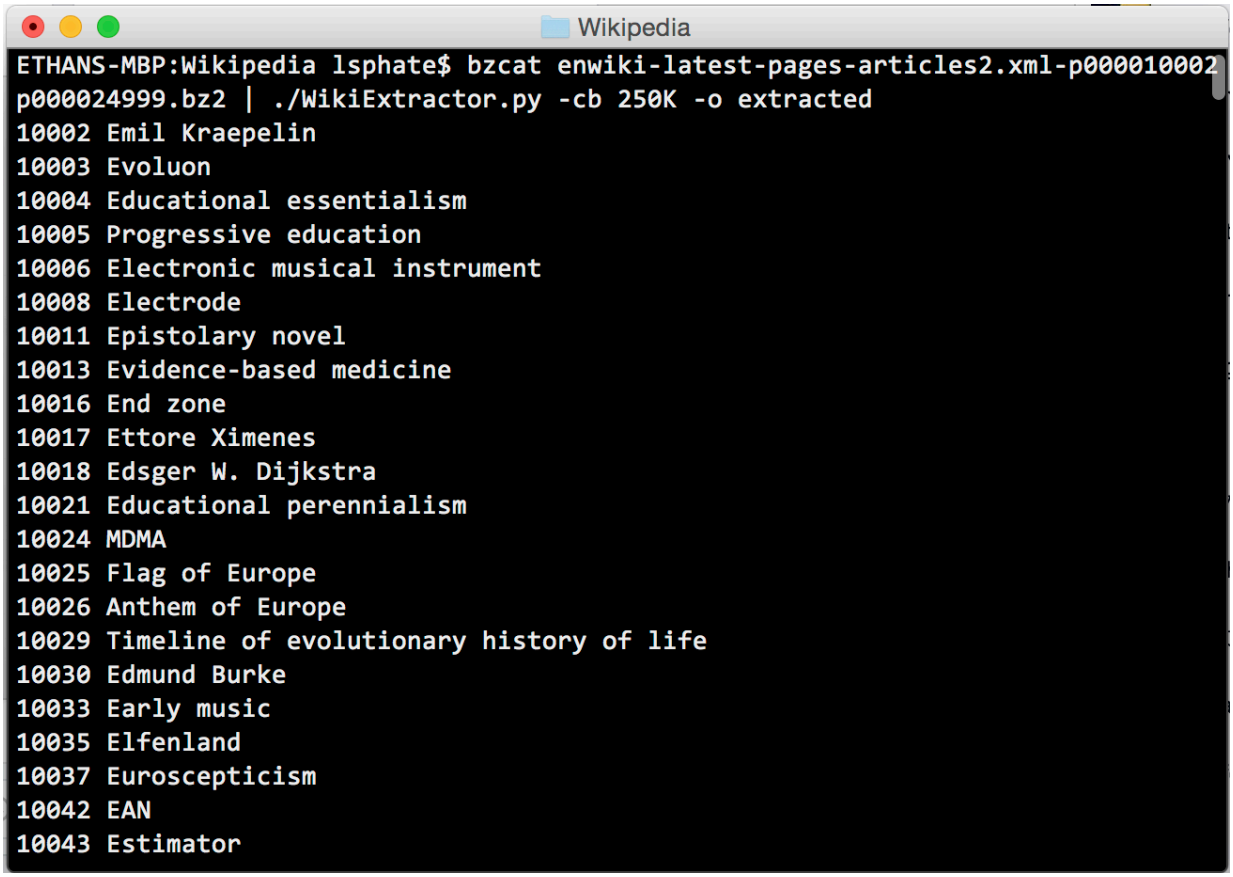
EECS6895 Adv. Bigdata Analytics HW 1 - Sun-Yi Lin(sl3833)

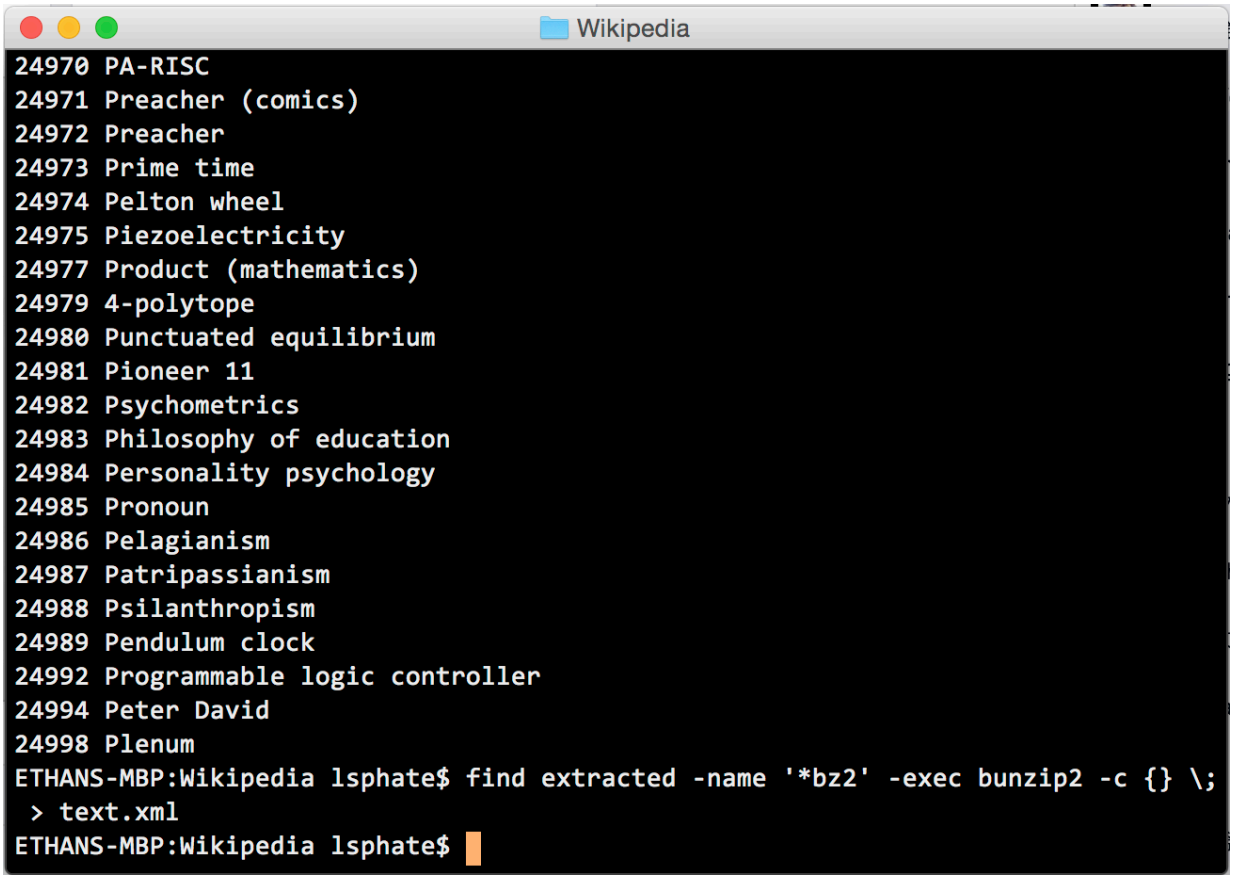
Prob. 2 Wikipedia Articles

Step 1. Download & extract Wikipedia articles

I downloaded the latest articles from http://dumps.wikimedia.org/enwiki/latest/. I choose the smallest file since we only need 1,000 pages. Then I use Wikipedia Extractor to extract the text of each articles.



Then we need to combine all the files into one text files by keeping the moderate numbers of outputs and use the command below:



The output format of **Wikipedia Extractor** is like below:

```
<doc id="10002" url="http://en.wikipedia.org/wiki?curid=10002" title="Emil Kraepelin">
Emil Kraepelin

Emil Kraepelin (15 February 1856 – 7 October 1926) was a German psychiatrist. H.J. Eysenck's
"Encyclopedia of Psychology" identifies him as the founder of modern scientific psychiatry, as well as of
psychopharmacology and psychiatric genetics. Kraepelin believed the chief origin of psychiatric disease
to be biological and genetic malfunction. His theories dominated psychiatry at the start of the twentieth
century and, despite the later psychodynamic influence of Sigmund Freud and his disciples, enjoyed a
revival at century's end.

Family and early life.
Kraepelin, the son of a civil servant, was born in 1856 in Neustrelitz, in the Duchy of Mecklenburg-
Strelitz in Germany. He was first introduced to biology by his brother Karl, 10 years older and, later,
the director of the Zoological Museum of Hamburg.

...

External links.
For biographies of Kraepelin see:
For English translations of Kraepelin's work see:

</doc>
```

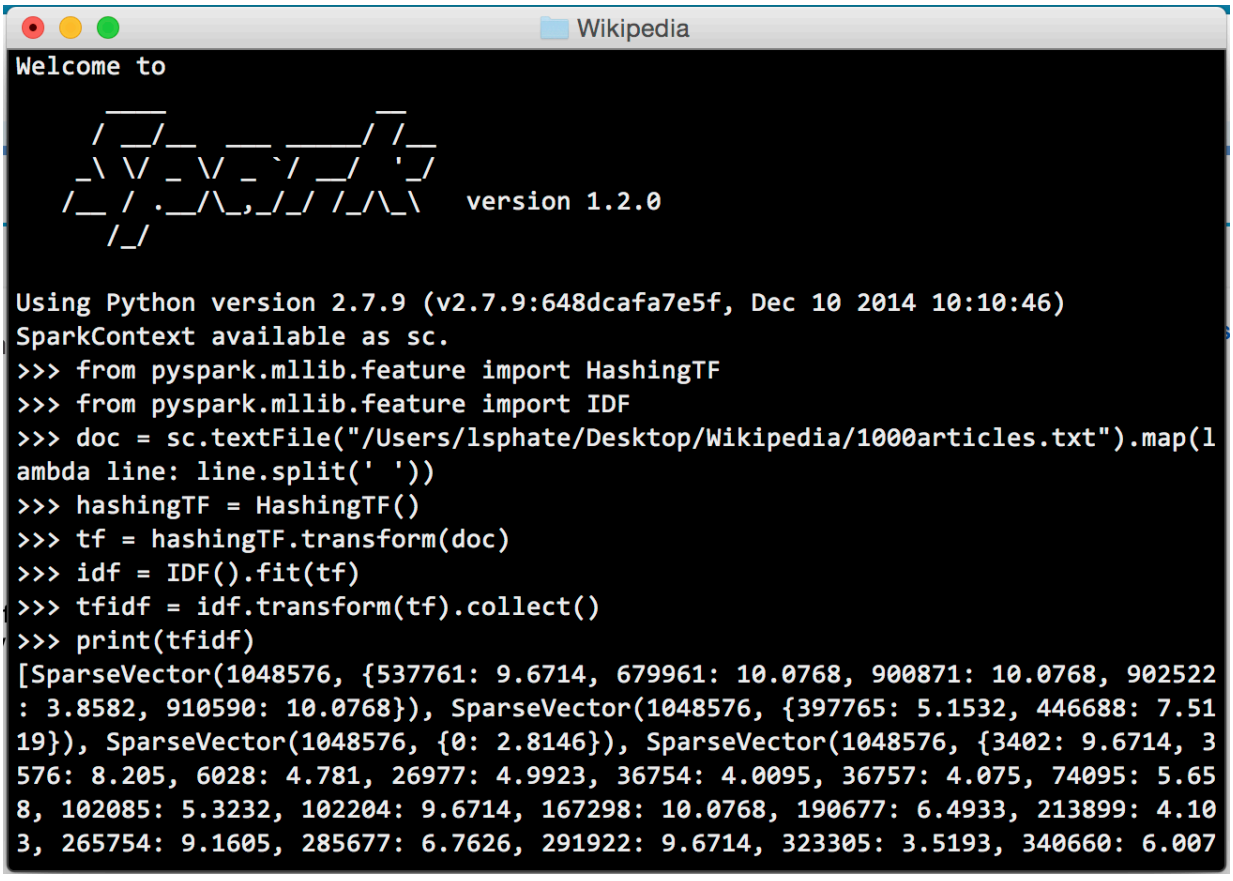
Each element contains the ID, title and text of one page, all the external links and other tags will be eliminated.

Step 2. Do TF-IDF

After we got we use **pyspark** to create TD-IDF of these articles.

```
from pyspark.mllib.feature import HashingTF
from pyspark.mllib.feature import IDF

doc = sc.textFile("PATH_OF_INPUT").map(lambda line: line.split(' '))
hashingTF = HashingTF()
tf = hashingTF.transform(doc)
idf = IDF().fit(tf)
tfidf = idf.transform(tf).collect()

print(tfidf)
```

After several minutes of computing, we'll get the result:

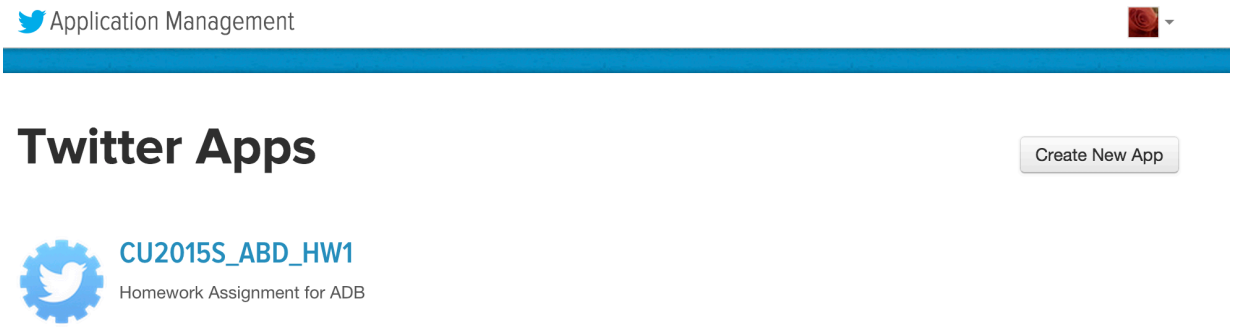


Prob. 3 Tweets Regarding 5 Companies

Step 1. Pull the real-time tweets data

As the request of this problem, we need to choose 5 companies to collect the real-time tweets mentions them. In order to be

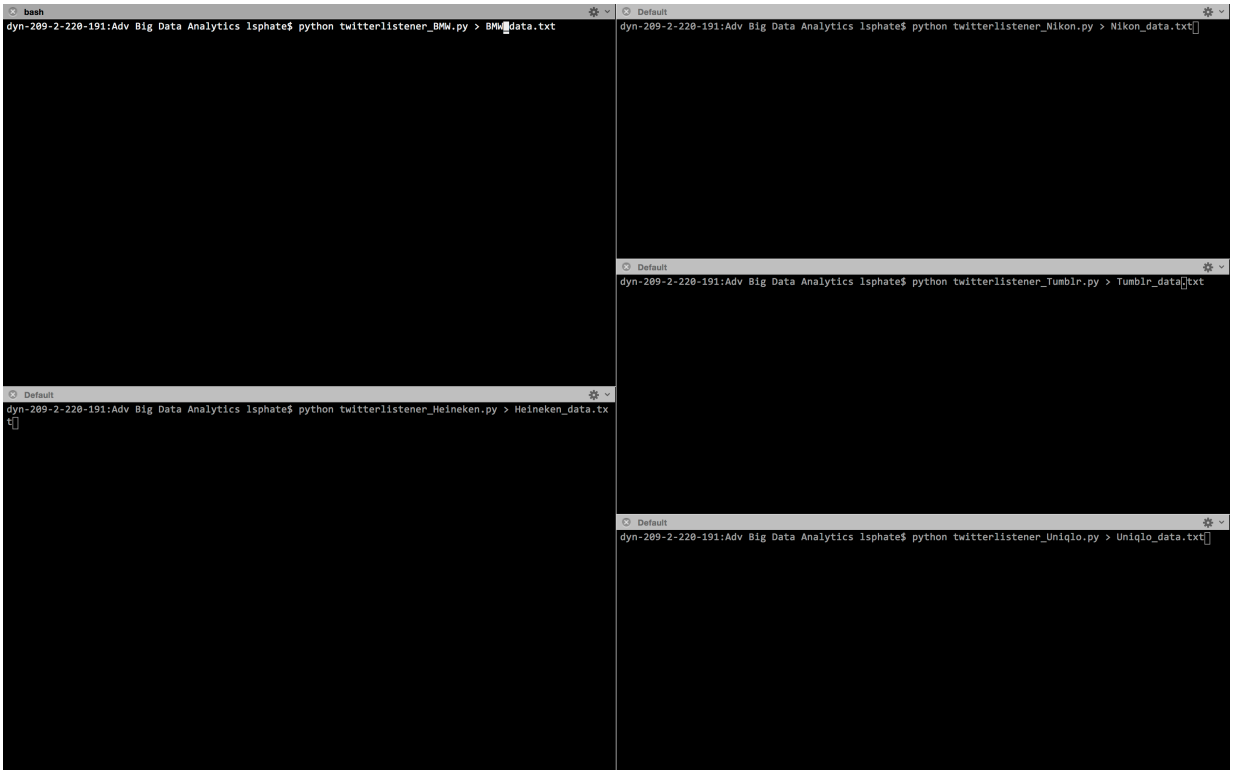
allowed to collect the data from Twitter, we need to create a twitter application and obtain the access token to use in the **Twitter Streaming API**.



And also we need the third-party library called Tweepy to pull down the tweets. I followed the instruction form here, and wrote the Python application to get the tweets.

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream

access_token = "YOUR_ACCESS_TOKEN"
access_token_secret = "YOUR_ACCESS_SECRET"
consumer_key = "YOUR_KEY"
consumer_secret = "YOUR_SECRET"

class StdOutListener(StreamListener):
    def on_data(self, data):
        print data
        return True
    def on_error(self, status):
        print status

if __name__ == '__main__':
    l = StdOutListener()
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    stream = Stream(auth, l)
    stream.filter(track=['KEYWORD'])
```

Where I replace the **KEYWORD** by the names of the 5 companies' name I choose: **BMW**, **Burberry**, **Heineken**, **Nikon** and **Tumblr**, then start these applications to grab the data.



The tweets pulled down are in these format:

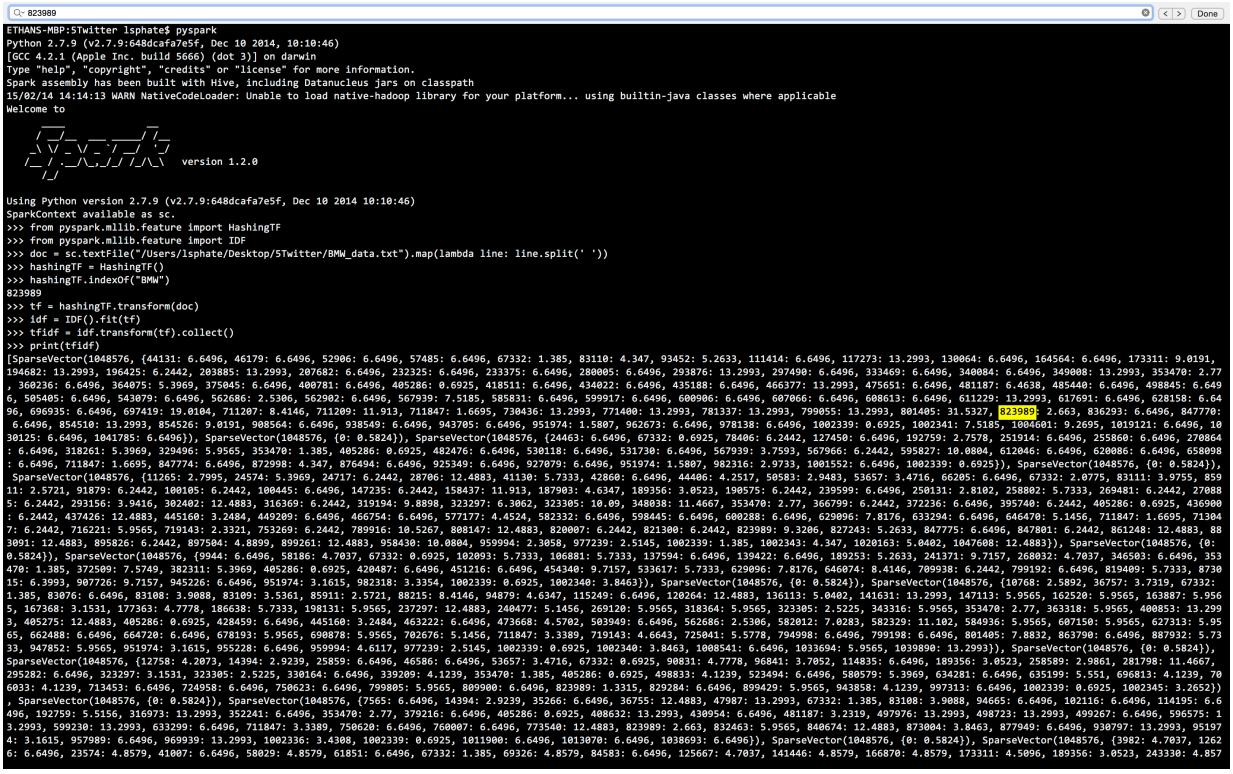
```
{"created_at":"Thu Feb 12 21:23:02 +0000
2015","id":565984447057895424,"id_str":"565984447057895424","text":"RT @PdeTannhauser: Hoy hace 8
a\u00f1os, en St Moritz, @NickHeidfeld se lo pasaba pipa con su BMW Sauber F1.07 CC @VirutasF1
http:\/\/t.co\/CDLQu1e\u2026",

...

"lang":"es","timestamp_ms":"1423776182318"}
```

Step 2. Do TF-IDF

Similar to the former problem, I use **pyspark** to create TF-IDF for each file.

```
from pyspark.mllib.feature import HashingTF
from pyspark.mllib.feature import IDF

doc = sc.textFile("INPUT TWITTER DATA").map(lambda line: line.split(' '))
hashingTF = HashingTF()
hashingTF.indexOf("COMPANY NAME")
tf = hashingTF.transform(doc)
idf = IDF().fit(tf)
tfidf = idf.transform(tf).collect()

print(tfidf)
```

And following are the results of each company:



```
823989
Done
ETHANS-MBP:5Twitter lshpate$ pyspark
Python 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014, 10:10:46)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
15/02/14 14:14:13 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/

Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)
SparkContext available as sc.
>>> from pyspark.mllib.feature import HashingTF
>>> from pyspark.mllib.feature import IDF
>>> doc = sc.textFile("/Users/lshpate/Desktop/5Twitter/BMW_data.txt").map(lambda line: line.split(' '))
>>> hashingTF = HashingTF()
>>> hashingTF.indexOf("BMW")
823989
>>> tf = hashingTF.transform(doc)
>>> idf = IDF().fit(tf)
>>> tfidf = idf.transform(tf).collect()
>>> print(tfidf)
[SparseVector(1048576, {44131: 6.6496, 46179: 6.6496, 52906: 6.6496, 57485: 6.6496, 67332: 1.385, 83110: 4.347, 93452: 5.2633, 111414: 6.6496, 117273: 13.2993, 130064: 6.6496, 164564: 6.6496, 173311: 9.0191, 194682: 13.2993, 196425: 6.2442, 203885: 13.2993, 207682: 6.6496, 232325: 6.6496, 233375: 6.6496, 280005: 6.6496, 293876: 13.2993, 297490: 6.6496, 333469: 6.6496, 340084: 6.6496, 349008: 13.2993, 353470: 2.77, 360236: 6.6496, 364075: 5.3969, 375045: 6.6496, 400781: 6.6496, 405286: 0.6925, 418511: 6.6496, 434022: 6.6496, 435188: 6.6496, 466377: 13.2993, 475651: 6.6496, 481187: 6.4638, 485440: 6.6496, 498845: 6.6496, 505405: 6.6496, 543079: 6.6496, 562686: 2.5306, 562902: 6.6496, 567939: 7.5185, 585831: 6.6496, 599917: 6.6496, 600906: 6.6496, 607066: 6.6496, 608613: 6.6496, 611229: 13.2993, 617691: 6.6496, 628158: 6.6496, 696935: 6.6496, 697419: 19.0104, 711207: 8.4146, 711209: 11.913, 711847: 1.6695, 730436: 13.2993, 771400: 13.2993, 781337: 13.2993, 799055: 13.2993, 801405: 31.5327, 823989: 2.663, 836293: 6.6496, 847770: 6.6496, 854510: 13.2993, 854526: 9.0191, 908564: 6.6496, 938549: 6.6496, 943705: 6.6496, 951974: 1.5807, 962673: 6.6496, 978138: 6.6496, 1002339: 0.6925, 1002341: 7.5185, 1004601: 9.2695, 1019121: 6.6496, 1030125: 6.6496, 1041785: 6.6496}), SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {24463: 6.6496, 67332: 0.6925, 78406: 6.2442, 127450: 6.6496, 192759: 2.7578, 251914: 6.6496, 255860: 6.6496, 270864: 6.6496, 318261: 5.3969, 329496: 5.9565, 353470: 1.385, 405286: 0.6925, 482476: 6.6496, 530118: 6.6496, 531730: 6.6496, 567939: 3.7593, 567966: 6.2442, 595827: 10.0804, 612046: 6.6496, 620086: 6.6496, 658098: 6.6496, 711847: 1.6695, 847774: 6.6496, 872998: 4.347, 876494: 6.6496, 925349: 6.6496, 927079: 6.6496, 951974: 1.5807, 982316: 2.9733, 1001552: 6.6496, 1002339: 0.6925}), SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {11265: 2.7995, 24574: 5.3969, 24717: 6.2442, 28706: 12.4883, 41130: 5.7333, 42860: 6.6496, 44406: 4.2517, 50583: 2.9483, 53657: 3.4716, 66205: 6.6496, 67332: 2.0775, 83111: 3.9755, 85911: 2.5721, 91879: 6.2442, 100105: 6.2442, 100445: 6.6496, 147235: 6.2442, 158437: 11.913, 187903: 4.6347, 189356: 3.0523, 190575: 6.2442, 239599: 6.6496, 250131: 2.8102, 258802: 5.7333, 269481: 6.2442, 270885: 6.2442, 293156: 3.9416, 302402: 12.4883, 316369: 6.2442, 319194: 9.8898, 323297: 6.3062, 323305: 10.09, 348038: 11.4667, 353470: 2.77, 366799: 6.2442, 372236: 6.6496, 395740: 6.2442, 405286: 0.6925, 436900: 6.2442, 437426: 12.4883, 445160: 3.2484, 449289: 6.6496, 466754: 6.6496, 577177: 4.4524, 582332: 6.6496, 598445: 6.6496, 600288: 6.6496, 629096: 7.8176, 633294: 6.6496, 646470: 5.1456, 711847: 1.6695, 713047: 6.2442, 716221: 5.9565, 719143: 2.3321, 753269: 6.2442, 789916: 10.5267, 808147: 12.4883, 820007: 6.2442, 821300: 6.2442, 823989: 9.3206, 827243: 5.2633, 847775: 6.6496, 847801: 6.2442, 861240: 12.4883, 883091: 12.4883, 895826: 6.2442, 897504: 4.8899, 899261: 12.4883, 958430: 10.0804, 959994: 2.3058, 977239: 2.5145, 1002339: 1.385, 1002343: 4.347, 1020163: 5.8402, 1047608: 12.4883}), SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {9944: 6.6496, 58186: 4.7037, 67332: 0.6925, 102093: 5.7333, 106881: 5.7333, 137594: 6.6496, 139422: 6.6496, 189253: 5.2633, 241371: 9.7157, 268032: 4.7037, 346503: 6.6496, 353470: 1.385, 372509: 7.5749, 382311: 5.3969, 405286: 0.6925, 420487: 6.6496, 451216: 6.6496, 454340: 9.7157, 533617: 5.7333, 629096: 7.8176, 646074: 8.4146, 709938: 6.2442, 799192: 6.6496, 819409: 5.7333, 873015: 6.3993, 907726: 9.7157, 945226: 6.6496, 951974: 3.1615, 982318: 3.3354, 1002339: 0.6925, 1002340: 3.8463}), SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {10768: 2.5892, 36757: 3.7319, 67332: 1.385, 83076: 6.6496, 83108: 3.9088, 83109: 3.5361, 85911: 2.5721, 88215: 8.4146, 94879: 4.6347, 115249: 6.6496, 120264: 12.4883, 136113: 5.0402, 141631: 13.2993, 147113: 5.9565, 162520: 5.9565, 163887: 5.9565, 167368: 3.1531, 177363: 4.7778, 186638: 5.7333, 198131: 5.9565, 237297: 12.4883, 240477: 5.1456, 269120: 5.9565, 318364: 5.9565, 323305: 2.5225, 343316: 5.9565, 353470: 2.77, 363318: 5.9565, 400853: 13.2993, 405275: 12.4883, 405286: 0.6925, 428459: 6.6496, 445160: 3.2484, 463222: 6.6496, 473668: 4.5702, 503949: 6.6496, 562686: 2.5306, 582012: 7.0283, 582329: 11.102, 584936: 5.9565, 607150: 5.9565, 627313: 5.9565, 662488: 6.6496, 664720: 6.6496, 678193: 5.9565, 690878: 5.9565, 702676: 5.1456, 711847: 3.3389, 719143: 4.6643, 725041: 5.5778, 794998: 6.6496, 799198: 6.6496, 801405: 7.8832, 863790: 6.6496, 887932: 5.7333, 947852: 5.9565, 951974: 3.1615, 955228: 6.6496, 959994: 4.6117, 977239: 2.5145, 1002339: 0.6925, 1002340: 3.8463, 1008541: 6.6496, 1033694: 5.9565, 1039890: 13.2993}), SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {12758: 4.2073, 14394: 2.9239, 25859: 6.6496, 46586: 6.6496, 53657: 3.4716, 67332: 0.6925, 90831: 4.7778, 96841: 3.7052, 114835: 6.6496, 189356: 3.0523, 258589: 2.9861, 281798: 11.4667, 295282: 6.6496, 323297: 3.1531, 323305: 2.5225, 330164: 6.6496, 339209: 4.1239, 353470: 1.385, 405286: 0.6925, 498833: 4.1239, 523494: 6.6496, 580579: 5.3969, 634281: 6.6496, 635199: 5.551, 696813: 4.1239, 706033: 4.1239, 713453: 6.6496, 724958: 6.6496, 750623: 6.6496, 799805: 5.9565, 809900: 6.6496, 823989: 1.3315, 829284: 6.6496, 899429: 5.9565, 943858: 4.1239, 997313: 6.6496, 1002339: 0.6925, 1002345: 3.2652)}, SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {7565: 6.6496, 14394: 2.9239, 35266: 6.6496, 36755: 12.4883, 47987: 13.2993, 67332: 1.385, 83108: 3.9088, 94665: 6.6496, 102116: 6.6496, 114195: 6.6496, 192759: 5.5156, 316973: 13.2993, 352241: 6.6496, 353470: 2.77, 379216: 6.6496, 405286: 0.6925, 408632: 13.2993, 430954: 6.6496, 481187: 3.2319, 497976: 13.2993, 498723: 13.2993, 499267: 6.6496, 596575: 13.2993, 599230: 13.2993, 633299: 6.6496, 711847: 3.3389, 750620: 6.6496, 760007: 6.6496, 773540: 12.4883, 823989: 2.663, 832463: 5.9565, 840674: 12.4883, 873004: 3.8463, 877949: 6.6496, 930797: 13.2993, 951974: 3.1615, 957989: 6.6496, 969939: 13.2993, 1002336: 3.4308, 1002339: 0.6925, 1011900: 6.6496, 1013070: 6.6496, 1038693: 6.6496}), SparseVector(1048576, {0: 0.5824}), SparseVector(1048576, {3982: 4.7037, 12626: 6.6496, 23574: 4.8579, 41007: 6.6496, 58029: 4.8579, 61851: 6.6496, 67332: 1.385, 69326: 4.8579, 84583: 6.6496, 125667: 4.7037, 141446: 4.8579, 166870: 4.8579, 173311: 4.5096, 189356: 3.0523, 243330: 4.857
```

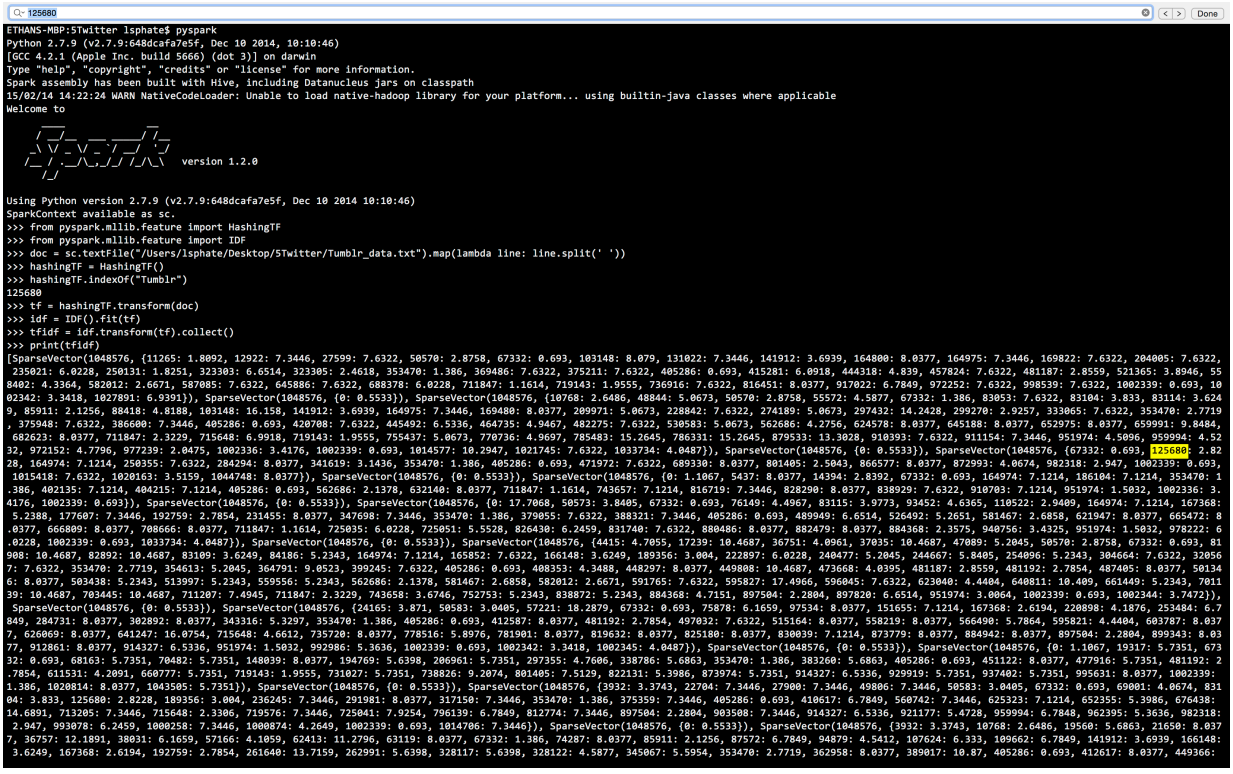
```
828897
Done
ETHANS-MBP:5Twitter lshpate$ pyspark
Python 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014, 10:10:46)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
15/02/14 14:17:50 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/

Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)
SparkContext available as sc.
>>> from pyspark.mllib.feature import HashingTF
>>> from pyspark.mllib.feature import IDF
>>> doc = sc.textFile("/Users/lshpate/Desktop/5Twitter/Burberry_data.txt").map(lambda line: line.split(' '))
>>> hashingTF = HashingTF()
>>> hashingTF.indexOf("Burberry")
828897
>>> tf = hashingTF.transform(doc)
>>> idf = IDF().fit(tf)
>>> tfidf = idf.transform(tf).collect()
>>> print(tfidf)
[SparseVector(1048576, {65562: 0.6886, 67332: 0.6886, 103234: 4.705, 138726: 4.705, 196521: 4.705, 263403: 4.705, 296522: 4.705, 300067: 4.705, 303929: 4.705, 353470: 1.3773, 373807: 3.2009, 394275: 4.705, 481187: 3.2009, 529785: 4.705, 571019: 4.2996, 625741: 4.705, 627606: 3.4523, 828897: 1.8428, 871950: 4.705, 964463: 2.3536, 970300: 4.705, 1002336: 3.2009, 1002338: 0.6886, 1033070: 4.705}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {11265: 2.3536, 59544: 4.705, 65562: 0.6886, 67332: 0.6886, 71826: 4.705, 85911: 2.1023, 166148: 4.2996, 180525: 4.705, 232298: 4.705, 250131: 2.4537, 293156: 2.7591, 294562: 4.705, 315376: 4.2996, 323305: 1.8146, 353470: 1.3773, 360474: 4.705, 415032: 3.7887, 481187: 3.2009, 531091: 4.705, 553125: 4.705, 568904: 4.705, 571019: 4.2996, 581759: 4.705, 585248: 1.8146, 604615: 3.7887, 627606: 3.4523, 650806: 3.7887, 704697: 3.3187, 711847: 1.2238, 715648: 1.8428, 719143: 3.5745, 725041: 2.066, 733117: 3.7887, 770308: 3.0956, 872993: 3.0956, 898900: 4.705, 956125: 3.4523, 964463: 2.3536, 977239: 1.9324, 999153: 4.705, 1002338: 0.6886, 1008716: 4.705}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {40231: 4.2996, 57930: 4.705, 64538: 4.705, 65562: 0.6886, 67332: 0.6886, 83115: 2.9133, 95566: 4.705, 129197: 4.2996, 345388: 4.705, 353470: 1.3773, 562686: 1.9017, 581632: 4.705, 711847: 1.2238, 719560: 4.705, 732779: 4.705, 747262: 4.705, 824454: 4.705, 828897: 1.8428, 839292: 4.705, 858724: 4.705, 890169: 4.705, 917583: 4.705, 951974: 1.1935, 982318: 2.5078, 1002338: 0.6886}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {0: 0.5078, 13360: 4.2996, 40410: 4.2996, 61031: 4.705, 65562: 0.6886, 67332: 0.6886, 104251: 3.4523, 115245: 3.7887, 124554: 4.2996, 198037: 4.2996, 202661: 4.2996, 244439: 4.2996, 247241: 4.0119, 257638: 4.0119, 258589: 2.7591, 270489: 4.2996, 303794: 4.2996, 319808: 4.2996, 339312: 4.2996, 353470: 1.3773, 397095: 4.2996, 401025: 4.2996, 420693: 8.5991, 442809: 8.5991, 446768: 4.2996, 453700: 8.5991, 454578: 4.705, 520583: 4.2996, 547777: 4.705, 579209: 4.2996, 580203: 4.2996, 599361: 3.6064, 670421: 4.2996, 678604: 4.2996, 709853: 4.2996, 711847: 1.2238, 800902: 4.2996, 844292: 4.705, 858761: 4.705, 872993: 3.0956, 900614: 8.0237, 928288: 4.2996, 951974: 1.1935, 989158: 4.2996, 999844: 4.2996, 1002338: 0.6886, 1009174: 4.2996}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {10768: 2.9133, 65562: 0.6886, 67332: 0.6886, 110522: 2.9133, 163098: 4.705, 176011: 9.41, 199934: 4.705, 282612: 4.705, 341051: 4.705, 353470: 1.3773, 360881: 4.705, 418086: 9.41, 546002: 4.705, 611589: 4.705, 620553: 4.705, 629096: 6.4019, 748456: 4.705, 800483: 4.705, 874286: 4.705, 1002338: 0.6886, 1002345: 4.0119, 1027967: 4.705}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {3932: 4.9074, 19019: 2.6901, 20838: 2.6256, 43394: 2.6256, 65562: 0.6886, 67332: 1.3773, 117847: 2.6256, 127209: 2.6256, 141486: 4.705, 166148: 4.2996, 185761: 4.705, 192759: 3.0956, 211678: 4.705, 213218: 4.705, 221003: 4.705, 255627: 2.6256, 268640: 2.6256, 284660: 2.6256, 293532: 5.3802, 299270: 3.6064, 300012: 5.3802, 323297: 4.8049, 323305: 3.6293, 353470: 2.7545, 359484: 4.705, 375246: 5.1299, 381164: 2.6901, 387619: 2.6256, 393139: 4.705, 411155: 2.6256, 423665: 4.705, 431935: 2.6901, 457035: 5.3802, 479134: 4.705, 479443: 2.6256, 484930: 9.41, 515914: 2.6256, 525870: 4.705, 567307: 2.6256, 582090: 2.6256, 585248: 3.6293, 618273: 4.705, 673776: 5.3802, 711847: 1.2238, 715648: 1.8428, 722824: 2.6256, 726619: 5.3802, 730091: 2.6901, 771136: 5.2511, 776355: 2.6901, 872992: 2.4537, 897504: 5.7973, 923416: 4.705, 925924: 2.6901, 932017: 2.6256, 940692: 2.6901, 943520: 4.705, 949743: 2.6256, 951974: 2.3869, 955651: 4.705, 959994: 3.5212, 964463: 4.7073, 965868: 4.705, 977943: 4.705, 982316: 2.1793, 982542: 4.705, 1002338: 0.6886, 1002339: 2.1793, 1002340: 2.5078, 1002341: 4.0119, 1027176: 5.3802, 1031610: 2.6256, 1043959: 3.6064, 1045028: 5.3802}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {4415: 4.0119, 60361: 4.705, 65562: 0.6886, 65938: 4.705, 67332: 0.6886, 199714: 4.705, 299871: 4.705, 337400: 4.705, 353470: 1.3773, 481631: 4.705, 535325: 4.2996, 562686: 1.9017, 581467: 2.9133, 581469: 4.2996, 665622: 4.705, 671977: 4.705, 711847: 1.2238, 787348: 4.0119, 828897: 1.8428, 872994: 4.0119, 886775: 4.2996, 920335: 4.2996, 921441: 4.2996, 948895: 3.4523, 951974: 1.1935, 959994: 1.7606, 1002338: 0.6886, 1008906: 4.705, 1040951: 4.705}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {3932: 2.4537, 11265: 2.3536, 13298: 4.705, 14394: 2.1793, 19019: 2.6901, 20838: 2.6256, 43394: 2.6256, 65562: 0.6886, 65940: 4.705, 67332: 1.3773, 83111: 4.2996, 85911: 2.1023, 117847: 2.6256, 127209: 2.6256, 133398: 4.705, 250131: 2.4537, 255627: 2.6256, 268640: 2.6256, 284660: 2.6256, 293532: 5.3802, 300012: 5.3802, 323297: 4.8049, 323305: 3.6293, 351085: 4.705, 353470: 2.7545, 375246: 5.1299, 381164: 2.6901, 385480: 4.705, 387619: 2.6256, 411155: 2.6256, 431935: 2.6901, 457035: 5.3802, 479443: 2.6256, 510873: 4.705, 515037: 4.705, 515914: 2.6256, 524029: 4.705, 567307: 2.6256, 582090: 2.6256, 585248: 3.6293, 669617: 4.705, 673776: 5.3802, 711847: 1.2238, 719143: 1.7872, 722824: 2.6256, 726619: 5.3802, 730091: 2.6901, 771136: 5.2511, 776355: 2.6901, 783644: 4.705, 838045: 4.705, 872992: 2.4537, 897504: 5.7973, 925924: 2.6901, 932017: 2.6256, 940692: 2.6901, 949743: 2.6256, 951974: 1.1935, 959994: 3.5212, 964463: 4.7073, 977239: 1.9324, 982316: 2.1793, 1002338: 0.6886, 1002339: 2.1793, 1002340: 2.5078, 1027176: 5.3802, 1031610: 2.6256, 1041257: 4.705, 1041684: 4.2996, 1045028: 5.3802}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {39768: 3.7887, 65562: 0.6886, 67332: 0.6886, 76510: 3.7887, 85911: 2.1023, 114524: 4.705, 127559: 3.7887, 140083: 4.705, 269607: 3.6064, 293156: 2.7591, 319194: 3.6064, 340634: 4.705, 353470: 1.3773, 497001: 3.7887, 578391: 4.0119, 583270: 3.7887, 584625: 3.7887, 604425: 3.7887, 622798: 3.7887, 715648: 1.8428, 719143: 1.7872, 725041: 2.066, 757050: 4.705, 770308: 3.0956, 773691: 4.2996, 820387: 4.705, 833837: 3.7887, 877032: 4.705, 897504: 1.9324, 977239: 1.9324, 982316: 2.1793, 984906: 3.7887, 1002336: 3.2009, 1002338: 0.6886, 1030563: 3.7887}), SparseVector(1048576, {0: 0.5078}), SparseVector(1048576, {39768: 3.7887, 65562: 0.6886, 67332: 0.6886, 76510: 3.7887, 85911: 2.1023, 114524: 4.705, 127559: 3.7887, 269607: 3.6064,
```

928201 Done

```
ETHANS-MBP:5Twitter lsphate$ pyspark
Python 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014, 10:10:46)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
15/02/14 14:19:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/

Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)
SparkContext available as sc.
>>> from pyspark.mllib.feature import HashingTF
>>> from pyspark.mllib.feature import IDF
>>> doc = sc.textFile("/Users/lsphate/Desktop/5TWitter/Heineken_data.txt").map(lambda line: line.split(' '))
>>> hashingTF = HashingTF()
>>> hashingTF.indexOf("Heineken")
928201
>>> tf = hashingTF.transform(doc)
>>> idf = IDF().fit(tf)
>>> tfidf = idf.transform(tf).collect()
>>> print(tfidf)
[SparseVector(1048576, {3932: 1.8326, 26942: 2.5257, 40140: 2.5257, 56805: 2.5257, 67332: 0.6539, 70933: 2.5257, 85911: 2.1203, 98147: 2.5257, 100335: 2.5257, 168353: 2.5257, 189632: 2.5257, 194731: 2.5257, 220371: 2.5257, 225462: 2.5257, 237694: 2.5257, 258589: 2.1203, 293156: 2.5257, 315202: 2.5257, 323305: 2.1203, 323325: 2.5257, 351085: 2.5257, 353470: 1.3079, 390422: 2.5257, 394554: 2.5257, 405286: 0.6539, 452545: 5.0515, 521365: 2.5257, 525510: 2.5257, 545752: 2.5257, 554605: 2.5257, 561031: 2.5257, 637757: 2.5257, 645467: 2.5257, 696315: 2.5257, 715648: 5.4977, 719143: 2.1203, 725041: 2.1203, 735901: 2.5257, 743658: 2.5257, 778958: 2.5257, 801002: 2.5257, 885769: 2.1203, 886775: 2.5257, 897504: 5.4977, 923062: 2.5257, 928201: 1.273, 951974: 2.8542, 956666: 2.5257, 959994: 4.2405, 977239: 2.1203, 979160: 2.5257, 1002339: 0.6539, 1017729: 2.5257}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {28777: 2.5257, 50570: 2.5257, 67332: 1.3079, 83115: 2.1203, 110522: 2.5257, 123936: 2.5257, 141912: 2.5257, 175051: 2.5257, 220898: 2.5257, 307883: 2.5257, 353470: 1.3079, 371467: 2.5257, 405286: 0.6539, 414692: 2.5257, 451016: 2.5257, 480044: 2.5257, 533799: 2.5257, 562686: 2.5257, 629096: 2.5257, 635809: 2.5257, 667214: 2.5257, 689935: 2.5257, 711847: 1.273, 714030: 2.5257, 714964: 2.5257, 715648: 1.8326, 725041: 2.1203, 746672: 2.5257, 757033: 2.5257, 803628: 2.5257, 850325: 2.5257, 855845: 2.5257, 863830: 2.5257, 897504: 1.8326, 928201: 1.273, 951974: 1.4271, 1002339: 0.6539, 1035098: 2.5257}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {19707: 2.5257, 36481: 2.5257, 67332: 0.6539, 94816: 2.5257, 103984: 2.5257, 142159: 2.5257, 145380: 1.8326, 151931: 2.5257, 173308: 2.5257, 174435: 2.5257, 183002: 2.5257, 258589: 2.1203, 323324: 2.1203, 353470: 1.3079, 405286: 0.6539, 452521: 2.5257, 503733: 2.5257, 507227: 2.5257, 595824: 2.1203, 656979: 2.5257, 668229: 2.5257, 697419: 2.2789, 708149: 2.5257, 763781: 2.1203, 770044: 2.5257, 872993: 2.1203, 874195: 2.5257, 885769: 2.1203, 895821: 2.5257, 903718: 1.8326, 1002339: 0.6539, 1023430: 2.5257, 1032731: 2.5257}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {3932: 1.8326, 6686: 2.5257, 67332: 0.6539, 83115: 2.1203, 112286: 2.5257, 200447: 2.5257, 212322: 2.5257, 250641: 2.5257, 287785: 2.5257, 315367: 2.5257, 325752: 2.5257, 332102: 2.5257, 353470: 1.3079, 360956: 2.5257, 405286: 0.6539, 481187: 1.8326, 509948: 2.5257, 538478: 2.5257, 607597: 2.5257, 621085: 2.5257, 634034: 2.5257, 684190: 2.5257, 697419: 3.4183, 708980: 2.1203, 711207: 1.8326, 732977: 2.5257, 738559: 2.5257, 760381: 2.5257, 782260: 2.5257, 782320: 2.5257, 806501: 2.5257, 854510: 2.5257, 857364: 2.5257, 911153: 2.5257, 928201: 1.273, 956260: 2.5257, 977122: 2.5257, 987910: 2.5257, 1002339: 0.6539, 1007909: 2.5257}), SparseVector(1048576, {3982: 2.5257, 66395: 2.5257, 67332: 0.6539, 145380: 1.8326, 192759: 2.5257, 195138: 2.5257, 245690: 2.5257, 251589: 2.5257, 276883: 2.5257, 327577: 2.5257, 334775: 2.5257, 338076: 2.5257, 353470: 1.3079, 354366: 2.5257, 375595: 2.5257, 405286: 0.6539, 406357: 2.5257, 452519: 1.8326, 501000: 2.5257, 555097: 2.5257, 577873: 2.5257, 697409: 4.2405, 697419: 2.2789, 711847: 1.273, 844458: 2.5257, 882579: 2.5257, 901831: 2.5257, 903718: 1.8326, 928201: 1.273, 940171: 2.5257, 951974: 1.4271, 982318: 1.8326, 987607: 2.5257, 1002339: 0.6539, 1002341: 2.5257, 1015179: 2.5257}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {11265: 2.1203, 18239: 2.5257, 20342: 2.5257, 43081: 2.5257, 52512: 2.5257, 66397: 2.5257, 67332: 0.6539, 95835: 2.5257, 191546: 2.5257, 196758: 2.5257, 250131: 2.1203, 263471: 2.5257, 342332: 5.0515, 353470: 1.3079, 404961: 2.5257, 405286: 0.6539, 452934: 2.5257, 481192: 2.5257, 499734: 5.0515, 526528: 2.5257, 567937: 2.5257, 579064: 5.0515, 616296: 2.5257, 617865: 2.5257, 623167: 2.5257, 624539: 2.5257, 627500: 2.5257, 687910: 2.5257, 697419: 4.5577, 711207: 1.8326, 711847: 1.273, 754221: 2.5257, 805052: 2.5257, 849988: 2.5257, 872993: 2.1203, 928201: 1.273, 950233: 2.5257, 1002339: 0.6539}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {0: 0.5798, 12758: 2.5257, 21955: 2.5257, 44831: 2.5257, 50591: 2.5257, 57166: 2.5257, 67332: 0.6539, 100895: 2.5257, 108488: 2.5257, 136113: 2.5257, 240477: 2.5257, 246320: 2.5257, 271680: 2.5257, 300113: 2.5257, 323305: 2.1203, 353470: 1.3079, 373202: 2.5257, 405286: 0.6539, 433205: 2.5257, 434774: 2.5257, 441832: 2.5257, 602353: 2.5257, 619616: 2.5257, 651783: 2.1203, 679471: 2.5257, 681566: 2.5257, 690636: 2.5257, 697409: 2.1203, 711847: 1.273, 715648: 3.6652, 727606: 2.5257, 730956: 2.5257, 738562: 2.5257, 771266: 2.5257, 777852: 2.5257, 809878: 2.5257, 833616: 2.5257, 845381: 2.5257, 872997: 2.5257, 897504: 1.8326, 903272: 2.5257, 951974: 2.8542, 956097: 2.5257, 969956: 2.5257, 982318: 1.8326, 994943: 2.5257, 1002339: 0.6539}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {38018: 2.5257, 46657: 2.5257, 67332: 0.6539, 69151: 2.5257, 83105: 2.5257, 93452: 2.5257, 132209: 2.5257, 145380: 1.8326, 165924: 2.5257, 166624: 2.5257, 175980: 2.5257, 189356: 2.5257, 202973: 2.5257, 218604: 2.5257, 293440: 2.5257, 323324: 2.1203, 343065: 2.5257, 353470: 1.3079, 368533: 2.5257, 397623: 2.5257, 405286: 0.6539, 452519: 1.8326, 453508: 2.5257, 481187: 1.8326, 489028: 2.5257, 579994: 2.5257, 595824: 2.1203, 611142: 2.5257, 612046: 2.5257, 667167: 2.5257, 697419: 1.1394, 763781: 2.1203, 778364: 2.5257, 803949: 2.5257, 832474: 2.5257, 872385: 2.5257, 886510: 2.5257, 887622: 2.5257, 888254: 2.5257, 893024: 2.5257, 897286: 2.5257, 901287: 2.5257, 903718: 1.8326, 927483: 2.5257, 940756: 2.5257, 962364: 2.5257, 1002339: 0.6539, 1031406: 2.5257}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {11265: 2.1203, 67332: 0.6539, 74578: 2.5257, 83221: 2.5257, 141915: 2.5257, 250131: 2.1203, 353025: 2.5257, 353470: 1.3079, 405286: 0.6539, 425412: 2.5257, 497933: 2.5257, 583546: 2.5257, 606109: 2.5257, 613209: 2.5257, 629344: 2.5257, 630256: 2.5257, 673313: 2.5257, 700801: 2.5257, 702360: 2.5257, 708960: 2.5257, 711847: 1.273, 789239: 2.5257, 801405: 5.0515, 848704: 2.5257, 872996: 5.0515, 937259: 2.5257, 938324: 2.5257, 948768: 2.5257, 982318: 1.8326, 1002339: 0.6539}), SparseVector(1048576, {0: 0.5798}), SparseVector(1048576, {3932: 1.8326, 6292: 2.5257, 67332: 0.6539, 71628: 2.5257, 74816: 2.5257, 76511: 2.5257, 153127: 2.5257, 166005: 2.52
```

551216 Done

```
ETHANS-MBP:5Twitter lsphate$ pyspark
Python 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014, 10:10:46)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
15/02/14 14:21:05 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/

Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)
SparkContext available as sc.
>>> from pyspark.mllib.feature import HashingTF
>>> from pyspark.mllib.feature import IDF
>>> doc = sc.textFile("/Users/lsphate/Desktop/5TWitter/Nikon_data.txt").map(lambda line: line.split(' '))
>>> hashingTF = HashingTF()
>>> hashingTF.indexOf("Nikon")
551216
>>> tf = hashingTF.transform(doc)
>>> idf = IDF().fit(tf)
>>> tfidf = idf.transform(tf).collect()
>>> print(tfidf)
[SparseVector(1048576, {6572: 4.9323, 41130: 4.6298, 67332: 0.6924, 96843: 6.5418, 255349: 6.1363, 261285: 5.6255, 284496: 5.0377, 353470: 1.3849, 373807: 1.1459, 405286: 0.6924, 411090: 2.1597, 412665: 6.5418, 412981: 6.5418, 459287: 4.837, 473392: 1.9466, 479860: 6.5418, 510812: 5.6255, 512713: 3.9391, 540395: 4.9323, 551216: 1.0466, 597291: 5.6255, 641942: 3.2276, 756062: 5.8486, 847769: 6.5418, 872993: 3.2837, 877218: 6.5418, 921822: 2.3829, 982318: 3.3431, 988209: 2.9864, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {11919: 6.5418, 67332: 0.6924, 94879: 5.289, 177342: 6.5418, 255349: 6.1363, 261285: 5.6255, 303183: 4.5958, 319194: 3.9027, 353470: 1.3849, 373807: 1.1459, 405286: 0.6924, 411090: 2.1597, 473392: 1.9466, 510812: 5.6255, 520421: 2.5905, 551216: 1.0466, 597291: 5.6255, 669843: 1.4919, 678963: 6.5418, 729840: 5.289, 760943: 5.4431, 768071: 5.1555, 793550: 1.8277, 795747: 3.8676, 828458: 5.0377, 847770: 6.5418, 860310: 5.6255, 872993: 3.2837, 893335: 1.7668, 921822: 2.3829, 982318: 3.3431, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {67332: 0.6924, 70425: 6.5418, 222557: 3.1574, 261285: 5.6255, 303183: 4.5958, 353470: 1.3849, 373807: 1.1459, 405286: 0.6924, 427429: 6.5418, 510812: 5.6255, 551216: 1.0466, 590524: 5.8486, 597291: 5.6255, 641929: 5.1555, 642463: 6.5418, 653028: 6.5418, 703110: 6.5418, 740142: 4.75, 778615: 6.5418, 779275: 3.7692, 796426: 6.5418, 847773: 6.5418, 849302: 6.1363, 872993: 3.2837, 898379: 6.1363, 921822: 2.3829, 951974: 2.2377, 982318: 3.3431, 1002339: 0.6924, 1043920: 4.3445}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {41130: 2.3149, 44599: 2.3986, 58451: 6.1363, 67332: 0.6924, 211451: 6.5418, 218031: 2.9042, 261285: 5.6255, 296471: 6.5418, 353470: 1.3849, 373807: 1.1459, 405286: 0.6924, 411090: 2.1597, 449584: 6.5418, 473392: 3.8933, 510812: 5.6255, 551216: 1.0466, 597291: 5.6255, 634034: 4.2392, 645010: 6.5418, 669843: 1.4919, 788773: 3.9027, 793550: 1.8277, 847760: 6.5418, 872993: 3.2837, 893335: 1.7668, 918744: 6.1363, 921822: 2.3829, 982318: 3.3431, 997920: 3.4972, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {16935: 2.5344, 25953: 2.6002, 39873: 4.1904, 67332: 0.6924, 91404: 6.5418, 136704: 2.231, 152123: 2.6099, 182346: 6.1363, 189356: 2.4727, 201135: 2.5528, 264272: 2.6198, 279073: 6.1363, 334786: 6.1363, 353470: 1.3849, 374958: 2.2933, 405286: 0.6924, 405672: 5.6255, 415258: 2.5528, 436947: 6.5418, 443628: 2.5436, 445160: 3.7692, 450747: 2.5528, 521379: 2.5344, 526581: 3.1745, 551216: 1.0466, 579017: 5.0507, 591779: 2.5528, 599058: 2.5528, 623235: 6.5418, 625434: 3.3229, 669843: 1.4919, 685641: 6.5418, 756338: 6.5418, 799195: 6.5418, 872997: 4.016, 893335: 1.7668, 910325: 2.5436, 921822: 2.3829, 951760: 5.6255, 974763: 5.6255, 1002339: 0.6924, 1027233: 2.5528}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {67332: 0.6924, 141912: 5.6255, 160272: 6.5418, 209208: 6.5418, 228021: 6.5418, 249630: 6.5418, 252992: 6.1363, 255158: 6.5418, 258589: 2.7131, 265812: 6.5418, 294285: 6.5418, 298771: 6.5418, 303344: 6.5418, 323297: 9.5, 353470: 1.3849, 364075: 6.1363, 405286: 0.6924, 496362: 6.5418, 521365: 6.1363, 542724: 6.5418, 594160: 6.5418, 600194: 6.5418, 606388: 6.5418, 667485: 4.2905, 682944: 12.2726, 686021: 6.5418, 709040: 6.5418, 715648: 3.3431, 765663: 6.5418, 781415: 6.5418, 799193: 6.5418, 811753: 5.8486, 882415: 6.5418, 897504: 3.5713, 929795: 6.5418, 951974: 2.2377, 956125: 3.8676, 981090: 6.5418, 1002339: 0.6924, 1002341: 4.6699}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {19220: 3.2837, 67332: 0.6924, 83109: 3.9768, 161417: 5.1555, 189356: 2.4727, 258589: 2.7131, 353470: 1.3849, 373807: 1.1459, 405286: 0.6924, 420848: 5.0377, 446852: 4.9323, 454128: 4.75, 489776: 6.5418, 526581: 3.1745, 545298: 6.5418, 551216: 1.0466, 572636: 6.5418, 659583: 6.5418, 669843: 1.4919, 724962: 6.5418, 793550: 1.8277, 799198: 6.5418, 823080: 5.8486, 842910: 5.6255, 869230: 5.4431, 893335: 1.7668, 931607: 4.9323, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {14394: 2.9174, 32836: 6.1363, 67332: 0.6924, 71512: 6.5418, 116858: 6.5418, 230498: 3.1574, 339812: 6.5418, 343705: 6.5418, 348062: 6.5418, 353470: 1.3849, 373807: 1.1459, 395403: 3.7692, 405286: 0.6924, 432117: 6.5418, 488922: 6.5418, 545276: 6.5418, 551216: 1.0466, 669843: 1.4919, 743345: 6.5418, 793550: 3.6555, 799197: 6.1363, 826121: 6.1363, 872995: 3.6514, 893335: 1.7668, 951974: 2.2377, 999892: 6.1363, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {19220: 3.2837, 31839: 5.8486, 39988: 6.5418, 67332: 0.6924, 69505: 6.5418, 83109: 3.9768, 178093: 6.5418, 258589: 2.7131, 269891: 6.1363, 348038: 4.0994, 353470: 1.3849, 369921: 4.3445, 373807: 1.1459, 395403: 3.7692, 405286: 0.6924, 435863: 6.5418, 551216: 1.0466, 592438: 6.5418, 593806: 5.6255, 669843: 1.4919, 799197: 6.1363, 826814: 5.8486, 885693: 6.5418, 893335: 1.7668, 931607: 4.9323, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {67332: 0.6924, 83109: 3.9768, 258589: 2.7131, 283452: 6.5418, 345589: 6.5418, 349197: 6.5418, 353470: 1.3849, 373807: 1.1459, 391240: 6.5418, 395403: 3.7692, 405286: 0.6924, 420848: 5.0377, 499891: 3.9027, 636664: 6.1363, 646961: 6.1363, 667487: 6.1363, 793550: 1.8277, 799196: 6.5418, 823083: 6.1363, 852669: 6.1363, 868906: 6.1363, 931607: 4.9323, 950704: 4.5268, 1002339: 0.6924, 1010135: 4.5958, 1021158: 6.5418}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {19220: 3.2837, 67332: 0.6924, 83109: 3.9768, 143758: 6.5418, 189356: 2.4727, 258589: 2.7131, 353470: 1.3849, 360261: 5.8486, 373807: 1.1459, 405286: 0.6924, 520421: 2.5905, 551216: 2.0933, 551604: 6.5418, 554148: 6.1363, 669843: 1.4919, 697155: 6.1363, 760084: 5.289, 770872: 6.5418, 788773: 3.9027, 791982: 6.1363, 799131: 6.5418, 799187: 6.5418, 874271: 6.5418, 931607: 4.9323, 956125: 3.8676, 1002339: 0.6924}), SparseVector(1048576, {0: 0.6215}), SparseVector(1048576, {14394: 2.9174, 32836: 6.1363, 67332: 0.6924, 67574: 6.1363, 71446: 6.5418, 83940: 6.1363, 218031: 2.9042, 221326: 6.1363, 269891: 6.1363, 275384: 6.1363, 318464: 6.1363, 353470: 1.3849, 369921: 4.3445, 373807: 1.1459, 405286: 0.6924, 431711: 6.5418, 432164: 6.5418, 473392: 1.9466, 496074: 5.8486, 540395: 4.9323, 551216: 1.0466, 750620: 6.5418, 812436: 5.8486, 825348: 5.8486, 826121: 6.1363, 8
```



Prob. 4 The Stock Prices of 5 Companies

Step 1. Pull the real-time stock prices

We need the yahoo-finance library to get the shared data of stocks from Yahoo Finance. I downloaded, installed the library, and wrote a Python application to get the data.

```
import time
from yahoo_finance import Share

price = Share('COMPANY CODE')
print price.get_open() + " at open"
print price.get_price() + " at " + price.get_trade_datetime()
for x in range(0, 28):
    time.sleep(60)
    price = Share('COMPANY CODE')
    print price.get_price() + " at " + price.get_trade_datetime()

print "Data pulling completed."
```

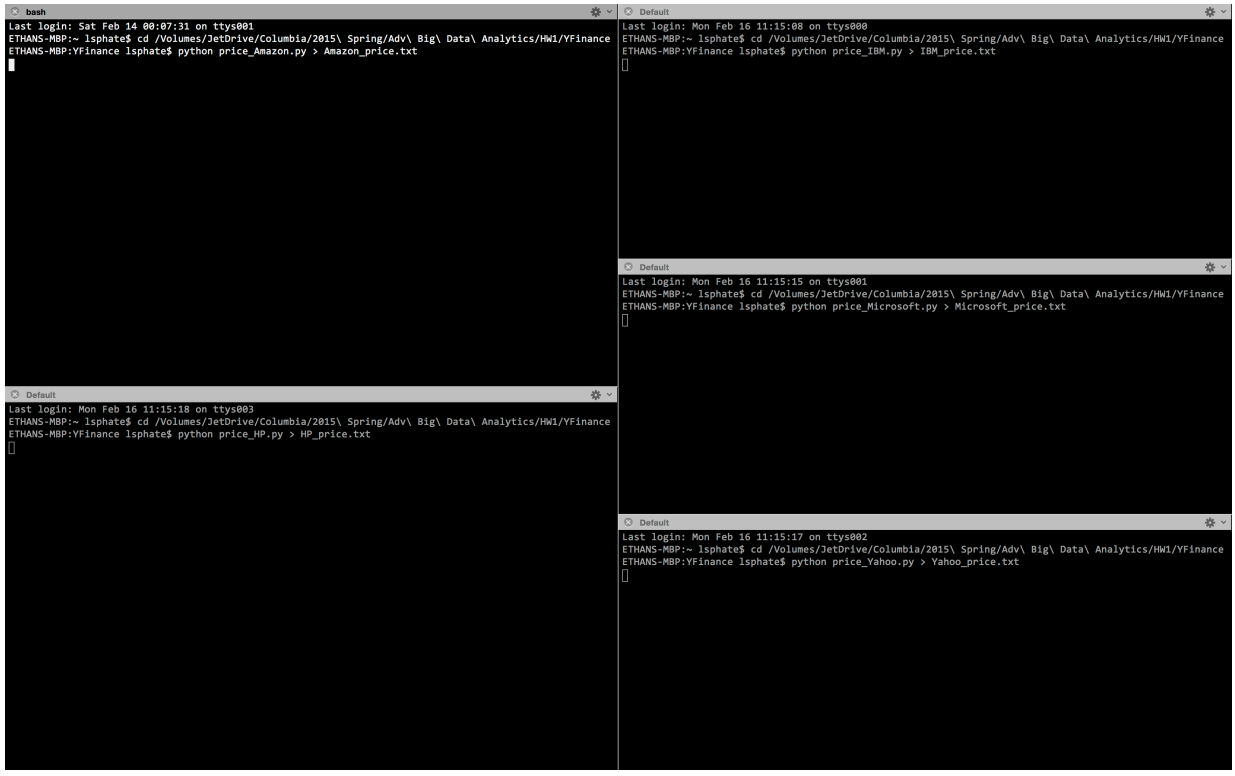
My choises of 5 companies are **Amazon**, **HP**, **IBM**, **Microsof**t and **Yahoo**. The datas I pulled down and outputed were in thi format:

```
{'Sector': 'Services', 'end': '2015-02-17', 'CompanyName': None, 'symbol': 'AMZN', 'start': '1997-05-16',
'FullTimeEmployees': '154100', 'Industry': 'Catalog & Mail Order Houses'}
378.10 at open
375.2698 at 2015-02-17 18:01:00 UTC+0000
375.12 at 2015-02-17 18:01:00 UTC+0000
375.01 at 2015-02-17 18:03:00 UTC+0000
375.01 at 2015-02-17 18:03:00 UTC+0000
375.29 at 2015-02-17 18:05:00 UTC+0000
375.25 at 2015-02-17 18:06:00 UTC+0000
375.355 at 2015-02-17 18:07:00 UTC+0000
375.19 at 2015-02-17 18:08:00 UTC+0000
375.11 at 2015-02-17 18:09:00 UTC+0000
375.05 at 2015-02-17 18:10:00 UTC+0000
374.90 at 2015-02-17 18:11:00 UTC+0000
374.792 at 2015-02-17 18:12:00 UTC+0000
374.58 at 2015-02-17 18:13:00 UTC+0000
374.65 at 2015-02-17 18:14:00 UTC+0000
375.33 at 2015-02-17 18:14:00 UTC+0000
375.29 at 2015-02-17 18:16:00 UTC+0000
375.14 at 2015-02-17 18:17:00 UTC+0000
375.245 at 2015-02-17 18:18:00 UTC+0000
375.54 at 2015-02-17 18:19:00 UTC+0000
375.18 at 2015-02-17 18:20:00 UTC+0000
375.18 at 2015-02-17 18:20:00 UTC+0000
375.475 at 2015-02-17 18:22:00 UTC+0000
375.68 at 2015-02-17 18:23:00 UTC+0000
375.524 at 2015-02-17 18:24:00 UTC+0000
375.36 at 2015-02-17 18:25:00 UTC+0000
375.14 at 2015-02-17 18:26:00 UTC+0000
375.248 at 2015-02-17 18:27:00 UTC+0000
375.3852 at 2015-02-17 18:28:00 UTC+0000
375.16 at 2015-02-17 18:28:00 UTC+0000
Data pulling completed.
```

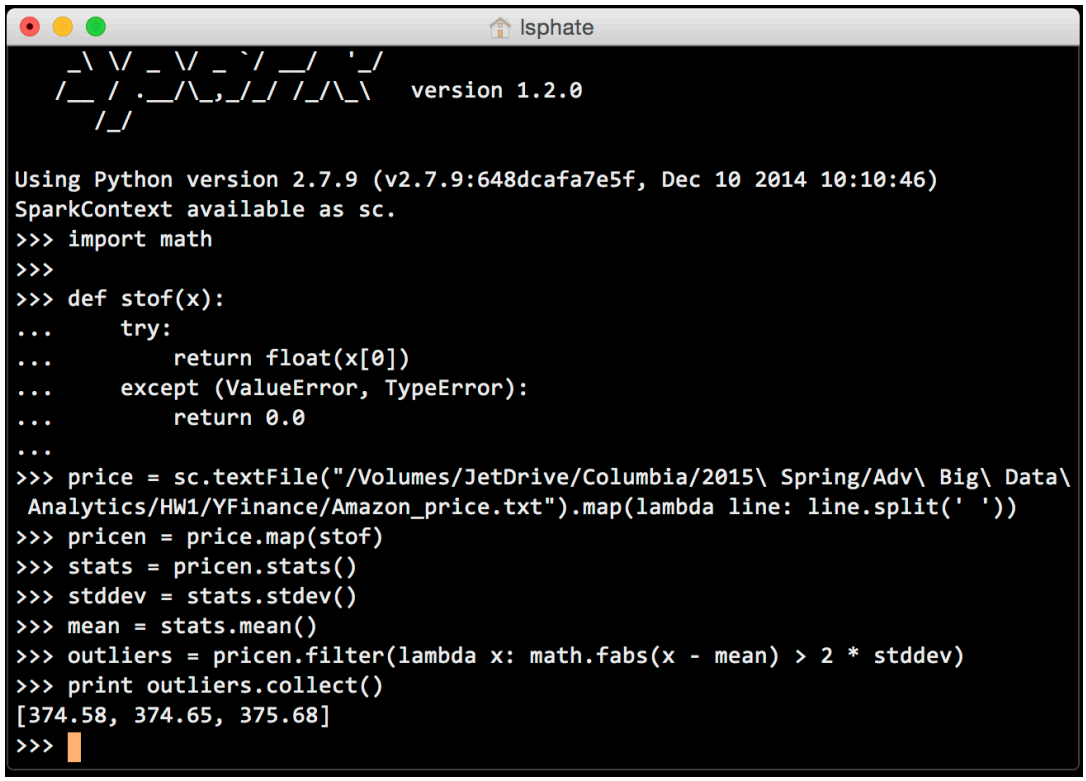
Step 2. Use filter to find outliers

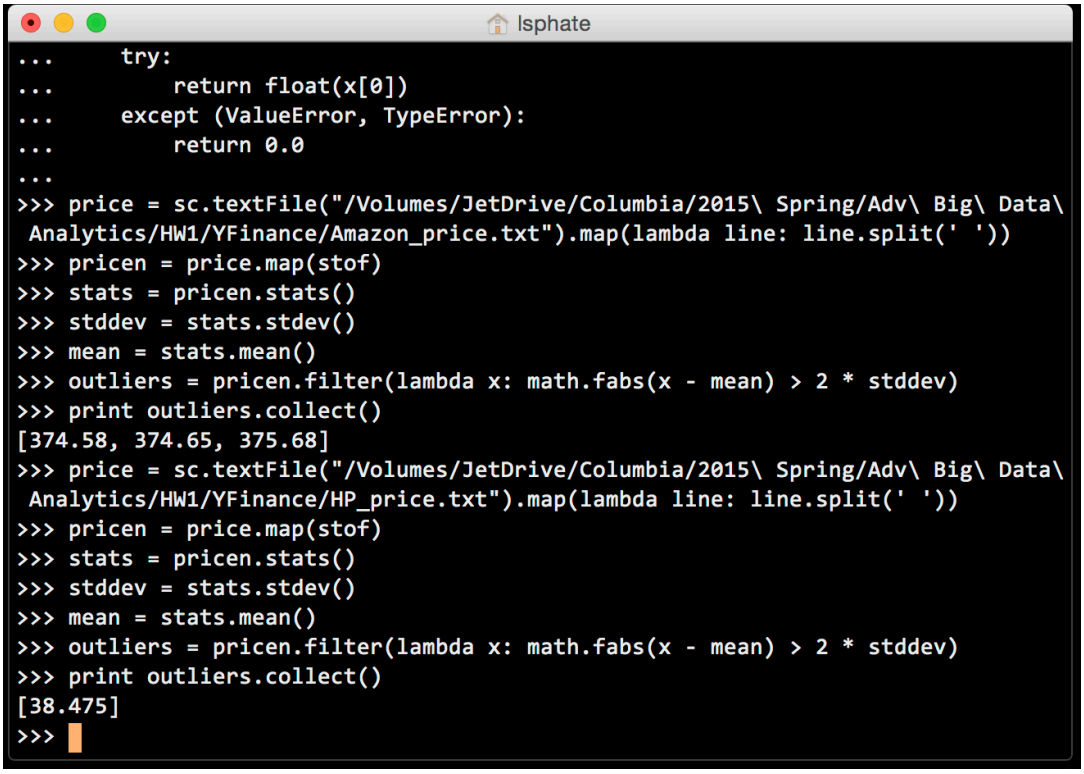
I wrote an application to extract the data points that are 2 standard deviations away against the mean of the prices. The data that imported has been pre-processed to left the price numbers only:

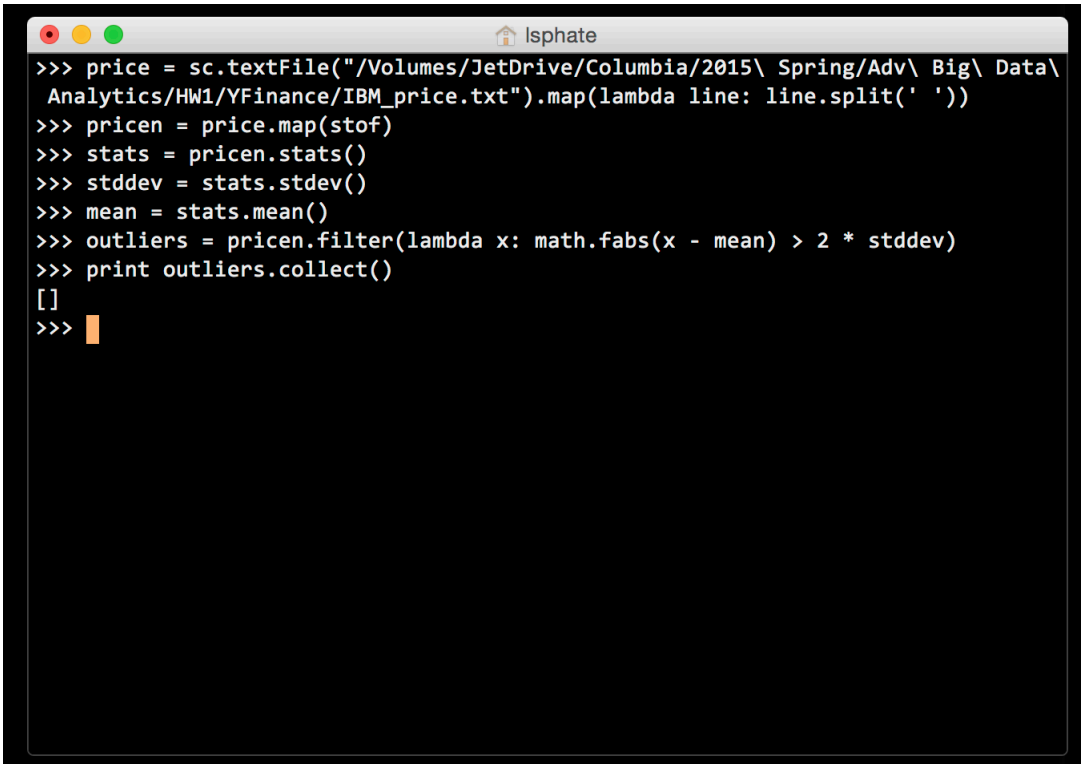
```
import math

def stof(x):
    try:
        return float(x[0])
    except (ValueError, TypeError):
        return 0.0

price = sc.textFile("PRICE_DATA_NAME").map(lambda line: line.split(' '))
pricen = price.map(stof)
stats = pricen.stats()
stddev = stats.stdev()
mean = stats.mean()
outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
print outliers.collect()
```

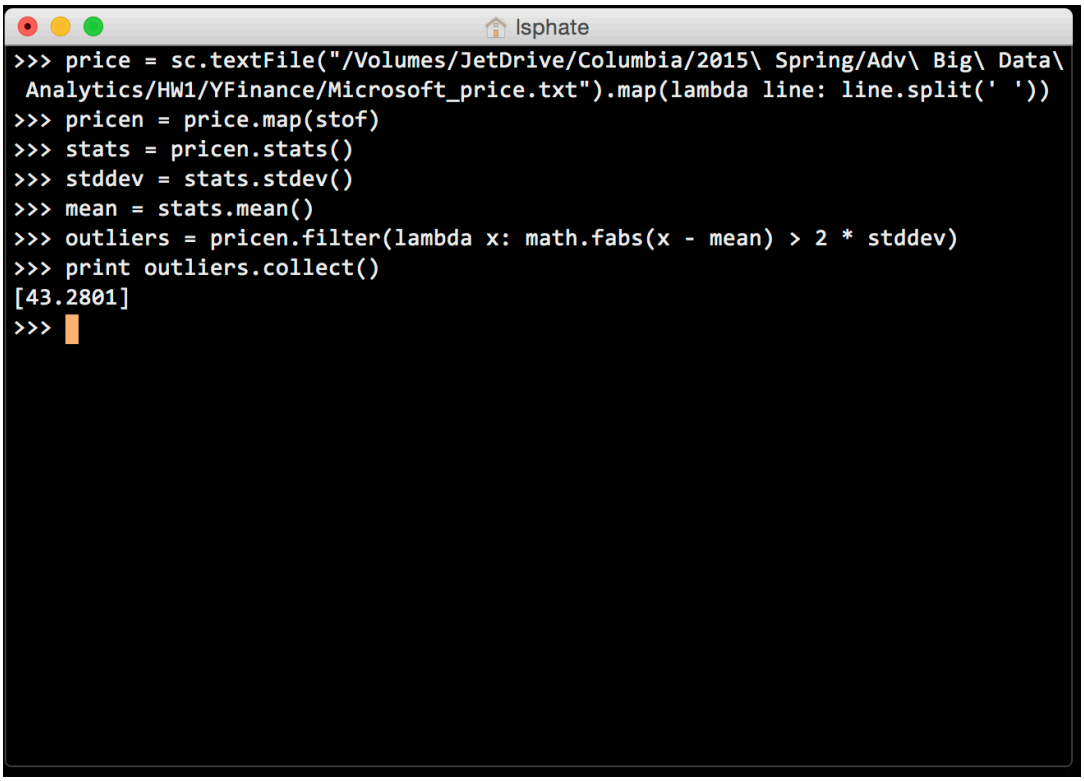


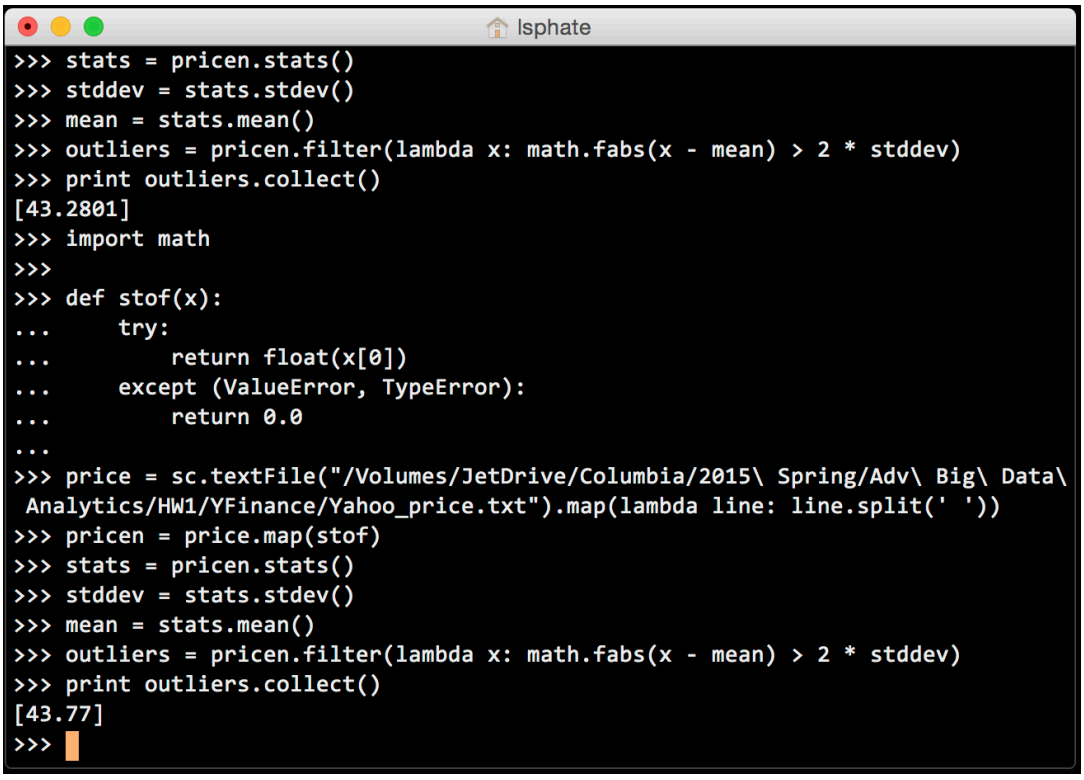
And we can get the results of the outliers:




lsphate
... try:
... return float(x[0])
... except (ValueError, TypeError):
... return 0.0
...
>>> price = sc.textFile("/Volumes/JetDrive/Columbia/2015\ Spring/Adv\ Big\ Data\
Analytics/HW1/YFinance/Amazon_price.txt").map(lambda line: line.split(' '))
>>> pricen = price.map(stof)
>>> stats = pricen.stats()
>>> stddev = stats.stdev()
>>> mean = stats.mean()
>>> outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
>>> print outliers.collect()
[374.58, 374.65, 375.68]
>>> price = sc.textFile("/Volumes/JetDrive/Columbia/2015\ Spring/Adv\ Big\ Data\
Analytics/HW1/YFinance/HP_price.txt").map(lambda line: line.split(' '))
>>> pricen = price.map(stof)
>>> stats = pricen.stats()
>>> stddev = stats.stdev()
>>> mean = stats.mean()
>>> outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
>>> print outliers.collect()
[38.475]
>>>


lsphate
>>> price = sc.textFile("/Volumes/JetDrive/Columbia/2015\ Spring/Adv\ Big\ Data\
Analytics/HW1/YFinance/IBM_price.txt").map(lambda line: line.split(' '))
>>> pricen = price.map(stof)
>>> stats = pricen.stats()
>>> stddev = stats.stdev()
>>> mean = stats.mean()
>>> outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
>>> print outliers.collect()
[]
>>>


lsphate
>>> price = sc.textFile("/Volumes/JetDrive/Columbia/2015\ Spring/Adv\ Big\ Data\
Analytics/HW1/YFinance/Microsoft_price.txt").map(lambda line: line.split(' '))
>>> pricen = price.map(stof)
>>> stats = pricen.stats()
>>> stddev = stats.stdev()
>>> mean = stats.mean()
>>> outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
>>> print outliers.collect()
[43.2801]
>>>


lsphate
>>> stats = pricen.stats()
>>> stddev = stats.stdev()
>>> mean = stats.mean()
>>> outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
>>> print outliers.collect()
[43.2801]
>>> import math
>>>
>>> def stof(x):
...   try:
...   return float(x[0])
...   except (ValueError, TypeError):
...   return 0.0
...
>>> price = sc.textFile("/Volumes/JetDrive/Columbia/2015\ Spring/Adv\ Big\ Data\
Analytics/HW1/YFinance/Yahoo_price.txt").map(lambda line: line.split(' '))
>>> pricen = price.map(stof)
>>> stats = pricen.stats()
>>> stddev = stats.stdev()
>>> mean = stats.mean()
>>> outliers = pricen.filter(lambda x: math.fabs(x - mean) > 2 * stddev)
>>> print outliers.collect()
[43.77]
>>>