BDA1-Spark - Lab Report

Mim Kemal Tekin (mimte666) & Andreas Stasinakis (andst745)

4/30/2019

Contents

Task 1

In this task we used temperatures-big.csv. First we find minimum and maximum temperatures for each year. We can see results of it as following:

Data import

```
# import pyspark
from pyspark import SparkContext

# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperatures-big.csv")

# transform the data by splitting each line
lines = temperature_file. \
    map(lambda line: line.split(";"))
```

Task 1: Maximum and Minimum Temperature for each Year

```
# transform the data by extracting year and temperature as tuple
year_temperature = lines. \
    map(lambda x: (x[1][0:4], float(x[3])))

# filter data by year
year_temperature = year_temperature. \
    filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014)

# reducer, to get the max temperature by KEY (year)
max_temperatures = year_temperature. \
    reduceByKey(max). \
    sortByKey(ascending=False)
# reducer, to get the min temperature by KEY (year)
min_temperatures = year_temperature. \
    reduceByKey(min). \
    sortByKey(ascending=False)

max_temperatures.saveAsTextFile("./res/t1_max")
min_temperatures.saveAsTextFile("./res/t1_min")
```

Maximum Temperature

```
(u'2014', 34.4)
(u'2013', 31.6)
(u'2012', 31.3)
(u'2011', 32.5)
(u'2010', 34.4)
(u'2009', 31.5)
(u'2008', 32.2)
(u'2007', 32.2)
(u'2006', 32.7)
(u'2005', 32.1)
(u'2004', 30.2)
(u'2003', 32.2)
(u'2002', 33.3)
(u'2001', 31.9)
(u'2000', 33.0)
(u'1999', 32.4)
(u'1998', 29.2)
(u'1997', 31.8)
(u'1996', 30.8)
(u'1995', 30.8)
```

Minimum Temperature

```
(u'2014', -42.5)
(u'2013', -40.7)
(u'2012', -42.7)
(u'2011', -42.0)
(u'2010', -41.7)
(u'2009', -38.5)
(u'2008', -39.3)
(u'2007', -40.7)
(u'2006', -40.6)
(u'2005', -39.4)
(u'2004', -39.7)
(u'2003', -41.5)
(u'2002', -42.2)
(u'2001', -44.0)
(u'2000', -37.6)
(u'1999', -49.0)
(u'1998', -42.7)
(u'1997', -40.2)
(u'1996', -41.7)
(u'1995', -37.6)
```

Task 1.a: Maximum and Minimum Temperatures For each Year Included Station Numbers

In this task we are asked for extending previous task to find max and min temperatures for each year and which station measured these values.

```
year_station_temp = lines. \
    map(lambda x: (x[1][0:4], (x[0], float(x[3]))))

year_temperature = year_station_temp. \
    filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014)

max_temperatures = year_temperature. \
    reduceByKey(lambda a,b: a if a>b else b). \
    sortByKey(False)

min_temperatures = year_temperature. \
    reduceByKey(lambda a,b: b if a<b else a). \
    sortByKey(False)

max_temperatures.saveAsTextFile("./res/t1_max_stations")
min_temperatures.saveAsTextFile("./res/t1_min_stations")
```

Maximum Temperature

```
(u'2014', (u'99450', 26.0))
(u'2013', (u'\ufeff102170', 6.8))
(u'2012', (u'99450', 21.3))
(u'2011', (u'99450', 21.8))
(u'2010', (u'99450', 25.2))
(u'2009', (u'99450', 22.4))
(u'2008', (u'99450', 24.7))
(u'2007', (u'99450', 21.5))
(u'2006', (u'99450', 24.5))
(u'2005', (u'99450', 25.7))
(u'2004', (u'99450', 25.1))
(u'2003', (u'99450', 25.5))
(u'2002', (u'99450', 24.7))
(u'2001', (u'99450', 24.7))
(u'2000', (u'99450', 18.8))
(u'1999', (u'99450', 24.1))
(u'1998', (u'99450', 19.6))
(u'1997', (u'99450', 25.5))
(u'1996', (u'99450', 23.7))
(u'1995', (u'99450', 24.1))
```

Minimum Temperature

```
(u'2014', (u'102170', -24.3))
(u'2013', (u'102170', -13.3))
(u'2012', (u'102190', -27.0))
(u'2011', (u'102190', -25.8))
(u'2010', (u'102190', -28.9))
(u'2009', (u'102190', -23.7))
(u'2008', (u'102190', -19.8))
(u'2007', (u'102190', -23.5))
(u'2006', (u'102190', -22.2))
(u'2005', (u'102190', -24.2))
(u'2004', (u'102190', -27.9))
(u'2003', (u'102190', -25.7))
(u'2002', (u'102190', -25.3))
(u'2001', (u'102190', -30.3))
(u'2000', (u'102190', -20.6))
(u'1999', (u'102210', -24.3))
(u'1998', (u'102210', -20.3))
(u'1997', (u'102210', -20.9))
(u'1996', (u'102210', -21.8))
(u'1995', (u'102210', -21.4))
```

Task 1.b

Now we run a benchmark test to see how much time non-parallel script that we wrote takes when we compare with parallel one. The script finds only max temperatures for each year. We run this script on "temperatures-big.csv". After running non-parallel script we get the results below.

```
RUNTIME = 1799.30932498

real    29m59.340s
user    29m49.337s
sys     0m8.945s
```

And we can see it took 1799 seconds, which is close to 30 minutes. This script has a for loop and it traverses each row that the data file has and in every iteration it checks for the year filter that we have and checks if it is max or not.
Also we modified the script that we wrote in previous task and we run a benchmark for this task. This script uses hdfs filesystem and it runs as parallel processing. We get the result as following:

```
229.759341955
```

This number presents seconds as before and it is close to 4 minutes. We can clearly see a performance difference. Parallel scipt finishes the task approximately 7 times faster than the normal script. The time above presents only the runtime of main algorithm. We can check the real starting time and end time of the process included with the time which includes queue time for scheduling on server and printing the results to the files below:

```
2019-05-03 18:58:36,356 INFO  [main] yarn.Client (Logging.scala:logInfo(58)) - Application report for application_1556787411054_0394 (state: RUNNING)
```

```
2019-05-03 19:02:26,800 INFO  [main] yarn.Client (Logging.scala:logInfo(58)) - Application report for application_1556787411054_0394 (state: FINISHED)
```

Non-Parallel Script

```
# - *- coding: utf- 8 - *-
# set year filter boundaries
lower_year = 1950
upper_year = 2014
# create empty dictionary for our response
response_max = {year:-10000 for year in range(lower_year,upper_year+1)}

line_no = 0
# read file
with open("/nfshome/hadoop_examples/shared_data/temperatures-big.csv","r") as file:
    for line in file:
        # We split the line by ; symbol.
        # after spliting er get the format as following:
        # ["station_no", "yyyy-mm-dd", "hh:mm:ss", "temp", "G new_line"]
        # we will check years and find the max and min of each year
        if not line_no%10000:
            print(line_no)
        line_no += 1
        reading = line.split(";")
        year = int(reading[1][0:4])
        # check our filter
        if year > 1950 and year < 2014:
            temp = float(reading[3])
            if temp > response_max[year]:
                response_max[year] = temp

end = time.time()

print "RUNTIME = {}".format(end-start)
with open("task1b_nonparallel_time.out","w") as file:
    file.write("RUNTIME = {}".format(end-start))

with open("t1b_max_stations", "w") as file:
    for k in response_max.keys():
        file.write("%s: %s\n" % (k, response_max[k]))
```

Parallel Script

```
# import pyspark
from pyspark import SparkContext
import time

start = time.time()
# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
```

```
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperatures-big.csv")

# transform the data by splitting each line
lines = temperature_file. \
    map(lambda line: line.split(";"))


year_temperature = lines. \
    map(lambda x: (x[1][0:4], float(x[3]))). \
    filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014). \
    reduceByKey(max). \
    sortByKey(ascending=False)


end = time.time()

print("RUNTIME = {}".format(end - start))


max_temperatures.saveAsTextFile("./res/t1_max")
```

Task 2

Count the number of readings for each month in the period of 1950-2014 which are higher than 10 degrees. 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.
In this exercise you will use the temperature-readings.csv file. The output should contain the following information:
Year, month, count

```
# import pyspark
from pyspark import SparkContext

# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperature-readings.csv")

# transform the data by splitting each line
lines = temperature_file. \
    map(lambda line: line.split(";"))

#############
#####   2.i
#############

# map as ((year, month), (station_no, temp, 1))
year_month = lines. \
    map(lambda x: ((x[1][0:4],x[1][5:7]), (x[0], float(x[3]))))
```

```
# filter by constraints and put 1 as value
year_month = year_month. \
    filter(lambda x: int(x[0][0])>=1950 and int(x[0][0])<=2014 and x[1][1]>10)

# map as pair rdd (key,1) and count them, sort them
count_reads = year_month. \
    map(lambda x: (x[0], 1)). \
    reduceByKey(lambda x,y: x+y). \
    sortByKey(ascending=False)

count_reads.saveAsTextFile("./res/t2_count_reads_month")


############
#####   2.ii
############
# get only one tuple for a station in one month.
# Remove duplicated reads in one month
year_month_unique = year_month. \
    map(lambda x: (x[0], (x[1][0], 1))). \
    distinct()

# map as pair rdd (key,1) and count them, sort them
station_month_counts = year_month_unique. \
    map(lambda x: (x[0], 1)). \
    reduceByKey(lambda x,y: x+y). \
    sortByKey(ascending=False)

count_reads.saveAsTextFile("./res/t2_count_reads_month_distinct")
```

Monthly Reading Count Greater than 10

```
((u'2014', u'12'), 3)
((u'2014', u'11'), 8139)
((u'2014', u'10'), 42191)
((u'2014', u'09'), 86090)
((u'2014', u'08'), 124045)
((u'2014', u'07'), 147681)
((u'2014', u'06'), 101711)
((u'2014', u'05'), 57250)
((u'2014', u'04'), 19862)
((u'2014', u'03'), 4213)
((u'2014', u'02'), 33)
((u'2013', u'12'), 16)
((u'2013', u'11'), 1595)
((u'2013', u'10'), 37839)
((u'2013', u'09'), 81960)
((u'2013', u'08'), 128235)
((u'2013', u'07'), 133657)
((u'2013', u'06'), 122181)
((u'2013', u'05'), 81996)
((u'2013', u'04'), 7169)
```

Monthly Distinct Station Reading Count Greater than 10

```
((u'2014', u'12'), 1)
((u'2014', u'11'), 158)
((u'2014', u'10'), 270)
((u'2014', u'09'), 296)
((u'2014', u'08'), 296)
((u'2014', u'07'), 297)
((u'2014', u'06'), 298)
((u'2014', u'05'), 296)
((u'2014', u'04'), 254)
((u'2014', u'03'), 169)
((u'2014', u'02'), 15)
((u'2013', u'12'), 8)
((u'2013', u'11'), 114)
((u'2013', u'10'), 270)
((u'2013', u'09'), 299)
((u'2013', u'08'), 300)
((u'2013', u'07'), 301)
((u'2013', u'06'), 302)
((u'2013', u'05'), 301)
((u'2013', u'04'), 208)
```

Task 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. Bear in mind that not every station has the readings for each month in this timeframe. In this exercise you will use the temperature-readings.csv file.
The output should contain the following information:
Year, month, station number, average monthly temperature

```
# import pyspark
from pyspark import SparkContext

# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperature-readings.csv")

# transform the data by splitting each line
lines = temperature_file. \
    map(lambda line: line.split(";"))


#############
#####   3
#############

# map as (date, station_no), (temperature)
# to calculate average temp of each day
yr_mn_st = lines. \
    map(lambda x: ((x[1], x[0]), (float(x[3]))))

# calculate avg temp for each day by using defined formula
# avg of min of day and max of day
# we grouped it by key in order to apply function to days for each station seperately
```

```
daily_avg = yr_mn_st.groupByKey().mapValues(lambda x: (max(x)+min(x))/2)

# calculate average of month for each station
# map as (year, month, station_no), (daily_avg, 1)
# 1 for counting element count while summing
# sum temperature and count how many elements we have
# map it again to find the average
monthly_avg = daily_avg. \
    map(lambda x: ((x[0][0][0:4], x[0][0][5:7], x[0][1]), (x[1],1))). \
    reduceByKey(lambda x,y: (x[0] + y[0], x[1] + y[1])). \
    map(lambda x: (x[0], x[1][0]/x[1][1])). \
    sortByKey(False)

monthly_avg.filter(lambda x: int(x[0][0])>1960 and int(x[0][0])<2014). \
    saveAsTextFile("./res/t3_avg_month")
```

```
((u'2013', u'12', u'99450'), 3.6709677419354843)
((u'2013', u'12', u'99280'), 3.6870967741935488)
((u'2013', u'12', u'99270'), 3.6306451612903223)
((u'2013', u'12', u'98490'), 2.138709677419355)
((u'2013', u'12', u'98290'), 3.0967741935483875)
((u'2013', u'12', u'98230'), 3.2661290322580645)
((u'2013', u'12', u'98210'), 3.470967741935484)
((u'2013', u'12', u'98180'), 3.17258064516129)
((u'2013', u'12', u'98160'), 3.4548387096774196)
((u'2013', u'12', u'98040'), 2.98548387096742)
((u'2013', u'12', u'97530'), 2.088709677419355)
((u'2013', u'12', u'97510'), 2.52258064516129)
((u'2013', u'12', u'97400'), 2.511290322580645)
((u'2013', u'12', u'97370'), 2.332258064516129)
((u'2013', u'12', u'97280'), 3.096774193548387)
((u'2013', u'12', u'97120'), 2.620967741935484)
((u'2013', u'12', u'97100'), 2.2483870967741937)
((u'2013', u'12', u'96560'), 1.696774193548387)
((u'2013', u'12', u'96550'), 1.6483870967741934)
((u'2013', u'12', u'96350'), 2.383870967741354)
```

Task 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.
The output should contain the following information:

Station number, maximum measured temperature, maximum daily precipitation

```
# import pyspark
from pyspark import SparkContext

# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperature-readings.csv")

# transform the data by splitting each line
lines = temperature_file. \
```

```
    map(lambda line: line.split(";"))

# import the data
# precipitation_file = sc.textFile("../station_data/short_precipitation-readings.csv")
precipitation_file = sc.textFile("/user/x_mimte/data/precipitation-readings.csv")

# transform the data by splitting each line
lines_precipitation = precipitation_file. \
    map(lambda line: line.split(";"))


############
#####   4
############


# map as (station_no, temp)
# find maximum read of station
station_temp = lines. \
    map(lambda x: (x[0], float(x[3]))). \
    reduceByKey(max)


# map as ((station, date), precipitation)
# calculate daily precipitation
# map as (station, precipitation)
# find max precipitation of station
station_precipitation = lines_precipitation. \
    map(lambda x: ((x[0], x[1]), float(x[3]))). \
    reduceByKey(lambda x,y: x+y). \
    map(lambda x: (x[0][0], x[1])). \
    reduceByKey(max)


# join them
station_temp_prec = station_temp.join(station_precipitation)

# filter the last data for constraints
station_temp_prec = station_temp_prec. \
    filter(lambda x: x[1][0]>25 and x[1][0]<30 \
    and x[1][1]>100 and x[1][1]<200)

station_temp_prec.saveAsTextFile("res/t4_station_temp_prec")
```

```
(u'136410', (31.1, 21.7))
(u'107140', (32.1, 19.2))
(u'115220', (30.6, 26.0))
(u'147560', (29.9, 29.3))
(u'167990', (30.1, 15.7))
(u'180770', (28.9, 34.5))
(u'170930', (31.0, 28.2))
(u'161710', (30.6, 20.8))
(u'83190', (32.7, 40.3))
(u'163960', (31.5, 14.4))
(u'122260', (28.5, 22.9))
(u'173900', (30.9, 18.8))
(u'98040', (35.0, 24.0))
(u'53530', (32.2, 52.4))
(u'144310', (28.4, 12.8))
(u'149340', (31.1, 29.2))
(u'146350', (30.8, 24.1))
(u'62040', (30.9, 21.9))
(u'159880', (30.1, 17.7))
(u'63590', (32.9, 24.8))
```

Task 5

Calculate the average monthly precipitation for the Ostergotland region (list of stations is provided in the separate file) for the period 1993-2016. In order to do this, you will first need to calculate the total monthly precipitation for each station before calculating the monthly average (by averaging over stations).
In this exercise you will use the precipitation-readings.csv and stations-Ostergotland.csv files.
The output should contain the following information:
Year, month, average monthly precipitation

```
# import pyspark
from pyspark import SparkContext

# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperature-readings.csv")

# transform the data by splitting each line
lines = temperature_file. \
    map(lambda line: line.split(";"))

# import the data
# precipitation_file = sc.textFile("../station_data/short_precipitation-readings.csv")
precipitation_file = sc.textFile("/user/x_mimte/data/precipitation-readings.csv")

# transform the data by splitting each line
lines_precipitation = precipitation_file. \
    map(lambda line: line.split(";"))


# import stations in Ostergotland
# ost_station_file = sc.textFile("../station_data/stations-Ostergotland.csv")
ost_station_file = sc.textFile("/user/x_mimte/data/stations-Ostergotland.csv")
```

```
############
#####   5
############

# transform the data by splitting each line
lines_ost = ost_station_file. \
    map(lambda line: line.split(";"))

# get station_ids in Ostergotland as an Array
ost_station_ids = lines_ost. \
    map(lambda x: x[0]).collect()

# map as ((station_no, yyyy, mm), (precipitation, 1))
# filter only ostergotland stations and date constraint
# map as ((yyyy, mm), (precipitation, 1))
# sum all values by reduceByKey
# map it again to find avg of month
# sort
ost_station_prec = lines_precipitation. \
    map(lambda x: ((x[0], x[1][0:4], x[1][5:7]), (float(x[3]), 1))). \
    filter(lambda x: x[0][0] in ost_station_ids and \
        int(x[0][1])>1993 and int(x[0][1])<2016). \
    map(lambda x: ((x[0][1], x[0][2]), x[1])). \
    reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1])). \
    map(lambda x: (x[0], x[1][0]/x[1][1])). \
    sortByKey(False)

ost_station_prec.saveAsTextFile("./res/t5_avg_ost_station")
```

```
((u'2015', u'12'), 0.04152907394113416)
((u'2015', u'11'), 0.09301182893539593)
((u'2015', u'10'), 0.00320183973111622)
((u'2015', u'09'), 0.14667873303167472)
((u'2015', u'08'), 0.03825301204819276)
((u'2015', u'07'), 0.16872675757039182)
((u'2015', u'06'), 0.115235304892148)
((u'2015', u'05'), 0.1307962118554897)
((u'2015', u'04'), 0.022284780239738455)
((u'2015', u'03'), 0.059807017543859586)
((u'2015', u'02'), 0.038925911407291104)
((u'2015', u'01'), 0.08375841303577758)
((u'2014', u'12'), 0.05039971575768328)
((u'2014', u'11'), 0.07664473684210539)
((u'2014', u'10'), 0.1020152024041014)
((u'2014', u'09'), 0.07054969057153256)
((u'2014', u'08'), 0.1299874753981037)
((u'2014', u'07'), 0.03264690218356114)
((u'2014', u'06'), 0.10947004188672389)
((u'2014', u'05'), 0.08136068735753121)
```

Task 6

Compare the average monthly temperature (find the difference) in the period 1950-2014 for all stations in Ostergotland with long-term monthly averages in the period of 1950-1980. Make a plot of your results. The output should contain the following information:

Year, month, difference

```
# import pyspark
from pyspark import SparkContext
```

```
# create the spark application
sc = SparkContext(appName = "Exercise App")

# import the data
# temperature_file = sc.textFile("../station_data/short_temperature_reads.csv")
temperature_file = sc.textFile("/user/x_mimte/data/temperature-readings.csv")

# transform the data by splitting each line
lines = temperature_file. \
    map(lambda line: line.split(";"))


# import stations in Ostergotland
# ost_station_file = sc.textFile("../station_data/stations-Ostergotland.csv")
ost_station_file = sc.textFile("/user/x_mimte/data/stations-Ostergotland.csv")


# transform the data by splitting each line
lines_ost = ost_station_file. \
    map(lambda line: line.split(";"))

# get station_ids in Ostergotland as an Array
ost_station_ids = lines_ost. \
    map(lambda x: x[0]).collect()


############
#####   3
############

# map as (date, station_no), (temperature)
# to calculate average temp of each day
yr_mn_st = lines. \
    map(lambda x: ((x[1], x[0]), (float(x[3]))))

# calculate avg temp for each day by using defined formula
# avg of min of day and max of day
# we grouped it by key in order to apply function to days for each station seperately
daily_avg = yr_mn_st.groupByKey().mapValues(lambda x: (max(x)+min(x))/2)

# calculate average of month for each station
# map as (year, month, station_no), (daily_avg, 1)
# filter them in order to get only ostergotland stations
# 1 for counting element count while summing
# sum temperature and count how many elements we have
# map it again to find the average
monthly_avg = daily_avg. \
    map(lambda x: ((x[0][0][0:4], x[0][0][5:7], x[0][1]), (x[1],1))). \
    filter(lambda x: x[0][2] in ost_station_ids). \
    reduceByKey(lambda x,y: (x[0] + y[0], x[1] + y[1])). \
    map(lambda x: (x[0], x[1][0]/x[1][1])). \
    sortByKey(False)
```

```
############
#####   6
############

# we use the avg temperature monthly that we found before in taks 3
# filter for the year constraint
# map as ((yyyy, mm),(avg_temp, 1))
# reduceByKey and found sum of values by keys
# map it again as ((yyyy, mm), avg) where avg is value[0]/value[1]
# sort them
monthly_avg50_14 = monthly_avg. \
    filter(lambda x: int(x[0][0])>=1950 and int(x[0][0])<=2014). \
    map(lambda x: ((x[0][0], x[0][1]),(x[1], 1))). \
    reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1])). \
    map(lambda x: (x[0], x[1][0]/x[1][1])). \
    sortByKey(ascending=False)

# filter again to get only before 1980
# map it as (mm, (avg_temp, 1))
# reduceByKey and found sum of values by keys
# map it again as (mm, avg)
# sort them
long_term_avg = monthly_avg50_14. \
    filter(lambda x: int(x[0][0])<=1980). \
    map(lambda x: (x[0][1], (x[1], 1))). \
    reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1])). \
    map(lambda x: (x[0], x[1][0] / x[1][1])). \
    sortByKey(ascending=False)


monthly_avg50_14.saveAsTextFile("res/t6_avg_monthly")
long_term_avg.saveAsTextFile("res/t6_long_term_avg")
```

Average Monthly

```
((u'2014', u'12'), -0.13577712609970674)
((u'2014', u'11'), 4.3805)
((u'2014', u'10'), 8.72683284457478)
((u'2014', u'09'), 11.869545454545452)
((u'2014', u'08'), 15.44325513196481)
((u'2014', u'07'), 19.029032258064515)
((u'2014', u'06'), 13.773030303030303)
((u'2014', u'05'), 10.580806451612903)
((u'2014', u'04'), 6.544666666666666)
((u'2014', u'03'), 3.831612903225806)
((u'2014', u'02'), 1.595535714285714)
((u'2014', u'01'), -2.2908064516129034)
((u'2013', u'12'), 2.92000651678071)
((u'2013', u'11'), 3.2512121212121214)
((u'2013', u'10'), 7.95718475073313)
((u'2013', u'09'), 10.803429752066116)
((u'2013', u'08'), 15.771260997067452)
((u'2013', u'07'), 16.943401759530794)
((u'2013', u'06'), 15.036212121212122)
((u'2013', u'05'), 11.887536656891495)
```

Long Term Average Monthly

```
(u'12', -0.9596664798954081)
(u'11', 2.316960327307101)
(u'10', 7.2042778605369655)
(u'09', 11.808487268108236)
(u'08', 16.0859022039355)
(u'07', 16.93512148217143)
(u'06', 15.580398922761825)
(u'05', 10.313615801472203)
(u'04', 4.47847350767512)
(u'03', -0.6551354403487598)
(u'02', -3.824775600280328)
(u'01', -3.2233944723330787)
```