A8 Regression

Karthik Chandranna
chandranna.k@husky.neu.edu

Naveen Aswathanarayana
aswathanarayana.n@husky.neu.edu

Ruinan Hu
hu.ru@husky.neu.edu

Sujith Narayan Rudrapatna Prakash
rudrapatnaprakash.s@husky.neu.edu

DESCRIPTION:

The aim of this assignment is to find the cheapest carrier and plot the evolution of its weekly median price over time using the Spark framework. We use the provided historical dataset containing airline data for the past 28 years and a value N representing the elapsed time in minutes to find the cheapest carrier through Linear Regression.

IMPLEMENTATION DETAILS:

DESIGN:

The key design idea is to have the data mapped to a dataframe by a combination of year and carrier as the key, and then implement Linear Regression to obtain the slope and intercept for each key using elapsed time and price. These values are then plugged in to the formula for slope-intercept form to calculate a value, say V, (intercept + slope*N) for each year and carrier combination.

We then use Spark SQL to get the cheapest carrier. This carrier has the least total value of V for over a period of 3 years.

Linear Regression:

The historical data is first read into a dataframe. The records are sanitized to get rid of bad data by calling the validate() method from the java utility. The records are then grouped by having a combination of year and carrier as the key and the elapsed time and price as the value. We then use the linear regression library to get the coefficients i.e. slope and intercept for each key. The result is a dataframe having a value (intercept + slope) * N for each year and carrier combination.

Spark SQL:

The regression dataframe is registered as a temporary table. Spark's SQLContext is used to get the cheapest carrier. This is done in two stages:

1) Group the data by year and get the carrier having the least value for each year. This result table is written to an intermediate dataframe.
2) Register the intermediate dataframe as a table and get the carrier that has the most occurrences. This is the cheapest carrier.

Weekly Median:

The input data is filtered to contain only the records of the cheapest carrier. The records are then mapped by year and week number (calculated by the java utility) as key and the price as values. After grouping by key, each year and week combination will have a list of prices. The median price is calculated by sorting this list. The result is a dataframe containing a median price for each year and week. This represents the evolution of price over time.

DESIGN JUSTIFICATIONS:

1. There were two approaches for getting the cheapest airline from the dataframe having year, carrier and a value. One was to use scala maps to group and compare the values, and the other was to use Spark SQL. We chose the latter because Spark SQL gives an efficient and seamless interface to using group by and aggregate queries to obtain the required results.

2. We reused the sanity from A4 by using a java utility class, as opposed to writing the same in scala. The reason was that the sanity code from A4 had been checked during many reviews and probably was at its best implementation. It really made sense to use a java utility class to reuse existing code, as opposed to writing it from scratch using scala, more so when you consider that the requirements do not place much emphasis on the language.

HADOOP VS SPARK:

From a pure software engineering perspective, it is much simpler in Spark to implement the map and reduce concepts, whereas in Hadoop, the process is more elaborate. Spark needs far fewer lines of code when compared to Hadoop. Our Hadoop implementation for A4 had around 900 lines of code, but our Spark implementation has just 120 lines of scala code.

Further, in Spark, it is easier to identify the data flow throughout the program. Hadoop involves writing the mapper and reducer output to files and as a result, it's harder to identify the data flow. Spark, on the other hand, has RDDs that simplify the visualization of data flow in the program.

Troubleshooting is harder in Spark due to the lack of intermediate output files. In Hadoop, we were able to analyze the intermediate outputs from the mapper.

Spark SQL was a huge plus when we had to perform joins on datasets. All we had to do was register a dataframe as a temporary table and all SQL operations could be performed on it.

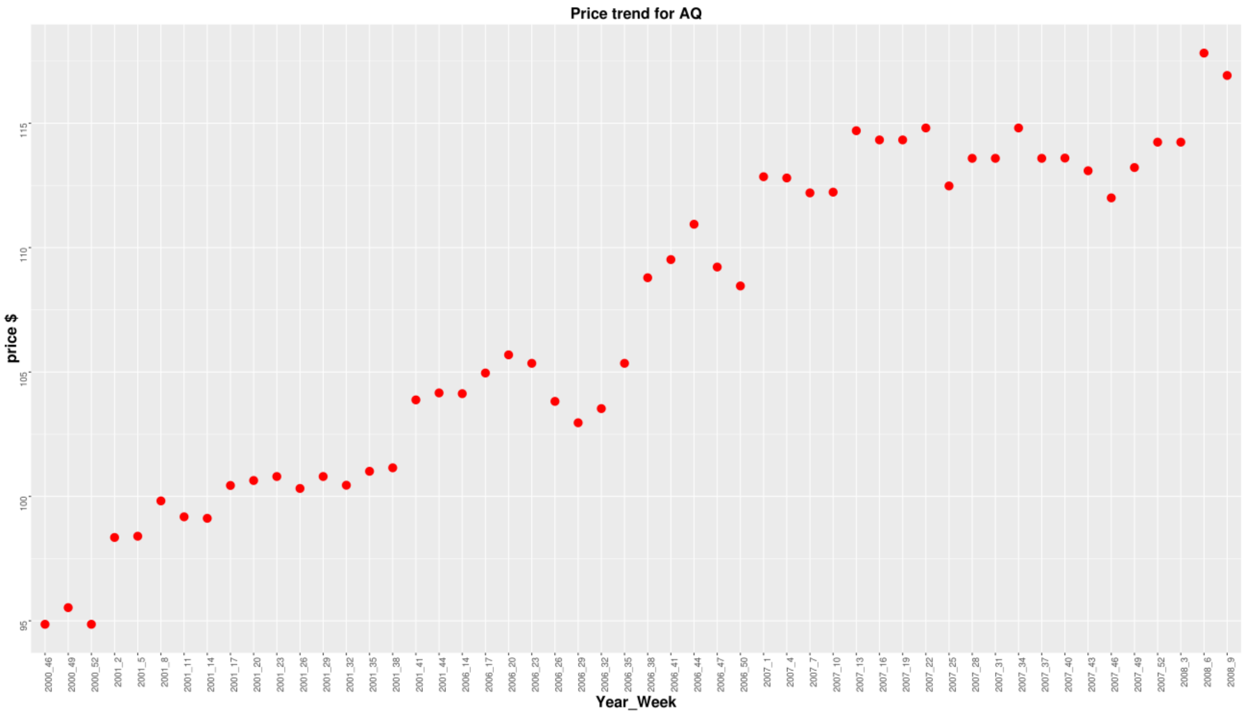
PERFORMANCE COMPARISON:

MACHINE CONFIGURATION	JOB TYPE	TIME IN MINUTES
AWS – 1 Master 2 Slave, m3.xLarge	Hadoop	49
	Spark SQL	37
AWS – 1 Master 4 Slaves, m3.xLarge	Hadoop	21
	Spark SQL	15
AWS – 1 Master 9 Slaves, m3.xLarge	Hadoop	8
	Spark SQL	5

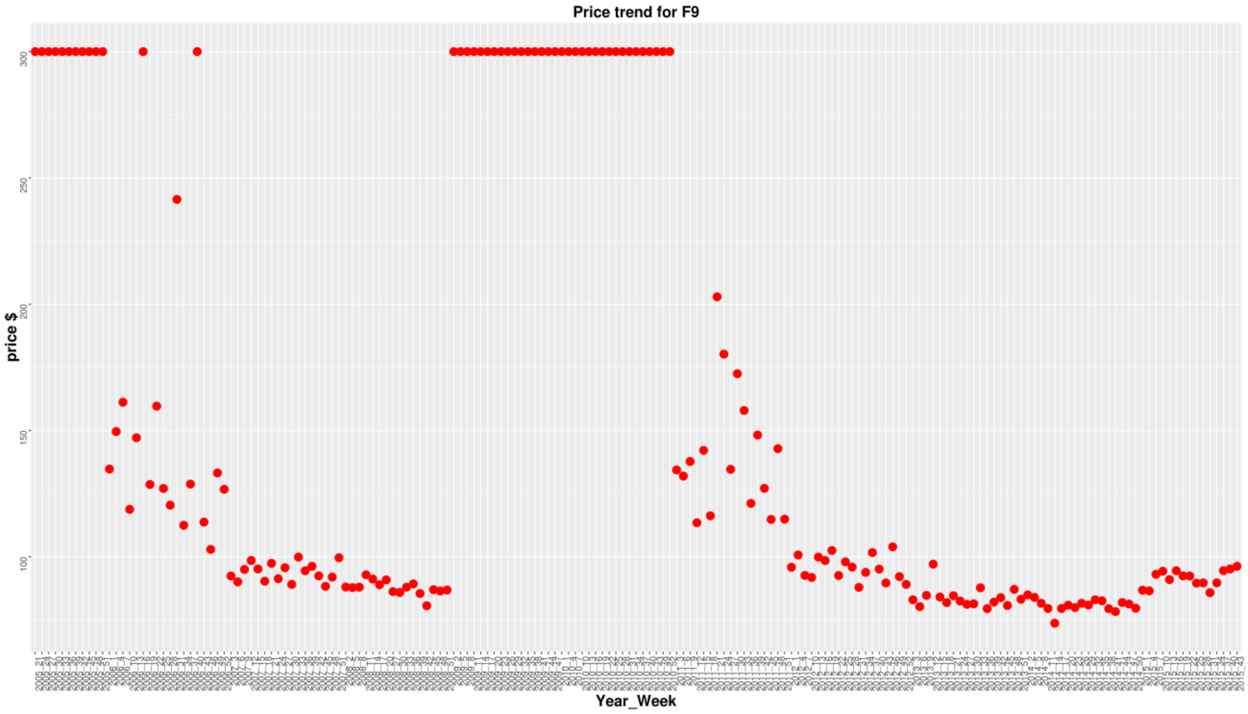
RESULTS:

For the value of N=1, we found out that AQ was the cheapest flight.

For the value of N=200, we found out that F9 was the cheapest flight.



Price trend for AQ
price $
Year_Week
95
100
105
110
115
2000_46
2000_49
2000_52
2001_2
2001_5
2001_8
2001_11
2001_14
2001_17
2001_20
2001_23
2001_26
2001_29
2001_32
2001_35
2001_38
2001_41
2001_44
2006_14
2006_17
2006_20
2006_23
2006_26
2006_29
2006_32
2006_35
2006_38
2006_41
2006_44
2006_47
2006_50
2007_1
2007_4
2007_7
2007_10
2007_13
2007_16
2007_19
2007_22
2007_25
2007_28
2007_31
2007_34
2007_37
2007_40
2007_43
2007_46
2007_49
2007_52
2008_3
2008_6
2008_9



Price trend for F9
price $
Year_Week
100
150
200
250
300