1.1 Methods:

1. Here are the detailed methods we designed and the tools we used to complete task1.

Github Link:

1. Go to directory: /user/hm74/NYCOpenData, reach all files related with task1
2. For each file, read it and convert it to dataframe through *spark.read* funtion
3. For each dataframe, we split into several columns. Then we process for every column.

* Get number of non-empty cells through *pyspark.sql filter* function
* Get number of empty cells through *pyspark.sql filter* function
* Get number of distinct values through *pyspark.sql groupBy* function
* Get top-5 most frequent value(s) through *pyspark.sql orderBy* function
* Traversal all cells in this column and get data type of every cell through get\_data\_type funtion, which is defined by ourselves.
* If data type is type INTEGER, add this cell value to *int\_list*

If data type is type REAL , add this cell value to *real\_list*

If data type is type DATE, add this cell value to *date\_list*

If data type is type TEXT, add this cell value to *text\_list*

* Process *int\_list* through *int\_report* function, where we will use *max, min, mean* and *stdev* function to get

-Maximum value

-Minimum value

-Mean

-Standard Deviation

* Process *real\_list* through real*\_report* function, where we will use *max, min, mean* and *stdev* function to get

-Maximum value

-Minimum value

-Mean

-Standard Deviation

* Process *date\_list* through date*\_report* function, where we will use *sort* function to get

-Maximum value

-Minimum value

* Process *text\_list* through text*\_report* function, where we will use *sort* and *len* function(sort by length) to get

- Top-5 Shortest value(s) (the values with shortest length)

- Top-5 Longest values(s) (the values with longest length)

- Average value length

* Store all requied information in *dataset* dictionary. Then use *json.dumps* function to convert dictionary to json and output to json file.

1. Discussion of challenges we have faced while designing and implementing your solution

|  |  |
| --- | --- |
| Challenge | Solution |
| How to run those different files together to speed up this process | We adopt *subprocess.check\_output* function ro run command '*hdfs dfs -ls /user/hm74/NYCOpenData*'. Throught which we get all the input file names. After that, we can use iteration to traversal all files and get output |
| For many columns, they contains space in their column name, which make it difficult to process with *pyspark.sql* | Firstly, we replace all ‘ ’ in original column name with ‘\_’, then use *pyspark.sql* *withColumnRenamed* function to conduct rename. Also when we refer to a specific column, we use *col(colName)* |
| How to identify if a cell value is INTEGER or not | We adopt function *ast.literal\_eval(),* which might give us a result. Then we check type of this result to see if this is int. If it throws Error, then it is not. |
| How to identify if a cell value is REAL or not | That’s very similar with previous solution. We adopt function *ast.literal\_eval(),* which might give us a result. Then we check type of this result to see if this is float. If it throws Error, then it is not. |
| How to identify if a cell value is Date or not | Firstly, we use *datetime.strptime()* funtion from *datetime* with some common datatime format. Then for higher precision, we call *parse(value, fuzzy=False)* function from *dateutil.parser* to get result |
| How to identify if a cell value is TEXT or not | In this task, if the cell value is not INTEGER, REAL or DATE. Then we would identify it with TEXT. |
| The input is really big so that run time of profiling tasks is long | Firstly, in coding part, we use spark instead of mapreduce. We adopt more dataframe instead of RDD. In addition, when running this task, we split the job to three members in our group. So that we’ll be able to run simultaneously. It’s about two days for every person. At last, we write some scripts to combine all those things together. |

Data quality issues we have identified:

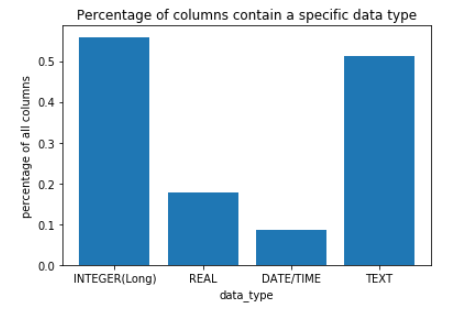
- too many missing values

- heterogeneous columns -- columns with values of multiple types.

- illegal column name which containing space and other illegal characters

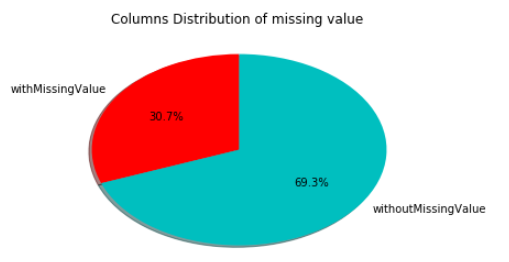
1.2 Evaluation:

For each data type, how many columns contain that type



The most common types that co-occur in columns

Missing values



Heterogeneous columns.

