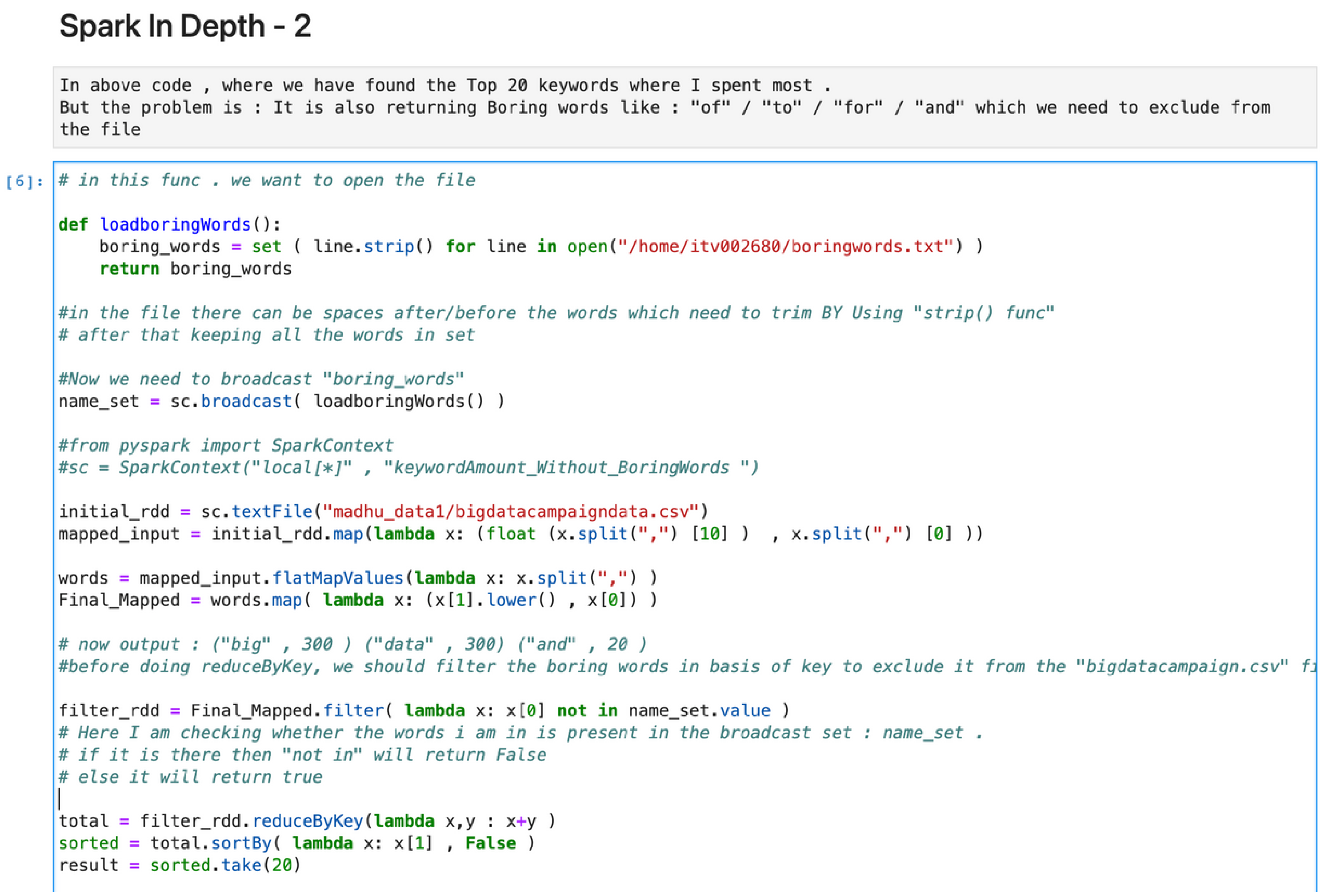
Apache Spark Practicals

DATA

DATA ENGINEER



Spark In Depth - 2

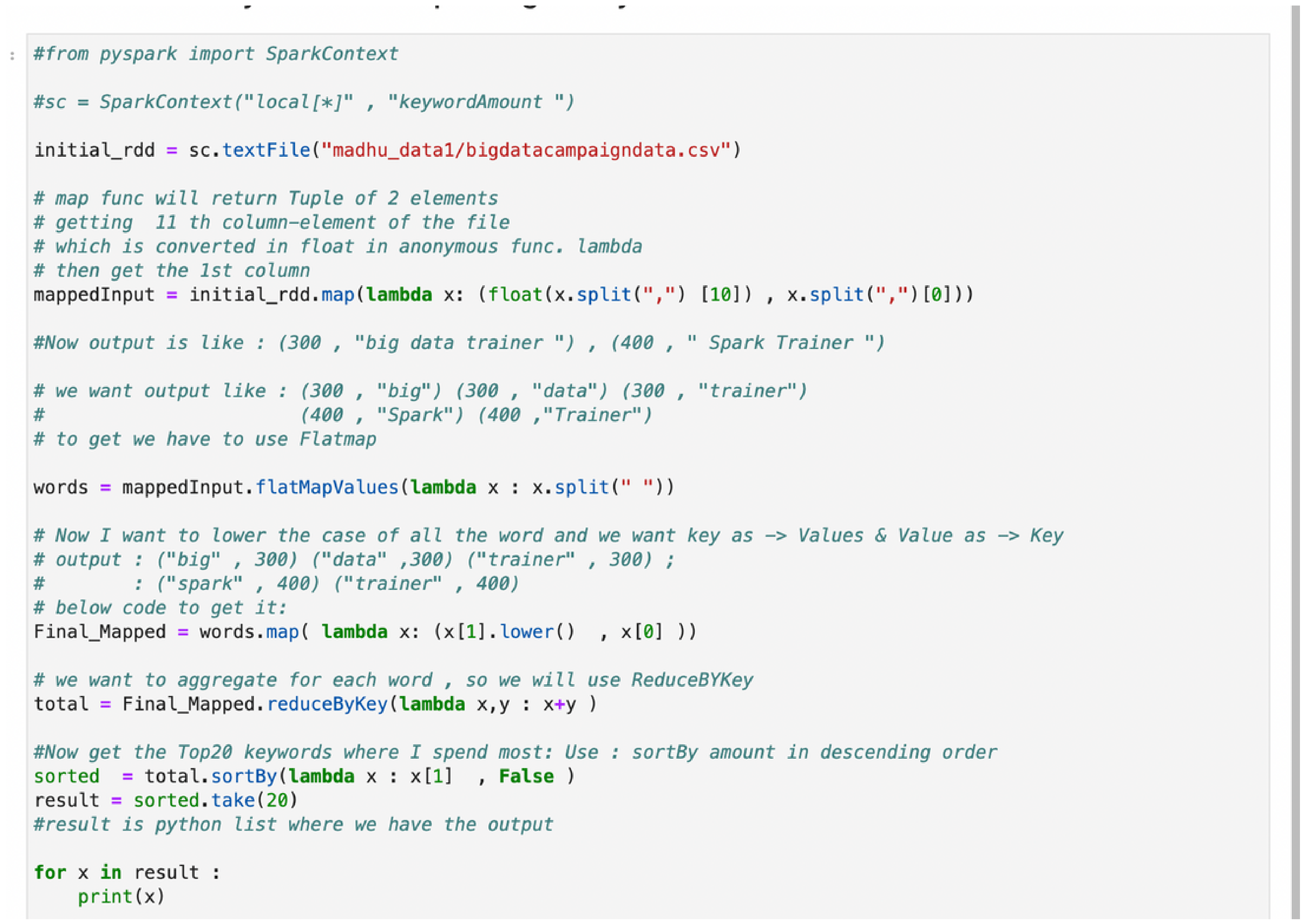
```
In above code , where we have found the Top 20 keywords where I spent most .
But the problem is : It is also returning Boring words like : "of" / "to" / "for" / "and" which we need to exclude from
the file
```

[6]:

```python
# in this func . we want to open the file

def loadboringWords():
    boring_words = set ( line.strip() for line in open("/home/itv002680/boringwords.txt") )
    return boring_words

#in the file there can be spaces after/before the words which need to trim BY Using "strip() func"
# after that keeping all the words in set

#Now we need to broadcast "boring_words"
name_set = sc.broadcast( loadboringWords() )

#from pyspark import SparkContext
#sc = SparkContext("local[*]" , "keywordAmount_Without_BoringWords ")

initial_rdd = sc.textFile("madhu_data1/bigdatacampaigndata.csv")
mapped_input = initial_rdd.map(lambda x: (float (x.split(",") [10] )  , x.split(",") [0] ))

words = mapped_input.flatMapValues(lambda x: x.split(",") )
Final_Mapped = words.map( lambda x: (x[1].lower() , x[0]) )

# now output : ("big" , 300 ) ("data" , 300) ("and" , 20 )
#before doing reduceByKey, we should filter the boring words in basis of key to exclude it from the "bigdatacampaign.csv" fi

filter_rdd = Final_Mapped.filter( lambda x: x[0] not in name_set.value )
# Here I am checking whether the words i am in is present in the broadcast set : name_set .
# if it is there then "not in" will return False
# else it will return true

total = filter_rdd.reduceByKey(lambda x,y : x+y )
sorted = total.sortBy( lambda x: x[1] , False )
result = sorted.take(20)
```

Apache Spark Practicals

DATA ENGINEER



```python
#from pyspark import SparkContext

#sc = SparkContext("local[*]" , "keywordAmount ")

initial_rdd = sc.textFile("madhu_data1/bigdatacampaigndata.csv")

# map func will return Tuple of 2 elements
# getting  11 th column-element of the file
# which is converted in float in anonymous func. lambda
# then get the 1st column
mappedInput = initial_rdd.map(lambda x: (float(x.split(",") [10]) , x.split(",")[0]))

#Now output is like : (300 , "big data trainer ") , (400 , " Spark Trainer ")

# we want output like : (300 , "big") (300 , "data") (300 , "trainer")
#                       (400 , "Spark") (400 ,"Trainer")
# to get we have to use Flatmap

words = mappedInput.flatMapValues(lambda x : x.split(" "))

# Now I want to lower the case of all the word and we want key as -> Values & Value as -> Key
# output : ("big" , 300) ("data" ,300) ("trainer" , 300) ;
#        : ("spark" , 400) ("trainer" , 400)
# below code to get it:
Final_Mapped = words.map( lambda x: (x[1].lower()  , x[0] ))

# we want to aggregate for each word , so we will use ReduceBYKey
total = Final_Mapped.reduceByKey(lambda x,y : x+y )

#Now get the Top20 keywords where I spend most: Use : sortBy amount in descending order
sorted  = total.sortBy(lambda x : x[1]  , False )
result = sorted.take(20)
#result is python list where we have the output

for x in result :
    print(x)
```

Apache Spark Practicals

Architecture of Spark on Yarn In Client Mode:

1. When we launch spark shell then automatically spark session is created
2. As soon as spark session is created , request goes to the Yarn Resource Manager
3. Yarn will create a container on one of the Node-Managers and will launch an Application Master for this Spark Application
4. Now Application Master will negotiate for resources From the Yarn Resource Manager in form of container
5. The Yarn RM will create containers on the Node Managers
6. Now the Application Masters will Launch these executors in these containers
7. Now the drivers & executors can communicate directly w/o the involvement of containers