Introduction to RavenR

Robert Chlumsky, Dr. James R. Craig

March 1, 2021

Introduction to RavenR Tutorial

This short document is intended to get you started with using RavenR to aid your analysis with the Raven Hydrologic Modelling Framework. This tutorial will get you up and running with the RavenR package and comfortable running a few commands. Some knowledge or R is presumed in this document. If you are not comfortable with R, take a look at any number of R training and Introductory resources, such as the tRaining repository) on Github.

This exercise will use the Nith River modelled output available from within the RavenR package, thus the functions to read in data from csv files are not required. However, it is recommended that you download the Nith river model files, and try to both run the model and read in the output files. The Nith river model can be downloaded from the Raven Tutorial #2.

Getting Acquainted with RavenR

If you don't have RavenR yet installed in your R library, run the following commands to install the RavenR package directly from the Comprehensive R Archive Network (CRAN), which is available on CRAN as version 2.0.1.

```
install.packages("RavenR")
```

For those interested in the latest versions of RavenR or in contributing to the development of RavenR, the package may be found on Github at https://github.com/rchlumsk/RavenR. Packages on Github may also be installed from within R using the `devtools` library.

Load the RavenR library from the console and view its contents with the following commands:

```
library(RavenR)

# view first 20 functions in RavenR
ls("package:RavenR") %>%
  head(., 20)
```

Each function in the package is documented, which includes a description of the function, its inputs and outputs, and an example. You can look at any of the function examples by typing out the name of the function beginning with a question mark, which will show the help information at the right of the RStudio environment.

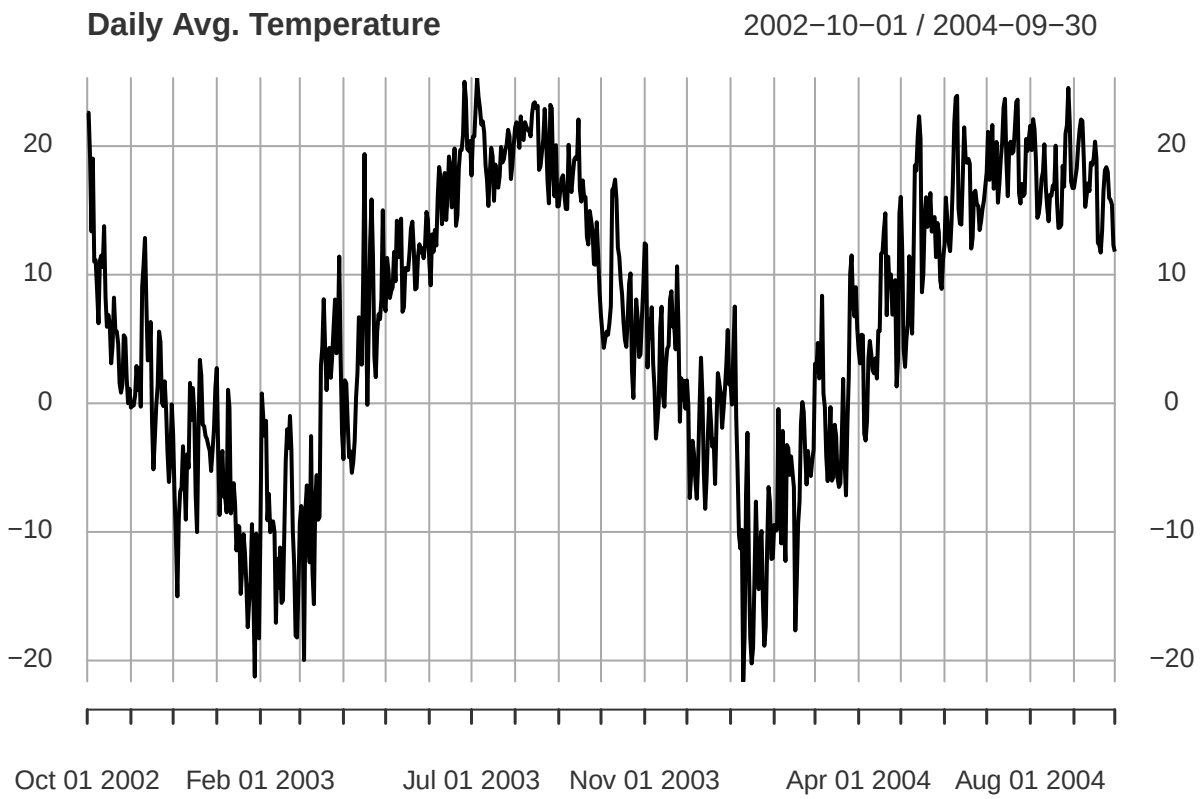
```
?rvn_flow_scatterplot
```

The name of each external function in the RavenR package begins with the "rvn_" prefix, so you in practice 'search' for functions by beginning to type them out. Try this to see what functions are available with "rvn_rvh_".

Sample Data in RavenR

The RavenR package contains a number of sample data files, which are useful for training purposes and testing of functions. The package contains sample data both in R format (under RavenR/data) and as raw data files in their native formats (RavenR/inst/extdata). The sample data set from the RavenR package (in R format) can be loaded in using the data function (with either quotes or just the name of the data), e.g.,

```
data("rvn_forcing_data")
# ?rvn_forcing_data
plot(rvn_forcing_data$forcings$temp_daily_ave,
     main="Daily Avg. Temperature")
```



Notice as well that the sample data set in R format also has a built in help file to describe the data.

To locate the raw data from the RavenR package, we will use a syntax to find the data by file name in the RavenR package directory, which ends up looking more similar to a raw file call. This raw data file comes from the **inst/extdata** folder in the RavenR package. Note that this is done so that the sample data in raw format can be used and tested with functions, and the syntax to locate the data file is more portable across various computer operating systems.

```
# read in hydrograph sample csv data from RavenR package
ff <- system.file("extdata","run1_Hydrographs.csv", package="RavenR")

# ff is a simple string, which can be substituted with any file location
ff
```

```
## [1] "C:/Users/rober/Documents/GitHub/RavenR/inst/extdata/run1_Hydrographs.csv"
```

```
# read in sample rvi file from the RavenR package
rvi_file <- system.file("extdata", "Nith.rvi", package="RavenR")

# show first 6 lines of the file
readLines(rvi_file) %>% head()
```

```
## [1] "# ------------------------------------------------"
## [2] "# Raven Input file"
## [3] "# HBV-EC Nith River emulation test case"
## [4] "# ------------------------------------------------"
## [5] "# --Simulation Details -------------------------"
## [6] ":StartDate       2002-10-01 00:00:00"
```

The `system.file` command will simply build a file path for where this data file is located on your machine with the RavenR package installation, which can then be passed to any function as required to provide a file location. This command will be used throughout this tutorial in place of local files for portability, however, your own data files may be swapped in place of the system.file locations. For example, you may wish to pass files from other Raven Tutorial files by changing the file paths throughout this tutorial.

Diagnostics and Plotting

Now you are ready to start using RavenR to directly visualize and manipulate model output. This section of the exercise will make use of raw sample data in the RavenR package to illustrate some of the diagnostics and plotting capabilities of RavenR.

Forcing Functions

Start by finding the raw **run1_ForcingFunctions.csv** file with the `system.file` command. Note that this can be replaced with your own forcing functions file location if preferred. We will store the forcing functions data into an object called ff (and obtain just the subobject using the '$' operator), and then view the first few rows using the **head** function. We will show only the first six columns of the data for brevity.

```
ff <- system.file("extdata","run1_ForcingFunctions.csv",package="RavenR")
# ff <- "C:/TEMP/Nith/output/ForcingFunctions.csv" # replace with your own file
ff_data <- RavenR::rvn_forcings_read(ff)
head(ff_data$forcings[,1:6])
```

```
## Warning: timezone of object (UTC) is different than current timezone ().
```

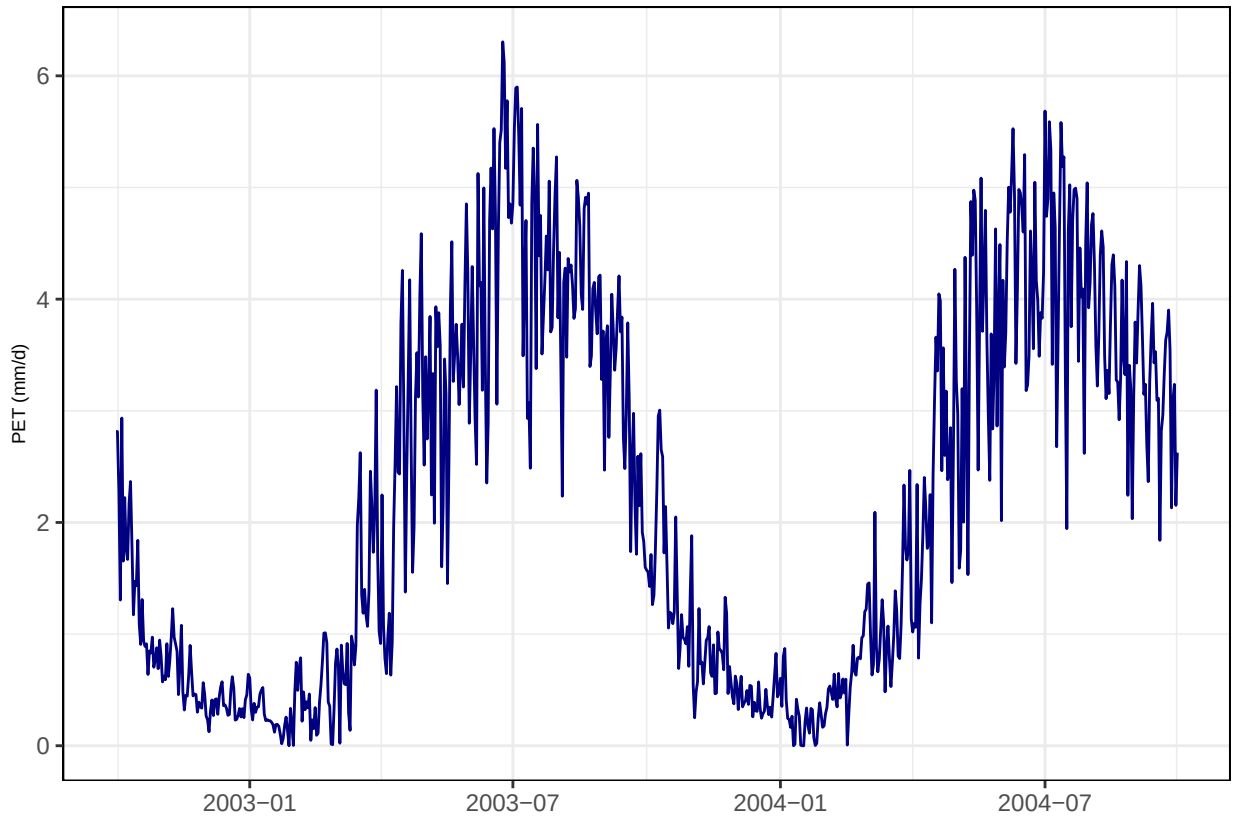
```
##            day_angle     rain snow    temp temp_daily_min temp_daily_max
## 2002-10-01   4.70809 3.468690    0 22.5956       17.92510        27.2662
## 2002-10-02   4.70809 3.468690    0 22.5956       17.92510        27.2662
```

```
## 2002-10-03   4.72530 1.189180     0 19.2076       15.40780        23.0075
## 2002-10-04   4.74251 2.083260     0 13.3714       11.49870        15.2440
## 2002-10-05   4.75973 6.474310     0 19.0304       12.50970        25.5510
## 2002-10-06   4.77694 0.125591     0 11.0186        7.43466        14.6024
```

Now we can plot the forcing data using the rvn_forcings_plot function. This creates an output of the five main forcings from the data set, from which we can plot one or more forcings, including a plot of the whole set of plots. This is typically a reasonable reality check on the model forcings.

Here, we plot the PET from the set of created plots.

```
myplots <- rvn_forcings_plot(ff_data$forcings)
# myplots$Temperature
# myplots$Radiation
# myplots$AllForcings
myplots$PET
```



Hydrograph and Diagnostics

We can similarly access the hydrograph fit. Here the hydrograph sample data is located with the usual `system.file` command, then read into R with the `rvn_hyd_read` function intended for reading Hydrographs file. The flows from a specific subbasin can be extracted using the `rvn_hyd_extract` function, which is done here for subbasin 36. The precipitation can be extracted similarly.

```
ff <- system.file("extdata","run1_Hydrographs.csv",package="RavenR")
# ff <- "mydirectory/Hydrographs.csv" # replace with your own file
hy <- rvn_hyd_read(ff)
head(hy$hyd)
```

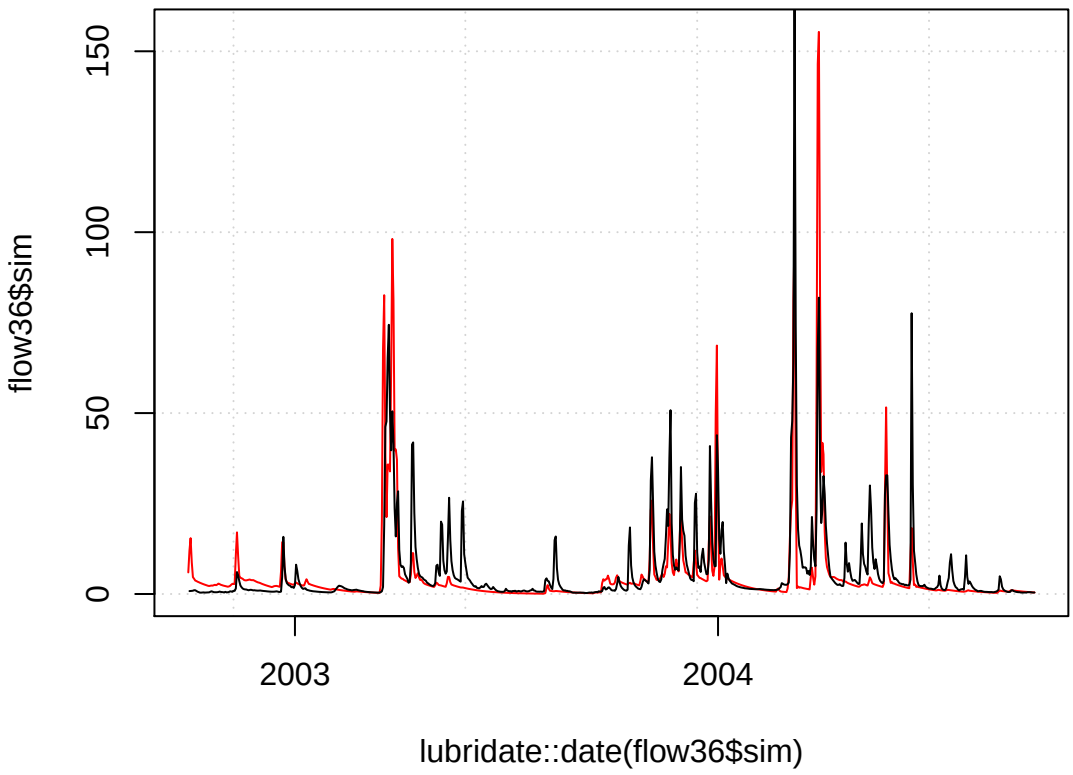
```
## Warning: timezone of object (UTC) is different than current timezone ().
```

```
##              precip    Sub36 Sub36_obs    Sub43 Sub43_obs
## 2002-10-01       NA  5.96354        NA 11.25050        NA
## 2002-10-02 3.468690 11.96430     0.801 18.59070      3.07
## 2002-10-03 1.189180 15.43700     0.828 25.74430      2.99
## 2002-10-04 2.083260  8.76948     0.860 18.68610      3.06
## 2002-10-05 6.474310  4.66501     0.903  9.82648      2.93
## 2002-10-06 0.125591  4.20829     1.040  7.90952      3.15
```

```
flow36 <- rvn_hyd_extract("Sub36",hy)
precip <- hy$hyd$precip
```

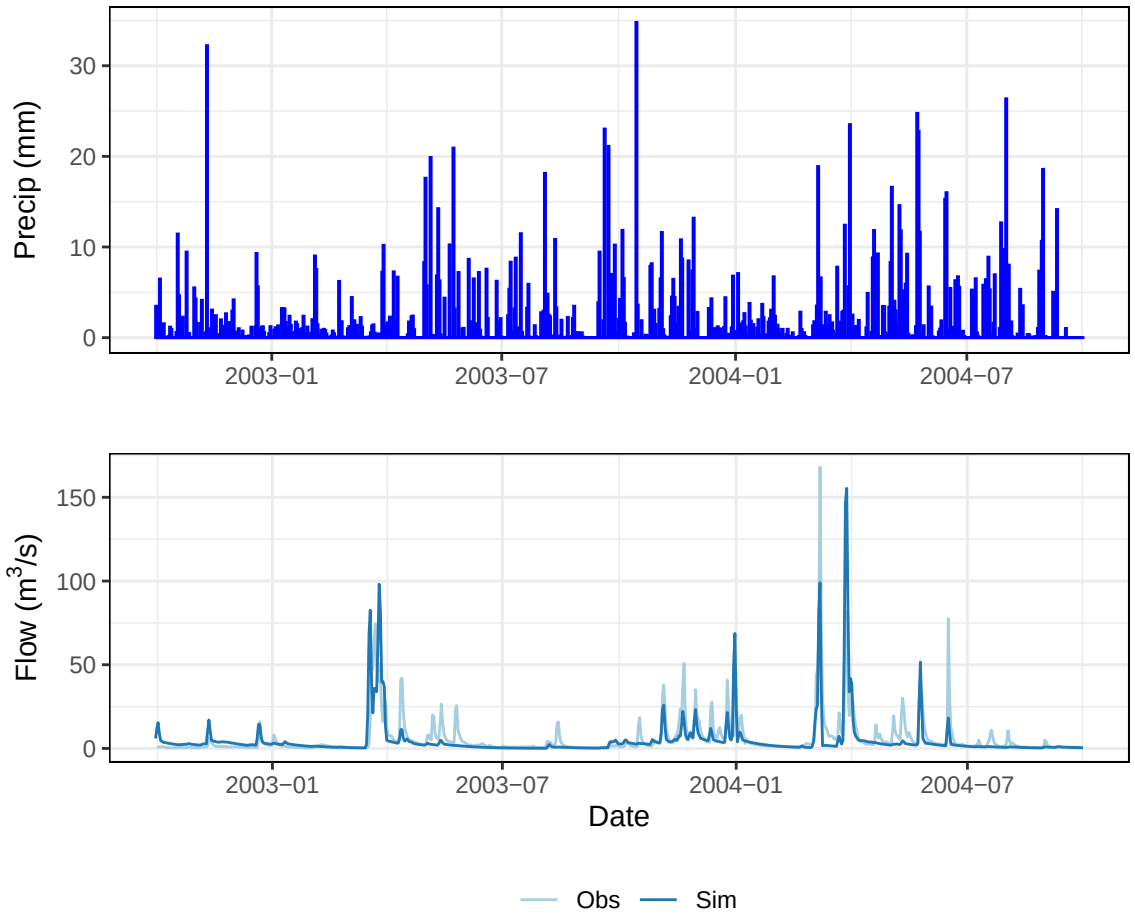
The hydrograph object flow36 now stores the simulated hydrograph (`flow36$sim`) and the observed hydrograph (`flow36$obs`), and the null subobject (`flow36$inflow`). The precip object stores the entire time series of watershed-averaged precip (`precip$sim`). We can plot the simulated and observed hydrograph with the following commands in base R, extracting the date:

```
plot(lubridate::date(flow36$sim), flow36$sim,col='red',
     type='l', panel.first=grid())
lines(lubridate::date(flow36$obs), flow36$obs,col='black')
```



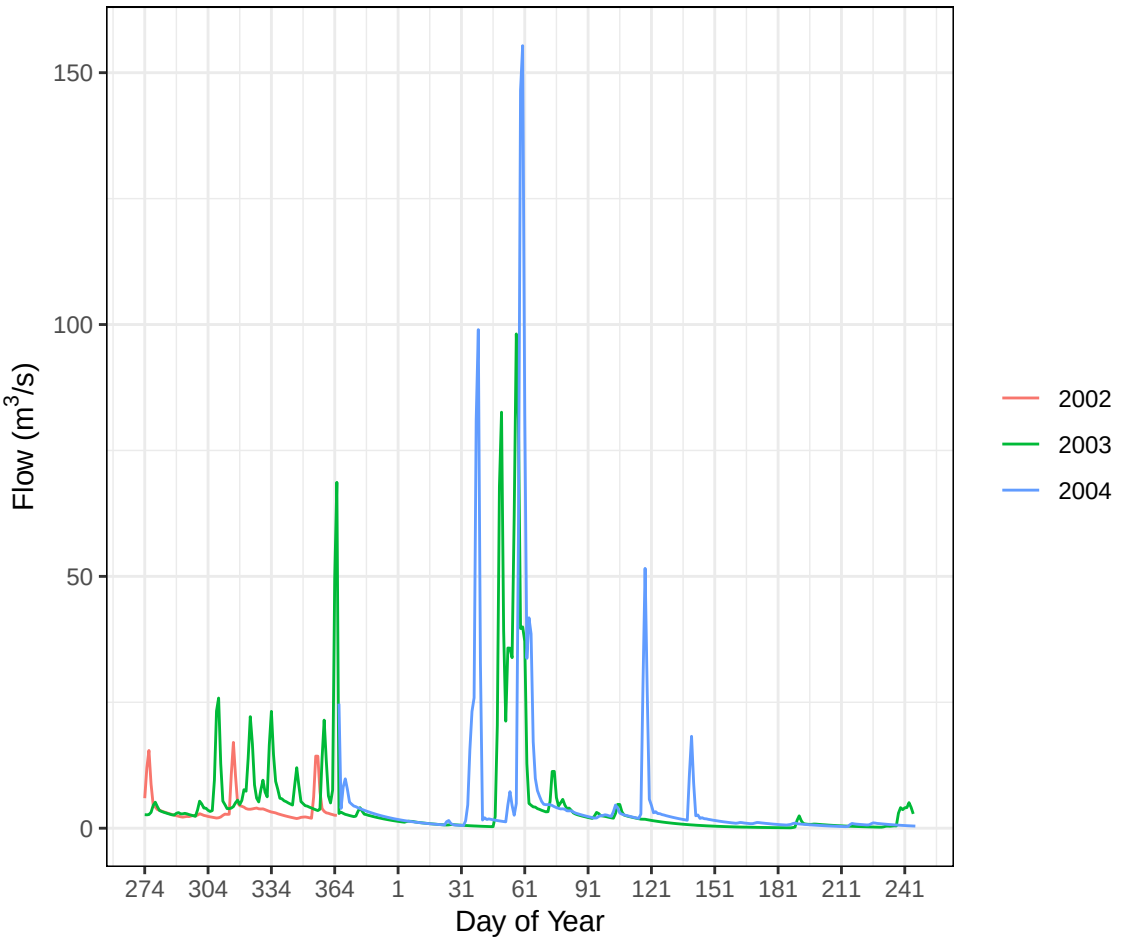
A ggplot format plot can also be created using the `rvn_hyd_plot` function in the RavenR library. This function can save some of the trouble of plotting the precipitation on the secondary axis.

```
rvn_hyd_plot(sim=flow36$sim, obs=flow36$obs, precip=precip)
```



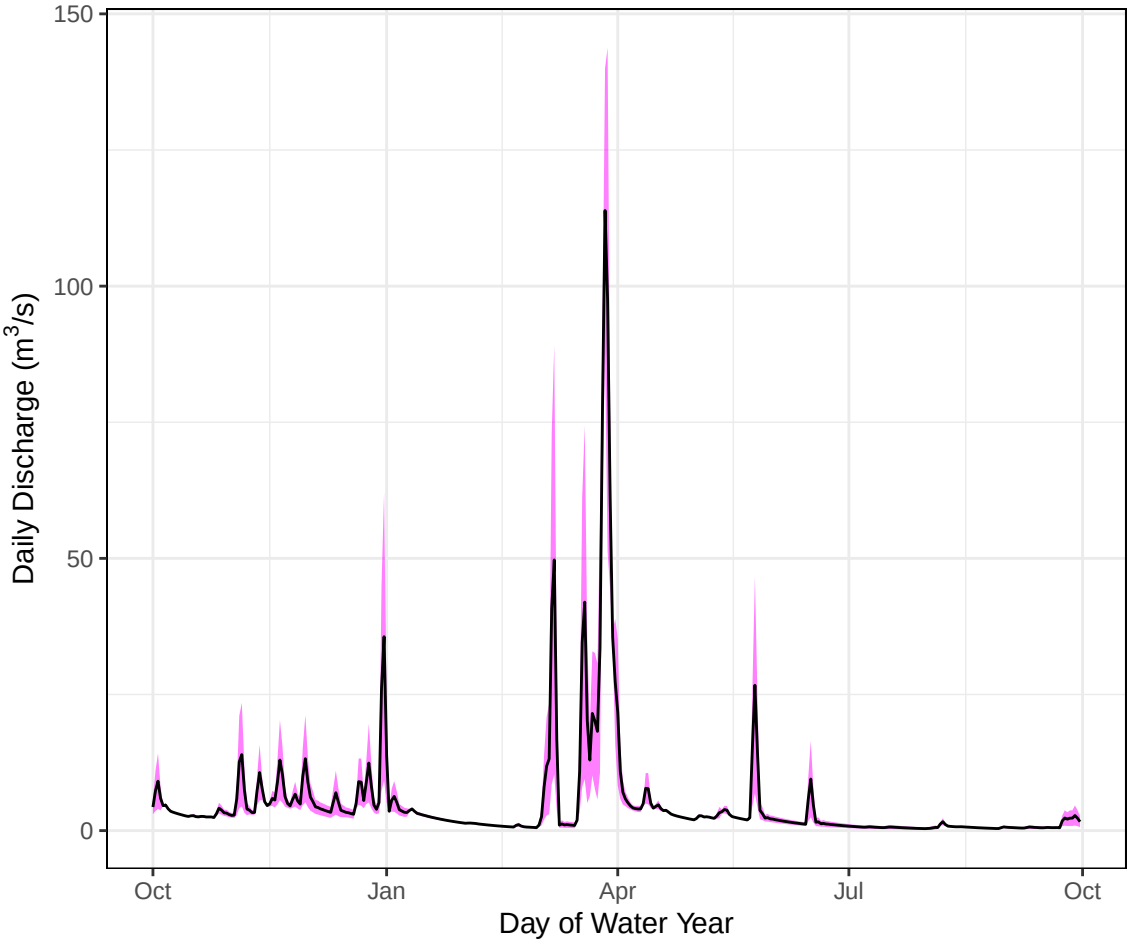
There are some other helpful functions in RavenR for understanding our hydrographs. For example, the 'spaghetti' plot overlays the hydrographs from the supplied series and plots them against day of year on the x-axis, facilitating a comparison across multiple years.

```
rvn_flow_spaghetti(flow36$sim)
```



The annual quantiles function compute the flow quantiles for a given time series for each day of the year, and plot those quantiles with the corresponding plot function. This provides a similar look to the spaghetti plot, but with smooth quantiles instead of overlaying time series.

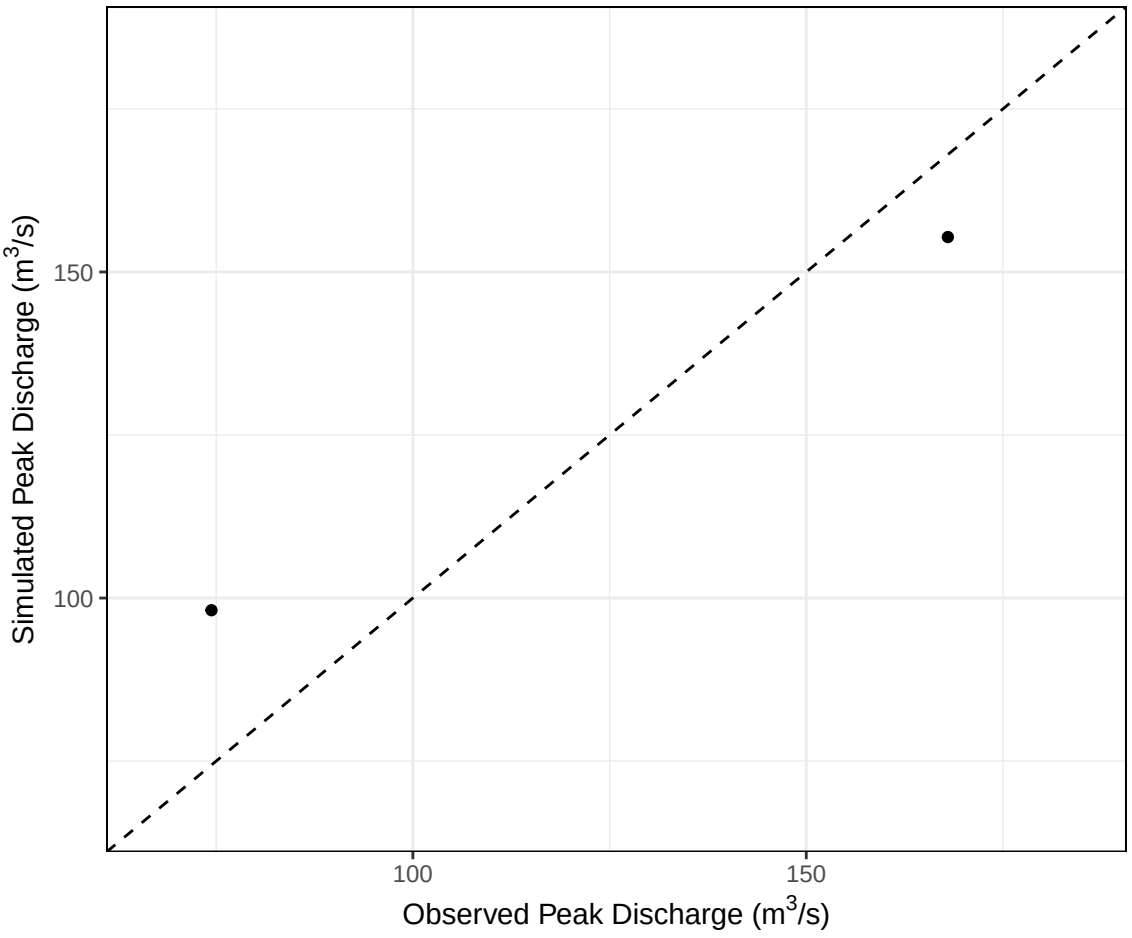
```
rvn_annual_quantiles(flow36$sim) %>%
  rvn_annual_quantiles_plot(., ribboncolor='magenta')
```



Other plots indicate the agreement between peak flows in the modelled and observed.

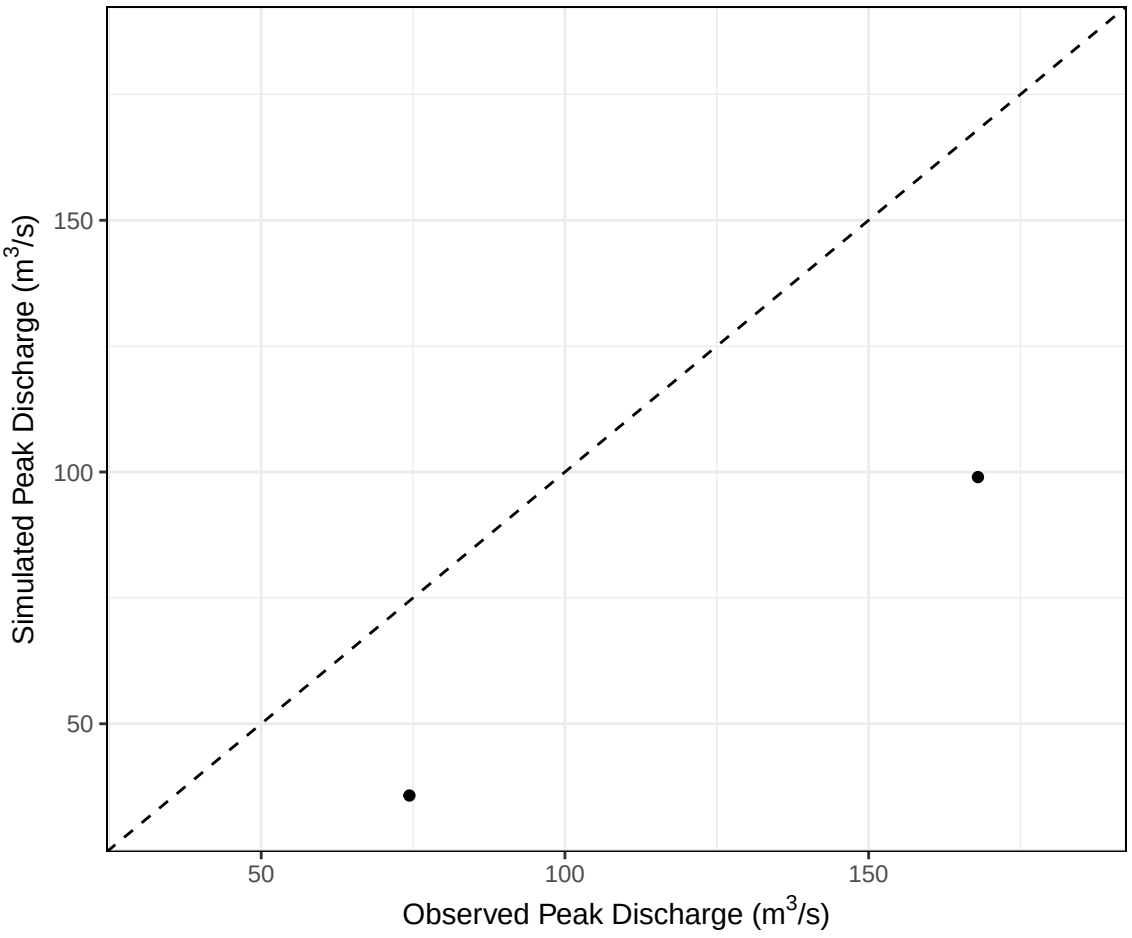
```
rvn_annual_peak(flow36$sim, obs=flow36$obs)
```

```
## $df_peak
##     date.end sim.peak obs.peak
## 1 2003-09-30  98.1327     74.4
## 2 2004-09-30 155.3560    168.0
##
## $p1
```



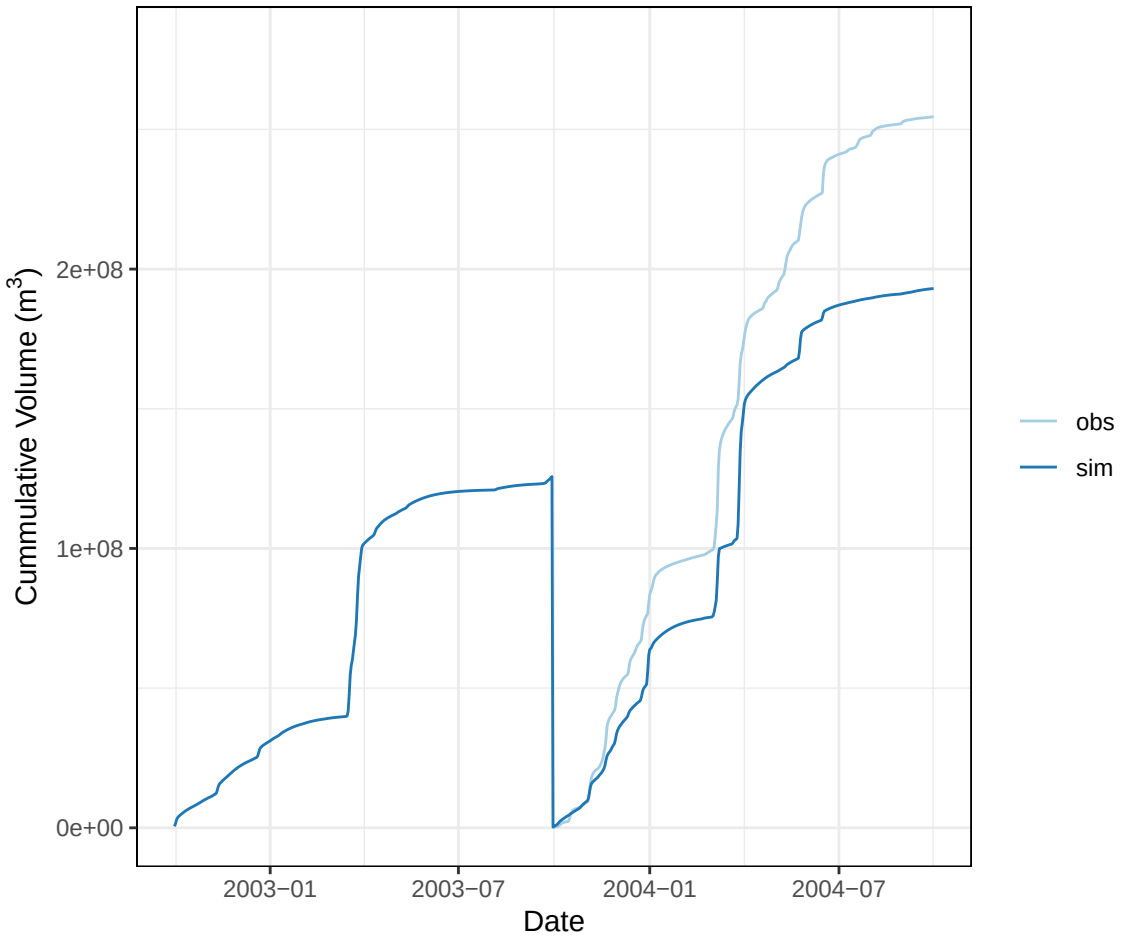

```
rvn_annual_peak_event(flow36$sim, obs=flow36$obs)
```

```
## $df_peak_event
##    obs.dates sim.peak.event obs.peak.event
## 1 2003-03-23        35.7503           74.4
## 2 2004-03-07        98.9858          168.0
##
## $p1
```



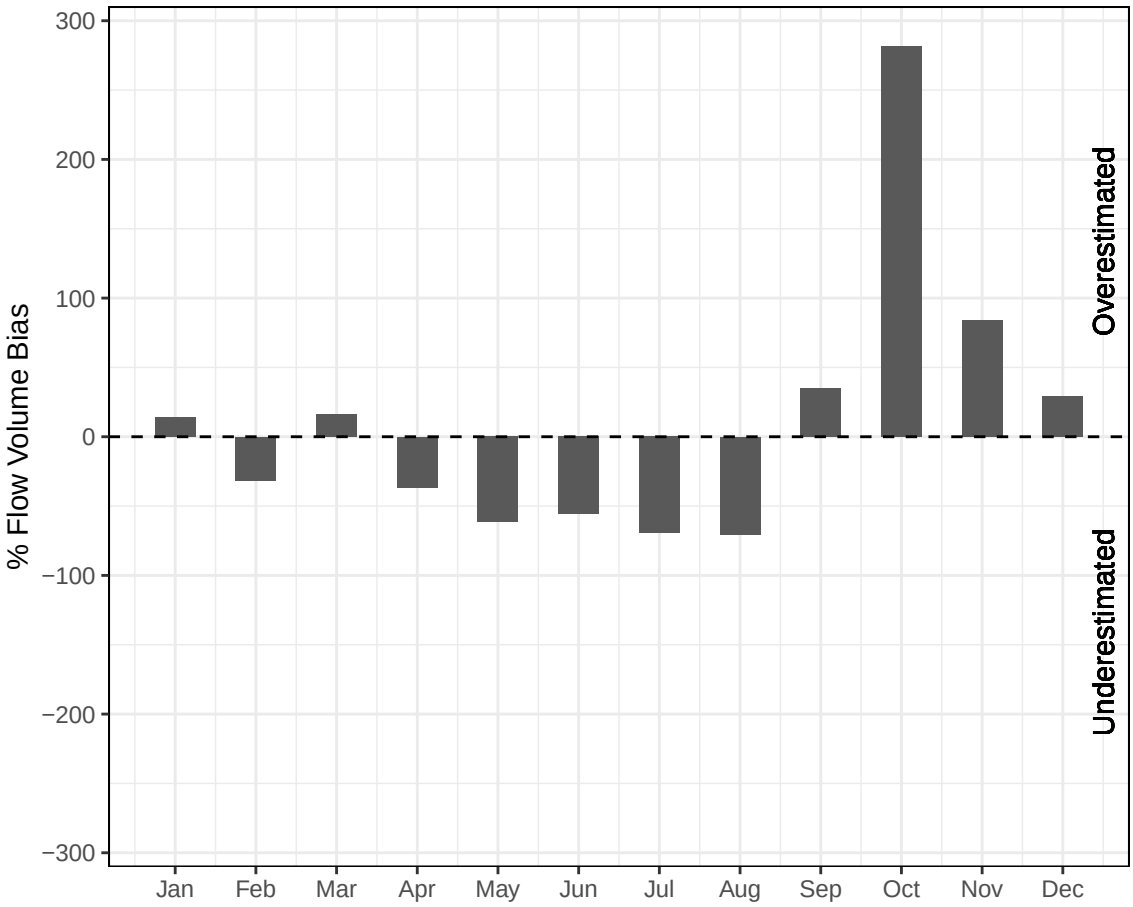
We can also use some of the Raven plots to get some diagnostics and comparisons on the simulated and observed hydrographs. For example, the plots below compare the annual cumulative flow and the monthly volume bias, respectively.

```
rvn_cum_plot_flow(flow36$sim, obs=flow36$obs)
```




```
rvn_monthly_vbias(flow36$sim, obs=flow36$obs)
```

```
## $df.mvbias
##        mvbias
## Jan  14.21804
## Feb -31.63489
## Mar  15.93412
## Apr -36.52680
## May -61.52621
## Jun -55.78284
## Jul -69.45787
## Aug -70.45509
## Sep  35.16982
## Oct 281.28584
## Nov  83.76182
## Dec  29.17106
##
## $plot
```



Flow DyGraphs

A fun feature in the RavenR package is the addition of dygraphs, which produces a dynamic hydrograph plot with a slider on the time scale. This is particularly helpful for viewing subsections of a hydrograph dynamically, and comparing the simulated and observed hydrographs in an interactive environment. Only the output object from the `rvn_hyd_read` function and the name of headings to extract is required. Note that this interactive plot does not work well in a PDF file, so try this out in R/RStudio.

```
rvn_hyd_dygraph(hy, basins="Sub36")
```

While this tutorial focuses on the forcing functions and hydrograph data, similar routines exist for reading in the reservoir stages output, the watershed mass energy balance files, and other Raven outputs.

RavenR Time Series Utilities

One utility that does not exist in the `xts` package, but is nonetheless useful in hydrology, is the application of functions to the water year period rather than the calendar year. In the RavenR package, the `rvn_apply_wyearly` function accomplishes this. The water year is by default set to index by the October 1st water year (specifying the period ending September 30th), but this may be provided as an argument to the function for different water years. Here, we apply the mean function to the water year using sample hydrograph data in RavenR.

```
myhyd <- system.file("extdata","run1_Hydrographs.csv", package="RavenR") %>%
  rvn_hyd_read()

library(xts)

# apply mean to calendar year in hydrograph data
xts::apply.yearly(myhyd$hyd$Sub36, mean, na.rm=TRUE)
```

```
##               Sub36
## 2002-12-31 3.886438
## 2003-12-31 4.962424
## 2004-09-30 5.549967
```

```
# apply mean as FUN to daily average temperature
RavenR::rvn_apply_wyearly(myhyd$hyd$Sub36, mean, na.rm=TRUE)
```

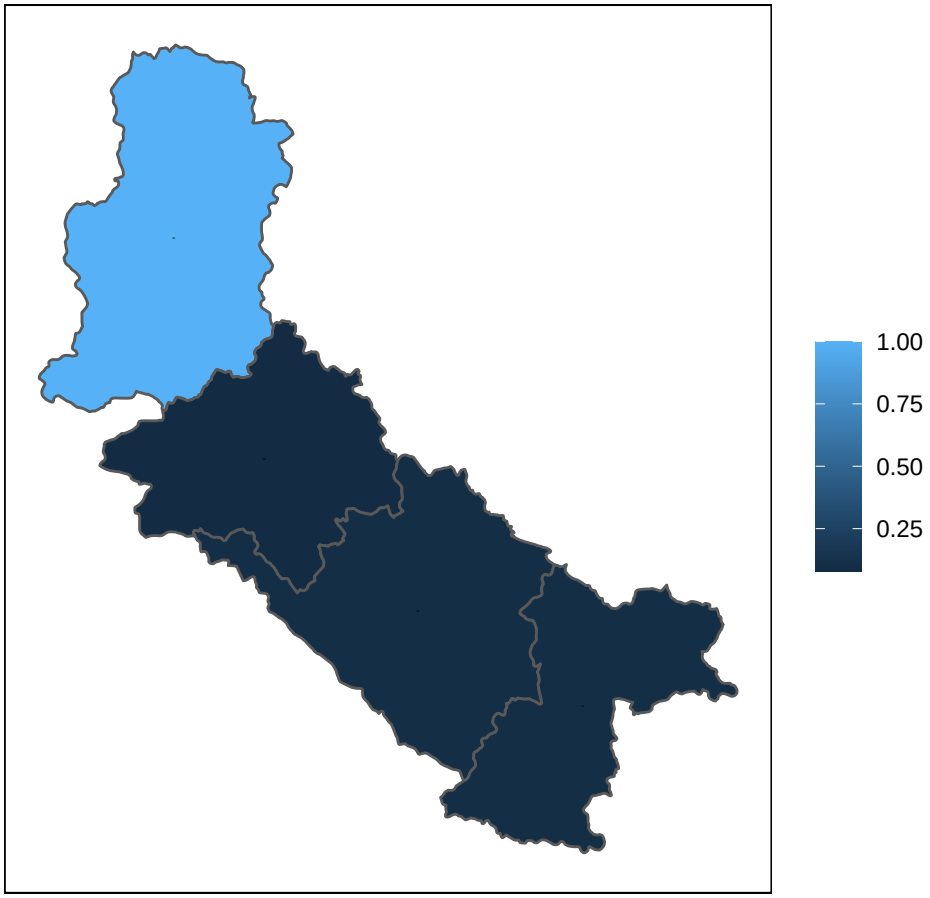
```
##               Sub36
## 2003-09-30 3.994746
## 2004-09-30 6.096847
```

Spatial Plotting with Custom Data

The RavenR package also has some basic functionality for spatial plots. This section will use some of the sample spatial data for the Nith basin to build a subbasin plot from custom output data, which can be modified to show any custom output data.

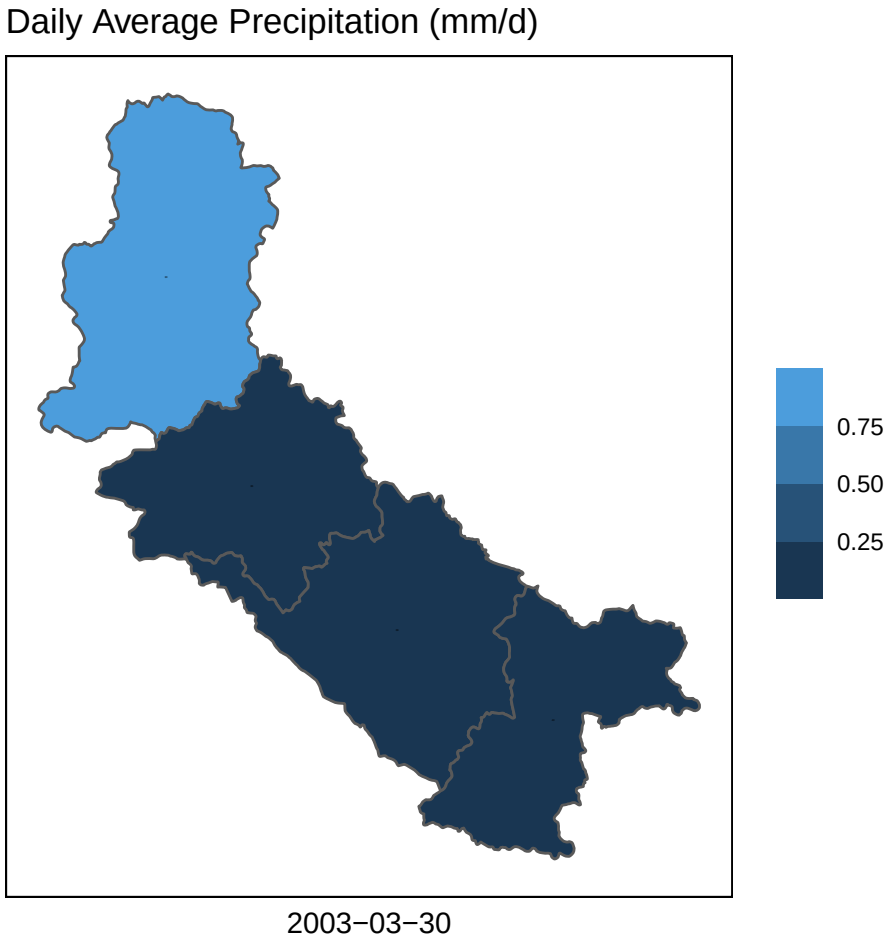
Begin by loading in the appropriate sample data files, and locating them on the local machine. This includes a custom output of daily average precipitation by subbasin for the Nith subbasin.

```
# Raw sample data
shpfilename <- system.file("extdata","Nith_shapefile_sample.shp",package="RavenR")

# plot shapefile with baseplot in R
sf::read_sf(shpfilename) -> shp
# plot(shp$geometry)

# Custom Output data from Raven for Nith basin
cust.data <- rvn_custom_read(system.file("extdata","run1_PRECIP_Daily_Average_BySubbasin.csv",
                                         package="RavenR"))

subIDcol <- 'subID' # attribute in shapefile with subbasin IDs
plot.date <- "2003-03-30" # date for which to plot custom data

# function call
rvn_subbasin_map(shpfilename, subIDcol, plot.date, cust.data)
```



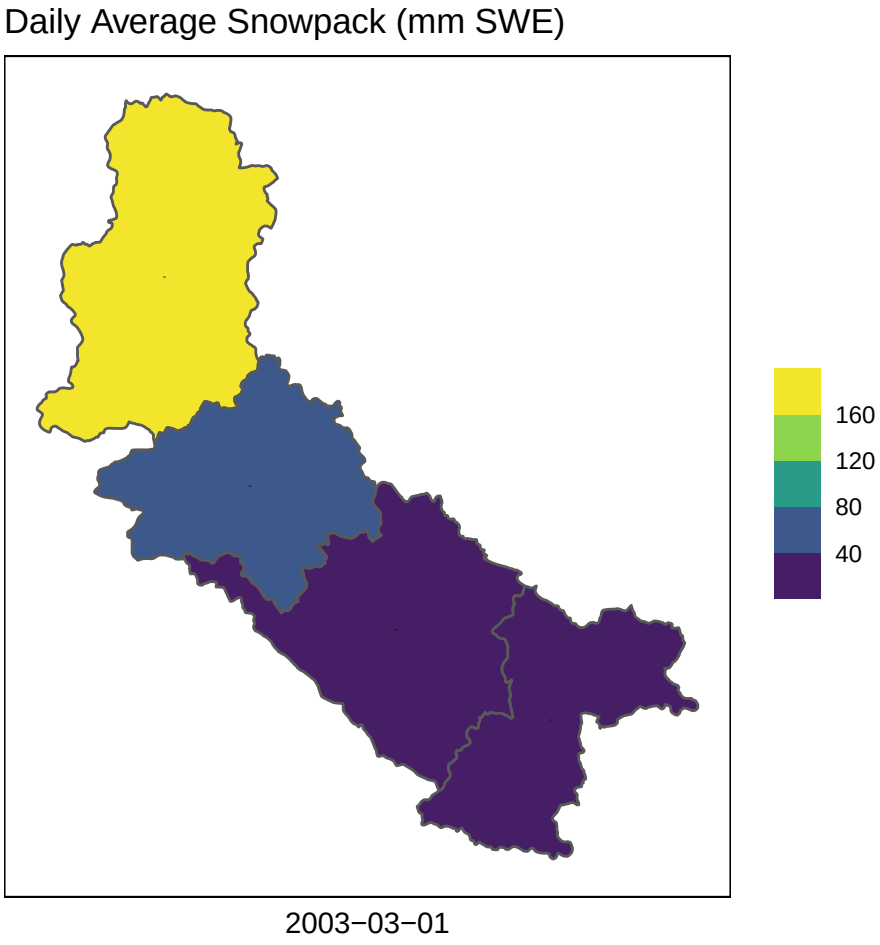
This produces a basic static map of the Nith subbasins provided in the sample file, with the precipitation data from the Custom Output data provided at the date specified. We can customize the plot now using the functionality of `ggplot2`, for example, adding a title to the plot and changing the legend axis to discrete rather than continuous.

```
library(ggplot2)

# create an updated plot
p1 <- rvn_subbasin_map(shpfilename, subIDcol, plot.date, cust.data)
p1 + ggtitle("Daily Average Precipitation (mm/d)") +
      scale_fill_binned()
```



This function can accept any custom output data as the input data for a given subbasin shapefile, provided that the numbering of the custom data and the labels on the subbasin IDs in the shapefile correspond. Try to use the sample external data for snowpack in this plot as well! Use the 'run1_SNOW_Daily_Average_BySubbasin.csv' file and change the colour scheme to the colour-blind friendly set 'viridis'.

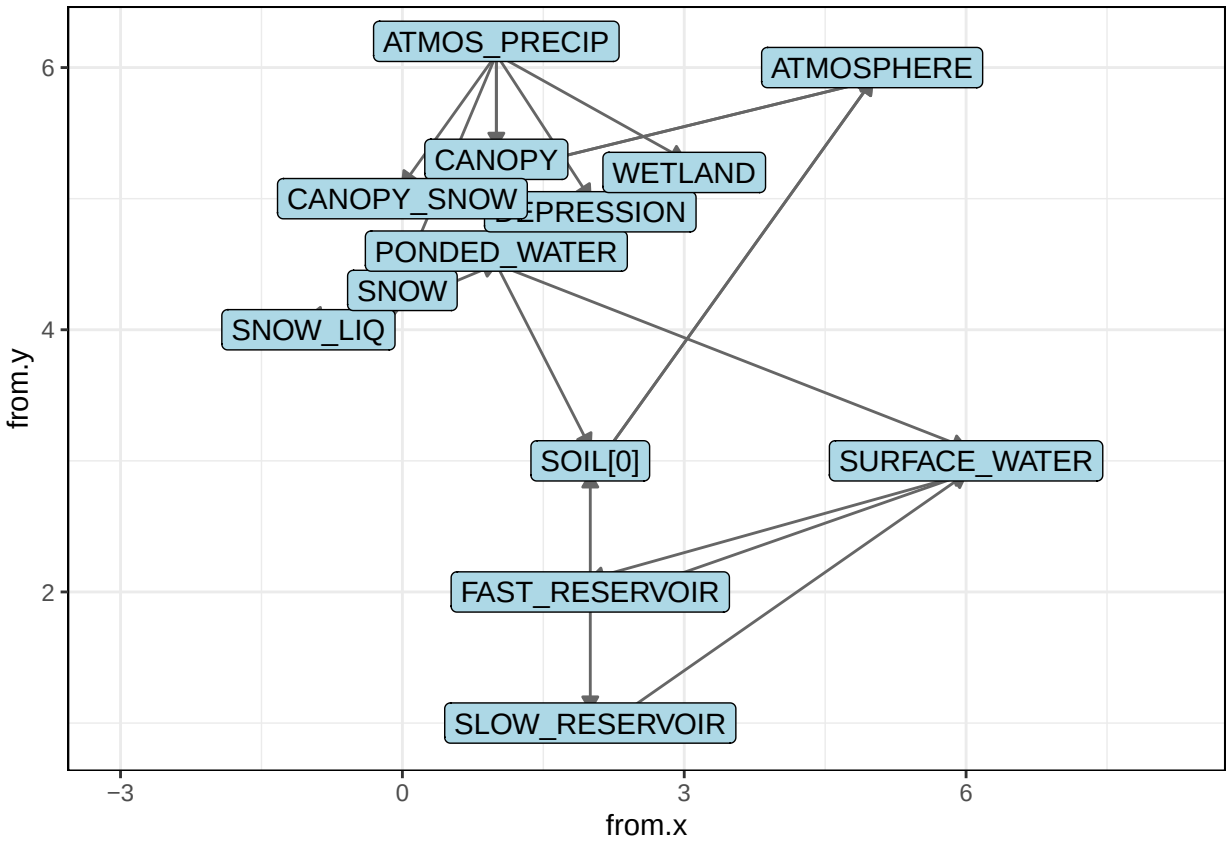
```
cust.data <- rvn_custom_read(system.file("extdata","run1_SNOW_Daily_Average_BySubbasin.csv",
                                         package="RavenR"))
plot.title <- 'Daily Average Snowpack (mm SWE)'
plot.date <- "2003-03-01" # date for which to plot custom data

# create an updated plot, change the colour scheme
rvn_subbasin_map(shpfilename, subIDcol, plot.date, cust.data)+
  ggtitle(plot.title)+
  scale_fill_binned(type="viridis")
```



RVI File Utilities

The RavenR package can read the rvi file and generate a network plot of the model connections, which is helpful in understanding the model configuration. This can be done as follows, again using the Nith rvi file as an example. The network plot may be directly written to file by supplying a file name to the `pdfout` argument, which is NULL by default.

```
rvi <- rvn_rvi_read(system.file("extdata","Nith.rvi", package="RavenR"))

rvn_rvi_connections(rvi) %>%
rvn_rvi_process_plot(., pdfout = NULL)
```



RVH File Utilities

The discretization file, rvh, may be read in and modified using the RavenR package. The `rvn_rvh_read` function also computes the upstream area of each subbasin. The corresponding `rvn_rvh_write` function also writes the rvh object back to file.

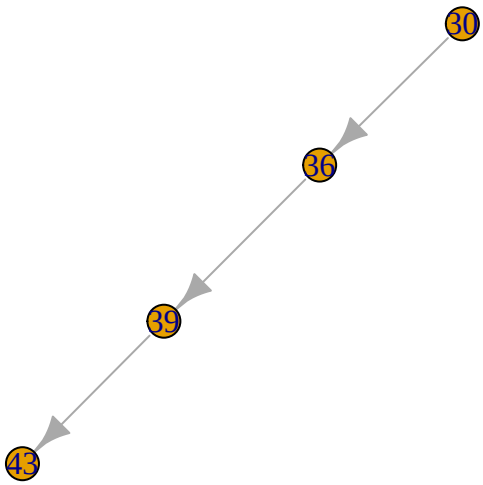
```
# read in rvh file
rvh <- rvn_rvh_read(system.file("extdata","Nith.rvh", package="RavenR"))

rvh$SBtable[, c("SBID","Downstream_ID","Area","TotalUpstreamArea")]
```

```
##    SBID Downstream_ID  Area TotalUpstreamArea
## 30   30            36 318.5             318.5
## 36   36            39 219.2             537.7
## 39   39            43 271.7             809.4
## 43   43            -1 205.0            1014.4
```

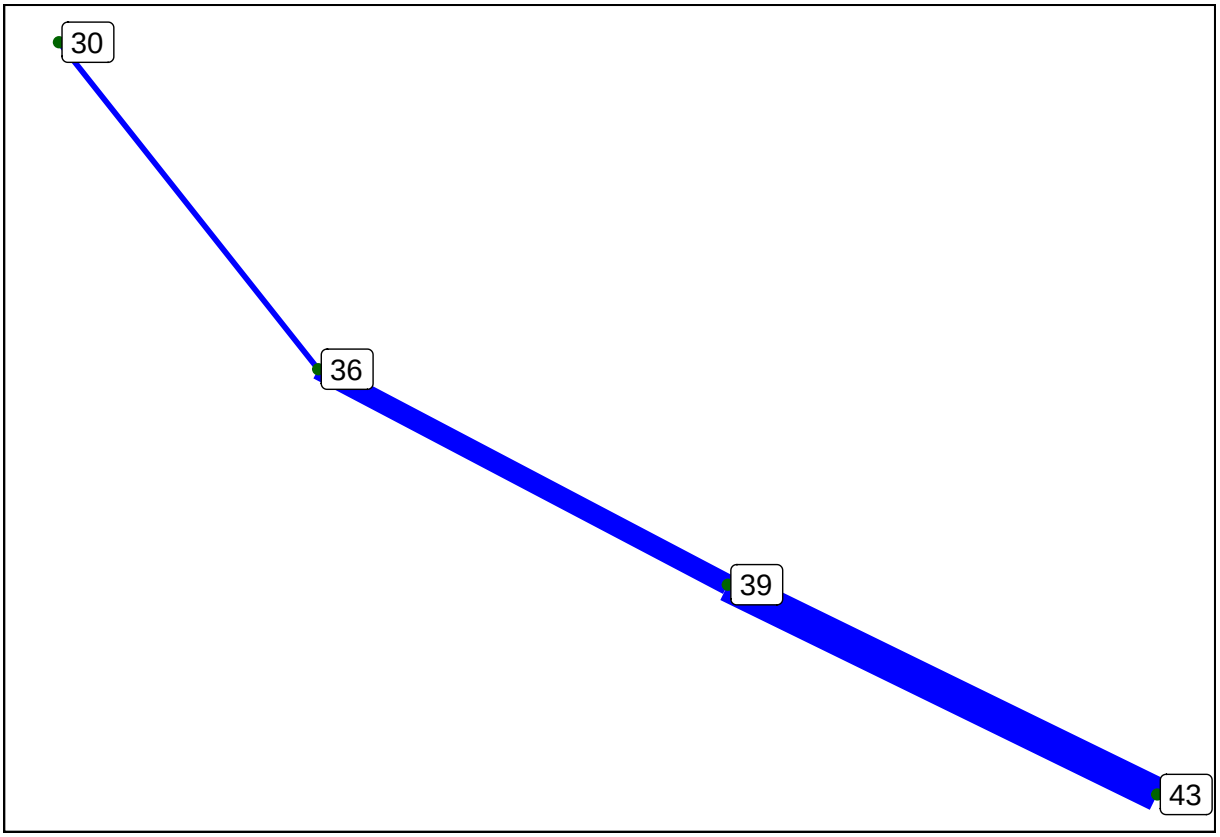
Similar to the rvi functionality, the subbasin connections from the rvh file may be plotted using one of two ways. The `rvn_rvh_read` function has built-in functionality to build a network plot, which may be plotted directly. The `rvn_subbasin_network_plot` function uses the subbasin table to plot the network from the lat/long coordinates, generating a more spatially accurate plot.

Compare these two plots to the map of the Nith basin from Tutorial #2.

```
# plot network from rvh file directly
plot(rvh$SBnetwork)
```




```
# create network plot of watershed structure from rvh file
rvn_subbasin_network_plot(rvh$SBtable, labeled=TRUE)
```



One more useful rvh functionality is the simplification of an HRU file. Often in the initial HRU discretization, many HRUs with similar properties and/or very small areas exist that are worthwhile consolidating for the purpose of modelling. The `rvn_rvh_cleanrhus` function accomplishes this.

Try the example in the help file - `?rvn_rvh_cleanhrus`.

RVT File Utilities

The types of rvt files in Raven are plentiful, as rvt files are used in Raven for time series observations, forcing data, gauge data, etc. A handful of highly useful rvt utilities exist in the RavenR package for building these input files. For example, gauge observation data can be written to rvt format from the `tidyhydat` package for scraping Water Survey of Canada data.

Here, we use sample data from the `tidyhydat` function, and write this to an rvt file for use within a Raven model. The commands that would be used to extract data with the `tidyhydat` function is also provided below (but commented out). The sample data is used here to help meet compliance, but you are encouraged to try using the `tidyhydat` commands to try this functionality for yourself.

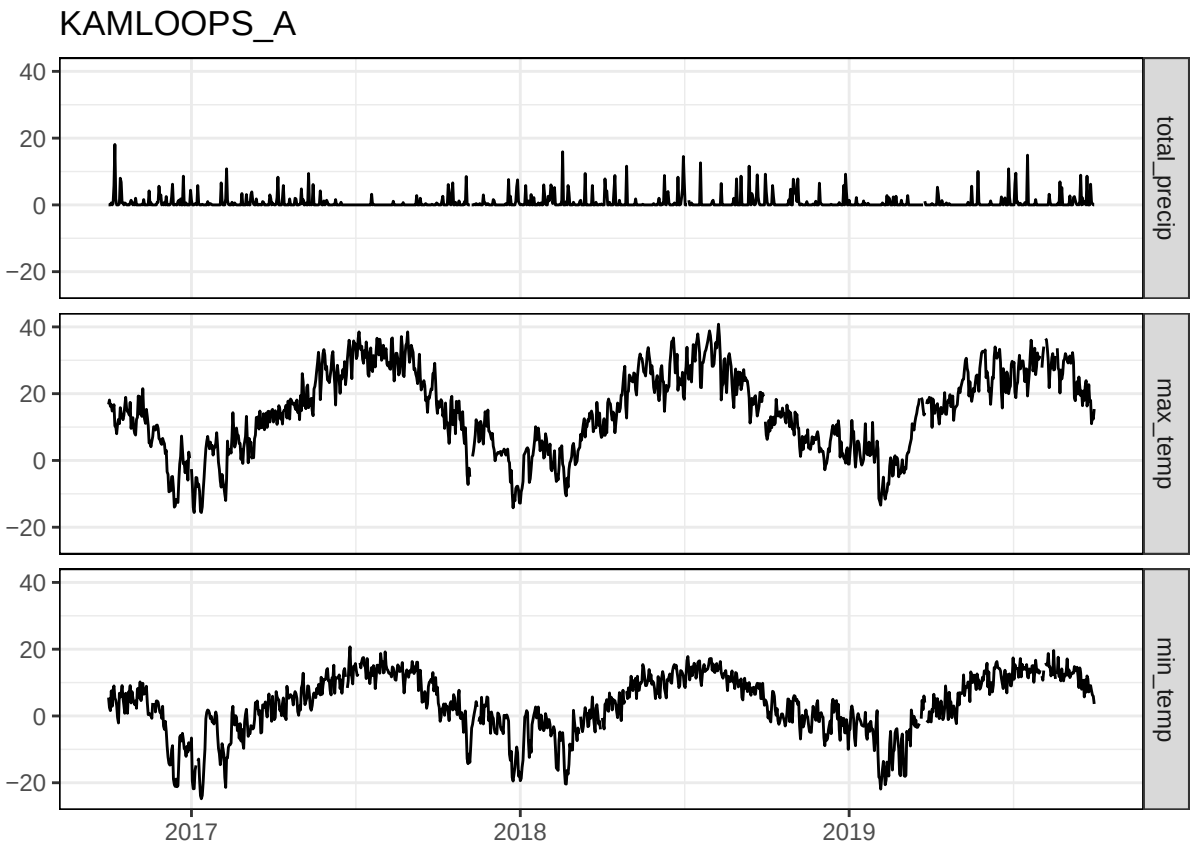
```
stations <- c("05CB004","05CA002")

## Gather station data/info using tidyhydat functions
# library(tidyhydat)
# hd <- tidyhydat::hy_daily_flows(station_number = stations,
#  start_date = "1996-01-01", end_date = "1997-01-01")

## load RavenR package sample data
```

```
data(rvn_tidyhydat_sample)
hd <- rvn_tidyhydat_sample

tf1 <- file.path(tempdir(), "station1.rvt")
tf2 <- file.path(tempdir(), "station2.rvt")

# Create RVT files
rvn_rvt_tidyhydat(hd, subIDs=c(3,11),
  filename=c(tf1,tf2))
```

Similar support for the `weathercan` package to write gauge forcing data exists within the `RavenR` package through the `rvn_rvt_ECmet` function. The `rvn_rvt_ECmet` function currently has two options for writing sets of forcing data. Here, we use the first option to write the total precipitation, maximum daily temperature, and minimum daily temperature from the 'Kamloops A' station to a Raven-compliant rvt format. The station metadata, also required for Raven gauge forcing data, is written to file when `write_stndata` is TRUE. The sample data obtained from the `weathercan` package is used here once again, but you are encouraged to try obtaining the data from the `weathercan` package directly using the commented out commands below.

```
## Obtain data using the weathercan package
# library(weathercan)
# kam <- weather_dl(station_ids = 51423,
#                   start = "2016-10-01", end = "2019-09-30", interval="day")

## load RavenR package sample data
data(rvn_weathercan_sample)
kam <- rvn_weathercan_sample

## basic use, override filename to temporary file
## default forcing_set (PRECIP, MAX TEMP, MIN TEMP)
result <- rvn_rvt_ECmet(metdata = kam, forcing_set = 1,
                        write_stndata = TRUE)
```

```
## [1] "Imported 1 data set"
## [1] "Overview of generated timeseries"
```


KAMLOOPS_A
40
20
0
−20
total_precip
max_temp
min_temp
2017
2018
2019

Exercise 1 - Build a model workflow

Now we will build a simple script which will provide a number of visualizations, which we can use to look at the Nith river model each time we run it. This can be made as complex as you want. You may also choose to run this for a different Raven model as part of this exercise.

Building a model workflow script

Start with a new script. From RStudio, go to the main menu. `Choose File -> New File -> R Script`. Populate the script with the following.

You can find the Nith model files in the Raven Tutorials. Please place a Raven.exe executable (or symbolic link called Raven.exe) inside this Nith folder, and update the modelfolder variable below to the appropriate folder location on your system.

```
modelfolder <- "C:/TEMP/Nith/"  # model folder with Raven.exe and Nith model files
fileprefix <- "Nith"            # prefix for model files (i.e. Nith.rvi should be in the modelfolder)
outdir <- "./output/"

## if this generates an error for you, the Nith folder cannot be found.
# Please update the modelfolder variable accordingly
if (!dir.exists(modelfolder)) {stop(sprintf("The folder %s does not exist, please verify!"))}

setwd(modelfolder)

# RUN RAVEN
# =====================================================
# writes complete command prompt command
# > Raven.exe [filename] -o [outputdir]
RavenCMD <- sprintf("Raven.exe %s -o %s", fileprefix, outdir)
system(RavenCMD) # this runs raven from the command prompt
```

Once the model is run, we can read in the output and save some of the plots to file. Here, we examine the forcing data and the snowpack.

```
# GENERATE OUTPUT PLOTS
# =====================================================
# read in the model output files

## use ggsave from ggplot2 to save plot pdf
ff_data <- rvn_forcings_read(paste0(outdir,"run1_ForcingFunctions.csv"))
myplots <- rvn_forcings_plot(ff_data$forcings)
myplots$AllForcings %>%
  ggsave("Forcings.pdf", ., width = 8.5, height=11, units='in')

# plot snowpack from xts format, save using base R commands
mywshd <- rvn_watershed_read(paste0(outdir, "run1_WatershedStorage.csv"))
png("snowpack.png") # create a png file to direct plot to
plot(mywshd$watershed_storage$Snow,
     main='Snowpack (mm SWE)', col='blue')
dev.off() #finishes writing plot to .png file
```

Modify the script

Modify the above script to generate png image plots of monthly subbasin-averaged PET in Subbasin 43 using the :CustomOutput option (you will have to add a :CustomOutput command to the Raven input rvi file). You will also want to use the RavenR `rvn_custom_read()` and `rvn_custom_output_plot()` commands.

Then, add a set of diagnostic plots related to the Hydrographs.csv file. For example, you may wish to compare annual peaks in the data or flow volumes.

Exercise 2 - Build Raven Input Files

If you have completed the workflow exercise above, you may build on the Introduction to R exercise from the earlier session to build Raven-compliant input files for the Raven River in Alberta. Using your knowledge of R and RavenR, undertake the following tasks using only R and a text file editor.

- build an rvt file of observed hydrograph flows from the Raven River WSC gauge;
- build rvt file(s) for at least one meteorological gauge located close to the WSC gauge;
- create a plot of the model structure used for the Raven River model (in the Tutorial files);
- create a plot of the subbasin network from the rvh file in the Raven River model;
- determine the total upstream area of basin in the model; and
- use the custom output functions to determine the aridity index (P/PET) of the watershed.

Conclusion

This tutorial is meant to serve as a brief introduction to the RavenR package. If you have any comments, suggestions or bug reports, please leave a note on the issues page of the Github project (RavenR Github page), email the authors of the package, or feel free to let us know on the Raven forum. Additional Raven materials can be found on the Raven downloads page.