

# Big Data Analytics - BD

*Stefano Toffol (steto820) and Nahid Farazmand (nahfa911)*

*07 May, 2019*

## Assignment 1

What are the lowest and highest temperatures measured each year for the period 1950-2014? Provide the lists sorted in the descending order with respect to the maximum temperature. In this exercise you will use the temperature-readings.csv file.

Code:

```
# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
splited_lines = tempreture_readings.map(lambda line: line.split(';'))

# extracing year and temp from each line as a tuple
extracted = splited_lines.map(lambda x: (int(x[1][0:4]),float(x[3])))
extracted = extracted.filter(lambda x: x[0]>=1950 and x[0]<=2014)

# maximum for each year (year is the key)
maximum_temps = extracted.reduceByKey(max)
maximum_temps = maximum_temps.sortBy(lambda x: x[1], ascending = False)

# minimum for each year (year is the key)
minimum_temps = extracted.reduceByKey(min)
minimum_temps = minimum_temps.sortBy(lambda x: x[1], ascending = False)

minimum_temps.saveAsTextFile('./results/task01_min')
maximum_temps.saveAsTextFile('./results/task01_max')
```

Result (maximum temperature):

```
[x.sttofo@heffal task01_max]$ head -20 all_task01_max.txt
(1975, 36.1)
(1992, 35.4)
(1994, 34.7)
(2014, 34.4)
(2010, 34.4)
(1989, 33.9)
(1982, 33.8)
(1968, 33.7)
(1966, 33.5)
(2002, 33.3)
(1983, 33.3)
(1986, 33.2)
(1970, 33.2)
(1956, 33.0)
(2000, 33.0)
(1959, 32.8)
(2006, 32.7)
(1991, 32.7)
(1988, 32.6)
(2011, 32.5)
```

Result (minimum temperature):

```
x sttof@heffal task01_min]$ head -20 all_task01_min.txt
(1990, -35.0)
(1952, -35.5)
(1974, -35.6)
(1954, -36.0)
(1992, -36.1)
(1975, -37.0)
(1972, -37.5)
(1995, -37.6)
(2000, -37.6)
(1957, -37.8)
(1989, -38.2)
(1983, -38.2)
(1953, -38.4)
(2009, -38.5)
(1993, -39.0)
(1984, -39.2)
(1991, -39.3)
(2008, -39.3)
(1973, -39.3)
(2005, -39.4)
```

Extend the program to include the station number (not the station name) where the maximum/minimum temperature was measured.

Code:

```
# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
splited_lines = tempreture_readings.map(lambda line: line.split(';'))

# Find temperature and station with max temperature per year
# We will organise the values in tuples (station_code, temp)
years_temp_station = splited_lines.map(lambda x: (int(x[1][0:4]), (int(x[0]),float(x[3])))) )
years_temp_station = years_temp_station.filter(lambda x: x[0]>=1950 and x[0]<=2014)

# Select max according to temperature only
max_station_temp = years_temp_station.reduceByKey(lambda x, y: (x[0] if x[1] >= y[1] else y[0], max(x[1], y[1])))

# Order according to temperature
sorted_max_station_temp = max_station_temp.sortBy(lambda a: a[1][1], ascending=False)

# Find temperature and station with min temperature per year
# Select min according to temperature only
min_station_temp = years_temp_station.reduceByKey(lambda x, y: (y[0] if x[1] >= y[1] else x[0], min(x[1], y[1])))

# Order according to temperature
sorted_min_station_temp = min_station_temp.sortBy(lambda a: a[1][1], ascending=False)

sorted_max_station_temp.saveAsTextFile('./results/task01a_max')
sorted_min_station_temp.saveAsTextFile('./results/task01a_min')
```

Result (maximum temperature):

```
[x.sttofo@heffal task01a_max]$ head -20 all_task01a_max.txt
(1975, (86200, 36.1))
(1992, (63600, 35.4))
(1994, (117160, 34.7))
(2014, (96560, 34.4))
(2010, (75250, 34.4))
(1989, (63050, 33.9))
(1982, (94050, 33.8))
(1968, (137100, 33.7))
(1966, (151640, 33.5))
(2002, (78290, 33.3))
(1983, (98210, 33.3))
(1986, (76470, 33.2))
(1970, (103080, 33.2))
(2000, (62400, 33.0))
(1956, (145340, 33.0))
(1959, (65160, 32.8))
(2006, (75240, 32.7))
(1991, (137040, 32.7))
(1988, (102540, 32.6))
(2011, (172770, 32.5))
```

Result (maximum temperature):

```
[x.sttofo@heffal task01a_min]$ head -20 all_task01a_min.txt
(1990, (166870, -35.0))
(1952, (192830, -35.5))
(1974, (179950, -35.6))
(1954, (113410, -36.0))
(1992, (179960, -36.1))
(1975, (157860, -37.0))
(1972, (167860, -37.5))
(2000, (169860, -37.6))
(1995, (182910, -37.6))
(1957, (159970, -37.8))
(1983, (191900, -38.2))
(1989, (166870, -38.2))
(1953, (183760, -38.4))
(2009, (179960, -38.5))
(1993, (191900, -39.0))
(1984, (191900, -39.2))
(2008, (179960, -39.3))
(1973, (166870, -39.3))
(1991, (179960, -39.3))
(2005, (155790, -39.4))
```

---

Write the non-parallelized program in Python to find the maximum temperatures for each year without using Spark. In this case you will run the program using:

```
python script.py
```

Code:

```
# Start the script
import time
start = time.time()

# Directly reading the data: we will organize the work in a dictionary
# Every year will be a key, to which temperature and station number will be associated
# We first create an empty dictionary where to store the data
max_station_temp = {year:-999 for year in range(1950, 2014+1)}

# Counter to check the progress of the script
i = 0
# Read the file line by line
with open("data/short_temp.csv","r") as file:
    for line in file:
        # Print the progress of our script (every million line)
        if i%1000000==0:
            print(i)
        i += 1
```

```

reading = line.split(";")
year = int(reading[1][0:4])
# Is the year the one we were looking for?
if year >= 1950 and year <= 2014:
    temp = float(reading[3])
    # Substitute the old max with the new max
    if temp > max_station_temp[year]:
        max_station_temp[year] = temp

# We will finally order according to the temperature
ordered_max_station_temp = [(i,max_station_temp[i]) for i in max_station_temp]
ordered_max_station_temp.sort(key=lambda x: x[1])

# Check the time of the start
end = time.time()
# Report the amount spent
final_time = time.strftime('%H:%M:%S', time.gmtime(end-start))

print ("RUNTIME = " + final_time)
# Save it as file
with open("task1b_noSpark_time","w") as file:
    file.write("RUNTIME = " + final_time)

with open("task1b_noSpark", "w") as file:
    for element in ordered_max_station_temp:
        file.write("%s: %s\n" % (element[0], element[1]))

```

Result:

```

[x sttof@heffal ~]$ python lab02_q01_b_noSpark.py
RUNTIME = 00:27:03

```

The required time was approximately of 27 minutes.

---

How does the runtime compare to the Spark version?

Code:

```

# Check the time needed
import time
start = time.time()

# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperatures-big.csv")
splited_lines = tempreture_readings.map(lambda line: line.split(';'))

# extracing year and temp from each line as a tuple
extracted = splited_lines.map(lambda x: (int(x[1][0:4]),float(x[3])))

```

```

extracted = extracted.filter(lambda x: x[0]>=1950 and x[0]<=2014)

# maximum for each year (year is the key)
maximum_temps = extracted.reduceByKey(max)
maximum_temps = maximum_temps.sortBy(lambda x: x[1], ascending = False)

# Check the time from the start
end = time.time()
# Report the amount spent
final_time = time.strftime('%H:%M:%S', time.gmtime(end-start))

print ("RUNTIME = " + final_time)
# Save it as file
with open("task1b_withSpark_time","w") as file:
    file.write("RUNTIME = " + final_time)

maximum_temps.saveAsTextFile('./results/task01b_withSpark_HDFS')

```

Result:

```

=====
RUNTIME = 00:03:36

```

The required time was approximately of 1 minute and an half.

The reason of this discrepancy is due to the way Spark optimizes how it handles the data: it parallelizes the computations allowing to dramatically reduce the computational time required.

## Assignment 2

Count the number of readings for each month in the period of 1950-2014 which are higher than 10 degrees.

Code:

```

# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
splited_lines = tempreture_readings.map(lambda line: line.split(';'))

month_temp = splited_lines.map(lambda x: ((int(x[1][0:4]),
                                           int(x[1][5:7])),float(x[3])))

# Filtering the RDD for the years
month_temp = month_temp.filter(lambda x: x[0][0]>=1950 and x[0][0]<=2014)
# Filtering the RDD by finding temp greater than 10

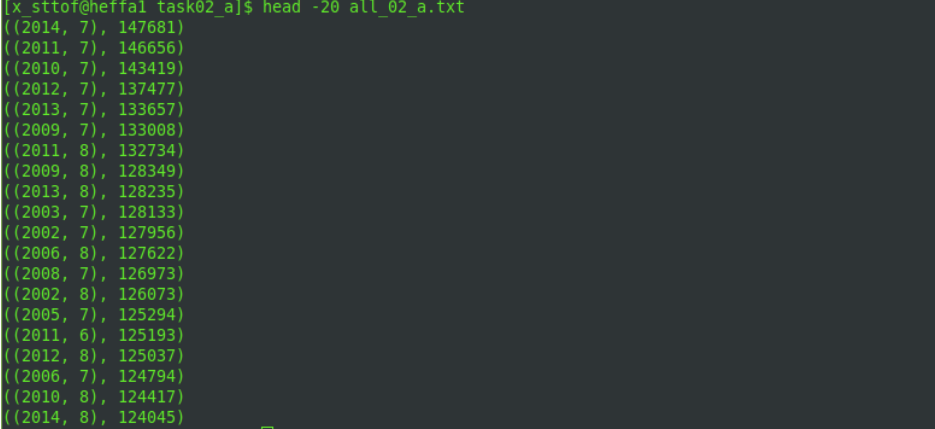
```

```
month_temp_greater_10 = month_temp.filter(lambda x: x[1] > 10)

# Counting the number of registered temp greater than 10 for each month of each year
month_temp_greater_10_count = month_temp_greater_10.countByKey()
month_temp_greater_10_count_sorted = sorted(
    month_temp_greater_10_count.items(), key=lambda x: x[1], reverse=True)

month_temp_greater_10_count_sorted = sc.parallelize(month_temp_greater_10_count_sorted)
month_temp_greater_10_count_sorted.saveAsTextFile('./results/task02_a')
```

Result:



```
[x sttof@heffal task02_a]$ head -20 all_02_a.txt
((2014, 7), 147681)
((2011, 7), 146656)
((2010, 7), 143419)
((2012, 7), 137477)
((2013, 7), 133657)
((2009, 7), 133008)
((2011, 8), 132734)
((2009, 8), 128349)
((2013, 8), 128235)
((2003, 7), 128133)
((2002, 7), 127956)
((2006, 8), 127622)
((2008, 7), 126973)
((2002, 8), 126073)
((2005, 7), 125294)
((2011, 6), 125193)
((2012, 8), 125037)
((2006, 7), 124794)
((2010, 8), 124417)
((2014, 8), 124045)
```

Repeat the exercise, this time taking only distinct readings from each station. That is, if a station reported a reading above 10 degrees in some month, then it appears only once in the count for that month.

Code:

```
# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
temperature_readings = sc.textFile("data/temperature-readings.csv")
splited_lines = temperature_readings.map(lambda line: line.split(';'))

month_temp = splited_lines.map(lambda x: (
    (int(x[1][0:4]), int(x[1][5:7]), int(x[0])), (float(x[3]))))

# Filtering the RDD for the years
month_temp = month_temp.filter(lambda x: x[0][0] >= 1950 and x[0][0] <= 2014)
# Filtering the RDD by finding temp greater than 10
month_temp_greater_10 = month_temp.filter(lambda x: x[1] > 10)

# Getting distinct pairs
distinct_stations = month_temp_greater_10.distinct()

# Counting distinct pairs by key
```

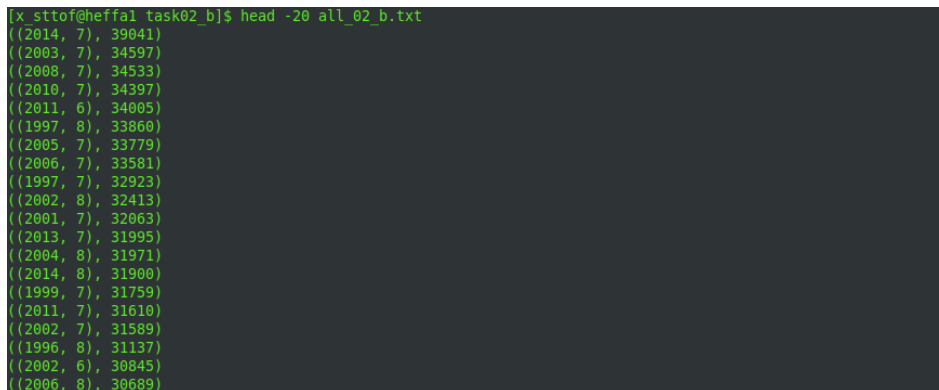
```

back_to_previos = distinct_stations.map( lambda x: ((x[0][0], x[0][1]), (x[1])) )
month_temp_greater_10_count = back_to_previos.countByKey()
month_temp_greater_10_count_sorted = sorted(
    month_temp_greater_10_count.items(), key=lambda x: x[1],reverse=True)

month_temp_greater_10_count_sorted = sc.parallelize(month_temp_greater_10_count_sorted)
distinct_month_temp_temp_sorted.saveAsTextFile('./results/task02_b')

```

Result:



```

[x.stto@heffal task02_b]$ head -20 all_02_b.txt
((2014, 7), 39041)
((2003, 7), 34597)
((2008, 7), 34533)
((2010, 7), 34397)
((2011, 6), 34005)
((1997, 8), 33860)
((2005, 7), 33779)
((2006, 7), 33581)
((1997, 7), 32923)
((2002, 8), 32413)
((2001, 7), 32063)
((2013, 7), 31995)
((2004, 8), 31971)
((2014, 8), 31900)
((1999, 7), 31759)
((2011, 7), 31610)
((2002, 7), 31589)
((1996, 8), 31137)
((2002, 6), 30845)
((2006, 8), 30689)

```

## Assignment 3

Find the average monthly temperature for each available station in Sweden. Your result should include average temperature for each station for each month in the period of 1960- 2014.

Code:

```

# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
splited_lines = tempreture_readings.map(lambda line: line.split(';'))

# creating keys of (y,m,d,stationNo) and Filtering based on year
year_month_station = splited_lines.map(
    lambda x: ((int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]), int(x[0])),float(x[3])))
year_month_station = year_month_station.filter(
    lambda x: x[0][0] >=1960 and x[0][0] <=2014)

# Calculating max-min of temp in each day for each station in each month of each year
maximums = year_month_station.reduceByKey(max)
minimum = year_month_station.reduceByKey(min)

# Calculating (max + min /2) for each day and extracting the keys as (y,m.stationNo)

```



```
max_min_join = maximums.join(minimum)
max_min_day_avg = max_min_join.map(lambda x: ((x[0][0],x[0][1],x[0][3]),
                                              (x[1][0] + x[1][1])/2))

# Summing up the average of max and mean for each month
from operator import add
monthly_max_min_avg = max_min_day_avg.reduceByKey(add)

# Counting the number of points for each month (it will be a dictionary)
monthly_max_min_count = max_min_day_avg.countByKey()

# Converting dictionary to RDD
monthly_max_min_count_rdd = sc.parallelize(monthly_max_min_count.items())

# Joining RDD of Sum Of avgs and counts
avg_count_join = monthly_max_min_avg.join(monthly_max_min_count_rdd)

# Dividing the sum of avgs by counts
monthly_average = avg_count_join.map(lambda x: (x[0],x[1][0]/x[1][1]))

monthly_average.saveAsTextFile('./results/task03')
```

Result:

```
[x_sttof@heffal monthly_average]$ head -20 all_03.txt
((1984, 11, 183760), -9.201666666666666)
((2014, 7, 86470), 19.637096774193548)
((2008, 7, 158740), 13.720967741935484)
((2014, 7, 167710), 17.774193548387096)
((1998, 1, 160960), -10.019354838709678)
((1967, 10, 83540), 9.412903225806453)
((2008, 5, 117430), 8.27741935483871)
((2010, 1, 162860), -12.158064516129032)
((1974, 3, 78250), 1.8096774193548388)
((1964, 9, 96030), 11.811666666666667)
((2003, 8, 97530), 15.93548387096774)
((1973, 6, 114330), 13.403333333333334)
((1998, 5, 158740), 4.229032258064516)
((1975, 10, 161940), 2.8935483870967738)
((2004, 3, 122260), -2.7048387096774187)
((1990, 9, 173960), 6.203333333333333)
((1971, 8, 91130), 16.06451612903226)
((1993, 4, 155900), -0.3683333333333333)
((1990, 9, 74440), 9.533333333333333)
((1984, 7, 74420), 16.75)
```

## Assignment 4

Provide a list of stations with their associated maximum measured temperatures and maximum measured daily precipitation. Show only those stations where the maximum temperature is between 25 and 30 degrees and maximum daily precipitation is between 100 mm and 200 mm. In this exercise you will use the temperature-readings.csv and precipitation-readings.csv files.

Code:

```
# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")
```

```

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
splitted_lines = tempreture_readings.map(lambda line: line.split(';'))

precipitation_readings = sc.textFile('data/precipitation-readings.csv')
splitted_lines = precipitation_readings.map(lambda line: line.split(';'))

# Extracting the key-value pairs (station number, temp)
precipitation_extracted_pairs = splitted_lines.map(lambda x: (int(x[0]),float(x[3])))
tempreture_extracted_pairs = splitted_lines.map(
    lambda x: ((int(x[0]),int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10])),float(x[3])))

# Filtering
tempreture_filterd = tempreture_extracted_pairs.filter(
    lambda x: x[1] >=25 and x[1]<=30)
precipitation_filterd = precipitation_extracted_pairs.filter(
    lambda x: x[1] >=100 and x[1]<=200)

# Finding the max temp and precipitation seperately
tempreture_max = tempreture_filterd.reduceByKey(max)
precipitation_max = precipitation_filterd.reduceByKey(max)

# Joining the results in a RDD
temp_percip_max = tempreture_max.join(precipitation_max)

temp_percip_max.saveAsTextFile('./results/task04')

```

Result:

```

[x_sttof@heffal temp_percip_max]$ head -20 all_task04.txt
[x_sttof@heffal temp_percip_max]$ 

```

The file returned is empty: no station satisfies the constrains.

## Assignment 5

Calculate the average monthly precipitation for the Östergötland region (list of stations is provided in the separate file) for the period 1993-2016.

In this exercise you will use the precipitation-readings.csv and stations-Ostergotland.csv files.

Code:

```

# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
precipitation_readings = sc.textFile('data/temperature-readings.csv')
splitted_lines = precipitation_readings.map(lambda line: line.split(';'))

```

```

stations_ostergotland = sc.textFile('data/stations-Ostergotland.csv')
splitted_lines_stations = stations_ostergotland.map(lambda x: x.split(';'))

extracted_pairs = splitted_lines.map(
    lambda x: ((int(x[0]),int(x[1][0:4]),int(x[1][5:7])),float(x[3])))
extracted_pairs = extracted_pairs.filter(lambda x: x[0][1] >=1993 and x[0][1] <=2016)

# List of ostergotland stations
OST_Stations = splitted_lines_stations.map(lambda x: int(x[0])).collect()

# Extracting the stations in Ostergotland
extracted_pairs_filtered = extracted_pairs.filter(lambda x: x[0][0] in OST_Stations)

# Summing up monthly observations for each station in each year
from operator import add
monthly_sum_perStation = extracted_pairs_filtered.reduceByKey(add)

# Countinh number of stations with registered observation in each month
counts = monthly_sum_perStation.countByKey()
counts = sc.parallelize(counts.items())

# Calculating the avg by dividing monthly sum on the number of stations
monthly_sum_perStation_count = monthly_sum_perStation.join(counts)
monthly_avg_over_stations = monthly_sum_perStation_count.map(
    lambda x: ((x[0][1],x[0][2]),x[1][0]/x[1][1]))

monthly_avg_over_stations.saveAsTextFile('./results/task05')

```

Result:

```

[x_sttof@heffal task05]$ head -20 all_task05.txt
((1998, 11), -68.6)
((1994, 7), 2149.1)
((1996, 5), 5572.799999999999)
((2009, 6), 862.3999999999999)
((2001, 6), 10326.099999999997)
((2000, 1), 296.4999999999999)
((2011, 4), 5199.899999999999)
((2005, 8), 935.6000000000001)
((1999, 8), 11245.400000000007)
((2002, 1), -39.39999999999998)
((2013, 4), 2765.099999999998)
((2006, 1), -264.40000000000003)
((2001, 4), 533.1)
((1996, 3), -81.30000000000001)
((1995, 12), -1994.0000000000007)
((2016, 1), -3254.3999999999999)
((1997, 10), 3168.6000000000017)
((2011, 2), -3852.6000000000004)
((1994, 11), 345.0)
((1994, 9), 2989.7000000000003)

```

## Assignment 6

Compare the average monthly temperature (find the difference) in the period 1950-2014 for all stations in Östergötland with long-term monthly averages in the period of 1950-1980.

Code:

```
# Creating Context
from pyspark import SparkContext
sc = SparkContext(appName = "test_app")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
splited_lines = tempreture_readings.map(lambda line: line.split(';'))

stations_ostergotland = sc.textFile('data/stations-Ostergotland.csv')
splitted_lines_stations = stations_ostergotland.map(lambda x: x.split(';'))

# List of ostergotland stations
OST_Stations = splitted_lines_stations.map(lambda x: int(x[0])).collect()

# Extracting key-value pairs of (y-m-d-station, temp) in ostergotland between 1950 and 2014
y_m_d_station_temp = splitted_lines.map(
    lambda x: ((int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]),int(x[0])),float(x[3])))
y_m_d_station_temp = y_m_d_station_temp.filter(lambda x: x[0][0]>= 1950 and x[0][0]<= 2014)
y_m_d_station_temp = y_m_d_station_temp.filter(lambda x: x[0][3] in OST_Stations)

# Calculating the max and min of temp in each day for each station in each month of each year
maximums = y_m_d_station_temp.reduceByKey(max)
minimum = y_m_d_station_temp.reduceByKey(min)

# Calculating the (max + min /2) for each day and extracting the keys as (y,m.stationNo)
max_min_join = maximums.join(minimum)
max_min_day_avg = max_min_join.map(lambda x: ((x[0][0],x[0][1],x[0][3]),
                                              (x[1][0] + x[1][1])/2))

# Summing up the average of max and mean for each month
from operator import add
monthly_max_min_avg = max_min_day_avg.reduceByKey(add)

# Counting the number of points for each month (it will be a dictionary)
monthly_max_min_count = max_min_day_avg.countByKey()

# Converting dictionary to RDD
monthly_max_min_count_rdd = sc.parallelize(monthly_max_min_count.items())

# Joining RDD of Sum Of avgs and counts
avg_count_join = monthly_max_min_avg.join(monthly_max_min_count_rdd)

# Dividing the sum of avgs by counts
monthly_average_perStation = avg_count_join.map(lambda x:(x[0],x[1][0]/x[1][1]))

# Counting the number of registration in each month
```

```

y_m_removedStation = monthly_average_perStation.map(lambda x: ((x[0][0],x[0][1]),
                                                                x[1]))
y_m_removedStation.filter(lambda x: x[0][0] >= 1950 and x[0][0] <= 1980)

# Sum of AVGs
from operator import add
y_m_removedStation_sum = y_m_removedStation.reduceByKey(add)

# Counts
counts = sc.parallelize(y_m_removedStation.countByKey().items())

# Calculating the long term average
y_m_removedStation_sum_counts = y_m_removedStation_sum.join(counts)
long_term_mavg = y_m_removedStation_sum_counts.map(lambda x: (x[0],x[1][0]/x[1][1]))

# Calculating the difference
monthly_average_perStation_station_removed = monthly_average_perStation.map(
    lambda x: ((x[0][0],x[0][1]),x[1]))
result = monthly_average_perStation_station_removed.join(long_term_mavg).map(
    lambda x: ((x[0][0],x[0][1]),x[1][0]-x[1][1]))

result_sum_by_key = result.reduceByKey(add)
result_count = result.countByKey()
result_count = sc.parallelize(result_count.items())
final_result = result_sum_by_key.join(result_count).map(
    lambda x: ((x[0][0],x[0][1]),x[1][0]/x[1][1]))

final_result.saveAsTextFile('./results/task06')

```

Result:

```

[x sttof@heffal task06]$ head -20 all_task06.txt
((1986, 8), 9.689219124001365e-16)
((1991, 11), 1.3664283380001927e-16)
((1980, 10), -1.6148698540002277e-16)
((1988, 2), -2.5232341468753557e-17)
((1957, 1), 4.4408920985006264e-17)
((1960, 6), -3.552713678800501e-16)
((2008, 6), -3.0682527226004327e-15)
((1971, 7), -5.465713352000771e-16)
((1977, 5), 1.3664283380001927e-16)
((1951, 3), 4.440892098500626e-16)
((1999, 3), 1.5372318802502167e-16)
((2005, 1), 1.5139404881252134e-17)
((1982, 4), -4.844609562000683e-16)
((1997, 9), -1.7763568394002505e-15)
((1974, 12), -9.516197353929913e-17)
((1963, 8), -2.4595710084003467e-15)
((1969, 10), -2.220446049250313e-16)
((1994, 7), -1.937843824800273e-15)
((1999, 4), -3.416070845000482e-16)
((2011, 8), 3.2297397080004555e-16)

```