# Introduction to R & RStudio

New Approach Methods (NAMs) Tools Training Workshop

2024-04-24

## Intro to R

This document is written in R markdown, a type of file that allows you to combine code with commentary. You can create a new Rmarkdown file by going to File -> New File -> R markdown. The code written here can also be written and run in an R script (File -> New File -> R script) or in your consol.

### R Packages

In R, shareable bundles of code, data, and documentation are called packages. Many functions are available in base R, but often times you will need to install a package to access something specific. Many packages are available on Comprehensive R Archive Network, or CRAN, the public clearing house for R packages. Other packages may not be available on CRAN, but able to be downloaded from Github.

#### Installing packages

The most common method is to install a package from CRAN. You can also do this by going to the bottom right screen, find the "Packages" tab, click the "install" button, and search for your package.

```
install.packages("httk")
```

You can install a package from a repository such as GitHub. First, you must install devtools with the usual "install.packages("devtools")"; you may also need to install Rtools if using Windows. Once devtools is installed, you can install other packages:

```
install_git() from a git repository,
install_github() from GitHub,
install_bitbucket() from Bitbucket,
install_version() from a specific version of a CRAN package
```

For example, to install "httk" from Github, use the user/rep form.

```
devtools::install_github("USEPA/CompTox-ExpoCast-httk")
```

Note: If you do not want to install devtools, you can bypass it by using the "remotes" package (first use "install.packages("remotes")")

```
remotes::install_github("USEPA/CompTox-ExpoCast-httk")
```

If you need to install a package stored in another external repository (neither CRAN nor Github), use the following parameters in your install.packages command:

```
install.packages('furrr', repos='http://cran.us.r-project.org', dependencies=TRUE)
```

If you have an R package downloaded on your local machine as a .zip or tar.gz file, you can install it using the install.packages() function and specify the path where the zip file is saved

```
install.packages('C:/Users/User/Downloads/abc_2.1.zip', repos=NULL, type='source')
```

### Dependencies

Dependencies are automatically installed from CRAN. Outdated dependencies are automatically upgraded by default. If you are downloading a package from GitHub, check the page for listed dependencies. If you are missing dependencies for the package you are trying to install, an error message will list which dependencies to install first. Older versions of packages/dependencies can be found by going to https://cran.r-project.org/web/packages, searching for your package, and clicking on "Old sources".

```
# check necessary dependencies for installed packages
pack <- available.packages()
pack["doParallel","Depends"]
```

### Loading Packages

Once you have your package installed (which only needs to happen once), you can load it into your library. You must do this with every new R session.

```
library("httk")
```

```
## Warning: package 'httk' was built under R version 4.3.2
```

Now you can get help with the package or check its version. You can also get help by going to the bottom right window and click the "Help" tab.

```
??httk
packageVersion("httk")
```

## Create a document or script

To create a new script, go to File -> New File -> R script. This opens a blank script. You can write your code in this document and it will save as a .R file. Make sure you save it in the appropriate directory on your computer (perhaps the same directory where you have data saved). If you want to run this script from a separate .R file, you use the "source" function. To ensure you are in the correct working directory, you can navigate to it using the Files tab in the bottom right pane, go to Session -> Set Working Directory -> To Files Pane Location. You can also choose Session -> Set Working Directory -> Choose Directory.

```
source("path/your_script.R")
```

## Base R Commands

Note: Commands can be run using ctrl + enter or the green "Run" button above.

Variables are containers for storing data values and can be assigned using "=" or "<-". A variable be of different data types For example, variables can be numeric, characters (a.k.a strings), or boolean. Note that strings must be enclosed in quotations.

```r
name = "John Doe"
person.name <- "John Doe"

age = 45

is.open = TRUE
```

You can find the class of the variables to tell you if it's a string, numeric, etc.

```r
class(name)
```

```
## [1] "character"
```

```r
class(age)
```

```
## [1] "numeric"
```

```r
class(is.open)
```

```
## [1] "logical"
```

Base R has a range of built-in functions.

```r
# Create vectors
y = c(10.1, 3, 1, 2.3, 1.2)

# built-in functions
max(y)
```

```
## [1] 10.1
```

```r
min(y)
```

```
## [1] 1
```

```r
range(y)
```

```
## [1]  1.0 10.1
```

```
tail(y,1)
```

```
## [1] 1.2
```

```
length(y)
```

```
## [1] 5
```

```
mean(y)
```

```
## [1] 3.52
```

```
sd(y)
```

```
## [1] 3.767891
```

```
runif(n=5, min = 1, max = 10) # runif randomly selects n numbers from a uniform distribution
```

```
## [1] 7.387547 9.842005 1.362561 3.481153 1.540182
```

```
# let's create a hypothetical x-vector to match y
x = seq(1,5,by=1)
```

Create a matrix or dataframe.

```
# combine the vectors into a matrix or dataframe
mat = cbind(x,y)
# rename columns
colnames(mat) = c("time","concentration")

df = data.frame(time = x, concentration = y)
df = data.frame(mat)


# extract elements from the matrix/df using matrix[row, column]
mat[1,2]
```

```
## concentration
##          10.1
```

```
# if the structure is a dataframe, you can use the "$" symbol
df$concentration
```

```
## [1] 10.1  3.0  1.0  2.3  1.2
```

```
df$concentration[1] # extracts first element of "concentration"
```

```
## [1] 10.1
```

Let's consider an actual dataset. R has preloaded data. Notice that R's preloaded data also includes httk's datasets.

```r
# Example: look at the "iris" data set
data()
data("iris")
head("iris")

# If you have not done so already, install and load the httk package
library("httk")
```

Look at one of httk's data sets chem.physical_and_invitro.data. This data set contains information about many chemicals; each row pertains to a single chemical.

```r
# Show the top few rows
head(chem.physical_and_invitro.data)
```

```r
# Find the names of the columns
colnames(chem.physical_and_invitro.data)
```

```
##  [1] "Compound"                           "CAS"
##  [3] "CAS.Checksum"                       "DTXSID"
##  [5] "Formula"                            "All.Compound.Names"
##  [7] "logHenry"                           "logHenry.Reference"
##  [9] "logMA"                              "logMA.Reference"
## [11] "logP"                               "logP.Reference"
## [13] "logPwa"                             "logPwa.Reference"
## [15] "logWSol"                            "logWSol.Reference"
## [17] "MP"                                 "MP.Reference"
## [19] "MW"                                 "MW.Reference"
## [21] "pKa_Accept"                         "pKa_Accept.Reference"
## [23] "pKa_Donor"                          "pKa_Donor.Reference"
## [25] "All.Species"                        "Dog.Foral"
## [27] "Dog.Foral.Reference"                "DTXSID.Reference"
## [29] "Formula.Reference"                  "Human.Caco2.Pab"
## [31] "Human.Caco2.Pab.Reference"          "Human.Clint"
## [33] "Human.Clint.pValue"                 "Human.Clint.pValue.Reference"
## [35] "Human.Clint.Reference"              "Human.Fabs"
## [37] "Human.Fabs.Reference"               "Human.Fgut"
## [39] "Human.Fgut.Reference"               "Human.Fhep"
## [41] "Human.Fhep.Reference"               "Human.Foral"
## [43] "Human.Foral.Reference"              "Human.Funbound.plasma"
## [45] "Human.Funbound.plasma.Reference"    "Human.Rblood2plasma"
## [47] "Human.Rblood2plasma.Reference"      "Monkey.Foral"
## [49] "Monkey.Foral.Reference"             "Mouse.Foral"
## [51] "Mouse.Foral.Reference"              "Mouse.Funbound.plasma"
## [53] "Mouse.Funbound.plasma.Reference"    "Rabbit.Funbound.plasma"
## [55] "Rabbit.Funbound.plasma.Reference" "Rat.Clint"
## [57] "Rat.Clint.pValue"                   "Rat.Clint.pValue.Reference"
## [59] "Rat.Clint.Reference"                "Rat.Foral"
## [61] "Rat.Foral.Reference"                "Rat.Funbound.plasma"
## [63] "Rat.Funbound.plasma.Reference"      "Rat.Rblood2plasma"
## [65] "Rat.Rblood2plasma.Reference"        "SMILES.desalt.Reference"
## [67] "Chemical.Class"
```

```
# Find the size
dim(chem.physical_and_invitro.data)
```

```
## [1] 11232    67
```

You can extract information about a certain chemical using functions such as which() and subset(). Consider a chemical of interest, Chloroform.

```
# which() finds the row number. You can save that row number to a variable named "our.row" (or any othe
our.row = which(chem.physical_and_invitro.data$Compound=="Chloroform")
A = chem.physical_and_invitro.data[our.row,] # we can save this subset into our environment
chem.physical_and_invitro.data[our.row, c("logP","MW")]
```

```
##         logP    MW
## 67-66-3 1.97 119.4
```

```
# or
A[, c("logP","MW")] # gives the same result
```

```
##         logP    MW
## 67-66-3 1.97 119.4
```

```
# subset() allows you to subset columns by row (B returns the same result as A)
B = subset(chem.physical_and_invitro.data, Compound=="Chloroform")
```

Subset dataframe by multiple conditions.

```
# Find all chemicals for which either 2.5 < logP < 3 or logP > 5
special.subset = subset(chem.physical_and_invitro.data, logP > 2.5 & logP < 3 | logP > 5)

# Refine your search further. Say you want to see if any of these chemicals have
# intrinsic clearance values for Rats.
# First, notice how many NA values there are
special.subset = subset(chem.physical_and_invitro.data, logP > 2.5 & logP < 3 | logP > 5)$Rat.Clint

# Filter out NA values: is.na(x) finds values that are NA; ! negates
new_df = subset(chem.physical_and_invitro.data, (logP > 2.5 & logP < 3 | logP > 5) & !(is.na(Rat.Clint))
```

You can find unique values when a table or dataframe may be large or have repeated values. For example, what unique species are in this table?

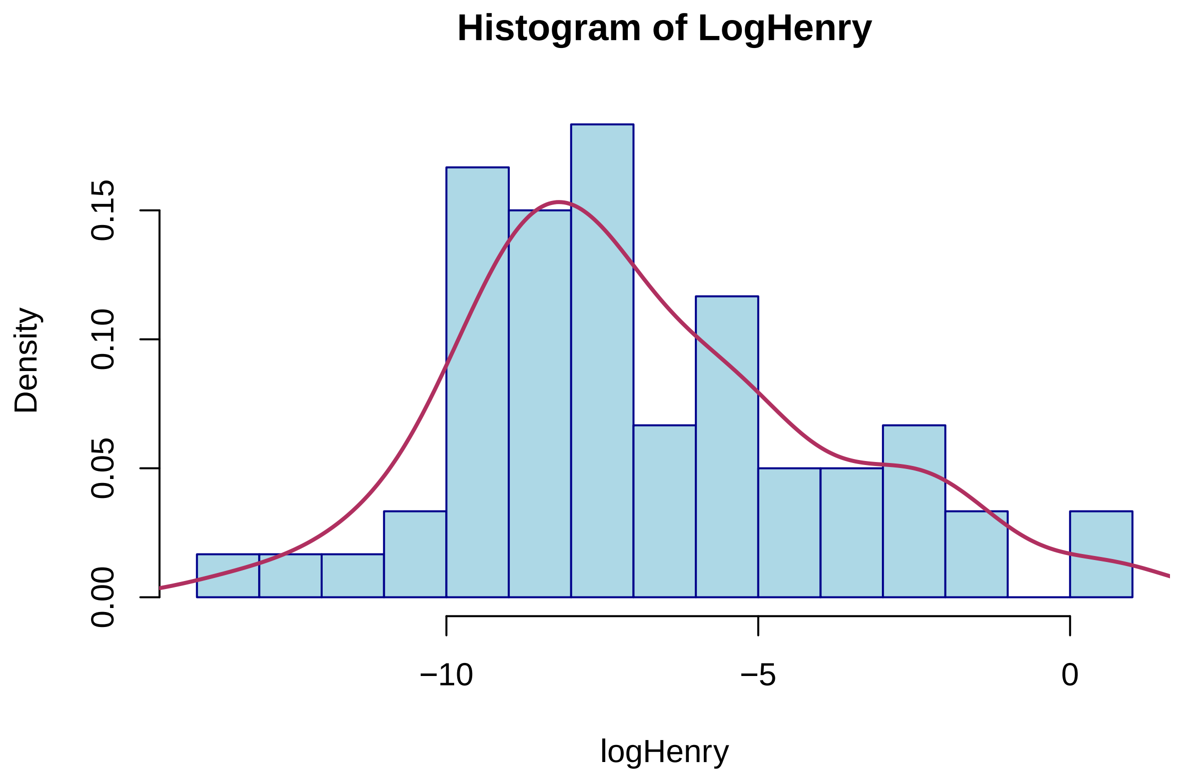
```
unique(chem.physical_and_invitro.data$All.Species)
```

```
##  [1] "Human"                       "Human|Rat"
##  [3] "Human|Rat|Dog|Monkey"        "Human|Dog"
##  [5] "Human|Monkey"                "Human|Rat|Dog"
##  [7] "Human|Mouse|Rat|Dog"         "Human|Rat|Mouse|Dog|Monkey"
##  [9] "Human|Rat|Mouse"             "Human|Rat|Monkey"
## [11] "Human|Rabbit|Dog"            "Rat|Human"
## [13] "Rat"                         "Rabbit|Human"
## [15] "Rat|Human|Mouse|Dog"        "Rat|Human|Dog"
## [17] "Mouse|Rat|Human"            "Human|Mouse|Rat|Dog|Monkey"
## [19] "Human|Mouse|Rat"            "Human|Rat|Mouse|Dog"
## [21] "Human|Mouse"                "None"
## [23] "Rat|Human|Mouse|Monkey"     "Human|Dog|Monkey"
## [25] "Human|Mouse|Dog|Monkey"
```

**Plotting in R**

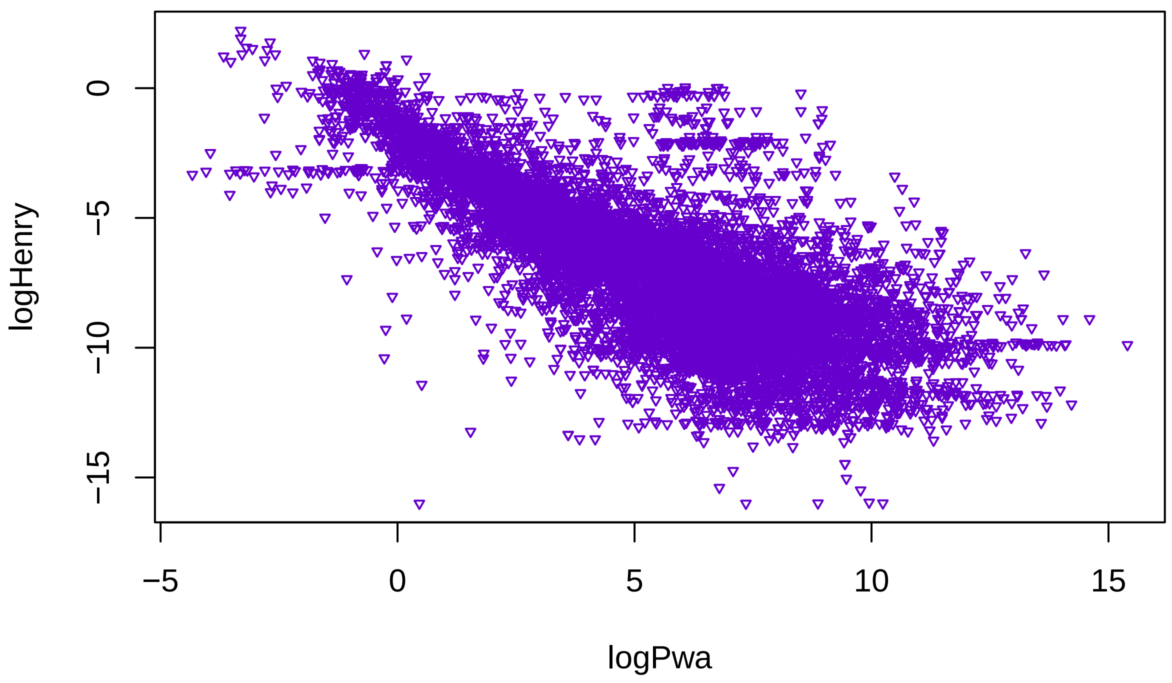
There are many ways to visualize data in R (ggplot2 is a very versatile package), but you can also easily plot in base R. Within base R plotting functions, specifications can be added (such as color). Below is an example of a histogram of the logHenry values from new_df.

```
# histogram of the logP values from new_df dataframe (above)
hist(new_df$logHenry,
     main = "Histogram of LogHenry",      # title
     xlab = "logHenry",                   # x-axis label
     breaks = 20,                         # change number of breaks
     col = "lightblue",                   # color of bars
     border = "darkblue",                 # border color
     prob = TRUE)                         # shows density instead of frequencies
 # lines() overlays a new line onto existing plot
lines(density(new_df$logHenry),           # density() shows density curve
       lwd = 2,                           # set line width
       col = "maroon")
```



You can also use the plot() function to plot numeric data. Plot logPwa (water:air partition coefficent) vs logHenry from chem.physical_and_invitro.data.

```
plot(chem.physical_and_invitro.data$logPwa, chem.physical_and_invitro.data$logHenry,
     xlab = "logPwa",
     ylab = "logHenry",
     pch = 6,                     # choose shape
     col = "#6600cc",             # color of the shape
     cex = 0.5)                   # shape size
```



**Bonus material**

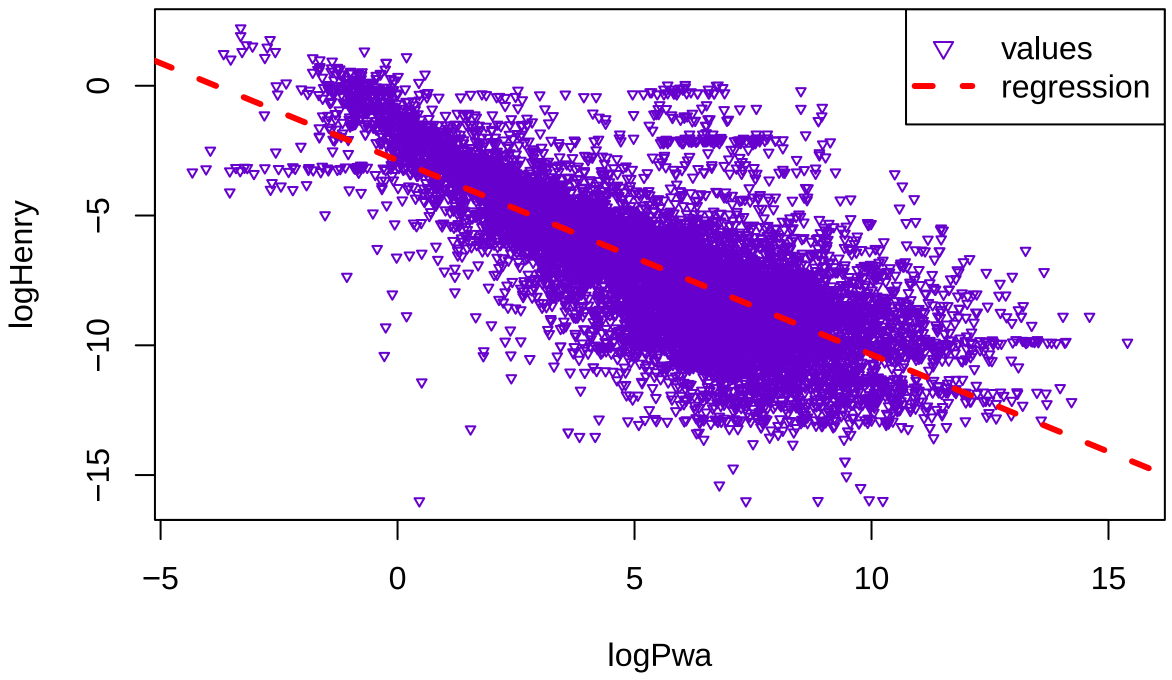
We can perform a linear regression on the data from the plot above using the lm() function.

```
lm(logHenry~logPwa, data = chem.physical_and_invitro.data)
```

```
##
## Call:
## lm(formula = logHenry ~ logPwa, data = chem.physical_and_invitro.data)
##
## Coefficients:
## (Intercept)       logPwa
##     -2.8721      -0.7485
```

To add the regression line to the plot, use abline()

```
plot(chem.physical_and_invitro.data$logPwa, chem.physical_and_invitro.data$logHenry,
     xlab = "logPwa",
     ylab = "logHenry",
     pch = 6,                    # choose shape
     col = "#6600cc",             # color of the shape
     cex = 0.5)                   # shape size
abline(lm(logHenry~logPwa, data = chem.physical_and_invitro.data),
       lwd = 3,
       col = "red",
       lty = 2)
# add a legend
legend("topright",                                 # location of legend
       legend = c("values","regression"),          # text of legend
       pch = c(6, NA),                             # symbols
       col = c("#6600cc","red"),                   # color per symbol
       lty = c(NA, 2),                             # linetype
       lwd = c(NA,3))                              # linewidth
```


values
regression
logHenry
logPwa