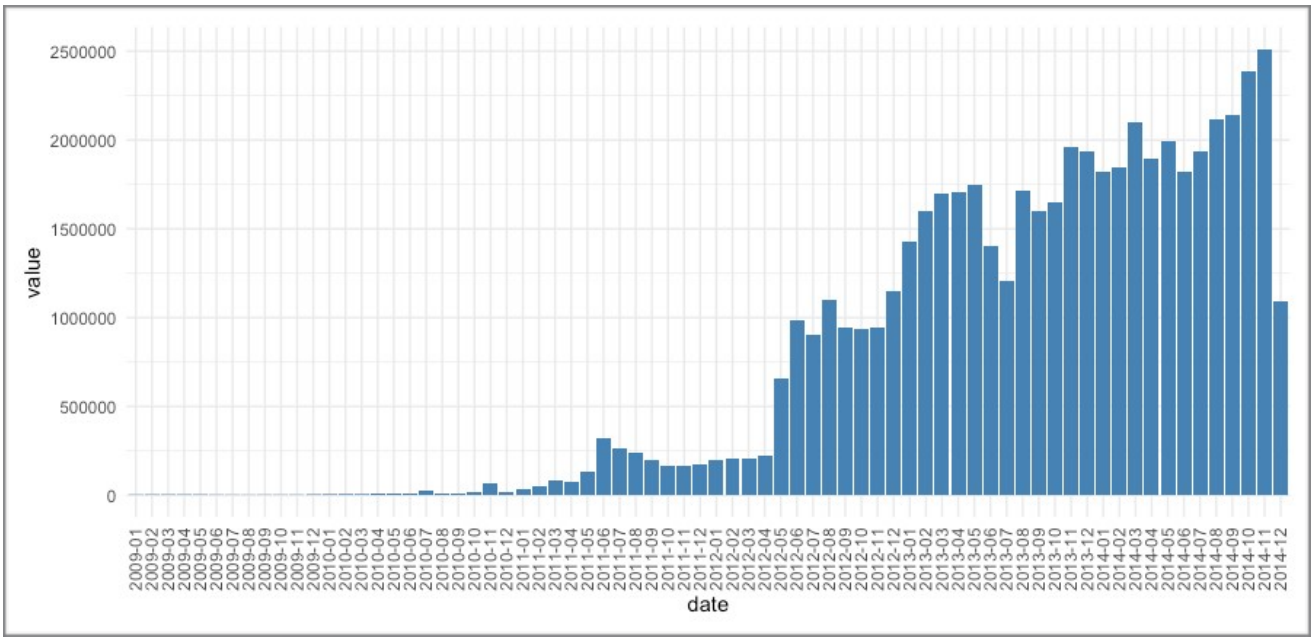
[submitting by 3rd of Dec]

PART A. TIME ANALYSIS

Create a bar plot showing the number of transactions which occurred every month between the start and end of the dataset. What do you notice about the overall trend in the utilisation of bitcoin?



The bar chart shows the number of bitcoin transactions each month from 2009 to 2014. At the beginning of 2009, the number of transactions was close to zero per month and remained relatively stable over the following months. However, the number of transactions rose considerably between 2012-April and 2014-November, reaching a peak of approximately 2,500,000 per month. Nonetheless, the number of transactions dropped dramatically in December-2014.

Code

```
"""
Create a bar plot showing the number of transactions which occurred every
month
"""
#Transactions.csv
#[0]tx_hash, [1]blockhash, [2]time, [3]tx_in_count, [4]tx_out_count

from mrjob.job import MRJob
import re
import time


#This line declares the class Lab1, that extends the MRJob format.
class partA(MRJob):
# this class will define two additional methods: the mapper method goes here
    def mapper(self, _, line):
        fields = line.split(',')
        try:
            if (len(fields)==5):
                time_epoch = int(fields[2])
                date = time.strftime("%Y-%m",time.gmtime(time_epoch))
                yield (date, 1)
        except:
            pass

    def combiner(self, date, counts):
        yield (date, sum(counts))

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

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

PART B. TOP TEN DONORS

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?

Top	Donor	BTC	GBP
1	{17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp}	46,515.189	9,596,083.590
2	{19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn}	5,770.000	1,190,351.000
3	{14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap}	1,931.482	398,464.737
4	{1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q}	1,848.738	381,394.609
5	{1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS}	806.134	166,305.450
6	{1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK}	648.520	133,789.672
7	{18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ}	637.044	131,422.106
8	{19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa}	576.835	119,001.061
9	{1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV}	556.700	114,847.210
10	{1AUGSxE5e8yPPLGd7BM2aUxfzbokT6ZYSq}	500.000	103,150.000

Note. The last column (GBP) is multiplied BTC by 206.3 (exchange rate of 31/12/14)

Code

Filter

```
from mrjob.job import MRJob
import re



class partBfilter(MRJob):

    def mapper(self, _, line):
        fields = line.split(",")
        hash = fields[0]
        publicKey = fields[3].strip()
        if (publicKey =="{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}"):
            yield(hash,publicKey)

if __name__ == '__main__':
    #partBfilter.JOBCONF = {'mapreduce.reduce.memory.mb': 4096 }
    partBfilter.run()
```

First join

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

class partBfirstjoin(MRJob):

    def steps(self):
           return [MRStep(mapper_init=self.mapper_join_init,
                          mapper=self.mapper_repl_join)]

    sector_table = {}
    def mapper_join_init(self):
        # load vout into a dictionary
        with open("out_partBfilter.txt") as f:
            for line in f:
                fields = line.split("\t")
                hash = fields[0].replace('"','')
                self.sector_table[hash] = '' #[value,n]

    def mapper_repl_join(self, _, line):
        fields = line.split(",")
        if len(fields) == 3:
            txid = fields[0]
            tx_hash = fields[1]
            vout = fields[2]
            if txid in self.sector_table:
                yield(tx_hash,vout)


if __name__ == '__main__':
    partBfirstjoin.JOBCONF = {'mapreduce.reduce.memory.mb': 4096 }
    partBfirstjoin.run()
```

Second join

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

import collections

def total(records):
    dct = collections.defaultdict(int)
    for cust_id, contrib in records:
        dct[cust_id] += contrib

    return dct.items()


class partB_secondjoin(MRJob):

    def steps(self):
           return [MRStep(mapper_init=self.mapper_join_init,
                          mapper=self.mapper_repl_join,
                          reducer=self.reducer_sum),
                   MRStep(mapper = self.mapper,
                          reducer=self.reducer_top)]

    sector_table = {}

    def mapper_join_init(self):
        with open("out_partBfirstjoin.txt") as f:
            for line in f:
                fields = line.split("\t")
                tx_hash = fields[0].replace('"','')
                vout = fields[1].replace('"','').strip()
                self.sector_table[(tx_hash,vout)] = ''


    def mapper_repl_join(self, _, line):
        fields = line.split(",")
        hash = fields[0]
        value = fields[1]
        n = fields[2]
        publicKey = fields[3]#.strip()
        if (hash,n) in self.sector_table:
            yield(publicKey,float(value))

    def reducer_sum(self, publicKey, value):
        yield('',(publicKey,sum(value)))

    def mapper(self, feature, value):
        yield('',value)

    def reducer_top(self,feature,pair):
        sorted_values = sorted(pair, reverse = True, key = lambda
pair:pair[1])[:10]
        for value in sorted_values:
             yield (value[0],value[1])


if __name__ == '__main__':
    partB_secondjoin.JOBCONF = {'mapreduce.reduce.memory.mb': 4096 }
    partB_secondjoin.run()
```

PART C. DATA EXPLORATION

3. 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?

Item	MapReduce	Spark
Execution Times	17 min 21 sec	38 min 1 sec
number of file	3	1
process of job	**1. Initial Filtering: filter out rows by wallet**	
	1 mapper	1 filter, 1 map, 1 broadcast
	2.First Join: filter out rows by transaction(tx_id), get tx_hash & vout	
	1 Dictionary, 1 mapper	1 filter, 1 map, 1 broadcast
	3. Second Join&Sort: Get Top10 Donor	
	1 Dictionary, 2 mapper, 2 reducer	1 filter, 1 map, 1 broadcast, 1 reduceByKey, 1 takeOrdered
number of transformation		7 transformation (3 map, 3 filter, 1 reduceByKey)

	MapReduce	Spark
1	17m 21s	1h 16m 36s
2	17m 11s	47m 55s
3	17m 23s	19m 18s
4	17m 14s	17m 18s
5	17m 37s	29m
Avg	17 m 21s	38m 1s

According to the tables above, I find out implementing partB by MapReduce job is much stable and faster than by spark. The reason might be that the the system needs to create RDDs from HDFS files in Spark but not in MapReduce (MapReduce job could read files directly). However, using MapReduce is inconvenient, because I need to detect the process and manually trigger jobs after previous job is complete. Also, results will be saved to HDFS for every iterations. Code

```
# System Setting ==============================================================
isLocal = False
if isLocal:
    dir = “/coursework"
    vout_path = dir + "/input/voutSample.csv"
    vin_path = dir + "/input/vinSample.csv"
else:
    dir = "hdfs://studoop.eecs.qmul.ac.uk/data/bitcoin"
    vout_path = dir + "/vout.csv"
    vin_path = dir + "/vin.csv"




# Start python ================================================================
import pyspark
import re

sc = pyspark.SparkContext()

# Define functions that will be used-----------------------------------------
def is_in_wallet(line):
    fields = line.split(',')
    publicKey = fields[3].strip()
    wallet = "{1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v}"
    if (publicKey == wallet):
        return True
    else:
        return False

def is_receiver_trans(line):
    tx_id = line.split(',')[0]
    if tx_id in filtered_hash.value :
        return True
    else:
        return False
    filtered_hash.unpersist()

def is_sender_trans(line):
    fields = line.split(',')
    hash = fields[0]
    n = fields[2]
    if (hash,n) in first_join.value :
        return True
    else:
        return False
    first_join.unpersist()



# Initial Filtering: filter rows by wallet, get associated transaction ID----
linesVout = sc.textFile(vout_path) \
              .filter(is_in_wallet)
filtered_hash = linesVout.map(lambda line: line.split(',')[0]).collect()
filtered_hash = sc.broadcast(filtered_hash)

del linesVout
```

```
# First Join: filter rows by transaction(tx_id),get tx_hash & vout ----------
linesVin = sc.textFile(vin_path) \
             .filter(is_receiver_trans)
first_join = linesVin.map(lambda l: (l.split(',')[1],
l.split(',')[2])).collect()
first_join = sc.broadcast(first_join)

del linesVin


# Second Join: filter rows by tx_hash & vout,get (wallet, value) ------------
linesVout = sc.textFile(vout_path) \
              .filter(is_sender_trans)
features = linesVout.map(lambda l:
(l.split(',')[3].strip(),float(l.split(',')[1])))

del linesVout


# Top ten: sort by value(money), get top 10 wallet and value -----------------
top10 = features.reduceByKey(lambda a, b: a + b).takeOrdered(10, key = lambda
x: -x[1])
for record in top10:
    print((record[0],record[1]))
```

5. Find a dataset containing the prices of bitcoin throughout time (Many sites exist with such information). Utilising the Spark MlLib library, combine this, characteristics extracted from the data (volume of bitcoins sold per day, transactions per pay), and any other external sources into a model for price forecasting. How far into the future from our subset end date (September 2014) is your model accurate? How does the volatility of bitcoin prices affect the effectiveness of your model?

Date	Price	total_trade_btc	number_of_transactions
2014-03-04	662.179993	1026802.034	272021
2014-05-16	453.630005	651741.425	193788
2014-11-21	357.299988	1319260.313	237094
2010-11-10	0.21	68695.560	508
2011-07-02	15.4	874963.593	33491

In this part, I use two python code.
[*preprocessing.py]* I summarize price, total_trade_bitcoin and number of transactions each date (see table above) by joining three different tables.

[prediction.py] I used linear regression to predict bitcoin price.
My model: price(day) = -1.435 total_trade_btc + 0.0023*number_of_transactions - 35.71
RMSE on Test Data = 153.812
R_2 = 0.572775

Preprocessing

```
import pyspark
from pyspark import SparkConf, SparkContext
from pyspark.ml.linalg import Vectors

def to_array(col):
    def to_array_(v):
        return v.toArray().tolist()
    return udf(to_array_, ArrayType(DoubleType()))(col)


# System Setting ====================================================
isLocal = True
if isLocal:
    dir =
"/Users/maoweng17/Documents/QMUL/BigDataProcessing/Lab/coursework/partC"
    pre_path = dir + "/input/out_preprocessing.csv"

from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext

# Load Data ---------------------------------------------------------
sc= SparkContext()
sqlContext = SQLContext(sc)
df = sqlContext.read.format('com.databricks.spark.csv') \
                    .options(header='false', inferschema='true') \
                    .load(pre_path) \
                    .toDF('time','price','total_trade_btc','n_trans')

# Transform into Vector for MlLib -----------------------------------
from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols =
['total_trade_btc','n_trans'], outputCol = 'features')
df = vectorAssembler.transform(df) \
                    .select(['features', 'price'])

# Split into Train and Test dataset ---------------------------------
splits = df.randomSplit([0.7, 0.3])
train_df = splits[0]
test_df = splits[1]

# Build LinearRegression Model--------------------------------------
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol = 'features', labelCol='price')
lr_model = lr.fit(train_df)
trainingSummary = lr_model.summary

# Check Result and Test Accuracy------------------------------------
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)

test_result = lr_model.evaluate(test_df)
print("RMSE on test data = %g" % test_result.rootMeanSquaredError)

# Get Prediction value of whole dataset -----------------------------
lr_predictions = lr_model.transform(df) \
                         .select('prediction','price','features')
```