Assignment Report - A4

Shabbir Hussain, Yu Wen & Navya Kuchibhotla

October 13, 2017

Program Design

Outputs

The goal of our mapreduce workflow is to output the candidate list of the most active airlines and airports along with its related information.

The reason of outputting the candidate list instead of the exact most active airlines/airports is that sorting according to values in mapreduce can be expensive. Here are two feasible approaches, both are expensive to do.We can either force the number of partition to be one and buffer the values to be sorted in memory, however, this means we are abandoning the distributed structure of mapreduce and costing huge amount of memory usage. Alternatively, we can run to map reduce job, the first one will calculate the top lists using multiple reducers the second one will calculate the mean delay information we need according to the top lists. However, this will lead to read the entire data set twice plus a lot of data will be transferred between map and reduce phase in the above two mapreduce job.

One advantage of outputting the candidate list is that the list is relatively small enough, so that we can sort the results in R markdown. Sorting a relatively small list in R is much cheaper than sorting a large list through mapreduce.

Another aspect we do differently is that in addition to just listing the mean delay of each most active airline/airport in each month, we also want to analysis the data in that month such that for each airport in the most active list, which airlines contribute the most number of flights and what was their mean delay in that month.

So, the output of candidate in list of most active airport is a bunch of records, where each record consists of

([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_DELAY_IN_PERCENTAGE, SUM_OF_DELAY_FREQUENCY, SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]).

The first “[]” indicates composite key, with each filed separated by “,”; The second “[]” indicates the value of the composite key, with each field separated by “,”. Similarly, for the airline , the record will be

([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_DELAY_IN_PERCENTAGE, SUM_OF_DELAY_FREQUENCY, SUM_OF_NUMBER_OF_FLIGHTS]).

In general the record outputted by the mapreduce will be

([KEY1TYPE, KEY1, KEY2, YEARMONTH], [SUM_OF_DELAY_IN_PERCENTAGE,SUM_OF_DELAY_FREQUENCY,SUM_OF_NUMBER_OF_FLIGHTS])

Mapreduce Design

Our design of mapreduce flow emphasis on only reading the dataset once and do another reduce side join, which is cheaper than reading the dataset twice.

First mapreduce job

Partitioner

In order to make sure all the flight number can be counted, for each [KEY1TYPE,KEY1,KEY2,YEARMONTH], a partitioner is used. The partition number are calculated base on KEY1TYPE and KEY1 such that all the records for KEY1 will go into the same partitioner.

Mapper

1.Input the records in the dataset 2. emit key value pairs according to the input：

([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [DELAY_IN_PERCENTAGE, DELAY_FREQUENCY, NUMBER_OF_FLIGHTS]) ([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [DELAY_IN_PERCENTAGE, DELAY_FREQUENCY, NUMBER_OF_FLIGHTS]).

Combiner:

combines some results from the mapper into

([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_DELAY_IN_PERCENTAGE, SUM_OF_DELAY_FREQUENCY, SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]).

([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_DELAY_IN_PERCENTAGE, SUM_OF_DELAY_FREQUENCY, SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]).

Reducer:

1. reduce the results into ([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_DELAY_IN_PERCENTAGE, SUM_OF_DELAY_FREQUENCY, SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]).

([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_DELAY_IN_PERCENTAGE, SUM_OF_DELAY_FREQUENCY, SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]).

2. Collect the information of ([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) and ([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES])

Then sort them in descending order according to “SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES”.

3.For the key

[AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] or [AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH]

in the top list, emit a special pair for the key such that

([AIRPORT_TAG, AIRPORT_NAME, BLANK_KEY,BLANK_KEY] , [-1, 0, -1])

or

([AIRLINE_TAG, AIRLINE_NAME, BLANK_KEY , BLANK_KEY] , [-1, 0, -1])

Second Mapreduce job:

This job performs a reduce side join which uses a reducer and a comparator.

The Comparator

The comparator group the data from output of the first mapreduce by using the keys of KEY1TYPE and KEY1. For keys in the top list, since a special pair ([KEY1TYPE, KEY1, BLANK_KEY,BLANK_KEY] , [-1, 0, -1]) are produced in previous step, the group comparator will concatenate the top lists records into following form.

([AIRPORT_TAG, AIRPORT_NAME, BLANK_KEY,BLANK_KEY] , [-1, 0, -1]) ([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) ([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) ([AIRPORT_TAG, AIRPORT_NAME, INCOMING_AIRLINE_NAME, MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) ……

Or

([AIRLINE_TAG, AIRLINE_NAME, BLANK_KEY , BLANK_KEY] , [-1, 0, -1]) ([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) ([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) ([AIRLINE_TAG, AIRLINE_NAME, DESTINATION_AIRPORT_NAME , MONTH] , [SUM_OF_NUMBER_OF_FLIGHTS_ARRIVES]) …..

The special pair can be ensured to appear in the first row since two field of the composite key is blank.

For keys that are not in the top list there are no special pair.

The Reducer

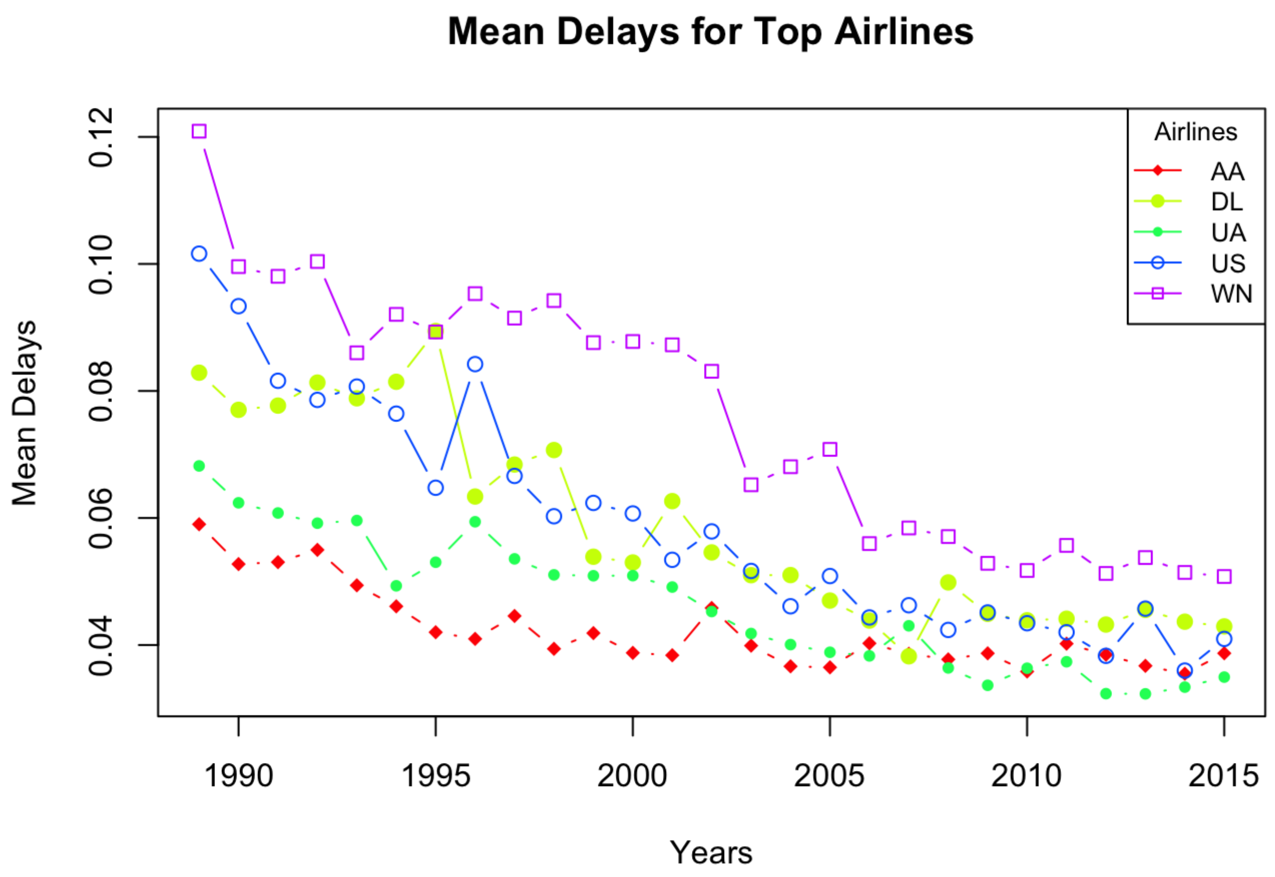
The reducer input the data mentioned mentioned in the comparator. It will read the row to see whether a special pair exists. If it exists, it indicates the key is in the top list, then the reducer output all the pairs of that key but the first row(it should be skipped since it was used as a marker for top list).

Visualization of Delays:

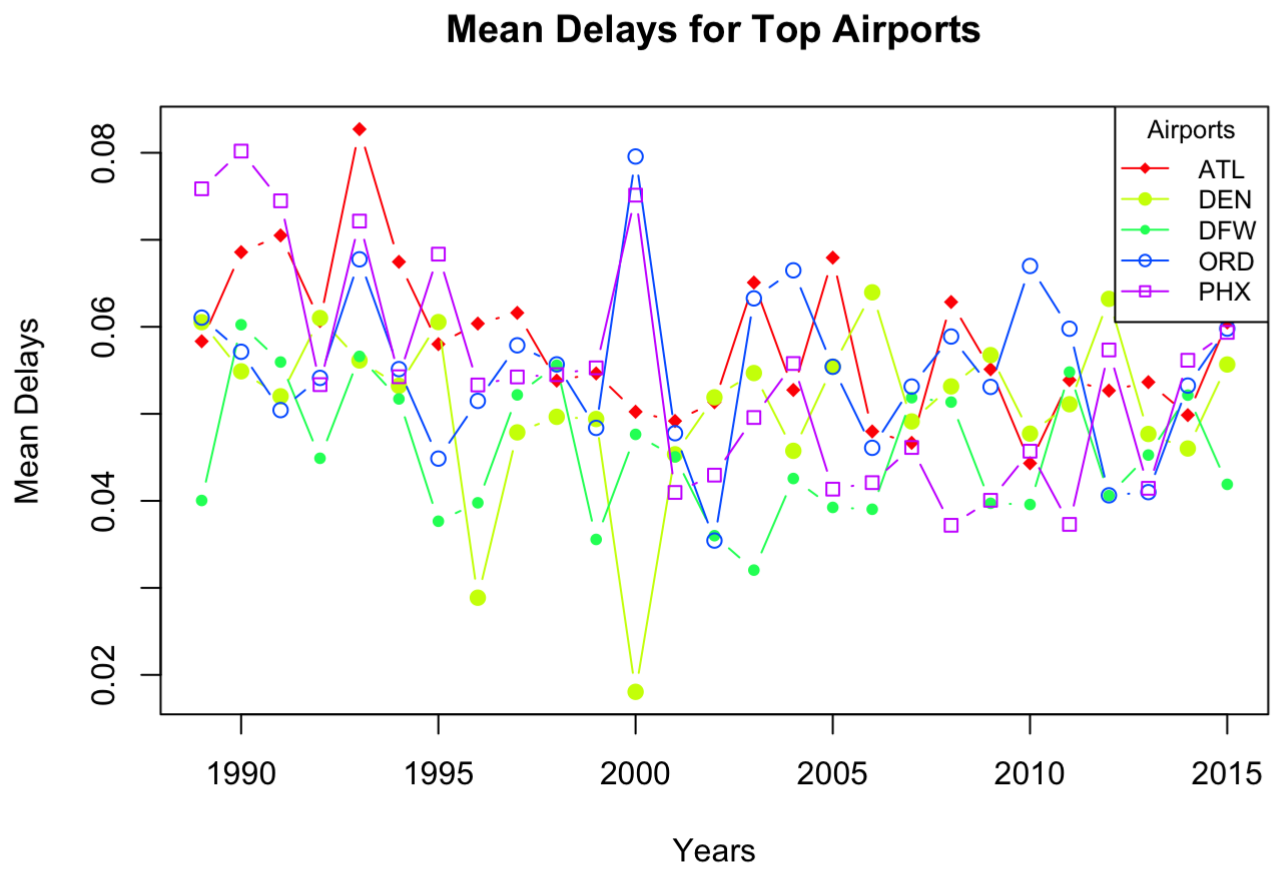
1. Aggregate Mean Delays:

The two graphs below plot the over all mean delays for five most active airlines and for the five most active airports in the country over the entire data set.

- The graph below plots mean delays for top 5 airlines observed for each year over the entire data set.



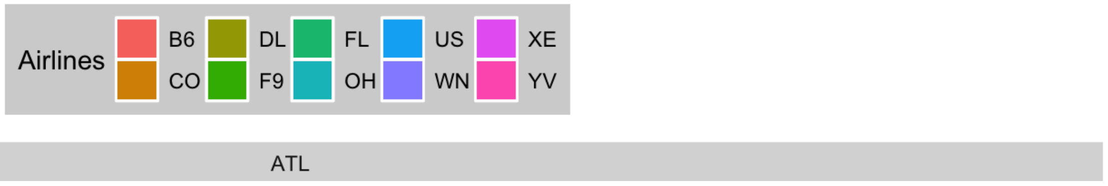
- The graph below plots mean delays for top 5 airports observed for each year over the entire data set.

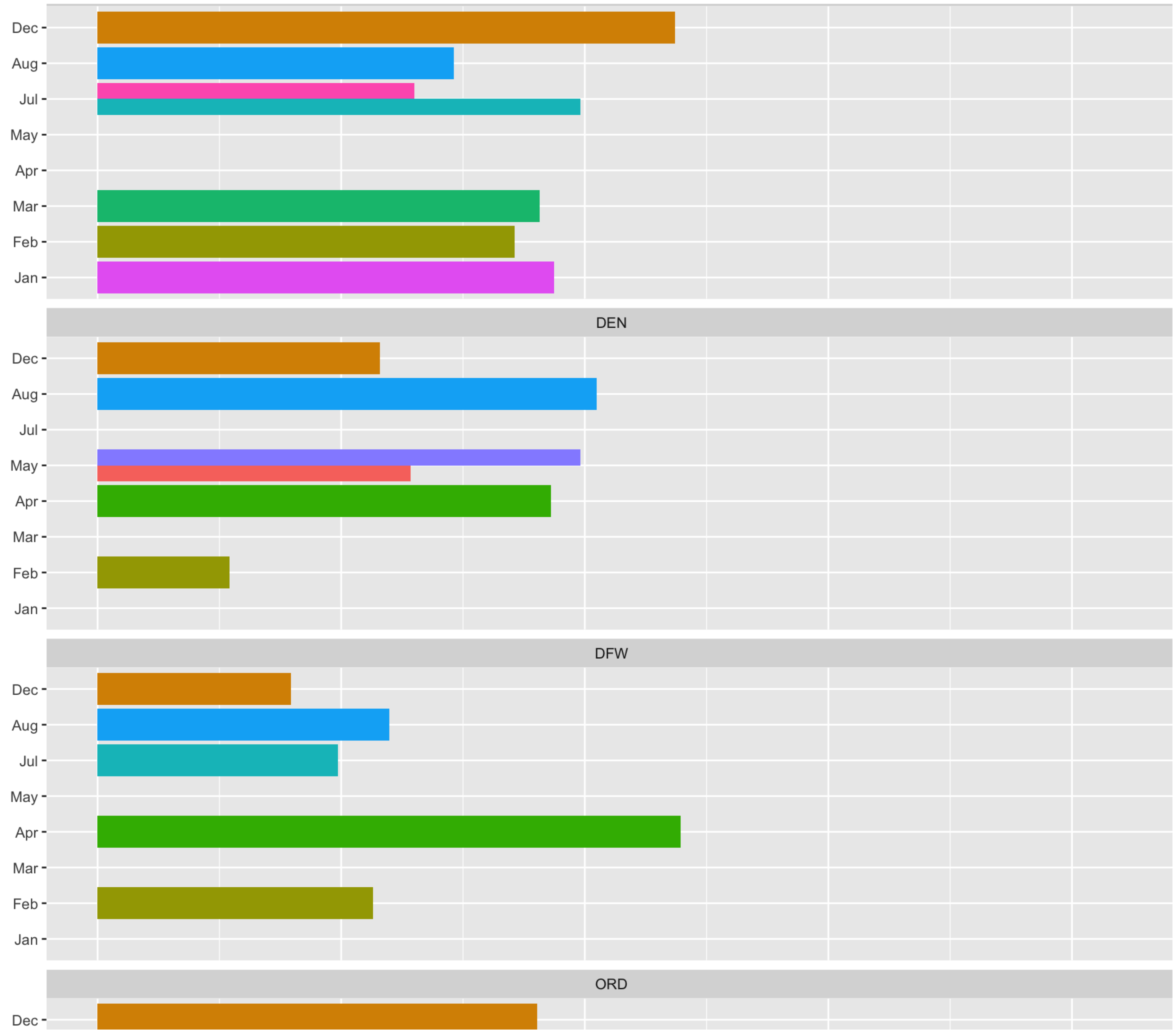


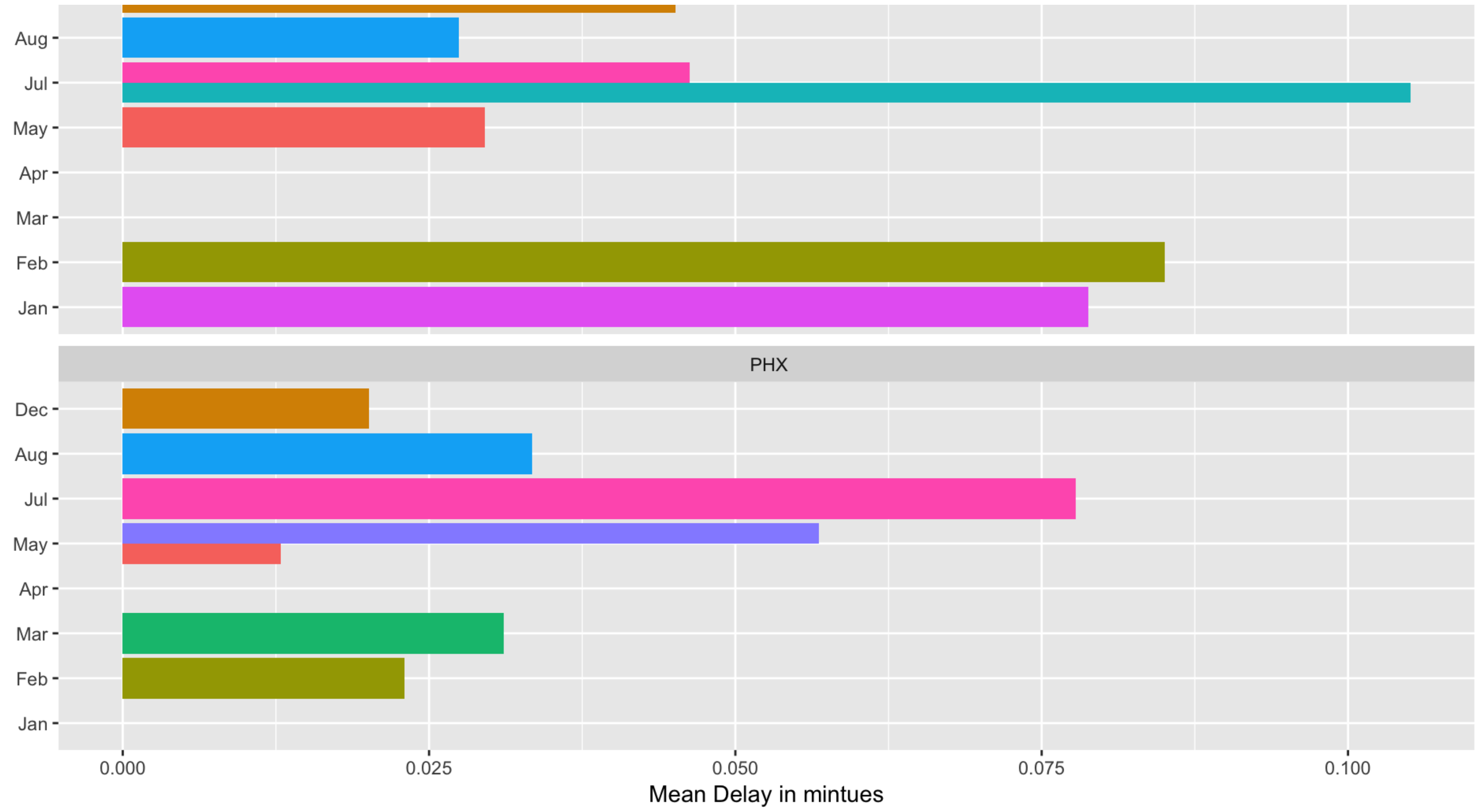
2. Mean Delays for year 2010:

Among the data from all the years we picked year 2010 at random, and graphs below plot two parts:

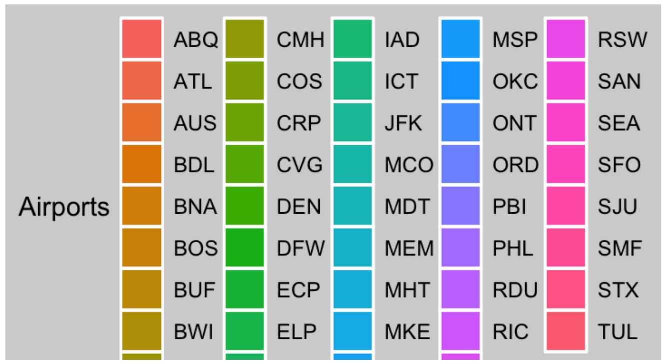
- Mean delay per flight, for all the airlines arriving at the top 5 airports, plotted for each month.
  For example if we were to look at the flights arriving at Atlanta airport (ATL) in the month of Jan, we know from the data, that our best bet to avoid delays would be to take the Continental Airlines(CO) rather than the EVA Air(EV). Similarly for the month of Aug, Delux Public Charter(XE) does much better than PSA Airlines(OH).

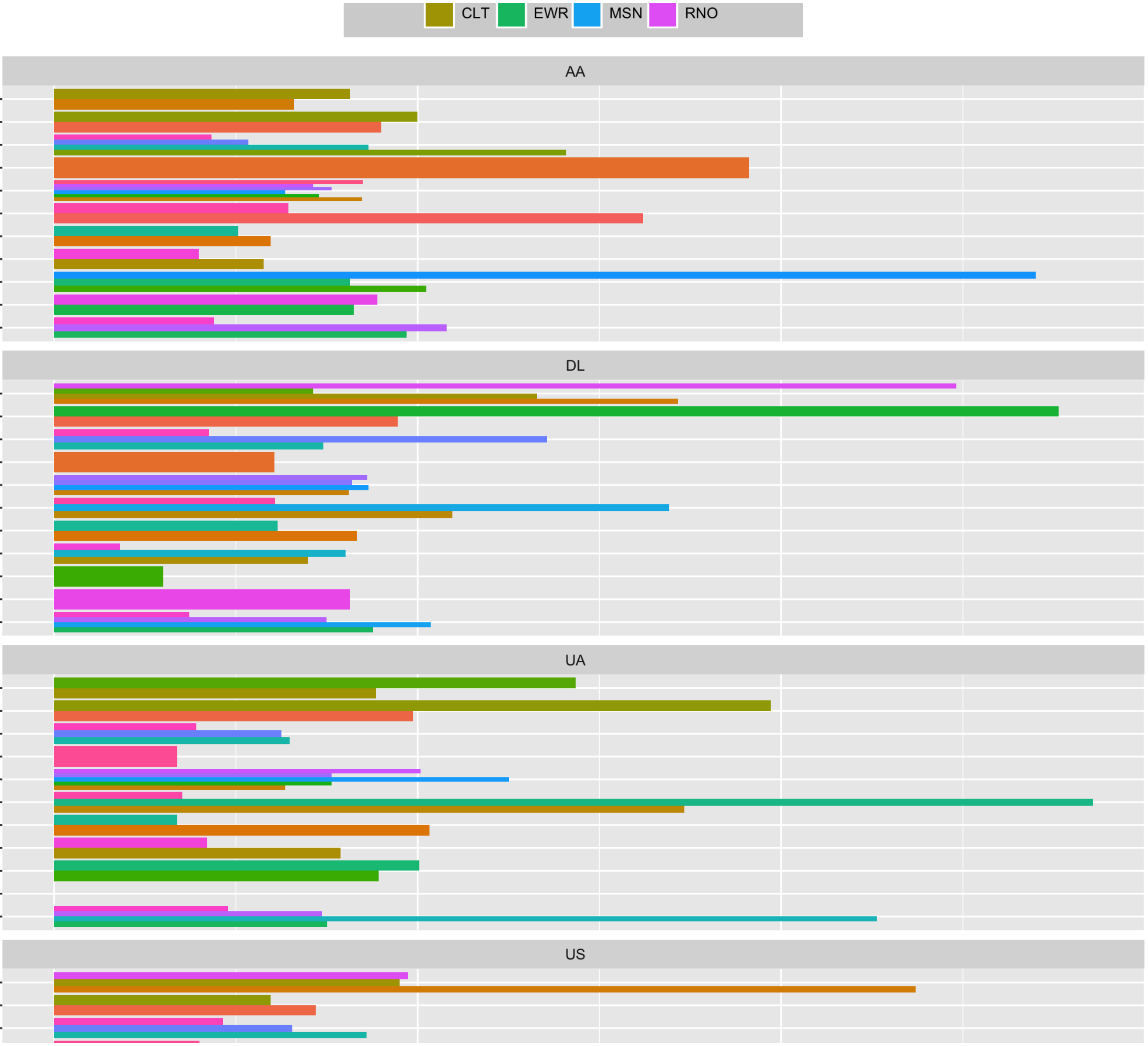


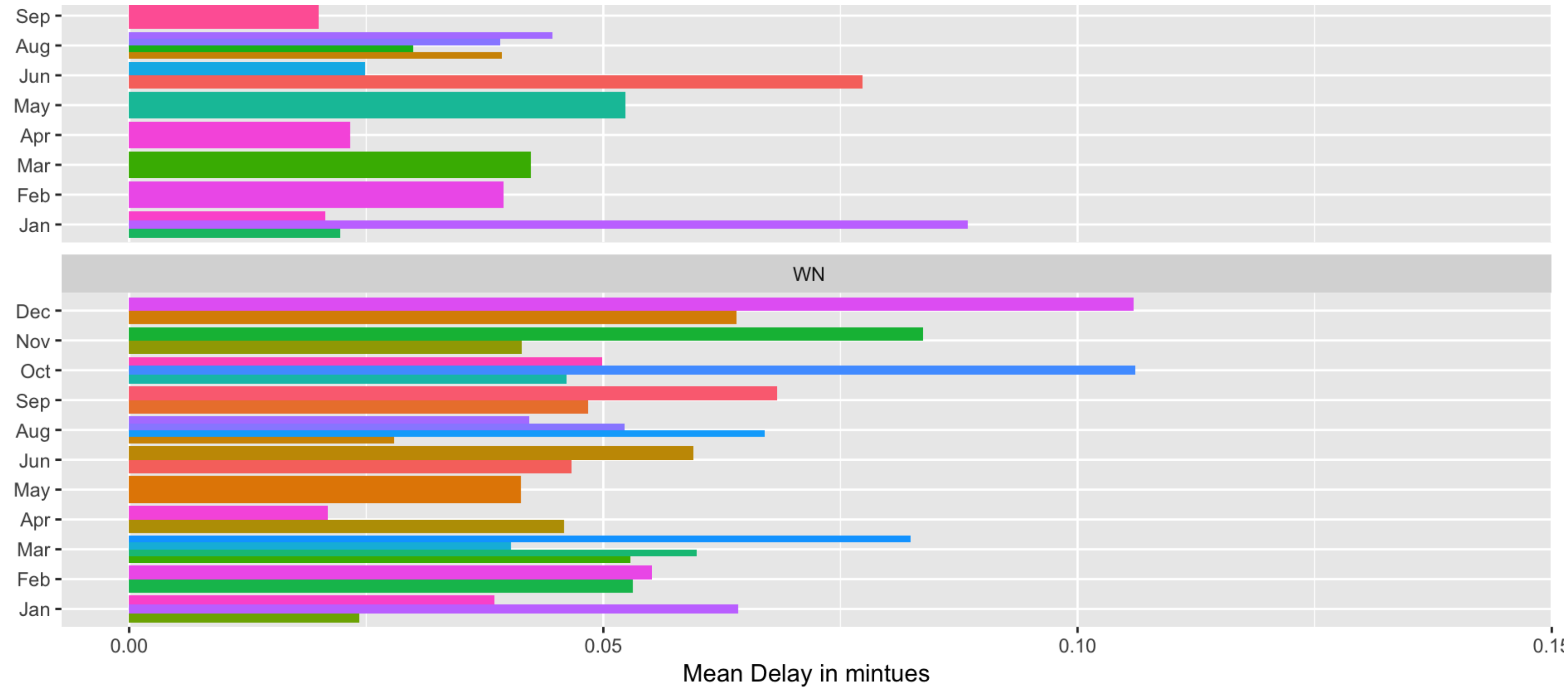

Dec
Aug
Jul
May
Apr
Mar
Feb
Jan
DEN
Dec
Aug
Jul
May
Apr
Mar
Feb
Jan
DFW
Dec
Aug
Jul
May
Apr
Mar
Feb
Jan
ORD
Dec



- Mean delay per flight, for all the airports that the top 5 airlines fly to, plotted for each month.
For example if we were to look at the American Airlines(AA) flying flights during the month of Sept, we can notice that flights going to Tucson International Airport, Arizona (TUS), have a much better performance compared to flights going to Manchester, New Hampshire (MTH).




CLT
EWR
MSN
RNO
AA
DL
UA
US
Dec
Nov
Oct
Sep
Aug
Jun
May
Apr
Mar
Feb
Jan



Conclusion:

1. Mean delay for Top Airlines is observed to decrease over the years, which is intuitive considering airlines have improved their performance over time.
2. To optimize number of jobs we had to implement multiple hacks obscuring the code readability.