First iteration of K-Means clustering algorithm using MapReduce

Assignment 2

Panourgia Evangelia (t8190130)
Papadatos Ioannis (t8190314)
Professor: Chatziantoniou Damianos

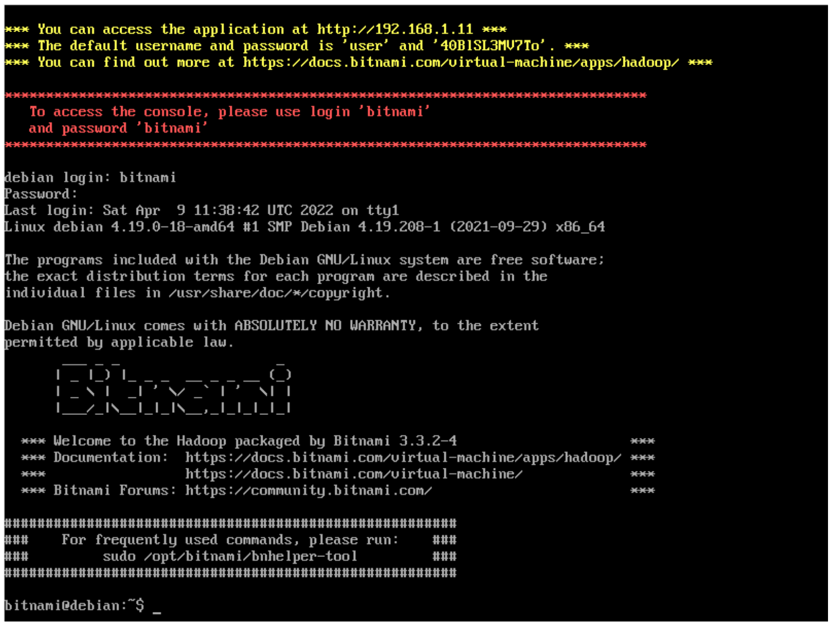
Latest Update: April 14, 2022

School of Management Science and Technology,
Athens University of Economics and Business

Contents

1 Hadoop Installation

As far as the installation of Hadoop, we decided to use Hadoop through a Virtual Machine. We chose to download Bitnami's Virtual Machine image for Hadoop [1], because it is lightweight and easy to use. Then, we launched a Virtual Machine by importing the downloaded image to the Oracle VirtualBox [2] virualization application. Once the startup process was completed, we logged in using the login prompt shown in the image below in order to access the console. Afterwards, we installed git (command: sudo apt install git) at the Virtual Machine through the console, in order to clone our Github repository containing the code. Finally, we also installed pip3 (command: sudo apt install python3-pip) in order to install the dependencies used in our code which are listed in the requirements.txt file (command: pip3 install -r requirements.txt).

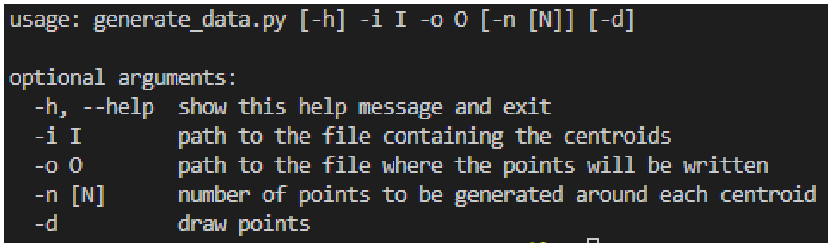


```
*** You can access the application at http://192.168.1.11 ***
*** The default username and password is 'user' and '40B1SL3MV7To'. ***
*** You can find out more at https://docs.bitnami.com/virtual-machine/apps/hadoop/ ***

******************************************************************************
   To access the console, please use login 'bitnami'
   and password 'bitnami'
******************************************************************************

debian login: bitnami
Password:
Last login: Sat Apr  9 11:38:42 UTC 2022 on tty1
Linux debian 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
       ___ _ _                   _
      | _ |_) |_ _ _  __ _ _ __ (_)
      | _ \ |  _| ' \/ _` | '  \| |
      |___/_|\__|_|_|\__,_|_|_|_|_|

  *** Welcome to the Hadoop packaged by Bitnami 3.3.2-4                    ***
  *** Documentation:  https://docs.bitnami.com/virtual-machine/apps/hadoop/ ***
  ***                 https://docs.bitnami.com/virtual-machine/             ***
  *** Bitnami Forums: https://community.bitnami.com/                        ***

#######################################################
###    For frequently used commands, please run:    ###
###         sudo /opt/bitnami/bnhelper-tool         ###
#######################################################

bitnami@debian:~$ _
```

Figure 1: Install Bitnami

2 Dataset Creation

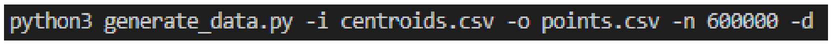
The script generate_data.py was written for generating data-points in the form (x, y). Specifically, it reads centroids in the form (x, y) from a csv file (-i) and generates the specified amount of data-points around each centroid (-n), which is then written to the specified file (-o). The argument -d is optional, but if provided it generates the graphical representation of the points in an image (points.png).



```
usage: generate_data.py [-h] -i I -o O [-n [N]] [-d]

optional arguments:
  -h, --help  show this help message and exit
  -i I        path to the file containing the centroids
  -o O        path to the file where the points will be written
  -n [N]      number of points to be generated around each centroid
  -d          draw points
```

Figure 2: Script arguments.

For our assignment needs, the script described above was used with the command shown below to read three centroids from the centroids.csv file and generate 600.000 data-points around each one of them, that was then written to the points.csv file. Also the points.png image was produced since the argument -d was provided.



```
python3 generate_data.py -i centroids.csv -o points.csv -n 600000 -d
```

Figure 3: Dataset generation command.

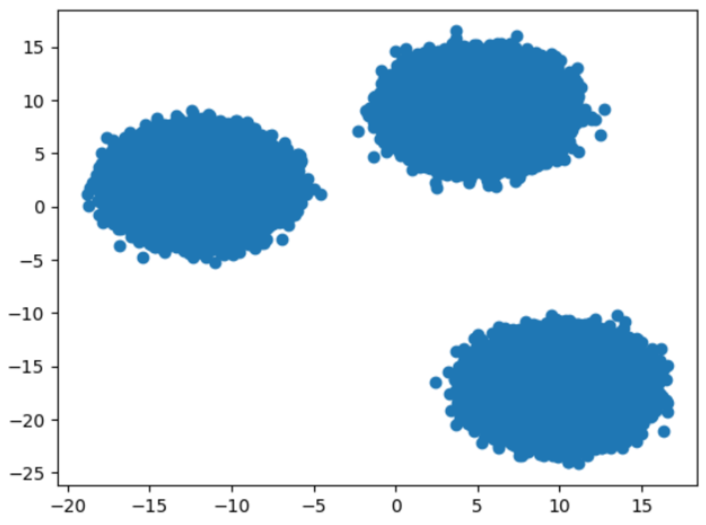
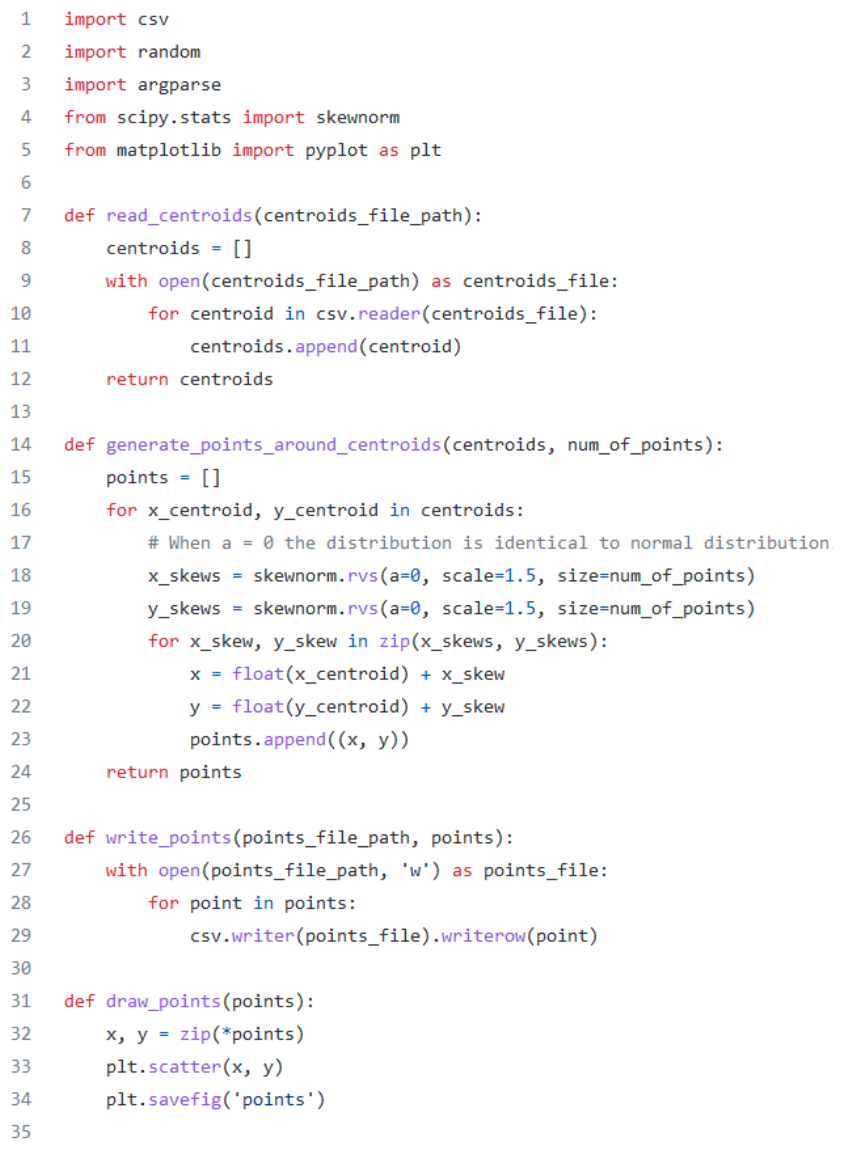
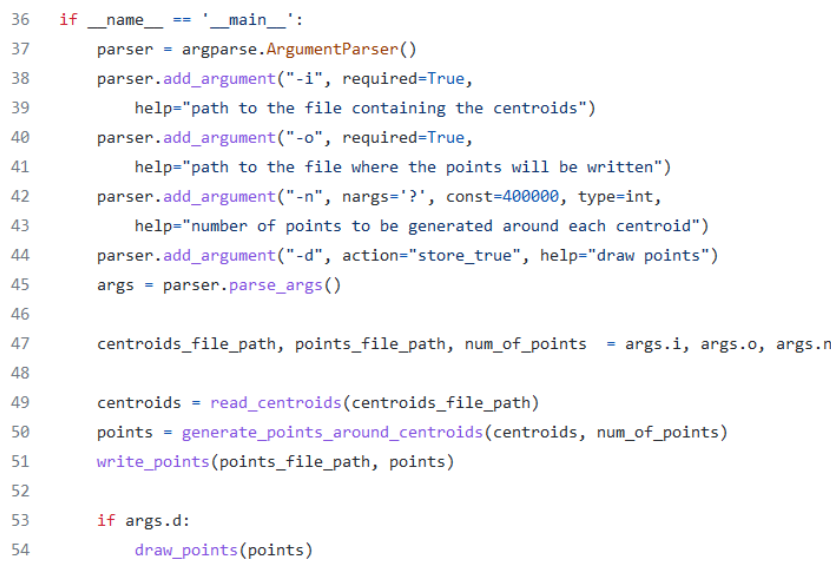


Figure 4: Dataset graphical representation.



```python
import csv
import random
import argparse
from scipy.stats import skewnorm
from matplotlib import pyplot as plt

def read_centroids(centroids_file_path):
    centroids = []
    with open(centroids_file_path) as centroids_file:
        for centroid in csv.reader(centroids_file):
            centroids.append(centroid)
    return centroids

def generate_points_around_centroids(centroids, num_of_points):
    points = []
    for x_centroid, y_centroid in centroids:
        # When a = 0 the distribution is identical to normal distribution
        x_skews = skewnorm.rvs(a=0, scale=1.5, size=num_of_points)
        y_skews = skewnorm.rvs(a=0, scale=1.5, size=num_of_points)
        for x_skew, y_skew in zip(x_skews, y_skews):
            x = float(x_centroid) + x_skew
            y = float(y_centroid) + y_skew
            points.append((x, y))
    return points

def write_points(points_file_path, points):
    with open(points_file_path, 'w') as points_file:
        for point in points:
            csv.writer(points_file).writerow(point)

def draw_points(points):
    x, y = zip(*points)
    plt.scatter(x, y)
    plt.savefig('points')

```

Figure 5: generate_data.py source code.



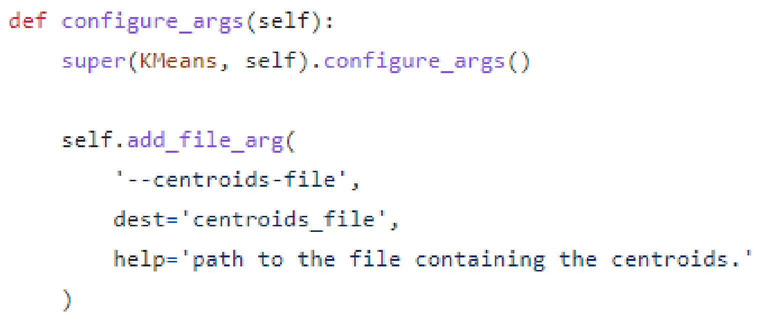
```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", required=True,
        help="path to the file containing the centroids")
    parser.add_argument("-o", required=True,
        help="path to the file where the points will be written")
    parser.add_argument("-n", nargs='?', const=400000, type=int,
        help="number of points to be generated around each centroid")
    parser.add_argument("-d", action="store_true", help="draw points")
    args = parser.parse_args()

    centroids_file_path, points_file_path, num_of_points  = args.i, args.o, args.n

    centroids = read_centroids(centroids_file_path)
    points = generate_points_around_centroids(centroids, num_of_points)
    write_points(points_file_path, points)

    if args.d:
        draw_points(points)
```

Figure 6: generate_data.py source code.

3 K-Means Clustering Algorithm

The script kmeans.py was written to implement the first iteration of the K-Means clustering algorithm [6] using MapReduce. To write the MapReduce job we used the Python library mrjob [3].

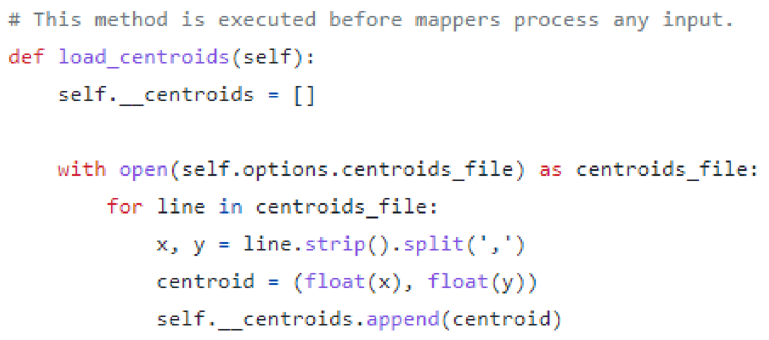
The configure_args method is used to add a file argument that allows the user to specify the path to the file containing the initial centroids:



```
def configure_args(self):
    super(KMeans, self).configure_args()

    self.add_file_arg(
        '--centroids-file',
        dest='centroids_file',
        help='path to the file containing the centroids.'
    )
```

Figure 7: configure_args method

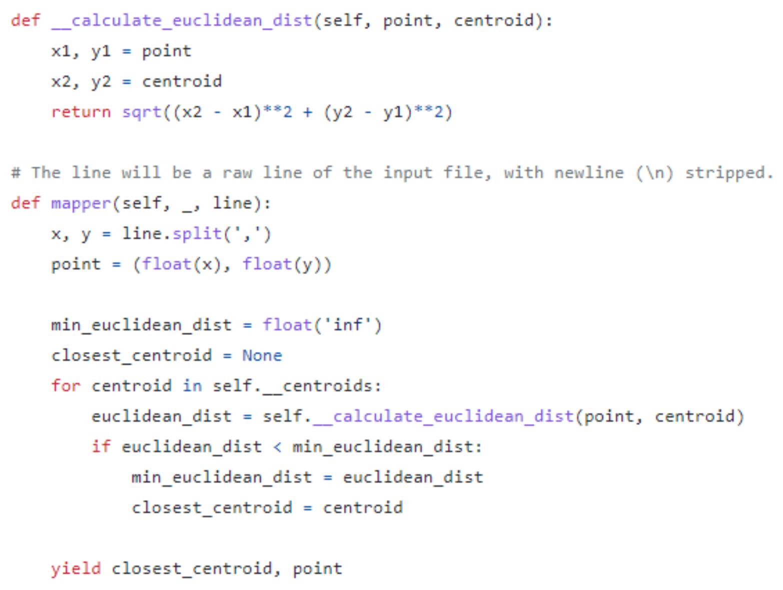
The load_centroids method is used to broadcast the initial centroids to the mappers:



```
# This method is executed before mappers process any input.
def load_centroids(self):
    self.__centroids = []

    with open(self.options.centroids_file) as centroids_file:
        for line in centroids_file:
            x, y = line.strip().split(',')
            centroid = (float(x), float(y))
            self.__centroids.append(centroid)
```

Figure 8: load_centroids method

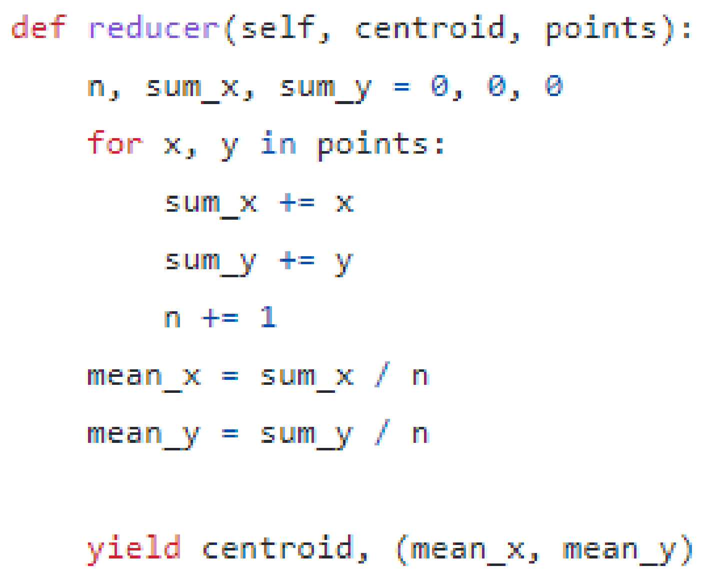
The mapper method is used to map each point to its closest centroid based on euclidean distance:



```
def __calculate_euclidean_dist(self, point, centroid):
    x1, y1 = point
    x2, y2 = centroid
    return sqrt((x2 - x1)**2 + (y2 - y1)**2)

# The line will be a raw line of the input file, with newline (\n) stripped.
def mapper(self, _, line):
    x, y = line.split(',')
    point = (float(x), float(y))

    min_euclidean_dist = float('inf')
    closest_centroid = None
    for centroid in self.__centroids:
        euclidean_dist = self.__calculate_euclidean_dist(point, centroid)
        if euclidean_dist < min_euclidean_dist:
            min_euclidean_dist = euclidean_dist
            closest_centroid = centroid

    yield closest_centroid, point
```

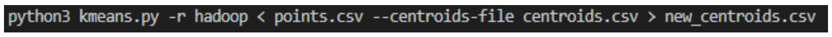
Figure 9: mapper method

The reducer method is used to compute the new centroids based on the data points that were mapped to each centroid:



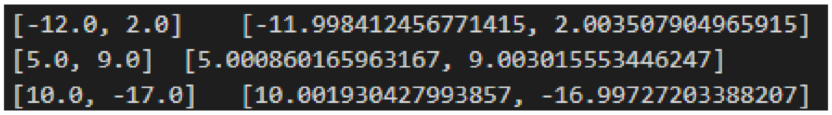
```
def reducer(self, centroid, points):
    n, sum_x, sum_y = 0, 0, 0
    for x, y in points:
        sum_x += x
        sum_y += y
        n += 1
    mean_x = sum_x / n
    mean_y = sum_y / n

    yield centroid, (mean_x, mean_y)
```

Figure 10: reducer method



```
python3 kmeans.py -r hadoop < points.csv --centroids-file centroids.csv > new_centroids.csv
```

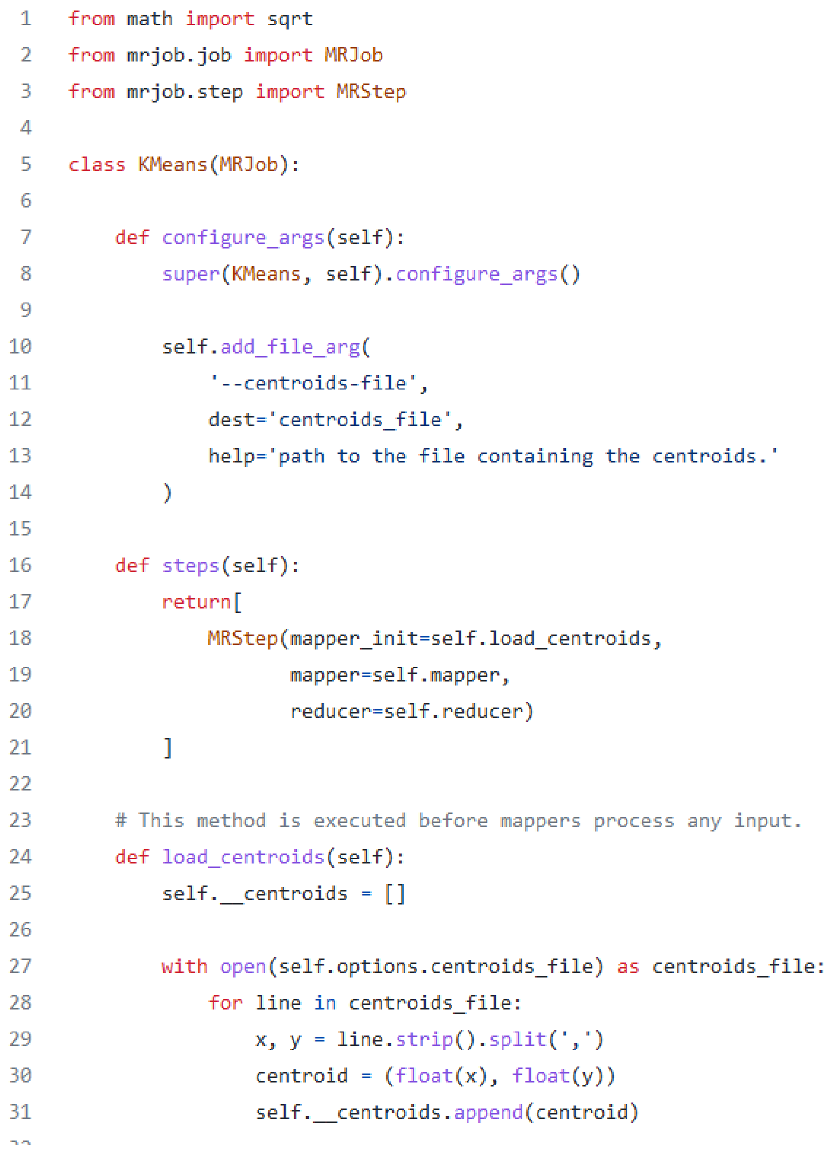
Figure 11: MapReduce job execution command



```
[-12.0, 2.0]    [-11.998412456771415, 2.003507904965915]
[5.0, 9.0]  [5.000860165963167, 9.003015553446247]
[10.0, -17.0]   [10.001930427993857, -16.99727203388207]
```

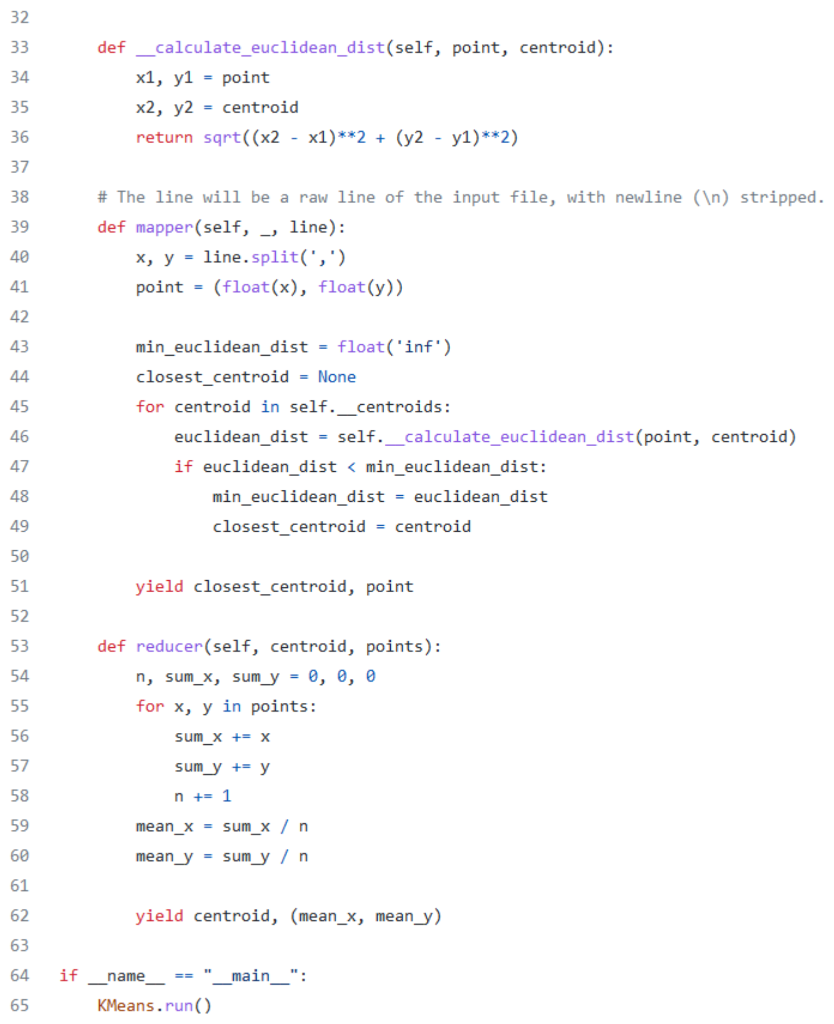
Figure 12: MapReduce job results

We observe that the deviation between the initial and the new centroids is small, which is normal since the data points were generated around the initial centroids using normal distribution.



```python
from math import sqrt
from mrjob.job import MRJob
from mrjob.step import MRStep

class KMeans(MRJob):

    def configure_args(self):
        super(KMeans, self).configure_args()

        self.add_file_arg(
            '--centroids-file',
            dest='centroids_file',
            help='path to the file containing the centroids.'
        )

    def steps(self):
        return[
            MRStep(mapper_init=self.load_centroids,
                   mapper=self.mapper,
                   reducer=self.reducer)
        ]

    # This method is executed before mappers process any input.
    def load_centroids(self):
        self.__centroids = []

        with open(self.options.centroids_file) as centroids_file:
            for line in centroids_file:
                x, y = line.strip().split(',')
                centroid = (float(x), float(y))
                self.__centroids.append(centroid)
```

Figure 13: kmeans.py source code.



```python

    def __calculate_euclidean_dist(self, point, centroid):
        x1, y1 = point
        x2, y2 = centroid
        return sqrt((x2 - x1)**2 + (y2 - y1)**2)

    # The line will be a raw line of the input file, with newline (\n) stripped.
    def mapper(self, _, line):
        x, y = line.split(',')
        point = (float(x), float(y))

        min_euclidean_dist = float('inf')
        closest_centroid = None
        for centroid in self.__centroids:
            euclidean_dist = self.__calculate_euclidean_dist(point, centroid)
            if euclidean_dist < min_euclidean_dist:
                min_euclidean_dist = euclidean_dist
                closest_centroid = centroid

        yield closest_centroid, point

    def reducer(self, centroid, points):
        n, sum_x, sum_y = 0, 0, 0
        for x, y in points:
            sum_x += x
            sum_y += y
            n += 1
        mean_x = sum_x / n
        mean_y = sum_y / n

        yield centroid, (mean_x, mean_y)

if __name__ == "__main__":
    KMeans.run()
```

Figure 14: kmeans.py source code.

References

[1] Hadoop packaged by Bitnami, ***https://bitnami.com/stack/hadoop/virtual-machine***

[2] Oracle VirtualBox, ***https://www.virtualbox.org/***

[3] mrjob, ***https://pypi.org/project/mrjob/***

[4] skewnorm, ***https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.skewnorm.html***

[5] matplotlib, ***https://matplotlib.org/***

[6] MapReduce Algorithms for k-means Clustering , title = MapReduce Algorithms for k-means Clustering, author = Max Bodoia, note = https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/bodoia.pdf, note = Accessed: 10-4-2022,