Capital bike share data for quarter 1, year 2014

[System Data | Capital Bikeshare](https://ride.capitalbikeshare.com/system-data)

**System Data**

Where do Capital Bikeshare riders go? When do they ride? How far do they go? Which stations are most popular? What days of the week are most rides taken on? We've heard all of these questions since we launched in 2010, and we're glad to provide the data that shows you the answers from our first trip to today.

## Trip History Data

Each quarter, we publish [downloadable files](https://s3.amazonaws.com/capitalbikeshare-data/index.html) of Capital Bikeshare trip data. The data includes:

* Duration – Duration of trip
* Start\_Date – Includes start date and time
* End\_Date – Includes end date and time
* Start\_Station\_number – Includes starting station number
* Start\_Station – Includes starting station name
* End\_Station\_number – Includes ending station number
* End Station – Includes ending station name
* Bike Number – Includes ID number of bike used for the trip
* Member Type – Indicates whether user was a "registered" member (Annual Member, 30-Day Member or Day Key Member) or a "casual" rider (Single Trip, 24-Hour Pass, 3-Day Pass or 5-Day Pass)

This data has been processed to remove trips that are taken by staff as they service and inspect the system, trips that are taken to/from any of our “test” stations at our warehouses and any trips lasting less than 60 seconds (potentially false starts or users trying to re-dock a bike to ensure it's secure).

Data source: [Index of bucket "capitalbikeshare-data"](https://s3.amazonaws.com/capitalbikeshare-data/index.html)

Queries:

Finding total rows

SELECT count(\*) FROM 2014q1\_capitalbikeshare

|  |
| --- |
| count |
| 401122 |

Finding riders count based on membership type

SELECT member\_type, count(\*) as count

FROM 2014q1\_capitalbikeshare

group by member\_type

|  |  |
| --- | --- |
| member\_type | count |
| Member | 355755 |
| Casual | 45367 |

Finding riders count based on station number

SELECT start\_station\_number, count(\*) as count

FROM 2014q1\_capitalbikeshare

group by start\_station\_number

order by count DESC

Table

Description automatically generated

Find average duration travelled based on member type:

SELECT member\_type, AVG(duration) as average\_duration

FROM 2014q1\_capitalbikeshare

group by member\_type

|  |  |
| --- | --- |
| member\_type | average\_duration |
| Member | 629.7344 |
| Casual | 2216.3460 |

A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.

The most practical example of this is a running total:

SELECT duration,

SUM(duration) OVER (ORDER BY start\_date) AS running\_total

FROM 2014q1\_capitalbikeshare

Table

Description automatically generated

If you'd like to narrow the window from the entire dataset to individual groups within the dataset, you can use PARTITION BY to do so

SELECT start\_station,

duration,

SUM(duration) OVER

(PARTITION BY start\_station ORDER BY start\_date)

AS running\_total

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-31'

Table

Description automatically generated

Partition without order by, will end up showing the final running total for all entries of the same station

SELECT start\_station,

duration,

SUM(duration) OVER

(PARTITION BY start\_station)

AS running\_total

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-31'

Table

Description automatically generated

Write a query modification of the above example query that shows the duration of each ride as a percentage of the total time accrued by riders from each start\_terminal

SELECT start\_station,

duration,

SUM(duration) OVER (PARTITION BY start\_station) AS start\_terminal\_sum,

(duration/sum(duration) OVER (PARTITION BY start\_station))\*100 as percent\_of\_total\_time

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-31'

ORDER BY 1, 4 DESC

Table

Description automatically generated

## **The usual suspects: SUM, COUNT, and AVG**

SELECT start\_station,

duration,

SUM(duration) OVER (PARTITION BY start\_station) AS running\_sum,

count(duration) OVER (PARTITION BY start\_station) AS running\_count,

AVG(duration) OVER (PARTITION BY start\_station) AS running\_avg

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-31'

Table

Description automatically generated

Including the order by

SELECT start\_station,

duration,

SUM(duration) OVER (PARTITION BY start\_station ORDER BY start\_date) AS running\_sum,

count(duration) OVER (PARTITION BY start\_station ORDER BY start\_date) AS running\_count,

AVG(duration) OVER (PARTITION BY start\_station ORDER BY start\_date) AS running\_avg

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-31'

Table

Description automatically generated

By End station

SELECT end\_station,

duration,

SUM(duration) OVER (PARTITION BY start\_station ORDER BY start\_date) AS running\_total

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-31'

Table

Description automatically generated

## **ROW\_NUMBER()**

ROW\_NUMBER() does just what it sounds like—displays the number of a given row. It starts are 1 and numbers the rows according to the ORDER BY part of the window statement. ROW\_NUMBER() does not require you to specify a variable within the parentheses:

SELECT start\_station,

start\_date,

duration,

ROW\_NUMBER() OVER (ORDER BY start\_date desc)

AS row\_number

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

Table

Description automatically generated

SELECT start\_station,

start\_date,

duration,

ROW\_NUMBER() OVER (PARTITION BY start\_station ORDER BY start\_date desc)

AS row\_number

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

Table

Description automatically generated

## **RANK() and DENSE\_RANK()**

RANK() is slightly different from ROW\_NUMBER(). If you order by start\_time, for example, it might be the case that some terminals have rides with two identical start times. In this case, they are given the same rank, whereas ROW\_NUMBER() gives them different numbers.

SELECT start\_station,

duration,

RANK() OVER (PARTITION BY start\_station

ORDER BY start\_date)

AS rank

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

Table

Description automatically generated

You can also use DENSE\_RANK() instead of RANK() depending on your application. Imagine a situation in which three entries have the same value. Using either command, they will all get the same rank. For the sake of this example, let's say it's "2." Here's how the two commands would evaluate the next results differently:

* RANK() would give the identical rows a rank of 2, then skip ranks 3 and 4, so the next result would be 5
* DENSE\_RANK() would still give all the identical rows a rank of 2, but the following row would be 3—no ranks would be skipped.

SELECT start\_station,

duration,

DENSE\_RANK() OVER (PARTITION BY start\_station

ORDER BY start\_date)

AS rank

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

## **NTILE**

You can use window functions to identify what percentile (or quartile, or any other subdivision) a given row falls into. The syntax is NTILE(\*# of buckets\*). In this case, ORDER BY determines which column to use to determine the quartiles (or whatever number of 'tiles you specify)

SELECT start\_station,

duration,

NTILE(4) OVER

(PARTITION BY start\_station ORDER BY duration)

AS quartile,

NTILE(5) OVER

(PARTITION BY start\_station ORDER BY duration)

AS quintile,

NTILE(100) OVER

(PARTITION BY start\_station ORDER BY duration)

AS percentile

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

ORDER BY start\_station, duration

Table

Description automatically generated

Write a query that shows only the duration of the trip and the percentile into which that duration falls (across the entire dataset—not partitioned by terminal).

SELECT duration,

NTILE(100) OVER

(ORDER BY duration)

AS percentile

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

ORDER BY duration

## **LAG and LEAD**

It can often be useful to compare rows to preceding or following rows, especially if you've got the data in an order that makes sense. You can use LAG or LEAD to create columns that pull values from other rows—all you need to do is enter which column to pull from and how many rows away you'd like to do the pull. LAG pulls from previous rows and LEAD pulls from following rows:

SELECT start\_station,

duration,

LAG(duration, 1) OVER

(PARTITION BY start\_station ORDER BY duration) AS lag,

LEAD(duration, 1) OVER

(PARTITION BY start\_station ORDER BY duration) AS lead

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

ORDER BY start\_station, duration

Table

Description automatically generated

SELECT start\_station,

duration,

duration -LAG(duration, 1) OVER

(PARTITION BY start\_station ORDER BY duration)

AS difference

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

ORDER BY start\_station, duration

Table

Description automatically generated

The first row of the difference column is null because there is no previous row from which to pull. Similarly, using LEAD will create nulls at the end of the dataset. If you'd like to make the results a bit cleaner, you can wrap it in an outer query to remove nulls:

SELECT \* from

(SELECT start\_station,

duration,

duration -LAG(duration, 1) OVER

(PARTITION BY start\_station ORDER BY duration)

AS difference

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

ORDER BY start\_station, duration) as sub

where sub.difference is not null

## **Defining a window alias**

If you're planning to write several window functions in to the same query, using the same window, you can create an alias. Take the NTILE example above:

SELECT start\_station,

duration,

NTILE(4) OVER

(PARTITION BY start\_station ORDER BY duration)

AS quartile,

NTILE(5) OVER

(PARTITION BY start\_station ORDER BY duration)

AS quintile,

NTILE(100) OVER

(PARTITION BY start\_station ORDER BY duration)

AS percentile

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

ORDER BY start\_station, duration

With Window alias:

SELECT start\_station,

duration,

NTILE(4) OVER ntile\_window AS quartile,

NTILE(5) OVER ntile\_window AS quintile,

NTILE(100) OVER ntile\_window AS percentile

FROM 2014q1\_capitalbikeshare

WHERE start\_date < '2014-01-08'

WINDOW ntile\_window AS

(PARTITION BY start\_station ORDER BY duration)

ORDER BY start\_station, duration

The WINDOW clause, if included, should always come after the WHERE clause