

# TDT4305 - Project part 1

Henrik Kjærnsli & Patrik Kjærran - MTIØT  
TDT4305 - Big Data Architecture

February 29, 2020

## 1 Introduction

Project part 1 in TDT4305 revolves around analysis of a given dataset consisting of three different files. We have chosen to use Python's API for Spark, called *pyspark*, to solve the given tasks. Tasks 1 - 4 asks us to load the dataset into three RDDs and use RDD functionalities to solve the exercises. Task 5 and 6 provides us with the challenge of loading the dataset into three DataFrames, and use Spark SQL API to implement solutions. In the following sections we will describe our design and approach to the exercise as well as the results we obtained.

## 2 Deliverables

In addition to the report, the .zip file contains the source codes and .csv files with the output of our solutions. We have made one csv-file for each subtask, although task 5a and 6b are delivered as .txt-files because of its format.

## 3 Setup

### Install

This project requires Python 3.7 or above, along with a working installation of PySpark. For detailed installation instructions, see "*spark\_install.pdf*" from course material on BlackBoard.

In terminal/command prompt, navigate to the project root directory and run:

```
pip install -r requirements.txt
```

**Hadoop (Windows only):** In order for file export to function correctly, precompiled Hadoop binaries is required ([link](#)).

## Run

The script can be run from within the project root directory using

```
python -m tdt4305 [args ... ]
```

or

```
python main.py [args ... ]
```

## Optional arguments:

**--action:** Denotes the action to perform for each task.

**run** (default): Prints results to console.

**export-csv:** Exports output as *.csv* files to project *output* directory.

**export-tsv:** Exports output as *.tsv* files to project *output* directory.

**export-txt:** Redirects console output to *output.txt* within project *output* directory.

**--tasks:** Specifies which tasks to execute (default: 1-6).

## Example:

```
python -m tdt4305 --action export-csv --tasks 1,3,5-6
```

## Input:

The script scans for the following files within the *data* directory:

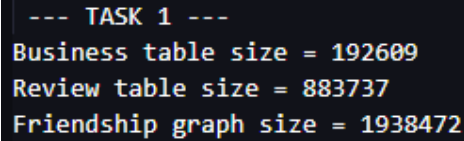
- *yelp\_businesses.csv*
- *yelp\_top\_reviewers\_with\_reviews.csv*
- *yelp\_top\_users\_friendship\_graph.csv*

Make sure to place the mentioned files into the data directory before running the code.

## 4 Tasks

### Task 1

To solve the RDD-tasks, we load the data files into separate RDDs using `SparkContext.textFile()`. We then remove the headers so it does not count toward any of the following results. The function `rdd.count()` is applied to each RDD to count the number of rows.



```
--- TASK 1 ---  
Business table size = 192609  
Review table size = 883737  
Friendship graph size = 1938472
```

Figure 1: Output from task 1.

### Task 2

In task 2 we do queries against the review-table to find our solutions.

- a) We first map the RDD to only contain the user IDs. We then use the functions `rdd.distinct()` before `rdd.count()` to find the number of distinct users.
- b) To find the average number of characters in a user review, we first map the RDD to only contain review lengths, before finding the average by summing the lengths and divide it by the number of reviews.
- c) We find the business ID of the top 10 businesses with the most number of reviews by grouping the RDD on business ID. We then map values to the length by using `rdd.mapValues(len)` before a sorting is performed. Finally we map the RDD to only contain business ID and retrieve the top 10 IDs from the sorted RDD by using `rdd.take(10)`.
- d) By grouping the RDD on year, we can then use `rdd.mapValues(len)` to obtain reviews per year.
- e) We first map RDD to only contain the review date. Because `datetime.fromtimestamp()` takes an integer as input, we need to convert the value to an integer. We then obtain the first and last review date using `rdd.min()` and `rdd.max()`.
- f) The goal in this subtask is to calculate Pearson Correlation Coefficient (PCC) between the number of reviews by a user and the average number of characters in the user's review.

Looking at the provided formula, the PCC can be divided into four parts:

- Element-wise difference from average number of reviews ( $x\_diffs$ )
- Element-wise difference from average review length ( $y\_diffs$ )
- Sum of squared differences for number of reviews ( $sum\_sqdiff\_x$ )

- Sum of squared differences for review length (*sum\_sqdiff\_y*)

We first group reviews by *user\_id* using the `rdd.groupBy()`, followed by `rdd.mapValues(len)` to obtain number of reviews per user. To find the average review count, we sum the values of the reviews per user and divide it by the user's review count.

We then find the review length per review along with a counter variable, which are used to calculate the average lengths per user. Average review length is given by `review_lengths.sum() / review_length.count()`.

To obtain the numerator of the expression, we then create two RDD where the first maps the review count count per user to *review count per user - average review count*. The second RDD is mapped to *review length per review - average review length*. For each review count, the deviation from average is obtained by subtracting the average number of reviews from earlier from each entry. This is done using the `rdd.map()` function. We then use the `.zip`-function to map the two RDDs so that we complete the two first parts.

The denominator is calculated as the square root of the squared sum of the RDDs. This is the third and fourth part. We obtain these by mapping the RDDs from the first and second part so that the RDDs contains only the squared differences.

As this part is fairly complicated to explain, we have attached a screenshot of the code for this subtask below.

```
1 def task_2f(rt_rdd):
2     """
3     Calculates the 'Pearson correlation coefficient' between the number of reviews
4     by a user and the average number of the characters in the user's reviews.
5     """
6     # Find number of reviews per user
7     rt_grouped_by_user = rt_rdd.groupBy(lambda row: row[1])
8     rt_reviews_count_per_user = rt_grouped_by_user.mapValues(len)
9
10    # Find review length per review, along with a counter variable
11    rt_extracted_lengths = rt_rdd.map(lambda row: [row[1], (1, len(row[3]))])
12    rt_tuples_per_user = rt_extracted_lengths.reduceByKey(lambda v1, v2: (v1[0] +
13    ↪ v2[0], v1[1] + v2[1]))
14    rt_average_length_per_user = rt_tuples_per_user.mapValues(lambda value: value[1] /
15    ↪ value[0])
16
17    # Average review count
18    rt_review_counts = rt_reviews_count_per_user.values()
19    X = rt_review_counts.sum() / rt_review_counts.count()
20
21    # Average review length
22    rt_review_lengths = rt_average_length_per_user.values()
23    Y = rt_review_lengths.sum() / rt_review_lengths.count()
24
25    x_diff = rt_review_counts.map(lambda x_i: x_i - X)
26    y_diff = rt_review_lengths.map(lambda y_i: y_i - Y)
27    sum_sqdiff_x = x_diff.map(lambda x_diff: x_diff**2).sum()
28    sum_sqdiff_y = y_diff.map(lambda y_diff: y_diff**2).sum()
29    numerator = x_diff.zip(y_diff).map(lambda diff: diff[0] * diff[1]).sum()
30    denominator = sum_sqdiff_x**0.5 * sum_sqdiff_y**0.5
31    return numerator / denominator
```

```

--- TASK 2a ---
Number of unique users = 4521

--- TASK 2b ---
Average review length = 1144.1648476865855

--- TASK 2c ---
Top 10 businesses: [
  'POSorDeYuzB3tMwIXSny7w', 'MPGrHmDFzXCr-EtNSDh2g',
  'uQo2-hQ9stWvKf8IiyPX0Q', 'vK_UbrtYjWExtXgFC7Ko_w',
  'b5wTvBzQNMGG4z1_wsSYHA', 'pfKrs4p1bQ5LVD_tztPHCQ',
  'vZbfukwCTj1BANm6jcBZg', 'l_ImU9UjDbHGxy_3l7fyGw',
  'I8zmaiEUVrhi_BhiQw3q5A', 'y5Z44Sr0SsrL83fELcRL9A'
]

--- TASK 2d ---
Reviews per year = [
  (2004, 2), (2005, 215), (2006, 915),
  (2007, 6245), (2008, 17269), (2009, 24880),
  (2010, 49394), (2011, 73952), (2012, 83388),
  (2013, 98324), (2014, 100526), (2015, 109816),
  (2016, 114738), (2017, 114141), (2018, 89932)
]

--- TASK 2e ---
Time of first review = 2004-12-19 20:47:24
Time of last review = 2018-11-14 18:10:56

--- TASK 2f ---
PCC = 0.12598043062131659

```

Figure 2: Output from task 2.

### Task 3

In task 3 we do queries against the business-table to find our solutions.

- We use the following approach to get the average rating for businesses in each city: Map the RDD so that it becomes on the form (city, (1, stars)). We then uses `rdd.reduceByKey()` to sum the stars for each city, as well as the ones in the tuple, before we divide the star count by the number of the review in the city.
- The first part is to map the RDD so that it only contains categories. We then uses `rdd.flatMap()` to create a row for each category-entry. Finally we use `rdd.countByValue()` to get the count for each review category, before using `rdd.top(10)` to extract the 10 most frequent entries.
- To calculate the centroids, we need to sum the latitude and longitude for each postalcode and then divide it by the number of entries for the postalcode. We first map the RDD to have postal code as key, with latitude and longitude along with a counter variable as values. Afterwards, `rdd.reduceByKey()` is used to sum the counter variable, latitude and longitude separately for each postal code. To find the average centroid, we create tuples where we take the sum of the latitudes and divide it by the counter variable for each postal code. The same is done for the longitudes.

## Task 4

In task 4 we do queries against the friendship-graph to find our solutions.

- Highest degree source is found by first mapping the RDD to contain the source user IDs along with a counter variable. We then add the counter variables by using `rdd.reduceByKey(add)`, before using the `rdd.top(10)` function to return the top 10 nodes with the most out-degrees. Highest destination is found by exactly the same approach.
- We start by finding the source degrees for each source user ID by using the same approach as in 4a. The mean is then found by taking the mean of the values of the RDD by calling `rdd.mean()` on `rdd.values()`.

To find the median, we first sort the RDD by the counter variable, before we merge the RDD by the use of `rdd.zipWithIndex()`. To be able to use `rdd.lookup()`, we need to map the RDD so that the index is at index 1, while the counter variable is at index 0. We then perform a integer division to find the median. Exactly the same approach is used to find the mean and median for the destination degrees.

```
--- TASK 4a ---
Top ten input nodes = [
  ("8DEyKVyp1n0cSKx39vatbg", 4919), ("ZIOcmdFaMIF56FR-nWr_2A", 4597),
  ("YttDgOC9A1M4HcA1DsB82A", 4222), ("djxnI8Ux8ZVQJh10QkrRhA", 4211),
  ("F_5_UNX-wrAFCXuAk8ZRDw", 3943), ("dIIKEf0go8KqUfGQvGikPg", 3651),
  ("GGTF7hnQ16D5W77_q1k1qg", 3609), ("NFU8zDaTMEQ4-X9dbQmd9A", 3557),
  ("3gRfkaVCEwr1-Ju70QX7uQ", 3396), ("NhgU7RhuVfmpkb1j1YJ6Q", 3330)
]

Top ten output nodes = [
  ("ZIOcmdFaMIF56FR-nWr_2A", 9564), ("F_5_UNX-wrAFCXuAk8ZRDw", 8586),
  ("djxnI8Ux8ZVQJh10QkrRhA", 8381), ("YttDgOC9A1M4HcA1DsB82A", 6758),
  ("NFU8zDaTMEQ4-X9dbQmd9A", 6506), ("dIIKEf0go8KqUfGQvGikPg", 6187),
  ("ACUVZ4S1N0gn17dzV0m9EQ", 6065), ("8DEyKVyp1n0cSKx39vatbg", 6026),
  ("w-w-k-QXosIKQ8HQVwJ61Q", 5987), ("Thc2zV-K-KLcvJn3fMPdqQ", 5821)
]

--- TASK 4b ---
Mean input node degree = 12.060049149228657
Median input node degree = 1

Mean output node degree = 3.864161354248208
Median output node degree = 1
```

Figure 3: Output from task 4.

## Task 5

To solve task 5, we specify the value types of each column, such that the types are in line with the ones given in the assignment. This is done using `sql.types.StructType` and `sql.types.StructField`.

```

--- TASK 5a ---
(Business table schema)
root
|-- business_id: string (nullable = true)
|-- name: string (nullable = true)
|-- address: string (nullable = true)
|-- city: string (nullable = true)
|-- state: string (nullable = true)
|-- postal_code: string (nullable = true)
|-- latitude: float (nullable = true)
|-- longitude: float (nullable = true)
|-- stars: float (nullable = true)
|-- review_count: integer (nullable = true)
|-- categories: string (nullable = true)

(Review table schema)
root
|-- review_id: string (nullable = true)
|-- user_id: string (nullable = true)
|-- business_id: string (nullable = true)
|-- review_text: string (nullable = true)
|-- review_date: string (nullable = true)

(Friendship graph schema)
root
|-- src_user_id: string (nullable = true)
|-- dst_user_id: string (nullable = true)

```

Figure 4: Output from task 5.

## Task 6

- a) To perform an inner join on the review table and business table, we use `rdd.join()`. The join is performed on business ID.
- b) In task 6b we save the table from 6a in a temporary table. This is done by using `SqlContext.registerDataFrameAsTable()`.
- c) We first group the DataFrame by user ID before sorting on the number of instances in descending order. Afterwards, we find the top 20 rows.

```

--- TASK 6b ---
SqlContext.tables() before execution
+-----+-----+
|database|tableName|isTemporary|
+-----+-----+

SqlContext.tables() after execution
+-----+-----+
|database|tableName|isTemporary|
+-----+-----+
|         |temp_table  |true       |
+-----+-----+

--- TASK 6c ---
Top 20 users by review count = [
  ('CxD0IDnH8g9KXzpBH3YXw', 4129), ('bLbSNKl ggFnqW4Nzzq-Ijw', 2354),
  ('PKEzKwv_FktMwZmGPjwd8Q', 1822), ('ELcQDlf69kb-1h3fxZyL8A', 1764),
  ('DK57YibC5Sh8mqQl97CKog', 1727), ('U4INQZ0P5Uaj8hHjLlZ3KA', 1559),
  ('QI90SEn6ujRctrX86vslw', 1496), ('d_TB633tW9y96ChqUEXkg', 1360),
  ('hMybu_KyVLSdEFzGrniTu', 1355), ('cMEtAIm60ISwE_vLftxoJQ', 1255),
  ('YRcaNlwG6QXPFDXitUwGdA', 1224), ('62GNF5FySKa3MbrQmngv', 1199),
  ('dIIKEF0go8KqUFGyGikPg', 1198), ('UVcmcbelzRe8Q6JqzLogw', 1196),
  ('n86871kbU28kxLFX_5aew', 1152), ('rCwrxURCB_pFagncHtp6A', 1148),
  ('3nDUQ6JkyVor5wV8w-eJchg', 1106), ('IDlkZ02iLL58Jwfdy7DP9A', 1088),
  ('USYQX_vHl_xQy8EQq1NQ', 1024), ('0BBUmH7Krcax1RZgH4f5A', 1018)
]

```

Figure 5: Output from task 6b and 6c.

## 5 Appendix

Table 1: Output format for our solutions.

Task	Output file	Format
1	a task_1a.tsv	<i>table_name, table_size</i>
	a task_2a.tsv	<i>unique_user_count</i>
	b task_2b.tsv	<i>avg_review_length</i>
2	c task_2c.tsv	<i>business_id</i>
	d task_2d.tsv	<i>year, review_count</i>
	e task_2e.tsv	<i>datetime</i>
	f task_2f.tsv	<i>pcc</i>
	a task_3a.tsv	<i>city, rating</i>
3	b task_3b.tsv	<i>category, frequency</i>
	c task_3c.tsv	<i>postal_code, coordinates</i>
	a task_4a.tsv	<i>(src_id_1, degree_1), (src_id_2, degree_2), ...</i> <i>(dest_id_1, degree_1), (dest_id_2, degree_2), ...</i>
4	b task_4b.tsv	<i>src_mean, dest_mean</i> <i>src_median, dest_median</i>
5	a task_5a.txt	—
	a task_6a.tsv	<i>city, rating</i>
6	b task_6b.txt	—
	c task_6c.tsv	<i>user_id, review_count</i>