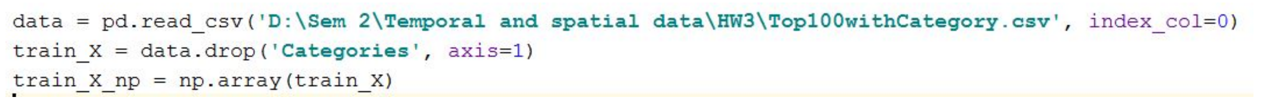
Tejal Bhangale

# **R** : Hands-on exercises

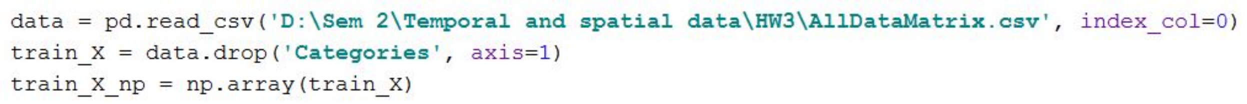
## 1. Use the same data matrix (after the feature selection step) as in Classification repository.

After feature selection - top 100 frequent words as features:
Converted previous hw Document Term Matrix (in R) into Top100withCategory.csv file



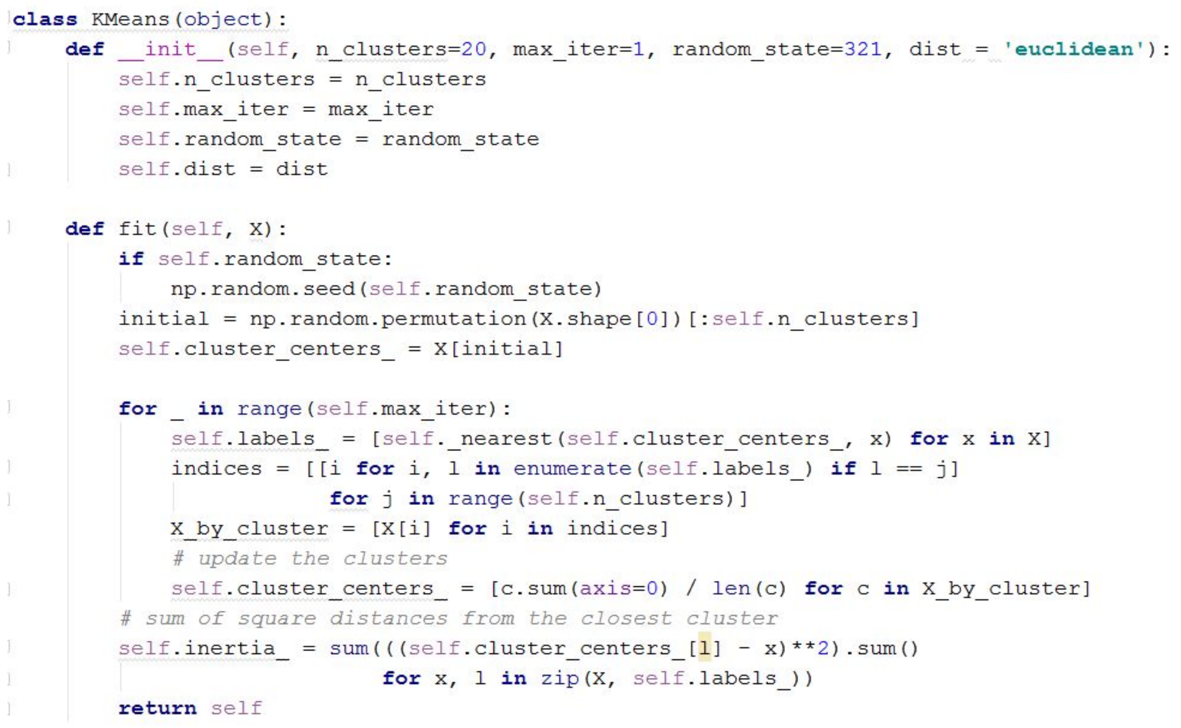
```
data = pd.read_csv('D:\Sem 2\Temporal and spatial data\HW3\Top100withCategory.csv', index_col=0)
train_X = data.drop('Categories', axis=1)
train_X_np = np.array(train_X)
```

Before feature selection - All the words from DTM as features:
Converted previous hw Document Term Matrix (in R) into Top100withCategory.csv file

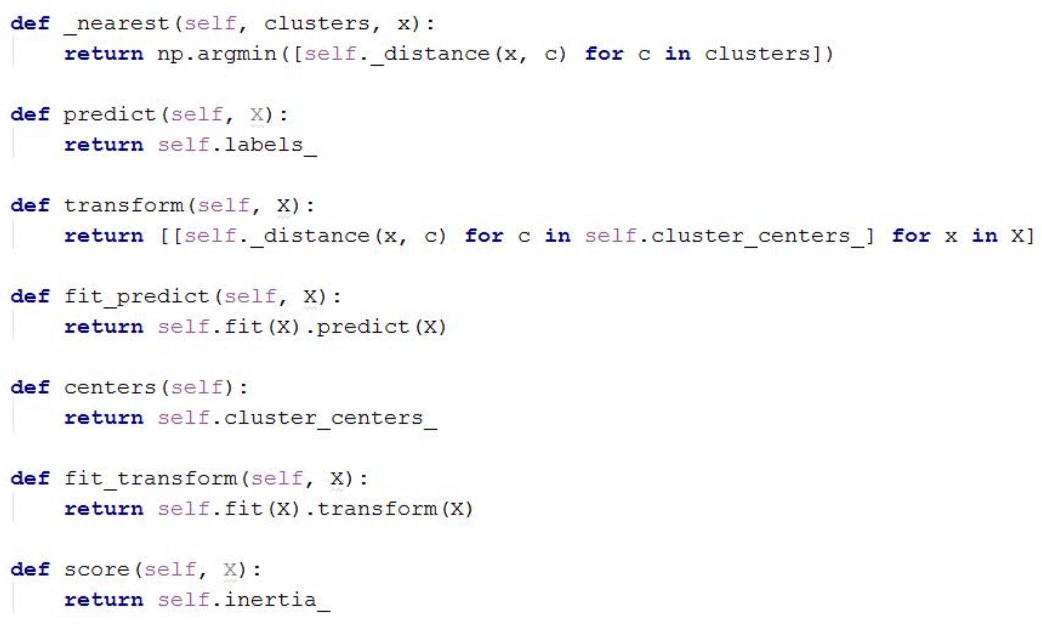


```
data = pd.read_csv('D:\Sem 2\Temporal and spatial data\HW3\AllDataMatrix.csv', index_col=0)
train_X = data.drop('Categories', axis=1)
train_X_np = np.array(train_X)
```

## 2. Implement the K-means algorithm.

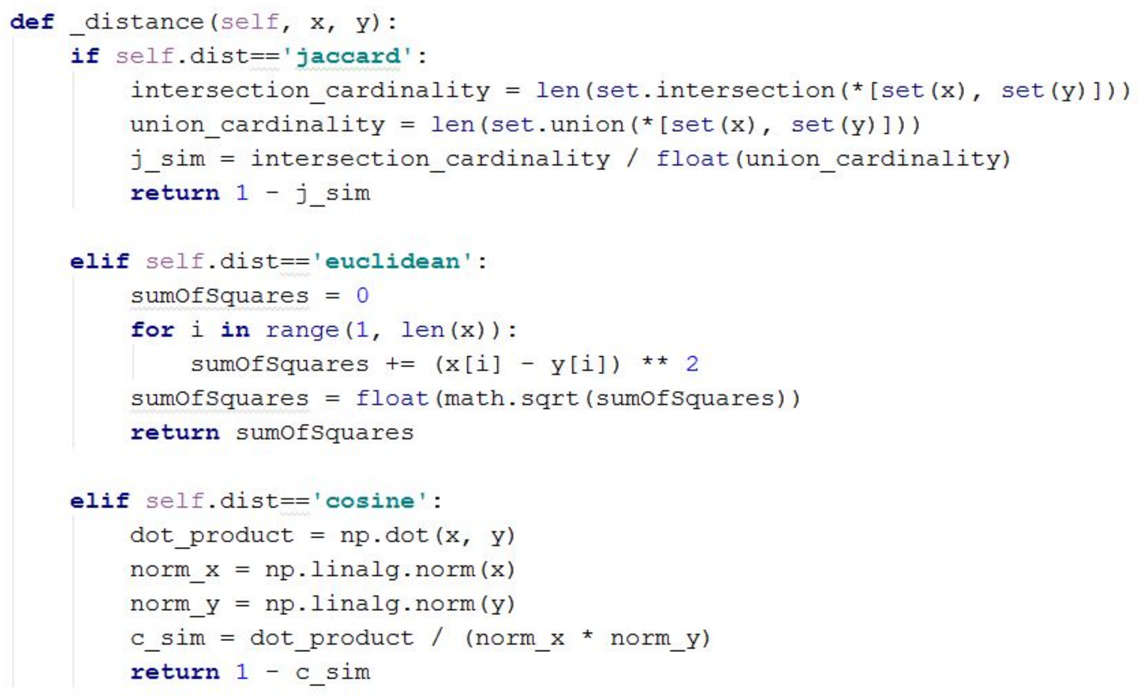


```
class KMeans(object):
    def __init__(self, n_clusters=20, max_iter=1, random_state=321, dist = 'euclidean'):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.random_state = random_state
        self.dist = dist

    def fit(self, X):
        if self.random_state:
            np.random.seed(self.random_state)
        initial = np.random.permutation(X.shape[0])[:self.n_clusters]
        self.cluster_centers_ = X[initial]

        for _ in range(self.max_iter):
            self.labels_ = [self._nearest(self.cluster_centers_, x) for x in X]
            indices = [[i for i, l in enumerate(self.labels_) if l == j]
                        for j in range(self.n_clusters)]
            X_by_cluster = [X[i] for i in indices]
            # update the clusters
            self.cluster_centers_ = [c.sum(axis=0) / len(c) for c in X_by_cluster]
        # sum of square distances from the closest cluster
        self.inertia_ = sum(((self.cluster_centers_[l] - x)**2).sum()
                            for x, l in zip(X, self.labels_))
        return self
```


```
def _nearest(self, clusters, x):
    return np.argmin([self._distance(x, c) for c in clusters])

def predict(self, X):
    return self.labels_

def transform(self, X):
    return [[self._distance(x, c) for c in self.cluster_centers_] for x in X]

def fit_predict(self, X):
    return self.fit(X).predict(X)

def centers(self):
    return self.cluster_centers_

def fit_transform(self, X):
    return self.fit(X).transform(X)

def score(self, X):
    return self.inertia_
```

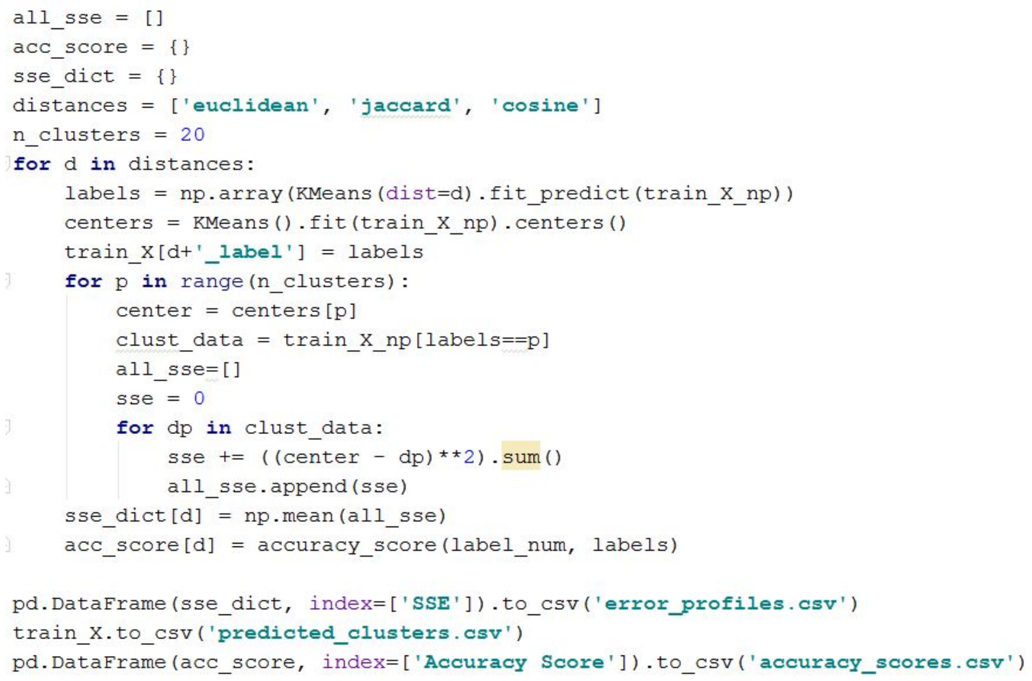
**3. K-means algorithm computes the distance of a given data point pair.**

Replace the computation function with Euclidean distance, 1- Cosine similarity, and 1 – Generalized Jaccard similarity. Implemented Euclidean, Jaccard and Cosine distance functions.


```
def _distance(self, x, y):
    if self.dist=='jaccard':
        intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
        union_cardinality = len(set.union(*[set(x), set(y)]))
        j_sim = intersection_cardinality / float(union_cardinality)
        return 1 - j_sim

    elif self.dist=='euclidean':
        sumOfSquares = 0
        for i in range(1, len(x)):
            sumOfSquares += (x[i] - y[i]) ** 2
        sumOfSquares = float(math.sqrt(sumOfSquares))
        return sumOfSquares

    elif self.dist=='cosine':
        dot_product = np.dot(x, y)
        norm_x = np.linalg.norm(x)
        norm_y = np.linalg.norm(y)
        c_sim = dot_product / (norm_x * norm_y)
        return 1 - c_sim
```

## 4. Run K-means clustering with Euclidean, Cosine and Jaccard similarity. (K = 20)

For the given input csv file, run k-means algorithm for all 3 distance metrics with number of clusters k = 20 (number of news article categories)

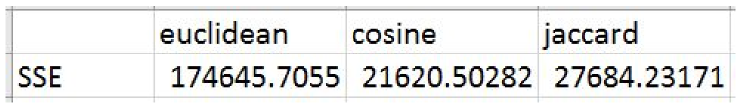


```
all_sse = []
acc_score = {}
sse_dict = {}
distances = ['euclidean', 'jaccard', 'cosine']
n_clusters = 20
for d in distances:
    labels = np.array(KMeans(dist=d).fit_predict(train_X_np))
    centers = KMeans().fit(train_X_np).centers()
    train_X[d+'_label'] = labels
    for p in range(n_clusters):
        center = centers[p]
        clust_data = train_X_np[labels==p]
        all_sse=[]
        sse = 0
        for dp in clust_data:
            sse += ((center - dp)**2).sum()
            all_sse.append(sse)
    sse_dict[d] = np.mean(all_sse)
    acc_score[d] = accuracy_score(label_num, labels)

pd.DataFrame(sse_dict, index=['SSE']).to_csv('error_profiles.csv')
train_X.to_csv('predicted_clusters.csv')
pd.DataFrame(acc_score, index=['Accuracy Score']).to_csv('accuracy_scores.csv')
```

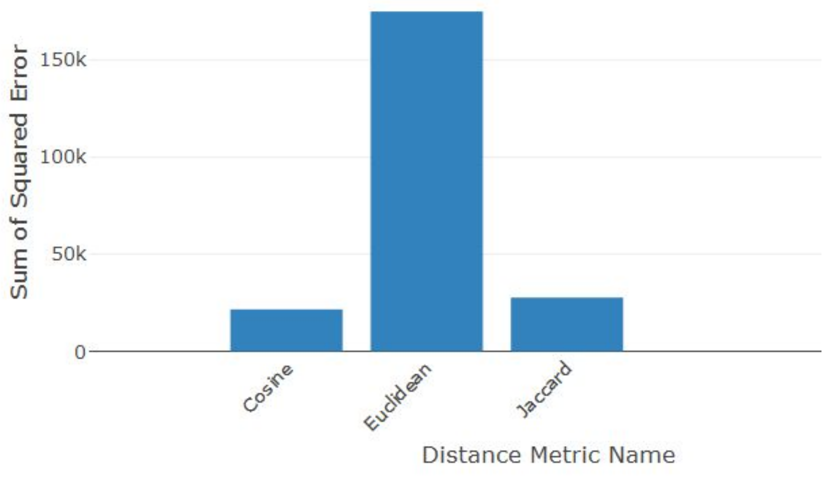
## 5. Compare the SSEs of Euclidean-K-means Cosine-K-means, Jaccard-K-means.

SSE is the sum of the squared differences between each observation and its group's mean. It can be used as a measure of variation within a cluster. If all cases within a cluster are identical the SSE would then be equal to 0. It means the smaller SSE, the better clustering.

Below is the comparison of SSE for each distance metric. According to it, Cosine k-means models better than other distance metrics as it has smallest Sum of squared error.

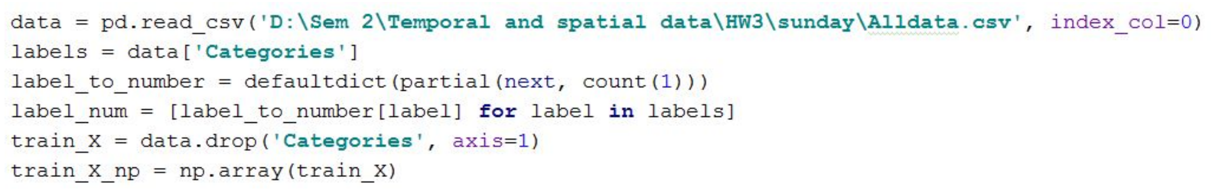


| | euclidean | cosine | jaccard |
|---|---|---|---|
| SSE | 174645.7055 | 21620.50282 | 27684.23171 |

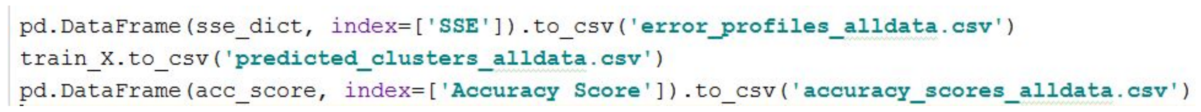


**6. Compare the accuracies of Euclidean-K-means Cosine-K-means, Jaccard-K-means.**

Implemented using `accuracy_score()` function of sklearn package.



```
data = pd.read_csv('D:\Sem 2\Temporal and spatial data\HW3\sunday\Alldata.csv', index_col=0)
labels = data['Categories']
label_to_number = defaultdict(partial(next, count(1)))
label_num = [label_to_number[label] for label in labels]
train_X = data.drop('Categories', axis=1)
train_X_np = np.array(train_X)
```

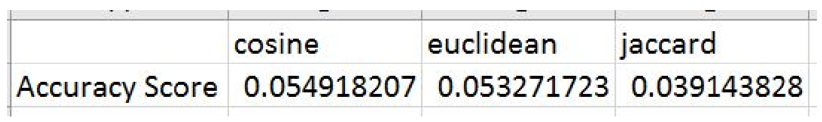


```
pd.DataFrame(sse_dict, index=['SSE']).to_csv('error_profiles_alldata.csv')
train_X.to_csv('predicted_clusters_alldata.csv')
pd.DataFrame(acc_score, index=['Accuracy Score']).to_csv('accuracy_scores_alldata.csv')
```

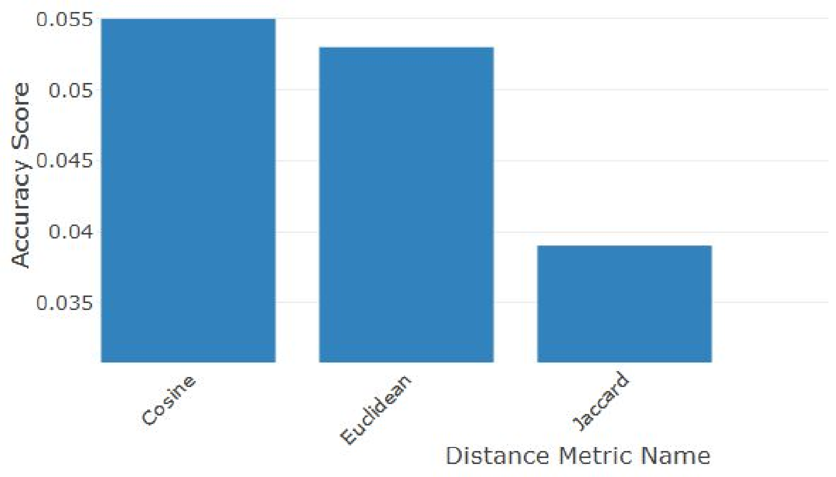
Accuracy of k-means is the metric to measure how good your clustering is in comparison to your known class labels. The higher the accuracy, the better clustering works.

Below is the comparison of Accuracy Score for each distance metric. According to accuracy score, Cosine k-means is better as it has highest accuracy score.

Accuracy score after feature selection:



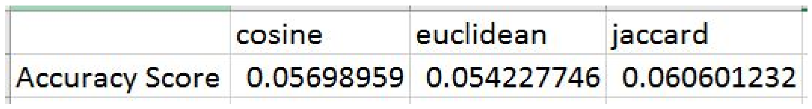
| | cosine | euclidean | jaccard |
|---|---|---|---|
| Accuracy Score | 0.054918207 | 0.053271723 | 0.039143828 |



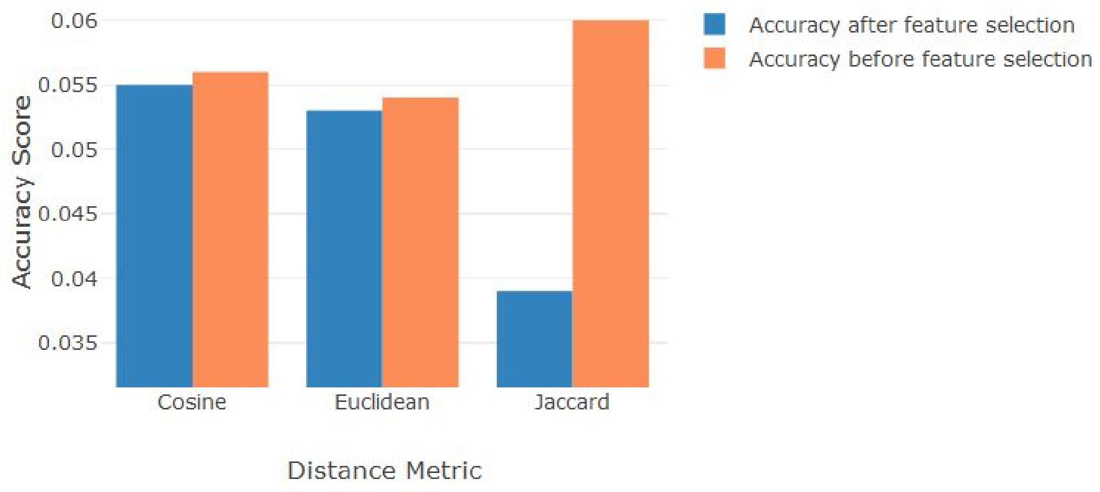
**7. Compare the accuracies (not SSE) of K-means before feature selection and after feature selection.**

Below is the comparison of Accuracy Score for each distance metric before and after feature Selection.

Accuracy score before feature selection:



| | cosine | euclidean | jaccard |
|---|---|---|---|
| Accuracy Score | 0.05698959 | 0.054227746 | 0.060601232 |



We can see that before feature selection, Jaccard k-means gives lower accuracy score. However, after feature selection Jaccard k-means gives higher accuracy score.

**8. Which of Euclidean-K-means, Cosine-K-means, Jaccard-K-means requires more iterations and times and why?**

In this implementation, Maximum Iterations limits the number of iterations in the k-means algorithm. Iteration stops after this many iterations even if the convergence criterion is not satisfied. This number must be between 1 and 999. To reproduce the algorithm, we need to set Maximum Iterations to 1.

Convergence Criterion determines when iteration ceases. It represents a proportion of the minimum distance between initial cluster centers, so it must be greater than 0 but not greater than 1. If the criterion equals 0.02, for example, iteration ceases when a complete iteration does not move any of the cluster centers by a distance of more than 2% of the smallest distance between any initial cluster centers.

Hence, it does not depend on distance metric such as Euclidean, Cosine and Jaccard k-means.

**9. Compare the SSEs of Euclidean-K-means Cosine-K-means, Jaccard-K-means.**

- **when there is no change in centroid position**
  When there is no change in centroid position, when we set maximum iterations to 1, the program will stop after first iteration.

- **when the SSE value increases in the next iteration**
  As we have set maximum iteration to 1, SSE value does not change in the next iteration.

- **when the maximum preset value (100) of iteration is complete**
  The SSEs of Euclidean k-means, Cosine k-means and Jaccard k-means will be same as we have set maximum iteration to 1 in this implementation.

Please find attached files with -

- Predicted cluster values for each distance metric
- SSE values for each distance metric
- Accuracy scores before and after feature selection for each distance metric