R Workshop Part 1: Base R

Tiffany Dias

2/22/2019

R Basics

Topics

0. Getting Started and How to Get Help
1. R Objects and Data Types
2. Math
3. Basic Statistical Analysis
4. Loops and Functions
5. Data Manipulation

PART 0: Getting Started and How to Get Help

What is RMarkdown? It is a way to include both formatted text and R code chunks in a single document. You can knit the output to a HTML, PDF, or Word file. For PDF files, you will need to install LaTeX, which can be found here: https://www.latex-project.org/get/

You can use the "Help" tab on the bottom right panel in RStudio. You can also use "?" before the function name, such as ?View

```
`?`(View)
```

You can also find help for a particular package by using: help(package = 'packagename').

```
help(package = "dplyr")
help(package = "ggplot2")
```

Sometimes, googling can be more useful than the official R documentation. Stackexchange, stackoverflow, and r-bloggers are all great resources.

1. The Working Directory

Your working directory is where your output and input files can be found. If you have an error reading in a file or finding your output files, check to see if the file is in your working directory.

You can check where your working directory is and set it using the following commands:

```
getwd()  #get the name of your working directory
```

```
## [1] "/Users/Rose/Desktop/Chen_lab/labdocs/R_workshop"
```

```
# setwd('C://file/path/goes/here')
```

2. Installing and Loading Packages

You can install and load a package by using the following commands.

```
# install install.packages('dplyr')
# install.packages('ggplot2')

# load packages
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

Packages do not need to be reinstalled each time you use them, BUT they do need to be loaded into your current session before you can use them.

3. Reading, Writing, and Viewing Data

If you want to import data from a website or file and assign it to a variable, you can use the following code (uncomment and insert the filename/website url in quotes):

```
# dataset <- read.csv('put your filename/website url') can
# also use 'read.table()' to read in a file in table format
# and create a data frame use stringsAsFactors = FALSE for
# read.table if you want the data imported as strings instead
# of factors
```

You can also assign row and column names to the dataset.

```
somedata <- c(22, 27, 13, 18, 15, 16)  #dataset
somedata
```

```
## [1] 22 27 13 18 15 16
```

```
dim(somedata) = c(3, 2)  #make into a table with 2 rows and 2 columns
colnames(somedata) <- c("C1", "C2")  #set column names
rownames(somedata) <- c("R1", "R2", "R3")  #set row names
somedata
```

```
##    C1 C2
## R1 22 18
## R2 27 15
## R3 13 16
```

```
somedata <- t(somedata)
somedata
```

```
##    R1 R2 R3
## C1 22 27 13
```

```
## C2 18 15 16
```

To practice manipulating data, you can also use one of the built-in datasets. To view all available datasets you can use "data()"

```
# data() #use to view built-in datasest
data(airquality)  #load in airquality dataset
head(airquality)  #see the first few data entries of the air quality dataset
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
nrow(airquality)  #output is number of rows in the dataset
```

```
## [1] 153
```

```
ncol(airquality)  #output is number of columns in the dataset
```

```
## [1] 6
```

```
View(airquality)  #view data
```

You can write data frames to a file using "write.table()"

```
write.table(somedata, file = "somedata.txt", quote = F)  #space delimited; can use `sep=` for a differe
# you can also use save() for writing tables
```

If successful, you should see the somedata.txt file appear in your working directory.

PART 1: R Objects and Data Types

1. Vectors

Vectors can be created using the following code:

```
v1 <- 1  #for one element
v2 <- c(5, 6, 7, 8, 9, 10)  #use c() for more than one element

v1
```

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

```
v2
```

```
## [1]  5  6  7  8  9 10
```

```
v2[2]  #access the second element in v2
```

```
## [1] 6
```

```
v2[-2]  #access all elements except the 2nd in v2
```

```
## [1]  5  7  8  9 10
```

```
v2[2:4]  #access 2nd-4th elements in v2
```

```
## [1] 6 7 8
```

```
v2[c(2, 4)]  #access the 2nd and 4th elements in v2
```

```
## [1] 6 8
```

To check the class of an object use "class()"

```
v <- c(1, 10, 2)
class(v)
```

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

```
l <- c(TRUE, FALSE, FALSE)
class(l)
```

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

```
b <- c("red", "orange", "yellow")
class(b)
```

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

You can also sort data using "sort()"

```
x <- c(5, 2, 9, 3)
x
```

```
## [1] 5 2 9 3
```

```
x <- sort(x)
x
```

```
## [1] 2 3 5 9
```

2. Lists

Lists can contain many different types of elements, including vectors, functions, and other lists.

```
lst1 <- list("character", 35.3, list("this", "is", "a", "list"))

lst1
```

```
## [[1]]
## [1] "character"
##
## [[2]]
## [1] 35.3
##
## [[3]]
## [[3]][[1]]
## [1] "this"
##
## [[3]][[2]]
## [1] "is"
##
## [[3]][[3]]
## [1] "a"
```

```
##
## [[3]][[4]]
## [1] "list"
```

```
# accessing info in lists
lst1[1]  #pulls out the first element of the list as a 1 item long list
```

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

```
lst1[[1]]  #pulls out the first element of the list as the original data type (in this case, character)
```

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

```
lst1[[3]][[1]]  #access the first element of the 3rd element of the list
```

```
## [1] "this"
```

```
lst1[[3]]  #pulls out the 3rd element of the list as the original data type (in this case, list)
```

```
## [[1]]
## [1] "this"
##
## [[2]]
## [1] "is"
##
## [[3]]
## [1] "a"
##
## [[4]]
## [1] "list"
```

3. Matrices

Matrices can be created in a similar way as vectors:

```
rownam <- c("one", "two", "three")
colnam <- c("one", "two", "three")
mat1 <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3,
    byrow = TRUE, dimnames = list(rownam, colnam))
mat2 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)

mat1
```

```
##       one two three
## one     1   2     3
## two     4   5     6
## three   7   8     9
```

```
mat2
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
mat1[1, ]  #row 1
```

```
##   one   two three
```

```
##     1     2     3
```

```
mat1[, 1]  #column 1
```

```
##   one   two three
##     1     4     7
```

```
mat1[1, 1]  #row 1, column 1 cell
```

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

```
t(mat1)  #transpose the matrix
```

```
##       one two three
## one     1   4     7
## two     2   5     8
## three   3   6     9
```

Matrices must contain all one data type. For example, you can have an all-integer matrix or an all-character matrix, but you can't have a matrix where some elements are integers and other elements are characters (for that you need a data frame.)

4. Arrays

Arrays are like matrices, but don't have to be 2-dimensional.

```
a <- array(c("red", "orange", "yellow"), c(1, 2, 3))  #creates 3 1x2 matrices
a
```

```
## , , 1
##
##      [,1]  [,2]
## [1,] "red" "orange"
##
## , , 2
##
##      [,1]     [,2]
## [1,] "yellow" "red"
##
## , , 3
##
##      [,1]     [,2]
## [1,] "orange" "yellow"
```

5. Factors

Factors are a type of categorical data that store vectors with the values of its elements as labels.

```
colors <- c("red", "yellow", "blue", "blue", "red", "green",
    "blue")
factorofcolors <- factor(colors)  #create a factor from vector; you can look at the environment panel t

colors
```

```
## [1] "red"    "yellow" "blue"   "blue"   "red"    "green"  "blue"
```

```
factorofcolors
```

```
## [1] red    yellow blue   blue   red    green  blue
## Levels: blue green red yellow
```

```
nlevels(factorofcolors)  #number of levels
```

```
## [1] 4
```

```
summary(factorofcolors)  #print summary
```

```
##   blue  green    red yellow
##      3      1      2      1
```

6. Data Frames

Data frames are like tables. They are similar to matrices, but unlike matrices, each column can hold a different type of data. As vectors are to lists, matrices are to data frames.

```
mice <- data.frame(color = c("black", "brown", "white", "brown"),
    weight = c(20, 25, 18, 27), age = c(5, 6, 4, 5))  #can also use stringsAsFactors = FALSE to have co

mice
```

```
##   color weight age
## 1 black     20   5
## 2 brown     25   6
## 3 white     18   4
## 4 brown     27   5
```

```
mousenames <- c("Mouse 1", "Mouse 2", "Mouse 3", "Mouse 4")
rownames(mice) <- mousenames  #rename rows
colnames(mice)[colnames(mice) == "weight"] <- "weight_grams"
names(mice)[3] <- "age_months"
mice
```

```
##         color weight_grams age_months
## Mouse 1 black           20          5
## Mouse 2 brown           25          6
## Mouse 3 white           18          4
## Mouse 4 brown           27          5
```

```
newrow <- list("white", 22, 4)  #data for a new row
mice <- rbind(mice, newrow)  #add a new row; you can use cbind() instead of rbind() to add a new column
mousenames <- c(mousenames, "Mouse 5")
attr(mice, "row.names") <- mousenames  #add row name to data frame; another way to rename rows
mice
```

```
##         color weight_grams age_months
## Mouse 1 black           20          5
## Mouse 2 brown           25          6
## Mouse 3 white           18          4
## Mouse 4 brown           27          5
## Mouse 5 white           22          4
```

```
mice <- mice[-c(3), ]  #remove row 3
mice
```

```
##         color weight_grams age_months
## Mouse 1 black           20          5
## Mouse 2 brown           25          6
## Mouse 4 brown           27          5
## Mouse 5 white           22          4
```

If you read in data using `read.csv()` or `read.table()`, it will be stored in a data frame.

You can access elements of the data frame in various ways:

```
mice[1:2]  #columns 1-2 of the mice dataset
```

```
##         color weight_grams
## Mouse 1 black           20
## Mouse 2 brown           25
## Mouse 4 brown           27
## Mouse 5 white           22
```

```
mice[c("color", "age_months")]  #access columns based on variable name
```

```
##         color age_months
## Mouse 1 black          5
## Mouse 2 brown          6
## Mouse 4 brown          5
## Mouse 5 white          4
```

```
mice$color  #access column based on variable name
```

```
## [1] black brown brown white
## Levels: black brown white
```

When in doubt about what kind of data you're manipulating, you can use the following ways to get more info:

```
str(mice)  #structure of data frame
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ color       : Factor w/ 3 levels "black","brown",..: 1 2 2 3
##  $ weight_grams: num  20 25 27 22
##  $ age_months  : num  5 6 5 4
```

```
class(mice$color)  #class of object
```

```
## [1] "factor"
```

```
length(mice)  #length
```

```
## [1] 3
```

```
attributes(mice)  #attributes of the dataset
```

```
## $names
## [1] "color"        "weight_grams" "age_months"
##
## $row.names
## [1] "Mouse 1" "Mouse 2" "Mouse 4" "Mouse 5"
##
## $class
## [1] "data.frame"
```

```
sapply(mice, class)  #show the class of each column in the mice dataset
```

```
##        color weight_grams   age_months
##     "factor"    "numeric"    "numeric"
```

```
is.numeric(mice$weight)
```

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

```
class(mice$color)
```

```
## [1] "factor"
```

```
is.character(mice$color)
```

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

```
is.vector(mice$color)
```

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

```
is.matrix(mice)
```

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

```
is.data.frame(mice)
```

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

You can also convert between data types.

```
x <- c(1, 2, 3, 4, 5)
y <- c(6, 7, 8, 9, 10)
mat1 <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3,
    byrow = TRUE)

x
```

```
## [1] 1 2 3 4 5
```

```
y
```

```
## [1]  6  7  8  9 10
```

```
z <- c(x, y)  #combine vectors x and y into one long vector
z
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
mat1
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
z <- as.vector(mat1)  #converts matrix to a long vector (adds by column)
z
```

```
## [1] 1 4 7 2 5 8 3 6 9
```

```
z <- as.vector(t(mat1))  #converts matrix to a long vector (adds by row)
z
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
mat2 <- cbind(x, y)  #convert vectors into matrix (x and y are columns)
mat2
```

```
##      x  y
## [1,] 1  6
## [2,] 2  7
## [3,] 3  8
## [4,] 4  9
## [5,] 5 10
```

```
mat2 <- rbind(x, y)  #convert vectors into matrix (x and y are rows)
mat2
```

```
##   [,1] [,2] [,3] [,4] [,5]
## x    1    2    3    4    5
## y    6    7    8    9   10
```

```
df <- data.frame(x, y)  #convert vectors to data.frame
df
```

```
##   x  y
## 1 1  6
## 2 2  7
## 3 3  8
## 4 4  9
## 5 5 10
```

```
df2 <- as.data.frame(mat2)  #convert matrix into data.frame
df2
```

```
##   V1 V2 V3 V4 V5
## x  1  2  3  4  5
## y  6  7  8  9 10
```

```
mat2 <- as.matrix(df2)  #convert data.frame into matrix
mat2
```

```
##   V1 V2 V3 V4 V5
## x  1  2  3  4  5
## y  6  7  8  9 10
```

PART 2: Math

1. Operation on Vectors

If you add/subtract two vectors of equal sizes together, the operation is carried out for the 2 elements in the same position of each vector. For example:

```
x <- c(1, 4, 7)
y <- c(2, 3, 8)

x + y
```

```
## [1]  3  7 15
```

When there is a difference in the lengths of the vectors, the elements in the shorter vector are cycled through to match the length of the longer vector.

```r
x <- c(2, 4, 6, 8, 10)
y <- c(1, 3, 5)

x + y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

```
## [1]  3  7 11  9 13
```

```r
x <- c(2, 4, 6, 8, 10)
y <- c(1, 3, 5)

x * y
```

```
## Warning in x * y: longer object length is not a multiple of shorter object
## length
```

```
## [1]  2 12 30  8 30
```

```r
mat1 <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3,
    byrow = TRUE, dimnames = list(rownam, colnam))
mat2 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)

mat1 * mat2  #element by element
```

```
##       one two three
## one     1   4     9
## two    16  25    36
## three  49  64    81
```

```r
mat2 <- matrix(1:3, nrow = 3, ncol = 1, byrow = TRUE)
mat1
```

```
##       one two three
## one     1   2     3
## two     4   5     6
## three   7   8     9
```

```r
mat2
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```r
mat1 %*% mat2  #matrix multiplication - number of rows of the 2nd matrix needs to equal the number of c
```

```
##       [,1]
## one     14
## two     32
## three   50
```

```r
# mat2%*%mat1 #error - non-conformable
```

2. Basic Statistical Functions

Here are some basic statistical functions in R:

```r
mean(airquality$Temp)  #mean
```

```
## [1] 77.88235
```

```r
median(airquality$Temp)  #median
```

```
## [1] 79
```

```r
range(airquality$Temp)  #range (highest and lowest values)
```

```
## [1] 56 97
```

```r
min(airquality$Temp)  #minimum
```

```
## [1] 56
```

```r
max(airquality$Temp)  #maximum
```

```
## [1] 97
```

```r
sd(airquality$Temp)  #standard deviation
```

```
## [1] 9.46527
```

```r
var(airquality$Temp)  #variance
```

```
## [1] 89.59133
```

```r
quantile(airquality$Temp, 0.5)  #quantile, where 0.5 = 50%
```

```
## 50%
##  79
```

```r
summary(airquality$Temp)  #summary: min, 1st quartile, median, mean, 3rd quartile, and max
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   56.00   72.00   79.00   77.88   85.00   97.00
```

PART 3: Basic Statistical Analysis

1. Statistical Tests

```r
# Z-TEST
xbar <- 15.8
n <- 40
sigma <- 0.4
mu <- 16
pnorm(xbar, mu, sigma/sqrt(n))  #1-sided z-test (lower); for upper, subtract from 1
```

```
## [1] 0.0007827011
```

```r
2 * pnorm(-abs((xbar - mu)/(sigma/sqrt(n))))  #2-sided z-test
```

```
## [1] 0.001565402
```

```r
# PAIRED/UNPAIRED T-TEST
before <- c(200.1, 190.9, 192.7, 213, 241.4, 196.9, 172.2, 185.5,
    205.2, 193.7)
after <- c(392.9, 393.2, 345.1, 393, 434, 427.9, 422, 383.9,
```

```
    392.3, 352.2)
t.test(x = before, y = after, alternative = "less", paired = T)  #paired t-test; for unpaired set 'pair
```

```
##
##  Paired t-test
##
## data:  before and after
## t = -20.883, df = 9, p-value = 3.1e-09
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##       -Inf -177.4177
## sample estimates:
## mean of the differences
##                  -194.49
```

```
# CHI-SQUARED TEST
categories <- c(100, 120, 140, 170, 180, 300)
nullprob <- c(0.1, 0.2, 0.1, 0.2, 0.2, 0.2)
chisq.test(categories, p = nullprob)  #goodness-of-fit
```

```
##
##  Chi-squared test for given probabilities
##
## data:  categories
## X-squared = 103.37, df = 5, p-value < 2.2e-16
```

```
observed <- matrix(c(10, 20, 30, 20, 25, 31, 35, 40), nrow = 4)
chisq.test(observed)  #independence
```

```
##
##  Pearson's Chi-squared test
##
## data:  observed
## X-squared = 3.7441, df = 3, p-value = 0.2905
```

```
# WILCOXON TESTS
wilcox.test(before, after, paired = T)  #signed rank; for rank sum set 'paired' equal to F
```

```
##
##  Wilcoxon signed rank test
##
## data:  before and after
## V = 0, p-value = 0.001953
## alternative hypothesis: true location shift is not equal to 0
```

```
# ANOVA
model1 = aov(airquality$Temp ~ airquality$Month)
summary(model1)
```

```
##                   Df Sum Sq Mean Sq F value   Pr(>F)
## airquality$Month   1   2413  2413.0   32.52 6.03e-08 ***
## Residuals        151  11205    74.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(model1)
```

```
## Analysis of Variance Table
```

```
##
## Response: airquality$Temp
##                   Df Sum Sq Mean Sq F value    Pr(>F)
## airquality$Month   1   2413  2413.0  32.519 6.026e-08 ***
## Residuals        151  11205    74.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2. Linear Regression

```
lm1 <- lm(airquality$Temp ~ airquality$Month)  #lm(y~x)
summary(lm1)
```

```
##
## Call:
## lm(formula = airquality$Temp ~ airquality$Month)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -20.5263  -6.2752   0.9121   6.2865  17.9121
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       58.2112     3.5191  16.541  < 2e-16 ***
## airquality$Month   2.8128     0.4933   5.703 6.03e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.614 on 151 degrees of freedom
## Multiple R-squared:  0.1772, Adjusted R-squared:  0.1717
## F-statistic: 32.52 on 1 and 151 DF,  p-value: 6.026e-08
```

```
anova(lm1)
```

```
## Analysis of Variance Table
##
## Response: airquality$Temp
##                   Df Sum Sq Mean Sq F value    Pr(>F)
## airquality$Month   1   2413  2413.0  32.519 6.026e-08 ***
## Residuals        151  11205    74.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
cor(airquality$Temp, airquality$Wind)
```

```
## [1] -0.4579879
```

3. Identifying Outliers

```
q1 <- quantile(airquality$Wind, 0.25)
q3 <- quantile(airquality$Wind, 0.75)

lowerfence <- q1 - 1.5 * (q3 - q1)
```

```r
upperfence <- q3 + 1.5 * (q3 - q1)

upperoutliers <- airquality[airquality$Wind >= upperfence, ]
loweroutliers <- airquality[airquality$Wind <= lowerfence, ]

cat("Upper Outliers: ", paste0(upperoutliers$Wind, collapse = ","),
    fill = TRUE)
```

```
## Upper Outliers:  20.1,18.4,20.7
```

```r
cat("Lower Outliers: ", paste0(loweroutliers$Wind, collapse = ","))
```

```
## Lower Outliers:
```

PART 4: Loops and Functions

1. If/Else

```r
i <- c(1, 2, 3, 4, 5)

if (3 %in% i) {
    print("yes")
} else {
    print("no")
}
```

```
## [1] "yes"
```

2. For

```r
for (i in 1:4) {
    j <- i + 10
    print(j)
}
```

```
## [1] 11
## [1] 12
## [1] 13
## [1] 14
```

3. While

```r
i <- 3
while (i < 5) {
    print(i)
    i <- i + 1
}
```

```
## [1] 3
## [1] 4
```

4. Functions

```
multiply <- function(x, y) {
    product <- x * y
    return(product)
}

multiply(3, 7)
```

```
## [1] 21
```

PART 5: Data Manipulation

When you have a data frame, you can access a column using "$". You can also access data as a vector using brackets

```
aqtemp <- airquality$Temp  #get data in the 'Temp' column in the 'airquality' dataset
aqtemp
```

```
##   [1] 67 72 74 62 56 66 65 59 61 69 74 69 66 68 58 64 66 57 68 62 59 73 61
##  [24] 61 57 58 57 67 81 79 76 78 74 67 84 85 79 82 87 90 87 93 92 82 80 79
##  [47] 77 72 65 73 76 77 76 76 76 75 78 73 80 77 83 84 85 81 84 83 83 88 92
##  [70] 92 89 82 73 81 91 80 81 82 84 87 85 74 81 82 86 85 82 86 88 86 83 81
##  [93] 81 81 82 86 85 87 89 90 90 92 86 86 82 80 79 77 79 76 78 78 77 72 75
## [116] 79 81 86 88 97 94 96 94 91 92 93 93 87 84 80 78 75 73 81 76 77 71 71
## [139] 78 67 76 68 82 64 71 81 69 63 70 77 75 76 68
```

```
aqdata13 <- airquality[1, 3]  #access data in row 1 column 3
aqdata13
```

```
## [1] 7.4
```

You can also save a subset of data by using "subset()" or brackets

```
aqsub <- subset(airquality, Temp > 65)  #makes a subset of air quality data for rows where the temperat
View(aqsub)

aqsub2 <- airquality$Wind[airquality$Temp > 65]  #wind data for when the temperature is above 65
View(aqsub2)

aqsub3 <- airquality[airquality$Temp > 65, c("Wind", "Temp")]  #wind and temp data for when the tempera
View(aqsub3)
```

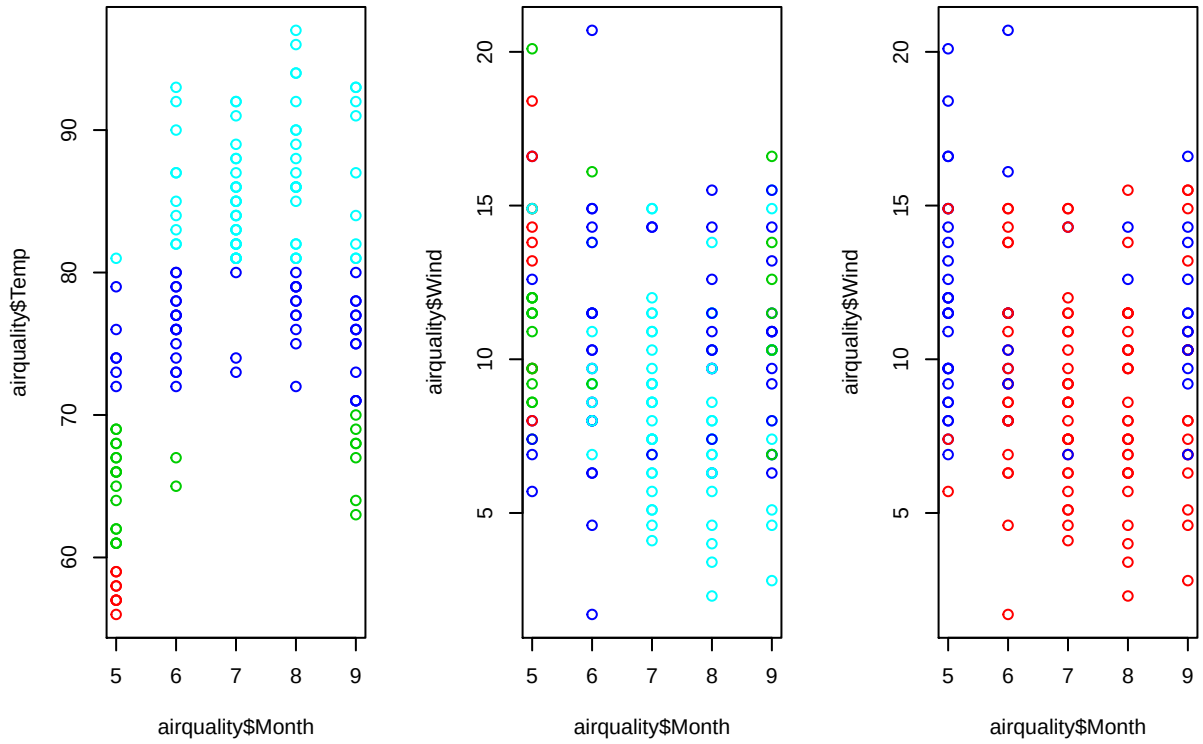
You can also split data into categories based on values. This can be useful for finding correlations and distinguishing particular points in plots.

```
aqtemp1 <- cut(airquality$Temp, breaks = c(0, 50, 60, 70, 80,
    max(airquality$Temp)), labels = c("Below 50", "50s", "60s",
    "70s", "Above 80"))

aqtemp2 <- ifelse(airquality$Temp > 75, "red", "blue")

par(mfrow = c(1, 3))
plot(airquality$Month, airquality$Temp, col = aqtemp1)
```

```
plot(airquality$Month, airquality$Wind, col = aqtemp1)
plot(airquality$Month, airquality$Wind, col = aqtemp2)
```



"tapply()" is also a useful function for analyzing data

```
tapply(airquality$Temp, airquality$Month, mean)  #mean temperature per month
```

```
##        5        6        7        8        9
## 65.54839 79.10000 83.90323 83.96774 76.90000
```

```
tapply(airquality$Temp, airquality$Month, median)  #median temperature per month
```

```
##  5  6  7  8  9
## 66 78 84 82 76
```

```
tapply(airquality$Temp, airquality$Month, var)  #variance in temperature per month
```

```
##        5        6        7        8        9
## 46.98925 43.54138 18.62366 43.36559 69.81724
```

The apply() family is useful for analyzing data. `apply()` is used to operate on arrays. `lapply()` applies a function to every element of a list and obtains a list as a result. It can be used for other objects like dataframes and lists (unlike `apply()`). `sapply()` is similar to `lapply()`, but simplifies the output into a simplified data structure (like a vector). `mapply()` applies a function to multiple vector arguments. `tapply()` applies a function to each non-empty group of values. A better explanation of what the apply() family is and what it is used for can be found here: https://ademos.people.uic.edu/Chapter4.html