BAN 602: Quantitative Fundamentals

Spring, 2020 Online Lecture Slides – Week 1

CAL STATE
EAST BAY

Agenda

- Set up R and R Studio
- Basics of R
- Data Structures in R
- Computing Summary Statistics using R
- Contingency Tables using R
- Data Visualization using R

Setup R and R-Studio

- To install R on your laptop/MacBook:
  - Go to http://www.r-project.org/ from your browser
  - Follow the download link on the page
  - Choose a CRAN location and follow the instructions for installation of the latest version for your laptop/MacBook
- Once R is properly installed, next install R-Studio:
  - Open https://rstudio.com/products/rstudio/ on your browser
  - Click on the button to download RStudio Desktop, Open Source Edition
  - Choose the latest version recommended for your laptop/MacBook and follow the instructions

Environment

- The “workspace” in R is called the global environment.
- This is where variables are stored
- The objects in the global environment can be listed using command ls()
- Objects can be removed using command rm()
- Set working directory using command setwd(“Working_Directory_Path”). In this location you should keep the data files you want to load in R.

What is R?

- R is both a computing environment and a programming language
- Born 26 years ago, rapid recent growth in popularity
- Enormous number of user-developed packages implementing multitudes of statistical methods
- Defacto language for academic research
- Rapidly becoming the language to use for industrial “data science”
- Formally, R language is:
  - functional (like Lisp)
  - interpreted (like Python)

Interacting with R

- You can use R interactively through its command line interface (CLI) or console:
  - enter a command
  - R executes the command
  - get a printed response
  - repeat
- You can use R in batch mode:
  - give R a script: a list of commands
  - R executes the commands
  - get result(s) displayed on screen or stored in files: text, tables, graphics, and more

R as a Calculator

```
7 + 10
```

```
## [1] 17
```

```
67.1 * 11
```

```
## [1] 738.1
```

```
5/7
```

```
## [1] 0.7142857
```

- R doesn't mind spaces
- Spaces can go wherever you like, just not in the middle of numbers or names

R works with vectors

```
c(5, 8, 100) + c(3, 1,7)
```

```
## [1]   8   9 107
```

```
c(5,7,10) * c(2,4,3)
```

```
## [1] 10 28 30
```

- c() is a function that concatenates scalars into a vector.
- Use c() to create a vector from individual values

R has built-in functions and constants

```
1 / sqrt(2*pi) * exp(-2)
```

```
## [1] 0.05399097
```

```
dnorm(0)
```

```
## [1] 0.3989423
```

- sqrt() computes the square root
- exp() computes e^x
- dnorm() computes the standard normal density function

CAL STATE EAST BAY

R has built-in help

```
help(dnorm)
help.search('normal distribution')
```

```
?dnorm
??'normal distribution'
```

- Text has to be enclosed between:
  - Two single quotes
  - Two double quotes

Scripts

- Most data analyses involve:
  - more than one command
  - mistakes
  - changes
- Don't enter commands one-by-one into the console
- Write your commands in a script:
  - plain text file containing your commands
  - easy to go back and make fixes/changes
  - easy to share
  - easy to add human-readable comments

Objects and Variables

- Objects are the “things” that we manipulate in R
- Conceptually, there are two kinds of objects:
  - data — things like 7 , "seven" , 7.00 , the vector [7 7 7]
  - functions — things like log , sin , +
- R considers both to be objects

- Variables
  - A variable is a storage location and an associated name that contains some object
  - Variables allow us to
    - abstract the data in our computations
    - generalize computations to different data
    - reuse data in different computations

Functions

- A function is a defined rule for turning
  - input objects (arguments) into an
  - output object (return value),
  - possibly with side effects

- Examples of side effects
  - plotting
  - saving a file
  - posting a Tweet

- Basic usage: f(arg_1, arg_2, ..., arg_n)

```
log(10)
```

```
## [1] 2.302585
```

```
10 + 5
```

```
## [1] 15
```

Variable Names and Assignment

- R variable names must obey rules:
  - Mix of alphabetical, digits, and periods
  - Cannot start with a number
- Pro tips:
  - avoid single character names
  - use underscores _ in place of spaces in names
- Variable assignment:
  - R uses the <- operator to an object to a variable
  - Pro tip: Avoid using = for assignment in R

```
name <- "Vince"
height <- 66
```

```
height <- 66
feet <- height %/% 12 # integer division
inches <- height %% 12 # modulus
```

Data type: Double

- R defines several primitive data types
- The four most common:
  - double
  - integer
  - logical
  - character
- These are the atoms of more complex data structures
- **Double**:
  - Default way of representing numbers in R
  - More specifically — double-precision floating point numbers
  - Subject to rounding error, e.g. 1/3 ≈ 0.33333

```
5.4
```

```
## [1] 5.4
```

```
10 / 3
```

```
## [1] 3.333333
```

Data types: Integer and Logical

- **Integer:**
  - Whole numbers, represented internally (by the computer) as a binary sequence (bits)
  - Most modern computers use 64-bits to represent integers
    - 2^{64} possible numbers
      Largest: $2^{63} - 1$
      Smallest: $-(2^{63})$
    - Often useful for counting or indexing things
- **Logical:**
  - Boolean TRUE or FALSE — binary values

```
10 < 15
```

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

```
10 > 15
```

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

Data types: Character

- **Character:**
  - Also known as text or character string or string
  - Internally represented by the computer as a sequence of integers and a predefined map between integers and symbols
  - Created by enclosing with single or double quotation marks

```
'v'
```

```
## [1] "v"
```

```
"Statistics is cool"
```

```
## [1] "Statistics is cool"
```

Data structure: Vectors

- Group related data values into one object: a data structure
- Most important data structures in R :
  - Atomic vector
  - List
  - Matrix
  - Data frame array

- **Vectors:**
  - Vectors come in two flavors in R:
    - Atomic vector
    - List
  - Two important properties (queried by functions):
    - length() — how many elements the vector contains
    - typeof() — what type of data it contains
  - Can create vectors with c() function

Atomic Vectors

- A homogeneous sequence of values, i.e. all of the same type
- Sometimes referred to generically as a vector
- Examples:
  - Double vector — [1.1, 5, 7.3]
  - Integer vector — [4L, 5L, 6L, 7L]
  - Character vector — [bob, alice, john]
  - Logical vector — [TRUE, FALSE, TRUE]
- Note: R does not have a scalar type. So the scalar in R is actually an atomic vector of length 1

```
x <- 5
length(x)
```

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

```
typeof(x)
```

```
## [1] "double"
```

```
is.vector(x)
```

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

Example of character vector

```r
x <- c('apples', 'oranges', 'bananas')
print(x)
```

```
## [1] "apples"  "oranges" "bananas"
```

```r
length(x)
```

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

```r
typeof(x)
```

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

Sequence

- Sequences of numbers are used frequently, so there are built- in functions to make sequences:
  - a:b – construct an integer sequence from a to b (note: a can be larger than b)
  - seq() – function to create arbitrary sequences

```
i <- 1:10
print(i)
```

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

```
j <- seq(1, 10, 2)
print(j)
```

```
## [1] 1 3 5 7 9
```

Indexing

- Elements of a vector are accessed via indexing
- Examples:
  - x[1] — first element
  - x[2] — second element
  - x[length(x)] — last element
- Negative indices are allowed too:
  - x[-1] — **all but** first element
  - x[-5] — **all but** 5th element

Example of Indexing

```
x <- c('lions', 'tigers', 'bears', 'oh my!')
x[1]
```

```
## [1] "lions"
```

```
x[length(x)]
```

```
## [1] "oh my!"
```

```
x[-2]
```

```
## [1] "lions"  "bears"  "oh my!"
```

Indexing by Vectors

- Vectors can be indexed by vectors

```
x <- c('lions', 'tigers', 'bears', 'oh my!')
x[1:2]
```

```
## [1] "lions"  "tigers"
```

```
x[c(1,3)]
```

```
## [1] "lions" "bears"
```

```
x[-(1:2)]
```

```
## [1] "bears"  "oh my!"
```

Type Coercion

```
x <- c(57, 'columbus', 1.56)
is.vector(x)
```

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

```
x <- c(57, 'columbus', 1.56)
typeof(x)
```

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

- c() coerces its arguments to be of the same type
- Here double got converted into character

Missing Values

- Real data sets often has missing values, for example:
  - survey questions not answered
  - measurements unknown/unrecorded
- R has a fundamental representation for missing values: NA

```
x <- c(5, 10, NA, 13)
print(x)
```

```
## [1]  5 10 NA 13
```

```
is.na(x)
```

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

Operators

- Operators are functions, but with special syntax
- They are of two types:
  - Unary — e.g. - negation, ! Boolean negation
  - Binary — e.g. + addition, - subtraction, < less-than
- How they operate depends on their arguments
- **Arithmetic Operators:** Operators defined on numbers
    - Take numbers as input and return a number as output
    - Examples:
      - - negation
      - + , - , * , / add, subtract, multiply, divide
      - %% modulus
- **Logical Operators:** Operators that return a logical
    - Most often used as conditions involving other data types
    - Examples:
      - < , <= — less than, less than or equal to
      - >, >= — greater than, greater than or equal to
      - == — is equal to

Boolean Indexing

- A vector can be indexed by a logical vector of the same length
- Useful for getting subsets of data, when combined with boolean operations

```
x <- c(1, 5, 7)
x[c(TRUE, FALSE, TRUE)]
```

```
## [1] 1 7
```

```
student <- c('bob', 'alice', 'john', 'mary')
gender <- c('M', 'F', 'M', 'F')
student[gender == 'M']
```

```
## [1] "bob"  "john"
```

Another Example of Boolean Indexing

```
fruits <- c('apple', 'orange', 'banana', 'cherry')
vegetables <- c('broccoli', 'carrot', 'asparagus', 'onion')
refrigerator <- c('apple', 'carrot', 'cheese')
refrigerator %in% fruits
```

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

```
refrigerator[refrigerator %in% fruits]
```

```
## [1] "apple"
```

Function Names and Defaults

- R allows function arguments to have names and default values
- This makes functions easier to use as you can refer to arguments by name. There is no need to memorize argument positions.
- Learn function names and defaults using help command
- For example: function rnorm() has 3 arguments: n — number of samples to generate (no default), mean — of the Normal distribution (default = 0), sd — of the Normal distribution (default = 1)

```
rnorm(n = 10, sd = 10)
```

```
##  [1]  -6.9568004  -9.8547346  -6.0635654   1.2393049 -18.2644090
##  [6] -12.8456064  -1.0497898   1.4678847  -2.8156671  -0.5414748
```

Data Structure: Matrix

- A matrix is like an atomic vector:
  - Collection of elements all of the same type
  - but with 2 dimensions
- Construct with matrix()
- Subset with []

```
x <- matrix(1:9, nrow = 3, ncol = 3)
print(x)
```

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

```
x[2, 3]
```

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

```
x[3, ]
```

```
## [1] 3 6 9
```

Array

- An array is a higher-dimensional generalization of a matrix:
  - Collection of elements all of the same type
  - Any fixed number of dimensions
- Construct with array()
- Subset with []

```
x <- array(1:12, dim = c(2, 3, 2))
print(x)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

```
x[2, 3, 1]
```

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

```
x[2, 3, 1]
```

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

Data Structure: List

- A vector, not necessarily all of the same type
- Elements can have names
- Create with list() function

```
my_distribution <- list('normal', 0, FALSE)
print(my_distribution)
```

```
## [[1]]
## [1] "normal"
##
## [[2]]
## [1] 0
##
## [[3]]
## [1] FALSE
```

List Index

- x[i] is a list containing the elements in x corresponding to i
- [] always returns a list

```
my_distribution[1:2]
```

```
## [[1]]
## [1] "normal"
##
## [[2]]
## [1] 0
```

```
my_distribution[2]
```

```
## [[1]]
## [1] 0
```

Content of a List

- [[]] returns a single value content of a list

```
typeof(my_distribution[2])
```

```
## [1] "list"
```

```
typeof(my_distribution[[2]])
```

```
## [1] "double"
```

"If list x is a train carrying objects, then x[[5]] is the object in car 5; x[4:6] is a train of cars 4-6."
— @RLangTip

```
my_distribution[[2]] + 10
```

```
## [1] 10
```

Naming the List Elements

```
names(my_distribution) <- c('family', 'mean', 'is.symmetric')
names(my_distribution)
```

```
## [1] "family"       "mean"         "is.symmetric"
```

```
print(my_distribution)
```

```
## $family
## [1] "normal"
##
## $mean
## [1] 0
##
## $is.symmetric
## [1] FALSE
```

Using Names to Access List Elements

```
# 3 ways:
my_distribution[[2]]
```

```
## [1] 0
```

```
my_distribution[["mean"]]
```

```
## [1] 0
```

```
my_distribution$mean # short-hand for [[]]
```

```
## [1] 0
```

Assigning Names during List Creation

```
another_distribution <- list(family = 'gaussian', mean = 7, sd = 1,
                             is.symmetric = TRUE)
print(another_distribution)
```

```
## $family
## [1] "gaussian"
##
## $mean
## [1] 7
##
## $sd
## [1] 1
##
## $is.symmetric
## [1] TRUE
```

Properties of List

- Lists give us a way to store and look up data by name
- They are like key-value pairs/associative array/dictionary in other languages
- If all our distributions have a component named family, then we can look it up by name without caring about its location.

```
another_distribution$family
```

```
## [1] "gaussian"
```

Data Structure: Data Frames

- A data frame is a classic data table, organized with
  - n rows (observations)
  - p columns (variables)
- Most statistical functions in R work with data frames
- A data frame is a list that
  - has atomic vector components all of the same length
  - can also be indexed like a matrix
- Construct with data.frame()

```
df <- data.frame(student = c('bob', 'alice', 'john', 'mary'),
                 score = c(70, 90, 85, 100))
print(df)
```

```
##   student score
## 1     bob    70
## 2   alice    90
## 3    john    85
## 4    mary   100
```

Properties of Data Frames

These functions are helpful to know the properties of data-frames:

- nrow() — returns number of rows in a data-frame
- ncol() — returns number of columns in a data-frame
- dim() — returns dimension of a data-frame i.e. number of rows and columns as a vector of length 2
- names() — inherited from lists
- rownames() — optional names for each row

```
nrow(df)
```

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

```
ncol(df)
```

```
## [1] 2
```

```
dim(df)
```

```
## [1] 4 2
```

Data Frame Indexing

- Data frames can be index like a matrix or a list

```
df[1,]
```

```
##   student score
## 1     bob    70
```

```
df[,1]
```

```
## [1] bob   alice john  mary
## Levels: alice bob john mary
```

```
df$score
```

```
## [1]  70  90  85 100
```

Attributes

- names is an example of an attribute
- R allows objects to have optional attributes

```
attributes(my_distribution)
```

```
## $names
## [1] "family"      "mean"        "is.symmetric"
```

- All vectors can have names
- Note: attributes() returns a list of the object's attributes

Vector Indexing by Name

- Vectors (atomic vectors and lists) can have an optional names attribute
- Vectors can be indexed by name

```
x <- c(john = 10, bob = 3, alice = 7)
x['bob']
```

```
## bob
##   3
```

```
x[c('john', 'alice')]
```

```
##  john alice
##    10     7
```

Matrix Indexing by Name

- Matrices and data frames can have optional rownames and colnames attributes
  - but for data frames, colnames is a synonym for names
- They can be indexed by name

```
x <- matrix(1:12, nrow=4,ncol=3)
rownames(x) <- c('john', 'bob', 'alice', 'mike')
colnames(x) <- c('homework', 'exam', 'final')
x['john', ]
```

```
## homework     exam    final
##        1        5        9
```

```
x['alice', 'final']
```

```
## [1] 11
```

Functions to Inspect Objects

- These functions are very helpful:
  - str() – returns the structure of an object
  - head() – returns the head (first few entries) of an object
  - tail() - returns the tail (last few entries) of an object

Reading Dataset

- There are a number of functions in R to read dataset. The name of the file to be read must be specified as an argument.
  - read.table() – reads data from a flat file. If the first row of the dataset contains column headers, pass an argument "header=T"
    - Can read data directly from an url: read.table(URL, header=T)
  - read.csv(), or read.csv2() – reads comma separated value files directly.
  - read.delim(), or read.delim2() – reads delimited text files
  - read.xlsx(), or read.xlsx2() – reads excel files

  - Example: To read the Babies.csv file from your working directory, execute the command:
    babies <- read.csv(file="Babies.csv", header = TRUE)

Descriptive Statistics: Statistical Summary

- summary() – this generic function summarizes an object
- For data frames, summary() provides a 6 number summary of each column: minimum, first quartile, median (i.e. second quartile), mean, third quartile, maximum.

```
summary(babies)
```

```
##       bwt          gestation         parity           age       
##  Min.   : 55.0   Min.   :148.0   Min.   :0.0000   Min.   :15.00  
##  1st Qu.:108.8   1st Qu.:272.0   1st Qu.:0.0000   1st Qu.:23.00  
##  Median :120.0   Median :280.0   Median :0.0000   Median :26.00  
##  Mean   :119.6   Mean   :286.9   Mean   :0.2549   Mean   :27.37  
##  3rd Qu.:131.0   3rd Qu.:288.0   3rd Qu.:1.0000   3rd Qu.:31.00  
##  Max.   :176.0   Max.   :999.0   Max.   :1.0000   Max.   :99.00  
##      height          weight        smoke       
##  Min.   :53.00   Min.   : 87   Min.   :0.0000  
##  1st Qu.:62.00   1st Qu.:115   1st Qu.:0.0000  
##  Median :64.00   Median :126   Median :0.0000  
##  Mean   :64.67   Mean   :154   Mean   :0.4644  
##  3rd Qu.:66.00   3rd Qu.:140   3rd Qu.:1.0000  
##  Max.   :99.00   Max.   :999   Max.   :9.0000
```

Simple Summary Statistics

- To calculate simple summary statistics of an object x use the following functions:
  - *mean(x)* – returns the mean of x values
  - *sd(x)* – returns the sample standard deviation of x values
  - *var(x)* - returns the sample variance of x values
  - *median(x)* – returns the median of x values
  - *quantile(x)* – returns 0^{th}, 25^{th}, 50^{th}, 75^{th}, 100^{th} percentile of x values
  - *range(x)* – returns the range (minimum and maximum) of x values
- Practice: Calculate mean, standard deviation, variance, median, first and third quartiles, range of the weight variable from the babies dataset.

Covariance and Correlation

- The cov(x, y) function is used to compute covariance between two variables x and y.
- The cor(x, y) function computes correlation between two variables x and y.
- How the covariance or correlation should be computed in presence of missing values can be mentioned through 'use' argument.

- Practice: Find out the covariance and correlation between the weight and height variables from the babies dataset. Use the following R commands:

cov(babies$height, babies$weight)
cor(babies$height, babies$weight)

Transforming a Data frame

- The function transform(dataframe, ...)
  - Copies input data frame,
  - Evaluates the remaining arguments, and
  - Either replaces matching columns or appends new columns
- Convenient because … arguments are evaluated within the data frame

- Example: To add two new columns (with values calculated based on existing columns) to the babies dataset - BMI of the mothers and whether or not a baby was born preterm:

```
# Create two new columns: BMI and preterm indicator
babies <- transform(babies, bmi = weight / height^2 * 703,
                            preterm = gestation < 37 * 7)
```

Missing Values in a Data frame

- In the babies dataset there are missing values. They are coded differently in different columns.
- For example, missing values in ‘smoke’ column is denoted by number 9. 0 in this column means the mother doesn’t smoke, and 1 means she does.
- To clean up the missing values from ‘smoke’ column:
  - Select entries whose values are the missing (i.e. of value 9)
  - Assign NA to those entries

```
babies$smoke[babies$smoke == 9] <- NA

# Convert smoking status into a logical
babies$smoke <- as.logical(babies$smoke)
```

- This code snippet would mark the missing values as NA, and convert the smoke variable into logical (true/false) type.

Crosstabulation: Contingency Table

- The function table() builds a contingency table of the counts of at each combination of factor levels of its input arguments
- Converts each argument into a factor (categorical data type)
- Counts each combination of factor levels (categories)

```
# Contingency table
x <- table(babies$smoke, babies$preterm)
print(x)
```

```
##
##         FALSE TRUE
##   FALSE   677   56
##   TRUE    439   41
```

- However the contingency table is not properly labeled here since the levels these two variables smoke and preterm could take have not been defined before.

Contingency Table with Factors Relabeled

```
babies <- transform(babies,
                    smoke = factor(smoke, levels = c(FALSE, TRUE),
                                   labels = c('non-smoking', 'smoking')),
                    preterm = factor(preterm, levels = c(FALSE, TRUE),
                                     labels = c('normal', 'preterm')))
```

- Now the contingency table looks meaningful:

```
# Contingency table
x <- table(babies$smoke, babies$preterm)
print(x)
```

```
##
##                normal preterm
##   non-smoking     677      56
##   smoking         439      41
```

Marginal Table and Relative Frequency

- To compute the margins of a table i.e. the sum across each rows or each columns of a table, the *margin.table()* function is used:
  - To compute **row-wise** sum: use *margin.table(table_name, **1**)* command
  - To compute **column-wise** sum: use *margin.table(table_name, **2**)* command
- Practice: Execute the R command:

  margin.table(x, 2) where x is the table from last slide.
- To compute relative frequency **row-wise** and **column-wise**, use the function *prop.table(table_name, **1**)* and *prop.table(table_name, **2**)* respectively.

```
prop.table(x, 1)
```

```
##
##                  normal    preterm
##   non-smoking 0.92360164 0.07639836
##   smoking     0.91458333 0.08541667
```

```
prop.table(x, 2)
```

```
##
##                  normal   preterm
##   non-smoking 0.6066308 0.5773196
##   smoking     0.3933692 0.4226804
```

Histogram

- The hist(x) function can be used to plot a histogram of object x values.
- You can get approximately n bars in the histogram by specifying the argument breaks=n
- Exact location of the breaks can be specified using a vector for the 'breaks' argument.
- Use the 'xlab' and 'ylab' arguments to label the x-axis and y-axis respectively.
- Use 'main' argument to add a title to the histogram

- Practice: Execute the R command:

hist(babies$bwt, breaks = c(40, 70, 100, 130, 160, 190), xlab="Birth Weight", main = "Histogram of Birth Weight")

and check the histogram plotted.

Boxplot

- The boxplot(x) function can be used to draw a Box and Whiskers plot of object x values.
- Use the 'xlab' and 'ylab' arguments to label the x-axis and y-axis respectively.
- Use 'main' argument to add a title to the boxplot
- Adding the argument horizontal=T changes the orientation of the boxplot to horizontal

- Practice: Execute the R command:

  boxplot(babies$height, horizontal = T, main="Boxplot", xlab="Height")

  and check the boxplot generated.

Bar Chart

- The barplot(x) function can be used to draw a bar chart of vector or matrix x.
- The bars created are stacked bars by default.
- If the bars are to stay side-by-side instead, use the argument 'beside=T'.
- If the bars are to be drawn horizontally, use the argument 'horiz=T'.
- The 'col' and 'border' arguments can be used to color the bars and their borders
- As usual xlab, ylab, and main can be used to add labels to the x-axis, y-axis and title of the chart

- Practice: Execute the R command:

  barplot(x, beside=T)

  and check the bar chart generated.

Pie Chart

- The pie(x) function can be used to draw a bar chart of vector or matrix x.
- The color of the slices can be mentioned in argument ‘col’.
- The labels for each slice can be specified by argument ‘labels’.
- The title can be specified through the ‘main’ argument.

- Practice: Execute the R command:

pie(x, labels = c("Normal-Non-smoking", "Normal-Smoking", "Preterm-Non-Smoking", "Preterm-Smoking"), col=c("red","green", "blue", "yellow"), main="Pie Chart")

and check the pie chart generated.

Scatter Plot

- The plot(x, y) function can be used to draw a scatter plot between two variables x and y.
- The title can be specified through the 'main' argument.
- The x-axis and y-axis can be labeled through xlab and ylab arguments

- Practice: Draw the scatter plot between baby weight and gestation period from the babies dataset:

```
plot(babies$bwt, babies$gestation, xlab = "Baby Weight", ylab="Gestation Period",
main="Scatter Plot")
```