

# **The R Guide for New Data Scientists**

By

Jeffrey Strickland



# The R Guide for New Data Scientists

Jeffrey Strickland

Copyright © 2022, Jeffrey S. Strickland

ISBN 978-1-6780-0244-2

This work is licensed under a Standard Copyright License. All rights reserved. Any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the author except for the use of brief quotations in a book review.

Printed in the United States of America





# Acknowledgements

---

I would like to thank the faculty and students that I worked with at the Vellore Institute of Technology (VIT) in Vellore, India, from 2016 to 2018. My experience with them was extraordinary. VIT is the top private engineering university in India and they are kind enough to consider me one of their own. I would particularly like to acknowledge my friends Aswani Kumar Cherukuri, Professor & Dean School of Information Technology & Engineering (SITE), and Chandra Mouliswaran S., Assistant Professor (SG), SITE. Along with the students I taught there in September 2016 and 2018, they inspired this book. My love for India is not surpassed by my love for its people.

*This is indeed India! ... The land of dreams and romance, of fabulous wealth and fabulous poverty, of splendour and rags, of palaces and hovels, of famine and pestilence, of genii and giants and Aladdin lamps, of tigers and elephants, the cobra and the jungle, the country of hundred nations and a hundred tongues, of a thousand religions and two million gods, cradle of the human race, birthplace of human speech, mother of history, grandmother of legend, great-grandmother of traditions, whose yesterday's bear date with the moderating antiquities for the rest of nations-the one sole country under the sun that is endowed with an imperishable interest for alien prince and alien peasant, for lettered and ignorant, wise and fool, rich and poor, bond and free, the one land that all men desire to see, and having seen once, by even a glimpse, would not give that glimpse for the shows of all the rest of the world combined.*

— Mark Twain

I owe a great deal of gratitude to my family for their support and endurance, Laurie, Miriah, and Evie. This is my 35<sup>th</sup> book along this life journey.



# Table of Contents

---

<b>ACKNOWLEDGEMENTS .....</b>	<b>I</b>
<b>TABLE OF CONTENTS.....</b>	<b>III</b>
<b>PREFACE .....</b>	<b>XIII</b>
<b>RELATED BOOKS BY THE AUTHOR.....</b>	<b>XIX</b>
<b>1. R PROGRAMMING PRIMER .....</b>	<b>1</b>
R OBJECTS AND DATA TYPES .....	1
<i>Integer Datatype</i> .....	1
<i>Numeric Datatype</i> .....	1
<i>Logical Datatype</i> .....	2
<i>Complex Datatype</i> .....	2
<i>Character Datatype</i> .....	2
<i>Raw Datatype</i> .....	2
<i>Vectors</i> .....	3
<i>Lists</i> .....	4
<i>Matrices</i> .....	4
<i>Arrays</i> .....	4
<i>Factors</i> .....	4
<i>Dataframe</i> .....	4
CREATING VECTORS.....	5
<i>Mixing Objects</i> .....	5
CREATING MATRICES.....	6
<i>cbind-ing and rbind-ing</i> .....	7
CREATING LISTS.....	7
CREATING FACTORS .....	8
DATA FRAMES.....	9
MISSING VALUES.....	10
SUBSETTING – BASICS.....	11
SUBSETTING – LISTS .....	12
SUBSETTING NESTED ELEMENTS OF A LIST.....	12
SUBSETTING – MATRICES .....	13
SUBSETTING - PARTIAL MATCHING.....	13
SUBSETTING - REMOVING MISSING VALUES.....	14
MISSING VALUES.....	15
CONTROL STRUCTURES .....	16
CONTROL STRUCTURES – IF-ELSE.....	16
CONTROL STRUCTURES - FOR.....	17
<i>Nested for loops</i> .....	18

CONTROL STRUCTURES - WHILE .....	19
CONTROL STRUCTURES – REPEAT .....	20
CONTROL STRUCTURES – NEXT, RETURN .....	20
CONTROL STRUCTURES SUMMARY .....	21
<b>2. GETTING AND CLEANING DATA.....</b>	<b>23</b>
OBTAINING DATA MOTIVATION .....	23
COMPONENTS OF TIDY DATA .....	29
DOWNLOADING FILES.....	34
<i>Downloading a File from the Web</i> .....	37
LOADING FLAT FILES - READ.TABLE().....	39
READING EXCEL FILES.....	40
READING XML.....	41
<i>Reading data from XML {XML package}</i> .....	42
<i>Directly access parts of the XML document</i> .....	42
<i>Programmatically extract parts of the file</i> .....	43
READING JSON .....	43
<i>Reading data from JSON {jsonlite package}</i> .....	43
<i>Nested objects in JSON</i> .....	44
READING FROM APIs.....	44
THE DATA.TABLE PACKAGE .....	46
<i>See all the data tables in memory</i> .....	46
<i>Subsetting rows</i> .....	47
<i>Subsetting columns</i> .....	47
<i>Calculating values for variables with expressions</i> .....	47
<i>Adding new columns</i> .....	47
<i>Adding new columns</i> .....	48
<i>Multiple operations</i> .....	48
<i>plyr like operations</i> .....	48
<i>Special variables</i> .....	48
<i>Keys</i> .....	48
<i>Joins</i> .....	49
<i>Fast Reading</i> .....	49
EXAMPLE 2-1. TIDY DATA CODEBOOK.....	49
<i>Solution to Example 1</i> .....	49
<i>Step 1. Download and Load the Data</i> .....	50
<i>STEP 2. Merging Training and Test Set into one Data Set</i> .....	50
<i>STEP 3. Set Merging</i> .....	51
<i>STEP 4. Measure Extraction</i> .....	52
<i>STEP 5. Activity Naming</i> .....	52
<i>STEP 6. Code Labeling Process</i> .....	52
<i>STEP 7. Tidy Data Creation &amp; Final Dataset</i> .....	53

<i>STEP 8. Final File Checking</i> .....	54
<i>Step 9. File Export FinalData into FinalData.txt file</i> .....	54
<i>STEP 10. Print Final Data Set</i> .....	54
EXAMPLE 2-2. HOSPITAL RANKINGS.....	56
<i>Read the downloaded data from Hospital Compare</i> .....	57
<i>Explore the Hospital Care Data</i> .....	57
<i>30-Day Mortality Rates for Heart Attacks</i> .....	58
EXAMPLE 2-3. BEST HOSPITAL IN A STATE.....	59
EXAMPLE 2-4. RANK OF HOSPITALS BY OUTCOME IN A STATE .....	61
<i>Sample Outputs from rankhospital</i> .....	62
EXAMPLE 2-5. RANKING HOSPITALS IN ALL STATES .....	62
<i>Rankings for Best Hospital for a medical condition in the State</i> .....	64
SUMMARY.....	65
<b>3. EXPLORATORY DATA ANALYSIS (EDA) .....</b>	<b>67</b>
INTRODUCTION .....	67
PRINCIPLES OF ANALYTIC GRAPHICS .....	67
<i>First Principle</i> .....	68
<i>Example 3-1</i> .....	69
<i>Second Principle</i> .....	71
<i>Example 3-2</i> .....	71
<i>Third Principle</i> .....	73
EXAMPLE 3-3 - PENGUINS.....	73
<i>MICE Package</i> .....	74
<i>Install and Load Libraries</i> .....	75
<i>Set working directory is needed and map the path</i> .....	75
<i>Get zip Data from GitHub</i> .....	75
<i>Create dataframe from penguin-size data</i> .....	75
<i>Check for Missing Values</i> .....	76
<i>Plot missing data patterns</i> .....	76
<i>Impute missing values</i> .....	78
<i>Check Imputed Dataset</i> .....	80
<i>Get complete data (3rd out of 7)</i> .....	80
<i>Recheck for missing values</i> .....	81
<i>Create dataframe</i> .....	81
<i>Define x and y variables</i> .....	81
<i>Create scatterplot</i> .....	81
<i>Create Coplot by Species</i> .....	82
<i>Conclusion</i> .....	83
FOURTH PRINCIPLE.....	83
EXAMPLE 3-4 – COURSE PARTICULATE MATTER.....	84
FIFTH PRINCIPLE .....	85

SIXTH PRINCIPLE .....	85
PRINCIPLES SUMMARY.....	85
EXAMPLE 3-5 – POLLUTION MONITORING.....	86
<i>Solution to Example 3-5.....</i>	86
EXAMPLE 3-6 – POLLUTION MONITORING (CONTINUED) .....	87
<i>Solution to Example 3-6.....</i>	87
<i>Conclusion to Example 3-6.....</i>	90
EXAMPLE 3-7 – EMISSIONS IN BALTIMORE.....	91
<i>Solution to Example 3-7.....</i>	91
<i>Conclusion to Example 3-7.....</i>	92
EXAMPLE 3-8 – EMISSION SOURCE COMPARISON.....	93
<i>Solution to Example 3-8.....</i>	93
<i>Conclusion to Example 3-8:.....</i>	95
EXAMPLE 3-9 – COAL COMBUSTION EMISSIONS .....	95
<i>Solution to Example 3-9.....</i>	95
<i>Conclusion to Example 3-9:.....</i>	96
EXAMPLE 3-10 – MOTOR VEHICLE EMISSIONS .....	96
<i>Solution to Example 3-10 .....</i>	97
<i>Conclusion to Example 3-10: .....</i>	97
EXAMPLE 3-11 - VEHICLE EMISSION SOURCES.....	97
<i>Solution to Example 3-11 .....</i>	98
<i>Conclusion to Example 3-11: .....</i>	102
<b>4. GITHUB.....</b>	<b>103</b>
USING GITHUB.....	103
GETTING STARTED WITH GITHUB.....	103
CREATING R PROJECTS.....	107
PUSHING FILES FROM RSTUDIO .....	110
<b>5. REPRODUCIBLE RESEARCH .....</b>	<b>119</b>
LITERATE (STATISTICAL) PROGRAMMING.....	121
SCRIPTING YOUR ANALYSIS .....	122
STRUCTURE OF A DATA ANALYSIS.....	123
SUMMARY.....	123
EXAMPLE 5-1. FITNESS.....	124
<i>Explore the Data .....</i>	125
<i>Mean Total Number of Steps per Day.....</i>	126
<i>Mean-Median number of Steps per day .....</i>	127
<i>Calculate Mean and Median Total Steps per Day .....</i>	128
<i>Plot the histogram.....</i>	129
AVERAGE DAILY ACTIVITY PATTERN.....	130
<i>Plot customization.....</i>	130

<i>Time series plot</i> .....	131
<i>Sample of some averages in steps_by_interval</i> .....	132
<i>Imputing Missing Values</i> .....	133
<i>Calculate number of missing values</i> .....	133
<i>Replace missing step values</i> .....	133
<i>Create object to hold imputed values</i> .....	134
<i>Find and replace missing values</i> .....	134
<i>Define comparison data</i> .....	134
<i>Calculate and Report the Mean and Median Total Steps per Day</i> .....	136
<i>Weekend and Weekday Activity Patterns</i> .....	137
<i>Create the names</i> .....	138
<i>Make a panel plot</i> .....	138
EXAMPLE 5-2. SEVERE WEATHER EFFECT.....	139
<i>Introduction</i> .....	139
<i>Analysis Questions and Outcomes</i> .....	139
DATA PROCESSING .....	140
<i>Install and Load Packages</i> .....	140
<i>Load the Weather Data</i> .....	140
<i>Exploratory Data Analysis (EDA)</i> .....	141
<i>Populate Initial Analysis Dataset</i> .....	141
<i>Prepare the Storm Data for Analysis</i> .....	142
<i>Compile the Data for Fatalities and Injuries Analysis</i> .....	146
<i>Compile the Data for Property and Crop damage Analysis</i> .....	146
<i>Build Lookup Tables</i> .....	147
<i>Merge Lookup Tables into Storm Data</i> .....	148
RESULTS.....	150
<i>Public Health - Fatalities</i> .....	150
<i>Public Health - Injuries</i> .....	153
<i>Property Damage</i> .....	154
<i>Crop Damage</i> .....	157
CONCLUSION.....	158
<b>6. INFERENTIAL STATISTICS .....</b>	<b>161</b>
OVERVIEW .....	161
ESTIMATING POPULATION PARAMETERS.....	161
CONFIDENCE INTERVALS.....	162
LEVELS OF MEASUREMENT .....	163
EXAMPLE 6-1. POINT ESTIMATE AND CONFIDENCE INTERVAL.....	164
COMPARISON TESTS.....	165
CORRELATION TESTS.....	166
EXAMPLE 6-2. THE CENTRAL LIMIT THEOREM.....	166
<i>Overview</i> .....	166

<i>The Exponential Distribution</i> .....	167
<i>Simulation of the Exponential(<math>\lambda</math>) distribution</i> .....	167
<i>Set up the Simulation</i> .....	167
<i>Applying the CLT</i> .....	168
<i>The Normal Distribution, <math>Normal(\mu, \sigma)</math></i> .....	169
<i>Normal Random Variates</i> .....	169
<i>The Mean Value</i> .....	169
<i>The Variance</i> .....	170
<i>Histogram Approximation of the Normal Distribution</i> .....	170
<b>EXAMPLE 6-3. TOOTH GROWTH AND NUTRIENTS</b> .....	172
<i>Assumptions</i> .....	172
<i>Exploratory Data Analysis (EDA)</i> .....	173
<i>ToothGrowth Description</i> .....	173
<i>ToothGrowth Summary</i> .....	173
<i>Data Visualization</i> .....	174
<i>Boxplot Tooth Growth Factors</i> .....	175
<i>Boxplot Comparison of Orange Juice vs Vitamin C</i> .....	175
<i>Boxplot Comparison of Three Dosage Levels</i> .....	177
<i>Hypothesis Test</i> .....	179
<i>Hypothesis Test on the Effect of Supplement Types</i> .....	179
<i>Hypothesis Test on the Effect of Dose Types</i> .....	181
<i>Supplement as a Factor within Dose Levels</i> .....	182
<b>7. REGRESSION MODELS</b> .....	185
<b>EXAMPLE 7-1 – GALTON STUDY</b> .....	185
<i>Finding the middle via least squares</i> .....	186
<i>Experiment</i> .....	187
<i>General least squares for linear equations</i> .....	189
<i>Fitting the best line</i> .....	190
<i>Regression through the origin</i> .....	191
<i>The solution</i> .....	193
<i>Interpreting regression coefficients, the slope</i> .....	195
<b>RESIDUALS</b> .....	196
<i>Properties of the residuals</i> .....	196
<i>Quantile-Quantile Plots</i> .....	197
<i>Density Plot of the Residuals</i> .....	197
<i>Estimating residual variation</i> .....	200
<b>ADJUSTMENT</b> .....	200
<b>EXAMPLE 7-2. MOTOR TREND MAGAZINE MPG STUDY</b> .....	201
<i>Data Processing</i> .....	201
<i>Exploratory Data Analysis (EDA)</i> .....	202
<i>Data Preparation</i> .....	203

<i>Hypothesis Test</i> .....	203
<i>MPG difference between automatic and manual transmissions</i> .....	204
<i>Conclusion</i> .....	207
<i>Appendix A</i> .....	207
<i>Appendix B</i> .....	208
<b>8. GENERALIZED LINEAR MODELS .....</b>	<b>209</b>
INTUITION .....	210
OVERVIEW .....	212
MODEL COMPONENTS .....	212
EXAMPLE 8-1 - MACROECONOMICS .....	216
<i>Viewing the Data</i> .....	216
EXAMPLE 8-2 – AIR QUALITY .....	221
DATA PREPARATION.....	222
<i>Training a GLM</i> .....	222
DEVIANCE RESIDUALS .....	223
RESPONSE RESIDUALS.....	224
WORKING RESIDUALS .....	225
PEARSON RESIDUALS.....	225
DEVIANCE RESIDUALS .....	225
DEVIANCE RESIDUALS IN PRACTICE .....	227
NULL AND RESIDUAL DEVIANCE .....	227
NULL DEVIANCE AND RESIDUAL DEVIANCE IN PRACTICE .....	228
OTHER OUTPUTS OF THE SUMMARY FUNCTION .....	228
<i>Dispersion parameter</i> .....	228
<i>AIC</i> .....	229
<i>Fisher scoring iterations</i> .....	229
THE PREDICTION FUNCTION OF GLMs .....	229
<i>The type argument</i> .....	229
<i>Obtaining confidence intervals</i> .....	231
<b>9. MACHINE LEARNING AND PREDICTION .....</b>	<b>235</b>
PREDICTION.....	235
<i>Cross-Validation</i> .....	237
<i>Types of Errors</i> .....	237
EXAMPLE 9-1 – SAMPLE SIZE .....	239
<i>t-tests</i> .....	240
<i>Tests of Proportions</i> .....	241
<i>Correlations</i> .....	242
<i>Creating Power or Sample Size Plots</i> .....	242
<i>Prediction Study Design</i> .....	243
EXAMPLE 9-2. EXERCISE GADGETS.....	244

<i>Data Preprocessing</i>	244
<i>Data Preparation</i>	244
<i>Cleaning the Data</i>	245
<i>Resetting the Training Set</i>	246
<i>Data Coercion</i>	246
<i>Finding the Best ML Model</i>	246
<i>Ensemble Learning</i>	246
<i>Random Forest</i>	247
<i>Stochastic Gradient Descent</i>	247
<i>Bagging (Bootstrap Aggregating)</i>	247
<i>Boosted Logistic Regression</i>	247
<i>Bagged Classification and Regression Trees (CART)</i>	247
<i>Train ML Algorithms</i>	248
<i>Prepare Training Scheme</i>	248
<i>boxplots of results</i>	250
<i>Cross-Validation</i>	250
<b>EXAMPLE 9-3. USING R FOR CRIME ANALYSIS</b>	252
<i>Introduction</i>	252
<i>The Crime Data</i>	252
<i>Installing Libraries &amp; Loading Data</i>	253
<i>Display Data</i>	254
<i>Preprocess the Data</i>	254
<i>Explore the Data</i>	255
<i>Crime Over Time</i>	259
<i>Crimes Series Plot</i>	259
<i>Aggregated Data</i>	261
<i>Creating a Bar Chart</i>	261
<i>Creating a pie chart</i>	262
<i>Temporal Trends</i>	263
<i>Theft Time Heatmap</i>	264
<i>Reorder and format Factors</i>	265
<i>Creating a Time Heatmap</i>	266
<i>Arrests Over Time</i>	268
<i>Daily Arrests</i>	268
<i>Number of Arrest by Time of Arrest</i>	269
<i>Factor by Crime Category</i>	272
<i>Arrests by Category and time of Arrest</i>	274
<i>Normalized Gradients</i>	276
<i>Factor by Police District</i>	278
<i>Factor by Month</i>	280
<i>Factor By Year</i>	282
<i>Police Arrest Normalized by Year</i>	282

<b>10. ML ENSEMBLES - RANDOM FORESTS.....</b>	<b>285</b>
HISTORY.....	285
ALGORITHM.....	286
<i>Bootstrap aggregating .....</i>	287
<i>Description of the technique.....</i>	287
EXAMPLE 10-1: OZONE DATA.....	288
<i>Bagging for nearest neighbor classifiers.....</i>	289
FROM BAGGING TO RANDOM FORESTS.....	290
<i>Random subspace method.....</i>	290
<i>Algorithm.....</i>	291
<i>Relationship to Nearest Neighbors.....</i>	291
VARIABLE IMPORTANCE .....	292
VARIANTS.....	293
EXAMPLE 10-2: RANDOM FOREST USING R.....	293
<i>Load libraries .....</i>	294
<i>Medical longevity Study of primary biliary cirrhosis (PBC) .....</i>	295
<i>Get transformed data .....</i>	297
<i>Reshaping Variable .....</i>	298
<i>Plot continuous variables .....</i>	299
<i>Show multiple plots in a window.....</i>	299
<i>Create Trial and Test Sets .....</i>	300
<i>Create the Survival Probability Function.....</i>	301
<i>Plot the survival Probability Function.....</i>	301
<i>Plot the cumulative hazard function.....</i>	302
<i>Using shiny GUI for colorspace.....</i>	305
<i>Analog to: choose_palette(gui = "shiny").....</i>	305
<i>Grow and Store the Random Survival Forest.....</i>	307
<i>Predict Patient Survival .....</i>	309
<i>Print prediction summary .....</i>	310
<i>Variable Importance .....</i>	311
<i>Minimal Depth .....</i>	314
<i>Return an object with both minimal depth and vimp measures.....</i>	315
<i>Both minimal depth and VIMP .....</i>	317
<i>Get the minimal depth selected variables .....</i>	317
<i>Data generation .....</i>	319
<i>Bilirubin variable dependence plot.....</i>	319
<i>Pull the categorical variables .....</i>	319
<i>Continuous Variable Dependence Plots .....</i>	320
<i>Partial Dependence Plot.....</i>	321
<i>Categorical Features .....</i>	323
<i>Conditional Dependence Plots .....</i>	333
<i>Partial Dependence Coplots.....</i>	335

<i>Partial Coplot</i> .....	336
EXAMPLE 10-3: BUSINESS APPLICATION .....	338
<i>Scenario and dataset</i> .....	338
<i>Read and Explore data</i> .....	339
<i>Make Formula</i> .....	340
<i>Building Random Forest using R</i> .....	340
<i>Variable Importance Plot</i> .....	341
<i>Variable Importance Table</i> .....	342
<i>Predict Response Variable Value using Random Forest</i> .....	343
<i>Confusion Matrix</i> .....	343
<i>Create Confusion Matrix</i> .....	344
<i>Predicting response variable</i> .....	344
<i>Create Confusion Matrix</i> .....	344
EXAMPLE 10-4: FIT AND COMPARE FGL DATA RF AND SVM .....	345
<b>11. SHINY APPS .....</b>	<b>349</b>
STRUCTURE OF A SHINY APP .....	349
EXAMPLE 11-1. HELLO SHINY! APP .....	350
<i>User Interface (ui)</i> .....	350
<i>Server</i> .....	350
<i>Running an App</i> .....	351
EXAMPLE 11-2. NEXT WORD PREDICTION MODEL.....	353
EXAMPLE 11-3: COVID-19 DATA DISCOVERY APP .....	358
<i>User Interface (ui) Code</i> .....	359
<i>Server Code</i> .....	361
DEPLOYING SHINY APPS ON THE WEB .....	364
<b>REFERENCES.....</b>	<b>365</b>
<b>INDEX.....</b>	<b>371</b>

# Preface

---

To write a single book about data science, at least as I view the discipline, would result in several volumes, and it has. I have come to view Data Science kind of like Engineering. We have all sorts of engineers: mechanical, electrical, civil, aeronautical, industrial, and so on. We still have them, but when we talk about them, we tend to use the general term "engineers" and their field as "engineering." I have thought about this for a few years, and I have concluded that we do something similar with the terms "data scientists" and "data science." Although this book covers a lot of what I include as data science, it is written with the beginner in mind. I have written other books that are more specialized for more experienced users.

*Data really powers everything that we do.*

– Jeff Weiner, LinkedIn

## What Comprises Data Science?

**Programming** (Computer Science). Data scientist convert data into a valuable format for investigation, inquiry, or analysis. This data conversion involves an extraction, transformation, and loading (ETL) effort to place data into a data repository, like a data warehouse or data lake. One of the data scientist's extremely valuable and sought-after skills is the capability to design, build, and execute computer code that structures data and manipulates unstructured data. Along with ETL, these three conceptual steps are the basis for the design and structure of most data pipelines. They serve as an outline or blueprint the way raw data are transformed to data that is ready for consumption.

**Data Preprocessing** (Information Technology). Many people who write about data science place preprocessing data in the field of information technology (IT), and indeed IT seems an appropriate. However, I now claim that this may be within the discipline of IT. The lines between IT and data science are blurred, enough so that the data scientist's primary training has been in IT. My data science students at the Vellore Institute of Technology (VIT) were in the

School of Information Technology & Engineering (SITE). Some were working on projects involving the writing of Hindi lexicons for text analytics using the Python programming language—quite different than may earlier image of IT.

**Data Mining** (Data Analysis, Statistics, Information Technology). Data mining is the semi-automatic or automatic process of taking large quantities of data and extracting data that is pertinent to an organization's data operations. Such mining pulls out interesting patterns such as groups of data records (cluster analysis); identifies items, events or observations which do not conform to an expected pattern (anomaly detection or outlier detection); finds frequent co-occurring associations among a collection of items (association rule mining or market basket analysis); and discovers statistically relevant patterns between data examples where the values are delivered in a sequence (sequential pattern mining). These mining tasks usually involve using database techniques to find patterns, which may be used in further analysis or, for example, in machine learning and predictive analytics. However, the data collection, data preparation, as well as the result interpretation and reporting is not part of the data mining step.

*In God we trust. All others must bring data.*

— W. Edwards Deming, statistician

**Business Intelligence** (BI Engineering, BI Development, BI Analysis). Business intelligence (BI), often referred to as business analytics, pursues the transform of data into actionable intelligence that informs an organization's strategic and tactical business decisions. In BI, analysts use software tools, such as Tableau and SAS, to access and analyze pertinent business data and present the results, in a variety of "story telling" forms, to provide stakeholders with detailed intelligence about the state of the business. Some experts distinguish between BI and business analytics, but I treat them as the same, with the additional idea that they are a specialized area of data analytics. Consequently, BI may describe a past or current state of the business and it also analyzes data to predict what will happen or what could happen by taking a certain approach.

*The big technology trend is to make systems intelligent and data is the raw material.*

— Amod Malviya, CTO at flipkart

**Machine Learning** (Data Science, Statistics, Computer Science, Applied Physic, Biomedicine, Cognitive Science). Machine learning (ML) is not the same thing as artificial intelligence (AI). ML is the science of getting computers to act without being explicitly programmed (the domain of AI). In fact, many researchers think that ML is the best way to make progress towards human-level AI. In data science, ML or the algorithms of ML are used to mine data, explore mined data, and predict future outcomes (among other things), and can be considered as a part of data analytics or data mining. I make a distinction here to emphasize that ML does not make use of traditional statistical of mathematics methods. ML algorithms may include artificial neural networks (ANN), random forests (RF), genetic algorithms (GA), and many other methods. ML is particularly useful in text analytics, where statistical methods are inappropriate.

*I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.*

— Alan Turing, Computing machinery and intelligence

**Data Analytics** (Data Scientist, Field Specific Data Analysis). Data analytics is a process of examining, cleaning, transforming, and modeling data with the goal of discovering useful information, informing conclusions, and providing decision support. Data analytics encompasses multiple methods and approaches within the realms of descriptive, predictive, and prescriptive analysis, while being used in different business, science, medical, psychological, and social science domains. The goals of data analytics are discovering useful information, informing conclusions, and supporting decision-making. Once the data is cleaned, data analysts apply a variety of techniques, referred to as exploratory data analysis, to begin understanding the information contained in the data. Data exploration can result in additional data cleaning or additional data

requirements, so these activities may be iterative. In descriptive analytics, unfolding the characteristics of the data with such measures as the mean and variance may help in understanding the data. Predictive analytics is concerned with forecasting future events based on the data that has been mined and explored. Prescriptive analytics is concerned with telling the story of why the data describes the present or predicts the future to help decision makers choose the best courses of action.

Descriptive Analytics, which use data aggregation and data mining to provide insight into the past and answer: "What has happened?"

Predictive Analytics, which use statistical models and forecasts techniques to understand the future and answer: "What could happen?"

*The goal is to turn data into information, and information into insight.*

— Carly Fiorina, former CEO, Hewlett-Packard

**Statistics** (Statistics, Applied Mathematics). In my assessment, this is the foundation of data science. It is the science that deals with the collection, classification, analysis, and interpretation of numerical facts or data. Supported directly by use of mathematical theories of probability, statistics imposes order and regularity on collections of dissimilar elements. This is the first time I have included a fair coverage of inferential statistics, regression models, and generalized linear models (GLMs) in a data science book. But, people generally do not know how to read, interpret, and use statistics, which must be governed by the following inquiries:

- How was the data collected?
- Does the evidence come from reliable sources?
- What is the data's background?
- Are all data reported?
- Have the data been interpreted correctly?

**Scientific Collaboration and Publication** (All Data-Centered Scientific Disciplines) GitHub has revolutionized programming in general and data science programming specifically. Git is a free and open-source distributed version control system (configuration management) designed to handle everything from small to very large projects with speed and efficiency. Anyone taking data science courses at an accredited college or university, or as no-credit training through Coursera, will need to be familiar with GitHub. We'll talk more about GitHub in Chapter 5.

**Data Dashboards and Apps** (Data Science, Business Intelligence, Other Domain-Specific Disciplines). Commercial software tools, like Tableau, has made the creation of dashboards convenient, cost-effective, useful, and should be included in the data scientists' toolbox. Google Analytics is where dashboards were born, and Google Analytics 360 is Google's current powerhouse. The COVID-19 pandemic of 2020 brought other dashboards to the fore, with the Johns Hopkins coronavirus tracker (JHU, 2021) and the UK government coronavirus tracker (GOV.UK, 2022) being good examples. Here, we'll talk about Shiny Apps with R Shiny (Chapter 11). If you want your data science products used customers, you have to build intuitive dashboards for them.

**Interdisciplinary.** In a nutshell, nearly any scientist may be a data scientist, even though there may be contention with this statement. In data science, we use statistics, which entails the application of mathematical and computational aspect of data science and suggests decision options to take advantage of the results of foundational descriptive and predictive analytics, where statistical methods and models prevail. We also use machine learning, which is highly specialized in terms of coding and algorithm construction. I have also performed data mining while doing the stuff of data science, so our IT friends are involved as well. It's like bringing a combined arms force to bear on a stubborn, defending enemy to drive them from their stronghold and reveal their vulnerabilities.

*Torture the data, and it will confess to anything.*

– Ronald Coase, winner of the Nobel Prize in Economics

## About R and R-Studio

I wrote about 80% of this book using my R scripts in R-Studio via R Markdown. Throughout the book, I have inserted "red-box" Markdown Notes like this:

**Markdown Note.** Markdown provides an authoring framework for data science. You can use a single R Markdown file to both

- save and execute code
- generate high quality reports that can be shared with an audience

R Markdown documents are fully reproducible and support dozens of static and dynamic output formats, including Word, HTML, PowerPoint, and more. Like the rest of R, R Markdown is free and open source.

## Pricing Philosophy

I try to make my books as affordable as possible. I do my own markdown, typing, graphics, interior design, and exterior design. Lulu.com prints and binds the books for an incredibly low cost and provide Global distribution. I take a very small percentage for myself—I would write without compensation, if necessary. Since about 2018, color printing has become very economical, so my books are now in full-color but does not drive the cost upward very much.

On the down-side, I do not pay an editor, a graphics design artist, or marketing. So, you might get a minor typo, a misaligned graphic, or some weird colors. If you can live with that, then enjoy the book...

## GitHub.

We'll discuss GitHub in Chapter 4, but for now, all the material in this book resides in my repositories there: <https://github.com/stricje1>.

## **Related Books by the Author**

---

*Time Series Analysis and Forecasting using Python & R.* Copyright© 2020, Lulu, Inc. ISBN 978-1-716-45113-3

*Data Science Applications using Python and R.* Copyright© 2020, Lulu, Inc. ISBN 978-1-716-89644-6

*Data Science Applications using R.* Copyright© 2020, Lulu, Inc. ISBN 978-0-359-81042-0

*Logistic Regression Inside-Out.* Copyright © 2020 by Jeffrey S. Strickland. Lulu, Inc. ISBN 978-1-365-81915-5

*Predictive Crime Analysis using R.* Copyright© 2018, Jeffrey Strickland. Lulu, Inc. ISBN 978-0-359-43159-5

*Time Series Analysis using Open-Source Tools.* Copyright© 2016, Jeffrey Strickland. Glasstree, Inc. ISBN 978-1-5342-0100-2

*Predictive Analytics using R.* Copyright © 2016 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-84101-7

*Introduction to Crime Analysis and Mapping.* © 2016 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-19311-6

*Data Analytics Using Open-Source Tools.* Copyright © 2015 by Jeffrey Strickland. Lulu Inc. ISBN 978-1-365-21384-7

*Data Science and Analytics for Ordinary People.* Copyright © 2015 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-329-28062-5

*Operations Research using Open-Source Tools.* Copyright © 2015 by Jeffrey Strickland. Lulu Inc. ISBN 978-1-329-00404-7

*Missile Flight Simulation - Surface-to-Air Missiles,* 2<sup>nd</sup> Edition. Copyright © 2015 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-329-64495-3

*Predictive Modeling and Analytics.* © 2014 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-37544-4

*Verification and Validation for Modeling and Simulation.* Copyright © 2014 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-74061-7

*Using Math to Defeat the Enemy: Combat Modeling for Simulation.* © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-257-83225-5

*Mathematical Modeling of Warfare and Combat Phenomenon.*  
Copyright © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-45839255-8

*Simulation Conceptual Modeling.* Copyright © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-105-18162-7.

*Discrete Event Simulation using ExtendSim 8.* Copyright © 2010 by Jeffrey S. Strickland. Lulu.com. ISBN 978-0-557-72821-3

# 1. R Programming Primer

## R Objects and Data Types

In this chapter, we look at different objects and data types that are used in R and some basic operations on those data types. First, let's talk about objects in general. All the things that we encounter in R are called objects. There are different kinds of objects that contain different kinds of data types. But everything in R, is an object. So, the objects in R hold particular data types. Let's establish some definitions.

**Definition 1-1.** A **data type** is a particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it.

In particular, R has five basic data types (see **Table 1-1**).

- Raw
- Numeric: integer
- Numeric: double (real)
- Character
- Complex
- Logical

### *Integer Datatype*

R supports integer data types which are the set of all integers. You can create as well as convert a value into an integer type using the `as.integer()` function. You can also use the capital '`L`' notation as a suffix to denote that a particular value is of the integer data type.

### *Numeric Datatype*

Decimal values are called numerics in R. It is the default data type for numbers in R. If you assign a decimal value to a variable `x` as follows, `x` will be of numeric type.

## **Logical Datatype**

R has logical data types that take either a value of true or false., often represents by 1 (true) and 0 (false). A logical value is often created via a comparison between variables.

## **Complex Datatype**

R supports complex data types that are set of all the complex numbers. The complex data type is to store numbers with an imaginary component.

## **Character Datatype**

R supports character data types where you have all the alphabets and special characters. It stores character values or strings. Strings in R can contain alphabets, numbers, and symbols. The easiest way to denote that a value is of character type in R is to wrap the value inside single or double inverted commas.

## **Raw Datatype**

A Raw data is a dataset that has been downloaded from web (or any other source) and has not been processed yet. Raw data is not ready for use in statistics. It needs various processing tools to be ready for analysis.

**Table 1-1.** Datatypes and verification steps in R

Data Type	Example	Verify
Logical	TRUE, FALSE	<pre>v &lt;- TRUE print(class(v))</pre> it produces the following result – [1] "logical"
Numeric	12.3, 5, 999	<pre>v &lt;- 23.5 print(class(v))</pre> it produces the following result – [1] "numeric"
Integer	2L, 34L, 0L	<pre>v &lt;- 2L print(class(v))</pre> it produces the following result – [1] "integer"

Complex	3 + 2i	<pre>v &lt;- 2+5i print(class(v))</pre> it produces the following result – [1] "complex"
Character	'a', "good", "TRUE", '23.4'	<pre>v &lt;- "TRUE" print(class(v))</pre> it produces the following result – [1] "character"
Raw	"Hello" is stored as 48 65 6c 6c 6f	<pre>v &lt;- charToRaw("Hello") print(class(v))</pre> it produces the following result – [1] "raw"

Now, we define a data object.

**Definition 1-2.** The **data object** is a location or region of storage that contains a collection of attributes or groups of values that act as an aspect, characteristic, quality, or descriptor of the object. A vehicle is a data object which can be defined or described with the help of a set of attributes or data.

R has six kinds of objects

- Atomic (vectors)
- Lists
- Matrices
- Arrays
- Factors
- Dataframes

### Vectors

Vectors are one of the basic R programming data objects. They are six types of atomic vectors, based on the data types they hold— logical, integer, character, raw, double, and complex. However, a vector cannot hold more than one type of data. Empty vectors can be created with the `vector()` function.

## **Lists**

Lists are data objects of R that contain various types of elements including strings, numbers, vectors, and a nested list inside it. It can also consist of matrices or functions as elements. It can be created with the help of the `list()` function. Unlike vectors, list can different data types in the same object.

## **Matrices**

Matrices in R Programming are used to arrange elements in the two-dimensional layout. They contain elements of the same data type. They usually contain numeric values in order to perform mathematical operations.

## **Arrays**

An **array** is used to store data in more than just 2 dimensions. It is used to store multi-dimensional data in the required format. It can be created with the help of an `array()` function.

## **Factors**

Factors are data objects that are used in order to categorize and store data as levels. They can be strings or integers. They are extremely useful in data analytics for statistical modeling. They can be created using `factor()` function.

## **Dataframe**

Dataframes are 2-dimensional data structures wherein each column consists of the value of one variable and each row consists of a value set from each column.

There's another special value called NAN or Nan. This represents an undefined value. For example, if you take a value divided by zero, the result is undefined and “not a number” or NaN. NaN can also be thought of as a missing value but we'll talk about missing values later.

R objects can have attributes

- names, dimnames
- dimensions (e.g., matrices, arrays)
- class

- length
- other user-defined attributes/metadata

Each object in R is an attribute. Not every object in R necessarily has attributes, but the object itself is an attribute. A common type of attributes that we'll encounter are names or dim names or dimension names. Attributes of an object can be accessed using the `attributes()` function.

A matrix will have dimensions, for example. It will have a number of rows and a number of columns and if you have a multidimensional array, we'll have more than two dimensions.

Every object also has a class. For example, numeric objects have the class numeric and integer objects have the class integer. Every object also has a length. For a vector, it's quite simple: the length of the object is just the number of elements in the vector.

There are other user-defined attributes or metadatas, which can be defined. There is a general function called attributes which allows you to set or modify the attributes for an R object.

Source Code: [https://github.com/stricje1/R\\_Programming](https://github.com/stricje1/R_Programming)

## Creating Vectors

The `c()` function can be used to create vectors of objects.

```
x <- c(0.5, 0.6)    ## numeric
x <- c(TRUE, FALSE) ## logical
x <- c(T, F) ## logical
x <- c("a", "b", "c")      ## character
x <- 9:29      ## integer
x <- c(1+0i, 2+4i) ## complex
```

Using the `vector()` function

```
x <- vector("numeric", length = 10)
x
## [1] 0 0 0 0 0 0 0 0 0 0
```

## Mixing Objects

What happens with the following vectors?

```
y <- c(1.7, "a")    ## character
y <- c(TRUE, 2)     ## numeric
y <- c("a", TRUE)   ## character
```

When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

```
x <- 0:6
class(x)
## [1] "integer"
as.numeric(x)
[1] 0 1 2 3 4 5 6
as.logical(x)
## [1] FALSE TRUE  TRUE  TRUE  TRUE  TRUE
as.character(x)
## [1] "0"  "1"  "2"  "3"  "4"  "5"  "6"

x <- c("a", "b", "c")
as.numeric(x)
[1] NA NA NA Warning message:
## NAs introduced by coercion
as.logical(x)
[1] NA NA NA
as.complex(x)
[1] NA NA NA Warning message:
## NAs introduced by coercion
```

## Creating Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (`nrow`, `ncol`)

```
m <- matrix (nrow = 2, ncol = 3)
m
## [,1] [,2] [,3] [1,] NA NA NA [2,] NA NA NA
dim(m)
## [1] 2 3

attributes(m)
$dim
## [1] 2 3
```

Matrices are constructed column-wise, so entries can be thought of starting in the “upper left” corner and running down the columns.

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Matrices can also be created directly from vectors by adding a dimension attribute.

```
m <- 1:10
m
##  [1] 1 2 3 4 5 6 7 8 9 10

dim(m) <- c(2, 5)
m
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

### *cbind-ing and rbind-ing*

Matrices can be created by column-binding or row-binding with `cbind()` and `rbind()`.

```
x <- 1:3
y <- 10:12
cbind (x, y)
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12

rbind (x, y)
##      [,1] [,2] [,3]
## x     1     2     3
## y    10    11    12
```

## **Creating Lists**

Lists are a special type of vector that can contain elements of different classes. Lists are a very important data type in R and you should get to know them well.

```
x <- list(1, "a", TRUE, 1 + 4i)
```

```

x [[1]]
## [1] 1
x[[2]]
## [1] "a"

x[[3]]
## [1] TRUE

x[[4]]
## [1] 1+4i

```

## Creating Factors

Factors are used to represent categorical data. Factors can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label.

- Factors are treated specially by modelling functions like `lm()` and `glm()`
- Using factors with labels is better than using integers because factors are self-describing; having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.

```

x <- factor (c("yes", "yes", "no", "yes", "no"))
x
## [1] yes yes no  yes no
## Levels: no yes

table(x)
## x
## no yes
## 2   3

unclass(x)
## [1] 2 2 1 2 1
## attr(,"levels")
## [1] "no" "yes"

```

The order of the levels can be set using the `levels` argument to `factor()`. This can be important in linear modeling because the first level is used as the baseline level.

```
x <- factor (c("yes", "yes", "no", "yes", "no"),

```

```
levels = c("yes", "no")
x
## [1] yes yes no  yes no
## Levels: yes no
```

## Data Frames

The data frame, or dataframe, is a key data type used in R . It is used to store tabular data. Tabular data accounts for much of what we do in statistics. Of course, not all types of data are tabular, but much data of the data we use becomes a tabular form.

Dataframes are represented as a special type of list, where every element of that list has the same length. So, we can think of each column of the dataframe as an element of the list, and in order to be a table, every column must have the same length. However, each column doesn't have to be the same data type. The first column could be numbers, the second column could be factor, the third column could be integers, and the fourth column could be logicals. It doesn't matter what the different types are. So, unlike matrices, which have to store the same type of object in every single element of the matrix, dataframes can store objects of different classes. And, dataframes also have some special attributes. The first attribute is called a row name. Hence, every row of a dataframe has a name, and this can be useful for annotating the data.

Each row in a dataframe might represent a subject enrolled in a study, and the row names might be the subject ID, for example. However, sometimes the row names are not interesting, and often just indexed with row names of 1, 2, 3, et cetera. Data frames can be created by calling the `read.table()` and the `read.csv()` functions, which we will cover in Chapter 2, *Getting and Cleaning Data*. We can also create a matrix from a dataframe by calling the `data.matrix()` function, but only if the dataframe has the same types of objects, If we try to coerce a dataframe with different data types into a matrix, it's going to force each object to be coerced so that they're all the same. Thus, we may get something that's not exactly expected.

Another important attribute is column names, and these are usually the variable names, like Name, Height, Address, Account Number, and so on. Also, the column names explicitly describe the different data types, i.e., Name is a string, while height is a number.

Dataframes can also be created using the `data.frame()` function.

```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
x
##   foo   bar
## 1   1  TRUE
## 2   2  TRUE
## 3   3 FALSE
## 4   4 FALSE

nrow(x)
## [1] 4

ncol(x)
## [1] 2
```

The `foo` variable is an integer sequence from one to four, and the `bar` variable is a logical vector with two true and two false. When we autoprint the dataframe we see two columns and row names default to 1, 2, 3, 4, because there are four rows. When I call the `nrow` function on `x`, we see that there are four rows, and the `ncol` function, shows us there are two rows.

In summary:

- They are represented as a special type of list where every element of the list has to have the same length
- Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
- Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
- Data frames also have a special attribute called `row.names`
- Data frames are usually created by calling `read.table()` or `read.csv()`
- Can be converted to a matrix by calling `data.matrix()`

## Missing Values

Missing values are denoted by `NA` or `NaN` for undefined mathematical operations.

`is.na()` is used to test objects if they are `NA`

`is.nan()` is used to test for `NAN`

- `NA` values have a class also, so there are integer `NA`, character `NA`, etc.
- A `NaN` value is also `NA` but the converse is not true

```
x <- c(1, 2, NA, 10, 3)
is.na(x)
## [1] FALSE FALSE TRUE FALSE FALSE

is.nan(x)
## [1] FALSE FALSE FALSE FALSE FALSE
x <- c(1, 2, NaN, NA, 4)
is.na(x)
## [1] FALSE FALSE TRUE TRUE FALSE

is.nan(x)
## [1] FALSE FALSE TRUE FALSE FALSE
```

## Subsetting – Basics

There are a number of operators that can be used to extract subsets of R objects.

- `[` always returns an object of the same class as the original; can be used to select more than one element (there is one exception)
- `[]` is used to extract elements of a list or a data frame; it can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame
- `$` is used to extract elements of a list or data frame by name; semantics are similar to that of `[]`.

```
x <- c("a", "b", "c", "c", "d", "a")
x[1] [1] "a"
x[2] [1] "b"
x[1:4]
## [1] "a" "b" "c" "c"

x[x > "a"]
## [1] "b" "c" "c" "d"

u <- x > "a"
u
## [1] FALSE TRUE TRUE TRUE TRUE FALSE
```

```
x[u]  
## [1] "b" "c" "c" "d"
```

## Subsetting – Lists

```
x <- list (foo = 1:4, bar = 0.6)  
x[1]  
## $foo  
## [1] 1 2 3 4  
  
x[[1]]  
## [1] 1 2 3 4  
## x$bar  
## [1] 0.6  
  
x[["bar"]]  
## [1] 0.6  
x["bar"]  
## $bar  
## [1] 0.6  
  
x <- list(foo = 1:4, bar = 0.6, baz = "hello")  
x[c(1, 3)]  
## $foo  
## [1] 1 2 3 4  
## $baz  
## [1] "hello"
```

The `[[` operator can be used with computed indices; `$` can only be used with literal names.

```
x <- list(foo = 1:4, bar = 0.6, baz = "hello")  
name <- "foo"  
x[[name]]    ## computed index for 'foo' [1] 1 2 3 4  
x$name ## element 'name' doesn't exist! NULL  
x$foo  
## [1] 1 2 3 4      ## element 'foo' does exist
```

## Subsetting Nested Elements of a List

The `[[` can take an integer sequence.

```
x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))  
x[[c(1, 3)]]  
## [1] 14  
x[[1]][[3]]
```

```
## [1] 14  
x[[c(2, 1)]]  
## [1] 3.14
```

## Subsetting – Matrices

Matrices can be subsetted in the usual way with (i, j) type indices

```
x <- matrix(1:6, 2, 3)  
x[1, 2]  
## [1] 3  
x[2, 1]  
## [1] 2
```

indices can also be missing.

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

Similarly, subsetting a single column or a single row will give you a vector, not a matrix (by default). By default, when a single element of a matrix is retrieved, it is returned as a vector of length 1 rather than a  $1 \times 1$  matrix. This behavior can be turned off by setting `drop = FALSE`.

```
x <- matrix(1:6, 2, 3)  
x[1, 2]  
## [1] 3  
x[1, 2, drop = FALSE]  
##      [,1]  
## [1,]    3
```

Similarly, subsetting a single column or a single row will give you a vector, not a matrix (by default).

```
x <- matrix(1:6, 2, 3)  
x[1, ] [1] 1 3 5  
x[1, , drop = FALSE]  
##      [,1] [,2] [,3]  
## [1,]    1    3    5
```

## Subsetting - Partial Matching

Partial matching of names is allowed with `[]` and `$`.

```

x <- list(aardvark = 1:5)
x$a
## [1] 1 2 3 4 5

x[["a"]] NULL
x[["a", exact = FALSE]]
## [1] 1 2 3 4 5

```

## Subsetting - Removing Missing Values

A common task is to remove missing values (NAs).

```

x <- c(1, 2, NA, 4, NA, 5)
bad <- is.na(x)
x[!bad]
## [1] 1 2 4 5

```

What if there are multiple things and you want to take the subset with no missing values?

```

x <- c(1, 2, NA, 4, NA, 5)
y <- c("a", "b", NA, "d", NA, "f")
good <- complete.cases(x, y)
good
## [1] TRUE  TRUE FALSE  TRUE FALSE  TRUE

x[good]
## [1] 1 2 4 5
y[good]
## [1] "a" "b" "d" "f"

airquality[1:6, ]
##      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

good <- complete.cases(airquality)
airquality[good, ][1:6, ]
##      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
## 7 23    299   8.6   65    5    7

```

## Missing Values

There is a special type of object for missing values. Missing values in R are denoted by either NA or NAN (Not a Number), which we briefly mentioned. NAN is used for undefined mathematical operations, and NA is used for everything else.

There is a function in R called `is.na`, which is used to test objects to see if they are NA if they are any missing values in the object. There is another function called `is.nan`, which is used to test for NaNs. NA values can have a class, too. So, we can have missing integer values or we can have missing character values and so on.

```

x <- c(1, 2, NA, 10, 3)
is.na(x)
## [1] FALSE FALSE  TRUE FALSE FALSE

is.nan(x)
## [1] FALSE FALSE FALSE FALSE FALSE

x <- c(1, 2, NaN, NA, 4)
is.na(x)
## [1] FALSE FALSE  TRUE  TRUE FALSE

is.nan(x)
## [1] FALSE FALSE  TRUE FALSE FALSE

```

In summary:

- `is.na()` is used to test objects if they are NA
- `is.nan()` is used to test for NaN
- NA values have a class also: integer NAs, character NAs, etc.
- A NaN value is also NA but the converse is not true

NA, so for example, an NAN value, a NAN value, is missing. Is considered to be missing.

## Control Structures

Control structures in R allow you to control the flow of execution of the program, depending on runtime conditions. Common structures are:

- **If, else**: testing a condition
- **For**: execute a loop a fixed number of times
- **While**: execute a loop while a condition is true
- **Repeat**: execute an infinite loop
- **Break**: break the execution of a loop
- **Next**: skip an iteration of a loop
- **Return**: exit a function

### Control Structures – if-else

The first control structure is **if-else**, which allows us to test logic conditions, and to execute R function do act, depending on whether the conditions are true or false. Now, **if** the condition is true then you do something, **else** if the condition false, you do something else. Although that is the typical construct, the **else** part is optional, so we could just have an **if** statement to do something if something is true.

Now we can have variations of the **else** part if we wanted to do something alternatively. We could have an **if-then-else** or an **if-else** in case there are other possibilities. There are a couple of different ways that you can formulate the if-else construct in R.

```
if(<condition>) {  
  ## do something  
} else {  
  ## do something else  
  
}  
if(<condition1>) {  
  ## do something  
} else if(<condition2>) {  
  ## do something different  
} else {  
  ## do something different
```

```
}
```

This is a valid **if-else** structure.

```
x = 2
if(x > 3) {
  y <- 10
} else {
  y <- 0
}
y
```

So is this one.

```
y <- if(x > 3) {
  10
} else {
  0
}
y
```

## Control Structures - for

The **for** loop take an iterator variable and assign it successive values from a sequence or vector. For loops are most commonly used for iterating over the elements of an object (list, vector, etc.)

```
for(i in 1:10) {
print (i)
}
```

This loop takes the **i** variable and in each iteration of the loop gives it values 1, 2, 3, ..., 10, and then exits.

These three loops have the same behavior.

```
x <- c("a", "b", "c", "d")
for(i in 1:4) {
  print(x[i])
}
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```

for(i in seq_along(x)) {
  print(x[i])
}
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"

for(letter in x) {
  print(letter)
}
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"

for(i in 1:4) print (x[i])
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"

```

### *Nested for loops*

for loops can be nested.

```

x <- matrix(1:6, 2, 3)
for(i in seq_len(nrow(x))) {
  for(j in seq_len(ncol(x))) {
    print(x[i, j])
  }
}
## [1] 1
## [1] 3
## [1] 5
## [1] 2
## [1] 4
## [1] 6

```

Be careful with nesting though. Nesting beyond 2–3 levels is often very difficult to read/understand.

## Control Structures - while

While loops begin by testing a condition. If it is true, then they execute the loop body. Once the loop body is executed, the condition is tested again, and so forth.

```
count <- 0
while (count < 10) {
  print(count)
  count <- count + 1
}
```

While loops can potentially result in infinite loops if not written properly. Use with care!

Sometimes there will be more than one condition in the test.

```
z <- 5
while(z >= 3 && z <= 10) {
  print(z)
  coin <- rbinom(1, 1, 0.5)
  if(coin == 1) { ## random walk
    z <- z + 1
  } else {
    z <- z - 1
  }
}

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

Conditions are always evaluated from left to right.

## Control Structures – repeat

Repeat initiates an infinite loop; these are not commonly used in statistical applications but they do have their uses. The only way to exit a repeat loop is to call break.

```
x0 <- 1
tol <- 1e-8
repeat {
  x1 <- (x0-1)
  if(abs(x1 - x0) < tol) {
    break
  } else {
    x0 <- x1
  }
}
x1
## [1] -15624136repeat
```

The loop in the previous slide is a bit dangerous because there's no guarantee it will stop. Better to set a hard limit on the number of iterations (e.g. using a for loop) and then report whether convergence was achieved or not.

## Control Structures – next, return

next is used to skip an iteration of a loop

```
for(i in 1:100) {
  if(i <= 20) {
    ## Skip the first 20 iterations
    next
  }
  ## Do something here
}
```

break is used to exit a loop immediately, regardless of what iteration the loop may be on.

```
for(i in 1:100) {
  print(i)

  if(i > 20) {
    ## Stop loop after 20 iterations
    break
  }
}
```

```
        break  
    }  
}
```

## Control Structures Summary

- Control structures like `if`, `while`, and `for` allow you to control the flow of an R program
- Infinite loops should generally be avoided, even if they are theoretically correct.
- Control structures mentioned here are primarily useful for writing programs; for command-line interactive work, the `*apply` functions are more useful.

### Markdown Note:

We specify the desired output format (including multi format) in the YAML metadata of our markdown code:

```
---
```

```
title: "Sample Document"  
output:  
  html_document:  
    toc: true  
    theme: united  
  word_document:  
    toc: true  
    highlight: zenburn  
---
```

```
---
```



## 2. Getting and Cleaning Data

### Obtaining Data Motivation

This chapter covers the basic ideas behind getting our data ready to actually perform a data analysis on it. This is often considered “data wrangling.” This is a step that is often skipped when taking a class or reading a book in machine learning or statistics.

***Definition 2-1.*** *Data wrangling is the process of cleaning and unifying messy and complicated data sets for easy access and analysis.*

We'll examine this topic using examples from coding (codebooks) and from hospitals (various),

Authors and instructors often assume that the data are already prepared to use, in a nice, neat format. Here, we will cover the process of finding the raw data, getting them in a readable format, cleaning them, and preparing usable subsets. This is so important that we will restate these steps.

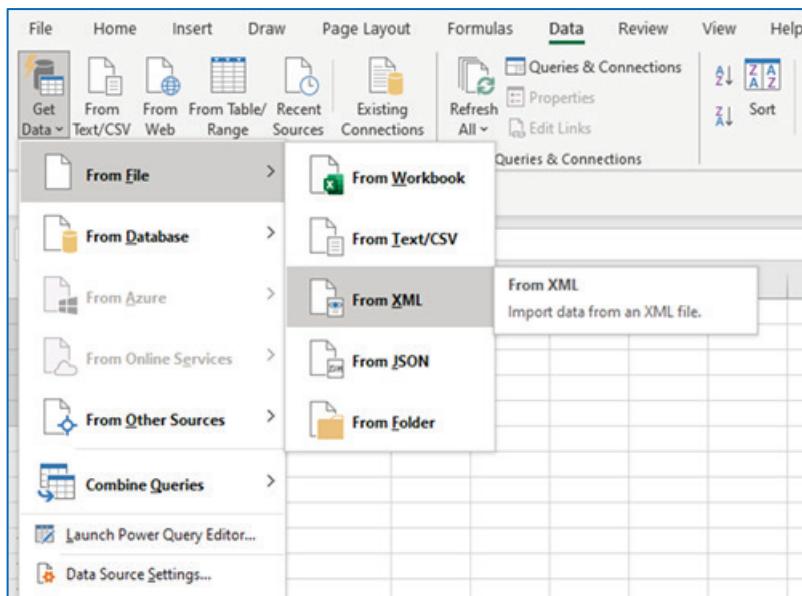
1. Find the raw data. We often do not know what data are available to help answer our analysis questions. This should not surprise us, as we should ask analysis questions apart from available data. In fact, data science may not be the appropriate approach and we sometimes discover that when we can't find appropriate data. However, for our purposes, we will assume that we have a data science problem and that data are available, somewhere in some form: spreadsheet (which is not a database) a database, an image file, a recording or something else. Extracting that data out its source also pertains to tidy data principles.
2. Data can come in many formats and can be consistent of mixed. For instance, JSON is ([JavaScript Object Notation](#)) uses **human-readable text to store and transmit data objects consisting of attribute-value pairs and arrays (or other serializable values)**. XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. DAT is a generic data file that

stores specific information relating to the program that created the file. .STAT is a Weather data summary report file. The data file extension for SAS is sas7bdat. All of these formats are correctible, perhaps with a little work for some. Also, it is possible that a data file has been corrupted but still contains some useful data for extraction.

R can convert JSON using the [rjson](#) package.

```
# Read a JSON file  
# Load the package required to read JSON files.  
library("rjson")  
# Give the input file name to the function.  
result <- fromJSON (file = "E:\\example.json")  
# Print the result.  
print(result)
```

Excel converts XML very well as long as the XML headers are correct using its data tools, as seen in **Figure 2-1**.



**Figure 2-1.** Excel's XML data import function.

Other database formats that are popular are MySQL and MongoDB, which are free. We may also have data from social media, like the Twitter API.

3. Clean the data. We usually start with “dirty” data. Putting the data into a useable format or a tidy data format requires cleaning the data first. We usually have to take raw data that are really complicated and may be formatted for a different use or not at all and ensure that there is data consistency and integrity. Often, the data has been entered by fallible human hands with mixed formatting, leading and trailing spaces, mixture of cases, and so on. Why does this matter? Computers cannot read! They only process numbers, and 1 and <space>”1” are different numbers. For example. Here are dirty data in Excel

ab	=	ab	TRUE	same
ab	=	Ab	FALSE	case
ab	=	ab	FALSE	leading space
ab	=	a b	FALSE	space gap
'ab	=	`ab	FALSE	` instead of '

In R, suppose we want to convert text to numeric. For instance, integers were entered as text.

```
x <- (c("12 ", " 12", "1 2", "1@"))
# Example character vector
x
x_num <- as.numeric(x)
x_num

## [1] "12" "12" "1 2" "1@"
Warning message:
NAs introduced by coercion
## [1] 12 12 NA NA
```

Here, `as.numeric()` caught the leading space in the second input, but it cannot adjudicate the space between characters in the third input, nor can it catch the “@” as “shift-2” in the last input.

Sometimes, we may have to remove leading spaces, check for consistent spelling, convert to lower case or integer format with nothing more than a text editor or Excel, because that may be all we have to work with. With that in mind, we'll include Notepad++, a text editor and more, as one of our tools.

4. Put them into a tidy format. This makes data compatible for use for our data analysis. We'll describe the tidy format later in this chapter. This tasks also involves practical implementation through a range of R packages and we'll be demonstrating a wide range of R packages in this book. This text assumes that you have some basic knowledge about one or more data science tool, even if it is limited to Excel's data analysis tool. Also, you may have been exposed to some programming logic, like IF-THEN-ELSE.
5. It would also be useful if we have taken some exploratory data analysis and also know a little R programming. Exploratory data analysis is relevant, because we might want to plot some of the data while we're data cleaning. Also, knowing reproducible research covers things like creating scripts that will reproduce our analysis, which is an important component of cleaning data. However, these aren't required for us to follow this chapter, and we will learn them soon
6. Now, the Excel file in **Figure 2-1** will have a single observation in each cell, and in every single column will have exactly one variable name. It is neatly organized into something that looks like a matrix, which makes it very easy to import into R. But the real data may look much different than that. For example, this could be a fast cue file (a metadata file which describes how the data is laid out). And in the fast cue file, we might be interested in extracting the sequence information from one line of this file or from each separate line of the file. So, to get the sequence information, we have to parse this raw text file and extract out the bits that we care about. So, one of the steps in getting data is getting into these raw files, figuring out their structure and being able to extract the relevant bits. Another thing that might happen is we might get data that's neatly structured, for example, data from the API of Twitter. Here we get data that's formatted in JSON. The JSON formatted data is neat, organized, and very easy

to distribute, but it's very hard to use for downstream analyses in many programming languages. So, we might want to reorganize the data in such a way that it's easy to analyze. And so that's another component of getting and cleaning data.

7. Finally, you might have situations like where we have instructions, like "take one tablet by mouth and take one-half tablet with grapefruit juice". These are free-text instructions that you might want to get into the free-text and extract it. So, the data might also be somewhere that you have to extract it. If the data is in a structured environment, we might be able to use Structured Query Language (SQL) to extract the data that we are interested in obtaining. We might also want to work with unstructured data, collect some of that data in its raw form, and then process it in such a way that we can use it for analysis. So, the data may be different places, in different formats, with different structures. For example, we might be getting information from Twitter API about specific tweets or specific users. Data might be available from other websites.
8. One of the websites we will be getting data from, contains a lot of datasets that are very interesting and useful for analysis. The website is Open Baltimore, which has free, publicly available data. We can also use the site's API to get downloadable files and clean them, so that we can do downstream analysis on them. So, think about the whole pipeline of going from a raw dataset to a dataset that might be useful for structured analysis. There are many steps to get what we need in most statistics and machine learning classes, where much of the focus is on the data analysis step, with very little emphasis on the data processing pipeline

## Raw and Processed Data

Raw data may be different depending on context or who is talking about it. Since there may not be a common understanding of raw data, let's start by defining data.

**Definition 2-2.** *Data is information that has been translated into a form that is efficient for movement or processing.*

Relative to computers and transmission media, data is information converted into binary digital form. It is acceptable for data to be used as a singular subject or a plural subject. Raw data is a term used to describe data in its most basic digital format. Here are some other important data terms:

- Data are values of qualitative or quantitative variables, belonging to a set of items.
- Set of items: might be the population or the set of objects that you might be interested in.
- Variables: A measurement or characteristic of an item
- Qualitative: country of origin, gender, treatment
- Quantitative: height, weight, blood pressure

When we think about data, we might think about qualitative things like country of origin, gender, treatment, and quantitative variables like height, weight, blood pressure. Many of these measurements are derived from lower-level measurements. For example, blood pressure is measured by calculating two pressure measurement. It's these low-level things we're going to be talking about in raw versus processed data. Hence, the raw data are

- The original source of the data
- Often very hard to use for data analyses
- Data analysis includes processing
- Raw data may only need to be processed once

Raw data are often very hard to use for data analyses because they're complicated or they're hard to parse or they're very hard to analyze. Data analysis includes the processing or the cleaning of the data. Processing of the data might include merging, subsetting, transforming, or we might go into a file and extract out a part of an image. We might go into a file and extract out a little bit of text or we may do a number of other things.

A big component of a data scientist's job is performing those sorts of processing operations. The raw data may only need to be processed once, but regardless of how often you process it, you need to keep a record of all the different things you did. Because it can have a major impact on the data stream analysis. The processed data are

- Data that is ready for analysis.
- Processing of the data might include merging, subsetting, transforming, etc.
- There may be standards for processing
- All steps should be recorded

Preprocessing often ends up being the most important component of a data analysis in terms of effect on the downstream data. So, paying attention to all the steps that we perform is critically important if we are going to be a data scientist who's careful about understanding what's really happening in the entire data processing pipeline. To summarize, getting data is all about taking raw data and turning it into processed data.

## Components of Tidy Data

We have talked about raw data, where we're starting from, the messy file that we're trying to extract some data from. Now we're going to talk about tidy data. There are four things that we should have when you finish going from a raw data set, to a tidy data set:

1. the raw data
2. tidy data set
3. a code book describing each variable, and its value in the tidy data set
4. an explicit and exact recipe that we use to go from steps one to steps two and three

The code book is often called the meta data. So, it's the data that surrounds the data, so-to-speak, and explains what the data is trying to say. We might define each column in our tidy data set as corresponding to one variable, and we might describe things like the units of those variables in the code book. The critical part that is sometimes missing, is an explicit and exact recipe that we use to go from steps one to steps two and three. In other words, we need to report the exact steps that we performed to get from the initial raw data to the finished processed data. We can do that in a variety of different ways, but for the purposes of this book, we'll put our scripts together that we could use to process data. And the scripts will be a

recipe that we can hand off to others and say, “This is how I got from the raw to the tidy data.”

**Raw Data.** It is important to remember that there are different levels of raw data, but for the purposes of our looking a particular data set raw data are the rawest form of the data that we had access to. For example, the raw data might be a strange binary file that is generated by measuring something like blood pressure. It could be an unformatted Excel file with ten worksheets that some company you contracted with sent you, or it could be something that was given to you as an Excel file, or it could be complicated JSON data you get from scraping an API. Or, it can be something like numbers you got from looking through a microscope and counting cells that appeared in the window. So, examples of raw data might include

- strange binary file that whatever your measuring spits out
- an unformatted Excel file with ten worksheets that some company you contracted with sent you
- complicated JSON data you get from scraping an API
- hand-written your numbers you got from looking through a microscope

We know the raw data is in the right format if, and this is a critical:

- we ran no software on the data
- we didn't manipulate any of your numbers in the data
- we move any of the data from the data set
- we didn't do any summarization to the data in any way

So, someone else might give us data that they have performed some preprocessing on, but the when it handed to us, we may consider it raw if we didn't manipulate any of the data set or move any of the data from the data set. Moreover, we have not performed any summarization of the data or explored its structure in any way. This is how we know that the data is in its rawest form. It is the raw data that we start to process of forming a tidy data set.

**Tidy Data.** The tidy data is, on the other hand, is the target or the end goal of the whole data preprocessing task. The idea is, we should have each variable that we've processed be in exactly one column. So,

there should be one variable per column, and each different observation should be in a different row. In other words, if you've measured a particular variable on a large number of people who have been tweeting, say you measure the number of tweets that were posted by a large number of users, then what we would get is the number of tweets in the column and then, for every single user, you would get the number of tweets in a corresponding row. And, there should be one table for every kind of variable.

- each variable that you've measured be in exactly one column
- each different observation should be in a different row
- there should be one table for every kind of variable
- if you have multiple tables, they should include a column in the table that allows them to be linked

For example, if you collect data from Twitter and Facebook, and so forth, we might have one table for each social media. If we have multiple tables, we should ensure that we include a column in the table that allows them to be linked together. This is called a key, and is often an ID like member number, customer number, VIN number and so on.

A tidy data set does not need column headings (variable names), but it's much more useful if they are. It's also much more useful if the variable names are human readable. For example, if you use something like AgeAtDiagnosis for a column name as opposed to AgeDx, which might be a little bit more confusing and harder for people to read.

In general, data should be saved in one file per table. So, it's a habit of some people to save data in multiple Excel spread sheets within a single Excel file, but it is a better idea to save each spreadsheet in a different file. So, some simple rules include:

- include a row at the top of each file for variable names
- make the variable names are human readable, like AgeAtDiagnosis as opposed to AgeDx
- put each spreadsheet in a different file

**The Codebook.** The next component is often a piece that is missing, called the codebook. So, now we have this tidy data set it's very nice and clean and neat and we got that data set by doing a whole bunch of things starting with a raw data set. At the end of the day, we end up with a data set that constitutes information about a bunch of different variables, and we might end up with just the variable names at the top of that file. But we might want to have more information about the variables. One common example is, we might want to know the units of measure. Thus, the column header might be the amount of money that we made this quarter, and we would want to know if that the values of those data were in units of thousands, of millions, or so on. We might also want to know some information about the summary choices that were made. So, it might be the variable measures something about the monthly revenue. We might also include information about the experimental study design that we used. In summary, we might require:

- information about a bunch of different variables
- information about the summary
- then information about the experimental study design that you used

Also, we might want to know something about the way that we collected these data, whether it was just in a database, we extracted it out of other data, or whether you performed an experiment, a randomized trial, or an A-B test or something like that. A common format for this document is a Word or a text file. It would be like a Markdown file. As we've seen, Markdown files are sort of a commonly used format in data science. There should be a section called Study Design, and it should contain a thorough description of how you collected the data. It should state things like how we picked which observations to collect. What did we extract out of the database, what did you exclude, and so on. There also should be a section called Code Book that describes all of the variables and its units.

- a commonly used format in data science
- a section called Study Design, and that has a thorough description of how you collected the data

- a section called Code Book that describes all of the variables and their units

**Instruction List.** Finally, we need the instruction list. So, even if you collected all that information and you made it available in the certain terms of the tidy data, we should be able to go back to the raw data and re-process it, getting the same tidy data set. If that can't happen, then there's something wrong in our data processing pipeline, and so, we want to be able to identify that and fix it. So ideally, this is going to be a computer script that will do this for you. For the purposes of this book, we use R to do it. The input for the script is the, raw data. And so, the output is going to be the processed, tidy data. After we have performed all the processing, and we have an exact recipe that doesn't have to be tweaked or modified by the end user.

- a computer script (in R or Python)
- the input for the script is the, raw data
- the output is going to be the processed, tidy data
- the script has no parameters

We would expect that they would get the same tidy data set out if they put the same raw data set in. In some cases, it's not possible to script every step. For example, not everything we can do to data, you can do in R. I know it's hard to believe that I'm saying that, but it's true. And so, what we might need, in addition to the R scripts, a set of commands written out in a text file. It says take the raw file, run some software, and you're going to run version 3.1.2, and we're going to run it with these specified parameters.

**Step 1:** take the raw file, and you're going to run some software, and you're going to run version 3.1.2, and you're going to run it with these spec, specified parameters

**Step 2:** run the software separately for each sample

**Step 3:** take column three of the `outoutfile.out` for each sample and that is the corresponding row in the output data set

We have to give all that information because if, for example, the version changes, we might get a different answer. And we should be specific with things like, "I ran the software separately for each

sample," so people know exactly how we produced the result. We should go overboard in the amount of detail we provide on how the data got from being raw data to processed data.

## Downloading Files

One way that we might get a data set is somebody might write down the numbers on a piece of paper and mail them to us. But more likely in this era of information technology, we're probably going to be downloading a lot of our data from the Internet. So, now we will discuss how to download files?

The reason why we want the user to download files, as opposed to pointing and clicking, is because the downloading process can be included in a processing script, and we get a more complete picture of how the data were collected and generated.

The first thing to keep in mind is we need to know what directories we're working in. The two main commands when we use with directories in R, are going `getwd()`, and `setwd()`. Now, `getwd()` does exactly what we would expect, it gets the working directory. It tells us what in what directory we are currently working. And `setwd()` sets a different working directory that we might want to move to instead. One way that we can do this is we can use a relative path. For example, if we use `setwd("../")`, we end up with a working directory one level up from where we store our scripts. Remember when we were talking about directory structure, directory is up if it's a super center, if it's above the directory that you're currently in. We can also use absolute paths. For example, we can type `setwd("/Users/jtleek/data/")` and that will move us into that directory. To summarize:

- Get/Set our working directory
- A basic component of working with data is knowing your working directory
- The two main commands are `getwd()` and `setwd()`
- Be aware of relative versus absolute paths
  - Relative – `setwd("./data")`, `setwd("../")`
  - Absolute – `setwd("/Users/jeff/data/")`
- In windows: `setwd ("C:\\\\Users\\\\jeff\\\\Documents")`

An important distinction in Windows is that we have to use backslashes rather than forward slashes. So, this is an example of the way that a path might look in a Windows machine as opposed to a Mac or Linux machine. One thing that we might want to do before we collect a bit of data is to create a directory for that data to fall into. So, `file.exists()`, is a function in R and if we are in a directory and you type `file.exists(directoryName)`, like this, it will look within that directory and see if there is a sub-directory called `directoryName`. And if it does exist, it will return TRUE, and if it doesn't exist it will return FALSE. Then, `dir.create()`, will create a directory if it doesn't exist.

`file.exists("directoryName")()` will check to see if the directory exists

`dir.create("directoryName")()` will create a directory if it does not exist

Here's a little bit of code that checks to see if there's a data directory.

```
if (!file.exists("data")) {  
    dir.create("data")  
}
```

The code says, if the data sub-directory exists, then it will return a TRUE. So, what will occur is it will be FALSE, and nothing will happen. If the data directory doesn't exist, it will return FALSE, and this exclamation point will negate it, so it will be TRUE. Then, it will create a directory called "data". Hence, the primary way we get data from the internet, is with the `download.file` function, and so this function downloads the file from the internet. We can do this by hand of course, i.e., go to the website that we're interested in and click and press download. But if we do it by using `download.file`, it will improve the reproducibility. So, it is easier to keep track of everything that we've done if we include the download process as part of our script. To summarize, getting data from the internet: `download.file()`

- the `download.file` function
- downloads the file from the internet

- it'll actually improve the reproducibility
- useful for downloading tab to limited, CSV files, Excel files, etc.

The important parameters are the `URL`, because that is where we will be getting the data. The `destfile` is the destination file where that data is going to go, and then the method. So, we'll talk a little bit about why the method needs to be specified, particularly when dealing with HTTPS. The `download()` function is sort of agnostic to the file type that you might be downloading.

As an example, we're going to be using the Baltimore fixed camera data. These are speed cameras that are set up around Baltimore, which detects speeding and results in citation and in a traffic ticket. If look at the Export button (**Figure 2**, Pointer 1) in on this website, you actually have different file formats to choose. For example, we might go down to CSV, which is one of the types of files that you would see over here on the left-hand side. However, if we look at the URL, we see file named "`explore?location=39.297050%2C-76.621400%2C12.80`". But, recall that the function `download.file()` agnostic in terms of file types, so we trust that the format is .csv.

The screenshot shows the Open Baltimore website interface. At the top, there's a navigation bar with links for Home, Data Catalog, Dashboards, Training & Resources, Data, Blog, Mayor's Office, What Works Cities, and Contact Us. Below the navigation is a search bar with placeholder text 'Search datasets'. The main content area displays a dataset titled 'Fixed Speed Cameras' from 'Baltimore City'. The dataset has 32 attributes, including Name, Type, and Action. A red arrow points to the 'Download' button for the 'OBJECTID' attribute. The 'Download Options' dropdown menu is also visible.

**Figure 2-2.** Open Baltimore, the Baltimore City public data site, showing the dataset for “Fixed Speed Cameras.”

### Downloading a File from the Web

```
fileUrl <-  
"https://data.baltimorecity.gov/datasets/baltimore::fixed-  
speed-cameras/explore?location=39.297050%2C-  
76.621400%2C12.80"  
download.file(fileUrl, destfile = "./data/cameras.csv",  
method= "curl")
```

%	Total	%	Received	%	Average Dload	Speed Upload	Current Speed
100	61266	100	61266	0	99k	0	100k

Hence, we save the file into a variable called `fileUrl`. and we pass a download file to our data directory in a file we name "`cameras.csv`". We're using .csv because that's the extension that's used for comma separated files. We also use the `curl` method because

the website is an HTTPS. Then if we list the files in this data directory, we see that we now have a file named cameras.csv.

```
list.files("./data")
## [1] "cameras.csv"    "community.csv" "EDU.csv"
"GDP.csv"          "image.jpeg"
dateDownloaded <- date()
dateDownloaded
## [1] "Wed Feb 02 10:06:43 2022"
```

**Markdown Note.** Knitr is an engine for dynamic report generation with R. It is a package in the statistical programming language R that enables integration of R code into LaTeX, LyX, HTML, Markdown, AsciiDoc, and reStructuredText documents.

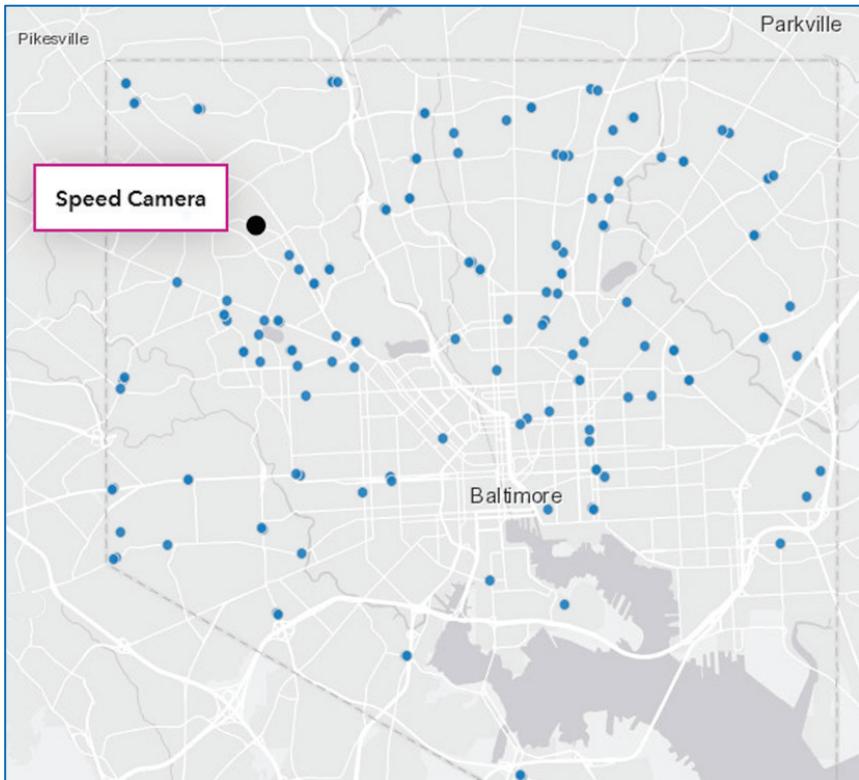


Figure 2-3. Map of Baltimore City with locations of Speed cameras, using the Create Map function from the website

## Notes about `download.file()`

- If the url starts with http, you can use `download.file ()`
- If the url starts with https, on Windows you may be ok
- If the url starts with https, on Mac you may need to set `method="curl"`
- If the file is big, this might take a while
- Be sure to record when you downloaded.

## Loading flat files - `read.table()`

This next section is about reading local or flat files. Prime examples of flat files include text files, hyper delimited text files, comma delimited text files and so on. If you have already familiar with R, you may be able to skip this section, because you've already seen how to load flat files into R. Again, we're going back to the Baltimore fixed camera data. We are going to download the data, check to see if the data directory exists. If it does not, we will create it, and then we will download the data with `download.file()`. And with the data downloaded from the website, it will be in our working directory on our computer.

```
If (!file.exists("data")) {  
    Dir.create("data")  
}  
fileUrl <-  
"https://data.baltimorecity.gov/datasets/baltimore::fixed-  
speed-cameras/explore?location=39.297050%2C-  
76.621400%2C12.80"  
download.file(fileUrl, destfile = "./data/cameras.csv",  
            method= "curl")  
dateDownloaded <- date()
```

So, the first thing we might use to load the data is with the `read.table()` function. So, `read.table()` is the most common function for loading data, because it's the most robust. It is flexible and requires just a few more parameters than we would pass to other loading functions, but it can be a little slow. So, there are actually faster ways if we need to scan through files to find specific elements (we might talk about those later)., `read.table()` reads the data into RAM, so it reads the data into memory. If we have a really big dataset,

`read.table()` may cause problems unless we read it in chunks. In summary, `read.table()`:

- is the main function for reading data into R
- is flexible and robust but requires more parameters
- reads the data into RAM - big data can cause problems
- has parameters `file`, `header`, `sep`, `row.names`, `nrows`
- related: `read.csv()`, `read.csv2()`

Thus, `read.table()` is probably not the best way to read large datasets in general into R. The important parameters here are what file you want to read, whether it has a header, what separates elements, whether it has row names and how many rows we want to read.

For the camera data, we write:

```
cameraData <- read.table("./data/cameras.csv", sep =",",  
header = TRUE )
```

The related function `read.csv()` is written as:

```
cameraData <- read.csv("./data/cameras.csv")
```

It will also take the parameter `header = TRUE`.

Some other important parameters include:

- `quote` - you can tell R whether there are any quoted values  
`quote=""` means no quotes.
- `na.strings` - set the character that represents a missing value.
- `nrows` - how many rows to read of the file (e.g. `nrows=10` reads 10 lines).
- `skip` - number of lines to skip before starting to read

## Reading Excel Files

Some data scientists show a certain snobbery about Excel files. But Excel files are still probably the most widely used format for sharing data. I think the reason is because a lot of people are used to using spreadsheets, so whether you're working in business applications or in science, people know how to use spreadsheets to collect and put

data in, and share it with each other. It's true now, even more than ever, with things like Google spreadsheets, which allow people to share them collaboratively over the internet. The problem for us is that when we're analyzing data with a scripting language like R we need to be able to extract the data out of those files so that we can perform downstream analyses and processing on them.

Now, we go back to the Baltimore camera data, using `download.file()` and recalling that it is agnostic to the file type so it just downloads the file and puts it into this file name that we state.

```
fileUrl <-  
"https://data.baltimorecity.gov/datasets/baltimore::fixed-  
speed-cameras/explore?location=39.297050%2C-  
76.621400%2C12.80"  
download.file(fileUrl, destfile = "./data/cameras.xlsx",  
    method= "curl")  
dateDownloaded <- date()
```

The R library that is useful for this is the `xlsx` package. So we write:

```
library(xlsx)  
cameraData <- read.xlsx("./data/cameras.xlsx", sheetname=  
"Fixed_Speed_Cameras", header=TRUE)
```

There is another package, `XLConnect` that can be used if you would like to. It might provide a little bit more flexible.

## Reading XML

XML is Extensible Markup Language. It is frequently used to score structured data. It's particularly widely used in internet applications. So, we'll see it a lot when we're doing things like web scraping or trying to get data from an internet API or trying to download data from an open government website. So, extracting HTML, XML is actually the basis for most of the web-scraping that we'll see. There are two components to an XML file. There's the markup. That's the labels that give the text structure. Thus, you can imagine if you just started typing, you would end up with sort of an unstructured text file.

## Tags, elements and attributes

- Tags correspond to general labels
  - Start tags <section>
  - End tags </section>
  - Empty tags <line-break />
- Elements are specific examples of tags
  - <Greeting> Hello, world </Greeting>
- Attributes are components of the label
  - 
  - <step number="3"> Connect A to B. </step>

<http://en.wikipedia.org/wiki/XML>

The markup is the way that you add labels so the file is structured. And then the content is the text that you enter in between the labels that give the structure to the text. Now, tags correspond to the labels, and the labels apply to particular parts of the text so that it will be structured. There are start tags that start with an open caret on one side and then they have phrase and then a closed caret on the other side. So, we'll have a tag at the beginning of a certain part of the text, and that will start a section.

### *Reading data from XML {XML package}*

```
library("XML")
fileURL <-
"https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fre-
staurants.xml"
doc <- XML::xmlTreeParse(sub("s", "", fileURL),
useInternal = TRUE)
```

### *Directly access parts of the XML document*

```
rootNode[[1]][[1]][[1]]
# <name>410</name>
rootNode[[1]][[1]][[2]]
# <zipcode>21206</zipcode>
rootNode[[1]][[1]][[3]]
# <neighborhood>Frankford</neighborhood>
```

```
rootNode[[1]][[1]][[4]]  
# <councildistrict>2</councildistrict>  
rootNode[[1]][[1]][[5]]  
# <policedistrict>NORTHEASTERN</policedistrict>
```

### *Programmatically extract parts of the file*

```
rootNode <- XML::xmlRoot(doc)  
xmlSApply (rootNode[[1]][[1]],xmlValue)  
  
# name zipcode neighborhood councildistrict policedistrict  
# "410" "21206" "Frankford" "2" "NORTHEASTERN"
```

## Reading JSON

JSON is another file format. It is similar to XML in the sense that it's structured, and it's also commonly used on the internet. JSON is short for Javascript Object Navigation, and it is lightweight for data storage. Consequently, it is very common in application programming interface, which is a way we can programmatically detect access to data for companies like Twitter or Facebook through URLs. The data are stored as Numbers, Strings, Booleans, Arrays, or Objects.

- Javascript Object Notation
- Lightweight data storage
- Common format for data from application programming interfaces (APIs) Similar structure to XML but different syntax/format
- Data stored as
  - Numbers (double)
  - Strings (double quoted)
  - Boolean ( true or false)
  - Array (ordered, comma separated enclosed in square brackets [ ])
  - Object (unordered, comma separated collection of key:value pairs in curly brackets { })

### *Reading data from JSON {jsonlite package}*

```
library(jsonlite)  
jsonData <-
```

```
fromJSON("https://api.github.com/users/jtleek/repos")
names(jsonData)
```

### Nested objects in JSON

```
names(jsonData$owner)
```

```
[1] "login"                  "id"
[3] "node_id"                "avatar_url"
[5] "gravatar_id"            "url"
[7] "html_url"                "followers_url"
[9] "following_url"          "gists_url"
[11] "starred_url"            "subscriptions_url"
[13] "organizations_url"      "repos URL"
[15] "events_url"              "received_events_url"
[17] "type"                   "site_admin"
```

```
jsonData$owner$login
```

```
[1] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
[7] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
[13] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
[19] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
[25] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
```

## Reading From APIs

APIs are application programming interfaces. For example, most social media companies like Twitter or Facebook will have an API, where we can download data. We can get data about which users are tweeting or what they're tweeting about. Where we can get information about what people are posting on Facebook. We can usually get these with GET requests with specific URLs as the arguments.

We will use the [HTTR](#) package to get data from Facebook and Twitter, but we will not actually do this now. To do this, the first thing that we would need an account. In this instance, we don't mean an account like a user account. We have to create an account with the API or with the development team out of each particular organization. For example, we would go to Twitter at <https://developer.twitter.com/en/products/twitter-api> and create an application. In this case, we would create an application. However,

we are not going to do that now, but will save the process for our discussion in Chapter 10, Creating Data Products.

For now, we will tap into an API I already have for a translation project. The project started because I was translating sentiment from Urdu. Urdu is an Indo-Aryan language spoken chiefly in South Asia. It is the official national language and lingua franca of Pakistan. If we want to download a Urdu dataset using the API, we would proceed as follows:

```
lang1 <-  
"https://console.cloud.google.com/translation/locations/us  
-  
central1/datasets/TRL174774520060575744/sentences?project=  
my-translation-project-266317"  
  
download.file(lang1, destfile = paste0(getwd(),  
'/lang10.csv'), method = "curl")  
  
## trying URL ##  
'https://console.cloud.google.com/translation/locations/us  
-##  
central1/datasets/TRL109753800440414208/sentences?project=  
## my-translation-project-266317'  
  
## Content type 'text/html; charset=UTF-8' length unknown  
  
## downloaded 99 KB  
  
urdu <- read.csv("./lang10.csv")  
head(urdu)  
  
X..DOCTYPE.html.  
1 <html lang=en>  
2 <head>  
3 <meta charset=utf-8>  
4 <meta content=width=300, initial-scale=1 name=viewport>  
5 <meta name=description content=Google Cloud Platform  
lets you build, deploy, and scale applications, websites,  
and services on the same infrastructure as Google.>  
6 <meta name=google-site-verification  
content=LrdTUW9psUAMbh4Ia074-BPEVmcpBxF6Gwf0MSgQXZs>
```

The dataset we down loaded has 2000 records. A record is comprised of an Urdu text entry (sentence) and sentiment (positive, neutral, or negative). In the API, [TRL109753800440414208](#) is the code for the dataset called `dataset_roman`. The first six row are metadata.

## The `data.table` Package

This section is about the `data.table` package which is an often faster more memory efficient version of the data frames, which we commonly use when your analyzing data. The `data.table` inherits from a data frame so all functions that accept that `data.frame` should work on `data.table`. It's written in C so it can be much faster than some of the functions that are done with `data.frame`.

Create data tables just like data frames

```
library(data.table)
DF = data.frame(x=rnorm(9) ,y=rep(c( "a", "b", "c")
,each=3) ,z=rnorm(9))
head(DF,3)

      x   y       z
1 -0.9999011 a  0.1469773
2 -0.3992725 a -0.1836456
3  1.0039080 a  1.1770095

DT = data.table (x=rnorm(9) ,y=rep(c( "a", "b", "c")
,each=3) ,z=rnorm(9))
head(DT,3)

      x   y       z
1: -0.3652361 a  0.7359411
2: -0.1637383 a  0.4647755
3: -0.4978154 a -0.2189378
```

*See all the data tables in memory*

```
tables()
```

	NAME	NROW	NCOL	MB	COLS	KEY
1:	DT	9	3	0		
2:	housing	6,496	188	5		
3:	xmlZipcodeDT	1,327	1	0		

```

1:                               x,y,z
2: RT,SERIALNO,DIVISION,PUMA,REGION,ST, ...
3:                               zipcode
Total: 5MB

```

### *Subsetting rows*

```

DT[2,]
      x   y       z
1: -0.1637383 a 0.4647755

DT [DT$y=="a",]
      x   y       z
1: -0.3652361 a 0.7359411
2: -0.1637383 a 0.4647755
3: -0.4978154 a -0.2189378

```

### *Subsetting columns*

```

DT[,c(2,3)]
      y       z
1: a -0.2818528
2: a -1.8083496
3: a 0.1554229
4: b -1.1141081
5: b -0.5586712
6: b 1.1237270
7: c 0.7219369
8: c 1.1713646
9: c 0.1105371

```

### *Calculating values for variables with expressions*

```

DT[,list(mean(x) ,sum(z) )]
      V1       V2
1: -0.0532023 0.8652344
DT [ ,table(y)]
y
a b c
3 3 3

```

### *Adding new columns*

```
DT[,w:=z^2]
```

## *Adding new columns*

```
DT2 <- DT
DT[, y:= 2]
head(DT, n = 3)
      x y          z          w
1: 0.1540239 2 -0.7092386 0.5030193
2: -0.1311806 2 -0.7394954 0.5468535
3: -0.3182240 2  0.8634417 0.7455315
head(DT2, n = 3)
      x y          z          w
1: 0.1540239 2 -0.7092386 0.5030193
2: -0.1311806 2 -0.7394954 0.5468535
3: -0.3182240 2  0.8634417 0.7455315
```

## *Multiple operations*

```
DT[,m:= {tmp <- (x+z); log2(tmp+5)}]
```

## *plyr like operations*

```
DT[, a:= x>0]
DT [, b:= mean(x+w), by=a]
```

## *Special variables*

```
.N An integer, length 1, containing the number
set.seed(123);
DT <- data.table(x=sample(letters[1:3], 1E5, TRUE))
DT[, . N, by=x]
      x      N
1: c 33294
2: b 33305
3: a 33401
```

## *Keys*

```
DT <- data.table(x=rep(c( "a", "b", "c"),each=100),
y=rnorm(300))
setkey(DT, x)
head(DT [ 'a'])

      x          y
1: a  0.8863126
```

```
2: a 2.8285813
3: a 2.0314543
4: a 1.9067541
5: a 0.2149083
6: a -0.8627341
```

## Joins

```
DT1 <- data.table (x=c('a', 'a', 'b', 'dt1'), y = 1:4)
DT2 <- data.table (x=c('a', 'b', 'dt2'), z = 5:7)
setkey(DT1, x);
setkey(DT2, x)
merge(DT1, DT2)

  x y z
1: a 1 5
2: a 2 5
3: b 3 6
```

## Fast Reading

```
big_df <- data.frame(x=rnorm(1E6), y=rnorm(1E6))
file <- tempfile()
write.table(big_df, file=file, row.names = FALSE,
col.names = TRUE, sep="\t", quote = FALSE)
system.time(fread(file))

  user  system elapsed
  0.15    0.03   0.27
system.time(read.table(file, header=TRUE, sep="\t"))
  user  system elapsed
  1.84    0.07   1.92
```

## Example 2-1. Tidy Data Codebook

A *codebook* describes the contents, structure, and layout of a data collection. A well-documented *codebook* "contains information intended to be complete and self-contained. Source Code: <https://github.com/stricje1/analysis.R>.

## Solution to Example 1

For this example, we'll use a template called `analysis.R` `Codebook`. First, we'll load required libraries (we'll install them if

necessary. For data retrieval, decompressing and naming assignments, we'll need the `dplyr` package.

```
library(dplyr)
```

### *Step 1. Download and Load the Data*

Here, we load the download and data into R. We also need to check the working directory and changing it if necessary.

```
getwd()  
  
##[1]"C:/Users/jeff/Documents/R/Getting_and_Cleaning_Data"  
  
filename <- "Coursera_DS3_Final.zip"  
# Checking if archive already exists.  
if (!file.exists(filename)){  
  fileURL <- "https://d396qusza40orc.cloudfront.net/getdat  
a%2Fprojectfiles%2FUCI%20HAR%20Dataset.zip"  
  download.file(fileURL, filename, method="curl")  
}  
# Checking if folder exists  
if (!file.exists("UCI HAR Dataset")) {  
  unzip (filename)  
}
```

### *STEP 2. Merging Training and Test Set into one Data Set*

- Assign each data to variables
- Merges feature sets
- merges activities code sets
- merges subject sets
- merges all prior sets

The features selected for this database come from the accelerometer and gyroscope 3-axial raw signals tAcc-XYZ and tGyro-XYZ.

```
features <- read.table ("UCI HAR Dataset/features.txt", co  
l.names = c("n", "functions"))
```

**Activities List.** List of activities performed when the corresponding measurements were taken and its codes (labels)

```
activities <- read.table ("UCI HAR Dataset/activity_labels.txt", col.names = c("code", "activity"))
```

**Volunteer Test Subject.** Contains test data of 9/30 volunteer test subjects being observed

```
subject_test <- read.table ("UCI HAR Dataset/test/subject_test.txt", col.names = "subject")
```

**Recorder Feature Data.** Contains recorded features testing data

```
x_test <- read.table ("UCI HAR Dataset/test/X_test.txt", col.names = features$functions)
```

**Activities Codes.** Contains testing data of activities'code labels

```
y_test <- read.table ("UCI HAR Dataset/test/y_test.txt", col.names = "code")
```

**Subject Training Set.** Contains train data of 21/30 volunteer subjects being observed

```
subject_train <- read.table ("UCI HAR Dataset/train/subject_train.txt", col.names = "subject")
```

**Feature Training Set.** Contains recorded features train data

```
x_train <- read.table ("UCI HAR Dataset/train/X_train.txt", col.names = features$functions)
```

**Activities Code Labels.** Contains train data of activities' code labels

```
y_train <- read.table ("UCI HAR Dataset/train/y_train.txt", col.names = "code")
```

### *STEP 3. Set Merging*

Merges the training and the test sets (x) to create one data set

```
X <- rbind(x_train, x_test)
```

Merges the train data of activities'code (y) labels

```
Y <- rbind(y_train, y_test)
```

The set Subject is created by merging subject\_train and subject\_test using rbind() function

```
Subject <- rbind(subject_train, subject_test)
```

The set Merged\_Data is created by merging Subject, Y and X using cbind() function

```
Merged_Data <- cbind(Subject, Y, X)
```

#### *STEP 4. Measure Extraction*

Extracts only the measurements on the mean and standard deviation for each measurement

```
TidyData <- Merged_Data %>% select(subject, code, contains("mean"), contains("std"))
```

#### *STEP 5. Activity Naming*

Uses descriptive activity names to name the activities in the data set

```
TidyData$code <- activities[TidyData$code, 2]
```

#### *STEP 6. Code Labeling Process*

Appropriately labels the data set with descriptive variable names \*

The code column in TidyData renamed into activities

```
names(TidyData)[2] = "activity"
```

All Acc in column's name replaced by Accelerometer

```
names(TidyData) <- gsub("Acc", "Accelerometer", names(TidyData))
```

All Gyro in column's name replaced by Gyroscope

```
names(TidyData) <- gsub("Gyro", "Gyroscope", names(TidyData))
```

All BodyBody in column's name replaced by Body

```
names(TidyData) <- gsub("BodyBody", "Body", names(TidyData))
```

All Mag in column's name replaced by Magnitude

```
names(TidyData) <- gsub("Mag", "Magnitude", names(TidyData))
```

All start with character t in column's name replaced by Time

```
names(TidyData) <- gsub("^t", "Time", names(TidyData))
```

All start with character f in column's name replaced by 'Frequency'

```
names(TidyData) <- gsub("^f", "Frequency", names(TidyData))
```

All `tbody` in column's name replaced by `TimeBody`  
`names(TidyData) <- gsub("tbody", "TimeBody", names(TidyData))`

All `-mean()` in column's name replaced by `Mean`  
`names(TidyData) <- gsub("-mean()", "Mean", names(TidyData), ignore.case = TRUE)`

All `-std()` in column's name replaced by `Stand_Dev`  
`names(TidyData) <- gsub("-std()", "STD", names(TidyData), ignore.case = TRUE)`

All `-freq()` in column's name replaced by `Frequency`  
`names(TidyData) <- gsub("-freq()", "Frequency", names(TidyData), ignore.case = TRUE)`

### *STEP 7. Tidy Data Creation & Final Dataset*

From the data set in step 4, creates a second, independent tidy data set with the average of each variable for each activity and each subject

The set `FinalData` (180 rows, 88 columns) is created by summarizing `TidyData` taking the means of each variable for each activity and each subject, after grouped by subject and activity.

```
FinalData <- TidyData %>%
  group_by(subject, activity)  %>%
  summarise_all(funs(mean))

## Warning: `fun` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
```

```
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

### *STEP 8. Final File Checking*

```
## str(FinalData) not printed
```

### *Step 9. File Export FinalData into FinalData.txt file.*

```
write.table(FinalData, "FinalData.txt", row.names=FALSE)
```

### *STEP 10. Print Final Data Set*

```
FinalData
```

```
## # A tibble: 180 x 88
## # Groups:   subject [30]
##   subject activity   TimeBodyAccelerome~ TimeBodyAccel
erome~ TimeBodyAccelerom~
##       <int> <chr>      <dbl>      <dbl>      <dbl>
## 1     1 LAYING      0.222    -0.0405    -0.113
## 2     1 SITTING     0.261    -0.00131   -0.105
## 3     1 STANDING    0.279    -0.0161    -0.111
## 4     1 WALKING     0.277    -0.0174    -0.111
## 5     1 WALKING_D~  0.289    -0.00992   -0.108
## 6     1 WALKING_U~  0.255    -0.02      -0.0973
## 7     2 LAYING      0.281    -0.0182    -0.107
## 8     2 SITTING     0.277    -0.0157    -0.109
## 9     2 STANDING    0.278    -0.0184    -0.106
## 10    2 WALKING     0.276    -0.0186    -0.106
## ... with 170 more rows, and 83 more variables:
##   TimeGravityAccelerometer.mean...X <dbl>,
##   TimeGravityAccelerometer.mean...Y <dbl>,
##   TimeGravityAccelerometer.mean...Z <dbl>,
##   TimeBodyAccelerometerJerk.mean...X <dbl>,
##   TimeBodyAccelerometerJerk.mean...Y <dbl>,
##   TimeBodyAccelerometerJerk.mean...Z <dbl>, TimeBodyGyr
##   oscope.mean...X <dbl>,
##   TimeBodyGyroscope.mean...Y <dbl>, TimeBodyGyroscope.m
##   ean...Z <dbl>,
##   TimeBodyGyroscopeJerk.mean...X <dbl>, TimeBodyGyrosco
##   peJerk.mean...Y <dbl>,
##   TimeBodyGyroscopeJerk.mean...Z <dbl>,
##   TimeBodyAccelerometerMagnitude.mean.. <dbl>,
```

```

## TimeGravityAccelerometerMagnitude.mean.. <dbl>,
## TimeBodyAccelerometerJerkMagnitude.mean.. <dbl>,
## TimeBodyGyroscopeMagnitude.mean.. <dbl>,
## TimeBodyGyroscopeJerkMagnitude.mean.. <dbl>,
## FrequencyBodyAccelerometer.mean...X <dbl>,
## FrequencyBodyAccelerometer.mean...Y <dbl>,
## FrequencyBodyAccelerometer.mean...Z <dbl>,
## FrequencyBodyAccelerometer.meanFreq...X <dbl>,
## FrequencyBodyAccelerometer.meanFreq...Y <dbl>,
## FrequencyBodyAccelerometer.meanFreq...Z <dbl>,
## FrequencyBodyAccelerometerJerk.mean...X <dbl>,
## FrequencyBodyAccelerometerJerk.mean...Y <dbl>,
## FrequencyBodyAccelerometerJerk.mean...Z <dbl>,
## FrequencyBodyAccelerometerJerk.meanFreq...X <dbl>,
## FrequencyBodyAccelerometerJerk.meanFreq...Y <dbl>,
## FrequencyBodyAccelerometerJerk.meanFreq...Z <dbl>,
## FrequencyBodyGyroscope.mean...X <dbl>,
## FrequencyBodyGyroscope.mean...Y <dbl>,
## FrequencyBodyGyroscope.mean...Z <dbl>,
## FrequencyBodyGyroscope.meanFreq...X <dbl>,
## FrequencyBodyGyroscope.meanFreq...Y <dbl>,
## FrequencyBodyGyroscope.meanFreq...Z <dbl>,
## FrequencyBodyAccelerometerMagnitude.mean.. <dbl>,
## FrequencyBodyAccelerometerMagnitude.meanFreq.. <dbl>,
## FrequencyBodyAccelerometerJerkMagnitude.mean.. <dbl>,
## FrequencyBodyAccelerometerJerkMagnitude.meanFreq..<dbl>
## FrequencyBodyGyroscopeMagnitude.mean.. <dbl>,
## FrequencyBodyGyroscopeMagnitude.meanFreq... <dbl>,
## FrequencyBodyGyroscopeJerkMagnitude.mean.. <dbl>,
## FrequencyBodyGyroscopeJerkMagnitude.meanFreq... <dbl>,
## angle.TimeBodyAccelerometerMean.gravity. <dbl>,
## angle.TimeBodyAccelerometerJerkMean..gravityMean.<dbl>,
## angle.TimeBodyGyroscopeMean.gravityMean. <dbl>,
## angle.TimeBodyGyroscopeJerkMean.gravityMean. <dbl>,
## angle.X.gravityMean.<dbl>,angle.Y.gravityMean. <dbl>,
## angle.Z.gravityMean. <dbl>, TimeBodyAccelerometer.std..
.X <dbl>,
## TimeBodyAccelerometer.std...Y <dbl>, TimeBodyAccelerome
ter.std...Z <dbl>,
## TimeGravityAccelerometer.std...X <dbl>,
## TimeGravityAccelerometer.std...Y <dbl>,
## TimeGravityAccelerometer.std...Z <dbl>,

```

```

## TimeBodyAccelerometerJerk.std...X <dbl>,
## TimeBodyAccelerometerJerk.std...Y <dbl>,
## TimeBodyAccelerometerJerk.std...Z <dbl>, TimeBodyGyroscopicstd...X <dbl>,
## TimeBodyGyroscope.std...Y <dbl>, TimeBodyGyroscope.std...Z <dbl>,
## TimeBodyGyroscopeJerk.std...X <dbl>, TimeBodyGyroscopeJerk.std...Y <dbl>,
## TimeBodyGyroscopeJerk.std...Z <dbl>,
## TimeBodyAccelerometerMagnitude.std.. <dbl>,
## TimeGravityAccelerometerMagnitude.std.. <dbl>,
## TimeBodyAccelerometerJerkMagnitude.std.. <dbl>,
## TimeBodyGyroscopeMagnitude.std.. <dbl>,
## TimeBodyGyroscopeJerkMagnitude.std.. <dbl>,
## FrequencyBodyAccelerometer.std...X <dbl>,
## FrequencyBodyAccelerometer.std...Y <dbl>,
## FrequencyBodyAccelerometer.std...Z <dbl>,
## FrequencyBodyAccelerometerJerk.std...X <dbl>,
## FrequencyBodyAccelerometerJerk.std...Y <dbl>,
## FrequencyBodyAccelerometerJerk.std...Z <dbl>,
## FrequencyBodyGyroscope.std...X <dbl>, FrequencyBodyGyroscope.std...Y <dbl>,
## FrequencyBodyGyroscope.std...Z <dbl>,
## FrequencyBodyAccelerometerMagnitude.std.. <dbl>,
## Freque https://www.coursera.org/learn/data-science-project/peer/EI114/final-project-submission/review/oEQ3UYHtEeypAw6khCQ1MwnccyBodyAccelerometerJerkMagnitude.std.. <dbl>,
## FrequencyBodyGyroscopeMagnitude.std.. <dbl>,
## FrequencyBodyGyroscopeJerkMagnitude.std.. <dbl>

```

## Example 2-2. Hospital Rankings

This is our scenario for example 2 through 4. U.S. Department of Health and Human Services administers the website Hospital Compare web site

(<http://hospitalcompare.hhs.gov>).

The purpose of the web site is to provide data and information regarding the quality of care at over 4,000 Medicare-certified hospitals in the U.S. AS such, the datasets for this assignment covers all major U.S. hospitals.

The source code: [https://github.com/stricje1/hospital\\_rankings](https://github.com/stricje1/hospital_rankings).

### ***Read the downloaded data from Hospital Compare***

```
library (readr)
getwd ()
setwd ("C:/Users/jeff/Documents/R_Programming/")
path <- getwd()
setwd("C:/Users/jeff/Documents/R_Programming/")
path <- "C:/Users/jeff/Documents/R_Programming/"
outcome <- read.csv("outcome-of-care-measures.csv", colClasses = "character")
```

### ***Explore the Hospital Care Data***

We'll use the `names()` function to get the names of the outcome object.

```
# head(outcome) # Not helpful in the case
names(outcome)

## [1] "Provider.Number"
## [2] "Hospital.Name"
## [3] "Address.1"
## [4] "Address.2"
## [5] "Address.3"
## [6] "City"
## [7] "State"
## [8] "ZIP.Code"
## [9] "County.Name"
## [10] "Phone.Number"
## [11] "Hospital.30.Day.Death..Mortality..Rates.from.Heart.Attack"
...
## [42] "Comparison.to.U.S..Rate...Hospital.30.Day.Readmission.Rates.from.Pneumonia"
## [43] "Lower.Readmission.Estimate...Hospital.30.Day.Readmission.Rates.from.Pneumonia"
## [44] "Upper.Readmission.Estimate...Hospital.30.Day.Readmission.Rates.from.Pneumonia"
## [45] "Number.of.Patients...Hospital.30.Day.Readmission.Rates.from.Pneumonia"
## [46] "Footnote...Hospital.30.Day.Readmission.Rates.from.Pneumonia"
```

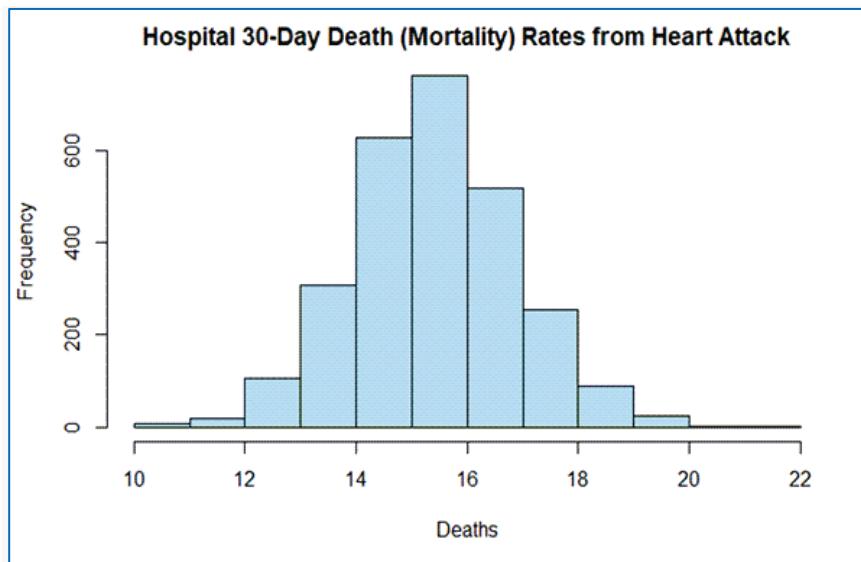
Looking ahead to Example 2, we need to know what column in the dataset holds the data for 30-day mortality rates for heart attack. This data appears in the column named (column 11):

```
"Hospital.30.Day.Death..Mortality.Rates.from.Heart.Attack"
```

### **30-Day Mortality Rates for Heart Attacks**

In this example, our task is to plot the 30-day mortality rates for heart attack patients at U.S. major hospitals (see **Figure 2**). From our explorations so far, we determined that the data appears in Column 11.

```
outcome[, 11] <- as.numeric(outcome[, 11])
# Column 11 for heart attack rates
png(file = "hospital_mortality3.png", width=900,
    Height = 660, res = 150)
hist(outcome[, 11], xlab='Deaths',
      ,main = 'Hospital 30-Day Death (Mortality) Rates from
Heart Attack', col = "lightblue")
dev.off()
## png
## 2
```



**Figure 2-4.** Histogram representing hospital 30-day mortality (death) rates from heart attacks.

## Example 2-3. Best Hospital in a State

In this example, our task is to write a function called `best()` that take two arguments:

- the 2-character abbreviated name of a state, and
- an outcome name.

The function reads the `outcome-of-care-measures.csv` file and returns a character vector with the name of the hospital that has the best (i.e., lowest) 30-day mortality rate for the specified outcome in that state. The hospital name is the name provided in the `Hospital.Name` variable. The outcomes can be one of “heart attack”, “heart failure”, or “pneumonia”. Hospitals that do not have data on a particular outcome will be excluded from the set of hospitals when deciding the rankings.

Note that we use the column-binding function `cbind()` to extract the columns from the dataset where the outcomes are conditions we are interested in, like “heart attack”, and binds them together as a new dataframe called `rates`, comprised of state, hospital, heart attack, heart failure, and pneumonia (which are the corresponding column names defined in the next code chunk).

Also note that we use two logical operators, `!` (not or negation) and `%in%`, a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for its left operand. For example, `!state %in% rates[,state]` provides a condition where the names of the state requested is not in the list of states in the dataset. Also, `!is.na`, uses `is.na` (are there any NAs in the data) and `!` as a condition for determining if valid rate value is present or not. For example, if the value is not NA, `!is.na`, the rates value is valid, otherwise it is not.

```
best <- function(state, outcome) {  
  ## Read outcome data  
  outcomes <- read.csv("outcome-of-care-measures.csv",  
    colClasses = "character",  
    header = TRUE)  
  ## Get data we're interested in
```

```

rates <- as.data.frame(cbind(outcomes[, 2],    # hospital
                             outcomes[, 7],    # state
                             outcomes[, 11],   # heart attack
                             outcomes[, 17],   # heart failure
                             outcomes[, 23]),  # pneumonia
                        stringsAsFactors = FALSE)

## Rename columns
Col names(rates) <- c("hospital", "state", "heart attack",
                      "heart failure", "pneumonia")
## Check that state and outcome are valid.
if(!state %in% rates[, "state"]){
  stop('invalid state')
}
if(!outcome %in% c("heart attack", "heart failure",
                    "pneumonia")){
  stop('invalid outcome')
}

## Return hospital name in that state with lowest 30-day
death rate
## Get only the hospitals in chosen state
hRates <- rates[(rates[, "state"] == state), ]
## Convert outcome rate to numeric values
hRates[, outcome] <- as.numeric(hRates[, outcome])
## Only include values that are not NA
hRates <- hRates[!is.na(hRates[, outcome])]

## Order by outcome rate
hRates <- hRates[order(hRates[, outcome]), ]
## Get names of hospital with the lowest rate
hNames <- hRates[hRates[, outcome] == min(hRates
                                             [, outcome]), 1]
## Sort by hospital name if tie
sort(hNames)[1]
}

```

Here are some sample outputs from `best()`. First, we look for the best hospital in Texas for heart attacks, and the best hospital in Maryland for pneumonia.

```

print(best("TX", "heart attack"))
## [1] "CYPRESS FAIRBANKS MEDICAL CENTER"
print(best("MD", "pneumonia"))
## [1] "GREATER BALTIMORE MEDICAL CENTER"

```

## Example 2-4. Rank of Hospitals by Outcome in a State

Here, we want to write a function called `rankhospital` that takes three arguments:

- the 2-character abbreviated name of a state (`state`)
- an outcome (`outcome`), and
- the ranking of a hospital in that state for that outcome (`num`)

The function reads the `outcome-of-care-measures.csv` file and returns a character vector with the name of the hospital that has the ranking specified by the `num` argument (argument 3). For example, the call `rankhospital("MD", "heart failure", 5)` returns a character vector containing the name of the hospital with the 5th lowest 30-day death rate for heart failure. The `num` argument can take values “best”, “worst”, or an integer indicating the ranking (smaller numbers are better). If the number given by `num` is larger than the number of hospitals in that state, then the function will return `NA`. Hospitals that do not have data on a particular outcome should be excluded from the set of hospitals when deciding the rankings.

```
rankhospital <- function(state, outcome, num = 'best') {  
  ## Read outcome data  
  outcomes <- read.csv("outcome-of-care-measures.csv",  
    colClasses = "character",  
    header = TRUE)  
  ## Get data we're interested in  
  rates <- as.data.frame(cbind(outcomes[, 2],    # hospita  
  L  
    outcomes[, 7],    # state  
    outcomes[, 11],   # heart attack  
    outcomes[, 17],   # heart failure  
    outcomes[, 23]),  # pneumonia  
    stringsAsFactors = FALSE)  
  ## Rename columns  
  colnames(rates) <- c("hospital", "state",  
    "heart attack", "heart failure", "pneumonia")  
  ## Check that state and outcome are valid  
  if(!state %in% rates[, "state"]){
    stop('invalid state')
```

```

}

if(!outcome %in% c("heart attack", "heart failure",
  "pneumonia")){
  stop('invalid outcome')
}

## Return hospital name in that state with lowest 30-day
## death rate
## Get only the hospitals in chosen state
hRates <- rates[(rates[, "state"] == state), ]
## Convert outcome rate to numeric, gets a warning
hRates[, outcome] <- as.numeric(hRates[, outcome])
## Remove NA values
hRates <- hRates[!is.na(hRates[, outcome]) , ]
## convert num argument to valid rank
if(num == "best") {
  num <- 1
}
if (num == "worst") {
  num <- nrow(hRates)
}
## Order by outcome rate
hRates <- hRates[order(hRates[, outcome], hRates[, "hospital"]), ]
## Get names of hospital
hRates[num,1]
}

```

### *Sample Outputs from rankhospital*

```

rankhospital("TX", "heart failure", 4)
# 4th best hospital in Texas for heart failure
## [1] "DETAR HOSPITAL NAVARRO"
rankhospital("MD", "heart attack", "worst")
# worse hospitals in Maryland for heart attack
## [1] "HARFORD MEMORIAL HOSPITAL"

```

## Example 2-5. Ranking Hospitals in all States

Next, we were to write a function called `rankall()` that takes two arguments:

- an outcome name (outcome) and

- a hospital ranking (num)

The function reads the `outcome-of-care-measures.csv` file and returns a 2-column data frame containing the hospital in each state that has the ranking specified in num. For example, the function call `rankall("heart attack", "best")` would return a data frame containing the names of the hospitals that are the best in their respective states for 30-day heart attack death rates. The function should return a value for every state (some may be `NA`). The first column in the data frame is named `hospital`, which contains the hospital name, and the second column is named `state`, which contains the 2-character abbreviation for the state name. Hospitals that do not have data on a particular outcome are excluded from the set of hospitals when deciding the rankings.

Notice that we reuse `rates <- as.data.frame...` from `rankhospital()`. We also include a simple for loop, explained in the code comments, and two `if-then-else` statements, also explained in the code comments.

```
rankall <- function(outcome, num = 'best') {
  ## Read outcome data
  outcomes <- read.csv()( "outcome-of-care-measures.csv",
    colClasses = "character",
    header = TRUE)
  ## Get data we're interested in
  rates <- as.data.frame()(cbind())(outcomes[, 2],    # hospital
    outcomes[, 7],    # state
    outcomes[, 11],   # heart attack
    outcomes[, 17],   # heart failure
    outcomes[, 23]),  # pneumonia
    stringsAsFactors = FALSE)

  ## Rename columns
  colnames(rates) <- c("hospital", "state",
    "heart attack", "heart failure", "pneumonia")
  ## Check outcome is valid: if the medical condition is not
  ## one of the 3 of interest, escape from the function
  if(!outcome %in% c("heart attack", "heart failure",
    "pneumonia")){
    stop('invalid outcome')}
```

```

}

## Return hospital name in that state with lowest 30-day
## death rate
hRank <- data.frame()
## For the requested state in a list of unique, sorted s
tates, do...
for(state in sort(unique(rates[, "state"]))) {
## Get only the hospitals in this state (uses the comparis
on operator ==)
## type ?== to learn more
hRates <- rates[(rates[, "state"] == state), ]
## Convert outcome rate to numeric, gets a warning
hRates[, outcome] <- as.numeric(hRates[, outcome])
## Remove NA values
hRates <- hRates[!is.na(hRates[, outcome]) (), ]
## convert num argument to valid rank number (rnum):
## if the value is "best" assign it a value of 1.
## if the value is "worst" assign it the corresponding
row-value
## for example, if Texas is the state in row 20 and is
rated as "worst", the rnum returned is 20.
if(num == "best") {
  rnum <- 1
} else if (num == "worst") {
  rnum <- nrow(hRates)
}
else {rnum = num}
## Order by outcome rate & hospital name
hRates <- hRates[order(hRates[, outcome],
  hRates[, "hospital"]), ]
hName <- hRates[rnum, 1]
hRank <- rbind (hRank,
  data.frame(hospital = hName,
  state = state))
}
## Return dataframe
hRank
}

```

### *Rankings for Best Hospital for a medical condition in the State*

First, let's look for the hospital ranked 20 for heart attacks.

```

head(rankall("heart attack", 20), 10)

##                                     hospital state
## 1                               <NA>    AK
## 2      D W MCMILLAN MEMORIAL HOSPITAL    AL
## 3      ARKANSAS METHODIST MEDICAL CENTER   AR
## 4  JOHN C LINCOLN DEER VALLEY HOSPITAL    AZ
## 5          SHERMAN OAKS HOSPITAL        CA
## 6      SKY RIDGE MEDICAL CENTER        CO
## 7      MIDSTATE MEDICAL CENTER        CT
## 8                               <NA>    DC
## 9                               <NA>    DE
## 10     SOUTH FLORIDA BAPTIST HOSPITAL     FL

```

Next, let's look for the worst ranked hospital for pneumonia.

```

tail(rankall("pneumonia", "worst"), 3)

##                                     hospital state
## 52  MAYO CLINIC HEALTH SYSTEM - NORTHLAND, INC    WI
## 53          PLATEAU MEDICAL CENTER        WV
## 54  NORTH BIG HORN HOSPITAL DISTRICT     WY

```

## Summary

The primary “take-aways” from this chapter are:

- User defined functions can be created for customized functions
- Writing functions in R can include previously defined operators, like `==`
- Writing functions in R can include functions that have already been written (packages)
- Writing functions in R can be done by nesting previously defined functions (packages)
- Writing functions in R is fairly intuitive



### 3. Exploratory Data Analysis (EDA)

#### Introduction

If you are reading this chapter, you should be familiar with all the kind of basic tools in the data scientist's toolbox. You should be familiar with the basics of R programming in terms of writing functions and writing basic operations on data. We should have covered the aspects of getting data from the internet and through various APIs and basic data processing.

Now we consider a data set that's been through a variety of processing steps and it's time to examine what the data can possibly tell us, whether it will be useful in answering our analysis questions. We do this by looking at data summaries, structures, plots of potential relationships, and so on. This is before we perform modeling, before we make predictions, before we do any sort of statistical inference. Consequently, we are going to talk about plotting data, beyond the basics and we'll look at some examples or case studies of exploratory data analysis. Let's start with a definition.

**Definition 3-1.** *Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.*

#### Principles of Analytic Graphics

First, we are going to discuss some basic principles for building analytic graphs. The goal is to provide some general rules that we can follow when we're building analytic graphics from data, to trying to tell a story about what's happening with the data. We have found that rules to be useful when thinking through the process of building data graphics and applying the rules to many different situations.

Edward Tufte, in his book Beautiful Evidence, six principles that we can adopt in our context (Tufte, 2006).

- **Principle 1:** Show comparisons
  - Evidence for a hypothesis is always *relative* to another

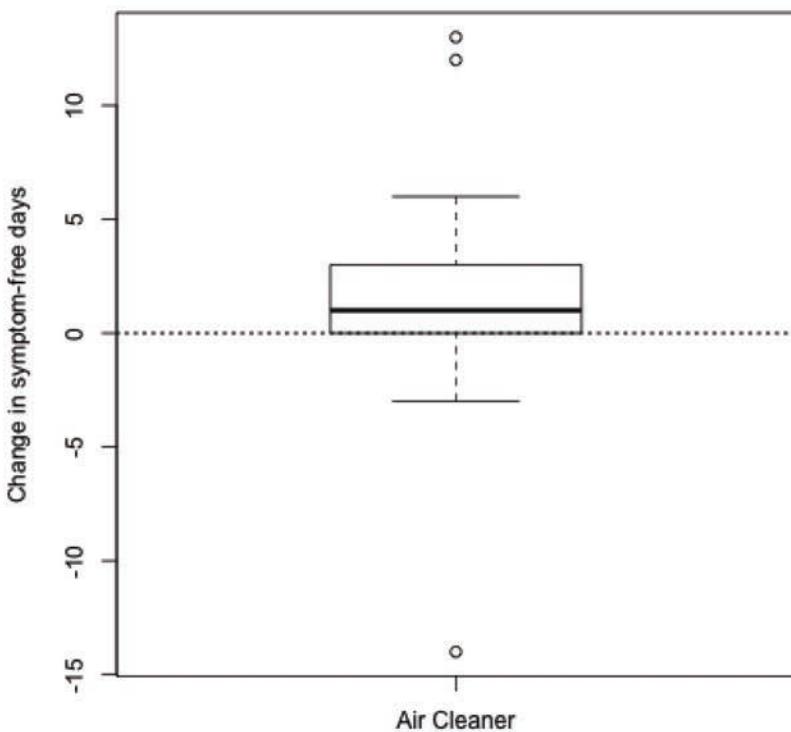
- competing hypothesis.
- Always ask "Compared to What?"
- **Principle 2:** Show causality, mechanism, explanation, systematic structure
  - What is your causal framework for thinking about a question?
- **Principle 3:** Show multivariate data
  - Multivariate = more than 2 variables
  - The real world is multivariate
  - Need to "escape flatland"
- **Principle 4:** Integration of evidence
  - Completely integrate words, numbers, images, diagrams
  - Data graphics should make use of many modes of data presentation
  - Don't let the tool drive the analysis
- **Principle 5:** Describe and document the evidence with appropriate labels, scales, sources, etc.
  - Completely integrate words, numbers, images, diagrams
  - Data graphics should make use of many modes of data presentation
  - Don't let the tool drive the analysis
  - A data graphic should tell a complete story that is credible
- **Principle 6:** Content is king
  - Analytical presentations ultimately stand or fall depending on the quality, relevance, and integrity of their content

### *First Principle*

The first principle is to show comparisons and this is a basic idea in all of science. The idea is that evidence for a hypothesis is always going to be relative to another hypothesis. So, evidence is always relative. If we're working with Hypothesis A, there must be some alternative hypothesis that we're going to use for comparison. Thus, whenever we hear a statement or summary of evidence, based on data, we should always ask the question, "Compared to what?"

### Example 3-1

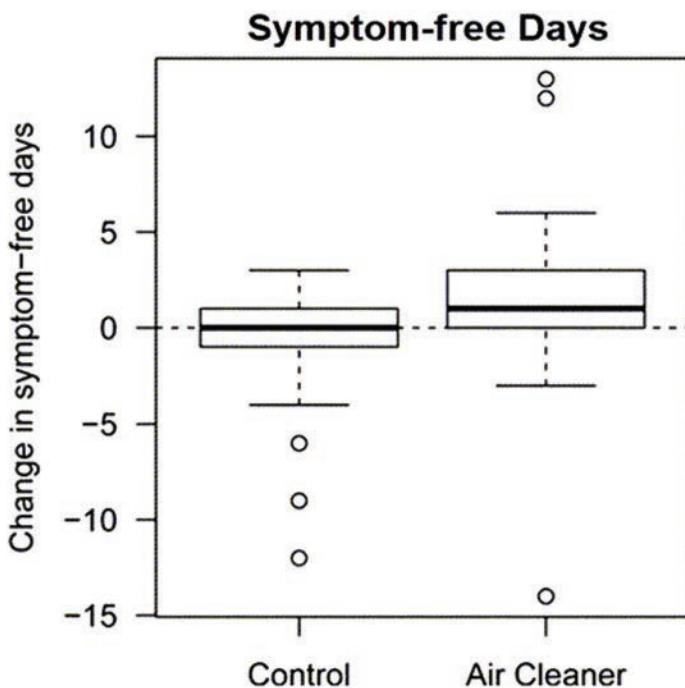
In this example we have a boxplot that represents the effect of an air cleaner on the asthma symptoms of children in a certain class of homes. In an experiment an air cleaner was introduced into a child's home, to increase indoor air quality levels, and we want to see if their asthma symptoms are improving. Thus, the outcome is whether a child is symptom-free for a number of days. So, a higher number is better. Referring to the boxplot in **Figure 3-1**, we can see that the group of children *that received the air cleaner in their home had an increase in their symptom-free days*.



**Figure 3-1.** Boxplot of the change in days of symptom-free asthma due to home installed air cleaners. Reference: Butz AM, et al., JAMA Pediatrics, 2011.

The plot in **Figure 3-2** shows a median increase in symptom-free days over 2 weeks. We might be inclined to conclude that over the course of time the air cleaner works. However, the real question is, “Compared to what?” So, what do we compare the air cleaner to?

Then, what do we compare the air cleaner to? The “what we compare them to” in this case, is nothing. Hence, our control could be the set of houses that have no air cleaner, i.e., no intervention to improve air quality. So, this was a randomized control trial that looked at installing an air cleaner in a child's home versus installing nothing in the child's home. From the pair of boxplots, in **Figure 3-2**, we can see that in the control homes, the average change in the symptom free case was about 0. Thus, there's really no change. However, the average change in symptoms in the air cleaner homes was about 1 symptom free day for, 2 weeks.



**Figure 3-2.** Boxplot of the change in days of symptom-free asthma due to home installed air cleaners compared with homes not installed with air cleaners. (Peng, et al., 2015, p. 20)

Now, we might say, relative to doing nothing, the air cleaner is actually a little bit better, showing an improvement in the child's symptoms. It's always important to show a comparison in a plot, so you need show a comparison of evidence between two different hypotheses.

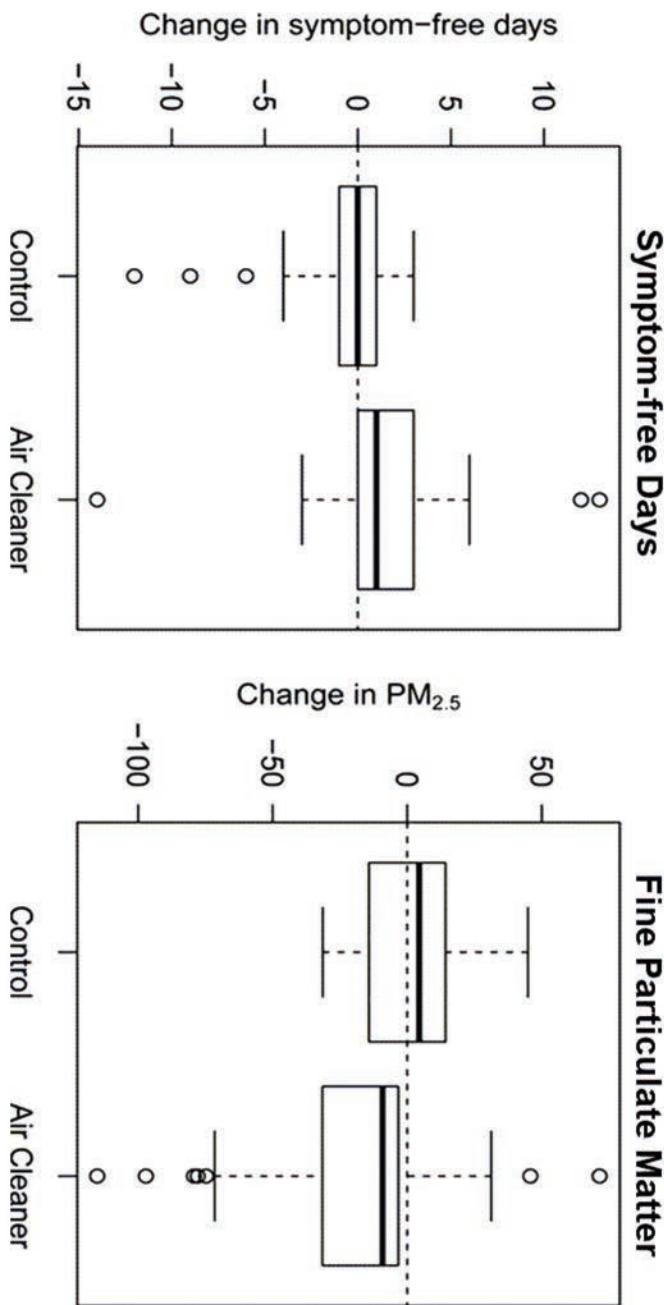
## *Second Principle*

The second principle, is to show causality or mechanism or, to provide an explanation showing some systematic structure. Now, the use of the word causality here is not supposed to be formal, as in statistical causality. Rather, it is just to show what we believe and what brought us to that belief. So, we need to be able to show what, how you believe the system is kind of operating, so to speak. That is, what is the causal framework for thinking about the question that we're of interest?

### *Example 3-2*

We might extend the example cleaning the air in a child's home, where on average, they're going to experience a one symptom-free day increase, which is a better outcome in their asthma symptoms, compared with no air treatment. We might ask, for example, why does that occur? Why does installing an air cleaner in a child's home, improves their symptoms? Of course, we hypothesize that the air cleaner is cleaning the air—it's removing particulate matter from the air, reducing the particulate matter going into the child's lungs, and may be triggering a reduction in their asthma symptoms. Now, that's how we believe things work, and we can show a plot, that might corroborate that evidence.

Here, we can show a plot in **Figure 3-3**, which has the symptom free days on the left-hand side, and the particulate matter, on the right-hand side. Now we can see the effect of the air cleaner on the particulate matter levels inside the child's home. We also can see that for the control group, there was basically no change in the particulate matter levels. But there was a substantial decrease in particulate matter levels in the homes with air cleaners. So now we can see that not only did a child's symptom free days increase when they got the air cleaner, but also their indoor air particulate matter levels decreased.



**Figure 3-3.** Symptom free days on the top, and the particulate matter, on the left-hand side. (Peng, et al., 2015, p. 20)

So, using the data we observed, we can say that adding an air cleaner does seem to decrease their particulate matter levels. Of course, to confirm that this hypothesis that the air cleaner reduces particulate matter, we'd have to perform a little more investigation and perhaps more experimentation. But the does suggests a possible explanation.

### ***Third Principle***

The third principle is to show multivariate data. The basic this rule is to show as much data on a single plot as we can. The reason is because data are inherently multivariate—there's lots of things going on all the time. If we just plot 2 or 3 variables, they're not going to show the real picture of what's occurring in the world. If we can, we want put a lot of data on a plot, then you'll be able to tell a much richer story.

### ***Example 3-3 - Penguins***

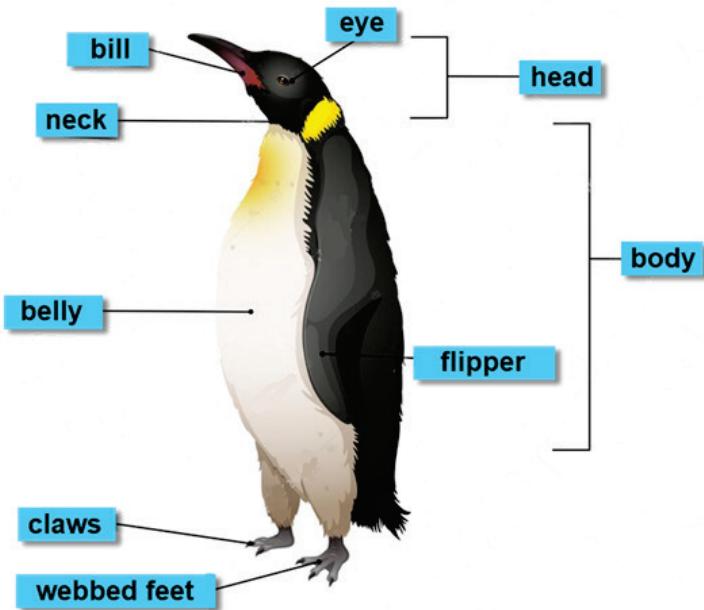
Our goal here is to look at a situation where we can misinterpret data if our EDA is not thorough. We'll see a phenomenon called *Yule-Simpson effect* or *Simpson's Paradox*.

***Definition 3-2.*** *Simpson's Paradox is a phenomenon in probability and statistics, in which a trend appears in several different groups of data but disappears or reverses when these groups are combined.*

In other words, the same data set can appear to show opposite trends depending on how it's grouped. The reason there are different interpretations of the same data, and what is evading our eye is something called the Lurking variable — a conditional variable that can affect our conclusions about the relationship between two variables.

To examine this phenomenon, we'll use a drop-in replacement for the famous iris dataset. The dataset consists of details about three species of penguins, including their culmen length and culmen depth, their flipper length, body mass, and sex. The culmen is essentially the upper ridge of a penguin's beak, while their wings are called flippers. These are shown in **Figure 3-4**. We'll look at `culmen_length_mm` versus `culmen_depth_mm`.

# Parts of a Penguin



**Figure 3-4.** The parts of the penguin (Source: Nature vector created by brgfx - [www.freepik.com](http://www.freepik.com))

## MICE Package

There are few missing values in the dataset. Let's get rid of those. MICE (Multivariate Imputation via Chained Equations) is one of the commonly used packages by R users. Creating multiple imputations as compared to a single imputation (such as mean) takes care of uncertainty in missing values.

MICE assumes that the missing data are Missing at Random (MAR), which means that the probability that a value is missing depends only on observed value and can be predicted using them. It imputes data on a variable-by-variable basis by specifying an imputation model per variable.

## *Install and Load Libraries*

```
library(mice)
library(missForest)
library(VIM)
library(ggplot2)
library(readr)
```

## *Set working directory is needed and map the path*

```
path = getwd()
cat("The working directory is ", path)

## The working directory is C:/Users/jeff/Documents/R/eda1
```

## *Get zip Data from GitHub*

There are two .csv files in penguin.zip \* penguin-size.csv \* penguin\_lter.csv

```
filename = "penguins_size.csv"
if (!file.exists(filename)) {
  urlzip <- "https://github.com/stricje1/eda1/raw/main/penguins.zip"
  download.file(urlzip, destfile = "./penguins.zip", method = "curl", extra = "-L")
  unzip("./penguins.zip", exdir = path )
}
```

## *Create dataframe from penguin-size data*

```
df1 <- read.csv("penguins_size.csv")
head(df1, 11)

##   species   island culmen_length culmen_depth flipper_length
## 1   Adelie Torgersen      39.1       18.7          181
## 2   Adelie Torgersen      39.5       17.4          186
## 3   Adelie Torgersen      40.3       18.0          195
## 4   Adelie Torgersen       NA          NA           NA
## 5   Adelie Torgersen      36.7       19.3          193
## 6   Adelie Torgersen      39.3       20.6          190
## 7   Adelie Torgersen      38.9       17.8          181
## 8   Adelie Torgersen      39.2       19.6          195
## 9   Adelie Torgersen      34.1       18.1          193
## 10  Adelie Torgersen      42.0       20.2          190
```

```

## 11 Adelie Torgersen      37.8       17.1       186
##   body_mass_g    sex
## 1        3750  MALE
## 2        3800 FEMALE
## 3        3250 FEMALE
## 4          NA <NA>
## 5        3450 FEMALE
## 6        3650  MALE
## 7        3625 FEMALE
## 8        4675  MALE
## 9        3475 <NA>
## 10       4250 <NA>
## 11       3300 <NA>

```

### *Check for Missing Values*

```
head(is.na(df1[,1:5]),8)
```

```

##      species island culmen_length culmen_depth flipper_length
## [1,] FALSE  FALSE      FALSE      FALSE      FALSE
## [2,] FALSE  FALSE      FALSE      FALSE      FALSE
## [3,] FALSE  FALSE      FALSE      FALSE      FALSE
## [4,] FALSE  FALSE      TRUE       TRUE       TRUE
## [5,] FALSE  FALSE      FALSE      FALSE      FALSE
## [6,] FALSE  FALSE      FALSE      FALSE      FALSE
## [7,] FALSE  FALSE      FALSE      FALSE      FALSE
## [8,] FALSE  FALSE      FALSE      FALSE      FALSE

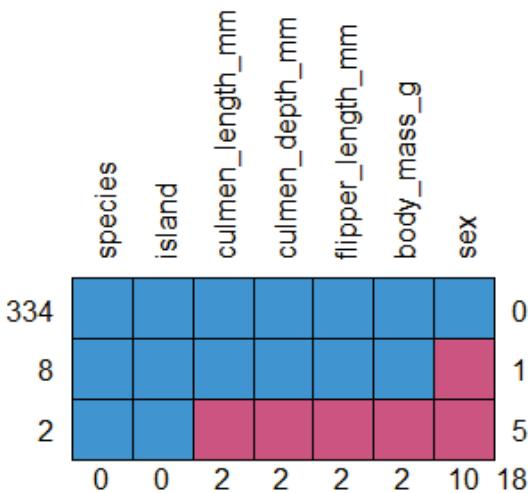
```

We already know there are missing values in the dataset, so let's look at a visual for insight.

### *Plot missing data patterns*

The `md.pattern()` function displays missing data. is useful for investigating any structure of missing observations in the data as seen in **Figure 3-5**. In specific case, the missing data pattern could be (nearly) monotone. Monotonicity can be used to simplify the imputation model. See Schafer (1997) for details. Also, the missing pattern could suggest which variables could potentially be useful for imputation of missing entries.

```
md.pattern(df1, plot = TRUE, rotate.names = TRUE)
```

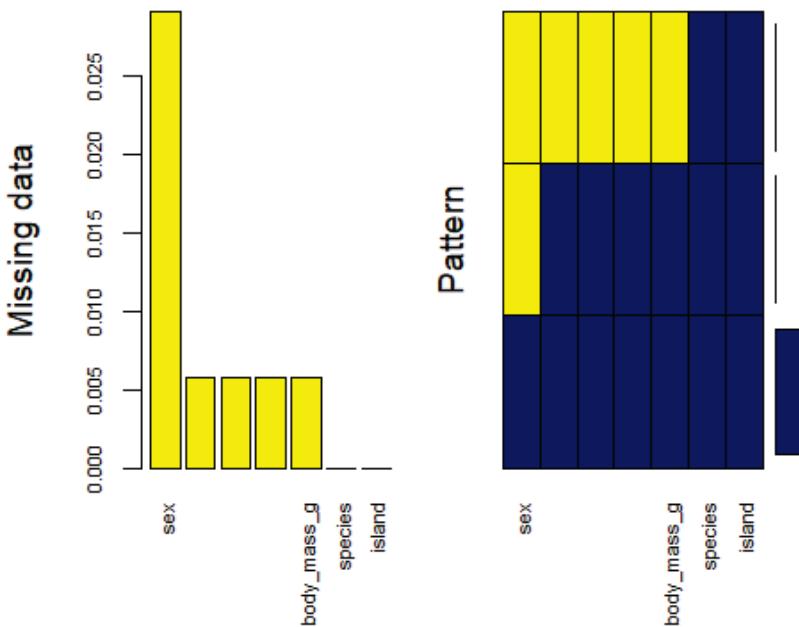


**Figure 3-5.** Pattern plot for Penguin missing data

```
##      species island culmen_length culmen_depth flipper_length
## 334        1      1                 1                 1                 1
## 8          1      1                 1                 1                 1
## 2          1      1                 0                 0                 0
##          0      0                 2                 2                 2
##      body_mass_g sex
## 334        1      1     0
## 8          1      0     1
## 2          0      0     5
##          2     10    18
```

We use the `aggr()` function to plot the number of missing/imputed values in each variable and the number of missing/imputed values in certain combinations of variables. We are often interested in how many missing/imputed values are contained in each variable. Even more interesting, there may be certain combinations of variables with a high number of missing/imputed values, as seen in **Figure 3-6**.

```
mice_plot <- aggr(df1, col = c('navyblue','yellow'),
                    numbers = TRUE, sortVars = TRUE,
                    labels = names(df1), cex.axis = .7,
                    gap = 3,
                    ylab = c("Missing data","Pattern"))
```



**Figure 3-6.** Aggregations for missing/imputed values in the Penguin dataset. The variable, `sex`, has more missing values.

```
##  
## Variables sorted by number of missings:  
##           Variable      Count  
##             sex  0.029069767  
##   culmen_length_mm  0.005813953  
##   culmen_depth_mm  0.005813953  
##   flipper_length_mm  0.005813953  
##         body_mass_g  0.005813953  
##           species  0.000000000  
##           island  0.000000000
```

### Impute missing values

We now use the `mice` package to input missing values. The `mice()` function can impute mixes of continuous, binary, unordered categorical and ordered categorical data. In addition, `mice` can impute continuous two-level data, and maintain consistency between imputations by means of passive imputation. Among the parameters are the `data`, `df1` in this case, `m` or the number of multiple

imputations, with the default being 5. `maxit` is a scalar giving the number of iterations, with the default being 5. `method` can be either a single string, or a vector of strings with length `ncol(data)`, specifying the univariate imputation method to be used for each column in `data`. we'll use `pmm`, predictive mean matching (numeric data). Finally, we'll set the random number seed to 500.

```
imputed_Data <- mice(df1, m=5, maxit = 50, method = 'pmm',
seed = 500)

## iter imp variable
## 1   1 culmen_length_mm culmen_depth_mm flipper_length_mm
## 1   2 culmen_length_mm culmen_depth_mm flipper_length_mm
## 1   3 culmen_length_mm culmen_depth_mm flipper_length_mm
## 1   4 culmen_length_mm culmen_depth_mm flipper_length_mm
## 1   5 culmen_length_mm culmen_depth_mm flipper_length_mm
## 2   1 culmen_length_mm culmen_depth_mm flipper_length_mm
## 2   2 culmen_length_mm culmen_depth_mm flipper_length_mm
## 2   3 culmen_length_mm culmen_depth_mm flipper_length_mm
## 2   4 culmen_length_mm culmen_depth_mm flipper_length_mm
## 2   5 culmen_length_mm culmen_depth_mm flipper_length_mm
## 3   1 culmen_length_mm culmen_depth_mm flipper_length_mm
## 3   2 culmen_length_mm culmen_depth_mm flipper_length_mm
## 3   3 culmen_length_mm culmen_depth_mm flipper_length_mm
## 3   4 culmen_length_mm culmen_depth_mm flipper_length_mm
## 3   5 culmen_length_mm culmen_depth_mm flipper_length_mm
## 4   1 culmen_length_mm culmen_depth_mm flipper_length_mm
## 4   2 culmen_length_mm culmen_depth_mm flipper_length_mm
## 4   3 culmen_length_mm culmen_depth_mm flipper_length_mm
## 4   4 culmen_length_mm culmen_depth_mm flipper_length_mm
## 4   5 culmen_length_mm culmen_depth_mm flipper_length_mm
...
## 49  5 culmen_length_mm culmen_depth_mm flipper_length_mm
## 50  1 culmen_length_mm culmen_depth_mm flipper_length_mm
## 50  2 culmen_length_mm culmen_depth_mm flipper_length_mm
## 50  3 culmen_length_mm culmen_depth_mm flipper_length_mm
## 50  4 culmen_length_mm culmen_depth_mm flipper_length_mm
## 50  5 culmen_length_mm culmen_depth_mm flipper_length_mm

summary(imputed_Data)

## Class: mids
## Number of multiple imputations:  5
## Imputation methods:
```

```

##      species      island  culmen_length_mm  culmen_depth_mm
##      ""          ""          "pmm"           "pmm"
## flipper_length_mm      body_mass_g      sex
##          "pmm"           "pmm"           ""
## PredictorMatrix:
##      species island culmen_length_mm culmen_depth_mm
## species      0      0          1          1
## island       0      0          1          1
## culmen_length_mm  0      0          0          1
## culmen_depth_mm   0      0          1          0
## flipper_length_mm 0      0          1          1
## body_mass_g      0      0          1          1
##      flipper_length_mm body_mass_g sex
## species            1      1      0
## island             1      1      0
## culmen_length_mm  1      1      0
## culmen_depth_mm   1      1      0
## flipper_length_mm 0      1      0
## body_mass_g        1      0      0
## Number of logged events: 3
##    it im dep     meth     out
## 1  0  0  constant species
## 2  0  0  constant island
## 3  0  0  constant   sex

```

### *Check Imputed Dataset*

Now that we have imputed missing values, we want to verify that we were successful. The next code chunk shows us the values that replaced the missing rows (4 and 430) for each of the 5 imputations.

```

imputed_Data$imp$culmen_length_mm

##      1      2      3      4      5
## 4  46.6  38.8  41.6  39.7  42.5
## 340 49.5  50.5  39.7  35.0  37.3

```

From the output, we'll select the third imputation values to replace the missing values.

### *Get complete data (3rd out of 7)*

The `complete()` function takes an object of class `mids`, fills in the missing data, and returns the completed data in a specified format.

To repeat, we'll fill in the third imputation (recall that we did `m = 5` imputations.

```
completeData <- complete(imputed_Data,3)
```

	species	island	culmen_length_mm	culmen_depth_mm
[1,]	TRUE	TRUE	TRUE	TRUE
[2,]	TRUE	TRUE	TRUE	TRUE
[3,]	TRUE	TRUE	TRUE	TRUE
[4,]	TRUE	TRUE	TRUE	TRUE
[5,]	TRUE	TRUE	TRUE	TRUE
[6,]	TRUE	TRUE	TRUE	TRUE

### *Recheck for missing values*

Now we'll reinspect the data for missing values. The output shows there are none.

```
is.na(completeData) == 0
```

### *Create dataframe*

Now, we'll create a dataframe from the `complteData` imputed data.

```
df2 <- data.frame(completeData)
```

### *Define x and y variables*

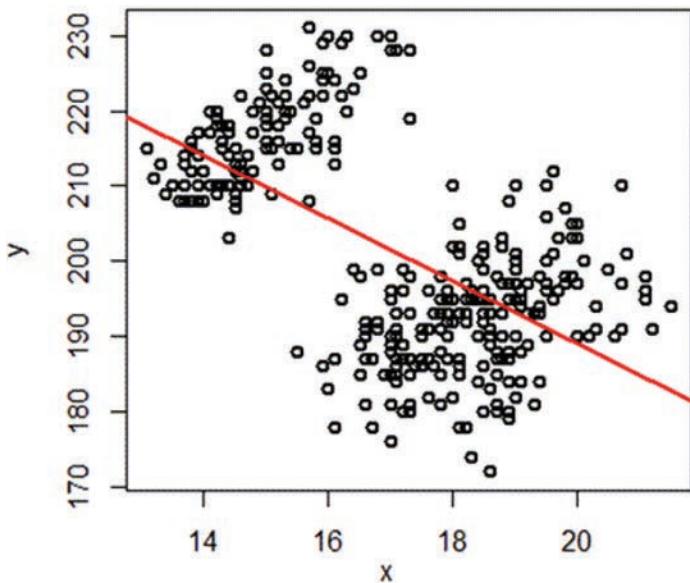
Here, we'll define the variables `x` and `y` as `culmen_length_mm` and `culmen_depth_mm`, respectively.

```
x <- df2[,3]
y <- df2[,4]
```

### *Create scatterplot*

Now, we'll generate a scatterplot of the `x` and `y` variables, with a line fitted for the data. The plot in **Figure 3-7** will show a trend, indicated by the fitted line, of the two plotted variables. It shows that there is a slightly downward trend (negative slope) in the data when considering `culmen_length_mm` versus `culmen_depth_mm` only. This demonstrates Simpson's Paradox.

```
plot(x,y, lwd = 2)
abline(lm(y ~ x, data = df2), col = "red", lwd = 2)
```

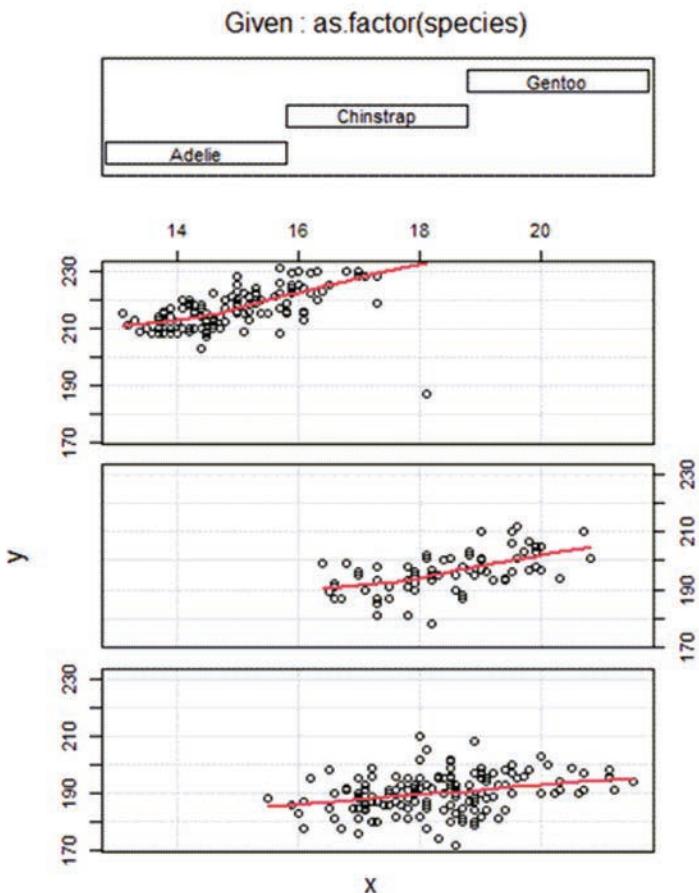


**Figure 3-7.** Scatterplot of culmen length (mm) and culmen depth (mm)

### Create Coplot by Species

If we stopped now, we might not understand what the data is trying to tell us as we interrogate it. Plotting `culmen_length_mm` versus `culmen_depth_mm` by the three penguin species yields a different insight. A conditioning plot or co-plot is a scatterplot of two variables when conditioned on a third variable. The third variable is called the conditioning variable. This variable can have both values either continuous or categorical. In **Figure 3-8** we can see that the actual trends are slightly positive.

```
require(graphics)
coplot(y ~ x | as.factor(species), data = df2,
       panel = panel.smooth, rows = 3, lwd = 2)
```



**Figure 3-8.** An example of Simpson's Paradox

### Conclusion

### Fourth Principle

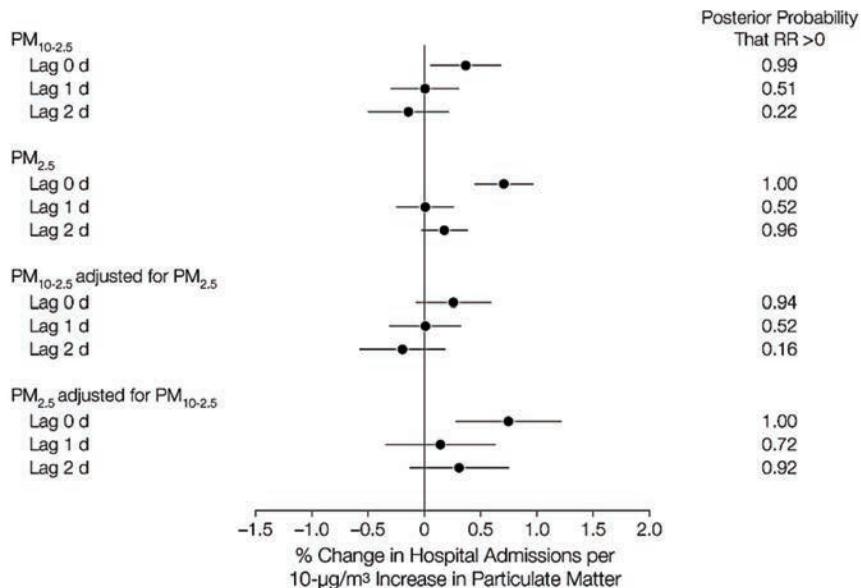
The **fourth principle** is to integrate the evidence we have. The idea here is that we want to use as many different modes of evidence to display. So, there's no reason to think that if we have a tool that makes a plot, that we should only show a plot. Likewise, if we only have the ability to make a table, we only show a table. We should be able to combine different modes of evidence into a single representation, and make it as information rich as possible. One of the advantages of a system like R, is the tools are very flexible, allowing us to construct

all kinds of customized plots to show the data and to kind of integrate different modes of evidence.

### Example 3-4 – Course Particulate Matter

Here is a quick example from the Journal of the American Medical Association, looking at the relationship between coarse particulate matter and hospitalizations in the elderly, depicted in **Figure 3-9**.

Percentage Change in Emergency Hospital Admissions Rate for Cardiovascular Diseases per a  $10\text{-}\mu\text{g}/\text{m}^3$  Increase in Particulate Matter



**Figure 3-9.** Estimates are on average across 108 counties.  $\text{PM}_{2.5}$  indicates particulate matter is  $2.5\text{ }\mu\text{m}$  or less in aerodynamic diameter;  $\text{PM}_{10}$ , particulate matter is  $10\text{ }\mu\text{m}$  or less in aerodynamic diameter;  $\text{PM}_{10-2.5}$ , particulate matter is greater than  $2.5\text{ }\mu\text{m}$  and  $10\text{ }\mu\text{m}$  or less in aerodynamic diameter; RR, relative risk. Error bars indicate 95% posterior intervals.

The details of this plot are not particularly important, but we want to see that there are point estimates, which are in the solid circles, and confidence intervals indicated by the lines going through the solid circles. But then on the right-hand side we see the label called

posterior probability that represent the relative risk posed by the different concentrations. This is a measure of the strength of the evidence that the, the kind of the association between coarse particulate matter and hospitalizations is, in fact, different from zero. Hence, we integrated the point estimates with confidence intervals. But we also have values on the right showing another piece of evidence, which is the strength of that evidence, as encoded by the posterior probability.

## Fifth Principle

The fifth principle is to describe and document the evidence that we present, by using labels and sources and, and whatever we deem appropriate. If we're going to build a plot with a system like R, it's important to preserve the computer code that made the plot. The idea is that we want to lend some credibility to the evidence we present, such as the sources of the data came and how we made the plots from them. This is a basic principle and it's important for our credibility.

## Sixth Principle

The sixth principle is, "context is king." If we don't have an interesting story to tell, then there's no presentation that will make it interesting. When we're making plots, figures, and graphs, the first thing to consider is, what is the content that we're trying to present? What's the story we're trying to tell? What's the data we have? Then we think about what's the best way to present our story? How am I And we think about what it's going to look like. If we don't have very good content, then there's really not much we're going to be able to do beyond that. Thus, context has to drive everything we want to show or we run the risk that our presentations have little to no meaning.

## Principles Summary

To summarize, the six basic principles help us explore the data for pertinent parts, that is, parts that are relative to our context. Based on our analysis questions, we should have an idea of the story we want to tell, and then find a way to present the data that will help us tell the story. On the other hand, we may discover that the data does

not support our story and cause us to search for different data. Or, our story isn't supportable by the available data, which may drive us to search unstructured data or consider a different story. Hence EDA is the backbone of our analysis, and without it we cannot claim any information gain from the data.

- Principle 1: Show comparisons
- Principle 2: Show causality, mechanism, explanation
- Principle 3: Show multivariate data
- Principle 4: Integrate multiple modes of evidence
- Principle 5: Describe and document the evidence
- Principle 6: Context is king

### Air Pollution in the United States

- The U.S. Environmental Protection Agency (EPA) sets national ambient air quality standards for outdoor air pollution
  - **U.S. National Ambient Air Quality Standards**
- For fine particle pollution ( $PM_{2.5}$ ), the "annual mean, averaged over 3 years" cannot exceed  $12 \mu\text{g}/\text{m}^3$ .
- Data on daily  $PM_{2.5}$  are available from the U.S. EPA web site
  - **EPA Air Quality System**

### Example 3-5 – Pollution Monitoring

**Analysis Question:** Are there any counties in the U.S. that exceed that national standard for fine particle pollution?

#### *Solution to Example 3-5*

The U.S. EPA uses Source Classification Codes (SCCs) to classify different types of activities that generate emissions. Each SCC represents a unique source category-specific process or function that emits air pollutants. The SCCs are used as a primary identifying data element in EPA's WebFIRE (where SCCs are used to link emissions factors to an emission process), the National Emissions Inventory (NEI), and other EPA databases. The SCCs are also used by many regional, state, local and tribal agency emissions data systems. Examples of processes described by SCCs and some of the emissions they generate include:

- Burning fuel in a boiler produces oxides of nitrogen (NOx) and other criteria and hazardous air pollutants (HAP).
- An industrial process such as paint coating produces volatile organic compounds (VOC).
- Fires produce particulate matter (PM).

Sources in the SCC table are classified into the following five broad types: point, non-point, events, non-road and on-road (see EPA.GOV, Introduction to Source Classification Codes and their Use for EIS Submissions.) The source code the remaining examples in this chapter: [https://github.com/stricje1/EPA\\_emissions](https://github.com/stricje1/EPA_emissions).

**Note:** EDA and Data Wangling are often used interchangeably, but they are not the same. By our definitions, we hope it is clear that they are complementary and that data wrangling must occur before EDA can be performed. Consequently, we will perform data wangling every time we have a question that can be answered using data. So, let's look at an example for applying these tasks.

## Example 3-6 – Pollution Monitoring (continued)

**Analysis Question:** Have total emissions from PM<sub>2.5</sub> decreased in the United States from 1999 to 2008?

### *Solution to Example 3-6*

Using the base plotting system, make a plot showing the total PM<sub>2.5</sub> emission from all sources for each of the years 1999, 2002, 2005, and 2008. Here we load the packages we'll need.

```
library(ggplot2)
library(RColorBrewer)
library(dplyr)
library(ggformula)
```

We also check the working directory to ensure we have the right oath to the data.

```
getwd()
## [1] "C:/Users/jeff/Documents/R/Exploratory Data
Analysis/EPA"
```

Now, we download and unzip the file:

1. Set the filename to match “[summarySCC\\_PM25.rds](#)”
2. Check to see if already downloaded and unzipped working directory (WD)
3. If filename is missing, download to the WD from the given URL
4. Unzip the air\_pollution.zip into the WD
5. [SummarySCC\\_PM25.rds](#) & [Source\\_Classification\\_Code.rds](#) appear in the WD

The code used to load the data is conditional, saying that if the file does not exist in the working directory

```
filename = "summarySCC_PM25.rds" if  
(!file.exists(filename)){urlzip <-  
https://d396qusza40orc.cloudfront.net/exdata%2Fdata%2FNEI_  
data.zip, download.file(urlzip,  
destfile = "./air_pollution.zip")  
unzip("./air_pollution.zip", exdir = ".") }
```

Next, we load the data:

```
NEI <- readRDS("./summarySCC_PM25.rds")  
SCC <- readRDS("./Source_Classification_Code.rds")
```

Checking the data summary will help ensure we have the right data.

```
summary(NEI)  
  
##      fips          SCC        Pollutant       Emissions  
##  Length:64977  Length:64977  Length:64977  Min.: 0.0  
##  Class:character  Class:character  Class:character  
1st Qu.:     0.0  
##  Mode:character  Mode:character  Mode:character  
Median :     0.0  
Mean   :     3.4  
3rd Qu.:     0.1  
Max.    :646952.0  
##      type            year  
##  Length:6497651  Min.   :1999  
##  Class :character  1st Qu.:2002  
##  Mode  :character  Median :2005
```

```

##               Mean   :2004
##             3rd Qu.:2008
##            Max.    :2008

```

Checking the data structure will give us insight into the nature of the SCC data, the length, variable types, and summary statistics, etc. We use the `str()` function to

```

ls.str(SCC)

## Created_Date : Factor w/ 57 levels "", "1/27/2000
## 0:00:00", ... : 1 1 1 1 1 1 1 1 1 1 ...
## Data.Category : Factor w/ 6 levels
## "Biogenic", "Event", ... : 6 6 6 6 6 6 6 6 6 ...
## EI.Sector : Factor w/ 59 levels "Agriculture - Crops &
## Livestock Dust", ... : 18 18 18 18 18 18 18 18 18 ...
## Last.Inventory.Year : int [1:11717] NA NA NA NA NA NA
## NA NA NA NA ...
## Map.To : num [1:11717] NA NA NA NA NA NA NA NA NA NA
## ...
## Option.Group : Factor w/ 25 levels "", "C/I
## Kerosene", ... : 1 1 1 1 1 1 1 1 1 ...
## Option.Set : Factor w/ 18 levels "", "A", "B", "B1A", ...
## 1 1 1 1 1 1 1 1 1 ...
## Revised_Date : Factor w/ 44 levels "", "1/27/2000
## 0:00:00", ... : 1 1 1 1 1 1 1 1 1 ...
## SCC : Factor w/ 11717 levels "10100101", "10100102", ...
## 1 2 3 4 5 6 7 8 9 10 ...
## SCC.Level.Four : Factor w/ 6084 levels "", "(NH4)2 SO4
## Acid Bath System and Evaporator", ... : 4455 5583 4466 4458
## 1341 5246 5584 5983 4461 776 ...
## SCC.Level.One : Factor w/ 17 levels "Brick Kilns", ...
## 3 3 3 3 3 3 3 3 3 ...
## SCC.Level.Three : Factor w/ 1061 levels "", "100%
## Biosolids (e.g., sewage sludge, manure, mixtures of these
## matls)", ... : 88 88 156 156 156 156 156 156 156 ...
## SCC.Level.Two : Factor w/ 146 levels "", "Agricultural
## Chemicals Production", ... : 32 32 32 32 32 32 32 32 32
## ...
## Short.Name : Factor w/ 11238 levels "", "2,4-D Salts
## and Esters Prod /Process Vents, 2,4-D Recovery:
## Filtration", ... : 3283 3284 3293 3291 3290 3294 3295 3296
## 3292 3289 ...

```

```
## Usage.Notes : Factor w/ 21 levels "",,"includes  
bleaching towers, washer hoods, filtrate tanks, vacuum  
pump exhausts",...: 1 1 1 1 1 1 1 1 1 1 1 ...
```

### ***Define dataset for analysis and plotting***

Next, we aggregate the Emissions data from NEI by year and sum them.

```
totalNEI <- aggregate(Emissions ~ year, NEI, sum)
```

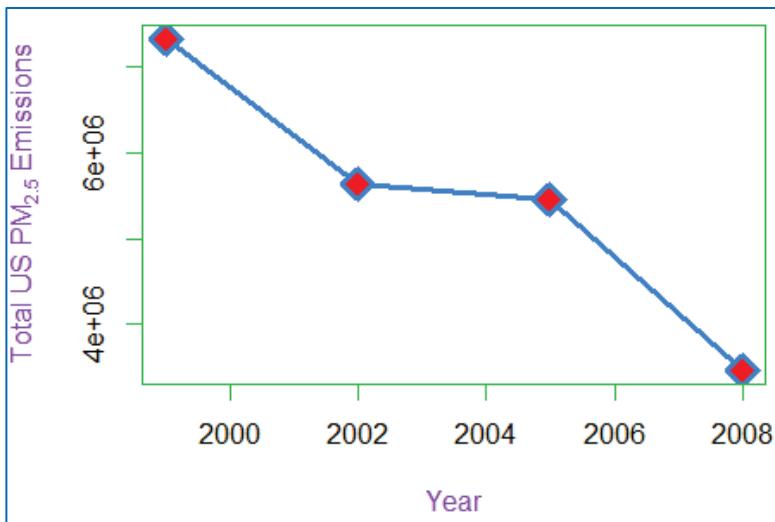
### ***Plot Construction***

In **Figure 3-10**, we construct a combined line-point plot showing emissions from the given years and save it as a PNG graphic.

```
plot(totalNEI$year, totalNEI$Emissions, # Plot variables  
      type = "o",    # Plot type is overlaid (line & points)  
      col = "dodgerblue",          # Plot line color  
      lwd = 3,           # Plot line thickness  
      font.main = 3,        # Main title font size  
      col.main = "darkred",       # Main title color  
      main = expression("Total US " ~  
                         PM[2.5] ~ "Emissions by Year"),  
                         # Title label  
      col.lab = "purple",        # Axes labels text color  
      ylab = expression("Total US " ~  
                         PM[2.5] ~ "Emissions"),   # y-axis label text  
      xlab = "Year",            # x-axis label text  
      fg = "green3",           # plot axes color  
      pch = 23,                # plot point type  
      bg = "red",              # plot point fill color  
      cex = 2)                 # Plot point size
```

### ***Conclusion to Example 3-6.***

Total emissions from PM<sub>2.5</sub> have decreased in the United States from 1999 to 2008.



**Figure 3-10.** Overlaid points and line plot of Total us PM<sub>2.5</sub> emissions by year

### Example 3-7 – Emissions in Baltimore

Have total emissions from PM<sub>2.5</sub> decreased in the Baltimore City, Maryland (fips == "24510") from 1999 to 2008? Use the base plotting system to make a plot answering this question.

#### Solution to Example 3-7

Extract Baltimore City (FIPS Code 24510) Emissions data for the corresponding years and sum them for each year. Then aggregate the data, summing by year.

```
baltimore <- subset(NEI, NEI$fips == "24510")
totalBaltimore <- aggregate(Emissions ~ year, baltimore,
sum)
```

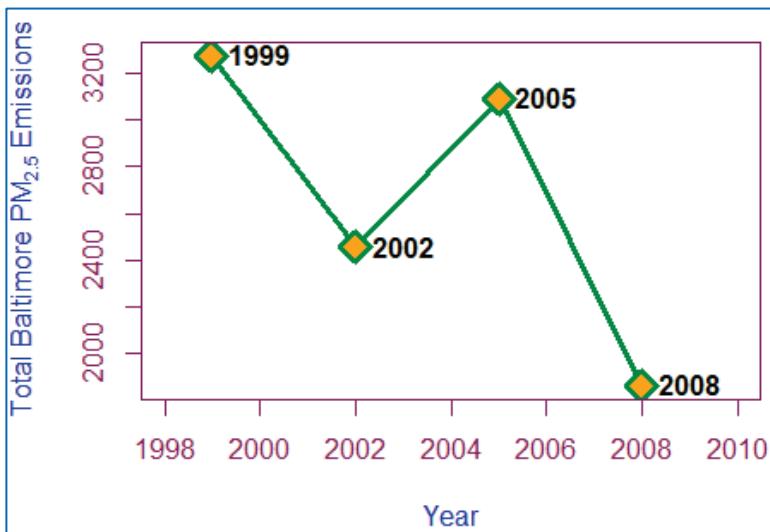
Now we construct a combined line-point plot showing emissions for Baltimore City for the given years 1999 to 2008 is shown in **Figure 3-11**.

```
p1 <- plot(totalBaltimore$year, totalBaltimore$Emissions,
           type = "o",    # Plot type is overlaid (line & points)
           col = "green4", # Plot line color
```

```

lwd = 3,                                # Plot line thickness
font.main = 3,                            # Main title font size
col.main = "darkgreen",                   # Main title color
main = expression("Total Baltimore" ~ PM[2.5]
~ "Emissions by Year"),
xlim = c(1998,2010),
xlab = "Year",                           # x-axis label text
ylab = expression("Total Baltimore " ~ PM[2.5]
~ "Emissions"),
col.lab = "blue3",
fg = "maroon4",                          # plot axes color
col.axis = "maroon4",                     # axis labels color
pch = 23,                                # plot point type
bg = "orange",                            # plot point fill color
cex = 2)                                 # Plot point size
p1 + text(Emissions ~ year, labels = totalBaltimore$year,
data = totalBaltimore, cex = 1, font = 2, pos = 4)

```



**Figure 3-11.** Overlaid points and line plot of Total us PM<sub>2.5</sub> emissions by year (annotated)

### Conclusion to Example 3-7

The total emissions from PM<sub>2.5</sub> did decrease in the Baltimore City, Maryland from 1999 to 2008. However, this is not a complete picture. Although decreasing from 1999 to 2002, there was a steep increase

from 2002 to 2005, nearly at the 1999 value. After that there was another drastic decline from 2005 to 2008.

## Example 3-8 – Emission Source Comparison

Of the four types of sources indicated by the type (point, nonpoint, onroad, nonroad) variable, which of these four sources have seen decreases in emissions from 1999-2008 for Baltimore City? Which have seen increases in emissions from 1999-2008? Use the `ggplot2` plotting system to make a plot answer this question.

### *Solution to Example 3-8*

Construct the data set comprised of emissions by source (type) in Baltimore City (FIPS Code 24510) from 1999 to 2008

```
baltimore <- subset(NEI, NEI$fips == "24510")
baltimoreType <- aggregate(Emissions ~ year + type,
baltimore, sum)
```

In Figure 3-12, we plot the four types of sources indicated by the type (point, nonpoint, onroad, nonroad) variable and the corresponding emissions from 1999 to 2008.

```
g <- ggplot(baltimoreType,
    aes(year, Emissions, col = type))
```

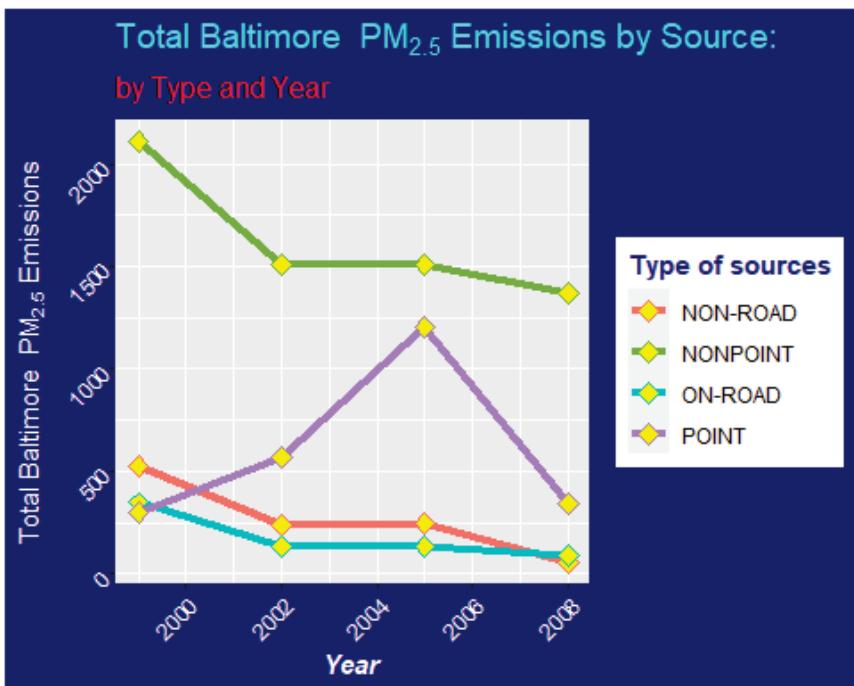
Add plot as a line type with 50% increase in size

```
g + geom_line(size = 1.5) +
  # Plot is a line type with 50% increase in size
  geom_point(size = 3, pch = 23, fill = "yellow") +
  # Main title label text
  ggtitle(expression("Total Baltimore " ~
    PM[2.5] ~ "Emissions by Source:")) + #Title label
  # Main subtitle label text
  labs(subtitle=expression("by Type and Year")) +
  # y-axis label text
  ylab(expression("Total Baltimore " ~
    PM[2.5] ~ "Emissions")) +
  # x-axis label text
  xlab("Year") +
  # plot line colors varied by pollution sources
  scale_colour_discrete(name = "Type of sources") +
```

```

# plot themes including: background color
theme(plot.background =
      element_rect(fill = "darkblue"),
# colors of axis titles
axis.title=element_text(face = "bold", color = "white"),
# color of legend title
legend.title = element_text(face="bold", color="blue4"),
# orientation, size, and color of axes labels
axis.text.x = element_text(angle = 45,
                            hjust = 1, color = "white"),
axis.text.y = element_text(angle = 45,
                            hjust = 1, color = "white"),
# font size, color, and facing of main title
plot.title= element_text(size = 14, color = "cyan",
                           face = "bold"),
# font size, color, and facing of main subtitle
plot.subtitle = element_text(size = 12, color = "red",
                             face = "bold"))

```



**Figure 3-12.** Overlaid point and line plots for the four sources of PM<sub>2.5</sub> by year, from 2000 to 2008.

### **Conclusion to Example 3-8:**

The emissions from point sources are higher in 2008 than it was in 1999, even though it has risen and fell. For the sources non-point, road, and on-road the 2008 levels are lower than those of 1999, and the overall trend is decreasing.

## **Example 3-9 – Coal Combustion Emissions**

Across the United States, how have emissions from coal combustion-related sources changed from 1999-2008?

### **Solution to Example 3-9**

Construct the data set comprised of emissions from coal related combustion across the United States from 1999 to 2008. This requires aggregating data from both base data sets and use of the pattern-matching and replacement function `grepl()` for “coal”.

```
SCCcoal <- SCC[grepl("coal", SCC$Short.Name,
                     ignore.case = T),]
NEIcoal <- NEI[NEI$SCC %in% SCCcoal$SCC, ]
totalCoal <- aggregate(Emissions ~ year + type,
                        NEIcoal, sum)
totalCoal2 <- aggregate(Emissions ~ year, NEIcoal, sum)
```

### **Construct the Plot**

Here, we'll use a combined line-point plot as we did in examples 1 and 2, but will add a spline (fitted line) to the plot as seen in **Figure 3-13**. First, we define a ggplot combining line-point plots for coal related emissions

```
g= ggplot(totalCoal2, aes(year, Emissions, col =
                           "Emissons"))
```

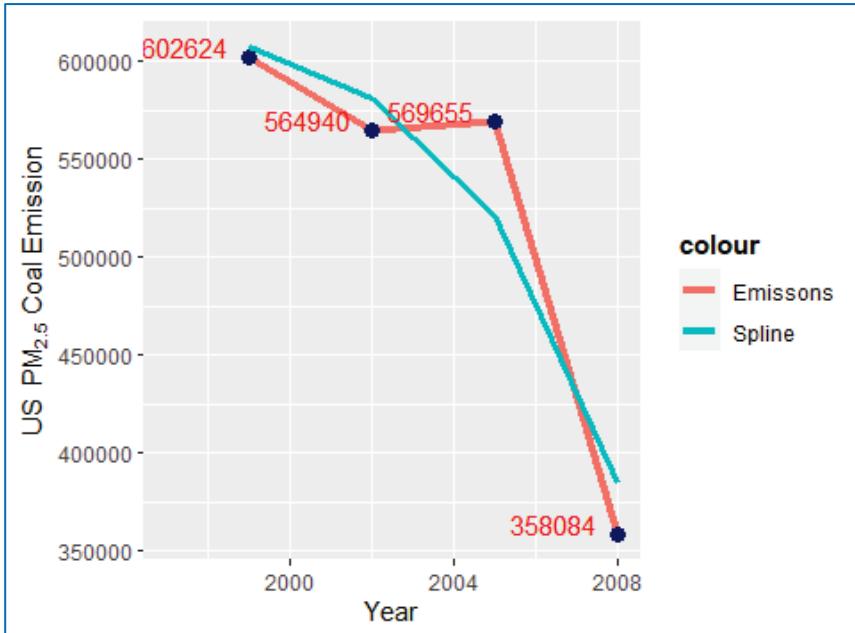
Next, we'll add the line-point-spline with aesthetics. The spline function projects points between data years in the context of the total interval (1999-2008) to form a trend-line. This is different than fitting a line for the entire plot from 1999-2008.

```
g+geom_line(size = 1.5) +
  geom_point(col = "navyblue", size = 3) +  xlim(1997,
2008) +
```

```

geom_spline(aes(x = year, y = Emissions, color =
"Spline"), size = 1.25) +
  geom_text(aes(label=round(Emissions,0)), hjust=1.25,
vjust=0, col = "red") +
  ggtitle(expression("Total" ~ PM[2.5] ~ "Coal Emission by
Year Across the United States")) +
  xlab("Year") +
  ylab(expression("US " ~ PM[2.5] ~ "Coal Emission")) +
  theme(legend.title = element_text(face = "bold"))

```



**Figure 3-13.** Total PM<sub>2.5</sub> coal emission by year across units with overlaid tend line (spine).

#### Conclusion to Example 3-9:

Across the United States, emissions from coal combustion-related sources have decreased from 1999-2008, and the overall trend is negative (decreasing) and seen by the spline in plot4.png.

#### Example 3-10 – Motor Vehicle Emissions

How have emissions from motor vehicle sources changed from 1999-2008 in Baltimore City?

### *Solution to Example 3-10*

Construct a dataset comprised of emissions data for Baltimore City (FIPS Code 24510) from 199 to 2008, aggregated by year and summed.

```
baltimoreMotor <- subset(NEI, NEI$fips == "24510" &
NEI$type == "ON-ROAD")
baltimoreMotorAGG <- aggregate(Emissions ~ year,
baltimoreMotor, sum)
```

### *Build the Barplot*

We construct a bar chart in **Figure 3-14** comprised of four years (one bar for each) with summed emissions data for Baltimore City. First, we define a base plotting object “g” using ggplot.

```
g <- ggplot(baltimoreMotorAGG, aes(year, Emissions))
```

Next, we add geom functions to augment the plot and add aesthetics

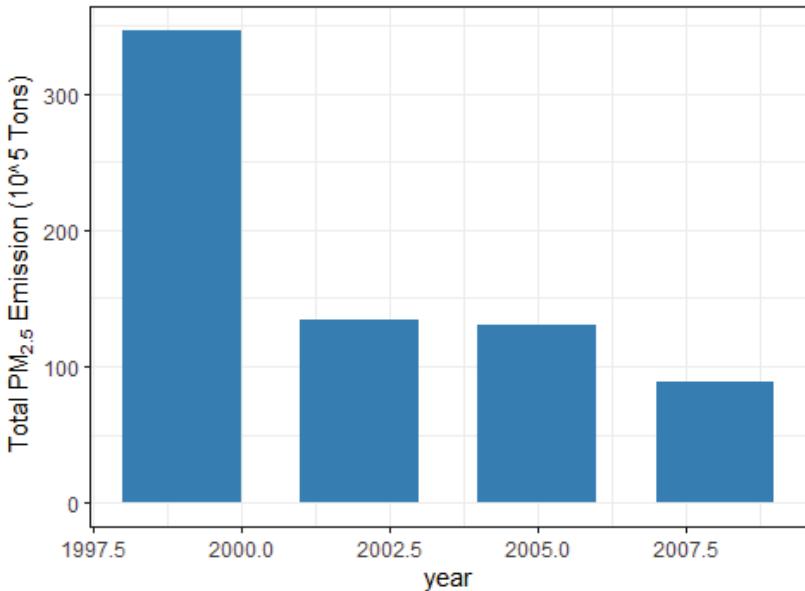
```
g + geom_bar(stat="identity", fill="steelblue", width=2) +
  theme_bw() +      # Add theme
  guides(fill=FALSE) +  # Set guides for scale to FALSE
  labs(x="year", y=expression("Total PM"[2.5]*" Emission
(10^5 Tons)")) + # Add x and y axes labels
  labs(title=expression("PM"[2.5]*" Motor Vehicle Source
Emissions in Baltimore from 1999-2008")) # Add main title
```

### *Conclusion to Example 3-10:*

Total emissions from motor vehicle sources in Baltimore City have decreased from 1999-2008, with a clear downward trend.

## **Example 3-11 - Vehicle Emission Sources**

Here, we compare emissions from motor vehicle sources in Baltimore City with emissions from motor vehicle sources in Los Angeles County, California (fips == “06037”). Which city has seen greater changes over time in motor vehicle emissions?



**Figure 3-14.** PM<sub>2.5</sub> motor vehicle emission in Baltimore by total emission

### Solution to Example 3-11

```
vehiclesNEI <- subset(NEI, NEI$fips %in%
c("24510","06037") & NEI$type == "O`N-ROAD")
```

We use a filter to get vehicle data from SCC. Level 2 is classified as industrial category sources, like industrial solid waste disposal or commercial marine vessels.

```
SCC_Vehicles <- SCC %>%
  filter(grepl('Vv]ehicle', SCC.Level.Two)) %>%
  select(SCC, SCC.Level.Two)
```

Now, we get the location of emissions data

- Filter the data to get Baltimore City (FIPS code = "24510") and Los Angeles County (FIPS Code "06037")
- Select the variables "fips", "SCC", "Emissions", and "year" Inner-join the data set with SCC\_Vehicle
- Group cities (fips) by year (four groups for each city)
- Select the variables Total Emissions, fips (Cities), and year.

```
Balt_LA_Emissions <- NEI %>%
  filter(fips == "24510" | fips == "06037") %>%
  select(fips, SCC, Emissions, year) %>%
  inner_join(SCC_Vehicles, by = "SCC") %>%
  group_by(fips, year) %>%
  summarise(Total_Emissions = sum(Emissions,
    na.rm = TRUE)) %>%
  select(Total_Emissions, fips, year)
```

`summarise()` has grouped output by '`fips`'. You can change this by using the `.groups()` argument.

To prepare the data for constructing a barplot, we merge the emission data into one superset

```
Balt_LA_Emissions$fips <- gsub("24510", "Baltimore City",
Balt_LA_Emissions$fips)
Balt_LA_Emissions$fips <- gsub("06037", "Los Angeles
County", Balt_LA_Emissions$fips)
```

Now, we construct a set of bar charts, one for Baltimore City and one for Los Angeles County, comprised of four years (one bar for each) with summed emissions. To do this, we first define a base plotting object "g" using `ggplot()`.

```
g <- ggplot(Balt_LA_Emissions, aes(x = factor(year), y =
Total_Emissions, fill = fips))
```

Get a description of the structure of the base graphic object. To decide what plotting functions to apply, we need to explore the structure of the plotting object, `g`.

```
str(g)

## List of 9
## $ data: grouped_df [8 x 4] (S3:
grouped_df/tbl_df/tbl/data.frame)
##..$ Total_Emissions: num[1:8] 6110 7189 7304 6421 404...
##..$ fips: chr [1:8] "Los Angeles County" "Los Angeles County"
"Los Angeles County" "Los Angeles County" ...
## ..$ year: int [1:8] 1999 2002 2005 ...
## ..$ .group: int [1:8] 2 2 2 2 1 1 1 1
## -attr(*,"groups")=tibble[2x2](S3:tbl_df/tbl/data.frame)
## ..$ fips: chr[1:2]"Baltimore City" "Los Angeles County"
## ..$ .rows: list<int> [1:2]
```

```

## ... .$. : int [1:4] 5 6 7 8
## ... .$. : int [1:4] 1 2 3 4
## ... .@ ptype: int(0)
## ... - attr(*, ".drop")= logi TRUE
## $ layers      : list()
## $ scales: Classes 'ScalesList','ggproto','gg'<ggproto
##   object: Class ScalesList, gg>
## super: <ggproto object: Class ScalesList, gg>
## $ mapping     :List of 3
## ... $.x     : language ~factor(year)
## ... -attr(*,".Environment")=<environment: R_GlobalEnv>
## ... $.y     : language ~Total_Emissions
## ... -attr(*,".Environment")=<environment: R_GlobalEnv>
## ... $.fill: language ~fips
## ... -attr(*,".Environment")=<environment: R_GlobalEnv>
## ... - attr(*, "class")= chr "uneval"
## $ theme       : list()
## $ coordinates: Classes 'CoordCartesian', 'Coord',
## 'ggproto','gg'<ggproto object: Class CoordCartesian, Coord, gg>
## super: <ggproto object:Class CoordCartesian,Coord, gg>
## $ facet:Classes 'FacetNull', 'Facet', 'ggproto', 'gg'
##   < ggproto object: Class FacetNull, Facet, gg>
##   params: list
##   shrink: TRUE
##   train_scales: function
##   vars: function
##   super: <ggproto object: Class FacetNull, Facet, gg>
## $ plot_env    :<environment: R_GlobalEnv>
## $ labels      :List of 3
## ... $.x     : chr "factor(year)"
## ... $.y     : chr "Total_Emissions"
## ... $.fill: chr "fips"
## - attr(*, "class")= chr [1:2] "gg" "ggplot"

```

Use the basic `ggplot` to generate a bar chart in **Figure 3-15** for emission from each location, and add aesthetics to the basic plot, including:

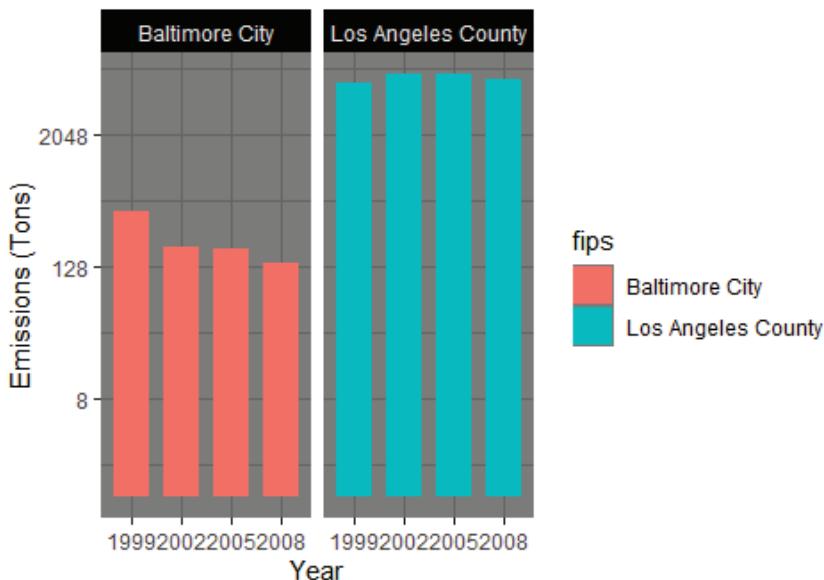
- plot theme
- main title label
- main subtitle label
- x-axis and y-axis labels
- `facet_grid()` forms a matrix of panels defined by row and column faceting variables.

- binary logarithm transformation of the y-axis for scaling effect

```
g + geom_bar(stat = "identity", width = 0.7) +
  facet_grid(.~fips) + scale_y_continuous(trans='log2') +
  labs(x = "Year", y = "Emissions (Tons)",
       title = "Comparison of Motor Vehicle Related Emissions",
       subtitle = "Between Baltimore City and Los Angeles From 1999 - 2008") +
  theme(plot.title = element_text(size = 14),
        plot.subtitle = element_text(size = 14),
        axis.title.x = element_text(size = 12),
        axis.title.y = element_text(size = 12),
        strip.text.x = element_text(size = 12)) +
  theme_dark()
```

### Comparison of Motor Vehicle Related Emissions

Between Baltimore City and Los Angeles From 1999 - 2008



**Figure 3-15.** Bar charts for a comparison between emission in Baltimore versus emission in Los Angeles

Now, we save the plots using `ggsave()`

```
ggsave("plot6.png", width = 20, height = 10, units = "cm")
ggsave("plot6_log.png", width = 20, height = 10, units =
"cm")
```

### ***Conclusion to Example 3-11:***

Comparing emissions from motor vehicle sources in Baltimore City with emissions from motor vehicle sources in Los Angeles County, California are as follows:

- Los Angeles County has seen significant change in scale than Baltimore City.
- However, while not at the same scale, Baltimore city has seen a decline in vehicle emissions.
- While Los Angeles County has seen greater changes over time in motor vehicle emissions, the 2008 level is high than the 1999 level.
- The picture is clearer in plot6\_log.png with the y-axis logarithm transformation, with Baltimore City seeing greater downward changes over time.

### **Markdown Note.**

Markdown provides an authoring framework for data science. You can use a single R Markdown file to both

- save and execute code
- generate high quality reports; share with an audience

R Markdown documents are fully reproducible and support dozens of static and dynamic output formats, including Word, HTML, PowerPoint, and more. Like the rest of R, R Markdown is free and open source. You can install the R Markdown package from CRAN with:

```
install.packages("rmarkdown")
```

## 4. GitHub

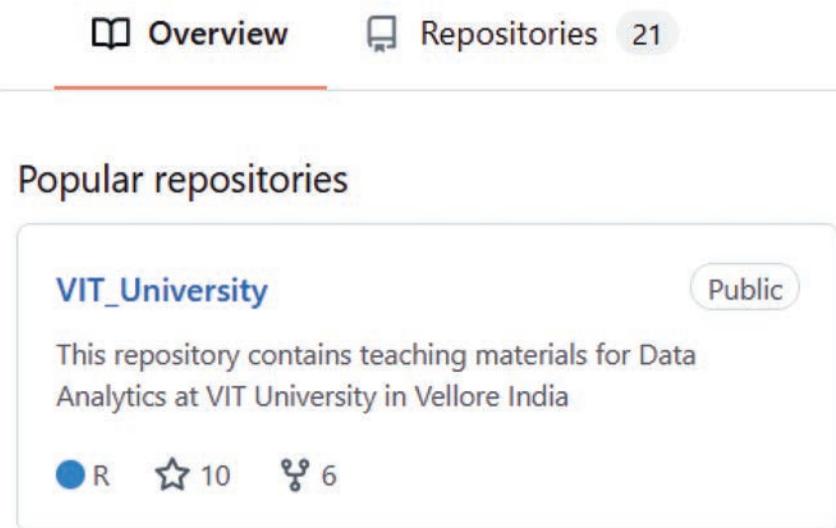
Let's assume you are not a user of GitHub or perhaps you are a user but have not used it in conjunction with R. GitHub is comprised of two elements Git and Github. Git is software for tracking changes in any set of files, usually used for collaborating work in developing source code during software development. GitHub is a web-based version of Git but has additional features.

### Using GitHub

Many R developers use GitHub to keep app and package source code for configuration control and peer review. To use GitHub you'll need to set up a GitHub account at <https://github.com>.

### Getting Started with GitHub

The first feature of GitHub is the repositories. I currently have 20-plus repositories, as shown in **Figure 4-1**. I generated this repository in 2018 as a resource for a data analytics course I taught at the Vellore Institute of Technology, Vellore India, that year. I put all of the code and data I used in this repository.

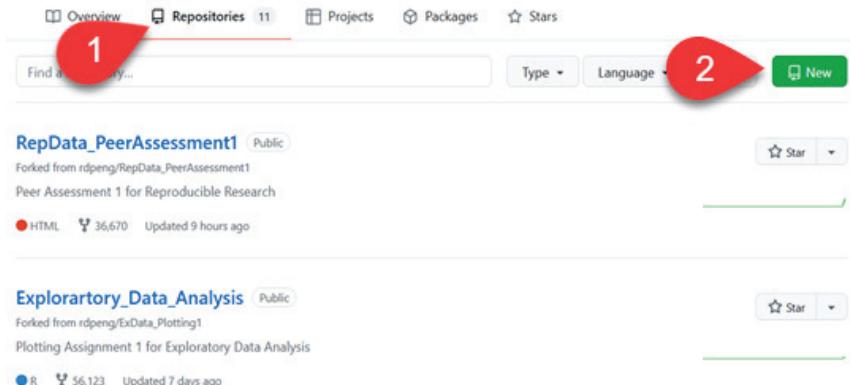


**Figure 4-1.** Example of a GitHub repository

To start, I will assume you can discover how to get an account (it's free), and refers you to, In GitHub:

1. click on the Repositories tab
2. then click on the New icon

Next, refer to **Figure 4-2**, where you should see, "Create a new repository."



**Figure 4-2.** Setting up a GitHub repository or “repo” steps 1 and 2.

3. in the dialog box, give the repository a name. This makes the repository url:

4. select whether you want the repository to be Public or Private. Since the point is sharing “stuff” with the GitHub community, all of my repositories are public. Also, I do not use it for storing personal items (I use Dropbox for that). Now refer to **Figure 4-3**.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Owner \*      Repository name \*

 stricje1 ✓ / R\_Programming ✓ 3

Great repository names! R\_Programming is available. Need inspiration? How about [turbo-waddle?](#)

Description (optional)

 Public 4  
Anyone on the internet can see this repository. You choose who can commit.

 Private  
You choose who can see and commit to this repository.

**Figure 4-3.** Naming the repository steps 3 and 4

5. Select “Add a README file (**Figure 4-4**), which will be generated in the next step. You can edit it later, or generate a replacement using R markdown.
6. Click on “Create repository”

### Initialize this repository with:

Skip this step if you’re importing an existing repository.

- Add a README file 5

This is where you can write a long description for your project. [Learn more.](#)

- Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

- Choose a license

A license tells others what they can and can’t do with your code. [Learn more.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

**Create repository** 6

**Figure 4-4.** Generating a README file and creating the repository Steps 5 and 6.

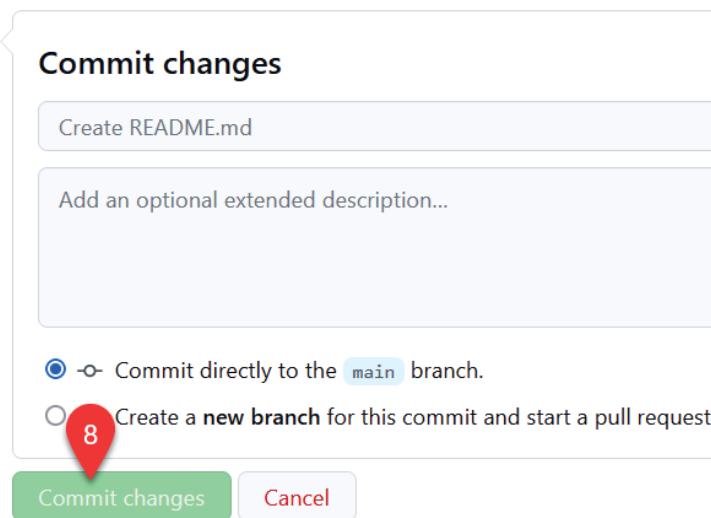
7. After step 6, you see something like **Figure 4-5**. It shows the README file in editing mode, which will be blank. I entered a little bit of text that I'll complete later using markdown.

The screenshot shows a GitHub repository named 'stricje1 / R\_Handbook\_Project'. The 'Code' tab is selected. A red circle with the number '7' is placed over the 'Edit file' button. The file path 'R\_Handbook\_Project / README.md' is shown, along with a 'main' branch indicator. The code editor contains the following text:

```
1 # R_Handbook_Project
2 Data and Code supporting my new book, R Handbook for Data Scientists.|
```

**Figure 4-5.** Step 7, README editing

8. Referring to **Figure 4-6**, scroll down and select Commit Changes



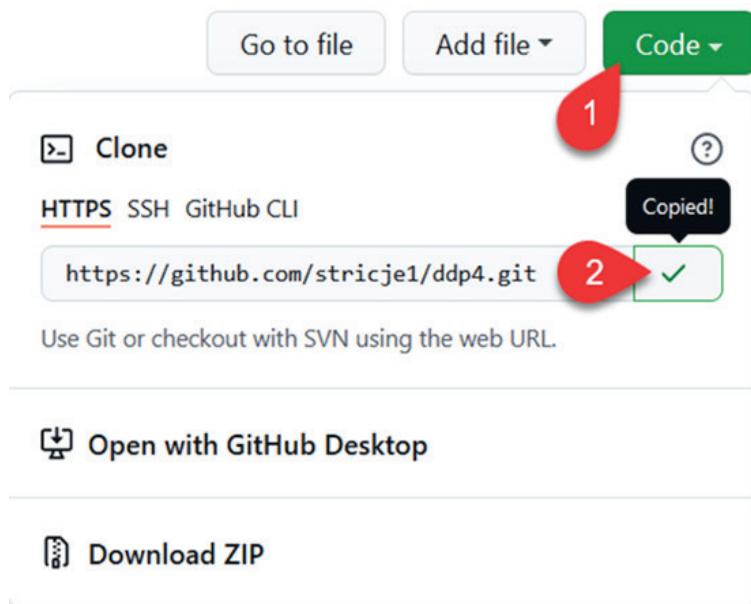
**Figure 4-6.** Committing changes to the README file

## Creating R Projects

So, you have now created a GitHub repository. Now let's use R and GitHub together, starting with an R project. There are really two reason we want to do this. First, we want to be able to publish some of our work, say a blog. Second, we want control configuration (version control). If you are currently emailing documents and code for collaboration, stop. You will lose control of content, rapidly. Third, we want to engage in peer collaboration.

For our first step, we need to copy the link to our repository as shown in **Figure 4-7**.

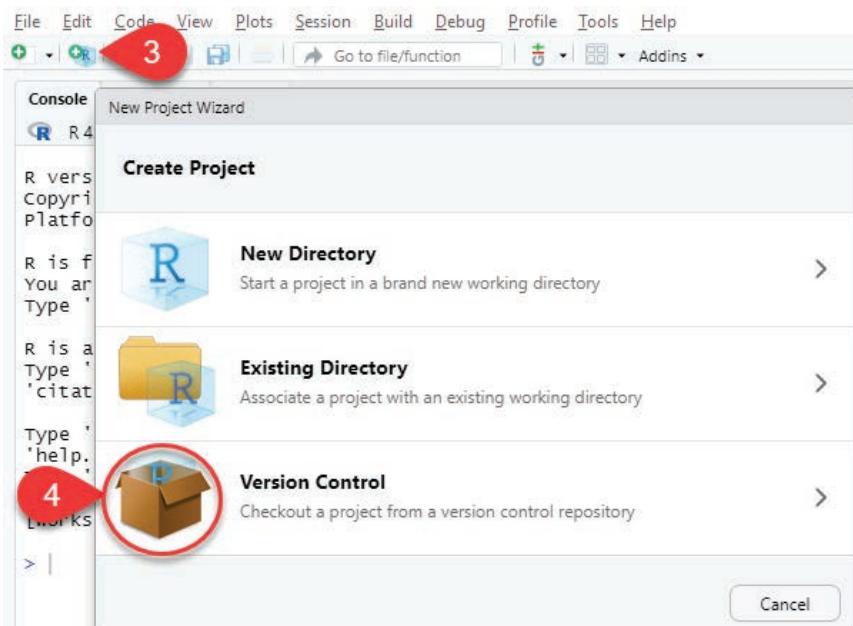
1. Click on the Code button in the main repository window.
2. Click on the copy button and it changes to a green checkmark indicating the copy was made



**Figure 4-7.** Copying the repository link for connecting R, steps 1 and 2

To maximize the “R – GitHub Advantage,” we’ll want to create an R project in RStudio, as soon will be apparent. Referring to Figure 4-8.

3. Click on the “Create a project.” This will open the “Create Project” dialog box.
4. There are three project choices. The first two choices, “New Directory” and “Existing Directory” are self-explanatory. The third choice is “Version Control.” Click on it.



**Figure 4-8.** Creating an R Project steps 1 and 2

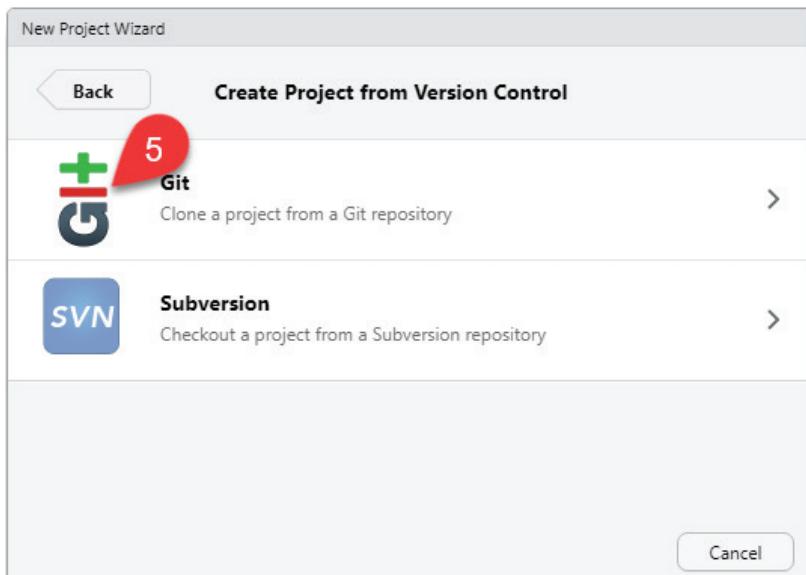
Version control creates a project. Version control help data scientist and team manage changes to their R source code over time. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members. Version control systems have been around for a long time but continue to increase in popularity with data science workflows.

5. After selecting a version control project, we are presented an option to connect it to Git or Subversion, as shown in **Figure 4-9**.

Remember the symbol for Git as you'll see it again in R (a smaller version, like  ).

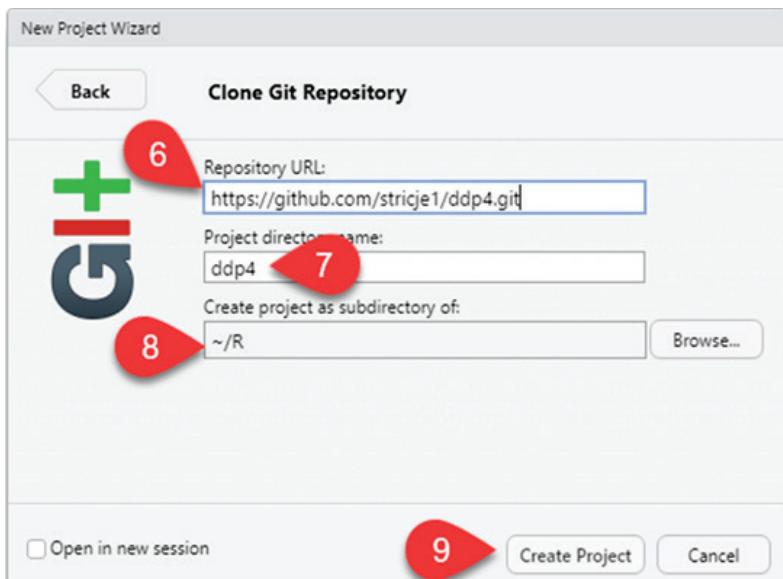
Now we will have an opportunity to use the directory path we copied. Referring to **Figure 4-9**:

6. Paste the directory link in the Repository URL box.
7. Enter a directory name in the Project directory name box



**Figure 4-9.** Clone a project from a Git repository

8. Notice the root directory is the project directory. You can keep the project as is or Browse to select a different directory (see **Figure 3-10**)



**Figure 4-10.** Connecting to a GitHub repository

9. Click on the Create Project button as shown in **Figure 4-11**.  
That completes the process.

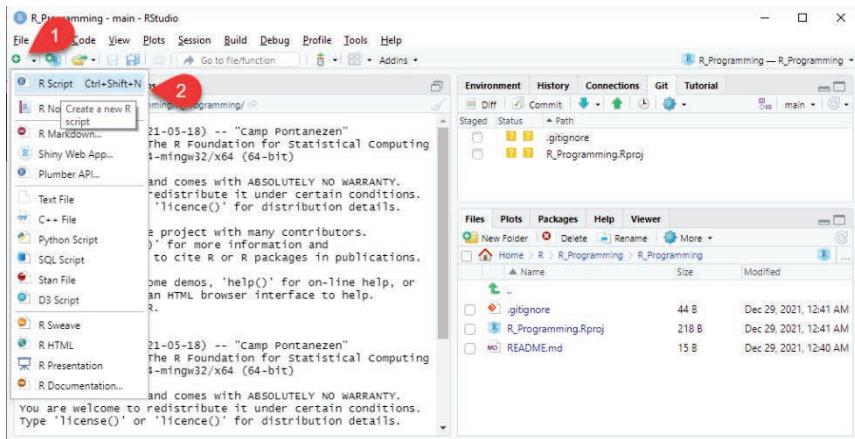
We can go to the root directory we chose or created to verify that it exists.

Name	Date modified	Type	Size
.git	1/15/2022 9:07 PM	File folder	
.Rproj.user	1/15/2022 7:02 PM	File folder	
.gitignore	1/15/2022 7:02 PM	Text Document	1 KB
ddp2	1/15/2022 8:56 PM	R Project	1 KB
README	1/15/2022 7:02 PM	Markdown Source...	1 KB

**Figure 4-11.** Working directory files

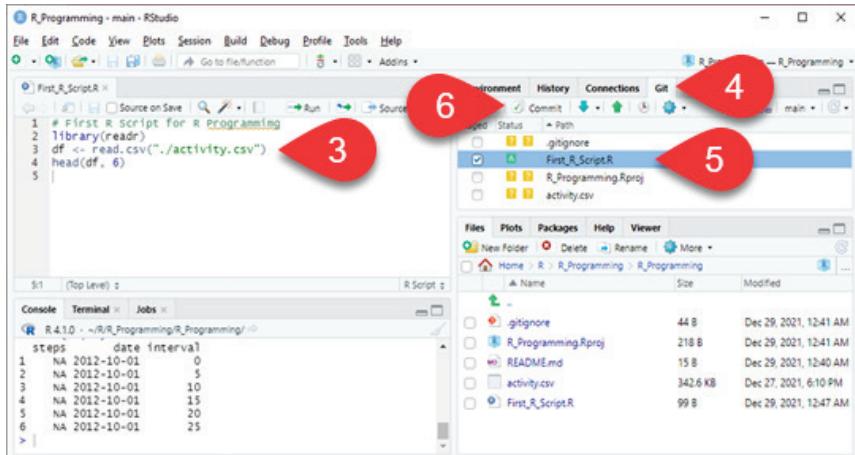
## Pushing files from RStudio

Now we need create a script in RStudio (**Figure 4-12**). Here we'll use the new file button (1) to create a new script (2).



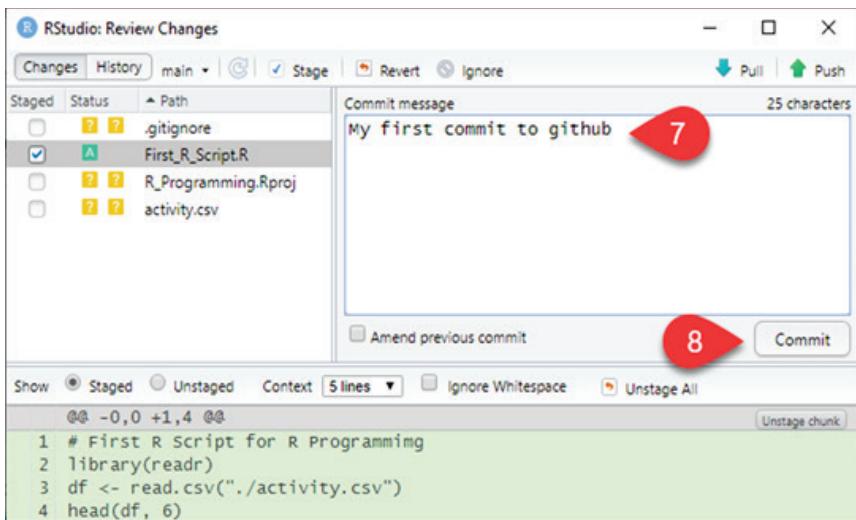
**Figure 4-12.** RStudio showing creating an R script file with new file button

Once the new script opens, we can enter some text in it, just enough to push it to GitHub (**Figure 4-13**), after which we click on the Git tab (4), and select the script to push (5). Next, we click on the Commit button (6).



**Figure 4-13.** Creating a script and pushing it to a GitHub repository

Next, a commit message window will appear (Figure 33) and we enter a simple commit message that will appear in the commit log in GitHub (7). Then we click on the Commit button (8).

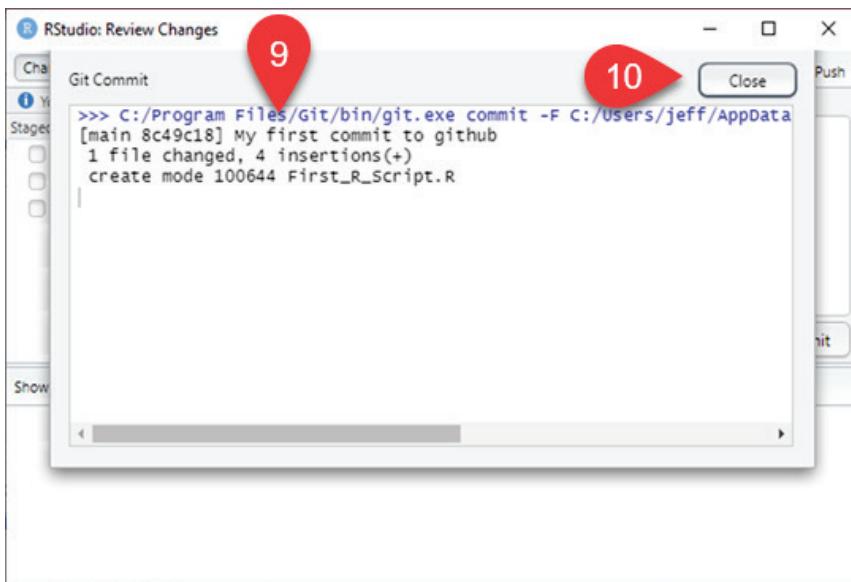


**Figure 4-14.** Committing a file from RStudio to GitHub using RStudio’s Git Interface

Note that we could also do this with Git bash, the terminal interface for Git.

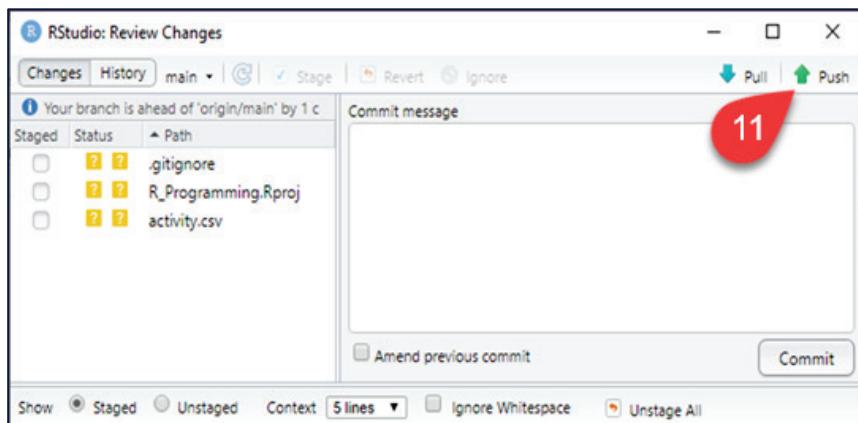
Once we commit a Git commit window (**Figure 4-15**) will pop up showing the status of the commit (9), and we can close the window (10).

**Markdown Note.** Markdown provides an excellent way to generate markdown-like readme documents. However, .Rmd does not render in GitHub like it does in Word or HTML. Instead, the markdown has to be saved as .md. A simple change of files extension does the trick.



**Figure 4-15.** Confirmation of committing a file

Once the Git Commit window closes, we click on the push button (11) as in **Figure 4-16**.



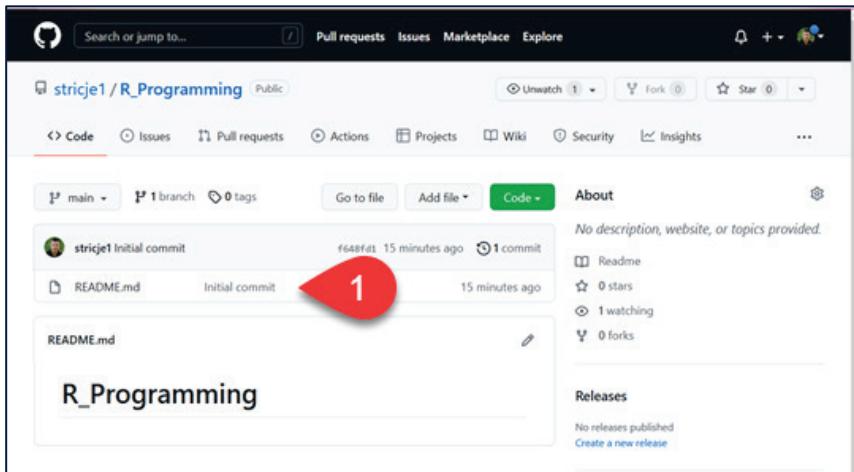
**Figure 4-16.** Pushing a committed file to GitHub

Next, the Git Push window (**Figure 4-17**) will open and a status message will appear (12). Barring errors, we can close the window (13).



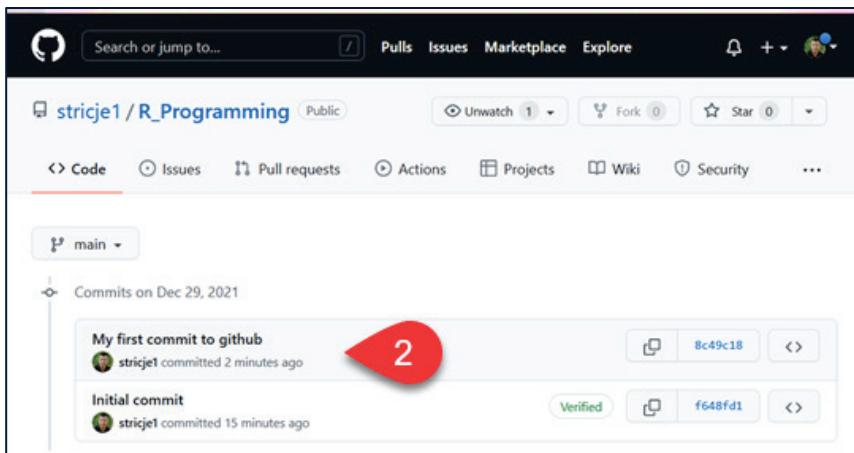
**Figure 4-17.** File push progress window

Next, we go to our GitHub repository (**Figure 4-18**) and we should see Initial Commit (1), although there may be some lag.



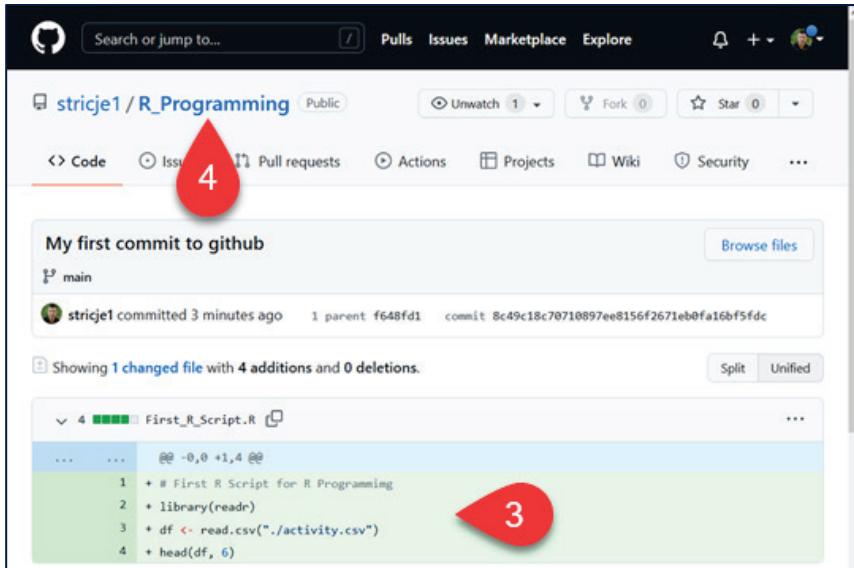
**Figure 4-18.** Initial commit in the GitHub repository

Once we see the Initial Commit message, we can open the commit log (**Figure 4-19**), and assuming this is our first commit, well see one entry (2) although there are two in my own.



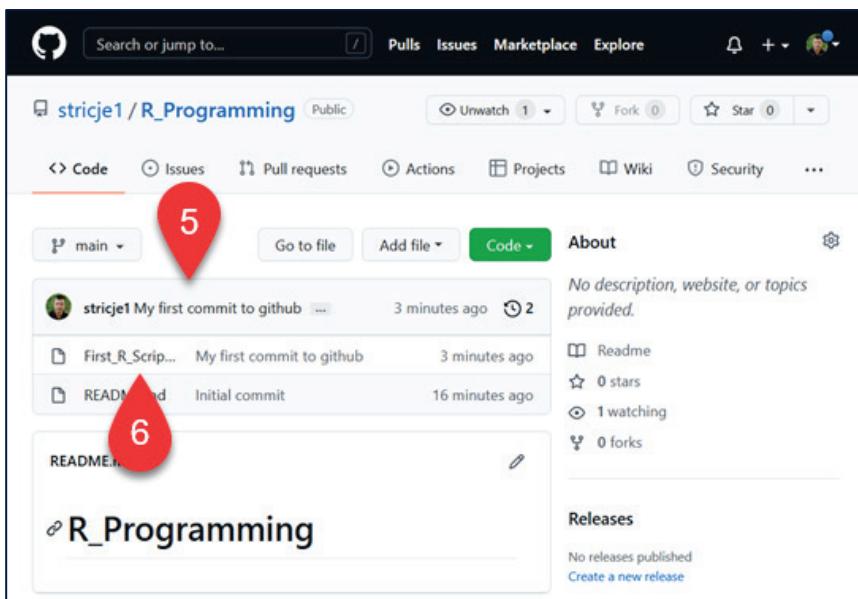
**Figure 4-19.** File commit window in GitHub

Opening the log entry (**Figure 4-20**), we should see the script (3) we wrote in RStudio. Next, we can go back to the main page of the repository (4).



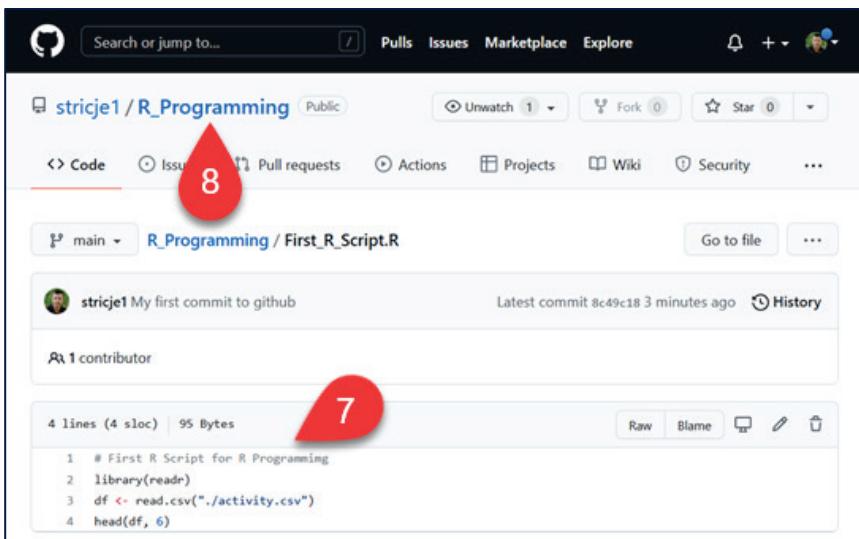
**Figure 4-20.** Committed script in GitHub

Back in the main repository window (**Figure 4-21**), there should be a message that refers to the first commit (5) at least two files, the script from RStudio and a readme file (6).



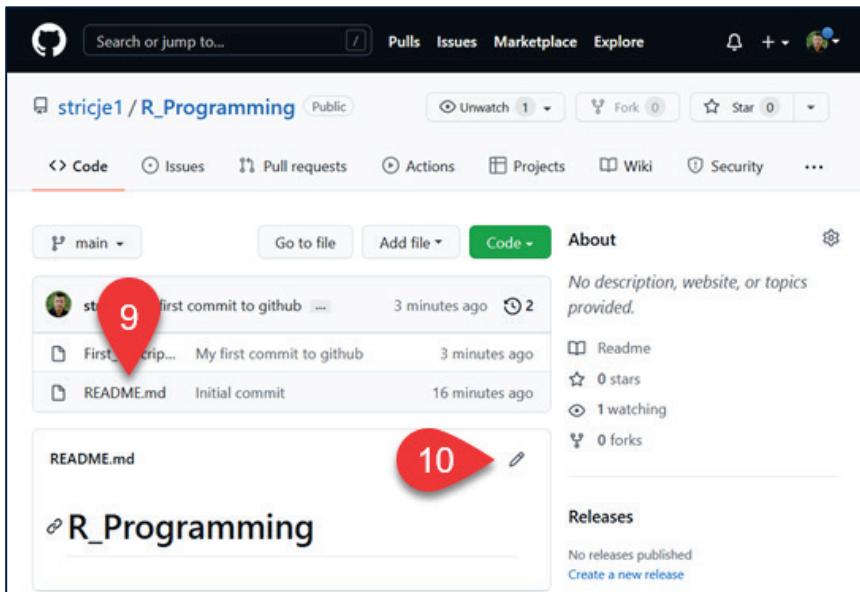
**Figure 4-21.** Script file in the GitHub repository

Now, we open the script we pushed to the repository (**Figure 4-22**) in the editing window (7), but we can go back the main repository (8) without editing.



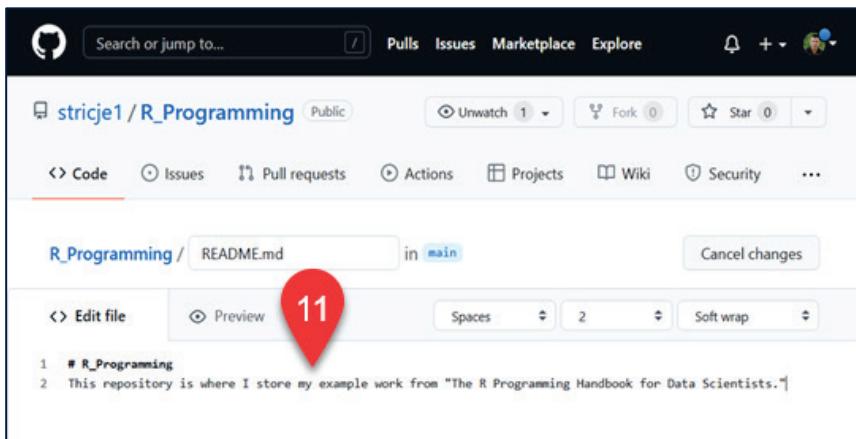
**Figure 4-22.** Script file open in GitHub

Next, we open the readme file (9) (**Figure 4-23**) and click on edit (10).



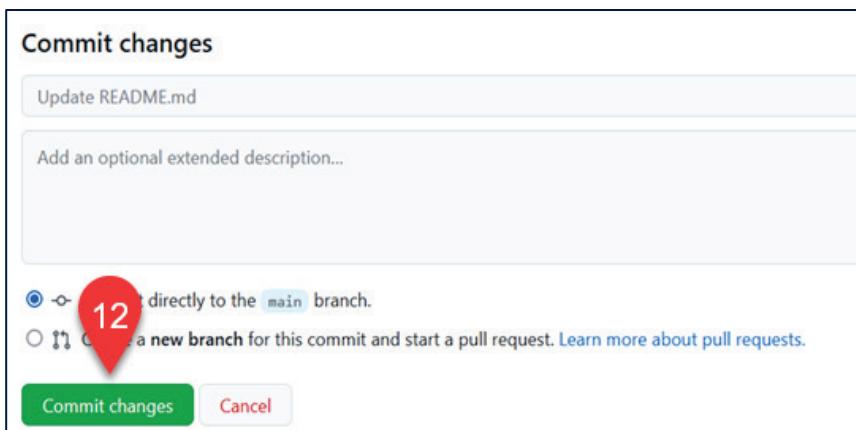
**Figure 4-23.** Editing a readme file in GitHub

We then write some text in the file (11) (Figure 4-24). We can always write more later.



**Figure 4-24.** Entering text in the *readme* file

Finally, we commit to changes (12) (Figure 4-25), that's like saving.



**Figure 4-25.** Committing changes to the *readme* file

Now we should be all set to use the basic aspects of GitHub with RStudio. Having this framework in our toolkit will facilitate our next task of generating reproducible research.

## 5. Reproducible Research

A basic idea is that in scientific research is replication. It is the most important element of verifying and validating findings scientists discover. If we claim that x causes y, or that Vitamin C reduces the occurrence of disease, then our claim will be scrutinized. That is, other scientist will independently want to investigate the claim to see if they come up with the same result. If a sufficient number of people arrive at the same result, then we can have more confidence that the original finding is probably true. Consequently, the ultimate standard in strengthening scientific evidence is replication.

The goal of replication research is to have independent scientists, using independent methods, with different data samples, and different laboratories, get the same result. Replication is particularly important in studies that have significant policy impacts or can influence regulatory types of decisions. However, with advances in technology and methodology, studies are becoming more difficult to replicate. Studies are getting larger, source data is getting bigger, research costs are increasing, and it is becoming more infeasible to replicate.

Thus, we are left with incurring the cost of replication or doing nothing. Or are we? There is middle ground, and this is the idea behind reproducible research is to create a minimum standard or a middle-ground. So, reproducibility can kind of bridge the gap between replication and nothing. The basic idea is we make available the data from the original study, and we make the computational methods available so that other people can look at our data and can run the kind of analyses that we've ran, and then come to the same conclusion. Because we're not collecting independent data using independent methods, we can, we can verify the question that we're asking.

So, by sharing our findings, our data, our methods, and other items, we are increase our chances that we arrived at our conclusions correctly (verification) and that the finding answer the analyses questions that were posed (validation). Likewise, if we can render reproducible results for someone else's research, the we are

contributing to the body of knowledge associated with the findings of the original research. This is how science is advanced.

Technological advances like parallel processing, cloud-based tools and algorithms, cloud storage, big data and so on, allow us to collect data at a much higher throughput to get very complicated and very high dimensional data sets, almost instantaneously, and mine the data, preprocess it, and explore it in the same environment in which we collect it, an environment that makes it easy to share it. And this same environment provides the means sharing complicated algorithms and implementing rigorous configuration control.

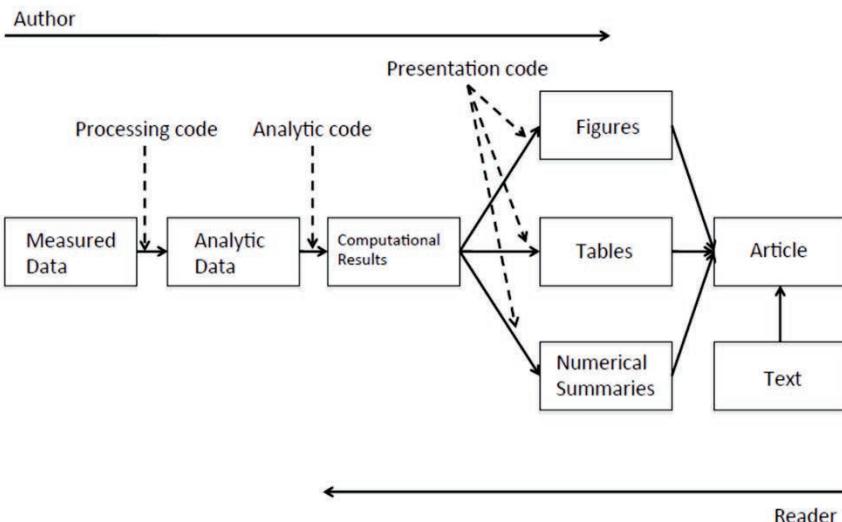
Consequently, scientific research is informing policy formation and decisions, regulatory requirements, and medical advances, for example. And the contributions to policies and regulations based on cost effective reproducible research outweigh the dwindling need for replication.

The basic issue here is that when you perform a study and present our findings, say in an article in literature or a research article, then we are obligated to provide our data, our methods, our algorithms, and even our code, so that others can reproduce our results. This is the idea of the Research Pipeline depicted in **Figure 5-1**.

On the left side of the pipeline, the author begins and moves to right. Then, we (the reader) start on the right and move to the left, as we read the article, examine its figures, tables, and summaries, uncover its code, computational results, its analytic data, and its source data. And we are able to reverse engineer the results, so to speak. In other words, the authors make transparent everything they used to reach their conclusion, so that we can economically trace their works and reproduce the results.

So, how do we make research reproducible. In context of this book, we do it with R programming. We say this because with R we have an integrated environment for mining data, writing code, rendering documents, producing apps, and performing configuration control.

# Research Pipeline



**Figure 5-1.** Research pipeline for reproducible research

## Literate (Statistical) Programming

One basic idea for generating reproducible research is known as Literate (Statistical) Programming. The idea is to think of an article, or a publication, or a report, as a stream of text and code. Then the generated text is readable by people and the code is readable by computers. Then the analysis is described in a series of text and code chunks. And each kind of code chunk will do something. It'll load some data and compute results. And each text chunk will relay something in a human readable language. So, there will be presentation code that formats tables and figures, and so on. This can be summarized as:

- An article is a stream of **text** and **code**
- Analysis code is divided into text and code “chunks”
- Each code chunk loads data and computes results
- Presentation code formats results (tables, figures, etc.)
- Article text explains what is going on

- Literate programs can be **weaved** to produce human—readable documents and **tangled** to produce machine—readable documents

Literate programming is a general concept that requires a documentation language (human readable) and a programming language (machine readable)

- knitr is a tool that turns machine readable language into human readable language
- knitr uses R as the programming language (although others are allowed) and variety of documentation languages – LaTeX, Markdown, HTML
- knitr was developed by Yihui Xie (while a graduate student in statistics at Iowa State)
- See <http://yihui.name/knitr/>

With knitr, we can use a variety of documentation languages. We can use LaTeX; we can also use something called Markdown and we can use HTML. The vast majority of this book was written with R markdown to word documents I used as student handout material and lecture notes. And all the data, code, and markdown are available in public GitHub repositories, so you can access the material and reproduce my work.

## Scripting Your Analysis

One sure way to make our work as reproducible as possible is that we script everything. In fact, the kind of file we use in R's text editor is called a Script. We write our code and document (comment on it) all within the Script file or files. Then we use the script to generate the markdown. We'll talk about the details as we progress through the book.

Scripting is really important because it's the basic idea of writing everything down. Back in the day we kept hand-written notes in volumes of notebooks for publishing papers, making lab notes, and so on. We wrote our procedures, our equations and formulas, and everything else that we could possibly write about to make our work reproducible. Now, we have integrated environments to do this in.

We have cloud environments with electronic notebooks, data search tools, analysis tools, programming tools, data lakes and data warehouses, and so on. We have Azure, Amazon Web Services (AWS), Google Web Services, Databricks, and RStudio, for example. We are like conductors of an orchestra comprised pf data woodwinds, brass notebooks, percussion code editors, and so on, coming together to make our research as repeatable as Beethoven's 5<sup>th</sup> Symphony, a musical script.

## Structure of a Data Analysis

In order to accomplish our task, we need a basic process by which our data analysis will unfold. So, we should define some steps that we can take to ensure reproducibility. Here are some steps to follow, not necessarily in this order:

1. Define the question
2. Define the ideal data set
3. Determine what data we can access
4. Try to fill the gap between the ideal and the actual
5. Obtain the data
6. Clean the data
7. Perform exploratory data analysis
8. Perform statistical prediction/modeling
9. Interpret results
10. Challenge results
11. Synthesize/write up results
12. Create reproducible code
13. Generate a codebook
14. Generate a notebook

In this chapter, we cover first seven steps, which we can think of as data pre-processing or data wrangling (recall Definition 3). We approach this using a primary example.

## Summary

- Reproducible research is important as a minimum standard, particularly for studies that are difficult to replicate

- Infrastructure is needed for creating and distributing reproducible documents, beyond what is currently available
- There is a growing number of tools for creating reproducible documents

## Example 5-1. Fitness

On Github, you can find the code and markdown in my RepData\_Project1 ([https://github.com/sticje1/RepData\\_Project1](https://github.com/sticje1/RepData_Project1)) repository. It is now possible to collect a large amount of data about personal movement using activity monitoring devices such as a Fitbit, Nike Fuelband, or Jawbone Up. These types of devices are part of the “quantified self” movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. But these data remain under-utilized both because the raw data are hard to obtain and there is a lack of statistical methods and software for processing and interpreting the data.

First, we load required packages

```
library(readr)
library(dplyr)
library(zoo)
```

First, we check the working directory, as well as download and unzip the data file:

```
path <- getwd()
```

- Set the filename to match “activity.csv”
- Check to see if already downloaded and unzipped working directory (WD)
- If filename is missing, download to the WD from the given URL
- Unzip the air\_pollution.zip into the WD
- activity.csv appears in the WD

```
filename = "activity.csv"
if (!file.exists(filename)) {
  urlzip <- "https://d396qusza40orc.cloudfront.net/repda%2Fdata%2Factivity.zip"
```

```

    download.file(urlzip, destfile = "./activity.zip" )
    unzip("./activity.zip", exdir = path )
}
cat("The working directory is ", path)

```

Recall that data wrangling is comprised of pre-processing tasks like obtaining, loading, cleaning, etc. In this code chunk we do the following:

- Set the path to match the working directory
- read filename that was unzipped above and indicate the header is present
- print the record of the activity file

```

activity_main <- read.csv("./activity.csv", header = TRUE
)
head(activity_main, 5)

##   steps      date interval
## 1   NA 2012-10-01       0
## 2   NA 2012-10-01       5
## 3   NA 2012-10-01      10
## 4   NA 2012-10-01      15
## 5   NA 2012-10-01      20

```

No, we make a copy of original data. It is always a good idea to not manipulate the original data file you upload (`activity_main`), so we make a copy:

```
activity <- activity_main
```

### *Explore the Data*

- Note that there are numerous missing values (NAs) from the file printouts
- Since imputation will be dealt with more thoroughly later, let's exclude the missing values for now
- Calculate the mean and total number of steps

```

activity <- activity_main[!is.na(activity[, 1]) , ]
act_count <- count(activity[, 0])
act_mean <- mean(activity[, 1])

```

```

act_sum <- sum(activity[, 1])
cat("Mean number of steps =", act_mean)
## Mean number of steps = 37.3826
cat("Total number of steps =", act_sum)
## Total number of steps = 570608
act_count
##      n
## 1 15264

```

Let's also look at the following functions that provide information about the dataset.

- **Dimension**

```

dim(activity)      # Dimension
## [1] 15264      3

```

- **Summary statistics**

```

summary(activity)  # Summary
##      steps          date        interval
##  Min.   : 0.00  Length:15264    Min.   : 0.0
##  1st Qu.: 0.00  Class :character  1st Qu.: 588.8
##  Median : 0.00  Mode  :character  Median :1177.5
##  Mean   : 37.38                    Mean   :1177.5
##  3rd Qu.: 12.00                    3rd Qu.:1766.2
##  Max.   :806.00                    Max.   :2355.0

```

- **Structure.** `str()` compactly display the internal structure of an R object, a diagnostic function and an alternative to `summary`. Ideally, only one line for each 'basic' structure is displayed. The structure that our R object is a dataframe. The `summary()` function does not provide the object type.

```

str(activity)      # Structure
## 'data.frame': 15264 obs. of 3 variables:
## $ steps  : int 0 0 0 0 0 0 0 0 ...
## $ date   : chr "2012-10-02" "2012-10-02" "2012-10-02"
## "2012-10-02" ...
## $ interval: int 0 5 10 15 20 25 30 35 40 45 ...

```

### *Mean Total Number of Steps per Day*

Now we answer the question, "What is mean total number of steps taken per day?"

First, we calculate the total number of steps per day, which is different than the calculation for total number of steps we did previously. Also, we will look ahead and note that we might want two options:

- one with missing values (our original activity data)
- one without missing values (`activity0`, the activity data excluding NAs)
- note that there are several ways to exclude the NAs
- using the `is.na()` function, which queries the data for NAs and record the value `TRUE` if or each `NA` found,
- **negating `is.na()`**, i.e., `!is.na` (we'll use this here),
- using `na.rm`, which is a logical value indicating whether NA values should be stripped before the computation proceeds, and
- using the `complete.cases()` function (we'll use this later), which is used to return a logical vector with cases which are complete, i.e., no `NAs`.

Note that by applying a filter `!is.na` as a filter, we can turn it off by commenting it out

```
steps_by_date<-select(activity, steps, date, interval) %>%
  filter(!is.na(steps)) %>%
# remove filter to include NAs as 0-step days in histogram
  group_by(date) %>%
  summarize(total_steps = sum(steps, na.rm = TRUE))
#has the same effect as !is.na
```

Second, using the total number of steps per day calculation, we can now compute the mean and median number of steps per day. Here, we assume that each date represents a day, and that there are multiple periods recorded each day, i.e., the 5-minute intervals.

### **Mean-Median number of Steps per day**

Here, we use the `complete.cases()` function as, previously mentioned to “remove” missing values

```

steps_by_date <- activity[complete.cases(activity),] %>%
  group_by(date) %>%
  summarize(total_steps=sum(steps))
steps_by_date

## # A tibble: 53 x 2
##   date      total_steps
##   <chr>        <int>
## 1 2012-10-02     126
## 2 2012-10-03    11352
## 3 2012-10-04    12116
## 4 2012-10-05    13294
## 5 2012-10-06    15420
## 6 2012-10-07    11015
## 7 2012-10-09    12811
## 8 2012-10-10    9900
## 9 2012-10-11   10304
## 10 2012-10-12   17382
## # ... with 43 more rows

```

### ***Calculate Mean and Median Total Steps per Day***

Next, we calculate and report the mean and median of the total number of steps taken per day. We compute them using the `mean()` and `median()` functions and name the computation, `MeanMedian_Raw`, for the unimputed data, and `MeanMedian_Imputed` (used later).

```

MeanMedian_Raw <-
  activity[complete.cases(activity),] %>%
  group_by(date) %>%
  summarize(average_steps=mean(steps),median_steps=median(
  steps)) %>%
  as.data.frame(MeanMedian_Raw)

mean_total_steps <- mean(steps_by_date$total_steps)
median_total_steps <- median(steps_by_date$total_steps)
cat("The mean of the total number of steps taken per day i
s ", mean_total_steps)
## The mean of the total number of steps taken per day is
10766.19
cat("The median of the total number of steps taken per day
is ", median_total_steps)

```

```
## The median of the total number of steps taken per day is
s 10765
```

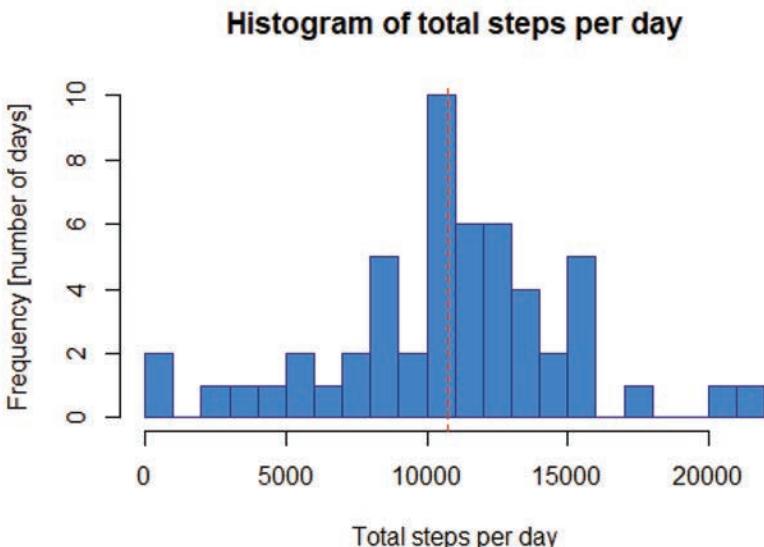
### Plot the histogram

Here, we use R's base function to plot a histogram of the total steps per day, as shown in **Figure 5-2**. We also add aesthetics, including

- A plot title (main)
- x and y labels
- customer colors
- breaks, which is a single number, treated as a suggestion as it gives pretty breakpoints

We also generate ablines using the `abline()` function, which can be used to add vertical, horizontal or regression lines to a graph. We add the vertical lines to the histogram we created earlier at the mean and median points. This also demonstrates how additional plotting objects can be added to an existing plot.

```
hist(steps_by_date$total_steps,
      main = "Histogram of total steps per day",
      xlab = "Total steps per day",
      ylab = "Frequency [number of days]",
      breaks = 20,
      border = "slateblue4",
      col = "dodgerblue"
)
abline(v=mean_total_steps, lwd = 1, lty = 2, col ="red")
abline (v=median_total_steps, lwd = 1, lty = 2, col ="red")
)
```



**Figure 5-2.** Histogram of total steps per day

## Average Daily Activity Pattern

Here, we answer the question, “What is the average daily activity pattern?”

To do so, we create a quasi-time-series plot (the plot is technical a scatterplot, as the activity data is not properly formatted with a time-based index, which we could do using the `xts`-package) as shown in **Figure 5-3**. However, we will apply a 100-interval moving average to investigate how we might impute missing values with a mean value, for example.

### Plot customization

We'll also customize the plot with the following aesthetics:

- A line type scatterplot using `type = "l"`
- a wider line width, using `lwd`
- a custom color, `royalblue4`, using `col`
- a plot title, using `main`

- x- and y-axis labels, with `xlab` and `ylab`, respectively
- the 100-period moving average line, and
- a customized legend

Finally, we will add vertical and horizontal lines, using `abline()`, representing the point where the maximum average steps occur.

```
steps_by_interval <- select(activity, steps, date, interval)
  %>%
  group_by(interval) %>%
  summarize(average_steps = mean(steps, na.rm = TRUE))
dim0 <- dim(steps_by_interval)
cat("The dimension of steps_by_interval is: ", dim0)

## The dimension of steps_by_interval is: 288 2
```

Which 5-minute interval, on average across all the days in the dataset, contains the maximum number of steps?

```
x <- steps_by_interval [steps_by_interval$average_steps==max(steps_by_interval$average_steps) ,1]
cat("The 5-minute interval with the maximum steps is: ", as.character(x))

## The 5-minute interval with the maximum steps is: 835
```

### *Time series plot*

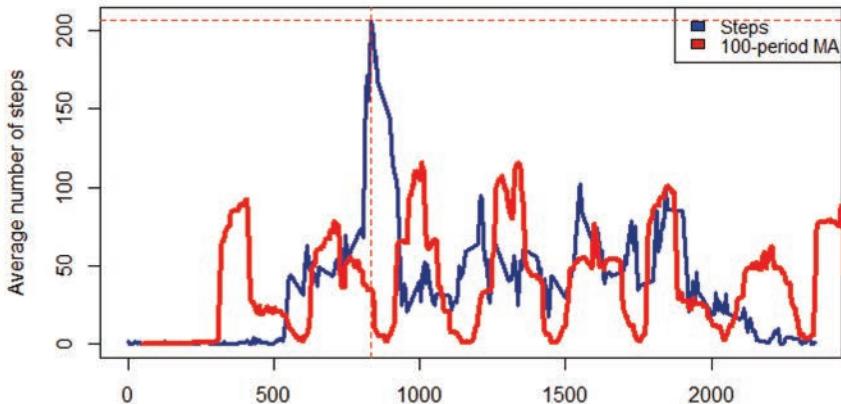
```
new_steps_100ma <- zoo::rollmean(activity["steps"], k = 100, fill = NA)
plot(steps_by_interval$interval, steps_by_interval$average_steps,
      type = "l", lwd = 3, col = 'royalblue4',
      main = 'Average Steps by 5-minute Interval', cex = .85,
      xlab = "Interval",
      ylab = "Average number of steps")
lines(new_steps_100ma, lwd = 4, col = 'red2')
legend('topright', deaths.xts,
       c('Steps', '100-period MA'),
       fill = c('blue2', 'red2', 'green2'), cex = .85,
       box.lty = 1)
max_average_steps <- max(steps_by_interval$average_steps)
```

```

max_average_steps_interval <- steps_by_interval[
  steps_by_interval$average_steps ==
  max_average_steps, ]$interval
abline (v = max_average_steps_interval, lwd = 1,
  lty = 2, col = "red")
abline(h = max_average_steps, lwd = 1, lty = 2,
  col = "red")

```

**Average Steps by 5-minute Interval**



**Figure 5-3.** Average steps by a 5-minute interval with moving average

#### *Sample of some averages in steps\_by\_interval:*

The moving average plot shows that there is an overall pattern to the daily total average steps, five or six periods. We can estimate this as  $2000/5 = 500$  periods. We might also consider the first 500 as a warmup period, to reach steady state in the 100-period MA, giving us points at 1000, 1500, 2000, and 2500

```

steps_by_interval [steps_by_interval$interval %in% c(1000,
1500,2000,2500),]

## # A tibble: 3 x 2
##   interval average_steps
##       <int>      <dbl>
## 1     1000      40.6
## 2     1500      30.0
## 3     2000      19.6

```

```

head(activity[which(is.na(activity$steps) & activity$interval %in% c(1000,1500,2000,2500)),], n = 30)

## [1] steps      date      interval
## <0 rows> (or 0-length row.names)

```

### *Imputing Missing Values*

Here, we investigate the *effect of imputing missing steps values*

Considering that steps are derived electronically, zeros are valid entries. Although the missing values, NAs, are curious, we will develop a strategy to impute them. Here, we can use the pattern we noted with the 100-period MA, in particular, (500, 1000, 1500, 2000, 2500), since zeros are valid.

### *Calculate number of missing values*

First, we calculate and report the total number of missing values, using the `is.na()` function.

```

x <- sum(is.na(activity[, 1]))
cat("Number of rows with missing values =",
    as.character(x))

## Number of rows with missing values = 0

```

### *Replace missing step values*

Here, we use the `mean()` function for a given interval, which we calculated above, to replace missing steps values (NAs). The object, `steps_by_interval`, contains means of steps for each interval (see above).

```

steps_by_interval [steps_by_interval$interval %in% c(500,1000,1500,2000),]

## # A tibble: 4 x 2
##   interval average_steps
##       <int>        <dbl>
## 1      500          0
## 2     1000         40.6
## 3     1500         30.0
## 4     2000         19.6

```

```
steps_by_interval <- as.data.frame(steps_by_interval)
```

### **Create object to hold imputed values**

Next, we create an object to hold activity and imputed activity.

```
Imputed <- activity
```

### **Find and replace missing values**

Next, we find rows with missing (NA) data and replace them with the mean we calculate for the given for the intervals.

```
for (i in 1:nrow(Imputed)) {  
  if (is.na(Imputed$steps[i])){  
    n <- steps_by_date[steps_by_date$interval==Imputed$interval[i],average_steps]  
    Imputed$steps[i]<- as.integer(round(n))  
  }  
}
```

### **Define comparison data**

Now, we are ready to compare the data we just imputed with the work accomplished earlier. We proceed as follows.

- Populate the Imputed object with NAs imputed
- Redefine the data with NAs present
- Redefine the data with NAs removed
- Generate histograms for each set of data. We show these in **Figure 5-4**

```
steps_by_date_imp <- Imputed %>%  
  group_by(date)  %>%  
  summarize(imp_total_steps=sum(steps))  
  
steps_by_date_na <- select(activity_main, steps, date,  
  interval)  %>%  
  #filter(!is.na(steps)) %>% # removed filter to include N  
As as 0-step days in histogram  
  group_by(date)  %>%  
  summarize(total_steps = sum(steps, na.rm = TRUE))  
steps_by_date <- select(activity, steps, date,
```

```

    interval) %>%
  filter(!is.na(steps)) %>% # removed filter to include NA
  s as 0-step days in histogram
  group_by(date) %>%
  summarize(total_steps = sum(steps, na.rm = TRUE)) # has
  the same effect as `!is.na`  

  par(mfrow=c(1,2))
  hist(steps_by_date$total_steps,
    main = "Histogram of Imputed Total Steps per Day (NAs
  Removed)",
    xlab = "Total steps per day",
    ylab = "Frequency [number of days]",
    cex.main = 1.1,
    cex.lab = 1.1,
    breaks = 20,
    border = "cadetblue4",
    col = "cadetblue3"
)
  hist(steps_by_date_na$total_steps,
    main = "Histogram of Total Steps per Day",
    xlab = "Total steps per day",
    ylab = "Frequency [number of days]",
    cex.main = 1.1,
    cex.lab = 1.1,
    breaks = 20,
    border = "dodgerblue4",
    col = "dodgerblue3"
)

```

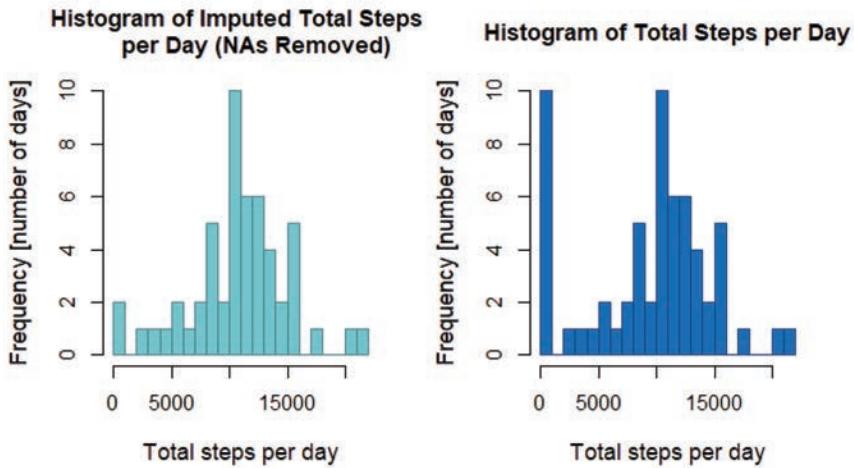
### Markdown Note.

This is a paragraph in an R Markdown document. Below is a code chunk that produces a plot output that follows:

```

```{r echo=TRUE}
plot(gg_dta, type="cum_haz") +
  labs(y = "Cumulative Hazard",
       x = "Observation Time (Years)",
       color = "Treatment", fill = "Treatment") +
  theme(legend.position = c(.2,.8))
```

```



**Figure 5-4.** Histograms of total steps per day versus imputed steps per day

```
par(mfrow=c(1,1))
```

The first two histograms demonstrate the imputation and removal of the NAs present in the third histogram. Zeros are still present, since zero is a valid entry.

### Calculate and Report the Mean and Median Total Steps per Day

Now we answer the question, “Are there differences in activity patterns between weekdays and weekends?”

```
MeanMedian_Imputed <-
  Imputed %>%
  group_by(date) %>%
  summarize(average_steps=mean(steps), median_steps=median(
    steps))
MeanMedian_Imputed

## # A tibble: 53 x 3
##   date      average_steps median_steps
##   <chr>          <dbl>        <dbl>
## 1 2012-10-02      0.438         0
## 2 2012-10-03      39.4          0
## 3 2012-10-04      42.1          0
## 4 2012-10-05      46.2          0
```

```

## 5 2012-10-06      53.5      0
## 6 2012-10-07      38.2      0
## 7 2012-10-09      44.5      0
## 8 2012-10-10      34.4      0
## 9 2012-10-11      35.8      0
## 10 2012-10-12     60.4      0
## # ... with 43 more rows

```

Do these values differ from the estimates from the first part of the example?

There is no significant difference

`summary(MeanMedian_Raw, 2:3)`

```

##       date      average_steps      median_steps
##  Length:53      Min.    : 0.1424      Min.    :0
##  Class :character 1st Qu.:30.6979  1st Qu.:0
##  Mode  :character Median  :37.3785  Median  :0
##                  Mean   :37.3826  Mean   :0
##                  3rd Qu.:46.1597 3rd Qu.:0
##                  Max.    :73.5903  Max.    :0

```

`summary(MeanMedian_Imputed, 2:3)`

```

##       date      average_steps      median_steps
##  Length:53      Min.    : 0.1424      Min.    :0
##  Class :character 1st Qu.:30.6979  1st Qu.:0
##  Mode  :character Median  :37.3785  Median  :0
##                  Mean   :37.3826  Mean   :0
##                  3rd Qu.:46.1597 3rd Qu.:0
##                  Max.    :73.5903  Max.    :0

```

### *Weekend and Weekday Activity Patterns*

Now, we answer the question, “Are there differences in activity patterns between weekdays and weekends?”

For this part the `weekdays()` function may be of some help here. Use the dataset with the filled-in missing values for this part. Create a new factor variable in the dataset with two levels - “weekday” and “weekend” indicating whether a given date is a weekday or weekend day.

## Create the names

First, we define the names of the days in a week and corresponding descriptors, weekday and weekend. We also attend to the necessary data imputations along the line of our previous work.

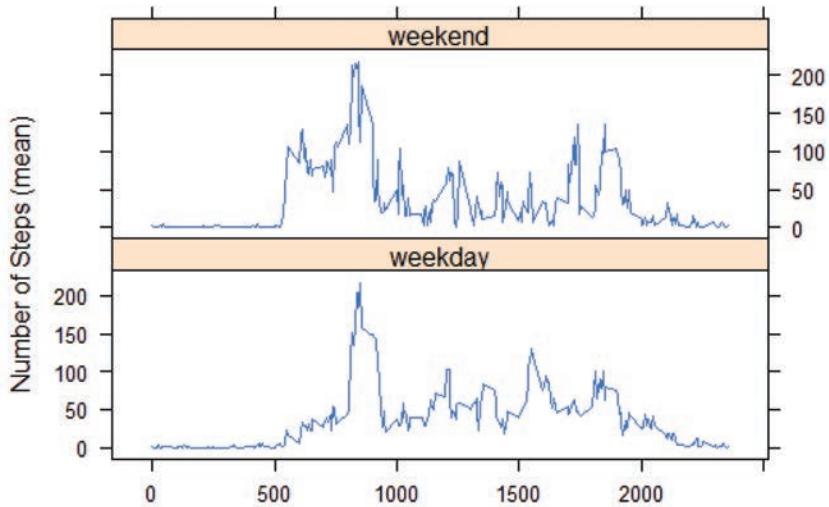
```
DayOfWeek <- data.frame(  
  name=c("Monday", "Tuesday", "Wednesday", "Thursday",  
    "Friday", "Saturday", "Sunday"),  
  part=c("weekday", "weekday", "weekday", "weekday",  
    "weekday", "weekend", "weekend"))  
  
Imputed$date <- as.Date(Imputed$date)  
  
## Create the column  
Imputed$WeekPart <- as.factor(weekdays(Imputed$date))  
## Set the value  
Imputed$WeekPart <- DayOfWeek[Imputed$WeekPart, 2]
```

## Make a panel plot

We also make a panel plot (see **Figure 5-5**) containing a time series plot (i.e., type = "l") of the 5-minute interval (x-axis) and the average number of steps taken, averaged across all weekday days or weekend days (y-axis). To do this, we:

- Calculate and store the means by group
- Construct a lattice plot for weekend and weekdays

```
PlotSummary <- Imputed %>%  
  group_by(WeekPart, interval) %>%  
  summarize(StepsMean=mean(steps))  
  
## Use the lattice graphics for easier multiple panels  
library(lattice)  
xyplot(PlotSummary$StepsMean ~ PlotSummary$interval | Plot  
  Summary$WeekPart,  
  data=PlotSummary, type = "l", layout=c(1:2),  
  ylab= "Number of Steps (mean)",  
  xlab= "Interval")
```



**Figure 5-5.** Mean number of steps per day weekdays versus weekends

Examination of the two plots indicate that the weekday and weekend activity is roughly the same over the same time intervals.

## Example 5-2. Severe Weather Effect

### Introduction

Storms and other severe weather events can cause both public health and economic problems for communities and municipalities. Many insurance companies use weather data for understanding the impacts of severe weather events in terms of fatalities, injuries, and property damage, and rates are based on their findings. For instance, some insurance companies will no longer cover property damage in Colorado due to the impact of wildfires and hail on property damage. While this analysis is focus on the overall storm affects across the United States, similar studies are regularly performed for states and regions. You can find all the material for this example in my RepData\_Project2 ([https://github.com/stricje1/RepData\\_Project2](https://github.com/stricje1/RepData_Project2)) repository.

### Analysis Questions and Outcomes

For this study, we have two analysis questions we want to answer:

1. Across the United States, which types of events (as indicated in the EVTYPE variable) are most harmful with respect to population health?

**Tornado** events have the greatest impacts on population health, both fatalities and injuries, and exceed other events at least three-fold.

2. Across the United States, which types of events have the greatest economic consequences?

For property damage, **flood** events (widespread) is overwhelmingly the leader, while the major cause of crop damage is drought.

## Data Processing

The Data acquired for this study comes from the U.S. National Oceanic and Atmospheric Administration's (NOAA) storm database. This data is available on NOAA's website at this link.

### *Install and Load Packages*

The code for installing packages not shown. The required packages are `captioner`, `readr`, `dplyr`, `knitr`, `kableExtra`, `ggplot2`, `grid`, `gridExtra`, and `ggplotify`.

### *Load the Weather Data*

Here, we load the dataset, using a conditional statement. The data used for the analysis is drawn from the U.S. NOAA storm database, as mentioned. The data for the analysis covers the period from 1950 to September 2011. The data can be downloaded from this link. The code we use first checks to see if the data has already been downloaded and unzipped to the working directory. If it has not, we generate a temporary file to hold the data that we download and unzip, thus saving disk space.

```
storm <- tempfile() # Create the temporary file
if (!file.exists(storm)){
  download.file
  ("http://d396qusza40orc.cloudfront.net/reldata%2FstormData.csv.bz2", storm)
  storm_data  read.csv (storm)
```

```
}
```

```
unlink(storm) # Release the temporary file
```

## Exploratory Data Analysis (EDA)

Here we perform EDA, as a part of our data processing or “wrangling.” First, we display the names of the data fields (variables) and compare them with (NOAA, 2021) for definitions and completeness, to ensure we have the necessary data and to better understand the content.

```
names(storm_data)
```

```
## [1] "STATE__"      "BGN_DATE"     "BGN_TIME"     "TIME_ZONE"
"COUNTY"
## [6] "COUNTYNAME"   "STATE"        "EVTYPE"       "BGN_RANGE"
"BGN_AZI"
## [11] "BGN_LOCATI"   "END_DATE"    "END_TIME"    "COUNTY_END"
"COUNTYENDN"
## [16] "END_RANGE"    "END_AZI"     "END_LOCATI"  "LENGTH"
"WIDTH"
## [21] "F"            "MAG"          "FATALITIES"  "INJURIES"
"PROPDMG"
## [26] "PROPDMGEXP"  "CROPDMG"     "CROPDMGEXP" "WFO"
"STATEOFFIC"
## [31] "ZONENAMES"    "LATITUDE"    "LONGITUDE"   "LATITUDE_E"
"LONGITUDE_"
## [36] "REMARKS"      "REFNUM"
```

## Populate Initial Analysis Dataset

The column names in storm\_data are clear enough to generate an initial analysis dataframe.

```
storm_data <- storm_data[,c("EVTYPE", "FATALITIES",
"INJURIES", "PROPDMG", "PROPDMGEXP", "CROPDMG",
"CROPDMGEXP", "COUNTYNAME", "STATE")]
```

Seeing that the data possesses the required fields, we examine the structure of those data. This will inform us as to the structure of the dataset (i.e., list, vector, dataframe, etc.), as well as reveal the type of variable we have, i.e., characters, integers, floating-point number, etc. It will also show us examples of the data in each field.

```

str(storm_data)

## 'data.frame': 902297 obs. of 9 variables:
## $ EVTYPE      : chr "TORNADO" "TORNADO" "TORNADO"
## "TORNADO" ...
## $ FATALITIES: num 0 0 0 0 0 0 0 1 0 ...
## $ INJURIES   : num 15 0 2 2 2 6 1 0 14 0 ...
## $ PROPDMG    : num 25 2.5 25 2.5 2.5 2.5 2.5 2.5 25 25
...
## $ PROPDMGEXP: chr "K" "K" "K" "K" ...
## $ CROPDMG    : num 0 0 0 0 0 0 0 0 0 ...
## $ CROPDMGEXP: chr "" "" "" ...
## $ COUNTYNAME: chr "MOBILE" "BALDWIN" "FAYETTE"
## "MADISON" ...
## $ STATE      : chr "AL" "AL" "AL" "AL" ...

```

The structure table shows the weather event-type, EVTYPE, recognizable by the Tornado events shown. This is the response variable for all our analysis. FATALITIES and INJURIES will help answer the Population health question. Finally, the fields for property damage (PROPDMG and PROPDMGEXP) and crop damage (CROPDMG and CROPDMGEXP). For instance, property damage (PROPDMG), a numerical field, and its associated exponent (PROPDMGEXP), a character field, provide the data for property damage effects. These fields are explained in (NOAA, 2021).

Now that we have a “feel” for the data, we generate a subset from storm\_data, comprised of the variables we need for our analysis. Subsets have computational efficiency and remove detractors.

### *Prepare the Storm Data for Analysis*

**Fatalities and Injuries** Recall our first analysis question: *Across the United States, which types of events (as indicated in the EVTYPE variable) are most harmful with respect to population health?*

Here we form a subset the contains the FATALITIES and INJURY fields with the Storm event type, EVTYPE. Although the larger set is not computationally constraining, it is good analysis practice to generate data subset for each analysis question. We name this subset, storm1, to indicate it is the storm data we need to answer analysis question 1. This is presented in **Table 5-1**.

```

storm1 <- storm_data[,c("EVTYPE", "FATALITIES",
"INJURIES")]
head(storm1,15) %>%
  kbl(align = "c",
    caption = "Table 1. Severe Weather Data Affecting
Population Health") %>%
  kable_classic(full_width = F, font = "Cambria") %>%
  column_spec(2, width = "10em") %>%
  column_spec(3, width = "10em")

Table 5-1. Severe Weather Data Affecting Population Health
```

| <b>Event Type</b> | <b>Fatalities</b> | <b>Injuries</b> |
|-------------------|-------------------|-----------------|
| TORNADO           | 0                 | 15              |
| TORNADO           | 0                 | 0               |
| TORNADO           | 0                 | 2               |
| TORNADO           | 0                 | 2               |
| TORNADO           | 0                 | 2               |
| TORNADO           | 0                 | 6               |
| TORNADO           | 0                 | 1               |
| TORNADO           | 0                 | 0               |
| TORNADO           | 1                 | 14              |
| TORNADO           | 0                 | 0               |
| TORNADO           | 0                 | 3               |
| TORNADO           | 0                 | 3               |
| TORNADO           | 1                 | 26              |
| TORNADO           | 0                 | 12              |
| TORNADO           | 0                 | 6               |

```

print(str(storm1))

## 'data.frame':   902297 obs. of  3 variables:
##   $ EVTYPE: chr  "TORNADO" "TORNADO" "TORNADO" "TORNADO"
...
##   $ FATALITIES: num  0 0 0 0 0 0 0 0 1 0 ...
##   $ INJURIES  : num  15 0 2 2 2 6 1 0 14 0 ...
##   NULL
```

```

print(summary(storm1))

##      EVTYPE          FATALITIES          INJURIES
## Length:902297    Min.   : 0.0000    Min.   : 0.0000
## Class:character  1st Qu.: 0.0000    1st Qu.: 0.0000
## Mode :character  Median : 0.0000    Median : 0.0000
##                  Mean   : 0.0168    Mean   : 0.1557
##                  3rd Qu.: 0.0000    3rd Qu.: 0.0000
##                  Max.   :583.0000    Max.   :1700.0000

```

To ensure we have what we asked for, we check the structure of this subset. Seeing no issues, we proceed with preparing the data for analysis question 2: *Across the United States, which types of events have the greatest economic consequences?* We generate this subset and name it Storm 2. Again, we examine the structure and data summary shown in **Table 5-2..**

```

storm2 <- storm_data[,c("EVTYPE", "PROPDMG", "PROPDMGEXP",
"CROPDMG", "CROPDMGEXP")]
head(storm2,15)  %>%
  kbl(align = "c",
    caption = "Table 2. Severe Weather Data Affecting
Population Health") %>%
  kable_classic(full_width = F, font = "Cambria")

```

**Table 5-2.** Severe Weather Data Affecting Population Health

| Event Type | Property Damage | Property Exponent | Crop Damage | Crop Exponent |
|------------|-----------------|-------------------|-------------|---------------|
| TORNADO    | 25.0            | K                 | 0           |               |
| TORNADO    | 2.5             | K                 | 0           |               |
| TORNADO    | 25.0            | K                 | 0           |               |
| TORNADO    | 2.5             | K                 | 0           |               |
| TORNADO    | 2.5             | K                 | 0           |               |
| TORNADO    | 2.5             | K                 | 0           |               |
| TORNADO    | 2.5             | K                 | 0           |               |
| TORNADO    | 2.5             | K                 | 0           |               |
| TORNADO    | 25.0            | K                 | 0           |               |
| TORNADO    | 25.0            | K                 | 0           |               |

**Table 5-2.** Severe Weather Data Affecting Population Health

| Event Type | Property Damage | Property Exponent | Crop Damage | Crop Exponent |
|------------|-----------------|-------------------|-------------|---------------|
| TORNADO    | 2.5             | M                 | 0           |               |
| TORNADO    | 2.5             | M                 | 0           |               |
| TORNADO    | 250.0           | K                 | 0           |               |
| TORNADO    | 0.0             | K                 | 0           |               |

```
print(str(storm2))

## 'data.frame': 902297 obs. of 5 variables:
## $ EVTYPE    : chr "TORNADO" "TORNADO" "TORNADO" ...
## $ PROPDMG   : num 25 2.5 25 2.5 2.5 2.5 2.5 2.5 ...
## $ PROPDMGEXP: chr "K" "K" "K" "K" ...
## $ CROPDMG   : num 0 0 0 0 0 0 0 0 0 ...
## $ CROPDMGEXP: chr "" "" "" "" ...
## NULL

print(summary(storm2))

##      EVTYPE                  PROPDMG                  PROPDMGEXP
CROPDMG
##  Length:902297      Min.   : 0.00  Length:902297
##  Min.   : 0.000          1st Qu.: 0.00  Class :character
##  Class :character      Median : 0.00  Mode   :character
##  1st Qu.: 0.000          Mean   : 12.06
##  Mode   :character      Median : 0.000
##  Median : 0.000          3rd Qu.: 0.50
##                               Mean   : 1.527
##                               3rd Qu.: 0.000
##                               Max.   :5000.00
##                               Max.   :990.000
##      CROPDMGEXP
##  Length:902297
##  Class :character
##  Mode  :character
##
```

We see that there is no missing value, so no need of inputting missing values. Next, let's calculate deaths and injuries by event type to

determine which storms and other weather events are most harmful to public health in the nation.

### *Compile the Data for Fatalities and Injuries Analysis*

Recall that we are looking for the top seven storm event types, so we'll get the representing the top seven for fatalities, and the top seven for injuries.

First, we sum the fatalities by storm event type, followed by the summed injuries for each storm event type.

```
fatalities <- aggregate(FATALITIES~EVTYPE, storm1, sum)
injuries <- aggregate(INJURIES~EVTYPE, storm1, sum)
```

Now, we get the top seven storm types for fatalities, and the for injuries.

```
top_7_fatality <- arrange(fatalities,
  desc(fatalities$FATALITIES))[1:7,]
top_7_injury <- arrange(injuries,
  desc(injuries$INJURIES))[1:7,]
```

### *Compile the Data for Property and Crop damage Analysis*

Next, let's similarly extract the seven storms and weather events that cause the most severe property and crop damage. Here, we need to investigate the “meaning” of the values in the property damage exponent field, PROPDGMGEXP, and crop damage field, CROPDGMGEXP. We need to convert the given “codes” into their numerical values to use them in calculating total damages, i.e., multiplying damage values by exponent values.

First, we extract the unique values contained in the PROPDGMGEXP and CROPDGMGEXP. field.

```
unique(storm2$PROPDGMGEXP)
## [1] "K" "M" "" "B" "m" "+" "0" "5" "6" "?" "4" "2"
"3" "h" "7" "H" "-" "1" "8"

unique(storm2$CROPDGMGEXP)
## [1] "" "M" "K" "m" "B" "?" "0" "k" "2"
```

Given these unique letter and integer “codes”, we identify the following patterns, and will use them to convert them to their respective exponent values.

- The number 1 places 1 zero to the right, yielding 10.
- We consider 2 as adding 2 zeros behind 1, so h, H, and 2 are 100.
- Also, we take 3 as adding 3 zeros after 1, So we take k and 3 as 1000.
- Likewise, 6 adds 6 zeros to 1, so M, m and 6 are 1000000.
- Then 9 adds 9 zeros to 1, so ‘B’ and 9 are 1 billion or 1000000000.
- The number 4 becomes 10000;
- 5 becomes 10000;
- 7 becomes 10000000;
- 8 becomes 100000000.
- Finally, we take 0, ?, and + as 1, adding no digits to 1.

Therefore, we can multiply the property and crop damage values by their respective exponents.

### ***Build Lookup Tables***

To facilitate this task, we build lookup tables for matching EXP and their Values. Then we use the lookup table and the storm dataset to get the value of property and crop damages. We should note that there are some inconsistencies in the crop and property exponents, i.e., the m, M and h, H entries. However, for the purpose of this analysis take them as the same.

```
PROPDGMGEXP <- sort(unique(storm2$PROPDGMGEXP)) #  
property damage levels  
# property damage multipliers  
propMult <-  
c(1,1,1,1,1,10,100,1000,10000,100000,1000000,10000000,  
100000000,1000000000,100,100,1000,1000000,10000000)  
propLookup <- data.frame(cbind (PROPDGMGEXP,propMult))  
# propLookup table  
propLookup$propMult <-  
as.numeric(as.character(propLookup$propMult))  
# convert to numeric  
CROPDMGEXP <- sort(unique(storm2$CROPDMGEXP))
```

```

# crop damage levels
cropMult <-
c(1,1,1,100,1000000000,1000,1000,1000000,1000000)
# crop damage multipliers
cropLookup <- data.frame(cbind (CROPDMGEXP, cropMult))
# cropLookup table
cropLookup$cropMult <-
  as.numeric(as.character(cropLookup$cropMult))
# convert to numeric

```

### *Merge Lookup Tables into Storm Data*

With the next code chunk, we merge property damage multiplier, `propMult`, and crop damage multiplier, `cropMult`, with severe weather (`storm2`) data set.

```

storm2 <- merge(storm2, propLookup)
# merge propMult into data set
storm2 <- merge(storm2, cropLookup)

```

Here, we merge cMultiplier into data set. For reference, we generate **Table 5-3** and **Table 5-4** to represent the lookup tables.

```

propLookup %>%
  kbl(col.names = c("EXONENT", "VALUE"), caption = "Table
3. Prpoerty Damage Exponent Codes with Values") %>%
  kable_classic(full_width = F, font = "Cambria") %>%
  column_spec(2, width = "18em")

```

**Table 5-3.** Property Damage Exponent Codes with Values

| EXONENT | VALUE |
|---------|-------|
| " "     | 1     |
| ?       | 1     |
| -       | 1     |
| 0       | 1     |
| 1       | 10    |
| 2       | 100   |
| 3       | 1000  |
| 4       | 10000 |

**Table 5-3.** Property Damage Exponent Codes with Values

| EXONENT | VALUE      |
|---------|------------|
| 5       | 100000     |
| 6       | 1000000    |
| 7       | 10000000   |
| 8       | 100000000  |
| B       | 1000000000 |
| h       | 100        |
| H       | 100        |
| K       | 1000       |
| m       | 1000000    |
| M       | 1000000    |

```
cropLookup %>%
  kbl(col.names = c("EXONENT", "VALUE"),
    caption = "Table 4. Crop Damage Exponent Codes with
Values") %>%
  kable_classic(full_width = F, font = "Cambria") %>%
  column_spec(2, width = "18em")
```

**Table 5-4.** Crop Damage Exponent Codes with Values

| EXONENT | VALUE      |
|---------|------------|
| 1       | 1          |
| ?       | 1          |
| 0       | 1          |
| 2       | 100        |
| B       | 1000000000 |
| k       | 1000       |
| K       | 1000       |
| m       | 1000000    |
| M       | 1000000    |

Now, we aggregate and sum the property damage by event type. Then, we generate the top seven most severe storms with the highest property damage values.

```

storm2$TOTALPROP <- storm2$PROPDMG *
as.numeric(storm2$propMult)
storm_prop <- aggregate(TOTALPROP ~ EVTYPE,
    data = storm2, sum)
top_property <- storm_prop[order(-storm_prop$TOTALPROP), ]
top_7_property <- top_property[1:7, ]

```

Next, we aggregate and sum the crop damage by event type. Then, we generate the top seven most severe storms with the highest crop damage values.

```

storm2$TOTALCROP <- storm2$CROPDMG *
as.numeric(storm2$cropMult)
storm_crop <- aggregate(TOTALCROP ~ EVTYPE,
    data = storm2, sum)
top_crop <- storm_crop[order(-storm_crop$TOTALCROP), ]
top_7_crop <- top_crop[1:7, ]

```

## Results

We are now ready to present and discuss the results of our analysis questions.

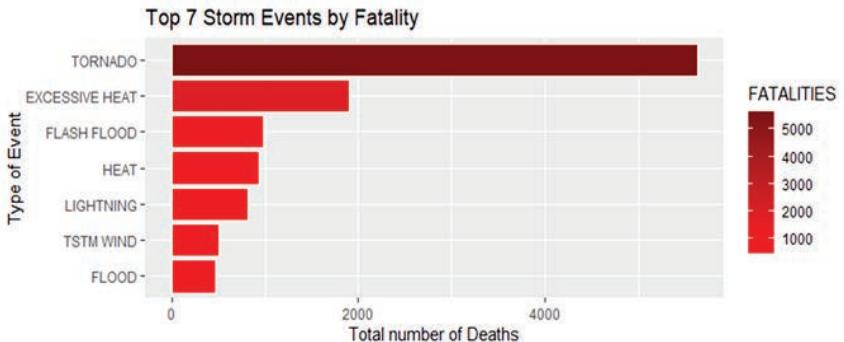
### *Public Health - Fatalities*

First, we generate a barplot with Fatality results, as seen in **Figure 5-6**.

```

f1 <- ggplot(top_7_fatality, aes(x = reorder(EVTYPE,
    FATALITIES),
y = FATALITIES, fill=FATALITIES))
f1 + geom_bar(stat = 'identity', color = 'white') +
    scale_fill_gradient(low = "red", high = "red4") +
    ggttitle('Top 7 Storm Events by Fatality') +
    xlab('Type of Event') +
    coord_flip() +
    ylab('Total number of Deaths')

```



*Figure 5-6.* indicates that tornadoes are responsible for the most fatalities, over three thousand more than those due to heat injuries (excessive heat). it is interesting to note the distinction between Excessive Heat and Heat. According to NOAA, the big distinction is “well above normal” versus “above normal” heat and high humidity (NOAA, 2021):

**“Excessive Heat (Z).** Excessive Heat results from a combination of high temperatures (well above normal) and high humidity. An Excessive Heat event occurs and is reported in Storm Data whenever heat index values meet or exceed locally/regionally established excessive heat warning thresholds. Fatalities (directly-related) or major impacts to human health that occur during excessive heat warning conditions are reported using this event category.”

**“Heat (Z).** A period of heat resulting from the combination of high temperatures (above normal) and relative humidity. A Heat event occurs and is reported in Storm Data whenever heat index values meet or exceed locally/regionally established advisory thresholds. Fatalities or major impacts on human health occurring when ambient weather conditions meet heat advisory criteria are reported using the Heat event.”

Lightning related deaths are well-understood by Coloradans, particularly at the higher altitudes. I’ve had a few close calls during high country hiking and horseback riding.

Flash floods are also common, especially with snow-melt and other natural phenomena. On July 31, 1976, the skies opened up over the Big Thompson Canyon, setting off the deadliest natural disaster in Colorado history that claimed 144 lives and caused \$35 million of damages. There is a dry wadi a little west of my house that is frequented by flash floods in the summer. The powerful effect of water is demonstrated by the manner it physical changes the terrain during a flash flood.

TSTM Winds are Thunderstorm Winds, which is defined as (see (Pohl, 2016)):

*"Thunderstorm Wind (C). Winds, arising from convection (occurring within 30 minutes of lightning being observed or detected), with speeds of at least 50 knots (58 mph), or winds of any speed (non-severe thunderstorm winds below 50 knots) producing a fatality, injury, or damage."*

These types of winds are commonplace where I live (on the Colorado prairie) with or without thunderstorms.

These results are summarized in **Table 5-5.** Top Seven Storm Type Affecting Public Health (Fatalities).

```
top_7_fatality %>%
  kbl(col.names = c("Event Type", "Fatalities"),
      align = "c",
      caption = "Table 5. Top Seven Storm Type
Affecting Public Health (Fatalities)") %>%
  kable_classic(full_width = F, font = "Cambria") %>%
  column_spec(2, width = "18em")
```

**Table 5-5.** Top Seven Storm Type Affecting Public Health (Fatalities)

| Event Type     | Fatalities |
|----------------|------------|
| TORNADO        | 5633       |
| EXCESSIVE HEAT | 1903       |
| FLASH          | 978        |
| FLOOD          |            |

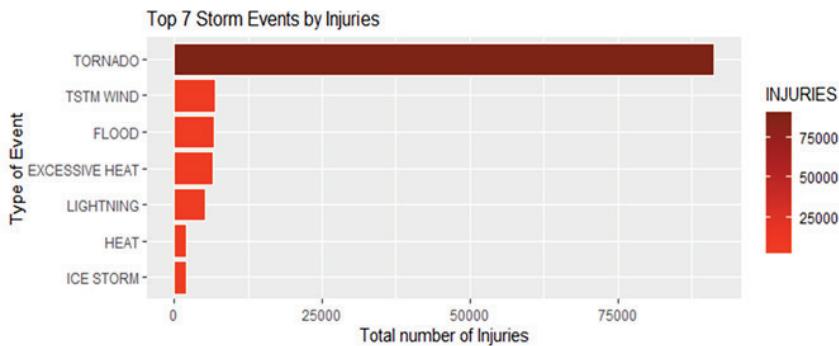
**Table 5-5.** Top Seven Storm Type Affecting Public Health (Fatalities)

| Event Type | Fatalities |
|------------|------------|
| HEAT       | 937        |
| LIGHTNING  | 816        |
| TSTM WIND  | 504        |
| FLOOD      | 470        |

### Public Health - Injuries

Next, we deal with injuries caused by severe whether events. As before, we present the results using a horizontal barplot shown in **Figure 5-7**.

```
f2<-ggplot(top_7_injury, aes(x=reorder(EVTTYPE, INJURIES),
                                y=INJURIES, fill=INJURIES))
f2 + geom_bar(stat='identity', color='white') +
  scale_fill_gradient(low = "orangered",
                      high = "orangered4") +
  ggttitle('Top 7 Storm Events by Injuries') +
  xlab('Type of Event') +
  coord_flip() +
  ylab('Total number of Injuries') +
  theme(plot.title = element_text(size = 12))
```



**Figure 5-7.** Storm events in severe consequences to public health - injuries (1950-2011)

The outcome is very similar to fatality results. One exception is the sheer scale of tornado related injuries. There are more than 75,000 more injuries caused by tornado than any other event.

Added to the injury results are ice storms, defined as:

*"Ice Storm (Z). Ice accretion meeting or exceeding locally/regionally defined warning criteria (typical value is 1/4 or 1/2 inch or more)."*

Table 6 summarizes these results visually.

```
top_7_injury %>%
  kbl(col.names = c("Event Type", "Injuries"),
  align = "c",
  caption = "Table 6.Top Seven Storm Type Affecting
  Public Health (Injuries)") %>%
  kable_classic(full_width = F, font = "Cambria") %>%
  column_spec(2, width = "20em")
```

**Table 5-6.** Top Seven Storm Type Affecting Public Health (Injuries)

| Event Type     | Injuries |
|----------------|----------|
| TORNADO        | 91346    |
| TSTM WIND      | 6957     |
| FLOOD          | 6789     |
| EXCESSIVE HEAT | 6525     |
| LIGHTNING      | 5230     |
| HEAT           | 2100     |
| ICE STORM      | 1975     |

### **Property Damage**

As we did with public health effects, we generate a table with the property damage resulting from severe weather, with an accompanying horizontal barchart.

```
top_7_property %>%
  kbl(col.names = c("Event Type", "Property Damage"),
```

```

align = "c",
caption = "Table 7. Top Seven Storm Type Affecting
Property Damage") %>%
kable_classic(full_width = F, font = "Cambria") %>%
column_spec(2, width = "20em")

```

**Table 5-7.** Top Seven Storm Type Affecting Property Damage

|     | Event Type        | Property Damage |
|-----|-------------------|-----------------|
| 170 | FLOOD             | 144657709807    |
| 411 | HURRICANE/TYPHOON | 69305840000     |
| 834 | TORNADO           | 56947380677     |
| 670 | STORM SURGE       | 43323536000     |
| 153 | FLASH FLOOD       | 16822673979     |
| 244 | HAIL              | 15735267513     |
| 402 | HURRICANE         | 11868319010     |

Property damage follows a similar pattern as fatalities, but the leader is different, with total costs for property damage due to flood coming in at \$144,657,709,807 or one hundred forty-four billion, six hundred fifty-seven million, seven hundred nine thousand eight hundred and seven U.S. dollars. or one trillion five hundred billion. Floods dominate property damage like tornado dominates fatalities. This is not surprising, since floods are generally more widespread as tornado. It is more common to hear that the Mississippi River went over its banks (an dikes) everywhere, just the other day, than it is to hear about a massive tornado. The most destructive tornado, or series of tornadoes, carved an approximately 250-mile path through northeast Arkansas, southeast Missouri, northwest Tennessee and western Kentucky on Dec 11, 2021, but was the first of such intensity in December since 1957.

How do we separate hurricanes from flooding and tornado, both of which can occur near simultaneously? The guidelines for reporting (see (Samenow, Feuerstein, & Livingston, 2021)) are clear:

*"Wind damage is the only individual hazard to be encoded in Hurricane/Typhoon, Tropical Storm, and Tropical Depression. This restriction prevents a 'double-count' from occurring in the national report entitled "A Summary of Natural Hazard Statistics for [Year] in the United States," which is based upon the header strips of Storm Data events.... Include all other impacts as separate events (e.g., storm surge/tide, freshwater flooding, tornadoes, debris flow, rip currents, etc.)."*

*"Flooding along the coast, even if it is from distant swells, will be entered as Storm Surge/Tide, not Coastal Flood. Rip Currents and High Surf can be entered in addition to Storm Surge/Tide, if applicable."*

It is surprising to see Hurricane as a separate event (from Hurricane/Typhoon), since the NATIONAL WEATHER SERVICE INSTRUCTION 10-1605, JULY 26, 2021, does not single it out. It is possible that earlier years did not combine hurricanes and typhoons, but the literature does not reveal it (see **Figure 5-8**).

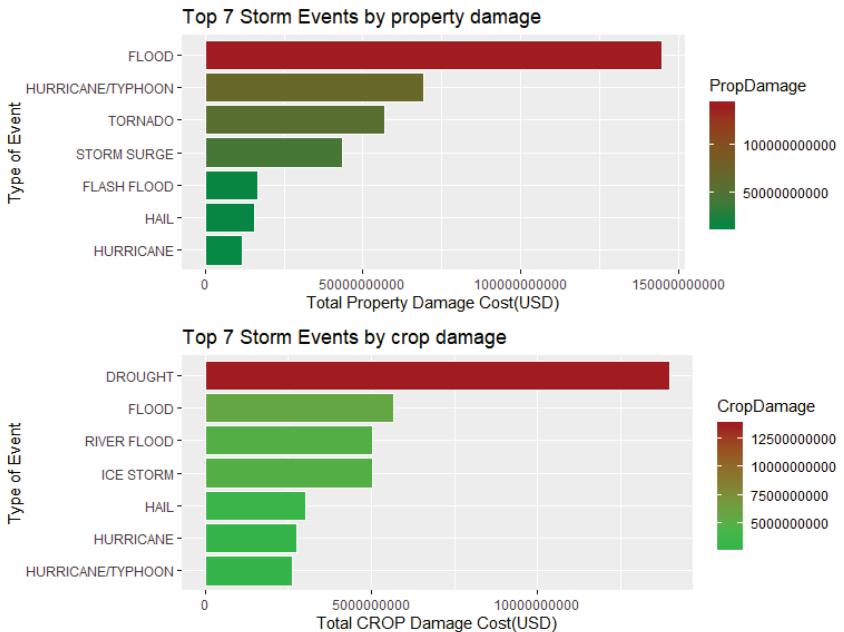
## Property Damage

```
names(top_7_property) <- c('EVTYPE', 'PropDamage')
d1 <- ggplot (top_7_property, aes(x=reorder(EVTYPE,
PropDamage), y=PropDamage, fill=PropDamage)) +
  geom_bar(stat='identity', colour='white') +
  scale_fill_gradient(low = "green4",
  high = "brown") +
  ggttitle('Top 7 Storm Events by property damage') +
  xlab('Type of Event') +
  coord_flip() +
  ylab('Total Property Damage Cost(USD)')
names(top_7_crop) <- c('EVTYPE', 'CropDamage')
d2 <- ggplot(top_7_crop, aes(x=reorder(EVTYPE,
CropDamage), y=CropDamage, fill=CropDamage)) +
  geom_bar(stat='identity', colour='white') +
  scale_fill_gradient(low = "green3",
  high = "brown") +
  ggttitle('Top 7 Storm Events by crop damage') +
  xlab('Type of Event') +
  coord_flip()
```

```

ylab('Total CROP Damage Cost(USD)')
grid.arrange(d1, d2) #Using the gridExtra-package

```



**Figure 5-8.** Storm events which have most severe consequences to the economy (1950-2011)

**Markdown Note.** In ````{r echo=TRUE}`

If `echo = FALSE`, knitr will not display the code in the code chunk above its results in the final document.

### Crop Damage

Here we demonstrate the crop damage resulting from severe weather, with the accompanying horizontal barplot at the bottom in Figure 3.

For this category, drought rules the day as is intuitive. The difference between flash floods and floods is clear, but what about river flooding. NOAA has no separate category for river floods and states (NOAA, 2021):

*"River flooding may be included as a Flood event. However, such entries should be confined to the effects of the river flooding, such as roads and bridges washed out, homes and businesses damaged, and the dollar estimates of such damage. The Water Resources Services Branch at National Weather Service Headquarters will maintain the official records of river stages, flood stages, and crests. Therefore, river stages need not be included in Storm Data."*

Like hurricanes, river flooding may be a legacy record, since this data goes back to 1950.

**Table 5-8** summarizes the damage to crops due to weather events.

```
top_7_crop %>%
  kbl(col.names = c("Event Type", "Crop Damage"),
      align = "c",
      caption = "Table 8. Top Seven Storm Type
Affecting Crop Damage") %>%
  kable_classic(full_width = F, font = "Cambria") %>%
  column_spec(2, width = "20em")
```

**Table 5-8.** Top Seven Storm Type Affecting Crop Damage

|     | Event Type        | Crop Damage |
|-----|-------------------|-------------|
| 95  | DROUGHT           | 13972566000 |
| 170 | FLOOD             | 5661968450  |
| 590 | RIVER FLOOD       | 5029459000  |
| 427 | ICE STORM         | 5022113500  |
| 244 | HAIL              | 3025954473  |
| 402 | HURRICANE         | 2741910000  |
| 411 | HURRICANE/TYPHOON | 2607872800  |

## Conclusion

Here, we recall our analysis questions and summarize the results:

**Analysis Question 1.** *Across the United States, which types of events (as indicated in the EVTYPE variable) are most harmful with respect to population health?*

With respect to fatalities, we found that tornado events dominate both aspects of population health, fatalities and injuries. Deaths due to tornado events total 5,633 from 1950 through 2011.

**Analysis Question 2.** *Across the United States, which types of events have the greatest economic consequences?*

In terms of property damage, flood events are the major contributor, resulting in nearly 145 billion US Dollars. For crop damage, drought is the leading cause, resulting in nearly 14 billion US dollars.



# 6. Inferential Statistics

## Overview

**Descriptive statistics** describes data (for example, a chart or graph) and **inferential statistics** allows us to make predictions ("inferences") from data. With inferential statistics, we take data from samples and make generalizations about a population.

For example, we might stand in a mall and ask a sample of 100 people if they like shopping at Macy's. You could make a bar chart of yes or no answers (that would be descriptive statistics) or you could use your research (and inferential statistics) to reason that around 75-80% of the population (all shoppers in all malls) like shopping at Macy's.

There are two main areas of inferential statistics:

1. **Parameter estimation.** This means taking a statistic from our sample data (for example the sample mean) and using it to say something about a population parameter (i.e., the population mean).
2. **Hypothesis tests.** This is where we can use sample data to answer research questions. For example, we might be interested in knowing if a new cancer drug is effective. Or if breakfast helps children perform better in schools.

## Estimating Population Parameters

The characteristics of samples and populations are described by numbers called statistics and parameters:

**Definition 6-1.** *A statistic is a measure that describes the sample (e.g., sample mean).*

**Definition 6-2.** *A parameter is a measure that describes the whole population (e.g., population mean).*

Sampling error is the difference between a parameter and a corresponding statistic. Since in most cases you don't know the real population parameter, you can use inferential statistics to estimate these parameters in a way that takes sampling error into account.

There are two important types of estimates you can make about the population: point estimates and interval estimates.

**Definition 6-3.** *A point estimate is a single value estimate of a parameter. For instance, a sample mean is a point estimate of a population mean.*

**Definition 6-4.** *An interval estimate gives you a range of values where the parameter is expected to lie. A confidence interval is the most common type of interval estimate.*

Both types of estimates are important for gathering a clear idea of where a parameter is likely to lie.

## Confidence Intervals

A **confidence interval** uses the variability around a statistic to come up with an interval estimate for a parameter. Confidence intervals are useful for estimating parameters because they take sampling error into account.

While a point estimate gives you a precise value for the parameter you are interested in, a confidence interval tells you the uncertainty of the point estimate. They are best used in combination with each other.

Each confidence interval is associated with a confidence level. A confidence level tells you the probability (in percentage) of the interval containing the parameter estimate if you repeat the study again. A 95% confidence interval means that if you repeat your study with a new sample in exactly the same way 100 times, you can expect your estimate to lie within the specified range of values 95 times.

Although you can say that your estimate will lie within the interval a certain percentage of the time, you cannot say for sure that the actual population parameter will. That's because you can't know the true

value of the population parameter without collecting data from the full population. However, with random sampling and a suitable sample size, you can reasonably expect your confidence interval to contain the parameter a certain percentage of the time.

## Levels of Measurement

The levels of measurements are nominal, ordinal, interval, and ratio data. Going from lowest to highest, the 4 levels of measurement are cumulative. This means that they each take on the properties of lower levels and add new properties.

| Nominal level  | Examples of nominal scales  |
|--|---|
| You can categorize your data by labelling them in mutually exclusive groups, but there is no order between the categories.   | <ul style="list-style-type: none"><li>• City of birth</li><li>• Gender</li><li>• Ethnicity</li><li>• Car brands</li><li>• Marital status</li></ul>  |
| Ordinal level  | Examples of ordinal scales  |
| You can categorize and rank your data in an order, but you cannot say anything about the intervals between the rankings.<br><br>Although you can rank the top 5 Olympic medalists, this scale does not tell you how close or far apart they are in number of wins. | <ul style="list-style-type: none"><li>• Top 5 Olympic medallists</li><li>• Language ability (e.g., beginner, intermediate, fluent)</li><li>• Likert-type questions (dissatisfied to very satisfied)</li></ul> |
| Interval level   | Examples of interval scales   |
| You can categorize, rank, and infer equal intervals between neighboring  | <ul style="list-style-type: none"><li>• Test scores (e.g., IQ or exams)</li><li>• Personality inventories</li></ul>   |

- data points, but there is no true zero point.
- Temperature in Fahrenheit or Celsius

The difference between any two adjacent temperatures is the same: one degree. But zero degrees is defined differently depending on the scale – it doesn't mean an absolute absence of temperature.

The same is true for test scores and personality inventories. A zero on a test is arbitrary; it does not mean that the test-taker has an absolute lack of the trait being measured.

| Ratio level   | Examples of ratio scales  |
|---|---|
| You can categorize, rank, and infer equal intervals between neighboring data points, and there is a true zero point.                        |   |
| A true zero means there is an absence of the variable of interest. In ratio scales, zero does mean an absolute lack of the variable.        | <ul style="list-style-type: none"><li>• Height</li><li>• Age</li><li>• Weight</li><li>• Temperature in Kelvin</li></ul> |
| For example, in the Kelvin temperature scale, there are no negative degrees of temperature – zero means an absolute lack of thermal energy. |   |

### Example 6-1. Point Estimate and Confidence Interval

We want to know the average number of paid vacation days that employees at an international company receive. After collecting survey responses from a random sample, we calculate a point

estimate and a confidence interval. Our point estimate of the population mean paid vacation days is the sample mean of 19 paid vacation days.

With random sampling, a 95% confidence interval of [16 22] means we can be reasonably confident that the average number of vacation days is between 16 and 22.

## Comparison Tests

**Comparison tests** assess whether there are differences in means, medians or rankings of scores of two or more groups. To decide which test suits your aim, consider whether your data meets the conditions necessary for parametric tests, the number of samples, and the levels of measurement of your variables.

Means can only be found for interval or ratio data, while medians and rankings are more appropriate measures for ordinal data. **Table 6-1** provides a list of comparison tests that are available in *R*, all of which are part of the *stats* package.

**Table 6-1. Comparison tests** assess whether there are differences in means, medians or rankings of scores of two or more groups.

| Comparison test  | Parametric? | What's being compared? | Samples |
|--|-------------|------------------------|---------|
| <b>t-test</b><br><code>t.test {stats}</code>                                   | Yes         | Means                  | 2       |
| <b>ANOVA</b> <code>anova{stats}</code>   | Yes         | Means                  | 3+      |
| <b>Mood's median</b><br><code>(mood.test{stats})</code>                        | No          | Medians                | 2+      |
| <b>Wilcoxon signed-rank</b><br><code>(wilcox.test{stats})</code>               | No          | Distributions          | 2       |
| <b>Wilcoxon rank-sum (Mann-Whitney U)</b><br><code>(wilcox.test{stats})</code> | No          | Sums of rankings       | 2       |
| <b>Kruskal-Wallis H</b>  | No          | Mean rankings          | 3+      |

| Comparison test      | Parametric? | What's being compared? | Samples |
|----------------------|-------------|------------------------|---------|
| kruskal.test {stats} |             |                        |         |

## Correlation Tests

**Correlation tests** determine the extent to which two variables are associated. Some of the tests (not all) available in R appear in **Table 6-2**. Although Pearson's  $r$  is the most statistically powerful test, Spearman's  $r$  is appropriate for interval and ratio variables when the data doesn't follow a normal distribution. The chi square test of independence is the only test that can be used with nominal variables.

**Table 6-2. Correlation tests available in R.**

| Correlation test   | Parametric? | Variables                        |
|--|-------------|----------------------------------|
| <b>Pearson's <math>r</math></b><br>chisq.test {stats}      | Yes         | Interval/ratio variables         |
| <b>Spearman's <math>r</math></b><br>cor.test {stats}       | No          | Ordinal/interval/ratio variables |
| <b>Chi square test of independence</b><br>Assocstats {vcd} | No          | Nominal/ordinal variables        |

## Example 6-2. The Central Limit Theorem

### Overview

In this analysis, we compare the distribution of 1000 random samples from an exponential distribution and the distribution of averages of 40 exponentials. The source code is located at: [https://github.com/stricje1/inferential\\_stats](https://github.com/stricje1/inferential_stats).

This problem is an application of the Central Limit Theorem (CLT). The CLT is defined in **Definition 6-1**. We offer it as a definition rather than a theorem since we have no intention of formally proving it.

**Definition 6-5.** If  $X_1, X_2, \dots, X_n$  are  $n$  random samples drawn from a population with overall mean  $\mu$  and finite variance  $\sigma^2$ , and if  $\bar{X}_n$  is the sample mean, then the limiting form of the distribution,  $Z = \lim_{n \rightarrow \infty} \sqrt{n} \left( \frac{\bar{X}_n - \mu}{\sigma} \right)$ , is a standard normal distribution.

The CLT is probably the most important theorem in inferential statistics. Simply, the CLT states that when a sufficient number of independent random variables are summed or averaged, their result forms a distribution of sums or averages that are approximately normally distributed. Hence, the 1000 averages of 40 random uniform variates appear in the shape of a normal distribution (40 is considered adequate by most authors for application of the CLT).

### ***The Exponential Distribution***

While samples have statistics, like the sample mean, population distributions have parameters, like the population mean. The exponential (lambda) distribution is defined by one parameter lambda, the rate. For our distribution, we take lambda to equal 0.2, with the mean being mu = 1/lambda = 5 and standard deviation, sigma = 1/lambda = 5.

### ***Simulation of the Exponential(lambda) distribution***

First, generate the distribution of 1000 random exponentials, using the `stats`-package, which provides the random uniform distribution function, `rexp`. To see the documentation of the stats-package, we use the command, `library(help = 'stats')`.

```
library(help = "stats") # Opens the stats-package documentation in the Editor window/view.
library("stats")
```

### ***Set up the Simulation***

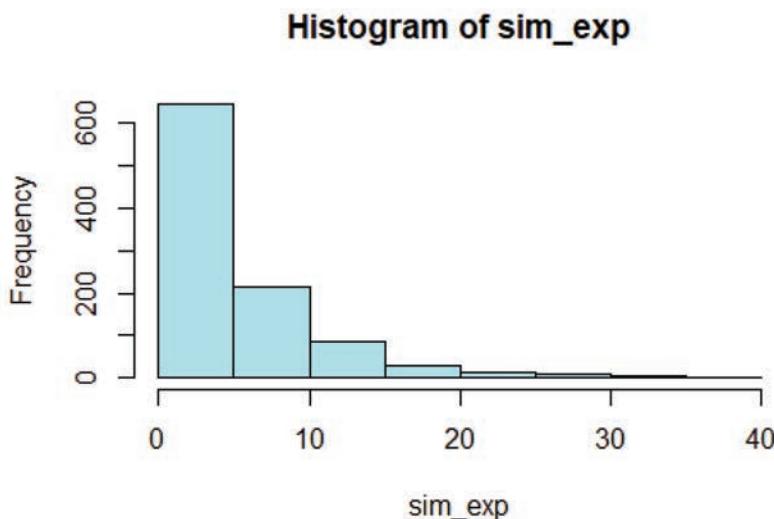
The documentation for the random uniform random variate provides the functions:

- `rexp` generates random deviates
- `n_sim` the number of simulations
- $\lambda$  is the rate of growth or decay

- $1/\lambda$  is the population mean (parameter)

Below, we generate 1000 random uniform distributions, calculate the sample means and plot a histogram. First, we construct the histogram as depicted in **Figure 6-1**.

```
n_sim = 1000
lambda = 0.2
sim_exp = rexp(n_sim, lambda)
hist(sim_exp, col="powderblue")
```



**Figure 6-1.** Histogram of simulated exponential random variable

Now, we calculate the sample mean.

```
cat("The sample mean is ", mean(sim_exp))
## The sample mean is 5.107197
```

### Applying the CLT

Now, we take our exponential distributions, and using their sample means, we generate a distribution of 40 of those means.

## **The Normal Distribution, $\text{Normal}(\mu, \sigma)$**

The normal distribution is defined by two parameters, population mean =  $\mu$ , the population standard deviation =  $\sigma$ , or  $\text{Normal}(\mu, \sigma)$ .

### **Normal Random Variates**

The sample MNS is approximately normal (due to the CLT) and has a sample mean near 5 and a sample standard deviation near calculated as  $(1/\lambda)^2/n_{dist}$ . So, now we form the random distribution from the means of 40 Uniform[0,1] random variates.

```
Lambda = 0.2
mu = 1/lambda
n_dist = 40
n_sim = 1000
mns_u = NULL
d = data.frame(mean = numeric())
for (i in 1 : n_sim){
  mns_u = c(mns_u, mean(rexp(n_dist, lambda)))
  d[i,1] = mean(mns_u)
}
cat("The mean value of the exponential is:", mean(d$mean),
"which is the mean of the normal distribution.")

## The mean value of the exponential is: 4.988691 which is the mean of the normal distribution.
```

### **The Mean Value**

As we have seen, the mean value of the exponential distribution is  $1/\lambda$  and in our example, it is equal to 5. The random variates drawn from this distribution have means near 5 and when we take form a distribution of these sample means, the new distribution also has a sample mean of 5.

```
mu = 1/lambda
x_bar = mean(d$mean)
cat("The mean value, mu, of the normal distro is ", mu,"and the sample mean is ", x_bar)

## The mean value, mu, of the normal distro is 5 and the sample mean is 4.988691
```

So, we see that the sample mean of the normal distribution of exponential means is approximately 5, and the parameter mu is 5.

### The Variance

The variance of the population parameter,  $\sigma^2$ , is given by  $(1/\lambda)^2/n_{dist}$ . Here, we calculate its value and compare to the sample variance of our normal distribution.

```
samp_var = round(var(mns_u),4)
sigma_sq = (1/lambda)^2/n_dist
cat("The population variance is ", sigma_sq, "and the sample variance is ", samp_var)

## The population variance is 0.625 and the sample variance is 0.6298
```

So, we see that the sample variance approximates the population parameter,  $\sigma^2$ .

#### Markdown Note.

```
```{r results = 'markup'}
```

If '`hide`', knitr will not display the code's results in the final document. If '`hold`', knitr will delay displaying all output pieces until the end of the chunk. If '`asis`', knitr will pass through results without reformatting them (useful if results return raw HTML, etc.)

### Histogram Approximation of the Normal Distribution

In **Figure 6-2**, we construct the histogram of our normal distribution overlaid by horizontal lines (using the `abline()` function) representing the sample mean and the population (theoretical) mean. The green line is the sample mean or statistic, while the red line is the theoretical mean or parameter.

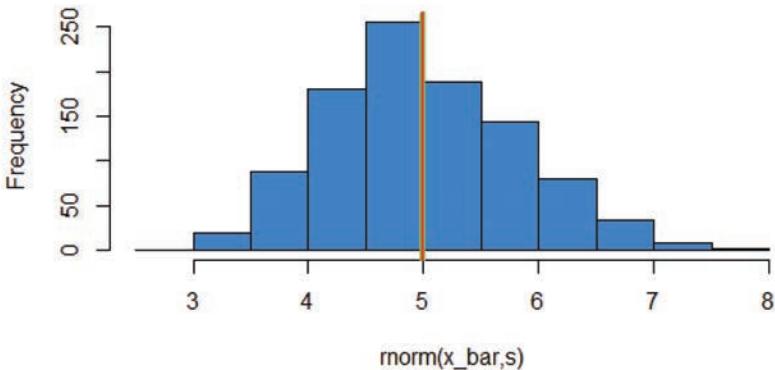
```
hist(mns_u, col="dodgerblue", xlab = "rnorm(x_bar,s)",
      main = "Histogram of an Approximate Normal
```

```

(mu,sigma) Distribution")
abline(v=mean(d$mean), col = "green", lwd=4) # Sample mean
abline(v=1/lambda, col = "red", lwd=2) # Theoretical mean

```

### Histogram of an Approximate Normal( $\mu, \sigma$ ) Distribution



**Figure 6-2.** Histogram of an approximate  $\text{Normal}(\mu, \sigma)$  distribution

Looking at the histogram, the distribution appears to be approximately normal, as expected. We now calculate the population parameters and sample statistics. The results are in **Table 6-3**.

```

df <- data.frame(
  Metric=rep(c('mean', 'variance', 'stdev', 'minimum',
  'maximum'), each=1),
  Statistic=rep(c(
    sprintf("%.6f", mean(mns_u)) ,
    sprintf("%.6f", var(mns_u)),
    sprintf("%.6f", sd(mns_u)),
    sprintf("%.6f", min(mns_u)),
    sprintf("%.6f", max(mns_u))), times=1),
  Parameter=rep(c(
    sprintf("%.6f", 1/lambda),
    sprintf("%.6f", (1/lambda)^2/n_dist),
    sprintf("%.6f", 1/lambda),
    "-infinity",
    "+infinity"), times=1))
knitr::kable(df)

```

**Table 6-3.** Sample statistic and population parameters

Metric	Statistic	Parameter
<b>mean</b>	4.994164	5.000000
<b>variance</b>	0.672584	0.625000
<b>standard deviation</b>	0.820112	0.790569
<b>minimum</b>	2.662689	-infinity
<b>maximum</b>	7.825034	+infinity

## Example 6-3. Tooth Growth and Nutrients

This example is about the effects of dosage levels and supplements on tooth growth. We present this as a data science report analyzing elements affecting tooth growth, specifically supplement type (VC or OJ) and dose in milligrams/day. In this analysis we examine whether various combinations of supplements and dose significantly impact tooth growth. We do this by performing hypothesis test to compare make these comparisons. The source code is located at: [https://github.com/stricje1/inferential\\_stats](https://github.com/stricje1/inferential_stats).

### Assumptions

This kind of analysis requires the assumptions of a two-sample t-test, which are as follows:

1. The data are continuous (not discrete).
2. The data (tooth length) follow the normal probability distribution.
3. The variances of the two populations are equal. (If not, the Aspin-Welch Unequal-Variance test is used.)
4. The two samples are independent (according to supplement types or level of dose). There is no relationship between the individuals in one sample as compared to the other (as there is in the paired t-test).
5. Both samples are simple random samples from their respective populations. Each individual in the population has an equal probability of being selected in the sample.

## *Exploratory Data Analysis (EDA)*

Before delving into the heart of the analysis, it is appropriate perform some basic exploratory data analysis.

### *ToothGrowth Description*

ToothGrowth is an R dataframe comprised of 60 observations and 3 variables. Length (len) is a numerical variable from 4.20 to 3.90 Supplement (supp) is a factor with 2 levels, "OJ" (orange juice) and "VC" (vitamin C) Dose (dose) is a factor with 3 levels, "0.5", "1", and "2"

```
data(ToothGrowth)
? ToothGrowth # To read the document on ToothGrowth
str(ToothGrowth)
## 'data.frame': 60 obs. of 3 variables:
## $ len : num 4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
## $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
## $ dose: num 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
tg <- ToothGrowth
dim(tg)
## [1] 60 3
```

### *ToothGrowth Summary*

```
library(dplyr)
group_data = group_by(tg, supp, dose)
dplyr::summarise(group_data, mean(len))
```

supp	dose	mean(len)
OJ	0.5	13.23
OJ	1.0	22.70
OJ	2.0	26.06
VC	0.5	7.98
VC	1.0	16.77
VC	2.0	26.14

6 rows

## Data Visualization

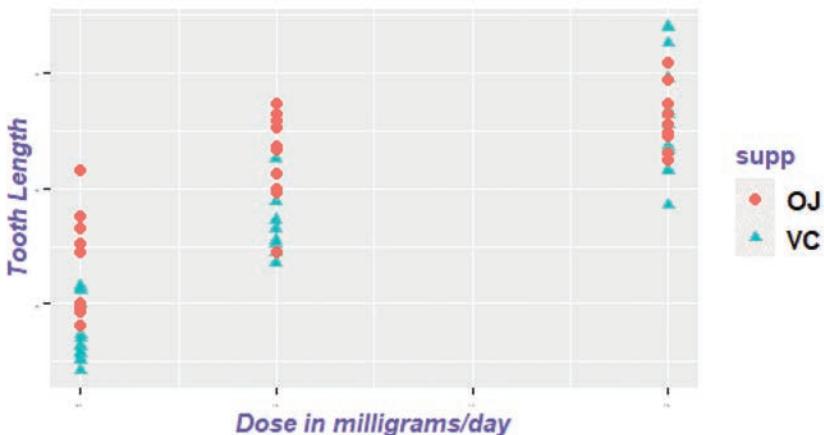
First, we construct a scatterplot (see **Figure 6-3**) of tooth length according to supplement type (Vitamin C and Orange Juice) by dosage levels, using `ggplot()` with `geom()` functions.

```
library(ggplot2)
p = ggplot(tg, aes(x = dose, y = len))
p = p + geom_point(aes(color = supp, shape = supp),
                    size = 2.0)
p = p + labs(x = "Dose in milligrams/day",
              y = "Tooth Length")
p = p + labs(title = "Tooth Growth")
axis_size = element_text(face= "bold", size = rep(1.5))
axis_title = element_text(face = "bold.italic",
                           color = "slateblue", size = rel(1))
main_title = element_text(size = rel(2.0),
                           color = "slateblue")
# title and labels
p = p + theme(axis.text = axis_size, axis.title =
               axis_title, plot.title = main_title)
# legend title
p = p + theme(legend.title =
               element_text(colour="slateblue", size=rel(1),
                            face="bold"))
# legend labels
p = p + theme(legend.text = element_text(colour="black",
   size=rel(1), face="bold"))
p
```

**Markdown Note.** There are numerous things you can do in a code chunk: you can generate text output, tables, and graphics. This gives you precise control over all these output via chunk options, which you can place inside curly braces (between ``{r} and }). For example, suppose you want to hide the output of `library(randomForest)`, which consist of warning that may be important to you but not so for your reader. You can choose to hide the text output using the chunk option `results = 'hide'` or set the figure height to 4 inches by `fig.height = 4`. Chunk options are separated by commas, like

```
```{r, chunk-label, results='hide', fig.height=4}
```

# Tooth Growth



**Figure 6-3.** Scatterplot of tooth growth by supplement dosage

## Boxplot Tooth Growth Factors

Boxplots can provide a visual aid in determining if one distribution's parameters are significantly different than another. This can help us determine whether a hypothesis test is unnecessary. We use three sets of boxplots as follows:

1. First, we examine the boxplots of the supplements, OJ and VC (see **Figure 6-4**)
2. Second, we explore the boxplots of the three levels of doses
3. Third, the dosage levels for each of the two different supplements

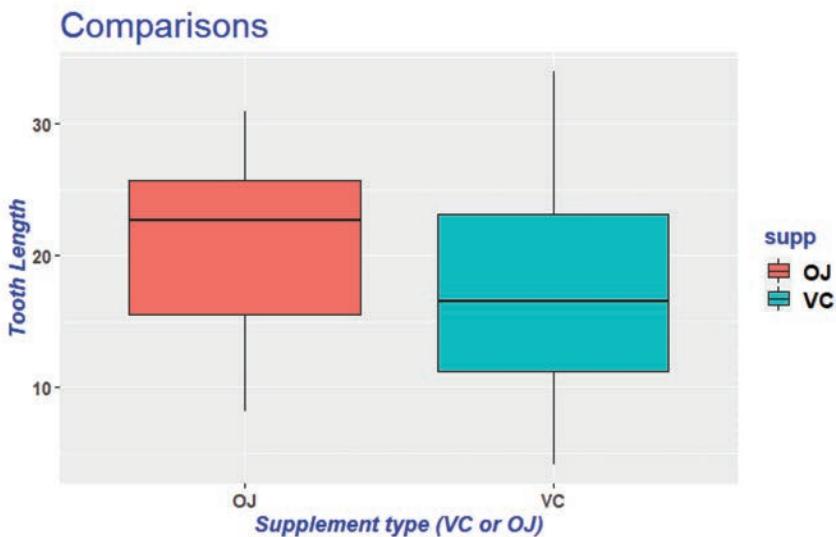
## Boxplot Comparison of Orange Juice vs Vitamin C

```
library(ggplot2)
p = ggplot(tg, aes(x = supp, y = len))
p = p + geom_boxplot(aes(fill = supp))
p = p + labs(x = "Supplement type (VC or OJ)",
              y = "Tooth Length")
p = p + labs(title = "Comparisons")
axis_size = element_text(face = "bold", size = rel(1.0))
axis_title = element_text(face = "bold.italic",
                          color = "blue2", size = rel(1.25))
```

```

main_title = element_text(size = rel(2.0),
                          color = "blue2")
# title and labels
p = p + theme(axis.text = axis_size, axis.title =
               axis_title, plot.title = main_title)
# legend title
p = p + theme(legend.title = element_text(colour="blue2",
                                           size=rel(1.25), face = "bold"))
# legend labels
p = p + theme(legend.text = element_text(colour="black",
                                           size=rel(1.25), face = "bold"))
p

```



**Figure 6-4.** Boxplots of tooth length by each supplement showing Orange Juice may be a better supplement on average

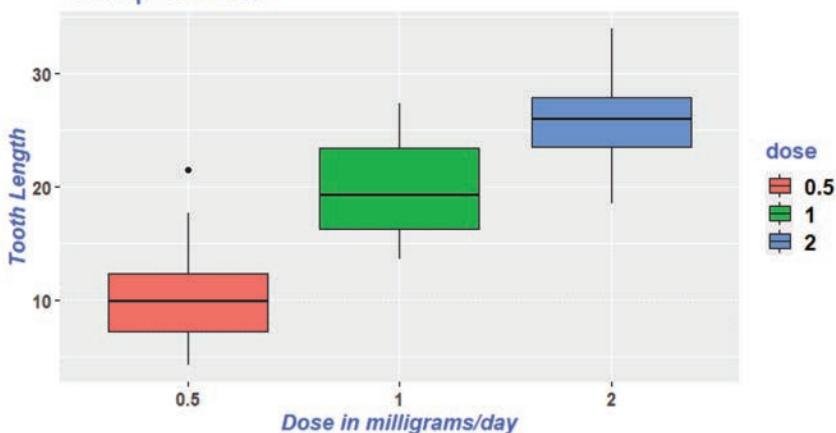
The first set of boxplots demonstrate distributions whose means may be significantly different warrant analysis using hypothesis testing. For the remaining boxplots, we omit the code using `echo=FALSE` in the markdown to save space. They are all similar to the previous.

## *Boxplot Comparison of Three Dosage Levels*

This set of boxplots also reveal distributions whose means may be significantly different warrant analysis using hypothesis testing (see **Figure 6-5**).

```
library(ggplot2)
tg$dose = as.factor(tg$dose)
p = ggplot(tg, aes(x = dose, y = len))
p = p + geom_boxplot(aes(fill = dose))
p = p + labs(x = "Dose in milligrams/day",
              y = "Tooth Length")
p = p + labs(title = "Comparisons")
axis_size = element_text(face = "bold", size = rel(1.0))
axis_title = element_text(face = "bold.italic",
                           color = "royalblue", size = rel(1.25))
main_title = element_text(size = rel(2.0),
                           color = "royalblue")
# title and labels
p = p + theme(axis.text = axis_size,
               axis.title = axis_title, plot.title = main_title)
# legend title
p = p + theme(legend.title = element_text(colour=
                                         "royalblue", size = rel(1.25), face = "bold"))
# legend labels
p = p + theme(legend.text = element_text(colour= "black",
                                         size = rel(1.25), face = "bold"))
p
```

## Comparisons

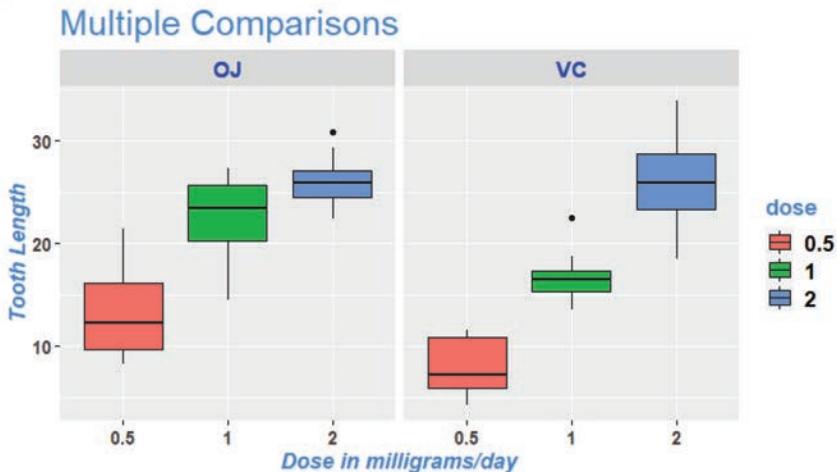


**Figure 6-5.** Boxplots for tooth length by distinct dosages

Boxplot Comparison of Orange Juice vs Vitamin C at Three Dosage Levels, as seen in **Figure 6-6**.

```
library(ggplot2)
tg$dose = as.factor(tg$dose)
p = ggplot(tg, aes(x = dose, y = len))
p = p + geom_boxplot(aes(fill = dose))
p = p + facet_wrap(~ supp)
p = p + labs(x = "Dose in milligrams/day",
              y = "Tooth Length")
p = p + labs(title = "Multiple Comparisons")
axis_size = element_text(face= "bold", size = rel(1.0))
axis_title = element_text(face = "bold.italic",
                           color = "dodgerblue", size = rel(1.25))
main_title = element_text(size = rel(2),
                           color = "dodgerblue")
# title and labels
p = p + theme(axis.text = axis_size,
               axis.title = axis_title, plot.title = main_title)
p = p + theme(strip.text = element_text(face = "bold",
                                         size = rel(1.25), colour = "blue"))
# legend title
p = p + theme(legend.title = element_text(colour=
                                         "dodgerblue", size =rel(1.25), face= "bold"))
# legend labels
p = p + theme(legend.text = element_text(colour="black",
```

```
size = rel(1.25), face = "bold"))
p
```



**Figure 6-6.** Boxplots for tooth length by supplement and by dosage

This set of boxplots shows that at dosage-level 0.5, the means of OJ and VC may be significantly different, as does the means of OJ and VC at dosage-level 1. However, the distributions of OJ and VC at dosage-level 2 appear not to be significantly different. Hence, we can perform the following hypothesis tests.

### Hypothesis Test

Plots are good visual aids in proving insights to the data we analyze. However, only hypothesis testing can provide the rigor of statistical significance. Even when things appear to be unequal, the differences we see may not be statistically significant.

### Hypothesis Test on the Effect of Supplement Types

To perform this hypothesis test we have to assume these two samples (OJ and VC) are taken from populations with a Gaussian distribution. Moreover, it is necessary to check that these samples have equal variances, by performing a t-test as follows:

```
var.test(len ~ supp, data = tg)
##
## F test to compare two variances
##
```

```

## data: len by supp
## F = 0.6386, num df = 29, denom df = 29, p-value =
0.2331
## alternative hypothesis: true ratio of variances is not
equal to 1
## 95 percent confidence interval:
## 0.3039488 1.3416857
## sample estimates:
## ratio of variances
## 0.6385951

```

Examining the p-value (0.2331), we see that it is greater than 0.05. Hence, we fail to reject the null hypothesis that two variances are equal. Now, we test the following hypothesis.

**H<sub>0</sub>:** There is no significant difference between the means supplement type (VC or OJ). That is, the two supplements have similar effects on tooth length.

**H<sub>A</sub>:** Supplement types OJ and VC have different means and hence different effects on tooth length.

```

t.test(len ~ supp, paired = F, var.equal = T, data = tg)
##
## Two Sample t-test
##
## data: len by supp
## t = 1.9153, df = 58, p-value = 0.06039
## alternative hypothesis: true difference in means
## between group OJ and group VC is not equal to 0
## 95 percent confidence interval:
## -0.1670064 7.5670064
## sample estimates:
## mean in group OJ mean in group VC
## 20.66333 16.96333

```

The p-value (0.06039) is greater than 0.05, which indicates that we cannot reject the null hypothesis. Thus, we conclude that the two supplement types have a similar effect on the tooth growth. We also see that the confidence interval for the difference of means is [-0.167,7.567]. Since this interval contains zero, we conclude that

there is no significant difference between the effect of these two supplement types on the growth of tooth.

### *Hypothesis Test on the Effect of Dose Types*

In keeping with class techniques, we will use pair by pair (F-paired) tests for our evaluating these hypotheses.

$H1_0$ : The means of dosage levels “0.5” and “1” are the same.

$H2_0$ : The means of dosage levels “0.5” and “2” are the same.

$H3_0$ : The means of dosage levels “1” and “2” are the same.

```
subset1 = subset(tg, dose %in% c(0.5, 1.0))
subset2 = subset(tg, dose %in% c(0.5, 2.0))
subset3 = subset(tg, dose %in% c(1.0, 2.0))
```

Again, we first test for equal variances using a t-test.

```
var.test(len ~ dose, data = subset1)
##
## F test to compare two variances
##
## data: len by dose
## F = 1.0386, num df = 19, denom df = 19, p-value = 0.9351
## alternative hypothesis: true ratio of variances is not
## equal to 1
## 95 percent confidence interval:
## 0.4110751 2.6238736
## sample estimates:
## ratio of variances
## 1.038561
```

Since these samples have equal variance, we choose

```
var.equal = TRUE in the following t-test.
t.test(len ~ dose, paired = FALSE,
var.equal = TRUE, data = subset1)

##
## Two Sample t-test
##
## data: len by dose
```

```

## t = -6.4766, df = 38, p-value = 1.266e-07
## alternative hypothesis: true difference in means
between group 0.5 and group 1 is not equal to 0
## 95 percent confidence interval:
## -11.983748 -6.276252
## sample estimates:
## mean in group 0.5 mean in group 1
## 10.605 19.735

```

The p-value is less than 0.05. Hence, we reject the null hypothesis conclude there is a difference in the effect of these two dosage levels. Also, the confidence interval for the difference of the means is [-11.9837,-6.2763]. This implies the first level (dose="0.5") is not effective as the second level (dose="1"). Similarly, we can conduct the above procedure to mydata2 and mydata3.

### *Supplement as a Factor within Dose Levels*

Similarly, we need to get the following three sub-group data.

```

subset4 = subset(tg, dose == 0.5)
subset5 = subset(tg, dose == 1.0)
subset6 = subset(tg, dose == 2.0)

```

Now, we assume that the variances of two supplement types are different. So, we choose `var.equal=F` in the flowing t-test.

```

t.test(len ~ supp, paired = F, var.equal = F, data =
subset4)

## Welch Two Sample t-test
##
## data: len by supp
## t = 3.1697, df = 14.969, p-value = 0.006359
## alternative hypothesis: true difference in means
between group OJ and group VC is not equal to 0
## 95 percent confidence interval:
## 1.719057 8.780943
## sample estimates:
## mean in group OJ mean in group VC
## 13.23 7.98

```

The p-value (0.006359) is much less than 0.05. Hence, we reject the null hypothesis. This the difference in means is not equal to zero, and

we conclude that there is a difference between orange juice and vitamin C as supplements for teeth growth. Also, the associated confidence interval for the difference of means is [1.719,8.781]. This implies the supplement type of orange juice (OJ) is much more effective than vitamin C (VC) at the level of dose equal to 0.5. Also, similarly, we can conduct the above procedure to subset5 and subset6.

## Summary

We concluded that the two supplement types have a similar effect on the tooth growth when taken at an aggregate of dose-levels. We also concluded there is a difference in the effect of these two dosage levels apart from supplement type. Finally, we learned there is a difference between orange juice and vitamin C as supplements for teeth growth.



## 7. Regression Models

Regression models are the workhorse of data science. They are the most well described, practical and theoretically understood models in statistics. A data scientist well versed in regression models will be able to solve an incredible array of problems.

Perhaps the key insight for regression models is that they produce highly interpretable model fits. This is unlike machine learning algorithms, which often sacrifice interpretability for improved prediction performance or automation. These are, of course, valuable attributes in their own rights. However, the benefit of simplicity, parsimony and interpretability offered by regression models (and their close generalizations) should make them a first tool of choice for any practical problem.

### Example 7-1 – Galton Study

Let's look at the data first, used by Francis Galton in 1885. Galton was a statistician who invented the term and concepts of regression and correlation, founded the journal Biometrika, and was the cousin of Charles Darwin. Galton is a data set from tabulated data set used by Galton in 1885 to study the relationship between a parent's height and their children's. The source code for this example is located at: [https://github.com/stricje1/linear\\_regression](https://github.com/stricje1/linear_regression).

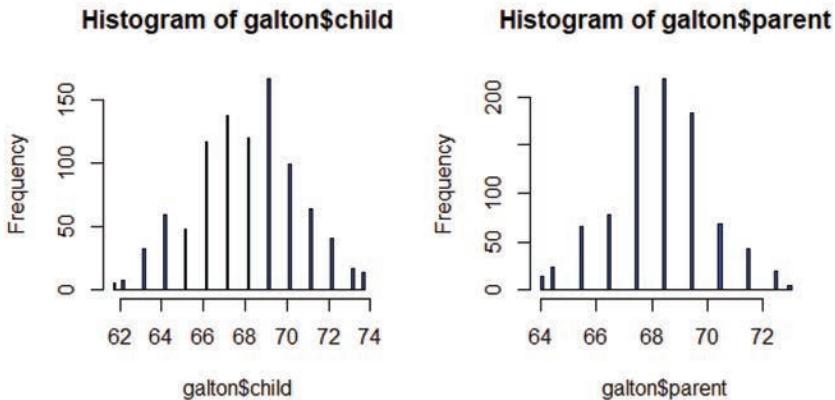
The midparent's height is an average of the father's height and 1.08 times the mother's. In the data there are 205 different parents and 928 children. The data here is truncated at the ends for both parents and children so that it can be treated as numeric data. The data were tabulated and consequently made discrete. The `father.son` data set is similar data used by Galton and is continuous. You may need to run `install.packages("UsingR")` if the UsingR library is not installed. Let's look at the marginal (parents disregarding children and children disregarding parents) distributions first (see **Figure 7-1**).

- Parent distribution is all heterosexual couples.
- Correction for gender via multiplying female heights by 1.08.
- Overplotting is an issue from discretization.

```

library(UsingR)
data(galton)
par(mfrow=c(1,2))
hist(galton$child, col="blue", breaks=100)
hist(galton$parent, col="blue", breaks=100)

```



**Figure 7-1.** Histograms of parent and children heights as recorded in the Galton dataset

### Finding the middle via least squares

In statistics, ordinary least squares (OLS) or linear least squares is a method for estimating the unknown parameters in a linear regression model. This method minimizes the sum of squared vertical distances between the observed responses in the dataset and the responses predicted by the linear approximation. The resulting estimator can be expressed by a simple formula, especially in the case of a single regressor on the right-hand side. Consider only the children's heights. How could one describe the "middle"?

**Definition 7-1.** Let  $Y$  be the height of child  $i$  for  $i = 1 \dots, n = 928$ , then define the middle as the value of  $\mu$  that minimizes

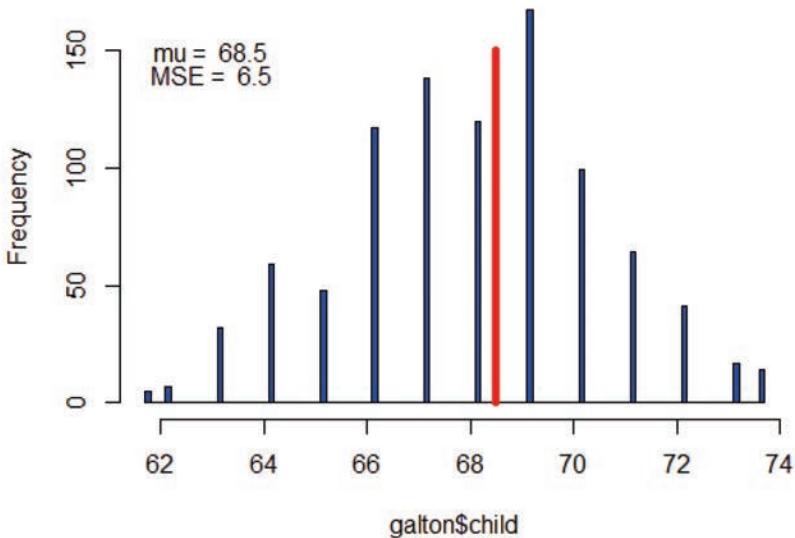
$$\sum_{i=1}^n (Y_i - \mu)^2$$

This is physical center of mass of the histogram. You might have guessed that the answer  $\mu = \bar{X}$ .

## Experiment

Use RStudio's manipulate to see what value of  $\mu$  minimizes the sum of the squared deviations (see **Figure 7-2**).

```
library(manipulate)
myHist <- function(mu){
  hist(galton$child, col= "blue", breaks=100)
  lines(c(mu, mu), c(0, 150), col="red", lwd=5)
  mse <- mean((galton$child - mu)^2)
  text(63, 150, paste("mu = ", mu))
  text(63, 140, paste("MSE = ", round(mse, 2)))
}
manipulate(myHist(mu), mu = slider(62, 74, step = 0.5))
```

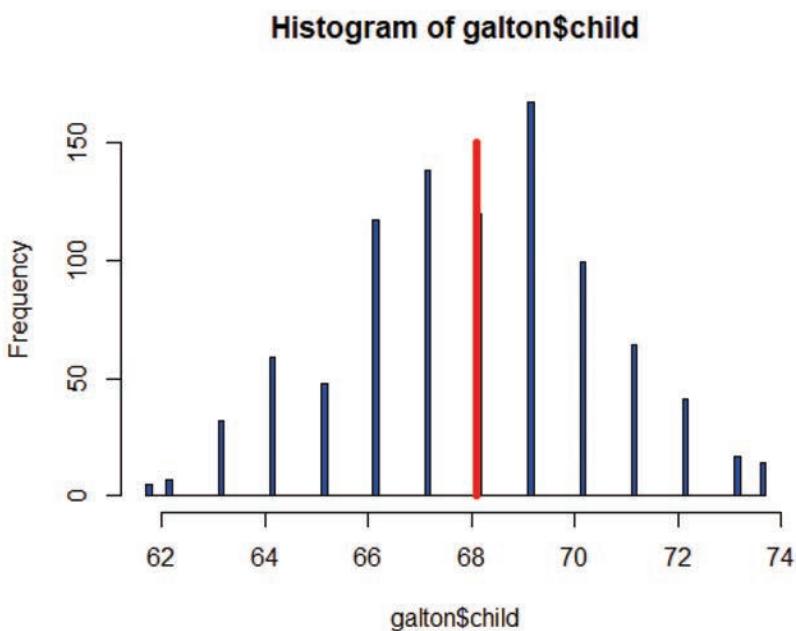


**Figure 7-2.** Histogram of children heights with approximated mean value represented by the red abline with  $MSE = 6.5$  and  $\mu = 68.5$

The least squares estimate is the empirical mean as shown in **Figure 7-3**.

```
hist(galton$child, col = "blue", breaks = 100)
meanChild <- mean(galton$child)
```

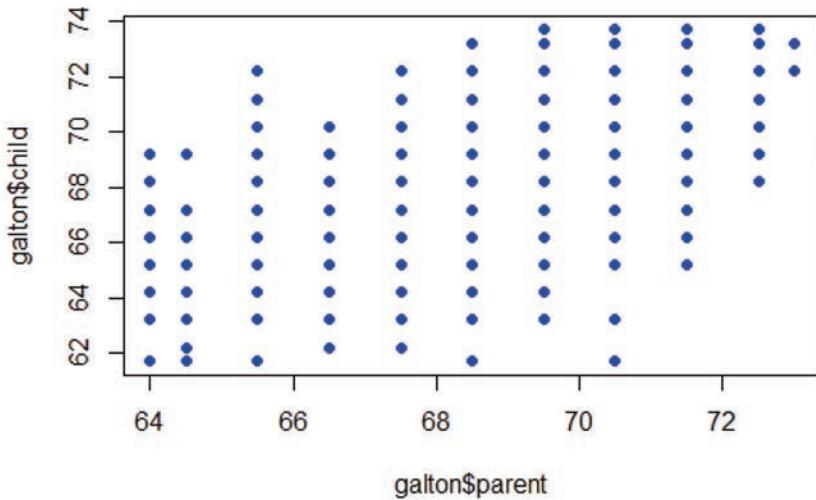
```
lines(rep(meanChild,100), seq(0,150,length=100),col="red",  
lwd=5)
```



**Figure 7-3.** Histogram of actual sample mean of children heights, represented by the red abline

Comparing childrens' heights and their parents' heights is shown in the scatter plot in **Figure 7-4**.

```
plot(galton$parent, galton$child, pch=19, col = "blue")
```

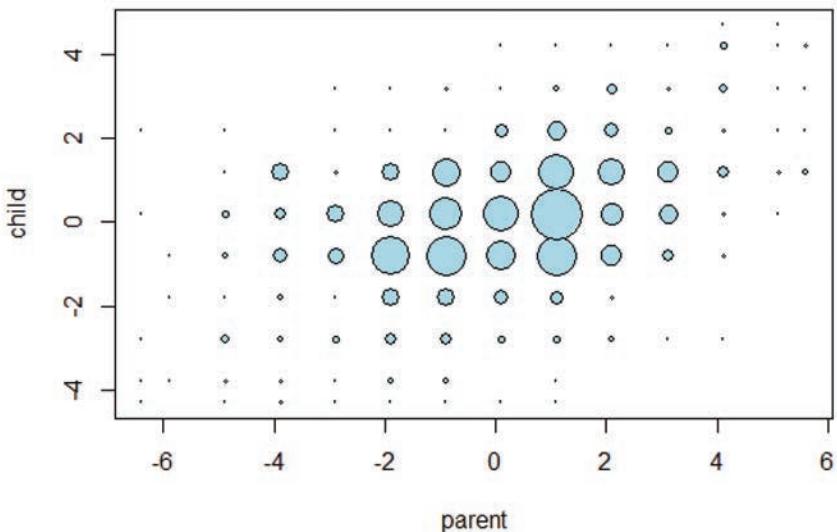


**Figure 7-4.** Scatterplot of parents heights versus children heights

### General least squares for linear equations

Consider again the parent and child height data from Galton. In **Figure 7-5**, we present a scatterplot with point size a function of frequency. That is, the more record with equal heights, the larger the bubble, i.e., 12 records with height equal to 53 inches, appears as a larger bubble than 2 records with height equal to 60 inches.

```
myPlot <- function(beta){
  y <- galton$child - mean(galton$child) # Child weight
  x <- galton$parent - mean(galton$parent) # Parent
  weight
  freqData <- as.data.frame(table(x, y))
  names(freqData) <- c("child", "parent", "freq")
  plot(
    as.numeric(as.vector(freqData$parent)),
    as.numeric(as.vector(freqData$child)),
    pch = 21, col = "black", bg = "lightblue",
    cex = .15 * freqData$freq, # Bubble size
    xlab = "parent",
    ylab = "child"
  )
}
myPlot(beta)
```



**Figure 7-5. Scatterplot of parent versus children heights**

### Fitting the best line

Since we are interested in summarizing the trend between two quantitative variables, the natural question arises — "what is the best fitting line?" At some point in your education, you were probably shown a scatter plot of  $(x, y)$  data and were asked to draw the "most appropriate" line through the data. Even if you weren't, you can try it now on a set of parent's heights ( $x$ ) and children's heights ( $y$ ) of 928 pairs, (Galton dataset). Now, what best summarizes the trend between height and weight?

In order to answer the question of better fit, we first need to introduce some common notation:

$y_i$  denotes the observed response for experimental unit  $i$

$x_i$  denotes the predictor value for experimental unit  $i$

$\hat{y}_i$  is the predicted response (or fitted value) for experimental unit  $i$

Then, the equation for the best fitting line is:

$$\hat{y}_i = b_0 + b_1 x_i$$

Incidentally, recall that an "**experimental unit**" is the object or person on which the measurement is made. In our Galton example, the experimental units are the children, i.e., can we predict the height of the child based on the height of the parent?

### *Regression through the origin*

To put this in terms of the simple linear regression model, suppose that  $X_i$  are the parents' heights, and  $Y_i$  are the heights of the children. Then, consider picking the slope  $\beta$  that minimizes. In other words, we want to find the slope that minimizes the squared distance between the heights of the parents and children.

$$\sum_{i=1}^n (Y_i - \beta X_i)^2$$

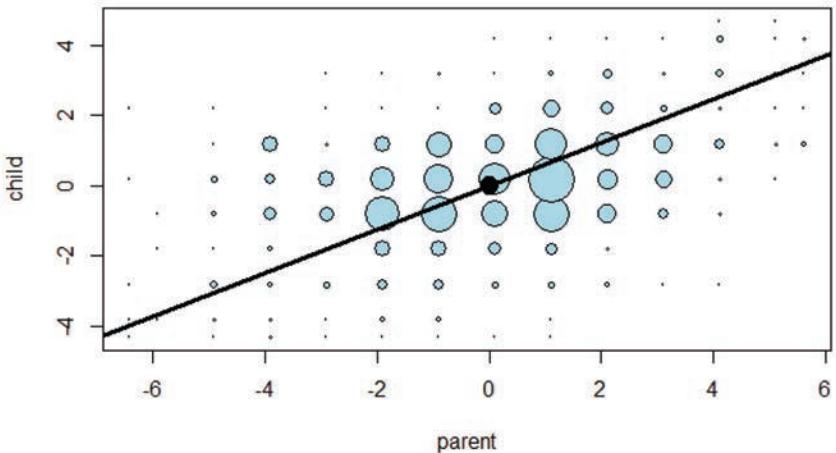
This is exactly using the origin as a pivot point picking the line that minimizes the sum of the squared vertical distances of the points to the line. We will use R's `manipulate()` function to experiment. And, we'll subtract the means so that the origin is the mean of the parent and children's heights. We make three plots (**Figure 7-6**, **Figure 7-7**, **Figure 7-8**) at three different values of  $\beta$  (0.62, 0.64, and 0.66, respectively) using the `manipulate()` function.

```
myPlot <- function(beta){
  y <- galton$child - mean(galton$child)
  x <- galton$parent - mean(galton$parent)
  freqData <- as.data.frame(table(x, y))
  names(freqData) <- c("child", "parent", "freq")
  plot(
    as.numeric(as.vector(freqData$parent)) ,
    as.numeric(as.vector(freqData$child)),
    pch = 21, col = "black", bg = "lightblue",
    cex = .15 * freqData$freq,
    xlab = "parent",
    ylab = "child"
  )
  abline(0, beta, lwd = 3) # Added for best fit line
  points(0, 0, cex = 2, pch = 19) # Added for center
  mse <- mean((y - beta * x)^2) # Added for metric
}
```

```

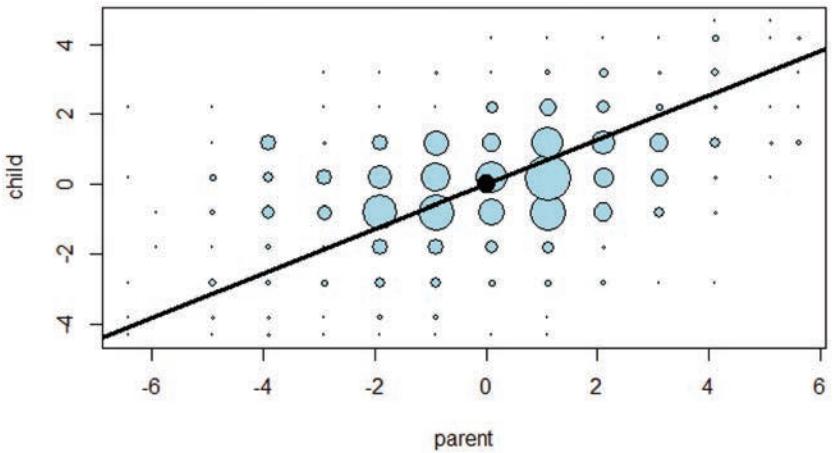
    title(paste("beta = ", beta, "mse = ", round(mse, 3)))
}
manipulate(myPlot(beta), beta=slider(0.6, 1.2, step=0.02))
beta = 0.62 mse = 5.002

```

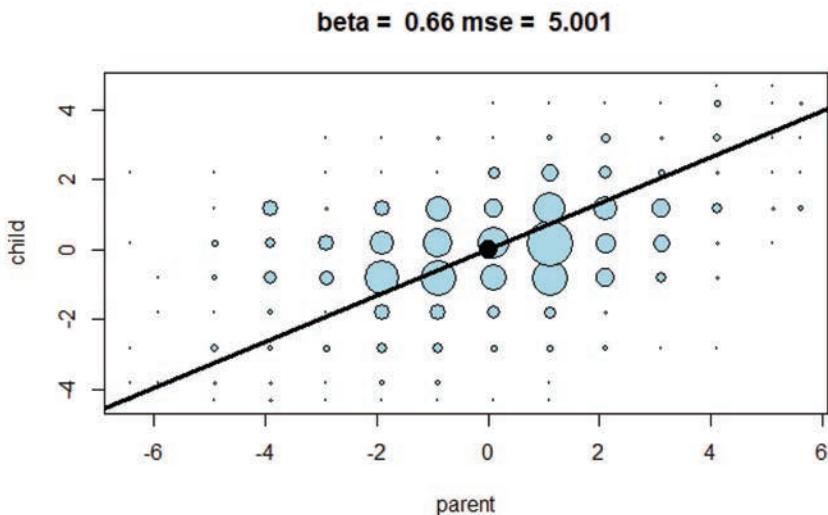


**Figure 7-6.** Parent versus children scatterplot and fitted line with  $\beta=0.62$  and  $mse = 5.002$

**beta = 0.64 mse = 5**



**Figure 7-7.** Parent versus children scatterplot and fitted line with  $\beta=0.64$  and  $mse = 5.000$



**Figure 7-8.** Parent versus children scatterplot and fitted line with  $\beta=0.666$  and  $mse = 5.001$

### The solution

```
lm(I(child - mean(child)) ~ I(parent - mean(parent)) - 1,
  data = galton)
Call:
lm(formula = I(child - mean(child)) ~ I(parent -
  mean(parent)) -
  1, data = galton)
```

Coefficients:

I(parent - mean(parent))	0.6463
--------------------------	--------

```
yhat <- predict(fit, type = 'response')
mse(fit, yhat)
```

[1] 5.000294

So, the beta that minimizes is 0.6463, which we approximated as 0.64 with our manipulation of the abline plots. And, the MSE is 5.000294, which we estimated as 5.0.

Now, we double check our approximations using R.

```
y <- galton$child
```

```

x <- galton$parent
beta1 <- cor(y, x) * sd(y) / sd(x)
beta0 <- mean(y) - beta1 * mean(x)
rbind(c(beta0, beta1), coef(lm(y ~ x)))

```

	(Intercept)	x
[1,]	23.94153	0.6462906
[2,]	23.94153	0.6462906

Reversing the outcome/predictor relationship

```

beta1 <- cor(y, x) * sd(x) / sd(y)
beta0 <- mean(x) - beta1 * mean(y)
rbind(c(beta0, beta1), coef(lm(x ~ y)))

```

	(Intercept)	y
[1,]	46.13535	0.3256475
[2,]	46.13535	0.3256475

Regression through the origin yields an equivalent slope if you center the data first

```

yc <- y - mean(y)
xc <- x - mean(x)
beta1 <- sum(yc * xc) / sum(xc ^ 2)
c(beta1, coef(lm(y ~ x))[2])

```

	x
0.6462906	0.6462906

Normalizing variables results in the slope being the correlation

```

yn <- (y - mean(y))/sd(y)
xn <- (x - mean(x))/sd(x)
c(cor(y, x), cor(yn, xn), coef(lm(yn ~ xn))[2])

```

	xn
0.4587624	0.4587624

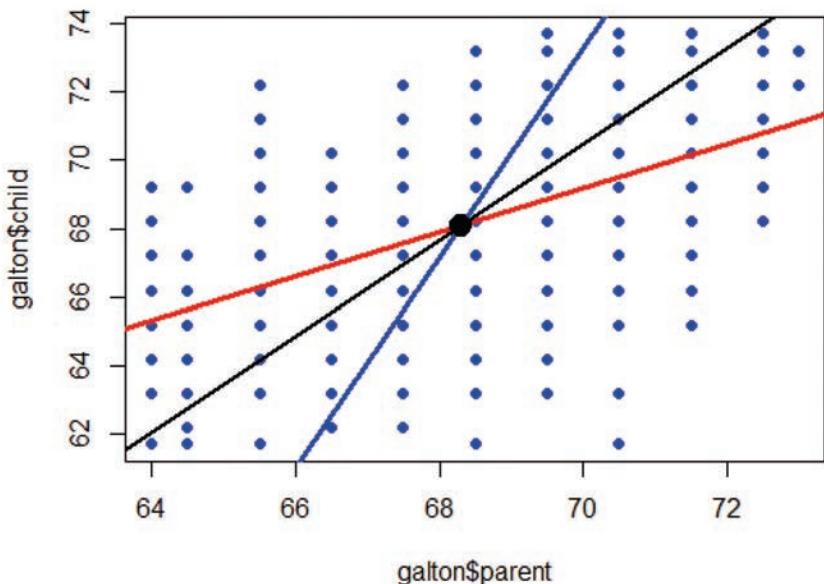
The code to add the lines, with the black line representing the best fit line in **Figure 7-9**.

```

abline(mean(y) - mean(x) * cor(y, x) * sd(y) / sd(x),
      sd(y) / sd(x) * cor(y, x), lwd = 3, col = "red")
abline(mean(y) - mean(x) * sd(y) / sd(x) / cor(y, x) ,
      sd(y) / cor(y, x) / sd(x), lwd = 3, col = "blue")
abline (mean(y) - mean(x) * sd(y) / sd(x),
      sd(y) / sd(x), lwd = 2)

```

```
points(mean(x), mean(y), cex = 2, pch = 19)
```



**Figure 7-9.** Parent versus children heights with best fit line depicted as the black abline.

### Interpreting regression coefficients, the slope

The parameter  $\beta_1$  is the expected change in response for a 1-unit change in the predictor.

$$E[Y|X = x + 1] - E[Y|X = x] = \beta_0 + \beta_1(x + 1) - (\beta_0 + \beta_1x) = \beta_1$$

**Definition 7-2.** The **expected value** of a discrete random variable  $X$ , symbolized as  $E(X)$ , is the average value we would expect of performing an experiment over and over, often referred to as the long-term average or mean.

$$E[X] = \sum_{i=1}^n X_i \times P(X_i)$$

where  $P(X_i)$  are the probabilities of an individual outcome.

**Definition 7-3.** The **expected value**,  $E[X]$ , of a continuous random value hold the same meaning as  $E[X]$  for the discrete case, but is given by:

$$E[X] = \int_{\mathbb{R}} x \times f(x) dx$$

where  $f(x)$  is as probability distribution function.

## Residuals

Now, any time we perform regression on random samples, we introduce a random error epsilon or  $\varepsilon$ , whereby our regression model is

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

where  $\varepsilon_i$  is follows a normal distribution  $N(0, \sigma^2)$ .

Now, least squares minimize the sum of the squared residuals, that is least squares minimize  $\sum_{i=1}^n e_i^2$ , where  $e_i$  are the observed errors. That it  $e_i$  is the observed and the predicted outcome:

$$e_i = Y_i - \hat{Y}_i$$

So,  $e_i$  is an estimate of  $\varepsilon_i$ .

### Properties of the residuals

- $E[e_i] = 0$
- If an intercept is included,  $\sum_{i=1}^n e_i = 0$
- If a regressor variable,  $X_i$ , is included, then  $\sum_{i=1}^n e_i X_i = 0$
- Residuals are useful for investigating poor model fit.
- Positive residuals are above the line, negative residuals are below.
- Residuals can be thought of as the outcome (Y) with the linear association of the predictor (X) removed.
- One differentiates residual variation (variation after removing the predictor) from systematic variation (variation explained by the regression model).
- Residual plots highlight poor model fit.

To calculate the fitted model residuals, we use the `resid()` function.

```
res <- resid(fit)
```

Now, to produce the residual plot, we generate a scatterplot of the fitted model, using the `fitted()` function, versus the residuals. `fitted()` is a generic function which extracts fitted values from objects returned by modeling functions. The top plot in **Figure 7-10**.

```
# produce residual vs. fitted plot
plot(fitted(fit), res, lwd=2, col="red")
# add a horizontal line at 0
abline(0, 0, lwd=2, col="blue")
```

### *Quantile-Quantile Plots*

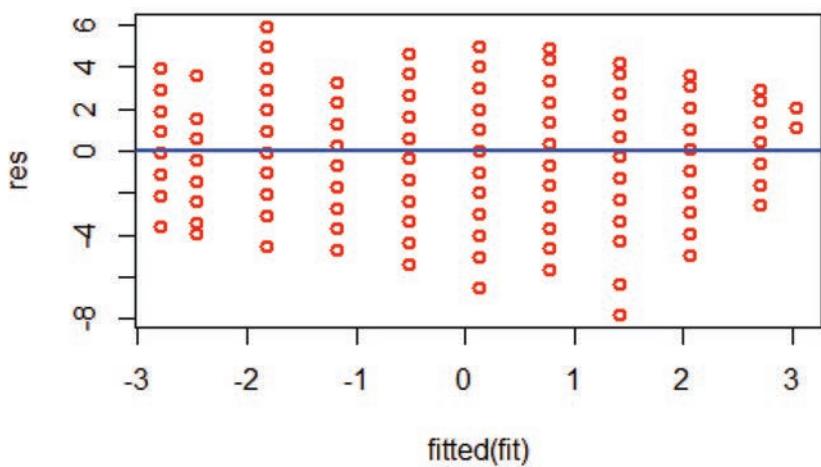
We can also produce a Q-Q plot, which is useful for determining if the residuals follow a normal distribution. If the data values in the plot fall along a roughly straight diagonal line, then the data is normally distributed.

The `qqnorm()` function the default method for producing a normal QQ plot of the values in `y`. The `qqline()` function adds a line to a “theoretical” (normal by default), quantile-quantile plot, which passes through the probability quantiles (first and third quartiles by default). These appear as the bottom plot in **Figure 7-10**. Also plotted in **Figure 7-11** are the scale-location and residuals-leverage plots.

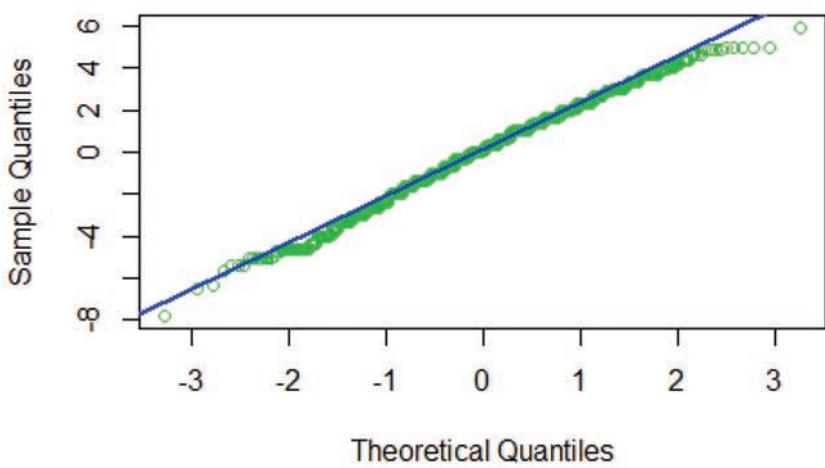
```
#create Q-Q plot for residuals
qqnorm(res, col="green3")
#add a straight diagonal line to the plot
qqline(data.frame(res), lwd=2, col="blue")
```

### *Density Plot of the Residuals*

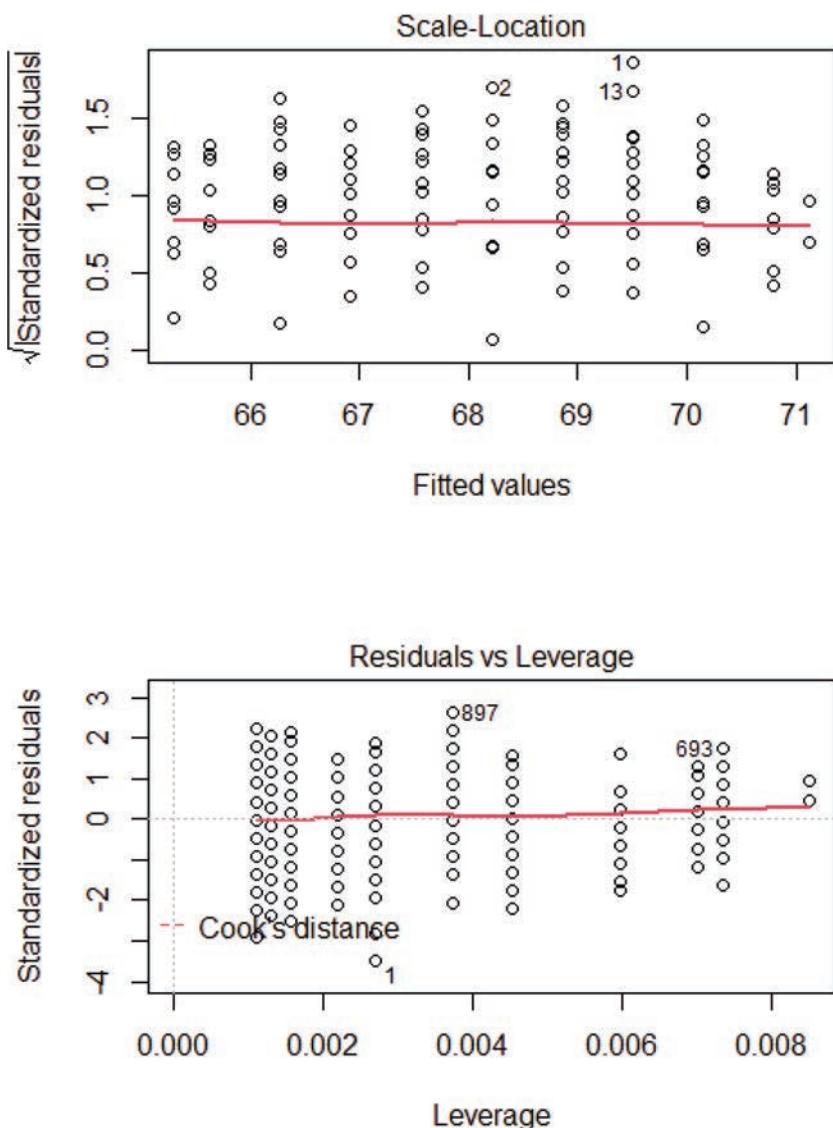
We can also produce a density plot, which is also useful for visually checking whether or not the residuals are normally distributed. If the plot is roughly bell-shaped, then the residuals likely follow a normal distribution (see **Figure 7-12**).



**Normal Q-Q Plot**

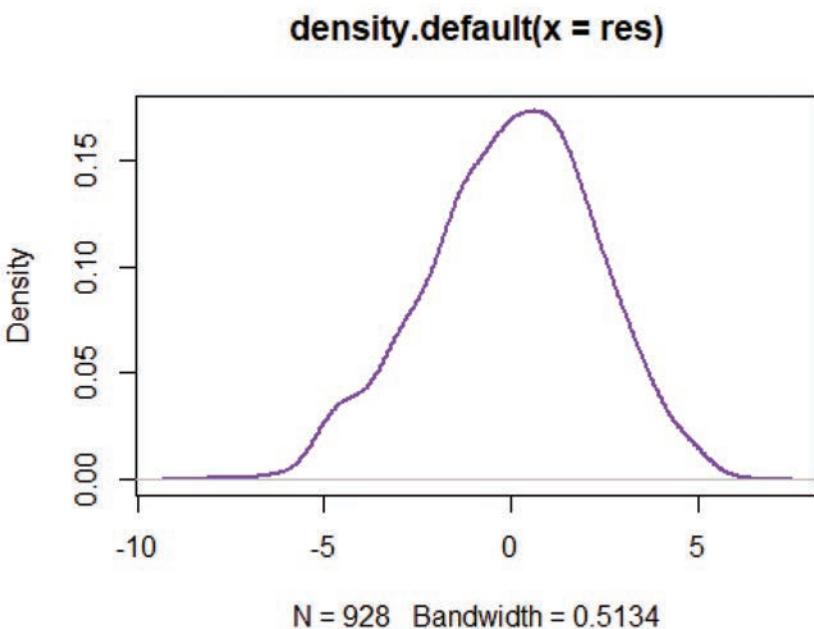


**Figure 7-10.** (top) The residual plot, and (bottom) the Q-Q plot



**Figure 7-11.** (top) residuals scale location and (bottom) residuals versus leverage

```
#Create density plot of residuals
plot(density(res), lwd=2, col="purple")
```



**Figure 7-12.** Residual density plot to check residual normality

### Estimating residual variation

```
summary(fit)$sigma  
[1] 2.237339
```

### Adjustment

Adjustment, is the idea of putting regressors into a linear model to investigate the role of a third variable on the relationship between another two. It is often the case that a third variable can distort, or confound, the relationship between two others.

As an example, consider looking at lung cancer rates and breath mint usage. For the sake of completeness, imagine if you were looking at forced expiratory volume (a measure of lung function) and breath mint usage. If you found a statistically significant regression relationship, it wouldn't be wise to rush off to the newspapers with the headline "Breath mint usage causes shortness

of breath!”, for a variety of reasons. First, even if the association is sound, you don’t know that it’s causal. But, more importantly in this case, the likely culprit is smoking habits. Smoking rates are likely related to both breath mint usage rates and lung function. How would you defend your finding against the accusation that it’s just variability in smoking habits?

If our findings held up among non-smokers and smokers analyzed separately, then our findings might be correct. In other words, people would not believe this finding, unless it held up, while holding smoking status constant. This is the idea of adding a regression variable into a model as an **adjustment**. The coefficient of interest is interpreted as the effect of the predictor on the response, holding the adjustment variable constant.

## Example 7-2. Motor Trend Magazine MPG Study

This report answers questions pertaining to Motor Trend, a magazine about the automobile industry. They wanted to understand the relationship between a set of variables and miles per gallon (MPG) (outcome). They were particularly interested in the following two questions:

1. “Is an automatic or manual transmission better for MPG”
2. “Quantify the MPG difference between automatic and manual transmissions”

The results show that manual transmission vehicles provide better gas mileage than automatic transmissions. It also shows quantifies the MPG difference between automatic and manual transmissions, with acceleration and vehicle weight as additional factors affecting gas mileage. The source code for this example is located at:  
[https://github.com/stricje1/linear\\_regression](https://github.com/stricje1/linear_regression).

### Data Processing

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

```
library(datasets)
```

```
data(mtcars)
help(mtcars)
Data Description
```

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

It consists of 32 observations on 11 variables.

1. [, 1] mpg Miles/(US) gallon
2. [, 2] cyl Number of cylinders
3. [, 3] disp Displacement (cu.in.)
4. [, 4] hp Gross horsepower
5. [, 5] drat Rear axle ratio
6. [, 6] wt Weight (lb/1000)
7. [, 7] qsec 1/4-mile time
8. [, 8] vs Engine (0 = V-shaped, 1 = straight)
9. [, 9] am Transmission (0 = automatic, 1 = manual)
10. [,10] gear Number of forward gears
11. [,11] carb Number of carburetors

### *Exploratory Data Analysis (EDA)*

The data summaries for automatic and manual transmissions are listed in Appendix A, following the conclusion to this example.

```
library(dplyr)
my_quantile <- function(x, probs) {
  tibble(x = quantile(x, probs), probs = probs)
}
mtcars %>%
  group_by(am)  %>%
  summarise(xbar = mean(mpg), sd = sd(mpg))
## # A tibble: 2 x 3
##   am   xbar    sd
##   <dbl> <dbl> <dbl>
## 1  0    17.1  3.83
## 2  1    24.4  6.17
```

More

```
mtcars %>%
  group_by(am) %>%
```

```
summarise(my_quantile(mpg, c(0.25, 0.5, 0.75)))
## # A tibble: 6 x 3
## # Groups: am [2]
##   am x probs
##   <dbl> <dbl> <dbl>
## 1 0    15.0  0.25
## 2 0    17.3  0.5
## 3 0    19.2  0.75
## 4 1    21.0  0.25
## 5 1    22.8  0.5
## 6 1    30.4  0.75
```

## Data Preparation

To perform this analysis, we need to transform the class of some variables. The function `factor()` is used to encode a vector as a factor.

```
mtcars$am <- as.factor (mtcars$am)
levels(mtcars$am) <-c("Automatic", "Manual")
```

**Example 7-3.** Determine whether an automatic or manual transmission better for MPG.

Here we quantify the MPG difference between automatic and manual transmissions. First, we generate a boxplot of MPG by transmission types at Appendix B, following the conclusion to this example. The boxplot shows that vehicles with manual transmissions may achieve better gas mileage than vehicles with automatic transmissions.

## Hypothesis Test

Now we address Question 1 statistically. The null hypothesis we are testing is:

$H_0$ : There is no different in mpg between automatic and manual transmissions.

We conduct a t-test to test this hypothesis.

```
t_test<-ttest(mtcars$mpg~mtcars$am, conf.level = 0.95)
df <- data.frame(
  "Welch Two Sample t-test"=rep(c('t-test p-value', 'Lower
  CI', 'Upper CI', 'Automatic
```

```

Estimate', 'Manual Estimate', ''), times = 1),
Value=rep(c(sprintf("%.6f", t_test$p.value),
sprintf("%.6f", t_test$conf.int) ,
sprintf("%.6f", t_test$estimate), times = 1)))
knitr::kable(df)

```

Welch.Two.Sample.t.test	Value
t-test p-value	0.001374
Lower CI	-11.280194
Upper CI	-3.209684
Automatic Estimate	17.147368
Manual Estimate	24.392308

Based on the results,  $p\text{-value} = 0.001374 < 0.05$ , we reject the null hypothesis that there is no difference in MPG, and conclude that manual transmission is better than automatic transmission for MPG, with assumption that all other conditions remain unchanged.

### *MPG difference between automatic and manual transmissions*

Here we try to quantify the MPG difference between transmission types, and find if there are other variables that account for the MPG differences.

First, do a multivariate linear regression with all variables. We use the step function in R for a step-wise regression, where the choice of predictor is carried out automatically by comparing certain criterion, for example, Akaike information criterion (AIC).

```

stepmodel = step(lm(data = mtcars, mpg ~ .) ,
trace=0, steps=10000)
summary(stepmodel)
##
## Call:
## lm(formula = mpg ~ wt + qsec + am, data = mtcars)
##
## Residuals:
##   Min 1Q Median 3Q Max
## -3.4811 -1.5555 -0.7257 1.4110 4.6610
##
## Coefficients:

```

```

## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 9.6178 6.9596 1.382 0.177915
## wt -3.9165 0.7112 -5.507 6.95e-06 ***
## qsec 1.2259 0.2887 4.247 0.000216 ***
## amManual 2.9358 1.4109 2.081 0.046716 *
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05
'.' 0.1 ' ' 1
##
## Residual standard error: 2.459 on 28 degrees of
## freedom
## Multiple R-squared: 0.8497, Adjusted R-squared:
## 0.8336
## F-statistic: 52.75 on 3 and 28 DF, p-value:
## 1.21e-11

```

To further optimize the model, we can examine  $\text{mpg} \sim \text{wt} + \text{qsec}$  correlation with  $\text{am}$ .

```

model <- lm(mpg ~ factor(am):wt +
             factor(am):qsec, data=mtcars)
summary(model)

##
## Call:
## lm(formula = mpg ~ factor(am):wt + factor(am):qsec,
## data = mtcars)
##
## Residuals:
## Min 1Q Median 3Q Max
## -3.9361 -1.4017 -0.1551 1.2695 3.8862
##
## Coefficients:
## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.9692 5.7756 2.419 0.02259 *
## factor(am)Automatic:wt -3.1759 0.6362 -4.992
## 3.11e-05 ***
## factor(am)Manual:wt -6.0992 0.9685 -6.297 9.70e-
## 07 ***
## factor(am)Automatic:qsec 0.8338 0.2602 3.205
## 0.00346 **

```

```

## factor(am)Manual:qsec 1.4464 0.2692 5.373 1.12e-
05 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05
'.' 0.1 ' ' 1
##
## Residual standard error: 2.097 on 27 degrees of
## freedom
## Multiple R-squared: 0.8946, Adjusted R-squared:
## 0.879
## F-statistic: 57.28 on 4 and 27 DF, p-value:
## 8.424e-13

```

The results suggests that the best model includes `weight` and `qsec`. The Adjusted R-squared increased from 0.83 to 0.88. The adjusted R-squared increases when the new term improves the model more than would be expected by chance. Thus, the model improved by adding the factors.

- that the model explains 88% of the variability of the response data around its mean. Now we can make the following conclusions:
- when the weight increased by 1000 lbs, the mpg decreased by -3.176 for automatic transmission cars, and -6.09 for manual transmission cars
- with increasing car weight, we should choose automatic transmission cars for better gas mileage
- when the acceleration speed dropped, and 1/4-mile time increased (by 1 sec), the mpg factor increased by 0.834 miles for automatic transmission cars, and 1.446 miles for
- manual transmission cars with lower acceleration speed and same weight, manual transmission cars get better gas mileage {mpg}

Residual plots seem to be randomly scattered, and some transformation may be needed for linearity (Appendix C). The Normal Q-Q plot shows the residuals are approximately normally distributed. Both scale-location and residual vs leverage show no issues.

## Conclusion

The hypothesis tests in part 1 showed that manual transmission vehicles are better than automatic transmission vehicles with no other factors considered. The factors in the optimized model quantifies the difference between automatic and manual transmissions, based on vehicle weight and acceleration, shows that manual transmission vehicles provide better gas mileage (increased mpg).

## Appendix A

Data summary for automatic transmissions:

```
summary(mtcars[mtcars$am==0,])
## mpg cyl disp hp drat
## Min. : NA Min. : NA Min. : NA Min. : NA Min. : NA
## 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA
## Median : NA Median : NA Median : NA Median : NA Median : NA
## Mean :NaN Mean :NaN Mean :NaN Mean :NaN Mean :NaN
## 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA
## Max. : NA Max. : NA Max. : NA Max. : NA Max. : NA
## wt qsec vs am gear
## Min. : NA Min. : NA Min. : NA Automatic:0 Min. : NA
## 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA Manual :0 1st Qu.: NA
## Median : NA Median : NA Median : NA Median : NA
## Mean :NaN Mean :NaN Mean :NaN Mean :NaN
## 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA
## Max. : NA Max. : NA Max. : NA Max. : NA
## carb
## Min. : NA
## 1st Qu.: NA
## Median : NA
## Mean :NaN
## 3rd Qu.: NA
## Max. : NA
```

Data summary for manual transmissions:

```
summary(mtcars[mtcars$am==1,])
## mpg cyl disp hp drat
```

```

## Min. : NA Min. : NA Min. : NA Min. : NA Min. : NA
## 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA
1st Qu.: NA
## Median : NA Median : NA Median : NA Median : NA
Median : NA
## Mean :NaN Mean :NaN Mean :NaN Mean :NaN Mean :NaN
## 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA
3rd Qu.: NA
## Max. : NA Max. : NA Max. : NA Max. : NA Max. : NA
## wt qsec vs am gear
## Min. : NA Min. : NA Min. : NA Automatic:0 Min. :
NA
## 1st Qu.: NA 1st Qu.: NA 1st Qu.: NA Manual :0 1st
Qu.: NA
## Median : NA Median : NA Median : NA Median : NA
## Mean :NaN Mean :NaN Mean :NaN Mean :NaN
## 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA 3rd Qu.: NA
## Max. : NA Max. : NA Max. : NA Max. : NA
## carb
## Min. : NA
## 1st Qu.: NA
## Median : NA
## Mean :NaN
## 3rd Qu.: NA
## Max. : NA

```

## *Appendix B*

```

boxplot(mpg~am, data = mtcars, xlab =
"Transmission", ylab = "Miles per Gallon",
main = "MPG by Transmission Type")
Appendix C
Residual plots
par(mfrow = c(2,2))
plot(model)

```

## 8. Generalized Linear Models

The generalized linear model is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution. The GLM generalizes the linear regression we looked at in Chapter 2 by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

GLMs were formulated by John Nelder and Robert Wedderburn as a way of unifying various other statistical models, including linear regression, logistic regression and Poisson regression (Nelder & Wedderburn, 1989). They proposed an iteratively reweighted least squares method for *maximum likelihood estimation* of the model parameters. Maximum-likelihood estimation remains popular and is the default method on many statistical computing packages. Other approaches, including Bayesian approaches and least squares fit to variance stabilized responses, have been developed.

In these models, the response variable  $y_i$  is assumed to follow an exponential family of distributions with mean  $\mu_i$ , which is assumed to be some (often nonlinear) function of  $x_i^T \beta$ . Some would call these “nonlinear” because  $\mu_i$  is often a nonlinear function of the covariates, but McCullagh and Nelder consider them to be linear, because the covariates affect the distribution of  $y_i$  only through the linear combination  $x_i^T \beta$ . (McCullagh & Nelder, 1989) The first widely used software package for fitting these models was called GLIM. Because of this program, “GLIM” became a well-accepted abbreviation for generalized linear models, as opposed to “GLM” which often is used for general linear models. Today, GLIM’s are fit by many packages, including the R function `glm()`.

The generalized linear models are a broad class of models that include linear regression, ANOVA, Poisson regression, log-linear models etc. The **Table 8-1** provides a good summary of GLMs.

**Table 8-1.** GLM Class of Models

Model	Random	Link	Systematic
Linear Regression	Normal	Identity	Continuous
ANOVA	Normal	Identity	Categorical
ANCOVA	Normal	Identity	Mixed
Logistic Regression	Binomial	Logit	Mixed
Log-Linear	Poisson	Log	Categorical
Poisson Regression	Poisson	Log	Mixed
Multinomial response	Multinomial	Generalized Logit	Mixed

## Intuition

Ordinary linear regression predicts the expected value of a given unknown quantity (the response variable, a random variable) as a linear combination of a set of observed values (predictors). This implies that a constant change in a predictor leads to a constant change in the response variable (i.e., a linear-response model). This is appropriate when the response variable has a normal distribution (intuitively, when a response variable can vary essentially indefinitely in either direction with no fixed “zero value”, or more generally for any quantity that only varies by a relatively small amount, e.g. human heights).

However, these assumptions are inappropriate for many types of response variables. For example, in many cases when the response variable must be positive and can vary over a wide scale, constant input changes lead to geometrically varying rather than constantly varying output changes. As an example, a model that predicts that each decrease in 10 degrees Fahrenheit leads to 1,000 fewer people going to a given beach is unlikely to generalize well over both small

beaches (e.g. those where the expected attendance was 50 at the lower temperature) and large beaches (e.g. those where the expected attendance was 10,000 at the lower temperature). An even worse problem is that, since the model also implies that a drop in 10 degrees leads 1,000 fewer people going to a given beach, a beach whose expected attendance was 50 at the higher temperature would now be predicted to have the impossible attendance value of -950. Logically, a more realistic model would instead predict a constant rate of increased beach attendance (e.g., an increase in 10 degrees leads to a doubling in beach attendance, and a drop in 10 degrees leads to a halving in attendance). Such a model is termed an exponential-response model (or log-linear model, since the logarithm of the response is predicted to vary linearly).

Similarly, a model that predicts a probability of making a yes/no choice (a Bernoulli variable) is even less suitable as a linear-response model, since probabilities are bounded on both ends (they must be between 0 and 1). Imagine, for example, a model that predicts the likelihood of a given person going to the beach as a function of temperature. A reasonable model might predict, for example, that a change in 10 degrees makes a person two times more or less likely to go to the beach. But what does “twice as likely” mean in terms of a probability? It cannot literally mean to double the probability value (e.g., 50% becomes 100%, 75% becomes 150%, etc.). Rather, it is the odds that are doubling: from 2:1 odds, to 4:1 odds, to 8:1 odds, etc. Such a model is a log-odds model.

Generalized linear models cover all these situations by allowing for response variables that have arbitrary distributions (rather than simply normal distributions), and for an arbitrary function of the response variable (the link function) to vary linearly with the predicted values (rather than assuming that the response itself must vary linearly). For example, the case above of predicted number of beach attendees would typically be modeled with a Poisson distribution and a log link, while the case of predicted probability of beach attendance would typically be modeled with a Bernoulli distribution (or binomial distribution, depending on exactly how the problem is phrased) and a log-odds (or logit) link function.

## Overview

In a generalized linear model (GLM), each outcome of the dependent variables,  $\mathbf{Y}$ , is assumed to be generated from a particular distribution in the exponential family, a large range of probability distributions that includes the normal, binomial, Poisson and gamma distributions, among others. The mean,  $\boldsymbol{\mu}$ , of the distribution depends on the independent variables,  $\mathbf{X}$ , through:

$$E(\mathbf{Y}) = \boldsymbol{\mu} = g^{-1}(\mathbf{X}\boldsymbol{\beta}), \quad (8.1)$$

where  $E(\mathbf{Y})$  is the expected value of  $\mathbf{Y}$  (see **Definition 7-2**);  $\mathbf{X}\boldsymbol{\beta}$  is the linear predictor, a linear combination of unknown parameters,  $\boldsymbol{\beta}$ ; and  $g$  is the link function.

In this framework, the variance is typically a function,  $V$ , of the mean:

$$\text{Var}(\mathbf{Y}) = V(\boldsymbol{\mu}) = V(g^{-1}(\mathbf{X}\boldsymbol{\beta})). \quad (8.2)$$

It is convenient if  $V$  follows from the exponential family distribution, but it may simply be that the variance is a function of the predicted value. The unknown parameters,  $\boldsymbol{\beta}$ , are typically estimated with maximum likelihood, maximum quasi-likelihood, or Bayesian techniques.

## Model components

The GLM consists of three elements:

1. **Random Component** - A probability distribution (i.e., the binomial distribution for binary logistic regression) of the response variable ( $Y$ ) from the exponential family.
2. **Systematic Component** - A linear predictor  $\eta = \mathbf{X}\boldsymbol{\beta}$ , specifies the explanatory variables ( $X_1, X_2, \dots, X_k$ ) in the model, more specifically their linear combination in creating the so called linear predictor;
3. A **link function**  $\eta$  (Greek eta) denotes a linear predictor  $\eta = g(\boldsymbol{\mu}) = \mathbf{X}\boldsymbol{\beta}$ , such that  $E(\mathbf{Y}) = \boldsymbol{\mu} = g^{-1}(\eta)$ , which specifies the link between random and systematic components. It says how the expected value of the response relates to the linear predictor of explanatory variables. For example,  $\eta = \text{logit}(\boldsymbol{\mu})$  is used for logistic

regression. When the link function is related to the mean of the distribution, we can it is the canonical form. That is, when the link function makes the linear predictor  $\eta_i$  the same as the canonical parameter  $\mu_i$ , we say that we have a *canonical link*. The identity is the canonical link for the normal distribution, the logit is the canonical link for the binomial distribution, and the natural logarithm,  $\ln$ , is the canonical link for the Poisson distribution. This leads to some natural pairings.

**Table 8-2** shows several exponential-family distributions in common use and the data they are typically used for, along with the canonical link functions and their inverses (sometimes referred to as the mean function, as done here).

In the cases of the exponential and gamma distributions, the domain of the canonical link function is not the same as the permitted range of the mean. In particular, the linear predictor may be negative, which would give an impossible negative mean. When maximizing the likelihood, precautions must be taken to avoid this. An alternative is to use a noncanonical link function.

Note also that in the case of the Bernoulli, binomial, categorical and multinomial distributions, the support of the distributions is not the same type of data as the parameter being predicted. In all of these cases, the predicted parameter is one or more probabilities, i.e., real numbers in the range  $[0,1]$ . The resulting model is known as logistic regression (or multinomial logistic regression in the case that  $K$ -way rather than binary values are being predicted).

For the Bernoulli and binomial distributions, the parameter is a single probability, indicating the likelihood of occurrence of a single event. The Bernoulli still satisfies the basic condition of the generalized linear model in that, even though a single outcome will always be either 0 or 1, the expected value will nonetheless be a real-valued probability, i.e., the probability of occurrence of a “yes” (or 1) outcome. Similarly, in a binomial distribution, the expected value is  $Np$ , i.e., the expected proportion of “yes” outcomes will be the probability to be predicted.

**Table 8-2.** Common distributions with typical uses and canonical link functions

Distribution	Support of distribution	Typical uses	Link name	Link function	Mean function
Normal	real: $(-\infty, \infty)$	Linear-response data	Identity	$X\beta = \mu$	$\mu = X\beta$
Exponential	real: $(0, \infty)$	Exponential-response data, scale parameters	Inverse	$X\beta = -\mu^{-1}$	$\mu = (-X\beta)^{-1}$
Gamma					
Inverse Gaussian	real: $(0, \infty)$				
Poisson	integer: $(0, \infty)$	count of occurrences in fixed amount of time/space	Log	$X\beta = \ln(\mu)$	$\mu = e^{X\beta}$
Bernoulli	integer: $[0, 1]$	outcome of single yes/no occurrence			
Binomial	integer: $[0, N]$	count of # of "yes" occurrences out of N yes/no occurrences	Logit	$X\beta = \ln\left(\frac{\mu}{1-\mu}\right)$	$\mu = \frac{1}{1+e^{X\beta}}$
Categorical	integer: $[0, K]$	K-vector of integer: $[0, 1]$ , where exactly one element in the vector has the value 1	outcome of single K-way occurrence		
Multinomial	K-vector of $[0, N]$			count of occurrences of different types $[1..K]$ out of N total K-way occurrences	

For categorical and multinomial distributions, the parameter to be predicted is a  $K$ -vector of probabilities, with the further restriction that all probabilities must add up to 1. Each probability indicates the likelihood of occurrence of one of the  $K$  possible values. For the multinomial distribution, and for the vector form of the categorical distribution, the expected values of the elements of the vector can be related to the predicted probabilities similarly to the binomial and Bernoulli distributions.

### **Assumptions:**

- The data  $Y_1, Y_2, \dots, Y_n$  are independently distributed, i.e., cases are independent.
- The dependent variable  $Y_i$  does NOT need to be normally distributed, but it typically assumes a distribution from an exponential family (e.g. binomial, Poisson, multinomial, normal,...)
- GLM does NOT assume a linear relationship between the dependent variable and the independent variables, but it does assume linear relationship between the transformed response in terms of the link function and the explanatory variables; e.g., for binary logistic regression  $\text{logit}(\mu) = \beta_0 + \beta_X$ .
- Independent (explanatory) variables can be even the power terms or some other nonlinear transformations of the original independent variables.
- The homogeneity of variance does NOT need to be satisfied. In fact, it is not even possible in many cases given the model structure, and overdispersion (when the observed variance is larger than what the model assumes) maybe present.
- Errors need to be independent but NOT normally distributed.
- It uses maximum likelihood estimation (MLE) rather than ordinary least squares (OLS) to estimate the parameters, and thus relies on large-sample approximations.

Goodness-of-fit measures rely on sufficiently large samples, where a heuristic rule is that not more than 20% of the expected cells counts are less than 5.

## Example 8-1 - Macroeconomics

We will use the *Longley* dataset, a macroeconomic data set which provides a well-known example for a *highly collinear regression*. The source code is located at: <https://github.com/stricje1/GLM>.

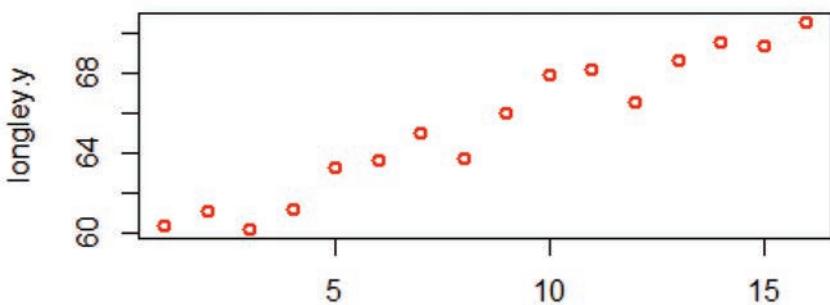
### Viewing the Data

A data frame with seven economic variables, observed yearly from 1947 to 1962 (n=16).

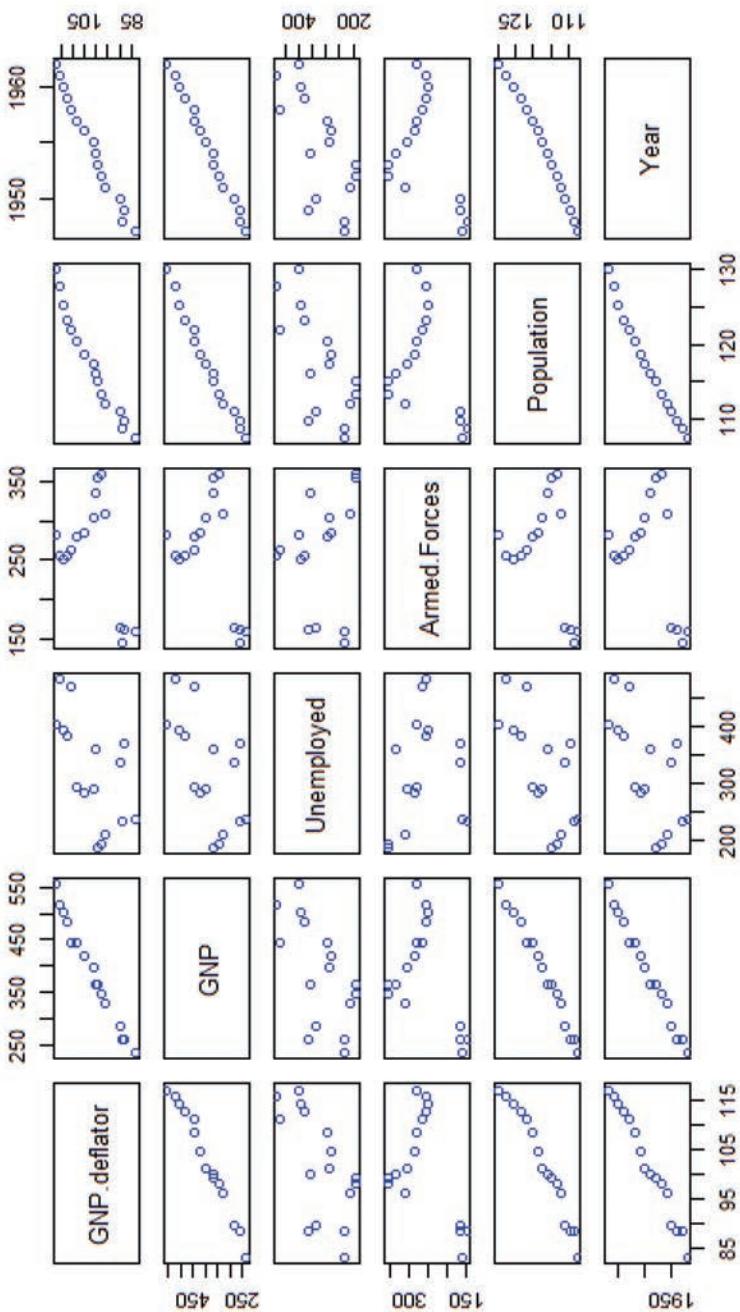
- GNP.deflator: GNP implicit price deflator (1954=100)
- GNP: Gross National Product
- Unemployed: number of unemployed
- Armed.Forces: number of people in the armed forces
- Population: 'noninstitutionalized' population  $\geq$  14 years of age
- Year: the year (time)
- Employed: number of people employed

**Figure 8-1** and **Figure 8-2** shows a scatterplot and matrix plot of the response and independent variables, respectively.

```
require(stats)
require(graphics)
longley.x <- data.matrix(longley[, 1:6])
longley.y <- longley[, "Employed"]
plot(longley.x, col="blue", lwd=2)
plot(longley.y, col="red", lwd=2)
```



**Figure 8-1.** Response variable (employment) scatter plot



**Figure 8-2.** Independent variables scatterplots matrix of the data

Here we will see the ability of R to discriminate between most appropriate function forms of the following models. We should expect, given the highly collinear nature of the regression `lm(Employed ~ .")` should be modeled with the Gaussian family.

- binomial (link = "logit")
- binomial (link = "probit")
- gaussian (link = "identity")
- poisson (link = "log")
- quasi (link = "identity", variance = "constant")

```
require(stats)
require(graphics)
longley.x <- data.matrix(longley[, 1:6])
summary(longley.x)
```

The summary function generates the following output, which shows us summary statics for each variable in the data set including the target variable, `"Employed"`.

	GNP.deflator	GNP	Unemployed
Min.	: 83.00	Min. :234.3	Min. :187.0
1st Qu.	: 94.53	1st Qu.:317.9	1st Qu.:234.8
Median	:100.60	Median :381.4	Median :314.4
Mean	:101.68	Mean :387.7	Mean :319.3
3rd Qu.	:111.25	3rd Qu.:454.1	3rd Qu.:384.2
Max.	:116.90	Max. :554.9	Max. :480.6
	Armed.Forces	Population	Year
Min.	:145.6	Min. :107.6	Min. :1947
1st Qu.	:229.8	1st Qu.:111.8	1st Qu.:1951
Median	:271.8	Median :116.8	Median :1954
Mean	:260.7	Mean :117.4	Mean :1954
3rd Qu.	:306.1	3rd Qu.:122.3	3rd Qu.:1958
Max.	:359.4	Max. :130.1	Max. :1962

Now, we set up a generalized linear model (we will talk about this in more detail in the next chapter), with `"Employed"` as the dependent variable and all the other variables as explanatory model effects. The statement `"Employed ~ ."` takes care of this requirement, while `"data=longley"` designates the data set, and `"family=Gaussian"` makes this a linear regression model having normal residuals.

```

longley.y <- longley[, "Employed"]
pairs(longley, main = "longley data")
longley.mod1<-glm(Employed ~ .,data=longley,family=gaussian
  n (identity))
summary(longley.mod1)

```

The “**Call**” statement merely shows us the GLM again. This is followed by output, including “Deviance Residuals,” which are critically important in logistic regression as we will see in later chapters. Then the output shows the model effect (explanatory variables) with their coefficient estimates and suitability statistics. At the end of the output, the Null Deviance and Residual Deviance are important goodness-of-fit metrics.

```

Call:
glm(formula = Employed ~ ., family = gaussian (identity),
data = longley)

Deviance Residuals:
    Min          1Q      Median          3Q      Max
-0.41011   -0.15767   -0.02816    0.10155   0.45539

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.482e+03  8.904e+02 -3.911 0.003560
GNP.deflator 1.506e-02  8.492e-02  0.177 0.863141
GNP          -3.582e-02  3.349e-02 -1.070 0.312681
Unemployed   -2.020e-02  4.884e-03 -4.136 0.002535
Armed.Forces -1.033e-02  2.143e-03 -4.822 0.000944
Population   -5.110e-02  2.261e-01 -0.226 0.826212
Year          1.829e+00  4.555e-01  4.016 0.003037

(Intercept) ***
GNP.deflator
GNP
Unemployed ***
Armed.Forces ***
Population
Year          **

---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

(Dispersion parameter for gaussian family taken to be 0.09  
293601)

Null deviance: 185.00883 on 15 degrees of freedom  
Residual deviance: 0.83642 on 9 degrees of freedom  
AIC: 14.187

Number of Fisher Scoring iterations: 2

The next section of R code gives use four GLM's with different link functions to compare: Gamma, Gaussian, Poisson, and Inverse Gaussian. When we run the code, we will get coefficient estimates for each model.

```
longley.mod1 <- glm(formula = Employed ~ .,  
family = Gamma(inverse), data = longley)  
longley.mod2 <- glm(formula = Employed ~ .,  
family = gaussian(identity), data = longley)  
longley.mod3 <- glm(formula = Employed ~ .,  
family = poisson(log), data = longley)  
longley.mod4 <- glm(formula = Employed ~ .,  
family = inverse.gaussian, data = longley)  
rbind (Gamma= longley.mod1$coef, gaussian= longley.mod2$co  
ef, poisson=longley.mod3$coeff, inverse.gaussian=longle  
y.mod4$coef)
```

	(Intercept)	GNP.deflator
Gamma	9.216863e-01	-7.325451e-06
gaussian	-3.482259e+03	1.506187e-02
poisson	-5.262691e+01	3.570455e-04
inverse.gaussian	2.907252e-02	-2.786624e-07
	GNP	Unemployed
Gamma	9.297934e-06	4.783348e-06
gaussian	-3.581918e-02	-2.020230e-02
poisson	-5.788587e-04	-3.111645e-04
inverse.gaussian	2.971755e-07	1.468006e-07
	Armed.Forces	Population
Gamma	2.126791e-06	3.201065e-05
gaussian	-1.033227e-02	-5.110411e-02
poisson	-1.489846e-04	-1.444313e-03
inverse. gaussian	5.999068e-08	1.277048e-06
	Year	
Gamma	-4.681685e-04	
gaussian	1.829151e+00	
poisson	2.931733e-02	

```
inverse.gaussian -1.490691e-05
```

The `rbind()` function takes a sequence of vector, matrix or data-frame arguments and combine by rows (`cbind()` does the same for columns). Here we want to look at a measure for model quality called AIC, or Akaike information criterion, for each of the four models. The `$aic` suffix after the model names does this for us. The lowest AIC value will indicate the best model.

```
rbind(Gamma= longley.mod1$aic, gaussian= longley.mod2$aic,  
      poisson=longley.mod3$aic, inverse.gaussian=longley.mod4  
      $aic)  
[ ,1]  
Gamma          15.42875  
gaussian       14.18670  
poisson         Inf  
inverse.gaussian 16.66554
```

It should be readily apparent that the Gaussian family with the identity link function works best with this collinear regression, and it is not improved upon by other functional forms.

## Example 8-2 – Air Quality

Daily air quality measurements in New York, May to September 1973. The data were obtained from the New York State Department of Conservation (ozone data) and the National Weather Service (meteorological data). The daily readings are for the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.

- Ozone: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
- Solar.R: Solar radiation in Langleys in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park
- Wind: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
- Temp: Maximum daily temperature in degrees Fahrenheit at La Guardia Airport.

The `airquality` data is part of the R datasets. It is a data frame with 153 observations on 6 variables.

```
[,1] Ozone numeric      Ozone (ppb)
[,2] Solar.R numeric   Solar R (lang)
[,3] Wind  numeric     Wind (mph)
[,4] Temp  numeric     Temperature (degrees F)
[,5] Month numeric    Month (1-12)
[,6] Day   numeric     Day of month (1-31)
```

For this example, we'll use a Poisson distribution, with canonical link function is  $g(\mu) = \ln(\mu)$ . Estimates on the original scale can be obtained by taking the inverse of the link function, in this case, the exponential function:  $\mu = \exp(X\beta)$ .

## Data preparation

We will take 70% of the airquality samples for training and 30% for testing:

```
data(airquality)
ozone <- subset(na.omit(airquality),
                 select = c("Ozone", "Solar.R", "Wind", "Temp"))
set.seed(123)
N.train <- ceiling(0.7 * nrow(ozone))
N.test <- nrow(ozone) - N.train
trainset <- sample(seq_len(nrow(ozone)), N.train)
testset <- setdiff(seq_len(nrow(ozone)), trainset)
```

## Training a GLM

For investigating the characteristics of GLMs, we will train a model, which assumes that errors are Poisson distributed.

By specifying `family = "poisson"`, `glm` automatically selects the appropriate canonical link function, which is the logarithm. More information on possible families and their canonical link functions can be obtained via `?family`.

```
model.pois <- glm(Ozone ~ Solar.R + Temp + Wind,
                    data = ozone, family = "poisson",
                    subset = trainset)
summary(model.pois)

##
## Call:
## glm(formula = Ozone ~ Solar.R + Temp + Wind,
```

```

## family = "poisson",
##       data = ozone, subset = trainset)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -4.0451   -2.4138   -0.8169    1.4265    8.7946
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
##(Intercept) 0.690227  0.218192  3.163  0.00156 **
## Solar.R    0.001815  0.000239  7.593 3.13e-14 ***
## Temp       0.043500  0.002295 18.956 < 2e-16 ***
## Wind       -0.084244  0.005960 -14.134 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
## '.' 0.1 ' ' 1
##
##(Dispersion parameter for poisson family taken to
## be 1)
##
## Null deviance: 1979.57 on 77 degrees of freedom
## Residual deviance: 530.51 on 74 degrees of freedom
## AIC: 951.97
##
## Number of Fisher Scoring iterations: 4

```

In terms of the GLM summary output, there are the following differences to the output obtained from the `lm` summary function:

- Deviance (deviance of residuals / null deviance / residual deviance)
- Other outputs: dispersion parameter, AIC, Fisher Scoring iterations

Moreover, the prediction function of GLMs is also a bit different. We will start with investigating the deviance.

## Deviance residuals

We already know residuals from the `lm` function. But what are deviance residuals? In ordinary least-squares, the residual associated with the  $i$ -th observation is defined as

$$r_i = y_i - \hat{f}(x_i)$$

where  $\hat{f}(x) = \beta_0 + x^T\beta$  is the prediction function of the fitted model.

For GLMs, there are several ways for specifying residuals. To understand deviance residuals, it is worthwhile to look at the other types of residuals first. For this, we define a few variables first:

```
expected <- ozone$Ozone[trainset]
g <- family(model.pois)$linkfun # log function
g.inv <- family(model.pois)$linkinv # exp function
# estimates on log scale
estimates.log <- model.pois$linear.predictors
# estimates on response scale (exponentiated)
estimates <- fitted(model.pois)
all.equal(g.inv(estimates.log), estimates)
## [1] TRUE
```

We will cover four types of residuals: response residuals, working residuals, Pearson residuals, and, deviance residuals. There is also another type of residual called *partial residual*, which is formed by determining residuals from models where individual features are excluded. This residual is not discussed here.

## Response residuals

For `type = "response"`, the *conventional* residual on the response level is computed, that is,

$$r_i = y_i - \hat{f}(x_i)$$

This means that the fitted residuals are transformed by taking the inverse of the link function:

```
# type = "response"
res.response1 <- residuals(model.pois, type = "response")
res.response2 <- expected - estimates
all.equal(res.response1, res.response2)
## [1] TRUE
```

## Working residuals

For `type = "working"`, the residuals are normalized by the estimates  $\hat{f}(x_i)$

$$r_i = y_i - \frac{\hat{f}(x_i)}{\hat{f}(x_i)}.$$

```
# type = "working"
res.working1 <- residuals(model.pois,
    type = "working")
res.working2 <- (expected - estimates) / estimates
all.equal(res.working1, res.working2)
## [1] TRUE
```

## Pearson residuals

For `type = "pearson"`, the Pearson residuals are computed. They are obtained by normalizing the residuals by the square root of the estimate:

$$r_i = \frac{y_i - \hat{f}(x_i)}{\sqrt{\hat{f}(x_i)}}.$$

```
# type = "pearson"
res.pearson1 <- residuals(model.pois,
    type = "pearson")
res.pearson2 <- (expected - estimates) /
    sqrt(estimates)
all.equal(res.pearson1, res.pearson2)
## [1] TRUE
```

## Deviance residuals

Deviance residuals are defined by the deviance. The deviance of a model is given by

$$D(y, \hat{\mu}) = 2(\log(p(y | \hat{\theta}_s)) - \log(p(y | \hat{\theta}_0))).$$

where

$y$  is the outcome

- $\hat{\mu}$  is the estimate of the model
- $\theta_s$  and  $\theta_0$  are the parameters of the fitted *saturated* and *proposed models*, respectively. A saturated model has as many parameters as it has training points, that is,  $p = n$
- Thus, it has a perfect fit. The proposed model can be the any other model.
- $p(y|\theta)$  is the likelihood of data given the model

The deviance indicates the extent to which the likelihood of the saturated model exceeds the likelihood of the proposed model. If the proposed model has a good fit, the deviance will be small. If the proposed model has a bad fit, the deviance will be high. For example, for the Poisson model, the deviance is

$$D = 2 \cdot \sum_{i=1}^n y_i \cdot \log(y_i \hat{\mu}_i) - (y_i - \hat{\mu}_i).$$

In R, the deviance residuals represent the contributions of individual samples to the deviance  $D$ . More specifically, they are defined as the signed square roots of the unit deviances. Thus, the deviance residuals are analogous to the conventional residuals: when they are squared, we obtain the sum of squares that we use for assessing the fit of the model. However, while the sum of squares is the residual sum of squares for linear models, for GLMs, this is the deviance.

How does such a deviance look like in practice? For example, for the Poisson distribution, the deviance residuals are defined as:

$$r_i = \text{sgn}(y - \hat{\mu}_i) \cdot \sqrt{2 \cdot y_i \cdot \log\left(\frac{y_i}{\hat{\mu}_i}\right) - (y_i - \hat{\mu}_i)}.$$

Let us verify this in R:

```
# type = "deviance"
res.dev1 <- residuals(model.pois, type = "deviance")
res.dev2 <- residuals(model.pois)
poisson.dev <- function (y, mu)
  # unit deviance
```

```

 2 * (y * log(ifelse(y==0, 1, y/mu)) - (y - mu))
res.dev3 <- sqrt(poisson.dev(expected, estimates)) *
  ifelse(expected > estimates, 1, -1)
all.equal(res.dev1, res.dev2, res.dev3)
## [1] TRUE

```

Note that, for ordinary least-squares models, the deviance residual is identical to the conventional residual.

## Deviance residuals in practice

We can obtain the deviance residuals of our model using the `residuals` function:

```

summary(residuals(model.pois))

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## -4.0451 -2.4138 -0.8169 -0.1840  1.4265  8.7946

```

Since the median deviance residual is close to zero, this means that our model is not biased in one direction (i.e. the outcome is neither over- nor underestimated).

## Null and residual deviance

Since we have already introduced the deviance, understanding the null and residual deviance is not a challenge anymore. Let us repeat the definition of the deviance once again:

$$D(y, \hat{\mu}) = 2(\log(p(y | \hat{\theta}_s)) - \log(p(y | \hat{\theta}_0))).$$

The null and residual deviance differ in  $\theta_0$

- Null deviance:  $\theta_0$  refers to the null model (i.e., an intercept-only model)
- Residual deviance:  $\theta_0$  refers to the trained model

How can we interpret these two quantities?

- Null deviance: A low null deviance implies that the data can be modeled well merely using the intercept. If the null deviance is

low, you should consider using few features for modeling the data.

- Residual deviance: A low residual deviance implies that the model you have trained is appropriate. Congratulations!

## Null deviance and residual deviance in practice

Let us investigate the null and residual deviance of our model:

```
paste0(c("Null deviance: ", "Residual deviance: "),
       round(c(model.pois$null.deviance,
               deviance(model.pois)), 2))
## [1] "Null deviance: 1979.57" "Residual deviance:
530.51"
```

These results are somehow reassuring. First, the null deviance is high, which means it makes sense to use more than a single parameter for fitting the model. Second, the residual deviance is relatively low, which indicates that the log likelihood of our model is close to the log likelihood of the saturated model.

However, for a well-fitting model, the residual deviance should be close to the degrees of freedom (74), which is not the case here. For example, this could be a result of overdispersion where the variation is greater than predicted by the model. This can happen for a Poisson model when the actual variance exceeds the assumed mean of  $\mu = \text{Var}(Y)$ .

## Other outputs of the summary function

Here, I deal with the other outputs of the GLM summary function: the dispersion parameter, the AIC, and the statement about Fisher scoring iterations.

### Dispersion parameter

Dispersion (variability/scatter/spread) simply indicates whether a distribution is wide or narrow. The GLM function can use a dispersion parameter to model the variability.

However, for likelihood-based model, the dispersion parameter is always fixed to 1. It is adjusted only for methods that are based on

quasi-likelihood estimation such as when `family = "quasipoisson"` or `family = "quasibinomial"`. These methods are particularly suited for dealing with overdispersion.

### AIC

The Akaike information criterion (AIC) is an information-theoretic measure that describes the quality of a model. It is defined as

$$AIC = 2p - 2\ln(\hat{L})$$

where  $p$

is the number of model parameters and  $\hat{L}$  is the maximum of the likelihood function. A model with a low AIC is characterized by low complexity (minimizes  $p$ ) and a good fit (maximizes  $\hat{L}$ ).

### Fisher scoring iterations

The information about *Fisher scoring iterations* is just verbose output of iterative weighted least squares. A high number of iterations may be a cause for concern indicating that the algorithm is not converging properly.

## The prediction function of GLMs

The GLM `predict` function has some peculiarities that should be noted.

### The type argument

Since models obtained via `lm` do not use a linker function, the predictions from `predict.lm` are always on the scale of the outcome (except if you have transformed the outcome earlier). For `predict.glm` this is not generally true. Here, the `type` parameter determines the scale on which the estimates are returned. The following two settings are important:

- `type = "link"`: the default setting returns the estimates on the scale of the link function. For example, for Poisson regression, the estimates would represent the logarithms of the outcomes. Given

the estimates on the link scale, you can transform them to the estimates on the response scale by taking the inverse link function.

- **type = "response"**: returns estimates on the level of the outcomes. This is the option you need if you want to evaluate predictive performance.

Let us see how the returned estimates differ depending on the type argument:

```
# prediction on link scale (log)
pred.l <- predict(model.pois, newdata = ozone[testset, ])
summary(pred.l)

##      Min. 1st Qu.   Median     Mean 3rd Qu.     Max.
##    2.219   3.118   3.680   3.603   4.136   4.832

# prediction on response scale
pred.r <- predict(model.pois, newdata = ozone[testset, ],
type = "response")
summary(pred.r)

##      Min. 1st Qu.   Median     Mean 3rd Qu.     Max.
##    9.20    22.60   39.63   44.25   62.58  125.46
```

Using the link and inverse link functions, we can transform the estimates into each other:

```
link <- family(model.pois)$linkfun # link function: log
for Poisson
ilink <- family(model.pois)$linkinv # inverse link
function: exp for Poisson
all.equal(ilink(pred.l), pred.r)
## [1] TRUE

all.equal(pred.l, link(pred.r))
## [1] TRUE
```

There is also the **type = "terms"** setting but this one is rarely used an also available in **predict.lm**.

## Obtaining confidence intervals

The predict function of GLMs does not support the output of confidence intervals via `interval = "confidence"` as for `predict.lm`. We can still obtain confidence intervals for predictions by accessing the standard errors of the fit by predicting with `se.fit = TRUE`:

```
predict.confidence <- function(object, newdata, level = 0.95, ...) {
  if (!is(object, "glm")) {
    stop("Model should be a glm")
  }
  if (!is(newdata, "data.frame")) {
    stop("Please input a data frame for newdata")
  }
  if (!is.numeric(level) | level < 0 | level > 1) {
    stop("level should be numeric and between 0 and 1")
  }
  ilink <- family(object)$linkinv
  ci.factor <- qnorm(1 - (1 - level)/2)
  # calculate CIs:
  fit <- predict(object, newdata = newdata,
                 level = level,
                 type = "link", se.fit = TRUE, ...)
  lwr <- ilink(fit$fit - ci.factor * fit$se.fit)
  upr <- ilink(fit$fit + ci.factor * fit$se.fit)
  df <- data.frame("fit" = ilink(fit$fit) ,
                   "lwr" = lwr, "upr" = upr)
  return(df)
}
```

Using this function, we get the following confidence intervals for the Poisson model:

```
conf.df <- predict.confidence(model.pois, ozone[testset,])
head(conf.df, 2)

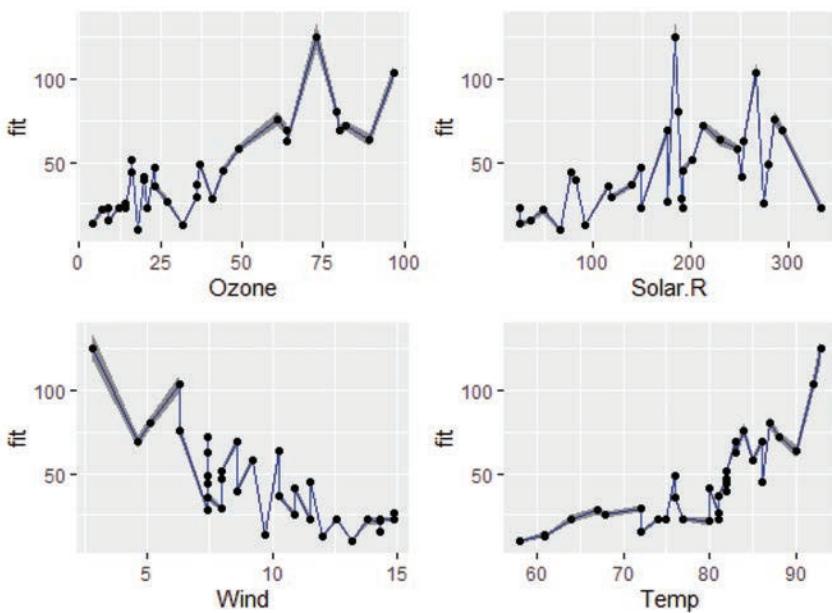
##
##      fit      lwr      upr
## 1 27.83060 25.59152 30.26559
## 2 28.85877 26.91558 30.94225
```

Using the confidence data, we can create a function for plotting the confidence of the estimates in relation to individual features:

```
plot.confidence <- function(df, feature) {  
  library(ggplot2)  
  p <- ggplot (df, aes_string(x = feature,  
                               y = "fit")) +  
    geom_line(colour = "blue") +  
    geom_point() +  
    geom_ribbon(aes(ymin = lwr, ymax = upr),  
                alpha = 0.5)  
  return(p)  
}  
  
plot.confidence.features <- function(data, features) {  
  plots <- list()  
  for (feature in features) {  
    p <- plot.confidence(data, feature)  
    plots[[feature]] <- p  
  }  
  library(gridExtra)  
  #grid.arrange  
(plots[[1]], plots[[2]], plots[[3]])  
  do.call(grid.arrange, plots)  
}
```

Using these functions, we can generate the plots shown in **Figure 8-3**.

```
data <- cbind(ozone[testset,], conf.df)  
plot.confidence.features(data, colnames(ozone))
```



**Figure 8-3.** Poisson fitted plots of Ozone (upper left), Solar.R (upper right), Wind (lower left) and Temp (lower right), with confidence features



## 9. Machine Learning and Prediction

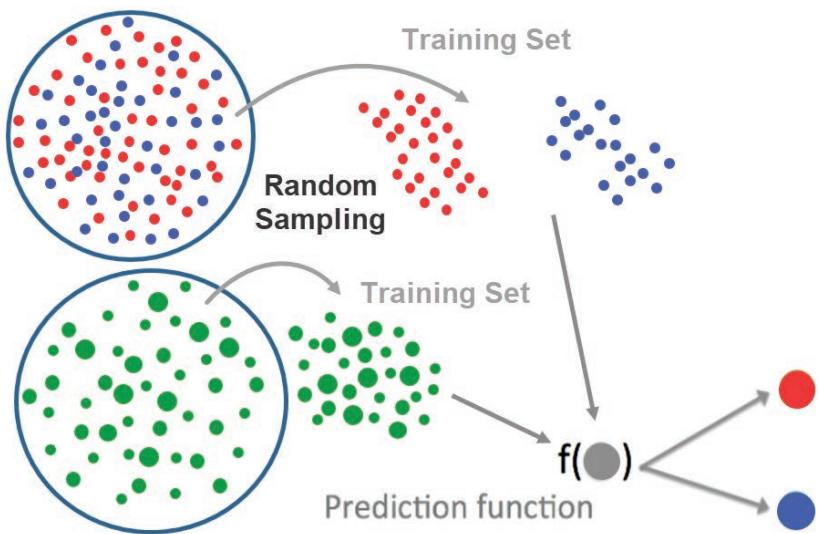
### Prediction

Most major organizations (and some less major) use machine learning in some form and simple to much more complicated forms. Local governments might try to predict pension payments so that they know whether their revenue generation mechanisms have sufficient funds generated to cover pension payments. Google might want to predict whether we're going to click on an ad so that they can show us only the ads that are most likely to get clicks to increase revenue. Amazon and Netflix like that will show us one free movie because they want us to buy the next movie, so that they need to our interests. In 2002, Oakland A's predicted wins using under evaluated players' on-base averages. With severe budget cuts and high player turnover, they won the 2002 American League West title (Lewis, 2003).

Insurance companies employ large groups of actuaries and statisticians to try to predict our risk of many different things, including death, so they can know the right price to set insurance premiums. Some of these prediction tasks are very simple and some are very complicated. Predicting which advertisement we might click on may have a lot of predictors, and might be based a complicated machine learning algorithm. In some cases, it might be a lot simpler in terms of what we're trying to predict. Some are much simpler.

A lot of focus in machine learning keyed on what algorithms are the best algorithms for extracting information and using it to predict. However, it's important to step back and look at the entire prediction problem. Fig illustrates some of the key issues in building a predictor. We start with what we want to predict, which customers will buy additional like insurance. The red and blue dots could represent what we want to predict, different levels of the response variable, like yes and no. and the green dots could represent the predictors or independent variables, with variable values (hence the sizes) that we'll use to make the predictions. Using probability and sampling, we'll select training sets from the data. The training set will consist of

some randomly selected red and blue dots, and an equal number of randomly selected blue dots. Then we split the training set to obtain a test set, usually much smaller, say 20% to 30%. We'll "train" the features of the predictor dots to build prediction function that distribute the response dots. Then, we'll take our smaller set of predictor dots and response dots and "test" that the prediction function will distribute the response dots in a similar manner. Then we'll use some metrics to determine if the train and test results are similar enough to use the model for prediction purposes.



**Figure 9-1.** Depiction of the training a prediction function with random sampling.

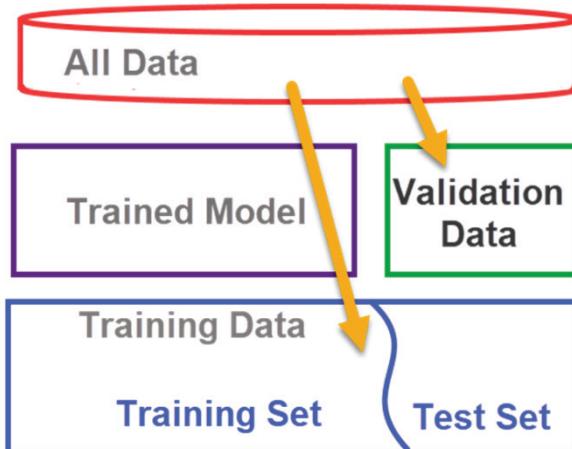
One thing that I think is very important and often underappreciated about building a machine learning algorithm is to look at probability and sampling step of building the training and test sets. The test set is removed from the data set before we begin training the predictor function. Otherwise, the test set may become part of the training set and we lose accuracy. So, before we start training, we have:

- A training response set
- A training predictor set
- A test response set

- A test predictor set

### Cross-Validation

Furthermore, we usually have another set of data taken from a time, say next month's response data, as a validation set. We sometimes refer to this as the set-aside data, and it includes both the response variable only. Then we "harden" the predictor function so that the predictors are set in the predictor function or model and are not fed new predictor data. Then the validation set of responses are run through the model and we evaluate how close the results are to the model we trained and tested (see **Table 9-2**). And for the model to be accurate, all of these subsets must be random samples. This impacts the kinds of errors we encounter.



**Figure 9-2.** Depiction of the cross-validation process

### Types of Errors

Some of the errors we deal with are, in general, positive = identified and negative = rejected errors. We usually adopt the following convention (Strickland, 2020):

- True positive = correctly identified
- False positive = incorrectly identified
- True negative = correctly rejected

- False negative = incorrectly rejected

Using a COVID testing example:

- True positive = People with COVID correctly diagnosed as sick
- False positive= Healthy people incorrectly diagnosed with COVID
- True negative = Healthy people correctly diagnosed as healthy
- False negative= People with COVID incorrectly diagnosed as healthy.

Using the error construct, we develop metrics for determining model prediction quality. The so-called confusion matrix (**Figure 9-3**) is often used to describe the errors and model quality.

		COVID	
		+	-
TEST	+	TP	FP
	-	FN	TN

**Figure 9-3.** Confusion matrix for COVID testing

Sensitivity	→	$\Pr(\text{positive test}   \text{disease})$
Specificity	→	$\Pr(\text{negative test}   \text{no disease})$
Positive Prediction Value	→	$\Pr(\text{disease}   \text{positive test})$
Negative Prediction Value	→	$\Pr(\text{no disease}   \text{negative test})$
Accuracy	→	$\Pr(\text{correct outcome})$

Here, what we desire is a model with greater sensitivity and greater specificity, resulting in greater accuracy. So, our common discrete error measures are specificity, sensitivity, and accuracy.

Continuous error measures include:

$$\text{Mean Squared Error (MSE)} = \frac{1}{n} \sum_{i=1}^n (Pred_i - Truth_i)^2$$

$$\text{Root Mean Squared Error (RMSE)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (Pred_i - Truth_i)^2}$$

$$\text{Mean Absolute Deviation (MAD)} = \frac{1}{n} \sum_{i=1}^n |x_i - m(X)|$$

And  $m(X)$  is the measure of central tendency.

There are two additional error concepts:

- **In-Sample Error:** The error rate you get on the same data set you used to build your predictor. Sometimes called re-substitution error.
- **Out-of-Sample Error:** The error rate you get on a new data set, the validation set. Sometimes called generalization error.

When the in-sample error is less than the out-of-sample error, we usually have an over-fit model. Since our primary concern is out-of-sample error, we have to build our models to address the out-of-sample error, rather than building a model that fits the training/test data extremely well.

The data has two parts: signal and noise. The goal of a predictor is to find signal. We can always design a perfect in-sample predictor and capture both signal and noise. When we do that, the predictor will not perform as well on new samples.

## Example 9-1 – Sample Size

In this example we will discuss how to determine sample size and power using various statistical tests for a variety of situations. The source code is located at: [https://rpubs.com/stricje1/sample\\_size](https://rpubs.com/stricje1/sample_size).

## *t*-tests

For t-tests, use the following functions:

```
pwr.t.test(n = , d = , sig.level = , power = , type =  
c("two.sample", "one.sample", "paired"))
```

where  $n$  is the sample size,  $d$  is the effect size, and type indicates a two-sample t-test, one-sample t-test or paired t-test. If you have unequal sample sizes, use

```
pwr.t2n.test(n1 = , n2= , d = , sig.level =, power = )
```

where  $n_1$  and  $n_2$  are the sample sizes.

For t-tests, the effect size is assessed as

$$d = \frac{|\mu_1 - \mu_2|}{\sigma} \quad \text{where} \quad \begin{aligned} \mu_1 &= \text{mean of group 1} \\ \mu_2 &= \text{mean of group 2} \\ \sigma^2 &= \text{common error variance} \end{aligned}$$

Cohen suggests that  $d$  values of 0.2, 0.5, and 0.8 represent small, medium, and large effect sizes respectively.

You can specify alternative="two.sided", "less", or "greater" to indicate a two-tailed, or one-tailed test. A two tailed test is the default.

```
power = pwr.t2n.test(n1 = 30 , n2 = 35 , d = 0.5,  
sig.level = 0.5)  
sig_lvl = pwr.t2n.test(n1 = 30 , n2 = 35 , d = 0.8,  
power = 0.85, sig.level=NULL)  
cat("power =", power$power)  
## power = 0.912156  
  
cat("significance level =", sig_lvl$sig.level)  
## significance level = 0.03390876  
  
samp_size = pwr.t.test(d = 0.5 , sig.level = 0.05 ,  
power = 0.85)
```

```
> cat("sample size =", samp_size$n)
## sample size = 72.80046
```

### Tests of Proportions

When comparing two proportions use

```
pwr.2p.test(h = , n = , sig.level = , power = )
```

where **h** is the effect size and **n** is the common sample size in each group.

$$h = 2 \arcsin \sqrt{(p_1)} - 2 \arcsin \sqrt{(p_2)}$$

Cohen suggests that h values of 0.2, 0.5, and 0.8 represent small, medium, and large effect sizes respectively.

For unequal n's use

```
pwr.2p2n.test(h = , n1 = , n2 = , sig.level = , power = )
```

To test a single proportion use

```
pwr.p.test(h = , n = , sig.level = power = )
```

For both two sample and one sample proportion tests, you can specify **alternative="two.sided"**, **"less"**, or **"greater"** to indicate a two-tailed, or one-tailed test. A two tailed test is the default.

```
library(pwr)
h = pwr.2p2n.test(n2 = 20 , n1 = 80,
sig.level=0.05,power=0.8 )
power = pwr.2p2n.test(h = 0.8 , n1 = 80, n2 = 20,
sig.level=0.05)
cat("n2 =",n2$n2)
cat("h =",h$h)
cat("power =", power$power)

## n2 = 14.48425> cat("h =",h$h)
```

```
## h = 0.700393> cat("power =", power$power)
## power = 0.8925191
```

## Correlations

For **correlation** coefficients use

```
pwr.r.test(n = , r = , sig.level = , power = )
```

where **n** is the sample size and **r** is the correlation. We use the population correlation coefficient as the effect size measure. Cohen suggests that **r** values of 0.1, 0.3, and 0.5 represent small, medium, and large effect sizes respectively.

```
samp_size = pwr.r.test(n = , r = 0.3, sig.level = 0.05,
power = 0.9)
cat("sample size =", samp_size$n)
## sample size = 111.8068
```

## Creating Power or Sample Size Plots

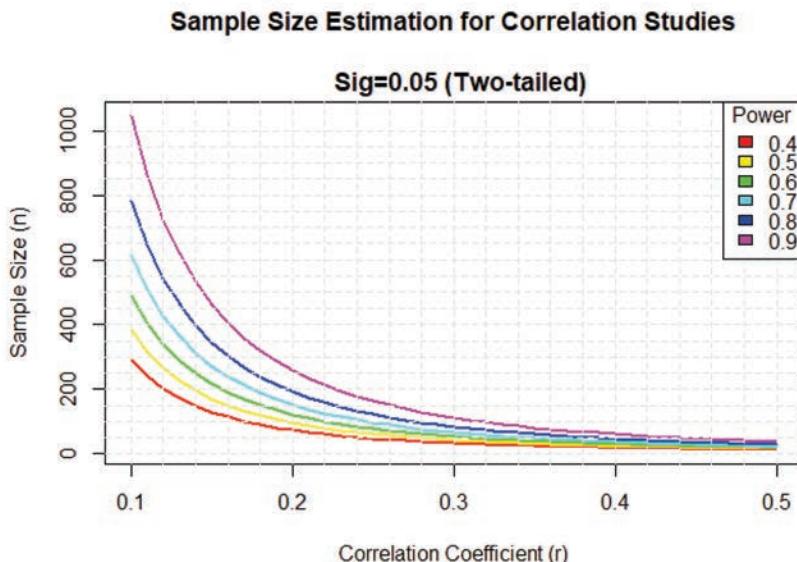
The functions in the **pwr** package can be used to generate power and sample size graphs, as shown in **Figure 9-4**.

```
# range of correlations
r <- seq(0.1,0.5,0.01)
nr <- length(r)
# power values
p <- seq(0.4,0.9,0.1)
np <- length(p)
# obtain sample sizes
samsize <- array(numeric(nr*np), dim=c(nr,np))
for (i in 1:np){
  for (j in 1:nr){
    result <- pwr.r.test(n = NULL, r = r[j],
                          sig.level = 0.05, power = p[i],
                          alternative = "two.sided")
    samsize[j,i] <- ceiling(result$n)
  }
}
# set up graph
xrange <- range(r)
```

```

yrange <- round(range(samsize))
colors <- rainbow(length(p))
plot(xrange, yrange, type="n",
      xlab= "Correlation Coefficient (r)",
      ylab= "Sample Size (n) ")
# add power curves
for (i in 1:np){
  lines(r, samsize[,i], type="l", lwd=2, col=colors[i])
}
# add annotation (grid lines, title, legend)
abline(v=0, h=seq(0,yrange[2],50), lty=2, col= "grey89")
abline(h=0, v=seq(xrange[1],xrange[2],0.02), lty=2,
       col= "grey89")
title("Sample Size Estimation for Correlation Studies\n
      Sig=0.05 (Two-tailed)")
legend("topright", title="Power", as.character(p),
       fill=colors)

```



**Figure 9-4.** Sample size estimates for correlation studies

### Prediction Study Design

1. Split the data sets
  - Train Set: random sample 60% to 80% of the training data

- Test Set: the rest of the training data or 20% to 40% of the data set
  - Validation Set: random sample from a different sample
2. Pick features from the training set
  3. Use the training set to select the prediction function, or fit the model
  4. Use the test set to “check” the prediction function
  5. If no validation set, use the test set to cross-validate
  6. If there is a validation set, apply to the hardened model

## Example 9-2. Exercise Gadgets

### *Data Preprocessing*

One thing that people regularly do is quantify how *much* of a particular activity they do, but they rarely quantify *how well they do it*. In this example, our goal will be to use data from accelerometers on the belt, forearm, arm, and dumbbell of six participants to predict the manner in which they did the exercise. The source data is located at: [https://github.com/stricje1/Machine\\_Learning\\_Project](https://github.com/stricje1/Machine_Learning_Project).

The training and test data for this project are available here:

```
trainUrl <-  
"http://d396qusza40orc.cloudfront.net/predmachlearn/pml-  
training.csv"  
testUrl <-  
"http://d396qusza40orc.cloudfront.net/predmachlearn/pml-  
testing.csv"
```

### *Data Preparation*

Now we read the data into memory and remove missing data

```
training <- read.csv(url(trainUrl),  
na.strings=c("NA","#DIV/0!",""))  
testing <- read.csv(url(testUrl),  
na.strings=c("NA","#DIV/0!",""))
```

Here, we construct the training and testing dataframes to prepare the data for cross-validation.

```
inTrain <- createDataPartition(y=training$classe, p=0.6,
list=FALSE)
myTraining <- training[inTrain, ]; myTesting <- training[-inTrain, ]
dim(myTraining); dim(myTesting)
## [1] 11776   160
## [1] 7846   160
```

### **Cleaning the Data**

The following transformations were used to clean the data:

**Transformation 1:** Cleaning NearZeroVariance Variables Run this code to view possible NZV Variables:

```
myDataNZV <- nearZeroVar(myTraining, saveMetrics=TRUE)
myTraining <- myTraining[!myNZVvars]
```

**Transformation 2:** Killing first column of Dataset - ID Removing first ID variable so that it does not interfere with ML Algorithms:

```
myTraining <- myTraining[c(-1)]
```

**Transformation 3:** Cleaning Variables with too many NAs. For Variables that have more than a 60% threshold of NAs I'm going to leave them out. We also establish the training set to have 60% of the data, leaving 40% for the testing set.

```
trainingV3 <- myTraining #creating another subset to
iterate in loop
for(i in 1:length(myTraining)) { #for every column in the
training dataset
  if( sum( is.na( myTraining[, i] ) ) /nrow(myTraining) >=
.6 ) { #if n?? NAs > 60% of total observations
    for(j in 1:length(trainingV3)) {
      if( length( grep(names(myTraining[i]),
names(trainingV3)[j]) ) ==1 ) { #if the columns are the
same:
        trainingV3 <- trainingV3[ , -j] #Remove that
column
      }
    }
  }
}
```

```
    }  
}
```

## *Resetting the Training Set*

Next, we put the data into its original training set and remove (rm) the last iteration or it (trainingV3).

```
myTraining <- trainingV3  
rm(trainingV3)
```

Now we perform the exact same 3 transformations for myTesting and testing data sets.

```
clean1 <- colnames(myTraining)  
clean2 <- colnames(myTraining[, -58]) #already with classe  
column removed  
myTesting <- myTesting[clean1]  
testing <- testing[clean2]
```

## *Data Coercion*

In order to ensure proper functioning of the classifier algorithms with the test data set, we need to coerce the data into the same type. We also make sure Coercion works by row-binding row two of myTraining and then removing the row.

```
for (i in 1:length(testing) ) {  
  for(j in 1:length(myTraining)) {  
    if( length( grep(names(myTraining[i]),  
names(testing)[j]) ) ==1) {  
      class(testing[j]) <- class(myTraining[i])  
    }  
  }  
}  
testing <- rbind(myTraining[2, -58] , testing)  
testing <- testing[-1, ]
```

## *Finding the Best ML Model*

### *Ensemble Learning*

Ensemble learning is a machine learning concept in which idea is to train multiple models (learners) to solve the same problem. The

main advantages of Ensemble learning methods are: (1) reduced variance, which helps overcome overfitting helps the model to be independent of training data; and (2) reduced bias, which overcome underfitting problem improves reliable classification over a single classifier.

### ***Random Forest***

The random forest algorithm is comprised of multiple trees, either classification or regression trees. This makes it an ensemble model. We'll talk more about this in Chapter 10.

### ***Stochastic Gradient Descent***

Stochastic Gradient Descent (SGD) is a simple, very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) SVMs and Logistic Regression. SGDs are efficient easy to complement, as there are many opportunities for tuning.

### ***Bagging (Bootstrap Aggregating)***

Bagging generates  $n$  new training data sets, each of which picks a sample of observations with replacement (bootstrap sample) from original data set. By sampling with replacement, some observations may be repeated in each new training data set. Then models are fitted using the above  $n$  bootstrap samples and combined them by averaging the output (for regression) or voting (for classification).

### ***Boosted Logistic Regression***

In logistic regression, boosting works by sequentially applying the algorithm to reweighted versions of the training data and then taking a weighted majority vote of the sequence of classifiers it produces

### ***Bagged Classification and Regression Trees (CART)***

Bagging algorithms with recursive partitioning as the estimator result in a prediction tool that is no longer a tree. Thus, these methods have superior predictive accuracy when compared to a single tree.

Recursive partitioning that incorporates cross-validation still results in a tree. Classification trees involve a categorical response variable and regression trees a continuous response variable. The approaches are similar enough to be referred to together as CART (Classification and Regression Trees). Predictors can be any combination of categorical or continuous variables. With bagging applied, multiple trees are evaluated.

### *Train ML Algorithms*

```
if (!require("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("logicFS") library(logicFS)
```

### *Prepare Training Scheme*

`trainControl` manages the computational nuances of the `train` function(). The method, `repeatedcv`, causes the parameters to be repeated, and 5-fold CV mean it divides the training set randomly into 5 parts and then using each of 5 parts as testing dataset for the model trained on other 4. With 3 repeats of 5-fold CV, the algorithms perform the average of 3 error terms obtained by performing 5-fold CV 3 times.

```
control <- trainControl(method="repeatedcv", number=5,
repeats=3)
# train the Logit Boost model
set.seed(7)
modelLog <- train(classe~, data=myTraining,
method="LogitBoost", trControl=control)
# train the RF model
set.seed(7)
modelRf <- train(classe~, data=myTraining, method="rf",
trControl=control)
# train the GBM model
set.seed(7)
modelGbm <- train(classe~, data=myTraining, method="gbm",
trControl=control, verbose=FALSE)
# train the CART model
set.seed(7)
modelTb <- train(classe~, data=myTraining,
method="treebag", trControl=control)
```

```

# collect resamples
results <- resamples(list(Log=modelLog, RF=modelRf,
GBM=modelGbm, Tb=modelTb))
# summarize the distributions
summary(results)
##
## Call:
## summary.resamples(object = results)
##
## Models: Log, RF, GBM, Tb
## Number of resamples: 15
##
## Accuracy
##             Min.   1st Qu.   Median   Mean   3rd Qu.
Max. NA's
## Log 0.9549795 0.9584772 0.9677858 0.9656414 0.9699031
0.9771376    0
## RF  0.9970276 0.9976651 0.9987261 0.9983866 0.9991506
0.9995752    0
## GBM 0.9944798 0.9966030 0.9970263 0.9969712 0.9974522
0.9987261    0
## Tb  0.9953291 0.9970282 0.9978769 0.9977638 0.9985144
0.9995756    0
##
## Kappa
##             Min.   1st Qu.   Median   Mean   3rd Qu.
Max. NA's
## Log 0.9427885 0.9472616 0.9590602 0.9563574 0.9617693
0.9709190    0
## RF  0.9962401 0.9970467 0.9983887 0.9979593 0.9989256
0.9994627    0
## GBM 0.9930176 0.9957030 0.9962389 0.9961691 0.9967775
0.9983889    0
## Tb  0.9940913 0.9962410 0.9973145 0.9971715 0.9981209
0.9994631    0

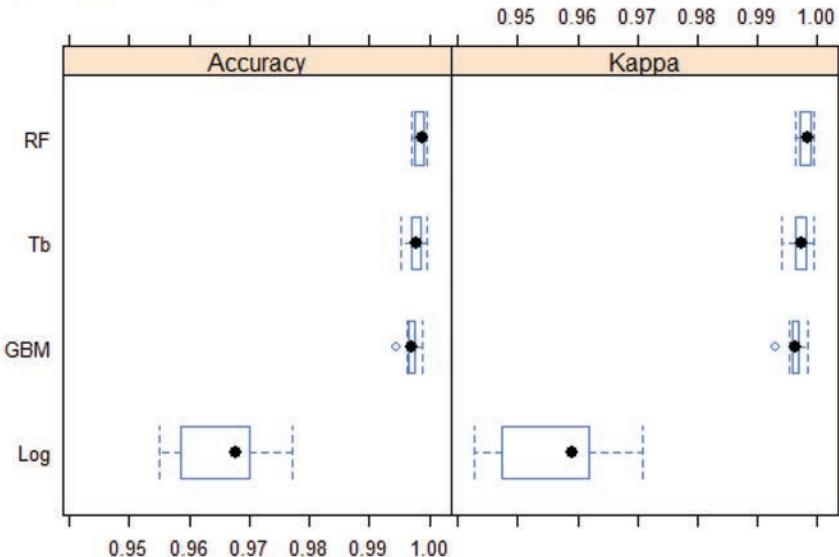
```

The Accuracy metric shows that all the models fit the data well, and practically, these RF, GB, and TB perform equally. However, we must pick a “best” model, so we will take the RF, which performs slight better than the TB (by 0.06). This is depicted in **Figure 9-5**. It is interesting to see that both RF and TB are tree-ensembles.

## *boxplots of results*

The boxplot provides visual support of the above results.

```
bwplot(results)
```

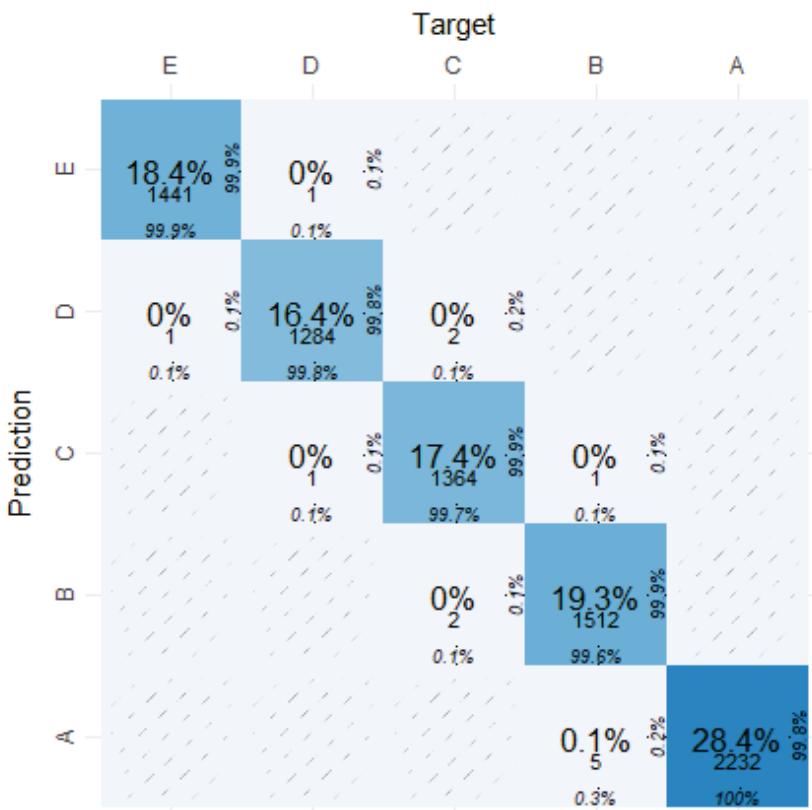


**Figure 9-5.** Boxplots of four competing word prediction models for two metrics, accuracy and kappa values

## *Cross-Validation*

Now, we use the testing data for cross-validation. This includes validating the model with the testing dataset. We also include interpret the results and explain them using a graphical confusion matrix (see **Figure 9-6**).

```
predictionOnTesting <- predict(modelRf, newdata=testing)
predictionOnTesting
## [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
Rf_pred <- predict(modelRf, myTesting)
conf_mat <- confusion_matrix(targets = myTesting$classe,
                               predictions = Rf_pred)
plot_confusion_matrix (conf_mat$`Confusion Matrix`[[1]])
```



**Figure 9-6.** Heat map of the confusion matrix for random forest word prediction model

In the middle of each tile, we have the normalized count (overall percentage) and, beneath it, the count. When all of the tile overall percentages are summed, they sum is 100%. For instance,  $18.4\% + 16.4\% + 17.4\% + 19.3\% + 28.4\% + 0.1\% = 100\%$ , allowing for round-off.

At the bottom of each tile, we have the column percentage. Of all the observations where Target is E, 99.9% of them were predicted to be E and 0.1% to be D. At the right side of each tile, we have the row percentage. Of all the observations where Prediction is A, 99.8% of them were actually A, while 23% were B. Note that the color intensity is based on the counts.

## Example 9-3. Using R for Crime Analysis

*“Sherlock Holmes was a man, however, who, when he had an unsolved problem upon his mind, would go for days, and even for a week, without rest, turning it over, rearranging his facts, looking at it from every point of view until he had either fathomed it or convinced himself that his data were insufficient.”*

—Dr. John Watson, *The Man with the Twisted Lip*

### ***Introduction***

Sherlock's data may have been insufficient, but at present we cannot make the same complaint, at least not entirely. If we have any burden whatsoever, then it is possessing too much data or having the data, but in an “unusable” format. In this study, will had three objectives: (1) use crime data to visualize the spatial and temporal trends of criminal activity, (2) analyze the data for factors that may be used to gain additional insights that the raw data does not provide, and (3) use specialized plotting functions to visual insightful aspects of the resulting crime information. Taken together, we generated reusable code to cluster and describe data that the police departments and/or city halls have already collected and turned over to the skillful hands of their analyst. The source code is located at my crime analysis repository: [https://github.com/stricje1/crime\\_analysis](https://github.com/stricje1/crime_analysis).

### ***The Crime Data***

You can find the R markdown for this example at <https://stricje1.github.io/ddp/>. The raw data we use for this study was downloaded from the Atlanta Police Department (PD) website ([www.atlantapd.org](http://www.atlantapd.org)). The data was contained in ten ZIP files, one for each year from 2009 to 2018. We joined the comma separated sets (CSV) sets together in Excel and generated a time-of-day variable. Then we compressed the CSV fil and uploaded it to GitHub as `atlanta_crime_10yr.zip`. The resulting set is `atlanta_crime_10yr.xls`, with 619,663 records and thirteen variables name as follows (Strickland, Data Science Applications using R. Lulu.com, 2019):

1. **IncidentNum** (N) Incident number
2. **Category** (C) Crime category, i.e., larceny/theft
3. **Descript** (C)
4. **DayOfWeek** (C)
5. **Date** (D) Date: DD/MM/YYYY
6. **Time** (T) Time: 24-hour system
7. **PdDistrict** (C) Police district where incident occurred
8. **Resolution** (C) Resolution of the crime
9. **X** (N) Longitude
10. **Y** (N) Latitude
11. **Location** (C) Lat/long
12. **PdId** (N) Police Department ID

(N = Numeric, T = Time, D = Date, C = Class)

The center of the Atlanta metropolitan area is (33.753746, -84.38633). Table 1 shows a partial view of the *enhanced* Atlanta crime data.

### *Installing Libraries & Loading Data*

```
if(!require(readr)) install.packages("readr")
if(!require(dplyr)) install.packages("dplyr")
if(!require(DT)) install.packages("DT")
if(!require(data.table)) install.packages("data.table")
if(!require(ggrepel)) install.packages("ggrepel")
if(!require(leaflet)) install.packages("leaflet")
```

The crime data was simulated using distributions derived from an analysis of major crimes in Atlanta, Georgia USA, San Francisco, California USA, and Boston, Massachusetts USA. For instance, Larceny/theft was approximately uniformly distributed  $\text{Uniform} \sim [120000, 160000]$  with mean 140000 and standard deviation of approximately 11547.

First, we check to see if we already have the data. If we don't, we download the ZIP file from GitHub and unzip it. Then we load the data into RStudio using `read_csv()`.

```
filename = "atlanta_crime_10yr.zip"
```

```

if (!file.exists(filename)){
urlzip <-
"https://github.com/stricje1/Data/blob/master/atlanta_crime_10yr.zip"
  download.file(urlzip, destfile =
"./atlanta_crime_10yr.zip" )
}
unzip("./atlanta_crime_10yr.zip", exdir = ".")
library(readr)
df <- read.csv ("atlanta_crime_10yr.csv")

```

## Display Data

Now, we display the data using `DT` and `data.table()`. (NOTE: After the publication of my book Predictive Crime Analysis using R, ISBN 978-0-359-43159-5, the function `datatable()` has been replaced by `data.table()`.) The table below shows a partial view of the Atlanta crime data

```

## NOT EXECUTED
df_sub <- df[1:100,] # display the first 100 rows
df_sub$Time <- as.character(df_sub$Time)
datatable(df_sub, options = list(pageLength = 5, scrollX='400px'))

```

## Preprocess the Data

The All-Caps text is difficult to read. Let's force the text in the appropriate columns into proper case. The result is shown in Table 10-1

```

proper_case <- function(x) {
  return (gsub("\\b([A-Z])([A-Z]+)", "\\U\\1\\L\\2" , x, perl = TRUE))
}
library(dplyr)
df <- df %>% mutate(Category = proper_case(Category),
                      Descript = proper_case(Descript),
                      PdDistrict = proper_case(PdDistrict),
                      Resolution = proper_case(Resolution),
                      Time = as.character(Time))

```

```
df_sub <- df[1:100,] # display the first 100 rows
datatable(df_sub, options = list(pageLength = 5,
scroll = '400px'))
```

**Table 9-1.** Atlanta crime data from 2009-2018

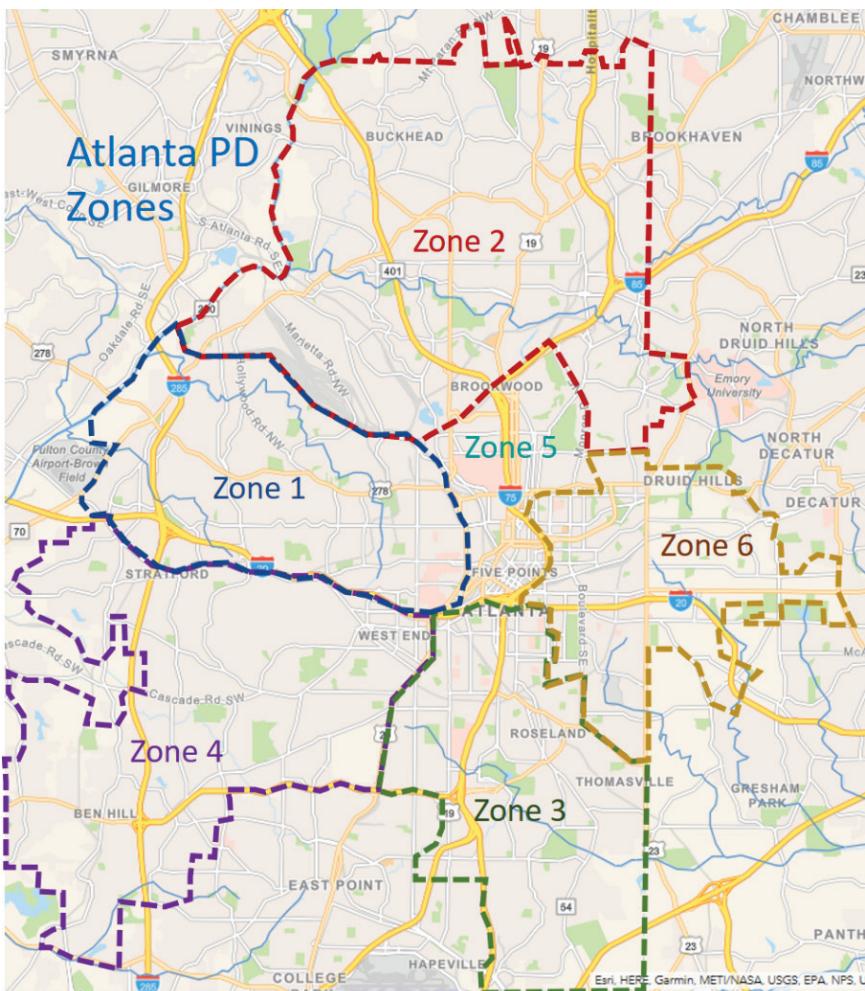
IncidentNum	Category	Date	Time	Resolution	Location
93120547	Homicide	4/10/2013	2100	Arrest	(33.75479,-84.48492)
93130369	Homicide	6/18/2014	1820	None	(33.82028,-84.45188)
93130482	Homicide	7/4/2014	0145	None	(33.71076,-84.44380)
93130637	Homicide	11/22/2014	0200	None	(33.75301,-84.48804)
93130640	Homicide	4/21/2016	1210	Arrest	(33.74898,-84.49168)
93140714	Homicide	11/10/2016	2035	Arrest	(33.69625,-94.53567)

### Explore the Data

In this section, we use map-markers to call out points on the map where crimes are concentrated. Marker locations are expressed in latitude/longitude coordinates and can either appear as icons or as circles. There are many markers on our map, so we cluster them together. To orient you toward our geographic area, **Figure 10-1** shows Atlanta with the general border defined by the Perimeter (I-285). Zones are generally divided by (1) I-75/85 running north and south to the split just north of the downtown area (I did not collect the data for the Airport zone, defined by the I-75/85 split south of the downtown area); and (2) I-20 divided Atlanta into north and south. The ArcGIS Online map shows an Esri World Navigation Map overlay. We added the layer of for PD zones. The seven PD zones are:

- Zone 1, encompassing northwest Atlanta
- Zone 2, encompassing north Atlanta
- Zone 3, encompassing southeast Atlanta
- Zone 4, encompassing southwest Atlanta
- Zone 5, encompassing downtown Atlanta

- Zone 6, encompassing east Atlanta
- Homicide, multi-jurisdictional



**Figure 9-7.** City of Atlanta with Police Department zones

In this section, we use the `leaflet` function. It creates a Leaflet map widget using `htmlwidgets`. The widget can be rendered on HTML pages generated from R Markdown. In addition to matrices and data frames, `leaflet` supports spatial objects from the `sp` package and spatial data frames from the `sf` package. We create a Leaflet map with these basic steps: First, create a map widget by calling

`leaflet()`. Next, we add layers (i.e., features) to the map by using layer functions (e.g., `addTiles`, `addMarkers`, `addPolygons`) to modify the map widget. Then you keep adding layers or stop when satisfied with the result. We will add a tile layer from a known map provider, using the leaflet function `addProviderTiles`. A list of providers can be found at <http://leaflet-extras.github.io/leaflet-providers/preview/>. We will also add graphics elements and layers to the map widget with `addCondrtoll` (`addTiles`). We use markers to call out points on the map. Marker locations are expressed in latitude/longitude coordinates, and can either appear as icons or as circles. When there are many markers on a map as in our case with crimes, we can cluster them together.

```
library(leaflet)
```

The following code defines the popups used on the Leaflet map we will create. The popups will appear on the interactive Leaflet map. When you click on a popup, it will contain the information defined below:

```
data <- df[1:10000,] # display the first 10,000 rows
data$popup <- paste("<b>Incident #: </b>", data$IncidentNum,
                    "<br>", "<b>Category: </b>", data$Category,
                    "<br>", "<b>Description: </b>", data$Description,
                    "<br>", "<b>Day of week: </b>", data$DayOfWeek,
                    "<br>", "<b>Date: </b>", data$Date,
                    "<br>", "<b>Time: </b>", data$Time,
                    "<br>", "<b>PD district: </b>", data$PdDistrict,
                    "<br>", "<b>Resolution: </b>", data$Resolution,
                    "<br>", "<b>Address: </b>", data$Address,
                    "<br>", "<b>Longitude: </b>", data$X,
                    "<br>", "<b>Latitude: </b>", data$Y)
```

Now, we define crime incident locations on the map using `leaflet`, which we use for our popups:

```
library(leaflet)
leaflet(data, width = "100%") %>% addTiles() %>%
  addTiles(group = "OSM (default)") %>%
  addProviderTiles(provider = "Esri.WorldStreetMap",
                   group = "World StreetMap") %>%
  addProviderTiles(provider = "Esri.WorldImagery",
```

```

group = "World Imagery") %>%
addMarkers(lng = ~X, lat = ~Y, popup = data$popup,
           clusterOptions = markerClusterOptions()) %>%
addLayersControl(
  baseGroups = c("OSM (default)", "World StreetMap",
                "World Imagery"),
  options = layersControlOptions(collapsed = FALSE)
)

```

In this manner, we can click icons on the map to show incident details, as shown in **Figure 10-2**. We need to set up some generate some parameters that we concatenate or “paste” together to form these incident descriptions. For example, the concatenated strings `pdata$popup`, provides the content of the second incident as shown here:

```
data$popup[1]
```

```
## [1] "<b>Incident #: </b> 150098210 <br> <b>Category: </b>
  Robbery <br> <b>Description: </b> Robbery, Bodily Force
<br> <b>Day of week: </b> Sunday <br> <b>Date: </b> 2/1/20
15 <br> <b>Time: </b> 15:45:00 <br> <b>PD district: </b> Z
one4 <br> <b>Resolution: </b> None <br> <b>Address: </b> N
one <br> <b>Longitude: </b> 80.26669397 <br> <b>Latitude:
</b> 13.09199072"
```

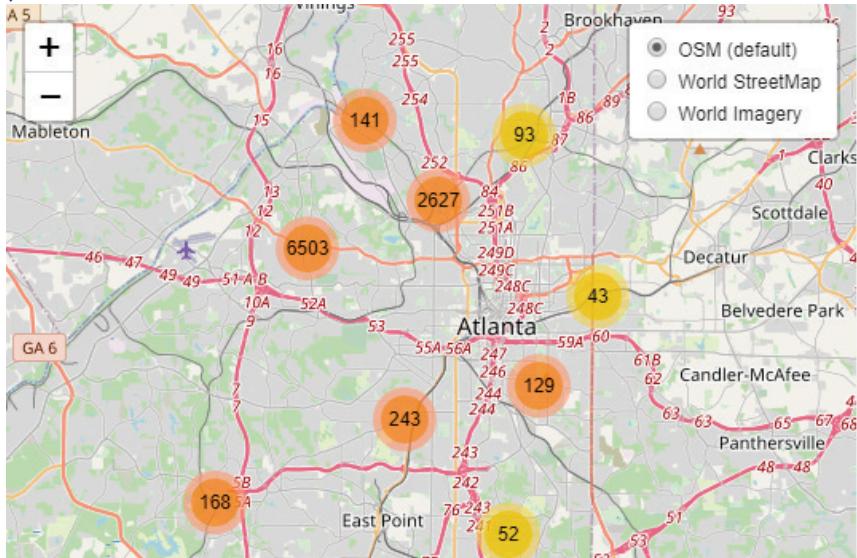
You may notice the “`%>%`” or forward-pipe operator in the `leaflet` arguments. The operators pipe their left-hand side values forward into expressions that appear on the right-hand side, rather than from the inside and out. For example:

```
leaflet(data, width = "100%") %>% addTiles() %>%
  addTiles(group = "OSM (default)") %>%
  addProviderTiles(provider = "Esri.WorldStreetMap",
                    group = "World StreetMap") %>%
  addProviderTiles(provider = "Esri.WorldImagery",
                    group = "World Imagery") %>%
  addProviderTiles(provider = "NASAGIBS.ViirsEarthAtNight2
012", group = "Nighttime Imagery") %>%
  addMarkers(lng = ~X, lat = ~Y, popup = data$popup,
             clusterOptions = markerClusterOptions()) %>%
  addLayersControl()
```

```

baseGroups = c("OSM (default)", "World StreetMap", "World Imagery"),
  options = layersControlOptions(collapsed = FALSE)
)

```



**Figure 9-8.** Crime cluster in Atlanta, 2009-2018

### Crime Over Time

In this section, we will manipulate the data using the `dplyr::mutate` function. `mutate` adds new variables while preserving existing variables. Below, we used “shades of blue” in the code for our plot, with a dark blue line that smooths the data.

```

library(dplyr)
df_crime_daily <- df %>%
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%
  group_by(Date)  %>%
  summarize(count = n()) %>%
  arrange(Date)

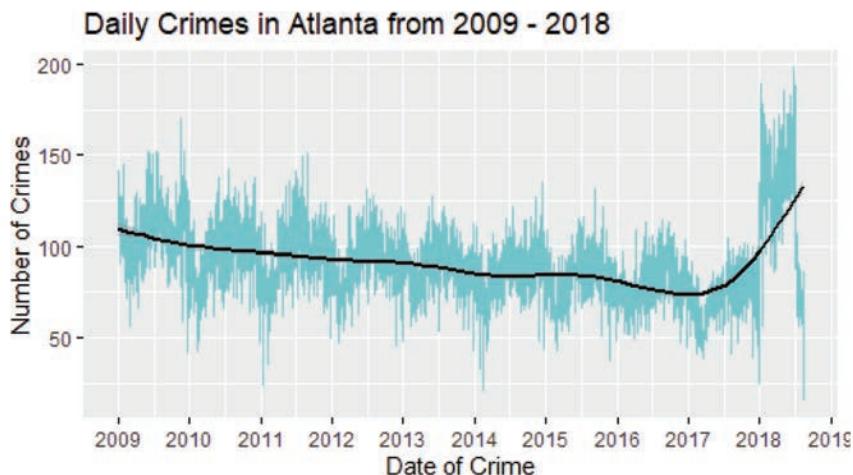
```

### Crimes Series Plot

A crimes series is a time series where the events are crimes. These have the usual components of a time series like seasonality, trend, and noise, as seen in **Figure 10-3**. However, the seasonality may

follow a pattern from day to night, where crimes may increase at night and fall off during the day. Another scenario might involve increased crime rate during certain events, like parades, rodeos, fairs, and so on. The 1996 Summer Olympics (not in the dataset) brought an increase in crime to Atlanta, including the Centennial Olympic Park bombing on July 27, attributed to domestic terrorism.

```
library (ggplot2)
library(scales)
plot <- ggplot(df_crime_daily, aes(x = Date, y = count)) +
  geom_line(color = "#F2CA27", size = 0.1) +
  geom_smooth(color = "#1A1A1A") +
  scale_x_date(breaks = date_breaks("1 year"), labels =
date_format("%Y")) +
  labs(x = "Date of Crime", y = "Number of Crimes",
       title = "Daily Crimes in Atlanta from 2009 - 2018")
plot
```



**Figure 9-9.** Daily crime-series (time series) for Atlanta, 2009-2018

The trend shown by Figure 4-3 shows a slight decrease in reported crimes up to the sharp rise beginning in 2017. Late summer of 2018 shows a possible decrease. The crime series also shows consistent seasonality, at least up to the sharp increase where it is difficult to observe.

## *Aggregated Data*

No, we can aggregate the data and create a table that summarizes the data by incident category as in **Table 10-2**. (In RStudio, this table shows in the Viewer pane.) We used the descending order of “decreasing” for sorting the incident category. `DT::datatable` or the `data.table()` function generates the HTML table widget.

```
df_category <- sort(table(df$Category), decreasing = TRUE)
df_category <- data.frame(df_category[df_category > 1000])
colnames(df_category) <- c("Category", "Frequency")
df_category$Percentage <- df_category$Frequency /
    sum(df_category$Frequency)
data.table(df_category, options = list(scrollX='400px'))
```

**Table 9-2.** Atlanta crime by category and frequency, 2009-2018

	Crime Category	Frequency	Percentage
1	Larceny/Theft	172657	0.54924734056
2	Burglary-Residence	47483	0.15105041482
3	Auto Theft	44247	0.14075622232
4	Agg-Assault	23244	0.07394258665
5	Robbery-Pedestrian	16488	0.05245075584
6	Burglary-Nonres	10233	0.03255267979

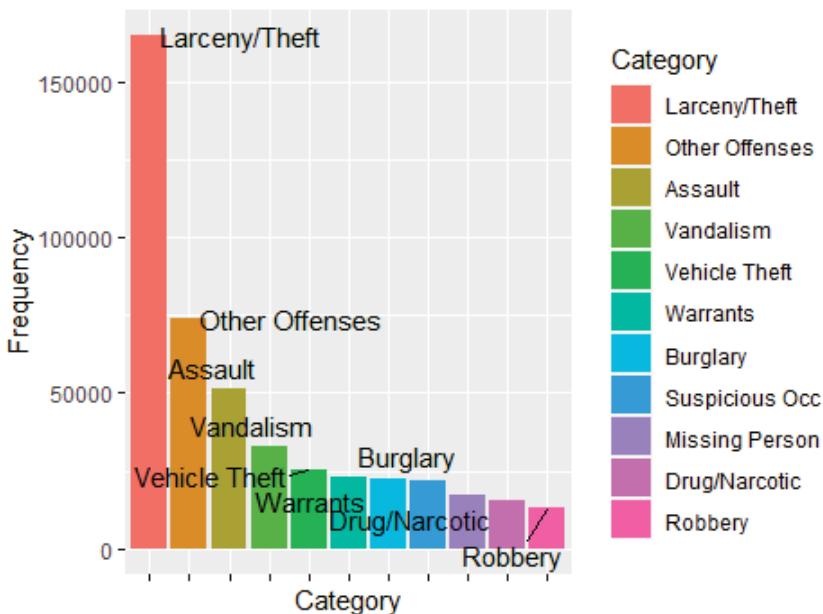
The table shows the frequencies (counts) of the reported crimes with 10,000 or more occurrences, indicating that the most frequent crime is Larceny/Theft (non-vehicular and from the vehicle). With all the hype regarding the dangers of Atlanta, there were only 886 murders in the 10-year period, or about 88 per year, and only 121 reported rapes for about 12 per year or 1 per month.

## *Creating a Bar Chart*

Now that we can aggregate the data, we will show the data with a bar graph, depicted in **Figure 10-4**. The bar graph (or histogram) shows the frequencies of the crimes recorded in Table 4-3. It makes it easy

to see the vast difference between Larceny/Theft and the other reported crimes.

```
library(ggplot2)
library(ggrepel)
bp <- ggplot(df_category, aes(x=Category, y=Frequency,
                               fill=Category)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_blank()) +
  geom_text_repel(data=df_category, aes(label=Category))
bp
```

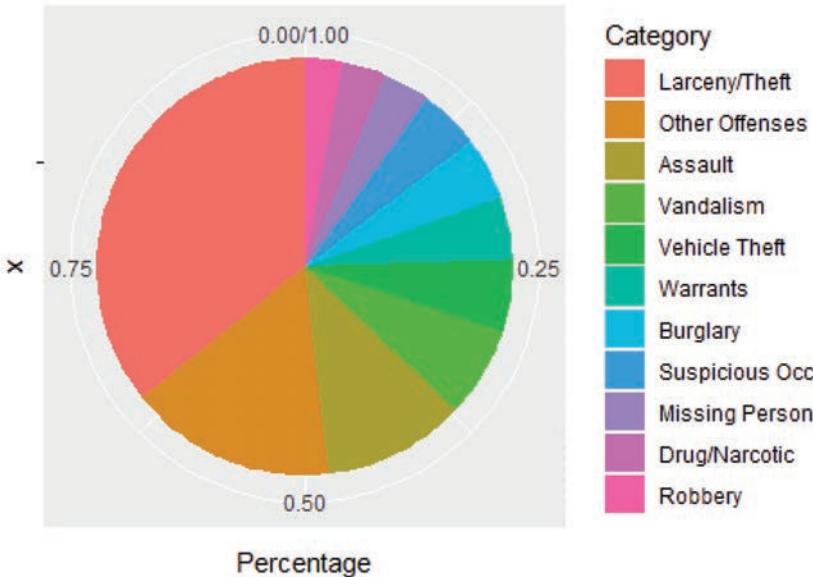


**Figure 9-10.** Histogram of the top crimes in Atlanta

### Creating a pie chart

To further illustrate the crime incident data, a subsequent pie chart is plotted in **Figure 10-5**. The chart illustrates the same data as does the bar chart, but it may be more understandable in this instance. It shows that Larceny/Theft occurs more than twice as much as all other reported crimes taken together. The chart is also ascetically pleasing.

```
Bp <- ggplot(df_category, aes(x="", y=Percentage,
  fill=Category)) +
  geom_bar(stat="identity")
pie <- bp + coord_polar("y")
pie
```



**Figure 9-11.** Pie chart of the top crimes in Atlanta

### Temporal Trends

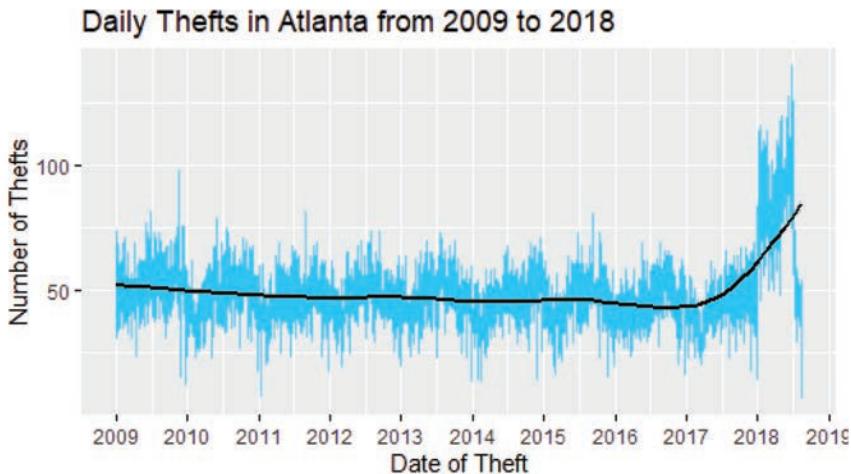
In this section, we create a chart of crimes (Larceny/Theft) over time. And for aesthetic effect as well as clarity, we make use color in depicting the series shown in **Figure 10-6**. This will provide us with a crime series for thefts, which we will smooth as we did before.

```
library(dplyr)
df <- read.csv(path)
df_theft <- df %>% filter(grepl("LARCENY/THEFT",Category))
df_theft_daily <- df_theft %>%
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%
  group_by(Date) %>%
  summarize(count = n()) %>%
  arrange(Date)
```

```

plot <- ggplot(df_theft_daily, aes(x = Date, y = count)) +
  geom_line(color = "#00ccff", size = 0.1) +
  geom_smooth(color = "#1A1A1A") + # fte_theme() +
  scale_x_date(breaks = date_breaks("1 year"),
  labels = date_format("%Y")) +
  labs(x = "Date of Theft", y = "Number of Thefts",
  title="Daily Thefts in Atlanta from 2009 to 2018")
plot

```



**Figure 9-12.** Crime-series of daily thefts in Atlanta, 2009-2018

### Theft Time Heatmap

Now, we aggregate counts of thefts by **Day-of-Week** and **Time** to create a heat map. Fortunately, the Day-Of-Week part is pre-derived, but Hour is slightly harder. We need a function that gets the hour from the time string in `atlanta_crime_10yr.csv`, so that we can use an approximate arrest time with day of the week. But R does not have one, or one I can find. So, we build the function below, using the colon delimiter to separate hours from minutes. Then we build **Table 10-3** that allows us to check that the code is doing what we expected.

```

get_hour <- function(x) {
  return(as.numeric(strsplit(x,":")[[1]][1]))
}
df_theft_time <- df_theft %>%
  mutate(Hour = sapply(Time, get_hour)) %>%

```

```

group_by(DayOfWeek, Hour)  %>%
summarize(count = n())
# df_theft_time %>% head(10)
data.table(df_theft_time, options = list(scrollX='400px'))

```

**Table 9-3.** Atlanta crime frequencies by hour and day of the week

#	DayOfWeek	Hour	Count
1	Friday	0	145
2	Friday	1	57
3	Friday	2	5
4	Friday	3	3
5	Friday	4	6
6	Friday	5	35
7	Friday	6	1
8	Friday	7	3
9	Friday	8	4
10	Friday	9	3

### Reorder and format Factors

In this section, we demonstrate how to reorder and format factors using the aggregated data. For instance, the `rev` function reverses elements so that the days of the week are “Saturday”, “Friday”, “Thursday”, “Wednesday”, “Tuesday”, “Monday” and “Sunday.” We use the `factor` function to encode a vector of times as a factor (the terms ‘category’ and ‘enumerated type’ are also used for factors)., thereby using the hours 12AM through 11PM for `Time`, as shown in the **Table 10-4**. This is the same data as in the previous table, but generates as a `kable` table, using the `knitr`-package.

```

dow_format <- c("Sunday", "Monday", "Tuesday", "Wednesday",
              "Thursday", "Friday", "Saturday")
hour_format <- c(paste(c(12,1:11),"AM"),
                 paste(c(12,1:11),"PM"))
df_theft_time$DayOfWeek <- factor(df_theft_time$DayOfWeek,
                                     level = rev(dow_format))
df_theft_time$Hour <- factor(df_theft_time$Hour,

```

```

    level = 0:23, label = hour_format)
#data.table(df_theft_time, options=list(scrollX='400px'))
knitr::kable(head(df_theft_time,11),
             caption = 'Thefts Recorded by Times')

```

**Table 9-4.** Atlanta crime frequencies by time and day of the week

	DayOfWeek	time	Time	Count
1	Friday	0	12 AM	145
2	Friday	1	1 AM	57
3	Friday	2	2 AM	5
4	Friday	3	3 AM	3
5	Friday	4	4 AM	6
6	Friday	5	5 AM	35
7	Friday	6	6 AM	1
8	Friday	7	7 AM	3
9	Friday	8	8 AM	4
10	Friday	9	9 AM	3

### *Creating a Time Heatmap*

Using our previous results, we build a “heatmap” and plot it with a red color scheme as seen in **Figure 10-7**. If you have not noticed, most of the colors I am using are in hexadecimal (hex) numbers, like “#000000” instead of black. A great interactive website to get colors with hex number is for any color is <https://www.colorhexa.com/000000>. The website also suggests “web safe colors” as alternatives.

```

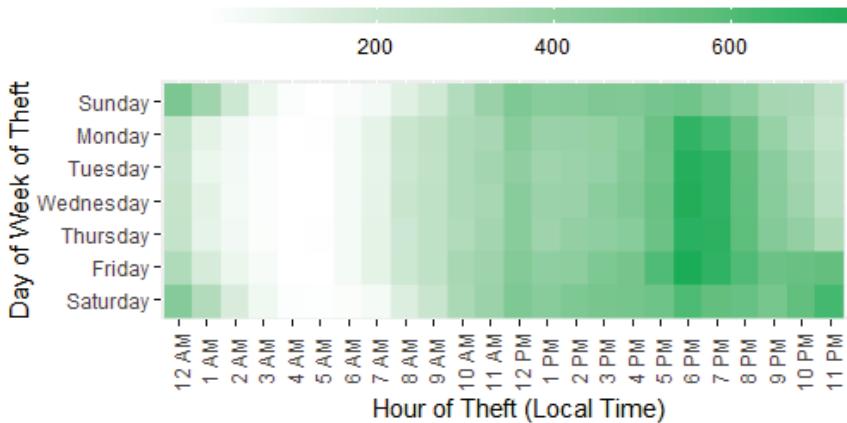
plot <- ggplot(df_theft_time, aes(x = Hour,
                                    y = DayOfWeek, fill = count)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90,
                                    vjust = 0.6),
        legend.title = element_blank(),

```

```

legend.position = "top",
legend.direction = "horizontal",
legend.key.width = unit(2, "cm"),
legend.key.height = unit(0.25, "cm"),
legend.margin = unit(-0.5,"cm"),
panel.margin = element_blank()) +
labs(x = "Hour of Theft (Local Time)",
y = "Day of Week of Theft",
title = "Number of Thefts in Atlanta from 2009 to
2018, by Time of Theft") +
scale_fill_gradient(low = "white", high = "#27AE60",
labels = comma)
plot

```



**Figure 9-13.** Heat map of the numbers of thefts in Atlanta, 2009-2018

Note that most of the code is for the legend and its formatting. The line of code that gives the heatmap its “heat” is the `scale_fill_gradient()` function with its low and high intensity fill colors.

The graph brings up a question: why is there a surge at 6-7 PM on weekdays? Note that Saturday and Sunday are in the middle running and the time axis is in 24-hour time. Law enforcement crime experts would probably be able to explain the “heat,” but without seeing the information provided by the data, they may not realize that 6-7 PM on weekdays is an issue.

## Arrests Over Time

Now, we create **Table 10-5** of arrests over time. First, we setup the data to get arrest counts by date. Then we plot the number of thefts given the date of the theft.

```
df_arrest <- df %>% filter(grepl("ARREST", Resolution))
df_arrest_daily <- df_arrest %>%
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%
  group_by(Date) %>%
  summarize(count = n()) %>%
  arrange(Date)
datatable(df_arrest_daily, options=list(scrollX='400px'))
```

**Table 9-5. Count of Arrests by date (daily), 2009-2018**

	Date	Count
1	2009-01-01	101
2	2009-01-02	117
3	2009-01-03	91
4	2009-01-04	89
5	2009-01-05	113
6	2009-01-06	115
7	2009-01-07	103
8	2009-01-08	86
9	2009-01-09	105
10	2009-01-10	105

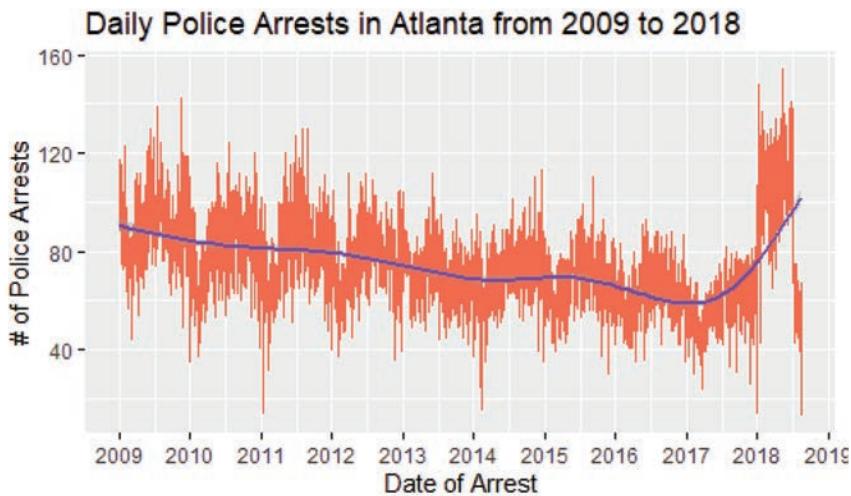
## Daily Arrests

Next, we build the plot shown in **Figure 10-8** of daily theft arrests by police or theft arrests over time. This will provide us with another crime series and we will smooth it as usual, noticing a downward trend in arrests over time until a sharp increase starting around 2017.

```

library(ggplot2)
library(scales)
plot <- ggplot(df_arrest_daily,
  aes(x = Date, y = count)) +
  geom_line(color = "#F2CA27", size = 0.1) +
  geom_smooth(color = "#1A1A1A") +
  # fte_theme() +
  scale_x_date(breaks = date_breaks("1 year"),
    labels = date_format("%Y")) +
  labs(x = "Date of Arrest", y = "# of Police Arrests",
    title = "Daily Police Arrests in Atlanta from 2009
    to 2018")
plot

```



**Figure 9-14.** Crime-series of daily police arrests in Atlanta, 2009-2018

### Number of Arrest by Time of Arrest

Here, we again use the function we created that gets the hour from the time string in [atlanta\\_crime\\_10yr.csv](#), so that we can use an approximate arrest time with the day of the week as shown in **Table 10-6**. This allows us to bin the crimes by hour, etc. Using **Figure 10-9**, most arrests are made on Thursdays and Fridays during the midnight hour. The fact that arrests seem to occur around 12AM, 5AM, 10AM, 3PM, and 8PM may simply be shift changes and perhaps the reporting

times correspond with the shift changes. At any rate, the resulting graph is not very informative. Consequently, we need to manipulate the data to produce plots that provide actionable information for consumption by law enforcement officials.

```
get_hour <- function(x) {
  return (as.numeric(strsplit(x,":"))[[1]][1]))
}
df_arrest_time <- df_arrest %>%
  mutate(Hour = sapply(Time, get_hour)) %>%
  group_by(DayOfWeek, Hour) %>%
  summarize(count = n())
dow_format <- c("Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday")
hour_format <- c(paste(c(12,1:11), "AM"),
  paste(c(12,1:11), "PM"))
df_arrest_time$DayOfWeek <- factor(df_arrest_time$DayOfWeek,
  level = rev(dow_format))
df_arrest_time$Hour <- factor(df_arrest_time$Hour, level =
  0:23, label = hour_format)
datatable(df_arrest_time, options = list(pageLength = 10, s
crollX='400px'))
```

**Table 9-6.** Arrest frequency table by day-of-week and hour (time)

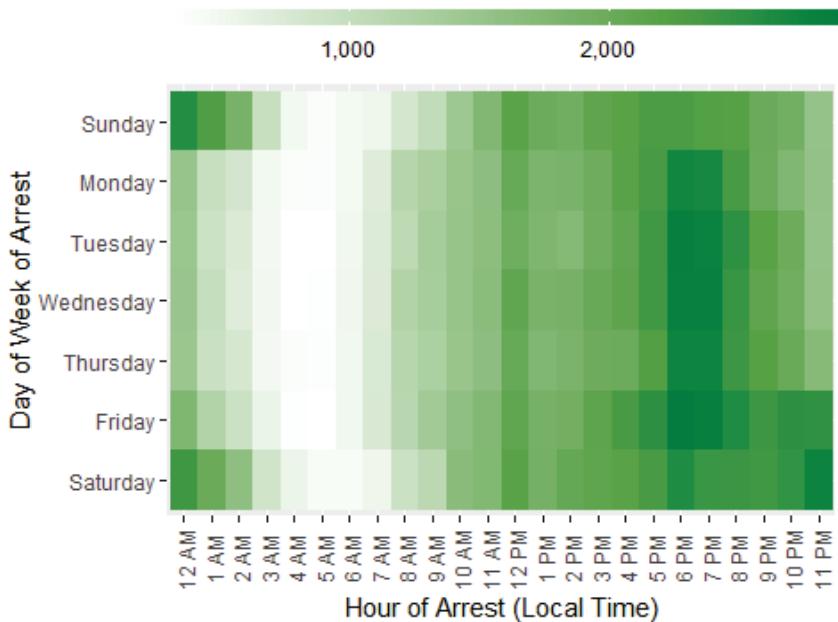
	Category	DayOfWeek	Hour	Count
1	Agg-Assault	Friday	12 AM	6
2	Agg-Assault	Friday	1 AM	6
3	Agg-Assault	Friday	2 AM	1
4	Agg-Assault	Friday	3 AM	1
5	Agg-Assault	Friday	5 AM	1
6	Agg-Assault	Friday	6 AM	1
7	Agg-Assault	Friday	7 AM	1
8	Agg-Assault	Friday	9 AM	1
9	Agg-Assault	Friday	10 AM	10
10	Agg-Assault	Friday	12 PM	2

```
plot <- ggplot(df_arrest_time, aes(x = Hour,
  y = DayOfWeek, fill = count)) +
  geom_tile() +
  # fte_theme() +
  theme(axis.text.x = element_text(angle = 90,
  vjust = 0.6),
  legend.title = element_blank(),
  legend.position = "top",
  legend.direction = "horizontal",
  legend.key.width = unit(2, "cm"),
  legend.key.height = unit(0.25, "cm"),
  legend.margin = unit(-0.5,"cm"),
  panel.margin = element_blank()) +
  labs(x = "Hour of Arrest (Local Time)",
  y = "Day of Week of Arrest",
  title = "Number of Police Arrests in Atlanta from
  2009 - 2018, by Time of Arrest") +
  scale_fill_gradient(low = "white",
  high = "#008000", labels = comma)
plot
```

### Markdown Note.

To generate the R markdown that yields the heatmap from the above code is easy. Just wrap it with the “code-chunk” code:

```
```{r echo=TRUE
plot <- ggplot...
#(plot code, parameters, etc.)
plot
```
```
```



**Figure 9-15.** Heat map of the number of police arrests in Atlanta, 2009-2018

From the figure, most arrests are made on Thursdays and Fridays during the midnight hour. The fact that arrests seem to occur around 12AM, 5 AM, 10AM, 3PM, and 8PM may simply be shift changes and perhaps the reporting times correspond with the shift changes.

Why is there a surge on Wednesday afternoon, and at 4-5PM on all days? Let's look at subgroups to verify there isn't a latent factor.

### Factor by Crime Category

Certain types of crime may be more time dependent. (e.g., more traffic violations when people leave work). While we are interested in crime frequencies as shown in **Table 10-7**, we can gain more information from the data. For instance, in **Table 10-8**. Arrest frequency table by day-of-week and hour (time), we look at frequency of the crime category per hour.

```
df_top_crimes <- df_arrest %>%
  group_by(Category)  %>%
```

```

    summarize(count = n()) %>%
    arrange(desc(count))
data.table(df_top_crimes, options = list(pageLength = 10, s
crollX='400px'))

```

**Table 9-7.** Atlanta's Top Crimes Table, 2009-2018

	Category	count
1	Larceny/Theft	172657
2	Burglary-Residence	47483
3	Auto Theft	44247
4	Agg-Assault	23244
5	Robbery-Pedestrian	16488
6	Burglary-Nonres	10233
7	Robbery-Commercial	2145
8	Robbery-Residence	2122
9	Homicide	886
10	Rape	141

```

df_arrest_time_crime <- df_arrest %>%
  filter(Category %in%
df_top_crimes$Category[2:19]) %>%
  mutate(Hour = sapply(Time, get_hour))  %>%
  group_by(Category, DayOfWeek, Hour) %>%
  summarize(count = n())
df_arrest_time_crime$DayOfWeek <- factor(
  df_arrest_time_crime$DayOfWeek, level = rev(dow_format))
df_arrest_time_crime$Hour <- factor(df_arrest_time_crime$H
our, level = 0:23, label = hour_format)
datatable(df_arrest_time_crime, options = list(pageLength
= 10, scrollX='400px'))

```

**Table 9-8.** Arrest frequency table by day-of-week and hour (time)

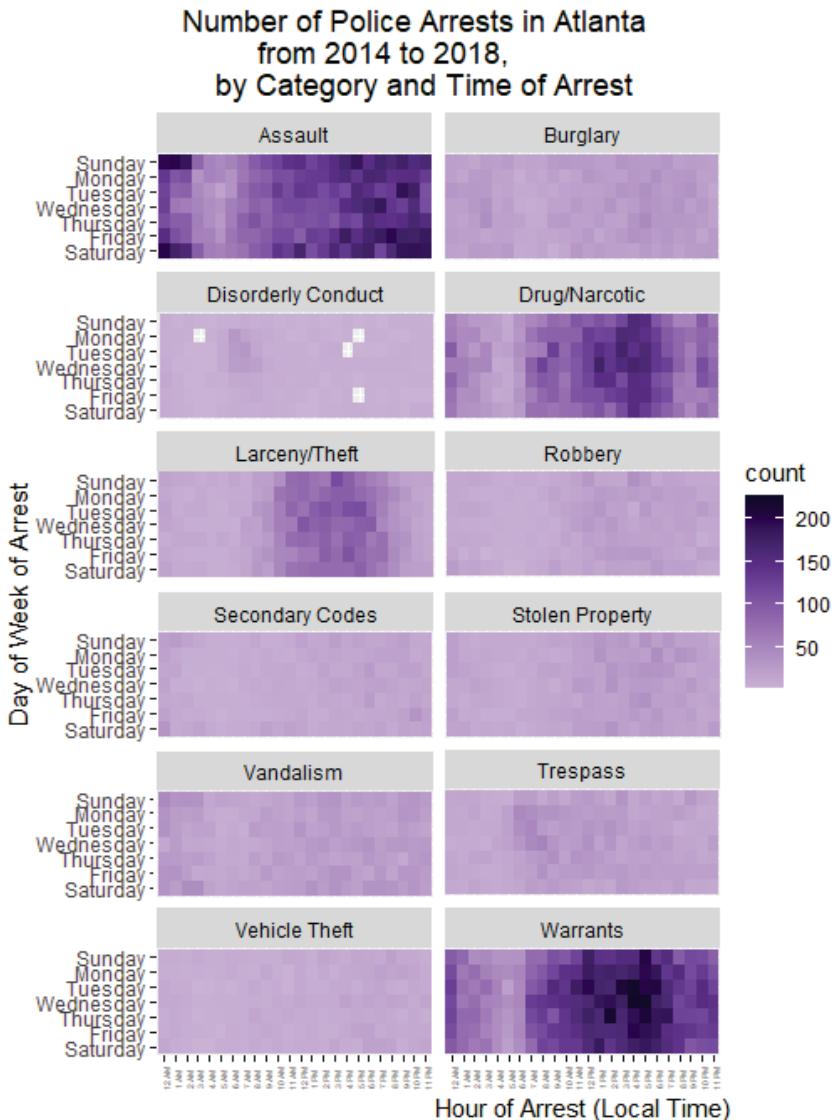
Category	DayOfWeek	Hour	count
AGG_ASSAULT	Friday	12 AM	143
AGG_ASSAULT	Friday	1 AM	135

AGG_ASSAULT	Friday	2 AM	110
AGG_ASSAULT	Friday	3 AM	52
AGG_ASSAULT	Friday	4 AM	35
AGG_ASSAULT	Friday	5 AM	28
AGG_ASSAULT	Friday	6 AM	33
AGG_ASSAULT	Friday	7 AM	49
AGG_ASSAULT	Friday	8 AM	58
AGG_ASSAULT	Friday	9 AM	64
AGG_ASSAULT	Friday	10 AM	50

### *Arrests by Category and time of Arrest*

In this section, we plot the number of arrests by category and time of arrest. This leads us to use a chart type that you may not have seen very often. We take the heat map application from the previous section and plot all the crime heat maps in one aggregated chart as seen in **Figure 10-10**. For each heat map in the chart, the horizontal axes are “hours of arrest (local time),” and the vertical axes are “days of the week.”

```
plot <- ggplot(df_arrest_time_crime, aes(x = Hour,
   y = DayOfWeek, fill = count)) +
  geom_tile() +
  # fte_theme() +
  theme(axis.text.x = element_text(angle = 90,
                                    vjust = 0.6, size = 4)) +
  labs(x = "Hour of Arrest (Local Time)",
       y = "Day of Week of Arrest",
       scale_fill_gradient(low = "#d7b4ff",
                           high = "#24004b") +
  facet_wrap(~ Category, nrow = 6)
plot
```



**Figure 9-16.** Heatmaps of police arrests for major crimes in Atlanta, 2014-2018

This graph looks good, but the gradients aren't helpful because they are not normalized. AS used here, normalization refers to adjustments in the measured scale where the intention is to bring the entire probability distributions of adjusted values into alignment. In this way, we can make one-to-one comparisons. We need to

normalize the range on each facet as we do for **Table 10-9** and the corresponding figure.

### Normalized Gradients

```
df_arrest_time_crime <- df_arrest_time_crime %>%
  group_by(Category) %>%
  mutate(norm = count/sum(count))
#data.table(df_arrest_time_crime, options = list(pageLength = 10, scrollX='400px'))
knitr::kable(head(df_arrest_time_crime,11) ,
  caption = 'Crime Arrest Times')
```

**Table 9-9.** Atlanta crimes by date-time with count and normalized count (showing aggravated assault)

Category	DayOfWeek	Hour	count	norm
AGG_ASSAULT	Friday	12 AM	143	0.0096556
AGG_ASSAULT	Friday	1 AM	135	0.0091155
AGG_ASSAULT	Friday	2 AM	110	0.0074274
AGG_ASSAULT	Friday	3 AM	52	0.0035111
AGG_ASSAULT	Friday	4 AM	35	0.0023633
AGG_ASSAULT	Friday	5 AM	28	0.0018906
AGG_ASSAULT	Friday	6 AM	33	0.0022282
AGG_ASSAULT	Friday	7 AM	49	0.0033086
AGG_ASSAULT	Friday	8 AM	58	0.0039163
AGG_ASSAULT	Friday	9 AM	64	0.0043214
AGG_ASSAULT	Friday	10 AM	50	0.0033761

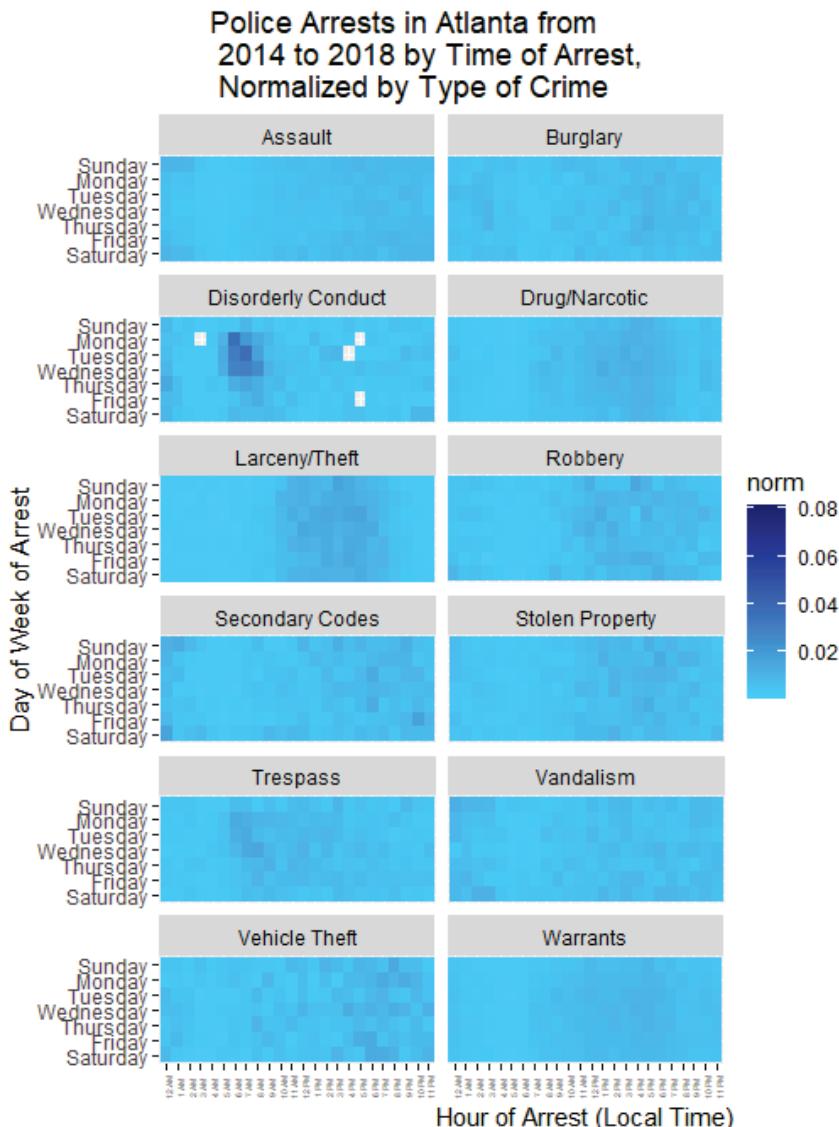
We normalized the number of arrests by category and time of arrest and plotted them in **Figure 10-11**.

```
plot <- ggplot(df_arrest_time_crime, aes(x = Hour, y = Day
OfWeek, fill = norm)) + geom_tile() +
  theme(axis.text.x = element_text(angle = 90,
  vjust = 0.6, size = 4)) +
  labs(x = "Hour of Arrest (Local Time)",
  y = "Day of Week of Arrest",
  title = "Police Arrests in Atlanta from 2014 to 2018 by")
```

```

Time of Arrest, Normalized by Type of Crime") +
  scale_fill_gradient(low = "#4dd2ff",
                      high = "#00008b") +
  facet_wrap(~ Category, nrow = 6)
plot

```



**Figure 9-17.** Heatmaps of police arrests, normalized by major crimes in Atlanta, 2014-2018

## *Factor by Police District*

In this section, we plot like we did for **Figure 10-11**, but with a different scope. In **Table 10-10** and its corresponding Figure 4-12, we want the normalized frequency of arrest from each PD district (or zone in this case) by day-of-week and hour.

```
df_arrest_time_district <- df_arrest %>%
  mutate(Hour = sapply(Time, get_hour)) %>%
  group_by(PdDistrict, DayOfWeek, Hour) %>%
  summarize(count = n()) %>%
  group_by(PdDistrict) %>%
  mutate(norm = count/sum(count))

df_arrest_time_district$DayOfWeek <- factor(df_arrest_time_district$DayOfWeek, level = rev(dow_format))
df_arrest_time_district$Hour <- factor(df_arrest_time_district$Hour, level = 0:23, label = hour_format)
knitr::kable(head(df_arrest_time_district, 11), caption = 'Crime Arrest Times by District')
```

**Table 9-10. Crime Arrest Times by District**

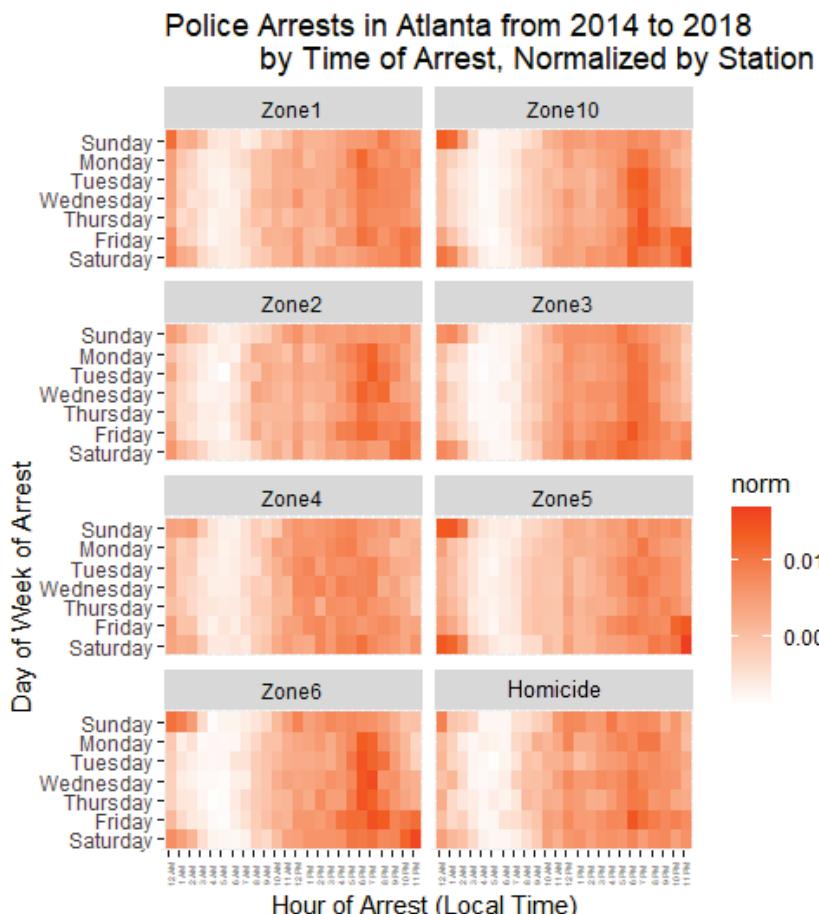
PdDistrict	DayOfWeek	Hour	count	norm
Zone1	Friday	12 AM	184	0.0082522
Zone1	Friday	1 AM	90	0.0040364
Zone1	Friday	2 AM	78	0.0034982
Zone1	Friday	3 AM	49	0.0021976
Zone1	Friday	4 AM	30	0.0013455
Zone1	Friday	5 AM	40	0.0017940
Zone1	Friday	6 AM	45	0.0020182
Zone1	Friday	7 AM	79	0.0035431
Zone1	Friday	8 AM	83	0.0037225
Zone1	Friday	9 AM	136	0.0060995
Zone1	Friday	10 AM	131	0.0058752

Now, we factor by police district and plot in **Figure 10-12**.

```

plot <- ggplot(df_arrest_time_district, aes(x = Hour, y = DayOfWeek, fill = norm)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90,
    vjust = 0.6, size = 4)) +
  labs(x = "Hour of Arrest (Local Time)",
    y = "Day of Week of Arrest",
    scale_fill_gradient(low = "white",
    high = "#ff4500") +
  facet_wrap(~ PdDistrict, nrow = 4)
plot

```



**Figure 9-18.** Heatmaps of police arrests for major crimes in Atlanta, by zones, 2014-2018

## Factor by Month

We now look at factor by month. If crime is tied to activities, the period at which activities end may impact.

```
df_arrest_time_month <- df_arrest %>%
  mutate(Month = format(as.Date(Date, "%m/%d/%Y"),
    "%B"), Hour = sapply(Time, get_hour)) %>%
  group_by(Month, DayOfWeek, Hour) %>%
  summarize(count = n()) %>%
  group_by(Month) %>%
  mutate(norm = count/sum(count))
```

Here, we set order of month facets by chronological order instead of alphabetical.

```
df_arrest_time_month$DayOfWeek <-
  factor(df_arrest_time_month$DayOfWeek,
    level = rev(dow_format))
df_arrest_time_month$Hour <-
  factor(df_arrest_time_month$Hour,
    level = 0:23, label = hour_format)
df_arrest_time_month$Month <-
  factor(df_arrest_time_month$Month,
    level = c("January", "February", "March", "April",
      "May", "June", "July", "August", "September",
      "October", "November", "December"))
```

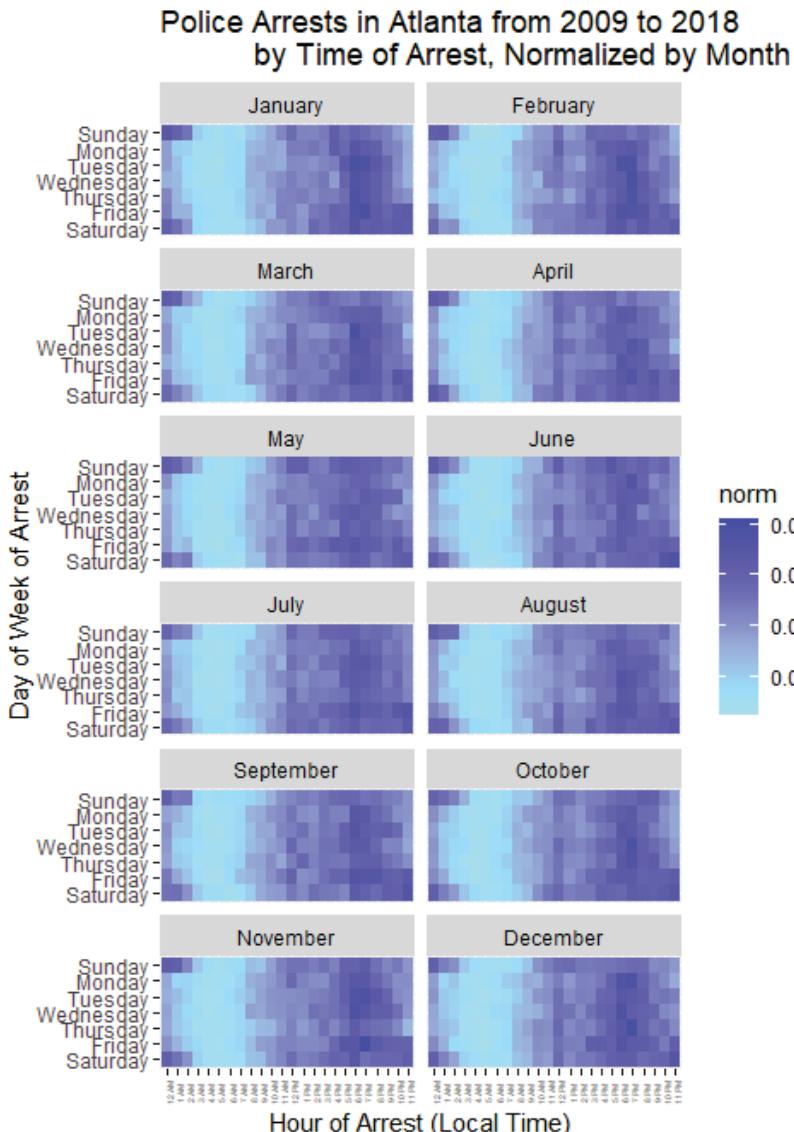
Now we plot the data as shown below and seen in **Figure 10-13**.

```
plot <- ggplot(df_arrest_time_month,
  aes(x = Hour, y = DayOfWeek, fill = norm)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90,
    vjust = 0.6, size = 4)) +
  labs(x = "Hour of Arrest (Local Time)",
    y = "Day of Week of Arrest",
    title = "Police Arrests in Atlanta from 2009
      to 2018 by Time of Arrest,
      Normalized by Month") +
  scale_fill_gradient(low = "#9bfdff", high =
```

```

      "#4401ff") +
  facet_wrap(~ Month, nrow = 4)
plot

```



**Figure 9-19.** Heatmaps of police arrests for major crimes in Atlanta, normalized by month 2014-2018

## Markdown Note.

R markdown code is as unforgiving as any other code when it comes to syntax, so pay attention to ` `` versus ' '' . The former is part of the ~ key on your keyboard. Also, R does not like smart (curly) quotes.

### Factor By Year

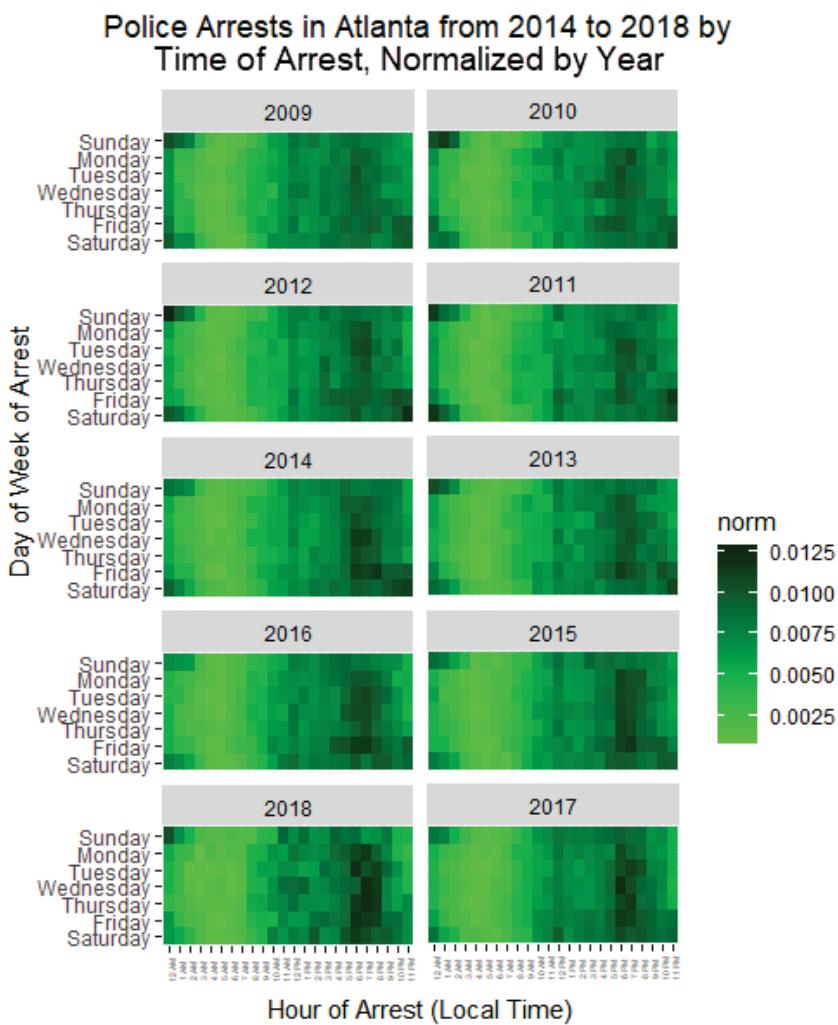
What if things changed overtime?

```
df_arrest_time_year <- df_arrest %>%
  mutate(Year = format(as.Date(Date, "%m/%d/%Y"), "%Y"),
        Hour = sapply(Time, get_hour)) %>%
  group_by(Year, DayOfWeek, Hour) %>%
  summarize(count = n()) %>%
  group_by(Year) %>%
  mutate(norm = count/sum(count))
df_arrest_time_year$DayOfWeek <- factor(df_arrest_time_yea
r$DayOfWeek, level = rev(dow_format))
df_arrest_time_year$Hour <- factor(df_arrest_time_year$Hou
r, level = 0:23, label = hour_format)
```

### Police Arrest Normalized by Year

In similar fashion, we can look at the arrests by year aggregated over day-of-week and time-of-day, as seen in **Figure 10-14**.

```
plot <- ggplot(df_arrest_time_year, aes(x = Hour, y = DayO
fWeek, fill = norm)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90,
    vjust = 0.6, size = 4)) +
  labs(x = "Hour of Arrest (Local Time)",
    y = "Day of Week of Arrest",
    title = "Police Arrests in Atlanta from 2014 to 201
    8 by Time of Arrest, Normalized by Year") +
  scale_fill_gradient(low = "#01ff44",
    high = "#00340e") +
  facet_wrap(~ Year, nrow = 6)
plot
```



**Figure 9-20.** Heatmaps of police arrests for major crimes in Atlanta, normalized by year 2014-2018



## 10. ML Ensembles - Random Forests

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman (2001) and Adele Cutler (2012), and "Random Forests" is their trademark. The term came from random decision forests that was first proposed by Tin Kam Ho of Bell Labs in 1995. The method combines Breiman's "bagging" idea and the random selection of features, introduced independently by Ho (1995) (1998) and Amit and Geman (1997) in order to construct a collection of decision trees with controlled variance. RFs are also *supervised learning* models.

The selection of a random subset of features is an example of the random subspace method, which, in Ho's formulation, is a way to implement classification proposed by Eugene Kleinberg (1996).

### History

The early development of random forests was influenced by the work of Amit and Geman (1997), which introduced the idea of searching over a random subset of the available decisions when splitting a node, in the context of growing a single tree. The idea of random subspace selection from Ho (1998) was also influential in the design of random forests. In this method a forest of trees is grown, and variation among the trees is introduced by projecting the training data into a randomly chosen subspace before fitting each tree. Finally, the idea of randomized node optimization, where the decision at each node is selected by a randomized procedure, rather than a deterministic optimization was first introduced by Dietterich (2000).

The introduction of random forests proper was first made in a paper by Leo Breiman (2001). This paper describes a method of building a forest of uncorrelated trees using a CART like procedure, combined with randomized node optimization and bagging. In addition, this paper combines several ingredients, some previously known and

some novel, which form the basis of the modern practice of random forests, in particular:

1. Using out-of-bag error as an estimate of the generalization error.
2. Measuring variable importance through permutation.

The report also offers the first theoretical result for random forests in the form of a bound on the generalization error which depends on the strength of the trees in the forest and their correlation.

More recently several major advances in this area have come from Microsoft Research (Criminisi, Shotton, & Konukoglu, 2011), which incorporate and extend the earlier work from Breiman.

## Algorithm

The training algorithm for random forests applies the general technique of bootstrap aggregating (see Bootstrap aggregating), or bagging, to tree learners. Given a training set  $X = x_1, \dots, x_n$  with responses  $Y = y_1$  through  $y_n$ , bagging repeatedly selects a bootstrap sample of the training set and fits trees to these samples:

For  $b = 1$  through  $B$ :

1. Sample, with replacement,  $n$  training examples from  $X, Y$ ; call these  $X_b, Y_b$ .
2. Train a decision or regression tree  $f_b$  on  $X_b, Y_b$ .

After training, predictions for unseen samples  $x'$  can be made by averaging the predictions from all the individual regression trees on  $x'$ :

$$\hat{f} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x'),$$

or by taking the majority vote in the case of decision trees.

In the above algorithm,  $B$  is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. Increasing the number of trees tends to decrease the variance of the model, without increasing the bias. As

a result, the training and test error tend to level off after some numbers of trees have been fit. An optimal number of trees  $B$  can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample  $x_i$ , using only the trees that did not have  $x_i$  in their bootstrap sample (James, Witten, Hastie, & Tibshirani, 2013).

### ***Bootstrap aggregating***

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

Bagging was proposed by Leo Breiman in 1994 to improve the classification by combining classifications of randomly generated training sets (Breiman L. , 1994).

### ***Description of the technique***

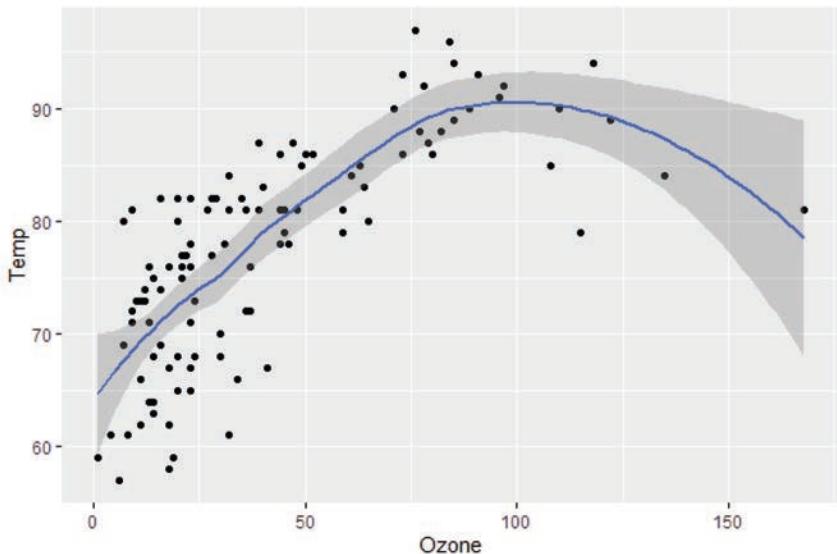
Given a standard training set  $D$  of size  $n$ , bagging generates  $m$  new training sets  $D_i$ , each of size  $n'$ , by sampling from  $D$  uniformly and with replacement. By sampling with replacement, some observations may be repeated in each. If  $n' = n$ , then for large  $n$  the set is expected to have the fraction  $(1 - 1/e)$  ( $\approx 63.2\%$ ) of the unique examples of  $D$ , the rest being duplicates (Aslam, Popa, & Rivest, 2007). This kind of sample is known as a bootstrap sample. The  $m$  models are fitted using the above  $m$  bootstrap samples and combined by averaging the output (for regression) or voting (for classification). Bagging leads to "improvements for unstable procedures" (Breiman L. , Random Forests, 2001), which include, for example, neural nets, classification and regression trees, and subset selection in linear regression (Breiman L. , Bagging predictors, 1996). An interesting application of bagging showing improvement in preimage learning is provided here (Sahu, Runger, & Apley, 2011) (Shinde, Sahu, Apley, & Runger, 2014). On the other hand, it can mildly degrade the performance of stable

methods such as  $K$ -nearest neighbors (Breiman L., Bagging predictors, 1996).

### Example 10-1: Ozone data

To show the basic principles of bagging, we use an analysis of the relationship between temperature and ozone (data from Rousseeuw and Leroy (2003), available at classic data sets, analysis done in R).

The relationship between temperature and ozone in the air quality data set is seemingly non-linear, based on the scatter plot. To mathematically describe this relationship, LOESS smoothers (with span 0.5) are used. First, we look at a simple smoother in **Figure 10-1**.



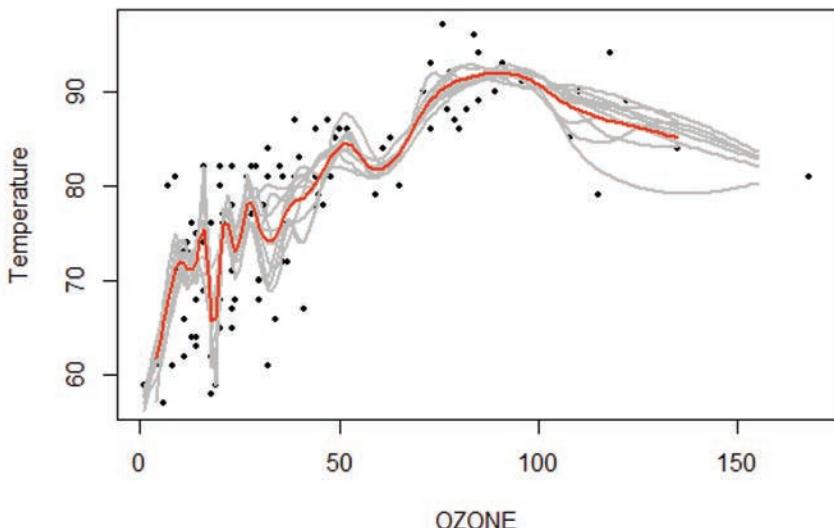
**Figure 10-1.** Ozone levels versus temperature data with LOESS smoother

Rather than constructing a single smoother from the complete data set, we drew 100 bootstrap samples of the data. Each sample is different from the original data set yet is similar to it in distribution and variability. A LOESS smoother was fit for each bootstrap sample. We then made Predictions from these 100 smoothers across the range of the data. The first 10 predicted smooth fits appear as grey

lines in the figure below. The lines are clearly very wiggly and they overfit the data—a result of the span being too low. But taking the average of 100 smoothers, each fitted to a subset of the original data set, we arrive at one bagged predictor (red line) in **Figure 10-2**. Clearly, the mean is more stable and there is less overfit.

### Bagging for nearest neighbor classifiers

It is well known that the risk of a 1 nearest neighbor (1NN) classifier is at most twice the risk of the Bayes classifier, but there are no guarantees that this classifier will be consistent. By careful choice of the size of the resamples, bagging can lead to substantial improvements of the performance of the 1NN classifier. By taking a large number of resamples of the data of size  $n'$ , the bagged nearest neighbor classifier will be consistent provided  $n' \rightarrow \infty$  diverges but  $n'/n \rightarrow 0$  as the sample size  $n \rightarrow \infty$ .



**Figure 10-2.** Ozone levels versus temperature data with LOESS smoothers for 100 bootstrap samples and one bagged predictor

Under infinite simulation, the bagged nearest neighbor classifier can be viewed as a weighted nearest neighbor classifier. Suppose that the feature space is  $d$  dimensional and denote by  $C_{n,n'}^{bnn}$  the bagged

nearest neighbor classifier based on a training set of size  $n$ , with resamples of size  $n'$ . In the infinite sampling case, under certain regularity conditions on the class distributions, the excess risk has the following asymptotic expansion

$$\mathcal{R}_{\mathcal{R}}(C_{n,n'}^{bnn}) - \mathcal{R}_{\mathcal{R}}(C^{bayes}) = \left( B_1 \frac{n'}{n} + B_2 \frac{1}{(n')^{4/d}} \right) \{1 + o(1)\},$$

for some constants  $B_1$  and  $B_2$ . The optimal choice of  $n'$ , that balances the two terms in the asymptotic expansion, is given by  $n' = Bn^{d/(d+4)}$  for some constant  $B$ .

## From bagging to random forests

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the  $B$  trees, causing them to become correlated.

Typically, for a dataset with  $p$  features,  $\sqrt{p}$  features are used in each split.

### *Random subspace method*

Random subspace method (or attribute bagging (Bryll, 2003)) is an ensemble classifier that consists of several classifiers and outputs the class based on the outputs of these individual classifiers. Random subspace method is a generalization of the random forest algorithm (Ho T. , 1998). Whereas random forests are composed of decision trees, a random subspace classifier can be composed from any underlying classifiers. Random subspace method has been used for linear classifiers (Skurichina, 2002), support vector machines (Tao, 2006), nearest neighbors (Tremblay, 2004) and other types of classifiers. This method is also applicable to one-class classifiers.

The algorithm is an attractive choice for classification problems where the number of features is much larger than the number of training objects, such as fMRI data or gene expression data (Kuncheva, Rodríguez, Plumpton, Linden, & Johnston, 2010).

### ***Algorithm***

The ensemble classifier is constructed using the following algorithm:

1. Let the number of training objects be  $N$  and the number of features in the training data be  $D$ .
2. Choose  $L$  to be the number of individual classifiers in the ensemble.
3. For each individual classifier,  $l$ , Choose  $d_l$  ( $d_l < D$ ) to be the number of input variables for  $l$ . It is common to have only one value of  $d_l$  for all the individual classifiers
4. For each individual classifier,  $l$ , create a training set by choosing  $d_l$  features from  $D$  without replacement and train the classifier.
5. For classifying a new object, combine the outputs of the  $L$  individual classifiers by majority voting or by combining the posterior probabilities.

### ***Relationship to Nearest Neighbors***

Given a set of training data

$$\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$$

a weighted neighborhood scheme makes a prediction for a query point  $X$ , by computing

$$\hat{Y} = \sum_{i=1}^n W_i(X) Y_i,$$

for some set of non-negative weights  $\{W_i(X)\}_{i=1}^n$  which sum to 1. The set of points  $X_i$  where  $W_i(X) > 0$  are called the neighbors of  $X$ . A common example of a weighted neighborhood scheme is the  $k$ -NN algorithm which sets  $W_i(X) = 1/k$  if  $X_i$  is among the  $k$  closest points to  $X$  in  $\mathcal{D}_n$  and 0 otherwise.

Random forests with constant leaf predictors can be interpreted as a weighted neighborhood scheme in the following way. Given a forest of  $M$  trees, the prediction that the  $m$ -th tree makes for  $X$  can be written as

$$T_m(X) = \sum_{i=1}^n W_{im}(X) Y_i,$$

where  $W_{im}(X)$  is equal to  $1/k_m$  if  $X$  and  $X_i$  are in the same leaf in the  $m$ -th tree and 0 otherwise, and  $k_m$  is the number of training data which fall in the same leaf as  $X$  in the  $m$ -th tree. The prediction of the whole forest is

$$F(X) = \sum_{i=1}^n T_m(X) = \frac{1}{M} \sum_{m=1}^M \sum_{i=1}^n W_{im}(X) Y_i = \sum_{i=1}^n \left( \frac{1}{M} \sum_{m=1}^M W_{im}(X) \right) Y_i,$$

which shows that the random forest prediction is a weighted average of the  $Y_i$ 's, with weights

$$W_i(X) = \frac{1}{M} \sum_{m=1}^M W_{im}(X).$$

The neighbors of  $X$  in this interpretation are the points  $X_i$  which fall in the same leaf as  $X$  in at least one tree of the forest. In this way, the neighborhood of  $X$  depends in a complicated way on the structure of the trees, and thus on the structure of the training set.

This connection was first described by Lin and Jeon in a technical report from 2001 where they show that the shape of the neighborhood used by a random forest adapts to the local importance of each feature (Lin & Jeon, 2001).

## Variable importance

Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. The following technique was described in Breiman's original paper (2001) and is implemented in the R package **randomForest** (Liaw, 2012).

The first step in measuring the variable importance in a data set  $\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$  is to fit a random forest to the data. During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest (errors on an independent test set can be substituted if bagging is not used during training).

To measure the importance of the  $j$ -th feature after training, the values of the  $j$ -th feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set. The importance score for the  $j$ -th feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees. The score is normalized by the standard deviation of these differences.

Features which produce large values for this score are ranked as more important than features which produce small values.

This method of determining variable importance has some drawbacks. For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Methods such as partial permutations can be used to solve the problem (Deng, Runger, & Tuv, 2011) (Altmann, Tolosi, Sander, & Lengauer, 2010). If the data contain groups of correlated features of similar relevance for the output, then smaller groups are favored over larger groups (Tolosi & Lengauer, 2011).

## Variants

Instead of decision trees, linear models have been proposed and evaluated as base estimators in random forests, in particular multinomial logistic regression and naïve Bayes classifiers (Prinzie & Van den Poel, 2008).

### Example 10-2: Random Forest using R

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or

mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. The source data for this and the following examples are at:

[https://github.com/stricje1/random\\_forest](https://github.com/stricje1/random_forest).

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set  $X = x_1, \dots, x_n$  with responses  $Y = y_1, \dots, y_n$ , bagging repeatedly ( $B$  times) selects random samples with replacement of the training set and fits trees to these samples:

1. Each sample may be comprised of different mixes members of the population, i.e., sample will have some members in common and some will be different
2. For  $n = 1, \dots, N$ , and  $m = 1, \dots, M$ , there will be  $N$  training sets comprised of  $m$  members sampled with replacement
3. Each sample may be comprised of a different mix of variables from the original set  $X$
4. Each tree (regression or classification) will train using one of the  $X_i$  training sets
5. After training, predictions are made by either averaging of the prediction from individual trees or by majority vote in the case of classification trees

Random Forest algorithm is built in `randomForest` package of R and same name function allows us to use the Random Forest in R.

### ***Load libraries***

```
if(!require(randomForest)) install.packages("randomForest")
if(!require(colorspace)) install.packages("colorspace")
if(!require(reshape)) install.packages("reshape")
if(!require(ggplot2)) install.packages("ggplot2")

library(randomForest)
```

```
library(colorspace)
library(reshape)
library(ggplot2)
```

Some of the commonly used parameters of randomForest functions are

- **x**: Random Forest Formula
- **data**: Input data frame
- **ntree**: Number of decision trees to be grown
- **replace**: Takes True and False and indicates whether to take sample with/without replacement
- **sampsize**: Sample size to be drawn from the input data for growing decision tree
- **importance**: Whether independent variable importance in random forest be assessed
- **proximity**: Whether to calculate proximity measures between rows of a data frame

### ***Medical longevity Study of primary biliary cirrhosis (PBC)***

Data was obtained from a Mayo Clinic randomized trial in primary biliary cirrhosis (PBC) of the liver conducted between 1974 and 1984. A total of 424 PBC patients, referred to Mayo Clinic during that ten-year interval met eligibility criteria for the randomized placebo-controlled trial of the drug D-penicillamine (DPCA). The data and partial likelihood model is described in Fleming and Harrington (1991). The variables in the data set are:

- case number
- number of days between registration and the earlier of death, transplantation, or study analysis time in July 1986
- status: 0=alive, 1=liver transplant, 2=dead
- drug: 1=D-penicillamine, 2=placebo
- age in days
- sex: 0=male, 1=female
- presence of ascites: 0=no 1=yes
- presence of hepatomegaly 0=no 1=yes
- presence of spiders 0=no 1=yes

- presence of edema 0=no edema and no diuretic therapy for edema; .5 = edema present without diuretics, or edema resolved by diuretics; 1 = edema despite diuretic therapy
- serum bilirubin in mg/dl
- serum cholesterol in mg/dl
- albumin in gm/dl
- urine copper in ug/day
- alkaline phosphatase in U/liter
- SGOT in U/ml
- triglycerides in mg/dl
- platelets per cubic ml / 1000
- prothrombin time in seconds
- histologic stage of disease

```
library (ggRandomForests)
```

```
library(randomForestSRC)
data(pbc)
summary(pbc)
```

	days	status	treatment	age
##	Min. : 41	Min. :0.0000	Min. :1.000	Min. : 9598
##	1st Qu.:1093	1st Qu.:0.0000	1st Qu.:1.000	1st Qu.:15644
##	Median :1730	Median :0.0000	Median :1.000	Median :18628
##	Mean :1918	Mean :0.3852	Mean :1.494	Mean :18533
##	3rd Qu.:2614	3rd Qu.:1.0000	3rd Qu.:2.000	3rd Qu.:21273
##	Max. :4795	Max. :1.0000	Max. :2.000	Max. :28650
##			NA's :106	
	sex	ascites	hepatom	spiders
##	Min. :0.0000	Min. :0.00000	Min. :0.0000	Min. :0.0000
##	1st Qu.:1.0000	1st Qu.:0.00000	1st Qu.:0.0000	1st Qu.:0.0000
##	Median :1.0000	Median :0.00000	Median :1.0000	Median :0.0000
##	Mean :0.8947	Mean :0.07692	Mean :0.5128	Mean :0.2885
##	3rd Qu.:1.0000	3rd Qu.:0.00000	3rd Qu.:1.0000	3rd Qu.:1.0000
##	Max. :1.0000	Max. :1.00000	Max. :1.0000	Max. :1.0000
##		NA's :106	NA's :106	NA's :106
	edema	bili	chol	albumin
##	Min. :0.0000	Min. : 0.300	Min. : 120.0	Min. :1.960
##	1st Qu.:0.0000	1st Qu.: 0.800	1st Qu.: 249.5	1st Qu.:3.243
##	Median :0.0000	Median : 1.400	Median : 309.5	Median :3.530
##	Mean :0.1005	Mean : 3.221	Mean : 369.5	Mean :3.497
##	3rd Qu.:0.0000	3rd Qu.: 3.400	3rd Qu.: 400.0	3rd Qu.:3.770

```

## Max.    :1.0000  Max.    :28.000  Max.    :1775.0  Max.    :4.640
##   NA's    :134
##      copper          alk          sgot          trig
## Min.   : 4.00  Min.   : 289.0  Min.   : 26.35  Min.   :33.00
## 1st Qu.:41.25 1st Qu.: 871.5  1st Qu.: 80.60  1st Qu.:84.25
## Median :73.00  Median :1259.0  Median :114.70  Median :108.00
## Mean   :97.65  Mean   :1982.7  Mean   :122.56  Mean   :124.70
## 3rd Qu.:123.00 3rd Qu.:1980.0  3rd Qu.:151.90  3rd Qu.:151.00
## Max.   :588.00  Max.   :13862.4  Max.   :457.25  Max.   :598.00
## NA's   :108    NA's   :106    NA's   :106    NA's   :136
##      platelet       prothrombin      stage
## Min.   :62.0   Min.   : 9.00  Min.   :1.000
## 1st Qu.:188.5 1st Qu.:10.00  1st Qu.:2.000
## Median :251.0  Median :10.60  Median :3.000
## Mean   :257.0  Mean   :10.73  Mean   :3.024
## 3rd Qu.:318.0 3rd Qu.:11.10  3rd Qu.:4.000
## Max.   :721.0  Max.   :18.00  Max.   :4.000
## NA's   :11     NA's   :2     NA's   :6

```

The first thing we want to do is transform variable values: years to days; 0-1 to T-F, transform age to years, and impute missing statuses.

```

pbc1 <- pbc
pbc1$Years <- pbc$days/365
pbc1$age <- pbc$age/365
pbc1$status <- NULL
pbc1$status

## OUTPUT OMITTED

```

The output for the last step was omitted but you might want to print it out. Now we pull in the transformed data

### *Get transformed data*

```

pbc2 <- pbc1[,2:20]
head(pbc2)

```

	status	treatment	age	sex	ascites	hepatom	spiders	edema	bili	chol	
## 1	1	1	58.80548	1	1	1	1	1	1.0	14.5	261
## 2	0	1	56.48493	1	0	1	1	1	0.0	1.1	302
## 3	1	1	70.12055	0	0	0	0	0	0.5	1.4	176
## 4	1	1	54.77808	1	0	1	1	0.5	1.8	244	
## 5	0	2	38.13151	1	0	1	1	0.0	3.4	279	
## 6	1	2	66.30411	1	0	1	0	0.0	0.8	248	

```

##   albumin copper    alk   sgot trig platelet prothrombin stage     Years
## 1    2.60    156 1718.0 137.95  172      190       12.2     4 1.095890
## 2    4.14     54 7394.8 113.52   88      221       10.6     3 12.328767
## 3    3.48    210  516.0  96.10   55      151       12.0     4  2.77260
## 4    2.54     64 6121.8  60.63   92      183       10.3     4  5.273973
## 5    3.53    143  671.0 113.15   72      136       10.9     3  4.120548
## 6    3.98     50  944.0  93.00   63       NA      11.0     3  6.857534

```

## *Reshaping Variable*

Let's talk briefly about reshaping data. Data Reshaping in R is something like arranged rows and columns in our own way to use it as per our requirements. We take most of the data we use as a data frame format in R to do data processing using functions like `rbind()`, `cbind()`, etc. (Smith, 2020) In this process, we reshape or re-organize the data into rows and columns. Reshaped data is reorganized data transformed a particular way which we need the data for further processing. Now we'll reshape continuous variables data for exploratory analysis, using the `melt()` function.

If you have used R before, you may recall a package called `reshape2`. Now, `tidyverse` supersedes `reshape2` (2010-2014) and `reshape` (2005-2010). `tidyverse` is designed specifically for tidying data, not general reshaping (`reshape2`), or the general aggregation (`reshape`). The `data.table` package provides high-performance implementations of `melt()` and `dcast()`. However, we used `reshape2` here and it seems to still work.

```

dtb1 <- melt(pbc2, id.vars=c("age","Years","status"))
dtb2 <- melt(pbc2, id.vars=c("bili","Years","status"))
dtb3 <- melt(pbc2, id.vars=c("albumin","Years","status"))
dtb4 <- melt(pbc2, id.vars=c("alk","Years","status"))
dtb5 <- melt(pbc2, id.vars=c("sgot","Years","status"))
dtb6 <- melt(pbc2, id.vars=c("prothrombin","Years","status"))
dtb7 <- melt(pbc2, id.vars=c("chol","Years","status"))
dtb8 <- melt(pbc2, id.vars=c("copper","Years","status"))
dtb9 <- melt(pbc2, id.vars=c("trig","Years","status"))
dtb10<- melt(pbc2, id.vars=c("platelet","Years","status"))

```

## *Plot continuous variables*

Next, we define 10 ggplots, one for each variable and year, (x,year). These will come in handy later.

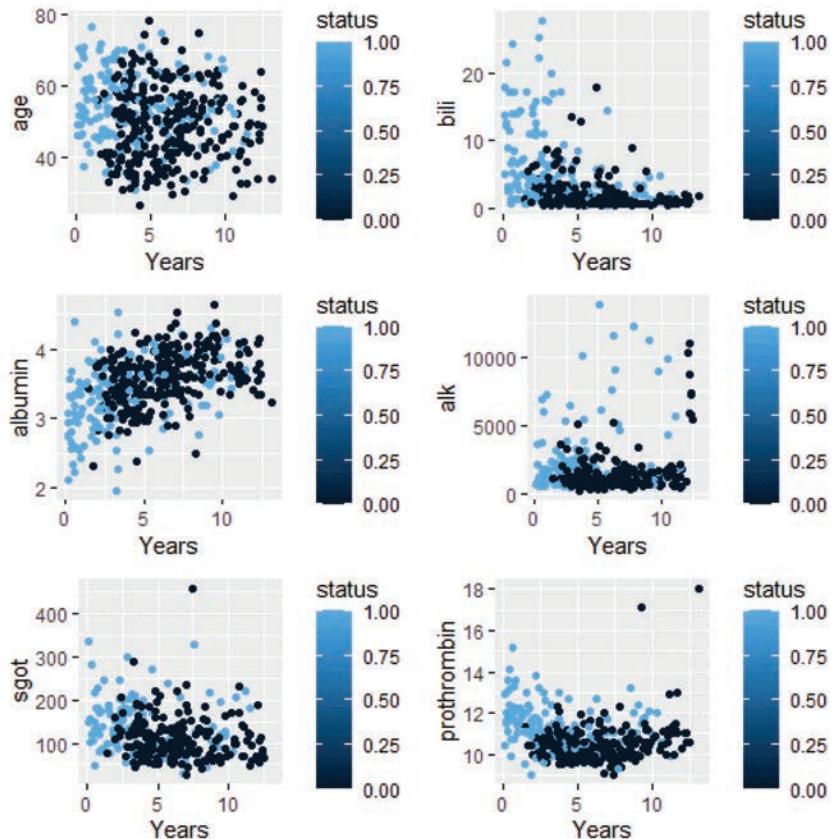
```
gg1 <- ggplot(data=dtb1, aes(x=Years, y=age)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg2 <- ggplot(data=dtb2, aes(x=Years, y=bili)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg3 <- ggplot(data=dtb3, aes (x=Years, y=albumin)) +  
  geom_point(aes (x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg4 <- ggplot(data=dtb4, aes (x=Years, y=alk)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg5 <- ggplot(data=dtb5, aes (x=Years, y=sgot)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg6 <- ggplot(data=dtb6, aes(x=Years, y=prothrombin)) +  
  geom_point(aes (x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg7 <- ggplot(data=dtb7, aes(x=Years, y=chol)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg8 <- ggplot(data=dtb8, aes(x=Years, y=copper)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg9 <- ggplot(data=dtb9, aes(x=Years, y=trig)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")  
gg10<- ggplot(data=dtb10, aes(x=Years, y=platelet)) +  
  geom_point(aes(x=Years, color=status)) +  
  scale_fill_brewer(type= "seq", palette = "Set1")
```

## *Show multiple plots in a window*

Here we demonstrate one way to print more than one plot in one window, simultaneously (there are other ways to do this). So, we will use the ggplots that we just defined shown in **Figure 10-3**.

```
if(require(!gridExtra)) install.packages("gridExtra")
```

```
library(gridExtra)
grid.arrange(gg1,gg2,gg3,gg4,gg5,gg6,nrow=3) ")
```



**Figure 10-3.** ggplots for the first six variables by years with coloring by normalized status

### Create Trial and Test Sets

Now, we create a trial using only randomized patients and a test set using the remaining patients, i.e., those that were not randomized. We will also create a survival object and plot the survival probability function.

```
pbc.trial <- pbc2[-which(is.na(pbc2$treatment)),]
```

```
pbc.test <- pbc2[which(is.na(pbc2$treatment)),]
head(pbc.trial)
```

```

##   status treatment age sex ascites hepatom spiders edema bili chol
## 1      1        1 58.80548  1      1      1      1 1.0 14.5 261
## 2      0        1 56.48493  1      0      1      1 0.0 1.1 302
## 3      1        1 70.12055  0      0      0      0 0.5 1.4 176
## 4      1        1 54.77808  1      0      1      1 0.5 1.8 244
## 5      0        2 38.13151  1      0      1      1 0.0 3.4 279
## 6      1        2 66.30411  1      0      1      0 0.0 0.8 248
##   albumin copper alk  sgot trig platelet prothrombin stage Years
## 1    2.60 156 1718.0 137.95 172     190     12.2    4 1.095890
## 2    4.14  54 7394.8 113.52  88     221     10.6    3 12.328767
## 3    3.48 210 516.0  96.10   55     151     12.0    4 2.772603
## 4    2.54  64 6121.8  60.63   92     183     10.3    4 5.273973
## 5    3.53 143 671.0 113.15   72     136     10.9    3 4.120548
## 6    3.98  50 944.0  93.00   63      NA     11.0    3 6.857534

```

### *Create the Survival Probability Function*

Here, we use gg\_survival to create an object representing the probability of surviving PBC.

```
gg_dta <- gg_survival(interval = "Years",
                      censor = "status",
                      by = "treatment",
                      data = pbc.trial,
                      conf.int = .95 )
```

### *Plot the survival Probability Function*

We now plot the probability function we just created. This will reflect both the treatment and the placebo (or treatment 2) and will show a range of values that increase over time. That is, we will see there is greater uncertainty regarding a patient's survival as the observation time increase. We show this in **Figure 10-4**.

```
plot(gg_dta) +
  labs(y = "Survival Probability",
       x = "Observation Time (Years)",
       color = "Treatment", fill = "Treatment") +
  theme(legend.position = c(.2,.2)) +
  coord_cartesian(y = c(0,1.01))
```

## Markdown Note.

Markdown provides an authoring framework for data science. You can use a single R Markdown file to both

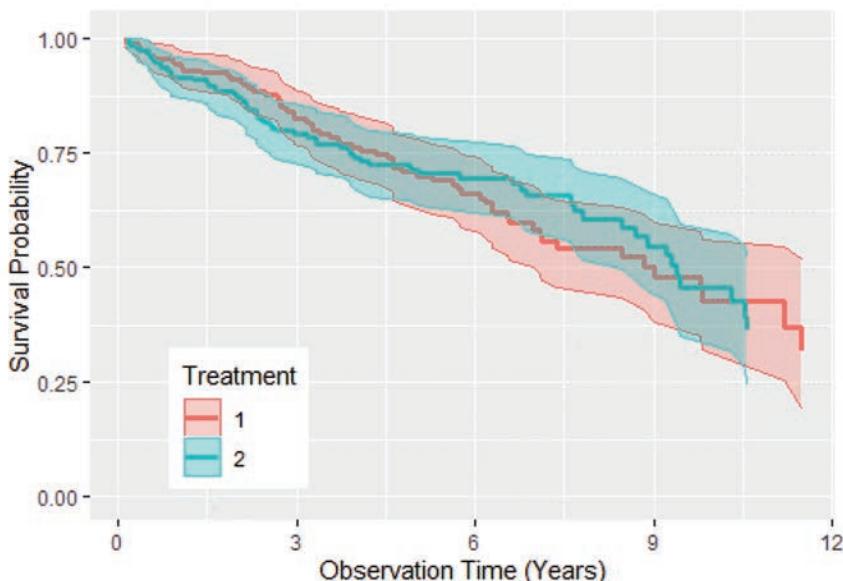
- save and execute code
- generate high quality reports; share with an audience

R Markdown documents are fully reproducible and support dozens of static and dynamic output formats, including Word, HTML, PowerPoint, and more. Like the rest of R, R Markdown is free and open source. You can install the R Markdown package from CRAN with:

```
install.packages("rmarkdown")
```

## *Plot the cumulative hazard function*

Now, we plot the cumulative hazard function for the treatment and placebo in **Figure 10-5**.



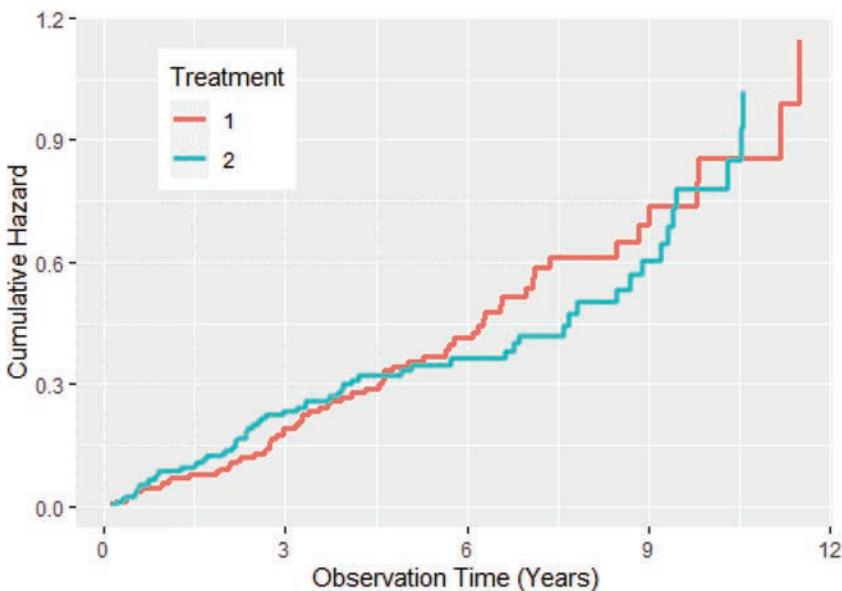
**Figure 10-4.** Probability of patient survival with treatment and placebo (treatment 2)

```
plot(gg_dta, type="cum_haz") +  
  labs(y = "Cumulative Hazard",  
        x = "Observation Time (Years)",  
        color = "Treatment", fill = "Treatment") +  
  theme(legend.position = c(.2,.8))
```

### Markdown Note.

This is a paragraph in an R Markdown document. Below is a code chunk that produces the output that follows:

```
```{r echo=TRUE}  
plot(gg_dta, type="cum_haz") +  
  labs(y = "Cumulative Hazard",  
        x = "Observation Time (Years)",  
        color = "Treatment", fill = "Treatment") +  
  theme(legend.position = c(.2,.8))  
```
```



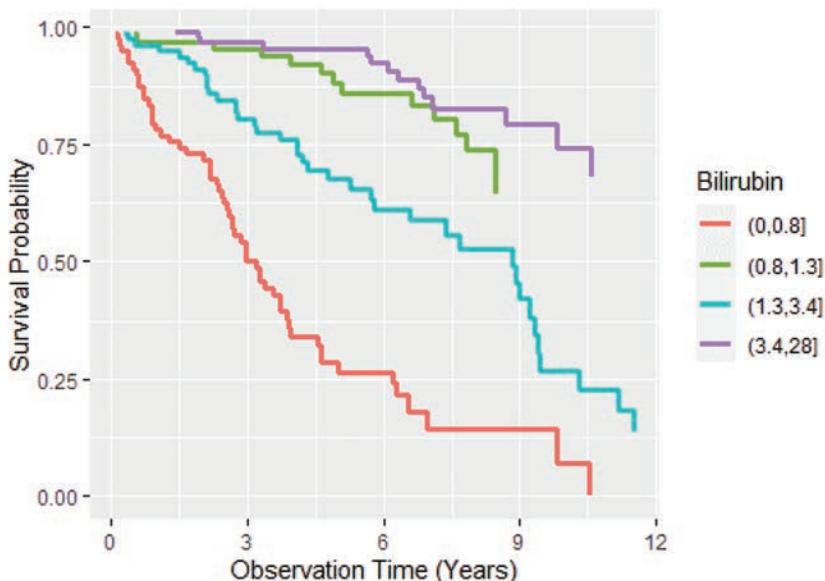
**Figure 10-5.** Plot of the cumulative hazard function for the treatment and placebo

Now, we want to duplicate the trial data and group it by bilirubin values. *Bilirubin* forms due to the natural breakdown of hemoglobin in red blood cells, so too much bilirubin in the urine or bloodstream is not a good thing. It indicates problems with your red blood cells or with your liver.

```
pbcbili <- pbctrial  
pbcbili$bili_grp <- cut(pbctrial$bili,  
                           breaks = c(0, .8, 1.3, 3.4,  
                           max(pbctrial$bili)))
```

When now plot the survival probability for four different values of bilirubin in **Figure 10-6**. The normal adult range is 0.8 to 1.3 mg/dL. If you are in the high range, you can develop jaundice.

```
plot(gg_survival(interval = "Years", censor = "status",  
                  by = "bili_grp", data = pbcbili),  
      error = "none") +  
  labs(y = "Survival Probability",  
       x = "Observation Time (Years)",  
       color = "Bilirubin")
```



**Figure 10-6.** Plot of the survival probability for four different values of bilirubin.

```
#if(!require(shape2)) install.packages(shape2)
library(reshape2)

dta <- melt(pbc2, id.vars=c("bili","Years"))
dtb <- melt(pbc2, id.vars=c("Years","status"))
head(dtb)

##      Years status variable value
## 1  1.095890     1 treatment    1
## 2 12.328767     0 treatment    1
## 3  2.772603     1 treatment    1
## 4  5.273973     1 treatment    1
## 5  4.120548     0 treatment    2
## 6  6.857534     1 treatment    2
```

**Markdown Note.** In ````{r echo=TRUE}`

If `echo = FALSE`, knitr will not display the code in the code chunk above its results in the final document.

### *Using shiny GUI for colorspace*

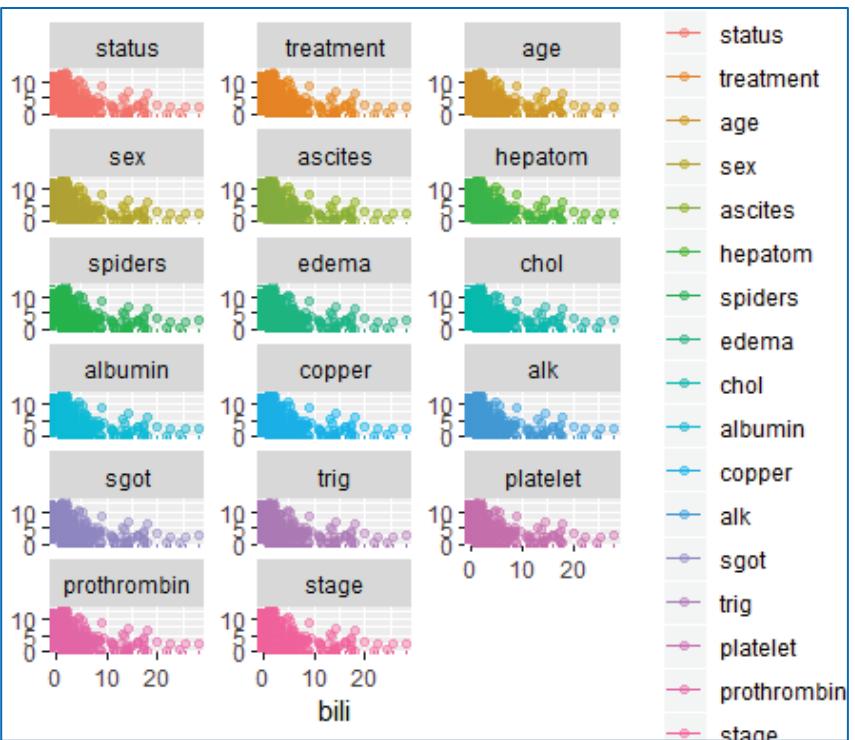
```
choose_palette("tcltk")
```

*Analog to: `choose_palette(gui = "shiny")`*

We'll talk more about building Shiny apps and their GUI. For now, Shiny carries out mapping between assorted color spaces including RGB, HSV, HLS, CIEXYZ, CIELUV, HCL (polar CIELUV), CIELAB and polar CIELAB. Qualitative, sequential, and diverging color palettes based on HCL colors are provided along with corresponding `ggplot2` color scales.

```
ggplot(dta, aes(x=bili, y=Years, color=variable)) + geom_point(alpha=.4) + geom_rug(data=dta) +
  labs(y="", x= "bili") + scale_fill_gradientn(colours =
colorspace::rainbow_hcl(17)) +
  facet_wrap(~variable, scales= "free_y", ncol=3)
```

The plots in gave us nice colors, but little information. So, let's look using shades of blue.



**Figure 10-7.** Variable scatterplots with different colors using the Shiny color mappings

### Markdown Note.

You can add images to an R Markdown report using markdown syntax as follows:

```
![alt text here](path-to-image-here)
```

However, when you knit the report, R will only be able to find your image if you have placed it in the right place - RELATIVE to your .Rmd file. This is where good file management becomes extremely important.

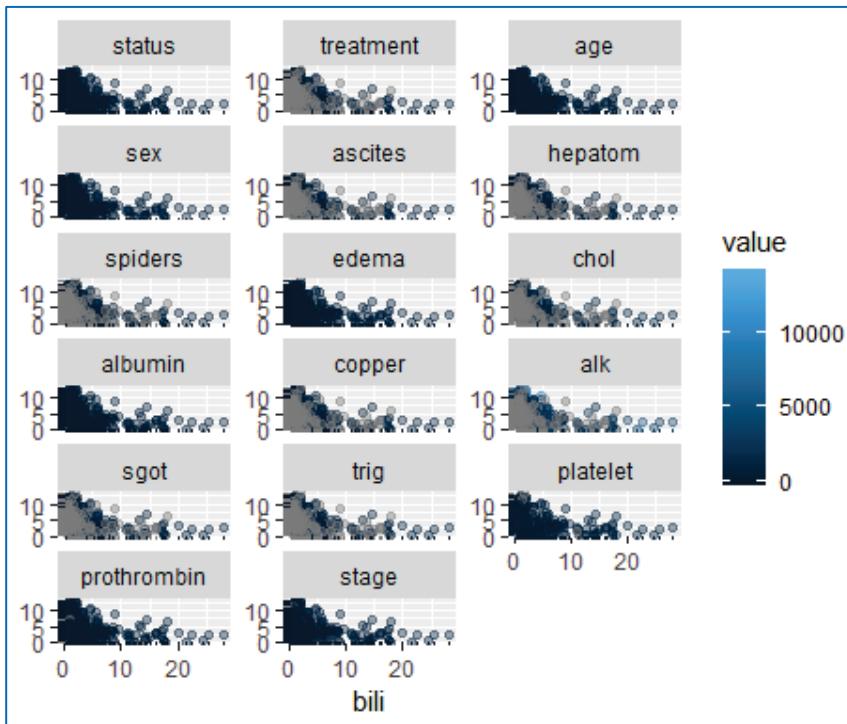
Now we repeat the plots with shade of blue as seen in **Figure 10-8**.

```
ggplot(dta, aes(x=bili, y=Years, color=value)) + geom_point(alpha=.4) + geom_rug(data=dta) +
```

```

  labs(y="", x="bili") + scale_fill_brewer (type = "seq",
  palette = "Set2") +
  facet_wrap(~variable, scales = "free_y", ncol = 3)

```



**Figure 10-8.** Same plots shown in **Figure 10-7** with shades of blue.

### ***Grow and Store the Random Survival Forest***

Bagging (Bootstrap Aggregating) generates  $m$  new training data sets. Each new training data set picks a sample of observations with replacement (bootstrap sample) from original data set. By sampling with replacement, some observations may be repeated in each new training data set. The  $m$  models are fitted using the above  $m$  bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

Out-of-Bag error (miscalculation rate) is equivalent to validation or test data. In random forests, there is no need for a separate test set to validate result. It is estimated internally, during the run, as follows:

As the forest is built on training data , each tree is tested on the 1/3rd of the samples (36.8%) not used in building that tree (similar to validation data set). This is the out of bag error estimate - an internal error estimate of a random forest as it is being constructed.

Fast Unified Random Forests for Survival, Regression, and Classification (RF-SRC) uses fast OpenMP parallel computing of random forests for regression, classification, survival analysis, competing risks, multivariate, unsupervised, quantile regression, and class imbalanced q-classification. Different splitting rules invoked under deterministic or random splitting are available for all families. Different types of variable importance (VIMP), holdout VIMP, as well as confidence regions can be calculated for single and grouped variables. Minimal depth variable selection. Fast interface for missing data imputation using a variety of different random forest methods.

**1. Types of forests.** There is no need to set the type of forest as the package automatically determines the underlying random forest requested from the type of outcome and the formula supplied. There are several possible scenarios:

- a. Regression forests for continuous outcomes.
- b. Classification forests for factor outcomes.
- c. Multivariate forests for continuous and/or factor outcomes and for mixed (both type) of outcomes.
- d. Unsupervised forests when there is no outcome.
- e. Survival forests for right-censored survival.
- f. Competing risk survival forests for competing risk.

**2. Splitting**

- a. Splitting rules are specified by the option split rule.
- b. For all families, pure random splitting can be invoked by setting `splitlevel = "random"`.
- c. For all families, computational speed can be increased using randomized splitting invoked by the option `nsplit`. See Improving Computational Speed.

```
rfsrc_pbc <- rfsrc(Surv(Years, status) ~ .,  
                      data = pbc.trial)
```

Use random splitting (`nsplit = 10`) and impute missing values  
(`na.action = "na.impute"`)

```
rfsrc_pbc2 <- rfsrc(Surv(Years, status) ~ .,  
                      data = pbc.trial,  
                      nsplit = 10,  
                      na.action = "na.impute")
```

### Predict Patient Survival

Now, we predict the survival for 106 patients not in randomized trial:

```
pbc.test$status<-ifelse(pbc.test$status == "T",1,0)  
head(pbc.test)
```

|     | <i>status</i>      | <i>treatment</i> | <i>age</i>  | <i>sex</i>     | <i>ascites</i> | <i>hepatom</i> | <i>spiders</i> |             |                 |
|-----|--------------------|------------------|-------------|----------------|----------------|----------------|----------------|-------------|-----------------|
| 313 | 0                  | NA               | 60.04110    | 1              | NA             | NA             | NA             |             |                 |
| 314 | 0                  | NA               | 65.04384    | 1              | NA             | NA             | NA             |             |                 |
| 315 | 0                  | NA               | 54.03836    | 1              | NA             | NA             | NA             |             |                 |
| 316 | 0                  | NA               | 75.05205    | 1              | NA             | NA             | NA             |             |                 |
| 317 | 0                  | NA               | 62.04384    | 1              | NA             | NA             | NA             |             |                 |
| 318 | 0                  | NA               | 43.03014    | 1              | NA             | NA             | NA             |             |                 |
|     | <i>edema</i>       | <i>bili</i>      | <i>chol</i> | <i>albumin</i> | <i>copper</i>  | <i>alk</i>     | <i>sgot</i>    | <i>trig</i> | <i>platelet</i> |
| 313 | 0.0                | 0.7              | NA          | 3.65           | NA             | NA             | NA             | NA          | 378             |
| 314 | 0.5                | 1.4              | NA          | 3.04           | NA             | NA             | NA             | NA          | 331             |
| 315 | 0.0                | 0.7              | NA          | 4.03           | NA             | NA             | NA             | NA          | 226             |
| 316 | 0.5                | 0.7              | NA          | 3.96           | NA             | NA             | NA             | NA          | NA              |
| 317 | 0.0                | 0.8              | NA          | 2.48           | NA             | NA             | NA             | NA          | 273             |
| 318 | 0.0                | 0.7              | NA          | 3.68           | NA             | NA             | NA             | NA          | 306             |
|     | <i>prothrombin</i> | <i>stage</i>     |             | <i>Years</i>   |                |                |                |             |                 |
| 313 |                    | 11.0             | NA          | 11.128767      |                |                |                |             |                 |
| 314 |                    | 12.1             | 4           | 9.756164       |                |                |                |             |                 |
| 315 |                    | 9.8              | 4           | 7.791781       |                |                |                |             |                 |
| 316 |                    | 11.3             | 4           | 5.673973       |                |                |                |             |                 |
| 317 |                    | 10.0             | NA          | 8.301370       |                |                |                |             |                 |
| 318 |                    | 9.5              | 2           | 4.602740       |                |                |                |             |                 |

```
rfsrc_pbc_test <- predict(rfsrc_pbc,  
                           newdata = pbc.test,  
                           na.action = "na.impute")
```

```
rfsrc_pbc
```

```
##                                     Sample size: 276  
##                                     Number of deaths: 111
```

```
##                               Number of trees: 1000
##           Forest terminal node size: 3
##           Average no. of terminal nodes: 66.925
## No. of variables tried at each split: 5
##           Total no. of variables: 17
##                               Analysis: RSF
##                               Family: surv
##           Splitting rule: logrank
##           Error rate: 17.46%
```

### rfsrc\_pbc2

```
##                               Sample size: 312
##           Number of deaths: 125
##           Was data imputed: yes
##           Number of trees: 1000
##           Forest terminal node size: 3
##           Average no. of terminal nodes: 74.26
## No. of variables tried at each split: 5
##           Total no. of variables: 17
##                               Analysis: RSF
##                               Family: surv
##           Splitting rule: logrank *random*
##           Number of random split points: 10
##           Error rate: 16.49%
```

The `print.rfsrc` function returns information on how the random forest was grown. Here the `family = "surv"` forest has `ntree = 1000` trees (the default `ntree` argument). We used `nsplit = 10` random split points to select random split rule, instead of an optimization on each variable at each split for performance reasons.

### *Print prediction summary*

#### rfsrc\_pbc\_test

```
##   Sample size of test (predict) data: 106
##           Was test data imputed: yes
##           Number of grow trees: 1000
##   Average no. of grow terminal nodes: 66.925
##           Total no. of grow variables: 17
##                               Analysis: RSF
##                               Family: surv
```

Next, we setup the subsequent prediction summary:

```
rfsrc_pbc_test2 <- predict(rfsrc_pbc2,  
                           newdata = pbc.test,  
                           na.action = "na.impute")
```

Then, we print the subsequent prediction summary:

```
rfsrc_pbc_test2  
  
##   Sample size of test (predict) data: 106  
##           Was test data imputed: yes  
##           Number of grow trees: 1000  
##   Average no. of grow terminal nodes: 74.26  
##           Total no. of grow variables: 17  
##                           Analysis: RSF  
##                           Family: surv
```

### *Variable Importance*

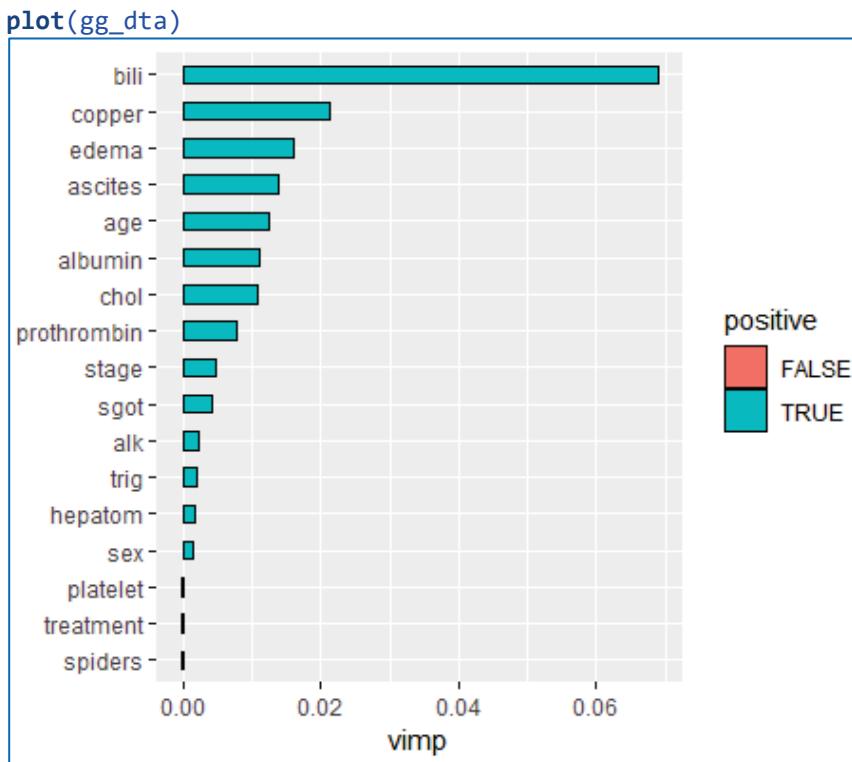
**Variable importance** (VIMP) was originally defined in CART using a measure involving surrogate variables (Breiman, H., Olshen, & Stone, 1984). The most popular VIMP method uses a prediction error approach involving "noising-up" each variable in turn. VIMP for a variable  $x_v$  is the difference between prediction error when  $x_v$  is noised up by randomly permuting its values, compared to prediction error under the observed values (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010).

Since VIMP is the difference in Out-of-Bag prediction error before and after permutation, a large VIMP value indicates that misspecification detracts from the predictive accuracy in the forest. If VIMP is close to zero the variable contributes nothing to predictive accuracy, and negative values indicate the predictive accuracy improves when the variable is mis-specified. In the latter case, we assume noise is more informative than the true variable. As such, we ignore variables with negative and near zero values of VIMP, relying on large positive values to indicate that the predictive power of the forest is dependent on those variables. The `gg_vimp` function extracts VIMP measures for each of the variables used to grow the forest. The `plot.gg_vimp` function in **Figure 10-9** shows the variables, in VIMP rank order in , labeled with the named vector in

the `lbls` argument. Now, we extract VIMP measures for each of the variables used to grow the forest.

```
gg_variable(rfsrc_pbc)
gg_dta<-gg_vimp(rfsrc_pbc)
gg_dta

##           vars   set      vimp positive
## 1       bili VIMP  0.0689719956    TRUE
## 2     copper VIMP  0.0213892597    TRUE
## 3      edema VIMP  0.0161389117    TRUE
## 4    ascites VIMP  0.0138731402    TRUE
## 5        age VIMP  0.0124575963    TRUE
## 6   albumin VIMP  0.0112094109    TRUE
## 7      chol VIMP  0.0109248146    TRUE
## 8 prothrombin VIMP  0.0077009750    TRUE
## 9     stage VIMP  0.0047462763    TRUE
## 10    sgot VIMP  0.0040700859    TRUE
## 11      alk VIMP  0.0021514165    TRUE
## 12     trig VIMP  0.0019009172    TRUE
## 13 hepatom VIMP  0.0016792764    TRUE
## 14      sex VIMP  0.0013872642    TRUE
## 15 platelet VIMP -0.0001474261 FALSE
## 16 treatment VIMP -0.0002531301 FALSE
```



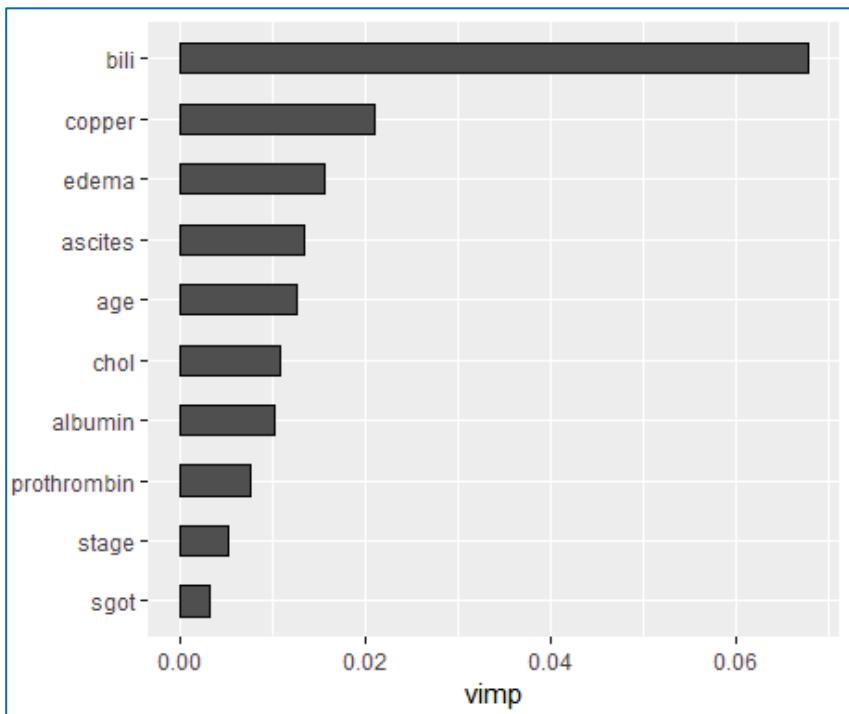
**Figure 10-9.** Variable importance (VIMP) plot

```
gg_dta_10<-gg_vimp(rfsrc_pbc, nvar=10); gg_dta_10
```

Now we plot the VIMP the top ten variables using default settings, as shown in **Figure 10-10**.

```
##          vars    set      vimp positive
## 1        bili VIMP 0.067855241    TRUE
## 2       copper VIMP 0.021075369    TRUE
## 3        edema VIMP 0.015652838    TRUE
## 4       ascites VIMP 0.013474512    TRUE
## 5         age VIMP 0.012696169    TRUE
## 6         chol VIMP 0.010789392    TRUE
## 7     albumin VIMP 0.010124282    TRUE
## 8  prothrombin VIMP 0.007659104    TRUE
## 9       stage VIMP 0.005156435    TRUE
## 10      sgot VIMP 0.003290305    TRUE
```

```
plot(gg_dta_10)
```



**Figure 10-10.** VIMP plot of the top ten variables

```
plot(rfsrc_pbc, lbls = st.labs) +  
  theme(legend.position = c(0.8,0.2)) +  
  labs(fill = "VIMP > 0") +  
  scale_fill_brewer(palette = "Set1")
```

### Minimal Depth

In VIMP, predictive risk factors are determined by testing the forest model prediction under alternative data settings, ranking the most important variables according to their impact on predictive ability of the forest. Another method uses inspection of the forest construction to rank variables. Minimal depth (Ishwaran H. , Kogalur, Chen, & J., 2011) (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010) assumes that variables with high impact on the prediction are those that most frequently split nodes nearest to the root node, where they partition the largest samples of the population.

Within each tree, node levels are numbered based on their relative distance to the root of the tree (with the root at 0). Minimal depth measures important risk factors by averaging the depth of the first split for each variable over all trees within the forest. We assume that in the metric smaller minimal depth values indicate the variable separates large groups of observations, and therefore has a large impact on the forest prediction.

In general, to select variables according to VIMP, we examine the VIMP values, looking for some point along the ranking where there is a large difference in VIMP measures. Given minimal depth is a quantitative property of the forest construction, Ishwaran et al. (2010) also derive an analytic threshold for evidence of variable impact. A simple optimistic threshold rule uses the mean of the minimal depth distribution, classifying variables with minimal depth lower than this threshold as important in forest prediction.

The `randomForestSRC var.select` function uses the minimal depth methodology for variable selection, returning an object with both minimal `depth` and `vimp` measures. The `ggRandomForests gg_minimal_depth` function is analogous to the `gg_vimp` function. Variables are ranked from most important at the top (minimal depth measure), to least at the bottom (maximal minimal depth) as shown in **Figure 10-11**.

### *Return an object with both minimal depth and vimp measures*

```
varsel_pbc <- var.select(rfsrc_pbc)

## minimal depth variable selection ...
## -----
## family          : surv
## var. selection  : Minimal Depth
## conservativeness : medium
## x-weighting used? : TRUE
## dimension      : 17
## sample size     : 276
## ntree           : 1000
## mtry             : 5
## nodesize         : 3
## refitted forest : FALSE
```

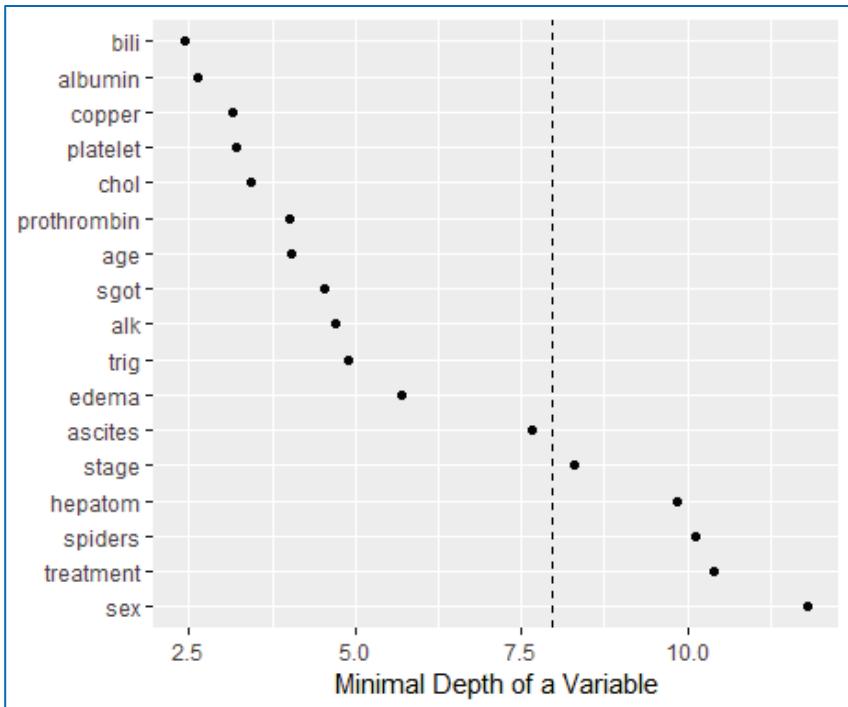
```

## model size      : 12
## depth threshold : 7.9681
## PE (true OOB)   : 17.4626
## Top variables:
##                  depth vimp
## bili            2.431  NA
## albumin         2.617  NA
## copper          3.139  NA
## platelet        3.193  NA
## chol             3.434  NA
## prothrombin    3.995  NA
## age              4.029  NA
## sgot             4.531  NA
## alk              4.708  NA
## trig             4.893  NA
## edema            5.706  NA
## ascites          7.658  NA
ggMindepth <- gg_minimal_depth(varsel_pbc, lbls = Years)
print(ggMindepth)

## gg_minimal_depth
## model size      : 12
## depth threshold : 7.9681
## PE :[1] 17.463
## Top variables:
##                  depth vimp
## bili            2.43  NA
## albumin         2.62  NA
## copper          3.14  NA
## platelet        3.19  NA
## chol             3.43  NA
## prothrombin    4.00  NA
## age              4.03  NA
## sgot             4.53  NA
## alk              4.71  NA

plot(ggMindepth)

```



**Figure 10-11.** Variables from minimal depth (top) to maximal minimal depth (bottom)

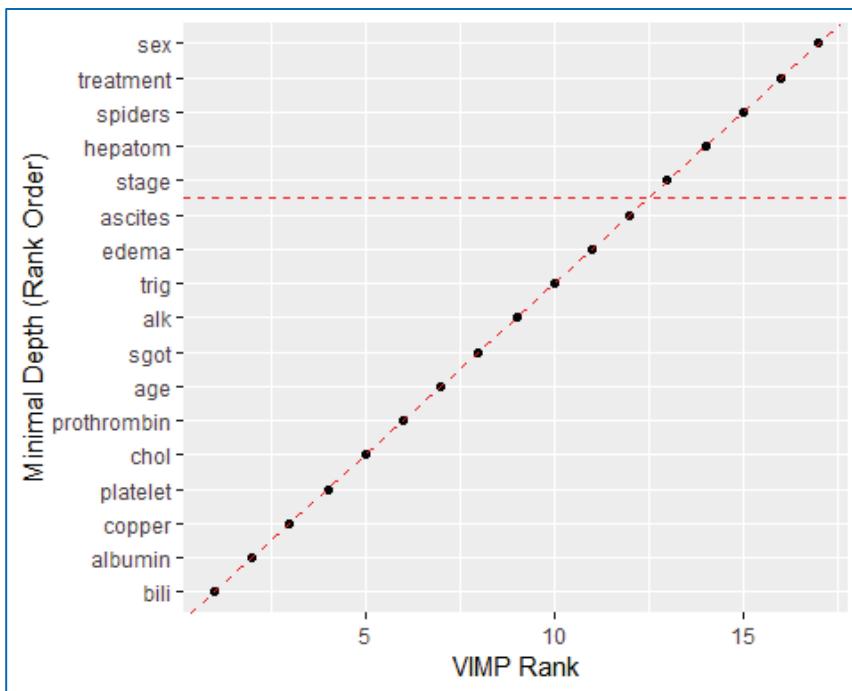
### Both minimal depth and VIMP

VIMP and Minimal Depth measures use different criteria, so we expect the variable ranking to be slightly different. We use `gg_minimal_vimp()` function to compare rankings between minimal depth and VIMP. In this call, we plot the stored `gg_minimal_depth` object (`gg_md`) in **Figure 10-12**, which would be equivalent to calling `plot.gg_minimal_vimp(ggMindepth)` or `plot(gg_minimal_vimp(ggMindepth))`.

### Get the minimal depth selected variables

```
xvar <- varsel_pbc$topvars
xvar

## [1] "bili"    "albumin"   "copper"    "platelet"   "chol"
## [6] "prothrombin" "age"       "sgot"      "alk"        "trig"
## [11] "edema"     "ascites"
```



**Figure 10-12.** Variables with minimal depth and VIMP rank

The points along the red dashed line indicate where the measures agree. Points above the red dashed line are ranked higher by VIMP than by minimal depth, indicating the variables are more sensitive to misspecification. Those below the line have a higher minimal depth ranking, indicating they are better at dividing large portions of the population. The further the points are from the line, the more the discrepancy between measures. **Table 10-1** below compares them.

**Table 10-1.** Comparison of variable minimum depth and VIMP

| Variable | Min depth | VIMP |
|----------|-----------|------|
| age      | 1         | 1    |
| albumin  | 2         | 5    |
| alk      | 3         | 2    |
| bili     | 4         | 4    |
| chol     | 5         | 10   |
| copper   | 6         | 3    |

| Variable    | Min depth | VIMP |
|-------------|-----------|------|
| edema       | 7         | 8    |
| platelet    | 8         | 12   |
| prothrombin | 9         | 9    |
| sgot        | 10        | 13   |
| stage       | 11        | 16   |
| trig        | 12        | 7    |

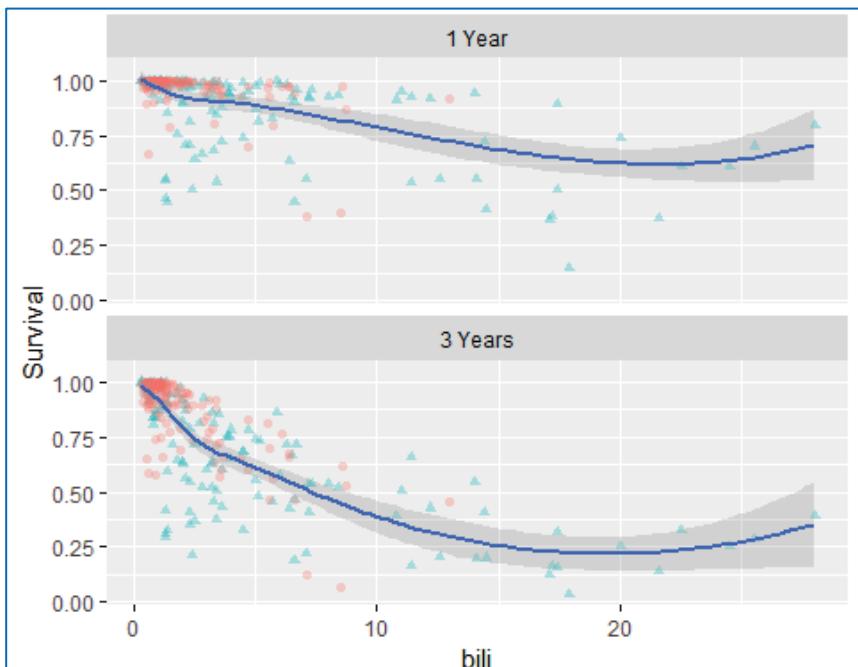
## Data generation

```
ggrf <- gg_variable(rfsrc_pbc, time = c(1, 3),  
                     time.labels = c("1 Year", "3 Years"))
```

## Bilirubin variable dependence plot

Here we generate variable dependence plots of survival at 1 and 3 years on bilirubin variable in **Figure 10-13**. Individual cases are marked with red circles (alive or censored) and green triangles (dead). Loess smooth curve with shaded 95% confidence band indicates decreasing survival with increasing bilirubin.

```
plot(ggrf, xvar = "bili", se = .95, alpha = .3) +  
  labs(y = "Survival", x = "bili") +  
  theme(legend.position = "none")
```



**Figure 10-13.** Bilirubin dependence plots at 1 and 3 years

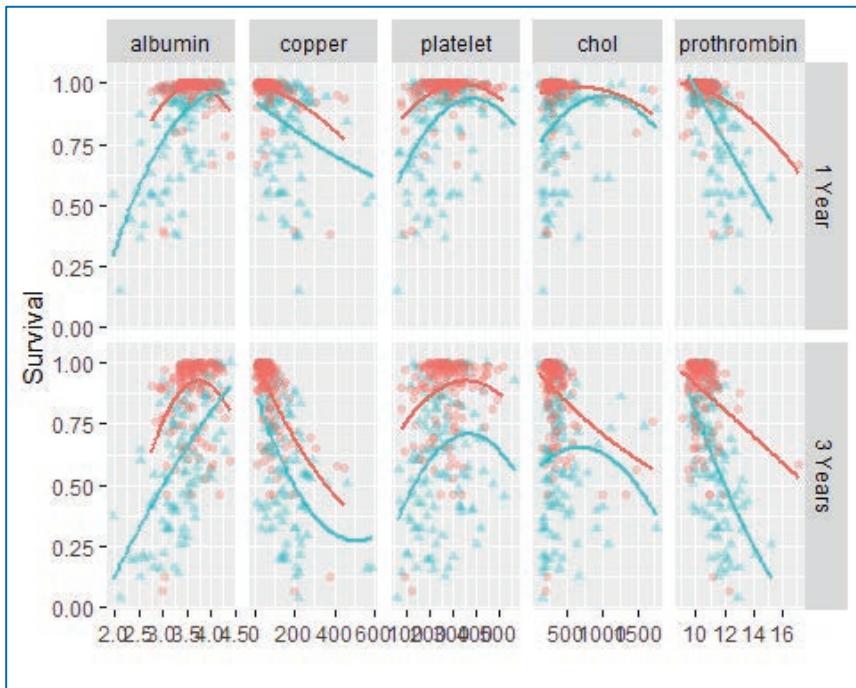
## Pull the categorical variables

```
xvar.cat <- c("edema", "stage")  
xvar <- xvar[-which(xvar %in% xvar.cat)]
```

## *Continuous Variable Dependence Plots.*

We now generate variable dependence plots of predicted survival at 1 and 3 years on continuous variables of interest in **Figure 10-14**. Individual cases are marked with red circles for censored cases and green triangles for death events. Loess smooth curve indicates the survival trend with increasing values.

```
plot(ggrf, xvar = xvar[2:6], panel = TRUE,  
     se = FALSE, alpha = .3,  
     method = "glm", formula = y~poly(x,2)) +  
     labs(y = "Survival") +  
     theme(legend.position = "none") #optional
```

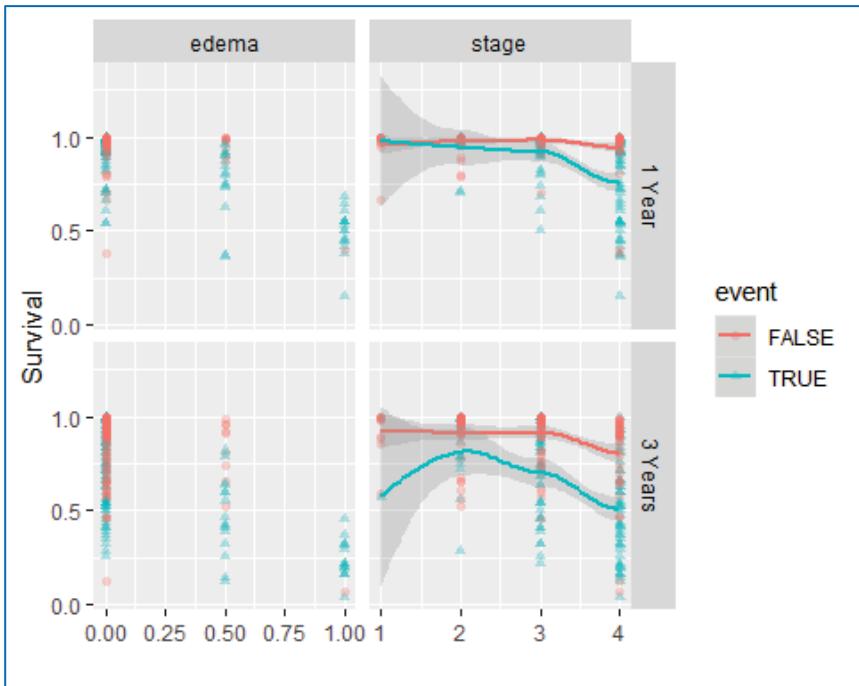


**Figure 10-14.** Variable dependence plots of predicted survival at 1 and 3 years on continuous variables of interest

Variable dependence plots for categorical variables are constructed using boxplots to show the distribution of the predictions within each category. Here we demonstrate variable dependence of survival 1 and 3 years on the edema categorical variable in **Figure 10-15**.

Symbols with red circles indicate censored cases and green triangles indicate death events. Boxplots indicate distribution of predicted survival for all observations within each edema group.

```
plot(ggrf, xvar = xvar.cat, panel = TRUE, notch = TRUE, alpha = .3) +
  labs(y = "Survival") + scale_fill_gradientn(colours = colorspace::rainbow_hcl(17))
```



**Figure 10-15.** Variable dependence of survival 1 and 3 years on the edema categorical variable

### Partial Dependence Plot

The partial dependence plot (PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a machine learning model (J. H. Friedman 2001). A partial dependence function is the partial distribution of a multi-variable distribution function and its plot can show whether the relationship between the target and a feature is linear, monotonic or more complicated. For

example, when applied to a linear regression model, partial dependence plots always show a linear relationship.

The partial dependence function for multi-variable regression is defined as:

$$\hat{f}_{x_s}(x_s) = E_{x_c}[\hat{f}(x_s, x_c)] = \int \hat{f}(x_s, x_c) d\mathbb{P}(x_c)$$

The  $x_s$  are the features for which the partial dependence function should be plotted and  $x_c$  are the other features used in the machine learning model  $\hat{f}$ . Usually, there are only one or two features in the set  $S$ . The feature(s) in  $S$  are those for which we want to know the effect on the prediction. The feature vectors  $x_s$  and  $x_c$  combined make up the total feature space  $x$ . Partial dependence works by marginalizing the machine learning model output over the distribution of the features in set  $C$ , so that the function shows the relationship between the features in set  $S$  we are interested in and the predicted outcome. By marginalizing over the other features, we get a function that depends only on features in  $S$ , interactions with other features included.

The partial function  $\hat{f}_{x_s}$  is estimated by calculating averages in the training data, also known as Monte Carlo method:

$$\hat{f}_{x_s}(x_s) = \frac{1}{n} \sum_{i=1}^n \hat{f}(x_s, x_c^{(i)})$$

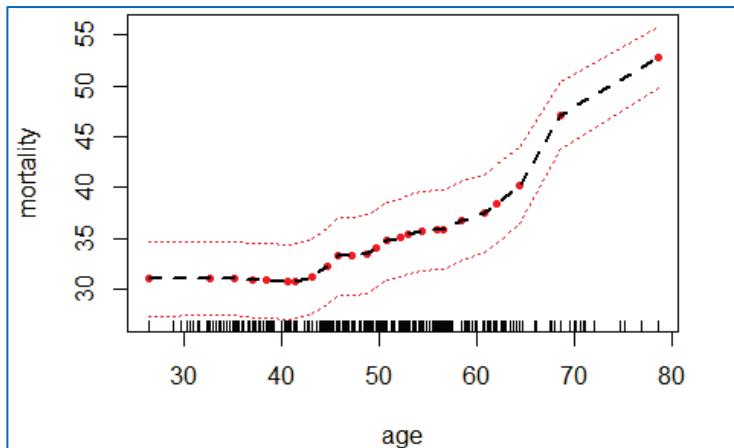
The partial function tells us for given value(s) of features  $S$  what the average marginal effect on the prediction is. In this formula,  $x_c^{(i)}$  are actual feature values from the dataset for the features in which we are not interested, and  $n$  is the number of instances in the dataset. An assumption of the PDP is that the features in  $C$  are not correlated with the features in  $S$ . If this assumption is violated, the averages calculated for the partial dependence plot will include data points that are very unlikely or even impossible.

For classification where the machine learning model outputs probabilities, the partial dependence plot displays the probability for

a certain class given different values for feature(s) in  $S$ . An easy way to deal with multiple classes is to draw one line or plot per class.

The partial dependence plot is a global method: The method considers all instances and gives a statement about the global relationship of a feature with the predicted outcome. We calculate the 1- and 3-year partial dependence plots for age (**Figure 10-16**) and alkaline phosphatase (**Figure 10-17**)

```
partial_age <- plot.variable(rfsrc_pbc, xvar.names="age",  
partial=TRUE)
```



**Figure 10-16.** Partial dependence plot for age

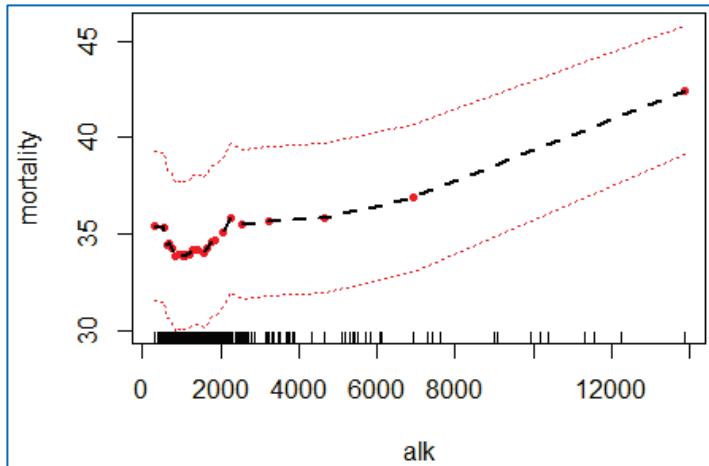
```
partial_alk <- plot.variable(rfsrc_pbc, xvar.names="alk",  
partial=TRUE)
```

Rather than plot all of the PDPs here, we will plot the most interesting, after an explanation of the PDP for categorical variables.

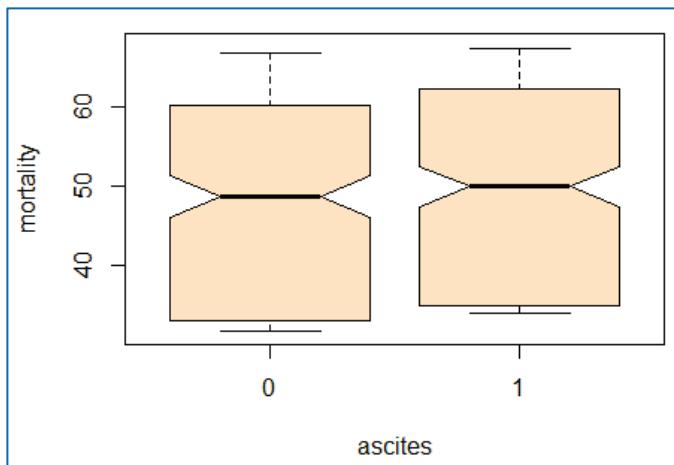
### Categorical Features

So far, we have only thought about numerical features. For categorical features, the partial dependence is very easy to compute. For each of the categories, like “yes” or “no” for the presence of ascites we get a PDP estimate by forcing all data instances to have the same category as in **Figure 10-18**. For another example, if we look at the PBC dataset and are interested in the partial dependence plot for the edema, we get 3 numbers, one for each category. To compute the

value for "no edema and no diuretic therapy for edema", we replace the edema of all data instances with "no edema and no diuretic therapy for edema" and average the predictions. We plot this instance along with for continuous variables in **Figure 10-19** and with 1-year and 3-year in plots in **Figure 10-20**.



**Figure 10-17.** Partial dependence plot for alkaline phosphatase



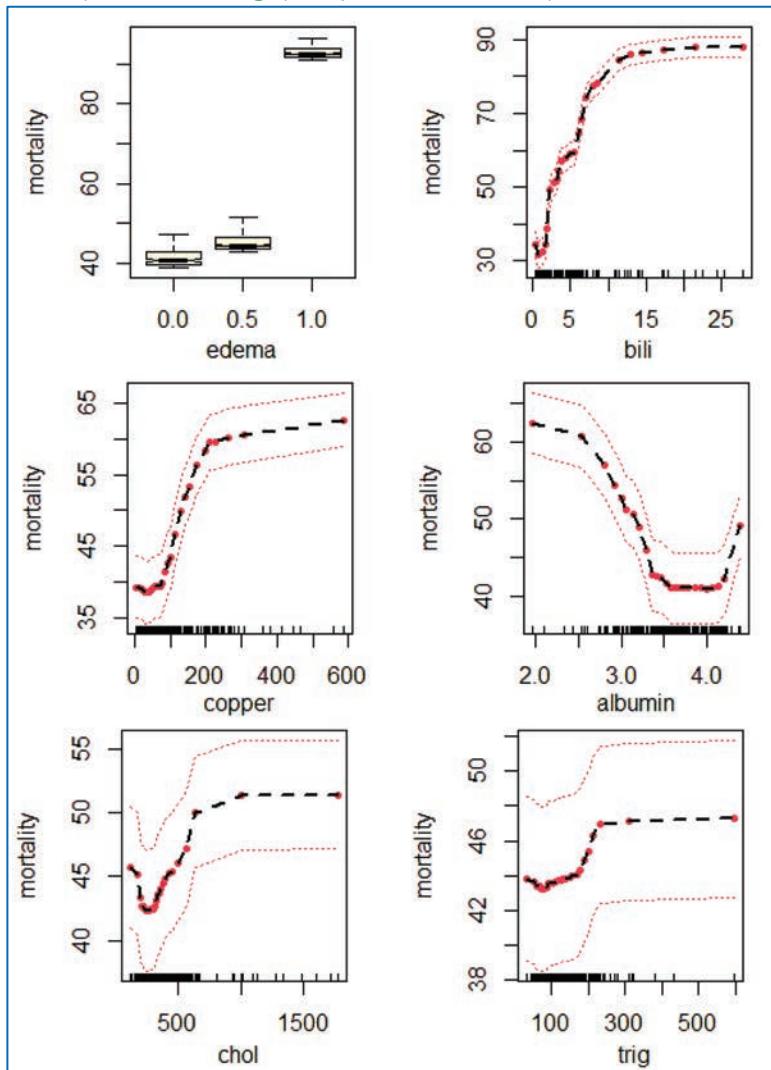
**Figure 10-18.** Partial dependence plot for the categorical variable "presence of ascites"

```
part_var01 <- plot.variable(rfsrc_pbc, xvar.names =
  c("edema", "bili"), partial = TRUE)
```

```

part_var02 <- plot.variable(rfsrc_pbc, xvar.names =
    c("copper", "albumin"), partial = TRUE)
part_var03 <- plot.variable(rfsrc_pbc, xvar.names =
    c("chol", "trig"), partial = TRUE)

```



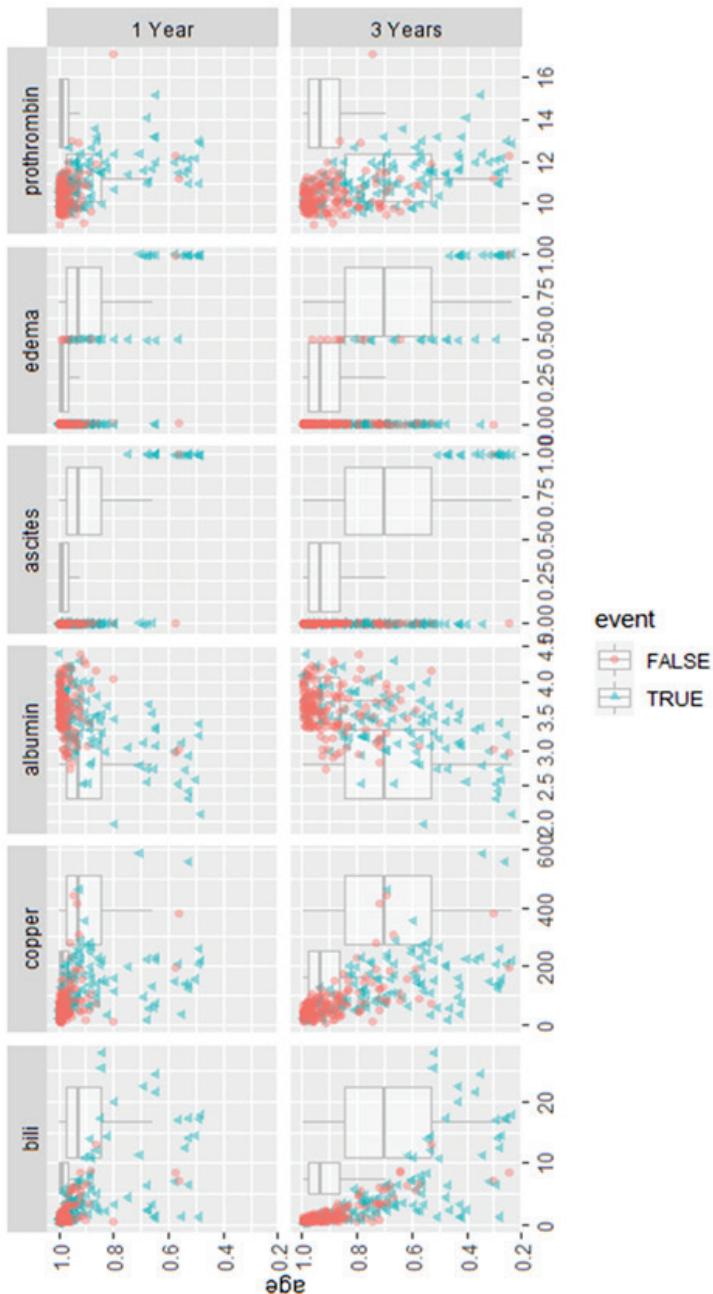
**Figure 10-19.** Partial dependence plots of the categorical variable edema and five continuous variables

```

xvar <- ggMindepth$topvars
plot(ggrf, xvar=xvar, panel=TRUE, alpha=.4) +

```

```
labs(y="age", x="")
```



**Figure 10-20.** Partial plots for 1-year and 3-year edema and four continuous variables

Recall that presence of edema has three values representing categorical flags:

- 0.0 = no edema and no diuretic therapy for edema;
- 0.5 = edema present without diuretics, or edema resolved by diuretics;
- 1.0 = edema despite diuretic therapy

The distributions for these categories is surprisingly similar and we might need to investigate the physiological and pathophysiological aspect.

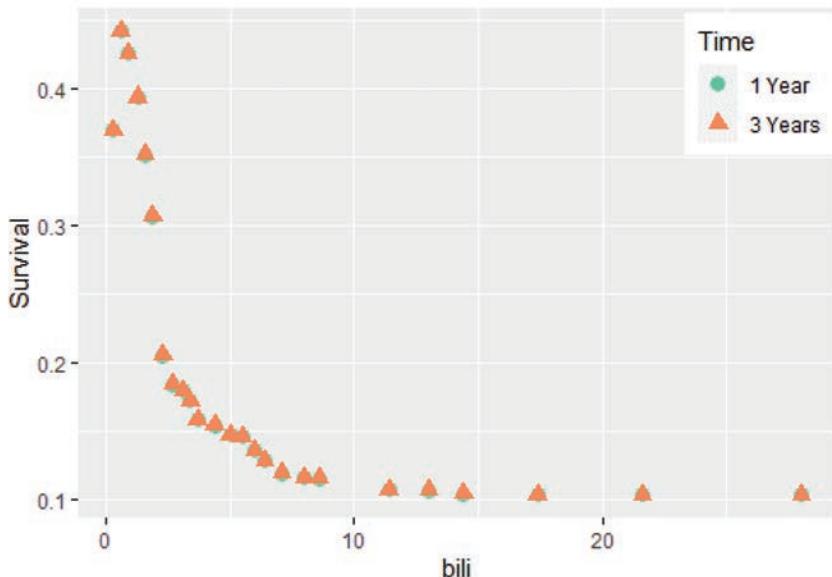
Next, we convert all partial plots to `gg_partial` objects.

```
gg_dta <- lapply(partial_pbc, gg_partial)
```

Now, we combine the objects to get multiple time curves along variables on a single figure as seen in **Figure 10-21**.

```
gg_dta <- lapply(partial_pbc, gg_partial)
pbc_ggpart <- combine.gg_partial(gg_dta[[1]], gg_dta[[2]],
lbls = c("1 Year", "3 Years"))
#pbc_ggpart2 <- combine.gg_partial(ggRandomForests::gg_partial(gg_dta[[1]], gg_dta[[2]]), lbls = c("1 Year", "3 Years"))
plot(pbc_ggpart[["bili"]]) +
  theme(legend.position = c(.9, .85)) +
  labs(y = "Survival",
       x = "bili",
       color = "Time", shape = "Time") +
  scale_color_brewer(palette = "Set2")
```

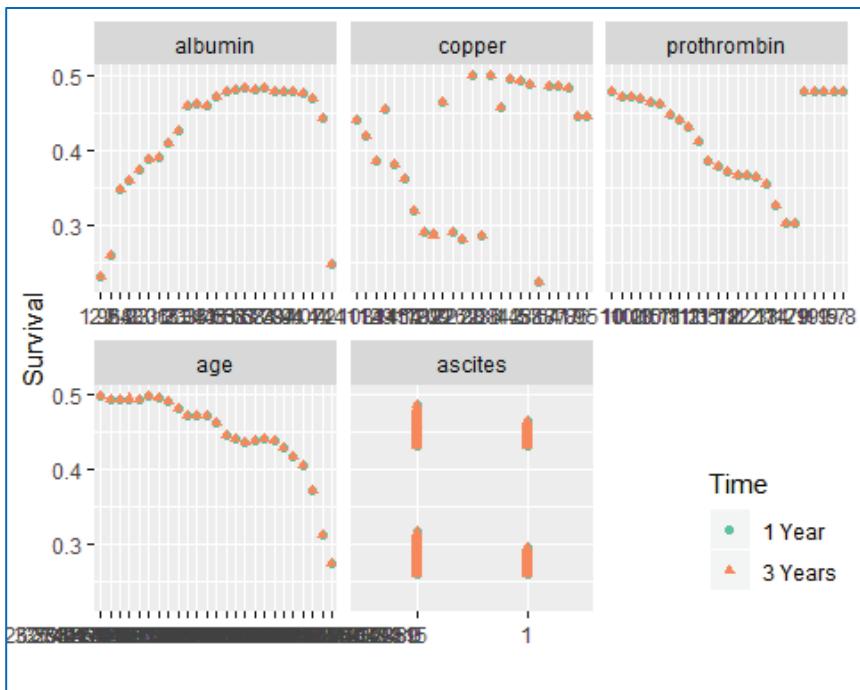
**Markdown Note.** Knitr is an engine for dynamic report generation with R. It is a package in the statistical programming language R that enables integration of R code into LaTeX, LyX, HTML, Markdown, AsciiDoc, and reStructuredText documents.



**Figure 10-21.** Combined objects to get multiple time curves (1 and 3 years) along variables on a single figure.

Let's create a temporary holder and remove the stage and edema data. Then, we plot this situation in **Figure 10-22**.

```
ggpart <- pbc_ggpart
ggpart$edema <- ggpart$stage <- NULL
ggpart$bili <- ggpart$sgot <- pipratecol <- NULL
ggpart$platelet <- ggpart$trig <- ggpart$alk <- NULL
# Panel plot the remainder.
plot(ggpart, panel = TRUE) +
  labs(x = "", y = "Survival", color = "Time", shape = "Ti
me") +
  scale_color_brewer(palette = "Set2") +
  theme(legend.position = c(.9, .15))
```



**Figure 10-22.** Partial dependence plots without stage and edema

Now, we generate the partial dependence plot for stage and edema by survival rates at 1-year and 3-year marks in **Figure 10-23**.

```
ggpart <- pbc_ggpart
head(ggpart$edema)

##          yhat edema   group
## 1    0.4341768     0 1 Year
## 2    0.4317110     0 1 Year
## 3    0.2670310     0 1 Year
## 4    0.4519583     0 1 Year
## 5    0.4299843     0 1 Year
ggpart$stage

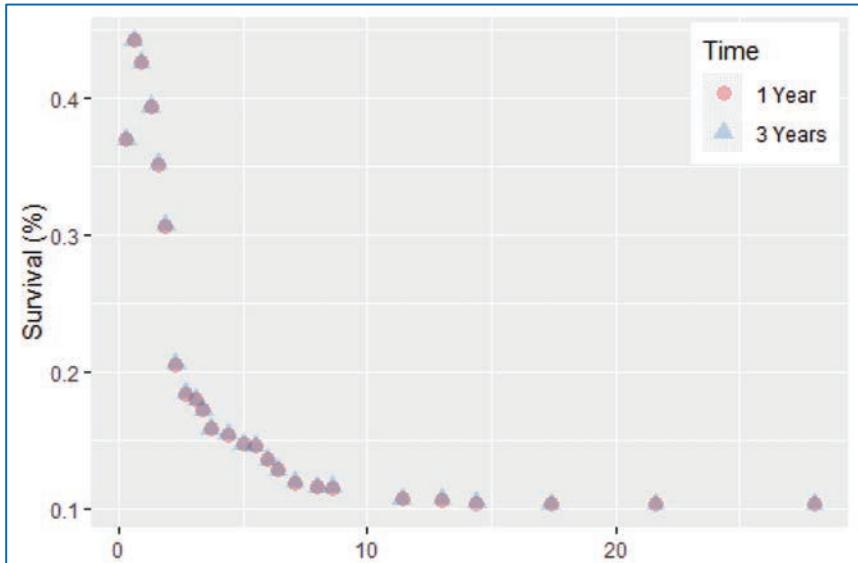
names(ggpart) <- c("edema", "stage")
class(ggpart) <- c("gg_partial_list", class(ggpart))
plot(ggpart$edema, panel=TRUE, notch = TRUE, alpha = .3) +
  labs(x = "", y = "Survival (%)", color = "Time",
```

```

shape= "Time") +
scale_color_brewer(palette = "Set1") +
theme(legend.position = c(.35, .1))

## Warning: Ignoring unknown parameters: panel, notch

```



**Figure 10-23.** Partial dependence plot for stage and edema by survival rates at 1-year and 3-year marks

The `gg_interaction()` function wraps the `find.interaction` matrix for use with the provided S3 plot and print functions.

```

interaction_pbc <- find.interaction(rfsrc_pbc)

##          No. of variables: 17
##  Variables sorted by minimal depth?: TRUE
##      bili albumin copper platelet chol prothrombin age sgot alk
## bili    0.14    0.25    0.28     0.29 0.25      0.33 0.26 0.32 0.30
## albumin 0.29    0.16    0.33     0.36 0.31      0.39 0.33 0.38 0.37
## copper  0.32    0.31    0.18     0.37 0.33      0.39 0.36 0.39 0.38
## platele 0.43    0.43    0.44     0.19 0.44      0.50 0.46 0.49 0.48
## chol    0.36    0.36    0.38     0.41 0.20      0.44 0.39 0.41 0.40
## prothro 0.44    0.44    0.46     0.50 0.46      0.23 0.46 0.49 0.50
## age     0.40    0.40    0.43     0.44 0.40      0.49 0.24 0.47 0.44
## sgot    0.49    0.48    0.51     0.50 0.49      0.55 0.50 0.27 0.51

```

```

## alk    0.48   0.47   0.51    0.53 0.48      0.57 0.53 0.52 0.27
## trig   0.49   0.50   0.52    0.53 0.49      0.58 0.52 0.53 0.53
## edema  0.55   0.55   0.57    0.58 0.56      0.62 0.58 0.62 0.58
## ascites 0.61   0.61   0.62    0.63 0.61      0.66 0.65 0.65 0.64
## stage   0.71   0.73   0.73    0.75 0.72      0.75 0.73 0.74 0.73
## hepatom 0.82   0.83   0.82    0.83 0.81      0.83 0.82 0.83 0.83
## spiders 0.85   0.85   0.85    0.85 0.83      0.85 0.85 0.85 0.85
## treat   0.91   0.91   0.91    0.90 0.91      0.91 0.91 0.90 0.90
## sex     0.89   0.89   0.91    0.90 0.89      0.90 0.90 0.90 0.89
##          trig edema ascites stage hepatom spiders treatment sex
## bili    0.32 0.59   0.78   0.55   0.65   0.66   0.65 0.73
## albumin 0.36 0.67   0.79   0.60   0.69   0.70   0.69 0.75
## copper  0.39 0.67   0.81   0.62   0.70   0.71   0.70 0.77
## platelet 0.49 0.74   0.84   0.68   0.76   0.78   0.76 0.82
## chol    0.42 0.75   0.89   0.63   0.72   0.74   0.73 0.79
## prothrombin 0.50 0.75   0.88   0.70   0.78   0.78   0.76 0.83
## age     0.45 0.79   0.89   0.67   0.74   0.76   0.74 0.83
## sgot    0.54 0.80   0.91   0.72   0.80   0.81   0.77 0.84
## alk     0.53 0.86   0.93   0.74   0.81   0.81   0.79 0.85
## trig    0.29 0.85   0.93   0.74   0.80   0.80   0.79 0.87
## edema   0.60 0.33   0.90   0.75   0.80   0.81   0.81 0.84
## ascites 0.66 0.84   0.44   0.79   0.83   0.84   0.83 0.87
## stage   0.75 0.90   0.97   0.48   0.88   0.90   0.88 0.92
## hepatom 0.84 0.95   0.98   0.90   0.57   0.93   0.93 0.95
## spiders 0.86 0.95   0.98   0.90   0.94   0.59   0.92 0.96
## treatment 0.91 0.98   0.99   0.95   0.96   0.96   0.61 0.97
## sex     0.90 0.97   0.99   0.95   0.96   0.96   0.95 0.68

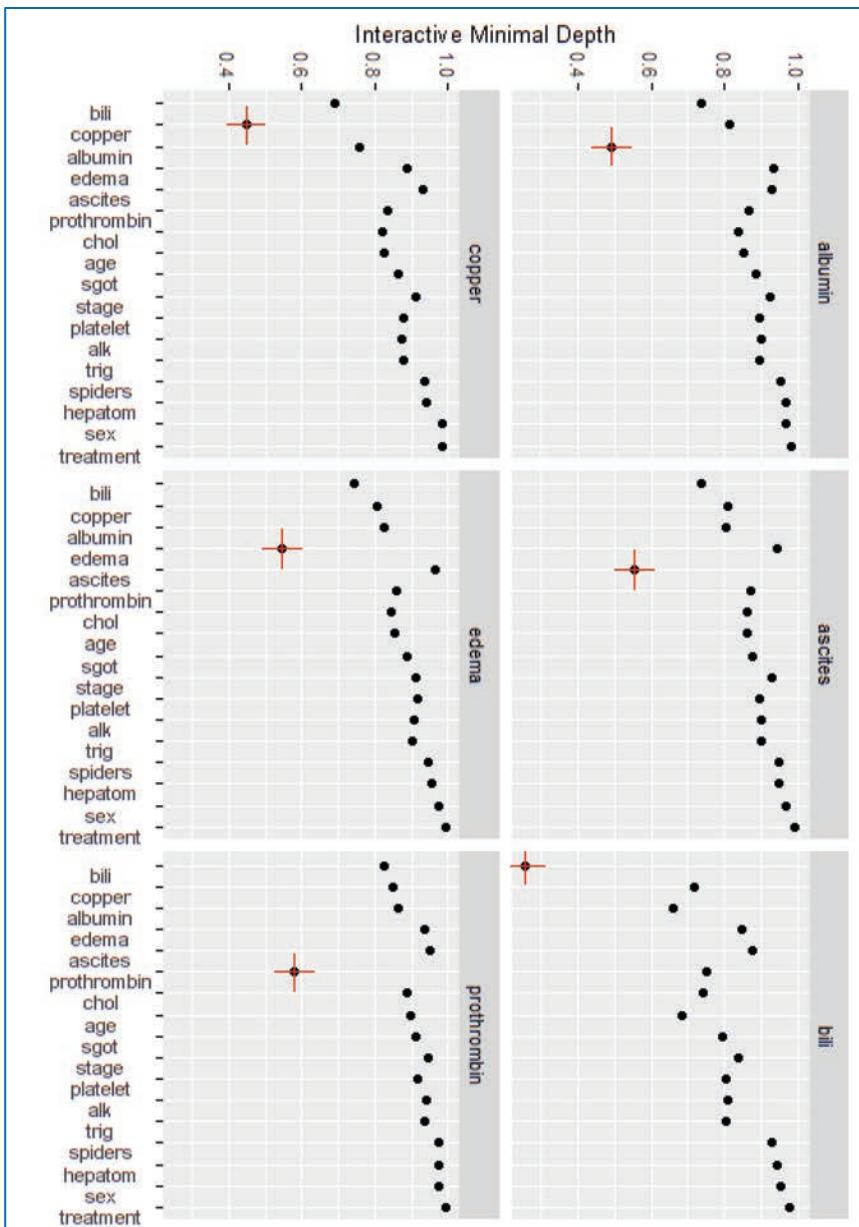
```

Now, we generate the interactive plots for selected variables, albumin, ascites, serum bilirubin, serum cholesterol, copper, edema, and prothrombin, in **Figure 10-24**.

```

ggint <- gg_interaction(interaction_pbc)
plot(ggint, xvar = xvar) +
  labs(y = "Interactive Minimal Depth") +
  theme(legend.position = "none")

```



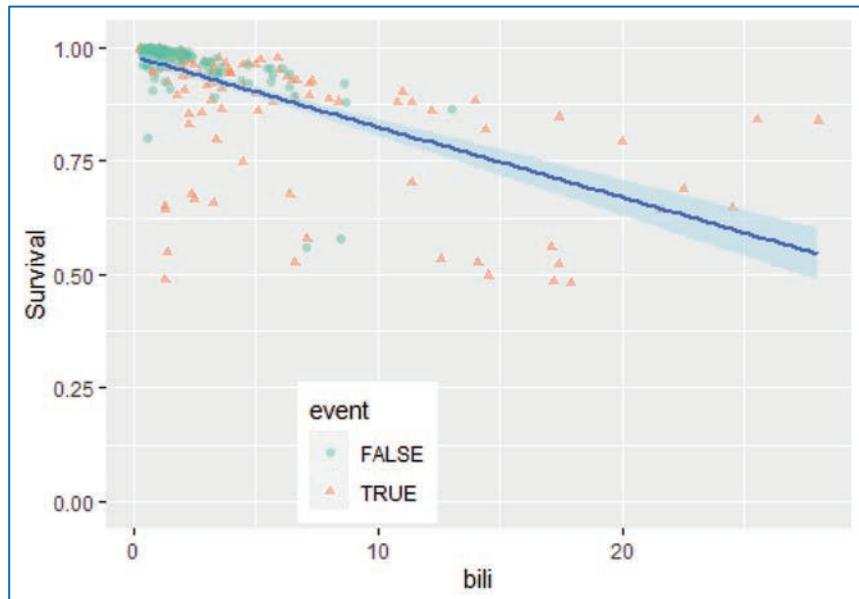
**Figure 10-24.** Selected interactive minimal depth plots

## Conditional Dependence Plots

Now, we generate the conditional dependence plot for serum bilirubin with fitted line in **Figure 10-25**.

```
ggvar <- gg_variable(rfsrc_pbc, time = 1)
ggvar$stage <- paste("stage = ", ggvar$stage, sep = "")
var_dep <- plot(ggvar, xvar = "bili", method = "glm", alpha
a = .5, se = FALSE) +
  labs(y = "Survival", x = "bili") +
  theme(legend.position = c(.35, .1)) +
  scale_color_brewer(palette = "Set2") +
  scale_shape(solid=TRUE) +
  coord_cartesian(y = c(-.01,1.01))
```

var\_dep



**Figure 10-25.** Conditional dependence plot for serum bilirubin with fitted line

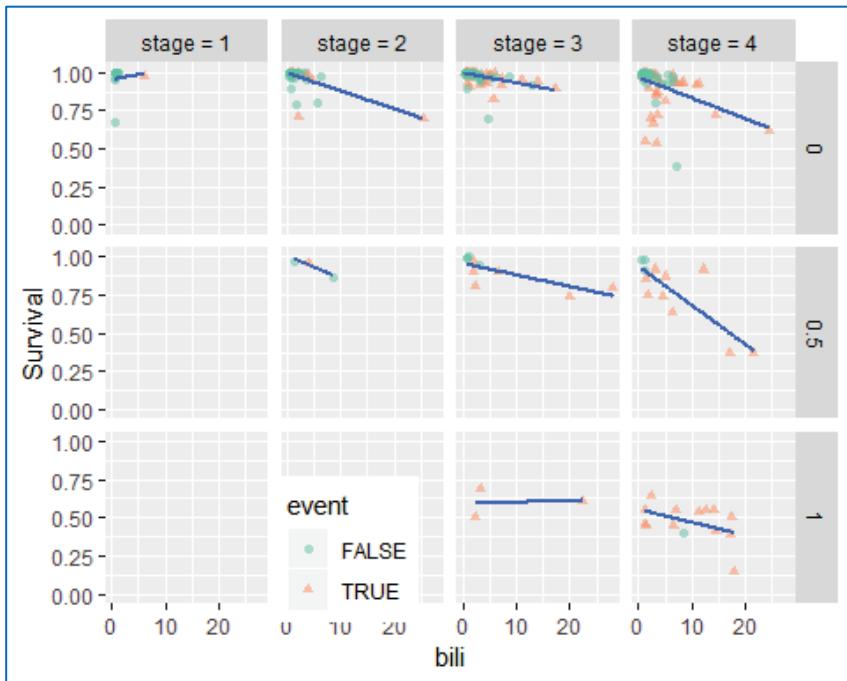
We now show the conditional dependence of survival against bilirubin, versus other categorical covariates, say edema and stage in **Figure 10-26**. We also find intervals with similar number of observations.

```

var_dep +
  facet_grid(edema~stage)

copper_cts <- quantile_pts(ggvar$copper, groups = 6,
                           intervals = TRUE)

```



**Figure 10-26.** Conditional dependence of survival against bilirubin, versus other categorical covariates, say edema and stag

Next, we create the conditional groups and add to the `gg_varible` object

```

copper_grp <- cut(ggvar$copper, breaks = copper_cts)
ggvar$copper_grp <- copper_grp

```

Finally, we adjust naming for facets before plotting.

```

levels(ggvar$copper_grp) <- paste("copper = ",
                                    levels(copper_grp), sep = "")

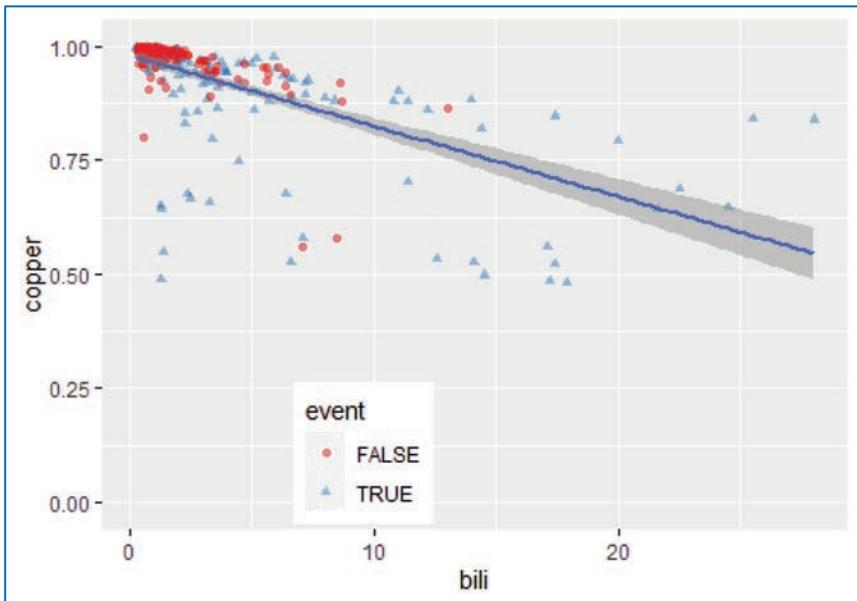
```

So, the variable dependency plot we generate is serum cholesterol versus serum bilirubin in **Figure 10-27**.

```

var_dep <- plot(ggvar, xvar = "bili", method = "glm",
                 alpha = .5, se = FALSE) +
  labs(y = "cholesterol", x = "bili") +
  theme(legend.position = c(.35, .1)) +
  scale_color_brewer(palette = "Set2") +
  scale_shape(solid=TRUE) +
  coord_cartesian(y = c(-.01,1.01))
var_dep

```



**Figure 10-27.** Variable dependency plot we generate is serum cholesterol versus serum bilirubin

### Partial Dependence Coplots

```
data(rfsrc_pbc, package="ggRandomForests")
```

For the partial dependence coplots, we create the variable plot.

```
ggvar <- gg_variable(rfsrc_pbc, time = 1)
```

Then, we find intervals with similar number of observations.

```
copper_cts <- quantile_pts(ggvar$copper, groups = 6,
                           intervals = TRUE)
```

Next, we create the conditional groups and add to the `gg_variable` object.

```
copper_grp <- cut(ggvar$copper, breaks = copper_cts)
partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc,
    xvar = "bili", groups = copper_grp,
    surv_type = "surv", time = 1, show.plots = FALSE)
```

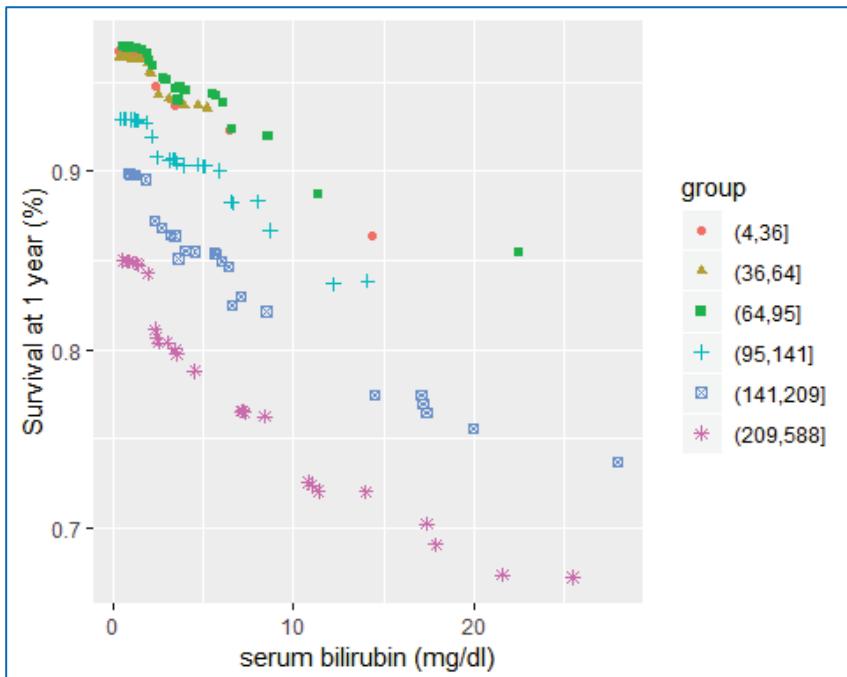
Now, we load the cached set.

```
data(partial_coplot_pbc, package="ggRandomForests")
```

### Partial Coplot

Finally, we display the partial dependence coplots by the conditional groups in **Figure 10-28**.

```
plot(partial_coplot_pbc) +
  labs(x = "serum bilirubin (mg/dl)",
       y = "Survival at 1 year (%)")
```

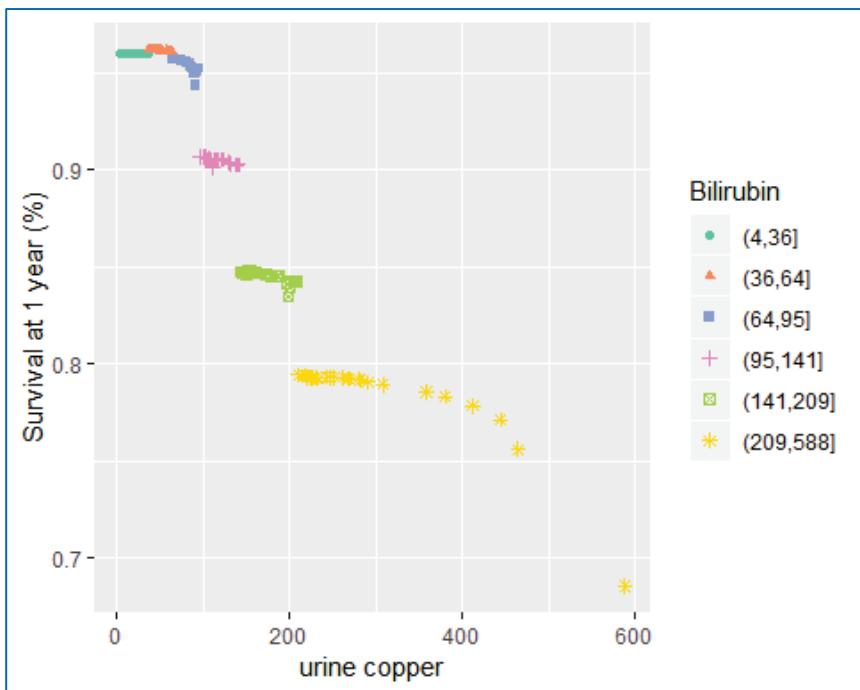


**Figure 10-28.** partial dependence coplots by the conditional groups

Next, we generate the plot of urine copper by serum bilirubin for 1-year survival percentage in **Figure 10-29**.

```
partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc,
  xvar = "copper", groups = copper_grp,
  surv_type = "surv", time = 1, show.plots = FALSE)
data(partial_coplot_pbc, package = "ggRandomForests")

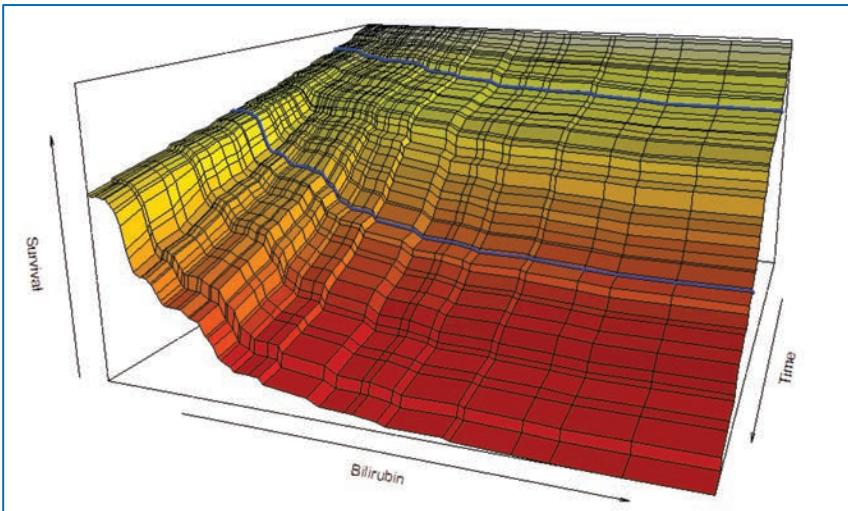
plot(partial_coplot_pbc) +
  labs(x = "urine copper",
       y = "Survival at 1 year (%)",
       color = "Bilirubin", shape = "Bilirubin") +
  scale_color_brewer(palette = "Set2")
```



**Figure 10-29.** Plot of urine copper by serum bilirubin for 1-year survival percentage

**Figure 10-30** below shows partial dependence of survival (Z-axis) as a function of bilirubin over a five year follow up time period. The lines perpendicular to the bilirubin axis are distributed along the bilirubin variable. Lines parallel to the bilirubin axis are taken at 50 training

set event times, the first event after  $t = 0$  at the back to last event before  $t = 5$  years at the front. The code is provided in (Ehrlinger, Rajeswaran, & Blackstone, 2019, pp. 38-40)



**Figure 10-30.** Partial dependence of survival (Z-axis) as a function of bilirubin over a five year follow up time period.

### Example 10-3: Business Application

#### Scenario and dataset

A marketing department of a bank runs various marketing campaigns for cross-selling products, improving customer retention and customer services. In this example, the bank wanted to cross-sell term deposit product to its customers. Contacting all customers is costly and does not create good customer experience. So, the bank wanted to build a predictive model which will identify customers who are more likely to respond to term deposit cross sell campaign. We will use sample Marketing Data sample for building random forest based model using R.

## *Read and Explore data*

```
Bank <- read.csv(file="C:/Users/Jeff/Documents/VIT_University/data/Banking.csv",header = T)
names(bank)

## [1] "job"      "marital"   "education" "age"     "balance"
## [6] "homeowner" "loans"    "default"   "contact"  "length"
## [11] "campaign"  "pdays"    "previous"  "poutcome" "RESP"
```

Input dataset has 20 independent variables and a target variable. The target variable y is binary.

```
table(bank$RESP)/nrow(bank)

##          NO         YES
## 0.8830178 0.1169822
```

Eleven % of the observations has target variable "yes" and remaining 89% observations take value "no".

Now, we will split the data sample into development and validation samples.

```
sample.ind <- sample(2,
                      nrow(bank),
                      replace = T,
                      prob = c(0.6,0.4))
#sample.ind <- sample(2, 10000, replace=F)
bank.train <- bank[sample.ind==1,]
bank.test <- bank[sample.ind==2,]
bank.train$RESP=as.factor(bank.train$RESP)
bank.test$RESP=as.factor(bank.test$RESP)
table(bank.train$RESP)/nrow(bank.train)

##          NO         YES
## 0.8824438 0.1175562

table(bank.test$RESP)/nrow(bank.test)

##          NO         YES
## 0.8835941 0.1164059

bank.test$RESP
```

```

## [1] NO NO
## [18] NO NO NO YES NO YES NO
## [35] NO NO
...
## [22628] YES YES YES YES YES YES NO YES NO NO YES NO NO NO NO NO
## [22645] NO NO YES YES NO YES YES YES YES
## Levels: NO YES

```

Both development and validation samples have similar target variable distribution. This is just a sample validation.

If target variable is factor, classification decision tree is built. We can check the type of response variable.

```

class(bank.train$RESP)
## [1] "factor"
class(bank.test$RESP)
## [1] "factor"

```

Class of target or response variable is factor, so a classification Random Forest will be built. The current data frame has a list of independent variables, so we can make it formula and then pass as a parameter value for `randomForest`.

### *Make Formula*

```

varNames <- names(bank.train)
# Exclude ID or Response variable
varNames <- varNames[!varNames %in% c("RESP")]
# add + sign between exploratory variables
varNames1 <- paste(varNames, collapse = "+")
# Add response variable and convert to a formula object
rf.form <- as.formula(paste("RESP", varNames1, sep=" ~ "))

```

### *Building Random Forest using R*

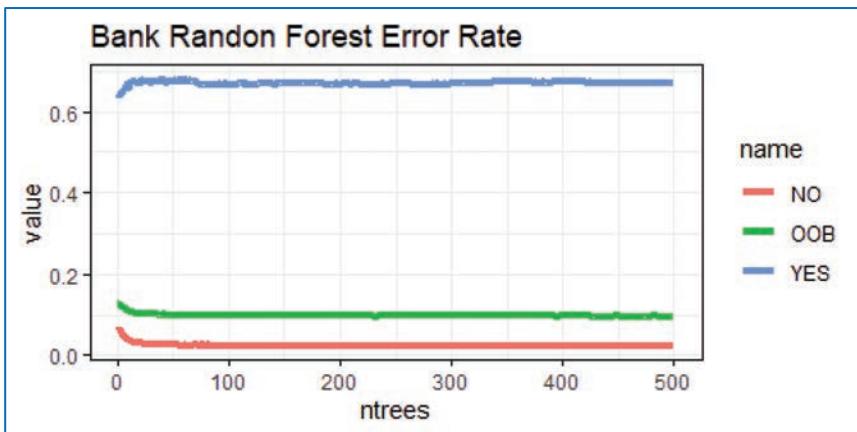
Now, we have a sample data and formula for building Random Forest model. Let's build 500 decision trees using Random Forest and plot the outcome in **Figure 10-31**.

```
bank.rf <- randomForest(rf.form,
```

```

bank.train, ntree = 500, importance = T)
plotdf <- pivot_longer(data.frame(
  ntrees = 1:nrow(lwt.rf$err.rate),
  bank.rf$err.rate), -ntrees)
ggplot(plotdf, aes(x= ntrees, y= value, col= name)) +
  geom_line(lwd = 1.5) +
  ggtitle("Bank Random Forest Error Rate") + theme_bw()

```



**Figure 10-31.** Cross-sell random forest model

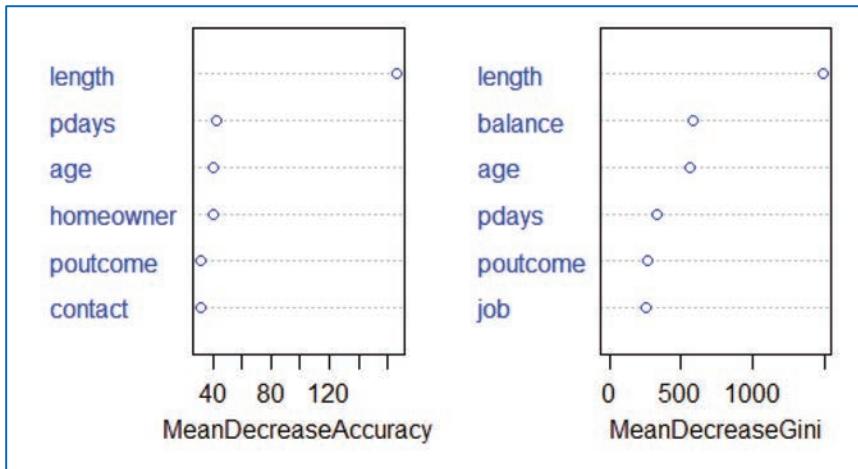
500 decision trees or a forest has been built using the random forest algorithm-based learning. We can plot the error rate across decision trees. The plot seems to indicate that after 100 decision trees, there is not a significant reduction in error rate.

YES and NO are lines for error in prediction for that specific label, and OOB (our first column) is simply the weighted-average of the two. As the number of trees increase, your OOB error gets lower because you get a better prediction from more trees.

### Variable Importance Plot

Variable importance plot is also a useful tool and can be plotted using `varImpPlot` function. Top 5 variables are selected and plotted based on Model Accuracy and Gini value. We can also get a table with decreasing order of importance based on a measure (1 for model accuracy and 2 node impurity). **Figure 10-32** contains the variable importance plot.

```
varImpPlot(bank.rf,
            sort = T,
            main="Variable Importance",
            n.var = 5)
```



**Figure 10-32.** Variable importance plot by mean decreasing accuracy (left) and mean decreasing Gini score (right)

### Variable Importance Table

```
var.imp <- data.frame(importance(bank.rf,
                                    type=2))
```

Here, we make row names as columns:

```
var.imp$Variables <- row.names(var.imp)
var.imp[order(var.imp,decreasing = T),]
```

|         | MeanDecreaseGini | Variables |
|---------|------------------|-----------|
| ## NA   | NA               | <NA>      |
| ## NA.1 | NA               | <NA>      |
| ## NA.2 | NA               | <NA>      |
| ## NA.3 | NA               | <NA>      |
| ## NA.4 | NA               | <NA>      |
| ## NA.5 | NA               | <NA>      |
| ## NA.6 | NA               | <NA>      |
| ## NA.7 | NA               | <NA>      |
| ## NA.8 | NA               | <NA>      |
| ## NA.9 | NA               | <NA>      |

```

## NA.10             NA      <NA>
## NA.11             NA      <NA>
## NA.12             NA      <NA>
## NA.13             NA      <NA>
## length          1300.789733  length
## balance         497.901416  balance
## age              456.717286   age
## job              320.024358   job
## pdays            276.070140   pdays
## poutcome        217.999505  poutcome
## campaign        174.409370  campaign
## homeowner       125.707781  homeowner
## education        123.510237  education
## previous         119.876718  previous
## marital           99.500687  marital
## contact           98.514174  contact
## loans             47.061799   loans
## default           9.970018   default

```

Based on random forest variable importance, the variables could be selected for any other predictive modelling techniques or machine learning.

Now, we want to measure the accuracy of the Random Forest model. Some of the other model performance statistics are: - KS - Lift Chart - ROC curve

### *Predict Response Variable Value using Random Forest*

Generic predict function can be used for predicting response variable using Random Forest object. Here we use a predicting response variable:

```
bank.train$predicted.response <- predict(
    bank.rf , bank.train)
```

### *Confusion Matrix*

`confusionMatrix` function from `caret` package can be used for creating confusion matrix based on actual response variable and predicted value.

## *Create Confusion Matrix*

```
library(e1071)
library(caret)
confusionMatrix(data=bank.train$predicted.response,
                reference=bank.train$RESP)

## Confusion Matrix and Statistics
##             Reference
## Prediction    NO     YES
##           NO 19604   1188
##           YES   235   1473
##                   Accuracy : 0.9368
##                   95% CI : (0.9335, 0.9399)
##       No Information Rate : 0.8817
##       P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6411
## Mcnemar's Test P-Value : < 2.2e-16
##
##                   Sensitivity : 0.9882
##                   Specificity : 0.5536
##      Pos Pred Value : 0.9429
##      Neg Pred Value : 0.8624
##          Prevalence : 0.8817
##      Detection Rate : 0.8713
## Detection Prevalence : 0.9241
##      Balanced Accuracy : 0.7709
##      'Positive' Class : NO
```

It has accuracy of 99.81%, which is fantastic. Now we can predict response for the validation sample and calculate model accuracy for the sample.

## *Predicting response variable*

```
bank.test$predicted.response <- predict(bank.rf,bank.test)
```

## *Create Confusion Matrix*

```
confusionMatrix(data=bank.test$predicted.response,
                reference=bank.test$RESP)
```

```

## Confusion Matrix and Statistics
##             Reference
## Prediction    NO     YES
##           NO 19820   1156
##           YES   264  1472
##                   Accuracy : 0.9375
##                   95% CI : (0.9343, 0.9406)
##       No Information Rate : 0.8843
##       P-Value [Acc > NIR] : < 2.2e-16
##                   Kappa : 0.6416
## McNemar's Test P-Value : < 2.2e-16
##                   Sensitivity : 0.9869
##                   Specificity : 0.5601
##      Pos Pred Value : 0.9449
##      Neg Pred Value : 0.8479
##                   Prevalence : 0.8843
##       Detection Rate : 0.8727
## Detection Prevalence : 0.9236
##       Balanced Accuracy : 0.7735
## 'Positive' Class : NO

```

Accuracy level has dropped to 91.4% but still significantly higher.

## Example 10-4: Fit and Compare fgl Data RF and SVM

The `fgl` data frame has 214 rows and 10 columns. It was collected by B. German on fragments of glass collected in forensic work.

```

library(MASS)
data(fgl)
set.seed(17)
fgl.rf <- randomForest(type ~ ., data = fgl, mtry = 2,
                        importance = TRUE, do.trace = 100)
## ntree      OOB      1      2      3      4      5      6
## 100: 20.09% 14.29% 18.42% 58.82% 23.08% 22.22% 13.79%
## 200: 21.03% 10.00% 22.37% 64.71% 23.08% 33.33% 13.79%
## 300: 18.69% 10.00% 18.42% 64.71% 23.08% 11.11% 13.79%
## 400: 19.16% 11.43% 17.11% 64.71% 23.08% 22.22% 13.79%
## 500: 19.63% 11.43% 19.74% 58.82% 23.08% 22.22% 13.79%

print(fgl.rf)

```

```

## Call:
## randomForest(formula = type ~ ., data = fgl, mtry = 2,
importance = TRUE,      do.trace = 100)
##                         Type of random forest: classification
##                               Number of trees: 500
## No. of variables tried at each split: 2
##             OOB estimate of  error rate: 19.63%
## Confusion matrix:
##              WinF WinNF Veh Con Tabl Head class.error
## Win       62     6   2   0     0     0    0.1142857
## WinNF     11    61   1   1     1     1    0.1973684
## Veh        7     3   7   0     0     0    0.5882353
## Con        0     2   0   10    0     1    0.2307692
## Tabl       0     2   0   0     7     0    0.2222222
## Head       1     3   0   0     0    25   0.1379310

par(mfrow = c(2, 2))
for (i in 1:4) plot(sort(fgl.rf$importance[,i], dec = TRUE),
, type = "h", main = paste("Measure", i))

```

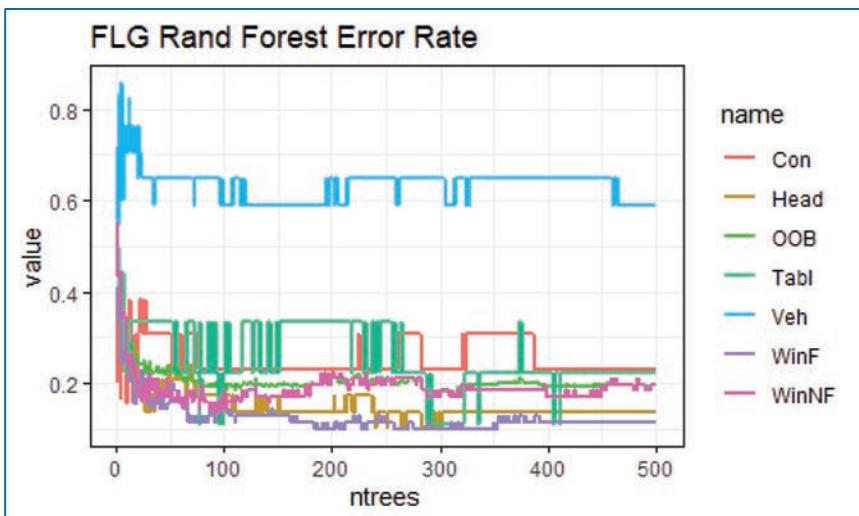
Now we can also check the error rate of the FLG random forest for all its variables, shown in **Figure 10-33**.

```

plotdf <- pivot_longer(data.frame(
  ntrees=1:nrow(fgl.rf$err.rate),
  fgl.rf$err.rate), -ntrees)
ggplot(plotdf,aes(x= ntrees, y= value, col= name)) +
  geom_line(lwd = 1.5) +
  ggtitle("FLG Rand Forest Error Rate") +
  theme_bw()

```

Referring to **Figure 10-33**, the legend shows that the variables are the colored lines for error in prediction (except the green line) for that specific label, and OOB (the green line) is simply the average of the all the variables. As the number of trees increase, your OOB error gets lower because you get a better prediction from more trees.



**Figure 10-33.** Error Rate plot for the FLG Random Forest Model

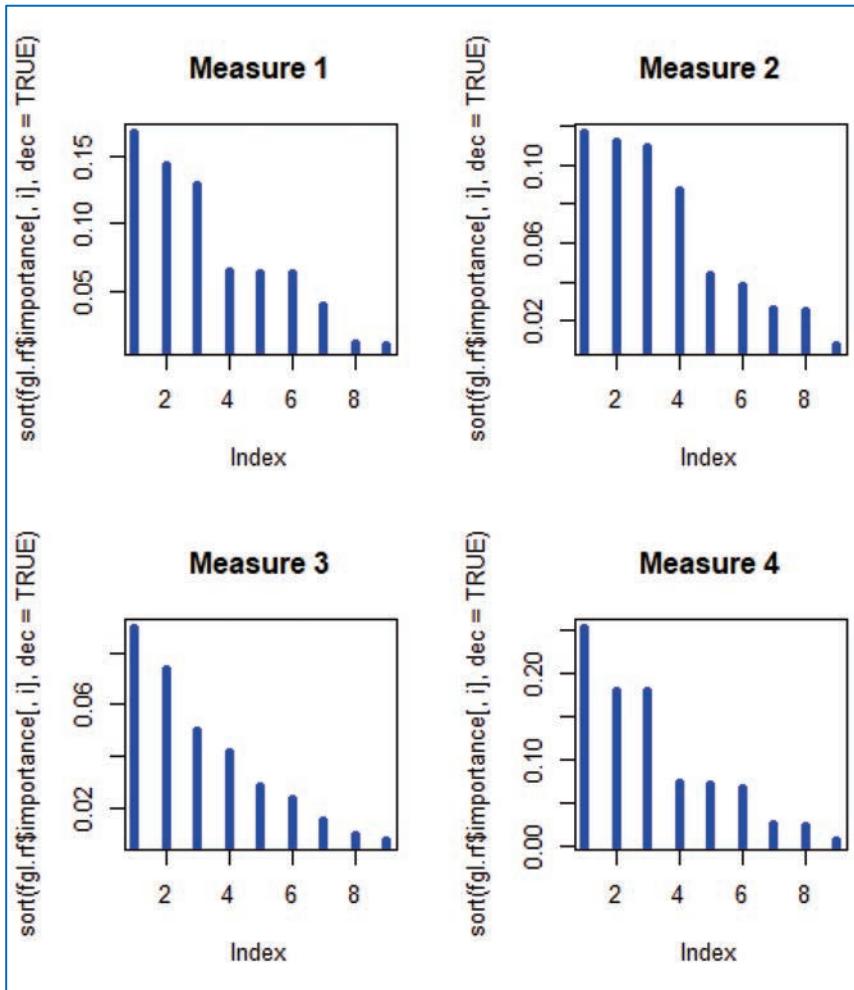
We can compare random forests with support vector machines by doing ten repetitions of 10-fold cross-validation, using the `errorest` functions in the `ipred` package. `errorest` performs resampling, based estimates of prediction error: misclassification error, root mean squared error or Brier score for survival data.

```
library(ipred)

set.seed(131)
error.RF <- numeric(10)
  for(i in 1:10) error.RF[i] <- errorest(type ~ .,
data = fgl, model = randomForest, mtry = 2)$error
summary(error.RF)
##      Min. 1st Qu. Median     Mean 3rd Qu.      Max.
##  0.1822  0.1928  0.2009  0.2033  0.2173  0.2243

set.seed(563)
error.SVM <- numeric(10)
for (i in 1:10) error.SVM[i] <- errorest(type ~ ., data =
fgl, model = svm, cost = 10, gamma = 1.5)$error
summary(error.SVM)
##      Min. 1st Qu. Median     Mean 3rd Qu.      Max.
##  0.3645  0.3785  0.3808  0.3794  0.3832  0.3879
```

We see that the random forest compares quite favorably with SVM. We have found that the variable importance measures produced by random forests can sometimes be useful for model reduction. **Figure 10-34** illustrates the previous conclusion.



**Figure 10-34.** Variable importance for random forest model of fgl data

# 11. Shiny Apps

## Structure of a Shiny App

Shiny is a framework for creating web applications using R code. It is designed primarily with data scientists in mind, and to that end, you can create pretty complicated Shiny apps with no knowledge of HTML, CSS, or JavaScript. On the other hand, Shiny doesn't limit you to creating trivial or prefabricated apps: its user interface components can be easily customized or extended, and its server uses reactive programming to let you create any type of back-end logic you want. Shiny is designed to feel almost magically easy when you're getting started, and yet the deeper you get into how it works, the more you realize it's built out of general building blocks that have strong software engineering principles behind them.

Shiny apps are contained in a single script called `app.R`. The script `app.R` lives in a directory (for example, `newdir/`) and the app can be run with `runApp("newdir")`.

`app.R` has three components:

- a user interface object
- a server function
- a call to the `shinyApp` function

The user interface (`ui`) object controls the layout and appearance of your app. The `server` function contains the instructions that your computer needs to build your app. Finally, the `shinyApp` function creates Shiny app objects from an explicit UI/server pair.

**Note:** Prior to version 0.10.2, Shiny did not support single-file apps and the `ui` object and `server` function needed to be contained in separate scripts called `ui.R` and `server.R`, respectively. This functionality is still supported in Shiny, however the tutorial and much of the supporting documentation focus on single-file apps.

One nice feature about single-file apps is that you can copy and paste the entire app into the R console, which makes it easy to quickly share code for others to experiment with. For example, if you copy and

paste the code above into the R command line, it will start a Shiny app.

## Example 11-1. Hello Shiny! App

### *User Interface (ui)*

Here is the ui object for the **Hello Shiny** example.

```
library(shiny)
# Define UI for app that draws a histogram ----
ui <- fluidPage(
  # App title ----
  titlePanel("Hello Shiny!"),
  # Sidebar layout with input and output definitions
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    # Main panel for displaying outputs ----
    mainPanel(
      # Output: Histogram ----
      plotOutput(outputId = "distPlot")
    )
  )
)
```

### *Server*

Here is the server function for the **Hello Shiny** example.

```
# Define server logic required to draw a histogram ----
server <- function(input, output) {
  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins
  # Generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
```

```

#
# 1. It is "reactive" and is automatically
#    re-executed when inputs (input$bins) change
# 2. Its output type is a plot
output$distPlot <- renderPlot({
  x     <- faithful$waiting
  bins <- seq(min(x), max(x), length.out =
              input$bins + 1)

  hist(x, breaks = bins, col = "#75AADB",
        border = "white",
        xlab = "Waiting time to next eruption (in mins)",
        main = "Histogram of waiting times")
})
}

```

At one level, the **Hello Shiny server** function is very simple. The script does some calculations and then plots a histogram with the requested number of bins.

However, you'll also notice that most of the script is wrapped in a call to `renderPlot`. The comment above the function explains a bit about this, but if you find it confusing, don't worry. We'll cover this concept in much more detail soon.

Play with the **Hello Shiny** app and review the source code. Try to develop a feel for how the app works. But before you do so, note that in your `app.R` file you will need to start with loading the Shiny package and end with a call to `shinyApp`:

```

library(shiny)
# Run the application
shinyApp(ui = ui, server = server)

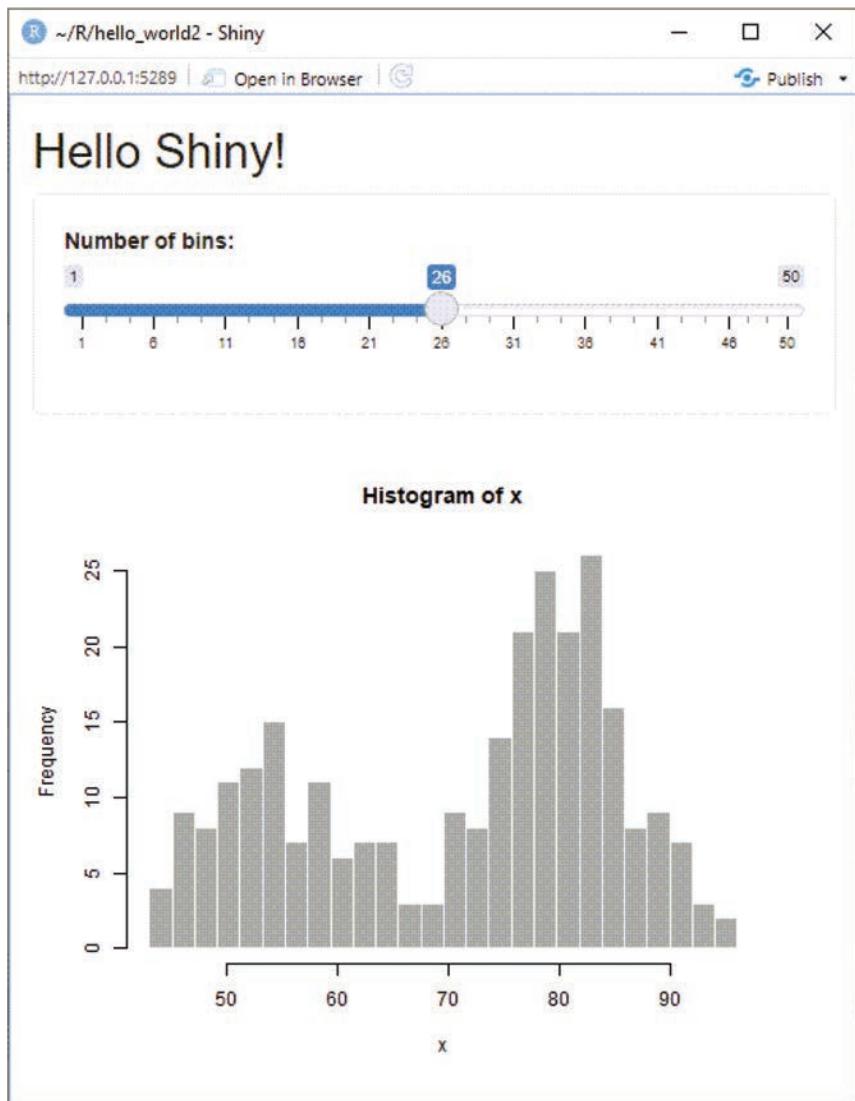
```

### *Running an App*

Every Shiny app has the same structure: an `app.R` file that contains `ui` and `server`. You can create a Shiny app by making a new directory and saving an `app.R` file inside it. It is recommended that each app will live in its own unique directory. You can run the app before deploying it, by running it in the RStudio viewer as in **Figure 11-1**.

You can run a Shiny app by giving the name of its directory to the function `runApp`. For example, if your Shiny app is in a directory called `my_app`, run it with the following code:

```
runApp("my_app")
```



**Figure 11-1.** Hello Shiny! App in the RStudio viewer

If you split your code in two scripts, `ui` and `server`, use:

```
shinyApp(ui = ui, server = server)
```

Your R session will be busy while the **Hello Shiny** app is active, so you will not be able to run any R commands. R is monitoring the app and executing the app's reactions. To get your R session back, hit escape or click the stop sign icon (found in the upper right corner of the RStudio console panel).

Note: `runApp` is similar to `read.csv`, `read.table`, and many other functions in R. The first argument of `runApp` is the filepath from your working directory to the app's directory. The code above assumes that the app directory is in your working directory. In this case, the filepath is just the name of the directory.

(In case you are wondering, the **Hello Shiny** app's files are saved in a special system directory called "`01_hello`". This directory is designed to work with the `runExample("01_hello")` call.)

## Example 11-2. Next Word Prediction Model

The source code is at: <https://github.com/stricje1/capstone>. The shiny app is at: [https://stricje1.shinyapps.io/next\\_word\\_predictor/](https://stricje1.shinyapps.io/next_word_predictor/).

```
#' Shiny App
#' This script creates a Shiny App that takes a word or
phrase input in a text box and outputs the next predicted
word.

library(shiny)
suppressPackageStartupMessages({
  library(tidyverse)
  library(stringr)
  library(rintrojs)
})
#' Source ngram function is an external file. It
# Loads the data and calculates the predicted values.
source("ngram.R")

#' Define UI for application that draws a histogram
```

```

ui <- fluidPage(


  # Application title
  titlePanel("Next Word Prediction Model"),
  p("This app that takes an input phrase (multiple words)
in a text box and outputs a prediction of the next
word."),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      h2("Instructions:"), # level-2 header
      h5("1. Enter a word or words in the text box."),
      h5("2. The predicted next word prints below it in
blue."),
      h5("3. No need to hit enter or submit."),
      h5("4. A question mark means no prediction, typically
due to mis-spelling"),
      h5("5. Additional tabs show plots of the top ngrams in
the dataset"),
      br(),
      a("Source Code", href =
"https://github.com/stricje1/capstone")
    ),
    # Show a plot of the generated distribution
    mainPanel(
      tabsetPanel(
        tabPanel("predict",
         textInput("user_input", h3("Your Input:"),
            value = "Your words"),
          h3("Predicted Next Word:"), # level-3 header
          h4(em(span(textOutput("ngram_output"),
            style="color:blue")))),
        # Shiny app Quadgram tab
        tabPanel("top quadgrams",
          br(),
          column(width = 12,
            h3("Simulated Response Estimates"),
            # plot output object
            plotOutput(outputId = "quadgram_plot")),
        introBox(numericInput(inputId = "QUAD_grams",
          20, label = "Number of Quad-grams")))

```



```

coord_flip() + scale_y_continuous(name =
  "Word Frequency") +
  scale_x_discrete(name = "Quad-grams") +
  theme(plot.title = element_text(face = "bold",
  color = "steelblue4", size=12),
        axis.text.x = element_text(face = "bold",
  color = "steelblue4", size=8, angle=0),
        existent = element_text(face = "bold",
  color = "steelblue4", size=8, angle=0))
})

# Trigram histogram
output$trigram_plot = renderPlot({
  options(repr.plot.width=8, repr.plot.height=3)
  ggplot(dfFreq3[1:input$TRI_grams, ],
    aes(word, freq)) +
    ggtitle("Tri-grams Frequencies") +
    geom_bar(stat = "identity", fill = "dodgerblue",
    colour = "skyblue") +
    coord_flip() + scale_y_continuous(name = "Word
Frequency") +
    scale_x_discrete(name = "Tri-grams") +
    theme(plot.title = element_text(face = "bold",
  color = "steelblue4", size=12),
        axis.text.x = element_text(face = "bold",
  color = "steelblue4", size=8, angle=0),
        axis.text.y = element_text(face = "bold",
  color = "steelblue4", size=8, angle=0))
})

# Bigram histogram
output$bigram_plot = renderPlot({
  options(repr.plot.width=8, repr.plot.height=3)
  ggplot(dfFreq2[1:input$BI_grams, ],
    aes(word, freq)) +
    ggtitle("Bi-grams Frequencies") +
    geom_bar(stat = "identity",
    fill = "dodgerblue", colour = "skyblue") +
    coord_flip() +
    scale_y_continuous(name = "Word Frequency") +
    scale_x_discrete(name = "Bi-grams") +
    theme(plot.title = element_text(face = "bold",
  color = "steelblue4", size=12),
        axis.text.x = element_text(face = "bold",

```

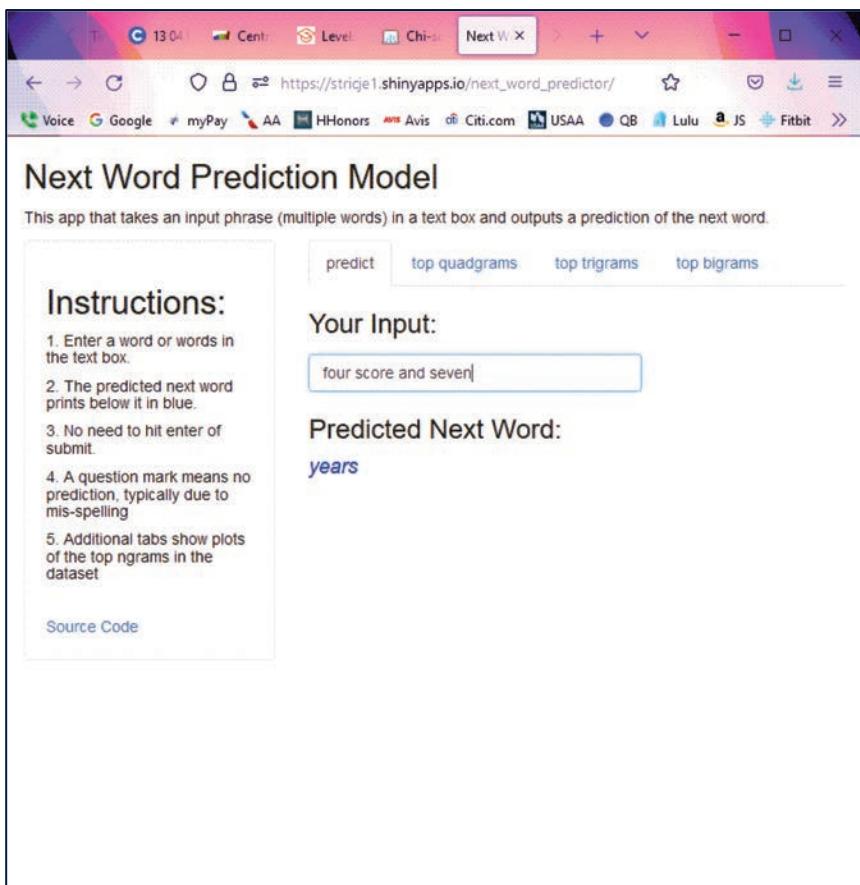
```

        color = "steelblue4", size=8, angle=0),
        axis.text.y = element_text(face = "bold",
        color = "steelblue4", size=8, angle=0))
    })
}

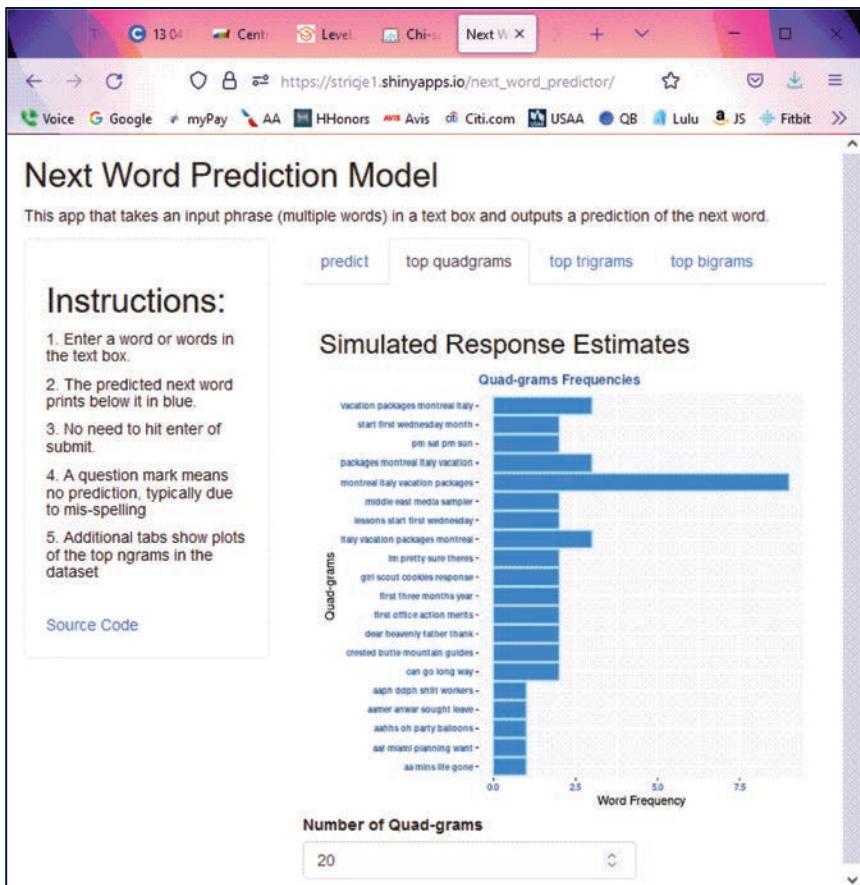
#' Run the application
# https://stricje1.shinyapps.io/next_word_predictor/
shinyApp(ui = ui, server = server)

```

The main tab of the app is shown in **Figure 11-2** and the gradients tab view is shown in **Figure 11-3**.



**Figure 11-2.** Next Word Prediction Model app main tab view



**Figure 11-3.** Next Word Prediction Model app quad-grams tab view

### Example 11-3: COVID-19 Data Discovery App

The source is at <https://github.com/stricje1/ddp4>. The Shiny app is at: <https://stricje1.shinyapps.io/ddp4/>. The COVID-19 Data Discovery app is comprised of easy-to-use user input (ui) with two basic entries:

1. Choosing a datasetInput
2. Selecting variables to plot

There are currently six datasets from Centers of Disease Control (CDC) data products:

1. CDC Monthly Disease tracking for Alaska from January 2020 to December 2021.
2. CDC Monthly Disease tracking for Colorado from January 2020 to December 2021.
3. CDC Monthly Disease tracking for Georgia from January 2020 to December 2021.
4. CDC Monthly Disease tracking for Florida from January 2020 to December 2021.
5. CDC Monthly Disease tracking for Missouri from January 2020 to December 2021.
6. CDC Monthly Disease tracking for New York from January 2020 to December 2021.

Tables can be added in the header of the source code, before the server code. Selecting a database results in a summary table, n-observations of the data set, and a plot of the first six variables. **Figure 11-4** shows the finalized app.

The datasets can be imported from the following site as CSV or JSON file formats:

["https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Public-Use-Data-with-Ge/n8mc-b4w4/data"](https://data.cdc.gov/Case-Surveillance/COVID-19-Case-Surveillance-Public-Use-Data-with-Ge/n8mc-b4w4/data)

These files can be imported in RStudio as JSON (faster download than CSV) using, for example:

```
url = "https://data.cdc.gov/resource/n8mc-b4w4.json"
covid <- jsonlite::fromJSON(url)
```

The supporting files can be found on by “dpp4” GitHub repository (<https://github.com/stricje1/ddp4>). The source code for the data (ui and server) can also be accessed there.

### *User Interface (ui) Code*

```
library(shiny)
library(shinythemes)
library(dplyr)
library(shinyhelper)
```

```

# Define UI for dataset viewer application
shinyUI(fixedPage(theme = shinytheme("flatly"),

                    # Application title.
                    titlePanel("Covid Data"),

                    fixedRow(
                      column(3,
                            selectInput("dataset", "Choose a dataset:",
   choices = c("Alaska COVID Cases",
   "Colorado COVID Cases",
   "Georgia COVID Cases",
   "Florida COVID Cases",
   "Missouri COVID Cases",
   "New York COVID Cases")))%>%
                            helper(icon = "question",
                                   colour = "green",
                                   type = "inline",
                                   content = c("Dataset Help:",
   "Choose a set from the pulldown menu."),
   buttonLabel = "Got it!",
   easyClose = FALSE,
   fade = TRUE,
   size = "s"),
                            radioButtons("varname", "Variable Name",
   c("status_num", "status_character")) %>%
                            helper(icon = "question",
                                   colour = "green",
                                   type = "inline",
                                   content = c("Variable Help:",
   "The variable select buttons work,",
   "but they are not connected to any",
   "functional code yet."),
   buttonLabel = "Got it!",
   easyClose = FALSE,
   fade = TRUE,
   size = "s"),
   br(),
   br()),
   column(9, tabsetPanel(
   h4("Selected Covid Data")),
   
```

```

        tabPanel("Data Table",
DT::dataTableOutput("view"), value="thistab"),
        tabPanel("Covid Plot", plotOutput("dataplot")),
        tabPanel("Documentation",
includeMarkdown("data_discovery_help.md")),
        selected = "thistab"
    )
)
))

```

### *Server Code*

```

## COVOD Data Discovery
library(shiny)
library(datasets)
library(dplyr)
library(formattable)
library(DT)
library(ggplot2)

options(encoding = "UTF-8") #required to avoid: warning in
readLines(con) incomplete final line

#need to consider
http://shiny.rstudio.com/articles/datatables.html
options("scipen" = 100)
options("scipen" = 100)
covid_case_surv_co <-
read.csv("./shiny/covid_case_surv_co.csv", header = TRUE)
covid_case_surv_ak <-
read.csv("./shiny/covid_case_surv_ak.csv", header = TRUE)
covid_case_surv_ga <-
read.csv("./shiny/covid_case_surv_ga.csv", header = TRUE)
covid_case_surv_fl <-
read.csv("./shiny/covid_case_surv_fl.csv", header = TRUE)
covid_case_surv_mo <-
read.csv("./shiny/covid_case_surv_mo.csv", header = TRUE)
covid_case_surv_ny <-
read.csv("./shiny/covid_case_surv_ny.csv", header = TRUE)

# Define server logic required to summarize and view the

```

```

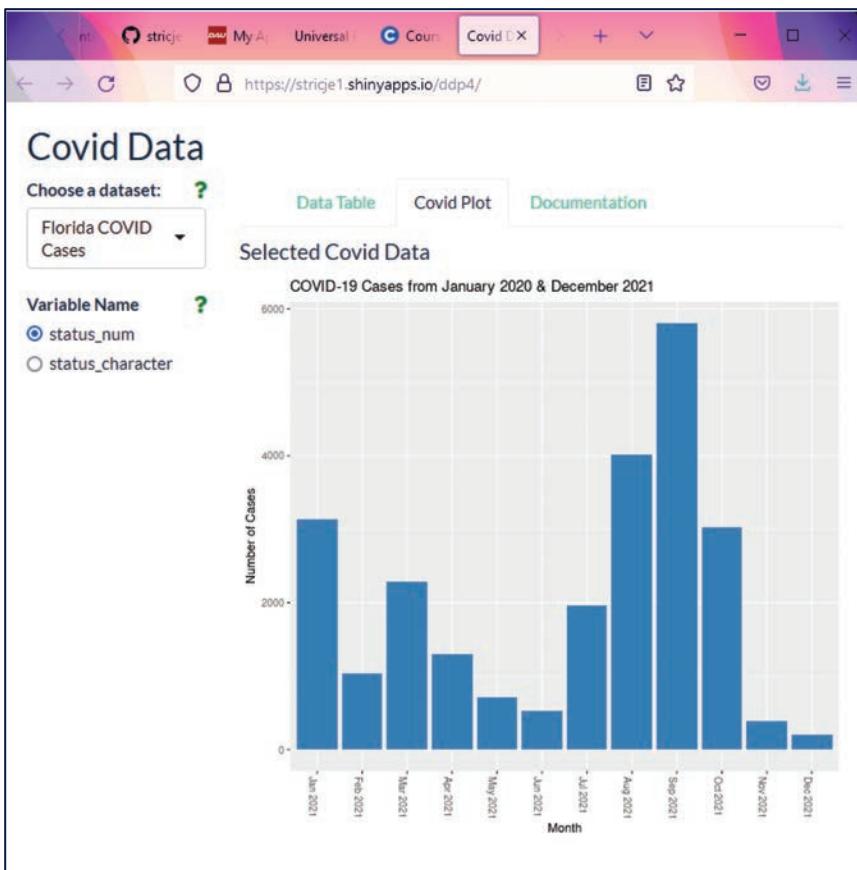
# selected dataset
shinyServer(function(input, output) {
  observe_helpers(session =
shiny::getDefaultReactiveDomain(),
  help_dir = "helpfiles", withMathJax = FALSE)
# Return the requested dataset
datasetInput <- reactive({
  switch(input$dataset,
    "Alaska COVID Cases" = covid_case_surv_ak,
    "Colorado COVID Cases" = covid_case_surv_co,
    "Georgia COVID Cases" = covid_case_surv_ga,
    "Florida COVID Cases" = covid_case_surv_fl,
    "Missouri COVID Cases" = covid_case_surv_mo,
    "New York COVID Cases" = covid_case_surv_ny
  )
})
varnameInput<-reactive({
  if(input$varname=="status_num"){
    switch(input$varname,
      "Variable Name" = status_num)
  } else{
    switch(input$varname,
      "Variable Name" = status_character)
  }
}

output$view <- DT:::renderDataTable({
  DT:::datatable(datasetInput(),
  options=list(searching=FALSE, paging=FALSE,
  rownames=FALSE))
})

library(tidyverse)
library(zoo)
output$dataplot <- renderPlot(height = 500,{
  par(mar=c(8.3, 4.1, 2, 1))
  ggplot(datasetInput(), aes(x =
(as.factor(as.yearmon(case_month)))) +
  geom_bar(stat = "count", fill="steelblue") +
  theme(axis.text.x = element_text(angle = -90, hjust =
0)) +
  labs(x= "Month",
  y = "Number of Cases",

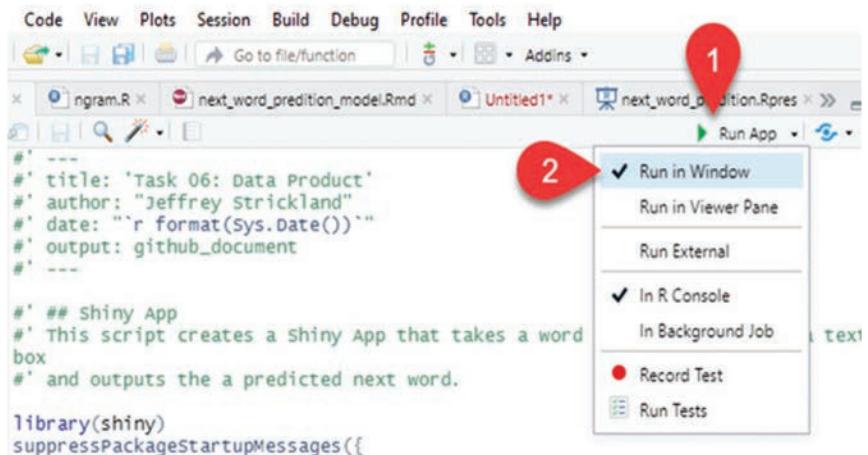
```

```
    title = "COVID-19 Cases from January 2020 &  
December 2021")  
})  
})
```



**Figure 11-4.** COVID Shiny App in web-view

In R studio, we can run the app from the Run App pull-down menu (1) and selecting Run in Window (2), shown in **Figure 11-5**.



**Figure 11-5.** Running a Shiny App in RStudio

## Deploying Shiny Apps on the Web

For information on how to deploy a Shiny App, Antoine Soetewey has written a step-by-step guide, *How to publish a Shiny app: example with shinyapps.io* at (Soetewey, 2020):

<https://towardsdatascience.com/how-to-publish-a-shiny-app-example-with-shinyapps-io-ec6c6604d8>

You can deploy 5 applications for free at:

<https://www.shinyapps.io/>

## References

---

- Altmann, A., Tolosi, L., Sander, O., & Lengauer, T. (2010). Permutation importance:a corrected feature importance measure. *Bioinformatics*. doi:10.1093/bioinformatics/btq134
- Amit, Y., & Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7), 1545–1588. doi:10.1162/neco.1997.9.7.1545
- Aslam, J. A., Popa, R. A., & Rivest, R. (2007). On Estimating the Size and Confidence of a Statistical Audit. *Proceedings of the Electronic Voting Technology Workshop (EVT '07)*. Boston, MA. Retrieved from <http://people.csail.mit.edu/rivest/pubs/APR07.pdf>
- Breiman, L. (1994). *Bagging Predictors*. Technical Report No. 421, University of California, Department of Statistics, Berkeley. Retrieved from <http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf>
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123-140. Retrieved from <http://link.springer.com/article/10.1023%2FA%3A1018054314350>
- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32. doi:10.1023/A:1010933404324
- Breiman, L., H., F. J., Olshen, R., & Stone, C. (1984). *Classification and Regression*. Monterey, CA.: Wadsworth and Brooks. Retrieved from <ftp://ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps.Z>.
- Bryll, R. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 20(6), 1291–1302.

- Criminisi, A., Shotton, J., & Konukoglu, E. (2011). Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Foundations and Trends in Computer Vision*, 7, 81–227. doi:10.1561/0600000035
- Deng, H., Runger, G., & Tuv, E. (2011). Bias of importance measures for multi-valued attributes and solutions. *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN)*, (pp. 293–300).
- Dietterich, T. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 139–157.
- Ehrlinger, J., Rajeswaran, J., & Blackstone, E. (2019, January 1). ggRandomForests: Exploring Random Forest Survival. *Statistical Methods in Medical Research*, 27(1), 126-141.
- GOV.UK. (2022, Feb 19). *UK Coronavirus Dashboard*. Retrieved from Coronavirus (COVID-19) in the UK: <https://coronavirus.data.gov.uk/>
- Ho, T. (1995). Random Decision Forest. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, (pp. 278–282). Montreal, QC. Retrieved from <http://cm.bell-labs.com/cm/cs/who/tkh/papers/odt.pdf>
- Ho, T. (1998). The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 832–844. doi:10.1109/34.709601
- Ishwaran, H., Kogalur, U. B., Chen, X., & J., M. A. (2011). Random Survival Forests for High- Dimensional Data. *Statist. Anal. Data Mining*, 4, 115-132.
- Ishwaran, H., Kogalur, U., Gorodeski, E. Z., & Minn, A. J. (2010). High- Dimensional Variable Selection for Survival Data. *J. Amer. Statist. Assoc.*, 105, 205–217.

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. New York: Springer.
- JHU. (2021). *Home*. Retrieved Feb 19, 2022, from Johns Hopkins Coronavirus Resource Center.: <https://coronavirus.jhu.edu/>
- Kleinberg, E. (1996). An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition. *Annals of Statistics*, 24(6), 2319–2349. doi:10.1214/aos/1032181157
- Kuncheva, L., Rodríguez, J., Plumpton, C., Linden, D., & Johnston, S. (2010). Random subspace ensembles for fMRI classification. *IEEE Transactions on Medical Imaging*, 29(2), 531–542. Retrieved from <http://pages.bangor.ac.uk/~mas00a/papers/lkjrcpdlsjtmi10.pdf>
- Lewis, M. (2003). *Moneyball: The Art of Winning an Unfair Game*. New York: W. W. Norton & Company.
- Liaw, A. (2012, October 16). *Documentation for R package randomForest*. Retrieved from The Comprehensive R Archive Network: <http://cran.r-project.org/web/packages/randomForest/randomForest.pdf>
- Lin, Y., & Jeon, Y. (2001). *Random forests and adaptive nearest neighbors*. Technical Report No. 1055, University of Wisconsin. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.9168>
- McCullagh, P., & Nelder, J. (1989). *Generalized Linear Models* (2nd ed.). Boca Raton: Chapman and Hall/CRC.
- Nelder, J., & Wedderburn, R. (1989). Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3), 370–384. doi:10.2307/2344614
- NOAA. (2021). *Storm Data Bulk Data Format*. Retrieved from <https://www.ncei.noaa.gov/pub/data/swdi/stormevents/csvfiles/Storm-Data-Bulk-csv-Format.pdf>

- Peng, R. D., Butz, A. M., Hackstadt, A. J., Williams, D. L., Diette, G. B., Breysse, P. N., & Matsui, E. C. (2015). Estimating the health benefit of reducing indoor air pollution in a randomized environmental intervention. *Journal of the Royal Statistical Society*.
- Pohl, J. (2016, July 29). 40 years later: Scores killed in Big Thompson Flood. *Coloradoan*. Retrieved from <https://www.coloradoan.com/story/news/2016/07/29/big-thompson-flood-killed-scores/87524858/>
- Prinzie, A., & Van den Poel, D. (2008). Random Forests for multiclass classification: Random MultiNomial Logit". *Expert Systems with Applications*, 34(3), 1721–1732.
- Rousseeuw, P. J., & Leroy, A. M. (2003). *Robust Regression and Outlier Detection*. New York: Wiley.
- Sahu, A., Runger, G., & Apley, D. (2011). Image denoising with a multi-phase kernel principal component approach and an ensemble version. *IEEE Applied Imagery Pattern Recognition Workshop*, (pp. 1-7).
- Samenow, J., Feuerstein, J., & Livingston, I. (2021, December 11). How Friday night's rare and deadly December tornado outbreak unfolded. *Seattle Times*. Retrieved from <https://www.washingtonpost.com/weather/2021/12/11/tornado-path-mayfield-kentucky-deaths/>
- Schafer, J. (1997). *Analysis of multivariate incomplete data*. London: Chapman&Hall.
- Shinde, A., Sahu, A., Apley, D., & Runger, G. (2014). Preimages for Variation Patterns from Kernel PCA and Bagging. *IIE Transactions*, 46(5).
- Skurichina, M. (2002). Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis and Applications*, 5(2), 121–135. doi:10.1007/s100440200011
- Smith, O. (2020, May 5). Data Reshaping in R. *datacamp.com/tutorials*, p. Online. Retrieved from

<https://www.datacamp.com/community/tutorials/data-reshaping-in-r>

Soetewey, A. (2020, May 28). *How to publish a Shiny app: example with shinyapps.io*. Retrieved from Towards Data Science: <https://towardsdatascience.com/how-to-publish-a-shiny-app-example-with-shinyapps-io-ec6c6604d8>

Strickland, J. (2019). *Data Science Applications using R*. Lulu.com. Lulu.com. Retrieved from [https://www.lulu.com/spotlight/strickland\\_jeffrey/](https://www.lulu.com/spotlight/strickland_jeffrey/)

Strickland, J. (2020). *Data Science Applications using Python and R*. Lulu, Inc. Retrieved from [https://www.lulu.com/spotlight/strickland\\_jeffrey/](https://www.lulu.com/spotlight/strickland_jeffrey/)

Tao, D. (2006). Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Tolosi, L., & Lengauer, T. (2011). Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*. doi:10.1093/bioinformatics/btr300

Tremblay, G. (2004). Optimizing Nearest Neighbour in Random Subspaces using a Multi-Objective Genetic Algorithm. *17th International Conference on Pattern Recognition*, (pp. 208–211).

Tufte, E. (2006). *Beautiful Evidence*. New York: Graphics Press LLC. Retrieved from [www.edwardtufte.com](http://www.edwardtufte.com)



# Index

---

---

## A

- accuracy .. 236, 238, 247, 249, 250, 287, 311, 341, 342, 343, 344, 345  
adjusted R-squared..... 206  
aggregate ..... 90, 91, 95, 149, 150, 183, 261, 264  
AIC ..... *See* Akaike information criterion  
Akaike information criterion..... 204, 220, 221  
analysis questions..... 23, 67, 85, 86, 87, 139, 142, 144, 150, 158, 159  
analytics ..... 4, 103  
ANOVA ..... 209, 210  
API ..... 25, 26, 27, 30, 41, 44, 45, 46  
array..... 3, 4, 5, 23, 43, 185, 242  
artificial neural network ..... xv  
Atlanta.... 252, 253, 254, 255, 256, 259, 260, 261, 262, 263, 264, 265,  
266, 267, 269, 272, 273, 275, 276, 277, 279, 281, 283  
attributes ..... 3, 4, 5, 6, 7, 9, 10, 23, 42, 185, 290, 293

## B

- bagging..... 247, 285, 286, 287, 288, 289, 290, 293  
Baltimore .... 27, 36, 37, 38, 39, 41, 91, 92, 93, 96, 97, 98, 99, 101, 102  
Bayes classifier ..... 289  
Bayesian ..... 209, 212  
Bernoulli..... 211  
bilirubin..... 296, 304, 319, 331, 333, 334, 336, 337, 338  
binary ..... 28, 30, 59, 101, 212, 213, 339  
binomial ..... 211, 212  
binomial distribution..... 211  
Boolean..... 43  
boosting..... 247  
bootstrap aggregating..... 286, 287  
bootstrap samples..... 247, 287, 288, 289

## C

- camera data ..... 36, 37, 38, 39, 40, 41

|                                 |                                                                      |
|---------------------------------|----------------------------------------------------------------------|
| CART .....                      | 285, <i>See</i> classification trees                                 |
| case.....                       | 211, 213                                                             |
| categorical distributions ..... | 213                                                                  |
| categorical variables .....     | 293, 320, 323                                                        |
| categories....                  | 86, 98, 151, 157, 163, 253, 261, 262, 263, 265, 272, 274,            |
|                                 | 276, 320, 323, 327                                                   |
| cbind .....                     | <i>See</i> column-binding                                            |
| Central Limit Theorem.....      | 166, 167, 169                                                        |
| character .....                 | <i>See</i> string                                                    |
| Chi square test.....            | 166                                                                  |
| class..                         | 4, 5, 6, 7, 9, 10, 11, 15, 23, 27, 69, 100, 181, 203, 209, 246, 285, |
|                                 | 290, 293, 323, 329, 340, 346                                         |
| classification.....             | xvi, 247, 285, 287, 291, 292, 294, 322, 340, 346                     |
| classification trees .....      | 248, 293, 294                                                        |
| CLT.....                        | <i>See</i> Central Limit Theorem                                     |
| code chunk .....                | 35, 40, 59, 121, 125, 148, 220, 303, 305                             |
| codebook.....                   | 23, 29, 32, 49, 123                                                  |
| coercion.....                   | 6, 25, 246                                                           |
| collinear.....                  | 216, 218, 221                                                        |
| column headings.....            | <i>See</i> variable names                                            |
| column-binding.....             | 7, 52, 59, 61, 63, 147, 148, 221, 298                                |
| comma delimited file ...        | 36, 37, 38, 39, 40, 45, 57, 59, 61, 63, 124, 125,                    |
|                                 | 140, 244, 253, 254, 264, 269, 339                                    |
| Comparison tests.....           | 165                                                                  |
| complex .....                   | 2, 3, 5, 6                                                           |
| complexity.....                 | 229                                                                  |
| confidence intervals .....      | 84, 162, 163, 165, 180, 181, 182, 183, 231                           |
| configuration control.....      | 103, 120                                                             |
| confusion matrix.....           | 343, 344, 345                                                        |
| control structure.....          | 16, 21                                                               |
| <b>break()</b> .....            | 16, 20                                                               |
| <b>for()</b> .....              | 16, 17, 18, 20, 64                                                   |
| <b>if-else()</b> .....          | 16, 17                                                               |
| <b>if-then-else()</b> .....     | 16                                                                   |
| <b>next()</b> .....             | 16                                                                   |
| <b>repeat()</b> .....           | 16, 20                                                               |
| <b>return()</b> .....           | 16                                                                   |
| <b>while()</b> .....            | 16, 19                                                               |
| correlation.....                | 286, 290                                                             |

|                                                                                                                                                                                                                                                                                              |                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| correlation tests .....                                                                                                                                                                                                                                                                      | 166                                         |
| counts .....                                                                                                                                                                                                                                                                                 | 215, 251, 261, 264, 268                     |
| COVID .....                                                                                                                                                                                                                                                                                  | xvii, 238, 358, 359, 363                    |
| crime data .....                                                                                                                                                                                                                                                                             | 252, 253, 254, 255                          |
| crop damage .....                                                                                                                                                                                                                                                                            | 140, 142, 146, 147, 148, 150, 156, 157, 159 |
| cross-validation .....                                                                                                                                                                                                                                                                       | 237, 245, 248, 250, 287, 347                |
| csv .....                                                                                                                                                                                                                                                                                    | <i>See</i> comma delimited file             |
| culmen depth .....                                                                                                                                                                                                                                                                           | 73, 82                                      |
| culmen length .....                                                                                                                                                                                                                                                                          | 73, 82                                      |
| <b>D</b>                                                                                                                                                                                                                                                                                     |                                             |
| data cleaning .....                                                                                                                                                                                                                                                                          | 23, 25, 26, 27, 28, 71, 125                 |
| data consistency .....                                                                                                                                                                                                                                                                       | 25                                          |
| data frame3, 4, 9, 10, 11, 46, 59, 63, 64, 126, 141, 173, 216, 221, 231, 245, 256, 295, 298, 340, 345                                                                                                                                                                                        |                                             |
| data integrity .....                                                                                                                                                                                                                                                                         | 25, 68                                      |
| data item .....                                                                                                                                                                                                                                                                              | 1                                           |
| data object .....                                                                                                                                                                                                                                                                            | 3                                           |
| data pre-processing .....                                                                                                                                                                                                                                                                    | 23, 29, 30, 87, 123, 125                    |
| data sets2, 23, 27, 29, 30, 31, 32, 33, 34, 37, 39, 40, 41, 45, 46, 51, 52, 53, 56, 58, 59, 67, 90, 93, 95, 97, 98, 120, 123, 126, 131, 137, 140, 141, 147, 148, 185, 186, 190, 216, 218, 221, 236, 239, 243, 244, 245, 246, 247, 248, 250, 260, 288, 289, 290, 293, 295, 322, 323, 339, 354 |                                             |
| data types .....                                                                                                                                                                                                                                                                             | 1, 2, 3, 4, 7, 9                            |
| data wrangling .....                                                                                                                                                                                                                                                                         | <i>See</i> data pre-processing              |
| data.table .....                                                                                                                                                                                                                                                                             | 48, 254, 261                                |
| database .....                                                                                                                                                                                                                                                                               | 23, 25, 32, 50, 108, 140                    |
| decision tree .....                                                                                                                                                                                                                                                                          | 285, 286, 287, 290, 293, 295, 340           |
| decision trees .....                                                                                                                                                                                                                                                                         | <i>See</i> classification trees             |
| degrees of freedom .....                                                                                                                                                                                                                                                                     | 220                                         |
| delimited text files .....                                                                                                                                                                                                                                                                   | 39                                          |
| dependent variable                                                                                                                                                                                                                                                                           |                                             |
| response variable .....                                                                                                                                                                                                                                                                      | 209, 212                                    |
| target variable .....                                                                                                                                                                                                                                                                        | 215, 218                                    |
| descriptive statistics .....                                                                                                                                                                                                                                                                 | 161                                         |
| Deviance Residuals .....                                                                                                                                                                                                                                                                     | 219                                         |
| dimension .....                                                                                                                                                                                                                                                                              | 4, 5, 6, 7, 131, 315                        |
| dirty data .....                                                                                                                                                                                                                                                                             | 25                                          |

|                            |                                                            |
|----------------------------|------------------------------------------------------------|
| dispersion parameter ..... | 228                                                        |
| distributions              |                                                            |
| Bernoulli.....             | 211, 213, 215                                              |
| binomial.....              | 210, 212, 213, 215                                         |
| binomial(logit) .....      | 218                                                        |
| binomial(probit).....      | 218                                                        |
| categorical .....          | 215                                                        |
| exponential .....          | 209, 212, 213, 215                                         |
| exponential (lambda).....  | 166, 167, 169                                              |
| gamma.....                 | 212, 213, 220                                              |
| Gaussian .....             | 179, 218, 219, 220, 221                                    |
| Inverse Gaussian.....      | 220, 221                                                   |
| multinomial .....          | 210, 213, 215                                              |
| normal.....                | 166, 167, 169, 170, 171, 196, 197, 209, 210, 212, 213, 215 |
| Poisson.....               | 210, 211, 212, 213, 215, 218, 220, 221, 222, 226           |
| population .....           | 167                                                        |
| probability.....           | 172, 196                                                   |
| uniform .....              | 167, 168                                                   |
| dplyr package.....         | 50                                                         |

## **E**

|                                 |                                                           |
|---------------------------------|-----------------------------------------------------------|
| EDA.....                        | <i>See</i> exploratory data analysis                      |
| effect size .....               | 240, 241, 242                                             |
| elements.....                   | 4, 5, 7, 11, 17, 39, 40, 42, 103, 172, 212, 215, 257, 265 |
| emissions data.....             | 86, 87, 90, 91, 92, 93, 95, 96, 97, 98, 99, 102           |
| ensemble learning.....          | 246, 247, 285                                             |
| Excel.....                      | 24, 25, 26, 30, 31, 36, 40, 252                           |
| expected value .....            | 210, 212, 213, 215                                        |
| exploratory data analysis.....  | 67, 87, 141                                               |
| exponential distribution.....   | 211, 212                                                  |
| Extensible Markup Language..... | 23, 24, 41, 42, 43                                        |
| extract .....                   | 11, 26, 27, 28, 29, 32, 41, 43, 59, 146, 312              |

## **F**

|                                                                                                                                     |                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| factors . 3, 4, 8, 9, 86, 99, 100, 137, 173, 177, 201, 203, 205, 206, 207,<br>231, 252, 265, 272, 273, 278, 280, 282, 314, 315, 340 |                                                 |
| false negative.....                                                                                                                 | 238                                             |
| false positive.....                                                                                                                 | 237, 238                                        |
| feature.....                                                                                                                        | 50, 103, 232, 289, 292, 293, 321, 322, 323, 349 |
| categorical .....                                                                                                                   | 323                                             |

|                      |                                                  |
|----------------------|--------------------------------------------------|
| numerical.....       | 323                                              |
| feature space.....   | 289                                              |
| Fisher scoring.....  | 220, 228, 229                                    |
| fitted model.....    | 231, 239, 244, 247, 249, 285, 287, 289, 294, 333 |
| fitting.....         | 95, 190, 209, 228, 247                           |
| fitting process..... | 293                                              |
| flat files .....     | 39                                               |
| floating-point ..... | <i>See numeric:double</i>                        |
| for loop .....       | 20, 63                                           |

## G

|                               |                                                                           |
|-------------------------------|---------------------------------------------------------------------------|
| generalized linear model..... | 209, 211, 212, 213, 218                                                   |
| genetic algorithms .....      | xv                                                                        |
| GitHub ....                   | 103, 104, 107, 110, 111, 112, 113, 114, 115, 116, 117, 122, 252, 253, 359 |
| goodness-of-fit.....          | 219                                                                       |

## H

|                       |                                                                   |
|-----------------------|-------------------------------------------------------------------|
| heatmap.....          | <i>See plot heatmap</i>                                           |
| homogeneity.....      | 215                                                               |
| hospital data.....    | 56, 58, 59, 60, 61, 62, 63, 64, 65                                |
| htmlwidgets.....      | 256                                                               |
| hypothesis tests .... | 67, 68, 73, 172, 175, 176, 177, 179, 180, 181, 182, 203, 204, 207 |

## I

|                             |                                                                               |
|-----------------------------|-------------------------------------------------------------------------------|
| if-then-else .....          | 63                                                                            |
| imputation.....             | 77, 78, 79, 80, 81, 125, 134, 136, 310, 311                                   |
| independent variables.....  | 212, 215, 216                                                                 |
| explanatory variables.....  | 212, 215, 219                                                                 |
| linear predictor .....      | 212                                                                           |
| index .....                 | 12, 130, 151                                                                  |
| indexed .....               | 9                                                                             |
| inferential statistics..... | 161, 162, 167                                                                 |
| in-sample error .....       | 239                                                                           |
| instruction list.....       | 33                                                                            |
| interval.....               | 95, 125, 126, 127, 130, 131, 132, 133, 138, 162, 163, 165, 166, 295, 301, 304 |
| interval estimate.....      | 162                                                                           |

|                             |     |
|-----------------------------|-----|
| inverse link function ..... | 230 |
| iterating.....              | 17  |
| iterator.....               | 17  |

## **J**

|                                   |                                         |
|-----------------------------------|-----------------------------------------|
| Javascript Object Navigation..... | 23, 24, 26, 30, 43                      |
| JSON .....                        | <i>See Javascript Object Navigation</i> |

## **K**

|                           |                         |
|---------------------------|-------------------------|
| k-nearest neighbors ..... | 288, 290, 291           |
| knitr package.....        | 122, 140, 171, 305, 327 |

## **L**

|                                                                                                                                                                                |                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| labels 8, 41, 42, 50, 51, 52, 68, 84, 85, 90, 92, 93, 94, 96, 97, 100, 129, 131, 174, 176, 177, 178, 260, 262, 264, 267, 269, 270, 271, 273, 278, 280, 282, 319, 350, 354, 355 |                                   |
| layers .....                                                                                                                                                                   | 255                               |
| leaflet.....                                                                                                                                                                   | 256, 257, 258                     |
| least squares.....                                                                                                                                                             | 209, 215, 229                     |
| levels of measurements .....                                                                                                                                                   | 163                               |
| library. 41, 87, 138, 167, 173, 175, 178, 185, 232, 241, 254, 257, 259, 295, 296, 300, 344, 347, 350, 351, 353                                                                 |                                   |
| linear combination.....                                                                                                                                                        | 209, 210, 212                     |
| linear predictor .....                                                                                                                                                         | 212, 213                          |
| linear regression model.....                                                                                                                                                   | 186, 210                          |
| link function.....                                                                                                                                                             | 209, 211, 212, 213, 215, 221      |
| identity.....                                                                                                                                                                  | 210                               |
| log.....                                                                                                                                                                       | 210                               |
| logit.....                                                                                                                                                                     | 210                               |
| lists . 3, 4, 7, 9, 10, 11, 12, 14, 17, 38, 47, 53, 59, 64, 99, 100, 141, 165, 232, 245, 249, 254, 255, 257, 261, 265, 266, 268, 273, 329, 340                                 |                                   |
| Literate (Statistical) Programming.....                                                                                                                                        | 121                               |
| Loess smoother .....                                                                                                                                                           | 288, 319, 320                     |
| logicals .....                                                                                                                                                                 | 1, 2                              |
| logistic regression .....                                                                                                                                                      | 209, 212, 213, 215, 219, 247, 293 |
| logit .....                                                                                                                                                                    | 211                               |
| Log-Linear .....                                                                                                                                                               | 211                               |
| Log-Linear regression .....                                                                                                                                                    | 209                               |
| Log-Linear Regression .....                                                                                                                                                    | 210                               |

|                                                                   |                                                                   |
|-------------------------------------------------------------------|-------------------------------------------------------------------|
| log-odds.....                                                     | 211                                                               |
| Longley data set.....                                             | 216                                                               |
| lookup tables.....                                                | 147, 148                                                          |
| loop body .....                                                   | 19                                                                |
| Los Angeles .....                                                 | 97, 98, 99, 101, 102                                              |
| lurking variable.....                                             | 73                                                                |
| <b>M</b>                                                          |                                                                   |
| machine learning.....                                             | 23, 27, 185, 235, 236, 246, 287, 321, 322, 343                    |
| map-markers.....                                                  | 255, 257                                                          |
| markdown                                                          | 32, 102, 105, 106, 122, 176, 271, 282, 302, 303, 305, 306,        |
| 327                                                               |                                                                   |
| Markdown Note .....                                               | xviii, 38, 102, 112, 135, 157, 170, 174, 271, 282,                |
| 302, 303, 305, 306, 327                                           |                                                                   |
| matrices.....                                                     | 3, 4, 6, 7, 9, 10, 13, 256                                        |
| matrix                                                            | 5, 7, 9, 10, 13, 26, 100, 216, 217, 218, 221, 238, 250, 251, 330, |
| 346                                                               |                                                                   |
| maximum likelihood estimation .....                               | 209, 212, 215                                                     |
| mean ....                                                         | 44, 48, 52, 53, 54, 86, 125, 126, 127, 128, 129, 130, 131, 133,   |
| 134, 161, 162, 163, 164, 165, 167, 168, 169, 170, 171, 172, 173,  |                                                                   |
| 180, 182, 187, 191, 193, 194, 195, 206, 209, 211, 212, 213, 228,  |                                                                   |
| 248, 253, 287, 289, 294, 315, 342, 347                            |                                                                   |
| measurements .....                                                | 28, 50, 52, 124, 221                                              |
| median .....                                                      | 53, 69, 127, 128, 129, 136, 137, 165, 227                         |
| melt .....                                                        | 298                                                               |
| MICE.....                                                         | <i>See</i> Multivariate Imputation via Chained Equations          |
| minimal depth.....                                                | 314, 315, 317, 318, 330                                           |
| missing values....                                                | 4, 6, 10, 11, 14, 15, 25, 59, 60, 61, 62, 63, 64, 65, 76,         |
| 77, 78, 80, 89, 125, 127, 130, 131, 133, 134, 137, 145, 207, 208, |                                                                   |
| 244, 249, 296, 297, 298, 301, 309, 316, 342, 343                  |                                                                   |
| MLE .....                                                         | <i>See</i> maximum likelihood estimation                          |
| model                                                             |                                                                   |
| effects .....                                                     | 218                                                               |
| linear .....                                                      | 209, 211                                                          |
| parameters.....                                                   | 209, 212, 215                                                     |
| predictive .....                                                  | 210                                                               |
| multidimensional .....                                            | 5                                                                 |
| multinomial logistic regression.....                              | 213                                                               |
| Multivariate Imputation via Chained Equations .....               | 74                                                                |

|                                                                            |                                                        |
|----------------------------------------------------------------------------|--------------------------------------------------------|
| mutate .....                                                               | 259, 263, 264, 268, 273, 282                           |
| <b>N</b>                                                                   |                                                        |
| NA .....                                                                   | <i>See</i> missing values                              |
| naïve Bayes .....                                                          | 293                                                    |
| NaN.....                                                                   | <i>See</i> Not-a-Number                                |
| noise.....                                                                 | 239, 259, 311                                          |
| nominal.....                                                               | 163, 166                                               |
| normal distribution.....                                                   | 210, 215                                               |
| normalized .....                                                           | 225, 251, 275, 276, 277, 278, 281, 283, 293, 300       |
| Not-a-Number.....                                                          | 4, 10, 11, 15, 207, 208                                |
| null deviance .....                                                        | 220                                                    |
| numeric                                                                    |                                                        |
| double.....                                                                | 1, 2, 3, 43, 156, 193, 211                             |
| integers .....                                                             | 1, 2, 3, 5, 6, 8, 10, 11, 12, 15, 26, 48, 61, 134, 147 |
| <b>O</b>                                                                   |                                                        |
| objects.1, 3, 4, 5, 6, 9, 10, 11, 15, 23, 28, 129, 197, 256, 327, 328, 349 |                                                        |
| OLS.....                                                                   | 186                                                    |
| operations                                                                 |                                                        |
| basic .....                                                                | 1, 67                                                  |
| mathematical.....                                                          | 4, 10, 15                                              |
| plyr like.....                                                             | 48                                                     |
| optimization .....                                                         | 285                                                    |
| ordinal.....                                                               | 163, 165, 166                                          |
| outcome-of-care-measures data .....                                        | 57, 59, 61, 63                                         |
| outcomes.....                                                              | 213                                                    |
| out-of-sample error.....                                                   | 239                                                    |
| overdispersion.....                                                        | 215                                                    |
| overfitting .....                                                          | 247, 287, 289, 294                                     |
| <b>P</b>                                                                   |                                                        |
| parameter.40, 161, 162, 168, 170, 195, 213, 215, 220, 223, 228, 229,       |                                                        |
| 286, 340                                                                   |                                                        |
| parse .....                                                                | 26, 28                                                 |
| partial dependence .....                                                   | 321, 322, 323, 329, 335, 337, 338                      |
| partial dependence plot.....                                               | <i>See</i> PDP                                         |
| PDP.....                                                                   | 322, 323                                               |
| Pearson's r.....                                                           | 166                                                    |

|                                                                                     |                                                                                          |
|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| penguin dataset.....                                                                | 75                                                                                       |
| <b>plot</b>                                                                         |                                                                                          |
| abline .....                                                                        | 129, 131, 132, 170, 171, 187, 188, 193, 194, 195, 243                                    |
| barplot.....                                                                        | 99, 150, 153, 157                                                                        |
| boxplot.....                                                                        | 70, 175, 176, 177, 179, 250, 320                                                         |
| heatmap.....                                                                        | 264, 266, 267, 274                                                                       |
| histogram.....                                                                      | 127, 129, 134, 135, 136, 168, 170, 171, 186, 261, 350, 351, 353, 355, 356                |
| line-point.....                                                                     | 95                                                                                       |
| minimal depth plots .....                                                           | 332                                                                                      |
| Normal Q-Q plot.....                                                                | 197, 206                                                                                 |
| partial dependence coplots.....                                                     | 335, 336                                                                                 |
| partial dependence plots .....                                                      | 321, 322, 323, 325, 329                                                                  |
| partial plots.....                                                                  | 327                                                                                      |
| Poisson fitted plots .....                                                          | 233                                                                                      |
| residual plots.....                                                                 | 196, 206                                                                                 |
| residuals-leverage plots .....                                                      | 197                                                                                      |
| sample size plots .....                                                             | 242                                                                                      |
| scale-location plots.....                                                           | 197                                                                                      |
| scatterplots .....                                                                  | 130, 217, 306                                                                            |
| variable dependence plots .....                                                     | 319, 320                                                                                 |
| <b>point estimate.....</b>                                                          | 162, 165                                                                                 |
| Poisson regression.....                                                             | 209, 226, 228, 229, 231                                                                  |
| <b>population.....</b>                                                              | 216                                                                                      |
| population parameters .....                                                         | 171, 172                                                                                 |
| popups .....                                                                        | 257, 258                                                                                 |
| posterior probability .....                                                         | 291                                                                                      |
| predicted outcome.....                                                              | 321, 322, 323                                                                            |
| predictors .....                                                                    | 190, 194, 195, 196, 201, 204, 210, 212, 213, 235, 236, 237, 239, 248, 289, 290, 292, 357 |
| probability85, 162, 195, 197, 211, 212, 213, 215, 235, 236, 275, 300, 301, 304, 322 |                                                                                          |
| property damage ....                                                                | 139, 140, 142, 146, 147, 148, 149, 154, 155, 156, 159                                    |
| proportion tests .....                                                              | 241                                                                                      |
| <b><i>Q</i></b>                                                                     |                                                                                          |
| qualitative data.....                                                               | 28, 305                                                                                  |
| quantitative data.....                                                              | 28, 190, 315                                                                             |

## R

### R dataset

|                               |                              |
|-------------------------------|------------------------------|
| <code>airquality</code> ..... | 222                          |
| <code>fgl</code> .....        | 345                          |
| <code>longley</code> .....    | 216                          |
| <code>mtcars</code> .....     | 202, 203, 204, 205, 207, 208 |
| <code>pbc</code> .....        | 296                          |

### R function

|                                         |                                                                                                                                                 |
|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>abline()</code> .....             | 129, 132, 191, 194, 197, 243                                                                                                                    |
| <code>addMarkers()</code> .....         | 258                                                                                                                                             |
| <code>addProviderTiles()</code> .....   | 258                                                                                                                                             |
| <code>addTiles()</code> .....           | 258                                                                                                                                             |
| <code>aes</code> ....                   | 93, 95, 96, 97, 99, 150, 153, 156, 174, 175, 177, 178, 232, 260, 262, 263, 264, 266, 269, 271, 274, 276, 279, 280, 282, 299, 305, 306, 355, 356 |
| <code>aggr()</code> .....               | 77                                                                                                                                              |
| <code>aggregate()</code> .....          | 90, 91, 93, 95, 97, 146, 150                                                                                                                    |
| <code>arrange()</code> .....            | 146, 259, 263, 273                                                                                                                              |
| <code>as.character()</code> .....       | 254                                                                                                                                             |
| <code>as.complex()</code> .....         | 6                                                                                                                                               |
| <code>as.data.frame()</code> .....      | 60, 61, 63, 134, 191                                                                                                                            |
| <code>as.factor()</code> .....          | 138, 178, 203                                                                                                                                   |
| <code>as.logical()</code> .....         | 6                                                                                                                                               |
| <code>as.numeric()</code> .....         | 6                                                                                                                                               |
| <code>as.numeric()</code> .....         | 6, 25, 58, 64, 147, 148, 150, 189                                                                                                               |
| <code>as.vector()</code> .....          | 189, 191                                                                                                                                        |
| <code>boxplot()</code> .....            | 208                                                                                                                                             |
| <code>bwplot()</code> .....             | 250                                                                                                                                             |
| <code>c()</code> .....                  | 5                                                                                                                                               |
| <code>cat()</code> .....                | 125, 126, 128, 131, 133, 168, 240                                                                                                               |
| <code>cbind()</code> .....              | 7, 52, 61, 63, 147                                                                                                                              |
| <code>ceiling()</code> .....            | 222                                                                                                                                             |
| <code>class()</code> .....              | 2, 3                                                                                                                                            |
| <code>coef()</code> .....               | 194                                                                                                                                             |
| <code>colnames()</code> .....           | 246                                                                                                                                             |
| <code>column_spec()</code> .....        | 148, 152, 155, 158                                                                                                                              |
| <code>combine.gg_partial()</code> ..... | 327                                                                                                                                             |
| <code>complete()</code> .....           | 80                                                                                                                                              |
| <code>complete.case()</code> .....      | 127                                                                                                                                             |
| <code>confusion_matrix()</code> .....   | 250                                                                                                                                             |
| <code>confusionMatrix()</code> .....    | 344                                                                                                                                             |

|                    |                                                                                        |
|--------------------|----------------------------------------------------------------------------------------|
| coplot             | 82                                                                                     |
| cor()              | 194                                                                                    |
| count()            | 125                                                                                    |
| curl()             | 37, 39, 45, 50                                                                         |
| cut()              | 304, 334                                                                               |
| data()             | 186                                                                                    |
| data.frame()       | 10, 64, 148, 169, 171, 231, 261                                                        |
| data.matrix()      | 9, 10, 218                                                                             |
| data.table()       | 46, 49, 265, 273                                                                       |
| datatable()        | 254, 261, 270                                                                          |
| date()             | 38                                                                                     |
| density()          | 199                                                                                    |
| deviance()         | 228                                                                                    |
| dim()              | 6, 126, 131, 173                                                                       |
| dir.create()       | 35                                                                                     |
| download.file()    | 35, 36, 37, 39, 41, 45, 88, 125, 140                                                   |
| dplyr()            | 259                                                                                    |
| element_text()     | 94, 96, 101, 153, 174, 175, 176, 177, 178, 266, 271, 274, 276, 279, 280, 282, 356, 357 |
| errorest()         | 347                                                                                    |
| expression()       | 90, 92, 93, 96, 97, 291                                                                |
| facet_grid()       | 100, 101                                                                               |
| facet_wrap()       | 178, 274, 277, 279                                                                     |
| factor()           | 8, 265, 270, 273, 278, 280                                                             |
| family()           | 224                                                                                    |
| file.exists()      | 35, 50, 88, 124                                                                        |
| filter()           | 98, 99                                                                                 |
| fromJSON()         | 24                                                                                     |
| geom_bar()         | 101, 150, 153, 156, 262, 263                                                           |
| geom_boxplot()     | 175, 177, 178                                                                          |
| geom_line()        | 93, 95, 232, 260, 264, 269                                                             |
| geom_point()       | 93, 95, 174, 232, 299, 305, 306                                                        |
| geom_rug()         | 305, 306                                                                               |
| geom_smooth()      | 260, 264                                                                               |
| geom_text_repel()  | 262                                                                                    |
| geom_title()       | 266, 274, 279, 282                                                                     |
| getwd()            | 34, 45, 57, 87, 124                                                                    |
| gg_interaction()   | 330                                                                                    |
| gg_minimal_depth() | 316                                                                                    |
| gg_minimal_vimp()  | 317                                                                                    |
| gg_survival()      | 301                                                                                    |

|                              |                                                                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gg_variable()</code>   | 312, 319, 333                                                                                                                                                               |
| <code>gg_vimp()</code>       | 312, 313                                                                                                                                                                    |
| <code>ggplot()</code>        | 93, 95, 97, 99, 150, 153, 156, 174, 177, 178, 232, 260, 262, 263, 264, 266, 269, 271, 274, 276, 279, 280, 282, 299, 305, 306                                                |
| <code>ggsave()</code>        | 101                                                                                                                                                                         |
| <code>ggtitle()</code>       | 93, 96, 150, 153, 156, 355, 356                                                                                                                                             |
| <code>glm()</code>           | 209, 219, 220, 222                                                                                                                                                          |
| <code>gradientn()</code>     | 321                                                                                                                                                                         |
| <code>grepl()</code>         | 263, 268                                                                                                                                                                    |
| <code>grid.arrange()</code>  | 157, 300                                                                                                                                                                    |
| <code>group_by()</code>      | 53, 127, 128, 134, 138, 173, 202, 259, 263, 265, 270, 272, 278, 280                                                                                                         |
| <code>groupby()</code>       | 99                                                                                                                                                                          |
| <code>gsub()</code>          | 52, 99                                                                                                                                                                      |
| <code>gsubns()</code>        | 53                                                                                                                                                                          |
| <code>head()</code>          | 48, 125, 133, 143, 144, 297                                                                                                                                                 |
| <code>hist()</code>          | 58, 129, 168, 186, 187                                                                                                                                                      |
| <code>ifelse()</code>        | 309                                                                                                                                                                         |
| <code>inner_join()</code>    | 99                                                                                                                                                                          |
| <code>is.na()</code>         | 11, 15, 62, 64, 125, 127, 133                                                                                                                                               |
| <code>kable()</code>         | 266, 276, 278                                                                                                                                                               |
| <code>kable_classic()</code> | 143, 144, 148, 149, 152, 154, 155, 158                                                                                                                                      |
| <code>labs()</code>          | 93, 97, 101, 174, 175, 177, 178, 260, 264, 267, 269, 271, 274, 276, 279, 280, 282, 301, 303, 304, 305, 307, 314, 319, 320, 321, 326, 327, 328, 329, 331, 333, 335, 336, 337 |
| <code>lapply()</code>        | 327                                                                                                                                                                         |
| <code>leaflet()</code>       | 257, 258                                                                                                                                                                    |
| <code>legend()</code>        | 131                                                                                                                                                                         |
| <code>length()</code>        | 242                                                                                                                                                                         |
| <code>library()</code>       | 24, 42, 43, 46, 57, 87, 124                                                                                                                                                 |
| <code>list()</code>          | 7, 12                                                                                                                                                                       |
| <code>list.files()</code>    | 38                                                                                                                                                                          |
| <code>lm()</code>            | 204, 205                                                                                                                                                                    |
| <code>manipulate()</code>    | 191                                                                                                                                                                         |
| <code>matrix()</code>        | 6, 13, 18                                                                                                                                                                   |
| <code>max()</code>           | 131                                                                                                                                                                         |
| <code>md.pattern()</code>    | 76                                                                                                                                                                          |
| <code>mean()</code>          | 47, 125, 128, 136, 168, 169, 173, 187, 194                                                                                                                                  |
| <code>median()</code>        | 128, 129, 136                                                                                                                                                               |
| <code>melt()</code>          | 305                                                                                                                                                                         |
| <code>mice()</code>          | 78                                                                                                                                                                          |

|                                      |                                                                                     |
|--------------------------------------|-------------------------------------------------------------------------------------|
| <code>mse()</code>                   | 193                                                                                 |
| <code>mutate()</code>                | 254, 259, 263, 270, 273, 276, 278, 280, 282                                         |
| <code>na.string()</code>             | 40                                                                                  |
| <code>names()</code>                 | 141                                                                                 |
| <code>nrow()</code>                  | 6                                                                                   |
| <code>nrows</code>                   | 40                                                                                  |
| <code>order()</code>                 | 62, 64, 150                                                                         |
| <code>pairs()</code>                 | 219                                                                                 |
| <code>par()</code>                   | 186, 346                                                                            |
| <code>paste()</code>                 | 270, 334, 340                                                                       |
| <code>plot()</code>                  | 90, 91, 188, 197, 216, 303, 313                                                     |
| <code>plot_confusion_matrix()</code> | 250                                                                                 |
| <code>points()</code>                | 195                                                                                 |
| <code>predict()</code>               | 193, 230, 231, 250, 311, 343, 344                                                   |
| <code>predict.confidence()</code>    | 231                                                                                 |
| <code>print()</code>                 | 2, 3, 17, 18, 19, 20, 24, 60, 125, 143, 144, 145, 297, 299, 310, 311, 316, 330, 345 |
| <code>pwr.2p.test()</code>           | 241                                                                                 |
| <code>pwr.2p2n.test()</code>         | 241                                                                                 |
| <code>pwr.r.test()</code>            | 242                                                                                 |
| <code>pwr.t.test()</code>            | 240                                                                                 |
| <code>pwr.t2n.test()</code>          | 240                                                                                 |
| <code>qqline()</code>                | 197                                                                                 |
| <code>qqnorm()</code>                | 197                                                                                 |
| <code>quantile()</code>              | 202                                                                                 |
| <code>quantile_pts()</code>          | 334                                                                                 |
| <code>randomForest()</code>          | 341, 345                                                                            |
| <code>range()</code>                 | 242                                                                                 |
| <code>rankall()</code>               | 65                                                                                  |
| <code>rbind()</code>                 | 7, 51, 64, 194, 221, 246                                                            |
| <code>read.csv()</code>              | 40, 45, 57, 59, 61, 63, 125, 140, 244, 254                                          |
| <code>read.frame()</code>            | 49                                                                                  |
| <code>read.table</code>              | 10                                                                                  |
| <code>read.table()</code>            | 39, 40, 49, 50, 51                                                                  |
| <code>read.xlsx()</code>             | 41                                                                                  |
| <code>readRDS()</code>               | 88                                                                                  |
| <code>require()</code>               | 216, 218, 248, 294                                                                  |
| <code>resamples()</code>             | 249                                                                                 |
| <code>resid()</code>                 | 197                                                                                 |
| <code>residuals()</code>             | 224, 225, 226                                                                       |
| <code>rexp()</code>                  | 168                                                                                 |

|                                            |                                                                                                                                                             |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rfsrc()</code> .....                 | 309                                                                                                                                                         |
| <code>rm()</code> .....                    | 246                                                                                                                                                         |
| <code>rollmean()</code> .....              | 131                                                                                                                                                         |
| <code>round()</code> .....                 | 170                                                                                                                                                         |
| <code>sample()</code> .....                | 222                                                                                                                                                         |
| <code>scale_colour_discrete()</code> ..... | 93                                                                                                                                                          |
| <code>scale_fill_brewer()</code> .....     | 299, 307                                                                                                                                                    |
| <code>scale_fill_gradient()</code> .....   | 150, 156, 267, 271, 279                                                                                                                                     |
| <code>scale_shape()</code> .....           | 335                                                                                                                                                         |
| <code>scale_y_continuous()</code> .....    | 101                                                                                                                                                         |
| <code>select()</code> .....                | 98, 99, 127, 131, 134                                                                                                                                       |
| <code>set.seed()</code> .....              | 48, 222, 248, 345, 347                                                                                                                                      |
| <code>setwd()</code> .....                 | 34, 57                                                                                                                                                      |
| <code>sort()</code> .....                  | 147, 261                                                                                                                                                    |
| <code>sprintf()</code> .....               | 171, 204                                                                                                                                                    |
| <code>str()</code> .....                   | 54, 89, 99, 126, 142, 173                                                                                                                                   |
| <code>subset()</code> .....                | 91, 93, 181, 182                                                                                                                                            |
| <code>sum()</code> .....                   | 47, 126, 133, 194                                                                                                                                           |
| <code>summarise()</code> .....             | 99, 202                                                                                                                                                     |
| <code>summarize()</code> .....             | 128, 134, 138, 259, 265, 273, 278                                                                                                                           |
| <code>summarize_all()</code> .....         | 53                                                                                                                                                          |
| <code>summary()</code> .....               | 88, 126, 137, 145, 200, 204, 207, 218, 222                                                                                                                  |
| <code>system.time()</code> .....           | 49                                                                                                                                                          |
| <code>t.test()</code> .....                | 180, 182                                                                                                                                                    |
| <code>table()</code> .....                 | 339                                                                                                                                                         |
| <code>tempfile()</code> .....              | 49, 140                                                                                                                                                     |
| <code>theme()</code> .....                 | 94, 96, 97, 100, 101, 153, 174, 176, 177, 178, 262, 264, 266, 269, 271, 274, 276, 279, 280, 282, 301, 303, 314, 319, 320, 327, 328, 330, 331, 333, 335, 356 |
| <code>train()</code> .....                 | 248                                                                                                                                                         |
| <code>trainControl()</code> .....          | 248                                                                                                                                                         |
| <code>unique()</code> .....                | 146                                                                                                                                                         |
| <code>unzip()</code> .....                 | 50, 88, 125, 254                                                                                                                                            |
| <code>var.select()</code> .....            | 315                                                                                                                                                         |
| <code>var.test()</code> .....              | 179                                                                                                                                                         |
| <code>varImpPlot()</code> .....            | 342                                                                                                                                                         |
| <code>vector()</code> .....                | 5                                                                                                                                                           |
| <code>write.table()</code> .....           | 49, 54                                                                                                                                                      |
| <code>xlab()</code> .                      | 58, 90, 92, 93, 96, 129, 131, 135, 138, 150, 153, 156, 170, 189, 191, 208, 243, 351                                                                         |
| <code>xmlSApply()</code> .....             | 43                                                                                                                                                          |

`ylab()` ... 90, 92, 93, 96, 129, 131, 135, 138, 150, 153, 156, 157, 189, 191, 208, 243

## R package

|                                    |                                                               |
|------------------------------------|---------------------------------------------------------------|
| <code>BiocManager</code> .....     | 248                                                           |
| <code>caret</code> .....           | 343, 344                                                      |
| <code>colorspace</code> .....      | 294, 295, 305, 321                                            |
| <code>data sets</code> .....       | 201                                                           |
| <code>data.table</code> .....      | 46, 253, 298                                                  |
| <code>dplyr</code> .....           | 87, 124, 202, 253                                             |
| <code>DT</code> .....              | 46, 47, 48, 253, 254, 261                                     |
| <code>e1071</code> .....           | 344                                                           |
| <code>ggformula</code> .....       | 87                                                            |
| <code>ggplot2</code> .....         | 87, 93, 140, 174, 175, 177, 178, 232, 260, 262, 269, 295, 305 |
| <code>ggplotify</code> .....       | 140                                                           |
| <code>ggRandomForest</code> .....  | 296                                                           |
| <code>ggRandomForests</code> ..... | 335, 337                                                      |
| <code>ggrepel</code> .....         | 253, 262                                                      |
| <code>graphics</code> .....        | 216, 218                                                      |
| <code>gridExtra</code> .....       | 140, 157, 232, 299, 300                                       |
| <code>HTTR</code> .....            | 44                                                            |
| <code>ipred</code> .....           | 347                                                           |
| <code>jsonlite</code> .....        | 43                                                            |
| <code>knitr</code> .....           | 204, 265, 266, 276, 278                                       |
| <code>lattice()</code> .....       | 138                                                           |
| <code>leaflet</code> .....         | 253, 257                                                      |
| <code>logicFS</code> .....         | 248                                                           |
| <code>manipulate</code> .....      | 187                                                           |
| <code>mice</code> .....            | 75, 78                                                        |
| <code>midForest</code> .....       | 75                                                            |
| <code>pwr</code> .....             | 241, 242                                                      |
| <code>randomForest</code> .....    | 294                                                           |
| <code>randomForestSRC</code> ..... | 296                                                           |
| <code>RColorBrewer</code> .....    | 87                                                            |
| <code>readr</code> .....           | 57, 124, 140, 253, 254                                        |
| <code>reshape</code> .....         | 294, 295, 298                                                 |
| <code>rjson</code> .....           | 24                                                            |
| <code>rmarkdown</code> .....       | 102, 256, 302                                                 |
| <code>scales</code> .....          | 260                                                           |
| <code>shiny</code> .....           | 361                                                           |
| <code>stats</code> .....           | 167, 216, 218                                                 |
| <code>tidyverse</code> .....       | 298                                                           |

|                              |                                                                           |
|------------------------------|---------------------------------------------------------------------------|
| <b>UsingR</b>                | 185, 186                                                                  |
| <b>VIM</b>                   | 75                                                                        |
| <b>XLConnect</b>             | 41                                                                        |
| <b>xlsx</b>                  | 41                                                                        |
| <b>XML</b>                   | 42                                                                        |
| <b>zoo</b>                   | 124, 131                                                                  |
| random forest                | 247, 251, 285, 286, 290, 292, 293, 294, 295, 310, 338, 341, 343, 346, 348 |
| random subspace              | 285, 290                                                                  |
| randomized node optimization | 285                                                                       |
| ratio                        | 163, 164, 165, 166, 180, 181, 202                                         |
| raw data                     | 2, 23, 25, 27, 28, 29, 30, 32, 33, 34, 124, 252                           |
| rbind                        | <i>See</i> row-binding                                                    |
| read.csv                     | 9, 10, 40, 339, 353                                                       |
| read.table                   | 9, 353                                                                    |
| recursive partitioning       | 247                                                                       |
| regression model             | 322                                                                       |
| regression tree              | 286, 287                                                                  |
| regression trees             | 247, 248, 286, 294                                                        |
| regressor                    | 186                                                                       |
| relative path                | 34                                                                        |
| repeat loop                  | 20                                                                        |
| replacement                  | 95, 105, 247, 286, 287, 291, 294, 295                                     |
| replication                  | 119, 120                                                                  |
| repositories                 | 103, 104, 105, 107, 109, 110, 111, 114, 115, 116, 122                     |
| residual deviance            | 220                                                                       |
| response variable            | 209, 210, 211, 290                                                        |
| row-binding                  | 7, 51, 64, 194, 220, 221, 246, 298                                        |
| <b>S</b>                     |                                                                           |
| sample mean                  | 161, 162, 165, 167, 168, 169, 170, 188                                    |
| sample size                  | 163, 240, 241, 242, 289, 315                                              |
| sample standard deviation    | 169                                                                       |
| sample statistics            | 171                                                                       |
| scripting                    | 122                                                                       |
| scripts                      | 26, 29, 33, 34, 35, 110, 111, 115, 116, 122, 123, 349, 351, 353           |
| seasonality                  | 259, 260                                                                  |
| sensitivity                  | 238, 344, 345                                                             |
| server function              | 349, 350, 351                                                             |

|                                                                           |                                                            |
|---------------------------------------------------------------------------|------------------------------------------------------------|
| set-aside data .....                                                      | 237                                                        |
| severe weather data.....                                                  | <i>See</i> storm data                                      |
| SGD .....                                                                 | <i>See</i> stochastic gradient descent                     |
| Shiny .....                                                               | 305, 306, 349, 350, 351, 352, 353, 354, 355                |
| signal - noise.....                                                       | 239                                                        |
| Simpson's Paradox.....                                                    | 73, 81, 83                                                 |
| Spearman's r.....                                                         | 166                                                        |
| specificity.....                                                          | 238, 344, 345                                              |
| split.....                                                                | 290, 310, 314, 315, 339, 346                               |
| splitting.....                                                            | 285                                                        |
| SQL.....                                                                  | <i>See</i> Structured Query Language                       |
| stochastic gradient descent.....                                          | 247                                                        |
| storm data .....                                                          | 139, 140, 141, 142, 143, 144, 146, 147, 150, 156           |
| string.....                                                               | 2, 4, 9, 43, 232, 244, 258, 264, 269                       |
| structure 26, 30, 34, 41, 42, 43, 49, 68, 71, 89, 99, 126, 141, 142, 144, | 215, 292, 351                                              |
| Structured Query Language .....                                           | 27                                                         |
| subsetted.....                                                            | 13                                                         |
| subsetting .....                                                          | 11, 13, 14                                                 |
| summarization .....                                                       | 30                                                         |
| supervised learning.....                                                  | 285                                                        |
| survival....                                                              | 300, 301, 304, 309, 319, 320, 321, 333, 334, 337, 338, 347 |
| <b>T</b>                                                                  |                                                            |
| tags .....                                                                | 42                                                         |
| temporal trends .....                                                     | 252                                                        |
| test sets.....                                                            | 50, 51, 236, 244, 246, 293, 300                            |
| tested model.....                                                         | 19, 237                                                    |
| text analytics .....                                                      | xv                                                         |
| text files .....                                                          | 39                                                         |
| tidy data.....                                                            | 23, 25, 29, 30, 31, 32, 33, 53                             |
| tidy format.....                                                          | 26                                                         |
| tooth growth data.....                                                    | 172, 173, 175, 180, 183                                    |
| trained model.....                                                        | 227, 228, 237, 248                                         |
| training data .....                                                       | <i>See</i> training sets                                   |
| training objects.....                                                     | 291                                                        |
| training sets.....                                                        | 51, 222, 235, 236, 243, 244, 245, 246, 247, 248, 285,      |
| 286, 287, 290, 291, 292, 293, 294, 322, 338                               |                                                            |

|                                                                                                      |                                                                      |
|------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| transformations .....                                                                                | 206, 215, 245, 246                                                   |
| trend .....                                                                                          | 95, 96, 97, 190, 259, 260, 268, 320                                  |
| true negative .....                                                                                  | 237, 238                                                             |
| true positive.....                                                                                   | 237, 238                                                             |
| t-test.....                                                                                          | 165, 172, 179, 180, 181, 182, 203, 204, 240                          |
| one-sample.....                                                                                      | 240                                                                  |
| paired .....                                                                                         | 240                                                                  |
| Twitter .....                                                                                        | 25, 26, 27, 31, 43, 44                                               |
| <b><i>U</i></b>                                                                                      |                                                                      |
| underfitting.....                                                                                    | 247                                                                  |
| units of measure .....                                                                               | 32                                                                   |
| unzip .....                                                                                          | 88, 124, 125, 140, 253                                               |
| user interface .....                                                                                 | 349                                                                  |
| <b><i>V</i></b>                                                                                      |                                                                      |
| validation.....                                                                                      | 119, 237, 239, 244, 339, 340, 344                                    |
| validation sample .....                                                                              | 344                                                                  |
| validation set.....                                                                                  | 237, 239, 244                                                        |
| variable importance .....                                                                            | <i>See</i> VIMP                                                      |
| variable names .....                                                                                 | 9, 31, 32, 52                                                        |
| variance..                                                                                           | 167, 170, 171, 172, 181, 209, 212, 215, 218, 228, 247, 285, 286, 287 |
| vectors...3, 4, 5, 6, 7, 8, 10, 13, 17, 25, 59, 61, 127, 141, 203, 215, 221, 265, 290, 311, 322, 347 |                                                                      |
| verification .....                                                                                   | 2, 45, 119                                                           |
| VIMP.....                                                                                            | 311, 312, 313, 314, 315, 317, 318                                    |
| <b><i>W</i></b>                                                                                      |                                                                      |
| web scraping .....                                                                                   | 30, 41                                                               |
| while loops .....                                                                                    | 19                                                                   |
| While loops .....                                                                                    | 19                                                                   |
| widget.....                                                                                          | 256, 261                                                             |
| working directory .....                                                                              | 34, 39, 50, 87, 88, 124, 125, 140, 353                               |
| <b><i>X</i></b>                                                                                      |                                                                      |
| XML .....                                                                                            | <i>See</i> Extensible Markup Language                                |



