SELECTING VALUES

SELECTING VALUES FROM VECTORS

- Use [ ] brackets to select values from a vector
- Can use the following to subset:
  - Positive integers
  - Negative integers
  - Zero
  - Blank spaces
  - Logical values
  - Names

YOUR TURN!

1. *Use the following commands to create a vector of 30 random values.*

```
set.seed(1234)
my_vector <- rnorm(n = 30, mean = 0, sd = 5)
names(my_vector) <- letters
```

YOUR TURN!

2. *Run each of the following commands. With a neighbor/partner, discuss how they subset my_vector.*

my_vector[]
my_vector[5]
my_vector[3:7]
my_vector[c(1,3,6)]
my_vector[-(2:14)]
my_vector[0]
my_vector[c(TRUE, FALSE, TRUE, FALSE, FALSE)]
my_vector[c("b", "n", "r")]

SOLUTION

```
# select every element of my_vector
my_vector[]
```

```
          a         b         c         d         e
-6.0353287  1.3871462  5.4222059 -11.7284885  2.1456234
          f         g         h         i         j
 2.5302795 -2.8736998 -2.7331593 -2.8222600 -4.4501891
          k         l         m         n         o
-2.3859635 -4.9919322 -3.8812695  0.3222941  4.7974703
          p         q         r         s         t
-0.5514275 -2.5550475 -4.5559771 -4.1858584 12.0791759
          u         v         w         x         y
 0.6704411 -2.4534295 -2.2027394  2.2979472 -3.4686012
          z      <NA>      <NA>      <NA>      <NA>
-7.2410246  2.8737786 -5.1182786 -0.0756915 -4.6797430
```

SOLUTION

```
# select the fifth element of my_vector
my_vector[5]
```

```
      e
2.145623
```

SOLUTION

```
# select the 3rd through 7th elements of my_vector
my_vector[3:7]

        c         d         e         f         g
5.422206 -11.728489  2.145623  2.530279 -2.873700
```

SOLUTION

```
# select the 1st, 3rd, and 6th elements of my_vector
my_vector[c(1,3,6)]

       a        c        f
-6.035329  5.422206  2.530279
```

SOLUTION

```
# select every element of my_vector EXCEPT the 2nd through 14th elements
my_vector[-(2:14)]

        a         o         p         q         r         s
-6.0353287  4.7974703 -0.5514275 -2.5550475 -4.5559771 -4.1858584
        t         u         v         w         x         y
12.0791759  0.6704411 -2.4534295 -2.2027394  2.2979472 -3.4686012
        z      <NA>      <NA>      <NA>      <NA>
-7.2410246  2.8737786 -5.1182786 -0.0756915 -4.6797430
```

SOLUTION

```
# returns an empty object
# this isn't very useful
my_vector[0]

named numeric(0)
```

SOLUTION

```
# returns the 1st element, excludes the 2nd element, returns the 3rd element
# excludes the 4th and 5th elements
# repeat this pattern through the entire vector
my_vector[c(TRUE, FALSE, TRUE, FALSE, FALSE)]
```

```
         a         c         f         h         k         m 
-6.0353287 5.4222059 2.5302795 -2.7331593 -2.3859635 -3.8812695 
         p         r         u         w         z      <NA> 
-0.5514275 -4.5559771 0.6704411 -2.2027394 -7.2410246 -5.1182786
```

SOLUTION

```
# select the elements named b, n, and r of my_vector
# can only do this when elements have names
my_vector[c("b", "n", "r")]
```

```
       a        c        f 
-6.035329  5.422206  2.530279 
```

YOUR TURN!

1. *Look at documentation for the WorldPhones data set in R (R has many built-in data sets to make learning easy and fun). Acquaint yourself by using some commands we've learned today (e.g., str()).*

2. *Using the hint that the command* WorldPhones[,] *will return all rows and columns of the data frame and your knowledge of subsetting vectors, extract the following:*

YOUR TURN!

a. *The 1st row and all columns from WorldPhones*

b. *The 2nd, 3rd, and 6th rows; the 2nd through 5th columns*

c. *All rows; repeat the 3rd column twice*

d. *All rows; every column except the 1st and 7th*

e. *An empty data frame (think an empty object)*

f. *The penultimate row by using a function and not the number 6; all columns*

SOLUTION

```
# The 1st row and all columns from WorldPhones
WorldPhones[1, ]
```

```
 N.Amer  Europe   Asia  S.Amer  Oceania  Africa Mid.Amer
  45939   21574   2876    1815     1646      89      555
```

SOLUTION

```
# The 2nd, 3rd, and 6th rows; the 2nd through 5th columns
WorldPhones[c(2, 3, 6), c(2, 5)]
```

```
     Europe Oceania
1956  29990    2366
1957  32510    2526
1960  40341    3054
```

SOLUTION

```
# All rows; repeat the 3rd column twice
WorldPhones[, c(3, 3)]

     Asia Asia
1951 2876 2876
1956 4708 4708
1957 5230 5230
1958 6662 6662
1959 6856 6856
1960 8220 8220
1961 9053 9053
```

SOLUTION

```
# All rows; every column except the 1st and 7th
WorldPhones[, -c(1, 7)]
```

```
     Europe Asia S.Amer Oceania Africa
1951  21574 2876   1815    1646     89
1956  29990 4708   2568    2366   1411
1957  32510 5230   2695    2526   1546
1958  35218 6662   2845    2691   1663
1959  37598 6856   3000    2868   1769
1960  40341 8220   3145    3054   1905
1961  43173 9053   3338    3224   2005
```

SOLUTION

```
# An empty object (think an empty object)
WorldPhones[0, 0]


<0 x 0 matrix>
```

SOLUTION

```
# The penultimate row by using a command and not the number 6; all columns
WorldPhones[nrow(WorldPhones) - 1, ]
```

```
 N.Amer  Europe   Asia  S.Amer  Oceania  Africa Mid.Amer
  76036   40341   8220    3145     3054    1905     1008
```

SELECTING COLUMNS FROM DATA FRAMES

- Able to use data_frame[,] notation to select rows/columns from a data frame
- We'll learn the dplyr package later to select rows/columns quickly and efficiently
- Base R way to select columns: $, [ ], and [[ ]]

SELECTING COLUMNS FROM DATA FRAMES

- Preserve the structure of the output to be the same as the input with data_frame[column]
  - Can use a column name in quotes or a column index

- Simplify the structure of the output with data_frame[[column]]
  - Can use a column name in quotes or a column index

- Simplify the structure of the output to be a smaller structure than the input with data_frame$column
  - Must use a column name with a $

SELECTING COLUMNS FROM DATA FRAMES

Preserve with data_frame[column]

```
# Preserve structure, use a column name
# The input mtcars is a data frame
    # and the output is also a data frame
mtcars["disp"]
```

```
                  disp
Mazda RX4         160.0
Mazda RX4 Wag     160.0
Datsun 710        108.0
Hornet 4 Drive    258.0
Hornet Sportabout 360.0
Valiant           225.0
Duster 360        360.0
Merc 240D         146.7
Merc 230          140.8
Merc 280          167.6
Merc 280C         167.6
Merc 450SE        275.8
```

```
# Preserve structure, use a column index
mtcars[3]
```

```
                  disp
Mazda RX4         160.0
Mazda RX4 Wag     160.0
Datsun 710        108.0
Hornet 4 Drive    258.0
Hornet Sportabout 360.0
Valiant           225.0
Duster 360        360.0
Merc 240D         146.7
Merc 230          140.8
Merc 280          167.6
Merc 280C         167.6
Merc 450SE        275.8
```

SELECTING COLUMNS FROM DATA FRAMES

Simplify with data_frame[[column]]

```
# Simplify structure, use a column name
# Notice the structure of the output is now a vector and NOT a
data frame
mtcars[["disp"]]
```

```
 [1] 160.0 160.0 108.0 258.0 360.0
 [6] 225.0 360.0 146.7 140.8 167.6
[11] 167.6 275.8 275.8 275.8 472.0
[16] 460.0 440.0  78.7  75.7  71.1
[21] 120.1 318.0 304.0 350.0 400.0
[26]  79.0 120.3  95.1 351.0 145.0
[31] 301.0 121.0
```

```
# Simplify structure, use a column index
mtcars[[3]]
```

```
 [1] 160.0 160.0 108.0 258.0 360.0
 [6] 225.0 360.0 146.7 140.8 167.6
[11] 167.6 275.8 275.8 275.8 472.0
[16] 460.0 440.0  78.7  75.7  71.1
[21] 120.1 318.0 304.0 350.0 400.0
[26]  79.0 120.3  95.1 351.0 145.0
[31] 301.0 121.0
```

SELECTING COLUMNS FROM DATA FRAMES

Simplify with data_frame$column

```
# Simplify structure, use a column name
# Notice the structure of the output is now a vector and NOT a
data frame
mtcars$disp
```

```
 [1] 160.0 160.0 108.0 258.0 360.0
 [6] 225.0 360.0 146.7 140.8 167.6
[11] 167.6 275.8 275.8 275.8 472.0
[16] 460.0 440.0  78.7  75.7  71.1
[21] 120.1 318.0 304.0 350.0 400.0
[26]  79.0 120.3  95.1 351.0 145.0
[31] 301.0 121.0
```

```
# Simplify structure, use a column index
# MUST use a column name with $
mtcars$3
```

```
Error: unexpected numeric constant in "mtcars$3"
```

SELECTING VALUES FROM LISTS

- Use $, [ ], and [[ ]] to select items and values from lists
- Sometimes need combinations of $, [ ], and [[ ]] to extract desired information
- Let’s practice!

YOUR TURN!

1. *Run the command model <- lm(mpg ~ ., data = mtcars).*

2. *Run the command str(model) to examine your model.*

3. *What do the following commands do? What structures should you expect?*
   a. *model["fitted.values"]*
   b. *model$coefficients[["wt"]]*
   c. *model[["residuals"]][1:10]*

SOLUTION

```
# Linear model, mpg as a function of every other variable in the mtcars data set
model <- lm(mpg ~ ., data = mtcars)

# look at structure of model
str(model)
```

```
List of 12
 $ coefficients : Named num [1:11] 12.3034 -0.1114 0.0133 -0.0215 0.7871 ...
  ..- attr(*, "names")= chr [1:11] "(Intercept)" "cyl" "disp" "hp" ...
 $ residuals    : Named num [1:32] -1.6 -1.112 -3.451 0.163 1.007 ...
  ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
 $ effects      : Named num [1:32] -113.65 -28.6 6.13 -3.06 -4.06 ...
  ..- attr(*, "names")= chr [1:32] "(Intercept)" "cyl" "disp" "hp" ...
 $ rank         : int 11
 $ fitted.values: Named num [1:32] 22.6 22.1 26.3 21.2 17.7 ...
  ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
 $ assign       : int [1:11] 0 1 2 3 4 5 6 7 8 9 ...
```

SOLUTION

```
# Look at the fitted values of the linear model
# Output structure is a list
model["fitted.values"]
```

```
        Mazda RX4       Mazda RX4 Wag          Datsun 710
         22.59951           22.11189           26.25064
    Hornet 4 Drive   Hornet Sportabout            Valiant
         21.23740           17.69343           20.38304
       Duster 360          Merc 240D           Merc 230
         14.38626           22.49601           24.41909
```

```
# look at class of output
class(model["fitted.values"])
```

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

SOLUTION

```
# Look at the weight coefficient
model$coefficients[["wt"]]

[1] -3.715304



# look at class of output
# Output structure is a numeric vector
class(model$coefficients[["wt"]])

[1] "numeric"
```

SOLUTION

```
# Look at the weight coefficient
model[["residuals"]][1:10]
```

```
       Mazda RX4     Mazda RX4 Wag        Datsun 710    Hornet 4 Drive
     -1.59950576      -1.11188608       -3.45064408        0.16259545
Hornet Sportabout          Valiant        Duster 360         Merc 240D
      1.00656597       -2.28303904       -0.08625625        1.90398812
        Merc 230          Merc 280
     -1.61908990        0.50097006
```

```
# look at class of output
# Output structure is a numeric vector
typeof(model[["residuals"]][1:10])
```

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

MODIFYING VALUES

MODIFYING VECTOR VALUES

- Describe the value(s) you want to modify
- Then use the <- assignment operator to overwrite the value(s) you described!
- Modifying elements overwrites; it doesn't create a new copy! Watch out!

```
# Make a lame dumb example vector
lame_dumb_vector <- c(1, 2, 3, 4, 5)

# Overwrite the first value with 8675309
lame_dumb_vector[1] <- 8675309

# QA every day! Check the vector!
lame_dumb_vector

[1] 8675309       2       3       4       5
```

MODIFYING **MANY** VECTOR VALUES

- Can change multiple values at once
- Can use vector recycling to your advantage

```
# Make a lame dumb example vector
lame_dumb_vector <- c(1, 2, 3, 4, 5)

# Overwrite the 1st and 3rd values with 867 and 5309
lame_dumb_vector[c(1, 3)] <- c(867, 5309)

# Overwrite the other values with -1
lame_dumb_vector[c(2, 4, 5)] <- -1



# QA every day! Check the vector!
lame_dumb_vector

[1]  867   -1 5309   -1   -1
```

MODIFYING DATA FRAME VALUES

To modify data frame values:

1. Select a column
2. Subset for the elements you want to modify
3. Assign a vector of values to replace them!

```
# save a copy of the iris data frame
iris_df <- iris

# select the Sepal.Length column
# then change the first 3 values in that column
iris_df$Sepal.Length[1:3] <- 1000

# QA every day
# check the data frame
head(iris_df)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1       1000.0         3.5          1.4         0.2  setosa
2       1000.0         3.0          1.4         0.2  setosa
3       1000.0         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

MODIFYING VECTORS WITH LOGIC

Logical operators return TRUE or FALSE for element-wise comparisons

Operator	Syntax
>	a > b
>=	a >= b
<	a< b
<=	a <= b
== (check for equality)	a == b
!= (check for not equal)	a != b
%in% (check for group membership)	a %in% c(a, b, c)

```
1 != 2
TRUE

3 > c(5, 1, 3)
FALSE  TRUE FALSE

c(5, 1, 3) == c(3, 1, 5)
FALSE  TRUE FALSE

1 %in% c(5, 1, 3)
TRUE

c(19, 3) %in% c(5, 1, 3)
FALSE  TRUE

c(1, 8, 0, 9) %in% c(5, 1, 3)
TRUE FALSE FALSE FALSE
```

MODIFYING VECTORS WITH LOGIC

- We can leverage logical comparisons to modify values in vectors
- Think through what each command outputs and how logical comparisons select which elements to modify!
- No, you don't need all three steps every time you want to logically subset. Thank heavens.

```
# save a copy of the iris data frame
iris_df <- iris

# TRUE/FALSE Boolean values for
# if Petal.Width is greater than 1
iris_df$Petal.Width > 1

# only observations with Petal.Width > 1
# show Petal.Width column
iris_df$Petal.Width[iris_df$Petal.Width > 1]

# change all Petal.Width > 1 values to -3
iris_df$Petal.Width[iris_df$Petal.Width > 1] <- -3
```

SUBSET THEN SELECT DF COLUMNS

- Use logical comparisons to filter for certain observations and then select all columns from a data frame
- Combine logical comparisons with & or | (and or…..or)

```
# filter for observations with a Sepal.Length
# greater than 7.3
# and show all columns for those observations
iris[iris$Sepal.Length > 7.3, ]


# filter for observations with:
# 1) Sepal.Length greater than 7.3, AND
# 2) Petal.Length greater than 6.3
iris[iris$Sepal.Length > 7.3 & iris$Petal.Length > 6.3, ]


# filter for observations with:
# 1) Sepal.Length greater than 7.3, OR
# 2) Petal.Width < 1.8
iris[iris$Sepal.Length > 7.3 | iris$Petal.Width < 1.8, ]
```

YOUR TURN!

1. *Read documentation for the esoph built-in R data set.*

2. *Filter the esoph data set for observations (showing all columns) with:*
   - ✓ *Tobacco consumptions either 0-9 grams per day or 10-19 grams per day, AND*
   - ✓ *Number of cases greater than zero*

SOLUTION

```
# Documentation
?esoph

# pull information
esoph[esoph$tobgp %in% c("0-9g/day", "10-19") & esoph$ncases > 0, ]
```

```
   agegp     alcgp    tobgp ncases ncontrols
13 25-34      120+    10-19      1         1
17 35-44 0-39g/day    10-19      1        14
21 35-44     40-79    10-19      3        23
28 35-44      120+ 0-9g/day      2         3
31 45-54 0-39g/day 0-9g/day      1        46
35 45-54     40-79 0-9g/day      6        38
```

MISSING DATA

MISSING VALUES IN R

- R shows missing values as NA
- Missing values prevent R from calculating sums, means, equality checks, etc.
- Use the is.na() function to check for missing values—this function is another logical operator in R

```
# adding missing values to numbers results in NA
8 + NA
```

```
[1] NA
```

```
# missing values mess up means
mean(c(1, 2, 3, NA, 5))
```

```
[1] NA
```

```
# this is why we use the na.rm = TRUE argument!
mean(c(1, 2, 3, NA, 5), na.rm = TRUE)
```

```
[1] 2.75
```

COUNTING MISSING VALUES

- Use the is.na() function to check for missing values—this function is another logical operator in R
- Use sum(is.na()) to calculate the total number of missing values
- Use colSums(is.na()) to calculate the number of missing values per column

```
# example data frame
df <- data.frame(col1 = c(1:3, NA),
                 col2 = c("this", NA,"is", "text"),
                 col3 = c(TRUE, FALSE, TRUE, TRUE),
                 col4 = c(2.5, NA, NA, NA),
                 stringsAsFactors = FALSE)

# illustrate is.na() function
is.na(df)
 col1  col2  col3  col4
[1,] FALSE FALSE FALSE FALSE
[2,] FALSE  TRUE FALSE  TRUE
[3,] FALSE FALSE FALSE  TRUE
[4,]  TRUE FALSE FALSE  TRUE

# count missing values
sum(is.na(df))
[1] 5
colSums(is.na(df))
col1 col2 col3 col4
   1    1    0    3
```

REMOVING INCOMPLETE OBSERVATIONS

- Use complete.cases() to keep only observations with no missing values
- Can also use na.omit() to remove incomplete observations
- Discuss with stakeholders/clients before simply removing observations! Maybe missing data exists for a reason!

```
# complete cases
complete.cases(df)
[1]  TRUE FALSE FALSE FALSE

# subset with complete observations only
df[complete.cases(df), ]
col1  col2  col3  col4
  col1 col2 col3 col4
   1 this TRUE  2.5

# or subset with `!` operator to retrieve incomplete cases
df[!complete.cases(df), ]
[1] 5
  col1 col2  col3 col4
2    2 <NA> FALSE   NA
3    3   is  TRUE   NA
4   NA text  TRUE   NA

# can also use na.omit() to subset for complete obs. Only
na.omit(df)
  col1 col2 col3 col4
1    1 this TRUE  2.5
```

YOUR TURN!

1. *How many missing values are in the built-in data set airquality?*
2. *Which variables are the missing values concentrated in?*
3. *How would you impute the mean or median for these values?*
4. *How would you omit all rows containing missing values?*

SOLUTION

```
# 1) how many missing values?
# number of missing values
sum(is.na(airquality))
```

```
[1] 44
```

```
# 2) which variables have missing values
# number of missing values by variable
colSums(is.na(airquality))
```

```
  Ozone Solar.R    Wind    Temp   Month     Day
     37       7       0       0       0       0
```

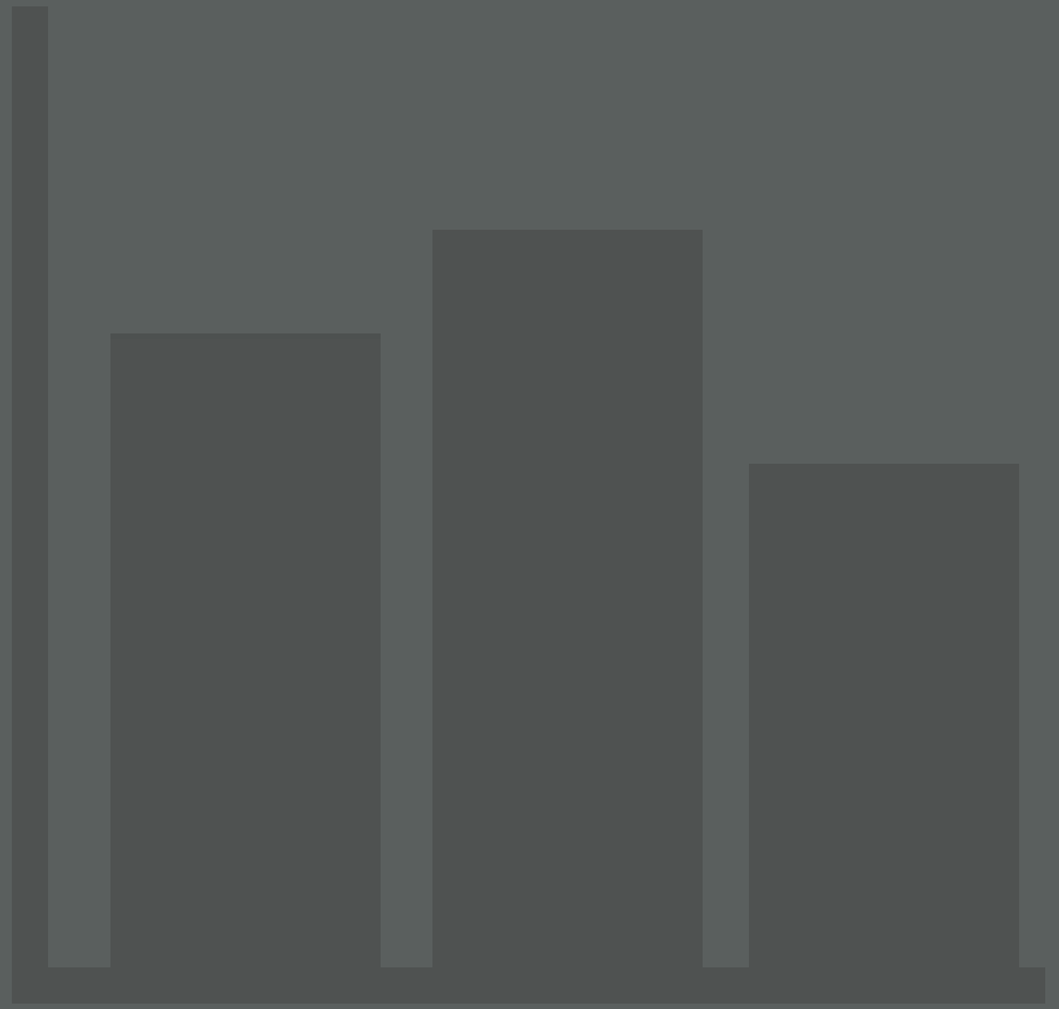
SOLUTION

```
# let’s save a copy of the data set, first
airquality_copy <- airquality


# 3) how to impute the mean/median
# imputing, for example, the mean for the Ozone variable
airquality_copy$Ozone[is.na(airquality_copy$Ozone)] <- mean(airquality_copy$Ozone, na.rm = TRUE)


# number of missing values by variable
colSums(is.na(airquality_copy))

  Ozone Solar.R    Wind    Temp   Month     Day
     0       7       0       0       0       0
```

SOLUTION

```
# 4) keep complete cases only
# use any of the following functions

airquality_complete <- complete.cases(airquality)

OR

airquality_complete <- na.omit(airquality)



# QA every day!
sum(is.na(airquality_complete))

[1] 0
```

GET TO KNOW DATA



LEARN ABOUT THE DATA

For quick data exploration, base R plotting functions can provide an expeditious and straightforward approach to understanding your data.

What are some functions to extract this information?

QUICK PLOTS

Function	Description
?	scatter plot
?	line chart
?	bar chart
?	histogram
?	box plot
?	stem & leaf plot

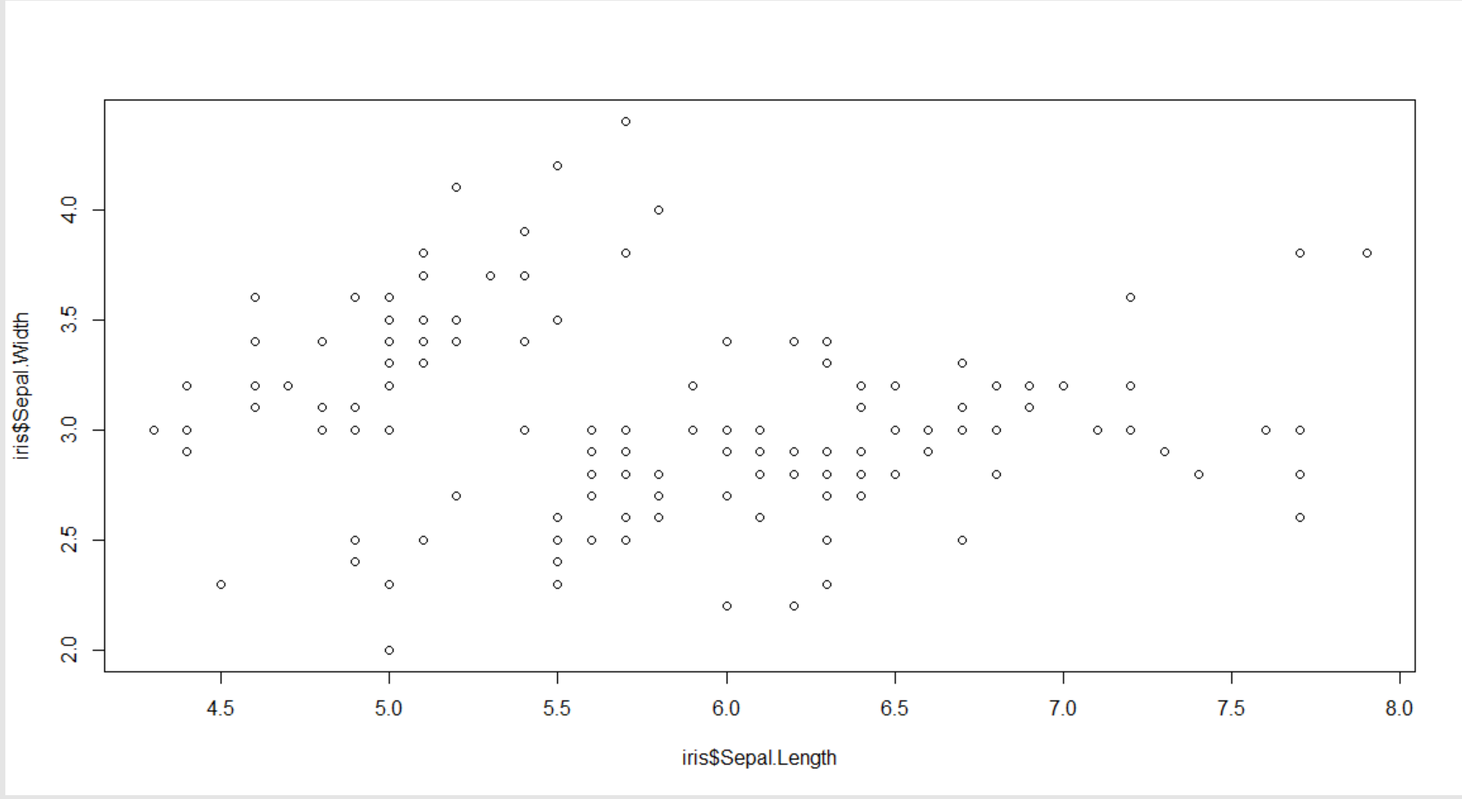
QUICK PLOTS

Function	Description
plot(x, y)	scatter plot
plot(x, y, type = “l”)	line chart
barplot(table(x))	bar chart
hist(x)	histogram
boxplot(y ~ x, data)	box plot
stem(x)	stem & leaf plot

EXAMPLES OF BASE R PLOTS

- The plot() function is very good at guessing what plot you want or should have
- Some base R plotting functions let you specify the data set without using $ to select variables for each argument
- Base R plots may not be pretty but they're quick to make!
- Don't worry! We'll learn the ggplot2 package for data visualization soon.

```
# super quick scatterplot with the (groan) iris data
plot(iris$Sepal.Length, iris$Sepal.Width)
```



YOUR TURN!

Use the chickwts data set to make the following visualizations.

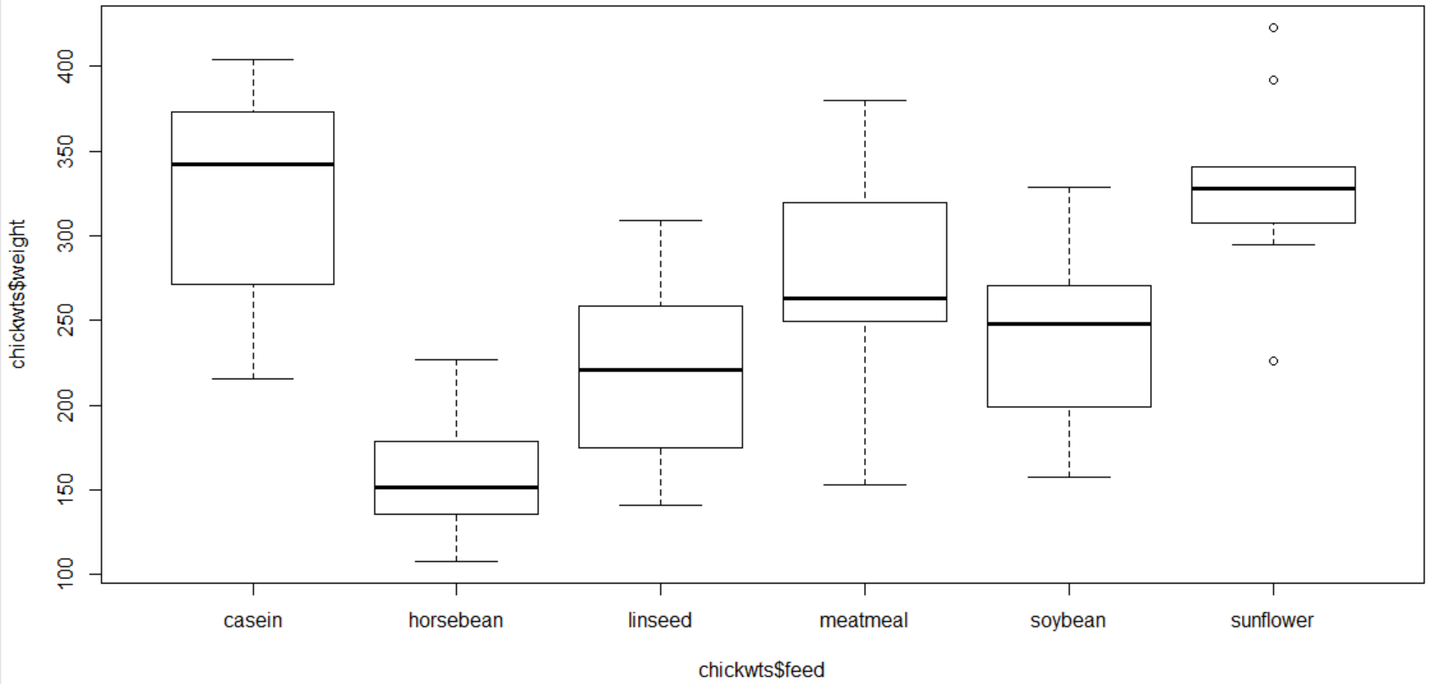
1. *Boxplots of the weight variable by feed. Look at documentation for the data set and for the boxplot() function if needed.*

2. *A table of the feed variable. Use the $ symbol to select the feed variable here.*

3. *A histogram of the weight variable.*

SOLUTION

```
# 1) boxplots
# use either command below
# both produce the same boxplots
boxplot(weight ~ feed, data = chickwts)
boxplot(chickwts$weight ~ chickwts$feed)
```



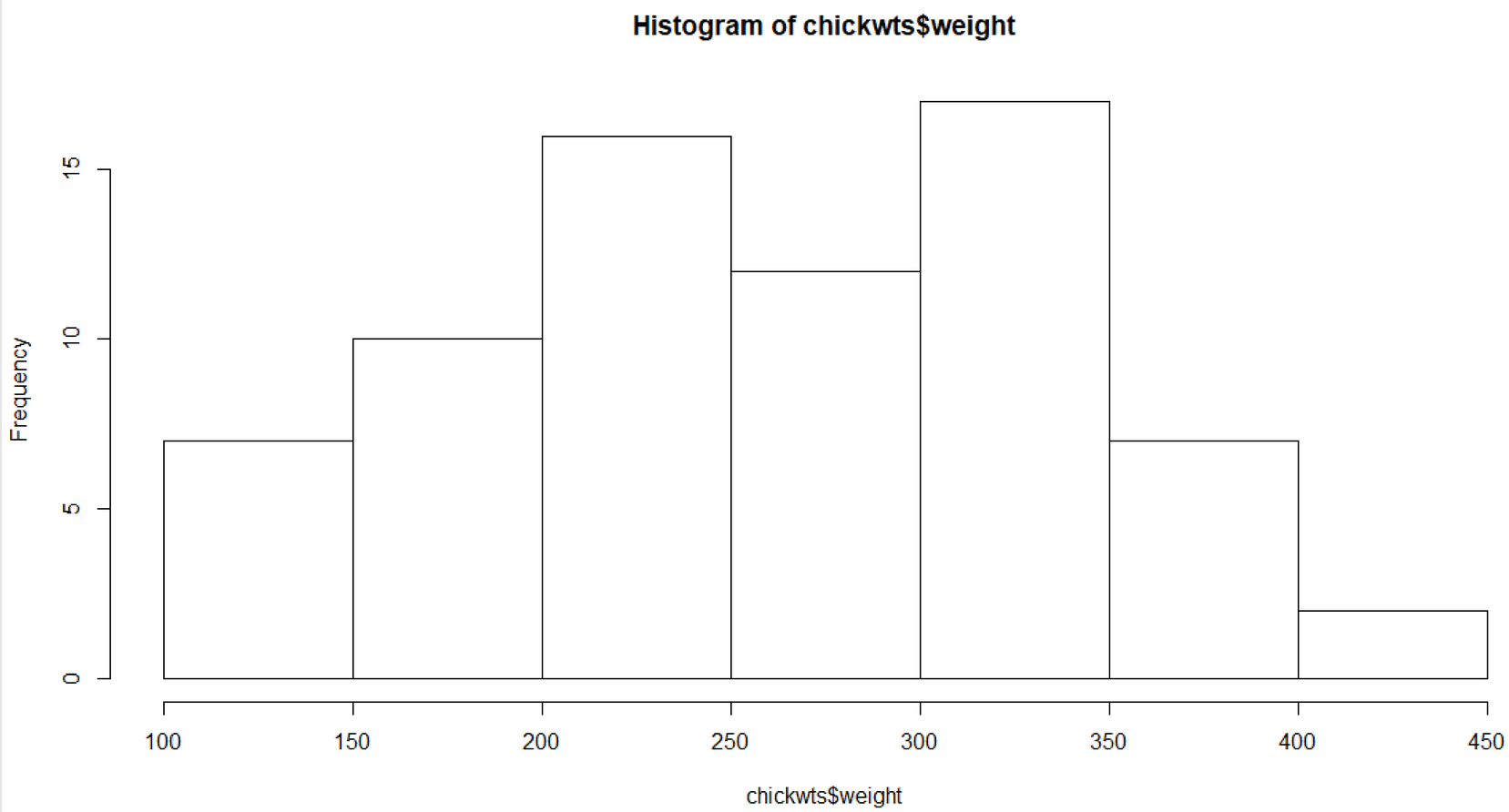
SOLUTION

```
# 2) table of the feed variable
table(chickwts$feed)
```

```
   casein horsebean   linseed  meatmeal   soybean sunflower
       12        10        12        11        14        12
```

SOLUTION

```
# 3) histogram of the weight variable
hist(chickwts$weight)
```



NUMERICAL SUMMARIES

Function	Description
?	Number of rows and columns
?	Variable names
?	First few rows or last few rows
?	Structure of data
?	Missing values
?	Summary statistics
?	Table of values

NUMERICAL SUMMARIES

Base R Function	Description
dim(), nrow(), ncol()	Number of rows and columns
names()	Variable names
head(), tail()	First few rows or last few rows
str()	Structure of data
is.na(), sum(is.na()), colSums(is.na())	Missing values
summary(), sd(), quantile(), range()	Summary statistics
table(), prop.table()	Table of values

BEFORE THE GROUP CHALLENGE

For next week (it's also on the course website):

- Read Chapter 27, sections 27.1 through 27.5 of R for Data Science, before coming to class next week.
  - This means actually reading and completing the follow-along exercises (check answers with the solutions guide on the website).
- Also read the midterm and final project pages on the course website.

- Complete homework #1 with your group (it's in the folder you downloaded for today's class).
  - In-class activities are very pointed and quick to answer. Homework assignments will have vague questions with more than one correct way to answer. Get used to it.
  - You have to explain your reasoning in homework—simply showing code or output with no explanation will earn you zero points.
  - One person in each group submits an .R script and a Word document.
  - (show *homework_1* file now)

GROUP CHALLENGE!

Spend the last 30-45 minutes of today's class session working through the *coding_exercises_1* file with your group members.