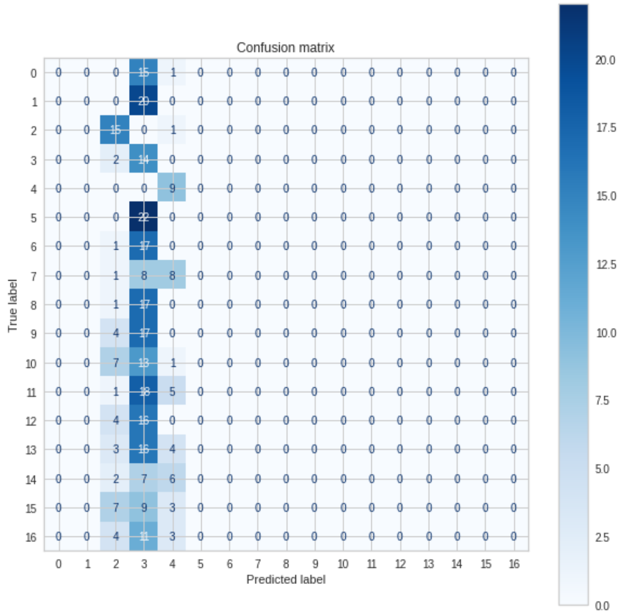
```
def print_accuracy(model, y_test, y_pred, X_test):
    print('\nClassification Report:\n')
    print(classification_report(y_test, y_pred))
    print("---------------------------------------------\n")

    acc = accuracy_score(y_test, y_pred)
    print("Accuracy: {:.2f}\n".format(acc))

    fig, ax = plt.subplots(figsize=(10, 10))
    print('\nConfusion Matrix:')
    plot_confusion_matrix(model, X_test, y_test,
                          xticks_rotation='horizontal',
                          ax=ax, cmap=plt.cm.Blues)
    plt.title('Confusion matrix')
    plt.show()

    return acc
# accuracy SVM
svm_acc = print_accuracy(svm_model, y_test, y_pred_svm, X_test)
```

3. SVM

- Build model based on RBF kernel
- Accuracy : **0.12**



```
Classification Report:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        16
           1       0.00      0.00      0.00        20
           2       0.29      0.94      0.44        16
           3       0.06      0.88      0.12        16
           4       0.22      1.00      0.36         9
           5       0.00      0.00      0.00        22
           6       0.00      0.00      0.00        18
           7       0.00      0.00      0.00        17
           8       0.00      0.00      0.00        18
           9       0.00      0.00      0.00        21
          10       0.00      0.00      0.00        21
          11       0.00      0.00      0.00        24
          12       0.00      0.00      0.00        20
          13       0.00      0.00      0.00        23
          14       0.00      0.00      0.00        15
          15       0.00      0.00      0.00        19
          16       0.00      0.00      0.00        18

    accuracy                           0.12       313
   macro avg       0.03      0.17      0.05       313
weighted avg       0.02      0.12      0.04       313
```

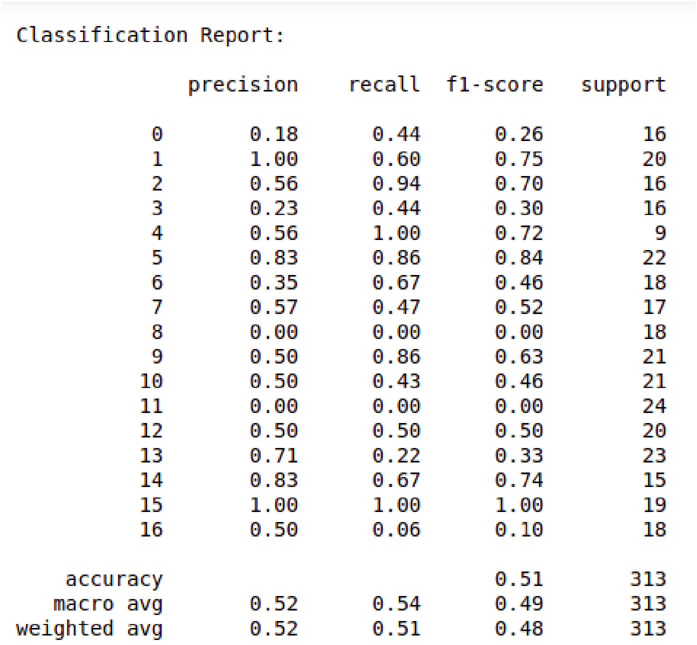


Confusion matrix

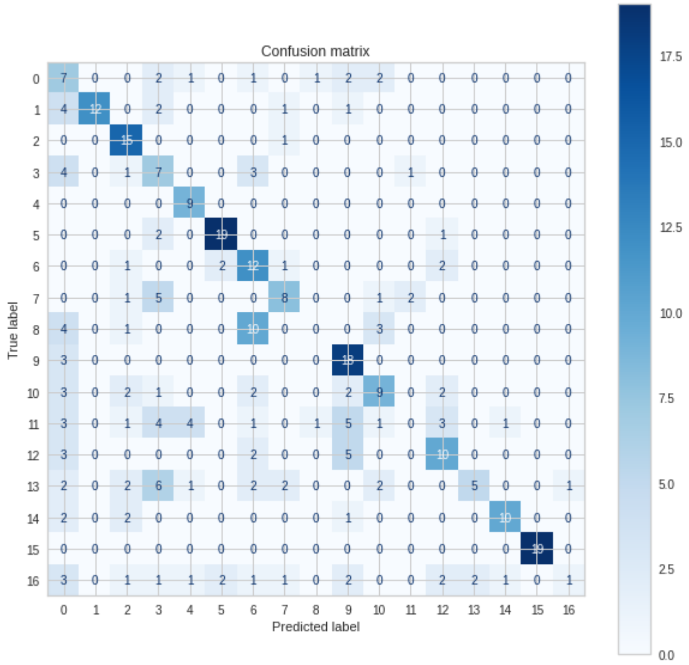
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	15	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	15	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	2	14	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	17	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	1	8	8	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	17	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	4	17	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	7	13	1	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	1	18	5	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	4	16	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	3	16	4	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	2	7	6	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	7	9	3	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	4	11	3	0	0	0	0	0	0	0	0	0	0	0	0

4. GNB

- Accuracy : **0.51**



```
Classification Report:

              precision    recall  f1-score   support

           0       0.18      0.44      0.26        16
           1       1.00      0.60      0.75        20
           2       0.56      0.94      0.70        16
           3       0.23      0.44      0.30        16
           4       0.56      1.00      0.72         9
           5       0.83      0.86      0.84        22
           6       0.35      0.67      0.46        18
           7       0.57      0.47      0.52        17
           8       0.00      0.00      0.00        18
           9       0.50      0.86      0.63        21
          10       0.50      0.43      0.46        21
          11       0.00      0.00      0.00        24
          12       0.50      0.50      0.50        20
          13       0.71      0.22      0.33        23
          14       0.83      0.67      0.74        15
          15       1.00      1.00      1.00        19
          16       0.50      0.06      0.10        18

    accuracy                           0.51       313
   macro avg       0.52      0.54      0.49       313
weighted avg       0.52      0.51      0.48       313
```

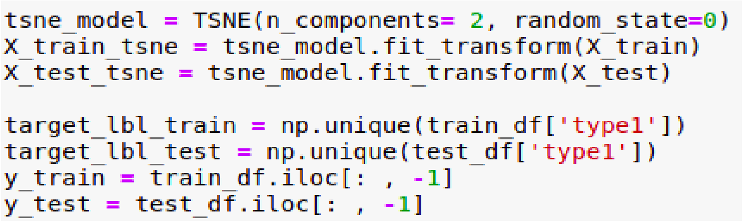


Confusion matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	7	0	0	2	1	0	1	0	1	2	2	0	0	0	0	0	0
1	4	12	0	2	0	0	0	1	0	1	0	0	0	0	0	0	0
2	0	0	15	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	4	0	1	7	0	0	3	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	2	0	19	0	0	0	0	0	0	1	0	0	0	0
6	0	0	1	0	0	2	12	1	0	0	0	0	2	0	0	0	0
7	0	0	1	5	0	0	0	8	0	0	1	2	0	0	0	0	0
8	4	0	1	0	0	0	10	0	0	0	3	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0
10	3	0	2	1	0	0	2	0	0	2	9	0	2	0	0	0	0
11	3	0	1	4	4	0	1	0	1	5	1	0	3	0	1	0	0
12	3	0	0	0	0	0	2	0	0	5	0	0	10	0	0	0	0
13	2	0	2	6	1	0	2	2	0	0	2	0	0	5	0	0	1
14	2	0	2	0	0	0	0	0	0	1	0	0	0	0	10	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
16	3	0	1	1	1	2	1	1	0	2	0	0	2	2	1	0	1

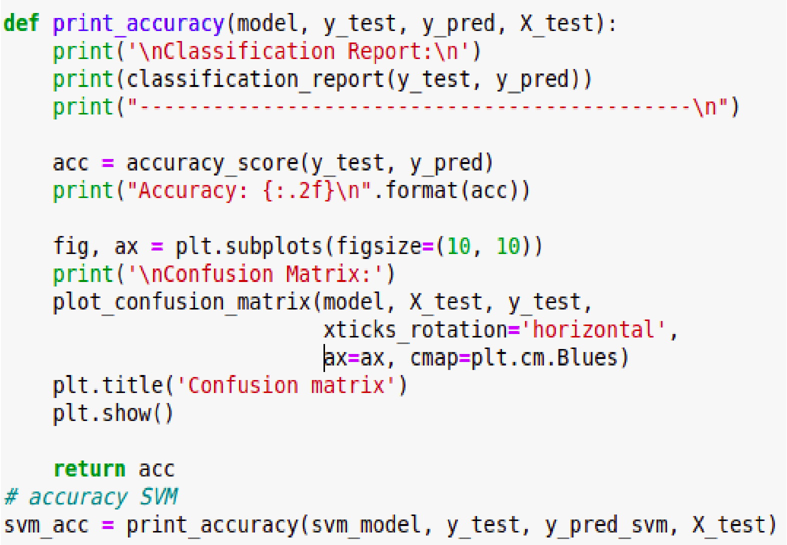
5. Apply TSNE to train and test

- Instantiate TSNE with 2 components and fit it to train and test data



```
tsne_model = TSNE(n_components= 2, random_state=0)
X_train_tsne = tsne_model.fit_transform(X_train)
X_test_tsne = tsne_model.fit_transform(X_test)

target_lbl_train = np.unique(train_df['type1'])
target_lbl_test = np.unique(test_df['type1'])
y_train = train_df.iloc[: , -1]
y_test = test_df.iloc[: , -1]
```

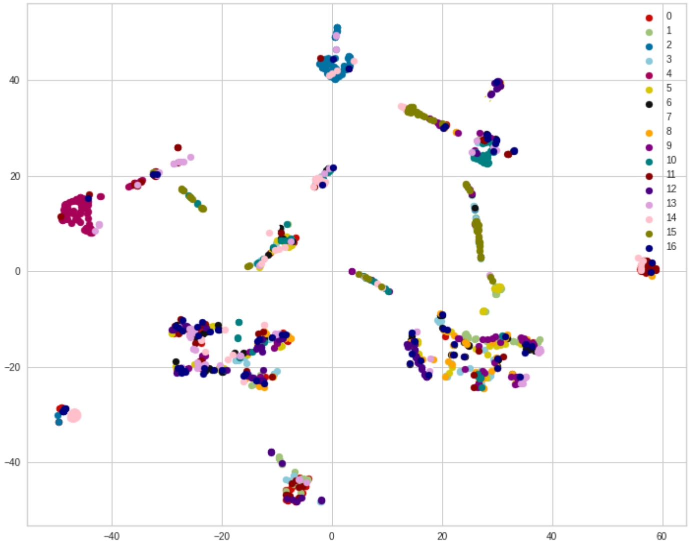
- Build a method to visualize the results



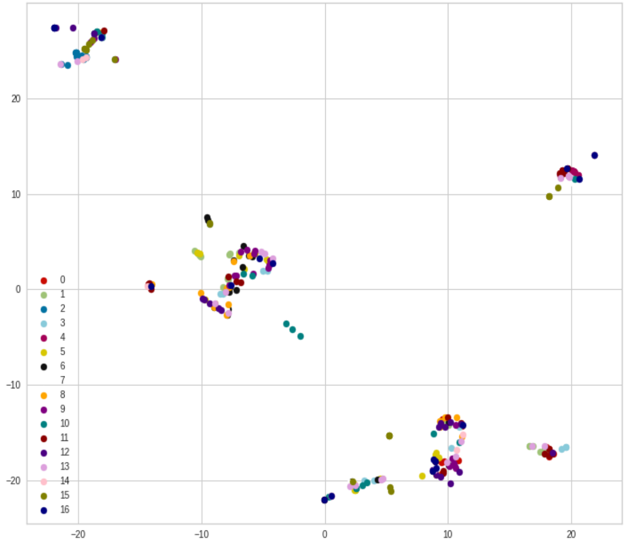
```
def print_accuracy(model, y_test, y_pred, X_test):
    print('\nClassification Report:\n')
    print(classification_report(y_test, y_pred))
    print("---------------------------------------------\n")

    acc = accuracy_score(y_test, y_pred)
    print("Accuracy: {:.2f}\n".format(acc))

    fig, ax = plt.subplots(figsize=(10, 10))
    print('\nConfusion Matrix:')
    plot_confusion_matrix(model, X_test, y_test,
                          xticks_rotation='horizontal',
                          ax=ax, cmap=plt.cm.Blues)
    plt.title('Confusion matrix')
    plt.show()

    return acc
# accuracy SVM
svm_acc = print_accuracy(svm_model, y_test, y_pred_svm, X_test)
```

- Visualize train and test using the method

Train data



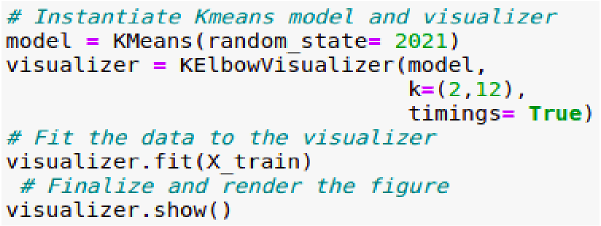
Test data



6. Select best number of clusters for k-means.

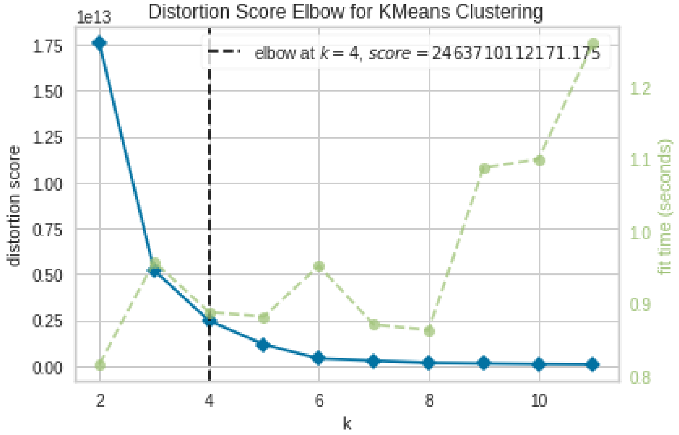
a. Plot the distortion score (a.k.a inertia) vs the number of clusters

Instantiate K-means model and visualizer



```
# Instantiate Kmeans model and visualizer
model = KMeans(random_state= 2021)
visualizer = KElbowVisualizer(model,
                              k=(2,12),
                              timings= True)
# Fit the data to the visualizer
visualizer.fit(X_train)
 # Finalize and render the figure
visualizer.show()
```

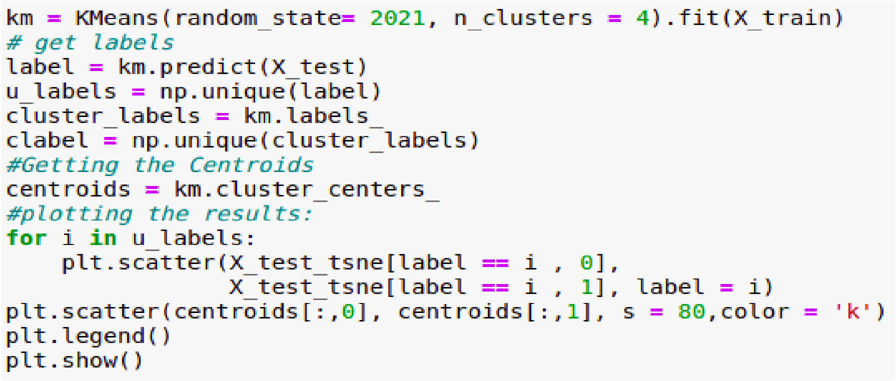
Elbow method



b. From Elbow graph, the best number of clusters is **4**

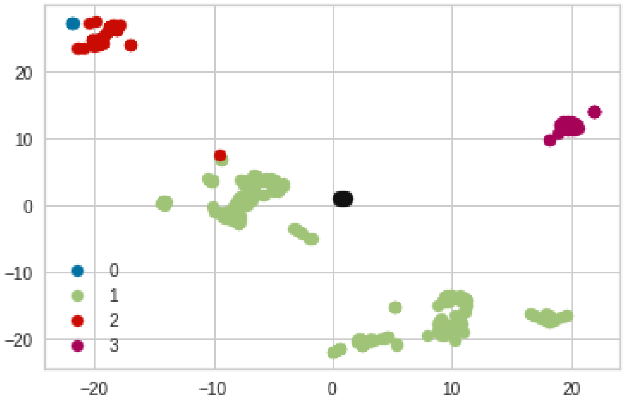
c. Plot clustered data with optimum number of clusters

- Build K-means model

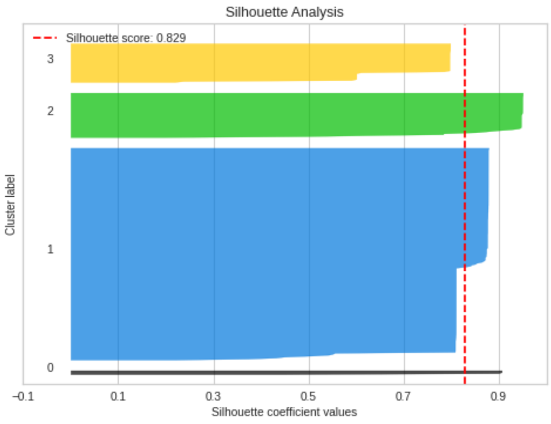


```
km = KMeans(random_state= 2021, n_clusters = 4).fit(X_train)
# get labels
label = km.predict(X_test)
u_labels = np.unique(label)
cluster_labels = km.labels_
clabel = np.unique(cluster_labels)
#Getting the Centroids
centroids = km.cluster_centers_
#plotting the results:
for i in u_labels:
    plt.scatter(X_test_tsne[label == i , 0],
                X_test_tsne[label == i , 1], label = i)
plt.scatter(centroids[:,0], centroids[:,1], s = 80,color = 'k')
plt.legend()
plt.show()
```

Clusters

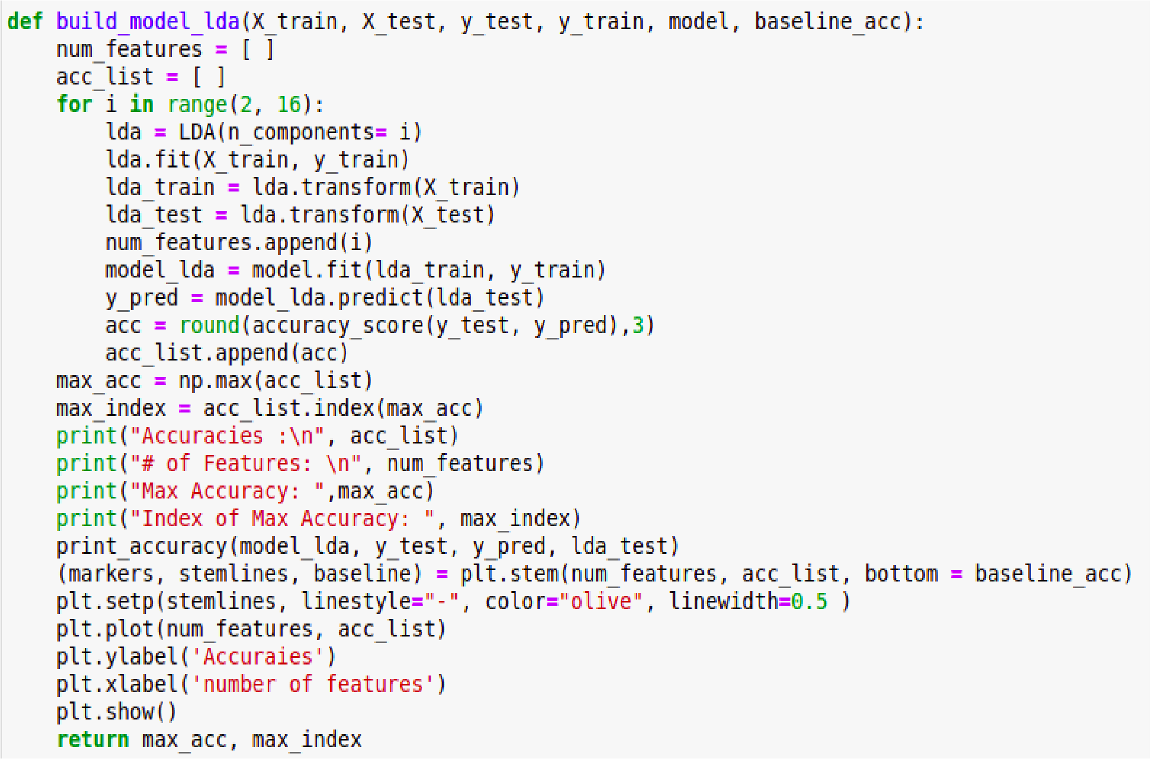


Silhouette



7. Apply one of the following Dimensionality Reduction (DR) methods to data

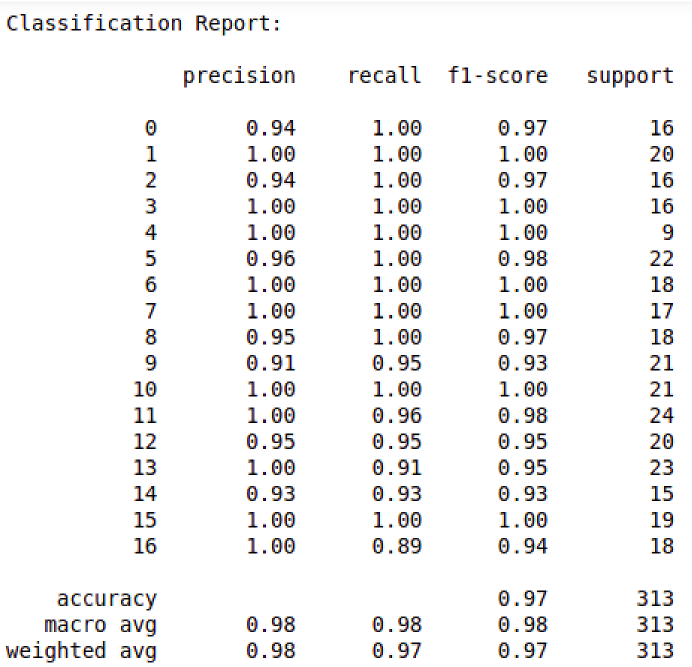
a. Here, we select LDA as DR technique



```
def build_model_lda(X_train, X_test, y_test, y_train, model, baseline_acc):
    num_features = [ ]
    acc_list = [ ]
    for i in range(2, 16):
        lda = LDA(n_components= i)
        lda.fit(X_train, y_train)
        lda_train = lda.transform(X_train)
        lda_test = lda.transform(X_test)
        num_features.append(i)
        model_lda = model.fit(lda_train, y_train)
        y_pred = model_lda.predict(lda_test)
        acc = round(accuracy_score(y_test, y_pred),3)
        acc_list.append(acc)
    max_acc = np.max(acc_list)
    max_index = acc_list.index(max_acc)
    print("Accuracies :\n", acc_list)
    print("# of Features: \n", num_features)
    print("Max Accuracy: ",max_acc)
    print("Index of Max Accuracy: ", max_index)
    print_accuracy(model_lda, y_test, y_pred, lda_test)
    (markers, stemlines, baseline) = plt.stem(num_features, acc_list, bottom = baseline_acc)
    plt.setp(stemlines, linestyle="-", color="olive", linewidth=0.5 )
    plt.plot(num_features, acc_list)
    plt.ylabel('Accuraies')
    plt.xlabel('number of features')
    plt.show()
    return max_acc, max_index
```

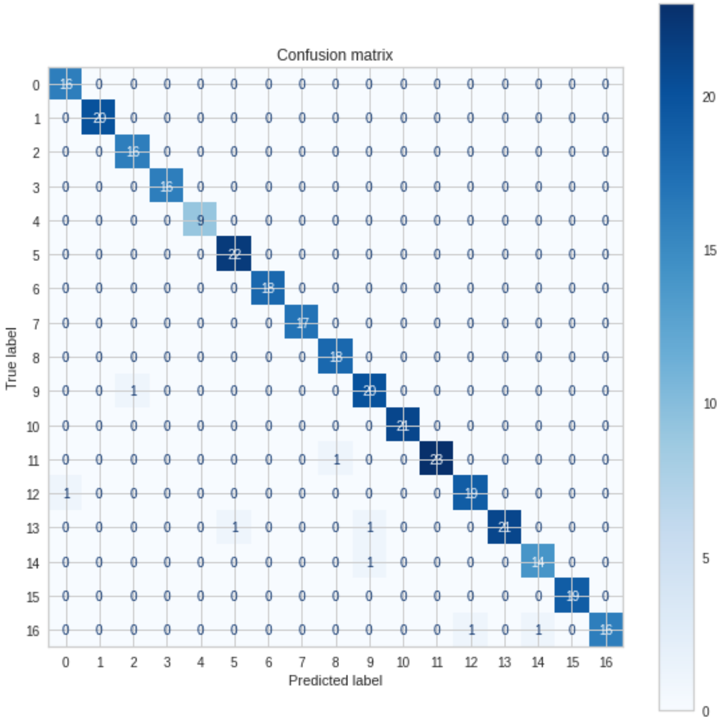
b. SVM based on LDA

SVM Classification Report



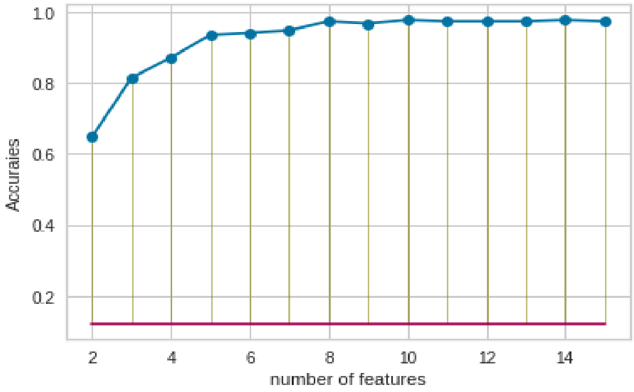
```
Classification Report:

              precision    recall  f1-score   support

           0       0.94      1.00      0.97        16
           1       1.00      1.00      1.00        20
           2       0.94      1.00      0.97        16
           3       1.00      1.00      1.00        16
           4       1.00      1.00      1.00         9
           5       0.96      1.00      0.98        22
           6       1.00      1.00      1.00        18
           7       1.00      1.00      1.00        17
           8       0.95      1.00      0.97        18
           9       0.91      0.95      0.93        21
          10       1.00      1.00      1.00        21
          11       1.00      0.96      0.98        24
          12       0.95      0.95      0.95        20
          13       1.00      0.91      0.95        23
          14       0.93      0.93      0.93        15
          15       1.00      1.00      1.00        19
          16       1.00      0.89      0.94        18

    accuracy                           0.97       313
   macro avg       0.98      0.98      0.98       313
weighted avg       0.98      0.97      0.97       313
```

Confusion Matrix



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	20	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	23	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0
13	0	0	0	0	0	1	0	0	0	1	0	0	0	21	0	0	0
14	0	0	0	0	0	0	0	0	0	1	0	0	0	0	14	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	16

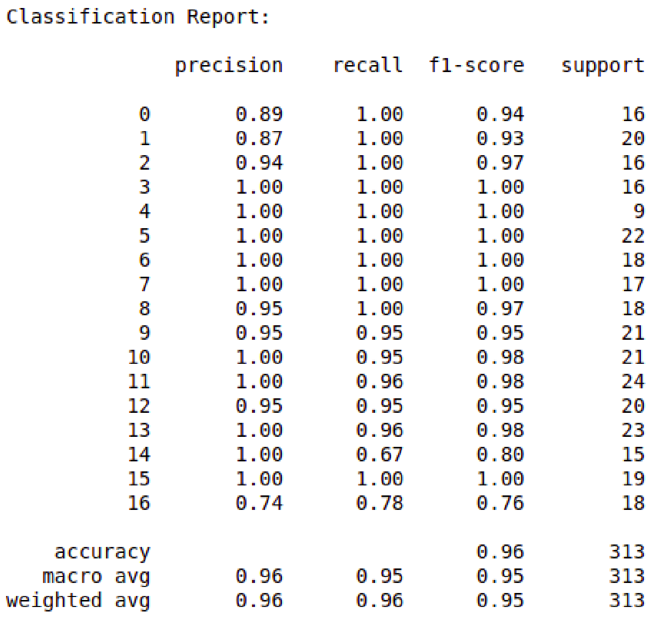
- After build SVM model based on LDA data, the accuracy rose dramatically from 0.12 to 0.978



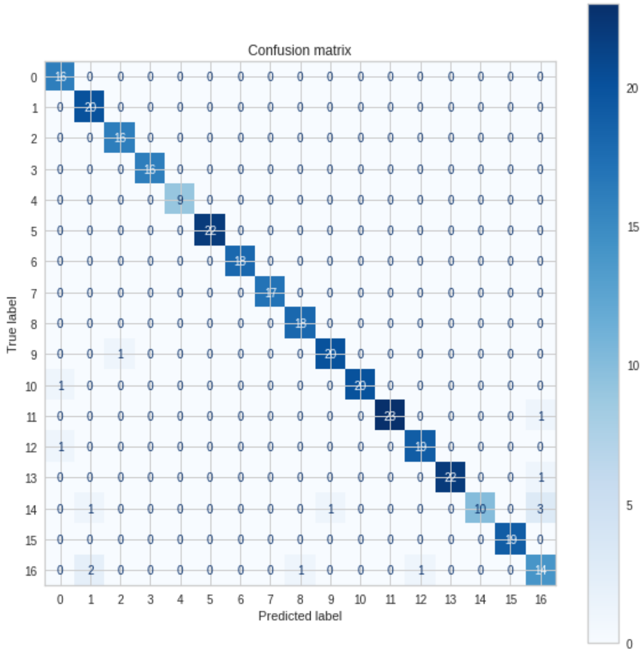
c. GNB based on LDA

GNB Classification Report

Confusion Matrix

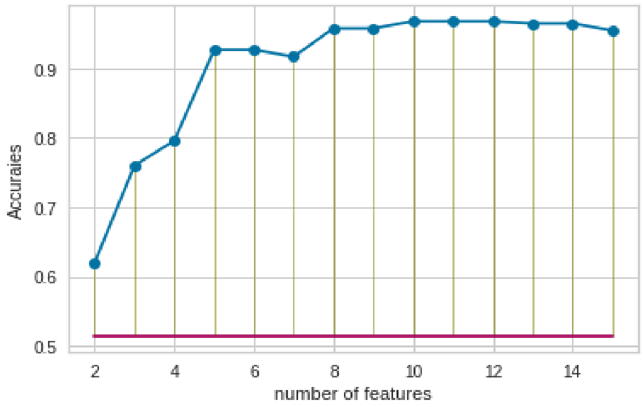


```
Classification Report:

              precision    recall  f1-score   support

           0       0.89      1.00      0.94        16
           1       0.87      1.00      0.93        20
           2       0.94      1.00      0.97        16
           3       1.00      1.00      1.00        16
           4       1.00      1.00      1.00         9
           5       1.00      1.00      1.00        22
           6       1.00      1.00      1.00        18
           7       1.00      1.00      1.00        17
           8       0.95      1.00      0.97        18
           9       0.95      0.95      0.95        21
          10       1.00      0.95      0.98        21
          11       1.00      0.96      0.98        24
          12       0.95      0.95      0.95        20
          13       1.00      0.96      0.98        23
          14       1.00      0.67      0.80        15
          15       1.00      1.00      1.00        19
          16       0.74      0.78      0.76        18

    accuracy                           0.96       313
   macro avg       0.96      0.95      0.95       313
weighted avg       0.96      0.96      0.95       313
```



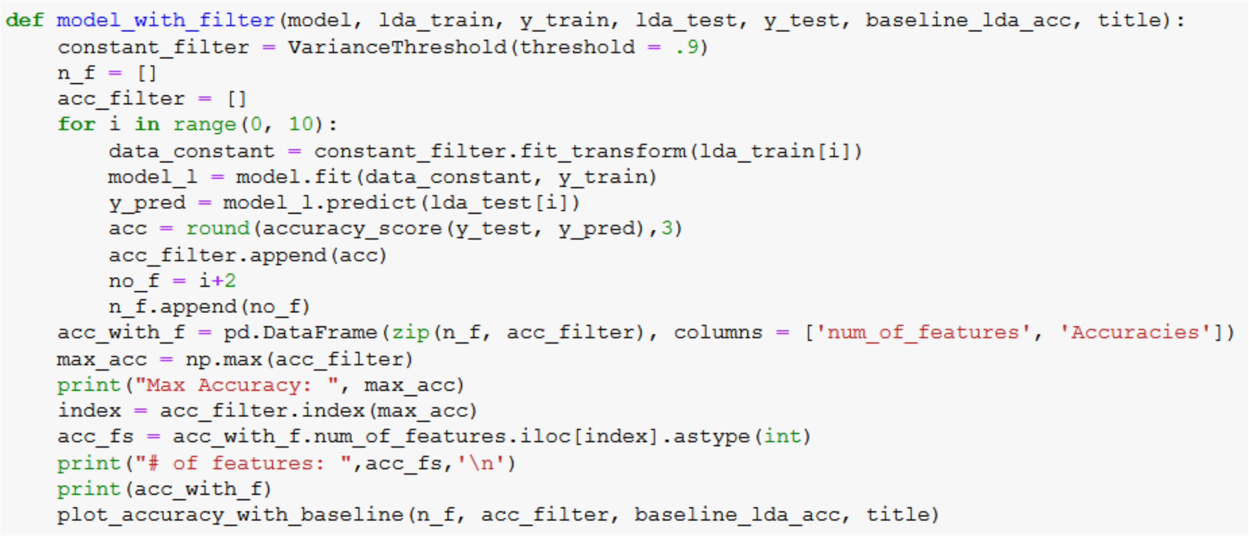
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	20	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	0	1
12	1	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	22	0	0	1
14	0	1	0	0	0	0	0	0	0	1	0	0	0	0	10	0	3
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
16	0	2	0	0	0	0	0	0	1	0	0	0	1	0	0	0	14

- After build GNB model based on LDA data, the accuracy rose from 0.51 to 0.96



Feature Selection methods
Filtering (Variance Threshold)

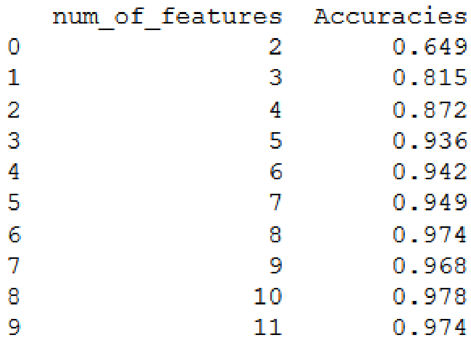
Here we use Variance threshold and no matter what the threshold is, it doesn't affect the results neither for SVM nor Naïve Bayes. The number of features isn't reduced, it seems like LDA didn't save an effort for making the features unique without any correlations or dependency to be filtered.



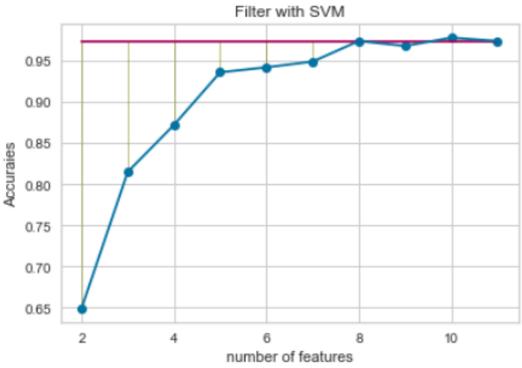
```python
def model_with_filter(model, lda_train, y_train, lda_test, y_test, baseline_lda_acc, title):
    constant_filter = VarianceThreshold(threshold = .9)
    n_f = []
    acc_filter = []
    for i in range(0, 10):
        data_constant = constant_filter.fit_transform(lda_train[i])
        model_l = model.fit(data_constant, y_train)
        y_pred = model_l.predict(lda_test[i])
        acc = round(accuracy_score(y_test, y_pred),3)
        acc_filter.append(acc)
        no_f = i+2
        n_f.append(no_f)
    acc_with_f = pd.DataFrame(zip(n_f, acc_filter), columns = ['num_of_features', 'Accuracies'])
    max_acc = np.max(acc_filter)
    print("Max Accuracy: ", max_acc)
    index = acc_filter.index(max_acc)
    acc_fs = acc_with_f.num_of_features.iloc[index].astype(int)
    print("# of features: ",acc_fs,'\n')
    print(acc_with_f)
    plot_accuracy_with_baseline(n_f, acc_filter, baseline_lda_acc, title)
```

1- SVM

Number of features: 10, Max accuracy: `0.978,` which is a little higher than SVM with LDA without filtering which is .97 as the baseline shows at 10.

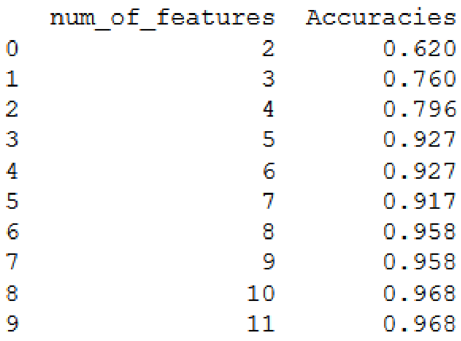


```
   num_of_features  Accuracies
0                2       0.649
1                3       0.815
2                4       0.872
3                5       0.936
4                6       0.942
5                7       0.949
6                8       0.974
7                9       0.968
8               10       0.978
9               11       0.974
```

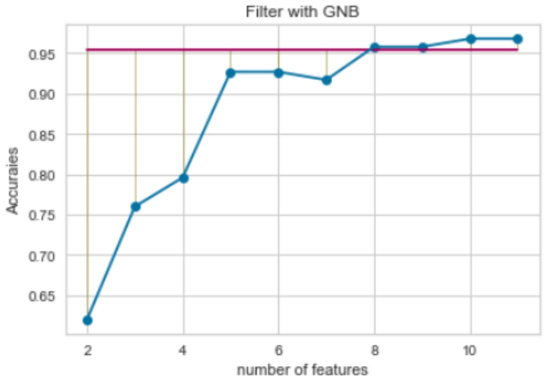


2- Gaussian Naïve Bayes (GNB)

Number of features: 10, Max accuracy `0.968`, which is a little higher than SVM with LDA without filtering which is .95 as the baseline shows at 10 also.

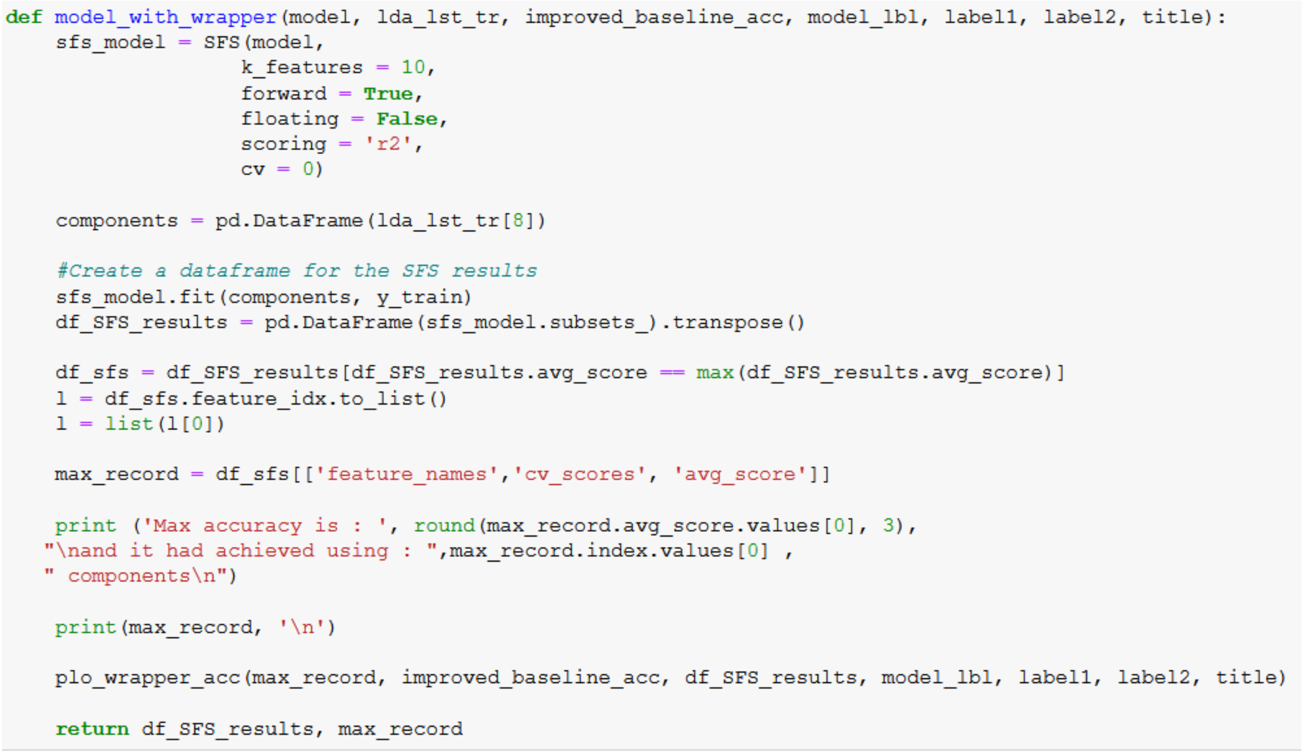


```
   num_of_features  Accuracies
0                2       0.620
1                3       0.760
2                4       0.796
3                5       0.927
4                6       0.927
5                7       0.917
6                8       0.958
7                9       0.958
8               10       0.968
9               11       0.968
```



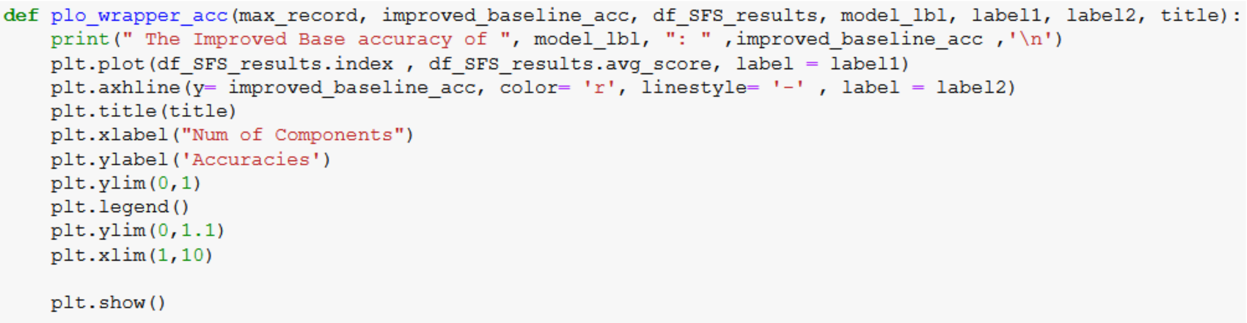
Wrapper Selection
Sequential Forward Selection

Here we used Sequential Forward Selection (SFS), passing the number of features (10) and trying to get the possible smaller number of features out of them using SFS method.



```
def model_with_wrapper(model, lda_lst_tr, improved_baseline_acc, model_lbl, label1, label2, title):
    sfs_model = SFS(model,
                   k_features = 10,
                   forward = True,
                   floating = False,
                   scoring = 'r2',
                   cv = 0)

    components = pd.DataFrame(lda_lst_tr[8])

    #Create a dataframe for the SFS results
    sfs_model.fit(components, y_train)
    df_SFS_results = pd.DataFrame(sfs_model.subsets_).transpose()

    df_sfs = df_SFS_results[df_SFS_results.avg_score == max(df_SFS_results.avg_score)]
    l = df_sfs.feature_idx.to_list()
    l = list(l[0])

    max_record = df_sfs[['feature_names','cv_scores', 'avg_score']]

    print ('Max accuracy is : ', round(max_record.avg_score.values[0], 3),
   "\nand it had achieved using : ",max_record.index.values[0] ,
   " components\n")

    print(max_record, '\n')

    plo_wrapper_acc(max_record, improved_baseline_acc, df_SFS_results, model_lbl, label1, label2, title)

    return df_SFS_results, max_record
```

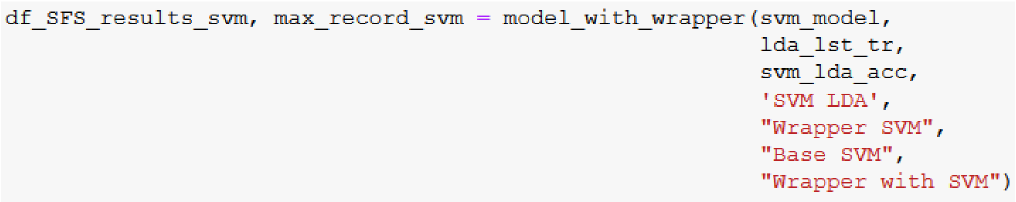
This function is used to plot the accuracy for each combination of features



```
def plo_wrapper_acc(max_record, improved_baseline_acc, df_SFS_results, model_lbl, label1, label2, title):
    print(" The Improved Base accuracy of ", model_lbl, ": " ,improved_baseline_acc ,'\n')
    plt.plot(df_SFS_results.index , df_SFS_results.avg_score, label = label1)
    plt.axhline(y= improved_baseline_acc, color= 'r', linestyle= '-' , label = label2)
    plt.title(title)
    plt.xlabel("Num of Components")
    plt.ylabel('Accuracies')
    plt.ylim(0,1)
    plt.legend()
    plt.ylim(0,1.1)
    plt.xlim(1,10)

    plt.show()
```

1- SVM

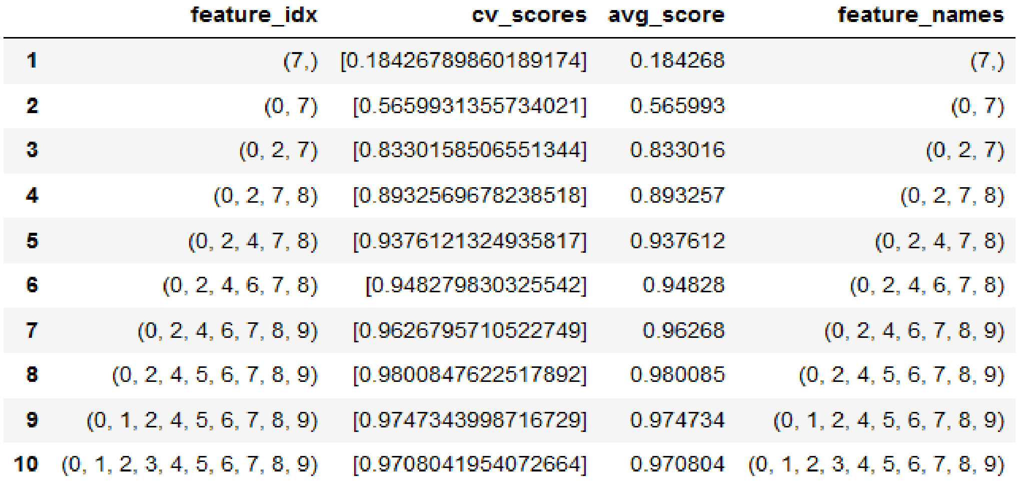
In the next code cell, we call the function to apply SFS wrapper with SVM model



```
df_SFS_results_svm, max_record_svm = model_with_wrapper(svm_model,
                                                        lda_lst_tr,
                                                        svm_lda_acc,
                                                        'SVM LDA',
                                                        "Wrapper SVM",
                                                        "Base SVM",
                                                        "Wrapper with SVM")
```

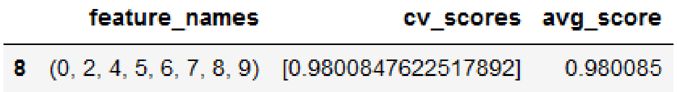
After displaying the combinations with their respect accuracies

display(df_SFS_results_svm, max_record_svm)



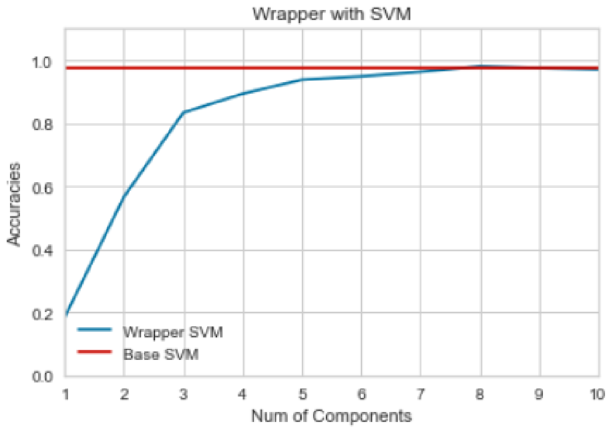
	feature_idx	cv_scores	avg_score	feature_names
1	(7,)	[0.18426789860189174]	0.184268	(7,)
2	(0, 7)	[0.5659931355734021]	0.565993	(0, 7)
3	(0, 2, 7)	[0.8330158506551344]	0.833016	(0, 2, 7)
4	(0, 2, 7, 8)	[0.8932569678238518]	0.893257	(0, 2, 7, 8)
5	(0, 2, 4, 7, 8)	[0.9376121324935817]	0.937612	(0, 2, 4, 7, 8)
6	(0, 2, 4, 6, 7, 8)	[0.948279830325542]	0.94828	(0, 2, 4, 6, 7, 8)
7	(0, 2, 4, 6, 7, 8, 9)	[0.9626795710522749]	0.96268	(0, 2, 4, 6, 7, 8, 9)
8	(0, 2, 4, 5, 6, 7, 8, 9)	[0.9800847622517892]	0.980085	(0, 2, 4, 5, 6, 7, 8, 9)
9	(0, 1, 2, 4, 5, 6, 7, 8, 9)	[0.9747343998716729]	0.974734	(0, 1, 2, 4, 5, 6, 7, 8, 9)
10	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)	[0.9708041954072664]	0.970804	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

The highest accuracy comes with just **8** combinations of features out of 10



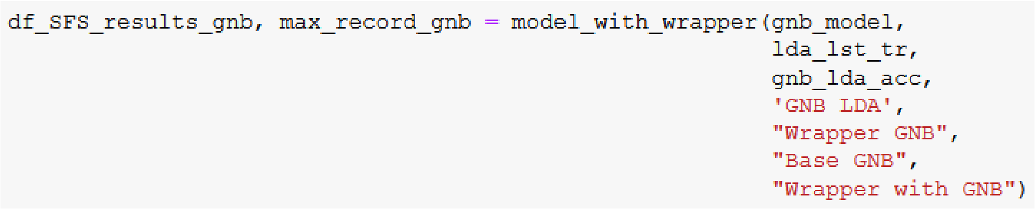
	feature_names	cv_scores	avg_score
8	(0, 2, 4, 5, 6, 7, 8, 9)	[0.9800847622517892]	0.980085

Here we see that at number of components equal to 8, the accuracy is high as much as the accuracy of using 10 components without using the wrapper as the baseline show here.



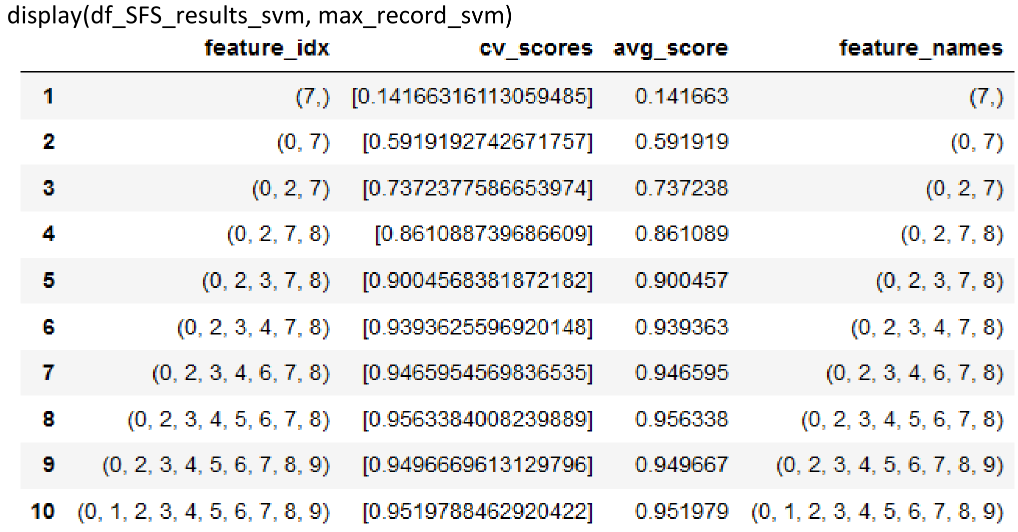
2- GNB

Here we call the function again with Naïve Bayes



```
df_SFS_results_gnb, max_record_gnb = model_with_wrapper(gnb_model,
                                                        lda_lst_tr,
                                                        gnb_lda_acc,
                                                        'GNB LDA',
                                                        "Wrapper GNB",
                                                        "Base GNB",
                                                        "Wrapper with GNB")
```

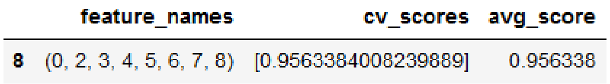
After displaying the combinations with their respect accuracies



display(df_SFS_results_svm, max_record_svm)

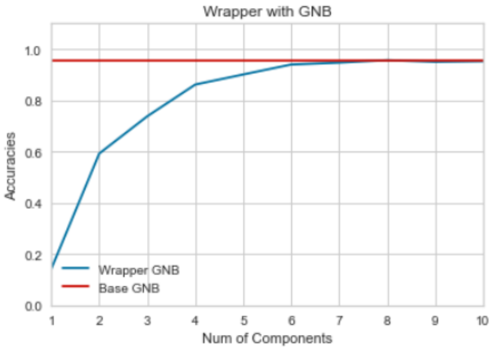
	feature_idx	cv_scores	avg_score	feature_names
1	(7,)	[0.14166316113059485]	0.141663	(7,)
2	(0, 7)	[0.5919192742671757]	0.591919	(0, 7)
3	(0, 2, 7)	[0.7372377586653974]	0.737238	(0, 2, 7)
4	(0, 2, 7, 8)	[0.861088739686609]	0.861089	(0, 2, 7, 8)
5	(0, 2, 3, 7, 8)	[0.9004568381872182]	0.900457	(0, 2, 3, 7, 8)
6	(0, 2, 3, 4, 7, 8)	[0.9393625596920148]	0.939363	(0, 2, 3, 4, 7, 8)
7	(0, 2, 3, 4, 6, 7, 8)	[0.9465954569836535]	0.946595	(0, 2, 3, 4, 6, 7, 8)
8	(0, 2, 3, 4, 5, 6, 7, 8)	[0.9563384008239889]	0.956338	(0, 2, 3, 4, 5, 6, 7, 8)
9	(0, 2, 3, 4, 5, 6, 7, 8, 9)	[0.9496669613129796]	0.949667	(0, 2, 3, 4, 5, 6, 7, 8, 9)
10	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)	[0.9519788462920422]	0.951979	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

The highest accuracy comes with just 8 combinations of features out of 10



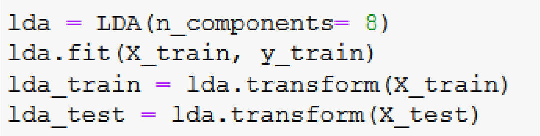
	feature_names	cv_scores	avg_score
8	(0, 2, 3, 4, 5, 6, 7, 8)	[0.9563384008239889]	0.956338

With Gaussian Naïve Bayes, we see that at number of components equal to 8, the accuracy is high as much as the accuracy of using 10 components without using the wrapper as the baseline show here. or even with the wrapper there's no need to more than 8 components as there's no enhancement in the accuracy by doing that, even it decreases.



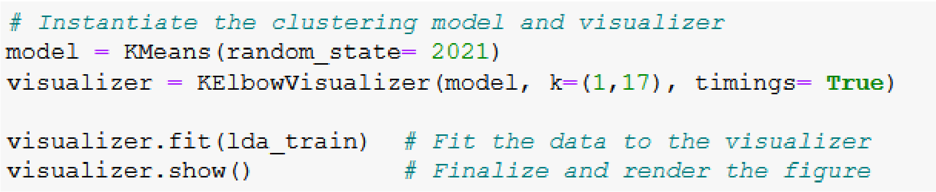
K-means based on optimal number of components

Depending on question 4 we will use 8 features



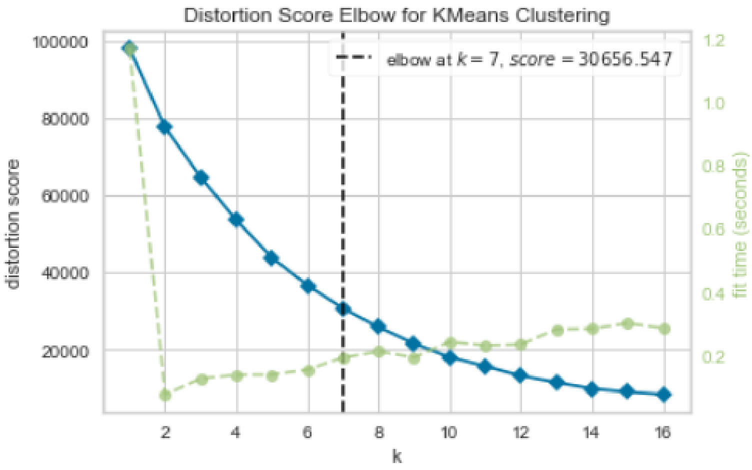
```
lda = LDA(n_components= 8)
lda.fit(X_train, y_train)
lda_train = lda.transform(X_train)
lda_test = lda.transform(X_test)
```

Code for plotting elbow method

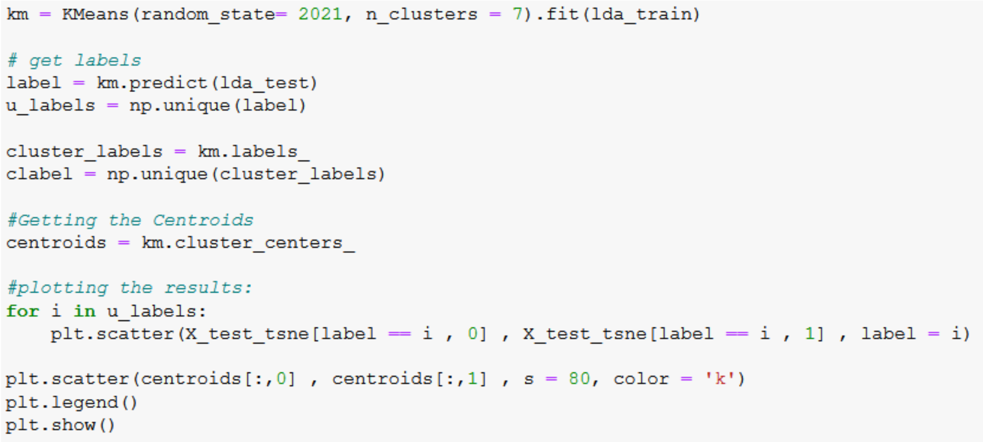


```
# Instantiate the clustering model and visualizer
model = KMeans(random_state= 2021)
visualizer = KElbowVisualizer(model, k=(1,17), timings= True)

visualizer.fit(lda_train)    # Fit the data to the visualizer
visualizer.show()            # Finalize and render the figure
```

Here's the elbow method shows that the best number of clusters is 7

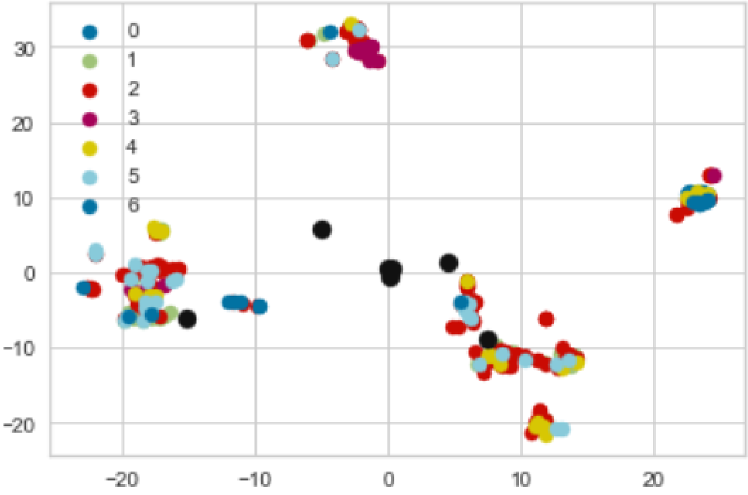


This code cell for plotting the clusters



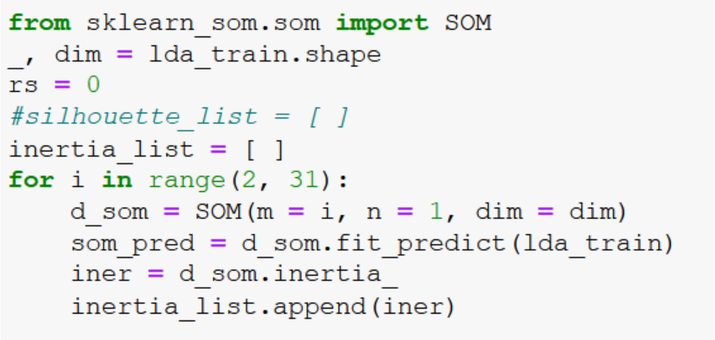
```
km = KMeans(random_state= 2021, n_clusters = 7).fit(lda_train)

# get labels
label = km.predict(lda_test)
u_labels = np.unique(label)

cluster_labels = km.labels_
clabel = np.unique(cluster_labels)

#Getting the Centroids
centroids = km.cluster_centers_

#plotting the results:
for i in u_labels:
    plt.scatter(X_test_tsne[label == i , 0] , X_test_tsne[label == i , 1] , label = i)

plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```

The optimal number of clusters for K-means with 8 components is 8



SOM

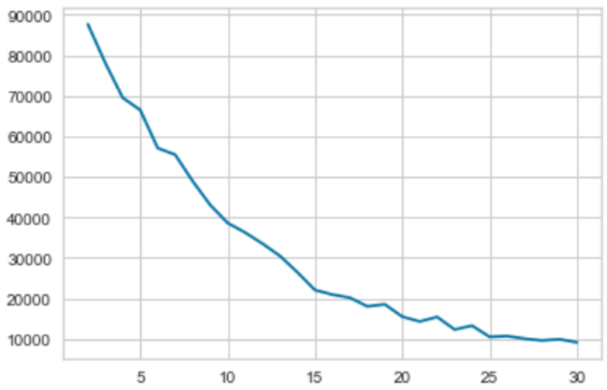
Here we use SOM from sklearn with 8 components from dimensionality in question 4, we calculate the accuracy for number of neurons from 2 to 30



```
from sklearn_som.som import SOM
_, dim = lda_train.shape
rs = 0
#silhouette_list = [ ]
inertia_list = [ ]
for i in range(2, 31):
    d_som = SOM(m = i, n = 1, dim = dim)
    som_pred = d_som.fit_predict(lda_train)
    iner = d_som.inertia_
    inertia_list.append(iner)
```

Here to plot the inertia for each number of neurons

```
from matplotlib import pyplot as plt
plt.plot(list(range(2, 31)), inertia_list)
plt.show()
```



The elbow method shows that the best number of neurons is laying between 15 and 17 we will go with 16 (also it will make a good-look matrix).

We will use minisom library as it is easier to use it in viewing the initial and final positions

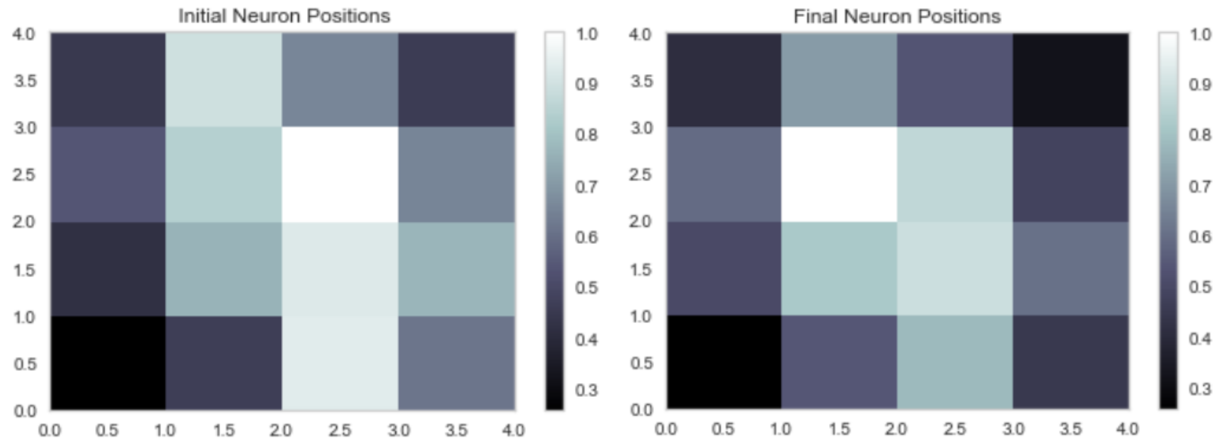
```
from minisom import MiniSom
# Set the hyper parameters
som_grid_rows = 4
som_grid_columns = 4
iterations = 10
sigma = 1
learning_rate = 0.5

# define SOM:
som = MiniSom(x = som_grid_rows, y = som_grid_columns, input_len=8, sigma=sigma, learning_rate=learning_rate)
# Initializing the weights
som.random_weights_init(lda_train)
s0 = som.distance_map().T
# Training
som.train_random(lda_train, iterations)
s1 = som.distance_map().T
```

And here we plot the initial and final positions

```
from pylab import plot, axis, show, pcolor, colorbar, bone
bone()
pcolor(s0)
colorbar()
plt.title(" Initial Neuron Positions")
show()
som.get_weights
```

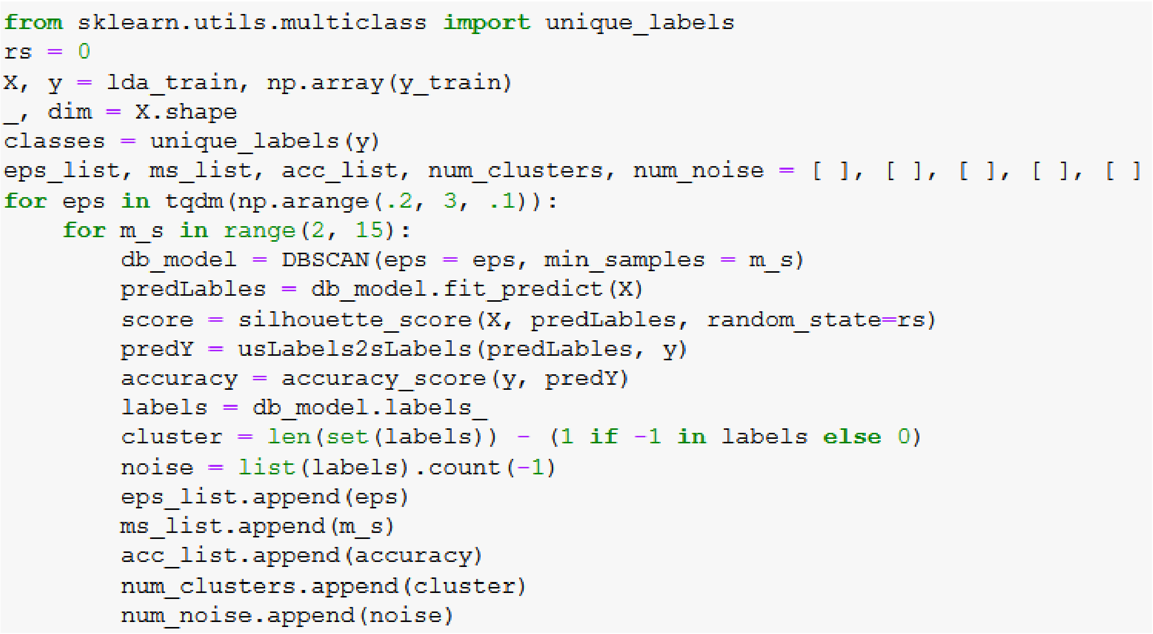
```
from pylab import plot, axis, show, pcolor, colorbar, bone
bone()
pcolor(s0)
colorbar()
plt.title("Final Neuron Positions")
show()
som.get_weights
```

The initial and final neurons positions



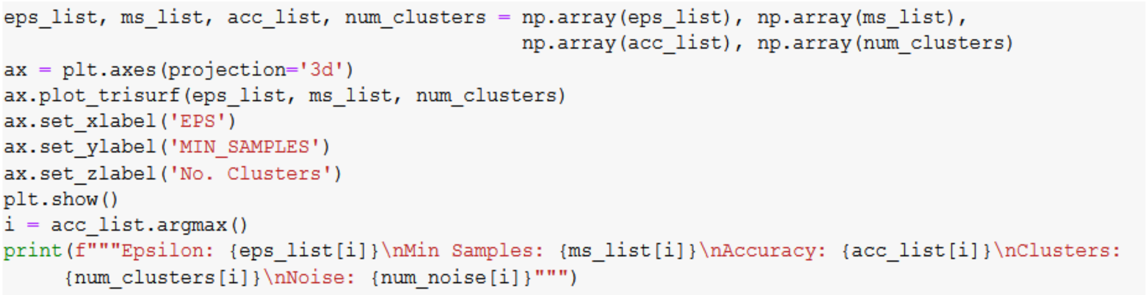
DBSCAN

Here we tune the parameters epsilon (0.2-3) and minpoints (2-15)



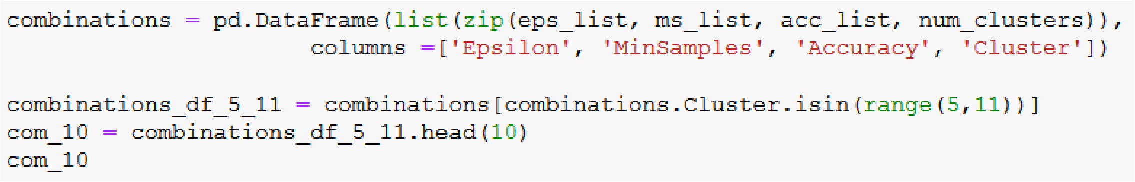
```
from sklearn.utils.multiclass import unique_labels
rs = 0
X, y = lda_train, np.array(y_train)
_, dim = X.shape
classes = unique_labels(y)
eps_list, ms_list, acc_list, num_clusters, num_noise = [ ], [ ], [ ], [ ], [ ]
for eps in tqdm(np.arange(.2, 3, .1)):
    for m_s in range(2, 15):
        db_model = DBSCAN(eps = eps, min_samples = m_s)
        predLables = db_model.fit_predict(X)
        score = silhouette_score(X, predLables, random_state=rs)
        predY = usLabels2sLabels(predLables, y)
        accuracy = accuracy_score(y, predY)
        labels = db_model.labels_
        cluster = len(set(labels)) - (1 if -1 in labels else 0)
        noise = list(labels).count(-1)
        eps_list.append(eps)
        ms_list.append(m_s)
        acc_list.append(accuracy)
        num_clusters.append(cluster)
        num_noise.append(noise)
```

This is how we plot the epsilon values and min samples with the clusters in 3d.



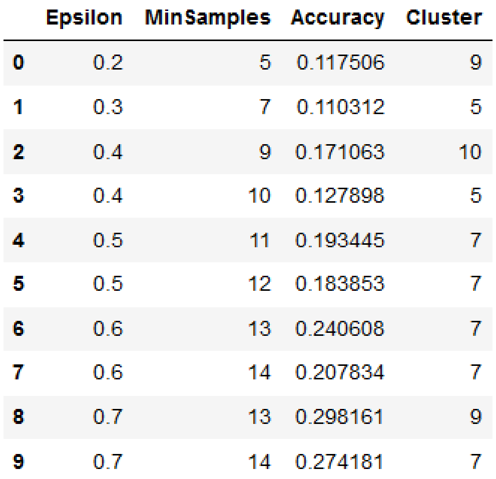
```
eps_list, ms_list, acc_list, num_clusters = np.array(eps_list), np.array(ms_list),
                                            np.array(acc_list), np.array(num_clusters)
ax = plt.axes(projection='3d')
ax.plot_trisurf(eps_list, ms_list, num_clusters)
ax.set_xlabel('EPS')
ax.set_ylabel('MIN_SAMPLES')
ax.set_zlabel('No. Clusters')
plt.show()
i = acc_list.argmax()
print(f"""Epsilon: {eps_list[i]}\nMin Samples: {ms_list[i]}\nAccuracy: {acc_list[i]}\nClusters:
    {num_clusters[i]}\nNoise: {num_noise[i]}""")
```

Here we choose the number of cluster 7 or around (5-11)



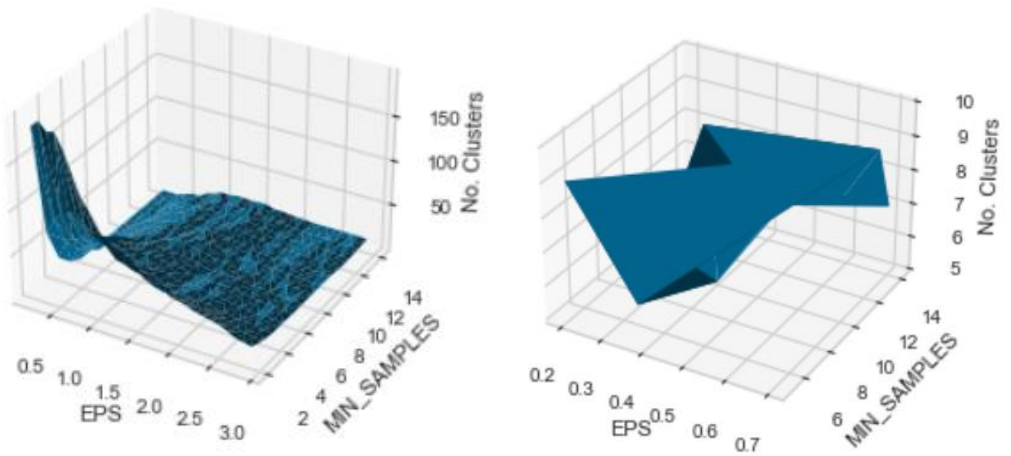
```
combinations = pd.DataFrame(list(zip(eps_list, ms_list, acc_list, num_clusters)),
                            columns =['Epsilon', 'MinSamples', 'Accuracy', 'Cluster'])

combinations_df_5_11 = combinations[combinations.Cluster.isin(range(5,11))]
com_10 = combinations_df_5_11.head(10)
com_10
```

The combination of parameters



	Epsilon	MinSamples	Accuracy	Cluster
0	0.2	5	0.117506	9
1	0.3	7	0.110312	5
2	0.4	9	0.171063	10
3	0.4	10	0.127898	5
4	0.5	11	0.193445	7
5	0.5	12	0.183853	7
6	0.6	13	0.240608	7
7	0.6	14	0.207834	7
8	0.7	13	0.298161	9
9	0.7	14	0.274181	7

3d Projection for all combinations on the left and the 10 combinations on the right.


No. Clusters
EPS
MIN_SAMPLES
No. Clusters
EPS
MIN_SAMPLES