

# Computing Summary Statistics for Daily Data

Dave Lorenz

December 8, 2015

## Abstract

These examples demonstrate how to compute selected summary statistics for daily streamflow data. The examples can easily be extended to other statistics or data types.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Computing Daily Mean Values</b>	<b>3</b>
<b>3</b>	<b>Computing Annual Mean Values</b>	<b>5</b>
<b>4</b>	<b>Computing Yearly and Monthly Mean Values</b>	<b>6</b>

# 1 Introduction

These examples use data from the smwrData package. The data are retrieved in the following code.

```
> # Load the smwrBase and smwrData packages
> library(smwrBase)
> library(smwrData)
> # Retrieve streamflow data for the Choptank River near Greensboro, Maryland
> data(ChoptankFlow)
> # Print the first and last few rows of the data
> head(ChoptankFlow)
```

	agency_cd	site_no	datetime	Flow	Flow_cd
1	USGS	01491000	1990-01-01	385	A
2	USGS	01491000	1990-01-02	649	A
3	USGS	01491000	1990-01-03	353	A
4	USGS	01491000	1990-01-04	246	A
5	USGS	01491000	1990-01-05	225	A
6	USGS	01491000	1990-01-06	214	A

```
> tail(ChoptankFlow)
```

	agency_cd	site_no	datetime	Flow	Flow_cd
8030	USGS	01491000	2011-12-26	245	A
8031	USGS	01491000	2011-12-27	224	A
8032	USGS	01491000	2011-12-28	337	A
8033	USGS	01491000	2011-12-29	370	A
8034	USGS	01491000	2011-12-30	255	A
8035	USGS	01491000	2011-12-31	221	A

```
> # Check for missing values
> with(ChoptankFlow, screenData(datetime, Flow, year = "calendar"))
```

No missing data between 1990-01-01 and 2011-12-31

## 2 Computing Daily Mean Values

The simplest and most straightforward way to compute summary statistics from arbitrarily grouped data is to use the `tapply` function. At its simplest, it requires only three arguments—`X`, the data to summarize; `INDEX`, the grouping data; and `FUN`, the summary statistic function.

The `smwrBase` package contains the `baseDay` function that can be used to group data by day, so that all data for each day, including February 29, can be summarized. The output can be arranged so that the sequence represents the calendar year, water year, or climate year, beginning January 1, October 1, or April 1, respectively.

The following script demonstrates how to use the `tapply` and `baseDay` functions to compute the daily mean streamflow for the previously retrieved data. It uses the `with` function to facilitate referring to columns in the dataset.

```
> # There are no missing values, so only need the basic
> # 3 arguments for tapply
> ChoptankFlow.daily <- with(ChoptankFlow, tapply(Flow,
+   baseDay(datetime, numeric=FALSE, year="calendar"), mean))
> # Print the first and last few values of the output
> head(ChoptankFlow.daily)
```

```
   Jan 01   Jan 02   Jan 03   Jan 04   Jan 05   Jan 06
144.8182 183.7727 174.6818 171.9091 159.4545 146.3182
```

```
> tail(ChoptankFlow.daily)
```

```
   Dec 26   Dec 27   Dec 28   Dec 29   Dec 30   Dec 31
245.2727 276.3636 192.6818 161.4091 143.2727 135.9545
```

The output from `tapply` is an array. Because the output from the `baseDay` function is an array of one dimension, it is printed in the form of a named vector. Different summary statistic functions will produce different outputs; for example, if the summary function had been `quantile`, the output would have been a list.

The `tapply` function is very powerful and easy to use; however, there are times when we want the output in the form of a dataset rather than a vector or array. In those cases, the `aggregate` function is a better alternative than the `tapply` function. The `aggregate` function has several usage options. The script below demonstrates how to build a formula to compute the same statistics that we computed in the previous script. Early versions of `aggregate` required the output of the summary statistic function to be a scalar, but that is no longer a limitation.

```
> # There are no missing values
> ChoptankFlow.dailyDF <- aggregate(Flow ~
+   baseDay(datetime, numeric=FALSE, year="calendar"),
+   data=ChoptankFlow, FUN=mean)
> # Print the first and last few values of the output
> head(ChoptankFlow.dailyDF)
```

```
   baseDay(datetime, numeric = FALSE, year = "calendar")      Flow
1                                           Jan 01 144.8182
```

2	Jan 02	183.7727
3	Jan 03	174.6818
4	Jan 04	171.9091
5	Jan 05	159.4545
6	Jan 06	146.3182

```
> tail(ChoptankFlow.dailyDF)
```

	baseDay(datetime, numeric = FALSE, year = "calendar")	Flow
361	Dec 26	245.2727
362	Dec 27	276.3636
363	Dec 28	192.6818
364	Dec 29	161.4091
365	Dec 30	143.2727
366	Dec 31	135.9545

```
> # Rename the grouping column
> names(ChoptankFlow.dailyDF)[1] <- "Day"
```

Note that the grouping column, renamed Day in the last line of code, is a factor. If character data are needed, executing the expression:

```
ChoptankFlow.dailyDF$Day <- as.character(ChoptankFlow.dailyDF$Day)
```

will convert the column Day to character.

### 3 Computing Annual Mean Values

The previous example can easily be expanded to any grouping. This example computes annual means by calendar year. The `year` function in `lubridate` is used to group the data by calendar year, and the grouping column is renamed `CalYear`. The `waterYear` function in `smwrBase` can be used to group the data by water year.

```
> # There are no missing values
> ChoptankFlow.yrDF <- aggregate(Flow ~
+   year(datetime),
+   data=ChoptankFlow, FUN=mean)
> # Rename the grouping column
> names(ChoptankFlow.yrDF)[1] <- "CalYear"
> # Print the first few values of the output
> head(ChoptankFlow.yrDF)
```

	CalYear	Flow
1	1990	114.67945
2	1991	99.54521
3	1992	85.62568
4	1993	119.65726
5	1994	203.11233
6	1995	94.18000

Other grouping functions include `month` (month) in `lubridate`, `seasons` (user-defined seasons) in `smwrBase`. Refer to the documentation for each of these functions for a description of the arguments.

## 4 Computing Yearly and Monthly Mean Values

Aggregation can also be done by multiple grouping variables. This example computes the mean streamflow for each month by year. This example uses the `year` and the `month` functions because the output is sorted by groups. The sequence of the groups in the call is important—the sorting is done in the order specified in the formula. For this example, the data are sorted by month and then by year, which in this case, keeps the order correct; grouping by water year would misplace October, November, and December. For a calendar year table, the months are in the correct order.

```
> # There are no missing values
> ChoptankFlow.my <- aggregate(Flow ~ month(datetime, label=TRUE) + year(datetime),
+   data=ChoptankFlow, FUN=mean)
> # Rename columns 1 and 2
> names(ChoptankFlow.my)[1:2] <- c("Month", "Year")
> # Print the first few values of the output
> head(ChoptankFlow.my)
```

	Month	Year	Flow
1	Jan	1990	238.5161
2	Feb	1990	152.0357
3	Mar	1990	137.5806
4	Apr	1990	223.6667
5	May	1990	288.8710
6	Jun	1990	108.4333

The output dataset may be used as is, or it could be restructured to a table of monthly values for each calendar year. To create a table by water year, the levels in the column Month must be reordered to begin in October and end in September.

```
> # Restructure the dataset
> ChoptankFlow.myTbl <- group2row(ChoptankFlow.my, "Year", "Month", "Flow")
> # Print the first few values of the output, set width for Vignette
> options(width=70)
> head(ChoptankFlow.myTbl)
```

	Year	Jan.Flow	Feb.Flow	Mar.Flow	Apr.Flow	May.Flow	Jun.Flow
1	1990	238.51613	152.03571	137.5806	223.66667	288.87097	108.43333
2	1991	252.35484	108.85714	182.6452	192.10000	88.74194	69.80000
3	1992	80.93548	84.10345	197.8065	109.76667	115.00000	79.80000
4	1993	174.29032	172.67857	512.1935	312.36667	107.77419	39.56667
5	1994	210.25806	372.57143	826.2903	331.33333	120.96774	55.83333
6	1995	183.54839	122.60714	221.8065	84.03333	107.16129	46.86667

  

		Jul.Flow	Aug.Flow	Sep.Flow	Oct.Flow	Nov.Flow	Dec.Flow
1	58.83871	44.580645	23.800000	29.58065	23.83333	47.48387	
2	93.93548	48.032258	25.833333	26.58065	24.63333	79.16129	
3	24.19355	62.838710	40.366667	39.19355	69.00000	123.00000	
4	13.03871	9.812903	9.526667	13.11935	20.63333	52.90323	
5	75.03226	157.387097	96.300000	55.51613	59.23333	84.32258	
6	23.87097	17.777419	15.520000	53.32258	123.90000	129.06452	

Note that this example used the `group2row` function in `smwrBase`. The `reshape` function in `stats` and `stack` and `unstack` functions in `utils` are other functions that will restructure data.