

BDA1 - Spark Lab 1

Roshni Sundaramurthy (rossu809) and Yusur Al-Mter (yusal621)

May 01, 2019

Contents

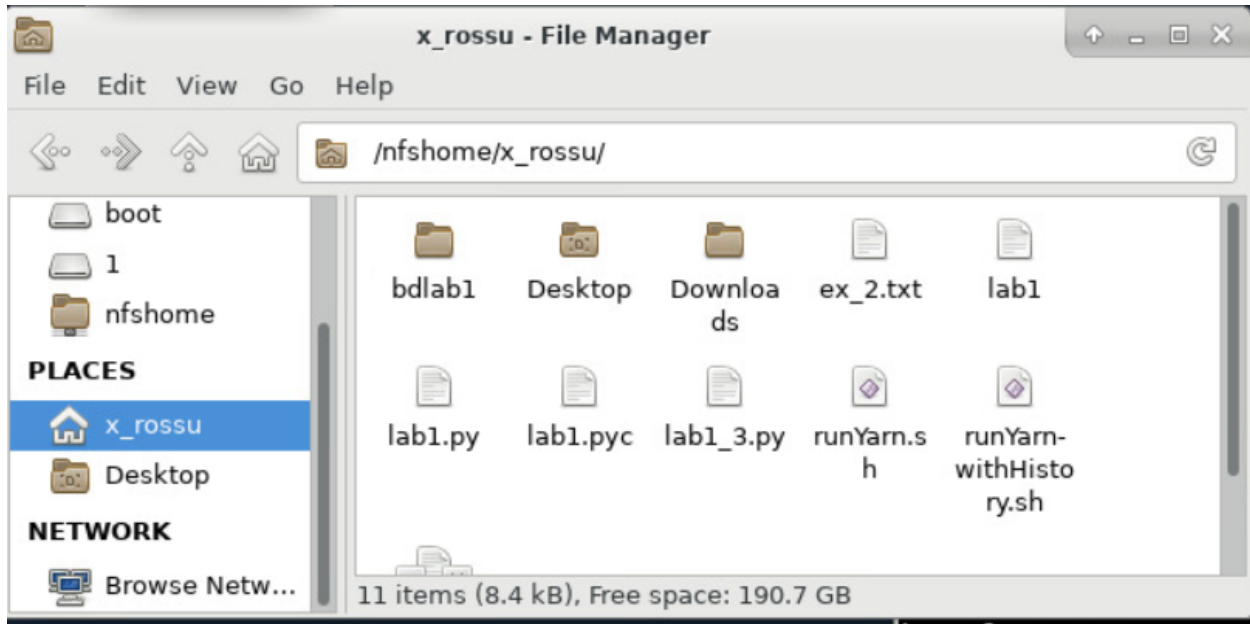
1	Question 1:	2
1.1	Spark code:	2
1.2	Non-parallelized program in Python:	4
2	Question 2:	5
2.1	Spark code:	5
3	Question 3:	6
3.1	Spark code:	6
4	Question 4:	7
4.1	Spark code:	8
5	Question 5:	9
5.1	Spark code:	9
6	Question 6:	10
6.1	Spark code:	10

Initial setups

```
# create directory
hdfs dfs mkdir bdlab1

# copy files to hdfs from local
hdfs dfs -copyFromLocal temperature-readings.csv bdlab1/
hdfs dfs -copyFromLocal precipitation-readings.csv bdlab1/
hdfs dfs -copyFromLocal stations-Ostergotland.csv bdlab1/

# check the files if it exist or not
hdfs dfs -ls /user/x_rossu/
```



1 Question 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.

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

1.1 Spark code:

```
from pyspark import SparkContext
from datetime import datetime

sc=SparkContext(appName="ex-1")

start = datetime.now()
temperature_file=sc.textFile("/user/x_rossu/bdlab1/temperature-readings.csv")
temperature_file.top(5)
lines=temperature_file.map(lambda line: line.split(";"))

# 1) a)
year_temperature= lines.map(lambda x: (x[1][0:4], (float(x[3]),x[0])))
year_temperature= year_temperature.filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014)

def max_temperature(a,b):
    if a[0]>=b[0]:
        return a
    else:
        return b
```

```

max_temperatures=year_temperature.reduceByKey(max_temperature)
min_temperatures=year_temperature.reduceByKey(min)

max_temp_sorted = max_temperatures.sortBy(ascending=False, keyfunc=lambda k: k[1][0])
min_temp_sorted = min_temperatures.sortBy(ascending=False, keyfunc=lambda k: k[1][0])

end = datetime.now()
print("Time taken by spark code is" + str(end - start))

max_temp_sorted.saveAsTextFile("max_temp")
min_temp_sorted.saveAsTextFile("min_temp")

```

1.1.1 Running code:

```

# Adding the execute permissions to the script before you run it
chmod u+x runYarn.sh
# to run lab1.py file
./runYarn.sh lab1.py # (for ex-1 exercise-1)

# Syntax for displaying the size of a file max_temp and directory in HDFS
hdfs dfs -du max_temp

# check outputs of file
hdfs dfs -cat max_temp/part-00000
hdfs dfs -cat min_temp/part-00001

```

1.1.2 Lowest and highest temperatures measured each year for the period 1950-2014:

```

# Max-temperatures (1st four)

Year, Max_Temperature, Station
(u'1975', (36.1, u'86200'))
(u'1992', (35.4, u'63600'))
(u'1994', (34.7, u'117160'))
(u'2014', (34.4, u'96560'))

# Min-temperaures (1st four)

Year, Min_Temperature, Station
(u'1992', (-36.1, u'179960'))
(u'1975', (-37.0, u'157860'))
(u'1972', (-37.5, u'167860'))
(u'1995', (-37.6, u'182910'))

```

Time taken by spark code is 0:01:52.245154

b) 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` This program will read the local file (not from HDFS). The local file is available under `/nfs/home/hadoop_examples/shared_data/temperatures-big.csv`. How does the runtime compare to the Spark version? Use logging (add the `-conf spark.eventLog.enabled=true` flag) to check the execution of the Spark program. Repeat the exercise, this time using `temperatures-big.csv`

file available on hdfs. Explain the differences and try to reason why such runtimes were observed.

1.2 Non-parallelized program in Python:

```
import time
start = time.time()

temp_dict = dict()
with open("/nfshome/hadoop_examples/shared_data/temperature-readings.csv") as f:
    for l in f:
        line = l.split(";")
        year = int(line[1].split("-")[0])
        if year >= 1950 and year <= 2014:
            station = line[0];
            temp = temp_dict.get(year)
            pres_temp = float(line[3]);
            if not temp:
                temp_dict[year] = [[station, pres_temp], [station, pres_temp]]
            else:
                min_temp = float(temp[0][1])
                max_temp = float(temp[1][1])
                if pres_temp < min_temp:
                    temp[0][0] = station
                    temp[0][1] = pres_temp
                if pres_temp > max_temp:
                    temp[1][0] = station
                    temp[1][1] = pres_temp
f.close()
sorted_temp = temp_dict.items()
sorted_temp = sorted(sorted_temp, key=lambda temp: temp[1][1][1], reverse=True)
with open("/nfshome/x_rossu/ex_2.txt", "w+") as f:
    for i in sorted_temp:
        f.write('%s,%s,%s,%s,%s\n' % (i[0], i[1][0][0], i[1][0][1], i[1][1][0], i[1][1][1]))
f.close()
end = time.time()
print("Time taken by non-parallelized program is",(end - start))
```

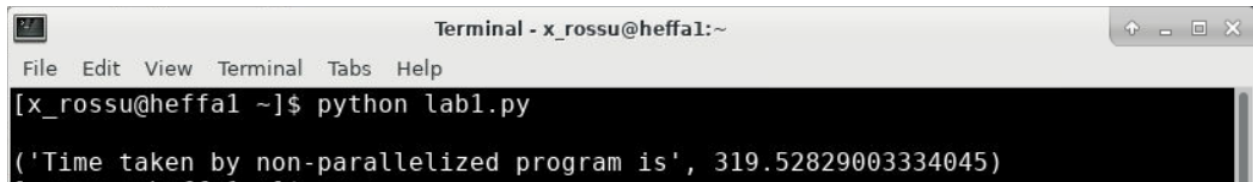
1.2.1 Running code:

```
python lab1.py
```

1.2.2 Lowest and highest temperatures measured each year for the period 1950-2014:

```
# first five readings
Year, Station, Min_temp, Station, Max_temp
1975,157860,-37.0,86200,36.1
1992,179960,-36.1,63600,35.4
1994,179960,-40.5,117160,34.7
2010,191910,-41.7,75250,34.4
2014,192840,-42.5,96560,34.4
```

For 2 GB temperature file

A terminal window titled "Terminal - x_rossu@heffal:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command prompt shows "[x_rossu@heffal ~]\$ python lab1.py" and the output is "('Time taken by non-parallelized program is', 319.52829003334045)".

```
Terminal - x_rossu@heffal:~
File Edit View Terminal Tabs Help
[x_rossu@heffal ~]$ python lab1.py
('Time taken by non-parallelized program is', 319.52829003334045)
```

Result:

Using both code, we got same result. But time taken by non-parallelized program is 5 min 31 sec for temperature-readings.csv (2 GB) file and 41 min 29 sec for temperatures-big.csv (17 GB) file. This is because, temperatures-big.csv is essentially a concatenation of 8 copies of temperature-readings.csv file. So it takes longer time. Also, using Spark code, the whole process just took 1 min 52 sec which is much faster than non parallelised code since the computations are done in parallelised way.

2 Question 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.

2.1 Spark code:

```
from pyspark import SparkContext

sc=SparkContext(appName="ex-2")

temperature_file=sc.textFile("/user/x_rossu/bdlab1/temperature-readings.csv")
temperature_file.top(5)
lines=temperature_file.map(lambda line: line.split(";"))

# Count the number of readings for each month in the period of 1950-2014
# which are higher than 10 degrees
year_temperature= lines.filter(lambda x: int(x[1][0:4])>=1950 and int(x[1][0:4])<=2014)

# count num of readings (default = 1)
year_temp = year_temperature.map(lambda x:((x[1][0:7],1), float(x[3])))

year_temp = year_temp.filter(lambda temp: (temp[1] > 10))
coun = year_temp.map(lambda x:(x[0]))
count = coun.reduceByKey(lambda x,y: x+y)
count.saveAsTextFile("count_of_readings")

## Distinct readings from each station

year_temp = year_temperature.map(lambda x:((x[0],x[1][0:7],1), float(x[3])))
year_temp = year_temp.filter(lambda temp: (temp[1] > 10))

distinct_temp= year_temp.map(lambda x:(x[0])).distinct()
```

```
coun = distinct_temp.map(lambda x:(x[1],x[2]))
count = coun.reduceByKey(lambda x,y: x+y)
count.saveAsTextFile("distinct_count_of_readings")
```

2.1.1 Running code

```
# run
./runYarn.sh lab1.py

# print o/p
hdfs dfs -du count_of_readings
hdfs dfs -cat count_of_readings/part-00000

hdfs dfs -du distinct_count_of_readings
hdfs dfs -cat distinct_count_of_readings/part-00000
```

2.1.2 Number of readings above 10 degrees for each month in the period of 1950-2014:

```
# first five readings
Year-Month, Count
(u'1967-07', 53813)
(u'1974-07', 66277)
(u'2003-05', 48264)
(u'1978-03', 306)
(u'1981-10', 9882)
```

2.1.3 Distinct readings above 10 degrees for each month in the period of 1950-2014:

```
# first five readings
Year-Month, Count
(u'1979-04', 227)
(u'1974-07', 362)
(u'2003-05', 321)
(u'2004-04', 267)
(u'1981-10', 325)
```

3 Question 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.

3.1 Spark code:

```

from pyspark import SparkContext

sc=SparkContext(appName="ex-3")

temperature_file=sc.textFile("/user/x_rossu/bdlab1/temperature-readings.csv")
temperature_file.top(5)
lines=temperature_file.map(lambda line: line.split(";"))

# Count the number of readings for each month in the period of 1950-2014
# which are higher than 10 degrees
year_temperature= lines.filter(lambda x: int(x[1][0:4])>=1960 and int(x[1][0:4])<=2014)
temp = year_temperature.map(lambda x: ((x[0], x[1]), (float(x[3]), float(x[3]))))
temp = temp.reduceByKey(lambda a,b: (a[0] if a[0]<b[0] else b[0], a[1] if a[1]>=b[1] else b[1]))
temp = temp.map(lambda x: ((x[0][0], x[0][1][:7]), (x[1][0] + x[1][1], 2)))
sum_temp = temp.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
avg_temp = sum_temp.map(lambda x: (x[0][1], x[0][0], x[1][0]/ x[1][1]))
avg_temp.saveAsTextFile("Avg_month_temperature")

```

3.1.1 Running code

```

# run the code
./runYarn.sh lab1.py

hdfs dfs -cat Avg_mon_temperature/part-00000 | tail

```

3.1.2 Average monthly temperature for each available station in Sweden in the period of 1960-2014:

```

Year-Month, Station, Average temperatue
(u'1999-02', u'124020', -6.451785714285714)
(u'1964-04', u'139200', 1.8616666666666668)
(u'2014-10', u'155970', 1.311290322580645)
(u'1997-12', u'87150', 1.1951612903225808)
(u'1976-06', u'63280', 12.889999999999999)
(u'1990-09', u'84310', 10.366666666666669)
(u'1986-04', u'103090', -0.09500000000000004)
(u'1981-11', u'127310', -2.2566666666666667)
(u'2000-06', u'102390', 11.388333333333333)
(u'1975-07', u'82380', 18.780645161290323)

```

4 Question 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.

4.1 Spark code:

```
from pyspark import SparkContext

sc=SparkContext(appName="ex-4")

temperature_file=sc.textFile("/user/x_rossu/bdlab1/temperature-readings.csv")
temperature_file.top(5)
lines=temperature_file.map(lambda line: line.split(";"))

temperature= lines.map(lambda x: (x[0],float(x[3])))
temperature= temperature.filter(lambda temp: (temp[1]>=25 and temp[1]<=30))
max_temp= temperature.reduceByKey(lambda a,b: a if a>=b else b )
max_temp.saveAsTextFile("Max_temp_task4")

precipitation_file=sc.textFile("/user/x_rossu/bdlab1/precipitation-readings.csv")
prec_lines=precipitation_file.map(lambda line: line.split(";"))
precipitation= prec_lines.map(lambda x: ((x[1],x[0]),float(x[3])))
daily_precip = precipitation.reduceByKey(lambda a,b: a+b)
precipitation = daily_precip.map(lambda x: (x[0][1], x[1]))
precipitation= precipitation.filter(lambda prec: (prec[1]>=100 and prec[1]<=200))
max_prec= precipitation.reduceByKey(max)
max_prec.saveAsTextFile("Max_prec_task4")
```

4.1.1 Running code

```
# running the code
./runYarn.sh lab1.py

# print o/p
hdfs dfs -cat Max_temp_task4/part-00000
hdfs dfs -cat Max_prec_task4/part-00001
```

4.1.2 Stations with their associated maximum measured temperatures and maximum measured daily precipitation:

```
# 1st five
Station,MaxTemp
(u'115230', 30.0)
(u'52510', 30.0)
(u'95160', 30.0)
(u'81060', 28.0)
(u'68560', 28.5)

# For precipitation, no results available since it has no values ranging between 100 and 200
```


5 Question 5:

Calculate the average monthly precipitation for the Östergötland region (list of stations is provided in the separate file). 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-Östergötland.csv` files.

5.1 Spark code:

```
from pyspark import SparkContext

sc=SparkContext(appName="ex-5")

Ostergotland_file=sc.textFile("/user/x_rossu/bdlab1/stations-Östergotland.csv")
lines= Ostergotland_file.map(lambda line: line.split(";"))
Ost_station = lines.map(lambda x: x[0])
Ost_station = Ost_station.coalesce(1).collect()
Ost_station = sc.broadcast(Ost_station)

precipitation_file=sc.textFile("/user/x_rossu/bdlab1/precipitation-readings.csv")
prec_lines=precipitation_file.map(lambda line: line.split(";"))
precip = prec_lines.map(lambda x: ((x[0], x[1][0:4], x[1][5:7]), (float(x[3]))))
precip = precip.filter(lambda x: x[0][0] in Ost_station.value)
year_precip = precip.filter(lambda x: int(x[0][1]) >= 1993 and int(x[0][1]) <= 2016)
sum_precip = year_precip.reduceByKey(lambda a, b: a + b)
count_precip = sum_precip.map(lambda x: ((x[0][1], x[0][2]), (x[1], 1)))
precip = count_precip.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
precip = precip.map(lambda x: (x[0][0], x[0][1], (x[1][0] / x[1][1])))
precip.saveAsTextFile("Avg_mon_precip")
```

5.1.1 Running code

```
# run
./runYarn.sh lab1.py

# print o/p

hdfs dfs -cat Avg_Precipitation/part-00001
```

5.1.2 Average monthly precipitation for the Östergötland region for the period 1993-2016:

```
Year,Month,Avg_precipitation
(u'2004', u'01', 26.400000000000001)
(u'2013', u'03', 7.3874999999999998)
(u'2016', u'03', 19.962500000000002)
(u'1998', u'10', 53.41666666666671)
(u'2006', u'12', 29.733333333333334)
```

6 Question 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.

6.1 Spark code:

```
from pyspark import SparkContext

sc = SparkContext(appName="ex-6")

Ostergotland_file=sc.textFile("/user/x_rossu/bdlab1/stations-Ostergotland.csv")
lines= Ostergotland_file.map(lambda line: line.split(";"))
Ost_station = lines.map(lambda a: int(a[0]))
Ost_station = Ost_station.coalesce(1).collect()
Ost_station = sc.broadcast(Ost_station)

temperature_file = sc.textFile("/user/x_rossu/bdlab1/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
temper = lines.map(lambda x: (int(x[0]), int(x[1][8:10]), int(x[1][:4]), int(x[1][5:7]),
                             x[2], float(x[3]), x[4]))
temp = temper.map(lambda x: ((x[0], x[1], x[2], x[3]),(x[5], x[5])))
year_temp = temp.filter(lambda x: x[0][2] >= 1950 and x[0][2] <= 2014)
temp = year_temp.filter(lambda x: x[0][0] in Ost_station.value)
temp = temp.reduceByKey(lambda a,b: (a[0] if a[0]<b[0] else b[0],a[1] if a[1]>=b[1] else b[1]))
temp = temp.map(lambda x: ((x[0][0], x[0][2], x[0][3]),(x[1][0] + x[1][1], 2)))
sum_temp = temp.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
avg_temp = sum_temp.map(lambda x: ((x[0][1], x[0][2]),(x[1][0]/ x[1][1], 1)))
avg_temp = avg_temp.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
avg_temp = avg_temp.map(lambda x: ((x[0][0], x[0][1]), x[1][0]/x[1][1]))

long_temp = avg_temp.filter(lambda x: x[0][0] >= 1950 and x[0][0] <= 1980)
count_temp = long_temp.map(lambda x: (x[0][1], (x[1], 1)))
sum_temp = count_temp.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
long_avg_temp = sum_temp.map(lambda x: (x[0], x[1][0]/x[1][1]))
month_temp = {a:b for a,b in long_avg_temp.collect()}

mon_Avgtemp = avg_temp.map(lambda x: (x[0][0], x[0][1], x[1] - month_temp[x[0][1]]))
mon_Avgtemp = mon_Avgtemp.sortBy(ascending=True, keyfunc=lambda x: (x[0], x[1]))
long_avg_temp = mon_Avgtemp.map(lambda x: '%s,%s,%s' % (x[0],x[1],x[2]))
long_avg_temp.coalesce(1).saveAsTextFile("Comp_temp")
```

6.1.1 Running code

```
# running the code
./runYarn.sh lab1.py

hdfs dfs -cat Comp_temp/part-00000 | tail
```

6.1.2 Compared average monthly temperature in the period 1950-2014 with long-term monthly averages in the period of 1950-1980:

```
Year, Month, Compared_Temperature
2014,3,4.48674834357
2014,4,2.06619315899
2014,5,0.267190650141
2014,6,-1.80736861973
2014,7,2.09391077589
2014,8,-0.642647071971
2014,9,0.0610581864372
2014,10,1.52255498404
2014,11,2.06353967269
2014,12,0.823889353796
```

The readings are saved in an excel file (separated by comma) for plotting.

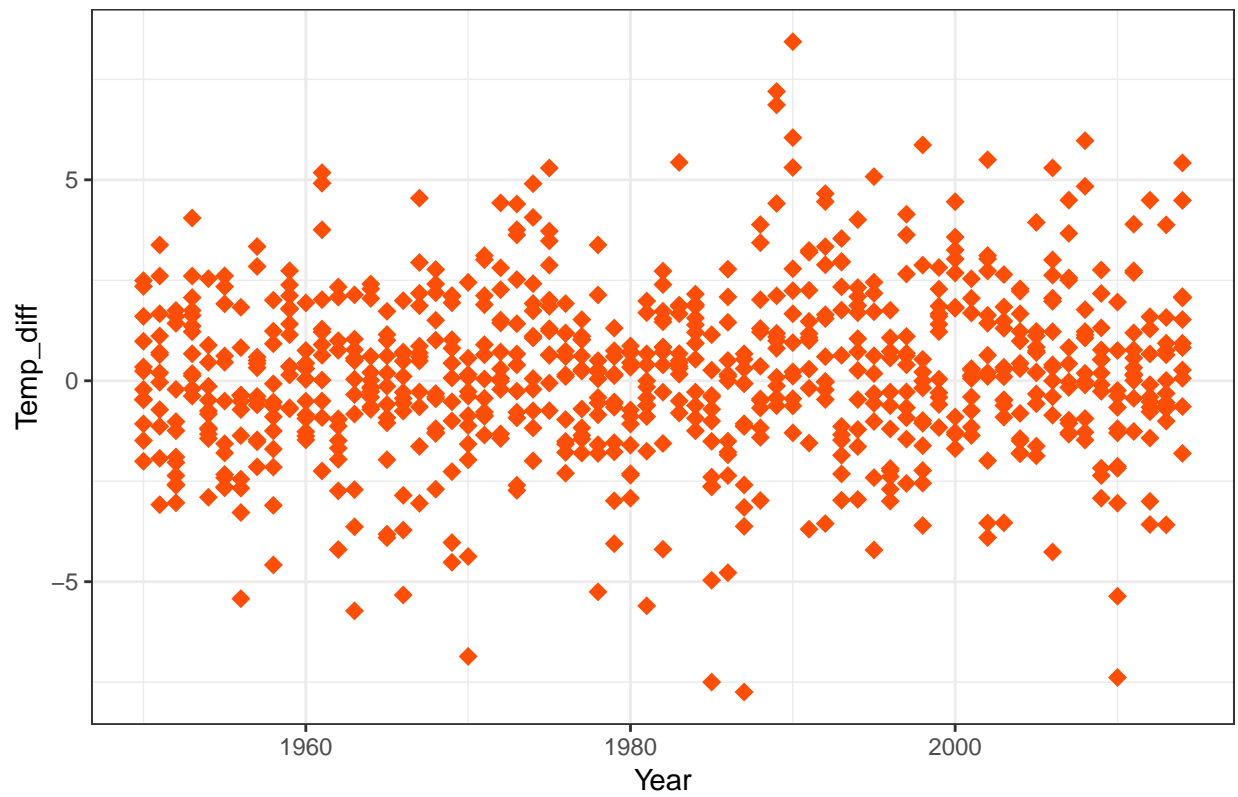
6.1.3 Plot:

```
#library(xlsx)
library(ggplot2)
Temp <- read.csv("Comp_temp.csv")
head(Temp)

##   Year Month  Temp_diff
## 1 1950     1 -2.0048313
## 2 1950     2  2.3479899
## 3 1950     3  2.4922322
## 4 1950     4  1.6006932
## 5 1950     5  0.9823519
## 6 1950     6 -0.2162323

# plot
ggplot(Temp) + geom_point(aes(x = Year, y=Temp_diff), shape = 18, color = "#FC4E07", size = 3) +
  ggtitle("Plot of average monthly temperature difference") + theme_bw()
```

Plot of average monthly temperature difference



In year 1990, one data point (temperature difference) seems to be above 7.5