KMeans Clustering Implementation

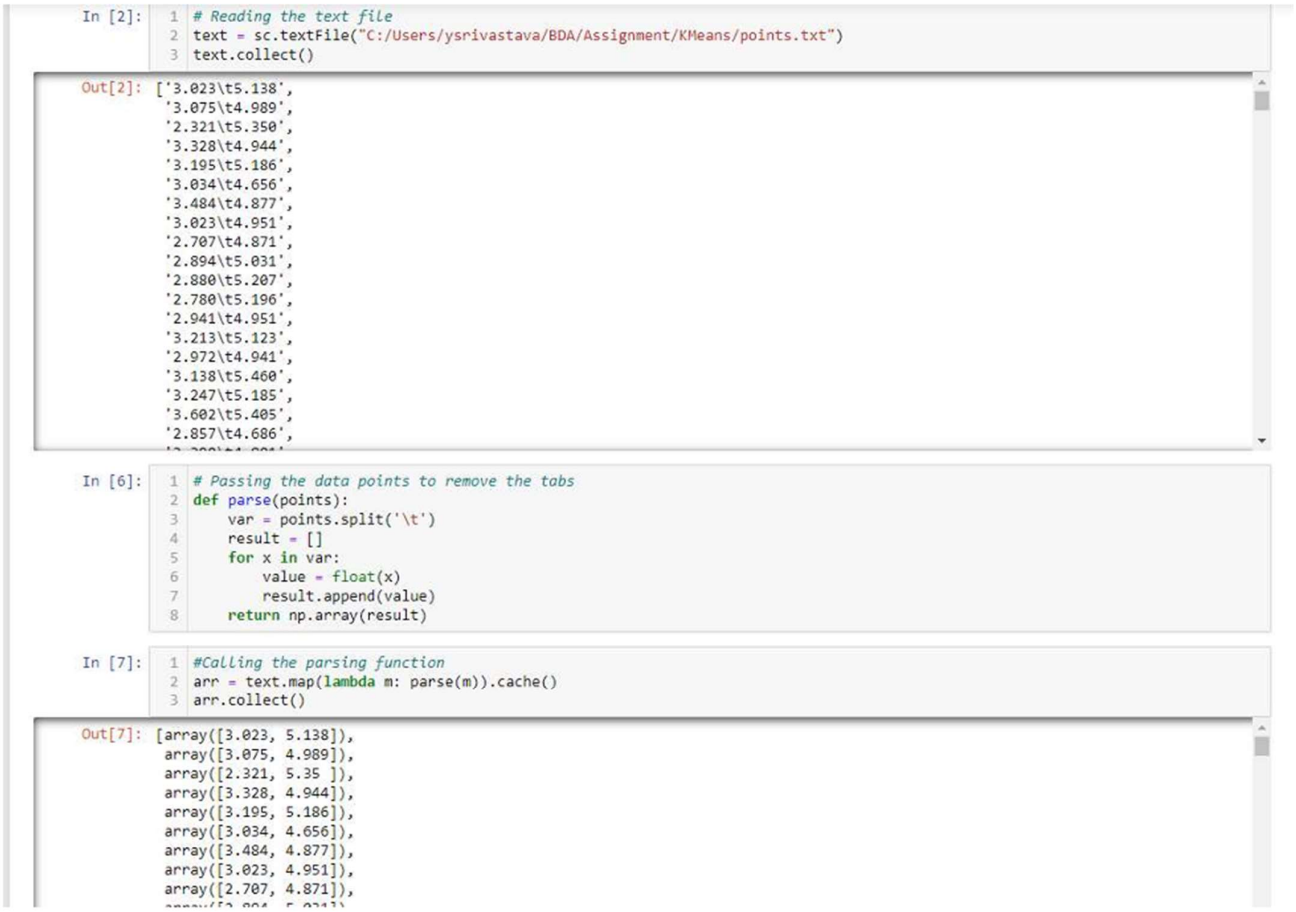
Technology/Concept:

- Distributed Machine Learning: Apache Spark (MLlib) can be used for large-scale data processing and machine learning tasks like KMeans clustering, enabling efficient handling of large datasets across distributed clusters.
- Hadoop: For large datasets, Hadoop with MapReduce can be used to parallelize the KMeans computation.
- Data Storage: Large input datasets can be stored in HDFS, Amazon S3, or other distributed storage systems for efficient access and processing.

Steps:

1. First step to read the info files.
2. Then parse the info.
3. Initialize the K clusters.
4. Now perform iteration until converse.
5. Get nearest points mapper.
6. Combined the mapper results using reducer.
7. Update the centroids.
8. Check the space between points and centroid.
9. After converse results as follows.

Output:



```
In [2]:
# Reading the text file
text = sc.textFile("C:/Users/ysrivastava/BDA/Assignment/KMeans/points.txt")
text.collect()
```

```
Out[2]: ['3.023\t5.138',
 '3.075\t4.989',
 '2.321\t5.350',
 '3.328\t4.944',
 '3.195\t5.186',
 '3.034\t4.656',
 '3.484\t4.877',
 '3.023\t4.951',
 '2.707\t4.871',
 '2.894\t5.031',
 '2.880\t5.207',
 '2.780\t5.196',
 '2.941\t4.951',
 '3.213\t5.123',
 '2.972\t4.941',
 '3.138\t5.460',
 '3.247\t5.185',
 '3.602\t5.405',
 '2.857\t4.686',
```

```
In [6]:
# Passing the data points to remove the tabs
def parse(points):
    var = points.split('\t')
    result = []
    for x in var:
        value = float(x)
        result.append(value)
    return np.array(result)
```

```
In [7]:
#Calling the parsing function
arr = text.map(lambda m: parse(m)).cache()
arr.collect()
```

```
Out[7]: [array([3.023, 5.138]),
 array([3.075, 4.989]),
 array([2.321, 5.35 ]),
 array([3.328, 4.944]),
 array([3.195, 5.186]),
 array([3.034, 4.656]),
 array([3.484, 4.877]),
 array([3.023, 4.951]),
 array([2.707, 4.871]),
```

In [18]:
```
Dist = float(0.1)
kp = [np.array([float(2) , float(5)]), np.array([float(6), float(2)]), np.array([float(1), float(1)])]
dist2 = 1.0
type(kp)
print(kp)
```

```
[array([2., 5.]), array([6., 2.]), array([1., 1.])]
```

In [21]:
```
nearest = arr.map(lambda p: process_map(p, kp))
print(nearest.take(5))
```

```
[(0, (array([3.023, 5.138]), 1)), (0, (array([3.075, 4.989]), 1)), (0, (array([2.321, 5.35 ]), 1)), (0, (array([3.328, 4.944]),
1)), (0, (array([3.195, 5.186]), 1))]
```

In [23]:
```
Status = nearest.reduceByKey(lambda pointer1 , pointer2: pairs(pointer1 , pointer2))
print(Status.take(4))
```

```
[(0, (array([1496.991, 2502.065]), 500)), (2, (array([1016.494,  501.689]), 500)), (1, (array([2496.329,  995.104]), 500))]
```

In [24]:
```
new_points = Status.map(lambda map_object: mapper(map_object)).collect()
print(new_points)
```

```
[(0, array([2.993982, 5.00413 ])), (2, array([2.032988, 1.003378])), (1, array([4.992658, 1.990208]))]
```

In [25]:
```
updated_distance = all_points_dist(new_points)
print(updated_distance)
```

```
51.92141868048006
```

In [27]:
```
def change_points(new_points, kp):
    for (cluster_number, point) in new_points:
        kp[cluster_number] = point
    return kp

change_points(new_points, kp)
print(kp)
```

```
[array([2.993982, 5.00413 ]), array([4.992658, 1.990208]), array([2.032988, 1.003378])]
```

In [30]:
```
while dist2 > Dist:
    print(dist2)
    near = arr.map(lambda p: process_map(p, kp))
    point_Stats = nearest.reduceByKey(lambda pointer1 , pointer2: pairs(pointer1 , pointer2))
    new_Points = point_Stats.map(lambda map_object: mapper(map_object)).collect()
    print(new_Points)
    dist2 = sum(np.sum((kp[iK] - p) ** 2) for (iK, p) in new_Points)

    for (iK, p) in new_Points:
        kp[iK] = p

results(kp)
```

```
1.0
[(0, array([2.993982, 5.00413 ])), (2, array([2.032988, 1.003378])), (1, array([4.992658, 1.990208]))]
2.99398#5.00413
4.99266#1.99021
2.03299#1.00338
```