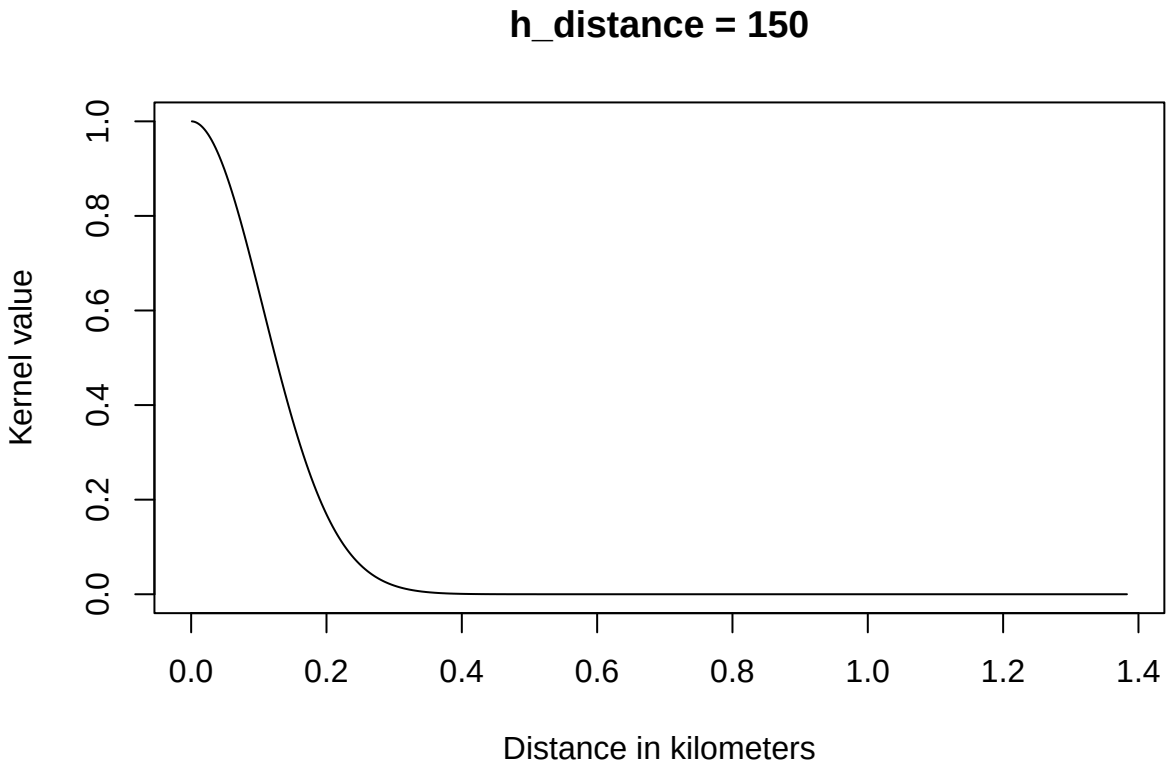
BDAML

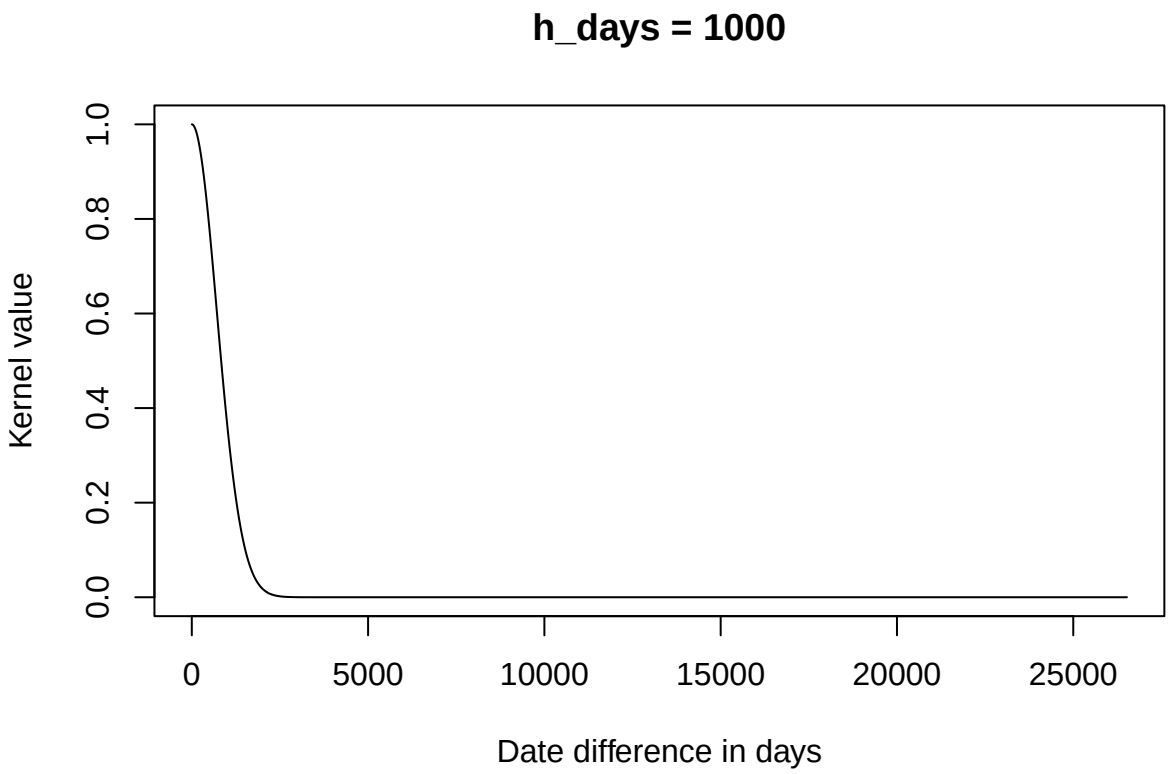
Adesijibomi Aderinto{adead268}

6/13/2022

```
h_distance = 150
h_days = 1000

##code to read the kernel output values and merge them into one file
listfile <- list.files(pattern = "part-*",full.names = T, recursive = TRUE)

for (i in 1:length(listfile)){
if(i==1){
  assign(paste0("Data"), read.table(listfile[i],header = FALSE, sep = ","))
}

if(!i==1){

  assign(paste0("Test",i), read.table(listfile[i],header = FALSE, sep = ","))
  Data <- rbind(Data,get(paste0("Test",i)))
  rm(list = ls(pattern = "Test"))
}
}

plot((seq(min(Data$V1):max(Data$V1)) / 1000),
     exp(-(seq(min(Data$V1):max(Data$V1)) / h_distance)^2),
     type = "l", xlab = "Distance in kilometers", ylab = "Kernel value", main = "h_distance = 150")
```

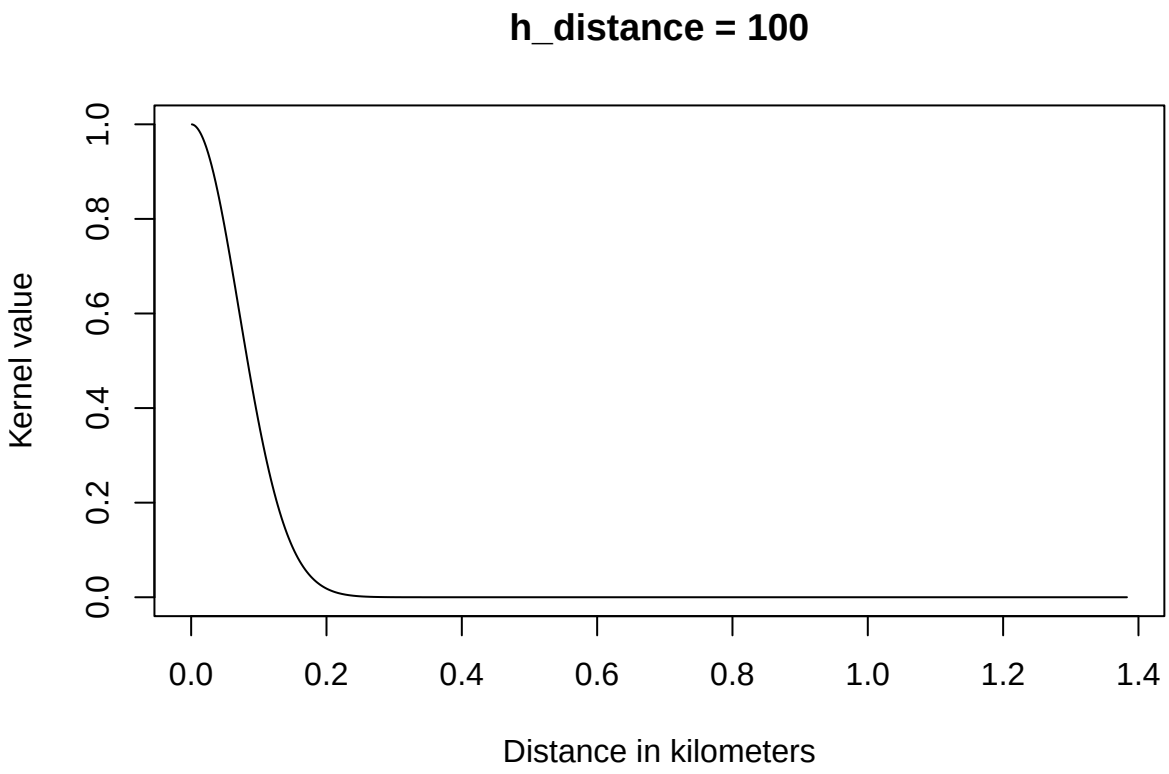



```
plot(seq(min(Data$V2):max(Data$V2)),
     exp(-(seq(min(Data$V2):max(Data$V2)) / h_days)^2),
     type = "l", xlab = "Date difference in days", ylab = "Kernel value",main = "h_days = 1000")
```

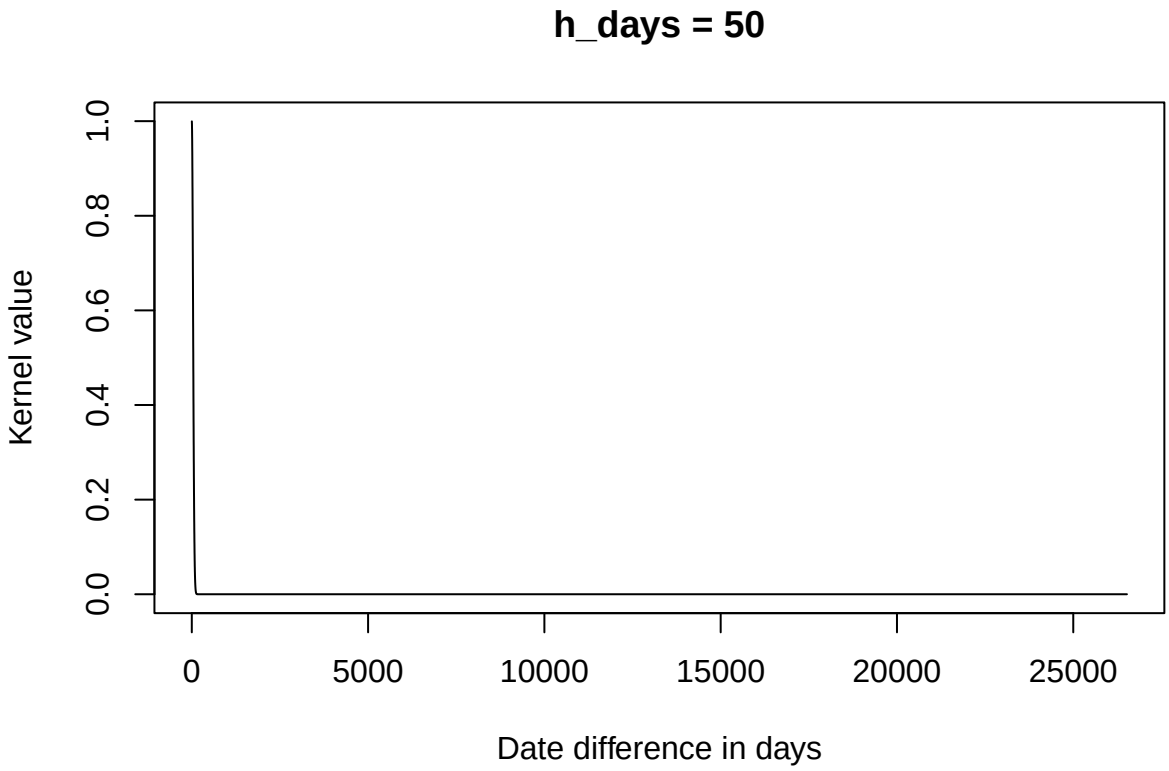


comment Plots above shows the effect of kernel values hows it affects the difference in H values. At a distance of 0.3km it tends to have no impact on the kernel, i.e low weights are assigned at larger distances which then to have no impact on the kernel probabilities, same can also be seen plot of days kernels

```
h_distance = 100
h_days = 50

plot((seq(min(Data$V1):max(Data$V1)) / 1000),
     exp(-(seq(min(Data$V1):max(Data$V1)) / h_distance)^2),
     type = "l", xlab = "Distance in kilometers", ylab = "Kernel value",main = "h_distance = 100")
```




```
plot(seq(min(Data$V2):max(Data$V2)),
     exp(-(seq(min(Data$V2):max(Data$V2)) / h_days)^2),
     type = "l", xlab = "Date difference in days", ylab = "Kernel value",main = "h_days = 50")
```



comments Plots showing different h_values for kernels

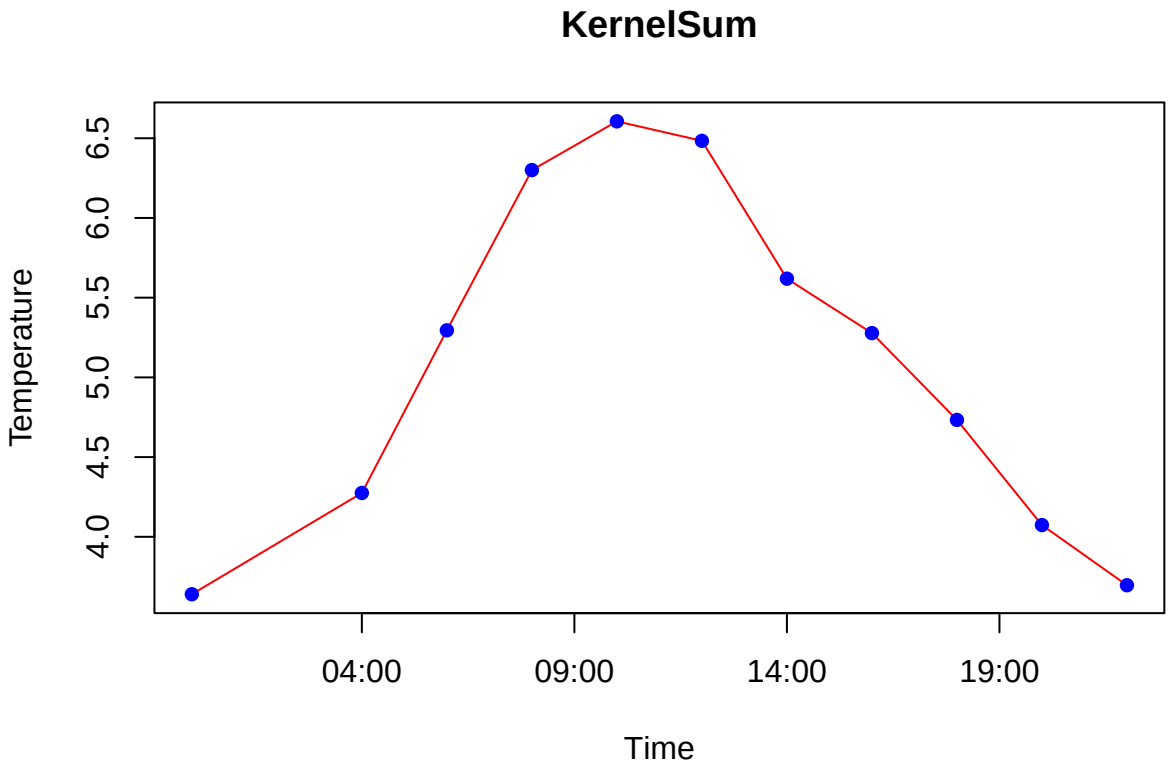
```
('04:00:00', 3.696250976132099)
('06:00:00', 3.6397725698648054)
('08:00:00', 4.27492877896444)
('10:00:00', 5.2951841336254315)
('12:00:00', 6.300181847439305)
('14:00:00', 6.605957535647397)
('16:00:00', 6.483739105179379)
('18:00:00', 5.619149929427509)
('20:00:00', 5.2776748871799395)
('22:00:00', 4.733071911757403)
('00:00:00', 4.073633050827442)
(serviceOption=None,
 services=List(),
 started=false)
```

Figure 1: Kernel Sum

```r
data_sum <- data.frame(times =c("00:00:00","04:00:00","06:00:00","08:00:00",
    predictions = c( 3.6397725698648054,  4.27492877896444,
                     5.2951841336254315, 6.300181847439305, 6.605957535647397,
                     6.483739105179379,  5.619149929427509, 5.2776748871799395,
                     4.733071911757403, 4.073633050827442,3.696250976132099))

data_sum
```

```
##       times predictions
## 1  00:00:00    3.639773
## 2  04:00:00    4.274929
## 3  06:00:00    5.295184
## 4  08:00:00    6.300182
## 5  10:00:00    6.605958
## 6  12:00:00    6.483739
## 7  14:00:00    5.619150
## 8  16:00:00    5.277675
## 9  18:00:00    4.733072
## 10 20:00:00    4.073633
## 11 22:00:00    3.696251
```

```r
plot(as.POSIXct(data_sum$times, format = "%H:%M:%OS"), data_sum$predictions,
     type = "l",xlab = "Time", ylab = "Temperature",ylim = c(min(data_sum[,2]),
                              max(data_sum[,2])),col = "red",main = "KernelSum")
points(as.POSIXct(data_sum$times, format = "%H:%M:%OS"),
       data_sum$predictions,pch = 16, col = "Blue")
```



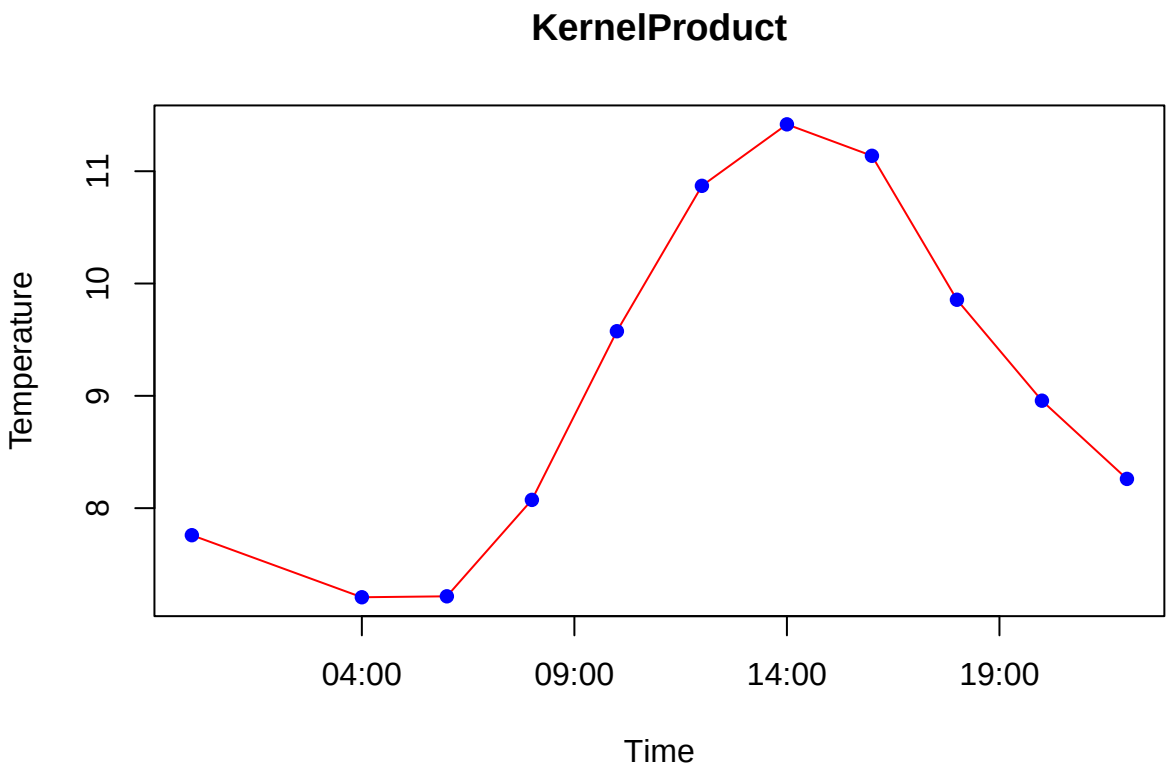

```
('04:00:00', 7.206854162532228)
('06:00:00', 7.2149717943017375)
('08:00:00', 8.073657761403423)
('10:00:00', 9.575235071996133)
('12:00:00', 10.869689622014663)
('14:00:00', 11.417215048023012)
('16:00:00', 11.137205806368883)
('18:00:00', 9.855527323963832)
('20:00:00', 8.956740656542651)
('22:00:00', 8.260453738271963)
('00:00:00', 7.75915580682052 4)
(serviceOption=None,
 services=List(),
 started=false)
```

Figure 2: Kernel Sum

```r
data_product <- data.frame(times =c("00:00:00","04:00:00","06:00:00","08:00:00",
    predictions = c(7.759155806820524, 7.206854162532228,
                    7.2149717943017375, 8.073657761403423, 9.575235071996133,
                    10.869689622014663, 11.417215048023012, 11.137205806368883,
                    9.855527323963832, 8.956740656542651,  8.260453738271963))

data_product
```

```
##       times predictions
## 1  00:00:00    7.759156
## 2  04:00:00    7.206854
## 3  06:00:00    7.214972
## 4  08:00:00    8.073658
## 5  10:00:00    9.575235
## 6  12:00:00   10.869690
## 7  14:00:00   11.417215
## 8  16:00:00   11.137206
## 9  18:00:00    9.855527
## 10 20:00:00    8.956741
## 11 22:00:00    8.260454
```

```r
plot(as.POSIXct(data_product$times, format = "%H:%M:%OS"), data_product$predictions,
     type = "l",xlab = "Time", ylab = "Temperature",ylim = c(min(data_product[,2]),
                              max(data_product[,2])),col = "red",main = "KernelProduct")
points(as.POSIXct(data_product$times, format = "%H:%M:%OS"),
       data_product$predictions,pch = 16, col = "Blue")
```



comments When comparing summation of kernels, if only one of the kernel values(h_distance,h_days,h_time)is high it will affect the overall kernel value. while as for the product of kernel, all three h_values must be high for the overall effect in kernel values to be seen. Therefore kernel sum is better.

Appendix

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

sc = SparkContext(appName="ML Lab3")

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
```

```
# init values
h_distance = 150 # Up to you
h_date  = 30 #  Up  to  you
h_time = 3 # Up to you

lon2 = 58.4274 # Up to you
lat2 = 14.826 # Up to you

times=["04:00:00", "06:00:00","08:00:00" ,"10:00:00","12:00:00","14:00:00",
       "16:00:00","18:00:00","20:00:00","22:00:00","00:00:00"]

date = "2013-11-04"

temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
stations_file = sc.textFile("BDA/input/stations.csv")

temps = temperature_file.map(lambda line: line.split(";"))
station = stations_file.map(lambda line: line.split(";"))

def as_date(date):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date, date_format)
    return a

def gaussian_kernel(diff, h):
    return exp(-(diff / h) ** 2)

dist_days = temps.map(lambda x:(x[0],(x[1]
                                ,x[2],float(x[3])))).filter(lambda x: as_date(x[1][0])
                                                            < as_date(date)).cache()

stations = station.map(lambda x: ((x[0],(float(x[3]),float(x[4])))))

stations = stations.collectAsMap()
bc = sc.broadcast(stations)
joined = dist_days.map(lambda x: (x[0], x[1],bc.value.get(x[0])))
kernel = joined.map(lambda x: (x[0],haversine(lon2,lat2,float(x[2][0]),float(x[2][1])),
                               (datetime.strptime(date,"%Y-%m-%d") -
                                  datetime.strptime(x[1][0],"%Y-%m-%d")).days,
                               x[1][2])).cache()
res = kernel.map(lambda x: (x[1],x[2],x[3]))
result = res.map(lambda x: str(x).replace('(', '').replace(')', ''))
result.sample(False, 0.001).saveAsTextFile("BDA/output/kernel_values")
```

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

sc = SparkContext(appName="ML Lab3")

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

# init values
h_distance = 100 # Up to you
h_date  = 50 #  Up  to  you
h_time = 3 # Up to you

lon2 = 58.4274 # Up to you
lat2 = 14.826 # Up to you

times=["04:00:00", "06:00:00","08:00:00" ,"10:00:00","12:00:00","14:00:00",
       "16:00:00","18:00:00","20:00:00","22:00:00","00:00:00"]

date = "2013-11-04"

temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
stations_file = sc.textFile("BDA/input/stations.csv")

temps = temperature_file.map(lambda line: line.split(";"))
station = stations_file.map(lambda line: line.split(";"))

def as_date(date):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date, date_format)
    return a

def gaussian_kernel(diff, h):
    return exp(-(diff / h) ** 2)

dist_days = temps.map(lambda x:(x[0],(x[1],
                                    x[2],float(x[3])))).filter(lambda x: as_date(x[1][0])
                                                               < as_date(date)).cache()

stations = station.map(lambda x: ((x[0],(float(x[3]),float(x[4])))))

stations = stations.collectAsMap()
bc = sc.broadcast(stations)
joined = dist_days.map(lambda x: (x[0], x[1],bc.value.get(x[0])))
kernel = joined.map(lambda x: (x[0],haversine(lon2,lat2,float(x[2][0]),float(x[2][1])),
                               (datetime.strptime(date,"%Y-%m-%d") -
                                  datetime.strptime(x[1][0], "%Y-%m-%d"))
                               .days,x[1][1],x[1][2]))
kernel_gaussian = kernel.map(lambda x: (gaussian_kernel(x[1],h_distance),
                                        gaussian_kernel(x[2], h_date),x[3],x[4])).cache()
```

```
for time in times:
    results = dict()
    kernel_rdd =  kernel_gaussian.map(lambda x: (x[0],x[1],
                                                 (datetime.strptime(time,'%H:%M:%S')
                                                  - datetime.strptime(x[2],'%H:%M:%S'))
                                                 .seconds/3600,float(x[3]))).cache()
    gaussian_time = kernel_rdd.map(lambda x: (x[3],x[0],x[1],
                                              gaussian_kernel(x[2],h_time))).cache()
    kernel_sum = gaussian_time.map(lambda x: (x[0], x[1] + x[2] + x[3]))
    kernel_agg = kernel_sum.map(lambda x: (float(x[0])*x[1],x[1]))
    reduced=kernel_agg.reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
    results=reduced[0]/reduced[1]
    print(time,results)
```

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

sc = SparkContext(appName="ML Lab3")

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

# init values
h_distance = 100 # Up to you
h_date  = 50 #  Up  to  you
h_time = 3 # Up to you

lon2 = 58.4274 # Up to you
lat2 = 14.826 # Up to you

times=["04:00:00", "06:00:00","08:00:00" ,"10:00:00","12:00:00","14:00:00",
       "16:00:00","18:00:00","20:00:00","22:00:00","00:00:00"]

date = "2013-11-04"

temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
stations_file = sc.textFile("BDA/input/stations.csv")

temps = temperature_file.map(lambda line: line.split(";"))
station = stations_file.map(lambda line: line.split(";"))
```

```
def as_date(date):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date, date_format)
    return a

def gaussian_kernel(diff, h):
    return exp(-(diff / h) ** 2)

dist_days = temps.map(lambda x:(x[0],(x[1]
                                      ,x[2],float(x[3])))).filter(lambda x: as_date(x[1][0])
                                                                  < as_date(date)).cache()

stations = station.map(lambda x: ((x[0],(float(x[3]),float(x[4])))))

stations = stations.collectAsMap()
bc = sc.broadcast(stations)
joined = dist_days.map(lambda x: (x[0], x[1],bc.value.get(x[0])))
kernel = joined.map(lambda x: (x[0],haversine(lon2,lat2,float(x[2][0]),float(x[2][1])),
                               (datetime.strptime(date,"%Y-%m-%d")
                                - datetime.strptime(x[1][0],"%Y-%m-%d")).
                               days,x[1][1],x[1][2]))
kernel_gaussian = kernel.map(lambda x: (gaussian_kernel(x[1],h_distance),
                                        gaussian_kernel(x[2], h_date),x[3],x[4])).cache()
for time in times:
    results = dict()
    kernel_rdd =  kernel_gaussian.map(lambda x: (x[0],x[1],
                                                 (datetime.strptime(time,'%H:%M:%S')
                                                  - datetime.strptime(x[2],'%H:%M:%S'))
                                                 .seconds/3600,float(x[3]))).cache()
    gaussian_time = kernel_rdd.map(lambda x: (x[3],x[0],x[1],
                                              gaussian_kernel(x[2],h_time))).cache()
    kernel_product = gaussian_time.map(lambda x: (x[0], x[1] * x[2] * x[3]))
    kernel_agg = kernel_product.map(lambda x: (float(x[0])*x[1],x[1]))
    reduced=kernel_agg.reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
    results=reduced[0]/reduced[1]
    print(time,results)
```