Assignment 3: Windowing

By Tony Fraser

2023-09-22

Assignment:

Use window functions to create partial dataframe groupings. Attach those as columns on the right.

A tidyverse toy example

First we'll create a toy time series dataframe. This dataframe simulates a person standing in front of the strip mall for two months asking people if their favorite color is red, yellow or blue. After we have the sample data, well use the use the cumulative sum `cumsum` and `lag` functions to create extra column.

```
library(tidyverse)
fav_colors <- data.frame(
  day = rep(seq(as.Date("2013-05-05"), by = "day", length.out = 60), each = 3),
  color = rep(c("red", "blue", "yellow"), 60),
  count = base::sample(1:10, size = 180, replace = TRUE))   # 180 -> 30 days x 3 colors)
head(fav_colors)
```

```
         day  color count
1 2013-05-05    red     1
2 2013-05-05   blue    10
3 2013-05-05 yellow     1
4 2013-05-06    red     4
5 2013-05-06   blue     3
6 2013-05-06 yellow    10
```

```r
fav_with_weekly_total <- fav_colors |>
  arrange(day, color) %>%
  group_by(color) %>%
  mutate(weekly_total = cumsum(count) - lag(cumsum(count), 7, default = 0)) |>
  arrange(day, color)
head(fav_with_weekly_total)
```

```
# A tibble: 6 x 4
# Groups:   color [3]
  day        color  count weekly_total
  <date>     <chr>  <int>        <int>
1 2013-05-05 blue      10           10
2 2013-05-05 red        1            1
3 2013-05-05 yellow     1            1
4 2013-05-06 blue       3           13
5 2013-05-06 red        4            5
6 2013-05-06 yellow    10           11
```

2022 NYC Traffic fatalities by week, a real world example using SparkR

We can use NYC's traffic dataset again[1] to demonstrate window over with real world data. First we need to set up our spark session.

```r
shp = "/usr/local/spark-3.3.0-bin-hadoop3/"
Sys.setenv(SPARK_HOME = shp)
print(Sys.getenv("SPARK_HOME"))

# Export these in your ~/.bash_profile
aws_access <- Sys.getenv("aws_access")
aws_secret <- Sys.getenv("aws_secret")

library(sparklyr)
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
library(magrittr)

sparkR.session(
  appName = "SparkR_S3_Example",
  sparkConfig = list(
    spark.executor.memory = "8g",
```

[1] https://catalog.data.gov/dataset/motor-vehicle-collisions-crashes

```
    spark.hadoop.fs.s3a.impl = "org.apache.hadoop.fs.s3a.S3AFileSystem",
    spark.hadoop.fs.s3a.access.key = aws_access,
    spark.hadoop.fs.s3a.secret.key = aws_secret)
)
```

We are loading part files without headers, so we'll need to build headers before we load data.

```
schema <- structType(
  structField("CRASH_DATE", "string"),
  structField("CRASH_TIME", "string"),
  structField("BOROUGH", "string"),
  structField("ZIP_CODE", "string"),
  structField("LATITUDE", "double"),
  structField("LONGITUDE", "double"),
  structField("LOCATION", "string"),
  structField("ON_STREET_NAME", "string"),
  structField("CROSS_STREET_NAME", "string"),
  structField("OFF_STREET_NAME", "string"),
  structField("NUMBER_OF_PERSONS_INJURED", "integer"),
  structField("NUMBER_OF_PERSONS_KILLED", "integer"),
  structField("NUMBER_OF_PEDESTRIANS_INJURED", "integer"),
  structField("NUMBER_OF_PEDESTRIANS_KILLED", "integer"),
  structField("NUMBER_OF_CYCLIST_INJURED", "integer"),
  structField("NUMBER_OF_CYCLIST_KILLED", "integer"),
  structField("NUMBER_OF_MOTORIST_INJURED", "integer"),
  structField("NUMBER_OF_MOTORIST_KILLED", "integer"),
  structField("CONTRIBUTING_FACTOR_VEHICLE_1", "string"),
  structField("CONTRIBUTING_FACTOR_VEHICLE_2", "string"),
  structField("CONTRIBUTING_FACTOR_VEHICLE_3", "string"),
  structField("CONTRIBUTING_FACTOR_VEHICLE_4", "string"),
  structField("CONTRIBUTING_FACTOR_VEHICLE_5", "string"),
  structField("COLLISION_ID", "integer"),
  structField("VEHICLE_TYPE_CODE_1", "string"),
  structField("VEHICLE_TYPE_CODE_2", "string"),
  structField("VEHICLE_TYPE_CODE_3", "string"),
  structField("VEHICLE_TYPE_CODE_4", "string"),
  structField("VEHICLE_TYPE_CODE_5", "string")
)
```

Build the 2022 fatalities dataframe

Now what we have spark and a structure, let's load the traffic data set, do some minor adjustments, then filter that down to just fatality records from 2022.

```
s3_path <- "s3a://tonyfraser-public/datasets/nyctraffic/*.csv"

# Why is CRASH_DATE not .asDate()? Because that doesn't run in parallel.
traffic <- read.df(s3_path, "csv", header = "false", schema=schema) %>%
  SparkR::withColumn(
      "CRASH_DATE",
      SparkR::from_unixtime(SparkR::unix_timestamp(.$CRASH_DATE, "MM/dd/yyyy"))) %>%
  SparkR::withColumn("week", SparkR::expr("EXTRACT(WEEK FROM CRASH_DATE)")) %>%
  SparkR::orderBy(SparkR::desc(.$CRASH_DATE))

fatalities <- SparkR::filter(
    traffic,
    (traffic$NUMBER_OF_PERSONS_KILLED > 0) |
    (traffic$NUMBER_OF_CYCLIST_KILLED > 0) |
    (traffic$NUMBER_OF_MOTORIST_KILLED > 0) ) %>%
    SparkR::filter(SparkR::year(.$CRASH_DATE) == 2022) %>%
    SparkR::withColumn("daily_tot",
        .$NUMBER_OF_PERSONS_KILLED +
        .$NUMBER_OF_CYCLIST_KILLED +
        .$NUMBER_OF_MOTORIST_KILLED) %>%
    SparkR::orderBy(SparkR::desc(.$CRASH_DATE)) %>%
    SparkR::select(.$CRASH_DATE,
                   .$NUMBER_OF_PERSONS_KILLED,
                   .$NUMBER_OF_CYCLIST_KILLED,
                   .$NUMBER_OF_MOTORIST_KILLED,
                   .$daily_tot,
                   .$week)  %>%
    SparkR::withColumnRenamed("NUMBER_OF_PERSONS_KILLED", "ppl") %>%
    SparkR::withColumnRenamed("NUMBER_OF_CYCLIST_KILLED", "cyc") %>%
    SparkR::withColumnRenamed("NUMBER_OF_MOTORIST_KILLED","car")

print(sprintf("Counts => raw dataset: %s  fatalities dataset: %s" , nrow(traffic), nrow(fa
```

```
[1] "Counts => raw dataset: 2021794  fatalities dataset: 277"
```

```
SparkR::showDF(fatalities, numRows = 10)
```

```
+-------------------+---+---+---+---------+----+
|         CRASH_DATE|ppl|cyc|car|daily_tot|week|
+-------------------+---+---+---+---------+----+
|2022-12-31 00:00:00|  1|  0|  0|        1|  52|
|2022-12-31 00:00:00|  1|  0|  0|        1|  52|
|2022-12-30 00:00:00|  1|  0|  1|        2|  52|
|2022-12-29 00:00:00|  1|  0|  1|        2|  52|
|2022-12-29 00:00:00|  1|  0|  1|        2|  52|
|2022-12-29 00:00:00|  1|  0|  0|        1|  52|
|2022-12-28 00:00:00|  1|  0|  0|        1|  52|
|2022-12-28 00:00:00|  1|  0|  1|        2|  52|
|2022-12-28 00:00:00|  1|  0|  1|        2|  52|
|2022-12-28 00:00:00|  1|  1|  0|        2|  52|
+-------------------+---+---+---+---------+----+
only showing top 10 rows
```

Add the window over column

This is how you do window over type functions with spark. Though the syntax of SparkR is very different to scala spark or pyspark, the method signatures are all the roughly the same.

```
windowSpec <- SparkR::windowPartitionBy("week")

fatalities_with_weekly_avg_rounded <- fatalities %>%
  SparkR::withColumn(
      "week_tot",
      SparkR::avg(.$daily_tot) %>% SparkR::over(windowSpec)) %>%
  SparkR::withColumn("week_tot", SparkR::expr("ROUND(week_tot, 2)"))

SparkR::showDF(fatalities_with_weekly_avg_rounded, numRows = 20)
```

```
+-------------------+---+---+---+---------+----+--------+
|         CRASH_DATE|ppl|cyc|car|daily_tot|week|week_tot|
+-------------------+---+---+---+---------+----+--------+
|2022-01-09 00:00:00|  1|  0|  1|        2|   1|     1.8|
|2022-01-08 00:00:00|  1|  0|  1|        2|   1|     1.8|
|2022-01-08 00:00:00|  1|  0|  1|        2|   1|     1.8|
|2022-01-04 00:00:00|  1|  0|  1|        2|   1|     1.8|
|2022-01-04 00:00:00|  1|  0|  0|        1|   1|     1.8|
|2022-01-16 00:00:00|  1|  1|  0|        2|   2|     1.4|
|2022-01-15 00:00:00|  1|  0|  0|        1|   2|     1.4|
```

```
|2022-01-14 00:00:00|  1|  0|  0|         1|   2|     1.4|
|2022-01-13 00:00:00|  1|  0|  1|         2|   2|     1.4|
|2022-01-13 00:00:00|  1|  0|  0|         1|   2|     1.4|
|2022-01-23 00:00:00|  1|  0|  0|         1|   3|     1.0|
|2022-01-23 00:00:00|  1|  0|  0|         1|   3|     1.0|
|2022-01-21 00:00:00|  1|  0|  0|         1|   3|     1.0|
|2022-01-17 00:00:00|  1|  0|  0|         1|   3|     1.0|
|2022-01-25 00:00:00|  1|  0|  1|         2|   4|     1.4|
|2022-01-25 00:00:00|  1|  0|  1|         2|   4|     1.4|
|2022-01-24 00:00:00|  1|  0|  0|         1|   4|     1.4|
|2022-01-24 00:00:00|  1|  0|  0|         1|   4|     1.4|
|2022-01-24 00:00:00|  1|  0|  0|         1|   4|     1.4|
|2022-02-06 00:00:00|  1|  0|  1|         2|   5|     1.5|
+-------------------+---+---+---+----------+----+--------+
only showing top 20 rows
```

Perfect!

The SS3AFileSystem not found error

Every time you set up a java, python, scala, or R versions of spark, and you try to get your spark to talk to AWS S3, you're going to see this error: `java.lang.ClassNotFoundException: Class org.apache.hadoop.fs.s3a.S3AFileSystem not found`. This error is telling you that the jar files in your SPARK_HOME/jars folder aren't set up to work with Amazon. Welcome to spark on AWS. Get used ot it.

Think of it like this. The open source community that built spark doesn't work for Amazon. Intead, they work for free, which means they don't hae to bother building AWS drivers to work with their spark. AWS has to build AWS drivers, and, it's up to you to go find those drivers and learn how to use them.

Bootstrapping spark

This is the shell script I used to build this local environment. As a professional spark developer, I suggest the reader take note and get working these exact versions before attempting to bump up version numbers.

```
#!/bin/bash

# use this spark
# wget https://archive.apache.org/dist/spark/spark-3.3.0/spark-3.3.0-bin-hadoop2.tgz -O -
#  sudo tar -xz -C /usr/local

dir=/usr/local/spark-3.3.0-bin-hadoop3/jars  # find SPARK_HOME?/jars folder
sudo -u root chown -R $(whoami) /usr/local/spark-3.3.0-bin-hadoop3

mvn=https://repo1.maven.org/maven2/
# Don't forget to remove guava* from your SPARK_HOME before you add this one in.
curl --silent $mvn/com/google/guava/guava/23.1-jre/guava-23.1-jre.jar \
    --output "$dir"/guava-23.1-jre.jar
curl --silent $mvn/org/apache/spark/spark-hadoop-cloud_2.12/3.3.0/spark-hadoop-cloud_2.12-
    --output "$dir"/spark-hadoop-cloud_2.12-3.3.0.jar
curl --silent $mvn/com/amazonaws/aws-java-sdk-bundle/1.11.1026/aws-java-sdk-bundle-1.11.10
  --output "$dir"/aws-java-sdk-bundle-1.11.1026.jar
curl --silent $mvn/org/apache/hadoop/hadoop-aws/3.3.2/hadoop-aws-3.3.2.jar  \
    --output "$dir"/hadoop-aws-3.3.2.jar
curl --silent $mvn/org/apache/hadoop/hadoop-client/3.3.2/hadoop-client-3.3.0.jar  \
    --output "$dir"/hadoop-client-3.3.0.jar
echo "Done. Set your SPARK_HOME to: ${dir%/jars}"
```