```
import pandas as pd
import numpy as np
import operator
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/ir
```

```
print(data)
```

```
     sepal_length  sepal_width  petal_length  petal_width           class
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

```
indices = np.random.permutation(data.shape[0])
div = int(0.75 * len(indices))
development_id, test_id = indices[:div], indices[div:]

development_set, test_set = data.loc[development_id,:], data.loc[test_id,:]
print("Development Set:\n", development_set, "\n\nTest Set:\n", test_set)
```

```
Development Set:
      sepal_length  sepal_width  petal_length  petal_width            class
44            5.1          3.8           1.9          0.4      Iris-setosa
37            4.9          3.1           1.5          0.1      Iris-setosa
140           6.7          3.1           5.6          2.4   Iris-virginica
19            5.1          3.8           1.5          0.3      Iris-setosa
28            5.2          3.4           1.4          0.2      Iris-setosa
..            ...          ...           ...          ...              ...
81            5.5          2.4           3.7          1.0  Iris-versicolor
78            6.0          2.9           4.5          1.5  Iris-versicolor
70            5.9          3.2           4.8          1.8  Iris-versicolor
10            5.4          3.7           1.5          0.2      Iris-setosa
90            5.5          2.6           4.4          1.2  Iris-versicolor

[112 rows x 5 columns]

Test Set:
      sepal_length  sepal_width  petal_length  petal_width            class
100           6.3          3.3           6.0          2.5   Iris-virginica
74            6.4          2.9           4.3          1.3  Iris-versicolor
15            5.7          4.4           1.5          0.4      Iris-setosa
141           6.9          3.1           5.1          2.3   Iris-virginica
```

139	6.9	3.1	5.4	2.1	Iris-virginica
33	5.5	4.2	1.4	0.2	Iris-setosa
61	5.9	3.0	4.2	1.5	Iris-versicolor
120	6.9	3.2	5.7	2.3	Iris-virginica
40	5.0	3.5	1.3	0.3	Iris-setosa
89	5.5	2.5	4.0	1.3	Iris-versicolor
113	5.7	2.5	5.0	2.0	Iris-virginica
95	5.7	3.0	4.2	1.2	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
59	5.2	2.7	3.9	1.4	Iris-versicolor
128	6.4	2.8	5.6	2.1	Iris-virginica
94	5.6	2.7	4.2	1.3	Iris-versicolor
5	5.4	3.9	1.7	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
66	5.6	3.0	4.5	1.5	Iris-versicolor
92	5.8	2.6	4.0	1.2	Iris-versicolor
34	4.9	3.1	1.5	0.1	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
119	6.0	2.2	5.0	1.5	Iris-virginica
109	7.2	3.6	6.1	2.5	Iris-virginica
14	5.8	4.0	1.2	0.2	Iris-setosa
127	6.1	3.0	4.9	1.8	Iris-virginica
87	6.3	2.3	4.4	1.3	Iris-versicolor
24	4.8	3.4	1.9	0.2	Iris-setosa
55	5.7	2.8	4.5	1.3	Iris-versicolor
84	5.4	3.0	4.5	1.5	Iris-versicolor
107	7.3	2.9	6.3	1.8	Iris-virginica

```
mean_development_set = development_set.mean()
mean_test_set = test_set.mean()
std_development_set = development_set.std()
std_test_set = test_set.std()


test_class = list(test_set.iloc[:,-1])
dev_class = list(development_set.iloc[:,-1])


def euclideanDistance(data_1, data_2, data_len):
    dist = 0
    for i in range(data_len):
        dist = dist + np.square(data_1[i] - data_2[i])
    return np.sqrt(dist)

def normalizedEuclideanDistance(data_1, data_2, data_len, data_mean, data_std):
    n_dist = 0
    for i in range(data_len):
        n_dist = n_dist + (np.square(((data_1[i] - data_mean[i])/data_std[i]) -
    return np.sqrt(n_dist)
```

```
        for x in range(len(dataset)):
            dist_up = euclideanDistance(testInstance, dataset.iloc[x], length)
            distances[x] = dist_up[0]
    elif dist_method == 'normalized_euclidean':
        for x in range(len(dataset)):
            dist_up = normalizedEuclideanDistance(testInstance, dataset.iloc[x],
            distances[x] = dist_up[0]
    elif dist_method == 'cosine':
        for x in range(len(dataset)):
            dist_up = cosineSimilarity(testInstance, dataset.iloc[x])
            distances[x] = dist_up[0]
    # Sort values based on distance
    sort_distances = sorted(distances.items(), key=operator.itemgetter(1))
    neighbors = []
    # Extracting nearest k neighbors
    for x in range(k):
        neighbors.append(sort_distances[x][0])
    # Initializing counts for 'class' labels counts as 0
    counts = {"Iris-setosa" : 0, "Iris-versicolor" : 0, "Iris-virginica" : 0}
    # Computing the most frequent class
    for x in range(len(neighbors)):
        response = dataset.iloc[neighbors[x]][-1]
        if response in counts:
            counts[response] += 1
        else:
            counts[response] = 1
    # Sorting the class in reverse order to get the most frequest class
    sort_counts = sorted(counts.items(), key=operator.itemgetter(1), reverse=Tru
    return(sort_counts[0][0])


# Creating a list of list of all columns except 'class' by iterating through the
row_list = []
for index, rows in development_set.iterrows():
    my_list =[rows.sepal_length, rows.sepal_width, rows.petal_length, rows.petal
    row_list.append([my_list])
# k values for the number of neighbors that need to be considered
k_n = [3, 5, 7]
# Distance metrics
distance_methods = ['euclidean', 'normalized_euclidean', 'cosine']
# Performing kNN on the development set by iterating all of the development set
obs_k = {}
for dist_method in distance_methods:
```

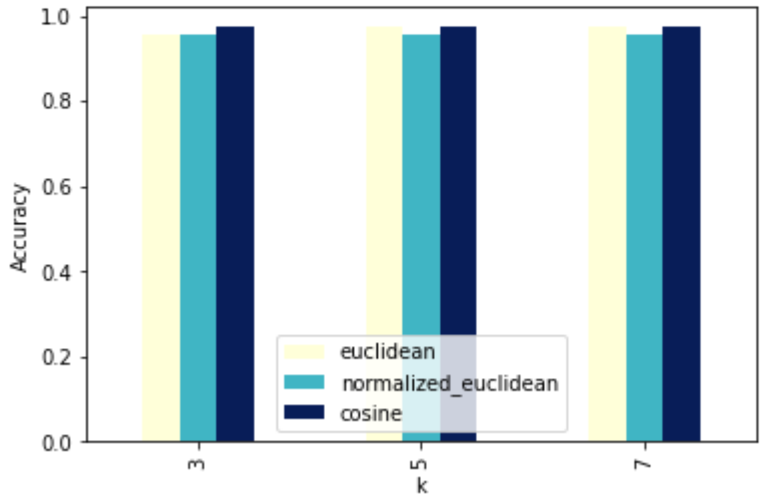
```
        for i,j in zip(dev_class, obs_k[key][k_value]):
            if i == j:
                count = count + 1
            else:
                pass
        accuracy[key][k_value] = count/(len(dev_class))

# Storing the accuracy for each k and each distance metric into a dataframe
df_res = pd.DataFrame({'k': k_n})
for key in accuracy.keys():
    value = list(accuracy[key].values())
    df_res[key] = value
print(df_res)
```

```
   k  euclidean  normalized_euclidean    cosine
0  3   0.955357              0.955357  0.973214
1  5   0.973214              0.955357  0.973214
2  7   0.973214              0.955357  0.973214
```

```
# Plotting a Bar Chart for accuracy
draw = df_res.plot(x='k', y=['euclidean', 'normalized_euclidean', 'cosine'], kin
draw.set(ylabel='Accuracy')
```

```
[Text(0, 0.5, 'Accuracy')]
```



```
for index, rows in test_set.iterrows():
    my_list =[rows.sepal_length, rows.sepal_width, rows.petal_length, rows.petal
    row_list_test.append([my_list])
test_set_obs = []
for i in range(len(row_list_test)):
    test_set_obs.append(knn(test_set, pd.DataFrame(row_list_test[i]), best_k, be
#print(test_set_obs)

count = 0
for i,j in zip(test_class, test_set_obs):
    if i == j:
        count = count + 1
    else:
        pass
accuracy_test = count/(len(test_class))
print('Final Accuracy of the Test dataset is ', accuracy_test)
```

```
Final Accuracy of the Test dataset is  0.9736842105263158
```