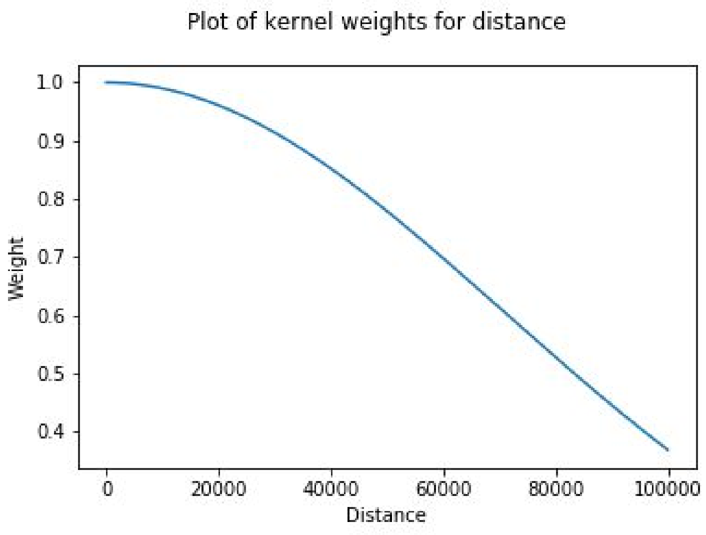
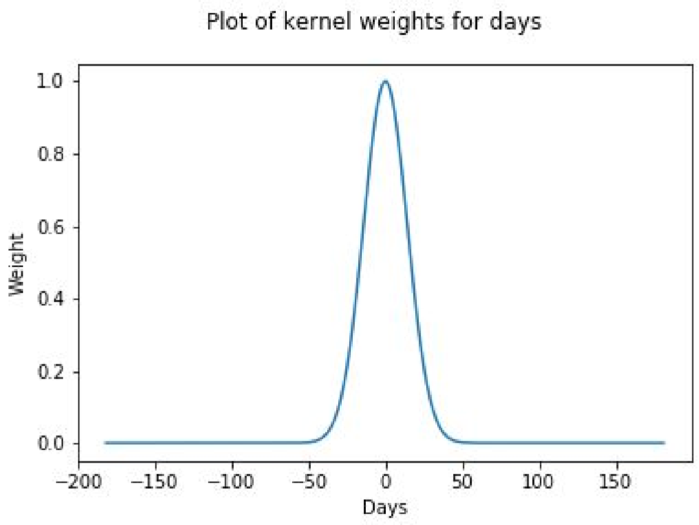
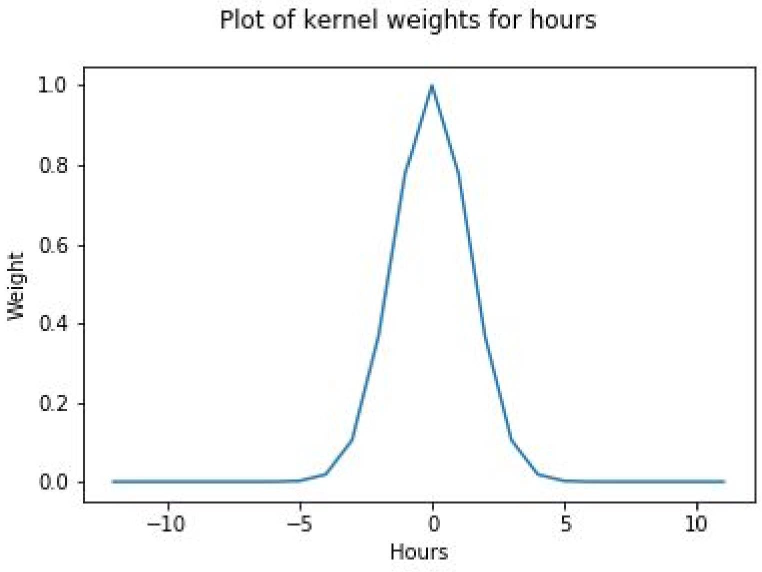
chrvo878
wilan057
2020-05-19

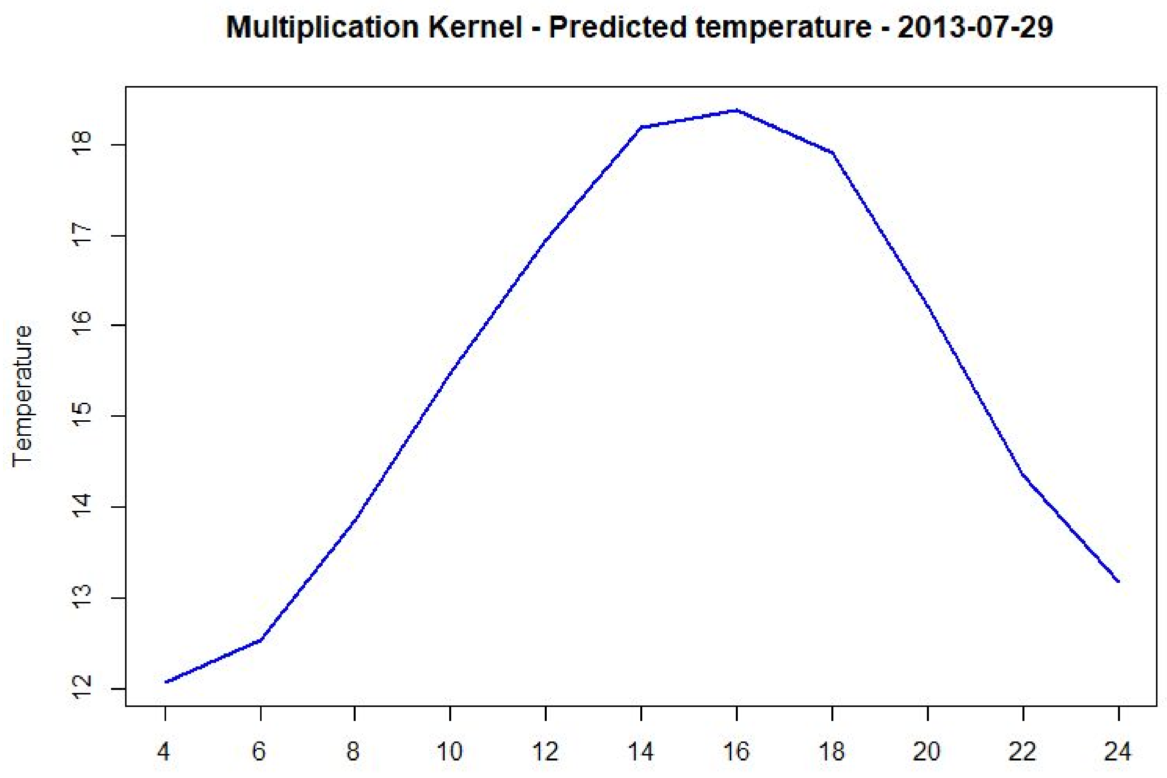
Lab report 3 - Big Data Analytics TDDE31

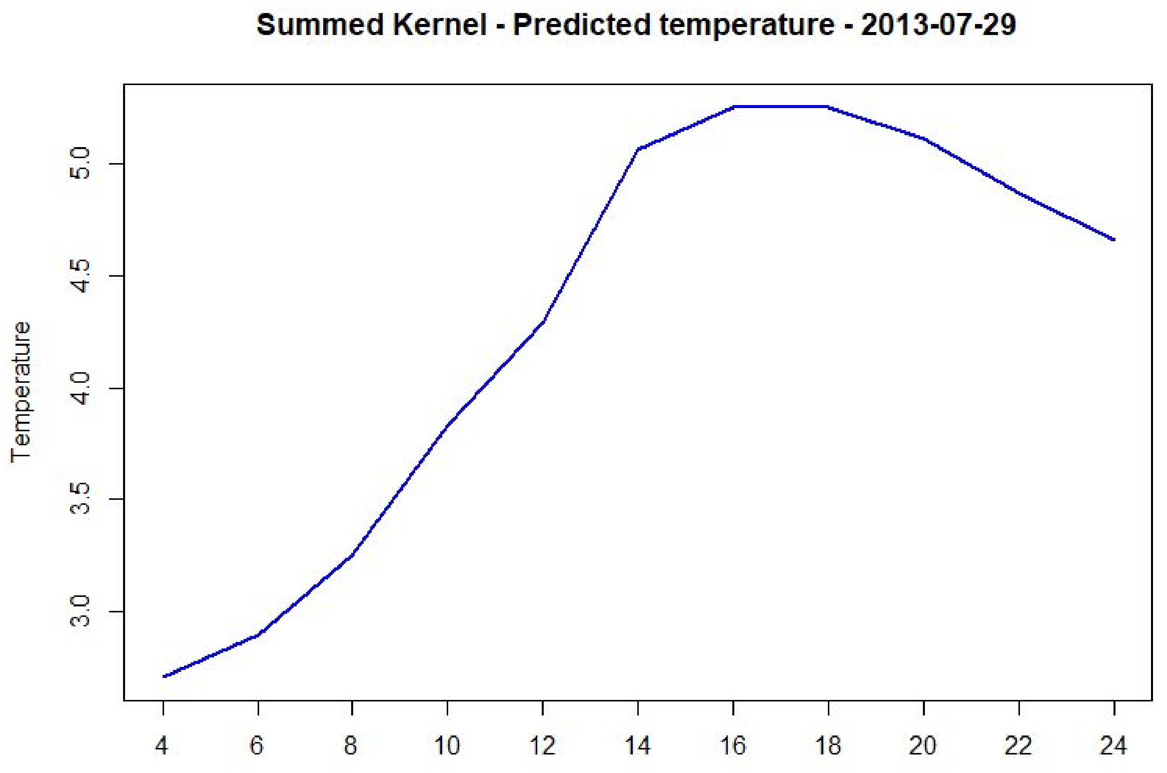






By examining the graphs above we can motivate the chosen h-values. Looking at the distance-graph, we see that the weights are significantly decreased (0.4) when at a distance of around 100 000 metres from our set location. This seems reasonable as it shows that the function takes the distance into account when predicting the temperature. Regarding the date-kernel, it seems reasonable that we only take measurements within 30 days of our prediction-date into account since the weather changes quite rapidly over the seasons in Sweden. When studying the time-kernel it seems reasonable to only take measurements within 5 hours of our predicted-time into account as the difference in temperature can vary quite a lot in 5 hours of time. The kernel takes all parameters into account when making the final prediction which makes sense.

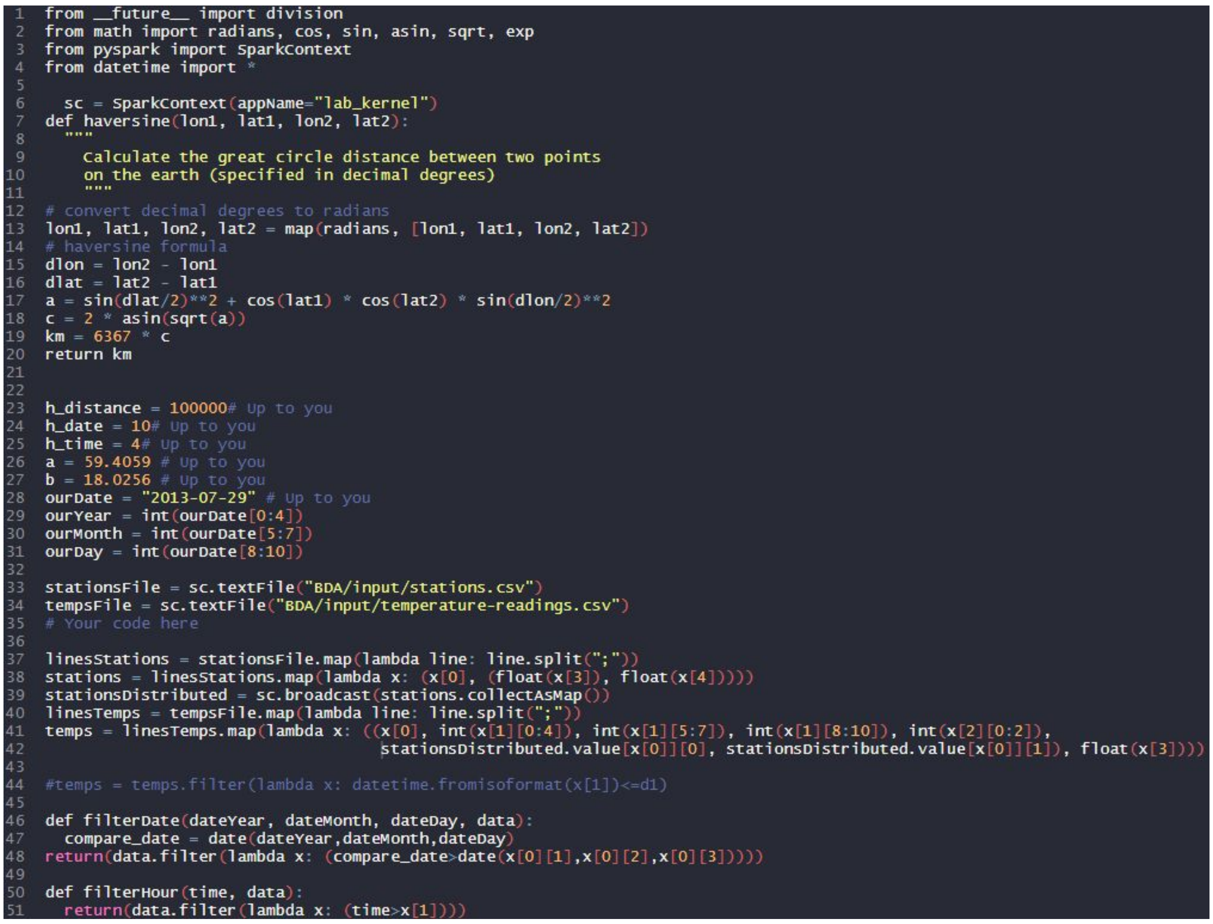




chrvo878
wilan057
2020-05-19
The difference between multiplying the kernels and adding them are clear. When multiplying the kernels the prediction becomes more sensitive to low values in any of the kernels, as it only takes one of the kernels to be close to 0 for the multiplied weight to obtain a value close to 0. Multiplying the kernels therefore makes the model dependent on all the factors. This makes the model more accurate as we make sure the observations that impact the prediction are highly significant, compared to the sum where an observation with one of the kernels equal to 0 can still impact our prediction. For example if we are to make a prediction in the north of Sweden, an observation taken in the most southern part can still have an effect on our prediction in the addition-kernel, while for the multiplication-kernel it will not have any significant effect.

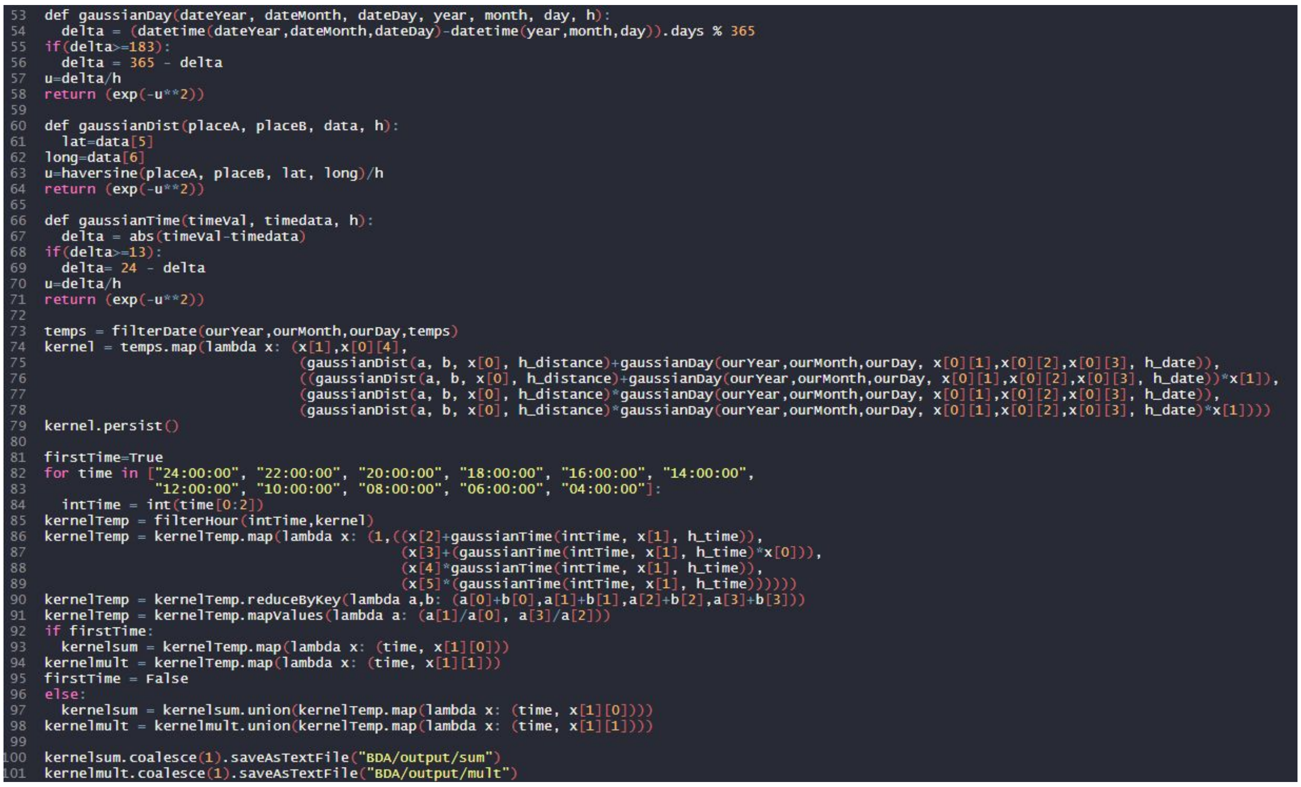
Appendix:

Picture:



```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from pyspark import SparkContext
from datetime import *

  sc = SparkContext(appName="lab_kernel")
def haversine(lon1, lat1, lon2, lat2):
  """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
# convert decimal degrees to radians
lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
# haversine formula
dlon = lon2 - lon1
dlat = lat2 - lat1
a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
c = 2 * asin(sqrt(a))
km = 6367 * c
return km


h_distance = 100000# Up to you
h_date = 10# Up to you
h_time = 4# Up to you
a = 59.4059 # Up to you
b = 18.0256 # Up to you
ourDate = "2013-07-29" # Up to you
ourYear = int(ourDate[0:4])
ourMonth = int(ourDate[5:7])
ourDay = int(ourDate[8:10])

stationsFile = sc.textFile("BDA/input/stations.csv")
tempsFile = sc.textFile("BDA/input/temperature-readings.csv")
# Your code here

linesStations = stationsFile.map(lambda line: line.split(";"))
stations = linesStations.map(lambda x: (x[0], (float(x[3]), float(x[4]))))
stationsDistributed = sc.broadcast(stations.collectAsMap())
linesTemps = tempsFile.map(lambda line: line.split(";"))
temps = linesTemps.map(lambda x: ((x[0], int(x[1][0:4]), int(x[1][5:7]), int(x[1][8:10]), int(x[2][0:2]),
                                   stationsDistributed.value[x[0]][0], stationsDistributed.value[x[0]][1]), float(x[3])))

#temps = temps.filter(lambda x: datetime.fromisoformat(x[1])<=d1)

def filterDate(dateYear, dateMonth, dateDay, data):
  compare_date = date(dateYear,dateMonth,dateDay)
return(data.filter(lambda x: (compare_date>date(x[0][1],x[0][2],x[0][3]))))

def filterHour(time, data):
  return(data.filter(lambda x: (time>x[1])))
```

chrvo878
wilan057
2020-05-19



```
def gaussianDay(dateYear, dateMonth, dateDay, year, month, day, h):
  delta = (datetime(dateYear,dateMonth,dateDay)-datetime(year,month,day)).days % 365
if(delta>=183):
  delta = 365 - delta
u=delta/h
return (exp(-u**2))

def gaussianDist(placeA, placeB, data, h):
  lat=data[5]
long=data[6]
u=haversine(placeA, placeB, lat, long)/h
return (exp(-u**2))

def gaussianTime(timeVal, timedata, h):
  delta = abs(timeVal-timedata)
if(delta>=13):
  delta= 24 - delta
u=delta/h
return (exp(-u**2))

temps = filterDate(ourYear,ourMonth,ourDay,temps)
kernel = temps.map(lambda x: (x[1],x[0][4],
                              (gaussianDist(a, b, x[0], h_distance)+gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date)),
                              ((gaussianDist(a, b, x[0], h_distance)+gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date))*x[1]),
                              (gaussianDist(a, b, x[0], h_distance)*gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date)),
                              (gaussianDist(a, b, x[0], h_distance)*gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date)*x[1])))
kernel.persist()

firstTime=True
for time in ["24:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
             "12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:
  intTime = int(time[0:2])
kernelTemp = filterHour(intTime,kernel)
kernelTemp = kernelTemp.map(lambda x: (1,((x[2]+gaussianTime(intTime, x[1], h_time)),
                                         (x[3]+(gaussianTime(intTime, x[1], h_time)*x[0])),
                                         (x[4]*gaussianTime(intTime, x[1], h_time)),
                                         (x[5]*(gaussianTime(intTime, x[1], h_time))))))
kernelTemp = kernelTemp.reduceByKey(lambda a,b: (a[0]+b[0],a[1]+b[1],a[2]+b[2],a[3]+b[3]))
kernelTemp = kernelTemp.mapValues(lambda a: (a[1]/a[0], a[3]/a[2]))
if firstTime:
  kernelsum = kernelTemp.map(lambda x: (time, x[1][0]))
kernelmult = kernelTemp.map(lambda x: (time, x[1][1]))
firstTime = False
else:
  kernelsum = kernelsum.union(kernelTemp.map(lambda x: (time, x[1][0])))
kernelmult = kernelmult.union(kernelTemp.map(lambda x: (time, x[1][1])))

kernelsum.coalesce(1).saveAsTextFile("BDA/output/sum")
kernelmult.coalesce(1).saveAsTextFile("BDA/output/mult")
```

Text:

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from pyspark import SparkContext
from datetime import *

sc = SparkContext(appName="lab_kernel")
def haversine(lon1, lat1, lon2, lat2):
   """
   Calculate the great circle distance between two points
   on the earth (specified in decimal degrees)
   """
   # convert decimal degrees to radians
   lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
   # haversine formula
   dlon = lon2 - lon1
   dlat = lat2 - lat1
   a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
   c = 2 * asin(sqrt(a))
   km = 6367 * c
   return km


h_distance = 100000# Up to you
h_date = 10# Up to you
h_time = 4# Up to you
```

chrvo878
wilan057
2020-05-19

```
a = 59.4059 # Up to you
b = 18.0256 # Up to you
ourDate = "2013-07-29" # Up to you
ourYear = int(ourDate[0:4])
ourMonth = int(ourDate[5:7])
ourDay = int(ourDate[8:10])

stationsFile = sc.textFile("BDA/input/stations.csv")
tempsFile = sc.textFile("BDA/input/temperature-readings.csv")
# Your code here

linesStations = stationsFile.map(lambda line: line.split(";"))
stations = linesStations.map(lambda x: (x[0], (float(x[3]), float(x[4]))))
stationsDistributed = sc.broadcast(stations.collectAsMap())
linesTemps = tempsFile.map(lambda line: line.split(";"))
temps = linesTemps.map(lambda x: ((x[0], int(x[1][0:4]), int(x[1][5:7]), int(x[1][8:10]),
int(x[2][0:2]), stationsDistributed.value[x[0]][0], stationsDistributed.value[x[0]][1]), float(x[3])))

#temps = temps.filter(lambda x: datetime.fromisoformat(x[1])<=d1)

def filterDate(dateYear, dateMonth, dateDay, data):
   compare_date = date(dateYear,dateMonth,dateDay)
   return(data.filter(lambda x: (compare_date>date(x[0][1],x[0][2],x[0][3]))))

def filterHour(time, data):
   return(data.filter(lambda x: (time>x[1])))

def gaussianDay(dateYear, dateMonth, dateDay, year, month, day, h):
   delta = (datetime(dateYear,dateMonth,dateDay)-datetime(year,month,day)).days % 365
   if(delta>=183):
      delta = 365 - delta
   u=delta/h
   return (exp(-u**2))

def gaussianDist(placeA, placeB, data, h):
   lat=data[5]
   long=data[6]
   u=haversine(placeA, placeB, lat, long)/h
   return (exp(-u**2))

def gaussianTime(timeVal, timedata, h):
   delta = abs(timeVal-timedata)
   if(delta>=13):
      delta= 24 - delta
   u=delta/h
```

chrvo878
wilan057
2020-05-19

```
    return (exp(-u**2))

temps = filterDate(ourYear,ourMonth,ourDay,temps)
kernel = temps.map(lambda x: (x[1],x[0][4],
                              (gaussianDist(a, b, x[0],
h_distance)+gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date)),
                               ((gaussianDist(a, b, x[0],
h_distance)+gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date))*x[1]),
                               (gaussianDist(a, b, x[0],
h_distance)*gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date)),
                            (gaussianDist(a, b, x[0],
h_distance)*gaussianDay(ourYear,ourMonth,ourDay, x[0][1],x[0][2],x[0][3], h_date)*x[1])))
kernel.persist()

firstTime=True
for time in ["24:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:
    intTime = int(time[0:2])
    kernelTemp = filterHour(intTime,kernel)
    kernelTemp = kernelTemp.map(lambda x: (1,((x[2]+gaussianTime(intTime, x[1], h_time)),
                                    (x[3]+(gaussianTime(intTime, x[1], h_time)*x[0])),
                                   (x[4]*gaussianTime(intTime, x[1], h_time)),
                                    (x[5]*(gaussianTime(intTime, x[1], h_time))))))
    kernelTemp = kernelTemp.reduceByKey(lambda a,b:
(a[0]+b[0],a[1]+b[1],a[2]+b[2],a[3]+b[3]))
    kernelTemp = kernelTemp.mapValues(lambda a: (a[1]/a[0], a[3]/a[2]))
    if firstTime:
        kernelsum = kernelTemp.map(lambda x: (time, x[1][0]))
        kernelmult = kernelTemp.map(lambda x: (time, x[1][1]))
        firstTime = False
    else:
        kernelsum = kernelsum.union(kernelTemp.map(lambda x: (time, x[1][0])))
        kernelmult = kernelmult.union(kernelTemp.map(lambda x: (time, x[1][1])))

kernelsum.coalesce(1).saveAsTextFile("BDA/output/sum")
kernelmult.coalesce(1).saveAsTextFile("BDA/output/mult")
```