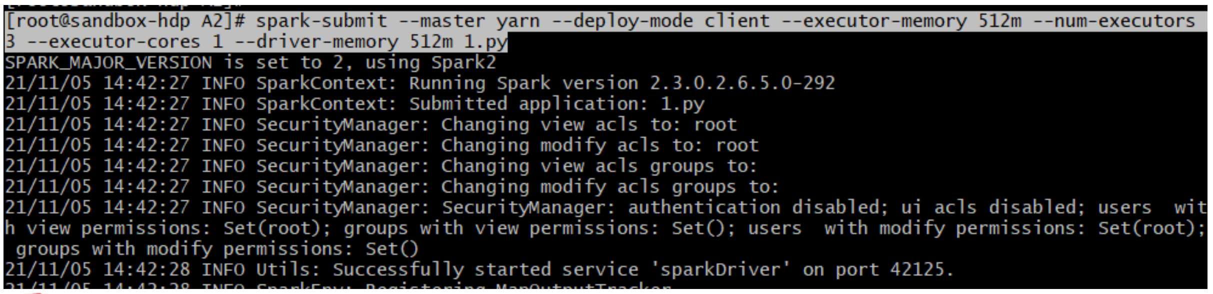
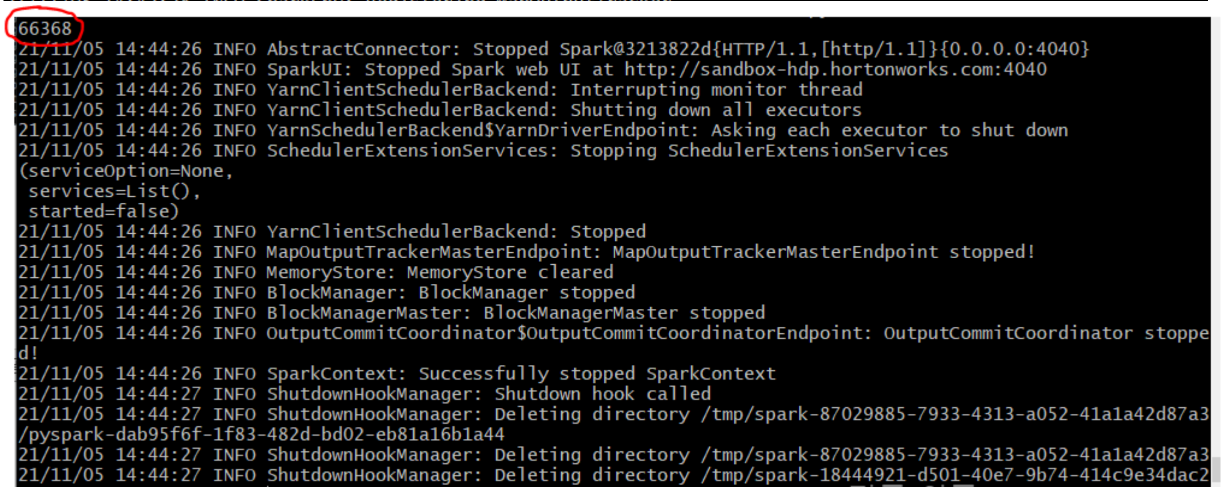
Q1. How many unique companies are present in the companies file? (10 pts)

```
from pyspark import SparkConf, SparkContext
def makeTuple(line):
    words = line.split('₩t')
    return (words[0]) # word[0] permalink    word[1]name
def main(sc):
    textFile = sc.textFile("/user/root/A2/companies.txt")
    #remove header
    header = textFile.first()
    textFile = textFile.filter(lambda line: line != header)
    wordList = textFile.map(lambda line: makeTuple(line))   #filter out header
    wordCount = wordList.map(lambda word: (word,1))
    reducedName = wordCount.reduceByKey(lambda v1,v2: v1+v2)
    print(reducedName.count())
if __name__   == "__main__":
    conf = SparkConf().setAppName("MyApp")
    sc = SparkContext.getOrCreate()
    main(sc)
    sc.stop()
```



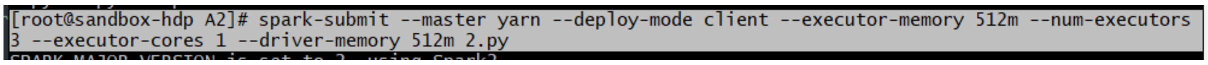
```
[root@sandbox-hdp A2]# spark-submit --master yarn --deploy-mode client --executor-memory 512m --num-executors
3 --executor-cores 1 --driver-memory 512m 1.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
21/11/05 14:42:27 INFO SparkContext: Running Spark version 2.3.0.2.6.5.0-292
21/11/05 14:42:27 INFO SparkContext: Submitted application: 1.py
21/11/05 14:42:27 INFO SecurityManager: Changing view acls to: root
21/11/05 14:42:27 INFO SecurityManager: Changing modify acls to: root
21/11/05 14:42:27 INFO SecurityManager: Changing view acls groups to:
21/11/05 14:42:27 INFO SecurityManager: Changing modify acls groups to:
21/11/05 14:42:27 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  wit
h view permissions: Set(root); groups with view permissions: Set(); users  with modify permissions: Set(root);
 groups with modify permissions: Set()
21/11/05 14:42:28 INFO Utils: Successfully started service 'sparkDriver' on port 42125.
```



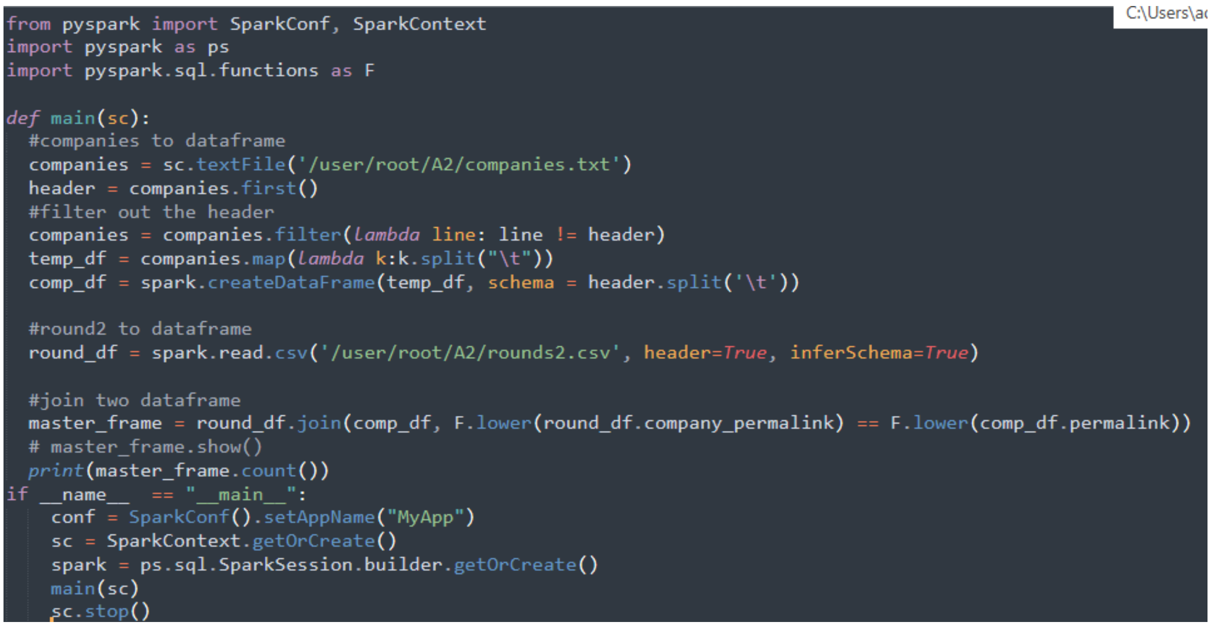
```
66368
21/11/05 14:44:26 INFO AbstractConnector: Stopped Spark@3213822d{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
21/11/05 14:44:26 INFO SparkUI: Stopped Spark web UI at http://sandbox-hdp.hortonworks.com:4040
21/11/05 14:44:26 INFO YarnClientSchedulerBackend: Interrupting monitor thread
21/11/05 14:44:26 INFO YarnClientSchedulerBackend: Shutting down all executors
21/11/05 14:44:26 INFO YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
21/11/05 14:44:26 INFO SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
 services=List(),
 started=false)
21/11/05 14:44:26 INFO YarnClientSchedulerBackend: Stopped
21/11/05 14:44:26 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/11/05 14:44:26 INFO MemoryStore: MemoryStore cleared
21/11/05 14:44:26 INFO BlockManager: BlockManager stopped
21/11/05 14:44:26 INFO BlockManagerMaster: BlockManagerMaster stopped
21/11/05 14:44:26 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stoppe
d!
21/11/05 14:44:26 INFO SparkContext: Successfully stopped SparkContext
21/11/05 14:44:27 INFO ShutdownHookManager: Shutdown hook called
21/11/05 14:44:27 INFO ShutdownHookManager: Deleting directory /tmp/spark-87029885-7933-4313-a052-41a1a42d87a3
/pyspark-dab95f6f-1f83-482d-bd02-eb81a16b1a44
21/11/05 14:44:27 INFO ShutdownHookManager: Deleting directory /tmp/spark-87029885-7933-4313-a052-41a1a42d87a3
21/11/05 14:44:27 INFO ShutdownHookManager: Deleting directory /tmp/spark-18444921-d501-40e7-9b74-414c9e34dac2
```

Q2. Merge the two data frames using inner join so that all variables (columns) in the companies frame are added to the rounds2 data frame. Name the merged frame master_frame. How many observations (rows) are present in master_frame? Hint: Find an attribute from both data frames that can serve as a unique key (10 pts)

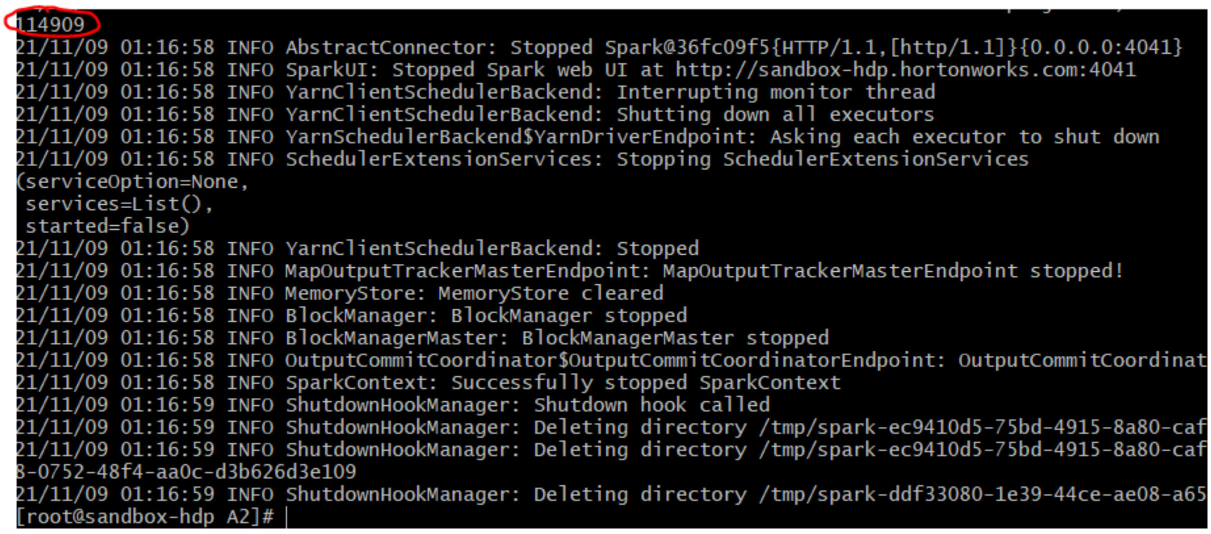
Ans: 114909 rows



```
[root@sandbox-hdp A2]# spark-submit --master yarn --deploy-mode client --executor-memory 512m --num-executors
3 --executor-cores 1 --driver-memory 512m 2.py
```



```python
from pyspark import SparkConf, SparkContext
import pyspark as ps
import pyspark.sql.functions as F

def main(sc):
  #companies to dataframe
  companies = sc.textFile('/user/root/A2/companies.txt')
  header = companies.first()
  #filter out the header
  companies = companies.filter(lambda line: line != header)
  temp_df = companies.map(lambda k:k.split("\t"))
  comp_df = spark.createDataFrame(temp_df, schema = header.split('\t'))

  #round2 to dataframe
  round_df = spark.read.csv('/user/root/A2/rounds2.csv', header=True, inferSchema=True)

  #join two dataframe
  master_frame = round_df.join(comp_df, F.lower(round_df.company_permalink) == F.lower(comp_df.permalink))
  # master_frame.show()
  print(master_frame.count())
if __name__ == "__main__":
    conf = SparkConf().setAppName("MyApp")
    sc = SparkContext.getOrCreate()
    spark = ps.sql.SparkSession.builder.getOrCreate()
    main(sc)
    sc.stop()
```

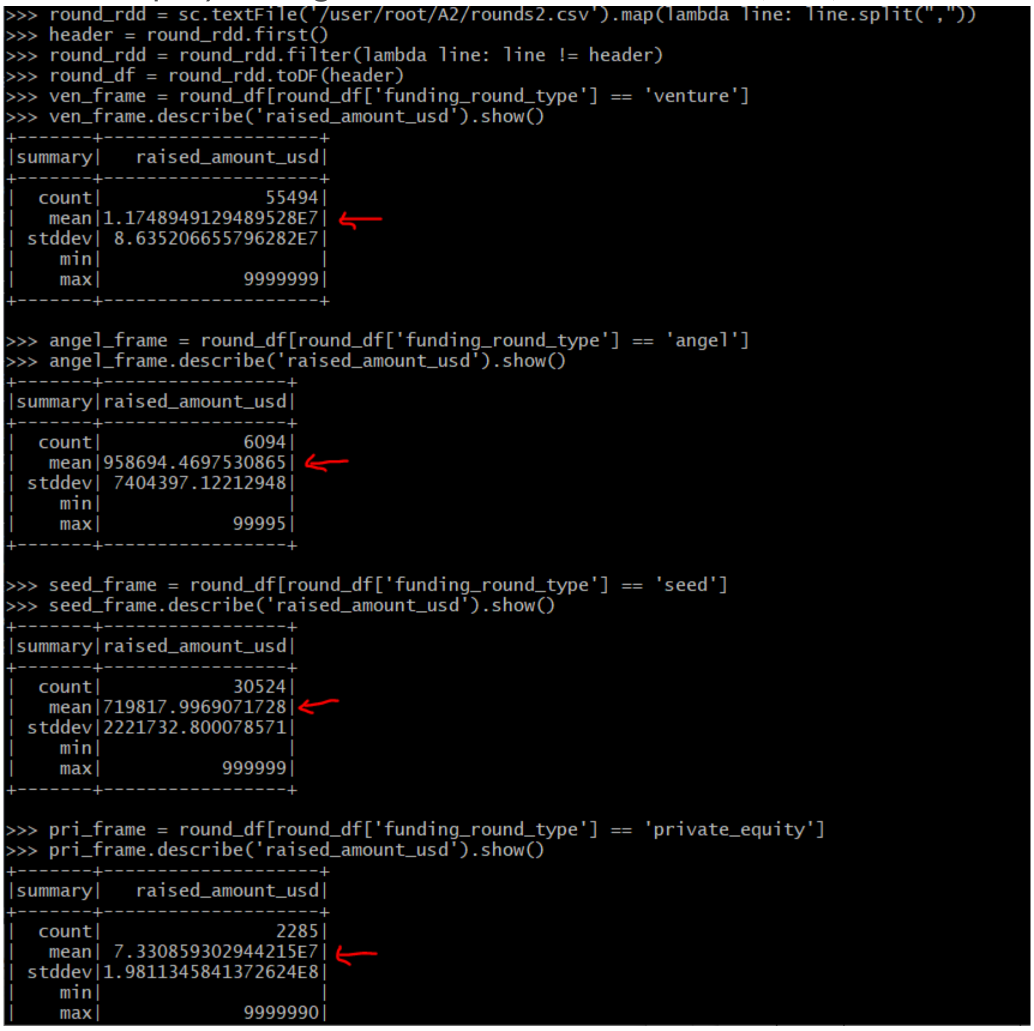


```
114909
21/11/09 01:16:58 INFO AbstractConnector: Stopped Spark@36fc09f5{HTTP/1.1,[http/1.1]}{0.0.0.0:4041}
21/11/09 01:16:58 INFO SparkUI: Stopped Spark web UI at http://sandbox-hdp.hortonworks.com:4041
21/11/09 01:16:58 INFO YarnClientSchedulerBackend: Interrupting monitor thread
21/11/09 01:16:58 INFO YarnClientSchedulerBackend: Shutting down all executors
21/11/09 01:16:58 INFO YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
21/11/09 01:16:58 INFO SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
 services=List(),
 started=false)
21/11/09 01:16:58 INFO YarnClientSchedulerBackend: Stopped
21/11/09 01:16:58 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/11/09 01:16:58 INFO MemoryStore: MemoryStore cleared
21/11/09 01:16:58 INFO BlockManager: BlockManager stopped
21/11/09 01:16:58 INFO BlockManagerMaster: BlockManagerMaster stopped
21/11/09 01:16:58 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinat
21/11/09 01:16:58 INFO SparkContext: Successfully stopped SparkContext
21/11/09 01:16:59 INFO ShutdownHookManager: Shutdown hook called
21/11/09 01:16:59 INFO ShutdownHookManager: Deleting directory /tmp/spark-ec9410d5-75bd-4915-8a80-caf
21/11/09 01:16:59 INFO ShutdownHookManager: Deleting directory /tmp/spark-ec9410d5-75bd-4915-8a80-caf
8-0752-48f4-aa0c-d3b626d3e109
21/11/09 01:16:59 INFO ShutdownHookManager: Deleting directory /tmp/spark-ddf33080-1e39-44ce-ae08-a65
[root@sandbox-hdp A2]# |
```

Q3. Calculate the average of the investment amount for each of the four funding types (venture, angel, seed, and private equity) (20 pts)

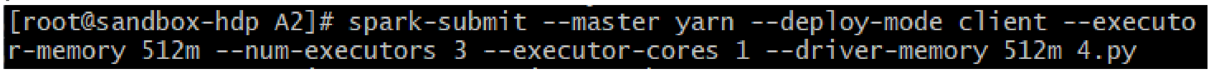
```
>>> round_rdd = sc.textFile('/user/root/A2/rounds2.csv').map(lambda line: line.split(","))
>>> header = round_rdd.first()
>>> round_rdd = round_rdd.filter(lambda line: line != header)
>>> round_df = round_rdd.toDF(header)
>>> ven_frame = round_df[round_df['funding_round_type'] == 'venture']
>>> ven_frame.describe('raised_amount_usd').show()
>>> angel_frame = round_df[round_df['funding_round_type'] == 'angel']
>>> angel_frame.describe('raised_amount_usd').show()
>>> seed_frame = round_df[round_df['funding_round_type'] == 'seed']
>>> seed_frame.describe('raised_amount_usd').show()
>>> pri_frame = round_df[round_df['funding_round_type'] == 'private_equity']
>>> pri_frame.describe('raised_amount_usd').show()
```

Venture average of investment amount: **11748949**
Angel average of investment amount: **958694.5**
Seed average of investment amount: **719818**
Private equity average of investment amount: **73,308,593**

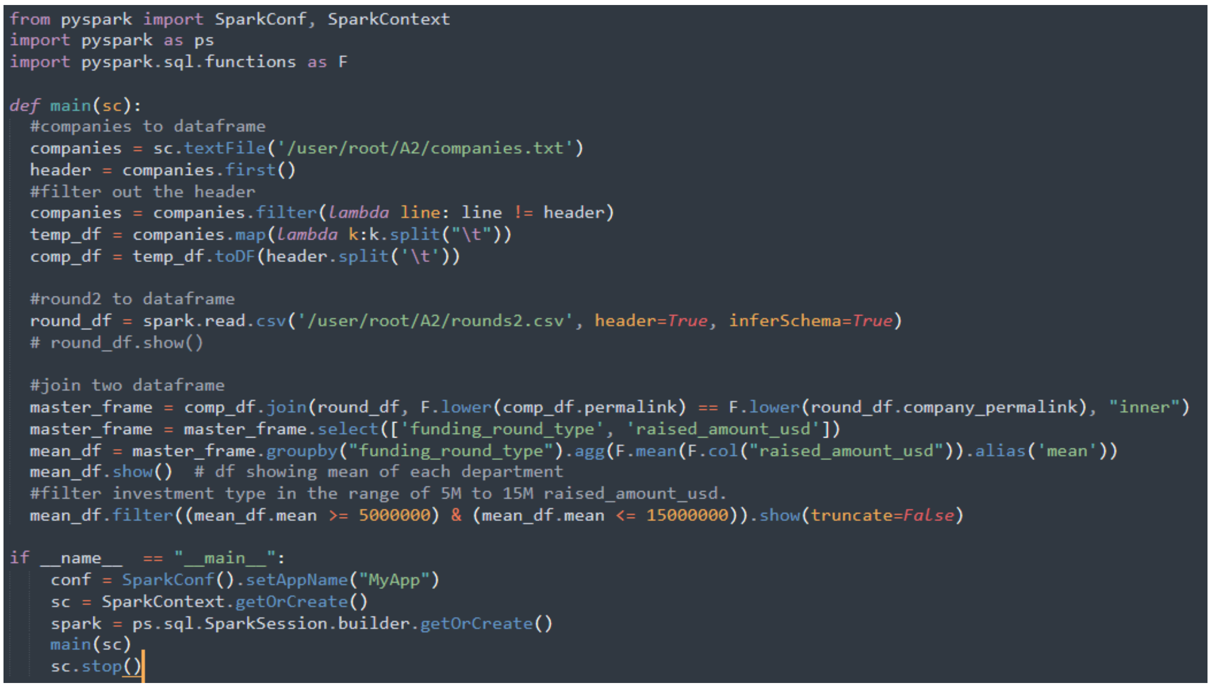


```
>>> round_rdd = sc.textFile('/user/root/A2/rounds2.csv').map(lambda line: line.split(","))
>>> header = round_rdd.first()
>>> round_rdd = round_rdd.filter(lambda line: line != header)
>>> round_df = round_rdd.toDF(header)
>>> ven_frame = round_df[round_df['funding_round_type'] == 'venture']
>>> ven_frame.describe('raised_amount_usd').show()
+-------+--------------------+
|summary|   raised_amount_usd|
+-------+--------------------+
|  count|               55494|
|   mean|1.1748949129489528E7|
| stddev| 8.635206655796282E7|
|    min|                    |
|    max|             9999999|
+-------+--------------------+

>>> angel_frame = round_df[round_df['funding_round_type'] == 'angel']
>>> angel_frame.describe('raised_amount_usd').show()
+-------+-----------------+
|summary|raised_amount_usd|
+-------+-----------------+
|  count|             6094|
|   mean|958694.4697530865|
| stddev| 7404397.12212948|
|    min|                 |
|    max|            99995|
+-------+-----------------+

>>> seed_frame = round_df[round_df['funding_round_type'] == 'seed']
>>> seed_frame.describe('raised_amount_usd').show()
+-------+-----------------+
|summary|raised_amount_usd|
+-------+-----------------+
|  count|            30524|
|   mean|719817.9969071728|
| stddev|2221732.800078571|
|    min|                 |
|    max|           999999|
+-------+-----------------+

>>> pri_frame = round_df[round_df['funding_round_type'] == 'private_equity']
>>> pri_frame.describe('raised_amount_usd').show()
+-------+--------------------+
|summary|   raised_amount_usd|
+-------+--------------------+
|  count|                2285|
|   mean| 7.330859302944215E7|
| stddev|1.9811345841372624E8|
|    min|                    |
|    max|             9999990|
```

Q4. Considering that X investor wants to invest between 5 to 15 million USD per investment round, which investment type is the most suitable for them? Hint: You can check the type of investment for this range of investment (20 pts)

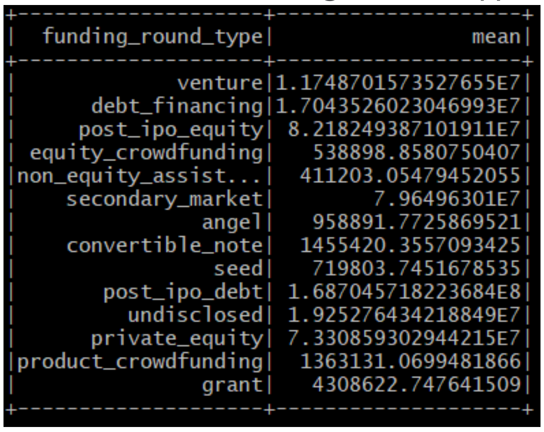


```
[root@sandbox-hdp A2]# spark-submit --master yarn --deploy-mode client --executo
r-memory 512m --num-executors 3 --executor-cores 1 --driver-memory 512m 4.py
```



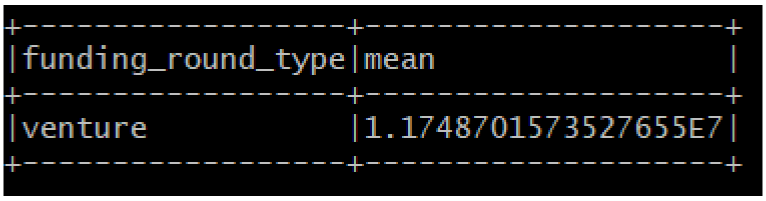
```python
from pyspark import SparkConf, SparkContext
import pyspark as ps
import pyspark.sql.functions as F

def main(sc):
  #companies to dataframe
  companies = sc.textFile('/user/root/A2/companies.txt')
  header = companies.first()
  #filter out the header
  companies = companies.filter(lambda line: line != header)
  temp_df = companies.map(lambda k:k.split("\t"))
  comp_df = temp_df.toDF(header.split('\t'))

  #round2 to dataframe
  round_df = spark.read.csv('/user/root/A2/rounds2.csv', header=True, inferSchema=True)
  # round_df.show()

  #join two dataframe
  master_frame = comp_df.join(round_df, F.lower(comp_df.permalink) == F.lower(round_df.company_permalink), "inner")
  master_frame = master_frame.select(['funding_round_type', 'raised_amount_usd'])
  mean_df = master_frame.groupby("funding_round_type").agg(F.mean(F.col("raised_amount_usd")).alias('mean'))
  mean_df.show()  # df showing mean of each department
  #filter investment type in the range of 5M to 15M raised_amount_usd.
  mean_df.filter((mean_df.mean >= 5000000) & (mean_df.mean <= 15000000)).show(truncate=False)

if __name__ == "__main__":
    conf = SparkConf().setAppName("MyApp")
    sc = SparkContext.getOrCreate()
    spark = ps.sql.SparkSession.builder.getOrCreate()
    main(sc)
    sc.stop()
```

mean of each funding_round_type



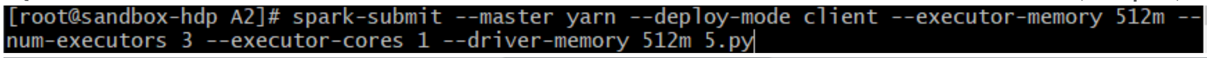
```
+--------------------+--------------------+
|  funding_round_type|                mean|
+--------------------+--------------------+
|             venture|1.1748701573527655E7|
|      debt_financing|1.7043526023046993E7|
|     post_ipo_equity| 8.218249387101911E7|
| equity_crowdfunding|   538898.8580750407|
|non_equity_assist...|  411203.05479452055|
|    secondary_market|        7.96496301E7|
|               angel|   958891.7725869521|
|    convertible_note|  1455420.3557093425|
|                seed|   719803.7451678535|
|       post_ipo_debt| 1.687045718223684E8|
|         undisclosed| 1.925276434218849E7|
|     private_equity| 7.330859302944215E7|
|product_crowdfunding|  1363131.0699481866|
|               grant|   4308622.747641509|
+--------------------+--------------------+
```

After filtered: Funding type in the range of 5M to 15M
Venture is the most suitable investment type

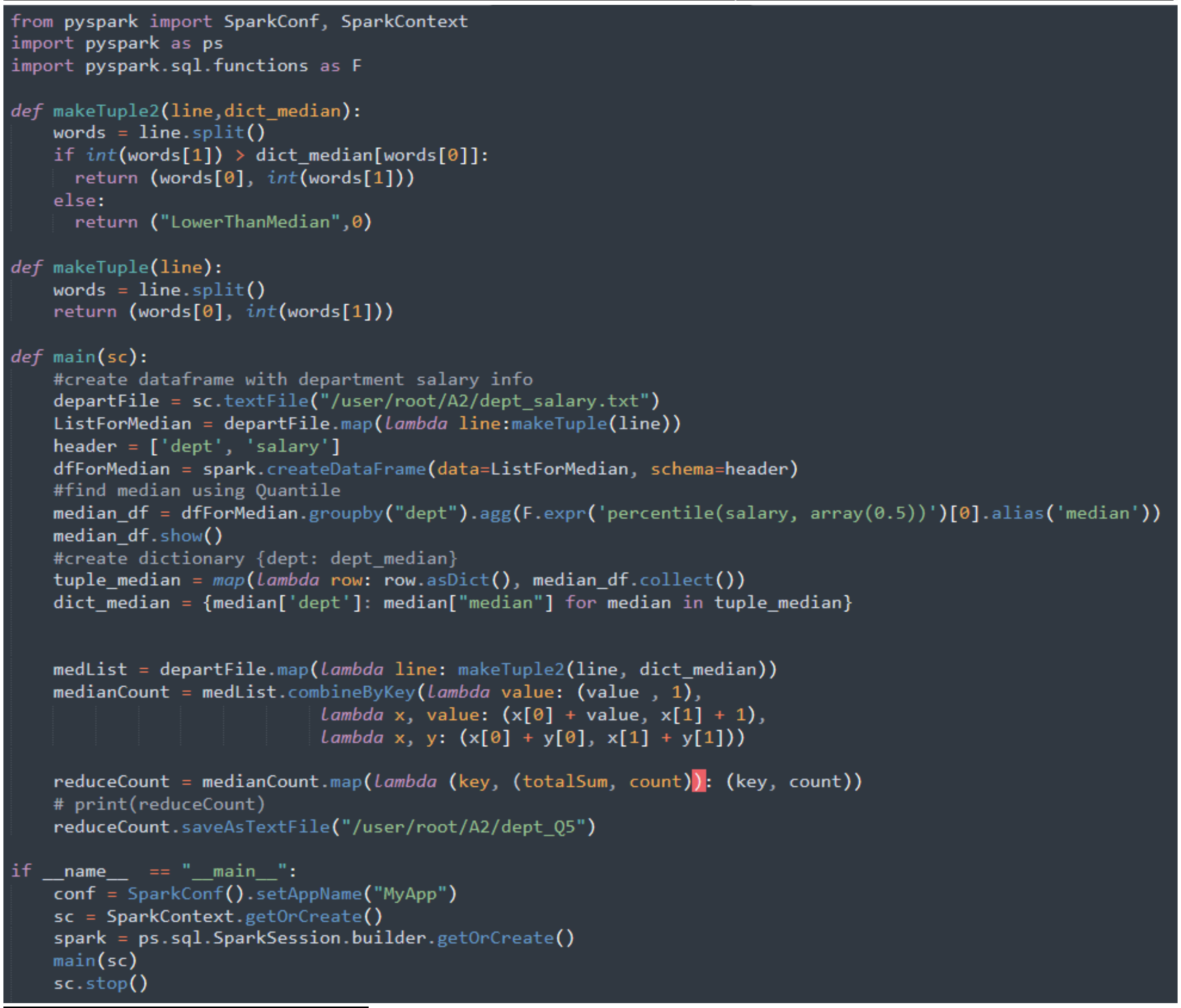


```
+------------------+--------------------+
|funding_round_type|mean                |
+------------------+--------------------+
|venture           |1.1748701573527655E7|
+------------------+--------------------+
```

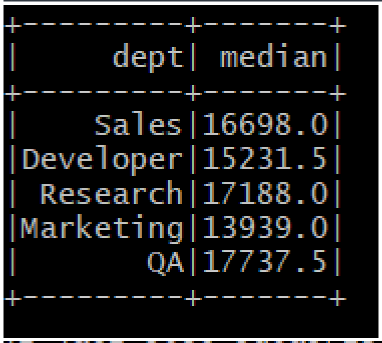
Q5. NUMBER OF SALARIES HIGHER THAN MEDIAN PER DEPARTMENT (10 pts)



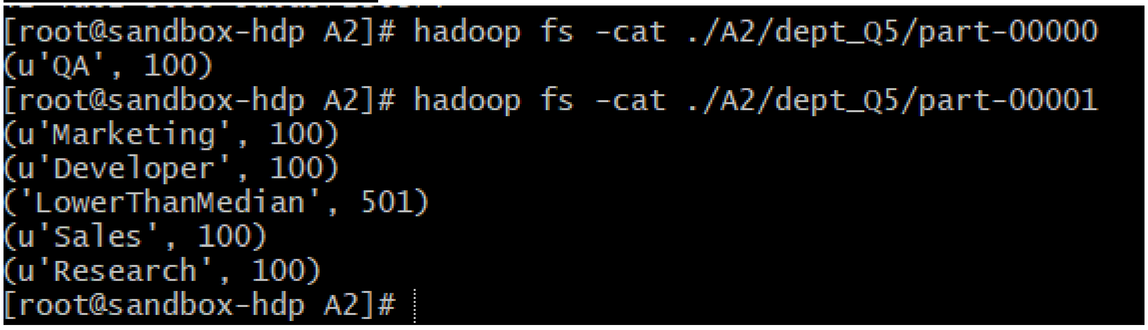
```
[root@sandbox-hdp A2]# spark-submit --master yarn --deploy-mode client --executor-memory 512m --
num-executors 3 --executor-cores 1 --driver-memory 512m 5.py
```



```python
from pyspark import SparkConf, SparkContext
import pyspark as ps
import pyspark.sql.functions as F

def makeTuple2(line,dict_median):
    words = line.split()
    if int(words[1]) > dict_median[words[0]]:
        return (words[0], int(words[1]))
    else:
        return ("LowerThanMedian",0)

def makeTuple(line):
    words = line.split()
    return (words[0], int(words[1]))

def main(sc):
    #create dataframe with department salary info
    departFile = sc.textFile("/user/root/A2/dept_salary.txt")
    ListForMedian = departFile.map(lambda line:makeTuple(line))
    header = ['dept', 'salary']
    dfForMedian = spark.createDataFrame(data=ListForMedian, schema=header)
    #find median using Quantile
    median_df = dfForMedian.groupby("dept").agg(F.expr('percentile(salary, array(0.5))')[0].alias('median'))
    median_df.show()
    #create dictionary {dept: dept_median}
    tuple_median = map(lambda row: row.asDict(), median_df.collect())
    dict_median = {median['dept']: median["median"] for median in tuple_median}


    medList = departFile.map(lambda line: makeTuple2(line, dict_median))
    medianCount = medList.combineByKey(lambda value: (value , 1),
                                       lambda x, value: (x[0] + value, x[1] + 1),
                                       lambda x, y: (x[0] + y[0], x[1] + y[1]))

    reduceCount = medianCount.map(lambda (key, (totalSum, count)): (key, count))
    # print(reduceCount)
    reduceCount.saveAsTextFile("/user/root/A2/dept_Q5")

if __name__  == "__main__":
    conf = SparkConf().setAppName("MyApp")
    sc = SparkContext.getOrCreate()
    spark = ps.sql.SparkSession.builder.getOrCreate()
    main(sc)
    sc.stop()
```



```
+---------+-------+
|     dept| median|
+---------+-------+
|    Sales|16698.0|
|Developer|15231.5|
| Research|17188.0|
|Marketing|13939.0|
|       QA|17737.5|
+---------+-------+
```

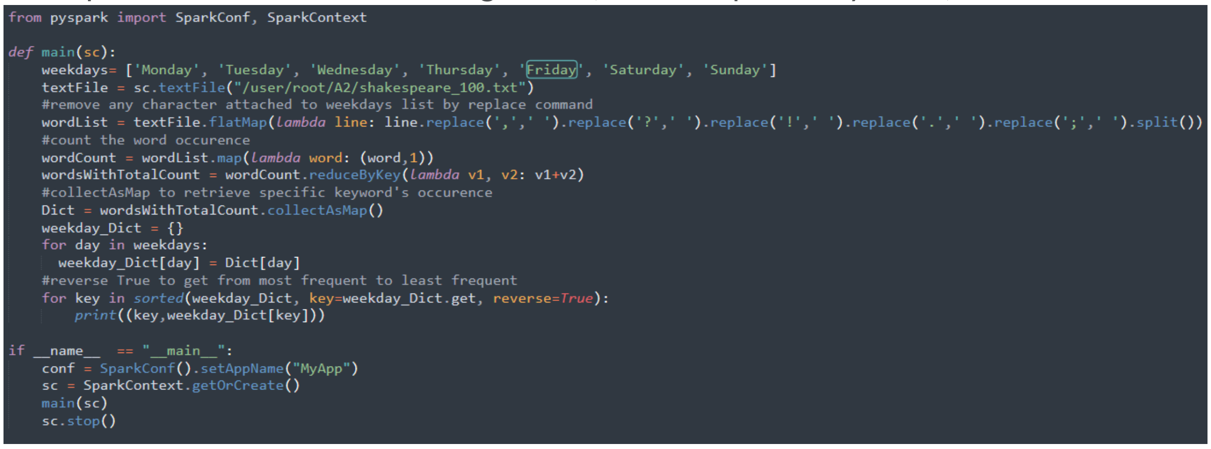


```
[root@sandbox-hdp A2]# hadoop fs -cat ./A2/dept_Q5/part-00000
(u'QA', 100)
[root@sandbox-hdp A2]# hadoop fs -cat ./A2/dept_Q5/part-00001
(u'Marketing', 100)
(u'Developer', 100)
('LowerThanMedian', 501)
(u'Sales', 100)
(u'Research', 100)
[root@sandbox-hdp A2]#
```

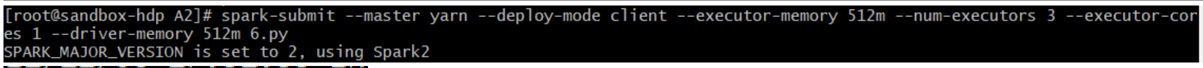
Result:
Sales department has **100** employees with salaries higher than median of $16698.
QA department has **100** employees with salaries higher than median of $17737.5.
Marketing department has **100** employees with salaries higher than median of $13939.0.
Developer department has **100** employees with salaries higher than median of 15231.5.
Research department has **100** employees with salaries higher than median of 17188.0.
501 employees with salaries **lower** than their department median.

Q6. Week days count (10 pts)

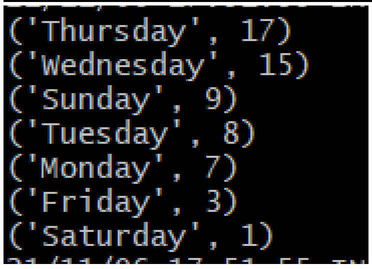
Input: shakespeare_100.txt
Output: list of week days ('Monday' to 'Sunday') and their number of occurrences in the input file shown in descending order (more frequent days first)



```python
from pyspark import SparkConf, SparkContext

def main(sc):
    weekdays= ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
    textFile = sc.textFile("/user/root/A2/shakespeare_100.txt")
    #remove any character attached to weekdays list by replace command
    wordList = textFile.flatMap(lambda line: line.replace(',',' ').replace('?',' ').replace('!',' ').replace('.',' ').replace(';',' ').split())
    #count the word occurence
    wordCount = wordList.map(lambda word: (word,1))
    wordsWithTotalCount = wordCount.reduceByKey(lambda v1, v2: v1+v2)
    #collectAsMap to retrieve specific keyword's occurence
    Dict = wordsWithTotalCount.collectAsMap()
    weekday_Dict = {}
    for day in weekdays:
     weekday_Dict[day] = Dict[day]
    #reverse True to get from most frequent to least frequent
    for key in sorted(weekday_Dict, key=weekday_Dict.get, reverse=True):
        print((key,weekday_Dict[key]))

if __name__  == "__main__":
    conf = SparkConf().setAppName("MyApp")
    sc = SparkContext.getOrCreate()
    main(sc)
    sc.stop()
```

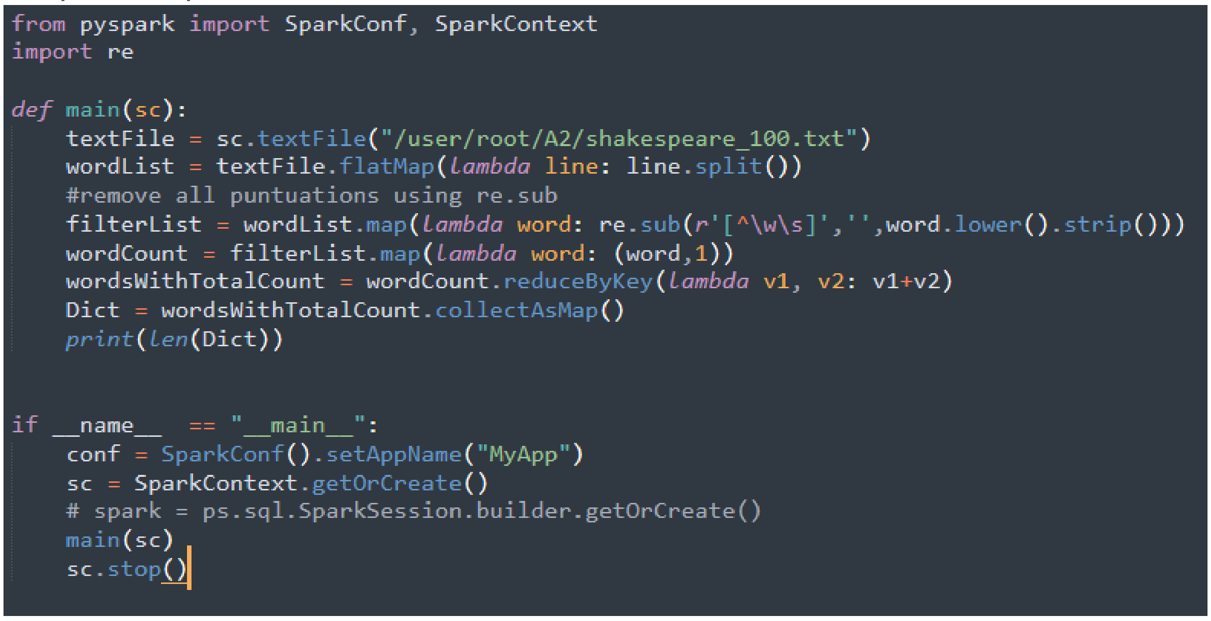


```
[root@sandbox-hdp A2]# spark-submit --master yarn --deploy-mode client --executor-memory 512m --num-executors 3 --executor-cor
es 1 --driver-memory 512m 6.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
```

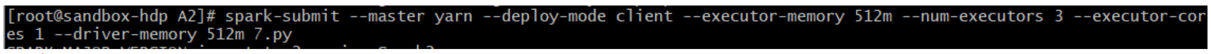


```
('Thursday', 17)
('Wednesday', 15)
('Sunday', 9)
('Tuesday', 8)
('Monday', 7)
('Friday', 3)
('Saturday', 1)
```

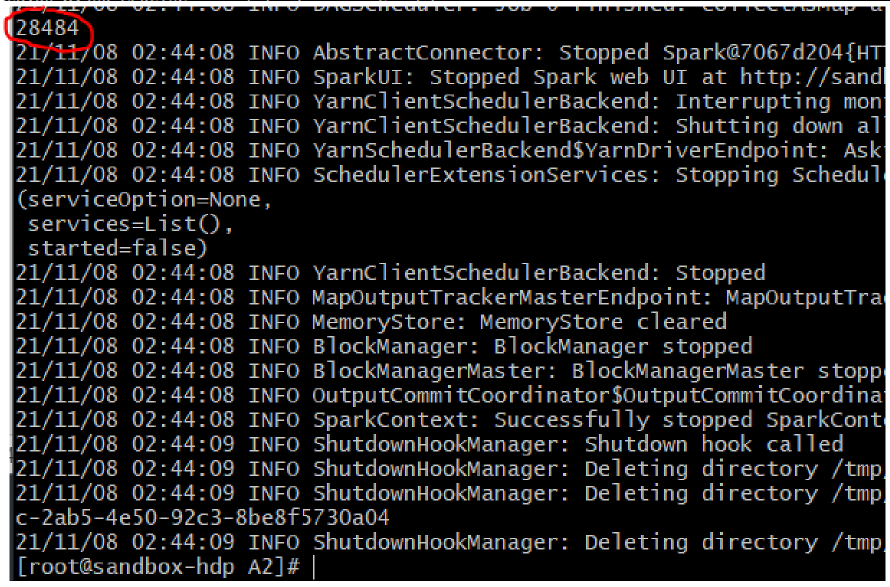
Q7. Calculate the number of unique words in input file (10 pts)
Input: shakespeare_100.txt
Output: Unique words



```python
from pyspark import SparkConf, SparkContext
import re

def main(sc):
    textFile = sc.textFile("/user/root/A2/shakespeare_100.txt")
    wordList = textFile.flatMap(lambda line: line.split())
    #remove all puntuations using re.sub
    filterList = wordList.map(lambda word: re.sub(r'[^\w\s]','',word.lower().strip()))
    wordCount = filterList.map(lambda word: (word,1))
    wordsWithTotalCount = wordCount.reduceByKey(lambda v1, v2: v1+v2)
    Dict = wordsWithTotalCount.collectAsMap()
    print(len(Dict))


if __name__ == "__main__":
    conf = SparkConf().setAppName("MyApp")
    sc = SparkContext.getOrCreate()
    # spark = ps.sql.SparkSession.builder.getOrCreate()
    main(sc)
    sc.stop()
```

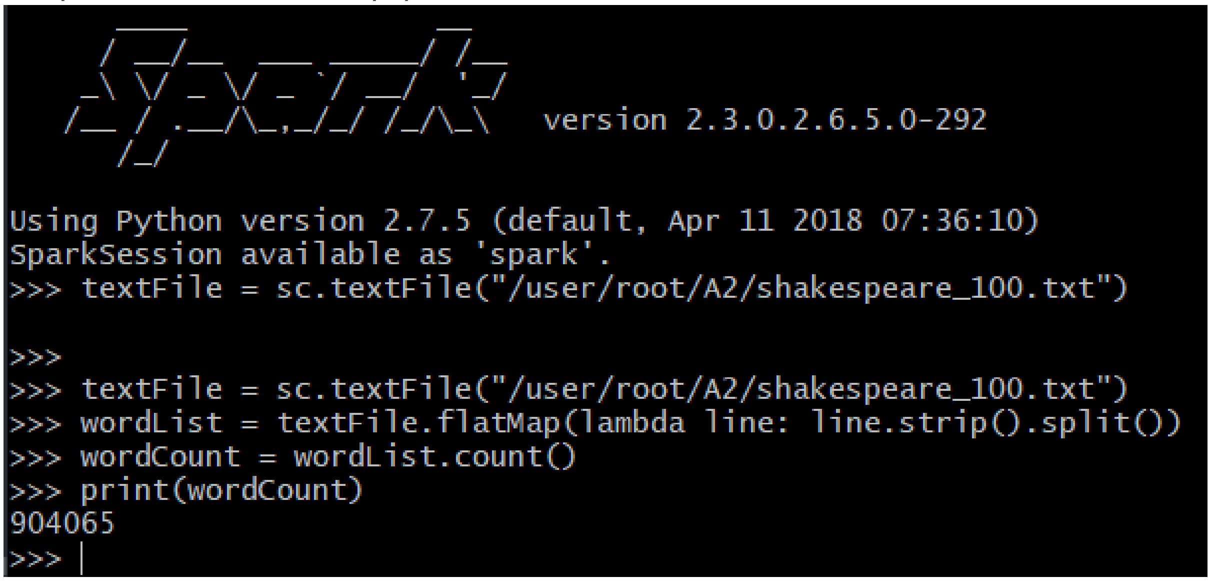


```
[root@sandbox-hdp A2]# spark-submit --master yarn --deploy-mode client --executor-memory 512m --num-executors 3 --executor-cor
es 1 --driver-memory 512m 7.py
```



```
28484
21/11/08 02:44:08 INFO AbstractConnector: Stopped Spark@7067d204{HT
21/11/08 02:44:08 INFO SparkUI: Stopped Spark web UI at http://sand
21/11/08 02:44:08 INFO YarnClientSchedulerBackend: Interrupting mon
21/11/08 02:44:08 INFO YarnClientSchedulerBackend: Shutting down al
21/11/08 02:44:08 INFO YarnSchedulerBackend$YarnDriverEndpoint: Ask
21/11/08 02:44:08 INFO SchedulerExtensionServices: Stopping Schedul
(serviceOption=None,
 services=List(),
 started=false)
21/11/08 02:44:08 INFO YarnClientSchedulerBackend: Stopped
21/11/08 02:44:08 INFO MapOutputTrackerMasterEndpoint: MapOutputTra
21/11/08 02:44:08 INFO MemoryStore: MemoryStore cleared
21/11/08 02:44:08 INFO BlockManager: BlockManager stopped
21/11/08 02:44:08 INFO BlockManagerMaster: BlockManagerMaster stopp
21/11/08 02:44:08 INFO OutputCommitCoordinator$OutputCommitCoordina
21/11/08 02:44:08 INFO SparkContext: Successfully stopped SparkCont
21/11/08 02:44:09 INFO ShutdownHookManager: Shutdown hook called
21/11/08 02:44:09 INFO ShutdownHookManager: Deleting directory /tmp
21/11/08 02:44:09 INFO ShutdownHookManager: Deleting directory /tmp
c-2ab5-4e50-92c3-8be8f5730a04
21/11/08 02:44:09 INFO ShutdownHookManager: Deleting directory /tmp
[root@sandbox-hdp A2]#
```

Q8. Remove all empty elements/entries where the word is '' (10 pts)
Input: shakespeare_100.txt
Output: Count of non-empty words



```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.0.2.6.5.0-292
      /_/

Using Python version 2.7.5 (default, Apr 11 2018 07:36:10)
SparkSession available as 'spark'.
>>> textFile = sc.textFile("/user/root/A2/shakespeare_100.txt")

>>>
>>> textFile = sc.textFile("/user/root/A2/shakespeare_100.txt")
>>> wordList = textFile.flatMap(lambda line: line.strip().split())
>>> wordCount = wordList.count()
>>> print(wordCount)
904065
>>> |
```