

Exploratory Data Analysis for Amazon Redshift with R & dplyr



Kan Nishida [Follow](#)
Aug 10, 2016 · 9 min read

Many of our Exploratory users store data in Amazon Redshift database for variety of reasons, but one thing in common among them is the need for quickly exploring the data to uncover patterns and trends that were unknown before. And true data exploration often involves transforming (or wrangling) the data, visualizing the data,

building statistical models, and testing with statistical tools, not in a sequential way, but in a very iterative way going back and forth among those tasks at the speed of our thoughts.

Writing SQL queries is hard and slow if you want to go beyond counting rows. And what you can do with SQL is too limited for a practical exploratory data analysis. So why not extracting the data into R so that you can explore the data in a super fast and iterative way with all the statistical tools R offers? And that's many of our users have already been doing, and I wanted to share how you, too, can do to take advantage of R to discover more about your data that is stored in Redshift database.

I'm going to demonstrate it by using dplyr commands inside Exploratory Desktop. dplyr is kind of like SQL but it's way more powerful and capable than that. It provides a set of the 'easy-to-read' grammar based commands to do data wrangling and analysis in a super fast and the most effective way in R.

hadley/dplyr

dplyr - Dplyr: A grammar of data manipulation

[github.com](https://github.com/hadley/dplyr)

Exploratory

Most of the data is sitting outside of the traditional databases today. You can quickly extract data from various data...

exploratory.io

Import Data from Amazon Redshift

Create Connection

First, you want to create a connection for Amazon Redshift database in Project List page in Exploratory Desktop.

Import Project

Account

R Packages

Connections

Created By Updated

kanaugust 8/4/2016, 10:06:53 AM

kanaugust 8/4/2016, 2:00:32 AM

After filling the database information, click ‘Test Connection’ button to make sure that the information you have typed is correct, before you save it.

Import Project

Connected to Redshift at exploratory-io.coezpt... successfully.

Name: Redshift

Type: Redshift

Host: [redacted]

Port: 5439

Database: exploratory

Username: exploratory

Password: [redacted]

Test Connection Update Cancel

Import data with SQL Query

Inside the project, you can select Amazon Redshift from Remote Data selection dialog.

Select the connection you created from Connection dropdown at left hand side and start typing SQL query in ‘SQL Query’ input field and click ‘Get Data’ button.

Import Amazon Redshift

Name: flight_redshift

Connection Type: Use Connection (Redshift)

SQL Query:

```

1 SELECT *
2 FROM airline
3 WHERE year = 2016
4 AND month = 1

```

SQL tip: Random data Sampling

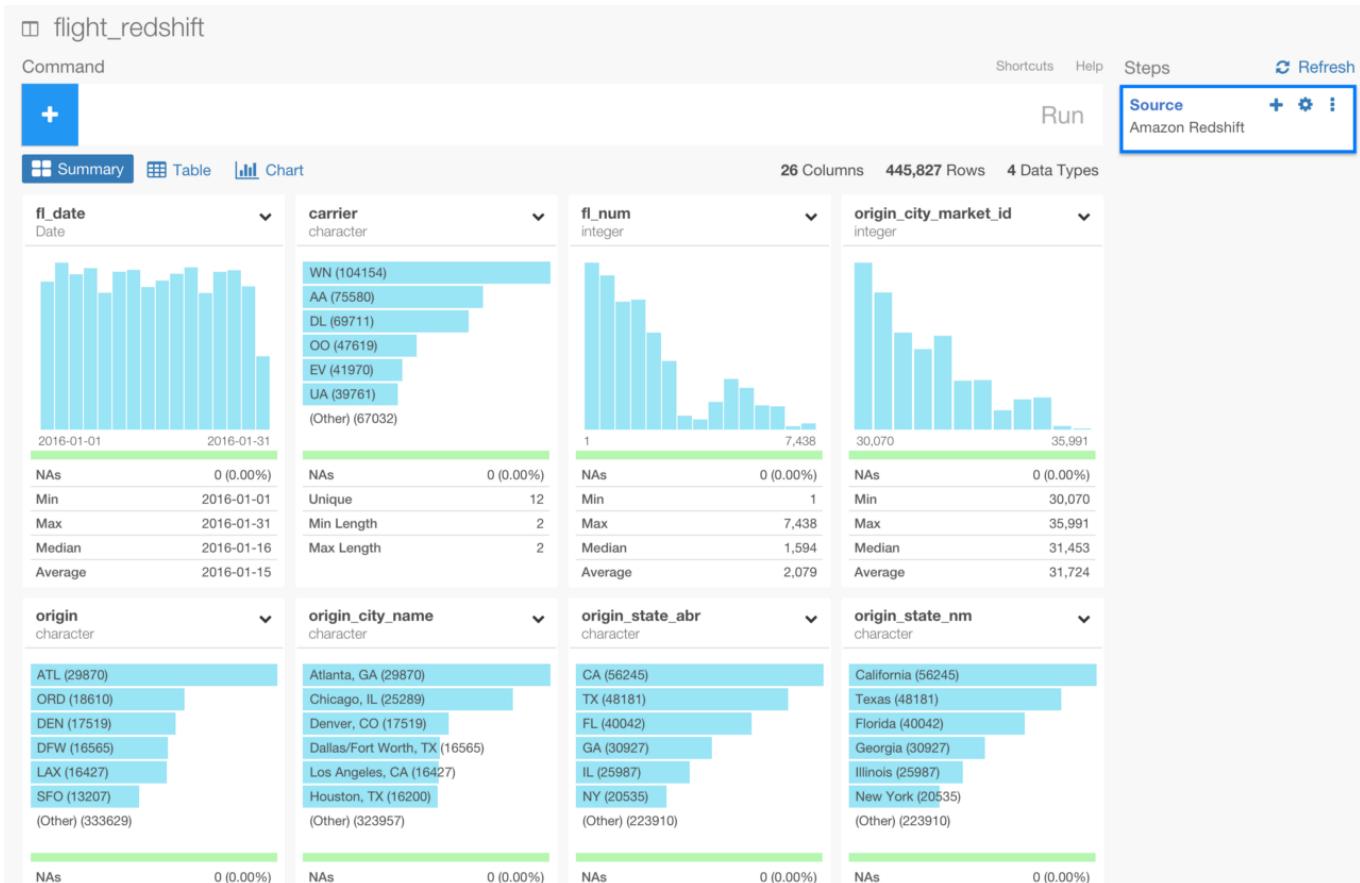
Get Data

	fl_date	carrier	fl_num	origin_city_market_id	origin	origin_city_name	origin_state_abr	o
1	2016-01-01	EV	2730	30194	DFW	Dallas/Fort Worth, TX	TX	T
2	2016-01-01	EV	2730	34109	PIB	Hattiesburg/Laurel, MS	MS	M
3	2016-01-01	EV	2771	30194	DFW	Dallas/Fort Worth, TX	TX	T
4	2016-01-01	EV	2749	30747	BRO	Brownsville, TX	TX	T
5	2016-01-01	EV	2750	30194	DFW	Dallas/Fort Worth, TX	TX	T
6	2016-01-01	EV	2751	30194	DFW	Dallas/Fort Worth, TX	TX	T

Showing first 100 of 445,827 rows

Update **Cancel**

After confirming that the data looks ok, click on ‘Save’ button to import. Then you will get this nice Summary view of your data from Redshift quickly!



Unique Min Length	294 3	Unique Min Length	290 8	Unique Min Length	52 2	Unique Min Length	52 4
----------------------	----------	----------------------	----------	----------------------	---------	----------------------	---------

Random Sample Data

Instead of filtering or aggregating the data, you might want to take a random sample of the data that would be reasonable size for your data analysis.

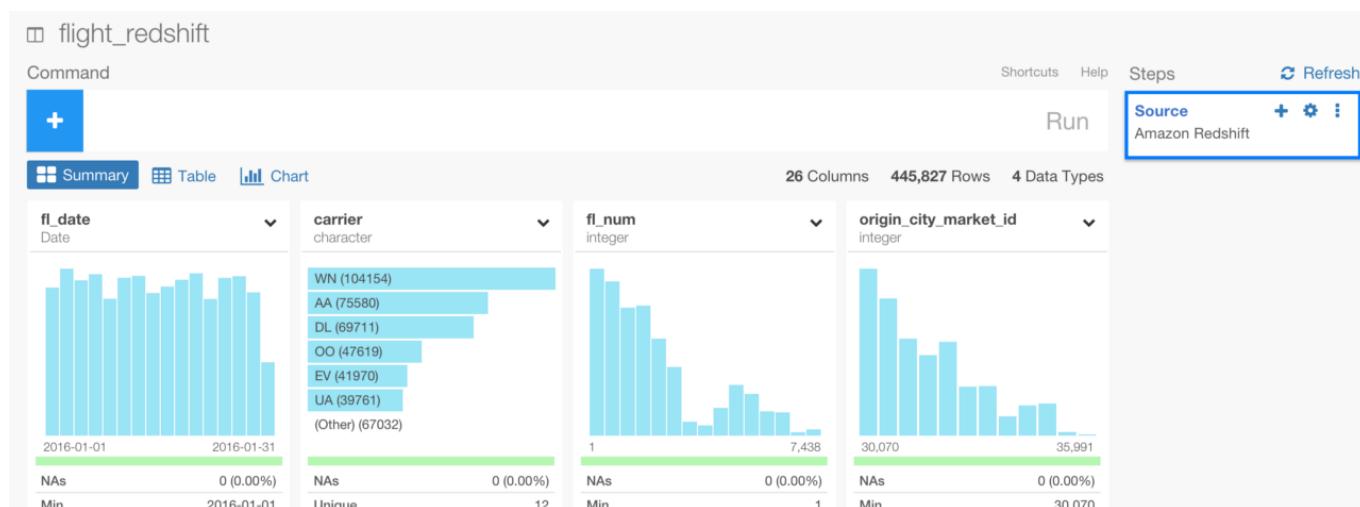
You can use ‘md5’ function, which returns hash code for a given text. Basically this function generates hash codes generated based on the actual data values so you can use it in ORDER BY clause to sort the data based on this hash code and limit to the first n numbers of rows, which would essentially return a random sample of the data while it would return a same set of the data every time you run the query.

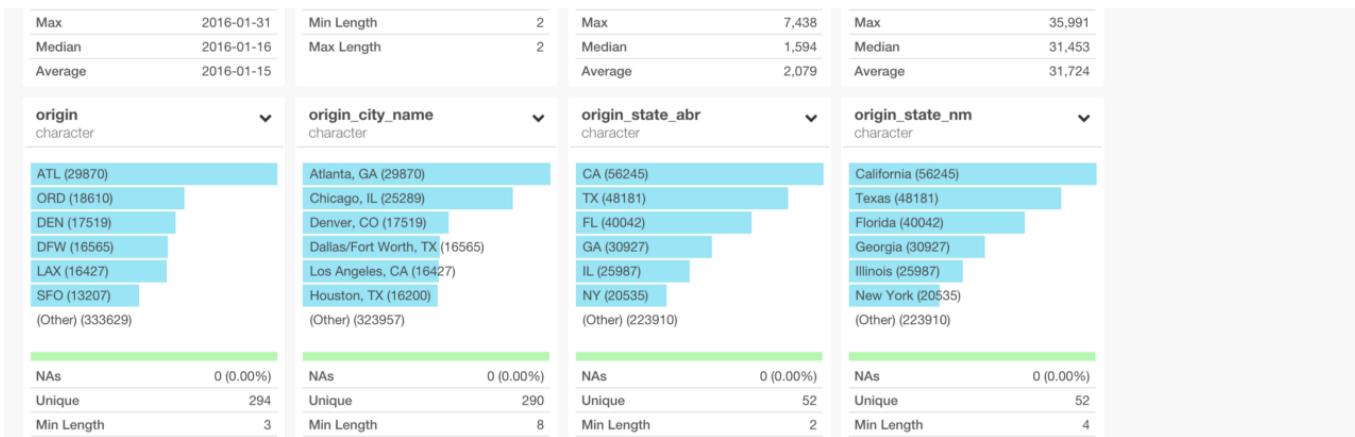
```
SELECT *
  FROM airline_2016_01
 ORDER BY md5('randomSeed' || flight_num)
 LIMIT 100000
```

Thanks to Sébastien for giving us a great guidance for how to effectively get the sample data from Redshift!

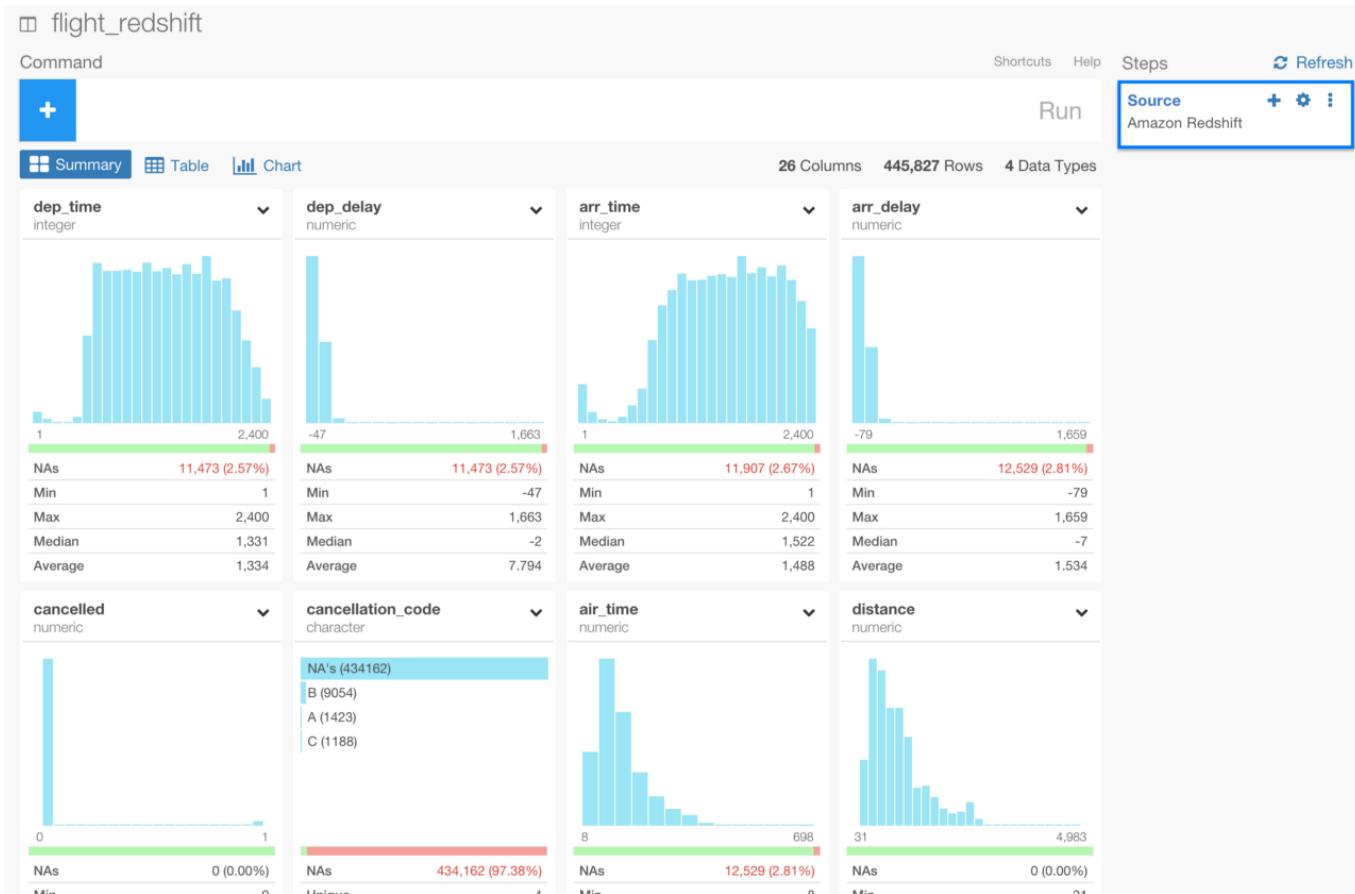
Exploratory Data Analysis

Just by importing the data you can quickly see the summary information for each column. For example, with this flight delay data, we can see ‘carrier’ column is ‘character’ data type and the top 6 carrier names along with the number of the rows for each carrier, and there are 12 carrier names in this data set.



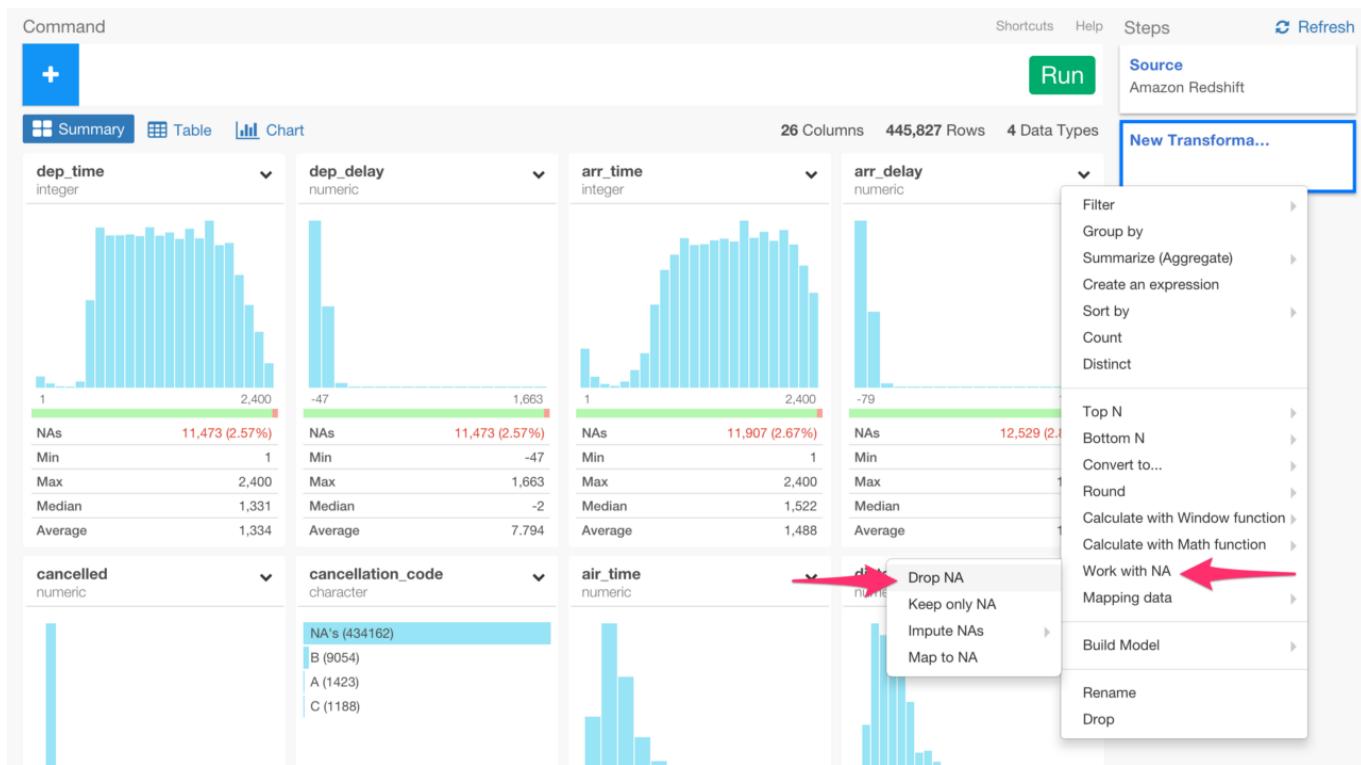


When I scroll down a bit, I can see other columns like ‘arr_delay’, which is ‘numeric’ data type, 2.81% of the data is NA. The average value is 1,534, but the median value is -7. When you look at the histogram you can see most of the data is skewed towards to the smaller values, and this is why ‘median’ value is more useful than ‘average’ value in this kind of scenario.



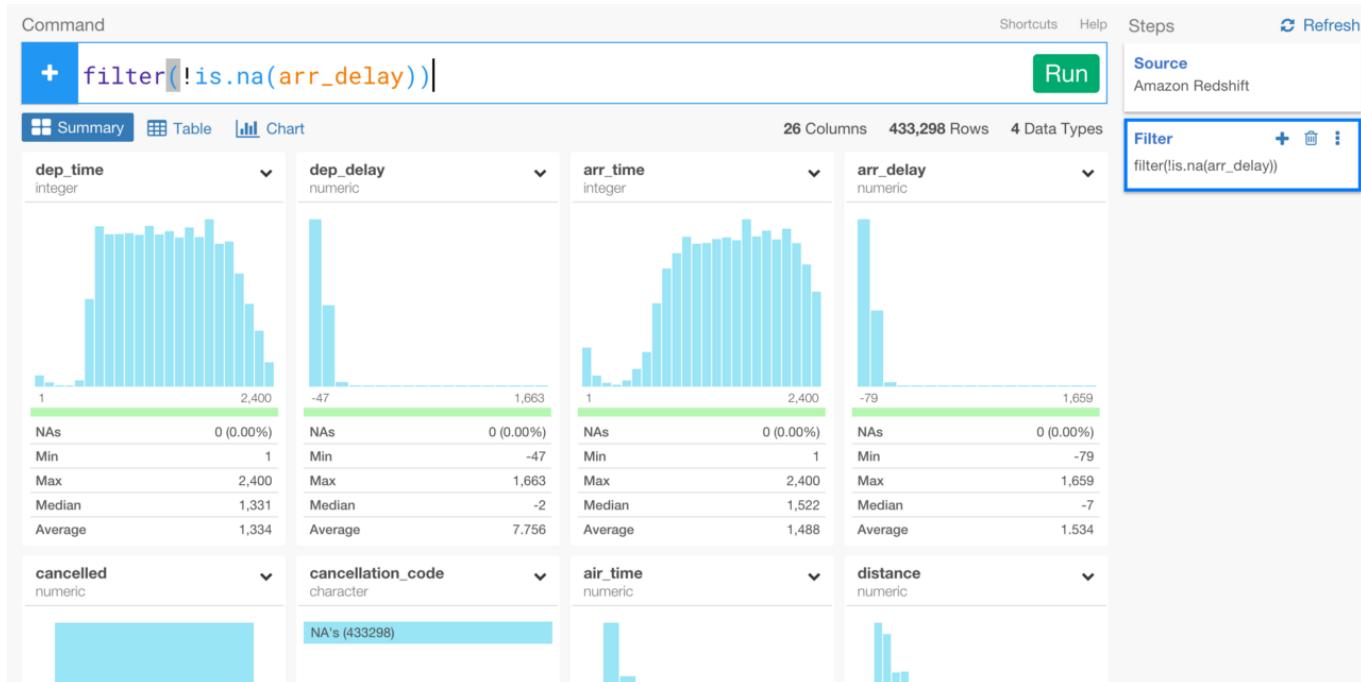
Remove NA

You can remove NA values for ‘arr_delay’ column by selecting ‘Drop NA’ under ‘Work with NA’ menu.



This will build a dplyr command automatically, I can click on Run button to execute.

```
filter(!is.na(arr_delay))
```



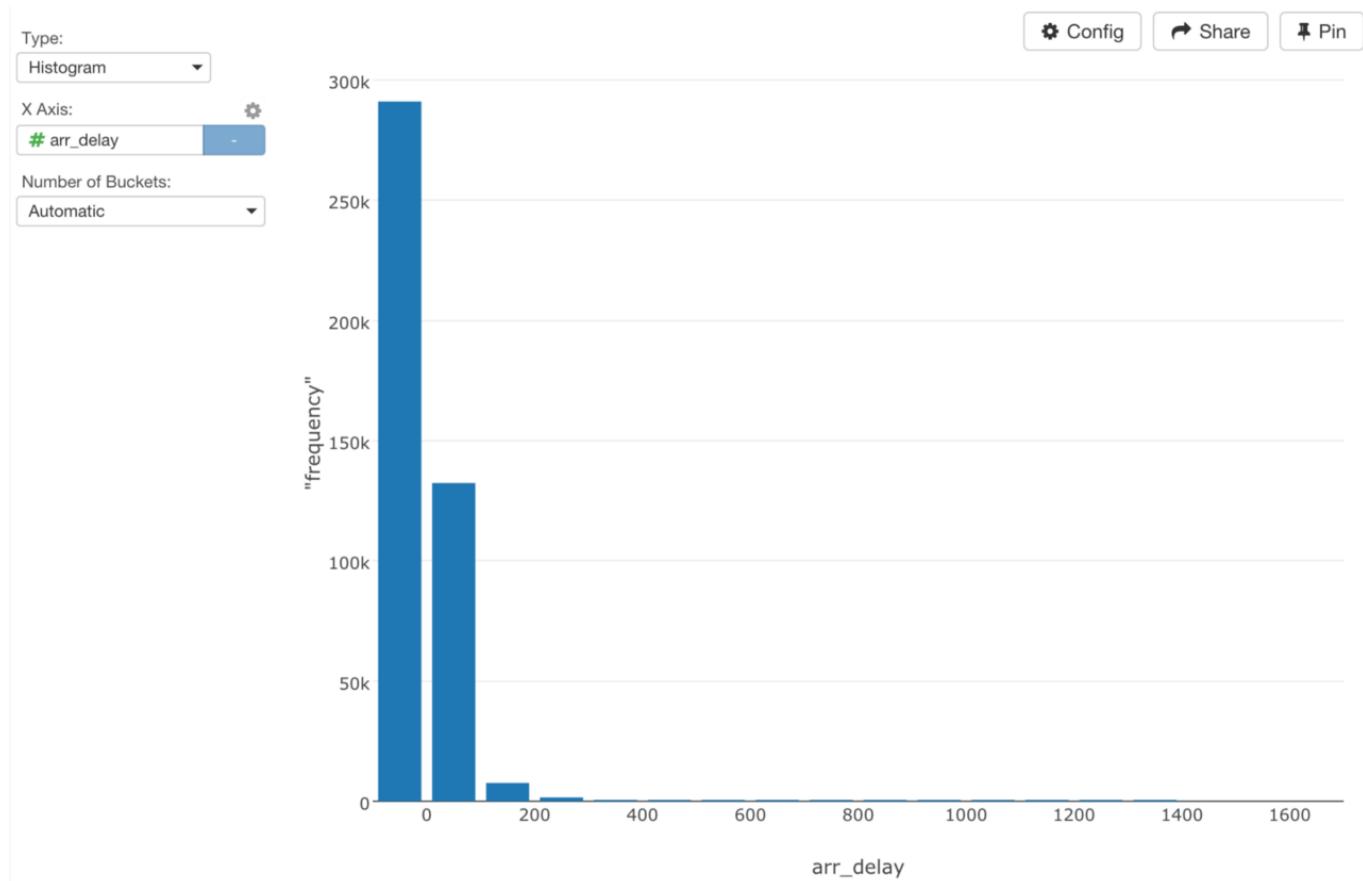
In case you are not familiar with R, the ‘!’ (exclamation mark) is to reverse the effect of the function right after it. ‘is.na(arr_delay)’ would return TRUE if a given value is NA, but combined with ‘!’ mark, it would return TRUE when the value is **not** NA. So ‘filter’ command would keep only the rows whose ‘arr_delay’ is **not** NA.

Visual Exploration

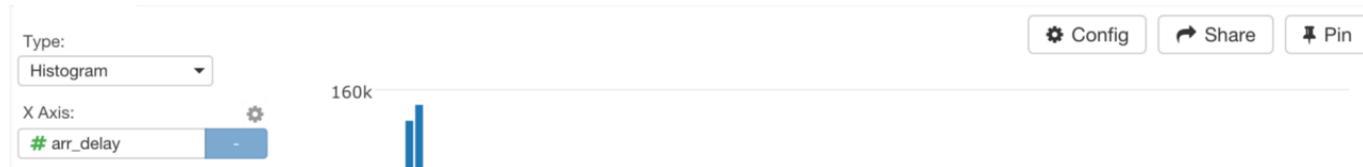
You can use Chart view to explore the data further by visualizing it.

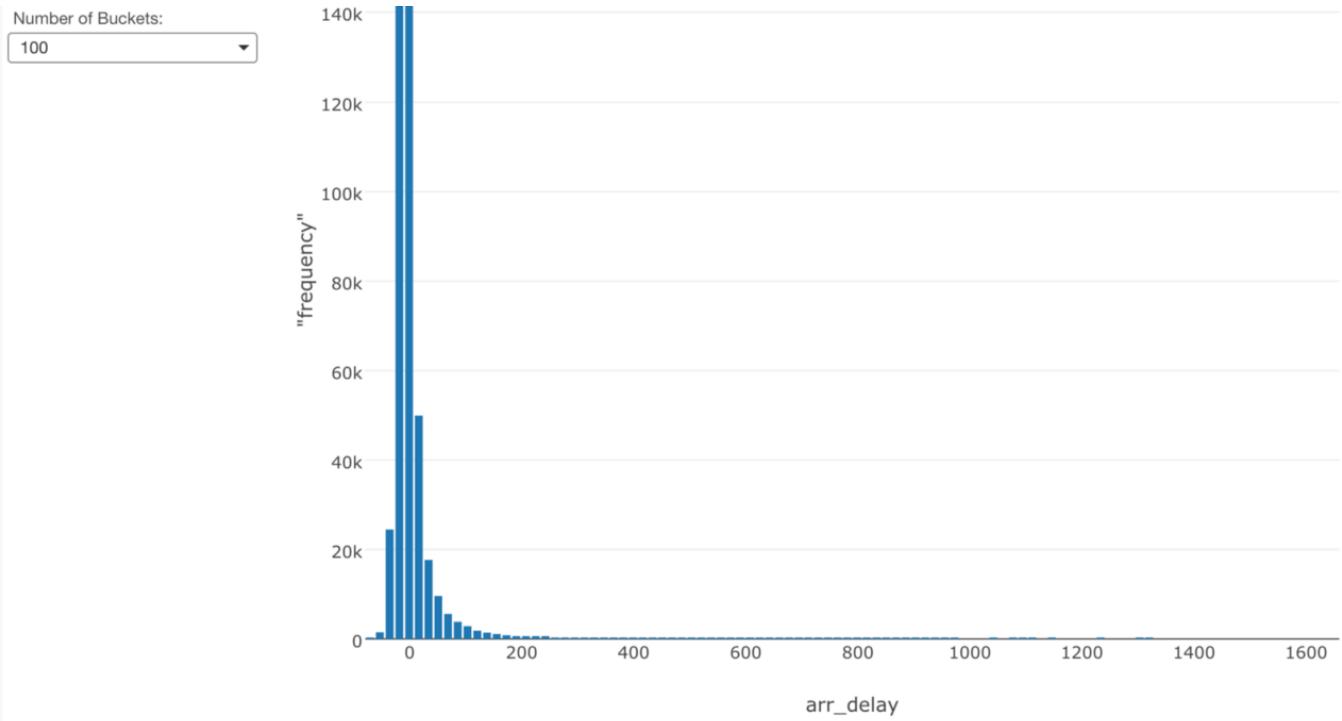
Histogram

By assigning ‘arr_delay’ column to X-Axis for Histogram chart type, we can see how the values for arrival delay time are distributed.

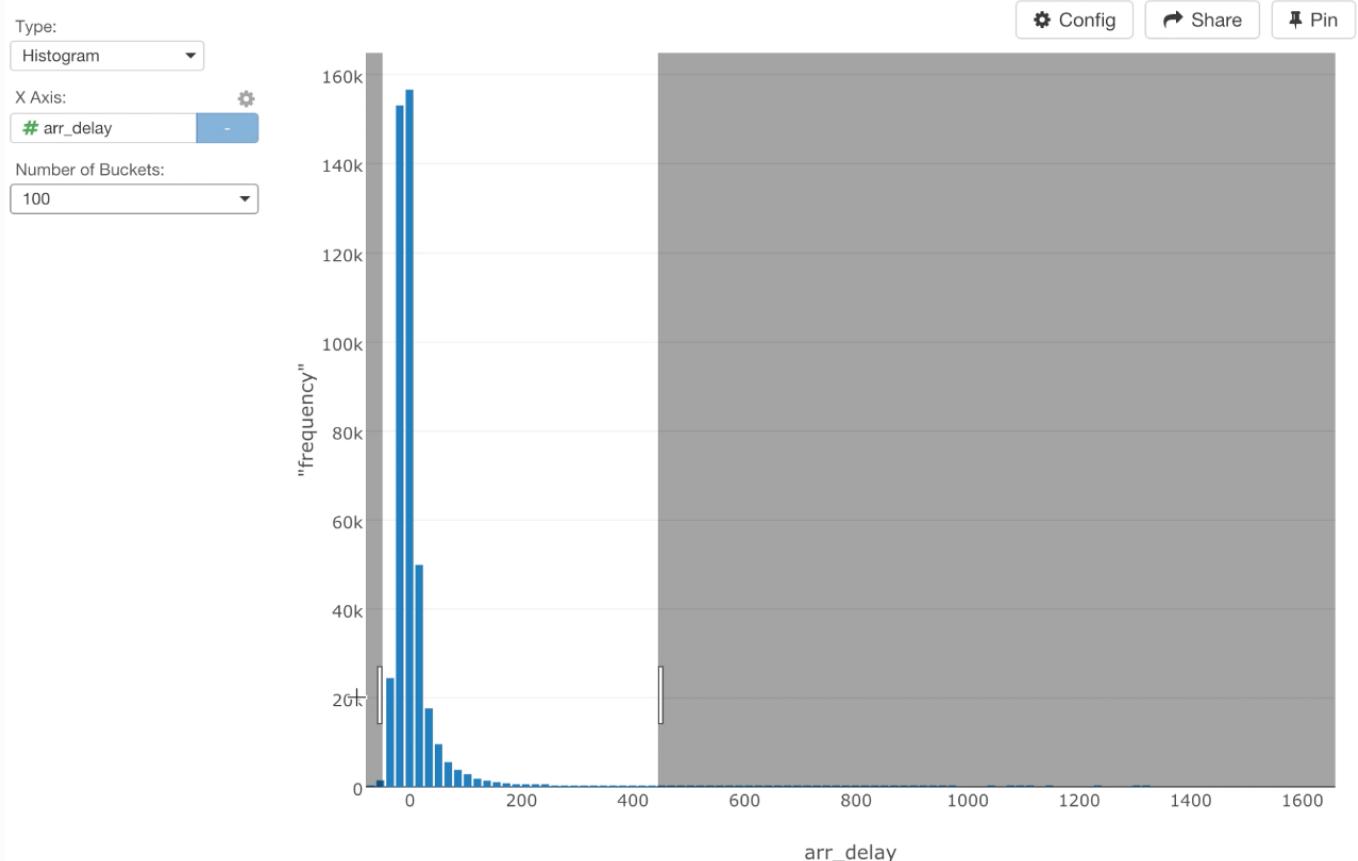


As you can see, the data is skewed highly towards to the smaller values. you can change “Number of Buckets” to 100 to see more detail level distribution.



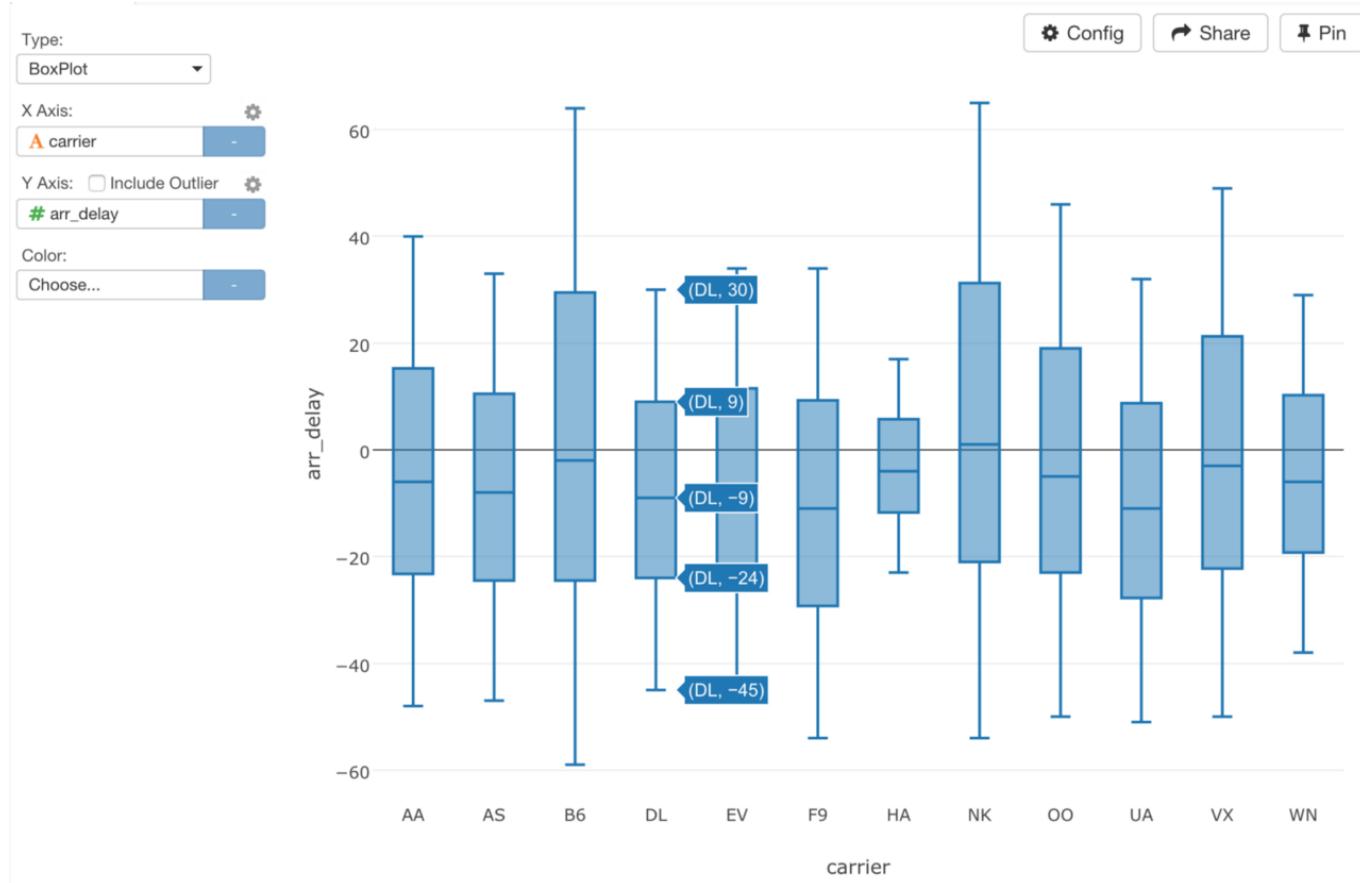


You can always zoom into a certain range of data by dragging the mouse cursor over the area you want to zoom in.



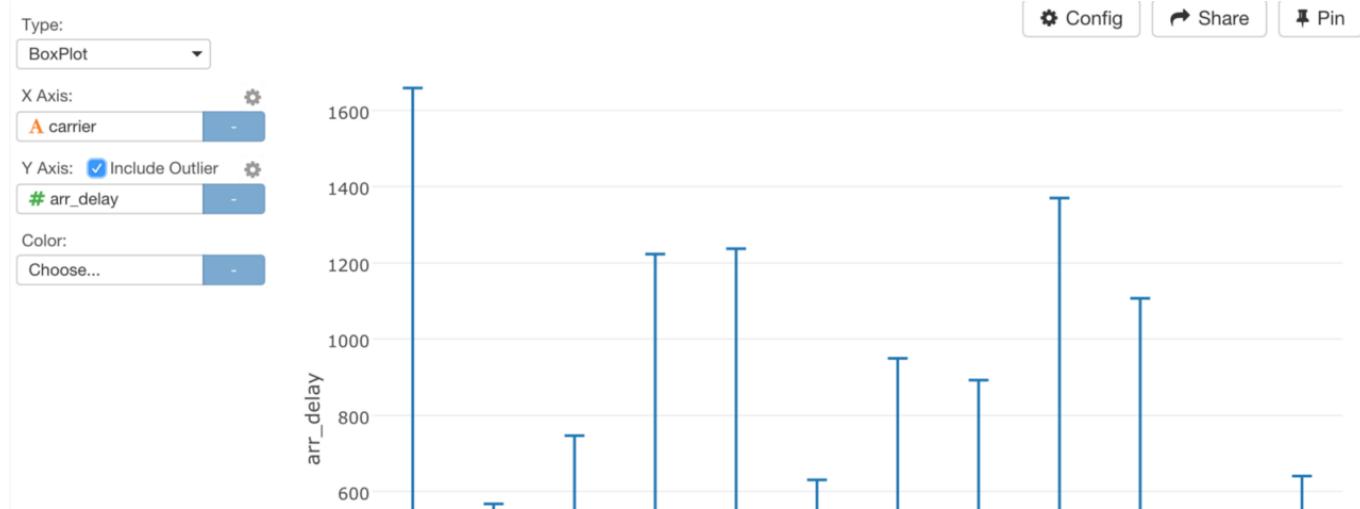
Boxplot

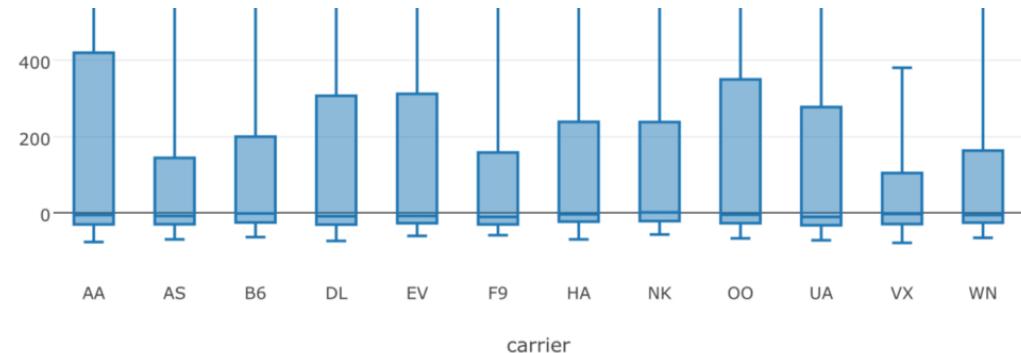
If you want to see how the data is distributed by categorical columns like ‘carrier’, Boxplot chart would be a good help.



It not only shows the data distribution for each carrier visually but also gives you Max, Min, Median, 1st Quartile, and 3rd Quartile values.

It excludes the outlier values as default, but you can include them by clicking on ‘Include Outlier’ checkbox if you like.





Above Average — Filter with Aggregate Function

Let's say you want to see only the flights whose arrival delay times are worse than the average. You can go back to Table view (or Summary view) and select 'Filter / greater than' menu from the column header dropdown.

The screenshot shows the Exploratory Data Analysis interface with a table of flight data. The table has 26 columns and 433,298 rows. The 'arr_delay' column is selected, and a context menu is open, showing options like 'Filter', 'Group by', 'Summarize (Aggregate)', etc. The '>=' option is highlighted with a red arrow.

state_abr	dest_state_nm	dep_time	dep_delay	arr_time	arr_delay	cancelled	cancellation_code	air_time
	Mississippi	1216	-9	1341	-15			
	Texas	1538	-2	1710	-19			
	Florida	955	-5	1134	-15			
	Texas	1358	-7	1525	-27			
	Oklahoma	845	-10	945	-7			
	Louisiana	1600	-5	1722	0			
	Tennessee	826	-4	1047	-18			
	Louisiana	1147	-8	1242	-13			
	Texas	1132	-2	1320	7			
	Texas	1036	-4	1213	-3			
	Texas	1007	-12	1051	-26			
	Texas	801	-10	1021	-6			

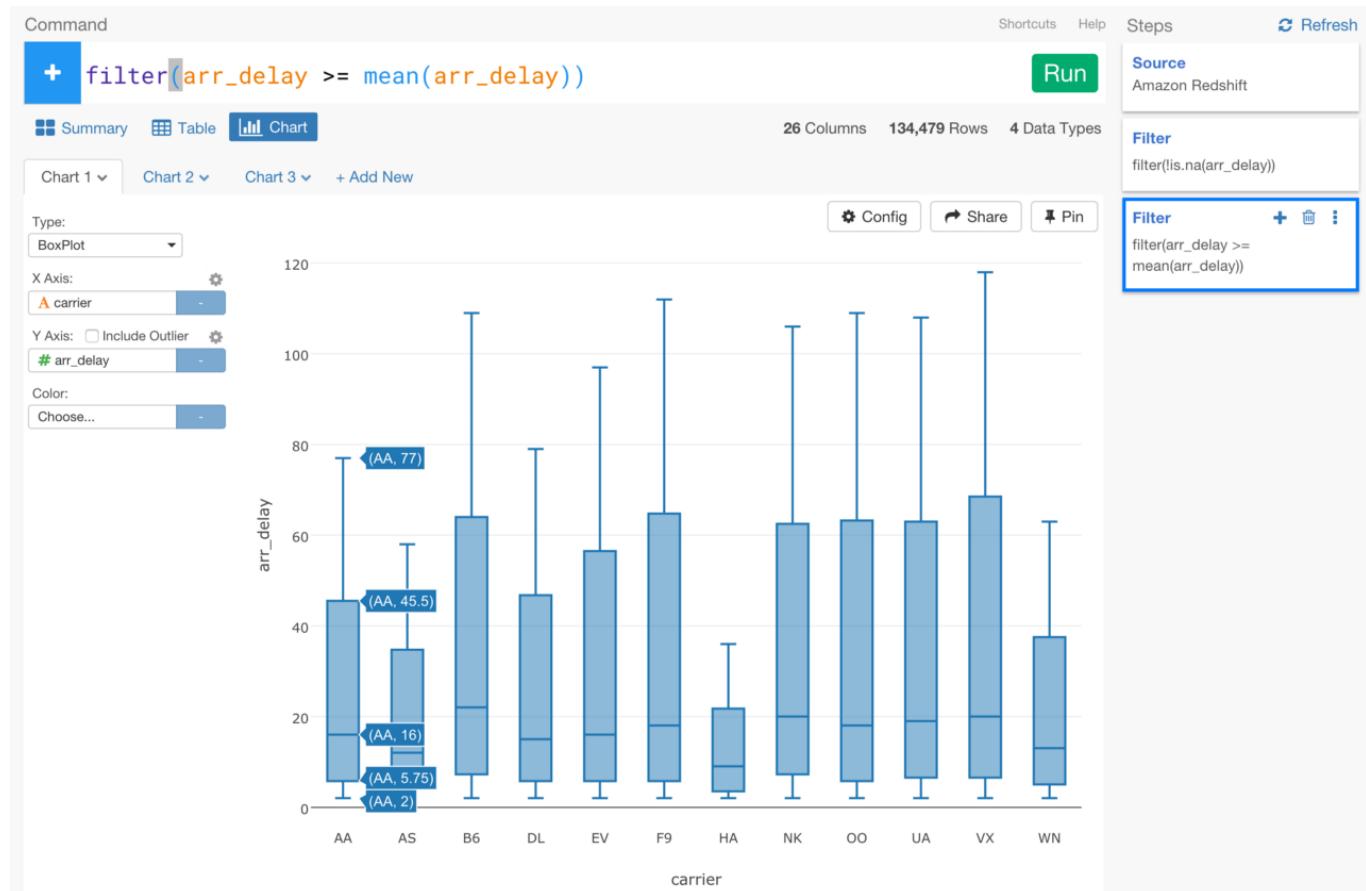
You can type 'mean' and select 'arr_delay' column from the suggested list.

The screenshot shows the Exploratory Data Analysis interface with a table of flight data. The table has 26 columns and 433,298 rows. The 'arr_delay' column is selected, and a context menu is open, showing options like '# air_time', '# arr_delay', '# arr_time', etc. The '# arr_delay' option is highlighted with a red arrow.

state_abr	dest_state_nm	dep_time	dep_delay	arr_time	arr_delay	cancelled	cancellation_code	air_time
	Mississippi	1216	-9	1341	-15			

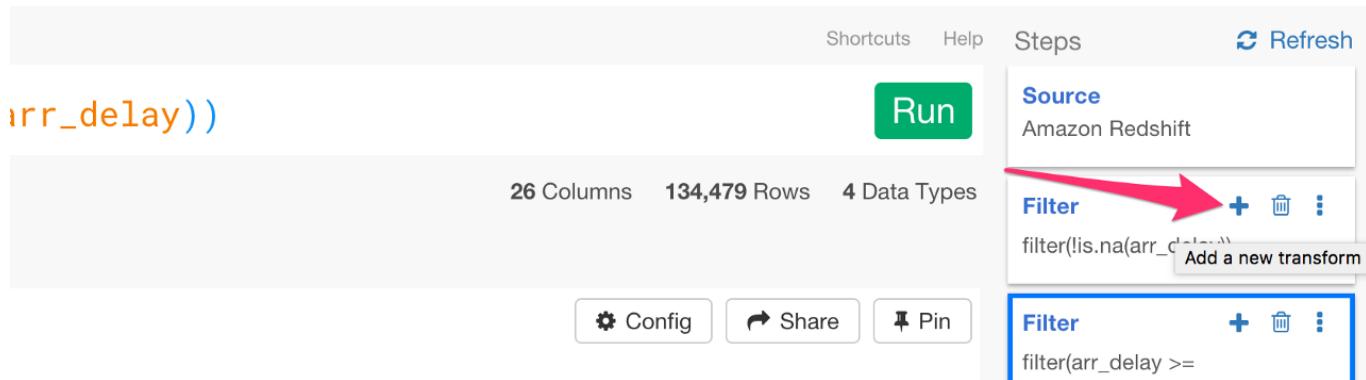
Texas	1538	# arr_delay	79
Florida	955	# dep_time	80

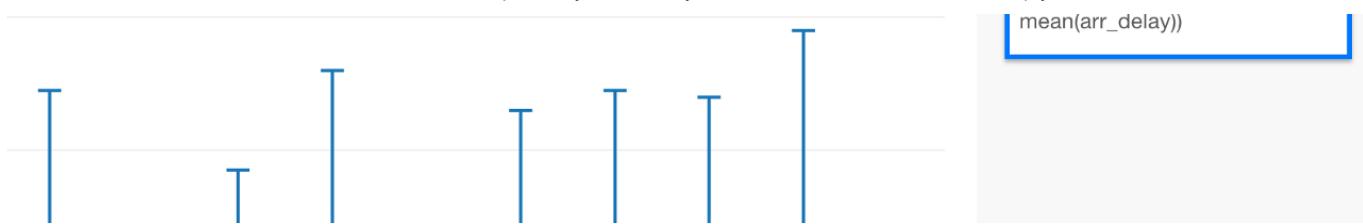
When you go back to Chart view you can see all the legs are starting from the average value of 2.



Keep Only Above Average of Each Carrier, not of All

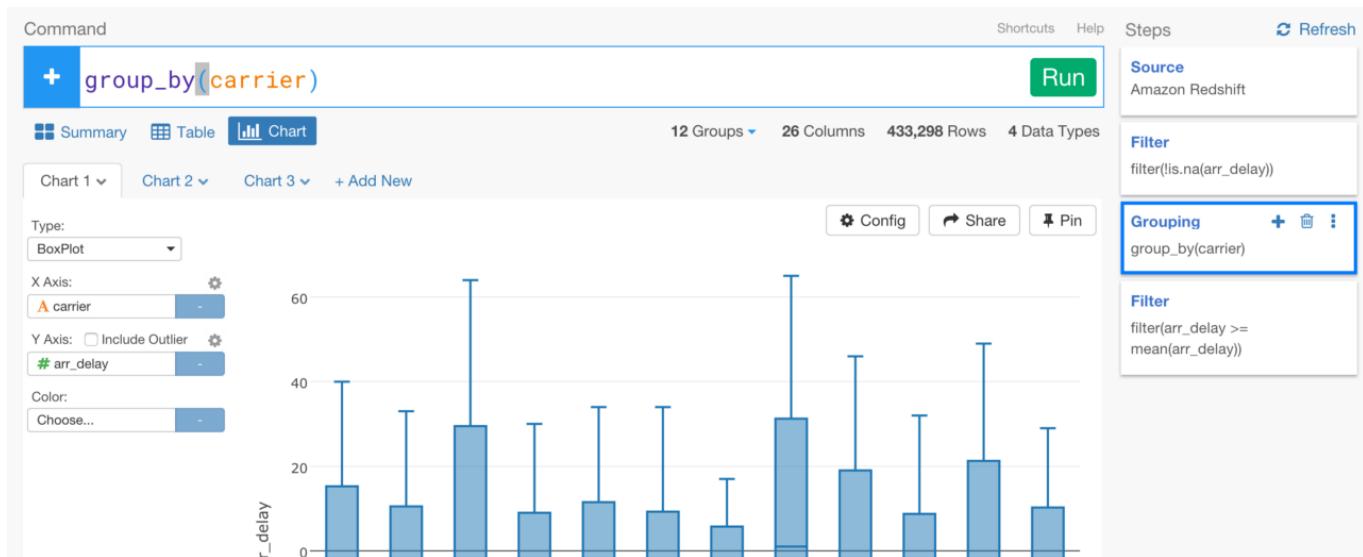
What if you want to filter the data by using average values for each carrier instead of the entire data? It's actually super simple with dplyr. All you need to do is to add a grouping step with 'group_by' command right before the 'Filter' step.



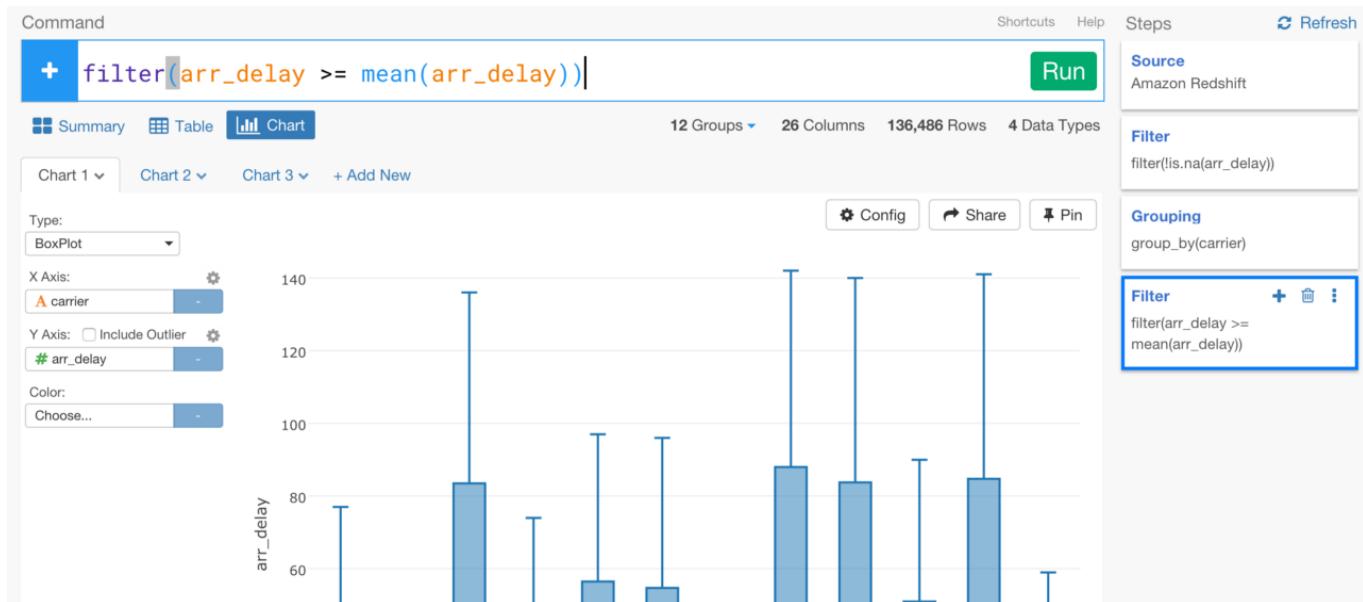


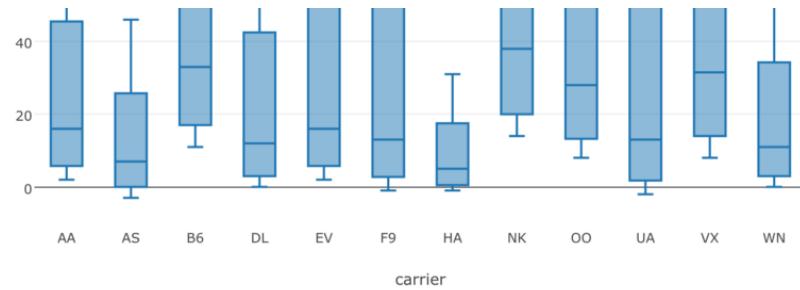
And type the following.

```
group_by(carrier)
```



Now you can go back to the last step and see that each boxplot leg is starting from each carrier's average arrival delay time.





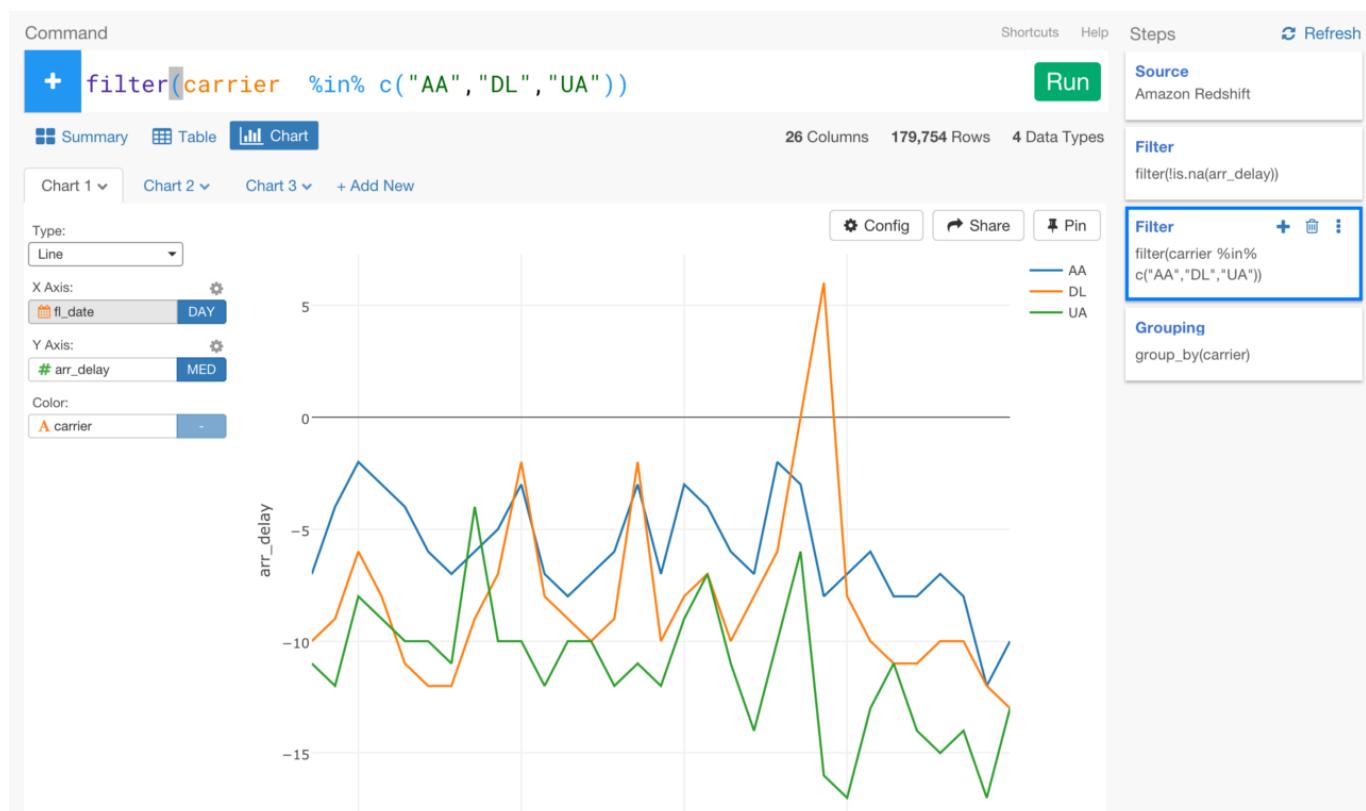
Time Series Analysis

You can find the trend of the arrival delay for each carrier by using Line chart.

I'm adding another Filter step after the first 'Filter' step to keep only 'American Airline (AA)', 'United Airline (UA)', and 'Delta Airline (DL)' to make it simple for this demonstration like below.

```
filter(carrier %in% c("AA", "UA", "DL"))
```

Change the chart type to Line and assign 'fl_date' to X-Axis, 'arr_delay' to Y-Axis, and 'carrier' to Color. Also, you can select 'Day' as the aggregation level for X-Axis and 'Median' as the aggregation function for Y-Axis.



Jan 3
2016 Jan 10 Jan 17 Jan 24 Jan 31
fl_date

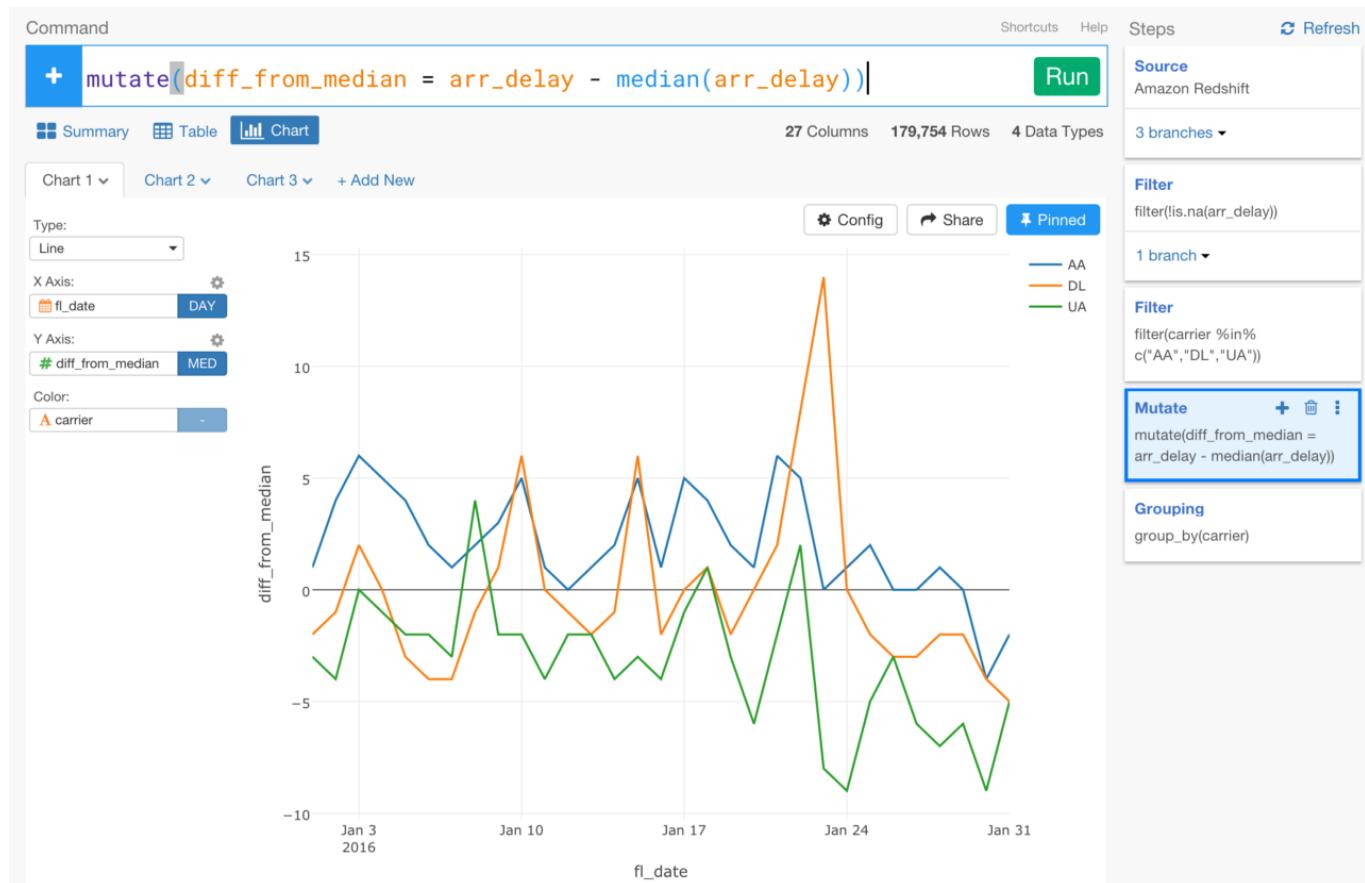
If we look at the median values of each carrier by day, they are mostly negative delay times, which means they are not delayed. But, is this really useful insight? What if all the other carriers also have not delayed? Don't we want to see how these three carriers are doing in a comparison to others?

Well, you can quickly change the baseline. you can use ‘mutate’ command to create a new column with an expression. In this case you can calculate the difference between the arrival delay time and the median of arrival_delay times of all.

```
mutate(diff_from_median = arr_delay - median(arr_delay))
```

Make sure you add this step before ‘Grouping’ step, otherwise the ‘median’ function would do the calculation for each group, which we don’t want for this time.

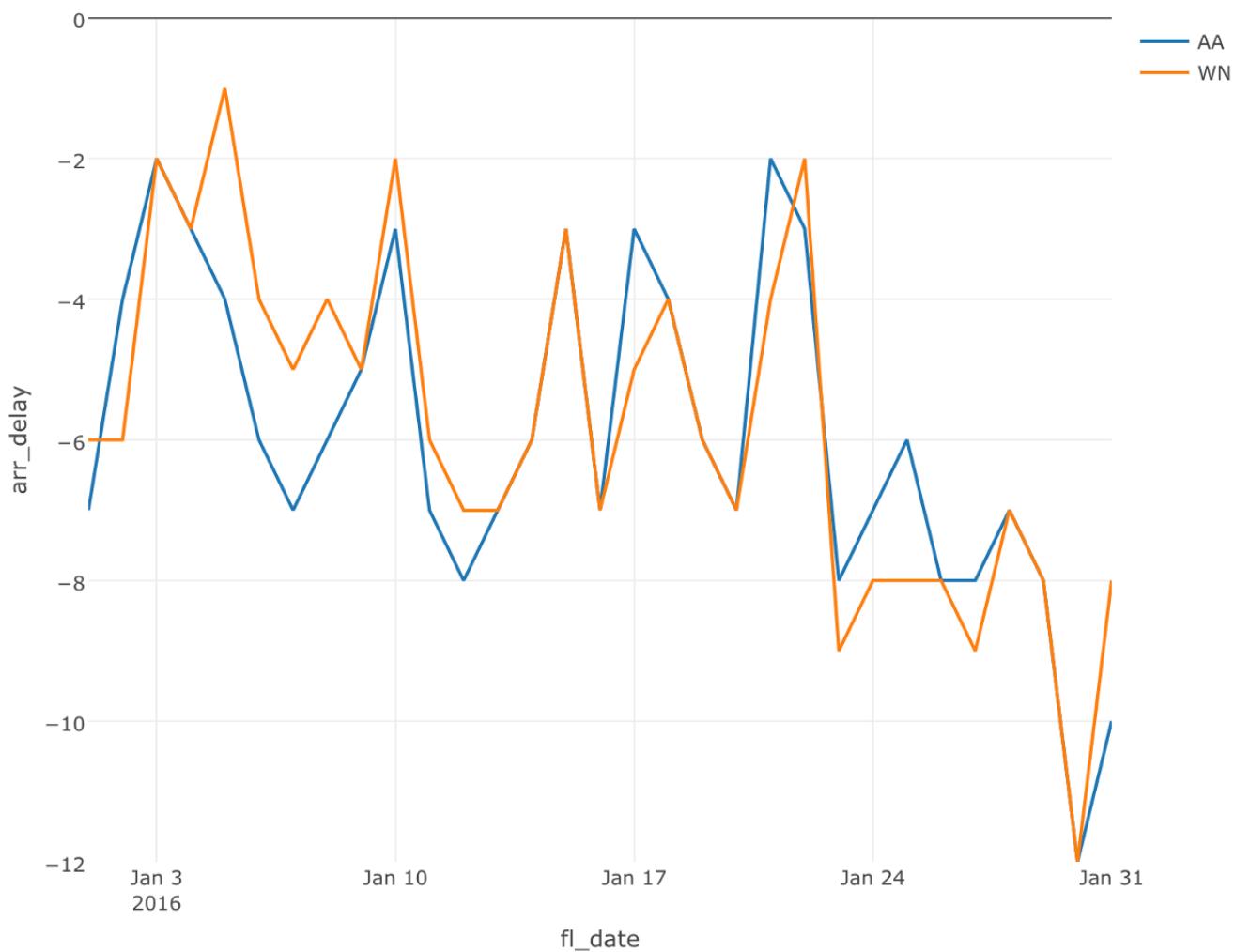
Assign this newly created column ‘diff_from_median’ to Y-Axis.



Now, we can see a slight different view here. For example, American Airline is mostly worse than the overall median arrival delay time while United Air is mostly better than the overall median.

Statistical Data Analysis — Correlation

Let's say we want to find which airline carriers are closely correlated based on the daily arrival delay trend. For example, if we see the trend lines for two carriers like below, then we consider them as highly correlated.



Instead of using our eyes on the visualization to see how the carriers are correlated to one another we can let the machine figure it all out automatically by applying the correlation algorithm.

```
do_cor.kv(carrier, fl_date, arr_delay)
```

The screenshot shows a data analysis interface with a command bar at the top containing the code: `+ do_cor.kv(carrier, fl_date, arr_delay)`. Below the command bar is a table view with three columns: `pair.name.1`, `pair.name.2`, and `cor.value`. The table has 13 rows, including a header row. The data shows correlations between flight carriers (AA, AS, DL, EV, F9, HA, NK, OO, UA, VX, WN) across different flight dates. The highest correlation is between AA and WN (0.3957). The interface also includes a sidebar with 'Source' (Amazon Redshift), 'Filter' (filter(!is.na(arr_delay))), and a 'Steps' panel containing the executed command.

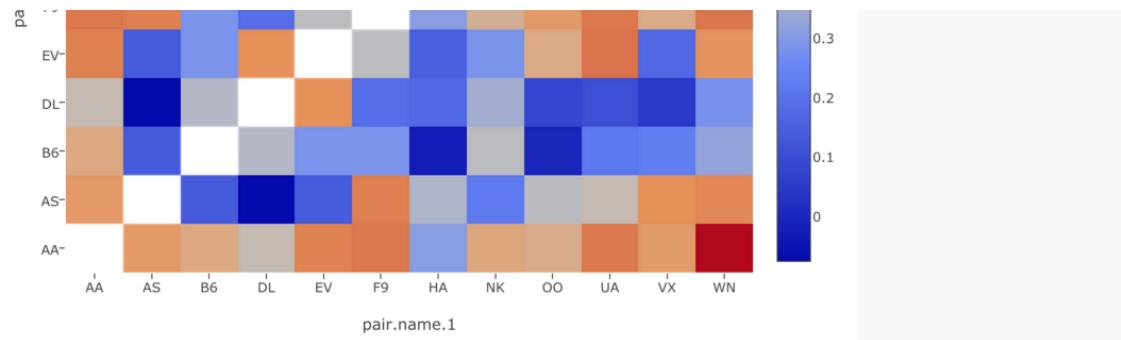
	<code>pair.name.1</code>	<code>pair.name.2</code>	<code>cor.value</code>
1	AA	AS	0.1679
2	AA	B6	0.5584
3	AA	DL	0.5661
4	AA	EV	0.2746
5	AA	F9	0.2371
6	AA	HA	0.0453
7	AA	NK	0.3962
8	AA	OO	0.2276
9	AA	UA	0.1974
10	AA	VX	0.2744
11	AA	WN	0.3957
12	AS	AA	0.1679

This ‘do_cor.kv’ function calculates the correlation for every single pair of the carriers based on how the arrival delay values (arr_delay) are for each flight date (fl_date). It aggregates the values at carrier and flight date level by using ‘mean’ function as default, but you can change this by using ‘fun.aggregate’ function. For example, I’m setting it to ‘median’ below.

```
do_cor.kv(carrier, fl_date, arr_delay, fun.aggregate=median)
```

We can go to Chart view and visualize the result to see the correlation among the carriers better. The darker the red color is the higher they are correlated, such as AA and WN.





When you click inside the function you can always see what the function does and how it works. Or, you can always see the reference document for all the functions.

Command Shortcuts Help

+ do_cor.kv(carrier, fl_date, arr_delay, fun.aggregate=median) Run

do_cor.kv(*subject_column*, *key_column*, *value_column*, *use* = *cor_na_operation*, *method* = *cor_method*, *distinct*=*logical*, *diag*=*logical*)

Chart

Type: Heatmap

X Axis: A pair.

Y Axis: A pair.

Color: # cor.

Package: exploratory

Summary

Calculates correlations for all the pairs of subject columns.

Syntax

```
do_cor.kv(<subject_column>, <key_column>, <value_column>, use = <cor_na_operation>, method = <cor_method>, distinct=<logical>, diag=<logical>)
```

Arguments

- *use* (Optional) - The default is "pairwise.complete.obs". Set an operation type for dealing with missing values.
- *method* (Optional) - The default is "pearson". Set a method to compute correlation coefficient among "pearson", "kendall", or "spearman".
- *distinct* (Optional) - The default is FALSE. Whether the pair of output should be unique. If this is TRUE, a pair appears only once but if it's FALSE, a pair appears twice in swapped order. If you want to filter the pairs by names, it's better to be FALSE.
- *diag* (Optional) - The default is FALSE. Whether the output should contain the similarity of documents with itself.

As you can see there are other arguments you can try it out. For example, if you don't want to see the duplicated pairs, you can set 'distinct' argument to TRUE.

```
do_cor.kv(carrier, fl_date, arr_delay, fun.aggregate=median,
distinct=TRUE)
```

Command Shortcuts Help Steps Refresh

+ r.kv(carrier, fl_date, arr_delay, fun.aggregate=median, **distinct=TRUE**) Run

Summary **Table** **Chart**

Chart 1 ▾ Chart 2 ▾ Chart 3 ▾ + Add New

3 Columns 66 Rows 2 Data Types

Type: Heatmap

X Axis: WN-

Config Share Pinned

Source

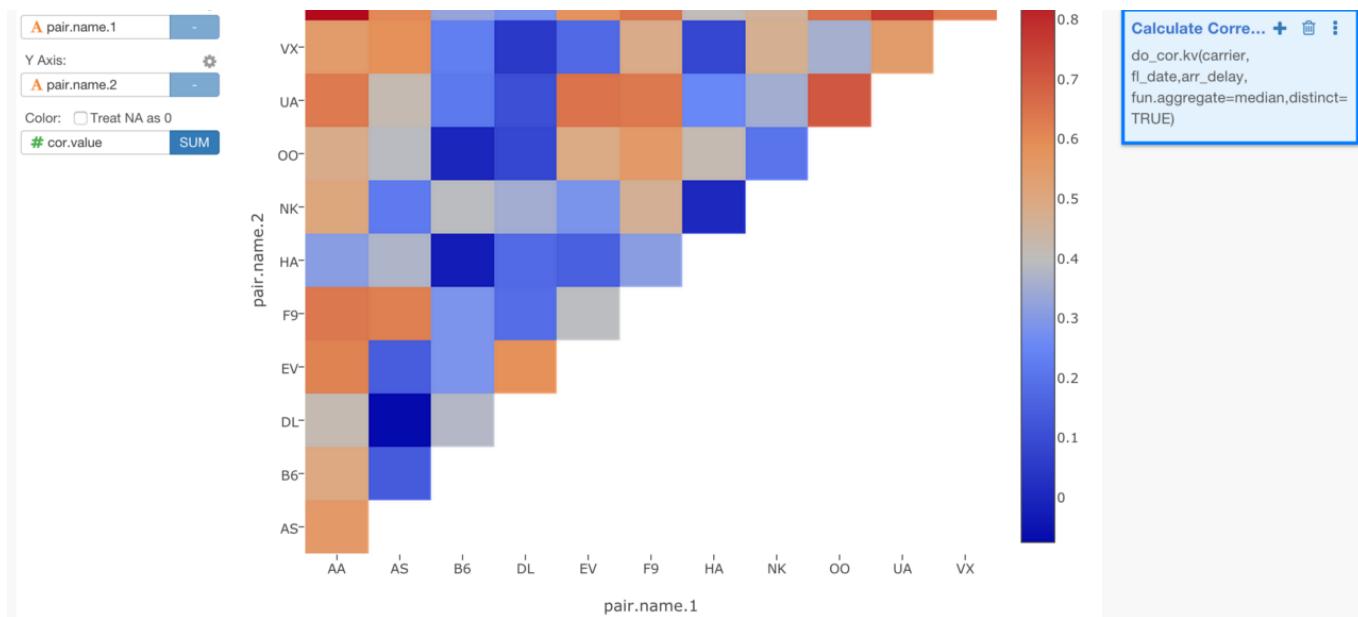
Amazon Redshift

1 branch ▾

Filter

filter(is.na(arr_delay))

1 branch ▾



As you have seen, with Exploratory you can quickly bring your data in Amazon Redshift database into R and start exploring the data interactively and iteratively with various visualization types and with so many R's algorithms.

If you would like to try it out quickly you can sign up for Exploratory Desktop beta from here for free!

• • •

Amazon Redshift support could not have been done without helps from the community. Special thanks to Sébastien for his advise on the random sampling and variable support!

• • •

R packages used in this post

hadley/dplyr

dplyr - Dplyr: A grammar of data manipulation

[github.com](https://github.com/hadley/dplyr)

exploratory-io/exploratory_func

exploratory_func - R functions for Exploratory

github.com

Thanks to hide kojima, Kei Saito, and Hideaki Hayashi.

Data Science Big Data Data Visualization Redshift

About Help Legal

Get the Medium app

