**Preview Questions**

1. What will this book teach me?
2. How is the book organized?
3. What won’t the book teach me?
4. What base level of programming experience must I have to be successful?
5. How do I run R code?
6. Where can I get help with the concepts covered in the book?
7. What is special about the online version?
8. What are the conventions used in the book
9. How should I use the code examples provided in the book?

**Summary Notes**

*Welcome {Web}*

* The book teaches how to do data science using the R programming language.
* Key tasks
  + Import data into R
  + Structure the data properly
  + Transform the data
  + Visualize the data
  + Model the data
* Key concepts
  + Grammar for graphics
  + Literate programming
  + Reproducible research
* Additional key skills
  + Cognitive frameworks to wrangle, visualize, and explore data

*What You Will Learn*

* Develop a solid foundation in the most important tools of data science.
* Workflow model
  1. Import (load data into a data frame in R)
  2. Wrangle
     1. Tidy (store data in a consistent form)
     2. Transform (narrowing in on observations, creating variables, calculating summary statistics)
  3. Visualize (map data for human interpretation; this step does not scale)
  4. Model (develop mathematical or computational tools to answer precise questions)
  5. Iterate through steps 3-5 as necessary
  6. Communicate (inform those who can take action based on the analysis)

*How This Book is Organized*

* Starts with transformation and visualization so that readers will understand the value without being disenchanted by the boredom of importing and tidying data.
* Easier to understand models if you have an understanding of visualization, tidy data, and programming.

*What You Won’t Learn*

* How to work with big data (i.e., data that is larger than 10 GB).
* Big data problems could be small data problems in disguise (i.e., use subset, subsample, multiple small data problems, etc.)
* Python, Julia, or other programming languages that are useful for data science.
* The book recommends mastering one tool at a time.
* R is specifically designed to support data science (i.e., more than just a programming language).
* Data that does not fit within the paradigm of variable and observation (i.e., rectangular data).
* Hypothesis confirmation (i.e., confirmatory analysis).
* Note that hypothesis generation (i.e., data exploration) looks at data more than once and hypothesis confirmation looks at data only one time.
* Models are useful for hypothesis generation as well as hypothesis confirmation.
* Visualizations are useful for hypothesis confirmation as well as hypothesis generation.

*Prerequisites*

* Numerical literacy and some programming experience.
* R-related tools from the comprehensive R archive network (CRAN) including R, RStudio, R packages from the tidyverse, data packages form outside the tidyverse.

*Running R Code*

* > is called the prompt
* Functions are followed by parentheses (e.g., sum(), mean(), etc.)
* Other R objects (i.e., data or function arguments) do not have parenthesis.

*Getting Help and Learning More*

* The troubleshooting process if you get stuck:
  + Perform a Google search and add “R” to the query to narrow the results.
  + Create and reprex (i.e., representative example).
  + Submit query with reprex to Stackoverflow.
* Invest a little time in learning R everyday.
  + Read the RStudio blog
  + Follow @rstudiotips on Twitter
  + Read <http://www.r-bloggers.com> which aggregates over 500 worldwide blogs about R.

*Using Code Examples*

* Generally, code examples from the book may be used in programs and documentation without permission so long as they are minimal in nature.
* Attribution is appreciated but not required.

**Preview Questions**

1. What is the objective of data exploration?
2. What is the objective of data visualization?
3. How is data visualization done?
4. What are the key functions used in data exploration and data visualization?
5. What is meant by the layered grammar of graphics?
6. What is a workflow?
7. What are objects?
8. How do you call a function?
9. How do you run code in R?
10. For what are RStudio diagnostics used?
11. What are projects?
12. What is considered real?
13. Where does your analysis live?
14. What are best practices regarding paths and directories?
15. What are RStudio projects?

**Summary Notes**

Web – Ch. 2 Introduction

* Data exploration 🡪generate many promising leads
  + Looking at data
  + Rapidly generating hypotheses
  + Quickly testing generated hypotheses
* Visualization
  + Learn basic structure of a ggplot2
  + Learn techniques for turning data into plots
* Exploratory data analysis combines visualization and transformation to ask and answer interesting questions about the data.
* Model is not addressed in the beginning because it requires additional skills in data wrangling and programming.

Web – Ch. 3 Visualization

* ggplot2 is one of several R systems for making graphs.
* ggplot2 is considered the most elegant and versatile.
* Data frame is a rectangular collection of variables (in columns) and observations (in rows).
* Complete a ggplot2 graph by adding one or more layers.
* Mapping argument is always paired with [aes()].
* Basic template  
  ggplot(data = <DATA>) +  
   <GEOM\_FUNCTION>(mapping = aes(<MAPPINGS>))
* An aesthetic is a visual property of the objects in a plot (e.g., size, shape, and color of points)
* Value refers to data; level refers to an aesthetic property.
* Scaling is assigning a unique level of an aesthetic to each unique value of a variable.
* Do not use size for a discrete variable.
* You can set the aesthetics of a geom manually.
  + Name of color as character string.
  + Size of data point in mm.
  + Shape of data point as a number from chart (R has 25 built in shapes).
* Common coding problems
  + Misplaced characters
  + Unmatched parentheses
  + Incomplete expression
  + Putting + in wrong location
* [ESCAPE] aborts processing the current command
* [?function\_name] call up help about the R function (can also select function name and press F1 in RStudio).
* Facets are subplots of one subset of the data.
* Geom is a geometrical object that a plot uses to represent data.
* Different geom functions use different visual objects to represent data.
* Not every aesthetic works with every geom.
* Use global mappings to minimize duplication in code.
* Mappings placed in a geom function are treated as local for the layer.
* A geom uses a stat (i.e., statistical transformation) to calculate new values for a graph based on the raw values of the dataset.
* [?geom\_name] shows the default stat used by the geom.
* Generally, geoms and stats can be used interchangeably.
* Every geom has a default stat; every stat has a default geom.
* Reasons to state stat explicity:
  + Override default stat
  + Override default mapping
  + Draw attention to the statistical transformation in the code
* Color bar chart using color or fill aesthetic.
* Position adjustment is used to stack bar charts.
  + Position = “identity” places each object exactly where if falls in the context of the graph.
  + Position = “fill” makes each set of stacked bars the same height.
  + Position =”dodge” places overlapping objects next to each other.
* Overplotting is when points on a grit overlap each other making it hard to see where the mass of the data is located.
* Position = “jitter” adds small amount of random noise to spread the points.
* Adding randomness makes graph less accurate at small scale but more revealing at large scale.
* Cartesian coordinate system is the default for ggplot2.
* The grammar of graphics enables you to uniquely describe any plot as a combination of:
  + Dataset
  + Geom
  + Mappings
  + Stat
  + Position adjustment
  + Coordinate system
  + Faceting scheme

Web – Ch. 4 Workflow: basics

* You can use R as a calculator
* Create new objects with < - read as “gets value” (keyboard shortcut is {Alt}+{-})
* Assignment statements have the form: object\_name < - value
* Object names must start with letter and can only have letters, numbers, “\_”, and “.”
* Use descriptive object names.
* Inspect an object in R by typing its name.
* Surrounding assignment statement with parentheses causes assignment and print to screen simultaneously.

Web – Ch. 6 Workflow: scripts

* Script editor provides more room to work than the console.
* Open script editor with File > New File > R script (keyboard shortcut {ctrl}+{shift}+{N})
* Experiment in the console and place functioning code in the script editor.
* When working in script editor {ctrl}+{enter} executes the current R expression in the console.
* In script editor {ctrl}+{shift}+{S} executes the complete script in one step.
* Always start script with the needed packages.

Web – Ch. 8 Workflow: projects

* Must decide:
  + What about analysis is “real” (i.e., lasting record of what happened).
  + Where the analysis will “live”.
* Consider R scripts as real instead of the objects listed in the environment pane.
* It’s easier to recreate the environment with R scripts than vice versa.
* Instruct RStudio not to preserve workspace between sessions.
* Use getwd() to print the working directory.
* Don’t set the working directory from within R.
* Always use Linux/Mac style forward slashes in path and directories because the backslash means something special in R.
* Always use relative paths in scripts and never absolute paths because they hinder sharing.
* Tilda points to the home directory (documents directory in Windows).
* Keep all files associated with a project together.

**Preview Questions**

1. What is a data transformation?
2. Why do we create data transformations?
3. How do you create data transformations?
4. What is a tibble?
5. How do you create a tibble?
6. How is a tibble different from a data frame?
7. What is meant by “tidy” data?
8. What are the characteristics of tidy data?

**Summary Notes**

Ch. 5 – Data Transformation

* You rarely get the data for an analysis in exactly the right form that you need.
* Data transformation is about putting the raw data into the right form for a given analysis.
* Transform data using the dplyr package, which is a core component of the tidyverse.
* dplyr overwrites some functions in base R.
* To use the base version of certain functions after loading dplyr use their full names:
  + stats::filter()
  + stats::lag()
* Tibbles are data frames slightly tweaked to work better in the tidyverse.
* Variable types:
  + int indicates integer
  + dbl indicates real numbers (i.e., doubles)
  + chr indicates character vectors (i.e., strings)
  + dttm indicates date-time (i.e., a date + a time)
  + lgl indicates logical (i.e., vectors containing only True or False)
  + fctr indicates factor (i.e., categorical variables with fixed possible values)
  + date indicates dates
* Data transformation functions
  + filter() picks observations by their values.
  + arrange() reorders rows.
  + select() picks variables by their names.
  + mutate() creates new variables with functions of existing variables.
  + summarise() collapses many values down to a single summary.
  + group\_by() operates on a dataset group-by-group.
* group\_by() can be used in conjunction with the other functions.

Chapter 10 – Tibbles

* Tibbles used instead of traditional data.frame.
* Tibbles are data frames with tweaks to some older behaviors to make tasks easier to perform.
* R is an old language; tibbles added as a package to avoid risk of breaking the existing code by changing base R.
* The terms tibble and data frame are used interchangeably.
* The tibble package is a core component of the tidyverse.
* Most R packages use regular data frames.
* as\_tibble()transform a traditional data frame to a tibble.
* tibble() creates a new tibble from individual vectors.
* Non-syntactic names are column names that are not valid R variable names.
  + Starting with character other than a letter.
  + Containing unusual characters (e.g., spaces).
* Tibbles can have non-syntactic names.
* Must use backticks to refer to non-syntactic names.
* tribble() is short for transported tibble is an alternate method of creating a tibble.
* The main differences between a tibble and a regular data.frame are printing and subsetting.
* [[ ]] extracts by name or position
* $ extracts only by name
* Format is <variable>$x or <variable>[[“x”]]
* Tibbles are more strict than data.frame; they never do partial matching.
* Some older functions don’t work with tibbles; us as.data.frame() to transform the tibble back to a data.frame when you encounter these functions.
* With tibbles, [ always returns another tibble.
* With R data frames, [ may return a data frame or a vector.

Chapter 12 – Tidy Data

* Tidy data is a consistent way to organize your data.
* The tidyr package is a core component of the tidyverse and provides tools to help tidy messy datasets.
* Three rules of tidy data:
  + Each variable must have its own column.
  + Each observation must have its own row.
  + Each value must have its own cell.
* Basic instructions for creating tidy data:
  + Put each dataset in a tibble.
  + Put each variable in a column.
* Consistent data structures make it easier to learn the analytical tools because the data have an underlying uniformity.
* Placing variables in columns makes the best use of R’s vectorized nature.
* Data may be untidy because:
  + Most people aren’t familiar with the principles of tidy data.
  + Data is often organized to for some purpose other than analysis.
* Process of tidying data:
  + First determine what are the variables and observations.
  + Resolve one of two common problems:
    - One variable is spread across multiple columns (i.e., wide data)
    - One observation is scattered across multiple rows (i.e., long data)
* gather() is used to gather a variable that is spread across columns.
* spread() is used to combined observations scattered across multiple rows.
* separate() pulls apart one column into multiple columns.
* unite() combines multiple columns into a single column.
* Values can be missing in one of two ways:
  + Explicitly when the missing data is flagged with NA.
  + Implicitly when the missing data is simply not present in the data.
* Transform explicit missing values into implicit missing values with na.rm = True in gather().
* complete() transforms implicit missing values into explicit missing values.
* fill() replaces the missing value with the most recent non-missing value.
* There are many useful and well-founded data structures that are not tidy data.
  + Substantial performance or space advantages.
  + Conventions used by specialized fields.
* Tidy data is not the only way.

Additional Questions

1. When is it every advantageous to use non-syntactic names in a tibble?

**Preview Questions**

1. Why do you need to manipulate strings?
2. How do you manipulate strings?
3. How do you manipulate dates and times?
4. What are the key challenges in working with dates and times?

**Reading Summary**

Strings

* Strings usually contain unstructured or semi-structured data.
* Regular expressions (regexps) concisely describe patterns in strings.
* stringr package is used for string manipulation.
* stringr is NOT part of the core tidyverse.
* Create strings with single quotes or doubles quotes.
* To include a literal single or double quote you must use \ to escape it.
* \n is used for a new line.
* \t is used for a tab.
* Don’t use R functions for manipulating strings because they are inconsistent.
* str\_c() combines strings.
* str\_replace\_na() prints missing values as “NA” .
* stringr drops string objects of 0 length.
* str\_sub(string, start, end) subsets a string.
* str\_to\_lower() changes text to lower case.
* str\_to\_upper() changes text to upper case.
* str\_to\_title() changes text to title case.
* str\_view(string, pattern) matches a string pattern.
* The “.” is used to match any character; escape the “.” to match it (i.e., \\.)
* Anchoring regular expressions match them from the start or end of the string.
  + Use ^ to match from the start of the string.
  + Use $ to match from the end of the string.
  + Use both ^ and $ to only match a complete string.
* \d matches any digit.
* \s matches any space, tab, or new line.
* [abc] matches a, b, or c.
* [^abc] matches anything except a, b, or c.
* To indicate how many times a pattern matches:
  + Use ? for 0 or 1
  + Use + for 1 or more
  + Use \* for 0 or more
* To specify the number of matches precisely:
  + {n} exactly n
  + {n,} n or more
  + {,m} at most m
  + {n,m} between n and m
* Refer to groups with backreferences (e.g., \1, \2)
* It’s often easier to write a series of simpler regexps.
* str\_detect() determines if a character vector matches a pattern and returns a logical.
* You can combine str\_detect() with logical operators.
* Matches never overlap.
* str\_extract() pulls out the actual text of a match.
* Use simplify = TRUE to return a matrix.
* Use parentheses to extract parts of a complex match.
* str\_replace() replaces matches with new strings.
* str\_split() breaks a string into pieces.
* str\_locate() gives the starting and ending positons of each match.
* A pattern that’s a string is automatically wrapped into a call to regex().
* Use other arguments of regex() to control match details.
  + ignore\_case = TRUE matches either upper or lowercase forms.
  + multiline = TRUE allows matches the start and end of each line.
  + comments = TRUE allows the use of comments and white space to make regexps more understandable.
  + dotall = TRUE allows “.” to match anything.
  + fixed() matches exactly the specified sequence.
  + coll() compares strings using collation rules.
* apropos() searches all objects available from the global environment.
* dir() lists all the files in a directory.
* The stringr package is built on top of the stringi package which has more functionality.

Dates and times

* lubridate package is used for working with dates.
* lubridate is NOT part of the core tidyverse.
* There are three types of date/time data:
  + Date <date>
  + Time within a day <time>
  + A date-time which is a date plus a time <dttm>
* R doesn’t have a native class for storing times; use the hms package is you need to use a time.
* Always use the simplest data type that works for your needs.
* Three ways to create a date/time:
  + From a string.
  + From individual date-time components
  + From an existing date/time object.
* To use lubridate accessor functions, identify the order in which year, month, and day appear in the dates
  + ymd, mdy, dmy
* ymd() can also accept unquoted numbers.
* To create date/time data add underscore plus “h”,”m”, and/or “s” or supply a timezone:
  + ymd\_hms(“2017-01-31 20:11:59”)
  + ymd (<date>, tz = <time zone>)
* Use make\_date() and make\_datetime() to create date/time data from individual components spread across variables.
  + Use with the mutate() function
* Use as\_datetime() and as\_date() to switch between date-time and date.
* You can pull out individual components of the date with accessor functions.
* For month() and wday() accessor functions set label = TRUE to get the abbreviated name of the month or day of the week.
* You can round the dates to a nearby unit of time.
* Use update() to modify a date-time.
* For time spans:
  + Durations are the exact number of seconds.
  + Periods are the human units (e.g., weeks, months).
  + Intervals are bound by a starting and ending point.
* Subtracting two dates returns a difftime object.
* Difftime objects records seconds, minutes, hours, days, or weeks.
* Durations have constructors:
  + dseconds(), dminutes(), dhours(), ddays(), dweeks(), dyears()
* Durations always record the time span in seconds.
* Periods have constructors:
  + seconds(), minutes(), hours(), days(), weeks(), years()
* Choosing between duration, periods, and intervals:
  + Physical time 🡪 durations
  + Human times 🡪 periods
  + Span in human units 🡪 intervals
* R uses the international standard IANA time zones.
  + <continent>/<city>
* Use Sys.timezone() to determine what time zone your computer is currently using.
* Use OlsonNames() to see the complete list of all time zone names.
* In R the time zone is an attribute of the date-time that only controls printing.
* lubridate() uses UTC (Coordinated Universal Time) unless otherwise specified.
* UTC is roughly equivalent to GMT (Greenwich Mean Time)
* with\_tz() only changes how the time is displayed.
* force\_tz() changes the underlying instant in time.
  + Use when an instant has been mislabeled with the incorrect time zone.

**Preview Questions**

1. What is relational data?
2. What are mutating joins?
3. What are filtering joins?
4. What are common join problems?
5. What are set operations?

**Reading Summary**

Relational Data

* Multiple tables of data are called relational data.
* There are three families of functions for working with relational data:
  + Mutating joins add new variables to one data frame matching observations in another.
  + Filtering joins filter observations from one data frame based on whether or not they match an observation in anther.
  + Set operations treat observations as if they were set elements.
* Primary key uniquely identifies an observation in its own table.
* Foreign key uniquely identifies an observation in another table.
* A variable can be both primary and foreign.
* Verify potential primary keys.
* A surrogate key is a variable added to a table that doesn’t have a primary key.
* Inner joins match pairs of observations whenever their keys are equal.
* Outer joins keeps observations that appear in at least one of the tables.
  + Left join keeps all observations in x.
  + Right join keeps all obervations in y.
  + Full join keeps all observations in x and y.
* Duplicate keys return all possible combinations.
* When key variables don’t have the same name, specify using the b = <variable>
* by = NULL is used for natural joins.
* Filtering joins match observations in much the same way as mutating joins.
* semi\_join() only keeps the rows in x that have a match in y.
* Use anti\_join() to diagnose join mismatches (i.e., observations in x that don’t have a match in y).
* Basic join workflow:
  + Identify primary key variables for each table.
  + Verify that none of the observations in the primary key variable are missing.
  + Verify that foreign keys match primary keys in another table [e.g., use anti\_join()].
* Simply checking the number of rows before and after a join is not sufficient to ensure the join has executed smoothly.
* Set operations are useful for breaking a single complex filter into simpler pieces.
  + intersect(x,y) returns only observations in both x and y.
  + union(x,y) returns unique observations in x and y.
  + setdiff(x,y) returns observations in x, but not in y.