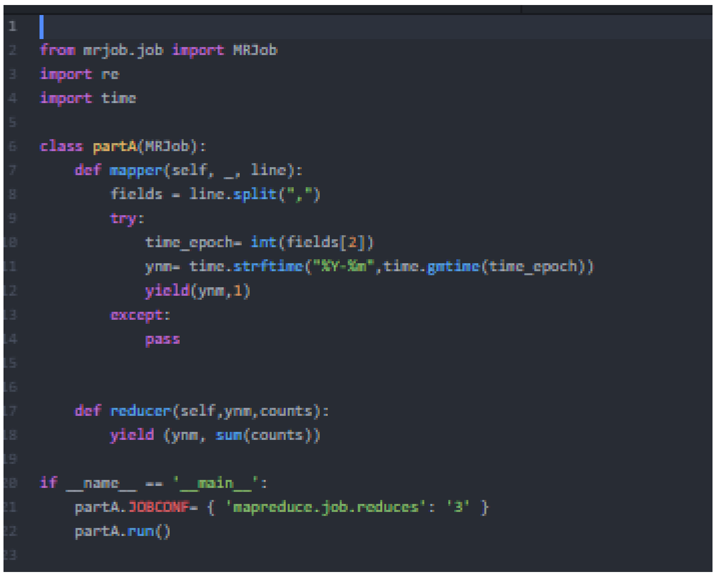
Coursework 1: Analysis of Bitcoin Transactions

Name: Amirhossein Layegh
Student Number : 180267837
Email: a.layeghkheirabadi@se18@qmul.ac.uk

PART A. TIME ANALYSIS (30%)

Obtain the top 10 donors over the whole dataset for the Wikileaks bitcoin address: {1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}. Is there any information on who these wallets belong to? Can you work out how much was being donated if converted into pounds?**(For this Part I used MRJob)**

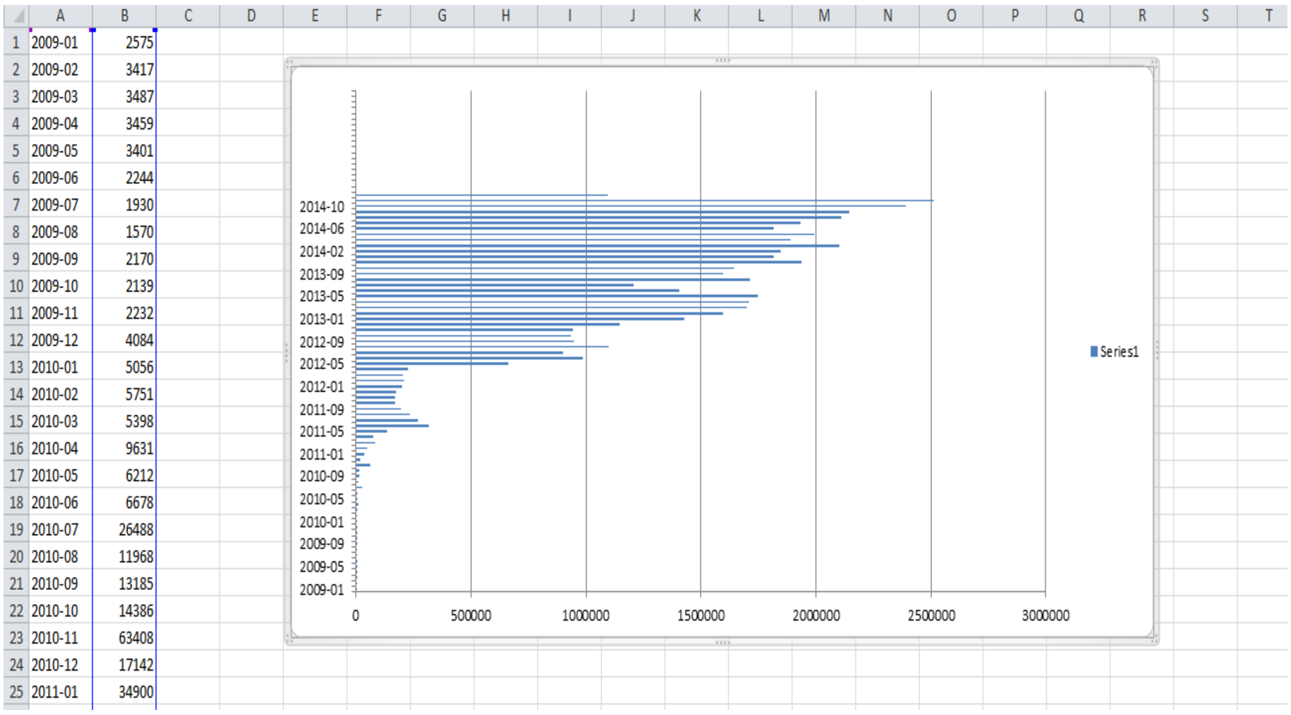


```python

from mrjob.job import MRJob
import re
import time

class partA(MRJob):
    def mapper(self, _, line):
        fields = line.split(",")
        try:
            time_epoch= int(fields[2])
            ynm= time.strftime("%Y-%m",time.gmtime(time_epoch))
            yield(ynm,1)
        except:
            pass


    def reducer(self,ynm,counts):
        yield (ynm, sum(counts))

if __name__ == '__main__':
    partA.JOBCONF= { 'mapreduce.job.reduces': '3' }
    partA.run()

```

As this file is comma delimited first we need to split them by "," and then separate them in terms of their time,

And the results are like this:

```
"2009-03"        3487
"2009-06"        2244
"2009-09"        2170
"2009-12"        4084
"2010-02"        5751
"2010-05"        6212
"2010-08"        11968
"2010-11"        63408
"2011-01"        34900
"2011-04"        73936
"2011-07"        267260
"2011-10"        168707
"2012-03"        203391
"2012-06"        986424
"2012-09"        946045
"2012-12"        1145572
"2013-02"        1597550
"2013-05"        1746928
"2013-08"        1714680
"2013-11"        1961108
"2014-01"        1817513
"2014-04"        1890902
"2014-07"        1932272
"2014-10"        2390212
```



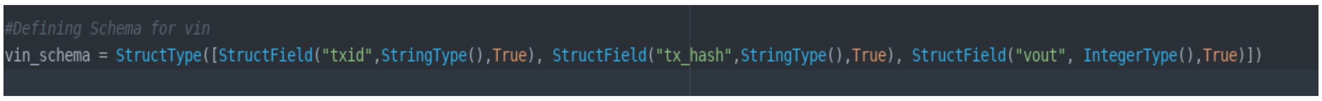
	A	B
1	2009-01	2575
2	2009-02	3417
3	2009-03	3487
4	2009-04	3459
5	2009-05	3401
6	2009-06	2244
7	2009-07	1930
8	2009-08	1570
9	2009-09	2170
10	2009-10	2139
11	2009-11	2232
12	2009-12	4084
13	2010-01	5056
14	2010-02	5751
15	2010-03	5398
16	2010-04	9631
17	2010-05	6212
18	2010-06	6678
19	2010-07	26488
20	2010-08	11968
21	2010-09	13185
22	2010-10	14386
23	2010-11	63408
24	2010-12	17142
25	2011-01	34900

Part B. Top ten donors (40%)

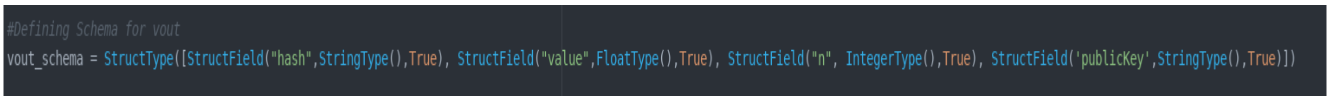
Obtain the top 10 donors over the whole dataset for the Wikileaks bitcoin address: {1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}. Is there any information on who these wallets belong to? Can you work out how much was being donated if converted into pounds?**(For this part I used pyspark)**

first of all we need to load the vin.csv and vout.csv into Dataframes which consists of two stages:

1. **Defining Schema:**

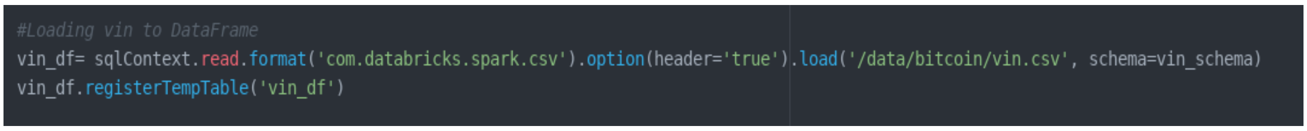


```
#Defining Schema for vin
vin_schema = StructType([StructField("txid",StringType(),True), StructField("tx_hash",StringType(),True), StructField("vout", IntegerType(),True)])
```

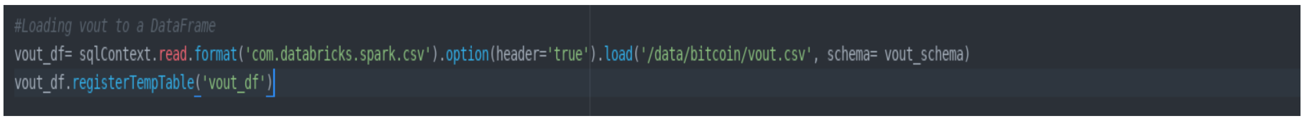


```
#Defining Schema for vout
vout_schema = StructType([StructField("hash",StringType(),True), StructField("value",FloatType(),True), StructField("n", IntegerType(),True), StructField('publicKey',StringType(),True)])
```

2.Loading to a Dataframe:

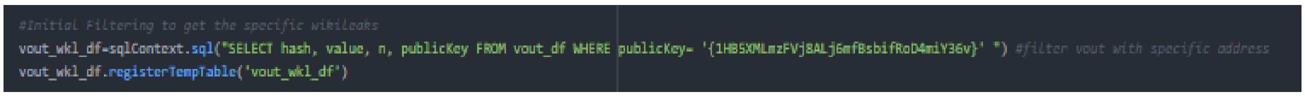


```
#Loading vin to DataFrame
vin_df= sqlContext.read.format('com.databricks.spark.csv').option(header='true').load('/data/bitcoin/vin.csv', schema=vin_schema)
vin_df.registerTempTable('vin_df')
```



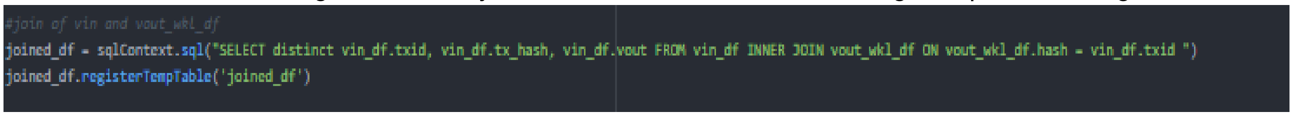
```
#Loading vout to a DataFrame
vout_df= sqlContext.read.format('com.databricks.spark.csv').option(header='true').load('/data/bitcoin/vout.csv', schema= vout_schema)
vout_df.registerTempTable('vout_df')
```

Initial Filtering: for this stage we need to filter wikileaks wallet on vout using the publicKey and create a DataFrame of them :



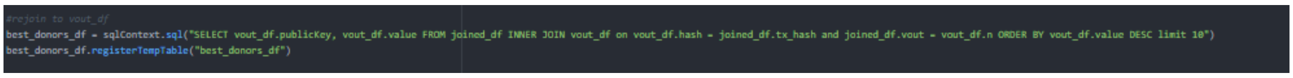
```
#Initial Filtering to get the specific wikileaks
vout_wkl_df=sqlContext.sql("SELECT hash, value, n, publicKey FROM vout_df WHERE publicKey= '{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}' ") #filter vout with specific address
vout_wkl_df.registerTempTable('vout_wkl_df')
```

First Join: At this stage we should join the smaller vout that we have got in previous stage with vin:



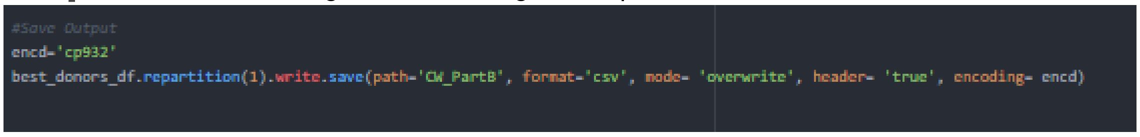
```
#join of vin and vout_wkl_df
joined_df = sqlContext.sql("SELECT distinct vin_df.txid, vin_df.tx_hash, vin_df.vout FROM vin_df INNER JOIN vout_wkl_df ON vout_wkl_df.hash = vin_df.txid ")
joined_df.registerTempTable('joined_df')
```

Second join: Now we just have the origins of the bitcoins so we have to rejoin our first join results to vout for checking their destination and ordering them to get the top 10:



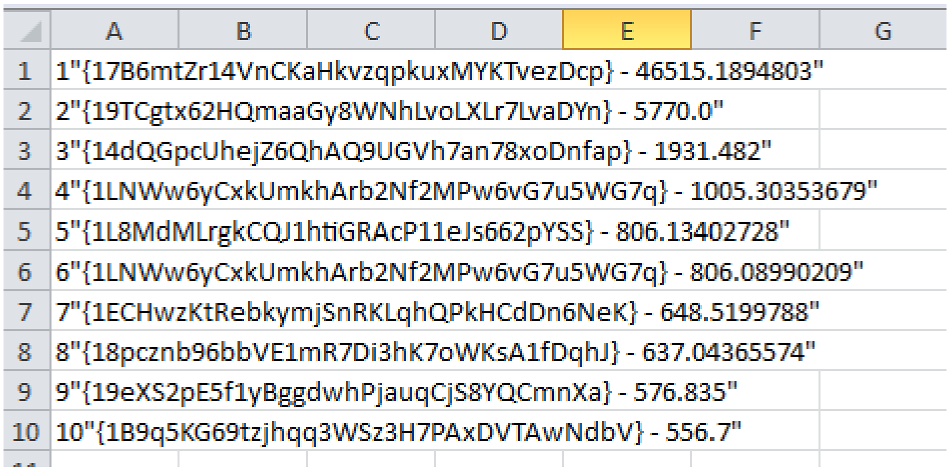
```
#rejoin to vout_df
best_donors_df = sqlContext.sql("SELECT vout_df.publicKey, vout_df.value FROM joined_df INNER JOIN vout_df on vout_df.hash = joined_df.tx_hash and joined_df.vout = vout_df.n ORDER BY vout_df.value DESC limit 10")
best_donors_df.registerTempTable("best_donors_df")
```

Output: At this final stage we are saving the output and then check them:



```
#Save Output
encd='cp932'
best_donors_df.repartition(1).write.save(path='CW_PartB', format='csv', mode= 'overwrite', header= 'true', encoding= encd)
```

The Result:

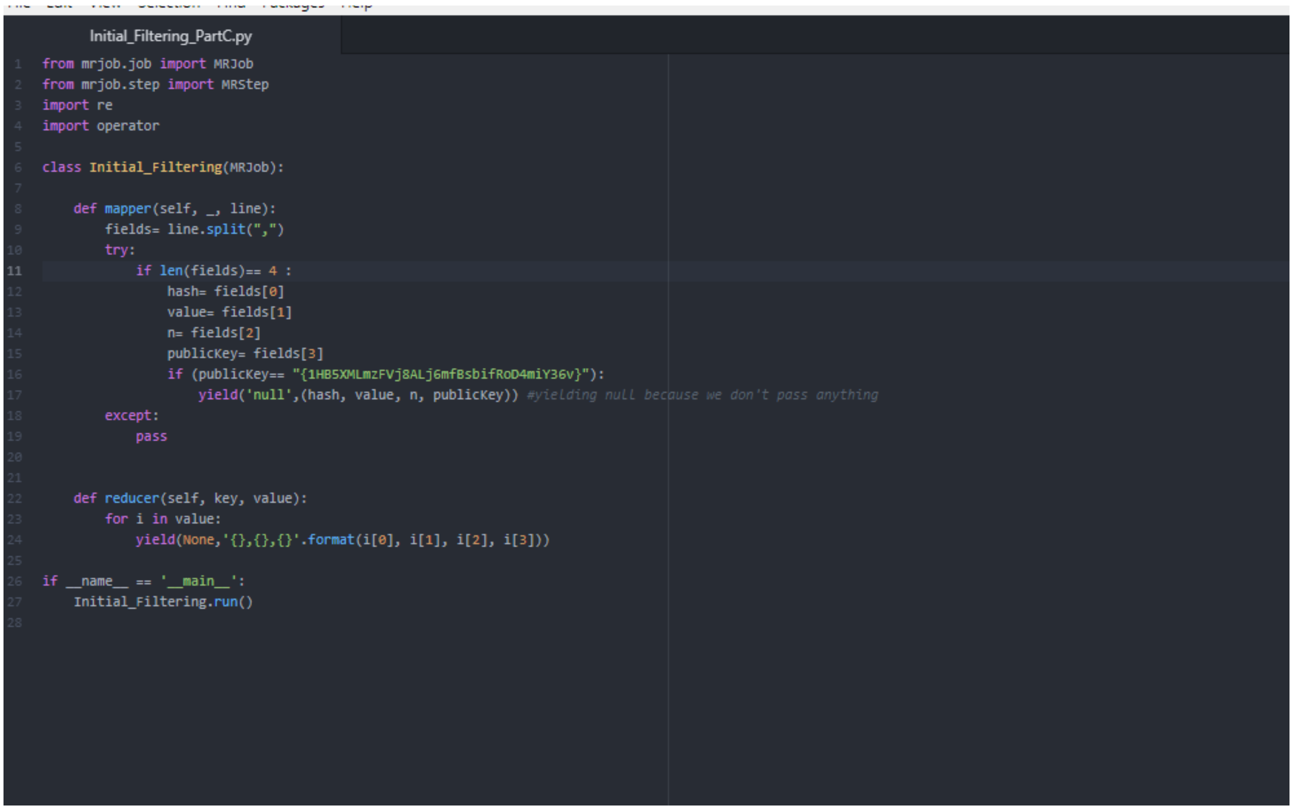


	A	B	C	D	E	F	G
1	1"{17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp} - 46515.1894803"						
2	2"{19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn} - 5770.0"						
3	3"{14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap} - 1931.482"						
4	4"{1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q} - 1005.30353679"						
5	5"{1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS} - 806.13402728"						
6	6"{1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q} - 806.08990209"						
7	7"{1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK} - 648.5199788"						
8	8"{18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ} - 637.04365574"						
9	9"{19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa} - 576.835"						
10	10"{1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV} - 556.7"						

PART C. DATA EXPLORATION (30%)

Implement Part B in both Spark and Map/Reduce. Provide a comparative analysis on how both implementations run, including consideration of observed execution times (you must run each job multiple times and compute averages to), as well as estimate on the resources involved in the cluster, as well as the number of transformations/ tasks required in each one. How does it run in comparison? What framework seems more appropriate for this task?

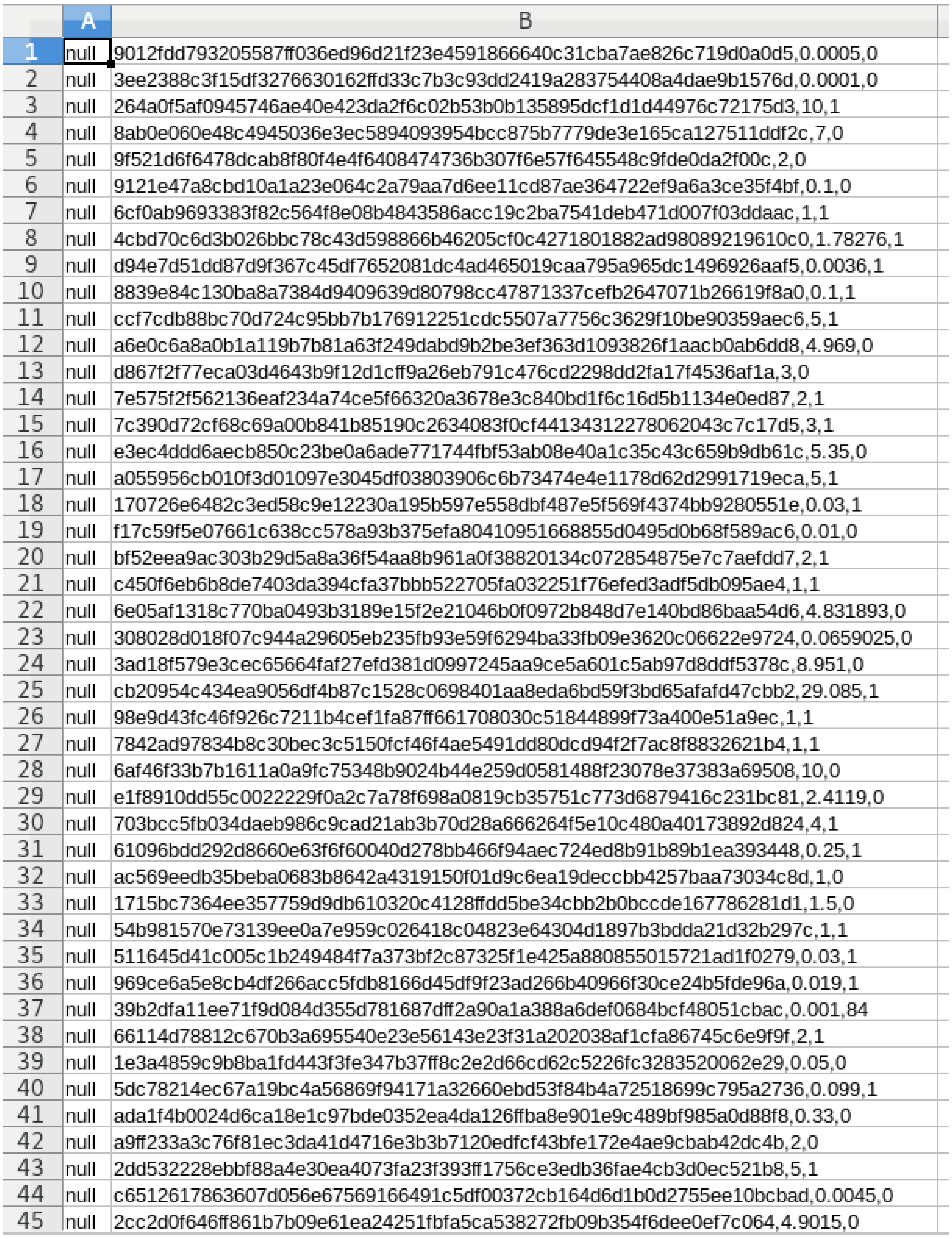
Like partB first we have to filter our data to get the smaller vout:



Initial_Filtering_PartC.py

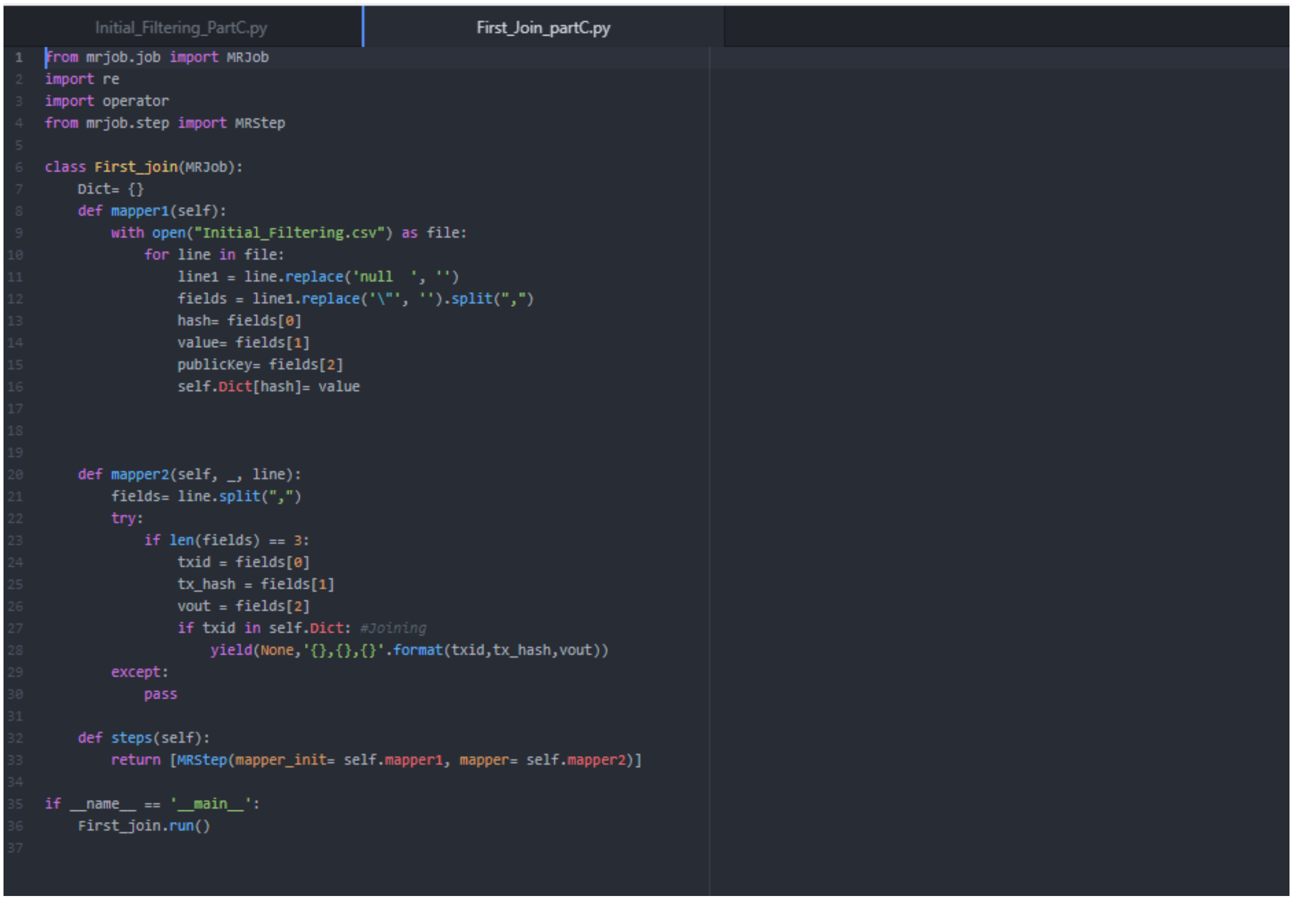
```python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import operator

class Initial_Filtering(MRJob):

    def mapper(self, _, line):
        fields= line.split(",")
        try:
            if len(fields)== 4 :
                hash= fields[0]
                value= fields[1]
                n= fields[2]
                publicKey= fields[3]
                if (publicKey== "{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}"):
                    yield('null',(hash, value, n, publicKey)) #yielding null because we don't pass anything
        except:
            pass


    def reducer(self, key, value):
        for i in value:
            yield(None,'{},{},{}'.format(i[0], i[1], i[2], i[3]))

if __name__ == '__main__':
    Initial_Filtering.run()
```

After running this code on hadoop, we need to copy the results to our local memory because we need them for the next stage (First Join). The results of this stage are like this:



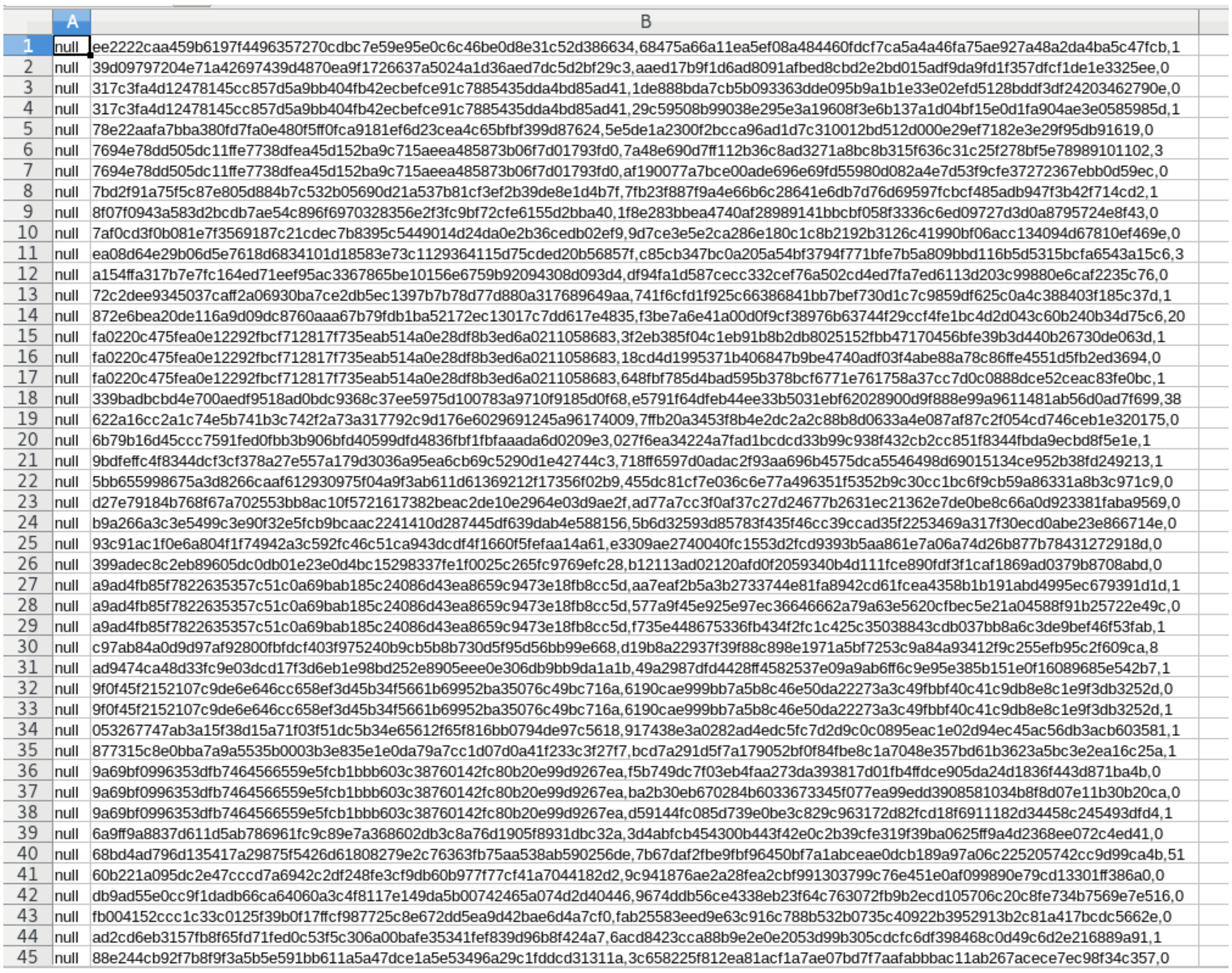
	A	B
1	null	9012fdd793205587ff036ed96d21f23e4591866640c31cba7ae826c719d0a0d5,0.0005,0
2	null	3ee2388c3f15df3276630162ffd33c7b3c93dd2419a283754408a4dae9b1576d,0.0001,0
3	null	264a0f5af0945746ae40e423da2f6c02b53b0b135895dcf1d1d44976c72175d3,10,1
4	null	8ab0e060e48c4945036e3ec5894093954bcc875b7779de3e165ca127511ddf2c,7,0
5	null	9f521d6f6478dcab8f80f4e4f6408474736b307f6e57f645548c9fde0da2f00c,2,0
6	null	9121e47a8cbd10a1a23e064c2a79aa7d6ee11cd87ae364722ef9a6a3ce35f4bf,0.1,0
7	null	6cf0ab9693383f82c564f8e08b4843586acc19c2ba7541deb471d007f03ddaac,1,1
8	null	4cbd70c6d3b026bbc78c43d598866b46205cf0c4271801882ad98089219610c0,1.78276,1
9	null	d94e7d51dd87d9f367c45df7652081dc4ad465019caa795a965dc1496926aaf5,0.0036,1
10	null	8839e84c130ba8a7384d9409639d80798cc47871337cefb2647071b26619f8a0,0.1,1
11	null	ccf7cdb88bc70d724c95bb7b176912251cdc5507a7756c3629f10be90359aec6,5,1
12	null	a6e0c6a8a0b1a119b7b81a63f249dabd9b2be3ef363d1093826f1aacb0ab6dd8,4.969,0
13	null	d867f2f77eca03d4643b9f12d1cff9a26eb791c476cd2298dd2fa17f4536af1a,3,0
14	null	7e575f2f562136eaf234a74ce5f66320a3678e3c840bd1f6c16d5b1134e0ed87,2,1
15	null	7c390d72cf68c69a00b841b85190c2634083f0cf44134312278062043c7c17d5,3,1
16	null	e3ec4ddd6aecb850c23be0a6ade771744fbf53ab08e40a1c35c43c659b9db61c,5.35,0
17	null	a055956cb010f3d01097e3045df03803906c6b73474e4e1178d62d2991719eca,5,1
18	null	170726e6482c3ed58c9e12230a195b597e558dbf487e5f569f4374bb9280551e,0.03,1
19	null	f17c59f5e07661c638cc578a93b375efa80410951668855d0495d0b68f589ac6,0.01,0
20	null	bf52eea9ac303b29d5a8a36f54aa8b961a0f38820134c072854875e7c7aefdd7,2,1
21	null	c450f6eb6b8de7403da394cfa37bbb522705fa032251f76efed3adf5db095ae4,1,1
22	null	6e05af1318c770ba0493b3189e15f2e21046b0f0972b848d7e140bd86baa54d6,4.831893,0
23	null	308028d018f07c944a29605eb235fb93e59f6294ba33fb09e3620c06622e9724,0.0659025,0
24	null	3ad18f579e3cec65664faf27efd381d0997245aa9ce5a601c5ab97d8ddf5378c,8.951,0
25	null	cb20954c434ea9056df4b87c1528c0698401aa8eda6bd59f3bd65afafd47cbb2,29.085,1
26	null	98e9d43fc46f926c7211b4cef1fa87ff661708030c51844899f73a400e51a9ec,1,1
27	null	7842ad97834b8c30bec3c5150fcf46f4ae5491dd80dcd94f2f7ac8f8832621b4,1,1
28	null	6af46f33b7b1611a0a9fc75348b9024b44e259d0581488f23078e37383a69508,10,0
29	null	e1f8910dd55c0022229f0a2c7a78f698a0819cb35751c773d6879416c231bc81,2.4119,0
30	null	703bcc5fb034daeb986c9cad21ab3b70d28a666264f5e10c480a40173892d824,4,1
31	null	61096bdd292d8660e63f6f60040d278bb466f94aec724ed8b91b89b1ea393448,0.25,1
32	null	ac569eedb35beba0683b8642a4319150f01d9c6ea19deccbb4257baa73034c8d,1,0
33	null	1715bc7364ee357759d9db610320c4128ffdd5be34cbb2b0bccde167786281d1,1.5,0
34	null	54b981570e73139ee0a7e959c026418c04823e64304d1897b3bdda21d32b297c,1,1
35	null	511645d41c005c1b249484f7a373bf2c87325f1e425a880855015721ad1f0279,0.03,1
36	null	969ce6a5e8cb4df266acc5fdb8166d45df9f23ad266b40966f30ce24b5fde96a,0.019,1
37	null	39b2dfa11ee71f9d084d355d781687dff2a90a1a388a6def0684bcf48051cbac,0.001,84
38	null	66114d78812c670b3a695540e23e56143e23f31a202038af1cfa86745c6e9f9f,2,1
39	null	1e3a4859c9b8ba1fd443f3fe347b37ff8c2e2d66cd62c5226fc3283520062e29,0.05,0
40	null	5dc78214ec67a19bc4a56869f94171a32660ebd53f84b4a72518699c795a2736,0.099,1
41	null	ada1f4b0024d6ca18e1c97bde0352ea4da126ffba8e901e9c489bf985a0d88f8,0.33,0
42	null	a9ff233a3c76f81ec3da41d4716e3b3b7120edfcf43bfe172e4ae9cbab42dc4b,2,0
43	null	2dd532228ebbf88a4e30ea4073fa23f393ff1756ce3edb36fae4cb3d0ec521b8,5,1
44	null	c6512617863607d056e67569166491c5df00372cb164d6d1b0d2755ee10bcbad,0.0045,0
45	null	2cc2d0f646ff861b7b09e61ea24251fbfa5ca538272fb09b354f6dee0ef7c064,4.9015,0

Next, the first join should be done. At first mapper we have to open the previous results (smaller vout) as a local file and then put the hash and value in a Dictionary (because we just need these two attributes). As we see we have a null column so they have to be replaced by space. At the second mapper we are doing the join and At reducer we have to mind the order of running the mappers (UsingMRStep).



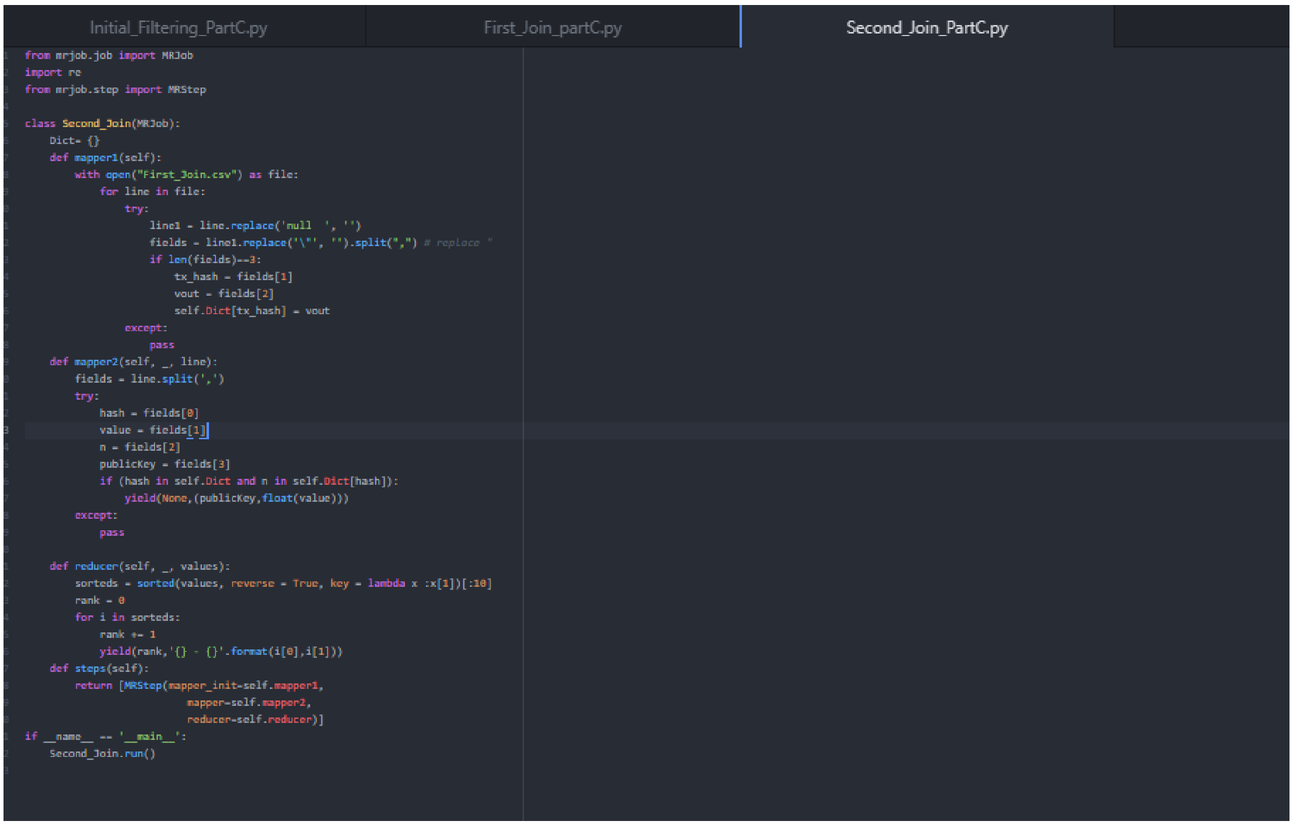
```python
from mrjob.job import MRJob
import re
import operator
from mrjob.step import MRStep

class First_join(MRJob):
    Dict= {}
    def mapper1(self):
        with open("Initial_Filtering.csv") as file:
            for line in file:
                line1 = line.replace('null  ', '')
                fields = line1.replace('\"', '').split(",")
                hash= fields[0]
                value= fields[1]
                publicKey= fields[2]
                self.Dict[hash]= value



    def mapper2(self, _, line):
        fields= line.split(",")
        try:
            if len(fields) == 3:
                txid = fields[0]
                tx_hash = fields[1]
                vout = fields[2]
                if txid in self.Dict: #Joining
                    yield(None,'{},{},{}'.format(txid,tx_hash,vout))
        except:
            pass

    def steps(self):
        return [MRStep(mapper_init= self.mapper1, mapper= self.mapper2)]

if __name__ == '__main__':
    First_join.run()
```

The results of first Join:



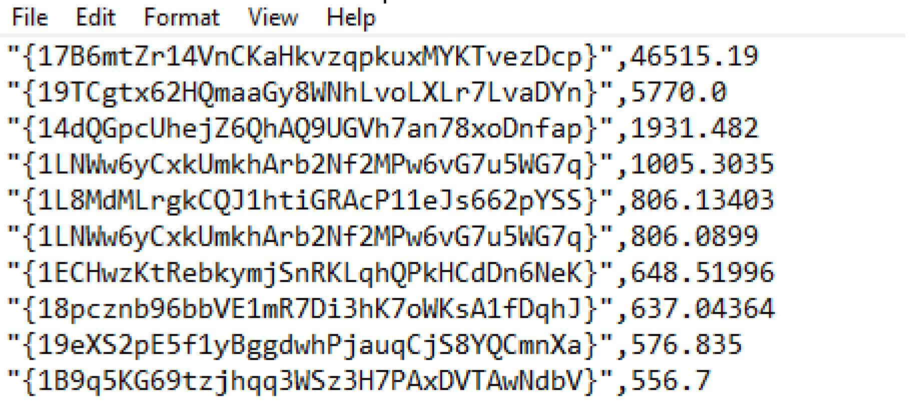
	A	B
1	null	ee2222caa459b6197f4496357270cdbc7e59e95e0c6c46be0d8e31c52d386634,68475a66a11ea5ef08a484460fdcf7ca5a4a46fa75ae927a48a2da4ba5c47fcb,1
2	null	39d09797204e71a42697439d4870ea9f1726637a5024a1d36aed7dc5d2bf29c3,aaed17b9f1d6ad8091afbed8cbd2e2bd015adf9da9fd1f357dfcf1de1e3325ee,0
3	null	317c3fa4d12478145cc857d5a9bb404fb42ecbefce91c7885435dda4bd85ad41,1de888bda7cb5b093363dde095b9a1b1e33e02efd5128bddf3df24203462790e,0
4	null	317c3fa4d12478145cc857d5a9bb404fb42ecbefce91c7885435dda4bd85ad41,29c59508b99038e295e3a19608f3e6b137a1d04bf15e0d1fa904ae3e0585985d,1
5	null	78e22aafa7bba380fd7fa0e480f5ff0fca9181ef6d23cea4c65bfbf399d87624,5e5de1a2300f2bcca96ad1d7c310012bd512d000e29ef7182e3e29f95db91619,0
6	null	7694e78dd505dc11ffe7738dfea45d152ba9c715aeea485873b06f7d01793fd0,7a48e690d7ff112b36c8ad3271a8bc8b315f636c31c25f278bf5e78989101102,3
7	null	7694e78dd505dc11ffe7738dfea45d152ba9c715aeea485873b06f7d01793fd0,af190077a7bce00ade696e69fd55980d082a4e7d53f9cfe37272367ebb0d59ec,0
8	null	7bd2f91a75f5c87e805d884b7c532b05690d21a537b81cf3ef2b39de8e1d4b7f,7fb23f887f9a4e66b6c28641e6db7d76d69597fcbcf485adb947f3b42f714cd2,1
9	null	8f07f0943a583d2bcdb7ae54c896f6970328356e2f3fc9bf72cfe6155d2bba40,1f8e283bbea4740af28989141bbcbf058f3336c6ed09727d3d0a8795724e8f43,0
10	null	7af0cd3f0b081e7f3569187c21cdec7b8395c5449014d24da0e2b36cedb02ef9,9d7ce3e5e2ca286e180c1c8b2192b3126c41990bf06acc134094d67810ef469e,0
11	null	ea08d64e29b06d5e7618d6834101d18583e73c1129364115d75cded20b56857f,c85cb347bc0a205a54bf3794f771bfe7b5a809bbd116b5d5315bcfa6543a15c6,3
12	null	a154ffa317b7e7fc164ed71eef95ac3367865be10156e6759b92094308d093d4,df94fa1d587cecc332cef76a502cd4ed7fa7ed6113d203c99880e6caf2235c76,0
13	null	72c2dee9345037caff2a06930ba7ce2db5ec1397b7b78d77d880a317689649aa,741f6cfd1f925c66386841bb7bef730d1c7c9859df625c0a4c388403f185c37d,1
14	null	872e6bea20de116a9d09dc8760aaa67b79fdb1ba52172ec13017c7dd617e4835,f3be7a6e41a00d0f9cf38976b63744f29ccf4fe1bc4d2d043c60b240b34d75c6,20
15	null	fa0220c475fea0e12292fbcf712817f735eab514a0e28df8b3ed6a0211058683,3f2eb385f04c1eb91b8b2db8025152fbb47170456bfe39b3d440b26730de063d,1
16	null	fa0220c475fea0e12292fbcf712817f735eab514a0e28df8b3ed6a0211058683,18cd4d1995371b406847b9be4740adf03f4abe88a78c86ffe4551d5fb2ed3694,0
17	null	fa0220c475fea0e12292fbcf712817f735eab514a0e28df8b3ed6a0211058683,648fbf785d4bad595b378bcf6771e761758a37cc7d0c0888dce52ceac83fe0bc,1
18	null	339badbcbd4e700aedf9518ad0bdc9368c37ee5975d100783a9710f9185d0f68,e5791f64dfeb44ee33b5031ebf62028900d9f888e99a9611481ab56d0ad7f699,38
19	null	622a16cc2a1c74e5b741b3c742f2a73a317792c9d176e6029691245a96174009,7ffb20a3453f8b4e2dc2a2c88b8d0633a4e087af87c2f054cd746ceb1e320175,0
20	null	6b79b16d45ccc7591fed0fbb3b906bfd40599dfd4836fbf1fbfaaada6d0209e3,027f6ea34224a7fad1bcdcd33b99c938f432cb2cc851f8344fbda9ecbd8f5e1e,1
21	null	9bdfeffc4f8344dcf3cf378a27e557a179d3036a95ea6cb69c5290d1e42744c3,718ff6597d0adac2f93aa696b4575dca5546498d69015134ce952b38fd249213,1
22	null	5bb655998675a3d8266caaf612930975f04a9f3ab611d61369212f17356f02b9,455dc81cf7e036c6e77a496351f5352b9c30cc1bc6f9cb59a86331a8b3c971c9,0
23	null	d27e79184b768f67a702553bb8ac10f5721617382beac2de10e2964e03d9ae2f,ad77a7cc3f0af37c27d24677b2631ec21362e7de0be8c66a0d923381faba9569,0
24	null	b9a266a3c3e5499c3e90f32e5fcb9bcaac2241410d287445df639dab4e588156,5b6d32593d85783f435f46cc39ccad35f2253469a317f30ecd0abe23e866714e,0
25	null	93c91ac1f0e6a804f1f74942a3c592fc46c51ca943dcdf4f1660f5fefaa14a61,e3309ae2740040fc1553d2fcd9393b5aa861e7a06a74d26b877b78431272918d,0
26	null	399adec8c2eb89605dc0db01e23e0d4bc15298337fe1f0025c265fc9769efc28,b12113ad02120afd0f2059340b4d111fce890fdf3f1caf1869ad0379b8708abd,0
27	null	a9ad4fb85f7822635357c51c0a69bab185c24086d43ea8659c9473e18fb8cc5d,aa7eaf2b5a3b2733744e81fa8942cd61fcea4358b1b191abd4995ec679391d1d,1
28	null	a9ad4fb85f7822635357c51c0a69bab185c24086d43ea8659c9473e18fb8cc5d,577a9f45e925e97ec36646662a79a63e5620cfbec5e21a04588f91b25722e49c,0
29	null	a9ad4fb85f7822635357c51c0a69bab185c24086d43ea8659c9473e18fb8cc5d,f735e448675336fb434f2fc1c425c35038843cdb037bb8a6c3de9bef46f53fab,1
30	null	c97ab84a0d9d97af92800fbfdcf403f975240b9cb5b8b730d5f95d56bb99e668,d19b8a22937f39f88c898e1971a5bf7253c9a84a93412f9c255efb95c2f609ca,8
31	null	ad9474ca48d33fc9e03dcd17f3d6eb1e98bd252e8905eee0e306db9bb9da1a1b,49a2987dfd4428ff4582537e09a9ab6ff6c9e95e385b151e0f16089685e542b7,1
32	null	9f0f45f2152107c9de6e646cc658ef3d45b34f5661b69952ba35076c49bc716a,6190cae999bb7a5b8c46e50da22273a3c49fbbf40c41c9db8e8c1e9f3db3252d,0
33	null	9f0f45f2152107c9de6e646cc658ef3d45b34f5661b69952ba35076c49bc716a,6190cae999bb7a5b8c46e50da22273a3c49fbbf40c41c9db8e8c1e9f3db3252d,1
34	null	053267747ab3a15f38d15a71f03f51dc5b34e65612f65f816bb0794de97c5618,917438e3a0282ad4edc5fc7d2d9c0c0895eac1e02d94ec45ac56db3acb603581,1
35	null	877315c8e0bba7a9a5535b0003b3e835e1e0da79a7cc1d07d0a41f233c3f27f7,bcd7a291d5f7a179052bf0f84fbe8c1a7048e357bd61b3623a5bc3e2ea16c25a,1
36	null	9a69bf0996353dfb7464566559e5fcb1bbb603c38760142fc80b20e99d9267ea,f5b749dc7f03eb4faa273da393817d01fb4ffdce905da24d1836f443d871ba4b,0
37	null	9a69bf0996353dfb7464566559e5fcb1bbb603c38760142fc80b20e99d9267ea,ba2b30eb670284b6033673345f077ea99edd3908581034b8f8d07e11b30b20ca,0
38	null	9a69bf0996353dfb7464566559e5fcb1bbb603c38760142fc80b20e99d9267ea,d59144fc085d739e0be3c829c963172d82fcd18f6911182d34458c245493dfd4,1
39	null	6a9ff9a8837d611d5ab786961fc9c89e7a368602db3c8a76d1905f8931dbc32a,3d4abfcb454300b443f42e0c2b39cfe319f39ba0625ff9a4d2368ee072c4ed41,0
40	null	68bd4ad796d135417a29875f5426d61808279e2c76363fb75aa538ab590256de,7b67daf2fbe9fbf96450bf7a1abceae0dcb189a97a06c225205742cc9d99ca4b,51
41	null	60b221a095dc2e47cccd7a6942c2df248fe3cf9db60b977f77cf41a7044182d2,9c941876ae2a28fea2cbf991303799c76e451e0af099890e79cd13301ff386a0,0
42	null	db9ad55e0cc9f1dadb66ca64060a3c4f8117e149da5b00742465a074d2d40446,9674ddb56ce4338eb23f64c763072fb9b2ecd105706c20c8fe734b7569e7e516,0
43	null	fb004152ccc1c33c0125f39b0f17ffcf987725c8e672dd5ea9d42bae6d4a7cf0,fab25583eed9e63c916c788b532b0735c40922b3952913b2c81a417bcdc5662e,0
44	null	ad2cd6eb3157fb8f65fd71fed0c53f5c306a00bafe35341fef839d96b8f424a7,6acd8423cca88b9e2e0e2053d99b305cdcfc6df398468c0d49c6d2e216889a91,1
45	null	88e244cb92f7b8f9f3a5b5e591bb611a5a47dce1a5e53496a29c1fddcd31311a,3c658225f812ea81acf1a7ae07bd7f7aafabbbac11ab267acece7ec98f34c357,0

At the final stage we need to re-join our these results to vout for checking their destination. Like the previous stage first get the tx_hash and vout and then change the criteria of our join:



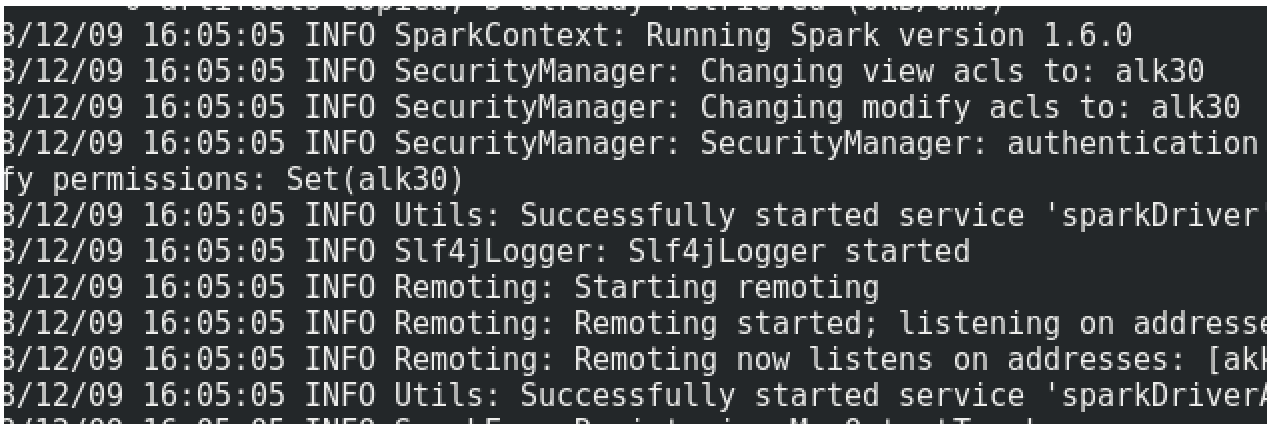
```python
from mrjob.job import MRJob
import re
from mrjob.step import MRStep

class Second_Join(MRJob):
    Dict= {}
    def mapper1(self):
        with open("First_Join.csv") as file:
            for line in file:
                try:
                    line1 = line.replace('null  ', '')
                    fields = line1.replace('\"', '').split(",") # replace "
                    if len(fields)==3:
                        tx_hash = fields[1]
                        vout = fields[2]
                        self.Dict[tx_hash] = vout
                except:
                    pass
    def mapper2(self, _, line):
        fields = line.split(',')
        try:
            hash = fields[0]
            value = fields[1]
            n = fields[2]
            publicKey = fields[3]
            if (hash in self.Dict and n in self.Dict[hash]):
                yield(None,(publicKey,float(value)))
        except:
            pass

    def reducer(self, _, values):
        sorteds = sorted(values, reverse = True, key = lambda x :x[1])[:10]
        rank = 0
        for i in sorteds:
            rank += 1
            yield(rank,'{} - {}'.format(i[0],i[1]))
    def steps(self):
        return [MRStep(mapper_init=self.mapper1,
                       mapper=self.mapper2,
                       reducer=self.reducer)]
if __name__ == '__main__':
    Second_Join.run()
```

The Results was the same as Spark:

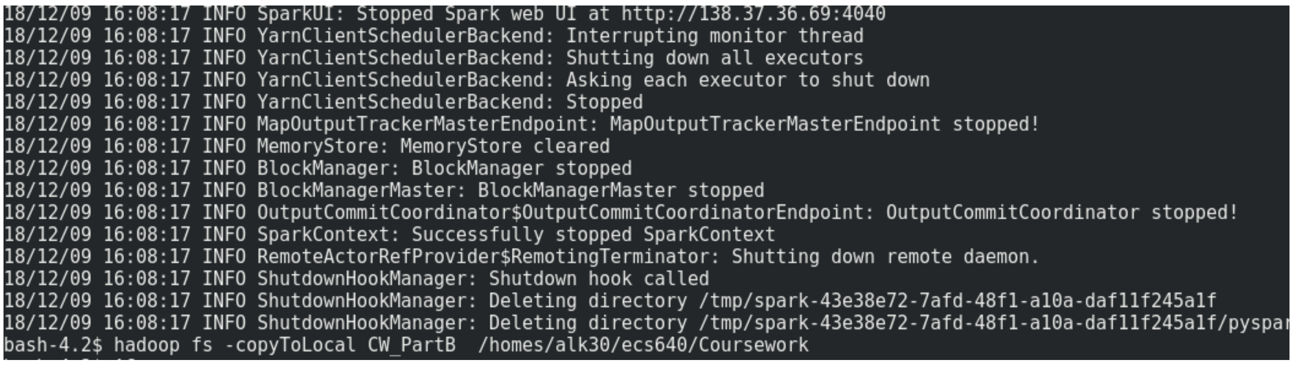


```
"{17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp}",46515.19
"{19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn}",5770.0
"{14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap}",1931.482
"{1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q}",1005.3035
"{1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS}",806.13403
"{1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q}",806.0899
"{1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK}",648.51996
"{18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ}",637.04364
"{19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa}",576.835
"{1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV}",556.7
```

With spark we could do the tasks in 3minutes and 12 seconds (16:08:17 – 16:05:02)



```
3/12/09 16:05:05 INFO SparkContext: Running Spark version 1.6.0
3/12/09 16:05:05 INFO SecurityManager: Changing view acls to: alk30
3/12/09 16:05:05 INFO SecurityManager: Changing modify acls to: alk30
3/12/09 16:05:05 INFO SecurityManager: SecurityManager: authentication
fy permissions: Set(alk30)
3/12/09 16:05:05 INFO Utils: Successfully started service 'sparkDriver'
3/12/09 16:05:05 INFO Slf4jLogger: Slf4jLogger started
3/12/09 16:05:05 INFO Remoting: Starting remoting
3/12/09 16:05:05 INFO Remoting: Remoting started; listening on addresse
3/12/09 16:05:05 INFO Remoting: Remoting now listens on addresses: [akk
3/12/09 16:05:05 INFO Utils: Successfully started service 'sparkDriverA
```



```
18/12/09 16:08:17 INFO SparkUI: Stopped Spark web UI at http://138.37.36.69:4040
18/12/09 16:08:17 INFO YarnClientSchedulerBackend: Interrupting monitor thread
18/12/09 16:08:17 INFO YarnClientSchedulerBackend: Shutting down all executors
18/12/09 16:08:17 INFO YarnClientSchedulerBackend: Asking each executor to shut down
18/12/09 16:08:17 INFO YarnClientSchedulerBackend: Stopped
18/12/09 16:08:17 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
18/12/09 16:08:17 INFO MemoryStore: MemoryStore cleared
18/12/09 16:08:17 INFO BlockManager: BlockManager stopped
18/12/09 16:08:17 INFO BlockManagerMaster: BlockManagerMaster stopped
18/12/09 16:08:17 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
18/12/09 16:08:17 INFO SparkContext: Successfully stopped SparkContext
18/12/09 16:08:17 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
18/12/09 16:08:17 INFO ShutdownHookManager: Shutdown hook called
18/12/09 16:08:17 INFO ShutdownHookManager: Deleting directory /tmp/spark-43e38e72-7afd-48f1-a10a-daf11f245a1f
18/12/09 16:08:17 INFO ShutdownHookManager: Deleting directory /tmp/spark-43e38e72-7afd-48f1-a10a-daf11f245a1f/pyspar
bash-4.2$ hadoop fs -copyToLocal CW_PartB  /homes/alk30/ecs640/Coursework
```

But with MRJob it takes us 2 minute and 45 second for initial filtering, 3minutes for first join and 5 minutes for second join.
So due to the fact that Spark is faster than Map/Reduce for this task, to my knowledge doing this task with Spark is better.