

# R Training

A beginner's guide to using R on Sapelo2



# Goals for this Training

1. Have R ready to use on your machine either through Sapelo2 or locally
2. Recognize and create the basic R objects: dataframes, vectors, list
3. Load data into your environment
4. Get quick info on the data through built in functions and graphics
5. Manipulate dataframes and create new dataframes from old
6. Use other's code, i.e packages, to expand your options
7. Save/export data and figures

# Features of the R language?

- 1.High-level language: Intuitive and easier to code.
- 2.R functions are vectorized, which allows for more efficient coding
- 3.Extensible via packages:Machine Learning, Genetics, High
- 4.Performance Computing and genetics
- 5.Graphics
- 6.Comparable to SAS, SPSS, and Stata, but FREE
- 7.Easy creation of documents such as MS-Word, PDF, websites and more.

# Downloading R

The R project website has installers for Microsoft Windows, Apple OSX, and Linux

<https://www.r-project.org/>

Once R is installed, it can be run in the command line.

```
keekov@keekov: ~  
File Edit View Search Terminal Help  
(base) keekov@keekov:~$ R  
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"  
Copyright (C) 2020 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> 
```

## Running on Sapelo2

On Sapelo2, first login to the interactive node using the `qlogin` command. Then enter the command :

```
ml R/4.0.0-foss-2019b
```

to load the R module.

Then type `R` and press enter.

This tutorial uses `xqlogin` for visuals

```
[keekov@c1-7 ~]$ ml R/4.0.0-foss-2019b
[keekov@c1-7 ~]$ R

R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> □
```

# Data types

Object Oriented Language: Everything is an object, which has a type and belongs to a class.

Objects have attributes and methods

Common Data types:

character	"Hello" , "x"
Numeric	1, 30.3, 1/3
Logical	TRUE, FALSE
vector	c(12, 34, 129, 16)
list	list(12, TRUE, "green")

Create an Object with = or <-

# Data Creation

Create an Object with = or <-

```
>
> variable1 = 3
> variable2 = 4
> variable4 <- TRUE
> variable3 <- "Hello"
> List = list(3,4,TRUE,"Hello")
> □
```

Functions to explore data:

class() - what kind of object is it

length() - how long is it?

attributes() - does it have any metadata?

ls() - shows what variables you have already

object.size() - about how many bytes is object?

rm() - removes an object

```
> class(variable1)
[1] "numeric"
> class(variable3)
[1] "character"
> class(variable4)
[1] "logical"
> class(List)
[1] "list"
> length(variable1)
[1] 1
> length(List)
[1] 4
>
```

# Let's make a dataframe

Most common data structure for tabular data is the Dataframe

Combine vectors with `data.frame()`

```
> Animal = c("snake", "chicken", "human")
> NumberOfHands = c(0, 0, 2)
> WarmBlooded = c(FALSE, TRUE, TRUE)
> ExampleDataFrame = data.frame(Animal, NumberOfHands, WarmBlooded)
> ExampleDataFrame
  Animal NumberOfHands WarmBlooded
1  snake              0         FALSE
2 chicken              0          TRUE
3  human              2          TRUE
> 
```

## Add and reference columns with \$

```
> ExampleDataFrame$Animal
[1] snake  chicken human
Levels: chicken human snake
> ExampleDataFrame$MultipleSpecies = c(TRUE,TRUE,FALSE)
> ExampleDataFrame
  Animal NumberOfHands WarmBlooded MultipleSpecies
1  snake             0        FALSE             TRUE
2 chicken             0         TRUE             TRUE
3  human             2         TRUE             FALSE
> 
```

Use ? in front on a function to open documentation

Close the documentation by typing q.

Functions can have arguments(inputs).  
Specify arguments by position, by complete name, or by partial name.

The following are the same:

```
mean(exampdata,0.5,FALSE)
```

```
mean(x=exampdata,trim=0.5,na.rm=FALSE)
```

```
mean(trim=0.5,x=exampdata,na.rm=FALSE)
```

The arguments can only be out of order if they are named!

## Example: ?mean

```
mean                                package:base                        R Documentation
Arithmetic Mean
Description:
  Generic function for the (trimmed) arithmetic mean.
Usage:
  mean(x, ...)
  ## Default S3 method:
  mean(x, trim = 0, na.rm = FALSE, ...)
Arguments:
  x: An R object. Currently there are methods for numeric/logical
    vectors and date, date-time and time interval objects.
    Complex vectors are allowed for 'trim = 0', only.
  trim: the fraction (0 to 0.5) of observations to be trimmed from
    each end of 'x' before the mean is computed. Values of trim
    outside that range are taken as the nearest endpoint.
  na.rm: a logical value indicating whether 'NA' values should be
    stripped before the computation proceeds.
  ...: further arguments passed to or from other methods.
```

# Vectorization and Subsetting

Always try to operate upon vectors when doing repetitive tasks.

In this example `abs_loop` takes the absolute value of each number in a vector.

`Abs_sets` is the vectorized version

`rep(c(-1,1),50000)` creates a vector of length 50000 with values of either -1 or 1.

`system.time()` records the time to run each function

```
> vec <- c(1, -2, 3, -4, 5, -6, 7, -8, 9, -10)
> vec < 0
[1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE
> vec[vec < 0]
[1] -2 -4 -6 -8 -10
> vec*3
[1] 3 -6 9 -12 15 -18 21 -24 27 -30
> []
```

```
> abs_loop <- function(vec){
+   for (i in 1:length(vec)) {
+     if (vec[i] < 0) {
+       vec[i] <- -vec[i]
+     }
+   }
+   vec
+ }
>
>
>
> abs_sets <- function(vec){
+   negs <- vec < 0
+   vec[negs] <- vec[negs] * -1
+   vec
+ }
>
> long <- rep(c(-1, 1), 50000)
> system.time(abs_loop(long))
  user system elapsed
0.012  0.000  0.012
> system.time(abs_sets(long))
  user system elapsed
0.001  0.000  0.001
> []
```

# Interacting with the file system

`getwd()` , `setwd()` - get and set the current directory

`list.files()` - List the files in the current directory or a specified location.

Example: `list.files("/scratch/keekov/testdir")`

`file.remove()`, `file.rename()`, `file.copy()`, `dir.create()` - file manipulation

# Importing data

The most common type of data is csv

`read.delim()` for txt files, `read.csv()` for csv , `read_excel()` for excel.

Data located at `/usr/local/training/RTraining/yf_Data.csv`

This was an experiment infecting immune cells with yellow fever. Three experimental groups are baseline (0 hour), mock infection (6 hours of culture but no virus), and YFV infection (6 hours). Each group included three biological samples, and each biological sample was run three times on a mass spectrometer coupled with liquid chromatography.

Sample `mock_6hr_03_3` is mock infection, sample 3, and second of three runs(labeled 1,3,5)

# Quick Analysis

Useful Data Frame Functions:

`head()` - shows first 6 rows

`tail()` - shows last 6 rows

`dim()` - returns the dimensions of data frame

`ncol()` - number of columns

`str()` - structure of data frame - name, type and preview of data in each column

`names()` or `colnames()` - both show the names attribute for a data frame

`table()` - builds a contingency table of a column

# Checking data distribution in samples

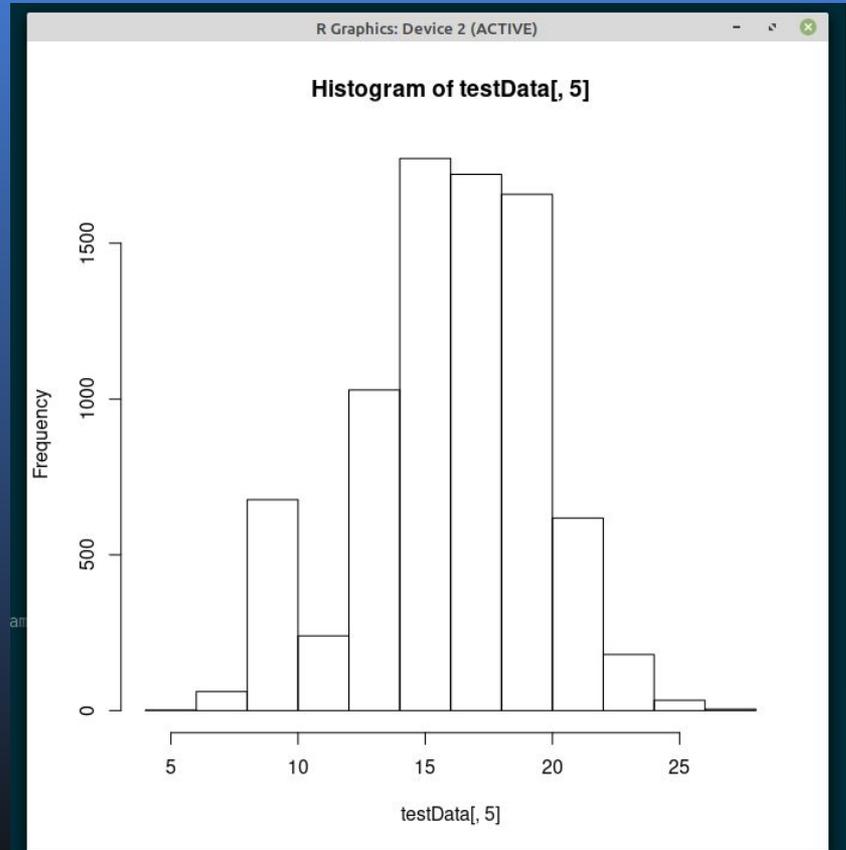
Subset using brackets:

`Dataframe[rows,columns]`

Plot Intensity distribution of first sample  
with `hist()`

`hist(yfData[,5])`

The data after log transformation should  
approximate a normal distribution



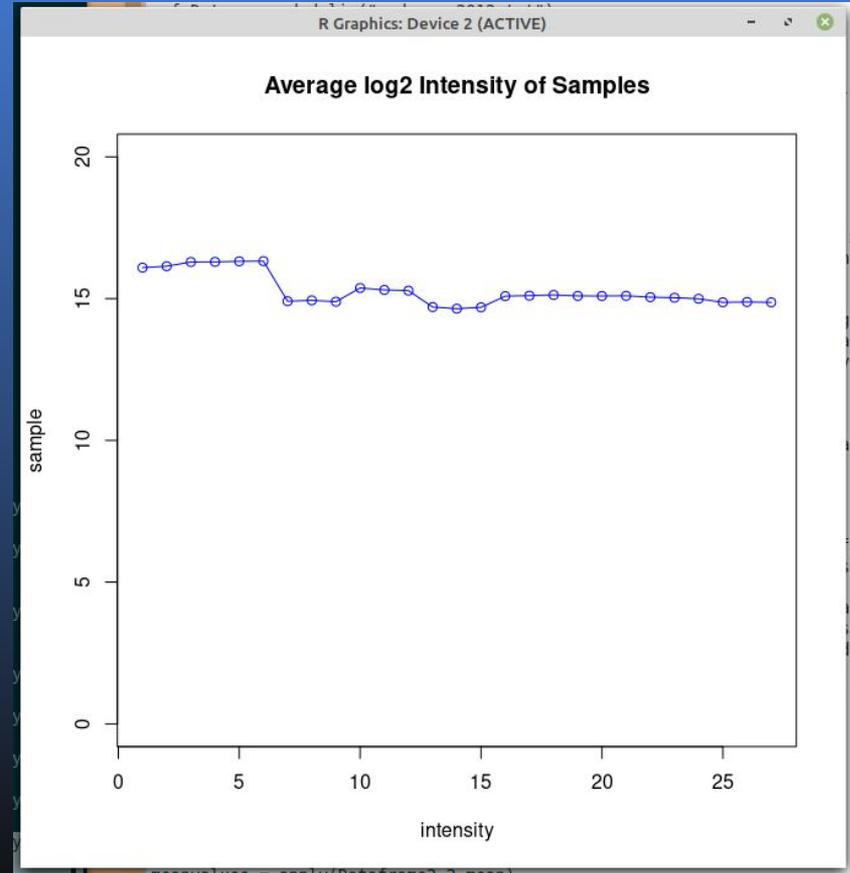
# Check data quality by average ion intensity in each sample

Use `apply()` to apply a function to rows  
or columns of a Dataframe

```
yf_Data2=yf_Data[,5:ncol(yf_Data)]
```

```
yf_means = apply(yf_Data2,2,mean)
```

```
plot(yf_means,ylim=c(0,20),  
main="Average log2 Intensity of Samples",  
xlab="intensity",ylab =  
"sample",type="o",col="blue")
```



# Packages

R packages are collections of functions and data sets developed by the community.

R installs a set of packages during installation. foundational or core packages offer basic functionality in terms of data manipulation, statistical tests, analysis, and visualization and are loaded when starting R.

Other packages have to be installed and loaded explicitly.

**On Sapelo:** you can request a package be installed or use your home directory:

[https://wiki.gacrc.uga.edu/wiki/Installing\\_Applications\\_on\\_Sapelo2#How\\_to\\_install\\_R\\_packages](https://wiki.gacrc.uga.edu/wiki/Installing_Applications_on_Sapelo2#How_to_install_R_packages)

**Locally:** `install.packages("Package Name")` followed by `library("package Name")`

Bioconductor is managed separately → installation of packages involves a different process

# ggplot2

```
datayf_means =  
as.data.frame(yf_means)
```

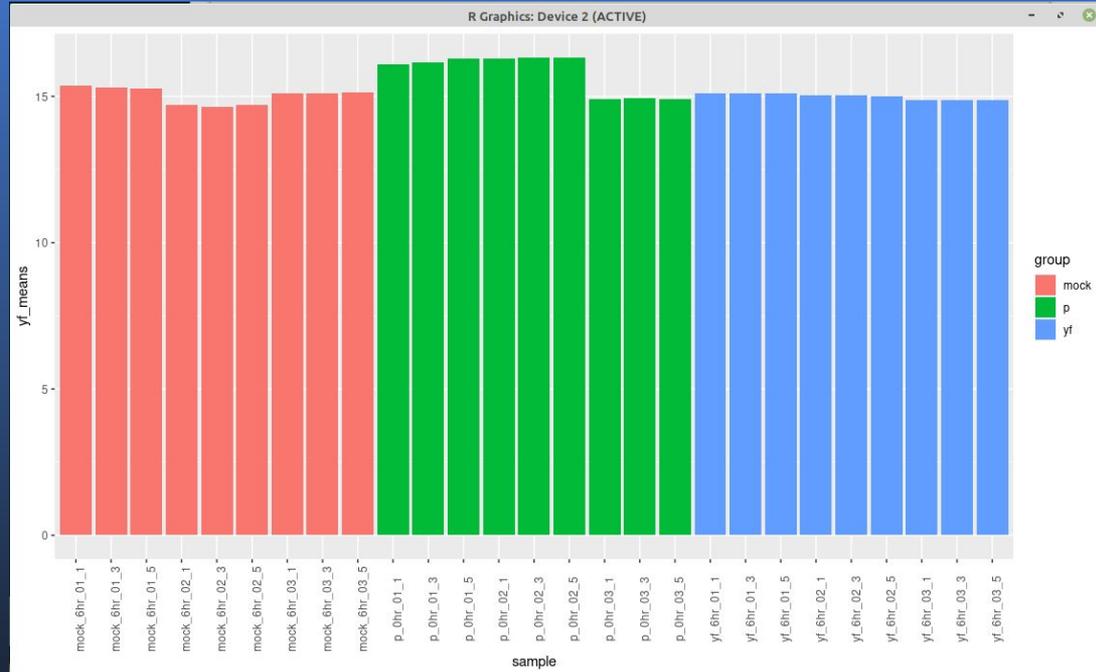
```
datayf_means$sample =  
rownames(datayf_means)
```

```
datayf_means$group =  
c(rep("p",9),rep("mock",9),rep("yf",9))
```

```
ggplot(data=datayf_means,aes(y=yf_  
means,x=sample,fill=group))+
```

```
+ geom_bar(stat="identity")+
```

```
+ theme(axis.text.x =  
element_text(angle = 90))
```



# Exporting Figures and Data

Figure output types include PDF, JPEG, PNG, SVG

Saving pdf Example:

```
pdf("rplot.pdf")
```

```
hist(yfData[,5])
```

```
dev.off()
```

`write.csv()` saves data as `.csv`

The workspace is your current R working environment and includes any user-defined objects. At the end of an R session, the user can save an image of the current workspace. They can also save their workspace or load a workspace using

```
save.image()
```

```
load("myfile.RData") will be hidden file! Show with ls -a
```

Includes console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

UI for connecting to git, managing packages and project management.

R studio desktop on Sap2

The screenshot displays the RStudio interface with the following components:

- Script Editor:** Contains R code for loading data, summarizing it, and creating a faceted scatter plot.
- Console:** Shows the execution output, including summary statistics for 'price' and the execution of the plotting commands.
- Workspace:** Lists the loaded data object 'diamonds' (53940 observations) and the function 'format.plot'.
- Plot:** A scatter plot titled 'Diamond Pricing' showing Price vs. Carat, faceted by Clarity (I1, SI2, SI1, VS2, VS1, VVS2, VVS1, IF).

```
1 library(ggplot2)
2 source("plots/formatPlot.R")
3
4 view(diamonds)
5 summary(diamonds)
6
7 summary(diamonds$price)
8 aveSize <- round(mean(diamonds$carat), 4)
9 clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12            data=diamonds, color=clarity,
13            xlab="Carat", ylab="Price",
14            main="Diamond Pricing")
15
```

```
> summary(diamonds$price)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  326    950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+           data=diamonds, color=clarity,
+           xlab="Carat", ylab="Price",
+           main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
```

x	y	z
Min. : 0.000	Min. : 0.000	Min. : 0.000
1st Qu.: 4.710	1st Qu.: 4.720	1st Qu.: 2.910
Median : 5.700	Median : 5.710	Median : 3.530
Mean : 5.731	Mean : 5.735	Mean : 3.539
3rd Qu.: 6.540	3rd Qu.: 6.540	3rd Qu.: 4.040
Max. :10.740	Max. :58.900	Max. :31.800

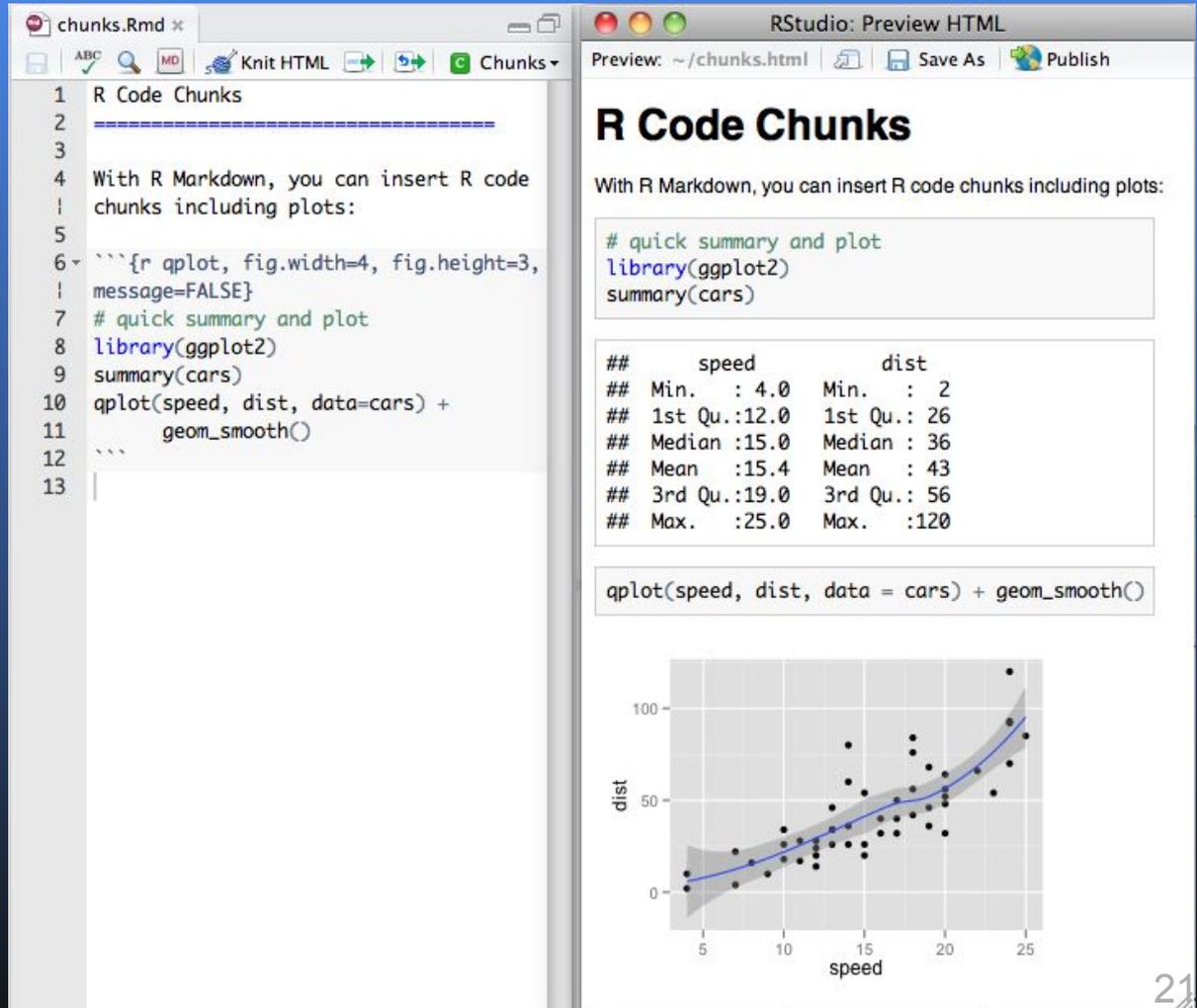
Price

Carat

Clarity

- I1
- SI2
- SI1
- VS2
- VS1
- VVS2
- VVS1
- IF

R markdown supports outputs such as HTML, PDFs, MS-Word documents, applications, websites and more.



The image shows a side-by-side comparison of R code in a source editor and its rendered HTML output in a preview window.

**Source Editor (Left):** The file is named 'chunks.Rmd'. It contains the following code:

```
1 R Code Chunks
2 -----
3
4 With R Markdown, you can insert R code
5 chunks including plots:
6
7 ```{r qplot, fig.width=4, fig.height=3,
8 | message=FALSE}
9 # quick summary and plot
10 library(ggplot2)
11 summary(cars)
12 qplot(speed, dist, data=cars) +
13 |   geom_smooth()
14 ```
```

**Preview Window (Right):** The window is titled 'RStudio: Preview HTML'. It shows the rendered HTML output of the code above.

**HTML Output:**

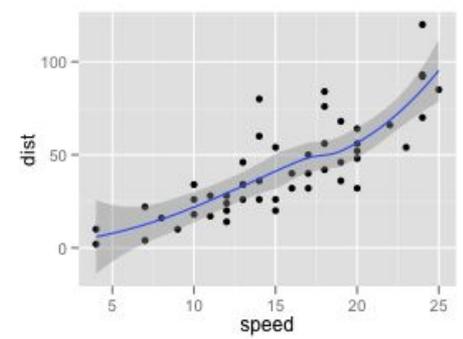
## R Code Chunks

With R Markdown, you can insert R code chunks including plots:

```
# quick summary and plot
library(ggplot2)
summary(cars)
```

##	speed	dist
##	Min. : 4.0	Min. : 2
##	1st Qu.:12.0	1st Qu.: 26
##	Median :15.0	Median : 36
##	Mean :15.4	Mean : 43
##	3rd Qu.:19.0	3rd Qu.: 56
##	Max. :25.0	Max. :120

```
qplot(speed, dist, data = cars) + geom_smooth()
```



Example Script located at `/usr/local/training/RTraining/Rsub.sh`