A4 - On-Time Performance Data

Jiangtao, Joyal and Bhanu

October 12, 2017

Contents

Problem

http://janvitek.org/pdpmr/f17/task-a4-delay.html

Solution

Architecture

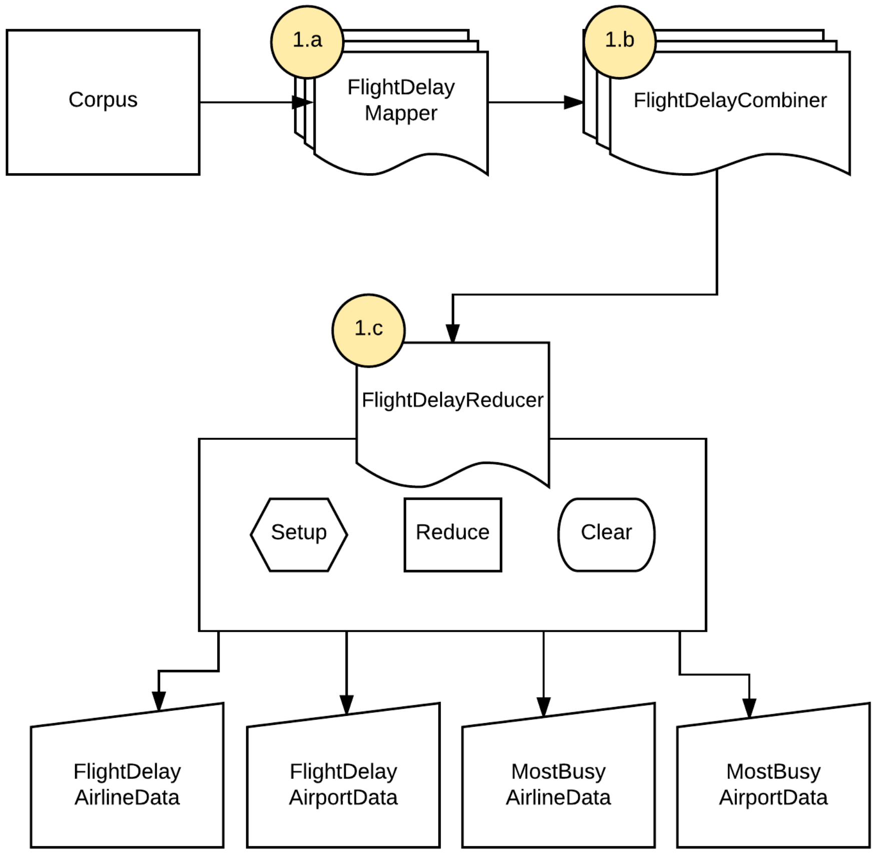


Figure 1:

Arrows head represent the data flow

1. FlightDelayJob

1a. FlightDelayMapper

The mapper reads each line from flight record csv file, cleans data as per given sanity rules and generates two Composite `<K,V>` pair for airline and airport. The `key` is type of `FlightCountCompositeKey` class which implements `WritableComparable` interface while the `value` is type of `FlightDataWritable` class which implements `Writable` interface.

```
# FlightDelayMapper Input k,v
LongWritable, Text
# FlightDelayMapper Output k,v
FlightCountCompositeKey, FlightDataWritable
```

FlightCountCompositeKey

The structure of `FlightCountCompositeKey` is as shown below:

```java
public class FlightCountCompositeKey implements WritableComparable<FlightCountCompositeKey> {
  private IntWritable year;
  private IntWritable month;
  private IntWritable aaCode;
  private Text aaName;
  private IntWritable recordType; //1-Airport , 2-Airline
}
```

This class was created to avoid reading file twice. Each input line of data contains information for both airline and airport. Hence, a mapper creates two instance of `FlightCompositeKey` and its corresponding value class `FlightDataWritable`. The following table shows difference in fields for each composite key instance.

Fields

AirLine.Instance

Airport.Instance

year

yyyy

yyyy

month

mm

mm

aaCode

Unique code for Airport like ‘JFK’

Unique code for Airline ‘UA’

aaName

Name for the Airport

Name of Airline

recordType

2

1

This class also overrides `compareTo` method in the following way

```java
  @Override
  public int compareTo(FlightCountCompositeKey key) {
    int code = this.year.compareTo(key.getYear());
    if (code == 0) {
      code = this.month.compareTo(key.getMonth());
      if (code == 0) {
        code = this.recordType.compareTo(key.getRecordType());
        if (code == 0) {
          code = this.getAaCode().compareTo(key.getAaCode());
        }
      }
    }
    return code;
  }
```

The overridden `compareTo` acts as both `SortComparator` and `GroupingComparator`.

FlightDataWritable

The structure of `FlightCountCompositeKey` is as shown below:

```
public class FlightDataWritable implements Writable {
// the value part of 1st map output pair <key, value>

FloatWritable delay;
IntWritable count;
}
```

The mapper class emits instance of this class as value to combiner/reducer. Each Value object contains two fields: `delay` and `count`. `delay` captures the normalized delay for airport/airline. `Count` is used to calculate globally active airport/airline, and it also contributes towards `mean delay`.

1.b FlightCountCombiner

Combines on keys for both `count` and `delay`. As `FlightCountReducer` is set to single instance, the combiner helps to reduce the input, thus improving performance.

```
# FlightCountCombiner Input k,v
FlightCountCompositeKey, FlightDataWritable
# FlightCountCombiner Output k,v
FlightCountCompositeKey, FlightDataWritable
```

The `reduce` method of combiner receives `FlightCountCompositeKey` key and `Iterable<FlightDataWritable>` values, and as explained above, sums up totalDelay for each airline and airport for each year each month.

At the end of the combiner phase, we will have combined delay and count for each airline and airport for each year each month.

This process is shown below:

Year

Month

aaCode

aaName

recordType

Delay

Count

1989

1

1247801

JFK

1

27

1

1989

1

19805

AA

2

27

1

1989

1

1247801

JFK

1

12

1

1989

1

19805

AA

2

12

1

As shown in the above table, there are four records, 2 each of type airport and airline. The data from the above table will be grouped on `FlightCountCompositeKey` and will be combined as shown below:

Year

Month

aaCode

aaName

recordType

Delay

Count

1989

1

1247801

JFK

1

39

2

1989

1

19805

AA

2

39

2

1.c FlightCountReducer

The Reducer phase receives input and emits output in the following `<k,v>` format

```
# FlightCountReducer Input k,v
FlightCountCompositeKey, FlightDataWritable
# FlightCountReducer Output k,v
FlightCountCompositeKey, FlightDataWritable
```

It has 3 sub phases:

- `Setup`: It initializes `topKCount` from job configuration, which is the value for the final top k flights for which we want to run the MR job.
- `Reduce`: The reduce phase works on the total following reducer local variables:

No.	Variable	Description
1	airportFlightCount	Holds Global flight counts for airport: k->aaCode, v->Count
2	airlineFlightCount	Holds Global flight counts for airline: k->aaCode, v->Count
3	airportFlightCountSorted	Local Map to hold sorted count for airport

4

airlineFlightCountSorted

Local Map to hold sorted count for airline

5

reducedValueMap

Local Map to hold reduced records for each <YEAR,MONTH,RECORD_TYPE,CODE>

6

mostBusyMap

Local Map to hold List of Most Busy Airport/Airline

It performs the following operations,

  - Reduces `count` and `delays`
  - Calcualtes `mean delays`
  - Stores `counts` in local map for global winner
  - Stores reduced `mean delays` in local map for filtered output.

- `Clear`: It performs the following operations,
  - Sort Global counts stored in local maps for Global winners.
  - Writes Global winners to file.
  - Filters reduced `mean delays` and writes to file.

Outputs

The output is explained below,

- mostBusyAirportData-r-00000 : Data for top 5 most busy airports. Every row represents,

  `<recordType=1>,<airportCode>`

- mostBusyAirlineData-r-00000 : Data for top 5 most busy airline. Every row represents,

  `<recordType=2>,<airlineCode>`

- flightDelayAirportData-r-00000 : The final `mean delays` for airports. Data is aggregated on Year,Month,AirportCode for top 5 most busy airports. Every row represents,

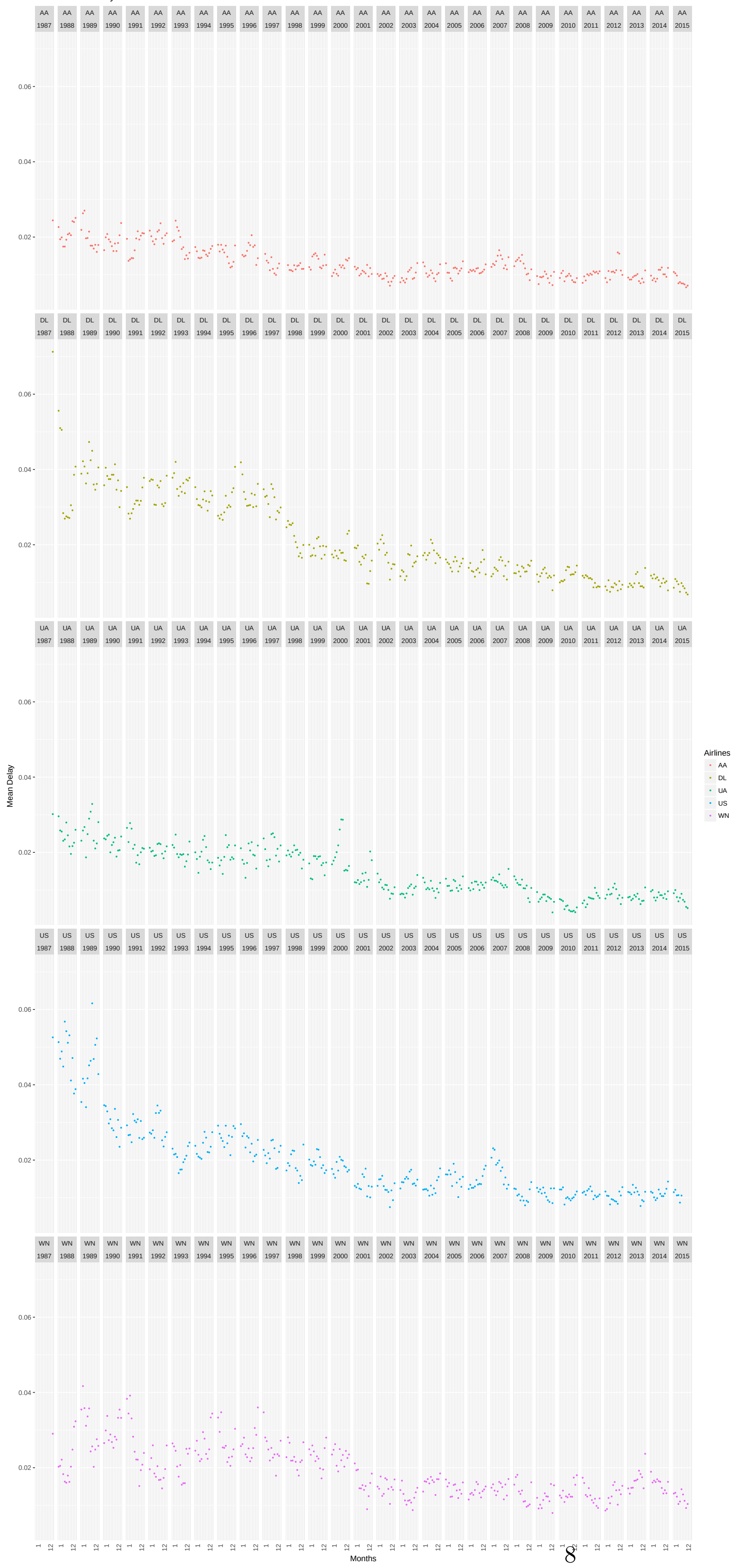
  `<year>,<month>,<airportCode>,<airportName>,<recordType=1>,<totalDelay>,<totalCount>`

- flightDelayAirlineData-r-00000 : The final `mean delays` for airlines. Data is aggregated on Year,Month,AirlineCode for top 5 most busy airlines. Every row represents,

  `<year>,<month>,<airlineCode>,<airlineName>,<recordType=1>,<totalDelay>,<totalCount>`

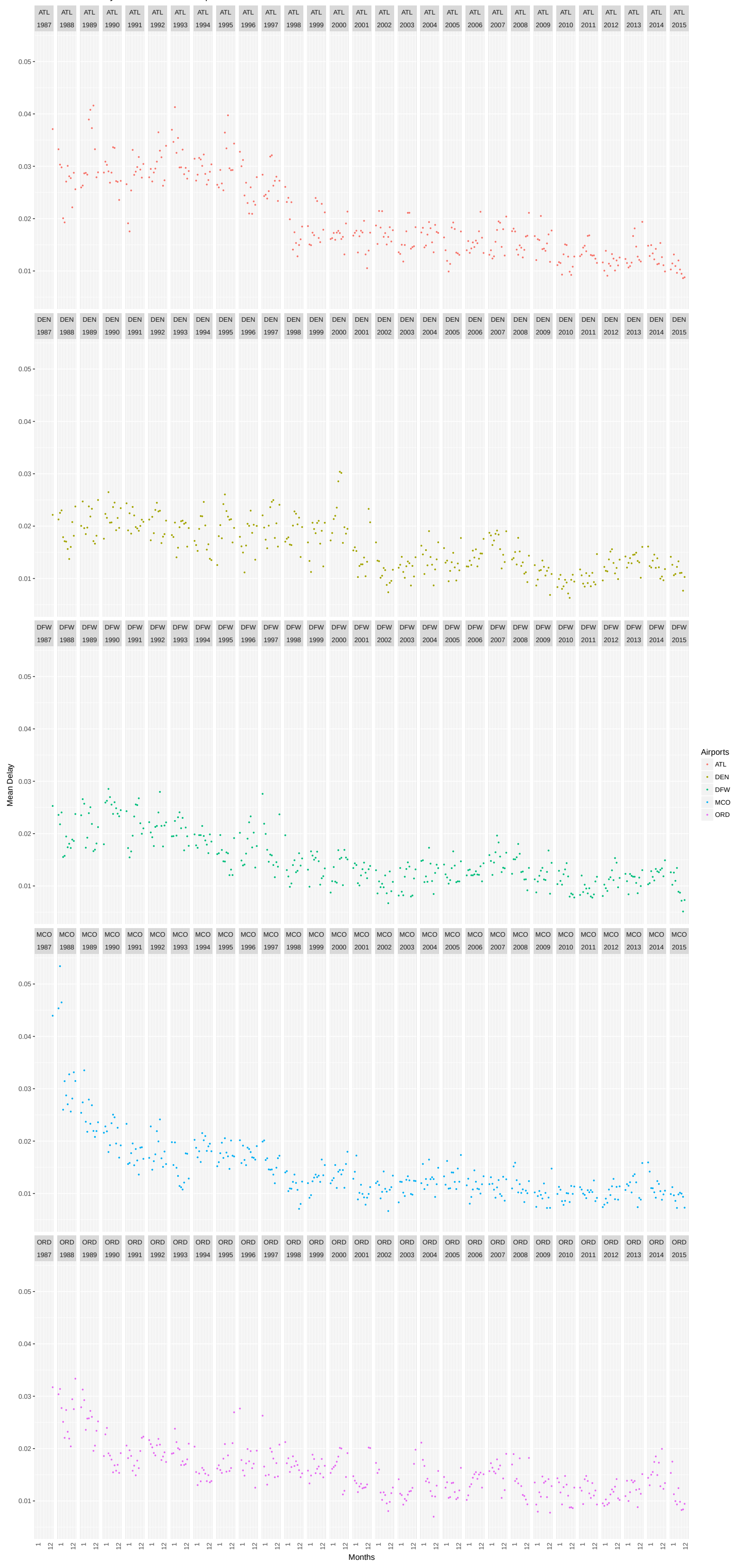
Analysis

Plot 1. Mean Delay For 5 Most Active Airlines

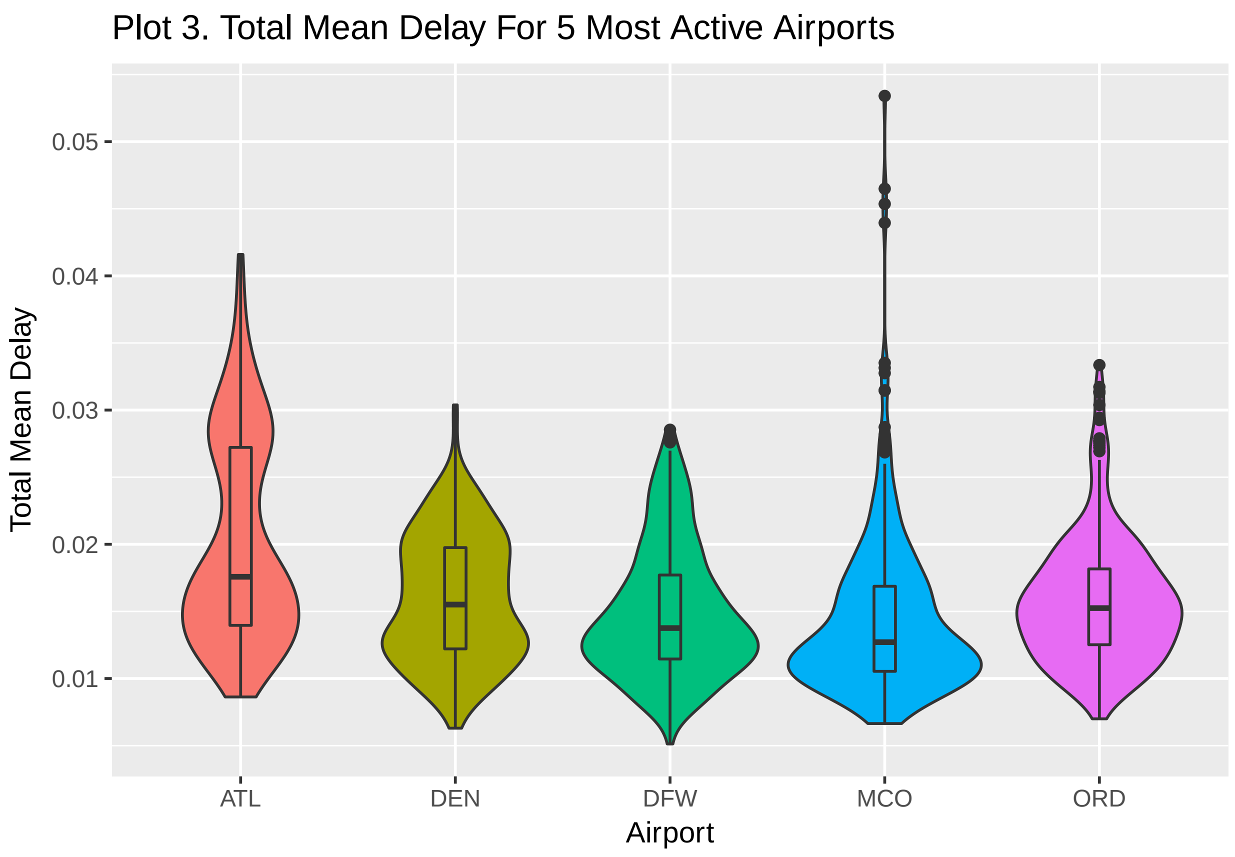


`Plot 1.` tells us about mean delay of top five most active airports across corpus for each month from year 1987-2015. The delay is normalized by length of flight. There is a clear trend of reduction in delays over the year range. This trend is seen for all the airports and indicates the improvement in flight control and technology over these years.

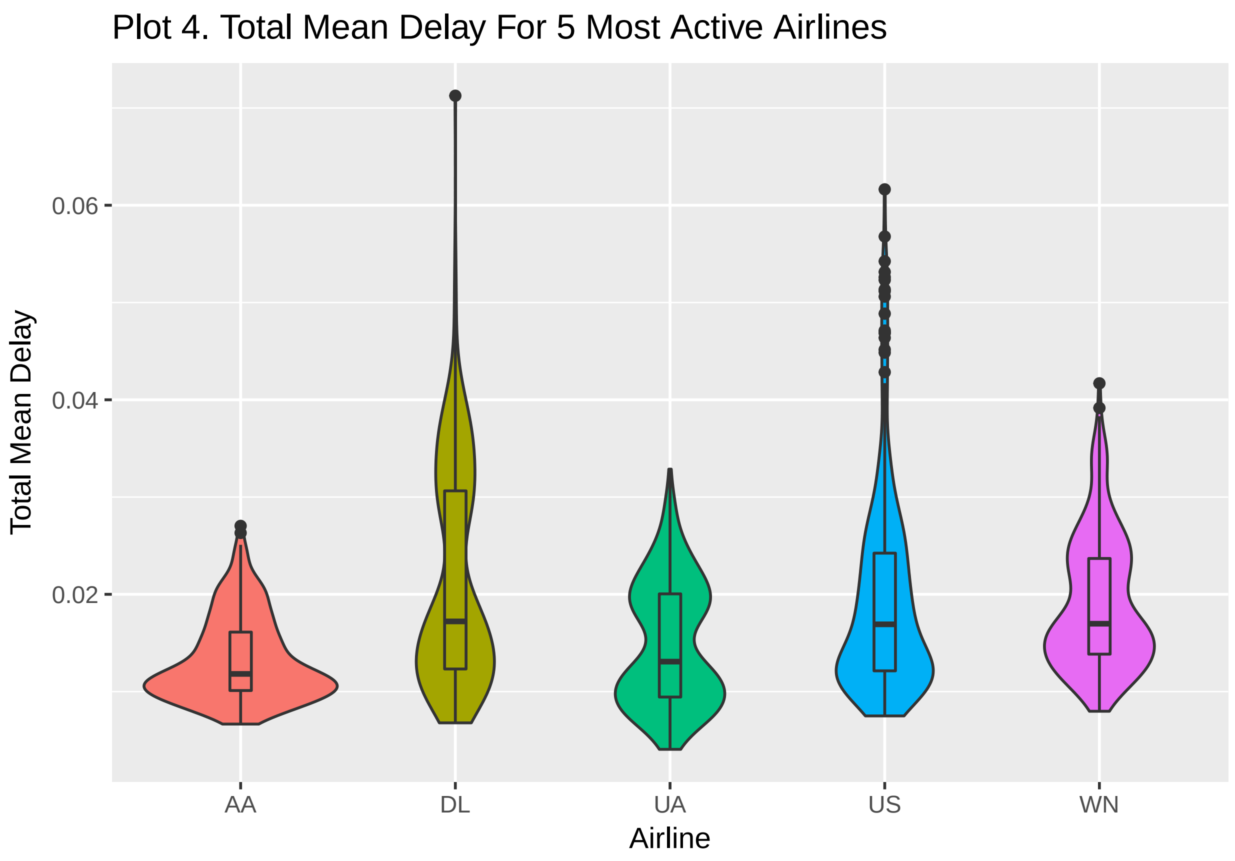
Plot 2. Mean Delay For 5 Most Active Airports



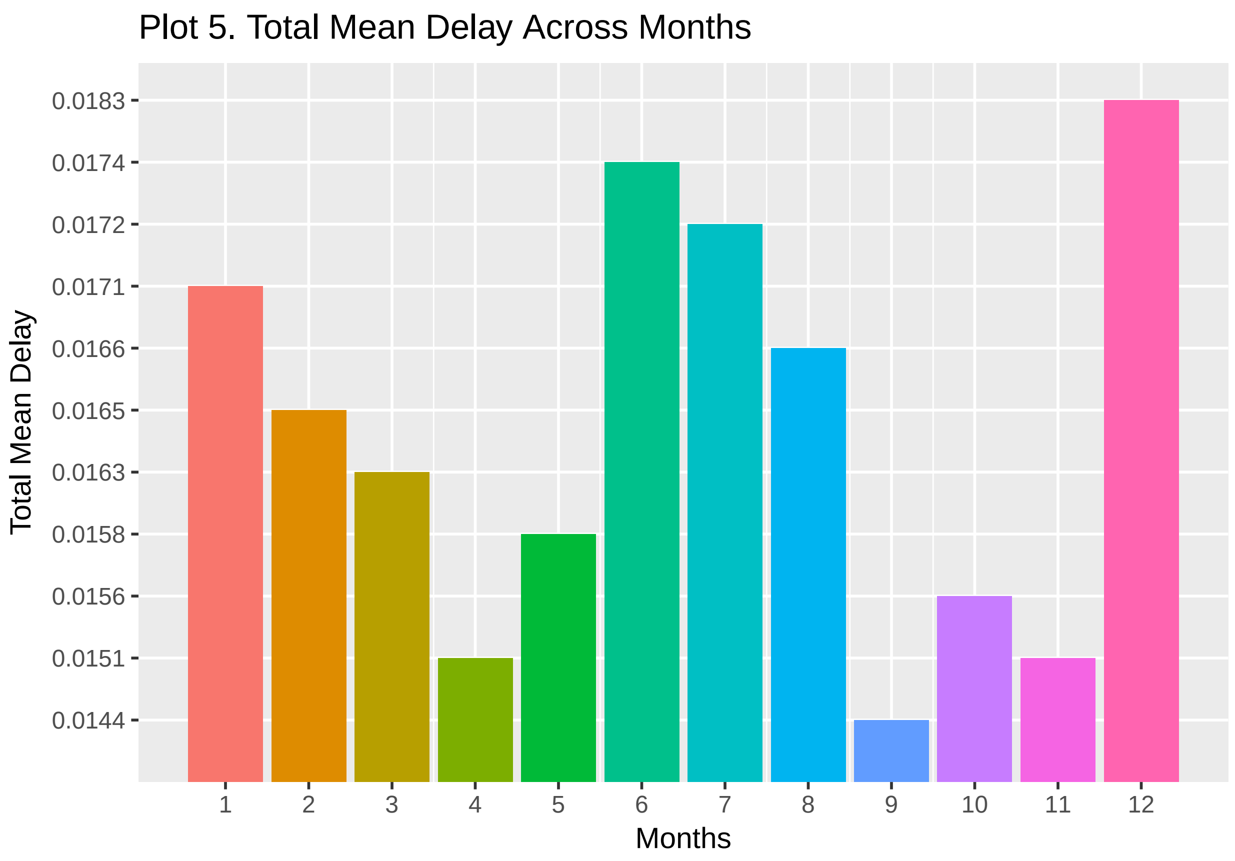
`Plot 2.` tells us about mean delay of top five most active airlines across corpus for each month from year 1987-2015. The delay is normalized by length of flight. There is a clear trend of reduction in delays over the year range. This trend is seen for all the airlines and indicates the improvement in flight control and technology over these years.



`Plot 3.` shows total mean delay for 5 most active airport from 1987-2015. We can see that Airport `ATL` has the highest median with frequency distribution along the complete range. We can infer that `ATL` has performed poorly over these 29 years as compared to all other airports.



`Plot 4.` shows total mean delay for 5 most active airlines from 1987-2015. `DL`,`US`,`WN` have median around .0175, but as `DL` is more elongated, we can concur that it is has most delays over the period of 29 yrs.



`Plot 5.` is analysis of seasonal trends of mean delays from 1987-2015. `x` axis represents the months and `y` axis gives total mean delay for all top 5 airports over 29 yrs. From the figure we can safely say that the month of December has the most delays across airports, maybe due the Holidays. Additionally, September saw, least delays, indicating the end of summer and less travel.

Job Execution

Machine Specifications

Improvements

- Improve heap performance.