Report

2.1 First RDD

```
l = list(range(0,3000))
rdd = sc.parallelize(l)
```

2.2 Computing the sum of cubes

```
C = rdd.map(lambda a : a*a*a)
C.sum()
```

First, we use map() in order to operate on every element in this RDD. Then use sum() to get the sum of cubes.

2.3 Last digits of elements in C

```
lastDig = C.map(lambda v : (v%10, 1))
totalLastDig = lastDig.reduceByKey(lambda a,b : a+b).sortByKey()
totalLastDig.collect()
```

Map: to get the last digit of every elements and store them in lastDig. The key of lastDig is 0 to 9, and the value is 1.
ReduceByKey: to count how many times every digit appears.
SortByKey: to make the key in an ascending sort order.
Collect: to output the RDD as a list.

2.4 Digits of C

```
def digits(i):
  return [(e,1) for e in str(i)]

dig = C.flatMap(lambda v: digits(v))
digTotal = dig.reduceByKey(lambda a,b : a+b).sortByKey()
digTotal.collect()
```

First, I define a new function, the input parameter is a number, and it will return a map in which the keys are the digits of the number and the value is always 1.
I use flatMap in order to separate every digit in a number into an independent part.
Then we give the same operations as above.

3.1 Step 1 : computing set of pairs

```
pairs = rdd.cartesian(rdd)
```

3.2 Step 2 : counting the pairs

```
filterPairs = pairs.filter(lambda p :(2*p[0]+1)*(2*p[0]+1)+(2*p[1]+1)*(2*p[1]+1) < (2*3000)*(2*3000)).count()
```

I use the function filter to select the eligible elements and the function count to count the number of elements after filtration.

3.3 Computing the approximation

```
print(filterPairs/(pairs.count())*4)
```

Just to check whether the result is π or not.

4.2 Getting the dataset into an RDD

```
import re
future_pattern = re.compile("""([^,"]+|"[^"]+")(?=,|$)""")
def parseCSV(line):
  return future_pattern.findall(line)
path_data = "/FileStore/tables/"
ratingsFile = sc.textFile(path_data+"/ratings.csv").map(parseCSV)
moviesFile = sc.textFile(path_data+"/movies.csv").map(parseCSV)
```

We import the two csv files into the datasets and get their locations. Then make them into RDD according to instruction.

4.3 Cleaning data

```
rating = ratingsFile.filter(lambda r : r[0] != 'userId')
r = rating.map(lambda r : (r[0],r[1],float(r[2]),r[3]))
movie = moviesFile.filter(lambda m : m[0] != 'movieId')
```

Use the function filter to take out the first line and use the function map to change the third column into float. Here we must pay attention that, we also need to write down r[0], r[1], r[3] even if we don't operate on them.

4.4 10 best movies of all times

```
best10List = r.map(lambda r : (r[0], r[2])).mapValues(lambda f : (f,1)).reduceByKey(lambda a,b : (a[0]+b[0], a[1]+b[1])).mapValues(lambda x : (x[0]/x[1])).sortBy((lambda x : x[1]), False).take(10)
```

Here I use mapValues and reduceByKey to calculate the total ratings and count the

number of ratings respectively. The parameter “False” in the function sortBy means a descending order.

4.5 Ordered list of movies with names

```
best10Rdd = sc.parallelize(best10List)
best10Rdd.join(movie).values().collect()
```

The function join makes the two RDDs into one. The same column in the two RDDs is the movieId.

4.6 Better ordered list

```
grade1 = r.map(lambda r : (r[0], r[2])).mapValues(lambda f : (f,1)).reduceByKey(lambda a,b : (a[0]+b[0], a[1]+b[1])).mapValues(lambda x : (x[0]/x[1]*(x[1]/(x[1]+1)))).sortBy((lambda x : x[1]), False).take(10)
best10_1 = sc.parallelize(grade1)
best10_1.join(movie).values().collect()

import math
grade2 = r.map(lambda r : (r[0], r[2])).mapValues(lambda f : (f,1)).reduceByKey(lambda a,b : (a[0]+b[0], a[1]+b[1])).mapValues(lambda x : (x[0]/x[1]*(math.log(x[1])))).sortBy((lambda x : x[1]), False).take(10)
best10_2 = sc.parallelize(grade2)
best10_2.join(movie).values().collect()
```

Almost the same as 4.4, just change the formula of calculating.