

Statistical Methods for Test and Evaluation,  
Volume 2: *Machine Learning Primer using R*

by Jeffrey Strickland, Ph.D.



Statistical Methods for Test and Evaluation, Volume 2: Machine Learning Primer using R

**By Jeffrey Strickland, Ph.D.**

Machine Learning Primer © 2024 by Jeffrey Strickland, Ph.D. is licensed under CC BY-NC-SA 4.0 

ISBN: 978-1-304-26226-4



# Table of Contents

<b>1</b>	<b>DESCRIBING DATA .....</b>	<b>1</b>
1.1	DESCRIBING PLANT GROWTH .....	1
1.2	ANALYSIS OF VARIANCE .....	5
1.3	EXPERIMENTAL DESIGNS .....	12
<b>2</b>	<b>COMPARING MEAN VALUES .....</b>	<b>17</b>
2.1	EFFECTS OF DOSAGE LEVELS AND SUPPLEMENTS ON TOOTH GROWTH ....	17
2.2	INTRODUCTION .....	17
2.3	ASSUMPTIONS .....	17
2.4	EXPLORATORY DATA ANALYSIS (EDA) .....	17
2.4.1	<i>ToothGrowth Description</i> .....	18
2.4.2	<i>ToothGrowth Summary</i> .....	18
2.4.3	<i>Data Visualization</i> .....	18
2.4.4	<i>Boxplot Tooth Growth Factors</i> .....	19
2.5	HYPOTHESIS TEST .....	22
2.5.1	<i>Hypothesis Test on the Effect of Supplement Types</i> .....	22
2.5.2	<i>Hypothesis Test on the Effect of Dose Types</i> .....	23
2.5.3	<i>Suplement as a Factor within Dose Levels</i> .....	24
2.6	SUMMARY .....	25
<b>3</b>	<b>THE CENTRAL LIMIT THEOREM .....</b>	<b>27</b>
3.1	OVERVIEW .....	27
3.2	SIMULATIONS .....	27
3.2.1	<i>The Exponential Distribution</i> .....	27
3.2.2	<i>Simulation of the Exponential(lambda) distribution</i> .....	27
3.2.3	<i>Set up the Simulation</i> .....	27
3.3	APPLICATION OF THE CLT .....	28
3.3.1	<i>The Normal Distribution, Normal(mu,sigma)</i> .....	29
3.3.2	<i>Normal Random Variates</i> .....	29
3.4	THE MEAN VALUE.....	29
3.5	THE VARIANCE.....	30
3.6	HISTOGRAM APPROXIMATION OF THE NORMAL DISTRIBUTION .....	30
3.7	BOXPLOT AND SAMPLE STATISTICS .....	32
3.8	SUMMARY .....	32
<b>4</b>	<b>IMPROVISED EXPLOSIVE DEVICE ANALYSIS IN BAGHDAD .....</b>	<b>33</b>
4.1	INTRODUCTION .....	33
4.2	INSTALL R LIBRARIES .....	34
4.3	THE IED DATA .....	34
4.4	READ THE DATA.....	35
4.5	DISPLAY DATA .....	35

4.6	PREPROCESS THE DATA.....	35
4.7	EXPLORE THE DATA .....	36
4.7.1	<i>Crime across space</i> .....	36
4.7.2	<i>Crime Over Time</i> .....	40
4.8	CRIMES SERIES PLOT .....	41
4.9	AGGREGATED DATA .....	42
4.10	CREATE A BAR CHART .....	43
4.11	CREATE A PIE CHART .....	44
4.12	TEMPORAL TRENDS .....	45
4.12.1	<i>Theft Over Time</i> .....	45
4.12.2	<i>IED Incident Time Heatmap</i> .....	46
4.12.3	<i>Reorder and format Factors</i> .....	46
4.12.4	<i>Create Time Heatmap</i> .....	47
4.12.5	<i>Arrest Over Time</i> .....	48
4.12.6	<i>Daily Arrests</i> .....	49
4.12.7	<i>Number of Arrest by Time of Arrest</i> .....	50
4.13	CORRELATION ANALYSIS .....	52
4.13.1	<i>Factor by Crime Category</i> .....	52
4.13.2	<i>Number of Arrests by Category and time of Arrest</i> .....	54
4.13.3	<i>Normalized Gradients</i> .....	56
4.13.4	<i>Normalized Number of Arrests by Category and Time</i> .....	56
4.14	FACTOR ANALYSIS .....	57
4.14.1	<i>Factor by Police Zone</i> .....	57
4.14.2	<i>Factor by Police Zone</i> .....	58
4.14.3	<i>Factor by Month</i> .....	59
4.14.4	<i>Factor By Year</i> .....	61
4.15	SESSION INFORMATION .....	63
<b>5</b>	<b>MULTIPLE LINEAR REGRESSION .....</b>	<b>67</b>
5.1	NOTATION FOR THE POPULATION MODEL.....	67
5.2	ESTIMATES OF THE MODEL PARAMETERS .....	67
5.3	INTERPRETATION OF THE MODEL PARAMETERS .....	68
5.4	PREDICTED VALUES AND RESIDUALS.....	68
5.5	COEFFICIENT OF DETERMINATION, R-SQUARED, AND ADJUSTED R-SQUARED.....	69
5.6	SIGNIFICANCE TESTING OF EACH VARIABLE .....	69
5.7	MULTIPLE LINEAR REGRESSION: SATELLITE SURVIVABILITY .....	71
5.8	MODEL 1: LINEAR MODEL WITH NUMERIC FEATURES .....	71
5.8.1	<i>Data Preprocessing</i> .....	71
5.8.2	<i>Load Required R Packages</i> .....	71
5.8.3	<i>Import the Data</i> .....	71
5.8.4	<i>Fill Missing Values</i> .....	72
5.8.5	<i>Create Dataset with Numeric Features</i> .....	72
5.8.6	<i>Train-Test Split</i> .....	73
5.8.7	<i>Model 1 Construction</i> .....	73

5.8.8	<i>Extract Model 1 Equation</i> .....	74
5.8.9	<i>Predict using Model 1</i> .....	74
5.8.10	<i>Calculate Model 1 Metrics</i> .....	74
5.8.11	<i>Plot Model 1 Coefficients</i> .....	75
5.9	MODEL 2: LINEAR MODEL WITH NUMERIC FEATURES .....	76
5.9.1	<i>Extract Model 2 Equation</i> .....	77
5.9.2	<i>Predict using Model 2</i> .....	77
5.9.3	<i>Calculate Model 2 Metrics</i> .....	77
5.9.4	<i>Model 2 Performance</i> .....	77
5.9.5	<i>Plot Model 2 Coefficients</i> .....	78
5.9.6	<i>Model 2 Residual Analysis</i> .....	78
5.9.7	<i>Residual Plots</i> .....	79
5.9.8	<i>Displaying the Confidence Interval</i> .....	79
5.10	MODEL 3: MIXED FEATURE TYPES .....	82
5.10.1	<i>Import the Data</i> .....	82
5.10.2	<i>Select Features</i> .....	83
5.10.3	<i>Fill Missing Feature Values</i> .....	83
5.10.4	<i>Create Response Set</i> .....	83
5.10.5	<i>Train-Test Split</i> .....	84
5.10.6	<i>Model 3 Construction</i> .....	84
5.10.7	<i>Extract Model 3 Equation</i> .....	85
5.10.8	<i>Use Model 3 to Predict</i> .....	85
5.10.9	<i>Model 3 Metrics</i> .....	85
5.10.10	<i>Plot Model 3 Coefficients</i> .....	85
5.10.11	<i>Model 3 Residual Analysis</i> .....	86
5.10.12	<i>Model 3 Residual Plots</i> .....	86
5.10.13	<i>Displaying the Confidence Interval</i> .....	87
5.10.14	<i>How to Interpret a Residual Plot</i> .....	89
5.10.15	<i>Error variance Analysis</i> .....	90
5.10.16	<i>Increasing Error Variance</i> .....	90
5.10.17	<i>Conclusions from Residual Analysis</i> .....	91
5.10.18	<i>Scatterplots</i> .....	92
5.10.19	<i>Holistic Scatterplot</i> .....	99
5.10.20	<i>Model 3 Evaluation</i> .....	99
5.10.21	<i>Bar Plot Observations</i> .....	100
5.11	CHAPTER SUMMARY .....	101
<b>6</b>	<b>REGULARIZATION AND REGRESSION</b> .....	<b>103</b>
6.1	MODEL 1 – SURVIVABILITY MODELING .....	103
6.1.1	<i>Load the Dataset</i> .....	103
6.1.2	<i>Data Preprocessing</i> .....	104
6.1.3	<i>Fill Empty Spaces</i> .....	104
6.1.4	<i>Create sets X and Y</i> .....	104
6.1.5	<i>Split X and Y</i> .....	105
6.1.6	<i>Form on Training Set</i> .....	105

6.1.7	<i>Model 1 Construction</i> .....	105
6.1.8	<i>Model 1 Output</i> .....	106
6.1.9	<i>Predict Survivability</i> .....	106
6.1.10	<i>Calculate MSE and R-Squared</i> .....	106
6.1.11	<i>Root Mean Square Error In R</i> .....	107
6.2	MODEL 2.....	107
6.2.1	<i>Splitting the Set</i> .....	108
6.2.2	<i>Model 2 Construction</i> .....	108
6.2.3	<i>Model Performance</i> .....	109
6.2.4	<i>Model 2 Coefficient Plot</i> .....	109
6.2.5	<i>Model 2 Residual Analysis</i> .....	110
6.2.6	<i>Residual Plots</i> .....	110
6.2.7	<i>Displaying the Confidence Interval</i> .....	111
6.2.8	<i>How to Interpret a Residual Plot</i> .....	113
6.2.9	<i>Error variance Analysis</i> .....	114
6.2.10	<i>Figure 1: Increasing Error Variance</i> .....	114
6.2.11	<i>Figure 2: Drifting Error Variance</i> .....	115
6.2.12	<i>Model 2 Performance</i> .....	116
6.3	MODEL 3.....	116
6.3.1	<i>Model 3 Construction</i> .....	117
6.3.2	<i>Model Performance</i> .....	119
6.3.3	<i>Residual Analysis</i> .....	119
6.3.4	<i>Conclusions from Residual analysis</i> .....	121
6.3.5	<i>Scatterplots</i> .....	122
6.3.6	<i>Holistic Scatterplot</i> .....	129
6.3.7	<i>Model 3 Performance</i> .....	129
6.3.8	<i>Bar Plot Observations</i> .....	130
6.4	A GLIMPSE OF LASSO REGRESSION .....	131
6.4.1	<i>Regularization</i> .....	131
6.4.2	<i>The Regularization Parameter</i> .....	132
6.4.3	<i>Regularization and Feature Reduction</i> .....	134
6.4.4	<i>Making the Tradeoff</i> .....	134
6.5	LASSO REGRESSION .....	135
6.5.1	<i>Lasso Regression Preliminaries</i> .....	135
6.5.2	<i>L1 Regularization</i> .....	136
6.5.3	<i>Load Imputed Data</i> .....	136
6.5.4	<i>Data Standardization</i> .....	136
6.5.5	<i>Define the Features and Response Sets</i> .....	138
6.5.6	<i>Matricize the Data</i> .....	138
6.5.7	<i>Define Lambda</i> .....	138
6.5.8	<i>Split the Data into Subsets</i> .....	138
6.6	LASSO REGRESSION MODEL CONSTRUCTION .....	139
6.7	MODEL 6: LASSO REGRESSION .....	139
6.8	MODEL EVALUATION .....	142

6.9	PREDICTION AND EVALUATION ON TRAIN DATA .....	142
6.10	PREDICTION AND EVALUATION ON TEST DATA.....	142
6.10.1	<i>Print Lasso Regressoin Coefficients.....</i>	146
6.10.2	<i>Lasso Model Performance .....</i>	147
6.10.3	<i>Cross-Validation Performance .....</i>	147
6.10.4	<i>Model Comparison Metrics Table.....</i>	148
6.11	CHAPTER SUMMARY .....	149
<b>7</b>	<b>RANDOM FOREST USING R .....</b>	<b>151</b>
7.1	RANDOM FOREST OVERVIEW.....	152
7.2	MEDICAL LONGEVITY STUDY OF PRIMARY BILIARY CIRRHOSIS (PBC)....	153
7.3	MODELING PREPARATION .....	154
7.4	CREATE A DATA DICTIONARY .....	155
7.5	PERFORM VARIABLE TRANSFORMATIONS.....	158
7.5.1	<i>Generate a Dataset with Integer Variables.....</i>	159
7.6	EXPLORATORY DATA ANALYSIS .....	159
7.6.1	<i>Categorical Plots.....</i>	160
7.6.2	<i>Plotting Continuous Variables.....</i>	164
7.6.3	<i>Covariate Panel Plot.....</i>	165
7.6.4	<i>Fitted Scatter Plots.....</i>	167
7.6.5	<i>Ph model from Fleming and Harrington 1991.....</i>	168
7.7	CREATE TRIAL AND TEST SETS.....	168
7.8	GENERALIZATION ERROR ESTIMATES.....	169
7.9	RANDOM FOREST PREDICTION.....	171
7.9.1	<i>Training Set Prediction .....</i>	171
7.9.2	<i>Plot the Survival Probability Function .....</i>	172
7.10	PLOT THE CUMULATIVE HAZARD FUNCTION .....	173
7.10.1	<i>Random forest imputation .....</i>	174
7.10.2	<i>Test set predictions.....</i>	175
7.11	VARIABLE SELECTION.....	176
7.11.1	<i>Variable Importance .....</i>	177
7.12	MINIMAL DEPTH.....	179
7.12.1	<i>Variable selection comparison.....</i>	182
7.13	RESPONSE/VARIABLE DEPENDENCE.....	184
7.13.1	<i>Variable dependence .....</i>	185
7.14	PARTIAL DEPENDENCE .....	190
7.15	VARIABLE INTERACTIONS .....	193
7.16	COPLOTS.....	196
7.17	PARTIAL DEPENDENCE COPLOTS.....	200
7.17.1	<i>Partial Dependence Plot.....</i>	202
7.17.2	<i>Conditional Dependence Plots.....</i>	204
7.17.3	<i>Partial dependence coplots.....</i>	206
7.17.4	<i>Create the conditional groups and add to the gg_variable object .....</i>	206

7.18	SUMMARY: HOW RANDOM FOREST WORKS.....	207
<b>8</b>	<b>CLUSTER MODELS .....</b>	<b>209</b>
8.1	DEFINITION.....	209
8.2	ALGORITHMS .....	211
8.3	K-MEANS CLUSTERING .....	212
8.4	STEPS.....	212
8.4.1	<i>Step 1. Install and import relevant libraries .....</i>	212
8.4.2	<i>Step 2. Create the data .....</i>	213
8.4.3	<i>Step 3. Use K-means and medoids to cluster data.....</i>	214
8.4.4	<i>Step 4. Use Hierarchical clustering .....</i>	220
8.4.5	<i>Step 5. Use Gaussian Mixed Models to cluster.....</i>	225
8.4.6	<i>Step 6. Compare the accuracy of each method.....</i>	228
8.5	OTHER CLUSTERING CONSIDERATIONS .....	229
8.5.1	<i>Choosing k for k-Means.....</i>	229
8.6	CONNECTIVITY BASED CLUSTERING (HIERARCHICAL CLUSTERING).....	234
8.6.1	<i>Metric.....</i>	235
8.6.2	<i>Linkage criteria .....</i>	236
8.6.3	<i>Agglomerative Nesting (Hierarchical Clustering).....</i>	237
8.6.4	<i>Plot with Parameters.....</i>	238
8.7	DENSITY-BASED CLUSTERING .....	243
8.8	CLUSTER MODEL – SEVERE WEATHER EVENTS AND THEIR EFFECTS .....	245
8.8.1	<i>Introduction.....</i>	245
8.8.2	<i>Analysis Questions and Outcomes.....</i>	246
8.8.3	<i>Data Processing.....</i>	246
8.8.4	<i>Install and Load Packages.....</i>	246
8.8.5	<i>Load the Weather Data.....</i>	246
8.8.6	<i>Exploratory Data Analysis (EDA).....</i>	248
8.8.7	<i>Populate Initial Analysis Dataset.....</i>	248
8.8.8	<i>Prepare the Storm Data for Analysis.....</i>	249
8.8.9	<i>Compile the Data for Fatalities and Injuries Analysis.....</i>	253
8.8.10	<i>Compile the Data for Property and Crop damage Analysis.....</i>	254
8.8.11	<i>Build Lookup Tables.....</i>	256
8.8.12	<i>Merge Lookup Tables into Storm Data.....</i>	257
8.9	RESULTS .....	258
8.9.1	<i>Public Health - Fatalities.....</i>	258
8.9.2	<i>Public Health - Injuries.....</i>	261
8.9.3	<i>Property Damage.....</i>	263
8.9.4	<i>Crop Damage.....</i>	265
8.10	CONCLUSION .....	266

## Preface

*"It is a capital mistake to theorize before one has data."*

~ Sherlock Holmes in "A study in Scarlet" by Arthur Conan Doyle

The greatest mistake an analyst-tester can make is to put the cart before the horse, so to speak, and jumps directly into specifying their favorite model of experimental design with little or no data analysis. When this occurs, it usually leads to testing the wrong thing and acquiring the wrong data. We can recover from bad analysis of the right data for the right problem by simply redoing the analysis. We cannot recover from framing the wrong problem and collecting the wrong data. Doyle understood this.

So, how do we get data before we design a test? In part, the answer lies in what we consider data or how we define data. Everything is data. The words I am writing here is data. The Instagram I just received is data. Testers often get stuck by classifying data as structured and derived from an experiment. However, we can perform a test without an experiment, but we cannot pose an experiment without data.

Here some sources of pretest data:

- Data from prototyping or technology demonstrations
- Data from similar systems
- Data from system design simulations
- Data from exercises
- Data from system-expert surveys

*"Data are just summaries of thousands of stories—tell a few of those stories to help make the data meaningful."*

~ Dan Heath, bestselling author

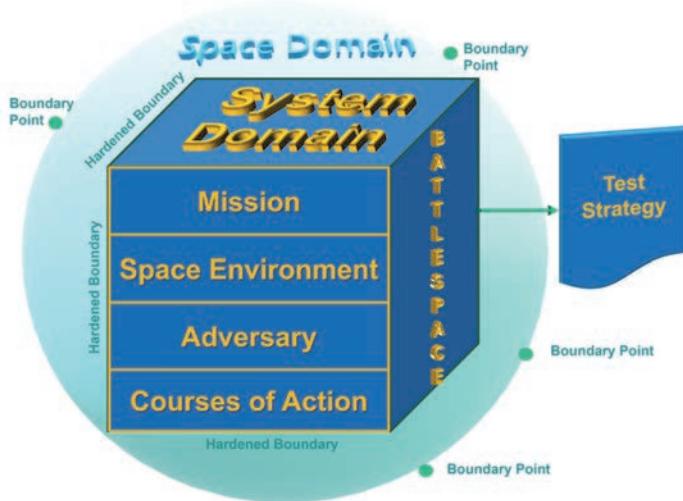
Characterizing a system is just like *data summaries*. If we are not telling the story about the system we are testing using the data we collected, then we would be better off admitting that we are not up to the task.

So, how much data do we need for telling the right story? We need all of it! We need every log file, every sound bite, every word spoken, and

every bit of debris that might not seem relevant at the moment. We need the structured data, the semi-structured data, and the unstructured data. With data pipelines, data warehouses, data lakes, data buckets, and machine learning, all data is assessable. It might not be free, but it is available.

With that said, why would we want to develop an experiment that restricts the data we collect, before we really understand what data we may require at any point during a system acquisition. The full-factorial design that we like so much, if useless is used before the time is right.

Moreover, in the Combat Test Planning Framework context, we would miss out on a lot of potential, useful data that lies in the boundaries of the battlespace, perhaps outside of the system space, as depicted by Figure i-1-1



*Figure i-1-1. Illustration of the System Domain (space) within the Space Domain with important boundary points.*

This Volume of *Statistical Methods for Test and Evaluation* explores the realm of machine learning that may be applicable to mining data, describing data, analyzing data (exploratory), and structuring the data we require for testing. And one last quote:

*"We do not perform test to see what a system can do. We perform them to get data that shows what a system can do. Every act that does not contribute to this end is useless."*

~ Dr. Jeffrey Strickland, data science author

## Other Technical Books by the Author

1. Statistical Methods for Test and Evaluation, Volume 1: The Combat Test Framework, Jeffrey Strickland, 2024, CC BY-NC-SA. Lulu, Inc. ISBN 978-1-304-26228-8
2. Statistical Methods for Test and Evaluation, Volume 3: Combinatorial Design using R, Jeffrey Strickland, 2024, CC BY-NC-SA. Lulu, Inc. ISBN 978-1-304-42779-3
3. Statistical Methods for Test and Evaluation, Volume 4: Experimental Design using R, Jeffrey Strickland, 2024, CC BY-NC-SA. Lulu, Inc. ISBN 978-1-304-26228-8
4. Statistical Methods for Test and Evaluation, Volume 4: Experimental Designs using Excel, Jeffrey Strickland, 2022, CC BY-NC-SA. Lulu, Inc. ISBN 978-1-387-90547-8
5. Practitioner's Guide to Military Modeling and Simulation, 2024 by Jeffrey Strickland, CC BY-NC-SA. Lulu, Inc. ISBN 978-1-304-42780-9
6. *Building Scenarios in Freeciv: The Templars*, 2024 by Jeffrey Strickland, CC BY-NC-SA. Lulu, Inc. ISBN: 9-781304633087
7. *Discrete Event Simulation Using ExtendSim 10*, Copyright 2023 by Jeffrey S. Strickland. Lulu, Inc., ISBN: 978-1-716-20282-7
8. *Regression totum modum*. Copyright © 2023 by Jeffrey S. Strickland. Lulu, Inc., ISBN: 978-1-329-33732-9
9. *Orbital Mechanics using Python and R*, Copyright © 2022 by Jeffrey S. Strickland. Lulu, Inc., ISBN: 978-1-387-50683-5
10. *Systems Engineering Processes and Practice*, Second Edition, Copyright © 2022, Lulu, Inc., ISBN 978-1-387-81105-2
11. *The Space Handbook*, Jeffrey Strickland, Copyright © 2022, Lulu, Inc., ISBN 978-1-387-97656-3
12. *The Python Guide for New Data Scientists*, Jeffrey Strickland, Copyright © 2022, Lulu, Inc. ISBN 978-1-4583-2161-9
13. *The R Guide for New Data Scientists*, Copyright © 2022, Lulu, Inc. ISBN 978-1-6780-0244-2
14. *Time Series Analysis and Forecasting using Python & R*. Jeffrey Strickland, Copyright © 2020, Lulu, Inc. ISBN 978-1-716-45113-3
15. *Data Science Applications using Python and R*. Jeffrey Strickland, Copyright © 2020, Lulu, Inc. ISBN 978-1-716-89644-6

16. *Data Science Applications using R.* Jeffrey Strickland, Copyright© 2020, Lulu, Inc. ISBN 978-0-359-81042-0
17. *Predictive Crime Analysis using R.* Jeffrey Strickland, Copyright© 2018, Jeffrey Strickland. Lulu, Inc. ISBN 978-0-359-43159-5
18. *Logistic Regression Inside-Out.* Copyright Jeffrey Strickland, © 2017 by Jeffrey S. Strickland. Lulu, Inc. ISBN 978-1-365-81915-5
19. *Time Series Analysis using Open-Source Tools.* Jeffrey Strickland, Copyright© 2016, Jeffrey Strickland. Glasstree, Inc. ISBN 978-1-5342-0100-2
20. *Data Analytics Using Open-Source Tools.* Copyright Jeffrey Strickland, © 2015 by Jeffrey Strickland. Lulu Inc. ISBN 978-1-365-21384-7
21. *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
22. *Predictive Analytics using R.* Copyright © 2015 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-84101-7
23. *Operations Research using Open-Source Tools.* Copyright © 2015 by Jeffrey Strickland. Lulu Inc. ISBN 978-1-329-00404-7.
24. *I Rode With Wallace.* Copyright © 2015 by Jeffrey S. Strickland. Lulu, Inc., ISBN: 978-1329565661
25. *Predictive Modeling and Analytics.* © 2014 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-37544-4.
26. *Verification and Validation for Modeling and Simulation.* Copyright © 2014 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-312-74061-7.
27. *Using Math to Defeat the Enemy: Combat Modeling for Simulation.* © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-257-83225-5.
28. *Mathematical Modeling of Warfare and Combat Phenomenon.* Copyright © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-45839255-8.
29. *Simulation Conceptual Modeling.* Copyright © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-105-18162-7.
30. *Systems Engineering Processes and Practice,* Copyright © 2011 by Jeffrey S. Strickland. Lulu.com. ISBN 978-1-257-09273-4.
31. *Discrete Event Simulation using ExtendSim 8.* Copyright © 2010 by Jeffrey S. Strickland. Lulu.com. ISBN 978-0-557-72821-3

32. *Fundamentals of Combat Modeling*, Copyright © 2011 by Jeffrey S. Strickland. Lulu.com, ISBN 978-1-257-00583-3.

## Non-Technical Books by the Author

33. *Knights of the Cross*, Copyright © 2011 by Jeffrey S. Strickland. Lulu.com, ISBN 978-1-105-35162-4.
34. A Mason's Mason, by Jeffrey S. Strickland. Lulu.com 2024 by Jeffrey S. Strickland. Lulu, Inc., ISBN 978-1-312-03214-9
35. *Weird Scientists – the Creators of Quantum Physics*, Copyright © 2011 by Jeffrey S. Strickland. Lulu.com, ISBN 978-1-257-97624-9
36. *Albert Einstein: "Nobody expected me to lay golden eggs"*, Copyright © 2011 by Jeffrey S. Strickland. ISBN 978-1-257-86014-2.
37. *The Men of Manhattan: Creators of the Nuclear Era*, Copyright © 2011 by Jeffrey S. Strickland. Lulu.com, ISBN 978-1-257-76188-3.
38. *The Devil Did Not Make Me Do It*, Copyright © 2018 by Jeffrey S. Strickland, Printed by Lulu, Inc., ISBN 9781365982149.
39. *To Watch Over Them Day and Night*, Copyright © by Jeffrey Strickland, Printed by Lulu, Inc. ISBN 9781365880759.
40. *Quantum Phaith*, Copyright © 2011 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781257645619
41. *Quantum Hope*, Copyright © 2015 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781329160781.
42. *Quantum Love*, Copyright © 2017 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781365835551
43. *I Rode with Wallace*, Copyright © 2015 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781329565661.
44. *Handbook of Handguns*, Copyright © 2013 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781300973294.
45. *LinkedIn Memoirs*, Copyright © 2014 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781312717329.
46. *Dear Mr. President*, Copyright © 2012 by Jeffrey Strickland, Printed by Lulu, Inc., ISBN 9781105555145.





# 1 Describing Data

One of the things I say about testing and evaluating defense systems is:

*We do not test to see what a system can do; we test to collect data that will show what a system can do.*

The statement might not be intuitive, so let me say plainly that the only reason we plan, design, and execute tests is to get data. The data may be comprised of numbers, documents, photos, vectors, and more.

Data, so to speak, has the loosest lips of all: if tortured, it will reveal everything. It is keener than our ears, and more insightful than our eyes. Data is everything. To describe it is more...

## 1.1 Describing Plant Growth

As an example we consider one of the data sets available with R relating to an experiment into plant growth. The purpose of the experiment was to compare the yields on the plants for a control group and two treatments of interest. The response variable was a measurement taken on the dried weight of the plants.

The first step in the investigation is to take a copy of the data frame so that we can make some adjustments as necessary while leaving the original data alone. We use the `factor` function to re-define the labels of the group variables that will appear in the output and graphs:

```
plant.df = PlantGrowth  
plant.df$group = factor(plant.df$group,  
                        labels = c("Control", "Treatment 1", "Treatment 2"))
```

The labels argument is a list of names corresponding to the levels of the group factor variable. A boxplot of the distributions of the dried weights for the three competing groups is created using the `ggplot` package:

```
require(ggplot2)
```

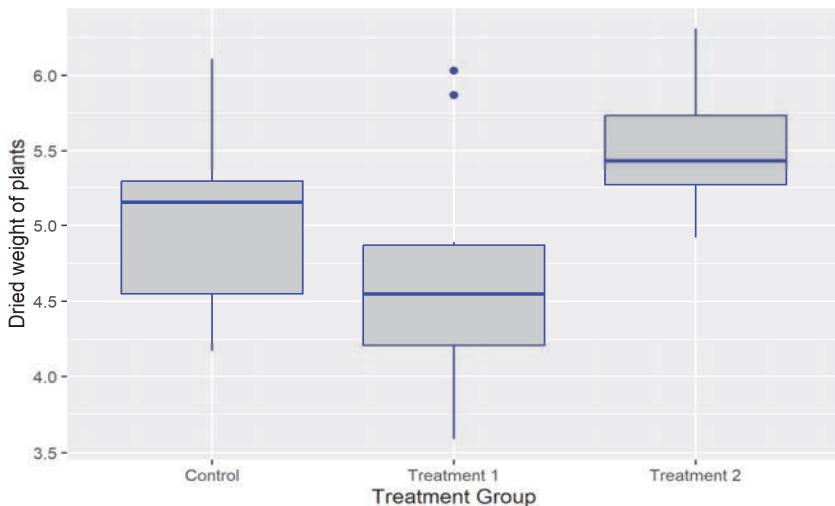
Loading required package: ggplot2

```
ggplot(plant.df, aes(x = group, y = weight)) +  
  geom_boxplot(fill = "grey80", color = "blue") +
```

```
scale_x_discrete() + xlab("Treatment Group") +  
ylab("Dried weight of plants")
```

The `geom_boxplot` option is used to specify background and outline colors for the boxes. The axis labels are created with the `xlab` and `ylab` options. The plot that is produced looks like this:

Initial inspection of the data shown in **Figure 1-1** suggests that there are differences in the dried weight for the two treatments, but it is not so clear cut to determine whether the treatments are different to the control group.



**Figure 1-1. Boxplots for comparing the distribution of dried weights of plants**

To investigate these differences we fit the one-way ANOVA model using the `lm` function and look at the parameter estimates and standard errors for the treatment effects. The function call is:

```
plant.mod1 = lm(weight ~ group, data = plant.df)
```

We save the model fitted to the data in an object so that we can undertake various actions to study the goodness of the fit to the data and other model assumptions. The standard summary of a `lm` object is used to produce the following output: `summary(plant.mod1)`

The model output indicates some evidence of a difference in the average growth for the 2nd treatment compared to the control group. An

analysis of variance (ANOVA) table for this model can be produced via the `anova` function:

```
anova(plant.mod1)
```

Analysis of Variance Table

```
Response: weight
  Df  Sum Sq Mean Sq F value Pr(>F)
group      2  3.7663  1.8832  4.8461 0.01591 *
Residuals 27 10.4921  0.3886
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This table confirms that there are differences between the groups which were highlighted in the model summary. The function `confint` is used to calculate confidence intervals on the treatment parameters, by default 95% confidence intervals:

```
confint(plant.mod1)
```

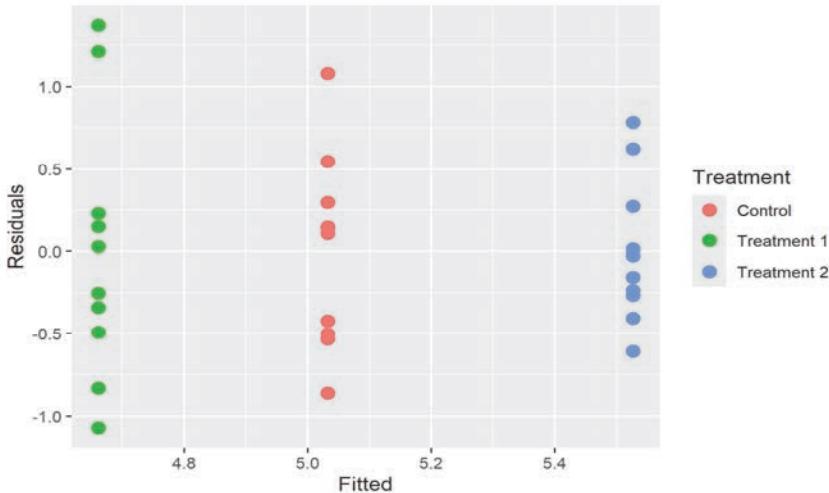
	2.5 %	97.5 %
(Intercept)	4.62752600	5.4364740
groupTreatment 1	-0.94301261	0.2010126
groupTreatment 2	-0.07801261	1.0660126

The model residuals can be plotted against the fitted values to investigate the model assumptions. First we create a data frame with the fitted values, residuals and treatment identifiers:

```
plant.mod = data.frame(Fitted = fitted(plant.mod1),
  Residuals = resid(plant.mod1), Treatment =
  plant.df$group)
```

And then produce the plot in *Figure 1-2*.

```
ggplot(plant.mod, aes(Fitted, Residuals,
  color = Treatment)) + geom_point(size = 3)
```



*Figure 1-2. Scatterplot of fitted vs. residual values*

We can see that there is no major problem with the diagnostic plot but some evidence of different vulnerabilities in the spread of the residuals for the three treatment-groups. The R function `aov` builds an ANOVA model from the data rather than from a model.

```
plant.aov <- aov(weight ~ group, plant.df)
```

The basic result does not give a great deal of information. We need to view the summary so try:

```
summary(plant.aov)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)						
group	2	3.766	1.8832	4.846	0.0159 *						
Residuals	27	10.492	0.3886								
---											
Signif. codes:	0	'***'	0.001	'**'	0.01	'*'	0.05	'. '	0.1	' '	1

So far we have conducted a simple one-way anova. In this instance we see that there is a significant effect of diet upon growth. However, there are 6 treatments. We would like to know which of these treatments are significantly different from the controls and from other treatments. We need a post-hoc test. R provides a simple function to carry out the *Tukey HSD test*.

This will show all the paired comparisons like so:

### TukeyHSD(plant.aov)

```
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = weight ~ group, data = plant.df)

$group
            diff      lwr      upr     p adj
Treatment 1-Control -0.371 -1.0622161 0.3202161 0.3908711
Treatment 2-Control  0.494 -0.1972161 1.1852161 0.1979960
Treatment 2-Treatment 1  0.865  0.1737839 1.5562161 0.0120064
```

The table/output shows us the difference between pairs, the 95% confidence interval(s) and the *p*-value of the pairwise comparisons. All we need to know!

## 1.2 Analysis of Variance

The analysis of variance (ANOVA) model can be extended from making a comparison between multiple groups to consider additional factors in an experiment. The simplest extension is from one-way to two-way ANOVA where a second factor is included in the model as well as a potential interaction between the two factors.

As an example consider a military medical logistics unit that regularly has to ship medical supplies to various (five for this example) field hospitals in forward operating bases. The 187th Medical Battalion has the option of using three subordinate supply companies, A, B, and D, all of which have roughly similar carrying capacities for each delivery. To determine which service to use, the battalion decides to run an experiment shipping three medical supply packages from its HQ to each of the five field hospitals. The delivery time for each package is recorded and the data loaded into R:

```
delivery.df = data.frame(
  Service = c(rep("A Company", 15), rep("B Company",
15),
  rep("D Company", 15)),
  Destination = c(rep(c("Hospital 1", "Hospital 2",
"Hospital 3", "Hospital 4", "Hospital 5"), 9)),
  Time = c(15.23, 14.32, 14.77, 15.12, 14.05,
15.48, 14.13, 14.46, 15.62, 14.23, 15.19, 14.67,
14.48, 15.34, 14.22, 16.66, 16.27, 16.35, 16.93,
```

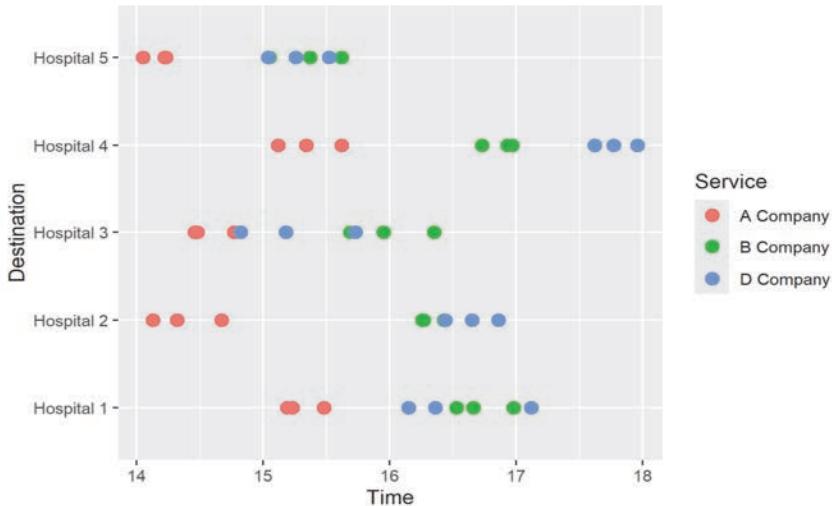
```

15.05, 16.98, 16.43, 15.95, 16.73, 15.62, 16.53,
16.26, 15.69, 16.97, 15.37, 17.12, 16.65, 15.73,
17.77, 15.52, 16.15, 16.86, 15.18, 17.96, 15.26,
16.36, 16.44, 14.82, 17.62, 15.04)
)

```

The data is then displayed (see **Figure 1-3**) using a dot plot for an initial visual investigation of any trends in delivery time between the three supply companies and their service and across the five field hospitals. The color aesthetic is used to distinguish between the three supply companies in the plot.

```
ggplot(delivery.df, aes(Time, Destination,
color = Service)) + geom_point(size = 3)
```



**Figure 1-3. Scatterplot of destination vs. delivery time**

The graph shows a general pattern of A Company having shorter delivery times than the other two companies. There is also an indication that the differences between the supply transport varies for the five hospitals and we might expect the interaction term to be significant in the two-way ANOVA model. To fit the two-way ANOVA model we use this code:

```
delivery.mod1 = aov(Time ~ Destination*Service,
data = delivery.df)
```

The \* symbol instructs R to create a formula that includes main effects for both Destination and Service as well as the two-way interaction

between these two factors. We save the fitted model to an object which we can summarize as follows to test for importance of the various model terms:

```
summary(delivery.mod1)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)						
Destination	4	17.542	4.385	61.155	5.41e-14 ***						
Service	2	23.171	11.585	161.560	< 2e-16 ***						
Destination:Service	8	4.189	0.524	7.302	2.36e-05 ***						
Residuals	30	2.151	0.072								
---											
Signif. codes:	0	'***'	0.001	'**'	0.01	'*'	0.05	'. '	0.1	' '	1

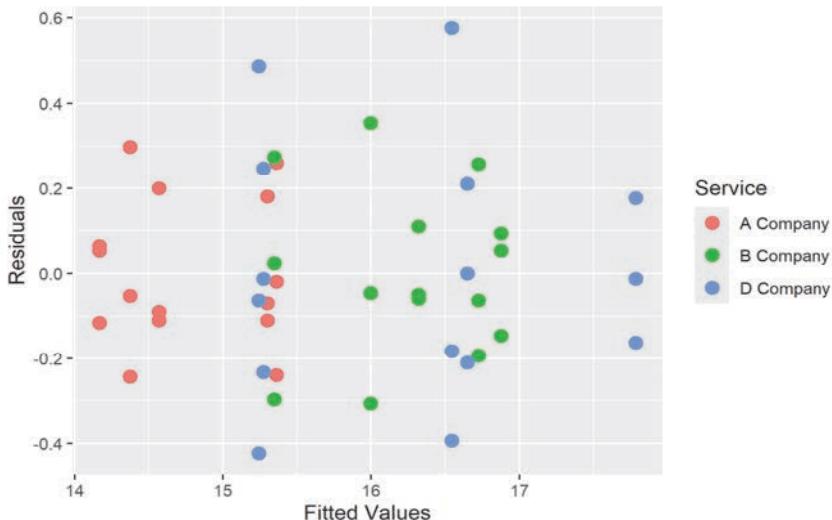
We have compelling evidence here that there are differences between the three supply companies, between the five field hospital destinations and that there is an interaction between destination and supply service in line with what we saw in the original plot of the data. Now that we have fitted the model and identified the key factors we need to investigate the model diagnostics to ensure that the various assumptions are broadly valid.

We can plot the model residuals against fitted values to look for obvious trends that are not consistent with the model assumptions about independence and common variance. The first step is to create a data frame with the fitted values and residuals from the above model:

```
delivery.res = delivery.df
delivery.res$M1.Fit = fitted(delivery.mod1)
delivery.res$M1.Resid = resid(delivery.mod1)
```

Then a scatter plot is used to display the fitted values and residuals where the color aesthetic highlights which points correspond to the three competing delivery services, as shown in **Figure 1-4**.

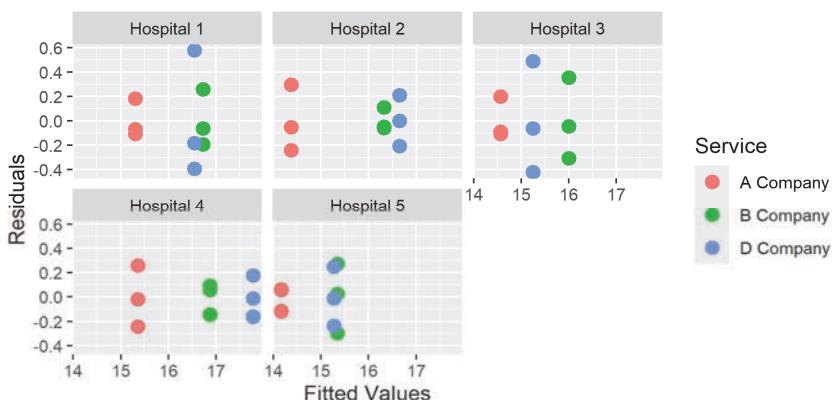
```
ggplot(delivery.res, aes(M1.Fit, M1.Resid,
  color = Service)) + geom_point(size=3) +
  xlab("Fitted Values") + ylab("Residuals")
```



*Figure 1-4. Scatterplot of residuals vs. fitted values*

There are no obvious patterns in this plot that suggest problems with the two-way ANOVA model that we fitted to the data. As an alternative display we could separate the residuals into destination field hospitals, where the `facet_wrap` function instructs ggplot to create a separate display (panel) for each of the destinations as shown in **Figure 1-5**.

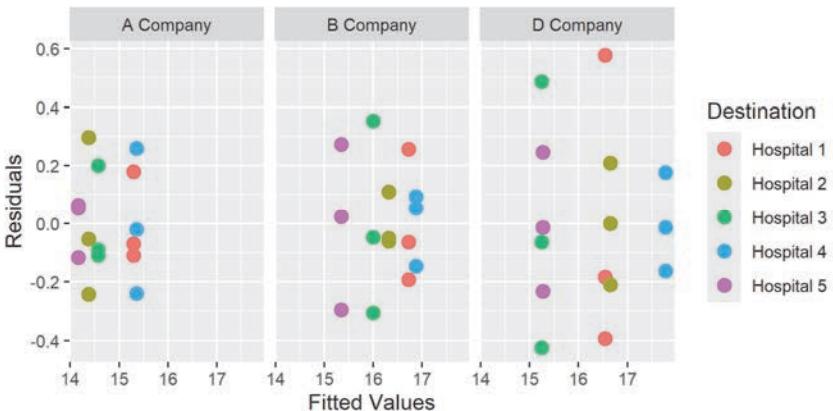
```
ggplot(delivery.res, aes(M1.Fit, M1.Resid,
  color = Service)) + geom_point(size = 3) +
  xlab("Fitted Values") + ylab("Residuals") +
  facet_wrap(~ Destination)
```



*Figure 1-5. Scatterplot matrix of residuals vs. fitted values for each field hospital*

No obvious problems in this diagnostic plot. We could also consider dividing the data by delivery service to get a different view of the residuals, like shown in **Figure 1-6**.

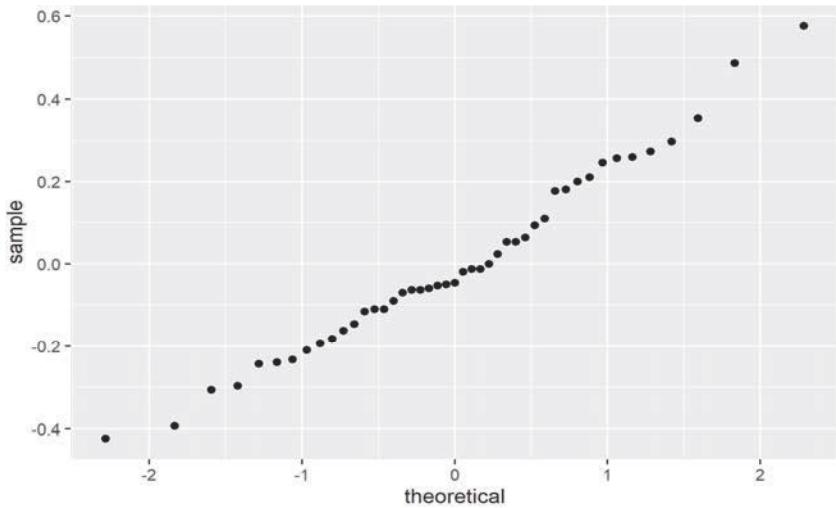
```
ggplot(delivery.res, aes(M1.Fit, M1.Resid,
  color = Destination)) +
  geom_point(size = 3) + xlab("Fitted Values") +
  ylab("Residuals") +
  facet_wrap(~ Service)
```



**Figure 1-6.** Scatterplot matrix for residuals vs. fitted values for each supply company

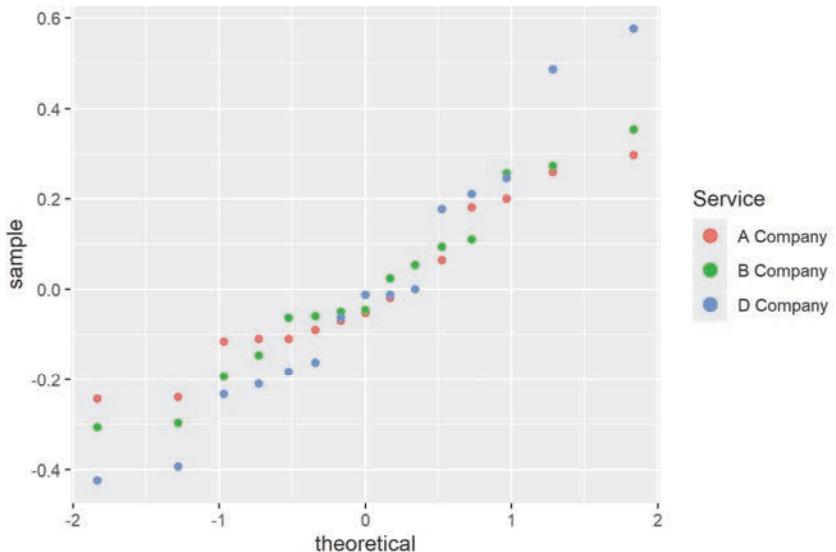
Again there is nothing substantial here to lead us to consider an alternative analysis. Lastly we consider the normal probability plot in **Figure 1-7** of the model residuals, using the `stat_qq` option and the normal probability plot of each company in **Figure 1-8**.

```
ggplot(delivery.res, aes(sample = M1.Resid)) + stat_qq()
```



*Figure 1-7. Q-Q plot of residuals showing evidence of normality*

```
ggplot(delivery.res, aes(sample = M1.Resid,
  color = Service)) + stat_qq() +
  geom_point(x = delivery.res$M1.Fit,
  y = delivery.res$M1.Resid, size = 3)
```



*Figure 1-8. Q-Q plot of residuals for each supply company showing evidence of normality (with noted outliers on both extremes)*

This plot is very close to the straight line we would expect to observe if the data was a close approximation to a normal distribution. To round off the analysis we look at the Tukey HSD multiple comparisons to confirm that the differences are between Company A's service and the other two companies' services:

```
TukeyHSD(delivery.mod1, which = "Service")
```

```
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = Time ~ Destination * Service, data =
delivery.df)

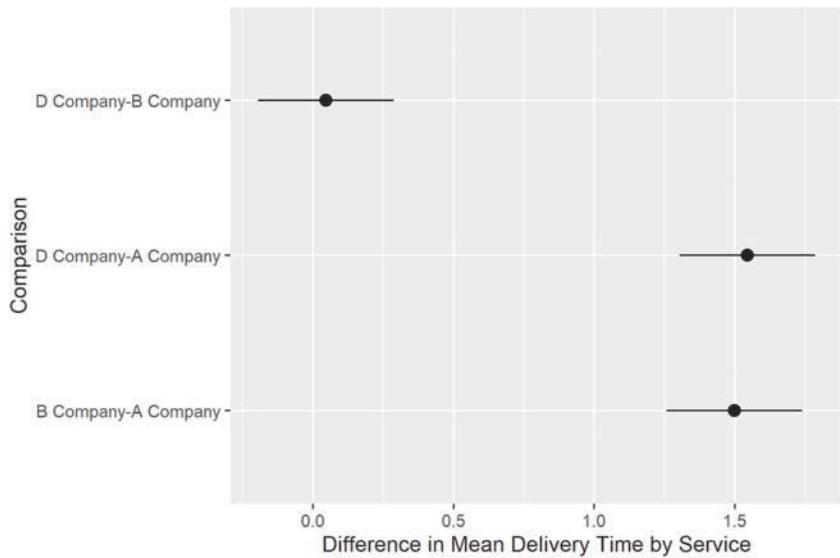
$Service
      diff      lwr      upr      p adj
B Company-A Company 1.498667  1.2576092 1.7397241 0.0000000
D Company-A Company 1.544667  1.3036092 1.7857241 0.0000000
D Company-B Company 0.046000 -0.1950575 0.2870575 0.8856246
```

Even with the multiple comparison post-hoc adjustment there is very strong evidence for the differences that we have consistently observed throughout the analysis. We can use `ggplot` to visualize the difference in mean delivery time for the services and the 95% confidence intervals on these differences. We create a data frame from the `TukeyHSD` output by extracting the component relating to the delivery service comparison and add the text labels by extracting the row names from the data frame.

```
delivery.hsd = data.frame(TukeyHSD(delivery.mod1,
                                    which = "Service")$Service)
delivery.hsd$Comparison = row.names(delivery.hsd)
```

We then use the `geom_pointrange` to specify lower, middle and upper values based on the three pairwise comparisons of interest as shown in *Figure 1-9*.

```
ggplot(delivery.hsd, aes(Comparison, y = diff,
                         ymin = lwr, ymax = upr)) +
  geom_pointrange() + ylab("Difference in Mean Delivery
Time by Service") +
  coord_flip()
```



*Figure 1-9. Interaction comparison of mean delivery times*

The `coord_flip` is used to make the confidence intervals horizontal rather than vertical on the graph.

### 1.3 Experimental Designs

This example requires the R stats package. There are three groups with seven observations per group. We denote group  $i$  values by  $y_i$ :

```
y1 = c(18.2, 20.1, 17.6, 16.8, 18.8, 19.7, 19.1)
y2 = c(17.4, 18.7, 19.1, 16.4, 15.9, 18.4, 17.7)
y3 = c(15.2, 18.8, 17.7, 16.5, 15.9, 17.1, 16.7)
```

```
local({pkg <- select.list(sort(.packages(all.available =
TRUE)), graphics=TRUE)
if(nchar(pkg)) library(pkg, character.only=TRUE)})
```

Now we combine them into one long vector, with a second vector, `group`, identifying group membership:

```
y = c(y1, y2, y3)
n = rep(7, 3)
n
```

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

```
group = rep(1:3, n)
group
```

```
[1] 1 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3
```

Here are summaries by group and for the combined data. First we show **stem-leaf diagrams**.

```
tmp = tapply(y, group, stem)
```

The decimal point is at the |

```
16 | 8
17 | 6
18 | 28
19 | 17
20 | 1
```

The decimal point is at the |

```
15 | 9
16 | 4
17 | 47
18 | 47
19 | 1
```

The decimal point is at the |

```
15 | 29
16 | 57
17 | 17
18 | 8
```

```
stem(y)
```

The decimal point is at the |

```
15 | 299
16 | 4578
17 | 14677
18 | 24788
19 | 117
20 | 1
```

Now we show summary statistics by group and overall. We locally define a temporary function, `tmpfn`, to make this easier.

```
tmpfn = function(x) c(sum = sum(x), mean = mean(x),
                      var = var(x), n = length(x))
tapply(y, group, tmpfn)
```

```
$`1`
      sum      mean      var      n
130.300000 18.614286 1.358095 7.000000

$`2`
      sum      mean      var      n
123.600000 17.657143 1.409524 7.000000

$`3`
      sum      mean      var      n
117.900000 16.842857 1.392857 7.000000
```

```
tmpfn(y)
```

```
      sum      mean      var      n
371.800000 17.704762 1.798476 21.000000
```

```
data = data.frame(y = y, group = factor(group))
fit = lm(y ~ group, data)
anova(fit)
```

#### Analysis of Variance Table

```
Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
group      2 11.007  5.5033  3.9683 0.03735 *
Residuals 18 24.963  1.3868
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
df = anova(fit)[, "Df"]
names(df) = c("trt", "err")
```

**Get  $F$  Values:** First we extract the treatment and error degrees of freedom. Then we use `qt` to get the tabled  $F$  values.

```
df
```

```
trt err  
2 18
```

```
alpha = c(0.05, 0.01)  
qf(alpha, df["trt"], df["err"], lower.tail = FALSE)
```

```
[1] 3.554557 6.012905
```

A confidence interval on the pooled variance can be computed as well using the `anova(fit)` object. First we get the residual sum of squares, SST<sub>Trt</sub>, then we divide by the appropriate chi-square tabled values.

```
anova(fit)[ "Residuals", "Sum Sq"]
```

```
[1] 24.96286
```

```
anova(fit)[ "Residuals", "Sum Sq"] / qchisq(c(0.025,  
0.975), 18)
```

```
[1] 3.0328790 0.7918086
```

```
anova(fit)[ "Residuals", "Sum Sq"] / qchisq(c(0.025,  
0.975), 18, lower.tail = FALSE)
```

```
[1] 0.7918086 3.0328790
```



## 2 Comparing Mean Values

### 2.1 Effects of Dosage levels and Supplements on Tooth Growth

### 2.2 Introduction

The purpose of this report is to analyze 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.

### 2.3 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.

### 2.4 Exploratory Data Analysis

Before delving into the heart of the analysis, it is appropriate perform some basic exploratory data analysis.(EDA)

### 2.4.1 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 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 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 ...
```

```
tg <- ToothGrowth
dim(tg)
```

```
[1] 60 3
```

### 2.4.2 ToothGrowth Summary

```
library(dplyr)
group_data = group_by(tg, supp, dose)
dplyr::summarise(group_data, mean(len))
```

```
# A tibble: 6 × 3
# Groups:   supp [2]
  supp   dose `mean(len)` 
  <fct> <dbl>      <dbl>
1 OJ     0.5       13.2
2 OJ     1          22.7
3 OJ     2          26.1
4 VC     0.5       7.98
5 VC     1          16.8
6 VC     2          26.1
```

### 2.4.3 Data Visualization

First, we construct a scatterplot of tooth length according to supplement type (Vitamin C and Orange Juice) by dosage levels, using `ggplot` with `geom` functions, as shown in *Figure 2-1..*

```
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(color="slateblue", size=rel(1),
face="bold"))
# Legend Labels
p = p + theme(legend.text = element_text(color="black",
size=rel(1), face="bold"))
p

```

## Tooth Growth

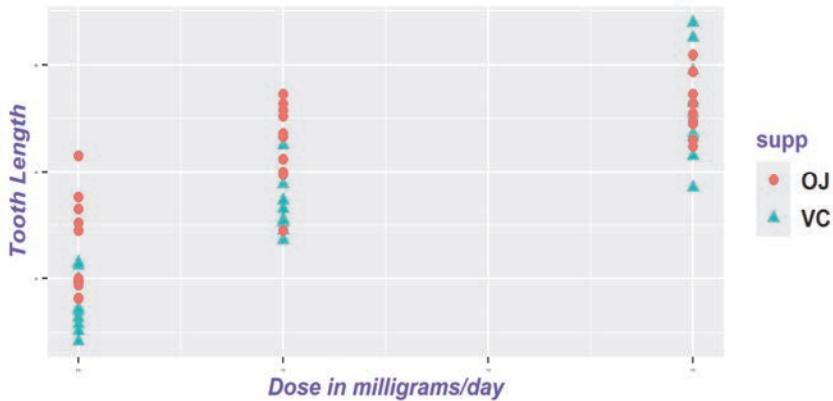


Figure 2-1. scatterplot of tooth length according to supplement type by dosage levels

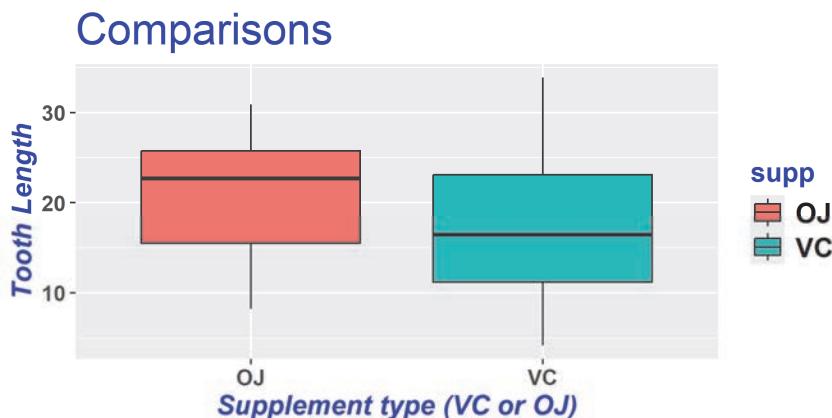
### 2.4.4 Boxplot Tooth Growth Factors

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 2-2**)
2. Second, we explore the boxplots of the three levels of doses (see **Figure 2-3**)
3. Third, the dosage levels for each of the two different supplements (see **Figure 2-4**)

### *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(color="blue2",
                                            size=rel(1.25), face="bold"))
# Legend Labels
p = p + theme(legend.text = element_text(color="black",
                                           size=rel(1.25), face="bold"))
p
```



*Figure 2-2. Comparison of two means whose differences may be significant*

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.

## Comparisons

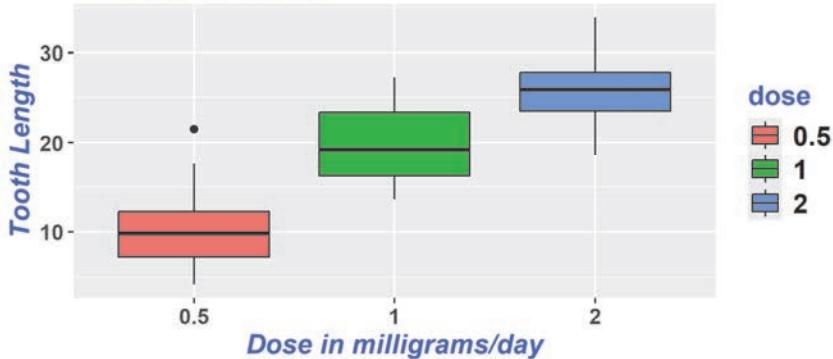


Figure 2-3. Boxplot Comparison of Three Dosage Levels

This set of boxplots also reveal distributions whose means may be significantly different warrant analysis using hypothesis testing.

## Multiple Comparisons

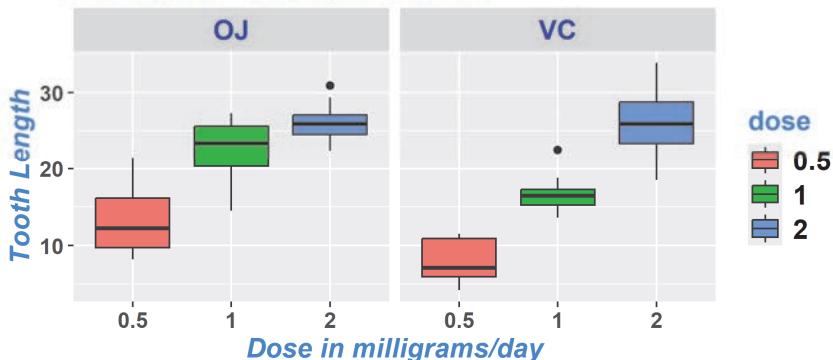


Figure 2-4. Boxplot Comparison of Orange Juice vs Vitamin C at Three Dosage Levels

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.

## 2.5 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.

### 2.5.1 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 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.

$H_0$  : There is no significant difference between the means supplement type (VC or OJ). That is, the two supplements have similar effects on tooth length.

$H_A$  : 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.

## 2.5.2 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.

H10: The means of dosage levels “0.5” and “1” are the same. H20: The means of dosage levels “0.5” and “2” are the same. H30: 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`.

### 2.5.3 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 following 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](#).

## 2.6 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.



## 3 The Central Limit Theorem

### 3.1 Overview

In this analysis, we compare the distribution of 1000 random samples from an exponential distribution and the distribution of averages of 40 exponentials.

This problem is an Application of the Central Limit Theorem (CLT). 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).

### 3.2 Simulations

#### 3.2.1 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 parameters  $\lambda$ , the rate. For our distribution, we take  $\lambda$  to equal 0.2, with the mean being  $\mu = 1/\lambda = 5$  and standard deviation,  $\sigma = 1/\lambda = 5$ .

#### 3.2.2 Simulation of the Exponential( $\lambda$ ) distribution

First, generate the distribution of 1000 random exponentials, using the stats-package, which provides the random exponential 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")
```

#### 3.2.3 Set up the Simulation

The documentation for the exponential 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 exponential distributions, calculate the sample means and plot a histogram. First, we construct the histogram, shown in **Figure 3-1**.

```
n_sim = 1000
lambda = 0.2
sim_exp = rexp(n_sim,lambda)
hist(sim_exp, col = "powderblue")
```

Histogram of `sim_exp`

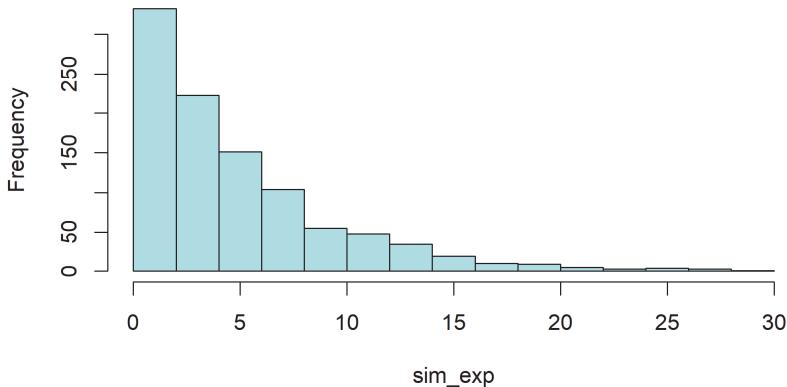


Figure 3-1. Histogram of 1000 random uniform distributions

Now, we calculate the sample mean.

```
cat("The sample mean is ", mean(sim_exp))
```

The sample mean is 4.878129

### 3.3 Application of the CLT

Now, we take our exponential distributions, and using their sample means, we generate as distribution of 40 of those means.

### 3.3.1 The Normal Distribution, Normal(mu,sigma)

The standard normal distribution is defined by two parameters, population mean = mu (or  $\mu$ ), the population standard deviation = sigma (or  $\sigma$ ) or  $Normal(\mu, \sigma)$ .

### 3.3.2 Normal Random Variates

The sampling distribution of the sample means 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 exponential( $\lambda$ ) 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 exponentials is:",
mean(d$mean), "which is the mean of the normal
distribution.")
```

The mean value of the exponential is: 4.924604 which is the mean of the normal distribution.

## 3.4 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 4.924604
```

So, we see that the sample mean of the normal distribution of exponential means is approximately 5, and the parameter mu is 5.

## 3.5 The Variance

The variance of the population parameter,  $\sigma^2$ , is given by  $(1/l\lambda)^2/n_{dist}$ . 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  
0.6329
```

So, we see that the sample variance approximates the population parameter,  $\sigma^2$ .

## 3.6 Histogram Approximation of the Normal Distribution

Here, we construct the histogram (see *Figure 3-2*) of our normal distribution overlaid by horizontal lines representing the sample mean and the population (theoretical) mean.

```
hist(mns_u, col = "dodgerblue", xlab = "rnorm(x_bar,s)",  
     main = "Histogram of an Approximate  
     Normal(mu,sigma) Distribution")  
# Sample mean  
abline(v = mean(d$mean), col = "green", lwd = 4)  
# Theoretical mean  
abline(v=1/lambda, col="red", lwd=2)
```

## Histogram of an Approximate Normal(mu,sigma) Distribution

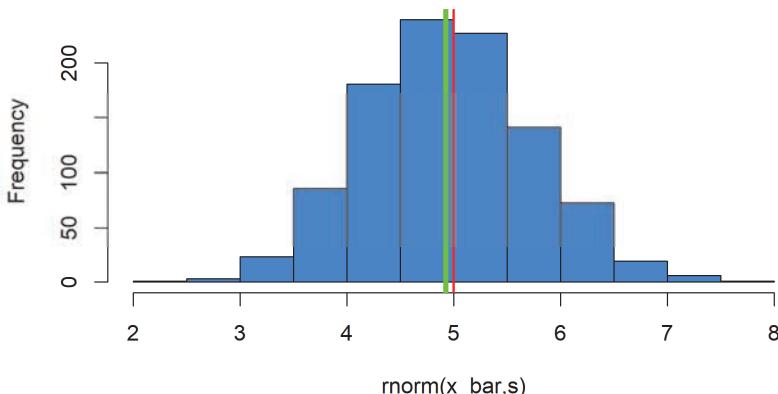


Figure 3-2. Example of a histogram showing approximate normality of the data

Looking at the histogram, the distribution appears to be approximately normal, as expected. We now calculate the population parameters and sample statistics.

```
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", sqrt((1/lambda)^2/n_dist)),
    "-infinity", "+infinity"), times=1))
knitr::kable(df)
```

Metric	Statistic	Parameter
mean	4.958859	5.000000
variance	0.632885	0.625000
stdev	0.795541	0.790569
minimum	2.432150	-infinity
maximum	7.880462	+infinity

## 3.7 Boxplot and Sample Statistics

The boxplot in **Figure 3-3** aids us in seeing the min, max, mean, quartiles, and outliers.

```
boxplot(mns_u, col="dodgerblue", horizontal=TRUE,  
       las=2, main="Boxplot for MNS", len=2,  
       boxcol="navyblue", medcol="orange",  
       whiskcol="green4", staplecol="blue",  
       outcol="purple")
```

Boxplot for MNS

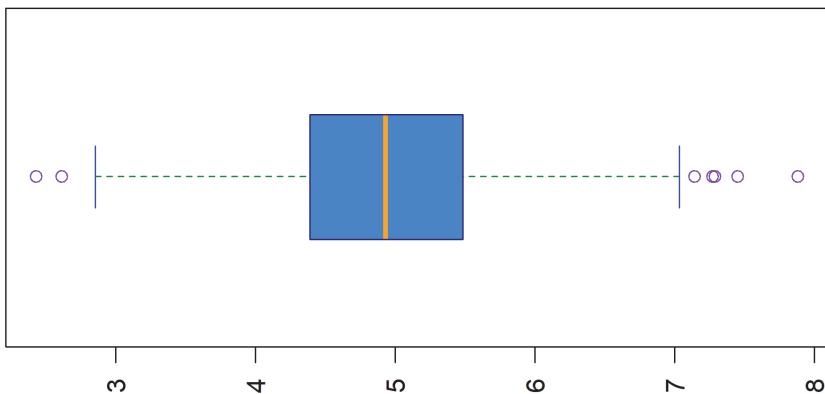


Figure 3-3. Illustration of the key features of a boxplot

```
summary(mns_u)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2.432	4.390	4.932	4.959	5.484	7.880

## 3.8 Summary

In this analysis we investigated an application of the CLT, using the means of 40 exponential random variates, approximating a standard normal distribution. It is important to state that the only definitive way of knowing that our distribution of sample mean is approximately normal is due to the CLT, and this hold true regardless of the underlying sample distributions. Note, we could also apply this to the sums of our exponential.

## 4 Improvised Explosive Device Analysis in Baghdad

*"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

### 4.1 Introduction

Sherlock's data may have been insufficient, but at present we cannot make the same complaint. If we have any burden whatsoever, then it is possessing too much data. In this chapter, will have two objectives: (1) visualizing spatial and temporal trends of terrorist improvised explosive devices (IEDs) activity based on data, (2) analyzing factors that may affect what may appear to be random behavior, and (3) turning the "randomness" into predictable patterns, based upon the data (see **Figure 4-1**). Taken together, we are really using R programming to cluster and describe data that the military combatant commands and local police departments have already collected and turned over to the skillful hands of the analyst.

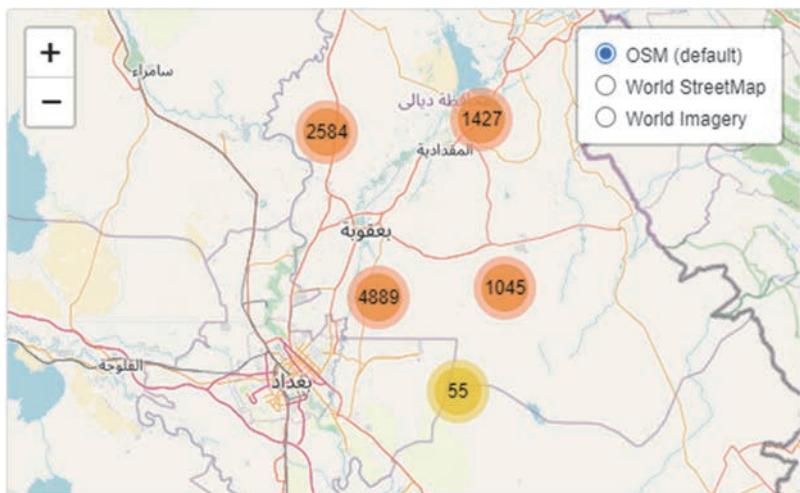


Figure 4-1. Five clusters of IED deployments around Baghdad

## 4.2 Install R libraries

```
if(!require(readr)) install.packages("readr")
if(!require(dplyr)) install.packages("dplyr")
if(!require(DT)) install.packages("DT")
if(!require(ggrepel)) install.packages("ggrepel")
if(!require(leaflet)) install.packages("leaflet")
if(!require(data.table)) install.packages("data.table")
if(!require(ggmap)) install.packages("ggmap")
if(!require(scales)) install.packages("scales")
```

## 4.3 The IED Data

The IED use data was simulated using distributions derived from an analysis of IEDs in their use in several foreign nations where there has been a US Military presence. The distributions were then applied to Baghdad Iraq using the inverse transform method. To simulate IED incidents in Baghdad, we used the reverse inverse method, which for the Uniform distribution is

$$X = a + (b - a)U,$$

where U is the Uniform[0,1] random number. In Excel, this is the RAND() function, so the inverse transform would be

$$X = a + (b - a) * RAND().$$

The IED are then geographically distributed in zones in a similar fashion.

The data was simulated due to the constraint that IED data was only available at the classified level. The data include the following factors:

- **IncidntNum** (N) Incident number
- **Category** (C) IED category, i.e., remote detonation, pressure detonated, etc.
- **Descript** (C) Additional descriptive details
- **DayOfWeek** (C)
- **Date** (D) Date: DD/MM/YYYY
- **Time** (T) Time: 24-hour system
- **PdDistrict** (C) military police district where incident occurred
- **Resolution** (C) Resolution of the criminal act
- **X** (N) Longitude

- **Y** (N) Latitude
- **Location** (C) Lat/long
- **PdId** (N) Military Department ID

(**N** = Numeric, **T** = Time, **D** = Date, **C** = Class)

## 4.4 Read the Data

First, we load the data using the `readr` package and `read_csv`.

```
library(readr)
path <- 
  "c:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\baghdad_ieds.csv"
df <- read_csv(path)
```

Rows: 499244 Columns: 13— Column specification

---

Delimiter: ","
chr (8): Category, Descript, DayOfWeek, Date, P...
dbl (4): IncidntNum, X, Y, PdId
time (1): Time

## 4.5 Display Data

Now, we display the data using the `DT` package and `datatable`. In our static view, we will not see the output here, since this produces an HTML interactive table

```
library(DT)
df_sub <- df[1:100,] # display the first 100 rows
df_sub$Time <- as.character(df_sub$Time)
#html viewable only
datatable(df_sub, options = list(pageLength = 5,
  scrollX='400px'))
```

## 4.6 Preprocess the Data

The All-Caps text is difficult to read. Let us force the text in the appropriate columns into proper case (see *Table 4-1*).

```
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, scrollX='400px'))
df %>% head(10)

```

A tibble: 10 × 13

*Table 4-1. Conversion from all caps to the proper cases*

IncidentNum <dbl>	Category <chr>	Descript <chr>
120944740	TimedIED	Vehicle Destruction
160476117	SuicideIED	Fatalities, Injuries
170276010	SuicideIED	Property Destruction
100533620	TimedIED	Civilian Injuries And Deaths
96030683	TimedIED	Civilian Injuries And Deaths
130142520	TimedIED	Infant Deaths
170441520	TimedIED	Infant Deaths
116154666	TimedIED	Vehicle Destruction
110445112	TimedIED	Vehicle Destruction
176053218	TimedIED	Vehicle Destruction

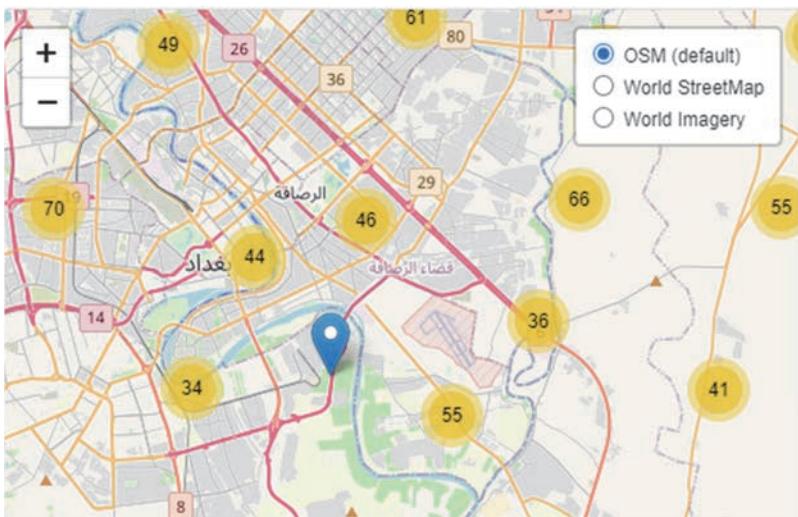
1-10 of 10 rows | 1-3 of 13 columns

## 4.7 Explore the Data

### 4.7.1 Crime Across Space

IED use can be modeled as a crime across space. 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 `addControl` (`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 IEDs, we can cluster them together `library(leaflet)` as shown in **Figure 4-2**.



*Figure 4-2. Zoomed-in portion of Baghdad from Figure 4-1*

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$Category,
"<br>", "<b>Description: </b>", data$Descript,
"<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. 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 in *Figure 4-3*.

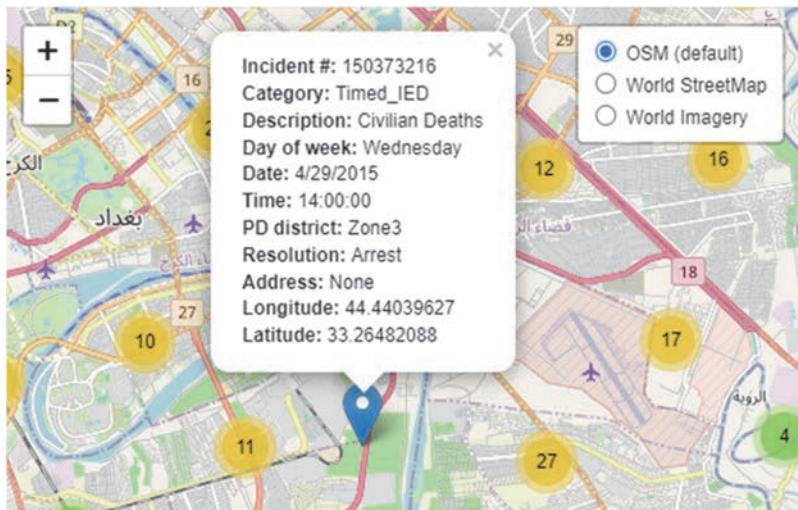


Figure 4-3. View of Figure 4-2 with popups activated

```
data$popup[1]
```

```
[1] "<b>Incident #: </b> 120944740 <br> <b>Category: </b>
Timed_IED <br> <b>Description: </b> Vehicle Destruction <br>
<b>Day of week: </b> Wednesday <br> <b>Date: </b> 11/21/2012
<br> <b>Time: </b> 17:50:00 <br> <b>PD district: </b> Zone3
<br> <b>Resolution: </b> Arrest <br> <b>Address: </b> None <br>
<b>Longitude: </b> 44.5915089 <br> <b>Latitude: </b>
33.52790997"
```

```
df_arrest <- df_arrest <- df %>% filter(grepl("Arrest",
                                                 Resolution))
df_arrest %>% head(10)
```

A tibble: 10 × 13

*Table 4-2. IED incident by category and outcomes*

IncidentNum <dbl>	Category <chr>	Description <chr>
120944740	Timed_IED	Vehicle Destruction
160476117	Suicide_IED	Fatalities, Injuries
170276010	Suicide_IED	Property Destruction
96030683	Timed_IED	Civilian Injuries And Deaths
110445112	Timed_IED	Vehicle Destruction
176053218	Timed_IED	Vehicle Destruction
156218874	Timed_IED	Vip Deaths
90831683	Suicide_IED	Vip Injuries
91166415	Suicide_IED	Vip Injuries
170511747	Remote_CONTROLLED	Detonated In Street

1-10 of 10 rows | 1-3 of 13 columns

```
sprintf("# of Rows in Dataframe: %s", nrow(df_arrest))
```

```
[1] "# of Rows in Dataframe: 269050"
```

```
sprintf("Dataframe Size: %s",
format(object.size(df_arrest), units = "MB"))
```

```
[1] "Dataframe Size: 28.8 Mb"
```

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)") %>% ...
```

## 4.7.2 IED Incidents 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.

```
df_arrest_daily <- df_arrest %>%  
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%  
  group_by(Date) %>%  
  summarize(count = n()) %>%  
  arrange(Date)  
  
df_arrest_daily %>% head(10)
```

A tibble: 10 × 2

*Table 4-3. Daily arrests count*

Date <date>	count <int>
2008-06-30	51
2008-07-01	80
2008-07-02	36
2008-07-03	48
2008-07-04	74
2008-07-05	69
2008-07-06	51
2008-07-07	44
2008-07-08	66
2008-07-09	51

1-10 of 10 rows

```
library(dplyr)  
df_ied_daily <- df %>%  
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%  
  group_by(Date) %>%  
  summarize(count = n()) %>%  
  arrange(Date)  
df_ied_daily %>% head(10)
```

A tibble:10 × 2

*Table 4-4. Daily IED incidents*

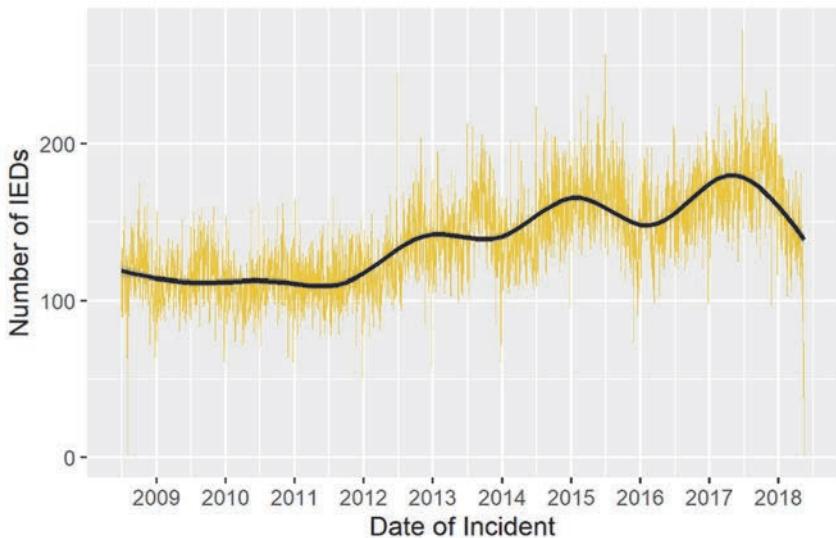
Date <date>	count <int>
2008-06-30	93
2008-07-01	126
2008-07-02	90
2008-07-03	95
2008-07-04	133
2008-07-05	119
2008-07-06	95
2008-07-07	90
2008-07-08	109
2008-07-09	93

1-10 of 10 rows

## 4.8 IED Incidents 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. However, the seasonality may follow a pattern from day to night, where IED incidents may increase during the day and fall off at night. Another scenario might involve increased IED detonation 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 Baghdad from 2009 - 2018")
plot
```



*Figure 4-4. Daily IED incidents in Baghdad from 2009 – 2018*

The trend shown by **Figure 4-4** shows a slight decrease in reported crimes up to the sharp rise beginning in 2017. Late summer of 2018 shows a possible increase. The crime series plot also shows consistent seasonality, at least up to the sharp increase where it is difficult to observe.

## 4.9 Aggregated Data

No, we can aggregate the data and create a table that summarizes the data by incident category. (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 `datatable` 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'))
```

Description:dt [5 × 4]

*Table 4-5. IED-type frequencies and percentages*

Category	Frequency	Percentage	options
<fctr>	<int>	<dbl>	<list>
TimedIED	338318	0.677660623	<chr [1]>
SuicideIED	120502	0.241368950	<chr [1]>
RemoteCONTROLLED	35481	0.071069457	<chr [1]>
PressureSWITCH	3711	0.007433239	<chr [1]>
PipeBOMB	1232	0.002467731	<chr [1]>

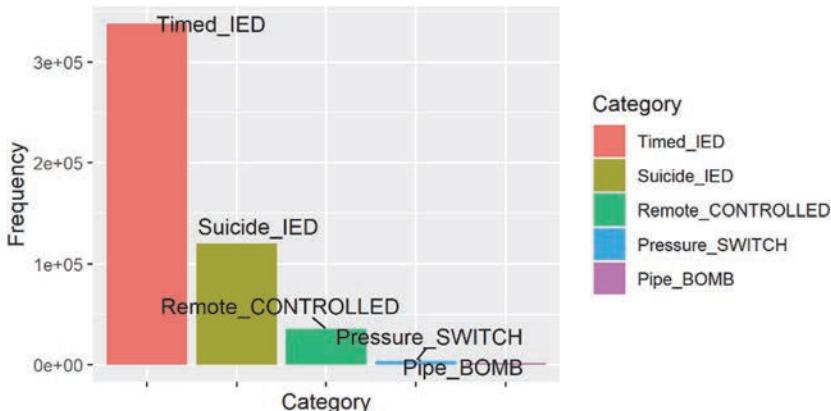
5 rows

The table shows the frequencies (counts) of the reported crimes with 10,000 or more occurrences, indicating that the most frequent IED type is timer detonated (non-vehicular and from the vehicle). With all the hype regarding the dangers of Baghdad, there were only 886 time-detonated IEDs in the 10-year period, or about 88 per year, and only 121 suicide-detonated IEDs for about 12 per year or 1 per month. I inflated the numbers to make a more interesting problem

## 4.10 Create a Bar Chart

Now that we can aggregate the data, we will show the data with a bar graph, depicted in *Figure 4-5*. The bar graph (or histogram) shows the frequencies of the IED Incidents recorded in *Table 4-5*. It makes it easy to see the vast difference between time-detonated IED and the other reported types.

```
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
```

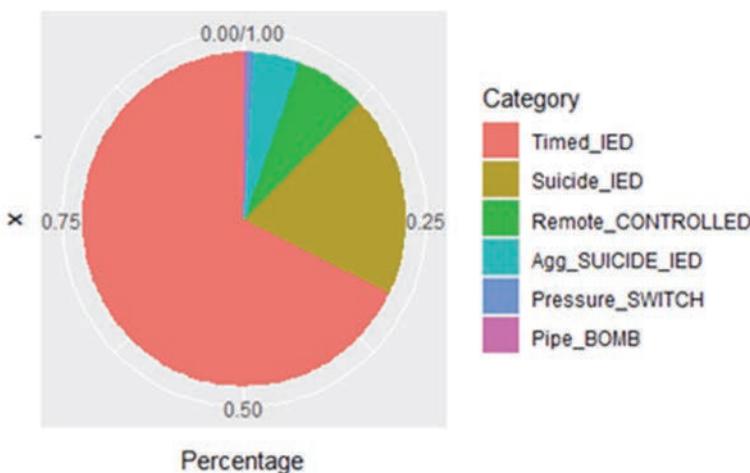


*Figure 4-5. Bar plot of IED type frequencies*

## 4.11 Create a pie chart

To further illustrate the IED incident data, a subsequent pie chart is plotted in *Figure 4-6*. The chart illustrates the same data as does the bar chart, but it may be more understandable in this instance. It shows that time-detonated IEDs occurs more than twice as much as all other reported IED types 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 4-6. Pie chart of IED type frequencies*

## 4.12 Temporal Trends

### 4.12.1 Theft Over Time

In this section, we create a chart of IED incidents over time. And for aesthetic effect as well as clarity, we make use color in depicting the series shown in **Figure 4-7**. This will provide us with a crime series for IEDs, which we will smooth as we did before.

```
library(dplyr)
df <- read.csv(path)
df_suicide <- df %>% filter(grepl("SUICIDE_IED",Category))
df_suicide_daily <- df_suicide %>%
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%
  group_by(Date) %>%
  summarize(count = n()) %>%
  arrange(Date)
plot <- ggplot(df_suicide_daily, aes(x=Date, count)) +
  geom_line(color = "#00ccff", size = 0.1) +
  geom_smooth(color = "#1A1A1A") +
  scale_x_date(breaks = date_breaks("1 year"),
  labels= date_format("%Y")) +
  labs(x = "Date of Suicide IED", y = "Number of
    Suicide IEDs", title = "Daily Suicide IEDs
    in Baghdad from 2008 to 2018")
plot
```

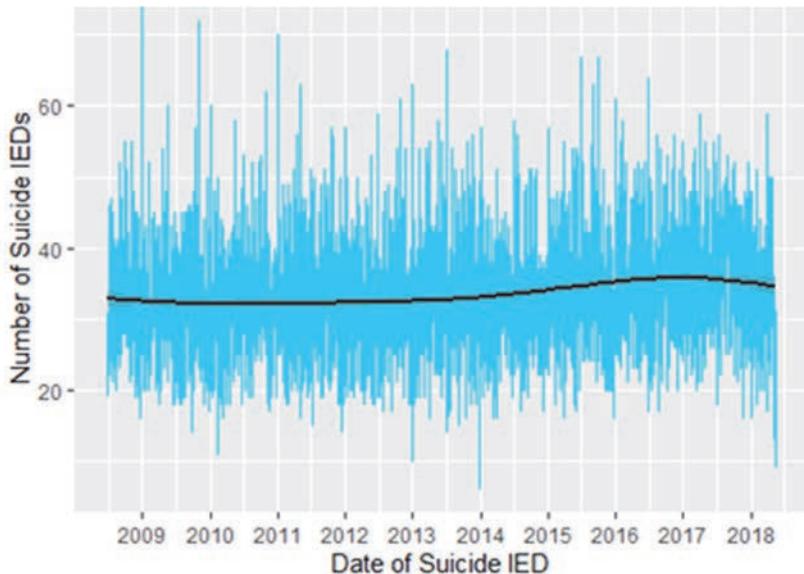


Figure 4-7. Daily suicide IEDs in Baghdad from 2008 to 2018

## 4.12.2 IED Incident Time Heatmap

Now, we aggregate counts of IED incidents by [Day-of-Week](#) and [Time](#) to create 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 [Baghdad\\_crime.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 built the function below, using the colon delimiter to separate hours from minutes. Then we build a table 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_suicide_time <- df_suicide %>%  
  mutate(Hour = sapply(Time, get_hour)) %>%  
  group_by(DayOfWeek, Hour) %>%  
  summarize(count = n())  
datatable(df_suicide_time, options=list  
  (scroll = '400px'))
```

## 4.12.3 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. 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_suicide_time$DayOfWeek <-  
  factor(df_suicide_time$DayOfWeek,  
  level = rev(dow_format))  
df_suicide_time$Hour <- factor(df_suicide_time$Hour,  
  level = 0:23, label = hour_format)  
knitr::kable(head(df_suicide_time , 11),  
  caption = 'Suicide IDEs Recorded by Times')
```

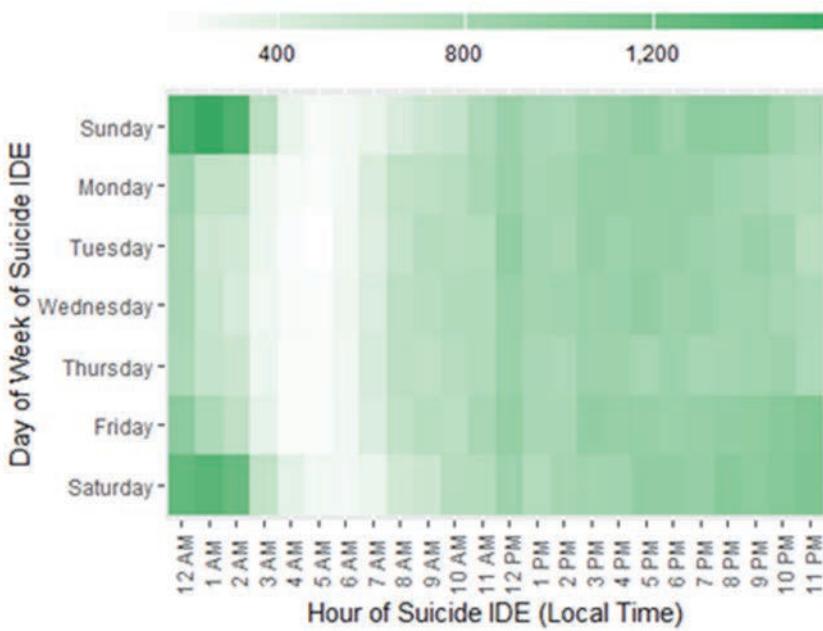
**Table 4-6. Suicide IDEs Recorded by Times**

DayOfWeek	Hour	count
Friday	12 AM	990
Friday	1 AM	743
Friday	2 AM	631
Friday	3 AM	338
Friday	4 AM	204
Friday	5 AM	193
Friday	6 AM	265
Friday	7 AM	423
Friday	8 AM	598
Friday	9 AM	679
Friday	10 AM	661

#### 4.12.4 Create Time Heatmap

Using our previous results, we build a “heatmap” and plot it with a red color scheme as seen in **Figure 4-8**. 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_suicide_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 Suicide IDE (Local Time)",
       y = "Day of Week of Suicide IDE",
       title = "Number of Suicide IDEs in Baghdad from
                 2008 to 2018, by Time of Suicide IDE") +
  scale_fill_gradient(low = "white", high = "#27AE60",
labels = comma)
plot
```



*Figure 4-8. Number of suicide IEDs in Baghdad from 2008 to 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 most likely 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.

#### 4.12.5 Arrest Over Time

Now, we create a chart 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.

```
# Returns the numeric hour component of a string formatted
# "HH:MM", e.g. "09:40" input returns 9
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())
df_arrest_time %>% head(10)

```

A tibble:10 × 3, Groups:DayOfWeek [1]

*Table 4-7. Arrest counts by ay of the week*

DayOfWeek <chr>	Hour <dbl>	count <int>
Friday	0	1768
Friday	1	1215
Friday	2	929
Friday	3	564
Friday	4	351
Friday	5	317
Friday	6	504
Friday	7	758
Friday	8	1130
Friday	9	1372

1-10 of 10 rows

```

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)
df_arrest_daily

```

#### 4.12.6 Daily Arrests

Next, we build the plot shown in *Figure 4-9* of daily IED incidents related arrests over time. This will provide us with another crime series, and we will smooth it as usual, noticing an upward trend in arrests over time until a sharp increase starting around 2017.

```

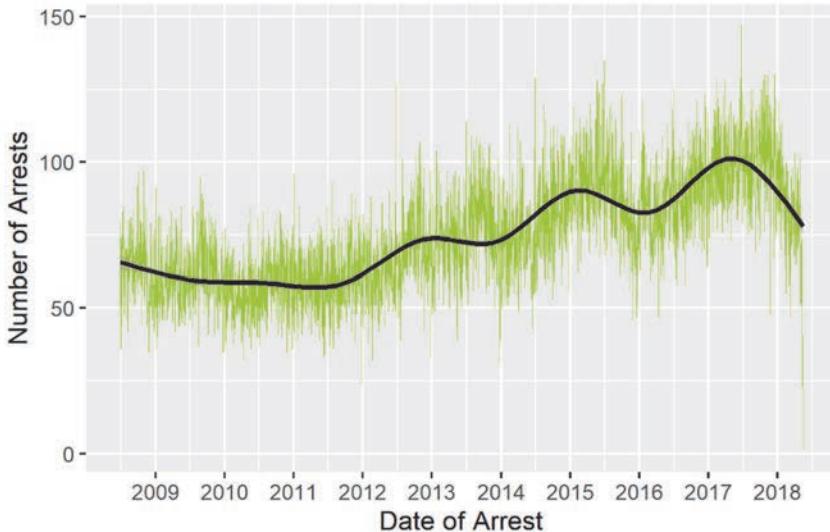
library(ggplot2)
library(scales)
plot <- ggplot(df_arrest_daily,aes(x= Date, y= count)) +
  geom_line(color = "#F2CA27", size = 0.1) +
  geom_smooth(color = "#1A1A1A") +
  scale_x_date(breaks = date_breaks("1 year"),

```

```

labels = date_format("%Y")) +
  labs(x = "Date of Arrest",
       y = "Number of Police Arrests", title = "Daily
Police Arrests in Baghdad from 2008 to 2018")
plot

```



*Figure 4-9. Daily IED related arrests in Baghdad from 2008-2018*

#### 4.12.7 Number of Arrest by Time of Arrest

Here, we again use the function we created that gets the hour from the time string in [Baghdad\\_crime.csv](#), so that we can use an approximate arrest time with the day of the week. This allows us to bin the incidents by hour, etc.

```

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)
```

Next, we use the out of the time extraction function to generate the number of IED arrests in Baghdad from 2009 - 2018, by time of arrest.

```
plot <- ggplot(df_arrest_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 Arrest (Local Time)",
       y = "Day of Week of Arrest",
       title = "Number of IED Arrests in Baghdad
                 from 2009 - 2018, by Time of Arrest") +
  scale_fill_gradient(low = "white",
                      high = "#008000", labels = comma)
plot
```

From the **Figure 4-10**, 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 the reporting times correspond with the shift changes.

Why is there a surge on Wednesday afternoon, and at 4-5PM on all days? Let us look at subgroups to verify there is not a latent factor.

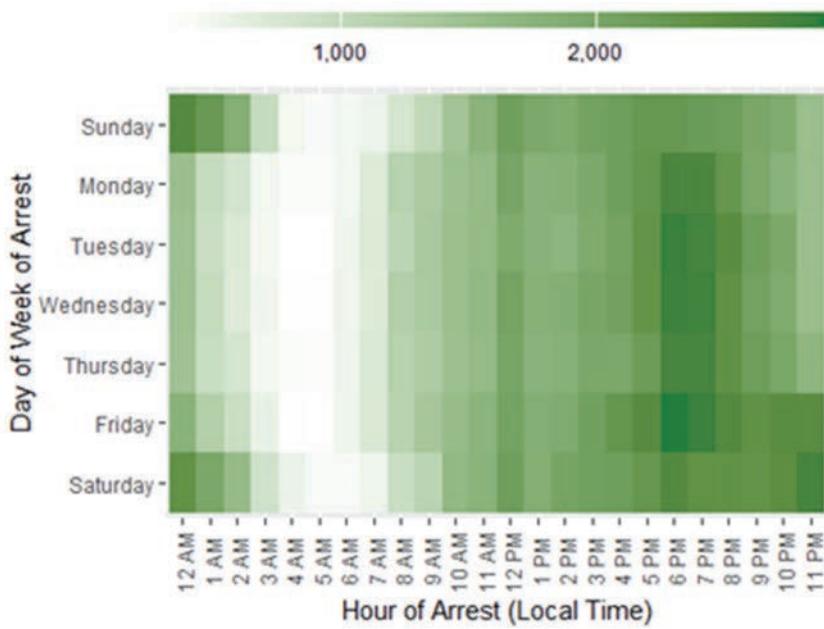


Figure 4-10. Number of arrests in Baghdad from 2008 – 2018

## 4.13 Correlation Analysis

**Correlation** is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It is a common tool for describing simple relationships without making a statement about cause and effect.

The sample **correlation coefficient**,  $r$ , quantifies the strength of the relationship. We test correlations for statistical significance.

Correlation cannot look at the presence or effect of other variables outside of the two being explored. Importantly, correlation does not tell us about cause and effect. Correlation also cannot accurately describe curvilinear relationships.

### 4.13.1 Factor by Crime Category

Certain types of IED use may be more time dependent. (e.g., more traffic violations when people leave work). While we are interested in IED frequencies as shown in **Table 4-9**, we can gain more information from the data. For instance, in the table below, we look at frequency of the IED category per hour.

```

df_top_crimes <- df_arrest %>%
  group_by(Category) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

df_top_crimes %>% head(20)

```

Description: dt [5 x 3]

*Table 4-8. High frequency IEDs by type*

Category <chr>	count <int>	options <list>
TIMEDIED	171532	<dbl [1]>
SUICIDEIED	74249	<chr [1]>
REMOTECONTROLLED	20226	<dbl [1]>
PRESSURESWITCH	2286	<chr [1]>
PIPEBOMB	757	<dbl [1]>

5 rows

Now we write the code to the category of the IED, the day it was used, the hour it was used, and the number of incidents.

```

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$Hour,
         level = 0:23, label = hour_format)
knitr::kable(head(df_arrest_time_crime,11),
            caption = ' Arrest frequency table by day-of-week  
and hour (time)')

```

*Table 4-9. Arrest frequency table by day-of-week and hour (time)*

Category <chr>	DayOfWeek <fctr>	Hour <fctr>	count <int>
PIPEBOMB	Friday	12 AM	1
PIPEBOMB	Friday	1 AM	5
PIPEBOMB	Friday	2 AM	2
PIPEBOMB	Friday	3 AM	3

Category <chr>	DayOfWeek <fctr>	Hour <fctr>	count <int>
PIPE_BOMB	Friday	4 AM	1
PIPE_BOMB	Friday	5 AM	1
PIPE_BOMB	Friday	6 AM	1
PIPE_BOMB	Friday	7 AM	3
PIPE_BOMB	Friday	8 AM	6
PIPE_BOMB	Friday	9 AM	9

1-10 of 10 rows

#### 4.13.2 Number of Arrests by Category and time of Arrest

In this section, we plot the number of IED related 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 4-11**. For each heatmap 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", title = "Number of Police Arrests in Baghdad from 2009 to 2018, by Category and Time of Arrest") +
  scale_fill_gradient(low = "#d7b4ff", high =
  "#24004b") +
  facet_wrap(~ Category, nrow = 6)
plot
```

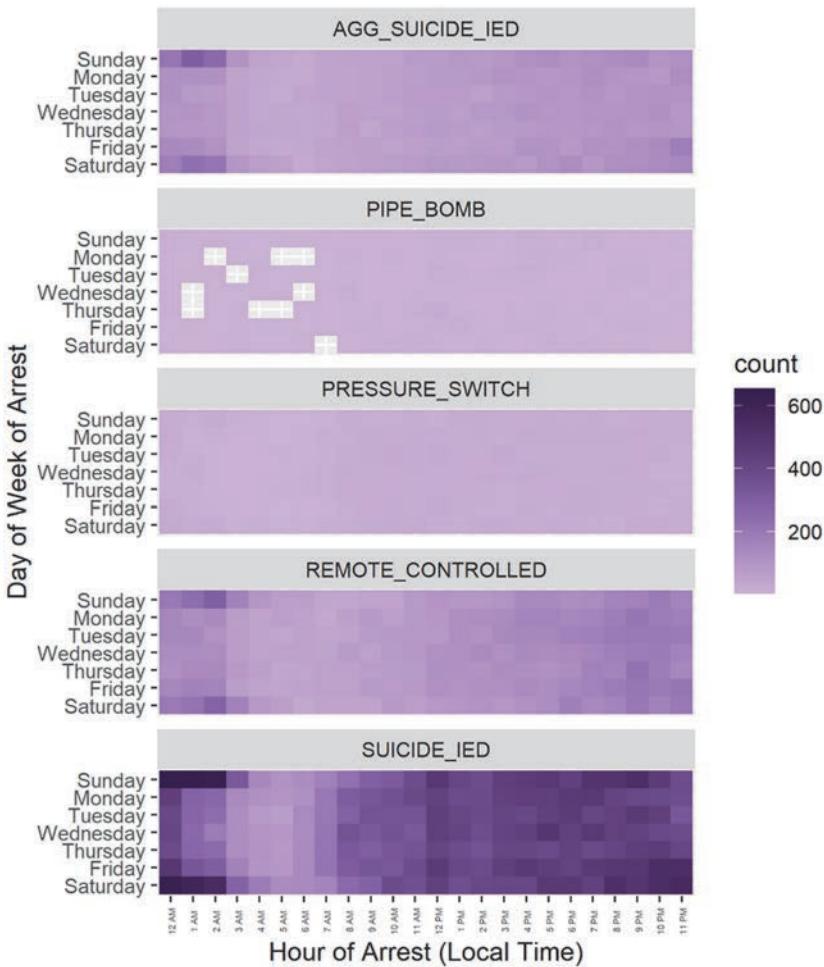


Figure 4-11. Number of IED related arrests in Baghdad from 2009-2018

```
plot <- ggplot(df_arrest_time_crime, aes(x = Hour, y =
  DayOfWeek, fill = count)) +
  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 = "# of Police Arrests in Baghdad from
  2003 - 2015, by Category and Time of Arrest") +
  scale_fill_gradient(low = "white", high = "#2980B9") +
  facet_wrap(~ Category, nrow = 6)
```

This graph looks good, but the gradients are not 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 **Figure 4-11** and the corresponding figure.

### 4.13.3 Normalized Gradients

```
df_arrest_time_crime <- df_arrest_time_crime %>%
  group_by(Category) %>%
  mutate(norm = count/sum(count))
knitr:::kable(head(df_arrest_time_crime, 11),
  caption = 'Crime Arrest Times')
```

*Table 4-10 IED related Arrest Times*

Category	DayOfWeek	Hour	count	norm
PIPE_BOMB	Friday	12 AM	1	0.0013210040
PIPE_BOMB	Friday	1 AM	5	0.0066050198
PIPE_BOMB	Friday	2 AM	2	0.0026420079
PIPE_BOMB	Friday	3 AM	3	0.0039630119
PIPE_BOMB	Friday	4 AM	1	0.0013210040
PIPE_BOMB	Friday	5 AM	1	0.0013210040
PIPE_BOMB	Friday	6 AM	1	0.0013210040
PIPE_BOMB	Friday	7 AM	3	0.0039630119
PIPE_BOMB	Friday	8 AM	6	0.0079260238
PIPE_BOMB	Friday	9 AM	9	0.0118890357

### 4.13.4 Normalized Number of Arrests by Category and Time

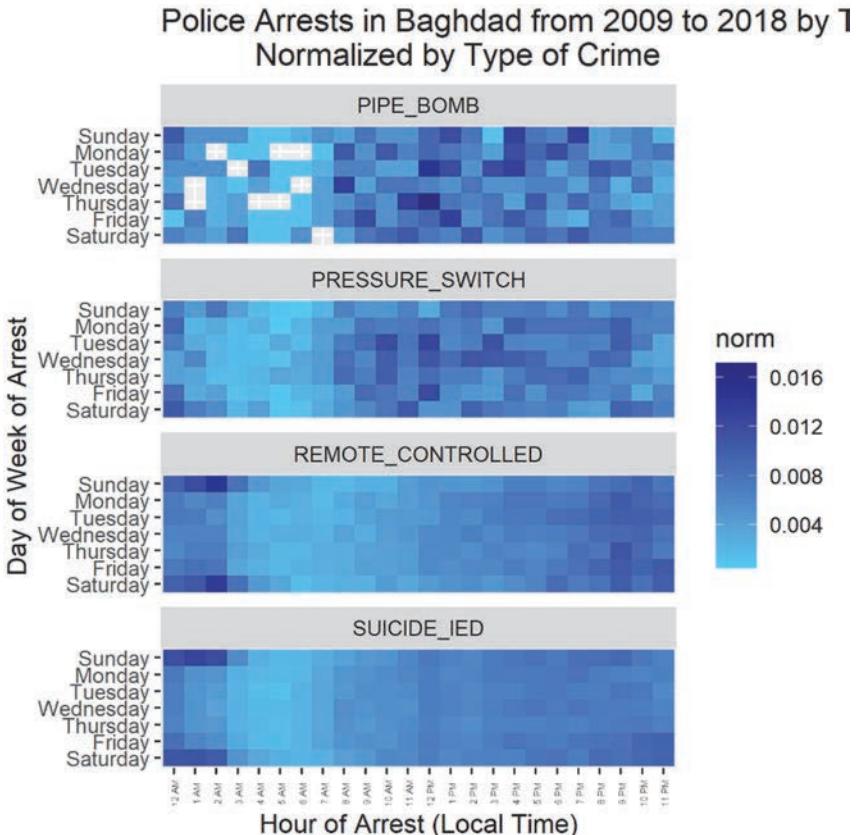
This view (see **Figure 4-12**) shows the distribution of IED related arrests by IED type over days of the week.

```
plot <- ggplot(df_arrest_time_crime, 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 Baghdad from 2009 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 4-12. IED related arrests in Baghdad from 2008 to 2018 IED normalized by type*

## 4.14 Factor Analysis

### 4.14.1 Factor by Police Zone

In this section, we plot like we did for Figure 4-10, but with a different scope. In **Table 4-11** and its corresponding **Figure 4-13**, 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 4-11. Crime Arrest Times by District*

PdDistrict	DayOfWeek	Hour	count	norm
Zone1	Friday	12 AM	184	0.0082559
Zone1	Friday	1 AM	90	0.0040382
Zone1	Friday	2 AM	78	0.0034998
Zone1	Friday	3 AM	49	0.0021986
Zone1	Friday	4 AM	30	0.0013461
Zone1	Friday	5 AM	40	0.0017948
Zone1	Friday	6 AM	45	0.0020191
Zone1	Friday	7 AM	79	0.0035447
Zone1	Friday	8 AM	83	0.0037241
Zone1	Friday	9 AM	136	0.0061022
Zone1	Friday	10 AM	131	0.0058779

#### 4.14.2 Factor by Police Zone

This view (see *Figure 4-13*) can aid us in allocating military and civil resource to a zone. This is especially useful when combined with a by-time view.

```

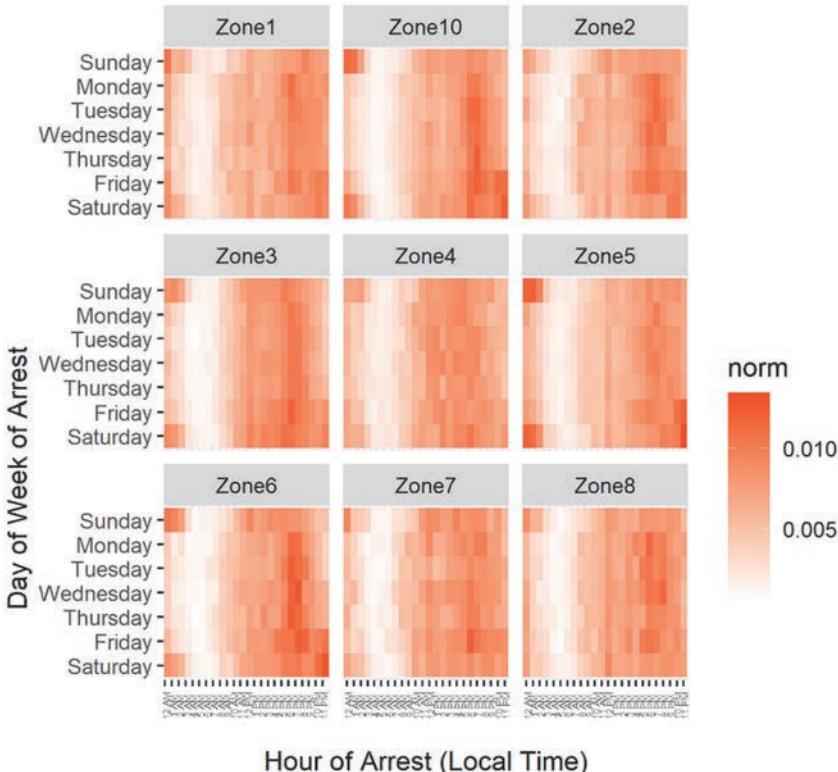
plot <- ggplot(df_arrest_time_district, aes(x = Hour, y =
DayOfWeek, fill = norm)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90,
                                   vjust = 0.6, size = 4)) +
  labs(x = "Hour of Arrest (Local Time)", y = "Day
of Week of Arrest", title = "Police Arrests in"

```

```

Baghdad from 2009 to 2018 by Time of Arrest,
Normalized by Station") +
scale_fill_gradient(low = "white", high ="#ff4500") +
facet_wrap(~ PdDistrict, nrow = 4)
plot

```



*Figure 4-13. IED arrests in Baghdad from 2009 to 2018*

#### 4.14.3 Factor by Month

We now look at factor by month. If IED incidents are related to activities, the period at which activities end may have an impact. For example, the Atlanta Olympic IED (1996) and the Boston Marathon IED (2013) are activities that influence usage of IEDs.

```

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))
```

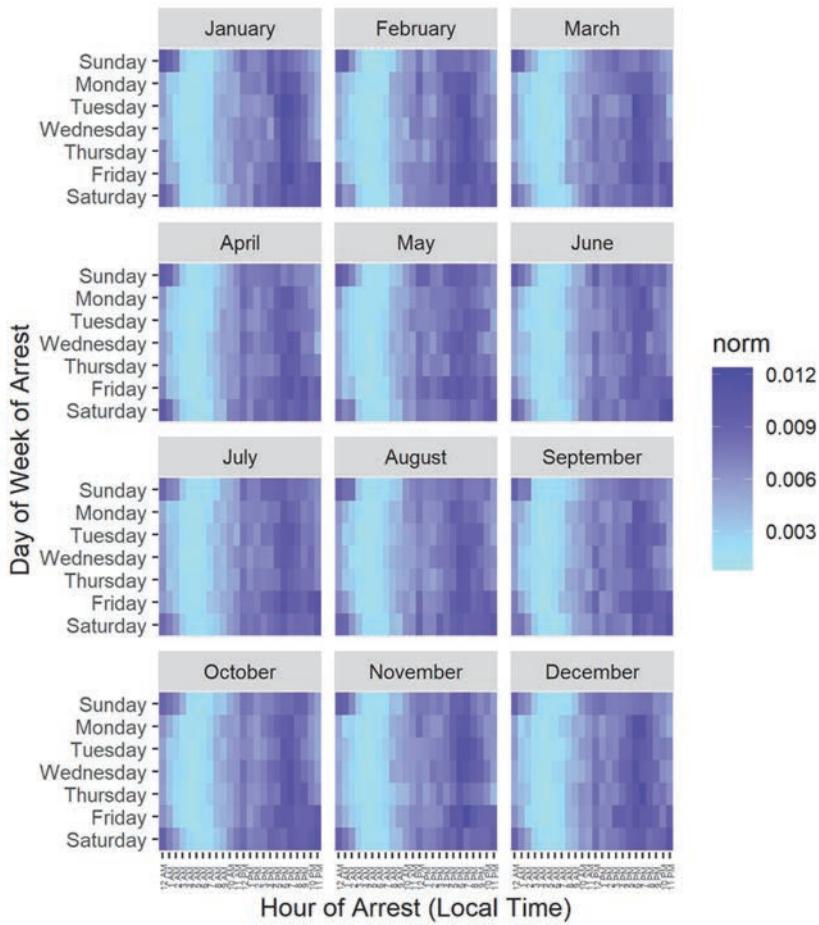
Here, we set order of month factors by chronological order instead of alphabetical.

```
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 in the **Figure 4-14**. The month by month distribution can aid in determining convoy routes, frequency of patrol, and so on.

```
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 Baghdad from 2008 to  
              2018 by Time of Arrest, Normalized by Month") +  
  scale_fill_gradient(low = "#9bfdff",  
                     high = "#4401ff") +  
  facet_wrap(~ Month, nrow = 4)  
plot
```

The viewpoint using **Figure 4-14** shows that the IED related arrests follow the same distribution over each month of the year.



*Figure 4-14. IED arrests in Baghdad from 2008 to 2018 normalized by month.*

#### 4.14.4 Factor By Year

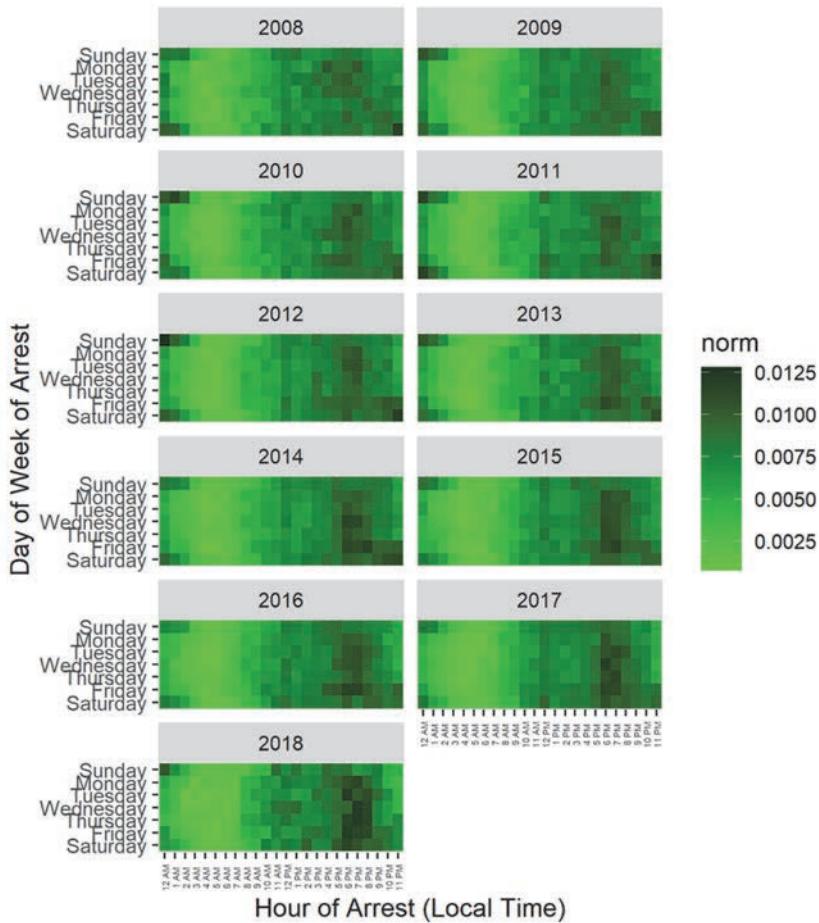
What if the situation changes overtime? We can look at the IED related incidents or arrests by year aggregated over day-of-week and time-of-day, as seen in **Figure 4-15**. This will provide a view that makes it easy to see changes, which in our case there are none

```
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$DayOfWeek <-  
  factor(df_arrest_time_year$DayOfWeek,  
         level = rev(dow_format))  
df_arrest_time_year$Hour <-  
  factor(df_arrest_time_year$Hour,  
         level = 0:23, label = hour_format)
```

Next, we plot arrest normalized by year.

```
plot <- ggplot(df_arrest_time_year, 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 Baghdad from 2014 to  
              2018 by Time of Arrest, Normalized by Year") +  
  scale_fill_gradient(low ="#01ff44", high ="#00340e") +  
  facet_wrap(~ Year, nrow = 6)  
plot
```



*Figure 4-15. IED Arrests in Baghdad from 2008 – 2018 normalized by Year*

## 4.15 Session Information

```
sessionInfo()
```

```
R version 4.3.3 (2024-02-29 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 19045)
```

```
Matrix products: default
```

```
locale:
```

```
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8
```

```

time zone: America/*****
tzcode source: internal

attached base packages:
[1] stats      graphics   grDevices utils
[5] datasets   methods    base

other attached packages:
[1] ggmap_4.0.0        scales_1.3.0
[3] leaflet_2.2.2     ggrepel_0.9.5
[5] ggplot2_3.5.0     data.table_1.15.4
[7] DT_0.33           dplyr_1.1.4
[9] readr_2.1.5

loaded via a namespace (and not attached):
[1] gtable_0.3.4        xfun_0.43
[3] bslib_0.7.0         ggRandomForests_2.2.1
[5] htmlwidgets_1.6.4   visNetwork_2.1.2
[7] lattice_0.22-5     tzdb_0.4.0
[9] bitops_1.0-7        leaflet.providers_2.0.0
[11] vctrs_0.6.5        tools_4.3.3
[13] crosstalk_1.2.1   generics_0.1.3
[15] parallel_4.3.3    tibble_3.2.1
[17] fansi_1.0.6        pkgconfig_2.0.3
[19] Matrix_1.6-5      RColorBrewer_1.1-3
[21] randomForestSRC_3.2.3 lifecycle_1.0.4
[23] stringr_1.5.1     farver_2.1.1
[25] compiler_4.3.3    munsell_0.5.1
[27] data.tree_1.1.0    htmltools_0.5.8.1
[29] sass_0.4.9         yaml_2.3.8
[31] pillar_1.9.0       crayon_1.5.2
[33] jquerylib_0.1.4   tidyverse_1.3.1
[35] cachem_1.0.8      neuralnet_1.44.2
[37] nlme_3.1-164       tidyselect_1.2.1
[39] digest_0.6.35     stringi_1.8.3
[41] purrr_1.0.2        labeling_0.4.3
[43] splines_4.3.3     fastmap_1.1.1
[45] grid_4.3.3         colorspace_2.1-0
[47] cli_3.6.2          magrittr_2.0.3
[49] DiagrammeR_1.0.11 randomForest_4.7-1.1
[51] survival_3.5-8    utf8_1.2.4
[53] withr_3.0.0        bit64_4.0.5
[55] httr_1.4.7         rmarkdown_2.26
[57] jpeg_0.1-10        bit_4.0.5
[59] gridExtra_2.3      png_0.1-8
[61] hms_1.1.3          evaluate_0.23
[63] knitr_1.46         mgcv_1.9-1
[65] rlang_1.1.3         Rcpp_1.0.12
[67] glue_1.7.0          rstudioapi_0.16.0
[69] vroom_1.6.5         jsonlite_1.8.8

```

[71] R6\_2.5.1

plyr\_1.8.9



## 5 Multiple Linear Regression

### 5.1 Notation for the Population Model

A population model for a multiple linear regression model that relates a  $y$ -variable to  $p - 1$   $x$ -variables is written as

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_{p-1} x_{i,p-1} + \varepsilon_i$$

We assume that the  $\varepsilon_i$  have a normal distribution with mean 0 and constant variance  $\sigma^2$ . These are the same assumptions that we used in simple regression with one  $x$ -variable.

The subscript  $i$  refers to the  $i^{th}$  individual or unit in the population. In the notation for the  $x$ -variables, the subscript following  $i$  simply denotes which  $x$ -variable it is.

The word “linear” in “multiple linear regression” refers to the fact that the model is *linear in the parameters*,  $\beta_0, \beta_1, \dots, \beta_{p-1}$ . This simply means that each parameter multiplies an  $x$ -variable, while the regression function is a sum of these “parameter times  $x$ -variable” terms. Each  $x$ -variable can be a predictor variable or a transformation of predictor variables (such as the square of a predictor variable or two predictor variables multiplied together). Allowing non-linear transformation of predictor variables like this enables the multiple linear regression model to represent non-linear relationships between the response variable and the predictor variables. Note that even  $\beta_0$  represents a “parameter times  $x$ -variable” term if you think of the  $x$ -variable that is multiplied by  $\beta_0$  as being the constant function “1.”

The model includes  $p - 1$   $x$ -variables, but  $p$  regression parameters (beta) because of the intercept term  $\beta_0$ .

### 5.2 Estimates of the Model Parameters

The estimates of the  $\beta$  parameters are the values that minimize the **sum of squared errors** for the sample. The exact formula for this is given in the next section on matrix notation.

We use the letter  $b$  to represent a sample estimate of a parameter  $\beta$ . Thus,  $b_0$  is the sample estimate of  $\beta_0$ ,  $b_1$  is the sample estimate of  $\beta_1$ , and so on.

$MSE = SSE/(n - p)$  (**mean square error**) estimates  $\sigma^2$ , the variance of the errors. In the formula,  $n$  = sample size,  $p$  = number of parameters in the model (including the intercept) and  $SSE$  = sum of squared errors. Notice that for simple linear regression  $p = 2$ . Thus, we get the formula for MSE that we introduced in that context of one predictor.

$S = \sqrt{MSE}$  or RMSE (**root mean square error**) estimates  $\sigma$  and is known as the regression standard error or the **residual standard error**. In the case of two predictors, the estimated regression equation yields a plane (as opposed to a line in the simple linear regression setting). For more than two predictors, the estimated regression equation yields a **hyperplane**.

## 5.3 Interpretation of the Model Parameters

Each  $\beta$  parameter represents the change in the mean response,  $E(y)$ , per unit increase in the associated predictor variable when we hold all the other predictors constant. For example,  $\beta_1$  represents the estimated change in the mean response,  $E(y)$ , per unit increase in  $x_1$  when we hold  $x_2, x_3, \dots, x_{p-1}$  constant. The intercept term,  $\beta_0$ , represents the **estimated mean response**,  $E(y)$ , when all the predictors  $x_1, x_2, \dots, x_{p-1}$ , are all zero (which may or may not have any practical meaning).

## 5.4 Predicted Values and Residuals

A predicted value is calculated as  $\hat{y}_i = b_0 + b_1x_{i,1} + b_2x_{i,2} + \dots + b_{p-1}x_{i,p-1}$ , where the  $b$  values come from statistical software and the  $x$ -values are specified by us.

A **residual (error)** term is calculated as  $e_i = y_i - \hat{y}_i$ , the difference between an actual and a predicted value of  $y$ .

A plot of residuals (vertical) versus predicted values (horizontal) ideally should resemble a horizontal random band. Departures from this form indicates difficulties with the model and/or data.

We can perform other residual analyses the same way we did in simple regression. For instance, we might wish to examine a normal probability plot (NPP) of the residuals. Additional plots to consider are plots of

residuals versus each  $x$ -variable separately. This might help us identify sources of curvature or nonconstant variance.

### ANOVA Table

Source	df	SS	MS	F
Regression	$p - 1$	SSR	$MSR = SSR/(p - 1)$	$MSR/MSE$
Error	$n - p$	SSE	$MSE = SSE/(n - p)$	
Total	$n - 1$	SSTO		

## 5.5 Coefficient of Determination, R-squared, and Adjusted R-squared

As in simple linear regression,  $R^2 = \frac{SSR}{SSTO} = 1 - \frac{SS}{SSTO}$ , and represents the proportion of variation in  $y$  (about its mean) “explained” by the multiple linear regression model with predictors,  $x_1, x_2, \dots$ . If we start with a simple linear regression model with one predictor variable,  $x_1$ , then add a second predictor variable,  $x_2$ , SSE will decrease (or stay the same) while SSTO remains constant, and so  $R^2$  will increase (or stay the same). In other words,  $R^2$  always increases (or stays the same) as more we add more predictors to a multiple linear regression model, even if the predictors added are unrelated to the response variable. Thus, by itself, we cannot use  $R^2$  to help us identify which predictors we should include in a model, and which we should exclude.

An alternative measure adjusted  $R^2$ , since adding predictors does not necessarily increase  $R^2$ , and can help us identify which predictors we should include in a model and which we should exclude. The **adjusted R-squared** is  $R^2 = 1 - \left(\frac{n-1}{n-p}\right)(1 - R^2)$ , and, while it has no practical interpretation, is useful for such model building purposes. Simply stated, when comparing two models used to predict the same response variable, we prefer the model with the higher value of adjusted  $R^2$ .

## 5.6 Significance Testing of Each Variable

Within a multiple regression model, we may want to know whether a particular  $x$ -variable is making a useful contribution to the model. That is, given the presence of the other  $x$ -variables in the model, does a

particular  $x$ -variable help us predict or explain the  $y$ -variable? For instance, suppose that we have three  $x$ -variables in the model. The general structure of the model could be

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_3 x_3 + \varepsilon$$

As an example, to determine whether variable  $x_1$  is a useful predictor variable in this model, we could assess

$$H_0: \beta_1 = 0$$

$$H_A: \beta_1 \neq 0$$

If the null hypothesis above were the case, then a change in the value of  $x_1$  would not change  $y$ , so  $y$  and  $x_1$  are not linearly related (considering  $x_2$  and  $x_3$ ). Also, we would still have variables present in the model. When we cannot reject the null hypothesis above, we should say that we do not need variable  $x_1$  in the model given that variables  $x_2$  and  $x_3$  will remain in the model. In general, the interpretation of a slope in multiple regression can be tricky. Correlations among the predictors can change the slope values dramatically from what they would be in separate simple regressions.

To conduct the test, statistical software will report  $p$ -values for all coefficients in the model. Each  $p$ -value will be based on a  $t$ -statistic calculated as

$$t^* = \frac{(\text{sample coefficient} - \text{hypothesized value})}{(\text{standard error of coefficient})}$$

For our example above, the  $t$ -statistic is:

$$t^* = \frac{b_1 - 0}{se(b_1)} = \frac{b_1}{se(b_1)}$$

Note that the hypothesized value is usually just zero, so this portion of the formula is often omitted.

Multiple linear regression, in contrast to simple linear regression, involves multiple predictors and so evaluating each variable can quickly become complicated. For example, suppose we apply two separate tests for two predictors, say  $x_1$  and  $x_2$ , and both tests have high  $p$ -values. One test suggests that the model does not need  $x_1$  with all the other

predictors included, while the other test suggests the model does not need  $x_2$  with all the other predictors included. But this does not necessarily mean that we need both  $x_1$  and  $x_2$  in a model with all the other predictors included. It may well turn out that we would do better to omit either  $x_1$  or  $x_2$  from the model, but not both.

## 5.7 Multiple Linear Regression: Satellite Survivability

Aside from potential nation-state threats, what factors might affect the survivability of resident space objects (i.e., satellites)? We constructed a database comprised of ten factors and a response variable using simulation. The data is purely hypothetical. The satellites used may not represent reality, although the names are real. The file named `Survive.csv` contains our data for this example.

## 5.8 Model 1: Linear Model with Numeric Features

### 5.8.1 Data Preprocessing

First, we need to perform several preprocessing tasks on the data to prepare it for modeling.

### 5.8.2 Load Required R Packages.

First we load the R packages that we anticipate needing. We may find that we need additional packages later.

```
#install.packages("rsq").
library(rsq)
library(ggplot2)
library(hrbrthemes)
```

### 5.8.3 Import the Data

Next, we will load the comma delimited data using the `read.csv` function and print the first six rows, including the headings.

```
train <-
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\Survive.csv"
)
head(train, 6)
```

	RSO_ID	RSO_Weight	RSO_Density	RSO_Visibility	RSO_Name	RSO_MRP
1	FDR37	NA	Regular	0.065928735	Worldview	283.0292

```

2 FDX09      9.00      Low   0.065515067   CBERS 278.1370
3 FDF04      17.50     High  0.013692598   PROBA 269.7304
4 FDA15      9.30      Low   0.016087659   Landsat 269.6092
5 FDA38      5.44      Low   0.025583714   Landsat 269.1538
6 FDX13      NA        Low   0.047551568   Pleiades 269.1092
  Orbit_ID Orbit_Estab_Year Orbit_Height Stealth_Type RSO_Type
1 OUT027      2009       MEO   Stealth_4 RSO_Type3
2 OUT018      2009       MEO   Stealth_4 RSO_Type2
3 OUT018      2009       MEO   Stealth_4 RSO_Type2
4 OUT018      2009       MEO   Stealth_4 RSO_Type2
5 OUT018      2009       MEO   Stealth_4 RSO_Type2
6 OUT027      2007       MEO   Stealth_4 RSO_Type3
Survivability
1 8209.3140
2 1058.6220
3 3616.6256
4 5976.2208
5 480.7076
6 8217.3036

```

In the dataset, we see characteristics of the RSOs (Weight, Visibility, etc.), characteristics of their orbits (Height, Type, etc.), and a survivability score. We want to predict survivability based on these “features.”

### 5.8.4 Fill Missing Values

Now, let us impute missing values (i.e., fill missing spaces) with “NA”.

```
train[train==""] <- NA # Filling blank values with NA.
names(train)
```

```
[1] "RSO_ID"           "RSO_Weight"        "RSO_Density"
"RSO_Visibility"
[5] "RSO_Name"         "RSO_MRP"          "Orbit_ID"
"Orbit_Estab_Year"
[9] "Orbit_Height"     "Stealth_Type"      "RSO_Type"
"Survivability"
```

### 5.8.5 Create Dataset with Numeric Features

Next, let us create two datasets, one with the numeric independent variables,  $X$ , and one with the dependent variable or response,  $Y$ .  $X$  will be comprised of columns 4, 6 and 8, or `RSO_Visibility`, `RSO_MRP` and `Orbit_Estab_Year`, respectively.  $Y$  is the response, `Survivability`.

We will also print the names of the variables in each set to confirm we have the right ones.

```
X <- train[c(4,6,8)]  
names((X))  
  
[1] "RSO_Visibility"    "RSO_MRP"           "Orbit_Estab_Year"  
  
Y <- train[c(12)] #Storing the dependent variable.  
names((Y))
```

```
[1] "Survivability"
```

## 5.8.6 Train-Test Split

Here, we split the datasets,  $X$  and  $Y$ , into two sets each. The training datasets will be comprised of 70% of the data from each set, and the **cross-validation** sets will be comprised of the remaining 30%.

```
set.seed(567)  
part <- sample(2, nrow(X), replace = TRUE, prob = c(0.7,  
0.3))  
X_train <- X[part == 1,]  
X_cv <- X[part == 2,]  
  
Y_train <- Y[part == 1,]  
Y_cv <- Y[part == 2,]
```

## 5.8.7 Model 1 Construction

We will train **Model 1** using two of the numeric features, `RSO_MRP` and `Orbit_Estab_Year`. We will also exclude the intercept term by adding `+0` to the regression formula.

```
model1 <- lm(Y_train ~ RSO_MRP + Orbit_Estab_Year + 0,  
data = X_train )  
summary(model1)
```

```
Call:  
lm(formula = Y_train ~ RSO_MRP + Orbit_Estab_Year + 0, data =  
X_train)
```

```
Residuals:  
      Min        1Q    Median        3Q       Max  
-3738.6   -771.5   -53.9    722.4   7969.8
```

```

Coefficients:
Estimate Std. Error t value Pr(>|t|)
RSO_MRP      15.88534   0.29474  53.895 <2e-16 ***
Orbit_Estab_Year -0.02804   0.02266  -1.238    0.216
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1389 on 5954 degrees of freedom
Multiple R-squared:  0.7474, Adjusted R-squared:  0.7473
F-statistic:  8806 on 2 and 5954 DF,  p-value: < 2.2e-16

```

## 5.8.8 Extract Model 1 Equation

Next, we use the R function `extract_eq` from the `equatiomatic` package, to extract the `Model 1` equation.

```

library(equatiomatic)
equatiomatic::extract_eq(model1)

```

$$Y_{\text{train}} = \beta_1(\text{RSO\_MRP}) + \beta_2(\text{Orbit\_Estab\_Year}) + \epsilon$$

## 5.8.9 Predict using Model 1

Now, we use the model and the cross-validation set to make predictions.

```

predict_1 <- predict(model1, X_cv) #Predicting the
values.

```

## 5.8.10 Calculate Model 1 Metrics

Next, we calculate the Model 1 performance metrics, RMSE and R-squared.

Now, let us calculate MSE, based on the model prediction, and R-squared based on the model. MSE or Mean Square Error,  $R^2$  is a metric that determines how much of the total variation in  $Y$  (dependent variable) is explained by the variation in  $X$  (independent variable). Mathematically, we can write  $R^2$  as:

$$R^2 = 1 - \left( \frac{\sum(Y_{\text{Actual}} - Y_{\text{Predicted}})^2}{\sum(Y_{\text{Actual}} - Y_{\text{Mean}})^2} \right)$$

The value of  $R^2$  is always between 0 and 1, where 0 means that the model does not explain any variability in the response variable and 1 meaning it explains full variability in the response variable.

**Root Mean Square Error:** In R, the root mean square error (RMSE) allows us to measure how far predicted values are from observed values in a regression analysis.

In other words, how concentrated the data around the line of best fit.

$$RMSE = \sqrt{\frac{\sum(P_i - O_i)^2}{n}}$$

where:  $\Sigma$  symbol indicates “sum”

$P_i$  is the predicted value for the  $i^{th}$  observation in the dataset

$O_i$  is the observed value for the  $i^{th}$  observation in the dataset

$n$  is the sample size

Now let us calculate these model metrics.

```
library(Metrics)
RMSE <- rmse(Y_cv, predict_1)
R.sq <- rsq(model1)
print(paste("R-square =", R.sq))
```

```
[1] "R-square = 0.747352678970367"
```

```
print(paste("RMSE =", RMSE))
```

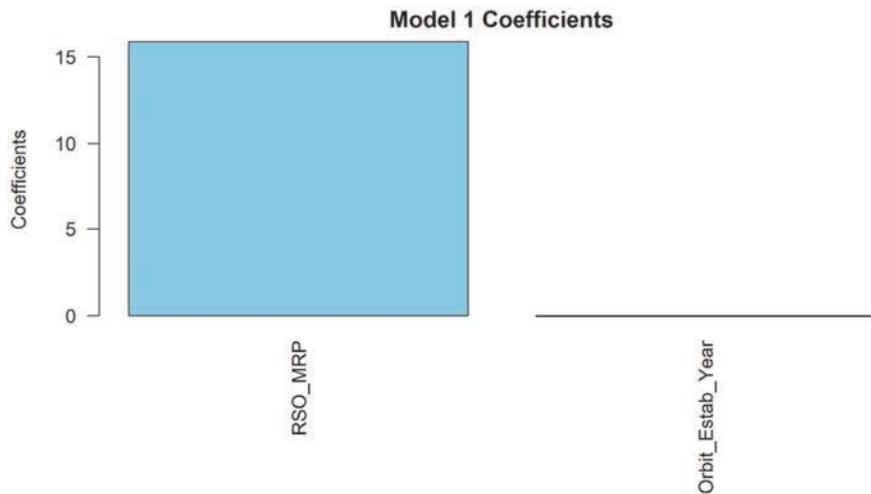
```
[1] "RMSE = 1411.26879741968"
```

We will use MSE for [Model 1](#) for comparison with subsequent models. In this case,  $R^2$  is 0.75, meaning, *year of establishment* and *MRP* explains 75% of variance in survivability. In other words, if we know year of establishment and the MRP, we will have 75% of the information to make an accurate prediction about survivability.

### 5.8.11 Plot Model 1 Coefficients

Finally, we plot the [Model 1](#) coefficient values using a [barplot](#). Let us look at the coefficients of this linear regression model. Since [RSO\\_MRP](#) has a high coefficient, having higher *RSO MRP* scores contributes to higher survivability.

```
par(mar=c(11,4,2,1))
barplot(model1$coefficients, main= "Model 1 Coefficients",
        ylab = "Coefficients", las = 2, col = 'skyblue')
```



*Figure 5-1. Bar plot for Model 1 coefficients*

## 5.9 Model 2: Linear Model with Numeric Features

We will train [Model 1](#) using two of the numeric features, `RSO_MRP` and `Orbit_Estab_Year` as before with the intercept excluded, but we add `RSO_Visibility`.

```
model2 <- lm(Y_train ~ RSO_MRP + Orbit_Estab_Year +
RSO_Visibility + 0, data = X_train )
summary(model1)
```

```
Call:
lm(formula = Y_train ~ RSO_MRP + Orbit_Estab_Year + 0, data =
X_train)

Residuals:
    Min      1Q  Median      3Q      Max 
-3738.6  -771.5   -53.9   722.4  7969.8 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
RSO_MRP       15.88534    0.29474  53.895 <2e-16 ***
Orbit_Estab_Year -0.02804    0.02266   -1.238    0.216  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1389 on 5954 degrees of freedom  
Multiple R-squared:  0.7474, Adjusted R-squared:  0.7473  
F-statistic: 8806 on 2 and 5954 DF,  p-value: < 2.2e-16
```

The model summary shows that all three features are significant as predictors, so let us get some predictions and check the MSE.

### 5.9.1 Extract Model 2 Equation

Next, we use the R function `extract_eq` from the `equatiomatic` package, to extract the `Model 2` equation.

```
library(equatiomatic)  
equatiomatic::extract_eq(model2)
```

$$Y_{train} = \beta_1(\text{RSO\_MRP}) + \beta_2(\text{Orbit\_Estab\_Year}) + \beta_3(\text{RSO\_Visibility}) + \epsilon$$

### 5.9.2 Predict using Model 2

Now, we use the model and the cross-validation set to make predictions.

```
predict_2 <- predict(model2, X_cv) #Predicting the  
values.
```

### 5.9.3 Calculate Model 2 Metrics

Next, we calculate the `Model 2` performance metrics, `RMSE` and `R-squared`.

```
library(Metrics)  
RMSE <- rmse(Y_cv, predict_2)  
R.sq <- rsq(model2)  
print(paste("R-square =", R.sq))
```

```
[1] "R-square = 0.752688015803263"
```

```
print(paste("RMSE =", RMSE))
```

```
[1] "RMSE = 1391.24486638781"
```

### 5.9.4 Model 2 Performance

The RMSE of 1391.245 did not change very much but is slightly smaller than **1411.269** from `Model 1`. `Model 2` introduces slightly more error.

$R^2$  is 0.75 and about the same as Model 1's. Moreover, we have learned from the residuals vs. fitted value plots that the error variance is increasing implying that the regression model is not a good one. So, back to the drawing board.

### 5.9.5 Plot Model 2 Coefficients

Finally, we plot the Model 2 coefficient values using a barplot.

```
par(mar=c(11,4,2,1))
barplot(model2$coefficients, main = "Model 1
Coefficients",
       ylab = "Coefficients", las = 2, col = 'skyblue')
```

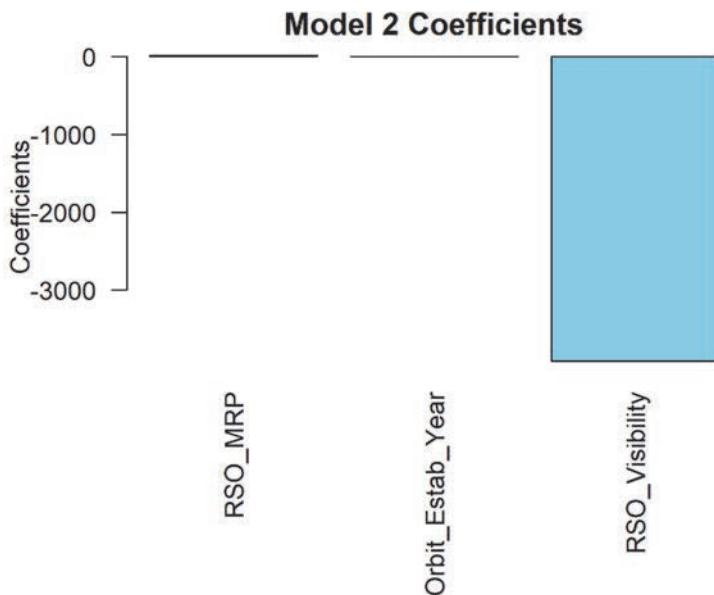


Figure 5-2. Bar plot for Model 2 coefficients

### 5.9.6 Model 2 Residual Analysis

Recall that residuals are differences between the one-step-predicted output from the model and the measured output from the validation data set. Thus, residuals represent the portion of the validation data not explained by the model.

### 5.9.7 Residual Plots

Residual analysis plots show different information depending on whether you use time-domain or frequency-domain input-output validation data. For frequency-domain validation data, the plot shows the following two axes:

1. Estimated power spectrum of the residuals for each output
2. Transfer-function amplitude from the input to the residuals for each input-output pair

For time-domain validation data, the plot shows the following two axes:

1. Autocorrelation function of the residuals for each output
2. Cross-correlation between the input and the residuals for each input-output pair

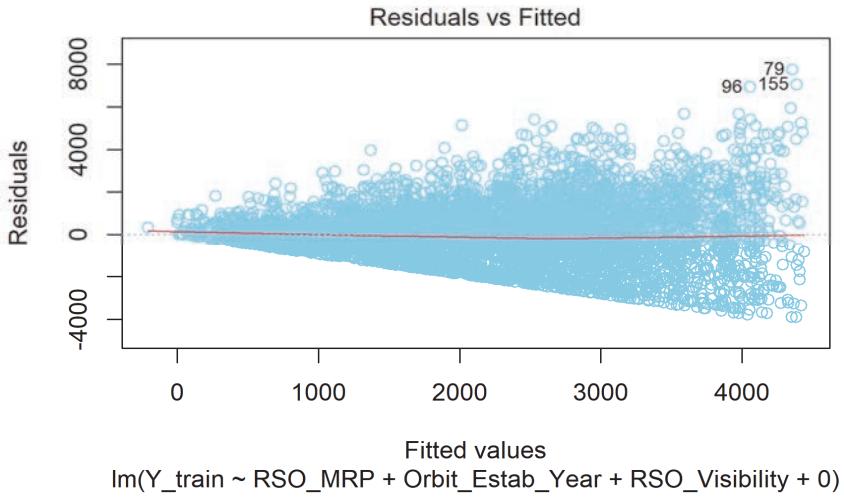
For linear models, you can estimate a model using time-domain data, and then validate the model using frequency domain data.

### 5.9.8 Displaying the Confidence Interval

The confidence interval corresponds to the range of residual values with a specific probability of being statistically insignificant for the system. If the condition of normality is not present, then we cannot calculate a confidence interval with any accuracy. So, before going down a dead-end street, we will plot and analyze the residuals, which if the data were [approximately] normal, would allow us to assume normality.

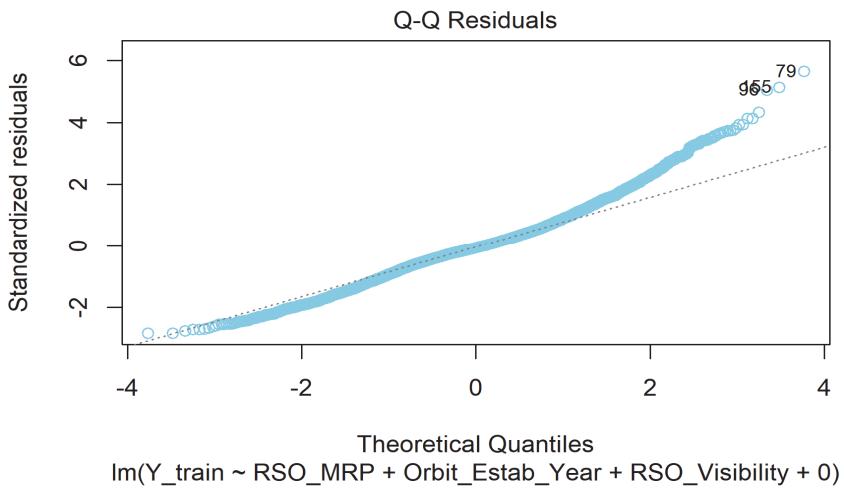
```
plot(model2, col = "skyblue")
```

The first residual plot (**Figure 5-3**) looks like a funnel shape. This shape indicates **heteroskedasticity**. The presence of non-constant variance in the error terms results in heteroskedasticity. We can clearly see that the variance of error terms(residuals) is not constant. Non-constant variance arises in presence of outliers or extreme leverage values. These values get too much weight, thereby disproportionately influencing the model's performance. When this phenomenon occurs, the confidence interval for out of sample prediction tends to be unrealistically wide or narrow. Here, we begin to see that the fitted model is not very good..



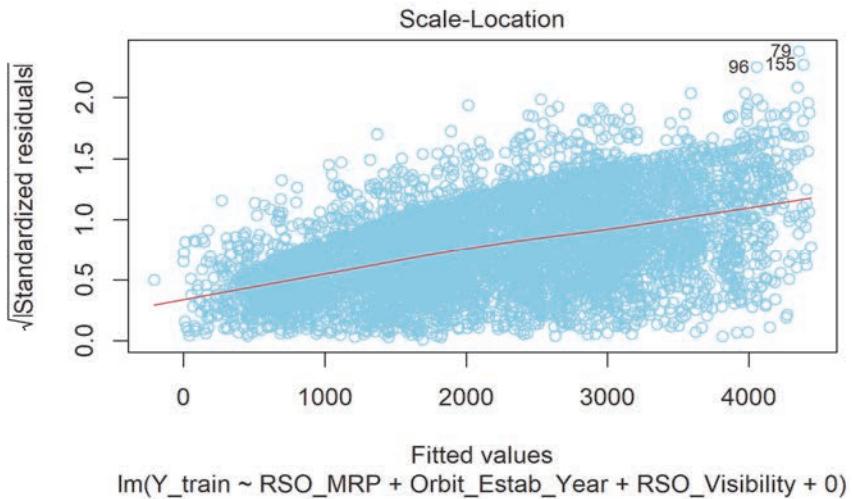
**Figure 5-3.** Model 2 residuals vs fitted values plot with increasing error variance

Next, we examine the Q-Q plot as another graphical diagnostic for normality. **Figure 5-4** shows the plot and we can see that the data begins to curve upward from the diagonal starting at about (0,0). If the model were a good fit, the data would be more linear.

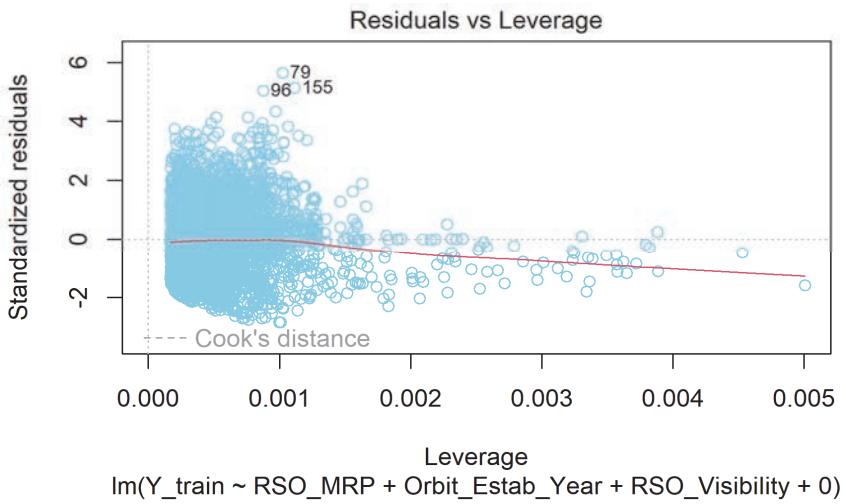


**Figure 5-4.** Normality plot for Model 2

**Figure 5-5** shows the scale-location plot with standardized residuals and does not provide any additional value to those shown in the first two plots.



*Figure 5-5. Scale-Location plot with standardized residuals*



*Figure 5-6. Plots of the standardized residuals vs model leverage.*

The fourth plot (**Figure 5-6**) shows *Cook's Distance*. Cook's Distance is an estimate of the influence of a data point. It considers both the leverage and residual of each observation. Cook's Distance is a summary of how much a regression model changes when we remove the  $i^{\text{th}}$  observation.

Collectively, these diagnostic plots indicate signs of non-linearity in the data which model has not captured. We also see three outliers in each plot: data points, 79, 96, and 155. Typically, we would remove these

points from the dataset, but given the amount of data we have, they do not have a significant impact on the model's fit.

## 5.10 Model 3: Mixed Feature Types

### 5.10.1 Import the Data

```
train <-  
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\Survive.csv")  
summary(train)
```

RSO_ID	RSO_Weight	RSO_Density
Length:8523	Min. : 4.5550	Length:8523
Class :character	1st Qu.: 8.7737	Class :character
Mode :character	Median :12.6000	Mode :character
	Mean :12.8576	
	3rd Qu.:16.8500	
	Max. :21.3500	
	NA's :1463	
RSO_Visibility	RSO_Name	RSO_MRP
Min. :0.000000	Length:8523	Min. : 31.290
1st Qu.:0.026989	Class :character	1st Qu.: 94.145
Median :0.053931	Mode :character	Median :143.547
Mean :0.066132		Mean :140.877
3rd Qu.:0.094585		3rd Qu.:186.621
Max. :0.328391		Max. :283.029
Orbit_ID	Orbit_Estab_Year	Orbit_Height
Length:8523	Min. :1985.0	Length:8523
Class :character	1st Qu.:1987.0	Class :character
Mode :character	Median :1999.0	Mode :character
	Mean :1998.4	
	3rd Qu.:2004.0	
	Max. :2009.0	
Stealth_Type	RSO_Type	
Length:8523	Length:8523	
Class :character	Class :character	
Mode :character	Mode :character	
Survivability		
Min. : 33.29		
1st Qu.: 848.89		
Median : 1805.65		
Mean : 2186.92		
3rd Qu.: 3102.63		

```
Max. :13086.96
```

## 5.10.2 Select Features

For Model 3, we select most of the features, categorical and numeric, for the dataset  $X$ .

```
X <- train[c(2,3,4,5,6,8,9,10,11)]  
names(X)
```

```
[1] "RSO_Weight"      "RSO_Density"      "RSO_Visibility"  
"RSO_Name"  
[5] "RSO_MRP"        "Orbit_Estab_Year" "Orbit_Height"  
"Stealth_Type"  
[9] "RSO_Type"
```

## 5.10.3 Fill Missing Feature Values

Next, we fill missing feature values using means for numeric fields and the lowest categories (most common) for categorical features.

```
X$RSO_Weight[is.na(X$RSO_Weight)] <- mean(X$RSO_Weight,  
na.rm = TRUE)  
X$RSO_Density[is.na(X$RSO_Density)] <- "Low"  
X$RSO_Visibility[X$RSO_Visibility == 0] <-  
  mean(X$RSO_Visibility)  
X$RSO_MRP[is.na(X$RSO_MRP)] <- mean(X$RSO_MRP,  
  na.rm = TRUE)  
X$Orbit_Estab_Year=2013 - X$Orbit_Estab_Year  
X$Orbit_Height[is.na(X$Orbit_Height)] <- "LEO"  
X$Stealth_Type[is.na(X$Stealth_Type)] <- "Stealth_1"  
X$RSO_Type[is.na(X$RSO_Type)] <- "RSO_Type1"
```

## 5.10.4 Create Response Set

We also create the response set  $Y$  (survivability) and fill missing values with the mean value.

```
Y <- train[c(12)]  
Y$Survivability[is.na(Y$Survivability)] <-  
  mean(Y$Survivability, na.rm = TRUE)  
names((Y))
```

```
[1] "Survivability"
```

## 5.10.5 Train-Test Split

Next, we split  $X$  and  $Y$  into training and cross-validation sets.

```
set.seed(567)
part <- sample(2, nrow(X), replace = TRUE, prob = c(0.7,
0.3))
X_train <- X[part == 1,]
X_cv <- X[part == 2,]

Y_train <- Y[part == 1,]
Y_cv <- Y[part == 2,]
```

## 5.10.6 Model 3 Construction

For [Model 3](#), we add the categorical values, and the model we build has mixed feature types.

```
model3 <- lm(Y_train ~ RSO_MRP + Orbit_Height +
RSO_Visibility + Orbit_Estab_Year + Stealth_Type + 0,
               data = X_train )
summary(model3)
```

Call:

```
lm(formula = Y_train ~ RSO_MRP + Orbit_Height + RSO_Visibility +
    Orbit_Estab_Year + Stealth_Type + 0, data = X_train)
```

Residuals:

Min	1Q	Median	3Q	Max
-4994.7	-634.8	-48.0	661.4	7791.6

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
RSO_MRP	11.1465	0.2571	43.347	< 2e-16 ***
Orbit_Height	-1754.1906	100.1200	-17.521	< 2e-16 ***
Orbit_HeightGEO	-1726.9934	127.0431	-13.594	< 2e-16 ***
Orbit_HeightLEO	-1495.5574	96.4788	-15.501	< 2e-16 ***
Orbit_HeightMEO	-1331.1593	104.8953	-12.690	< 2e-16 ***
RSO_Visibility	-1838.5004	297.7648	-6.174	7.08e-10 ***
Orbit_Estab_Year	24.2469	2.3962	10.119	< 2e-16 ***
Stealth_TypeStealth_1	1770.9649	71.2298	24.863	< 2e-16 ***
Stealth_TypeStealth_2	3475.6101	90.0222	38.608	< 2e-16 ***
Stealth_TypeStealth_3	4962.8902	110.6127	44.867	< 2e-16 ***
Stealth_TypeStealth_4	2588.0252	152.9347	16.922	< 2e-16 ***
---				
Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'
	0.1 ' '	1		

```
Residual standard error: 1090 on 5945 degrees of freedom  
Multiple R-squared:  0.8448, Adjusted R-squared:  0.8445  
F-statistic: 2942 on 11 and 5945 DF, p-value: < 2.2e-16
```

### 5.10.7 Extract Model 3 Equation

Next, we use the R function `extract_eq` from the `equatiomatic` package, to extract the `Model 3` equation. The `equatiomatic` package provides the function for expressing the regression model in its mathematical format.

```
equatiomatic::extract_eq(model3)
```

$$Y_{train} = \beta_1(\text{RSO\_MRP}) + \beta_2(\text{Orbit\_Height}) + \beta_3(\text{Orbit\_Height}_{\text{GEO}}) + \\ \beta_4(\text{Orbit\_Height}_{\text{LEO}}) + \beta_5(\text{Orbit\_Height}_{\text{MEO}}) + \\ \beta_6(\text{RSO\_Visibility}) + \beta_7(\text{Orbit\_Estab\_Year}) + \\ \beta_8(\text{Stealth\_Type}_{\text{Stealth\_1}}) + \beta_9(\text{Stealth\_Type}_{\text{Stealth\_2}}) + \\ \beta_{10}(\text{Stealth\_Type}_{\text{Stealth\_3}}) + \beta_{11}(\text{Stealth\_Type}_{\text{Stealth\_4}}) + \epsilon$$

**Figure 5-7** provides a picture of `Model 2`'s coefficients.

### 5.10.8 Use Model 3 to Predict

Now, we use the model and the cross-validation set to make predictions.

```
predict_3 <- predict(model3, X_cv)
```

### 5.10.9 Model 3 Metrics

Next, we calculate the `Model 3` performance metrics, `RMSE` and `R-squared`.

```
RMSE <- rmse(Y_cv, predict_3)  
R.sq <- rsq(model3)  
print(paste("R-square =", R.sq))
```

```
[1] "R-square = 0.84479059867493"
```

```
print(paste("RMSE =", RMSE))
```

```
[1] "RMSE = 1108.07626043642"
```

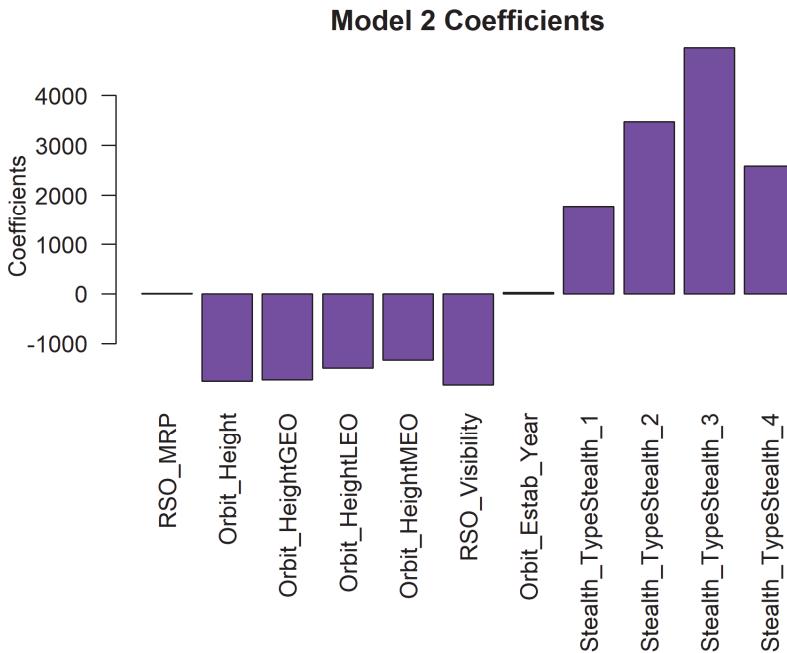
### 5.10.10 Plot Model 3 Coefficients

Finally, we plot the `Model 3` coefficient values using a `barplot`.

```

par(mar=c(11,4,2,1))
barplot(model3$coefficients, main = "Model 2
Coefficients",
        ylab = "Coefficients", las = 2, col = 'blueviolet')

```



*Figure 5-7. Bar plot of the model's coefficient values.*

### 5.10.11 Model 3 Residual Analysis

Recall that residuals are differences between the one-step-predicted output from the model and the measured output from the validation data set. Thus, residuals represent the portion of the validation data not explained by the model.

### 5.10.12 Model 3 Residual Plots

Residual analysis plots show different information depending on whether you use time-domain or frequency-domain input-output validation data. For linear models, you can estimate a model using time-domain data, and then validate the model using frequency domain data.

### 5.10.13 Displaying the Confidence Interval

The confidence interval corresponds to the range of residual values with a specific probability of being statistically insignificant for the system.

```
plot(model3, col = "green2")
```

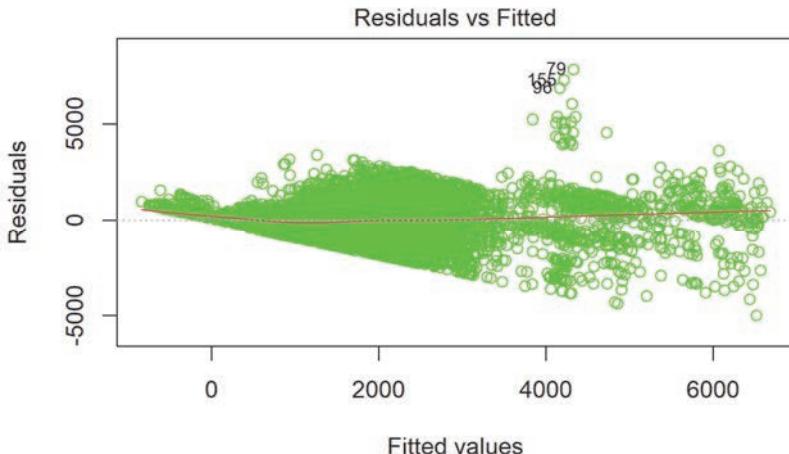


Figure 5-8. Plot of the residuals vs fitted values, with outliers at 79, 96, 155.

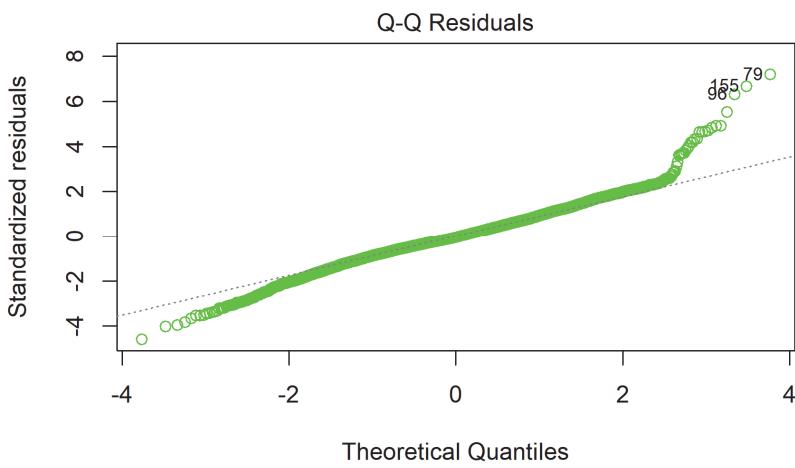
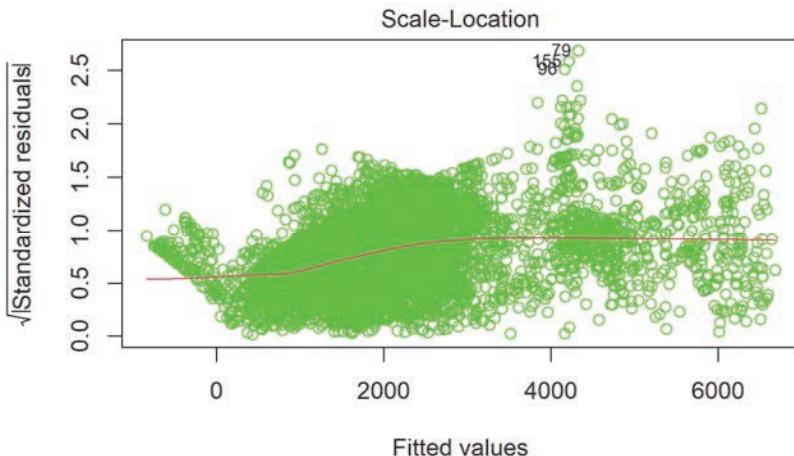
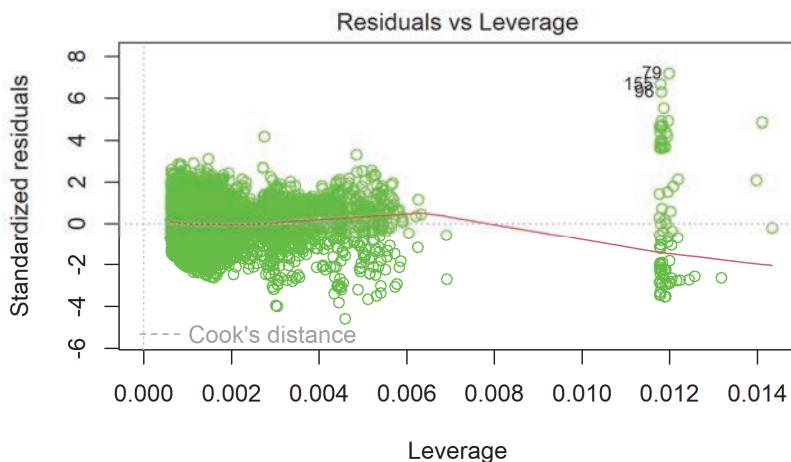


Figure 5-9. Normality plot showing a lack of fit in the upper quantiles, with outliers at 79, 96, 155.



**Figure 5-10.** Scale-location chart with standardized residuals, with outliers at 79, 96, 155.



**Figure 5-11.** Plot of standardized residuals vs model leverage, with outliers at 69, 295, 2242.

The first residual plot (**Figure 5-8**) looks like a funnel shape. This shape indicates heteroskedasticity. The presence of non-constant variance in the error terms results in heteroskedasticity. We can clearly see that the variance of error terms (residuals) is not constant. Non-constant variance arises in presence of outliers or extreme leverage values. These values get too much weight, thereby disproportionately influencing the model's performance. When this phenomenon occurs, the confidence

interval for out of sample prediction tends to be unrealistically wide or narrow.

We can easily check this by looking at residual vs fitted values plot. If heteroskedasticity exists, the plot would exhibit a funnel shape pattern as shown above. This indicates the model did not capture signs of non-linearity in the data.

Also in the first plot, we see three outliers: data points, 79, 96, and 155. Typically, we would remove these points from the dataset, but given the amount of data we have, they do not have a significant impact on the model's fit. The second plot (**Figure 5-9**) is the Normal Q-Q plot. The concern here is curvature of the data. If the model were a good fit, the data would be more linear.

The third plot (**Figure 5-10**) shows the standardized residuals and does not provide any additional value to those shown in the first plot.

The fourth plot (**Figure 5-11**) shows Cook's Distance. Cook's Distance is an estimate of the influence of a data point. It considers both the leverage and residual of each observation. Cook's Distance is a summary of how much a regression model changes when the  $i^{th}$  observation is removed.

#### 5.10.14 How to Interpret a Residual Plot

When the non-linearity does not exist, we follow this process for interpreting residuals:

**Step 1:** Locate the residual = 0 line in the residual plot.

**Step 2:** Look at the points in the plot and answer the following questions:

Are they scattered randomly around the residual = 0 line? Or are they clustered in a curved pattern, such as a U-shaped pattern?

If the points show no pattern, that is, the points are randomly dispersed, we can conclude that a linear model is an appropriate model. In other words, if the residuals are randomly scattered around the residual = 0, it means that a linear model approximates the data points well without favoring certain inputs.

If the points show a curved pattern, such as a U-shaped pattern, we can conclude that a linear model is not appropriate and that a non-linear model might fit better.

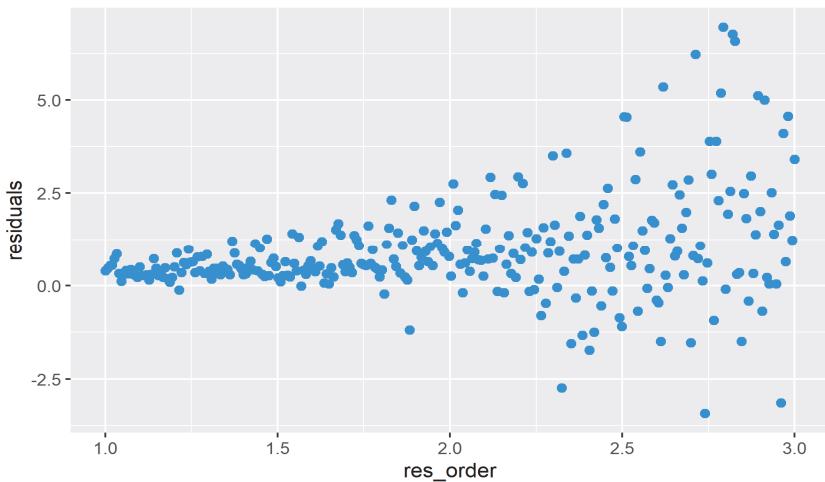
### 5.10.15 Error variance Analysis

Residuals plot that has an increasing trend suggests that the error variance increases with the independent variable (see **Figure 5-12**); while a distribution that reveals a decreasing trend indicates that the error variance decreases with the independent variable. Neither of these distributions are constant variance patterns. The foregoing is also true when we have drifting in the error variance, as demonstrated in Figure 2. Therefore, they indicate that the assumption of constant variance is not likely to be true and the regression is not a good one. On the other hand, a horizontal-band pattern suggests that the variance of the residuals is constant.

### 5.10.16 Increasing Error Variance

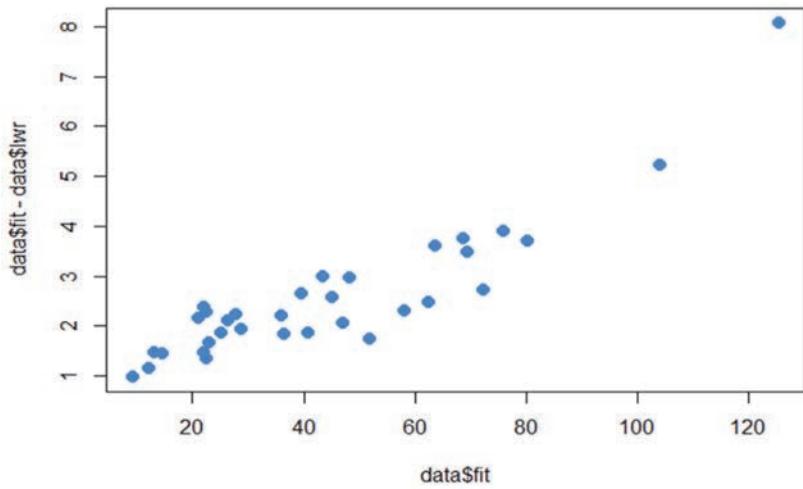
The generate the **Figure 5-12**, we simulated some basic data that would be demonstrate an increasing error variance.

```
set.seed(77)
n <- 300
res_order <- seq(1, 3, length.out = n)
g <- sample(c("m", "f"), size = n, replace = TRUE)
msd <- ifelse(g=="m", 2*2.5, 2) * exp(1.5*res_order)/10
residuals <- (1.2 + 2.1*res_order - 1.5*(g == "m") +
2.8*res_order*(g == "m") + rnorm(n, sd = msd))/10
d <- data.frame(residuals, res_order, g)
ggplot(d, aes(res_order, residuals)) +
geom_point(size = 2, col = 4) +
ggttitle("Residual that show Increasing Variance")
```



*Figure 5-12. Residuals with increasing error variance*

To generate **Figure 5-13**, we simulated some basic data that would demonstrate drifting error variance.



*Figure 5-13. Residuals with drifting error variance.*

### 5.10.17 Conclusions from Residual Analysis

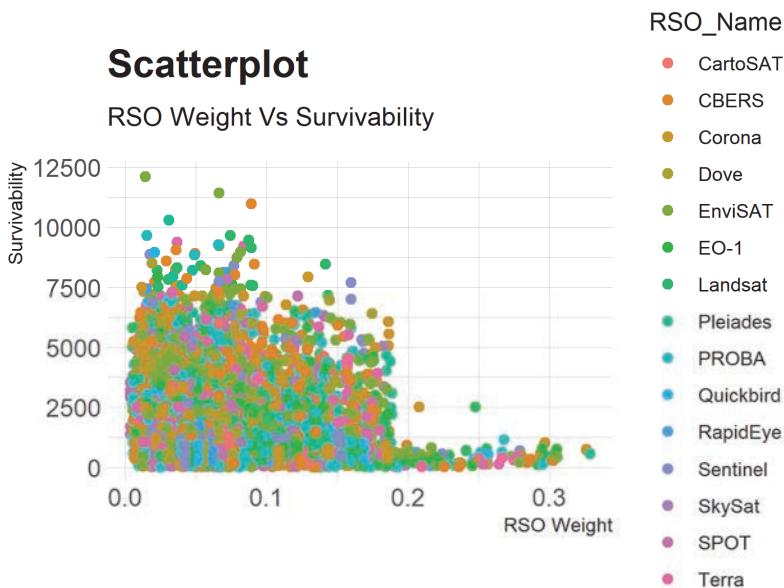
The plots reveal issues. One concern is introduction of outliers at data points 79, 96, and 155, which has the effect of squeezing the data toward the horizontal axis. This makes it difficult to analyze the residuals, precisely. To correct this, we would normally remove the data outlier and repeat the modeling and analysis. However, in the next section, we

are going to try a different modeling type. Another concern is the apparent, increasing error variance. The fourth plots show *Cook's Distance*. Cook's Distance is an estimate of the influence of a data point. It considers both the leverage and residual of each observation. Cook's Distance is a summary of how much a regression model changes when the  $i^{th}$  observation is removed.

### 5.10.18 Scatterplots

Before we delve into another model, let us look at some more plots that may be useful for our analysis. The first plot (**Figure 5-14**) is for *RSO Weight vs. Survivability*, which we categorized by *RSO Name*.

```
ggplot(X_train, aes(x = RSO_Visibility, y = Y_train,
                     color = RSO_Name)) +
  geom_point(size = 2) +
  theme_ipsum() +
  labs(subtitle = "RSO Weight Vs Survivability",
       y = "Survivability", x = "RSO Weight",
       title = "Scatterplot")
```



**Figure 5-14.** Resident space object weight vs survivability.

```
library(ggplot2)
gg <- ggplot(X_train, aes(x = RSO_Weight, y = Y_train,
```

```

        col = Stealth_Type)) +
geom_point(size = 2) +
theme_ipsum() +
labs(subtitle = "RSO Weight Vs Survivability",
     y = "Survivability",
     x = "RSO Weight",
     title = "Scatterplot")
gg

```



*Figure 5-15. Resident space object stealth type vs survivability.*

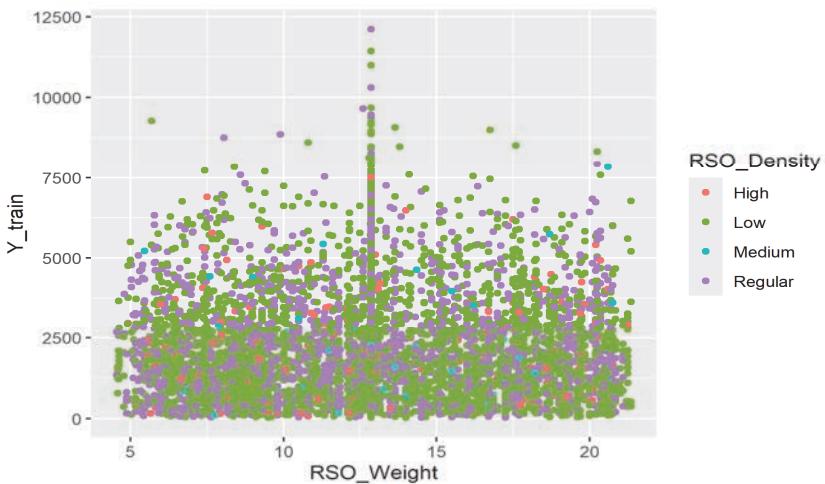
We could expect, it appears the more advanced the stealth technology is, the more survivable is the RSO. However, note that are a few exceptions. Now, we generate a scatterplot (**Figure 5-16**) using *RSO Weight* vs. *Survivability*, but this time we categorize it by *RSO Density*.

```

gg <- ggplot(X_train, aes(x = RSO_Weight, y = Y_train))
+ geom_point(aes(col = RSO_Density))
gg

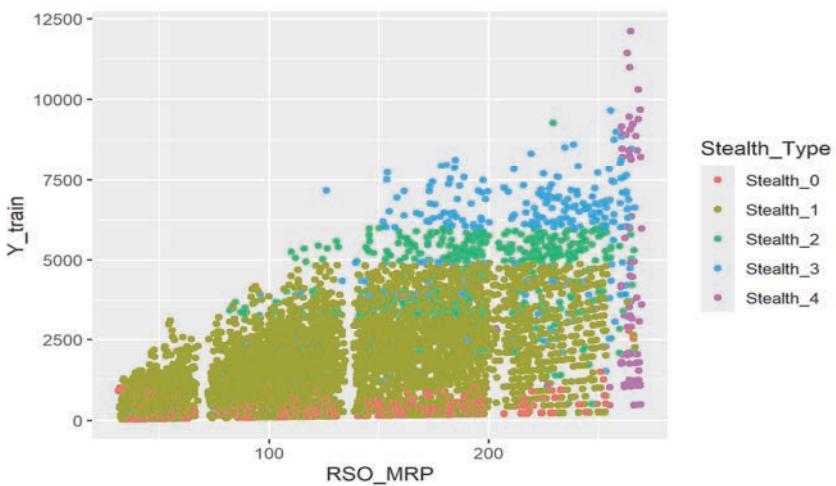
```

The plot adds validity to the model outcomes, since *RSO Density* was not a significant feature in the model. **Figure 5-17** shows *RSO MRP* vs. *Survivability*. Here, we categorize the data by *Stealth Type*.



*Figure 5-16. Resident space object weight vs survivability by density.*

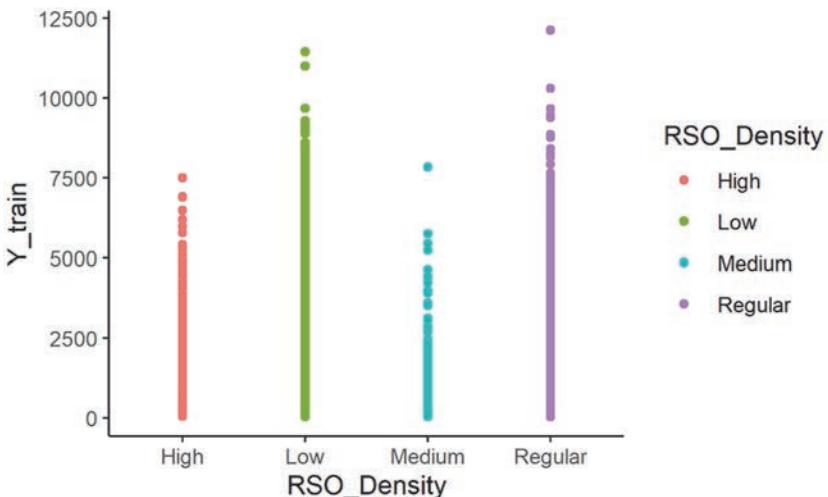
```
gg <- ggplot(X_train, aes(x = RSO_MRP, y = Y_train)) +
  geom_point(aes(col = Stealth_Type))
gg
```



*Figure 5-17. Resident space object MRO vs survivability by stealth type.*

Although it is difficult to see all the data points for each *Stealth Type*, there does seem to be a pattern of increasing survivability with increasing suitability. Now, let us do the same analysis but categorized as *RSO Density* (recall that it is not significant in the model).

```
gg <- ggplot(X_train, aes(x = RSO_MRP, y = Y_train)) +
  geom_point(aes(col = RSO_Density))
gg
```

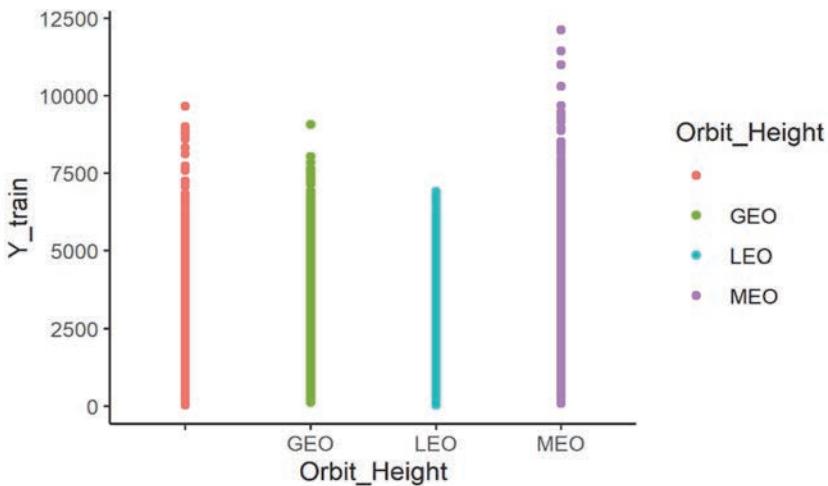


**Figure 5-18. Resident space object MRP vs survivability by density.**

Again, there does seem to be a pattern of increasing survivability with increasing suitability, even though *RSO Density* was not a significant feature. Now, let's look at *Establishment Year* vs. *Survivability*, categorized by *Orbit Height*.

```
gg <- ggplot(X_train, aes(x = Orbit_Estab_Year,
  y = Y_train)) +
  geom_point(aes(col = Orbit_Height))
gg
```

In **Figure 5-19**, the salmon-colored data points are missing values, so one thing we learned is that we should impute the missing values or remove them from the data. The plot seems to demonstrate that the established duration factor increases, the RSO is less survivable. Note that the father to the left an RSO is in the plot, it has been in orbit for less time than those on the right. Also, note that the data is skewed, i.e., not symmetric. Recall that skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive, zero, negative, or undefined.

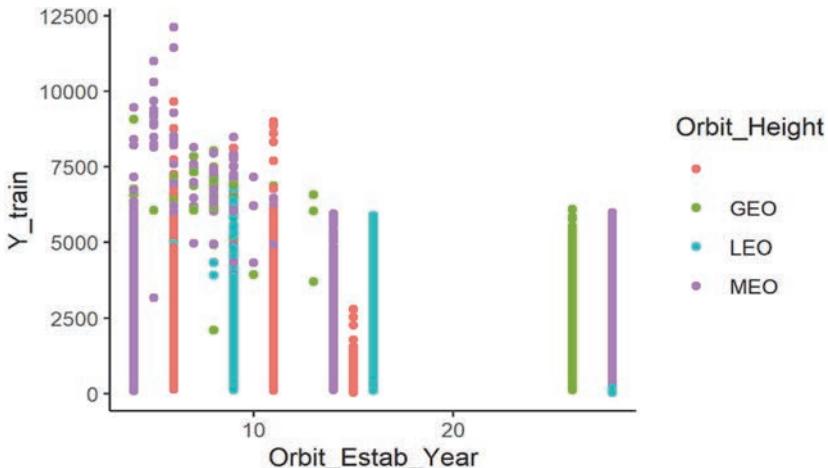


**Figure 5-19.** Resident space object orbit establishment year vs survivability by orbit height.

Let us do the same analysis but with *Stealth Type* as the category.

```
gg <- ggplot(X_train, aes(x = Orbit_Estab_Year,
y = Y_train)) + geom_point(aes(col = Stealth_Type))
gg
```

The plot in **Figure 5-20** tells the same story for *Orbit Establishment* vs. *Survivability*. Note that *Stealth Type 4* is a younger technology.



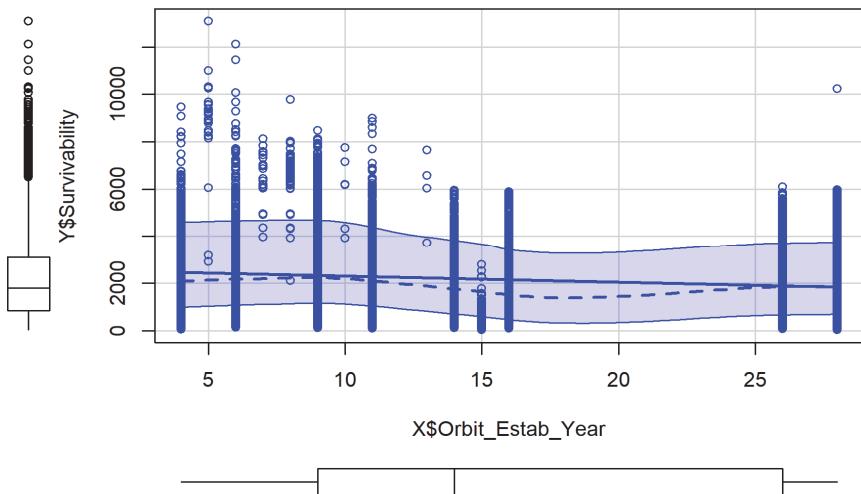
**Figure 5-20.** Resident space object orbit establishment year vs survivability by stealth type

Now let us switch gears a bit and look at a different type of scatterplot, which is available in the car-package. This kind of scatterplot shows the information provided in previous plots but adds more diagnostics.

First, we see a dashed line in **Figure 5-21** that represents the trend of the data and around it, in the shaded light blue areas, it adds a confidence interval. Second, there is a solid blue liner fit line slightly above the trend curve.

Finally, there are two box plots, one representing the survivability data and another representing the *Established Year* data. The bow plots clearly support the observation of skewness we made previously. We also see that a linear fit might not be the best one, although our observation is not conclusive.

```
library(car)
scatterplot(X$Orbit_Estab_Year, Y$Survivability)
```



**Figure 5-21.** Resident space object orbit establishment year vs survivability.

**Figure 5-22** shows a scatterplot for RSO MRP vs. Survivability. The plot supplies the same information as we saw previously but adds more. In this instance we can see that underlying trend is linear, but increasing, as well as the confidence interval. The boxplots show skewness in the data and point out the central tendency. This plot also implies that the linear regression model may not be the best modeling option.

```
scatterplot(X$RSO_MRP, Y$Survivability)
```

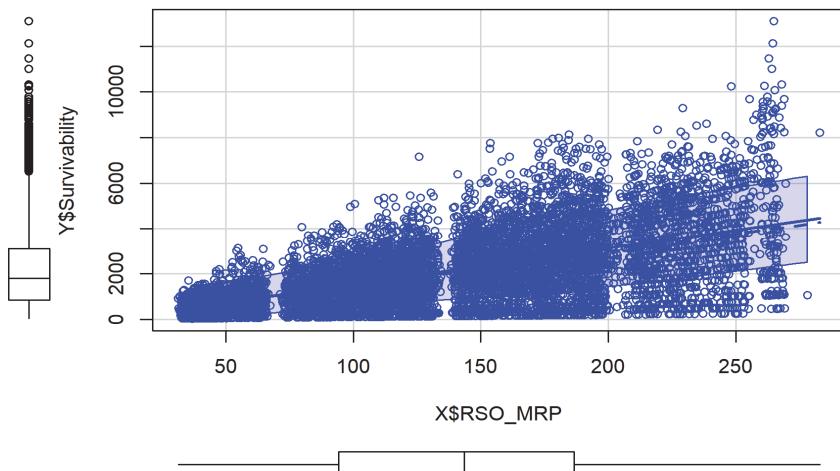


Figure 5-22. Resident space object MRP vs survivability

Finally, **Figure 5-23** shows RSO Weight vs. Survivability again. The plot does not add information, except that the fit may be curvilinear, implying again that linear regression may not lead to the best fit.

```
scatterplot(X$RSO_Weight, Y$Survivability)
```

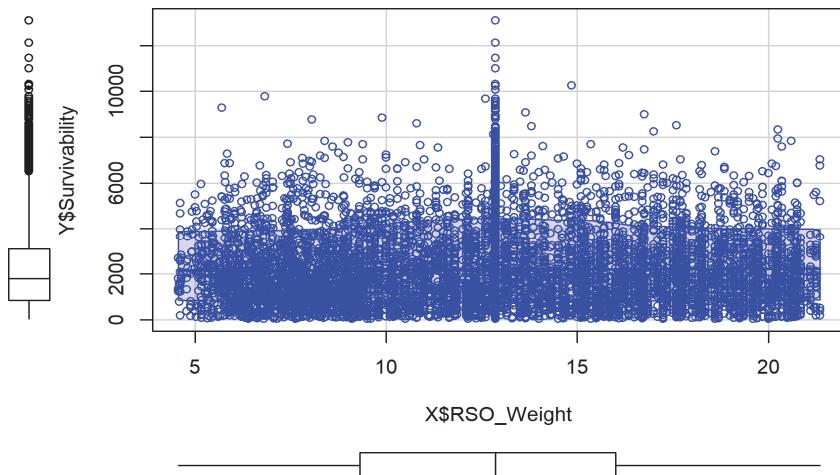
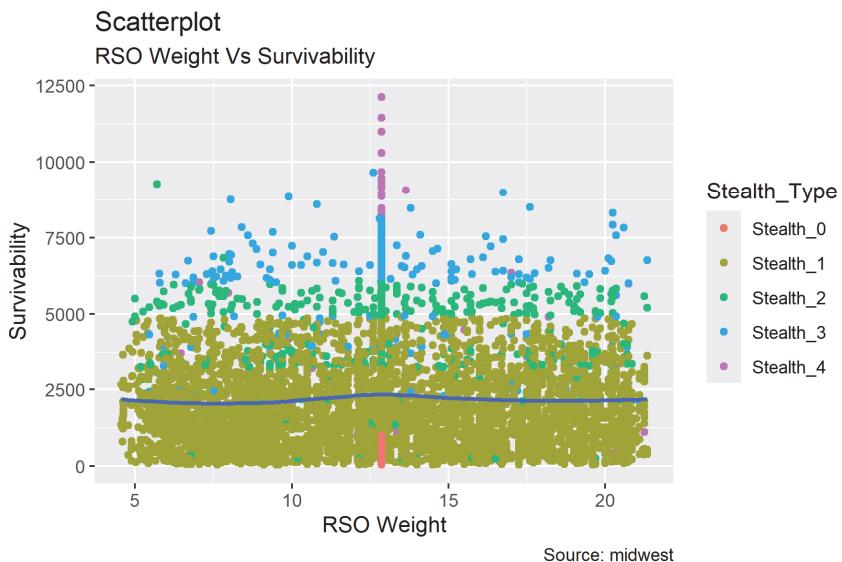


Figure 5-23. Resident space object weight vs survivability.

## 5.10.19 Holistic Scatterplot

This final scatterplot (**Figure 5-24**) is different from previous ones, in that we can see the data categorized by *Stealth Type* and observe the darker color blue trend line is curvilinear.

```
gg <- ggplot(X_train, aes(x = RSO_Weight,
  y = Y_train)) +
  geom_point(aes(col = Stealth_Type)) +
  geom_smooth(method = "loess", se = F) +
  labs(subtitle = "RSO Weight Vs Survivability",
       y = "Survivability",
       x = "RSO Weight",
       title = "Scatterplot",
       caption = "Source: midwest")  
gg
```



**Figure 5-24.** Resident space object weight vs survivability by stealth type.

## 5.10.20 Model 3 Evaluation

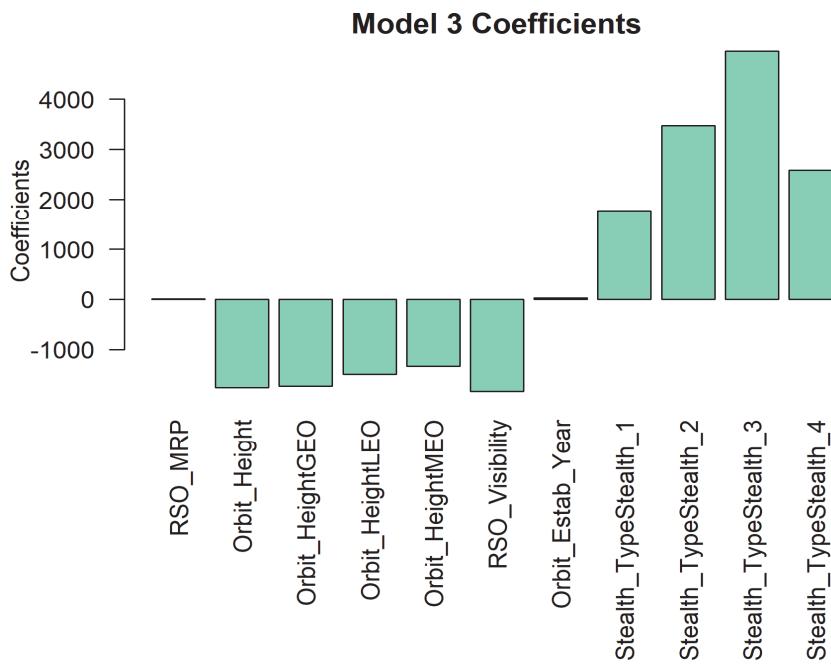
Aside from our earlier performance metrics and the diagnostics we performed with our plots, we will now look at some numerical metrics. Recall the MSE from **Model 2** is **0.8448** and its  $R^2$  value is **0.8445**. **Model 3**'s MSE is close to zero, which is a drastic improvement. Also, the  $R^2$  of 0.8445 shows that the model is a good fit. But wait! Didn't the plots

reveal some potential issues? They did, and this is probably a model that over-fits the data!

```
par(mar=c(11,4,2,1))
barplot(model3$coefficients, main = "Model 3
Coefficients",
ylab = "Coefficients", las = 2, col = "aquamarine2")
```

## 5.10.21 Bar Plot Observations

Referring to **Figure 5-25**, we can see that magnitude of the coefficients for `Orbit_Identifier_OUT013`, `Orbit_IdentifierOUT018`, `Orbit_IdentifierOUT027`, `1Orbit_Identifier_OUT0451`, `10orbit_IdentifierOUT0491`, `Orbit_HeightGEO`, `Orbit_HeightMEO`, `Stealth_Type2`, `Stealth_Type3`, and `RSO_Type3`, are much higher as compared to rest of the coefficients. Therefore, the survivability of an RSO would be more driven by these features or levels of features. This might indicate an over-fitted model and validates our prior observation.



*Figure 5-25. Bar plot of the model's coefficient values.*

How can we reduce the magnitude of coefficients in our model to correct for over-fitting? For this purpose, we have different types of regression

techniques, which use regularization to overcome this problem. So let us discuss them in the next chapter.

## 5.11 Chapter Summary

In this chapter, we discussed estimating and interpreting parameters of a multiple regression model. We saw how to produce predicted values and perform residual analysis. We examined the coefficient of determination,  $R^2$ , and adjusted  $R^2$  as model metrics. We also explored significance testing of model features. Next, we looked at examples of linear models with numeric features and linear models with mixed feature types.



## 6 Regularization and Regression

In this chapter, we will discuss regularization, which will be one of the mechanisms for ensuring that we choose a model that will give us the balance between bias and variance.

### 6.1 MODEL 1 – Survivability Modeling

Aside from potential nation-state threats, what factors might affect the survivability of resident space objects (i.e., satellites)? We constructed a database comprised of ten factors and a response variable using simulation. The data is purely hypothetical, and the satellites used may not represent reality, although the names are real. The dataset is named “Survive” for short, and is comma delimited (CSV).

#### 6.1.1 Load the Dataset

First, we will load the comma delimited data using the `read.csv` function and print the first six rows, including the headings.

```
#install.packages("rsq")
library(rsq)
library(ggplot2)
library(hrbrthemes)
#hrbrthemes::import_roboto_condensed()

train <-
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\Survive.csv"
) #Import the train set
head(train,6)
```

	RSO_ID	RSO_Weight	RSO_Density	RSO_Visibility	RSO_Name
1	FDR37	NA	Regular	0.065928735	Worldview
2	FDX09	9.00	Low	0.065515067	CBERS
3	FDF04	17.50	High	0.013692598	PROBA
4	FDA15	9.30	Low	0.016087659	Landsat
5	FDA38	5.44	Low	0.025583714	Landsat
6	FDX13	NA	Low	0.047551568	Pleiades
	RSO_MRP	Orbit_ID	Orbit_Estab_Year	Orbit_Height	
1	283.0292	OUT027	2009	MEO	
2	278.1370	OUT018	2009	MEO	
3	269.7304	OUT018	2009	MEO	
4	269.6092	OUT018	2009	MEO	
5	269.1538	OUT018	2009	MEO	

```

6 269.1092    OUT027          2007      MEO
  Stealth_Type  RSO_Type Survivability
1   Stealth_4 RSO_Type3     8209.3140
2   Stealth_4 RSO_Type2     1058.6220
3   Stealth_4 RSO_Type2     3616.6256
4   Stealth_4 RSO_Type2     5976.2208
5   Stealth_4 RSO_Type2     480.7076
6   Stealth_4 RSO_Type3     8217.3036

```

In the dataset, we see characteristics of the RSOs (Weight, Visibility, etc.), characteristics of their orbits (Height, Type, etc.), and a survivability score. We want to predict survivability based on these “features.”

### 6.1.2 Data Preprocessing

First, we need to perform a few preprocessing tasks on the data to prepare it for modeling.

### 6.1.3 Fill Empty Spaces

First, we impute missing values (i.e., fill missing spaces) with “NA”

```

train[train==""] <- NA #Filling blank values with NA
names(train)

```

```

[1] "RSO_ID"    "RSO_Weight"  "RSO_Density"  "RSO_Visibility"
[5] "RSO_Name"   "RSO_MRP"     "Orbit_ID"     "Orbit_Estab_Year"
[9] "Orbit_Height" "Stealth_Type" "RSO_Type"    "Survivability"

```

### 6.1.4 Create sets X and Y

Next, we create two datasets, one with two independent variables,  $X$ , and one with the dependent variable or response,  $Y$ .  $X$  will be comprised of columns 6 and 8, or “`RSO_MRP`” and “`Orbit_Estab_Year`”, respectively.  $Y$  is the response, “`Survivability`.”

```

X <- train[c(6,8)] #Creating new data with two variables
names((X))

```

```
[1] "RSO_MRP"           "Orbit_Estab_Year"
```

```

Y <- train[c(12)] #Storing the dependent variable
names((Y))

```

```
[1] "Survivability"
```

### 6.1.5 Split X and Y

Here, we split X and Y into Training (`X_Train` and `Y_Train`) and Cross-Validation (`X_cv` and `Y_cv`) sets, using a 70/30 split.

```
set.seed(567)
part <- sample(2, nrow(X), replace = TRUE, prob = c(0.7,
0.3))
X_train <- X[part == 1,]
X_cv <- X[part == 2,]

Y_train <- Y[part == 1,]
Y_cv <- Y[part == 2,]
```

### 6.1.6 Form on Training Set

Here, we merge `X_train` and `Y_train` into one training set to use in our first regression model.

```
train_2 <- data.frame(Y_train, X_train)
```

### 6.1.7 Model 1 Construction

Model 1 is a linear regression with two features, `RSO_MRP` and `Orbit_Estab_Year`.

```
model1 <- lm(Y_train ~ RSO_MRP + Orbit_Estab_Year,
data = train_2) #Linear model function
summary(model1)
```

Call:

```
lm(formula = Y_train ~ RSO_MRP + Orbit_Estab_Year, data =
train_2)
```

Residuals:

Min	1Q	Median	3Q	Max
-3847.8	-795.4	-83.8	736.5	7884.4

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-2.463e+04	4.384e+03	-5.617	2.03e-08 ***
RSO_MRP	1.572e+01	2.954e-01	53.234	< 2e-16 ***
Orbit_Estab_Year	1.231e+01	2.196e+00	5.604	2.19e-08 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

```
Residual standard error: 1386 on 5953 degrees of freedom  
Multiple R-squared:  0.3323, Adjusted R-squared:  0.332  
F-statistic: 1481 on 2 and 5953 DF,  p-value: < 2.2e-16
```

The equatiomatic package provides the function for expressing the regression model in its mathematical format.

```
library(equatiomatic)  
equatiomatic::extract_eq(model1)
```

$$Y_{\text{train}} = \alpha + \beta_1(\text{RSO\_MRP}) + \beta_2(\text{Orbit\_Estab\_Year}) + \epsilon$$

### 6.1.8 Model 1 Output

Let us take a look at the coefficients of this linear regression model. Since `RSO_MRP` has a high coefficient, having higher RSO MRP scores contributes to higher survivability.

### 6.1.9 Predict Survivability

Now, we use the `predict` function to calculate predictions and assess the model using Root Mean Square Error (RMSE).

```
predict_1 <- predict(model1, X_cv) #Predicting the values
```

### 6.1.10 Calculate MSE and R-Squared

Now, let us calculate MSE, based on the model prediction, and R-squared based on the model. MSE or Mean Square Error,  $R^2$  is a metric that determines how much of the total variation in  $Y$  (dependent variable) can be explained by the variation in  $X$  (independent variable). Mathematically, it can be written as:

$$R^2 = 1 - \frac{\sum(Y_{\text{Actual}} - Y_{\text{Predicted}})^2}{\sum(Y_{\text{Actual}} - Y_{\text{Mean}})^2}$$

The value of  $R^2$  is always between 0 and 1, where 0 means that the model does not explain any variability in the response variable and 1 meaning it explains full variability in the response variable.

### 6.1.11 Root Mean Square Error In R

The root mean square error (RMSE) allows us to measure how far predicted values are from observed values in a regression analysis. In other words, how concentrated the data around the line of best fit.

$$RMSE = \sqrt{\frac{\sum(P_i - O_i)^2}{n}}$$

where:

$\Sigma$  symbol indicates “sum”

$P_i$  is the predicted value for the  $i$ th observation in the dataset

$O_i$  is the observed value for the  $i$ th observation in the dataset

$n$  is the sample size

Now let us calculate these model metrics.

```
library(Metrics)
RMSE <- rmse(model1$fitted.values, predict_1)
R.sq <- rsq(model1)
print(paste("R-square =", R.sq))
```

```
[1] "R-square = 0.332266097561394"
```

```
print(paste("RMSE =", RMSE))
```

```
[1] "RMSE = 1320.48113542391"
```

We will use MSE for Model 1 for comparison with subsequent models. In this case,  $R^2$  is 33%, meaning, the model only explains 33% of variance in survivability by year of establishment and MRP. In other words, if you know the year of establishment and the MRP, you will have 33% information to make an accurate prediction about survivability.

## 6.2 MODEL 2

In the second model, we will introduce an additional feature, `RSO_Weight`. Hence, we will need to redefine the dataset.

```
Train <-
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\Survive.csv")
) #Import the train set
```

```
X<-train[c(2,6,8)]  
names((X))
```

```
[1] "RSO_Weight"      "RSO_MRP"          "Orbit_Estab_Year"  
X$RSO_Weight[is.na(X$RSO_Weight)] <- mean(X$RSO_Weight,  
    na.rm = TRUE)  
Y<-train[c(12)]  
names((Y))
```

```
[1] "Survivability"
```

## 6.2.1 Splitting the Set

Here, we will need to split the new set into train and cross-validation subsets.

```
set.seed(567)  
part <- sample(2, nrow(X), replace = TRUE, prob = c(0.7,  
0.3))  
X_train <- X[part == 1,]  
X_cv <- X[part == 2,]  
  
Y_train <- Y[part == 1,]  
Y_cv <- Y[part == 2,]  
  
train_2 <- data.frame(Y_train, X_train)
```

## 6.2.2 Model 2 Construction

Now, we build a regression model with survivability as the response and three features.

```
model2 <- lm(Y_train ~ RSO_Weight + RSO_MRP + Orbit_Estab_Year,  
    data = train_2 )  
equatiomatic::extract_eq(model2)
```

$$Y_{\text{train}} = \alpha + \beta_1(\text{RSO\_Weight}) + \beta_2(\text{RSO\_MRP}) + \beta_3(\text{Orbit\_Estab\_Year}) + \epsilon$$

```
summary(model1)
```

```
Call:  
lm(formula = Y_train ~ RSO_MRP + Orbit_Estab_Year, data =
```

```

train_2)

Residuals:
    Min      1Q  Median      3Q     Max 
-3847.8  -795.4   -83.8   736.5  7884.4 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.463e+04  4.384e+03  -5.617 2.03e-08 ***
RSO_MRP      1.572e+01  2.954e-01   53.234 < 2e-16 ***
Orbit_Estab_Year 1.231e+01  2.196e+00   5.604 2.19e-08 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1386 on 5953 degrees of freedom
Multiple R-squared:  0.3323, Adjusted R-squared:  0.332 
F-statistic:  1481 on 2 and 5953 DF,  p-value: < 2.2e-16

```

The model summary shows that all three features are significant as predictors, so let us get some predictions and check the MSE.

```
predict_2 <- predict(model2,X_cv)
```

### 6.2.3 Model Performance

Here we calculate  $R^2$  and RMSE to analyze the model's performance. The results show that the model features explain about 33% of the Survivability score with regression model.

```
RMSE <- rmse(model2$fitted.values, predict_2)
R.sq <- rsq(model2)
print(paste("R-square =", R.sq))
```

```
[1] "R-square = 0.332296368274434"
```

```
print(paste("RMSE =", RMSE))
```

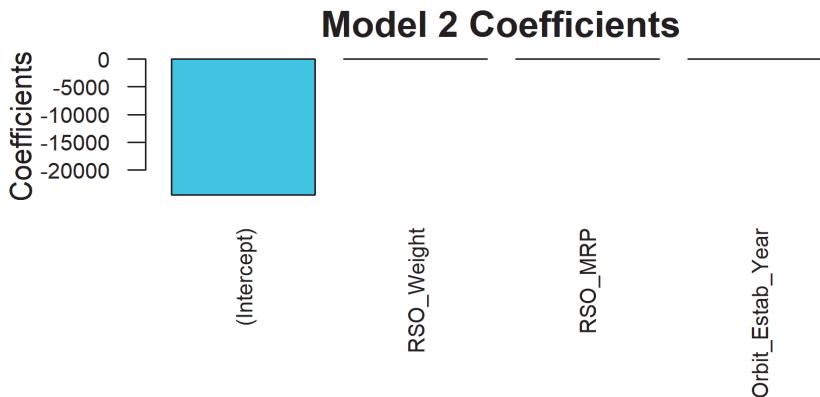
```
[1] "RMSE = 1320.72054958867"
```

### 6.2.4 Model 2 Coefficient Plot

Now use plot the coefficient values using a barplot. The plot in **Figure 6-1** shows that none of the features are significant, but well seek to verify this result with more numerical and residual analysis.

```
par(mar = c(9,4,2,1))
barplot(model2$coefficients, main = "Model 2")
```

```
Coefficients", ylab="Coefficients", cex=.75, cex.lab=1,
cex.main=1.25, cex.axis=.75, las=2, col="#12d1f2")
```



*Figure 6-1. Model 2 coefficient plot with an intercept only significance.*

### 6.2.5 Model 2 Residual Analysis

Recall that residuals are differences between the one-step-predicted output from the model and the measured output from the validation data set. Thus, residuals represent the portion of the validation data not explained by the model.

### 6.2.6 Residual Plots

Residual analysis plots show different information depending on whether you use time-domain or frequency-domain input-output validation data. For frequency-domain validation data, the plot in *Figure 6-2* (top) shows the following two axes:

- Estimated power spectrum of the residuals for each output
- Transfer-function amplitude from the input to the residuals for each input-output pair

For time-domain validation data, the plot in *Figure 6-2* (bottom) shows the following two axes:

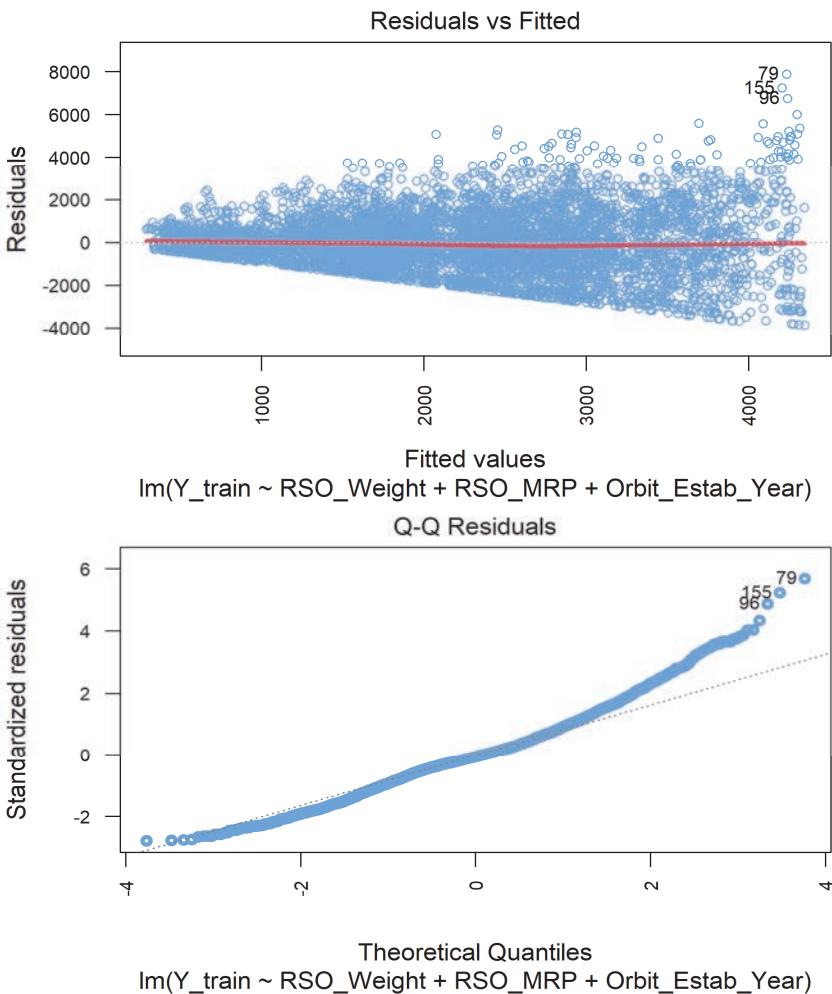
- Autocorrelation function of the residuals for each output
- Cross-correlation between the input and the residuals for each input-output pair

For linear models, you can estimate a model using time-domain data, and then validate the model using frequency domain data.

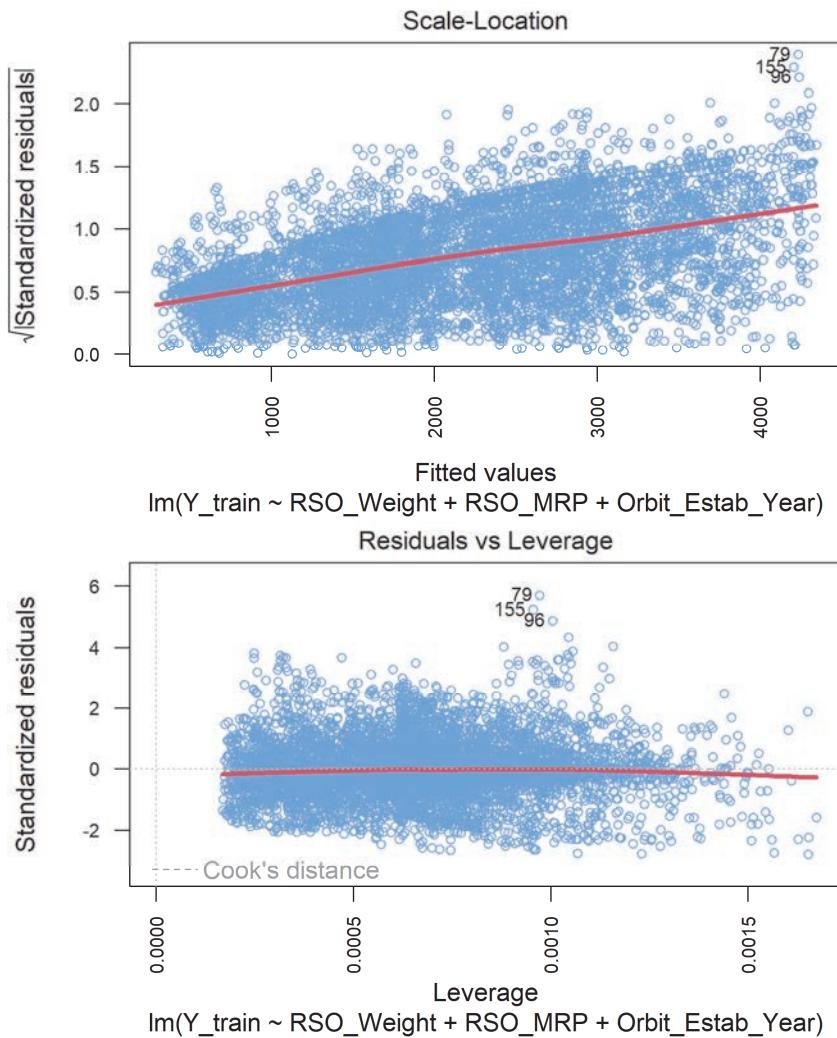
### 6.2.7 Displaying the Confidence Interval

The confidence interval corresponds to the range of residual values with a specific probability of being statistically insignificant for the system.

```
plot(model2, cex = .75, cex.lab = 1, cex.main = 1.25,
cex.axis = .75, las = 2, lwd = 3, col = "#5faaff")
```



*Figure 6-2. Model 2 residual plots: Residuals vs. fitted (top) and Q-Q Plot (bottom)*



**Figure 6-3. Model 2 standardized residual plots: Scale location (top) and Leverage plot (bottom)**

The first residual plot looks like a funnel shape. This shape indicates Heteroskedasticity. The presence of non-constant variance in the error terms results in heteroskedasticity. We can clearly see that the variance of error terms(residuals) is not constant. Non-constant variance arises in presence of outliers or extreme leverage values. These values get too much weight, thereby disproportionately influencing the model's performance. When this phenomenon occurs, the confidence interval for out of sample prediction tends to be unrealistically wide or narrow.

We can easily check this by looking at residual vs fitted values plot. If heteroskedasticity exists, the plot would exhibit a funnel shape pattern as shown above. This indicates signs of non-linearity in the data which the model did not capture. Also in the first plot, we see three outliers: data points, 79, 96, and 155. Typically, we would remove these points from the dataset, but given the amount of data we have, they do not have a significant impact on the model's fit.

The second plot is the Normal Q-Q plot. The concern here is curvature of the data. If the model were a good fit, the data would be more linear.

The third plot shows the standardized residuals and does not provide any additional value to those shown in the first plot.

The fourth plot shows Cook's Distance. Cook's Distance is an estimate of the influence of a data point. It considers both the leverage and residual of each observation. Cook's Distance is a summary of how much a regression model changes when we remove the  $i^{th}$  observation.

### 6.2.8 How to Interpret a Residual Plot

When the non-linearity does not exist, we follow this process for interpreting residuals:

**Step 1:** Locate the residual = 0 line in the residual plot.

**Step 2:** Look at the points in the plot and answer the following questions:

Are they scattered randomly around the residual = 0 line? Or are they clustered in a curved pattern, such as a U-shaped pattern?

If the points show no pattern, that is, the points are randomly dispersed, we can conclude that a linear model is an appropriate model. In other words, if the residuals are randomly scattered around the residual = 0, it means that a linear model approximates the data points well without favoring certain inputs. If the points show a curved pattern, such as a U-shaped pattern, we can conclude that a linear model is not appropriate and that a non-linear model might fit better.

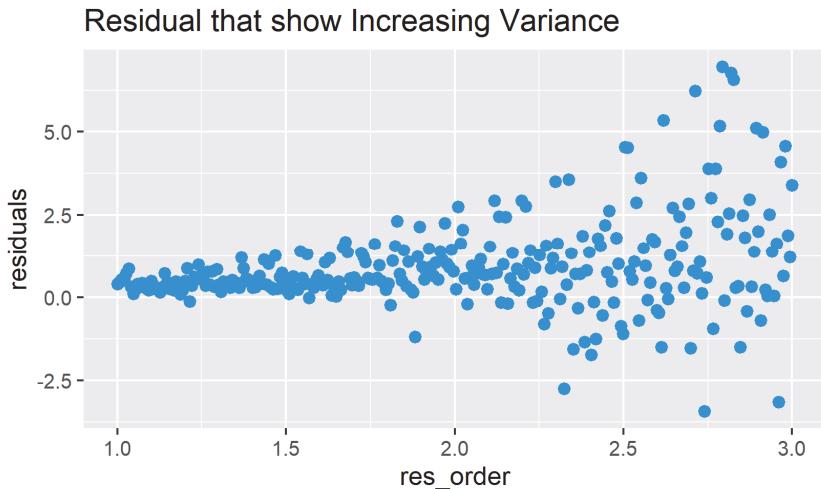
## 6.2.9 Error variance Analysis

Residuals plot that has an increasing trend suggests that the error variance increases with the independent variable (see Figure 1); while a distribution that reveals a decreasing trend indicates that the error variance decreases with the independent variable. Neither of these distributions are constant variance patterns. The foregoing is also true when we have drifting in the error variance, as demonstrated in Figure 2. Therefore, they indicate that the assumption of constant variance is not likely to be true and the regression is not a good one. On the other hand, a horizontal-band pattern suggests that the variance of the residuals is constant.

### 6.2.10 Figure 1: Increasing Error Variance

**Figure 6-4** illustrated increasing error variance that is present in the survivability model .

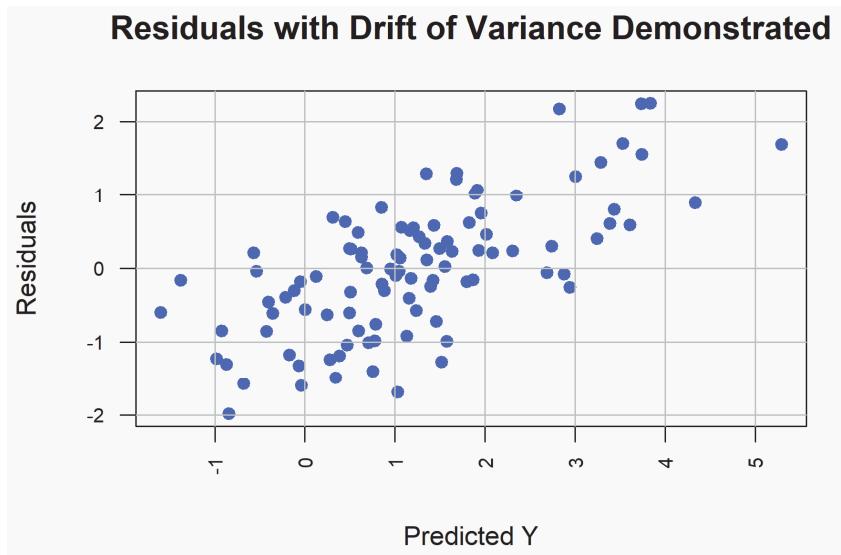
```
set.seed(77)
n <- 300
res_order <- seq(1, 3, length.out = n)
g <- sample(c("m","f"), size = n, replace = TRUE)
msd <- ifelse(g=="m", 2*2.5, 2) * exp(1.5*res_order)/10
residuals <- (1.2 + 2.1*res_order - 1.5*(g == "m") +
2.8*res_order*(g == "m") + rnorm(n, sd = msd))/10
d <- data.frame(residuals, res_order, g)
ggplot(d, aes(res_order, residuals)) +
  geom_point(size = 2, col = 4) + ggtitle("Residual that
show Increasing Variance")
```



*Figure 6-4. Increasing error variance in the survivability model*

### 6.2.11 Figure 2: Drifting Error Variance

To generate the figure below, we simulated some basic data that would demonstrate a drifting error variance.



*Figure 6-5. simulated drifting error variance*

In **Figure 6-6** we simulated some basic data that would demonstrate a constant or non-drifting error variance.

## Residuals with Constant Variance Demonstrated

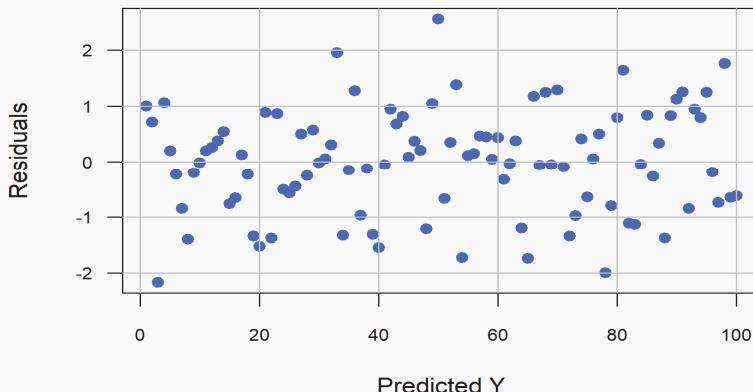


Figure 6-6. Simulated non-drifting (constant) error variance

### 6.2.12 Model 2 Performance

The RMSE of 1320.721 did not change very much but is slightly larger than 1320.481 from Model 1. Model 2 introduces slightly more error.  $R^2$  is 0.3322964 and also about the same as Model 1's. Moreover, we have learned from the residuals vs. fitted value plots that the error variance is increasing implying that the regression model is not a good one. So, back to the drawing board.

## 6.3 MODEL 3

Given the analysis of Model 2, a linear regression model may not be the best way to fit this data. Model 3 is a regression model using all the features to predict survivability. Also, we need to ensure we start with the same data, so we refresh/repeat all the data preparation we did previously.

```
Train <-  
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\Survive.csv"  
) #Import the train set  
X<-train[c(-1,-12)]  
Y<-train[c(12)]  
  
X$RSO_Weight[is.na(X$RSO_Weight)] <- mean(X$RSO_Weight,  
na.rm = TRUE)  
X$RSO_Density[is.na(X$RSO_Density)] <- "Low"  
X$RSO_Visibility[X$RSO_Visibility == 0] <-
```

```

mean(X$RSO_Visibility)
X$RSO_MRP[is.na(X$RSO_MRP)] <- mean(X$RSO_MRP, na.rm = TRUE)
X$Orbit_Establ_Year=2013 - X$Orbit_Estab_Year
X$Orbit_Height[is.na(X$Orbit_Height)] <- "LEO"
X$Stealth_Type[is.na(X$Stealth_Type)] <- "Stealth_1"
X$RSO_Type[is.na(X$RSO_Type)] <- "RSO_Type1"
set.seed(567)
part <- sample(2, nrow(X), replace = TRUE, prob = c(0.7, 0.3))
X_train <- X[part == 1,]
X_cv <- X[part == 2,]
Y_train <- Y[part == 1,]
Y_cv <- Y[part == 2,]
train_3 <- data.frame(Y_train, X_train)

```

### 6.3.1 Model 3 Construction

For this model, we include all of the features in our dataset.

```

model3<-lm(Y_train~.,data = train_3)
equatiomatic::extract_eq(model3)

```

$$\begin{aligned}
Y_{\text{train}} = & \alpha + \beta_1(\text{RSO\_Weight}) + \beta_2(\text{RSO\_Density}_{\text{Low}}) + \\
& \beta_3(\text{RSO\_Density}_{\text{Medium}}) + \beta_4(\text{RSO\_Density}_{\text{Regular}}) + \\
& \beta_5(\text{RSO\_Visibility}) + \beta_6(\text{RSO\_Name}_{\text{CBERS}}) + \beta_7(\text{RSO\_Name}_{\text{Corona}}) + \\
& \beta_8(\text{RSO\_Name}_{\text{Dove}}) + \beta_9(\text{RSO\_Name}_{\text{EnviSAT}}) + \beta_{10}(\text{RSO\_Name}_{\text{EO-1}}) + \\
& \beta_{11}(\text{RSO\_Name}_{\text{Landsat}}) + \beta_{12}(\text{RSO\_Name}_{\text{Pleiades}}) + \\
& \beta_{13}(\text{RSO\_Name}_{\text{PROBA}}) + \beta_{14}(\text{RSO\_Name}_{\text{Quickbird}}) + \\
& \beta_{15}(\text{RSO\_Name}_{\text{RapidEye}}) + \beta_{16}(\text{RSO\_Name}_{\text{Sentinel}}) + \\
& \beta_{17}(\text{RSO\_Name}_{\text{SkySat}}) + \beta_{18}(\text{RSO\_Name}_{\text{SPOT}}) + \beta_{19}(\text{RSO\_Name}_{\text{Terra}}) + \\
& \beta_{20}(\text{RSO\_Name}_{\text{Worldview}}) + \beta_{21}(\text{RSO\_MRP}) + \beta_{22}(\text{Orbit\_ID}_{\text{OUT013}}) + \\
& \beta_{23}(\text{Orbit\_ID}_{\text{OUT017}}) + \beta_{24}(\text{Orbit\_ID}_{\text{OUT018}}) + \beta_{25}(\text{Orbit\_ID}_{\text{OUT019}}) + \\
& \beta_{26}(\text{Orbit\_ID}_{\text{OUT027}}) + \beta_{27}(\text{Orbit\_ID}_{\text{OUT035}}) + \beta_{28}(\text{Orbit\_ID}_{\text{OUT045}}) + \\
& \beta_{29}(\text{Orbit\_ID}_{\text{OUT046}}) + \beta_{30}(\text{Orbit\_ID}_{\text{OUT049}}) + \beta_{31}(\text{Orbit\_Estab\_Year}) + \\
& \beta_{32}(\text{Orbit\_Height}_{\text{GEO}}) + \beta_{33}(\text{Orbit\_Height}_{\text{LEO}}) + \\
& \beta_{34}(\text{Orbit\_Height}_{\text{MEO}}) + \beta_{35}(\text{Stealth\_Type}_{\text{Stealth\_1}}) + \\
& \beta_{36}(\text{Stealth\_Type}_{\text{Stealth\_2}}) + \beta_{37}(\text{Stealth\_Type}_{\text{Stealth\_3}}) + \\
& \beta_{38}(\text{Stealth\_Type}_{\text{Stealth\_4}}) + \beta_{39}(\text{RSO\_Type}_2) + \beta_{40}(\text{RSO\_Type}_3) + \\
& \beta_{41}(\text{RSO\_Type}_4) + \beta_{42}(\text{Orbit\_Establ\_Year}) + \epsilon
\end{aligned}$$

```
summary(model3)
```

Call:

```
lm(formula = Y_train ~ ., data = train_3)
```

Residuals:					
	Min	1Q	Median	3Q	Max
	-3740.8	-585.7	-50.1	569.1	5871.0
Coefficients: (4 not defined because of singularities)					
		Estimate	Std. Error	t value	Pr(> t )
(Intercept)		-1.336e+05	1.046e+04	-12.771	< 2e-16 ***
RSO_Weight		-2.426e+00	2.891e+00	-0.839	0.401399
RSO_DensityLow		-1.892e+00	6.436e+01	-0.029	0.976545
RSO_DensityMedium		-3.987e+01	1.272e+02	-0.314	0.753856
RSO_DensityRegular		6.002e+01	6.681e+01	0.898	0.369031
RSO_Visibility		-1.112e+02	2.612e+02	-0.426	0.670308
RSO_NameCBERS		-1.896e+02	1.430e+02	-1.326	0.184766
RSO_NameCorona		-2.448e+02	1.444e+02	-1.695	0.090036 .
RSO_NameDove		-2.681e+02	1.479e+02	-1.812	0.070060 .
RSO_NameEnviSAT		-2.420e+02	1.427e+02	-1.695	0.090069 .
RSO_NameEO-1		-2.342e+02	1.457e+02	-1.608	0.107918
RSO_NameLandsat		-3.026e+02	1.456e+02	-2.079	0.037679 *
RSO_NamePleiades		-2.297e+02	1.458e+02	-1.576	0.115160
RSO_NamePROBA		-2.586e+02	1.443e+02	-1.792	0.073160 .
RSO_NameQuickbird		-2.480e+02	1.548e+02	-1.602	0.109214
RSO_NameRapidEye		-2.771e+02	1.687e+02	-1.642	0.100578
RSO_NameSentinel		-3.006e+02	1.496e+02	-2.009	0.044546 *
RSO_NameSkySat		-2.450e+02	1.639e+02	-1.495	0.134966
RSO_NameSPOT		-1.892e+02	1.587e+02	-1.192	0.233265
RSO_NameTerra		-2.896e+02	1.490e+02	-1.944	0.051997 .
RSO_NameWorldview		-3.464e+02	1.757e+02	-1.972	0.048686 *
RSO_MRP		1.150e+01	2.231e-01	51.534	< 2e-16 ***
Orbit_IDOUT013		1.561e+03	3.507e+02	4.450	8.75e-06 ***
Orbit_IDOUT017		1.175e+03	7.557e+01	15.545	< 2e-16 ***
Orbit_IDOUT018		-7.583e+02	3.466e+02	-2.188	0.028706 *
Orbit_IDOUT019		2.304e+03	3.946e+02	5.839	5.54e-09 ***
Orbit_IDOUT027		2.278e+03	3.435e+02	6.632	3.61e-11 ***
Orbit_IDOUT035		1.709e+03	3.693e+02	4.629	3.75e-06 ***
Orbit_IDOUT045		1.463e+03	6.354e+01	23.030	< 2e-16 ***
Orbit_IDOUT046		2.008e+03	3.668e+02	5.475	4.55e-08 ***
Orbit_IDOUT049		5.437e+02	3.419e+02	1.590	0.111911
Orbit_Estab_Year		6.583e+01	5.239e+00	12.564	< 2e-16 ***
Orbit_HeightGEO		8.300e+02	3.427e+02	2.422	0.015466 *
Orbit_HeightLEO		-2.985e+02	3.634e+02	-0.821	0.411528
Orbit_HeightMEO		1.189e+03	3.366e+02	3.532	0.000416 ***
Stealth_TypeStealth_1		1.116e+03	1.329e+02	8.399	< 2e-16 ***
Stealth_TypeStealth_2		2.603e+03	1.405e+02	18.528	< 2e-16 ***
Stealth_TypeStealth_3		3.260e+03	1.565e+02	20.833	< 2e-16 ***
Stealth_TypeStealth_4		1.440e+03	1.780e+02	8.091	7.13e-16 ***
RSO_TypeRSO_Type2		NA	NA	NA	NA
RSO_TypeRSO_Type3		NA	NA	NA	NA

```
RSO_TypeRSO_Type4           NA      NA      NA      NA  
Orbit_Establ_Year          NA      NA      NA      NA  
---  
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 932.1 on 5917 degrees of freedom  
Multiple R-squared:  0.6997, Adjusted R-squared:  0.6978  
F-statistic: 362.8 on 38 and 5917 DF,  p-value: < 2.2e-16
```

```
predict_3<-predict(model3,X_cv)
```

### 6.3.2 Model Performance

Now we compute the  $R^2$  and RMSE of our model and its predictions to assess its performance. The model is an improvement over Model 2. We can now state that nearly 70% of the variability in Survivability scores are explained by the model and its significant features. Moreover, the RMSE has not been reduced.

```
RMSE<-rmse(model3$fitted.values, predict_3)  
R.sq<-rsq(model3)  
print(paste("R-square =", R.sq))
```

```
[1] "R-square = 0.699705959823552"  
print(paste("RMSE =", RMSE))
```

```
[1] "RMSE = 2011.02791136764"
```

### 6.3.3 Residual Analysis

As before, we start our analysis by studying the residuals. We demonstrated a quick way to do this in the previous section, using the basic plot function.

```
plot(model3, lwd=3, col = 4, cex=.75, cex.lab=1,  
cex.main=1.25, cex.axis=.75)
```

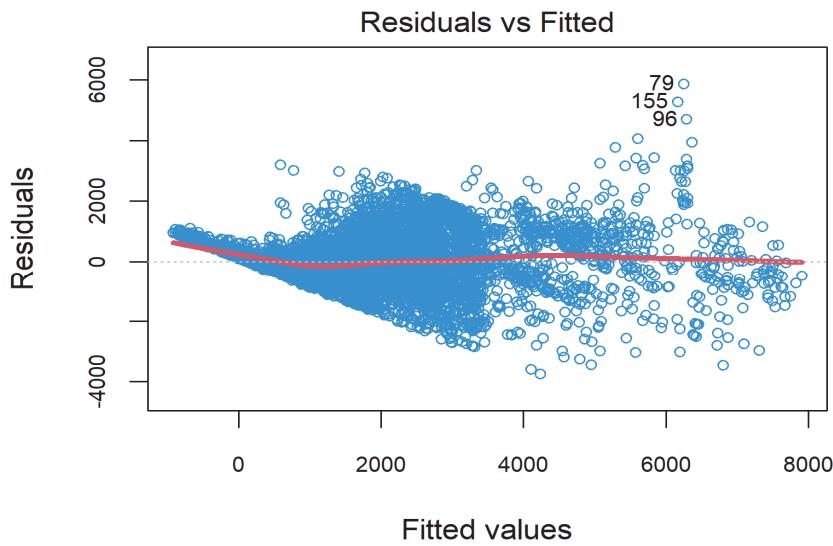


Figure 6-7. Model 3 residuals vs. fitted values plot showing non-linearity

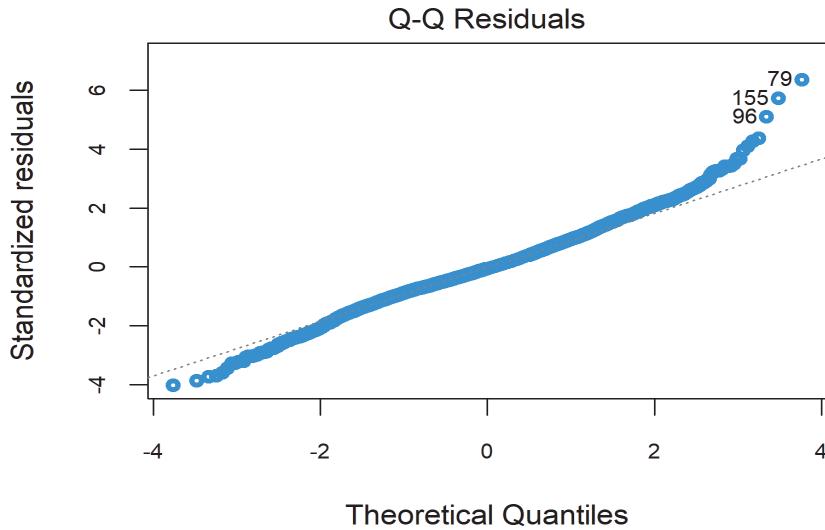
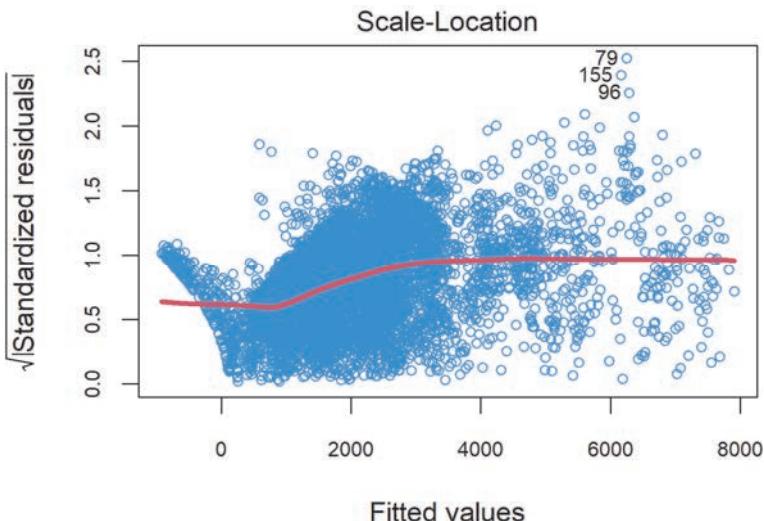
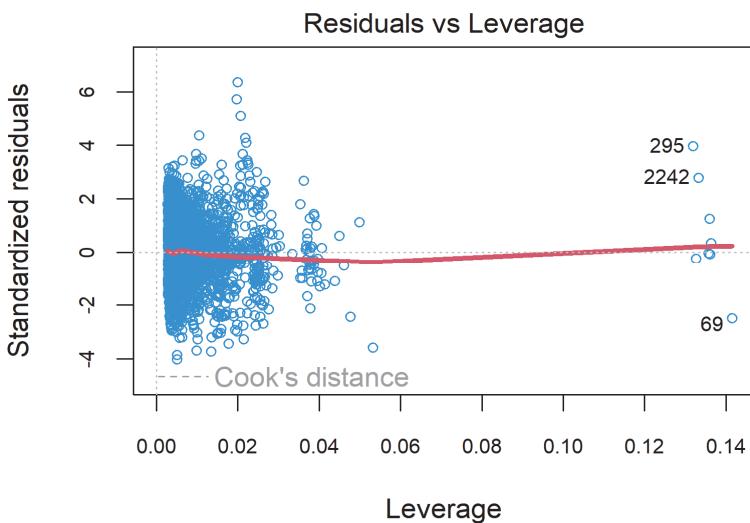


Figure 6-8. Model 3 Q\_Q plot showing non-linearity



*Figure 6-9. Scale-location plot using standardized residuals*



*Figure 6-10. Model 3 leverage plot using standardized residuals*

### 6.3.4 Conclusions from Residual analysis

The plots reveal several issues. One concern is introduction of outliers at data points 79, 96, and 155, which has the effect of squeezing the data toward the horizontal axis. This makes it difficult to analyze the residuals precisely. To correct this, we would normally remove the data outlier and repeat the modeling and analysis. However, in the next section, we

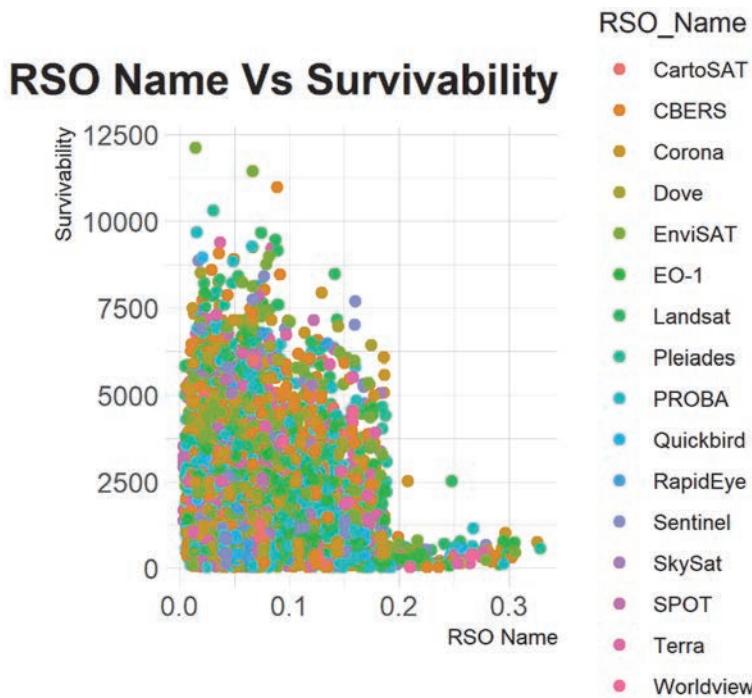
are going to try a different modeling type. Another concern is the apparent, increasing error variance.

The fourth plots shows Cook's Distance. Cook's Distance is an estimate of the influence of a data point. It considers both the leverage and residual of each observation. Cook's Distance is a summary of how much a regression model changes when the  $i^{th}$  observation is removed.

### 6.3.5 Scatterplots

Before we delve into another model, let us look at some more plots that may be useful for our analysis. The first plot in **Figure 6-11** we generate is for RSO Name vs. Survivability, which we categorized by RSO Name.

```
ggplot(train_3, aes(x=RSO_Name, y=Y_train,
color=RSO_Name)) +
  geom_point(size = 2) + theme_ipsum() +
  theme(plot.title = element_text(hjust=1)) +
  labs(title="RSO Name Vs Survivability",
y="Survivability", x="RSO Name")
```

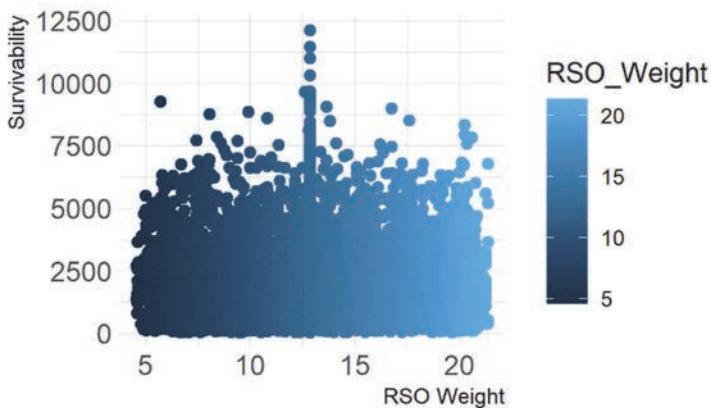


**Figure 6-11. RSO Name vs. Survivability**

From this scatterplot, we can see that most of the RSOs weigh less than 0.2 tons or approximately 4440 pounds. There is another pattern that implies the less an RSO weighs, the more survivable it is.

Now, we generate another scatterplot in **Figure 6-12** with RSO Weight vs. Survivability as before, but now we categorize the data by Stealth Type.

```
library(ggplot2)
gg <- ggplot(train_3, aes(x= RSO_Weight, y=Y_train, col=
RSO_Weight)) + geom_point(size = 2) + theme_ipsum() +
  labs(title="RSO Weight Vs Survivability",
       y="Survivability", x="RSO Weight")
gg
```

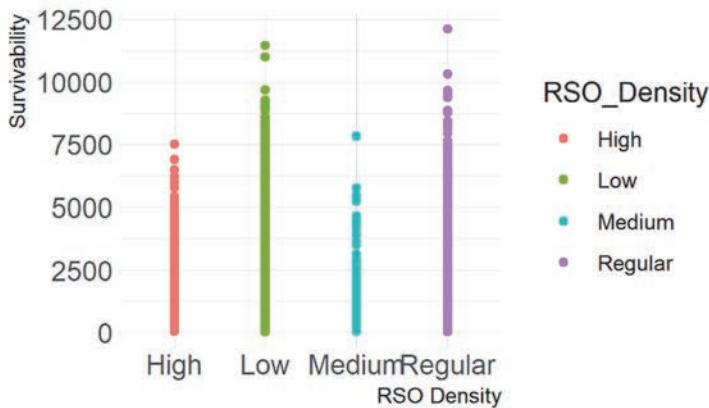


**Figure 6-12. Stealth Type vs. Survivability**

From this scatterplot, we can see that most of the RSOs weigh less than 0.2 tons or approximately 4440 pounds. There is another pattern that implies the less an RSO weighs, the more survivable it is.

Now, we generate a scatterplot in **Figure 6-13** using RSO Density vs. Survivability, but this time we categorize it by RSO Density.

```
gg <- ggplot(train_3, aes(x= RSO_Density, y=Y_train)) +
  geom_point(aes(col=RSO_Density)) + theme_ipsum() +
  labs(title="RSO Density Vs Survivability",
       y="Survivability", x="RSO Density")
gg
```

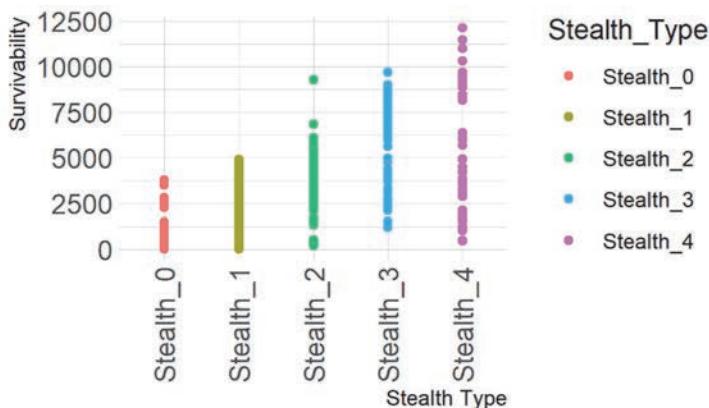


*Figure 6-13. RSO Density vs. Survivability*

There does seem to be a pattern of increasing survivability with increasing suitability, even though RSO Density was not a significant feature.

So, now let us look at RSO Stealth Type vs. Survivability (see **Figure 6-14**). Here, we categorize the data by Stealth Type.

```
gg <- ggplot(train_3, aes(x= Stealth_Type, y=Y_train)) +
  geom_point(aes(col=Stealth_Type)) + theme_ipsum() +
  labs(title="Stealth Type Vs Survivability",
       y="Survivability", x="Stealth Type")
gg
```



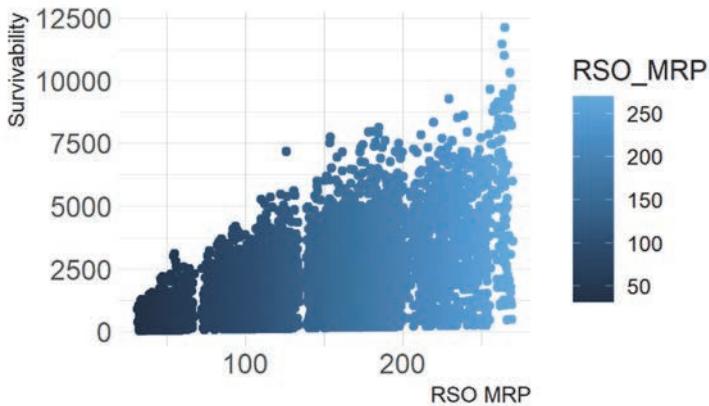
*Figure 6-14. Stealth type vs. Survivability*

Although it is difficult to see all the data points for each Stealth Type, there does seem to be a pattern of increasing survivability with increasing suitability.

Now, let us do the same analysis using **Figure 6-15** but categorize it as RSO MRP (recall that it is not significant in the model).

```
gg <- ggplot(train_3, aes(x=RSO_MRP, y=Y_train)) +  
  geom_point(aes(col=RSO_MRP)) + theme_ipsum() +  
  labs(title="RSO MRP Vs Survivability",  
       y="Survivability", x="RSO MRP")  
gg
```

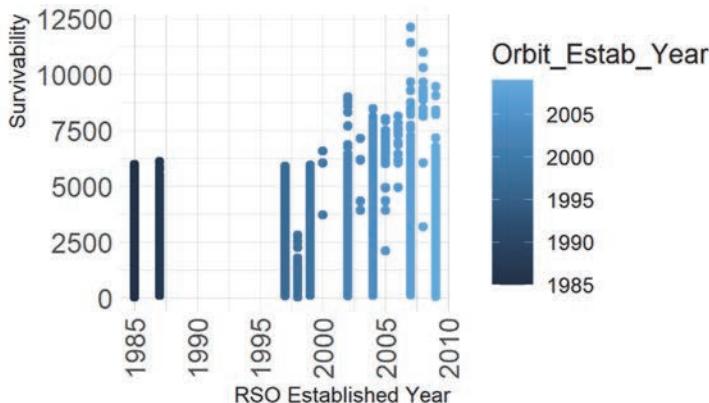
Again, there does seem to be a pattern of increasing survivability with increasing suitability, even though RSO MRP was not a significant feature.



*Figure 6-15. RSO MRP vs. Survivability*

Now, let us look at Establishment Year vs. Survivability in **Figure 6-16**, categorized by Orbit Height.

```
gg <- ggplot(train_3, aes(x=Orbit_Estab_Year,  
y=Y_train)) + geom_point(aes(col=Orbit_Estab_Year), size =  
2) + theme_ipsum() +  
  labs(title="Year Established Vs Survivability",  
       y="Survivability", x="RSO Year Established ")  
gg
```



*Figure 6-16. RSO Established Year vs. Survivability*

The plot seems to demonstrate that the longer the RSO has been established, the less survivable. Note that the farther to the left an RSO is located in the plot, it has been in orbit for less time than those on the right. Also note that the data is skewed, i.e., not symmetric.

The plot tells the same story for Orbit Establishment vs. Survivability. Note that Stealth Type 4 is a younger technology.

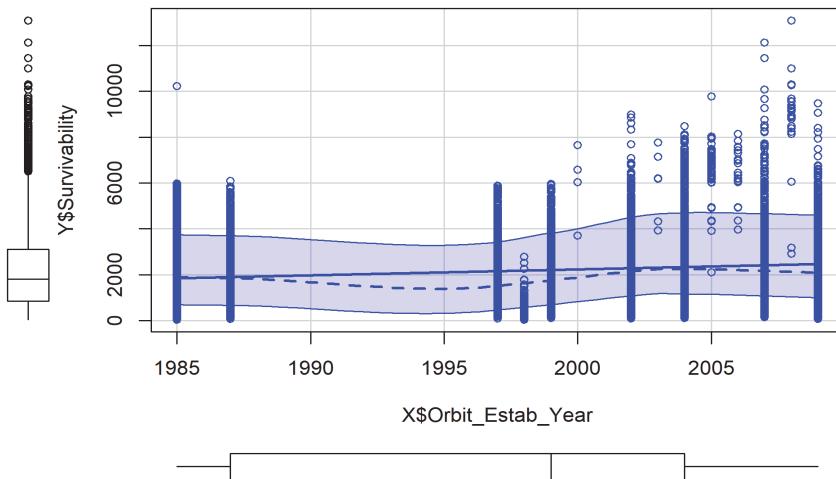
Now let us switch gears a bit and look at a different type of scatterplot, which is available in the car-package. This kind of scatterplot shows the information provided in previous plots but adds a few more diagnostics.

Referring to **Figure 6-17**, we first see a dashed-line in the plot that represents the trend of the data and around it, in the shaded light blue areas, a confidence interval is added.

Second, there is a solid blue liner fit line slightly above the trend curve that represents the average.

Finally, there are two box plots, one representing the survivability data and another representing the Established Year data. The box plots clearly supports the observation of skewness we made previously. We also see that a slight curvilinear fit might not be the best one, although our observation is not conclusive.

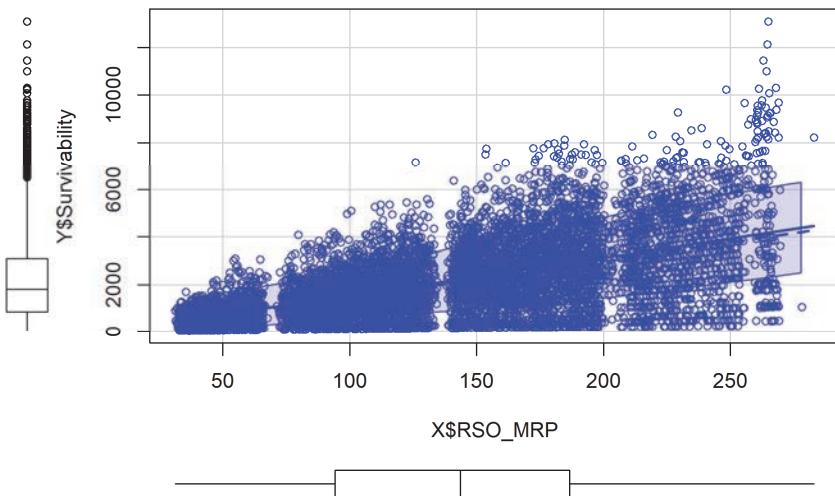
```
library(car)
scatterplot(X$Orbit_Estab_Year, Y$Survivability)
```



**Figure 6-17. Enhanced scatter plot of the year of orbit establishment vs. survivability. The trend seems to indicate that older the orbit, the less survivable is the RSO**

The next scatterplot in **Figure 6-18** is for RSO MRP vs. Survivability. The plot supplies the information we did not previously observe. In this instance we can see that the underlying trend is linear and increasing, as well as its confidence interval. The boxplots show skewness in the data, and also point out the central tendency. This plot also implies that the linear regression model may not be the complex enough for the best modelling option.

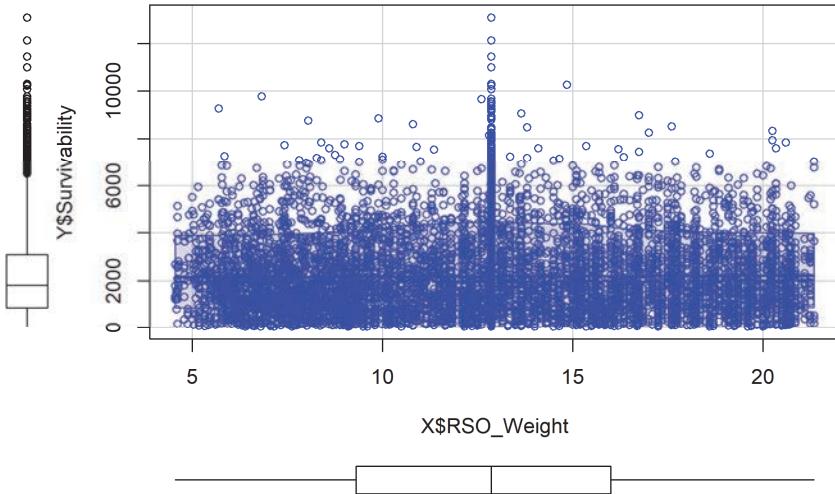
```
scatterplot(X$RSO_MRP, Y$Survivability)
```



**Figure 6-18. Scatterplot for RSO MRP and Survivability showing a leaner but increasing trend line.**

Finally, we look at RSO Weight vs. Survivability again in **Figure 6-19**. The plot does not add much information, except that the fit may be symmetric and showing an ideal weight of 13 kg for better survivability.

```
scatterplot(X$RSO_Weight, Y$Survivability)
```

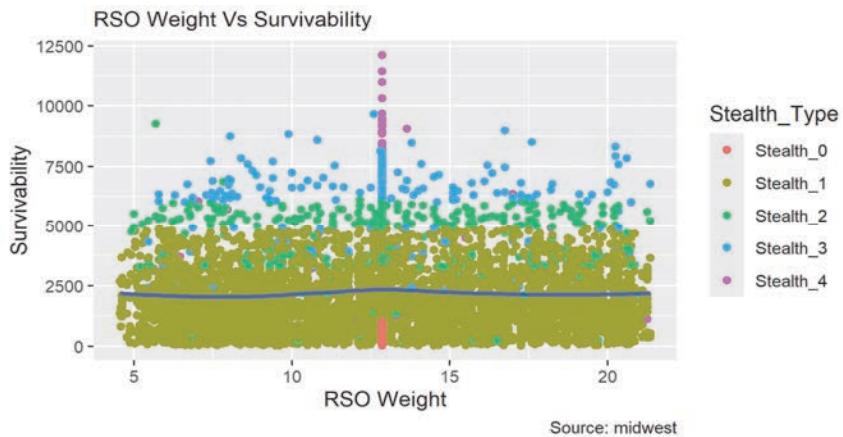


**Figure 6-19. Scatterplot of survivability by RSO weight**

### 6.3.6 Holistic Scatterplot

This final scatterplot (**Figure 6-20**) is different from previous ones, in that we can see the data categorized by Stealth Type and observe the darker color blue tend line is horizontal. This indicates a stratified layering of survivability by weight. One thing that stands our is the more sophisticated stealth types appear in the top layers of survivability. We also see the horizontal line that we saw in **Figure 6-17**.

```
gg <- ggplot(train_3, aes(x=RSO_Weight, y=Y_train)) +
  geom_point(aes(col=Stealth_Type)) +
  geom_smooth(method="loess", se=F) +
  labs(subtitle="RSO Weight Vs Survivability",
       y="Survivability",
       x="RSO Weight",
       title="Scatterplot",
       caption = "Source: Midwest")
gg
```



**Figure 6-20. Scatterplot of RSO weight vs. survivability by Stealth type**

### 6.3.7 Model 3 Performance

Aside from the diagnostics we performed with our plots, we will now look at some numerical metrics. Recall the MSE from Model 2 is 1979562 and its  $R^2$  value is 0.3322661. Model 3's MSE is close to zero, which is a drastic improvement. Also, the  $R^2$  of 1.0 shows that the model is a perfect fit. But wait! Didn't the plots reveal some potential issues? They did, and this is a model that over-fits the data!

```

par(mar=c(10,4,2,1))
barplot(model3$coefficients, main = "Model 3
Coefficients", ylab = "Coefficients", las = 2,
cex = .75, cex.lab = .75, cex.main = 1.25,
cex.sub = .75, cex.axis = .75, las = 2,
col = "#ffa3a2")

```

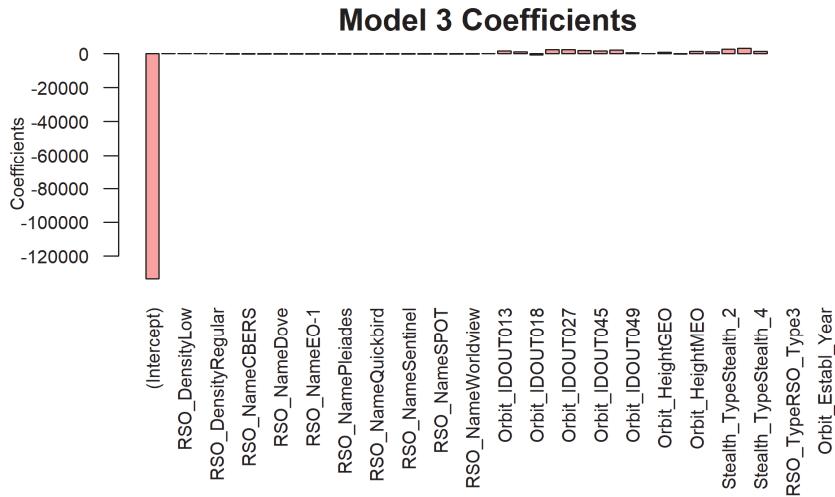


Figure 6-21. Model coefficient plot for model 3

### 6.3.8 Bar Plot Observations

We can see that magnitude of the coefficients for:

Orbit_IdentifierOUT013	Orbit_IdentifierOUT018
Orbit_IdentifierOUT027	Orbit_IdentifierOUT045
Orbit_IdentifierOUT049	Orbit_HeightGEO
Orbit_HeightMEO	Stealth_Type2
Stealth_Type3	RSO_Type3,

are much higher as compared to the rest of the coefficients. Therefore, the survivability of an RSO would be more driven by these features or levels of features. This might indicate an over-fitted model and validates our prior observation.

How can we reduce the magnitude of coefficients in our model to correct for over-fitting? For this purpose, we have different types of regression

techniques which uses regularization to overcome this problem. So let us discuss them.

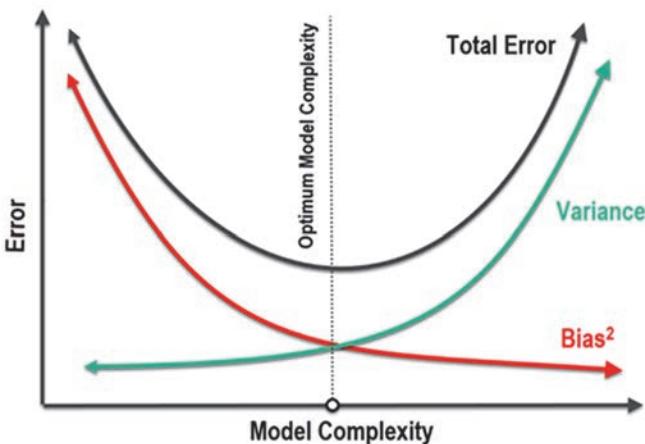
## 6.4 A Glimpse of Lasso Regression

### 6.4.1 Regularization

From Chapter 5, recall that when building a model, we want training and test errors to be as small as possible. To accomplish this, we used polynomial terms to add complexity to our models.

Referring to [Figure 6-22](#), the curve on top represents the total error of the trained model with the optimal complexity obtained where the dotted line crosses the black curve. Recall that increasing model complexity prevents underfitting, while more complex models may lead to overfitting, such that our model will not generalize well when we use the test data. We discussed how to find optimal complexity of a model to reduce both the fit error and prediction error.

Another way we can address model training vs generalization is to examine what we call **bias-variance tradeoff**. We show this by the red and green curves in [Figure 6-22](#). The method we use is regularization, which will take our existing model and find its optimal complexity.



*Figure 6-22. Illustration of the bias-variance tradeoff along with the total complexity error.*

So, what do we mean by this term **regularization**? Recall the method by which our model learns the parameters from the data is to increase

complexity while minimizing some cost function. As we saw for linear regression, the model would minimize the mean squared error, that is the error between the outcome variable and our predicted variable squared. We now introduce an added cost function for regularization:

$$M(w) + \lambda R(w)$$

where,  $M(w)$  is the model error or original cost function  $R(w)$  is the function of estimated parameters  $\lambda$  is the regularization power parameter

The regularization portion of the added cost function is  $R(w)$  multiplied by  $\lambda$ . We add this term to the original cost function so that we can increasingly penalize the model when it becomes too complex. This will allow us to “dumb-down” the model. So, by strengthening our parameter weights, and strengthening our model parameters, we increase the value of cost function. However, we are trying to minimize the cost function, so we’re not going to be able to fit the model as close to the training data, as we might like. So, we add a penalty.

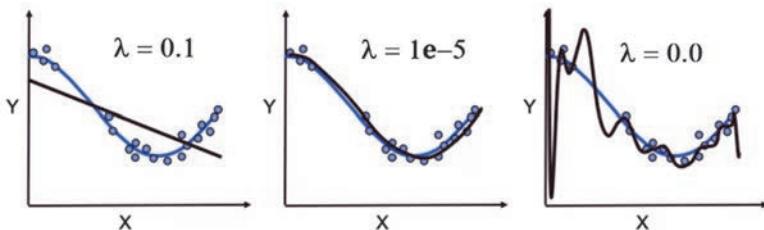
## 6.4.2 The Regularization Parameter

The  $\lambda$  is going to add a penalty proportional to the size of the strength of these parameters, or some function of these parameters, namely  $R(w)$ . We will examine more closely how  $\lambda R(w)$  can be a function of the parameters later on. The takeaway is that the larger we make  $\lambda$ , the more we penalize stronger parameters. And the more we penalize our model for being stronger and having stronger parameters, the less complex that model will be. This will cause us to try minimizing the strength of all of our parameters while minimizing our original cost function,  $M(w)$ . So, increasing the regularization parameter  $\lambda$  allows us to manage the complexity tradeoff, but increases the model bias as shown by the red curve in [Figure 6-22](#). On the other hand, less regularization makes it so that the model is more complex and will increase the variance. That is, if increase the complexity of a model, we may overfit the model (on the training set), and we will also increase the variance (the green curve in [Figure 6-22](#)).

Now let us take this concept of  $\lambda$  as it relates to three coefficients in their polynomial graphs. The idea here is to find the true function of the blue lines represent, given our sample data. So, let us say that we are starting with a model that is just polynomial degree fifteen. Rather than testing

different polynomials of degree 1, 5 and 15 we can introduce the blue  $\lambda$  term here, and as illustrated in [Figure 6-23](#).

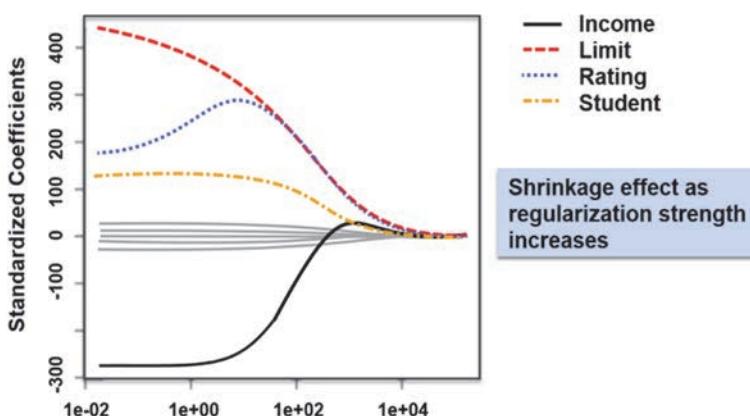
$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^m \left( (\beta_0 + \beta_1 x_{obs}^i) - y_{obs}^i \right)^2 + \lambda \sum_{j=1, j \neq k} \beta_{j2}$$



[Figure 6-23. The effect of using the  \$\lambda\$ -term rather than the polynomial degrees..](#)

Now, let us think about regularization in the context of feature selection, or determining which features are important to include in our model. So, regularization is going to be a form of feature selection. That is, regularization is going to take the contribution of each feature and eliminate or reduce features as it adds more weight to the penalty.

Now what happens with regards to the relationship between the  $\lambda$ -values and those standardized coefficients. Generally, as seen in [Figure 6-24](#), moving to the right as  $\lambda$  increases, the standardized coefficients should decrease, so there should be that inverse relationship.



[Figure 6-24. Coefficient value change vs regularization \( \$\lambda\$ \) change.](#)

We do see these ratings increase, as the other features are decreasing and that would just be something due to multicollinearity. At a certain point, we will see they all start to decrease, and they are all decreasing monotonically towards 0 once they reach a certain threshold. So, ideally as we increase alpha, the value we choose when working with *sklearn*, but here we call it lambda. As we increase lambda we will decrease each one of the coefficients.

```
lasso_reg_cv = cv.glmnet(X_train, Y_train, alpha = 1)
coef(lasso_reg_cv)
plot(lasso_reg_cv)
```

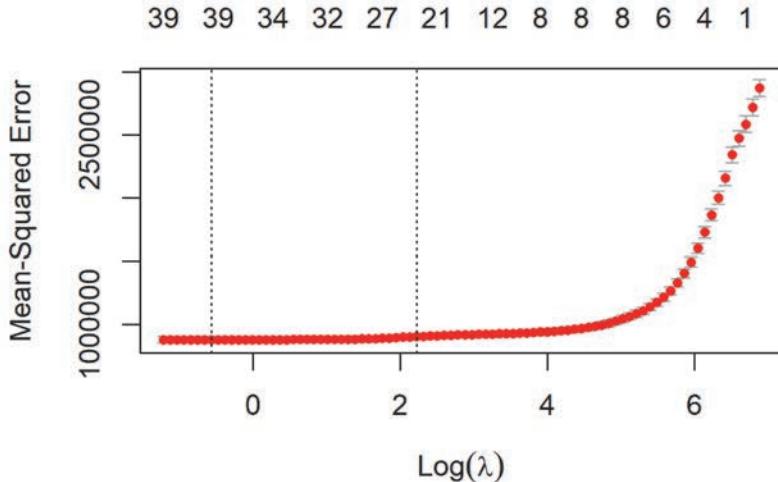


Figure 6-25. MSE plot of the logarithm of regularization parameter.

#### 6.4.3 Regularization and Feature Reduction

Feature reduction will be most obvious when we work with Lasso regression, as Lasso will drive some of the coefficients in our linear regression down to zero. So, if we may think about a zero-coefficient as removing the contribution of that feature altogether. This would have the same effect as manually removing some features prior to modeling. Lasso will find which ones to remove automatically according to some mathematical formula.

#### 6.4.4 Making the Tradeoff

Using *Figure 6-26*, we see this complexity trade-off and it is possible that variance reduction may actually outpace the increase in bias. So, we may

find a better fit model without having to increase bias too much. What we mean here is, we can reduce the complexity while still consistently having enough information to show that relationship between  $X$  and  $y$  in our training set. So, we are not increasing bias too much, rather we're able to reduce complexity for some time while barely affecting that bias. And this may happen if we are starting with an extremely over fit model, and we can keep reducing variance without increasing the bias. Then we eventually find that optimal value that is going to allow us to have the lowest mean squared error on our holdout set. So, the red "X" marks that spot, as we see variance and bias minimized as well.

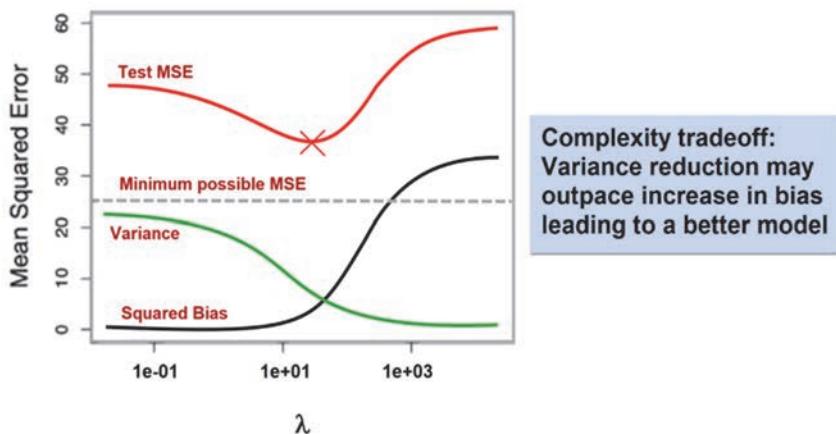


Figure 6-26. complexity trade-off and its possible that variance reduction

## 6.5 Lasso Regression

LASSO (Least Absolute Shrinkage Selector Operator) is quite like ridge, but we will try to understand the difference between them by implementing it in our satellite survivability problem. Let us first discuss some preliminaries.

### 6.5.1 Lasso Regression Preliminaries

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e., models with fewer parameters). This type of regression is well-suited for models showing important levels of multicollinearity or when you want

to automate certain parts of model selection, like variable selection/parameter elimination.

### 6.5.2 L1 Regularization

Lasso regression performs  $L^1$  regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models. On the other hand,  $L^2$  regularization (e.g., Ridge regression) does not result in elimination of coefficients or sparse models. This makes the LASSO far easier to interpret than the Ridge. The goal of the algorithm is to minimize:

$$\sum_{i=1}^n \left( y_i - \sum_j x_{(ij)} \beta_j \right)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

Which is the same as minimizing the sum of squares with constraint  $\sum |\beta_j| \leq s$ . Some of the  $\beta$ s are shrunk to exactly zero, resulting in a regression model that is easier to interpret. So, the basic difference between Ridge and Lasso is the way we penalize the regression.

```
if(!require(glmnet)) install.packages("glmnet")
if(!require(superml)) install.packages("superml")
if(!require(rsq)) install.packages("rsq")
if(!require(lars)) install.packages("lars")
if(!require(caret)) install.packages("caret")
```

### 6.5.3 Load Imputed Data

```
library(glmnet)
train2 <-
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\survive2.csv")
```

### 6.5.4 Data Standardization

Now, we want to keep in mind that with our original linear regression cost function, scaling would not have a large effect on our eventual outcome. But now that we have added coefficient weights to our cost function, scale will be of utmost importance.

For example, suppose we're working with two variables to predict survivability for a certain resident space objects (RSOs). One of our variables  $x_1$  is going to be the altitude of the RSO and the other  $x_2$  variable is going to be the stealth technology used for each RSO. `Orbit_Height` is an integer classification between 1 and 3 and the `RSO_Weight` is an integer between 4.5 kg and 21.4 kg, the coefficient for Survival of 1 unit-change will have a much larger coefficient with regards to its change in altitude,  $x_1$ , compared to a one-unit change in weight,  $x_2$ .

Therefore, if we end up with a higher price coefficient we will end up being highly penalized for that coefficient. We are going to penalize large coefficients using our new ridge regression. Hence, we need to first ensure that all of our distinctive features are on the same scale. And we can do that using the standardization technique that we see here, just subtracting the mean and dividing by the standard deviation.

$$\hat{x}' = \frac{(x - \bar{x})}{\sigma}$$

So, let us take or satellite survivability data and standardize it in R. We will write the standard variables as  $x'$  for each. Later, we will create a function that does all of the standardization automatically.

```
library(glmnet)
train <-
read.csv("C:\\\\Users\\\\jeff\\\\Documents\\\\Data\\\\Survive.csv")
#Import the train set

train$RSO_Weight[is.na(train$RSO_Weight)] <-
  mean(train$RSO_Weight, na.rm = TRUE)
train$RSO_Density[is.na(train$RSO_Density)] <- "Low"
train$RSO_Visibility[train$RSO_Visibility == 0] <-
  mean(train$RSO_Visibility)
train$RSO_MRP[is.na(train$RSO_MRP)] <- mean(train$RSO_MRP,
  na.rm = TRUE)
train$Orbit_Estab_Year=2013 - train$Orbit_Estab_Year
train$Orbit_Height[is.na(train$Orbit_Height)] <- "LEO"
train$Stealth_Type[is.na(train$Stealth_Type)] <-
  "Stealth_1"
train$RSO_Type[is.na(train$RSO_Type)] <- "RSO_Type1"
```

## 6.5.5 Define the Features and Response Sets

Now, that we have imputed missing values, we need to define the set of features and the response,  $Y$ .

```
Train <- train[c(-1)]  
Y <- train[c(11)]
```

## 6.5.6 Matricize the Data

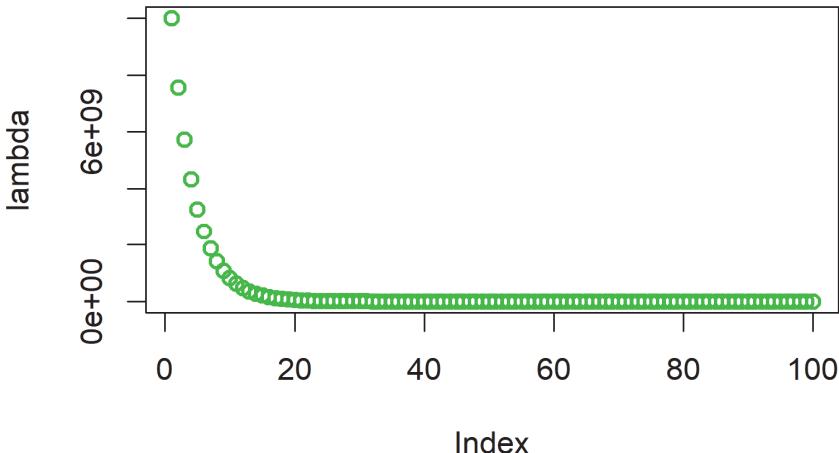
Next, we take the response and features and form a model matrix.

```
X <- model.matrix(Survivability ~ ., train)
```

## 6.5.7 Define Lambda

We also need to define the initial value of  $\lambda$  and the stop and step values that we will iterate through and plot the values (see *Figure 6-27*).

```
lambda <- 10^seq(10, -2, length = 100)  
plot(lambda, col = "green3", lwd = 2)
```



*Figure 6-27. Scatterplot of lambda values vs index.*

## 6.5.8 Split the Data into Subsets

Finally, we split the set into the training set and the cross-validation set, which will complete our data preprocessing.

```
set.seed(567)  
part <- sample(2, nrow(X), replace = TRUE,
```

```

prob = c(0.7, 0.3))
X_train <- X[part == 1,]
X_cv <- X[part == 2,]

X_train <- as.matrix(X_train)
X_cv <- as.matrix(X_cv)

Y_train <- Y[part == 1,]
Y_cv <- Y[part == 2,]

```

## 6.6 Lasso Regression Model Construction

Here, we also use the `glmnet` function to build the Lasso regression model and use it to predict future values. Recall  $\lambda$  is the parameter we use for hyperparameter tuning, rather than a model value or variable. If model parameters are variables that get adjusted by training with existing data, your hyperparameters are the variables about the training process itself. For example, part of setting up a deep neural network is deciding how many “hidden” layers of nodes to use between the input layer and the output layer, as well as how many nodes each layer should use. These variables are not directly related to the training data at all. They are configuration variables. Another difference is that parameters change during a training job, while the hyperparameters are usually constant during a job. A detailed discussion is beyond the scope of this book. See Strickland (Regression totum modum, 2023).

```

lasso_reg <- glmnet(X_train, Y_train, alpha = 1,
                      lambda = lambda, type.measure = "mse")
bestlam <- lasso_reg$lambda.min

```

## 6.7 MODEL 6: Lasso Regression

Now, we construct a lasso regression model using different lambda value settings to search for the **optimal lambda**. We use the `cv.glmnet` function to fit a model to the data and then the model evaluates the resulting lambdas until it finds an optimal value.

For our Lasso regression model, **Figure 6-28** shows the path followed to reach the optimal lambda.

```

library(glmnet)
lambda <- 10^seq(0, -3, by = -.05)

cv_lasso <- cv.glmnet(X[X_train,], Y[X_train], alpha = 1,

```

```
lambda = lambdas)
optimal_lambda <- cv_lasso$lambda.min
```

```
optimal_lambda
[1] 0.001
```

```
par(mar=c(4,4,1,1))
plot(lambdas, cv_lasso$cvm, ylab = "Mean-Squared Error", xlab =
= "Lambda", type = "l", lwd = 3, col = "dodgerblue")
```

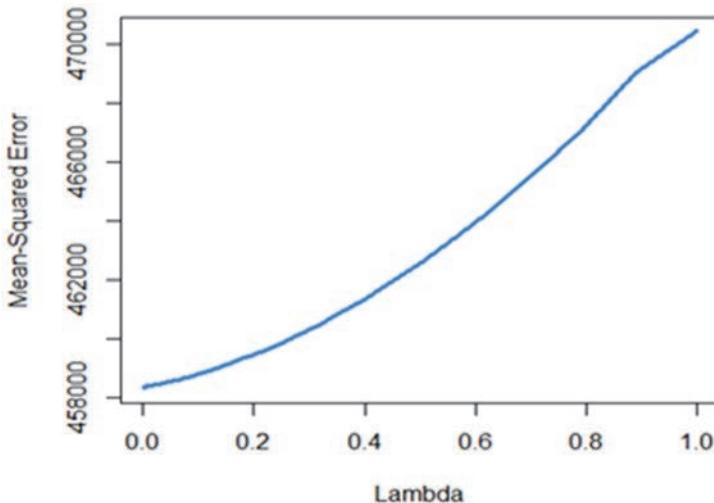


Figure 6-28. Search path for finding the optimal lambda

After finding an optimal lambda, we proceed to use it in building the Lasso regression model shown below. Notice that we use the optimal lambda name in the model rather than the value of a specific lambda.

```
lasso_reg = glmnet(X_train, Y_train, alpha = 1, family =
= 'gaussian', lambda = optimal_lambda, thresh = 1e-07)
summary(lasso_reg)
```

	Length	Class	Mode
a0	1	-none-	numeric
beta	42	dgCMatrix	S4
df	1	-none-	numeric
dim	2	-none-	numeric
lambda	1	-none-	numeric
dev.ratio	1	-none-	numeric
nulldev	1	-none-	numeric
npasses	1	-none-	numeric
jerr	1	-none-	numeric

```
offset      1     -none-    logical
call        7     -none-    call
nobs       1     -none-    numeric
```

```
coef(lasso_reg)
```

```
43 x 1 sparse Matrix of class "dgCMatrix"
          s0
(Intercept) -424.581637
(Intercept) .
RSO_Weight   -2.432813
RSO_DensityLow -2.833222
RSO_DensityMedium -40.628563
RSO_DensityRegular 59.461384
RSO_Visibility -109.468891
RSO_NameCBERS -141.834569
RSO_NameCorona -197.200116
RSO_NameDove   -220.414086
RSO_NameEnvisAT -194.254047
RSO_NameEO-1   -186.604290
RSO_NameLandsat -255.015177
RSO_NamePleiades -182.671418
RSO_NamePROBA  -211.732992
RSO_NameQuickbird -200.826216
RSO_NameRapidEye -229.945463
RSO_NameSentinel -253.511820
RSO_NameSkySat  -197.732627
RSO_NameSPOT    -141.864876
RSO_NameTerra   -242.403607
RSO_NameWorldview -299.262197
RSO_MRP        11.501461
Orbit_IDOUT013 758.857889
Orbit_IDOUT017 485.609225
Orbit_IDOUT018 -301.773709
Orbit_IDOUT019 2150.288083
Orbit_IDOUT027 1806.879279
Orbit_IDOUT035 893.153003
Orbit_IDOUT045 776.215479
Orbit_IDOUT046 1193.528655
Orbit_IDOUT049 -258.969855
Orbit_Estab_Year -66.045037
Orbit_HeightGEO 947.338301
Orbit_HeightLEO -170.094144
Orbit_HeightMEO 1305.138076
Stealth_TypeStealth_1 1089.700748
Stealth_TypeStealth_2 2576.287781
Stealth_TypeStealth_3 3232.967971
Stealth_TypeStealth_4 1411.182096
```

RSO_TypeRSO_Type2	-1261.037155
RSO_TypeRSO_Type3	-330.579798
RSO_TypeRSO_Type4	-685.041039

## 6.8 Model Evaluation

Here, we write a function to calculate **R-squared** and **RMSE**.

```
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SSR <- sum((true - mean(true))^2)
  SST<-SSR+SSE
  R_square <- SSR / SST
  RMSE = sqrt(SSE/nrow(df))
  return(data.frame(
    RMSE,
    Rsquare = R_square
  ))}
```

## 6.9 Prediction and evaluation on train data

Using the `predict` function and the training data set, we capture future values of the response.

```
predictions_train <- predict(lasso_reg, s = optimal_lambda,
                             newx = X_train)
res1 <- eval_results(Y_train, predictions_train, train)
```

## 6.10 Prediction and evaluation on test data

Using the `predict` function and the cross-validation data set, we capture future values of the response. Recall that the argument `newx` requires a hold-out or cross validation set, rather than the set we trained the model with.

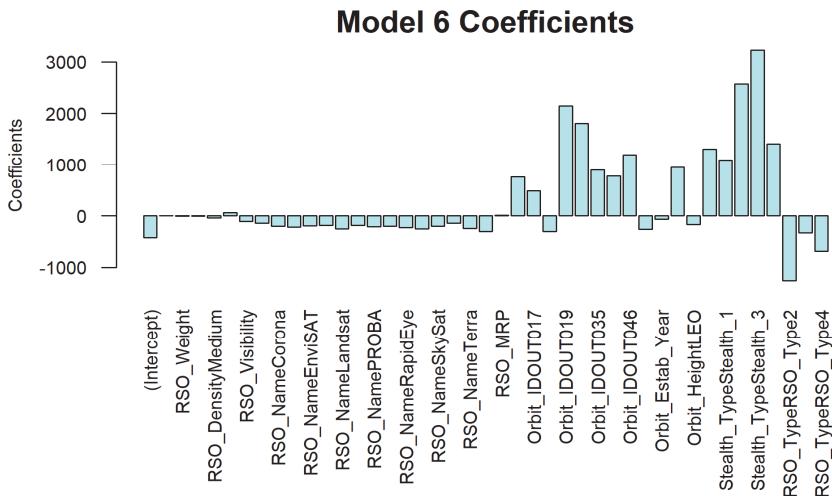
```
predictions_cv <- predict(lasso_reg, s = optimal_lambda,
                           newx = X_cv)
res2 <- eval_results(Y_cv, predictions_cv, X_cv)
print(paste("Lasso Regression Model =", res1))
```

```
[1] "Lasso Regression Model = 776.621664002712"
[2] "Lasso Regression Model = 0.769048165983"
```

```
print(paste("Lasso Cross Validation =", res2))
```

```
[1] "Lasso Cross Validation = 924.352437717525"
[2] "Lasso Cross Validation = 0.776788424284334"
```

```
par(mar=c(10,4,2,1))
lasso.coef<-predict(lasso_reg, type = "coefficients", s =
bestlam)[1:43,]
barplot(lasso.coef, main="Model 6 Coefficients",
ylab="Coefficients", las=2, cex=.75, cex.lab=.75,
cex.main=1.25, cex.sub=.75, cex.axis=.75, las=2,
col="#acffff")
```



*Figure 6-29. Plot of model coefficients with respect to their numeric values (not order of importance)*

The results show that Model 6 is the best fit thus far, with an RMSE of approximately 925 and an R-squared of 0.71. This means that the model features explain 71% of the variation in the Survivability measure.

```
library(caret)
V = varImp(lasso_reg, lambda = 0.0001)
#Insert new column
s <- nrow(V)
new <- seq(s)
V$Variables <- new
#Remove insignificant Overall importance values
#Insignificant values < median value
#Transform from numerical to logical
V_log<- V>median(V$Overall)
V1_log <- V_log == TRUE
```

```

#Transform to (0,1)
V2 = V1_log-FALSE
#Transform to numerical with insignificant = 0
V3 = V*V2
#Convert to data frame
V4 <- as.data.frame(V3)
#Remove rows containing 0 overall values
V5 <- V4[!(V4$Overall == 0),]
#Convert to data frame
V5 <- as.data.frame(V5)
#Rename "V5" column to "Overall"
names(V5)[1] <- paste('Overall')
#Count variable reduction
nrow(V)

```

[1] 43

**nrow**(V) - **nrow**(V5)

[1] 22

Here is code for plotting the model variables in the order of importance, as we see in **Figure 6-30**.

```

my_ggp <- ggplot2::ggplot(V5,
aes(x=reorder(rownames(V5),Overall), y=Overall)) +
  geom_point(aes(color= (factor(rownames(V5)))), size=5,
  alpha=0.6) +
  geom_segment(aes(x=rownames(V5), y=0 , xend=rownames(V5),
  yend=Overall),
  color='skyblue', size = 1.5) +
  ggtitle("Variable Importance using Lasso Regression") +
  guides(color = guide_legend(title = "Important
  Variables")) +
  xlab('') + ylab('Overall Importance') +
  coord_flip()

my_ggp + theme_light() +
  theme(axis.title = element_text(size = 14)) +
  theme(axis.text = element_text(size = 12)) +
  theme(plot.title = element_text(size = 14)) +
  theme(legend.title = element_text(size = 13)) +
  theme(legend.text = element_text(size = 11))

```

## Variable Importance using Lasso Regression

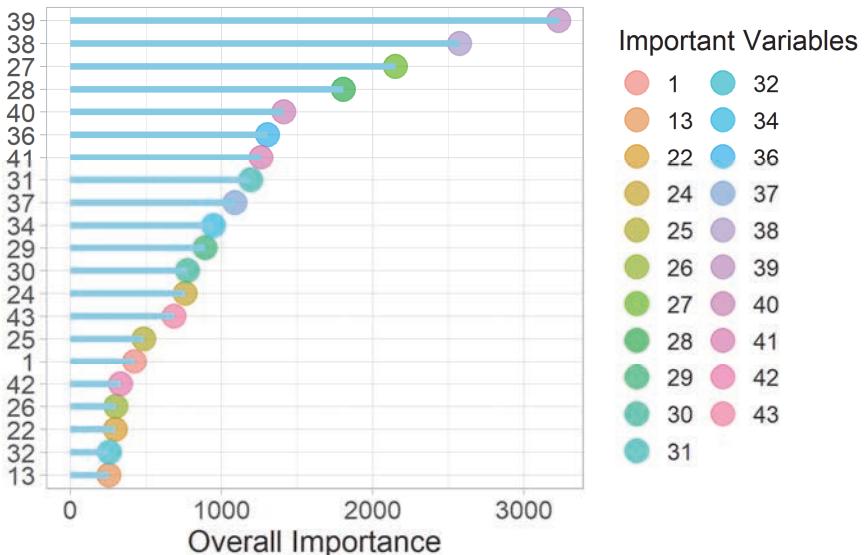


Figure 6-30. Horizontal bar-plot showing the model variables in the order of importance (top-to-bottom)

```
V = varImp(lasso_reg, lambda=0.0001)
#Insert new column
s <- nrow(V)
new <- c("Intercept",
        "RSO_Weight",
        "RSO_Density",
        "RSO_Visibility",
        "RSO_Name",
        "RSO_MRP",
        "Orbit_ID",
        "Orbit_Estab_Year",
        "Orbit_Height",
        "Stealth_Type",
        "RSO_Type",
        "Survivability",
        "Orbit_Estab_Year",
        "RSO_Weight",
        "RSO_Density",
        "RSO_Visibility",
        "RSO_Name",
        "RSO_MRP",
        "Orbit_ID",
        "Orbit_Estab_Year",
        "Orbit_Height",
```

```
"Stealth_Type",
"RSO_Type",
"Survivability",
"Orbit_Estab_Year",
"RSO_Weight",
"RSO_Density",
"RSO_Visibility",
"RSO_Name",
"RSO_MRP",
"Orbit_ID",
"Orbit_Estab_Year",
"Orbit_Height",
"Stealth_Type",
"RSO_Type",
"Survivability",
"Orbit_Estab_Year",
"RSO_Weight",
"RSO_Density",
"RSO_Visibility",
"RSO_Name",
"RSO_MRP",
"Orbit_Identifier")
```

```
V$Variables <- new
```

### 6.10.1 Print Lasso Regression Coefficients

```
lasso.coef <- predict(lasso_reg, type = "coefficients",
  s = bestlam)[1:43,]
print(paste("Variable: Intercept =", lasso.coef[1]))
```

```
[1] "Variable: Intercept = -424.581636743193"
```

```
print(paste("Variable:",
  names(train), "=" ,lasso.coef[2:43]))
```

```
[1] "Variable: RSO_Weight = 0"
[2] "Variable: RSO_Density = -2.43281299380114"
[3] "Variable: RSO_Visibility = -2.83322219181735"
[4] "Variable: RSO_Name = -40.6285632917687"
[5] "Variable: RSO_MRP = 59.4613840789671"
[6] "Variable: Orbit_ID = -109.468890601793"
[7] "Variable: Orbit_Estab_Year = -141.834569300889"
[8] "Variable: Orbit_Height = -197.200115750145"
[9] "Variable: Stealth_Type = -220.414086274024"
[10] "Variable: RSO_Type = -194.254047464834"
[11] "Variable: Survivability = -186.60428950293"
[12] "Variable: RSO_Weight = -255.015177192499"
```

```

[13] "Variable: RSO_Density = -182.671417601287"
[14] "Variable: RSO_Visibility = -211.732992205273"
[15] "Variable: RSO_Name = -200.826216432103"
[16] "Variable: RSO_MRP = -229.945462968522"
[17] "Variable: Orbit_ID = -253.511819591486"
[18] "Variable: Orbit_Estab_Year = -197.73262671526"
[19] "Variable: Orbit_Height = -141.864875661411"
[20] "Variable: Stealth_Type = -242.403607244088"
[21] "Variable: RSO_Type = -299.262197079571"
[22] "Variable: Survivability = 11.5014607991469"
[23] "Variable: RSO_Weight = 758.857888579293"
[24] "Variable: RSO_Density = 485.609224574507"
[25] "Variable: RSO_Visibility = -301.773708677179"
[26] "Variable: RSO_Name = 2150.28808285611"
[27] "Variable: RSO_MRP = 1806.87927893014"
[28] "Variable: Orbit_ID = 893.153003145922"
[29] "Variable: Orbit_Estab_Year = 776.215479116192"
[30] "Variable: Orbit_Height = 1193.52865515672"
[31] "Variable: Stealth_Type = -258.969855261881"
[32] "Variable: RSO_Type = -66.0450365308911"
[33] "Variable: Survivability = 947.338300704269"
[34] "Variable: RSO_Weight = -170.094143929963"
[35] "Variable: RSO_Density = 1305.13807589406"
[36] "Variable: RSO_Visibility = 1089.70074753377"
[37] "Variable: RSO_Name = 2576.28778120382"
[38] "Variable: RSO_MRP = 3232.96797145437"
[39] "Variable: Orbit_ID = 1411.18209552646"
[40] "Variable: Orbit_Estab_Year = -1261.03715456234"
[41] "Variable: Orbit_Height = -330.579797763194"
[42] "Variable: Stealth_Type = -685.041038790335"

```

## 6.10.2 Lasso Model Performance

```

eval_results <- function(true, predicted, df) {
  SST <- sum((predicted-true)^2)
  SSE <- sum((true-mean(true))^2)
  R_square <- 1-SSE/SST
  RMSE = sqrt(SSE/nrow(df))
  return(data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  ))}

```

## 6.10.3 Cross-Validation Performance

```

predictions_cv <- predict(lasso_reg, s = optimal_lambda,
newx = X_cv)
res2<-eval_results(Y_cv, predictions_cv, X_cv)
print(paste("Lasso Regression Model =", res1))

```

```
[1] "Lasso Regression Model = 10458.1970"
[2] "Lasso Regression Model =      0.7690"

print(paste("Lasso Cross Validation =", res2))
```

```
[1] "Lasso Cross Validation = 924.3524"
[2] "Lasso Cross Validation =    0.7768"
```

## 6.10.4 Model Comparison Metrics Table

Now we will compare the three models we built in this section, using the prediction and evaluation of train data. The results show that Model 1 and Model 2 (the multiple linear models) do not explain very much of the variance, approximately 68% (I usually look for 70% or greater). Model 4 appears to be a good fit. It has an Adjusted  $R^2$  of about 0.74 and their RMSE are closest to the other models.

If we were just looking at the performance metrics, I would say the Model 3 (the mixed model) is slightly better. However, I cannot pretend I do not know the data. Consequently, I conclude that Model 3 does not account for the complexity of the data we are modeling and may overfit the data. So, I conclude that Model 4, the LASSO model is the best of the four models we evaluated. Note that Model 4 also has the lowest RMSE.

```
library(Metrics)
RMSE1 <- rmse(Y_cv, predict_1)
R.sq1 <- rsq(model1)
RMSE2 <- rmse(Y_cv, predict_2)
R.sq2 <- rsq(model2)
RMSE3 <- rmse(Y_cv, predict_3)
R.sq3 <- rsq(model3x)
pred4 <- predict(lasso_reg, s=optimal_lambda, newx = X_cv)
res4 <- eval_results(Y_cv, pred4_cv, X_cv)
print(paste("Mod1 R-square =", round(R.sq1,4), " | Mod1 RMSE",
           "=", round(RMSE1,4)))
print(paste("Mod2 R-square =", round(R.sq2,4), " | Mod2 RMSE",
           "=", round(RMSE2,4)))
print(paste("Mod3 R-square =", round(R.sq3,4), " | Mod3 RMSE",
           "=", round(RMSE3,4)))
print(paste("Mod4 R-square =", round(res4$Rsquare,4), " | Mod4 RMSE",
           "=", round(res4$RMSE,4)))
```

```
[1] "Mod1 R-square = 0.7474 | Mod1 RMSE = 1411.2688"
[1] "Mod2 R-square = 0.7527 | Mod2 RMSE = 1391.2449"
[1] "Mod3 R-square = 0.8448 | Mod3 RMSE = 1108.0763"
[1] "Mod4 R-square = 0.7768 | Mod4 RMSE = 924.3524"
```

## 6.11 Chapter Summary

We have seen that multiple regression can help answer complex questions. However, multiple regression models are weak when data introduce multicollinearity and complexity. When either of these situations occur, we have other models to address them. Lasso regression is a type of linear regression that uses shrinkage, where shrinkage is where data values are shrunk towards a central point, like the mean. The procedure models with fewer parameters. This type of regression is well-suited for:

- Models showing elevated levels of multicollinearity.
- Models where want to automate certain parts of model selection, like variable selection/parameter elimination.

Lasso regression performs  $L^1$  regularization, which adds a penalty to the coefficients, resulting in sparse models.

We use two R functions for lasso regression: `glmnet` and `cv.glmnet`. We also saw that we can use the Least Angle Regression (LARS) algorithm (`lars`) for high dimensional data.



## 7 Random Forest using R

**Random forest (RF)** (Breiman 2001a) is a non-parametric statistical method which does not require distributional assumptions on covariate relation to the response. RF is a robust, non-linear technique that optimizes predictive accuracy by fitting an ensemble of trees to stabilize model estimates. **Random Survival Forest (RSF)** (Ishwaran and Kogalur 2007; Ishwaran et al. 2008) is an extension of Breiman’s RF techniques to survival settings, allowing efficient non-parametric analysis of time to event data. The `randomForestSRC` package (<http://CRAN.R-project.org/package=randomForestSRC>) (Ishwaran and Kogalur 2014) is a unified treatment of Breiman’s (2001a) random forest for survival, regression and classification problems.

Predictive accuracy make RF an attractive alternative to parametric models, though complexity and interpretability of the forest hinder wider application of the method. We introduce the `ggRandomForests` package (<http://CRAN.R-project.org/package=ggRandomForests>) for visually exploring random forest models. The `ggRandomForests` package is structured to extract intermediate data objects from `randomForestSRC` objects and generate figures using the `ggplot2` graphics package (<http://CRAN.R-project.org/package=ggplot2>) (Wickman, 2010)).

Many of the figures created by the `ggRandomForests` package are also available directly from within the `randomForestSRC` package. However `ggRandomForests` offers the following advantages:

- Separation of data and figures: `ggRandomForests` contains functions that operate on either the `rfsrc` forest object directly, or on the output from `randomForestSRC` post processing functions (i.e., `plot.variable`, `var.select`) to generate intermediate `ggRandomForests` data objects. `ggRandomForests` functions are provide to further process these objects and plot results using the `ggplot2` graphics package. Alternatively, users can use these data objects for their own custom plotting or analysis operations.
- Each data object/figure is a single, self-contained unit. This allows simple modification and manipulation of the data or `ggplot` objects to meet users specific needs and requirements.

- We chose to use the `ggplot2` package for our figures for flexibility in modifying the output. Each `ggRandomForests` plot function returns either a single `ggplot` object, or a list of `ggplot` objects, allowing the use of additional `ggplot2` functions to modify and customize the final figures.

We have structured this chapter as a tutorial for using the `randomForestSRC` package for building and post-processing random survival forest models and using the `ggRandomForests` package for understanding how to construct the forest. We will build a random survival forest for the **primary biliary cirrhosis (PBC)** of the liver data set (Fleming & Harrington, 1991), available in the `randomForestSRC` package.

## 7.1 Random Forest Overview

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, which 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, the model makes predictions by either averaging of the prediction from individual trees or by majority vote in the case of **classification trees**

## 7.2 Medical longevity Study of primary biliary cirrhosis (PBC)

Data summary: primary biliary cirrhosis (PBC) data set

The **Primary Biliary Cirrhosis** of the liver (**PBC**) study consists of 424 PBC patients referred to Department of Veterans Affairs between 1979 and 1989 who met eligibility criteria for a randomized placebo-controlled trial of the drug *D-penicillamine* (DPCA). The data is described by Fleming and Harrington (1991) and they developed a **partial likelihood model** (Cox proportional hazards). The **pbc** data set, included in the **randomForestSRC** package, contains 418 observations, of which 312 patients participated in the randomized trial. (Fleming & Harrington, 1991)

The Military Health System identified PBC as an incurable disease that could be caused by exposure to various battlefield conditions, such as prolonged exposure to certain chemical compounds. The Department of Veterans Affairs is performing research to identify key factors that lead to the progression of the disease (via Ludwig's classification, PBC histological stage(s) 1 through 4 (NIH, 2017)), and those that lead to death. Though there is no clear cure for PBC, identifying these important factors can assist in determining how severe the disease is on a quantitative scale. This may help with research studies and understanding factors to prioritize in the development of PBC.

Primary Biliary Cirrhosis (PBC), or cholangitis, is a chronic disease of the liver. In instances where the disease is present, the bile ducts of an individual's liver are slowly destroyed. Bile is a fluid made in the liver that aids digestion and absorbs vitamins. It helps filter out cholesterol, toxins, and worn blood cells. (Mayo Clinic, 2023)

Once the bile ducts of a liver are destroyed, the irreversible scarring of liver tissue - cirrhosis, occurs, and the liver will eventually fail. This can

contribute to complications such as decreased mental functioning, vitamin deficiencies, increased blood pressure, liver failure, liver cancer, and death. (Mayo Clinic, 2023)

As of now, there is no cure for PBC. The medications and treatments involve slowing down the active liver damage, and the other comorbidities to PBC (NHS, 2024). In severe cases, liver transplant is necessary to keep a patient alive (NHS, 2024).

### 7.3 Modeling Preparation

We begin by loading R plotting and data mining packages. As with any problem we model with R (and R-Studio in this case) we start by loading the packages we think we may need.

```
library("knitr")           # markdown
library("ggplot2")          # Graphics engine
library("RColorBrewer")     # Nice color palettes
library("plot3D")           # for 3d surfaces.
library("dplyr")            # Better data manipulations
library("tidyverse")         # gather variables into long format
library("parallel")          # mclapply for multicore processing
```

Next, we load the random survival forest (RSF) `randomForestSRC` and `ggRandomForests` analysis packages.

```
library("randomForestSRC") # random forests for survival,
                           regression and classification
library("ggRandomForests") # ggplot2 random forest
                           figures (This!)
```

Now, we make some default settings that we will use for plotting purposes.

```
theme_set(theme_bw()) #ggplot2 theme with white background
                      Set open circle for censored, and x for events
event.marks <- c(1, 4)
event.labels <- c(FALSE, TRUE)

# We want red for death events, so reorder this set.
strCol <- brewer.pal(3, "Set1")[c(2,1,3)]
```

Next, we will load the `pbc` data and view the first six rows.

```
data("pbc", package = "randomForestSRC")
head(pbc)
```

	days	status	treatment	age	sex	ascites	hepatom	spiders	
1	400	TRUE		1 21464	1	1	1	1	1
2	4500	FALSE		1 20617	1	0	1	1	1
3	1012	TRUE		1 25594	0	0	0	0	0
4	1925	TRUE		1 19994	1	0	1	1	1
5	1504	FALSE		2 13918	1	0	1	1	1
6	2503	TRUE		2 24201	1	0	1	0	
	edema	bili	chol	albumin	copper	alk	sgot	trig	platelet
1	1.0	14.5	261	2.60	156	1718.0	137.95	172	190
2	0.0	1.1	302	4.14	54	7394.8	113.52	88	221
3	0.5	1.4	176	3.48	210	516.0	96.10	55	151
4	0.5	1.8	244	2.54	64	6121.8	60.63	92	183
5	0.0	3.4	279	3.53	143	671.0	113.15	72	136
6	0.0	0.8	248	3.98	50	944.0	93.00	63	NA
	prothrombin	stage							
1		12.2		4					
2		10.6		3					
3		12.0		4					
4		10.3		4					
5		10.9		3					
6		11.0		3					

## 7.4 Create a Data Dictionary

We obtained data 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 (Fleming & Harrington, 1991). The variables in the data set are shown in **Table 7-1**

*Table 7-1. Data dictionary for PBC data set*

Variable	Description	type
days	Time (days)	integer
status	Event (F = censor, T = death)	logical
treatment	Treatment: 1=D-penicil, 2=placebo	integer
age	Age: age in days	integer
sex	Sex: 0=male, 1=female	integer
ascites	Ascites: 0=no, 1 =yes	integer
hepatom	Hepatom: 0=no, 1=yes	integer
spiders	Spiders: 0=no, 1=yes	integer
edema	Edema(0, 0.5, 1)	numeric

Variable	Description	type
bili	Serum Bilirubin (mg/dl)	numeric
chol	Serum Cholesterol (mg/dl).	integer
albumin	Albumin (gm/dl)	numeric
copper	Urine Copper (ug/day)	integer
alk	Alkaline Phosphatase (U/liter)	numeric
sgot	SGOT (U/ml)	numeric
trig	Triglycerides(mg/dl)	integer
platelet	Platelets per cubic ml/1000	integer
prothrombin	Prothrombin time (sec)	numeric
stage	Histologic Stage	integer

Now we generate a data summary `pbc` data set.

```
data(pbc)
summary(pbc)
```

days			status			treatment		
Min. : 41	Min. :0.0000	Min. :1.000	1st Qu.:1093	1st Qu.:0.0000	1st Qu.:1.000	Median :1730	Median :0.0000	Median :1.000
Mean :1918	Mean :0.3852	Mean :1.494	3rd Qu.:2614	3rd Qu.:1.0000	3rd Qu.:2.000	Max. :4795	Max. :1.0000	Max. :2.000
NA's :106								
age			sex			ascites		
Min. : 9598	Min. :0.0000	Min. :0.00000	1st Qu.:15644	1st Qu.:1.0000	1st Qu.:0.00000	Median :18628	Median :1.0000	Median :0.00000
Mean :18533	Mean :0.8947	Mean :0.07692	3rd Qu.:21273	3rd Qu.:1.0000	3rd Qu.:0.00000	Max. :28650	Max. :1.0000	Max. :1.00000
NA's :106								
hepatom			spiders			edema		
Min. :0.0000	Min. :0.0000	Min. :0.0000	1st Qu.:0.0000	1st Qu.:0.0000	1st Qu.:0.0000	Median :1.0000	Median :0.0000	Median :0.0000
Mean :0.5128	Mean :0.2885	Mean :0.1005	3rd Qu.:1.0000	3rd Qu.:1.0000	3rd Qu.:0.0000	Max. :1.0000	Max. :1.0000	Max. :1.0000
NA's :106	NA's :106	NA's :106						
bili			chol			albumin		
Min. : 0.300	Min. : 120.0	Min. :1.960	1st Qu.: 0.800	1st Qu.: 249.5	1st Qu.:3.243	Median : 1.400	Median : 309.5	Median :3.530

```

Mean    : 3.221   Mean    : 369.5   Mean    :3.497
3rd Qu.: 3.400   3rd Qu.: 400.0   3rd Qu.:3.770
Max.    :28.000   Max.    :1775.0   Max.    :4.640
NA's    :134

copper      alk          sgot
Min.    : 4.00   Min.    : 289.0   Min.    : 26.35
1st Qu.: 41.25  1st Qu.: 871.5   1st Qu.: 80.60
Median  : 73.00  Median  : 1259.0  Median  :114.70
Mean    : 97.65  Mean    : 1982.7  Mean    :122.56
3rd Qu.:123.00  3rd Qu.: 1980.0  3rd Qu.:151.90
Max.    :588.00   Max.    :13862.4  Max.    :457.25
NA's    :108     NA's    :106     NA's    :106
trig       platelet    prothrombin
Min.    : 33.00  Min.    : 62.0    Min.    : 9.00
1st Qu.: 84.25  1st Qu.:188.5   1st Qu.:10.00
Median  :108.00  Median  :251.0    Median  :10.60
Mean    :124.70  Mean    :257.0    Mean    :10.73
3rd Qu.:151.00  3rd Qu.:318.0   3rd Qu.:11.10
Max.    :598.00   Max.    :721.0    Max.    :18.00
NA's    :136     NA's    :11      NA's    :2
stage
Min.    :1.000
1st Qu.:2.000
Median  :3.000
Mean    :3.024
3rd Qu.:4.000
Max.    :4.000
NA's    :6

```

Both EDA figures indicate the `pb`c data set contains quite a bit of missing data. **Table 7-2** shows the number of missing values in each variable of the `pb`c data set. Of the nineteen variables in the data, twelve have missing values. The `pb`c column details variables with missing data in the full `pb`c data set, though there are patients that were not randomized into the trial. If we restrict the data to the trial only, most of the missing values are also removed, leaving only four variables with missing values. Therefore, we will focus on the 312 observations from the clinical trial for the remainder of this document. We will discuss how `randomForestSRC` handles missing values in subsection 3.1.1.

**Table 7-2. Missing value counts in 'pb'c data set and PBC clinical trial observations ('pb'.trial').**

	pb	pb'.trial
treatment	106	0
ascites	106	0

hepatom	106	0
spiders	106	0
chol	134	28
copper	108	2
alk	106	0
sgot	106	0
trig	136	30
platelet	11	4
prothrombin	2	0
stage	6	0

For this analysis, we modified some of the data for better formatting of our results. Since the data contains about 12 years of follow up, we prefer using years instead of days to describe survival. We also convert the age variable to years, and the treatment variable to a factor containing levels of `c("DPCA", "placebo")`. We listed the variable names, type and description in *Table 7-1*.

## 7.5 Perform Variable Transformations

Here, we perform several variable transformations, including the conversion of "days" to "years" and replacing the treatment levels (default are 1 and 2) with "DPCA" and "placebo". We also refresh the data by removing the "days" column. Finally, we print and review the data summary.

```
# Convert age to years
pbcl <- pbc
pbcl$years <- pbcl$days/365
pbcl$age <- pbcl$age/365

# Fill empty data places
pbcl$Status <- NULL
pbcl$status

# Convert numbers to names for the treatment
# pbcl$treatment <- as.numeric(pbcl$treatment)
#pbcl$treatment[which(pbcl$treatment == 1)] <- "DPCA"
#pbcl$treatment[which(pbcl$treatment == 2)] <- "placebo"
#pbcl$treatment <- factor(pbcl$treatment)

# Define a subset to work with
pbcl <- pbcl[,2:20]
head(pbcl)
```

	status	treatment	age	sex	ascites	hepatom	spiders	
1	1		58.80548	1	1	1	1	1
2	0		56.48493	1	0	1	1	1
3	1		70.12055	0	0	0	0	0
4	1		54.77808	1	0	1	1	1
5	0		38.13151	1	0	1	1	1
6	1		66.30411	1	0	1	0	0
	edema	bili	chol	albumin	copper	alk	sgot	trig platelet
1	1.0	14.5	261	2.60	156	1718.0	137.95	172 190
2	0.0	1.1	302	4.14	54	7394.8	113.52	88 221
3	0.5	1.4	176	3.48	210	516.0	96.10	55 151
4	0.5	1.8	244	2.54	64	6121.8	60.63	92 183
5	0.0	3.4	279	3.53	143	671.0	113.15	72 136
6	0.0	0.8	248	3.98	50	944.0	93.00	63 NA
	prothrombin	stage	years					
1	12.2	4	1.095890					
2	10.6	3	12.328767					
3	12.0	4	2.772603					
4	10.3	4	5.273973					
5	10.9	3	4.120548					
6	11.0	3	6.857534					

### 7.5.1 Generate a Dataset with Integer Variables

Here, we remove integer-valued variables from the modified dataset, using the `gather` function. While there is a new function named `pivot_longer` that is supposed to be better, I have not had the opportunity to test it. We use `gather` here to remove integer-valued variables from our dataset.

```
# Use tidyverse::gather to transform the data into long format.
dta <- gather(pbc2, variable, value, -status, -edema, -
spiders, -sex, -ascites, -hepatom, -treatment, -stage, -
trig, -years)
#head(dta)
```

## 7.6 Exploratory Data Analysis

It is good practice to view your data before beginning analysis. Exploratory Data Analysis (EDA) (Tukey, 1977) will help you to understand the data, and find outliers, missing values and other data anomalies within each variable before getting deep into the analysis. To this end, we use `ggplot2` figures with the `facet_wrap` function to create several sets of panel plots, of bar charts (like **Figure 7-1** and

**Figure 7-3**) histograms for categorical variables (like **Figure 7-2**), and another of scatter plots for continuous variables (like **Figure 7-4** and **Figure 7-5**). Variables are plotted along a continuous variable on the X-axis to separate the individual observations.

In categorical EDA plots (like **Figure 7-1**), we are looking for patterns of missing data (white portion of bars). We often use surgical date for our *x*-axis variable to look for periods with more deaths. There is not a comparable variable available in the `pb` data set, so instead we used follow-up time (years). Another reasonable choice may have been to use the patient age variable for the *x*-axis. The important quality of the selected variable is to spread the observations out to aid in finding data anomalies.

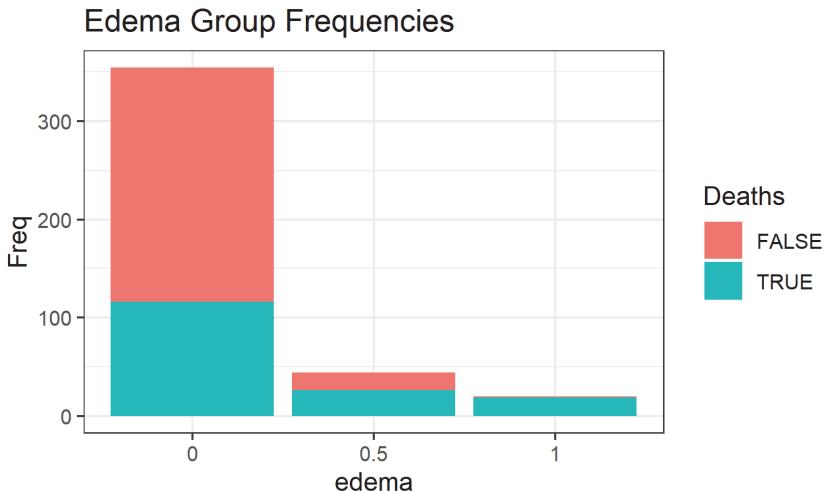
In continuous data EDA plots (**Figure 7-4**), we are looking for *missingness* (rug marks) and extreme or non-physical values. For survival settings, we color and shape the points as red circles to indicate death events, and blue circles to indicate censored observation.

### 7.6.1 Categorical Plots

We have several choices when it comes to plotting categorical variables for the purpose of discovery. We demonstrate three similar plots below. However, the similarities are in the shape of the distribution. First, we use a stacked bar plot in **Figure 7-1** as the frequency of occurrence of the three `edema` groups by status, i.e., deaths, either true or false.

```
edemacount <- as.data.frame(table(pbc2$edema, pbc2$years,
                                    pbc2$status))
colnames(edemacount)[1] <- "edema"
colnames(edemacount)[2] <- "years"
colnames(edemacount)[3] <- "status"

ggplot(edemacount, aes(x=edema, y=Freq, fill=status))+
  geom_bar(stat="Identity", position="stack")+
  labs(title="Edema Group Frequencies")+
  guides(fill=guide_legend(title="Deaths"))
```



*Figure 7-1. Bar plot of edema with three levels stratified by status (death or censor)*

The next plots in **Figure 7-2** are histograms, which are frequency distribution by definition. To create these plots over "years", we put the `years` into discrete groups by rounding them to an integer and used these as bin for the `edema` and `hepatom` data. First, we use the `tidyverse::gather` function to transform the data into long format. The `tidyverse::gather` function is read as `<package>::<function>` and is useful when more than one package has a function with the same name.

```
# Use tidyverse::gather to transform the data into long format.
tidyverse::gather(pbc2, variable, value, -status, -years)
# plot panels for each covariate colored by the logical
status variable.
gg1 <- ggplot(data.frame(dta$edema), aes(x = dta$years,
  fill = dta$status)) +
  geom_histogram(color=1) +
  labs(y = "", x = "Years", title = "Edema") +
  labs(fill = "Deaths?")

gg2 <- ggplot(data.frame(dta$hepatom), aes(x=dta$years,
fill=dta$status)) +
  geom_histogram(color = 1) +
  labs(y = "", x = "Years", title = "Hepatom")+
  labs(fill = "Deaths?")
```

Next, we use `grid.arrange` to show the plots in two rows.

```

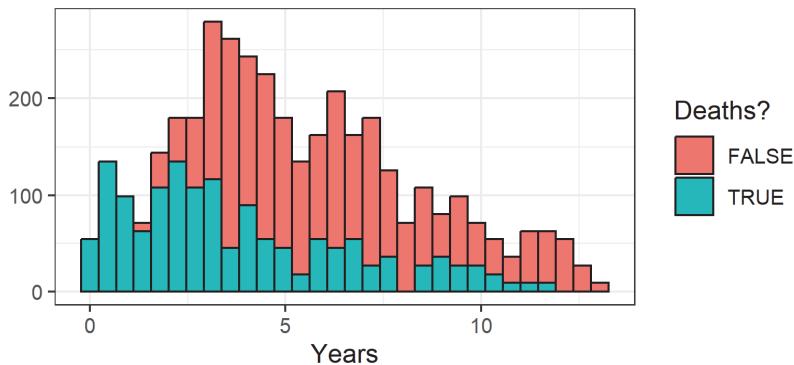
library(gridExtra)
pb2$years <- round(pb2$years,0)

edemacount <- as.data.frame(table(pb2$edema, pb2$years,
pb2$status))

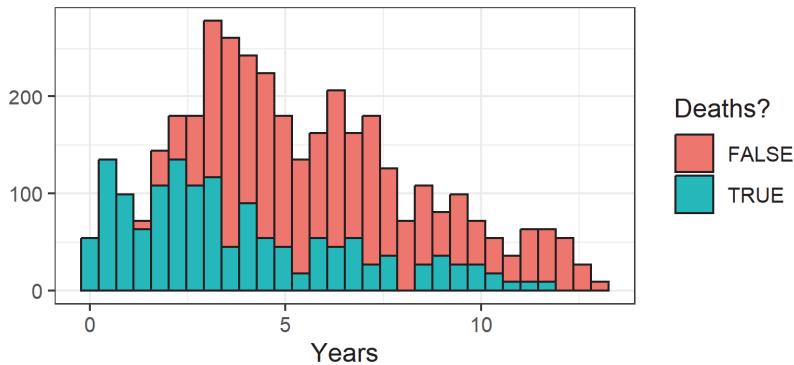
grid.arrange(gg1, gg2, nrow = 2)

```

Edema



Hepatom



*Figure 7-2. Histograms of edema and hepatoma over years and stratified by status*

The next set of plots in **Figure 7-3** are stacked, bar plots six discrete factors by their levels.

```

par(mfrow=c(3,2))

# Stacked Bar Plot with Colors and Legend
counts1 <- table(pbc$status,round(pbc1$years,0))
b1 <- barplot(counts1, main = "Status Distribution",
  xlab = "Years", col = c("dodgerblue","red"),
  legend = rownames(counts1))

counts2 <- table(pbc$edema,round(pbc1$years,0))
b2 <- barplot(counts2, main = "Edema Distribution",
  xlab = "Years", col = c("dodgerblue","red", "yellow"),
  legend = rownames(counts2))

counts3 <- table(pbc$hepatom,round(pbc1$years,0))
b3 <- barplot(counts3, main = "Hepatom Distribution",
  xlab = "Years", col = c("dodgerblue","red"),
  legend = rownames(counts3))

counts4 <- table(pbc$stage,round(pbc1$years,0))
b4 <- barplot(counts4, main="Stage Distribution",
  xlab = "Years", col =
c("dodgerblue","red","yellow","green"),
  legend = rownames(counts4))

counts5 <- table(pbc$sex,round(pbc1$years,0))
b5 <- barplot(counts5, main = "Sex Distribution",
  xlab = "Years", col = c("dodgerblue","red"),
  legend = rownames(counts5))

counts6 <- table(pbc$treatment,round(pbc1$years,0))
b6 <- barplot(counts6, main = "Treatment Distribution",
  xlab = "Years", col = c("dodgerblue","red"),
  legend = rownames(counts6))

```

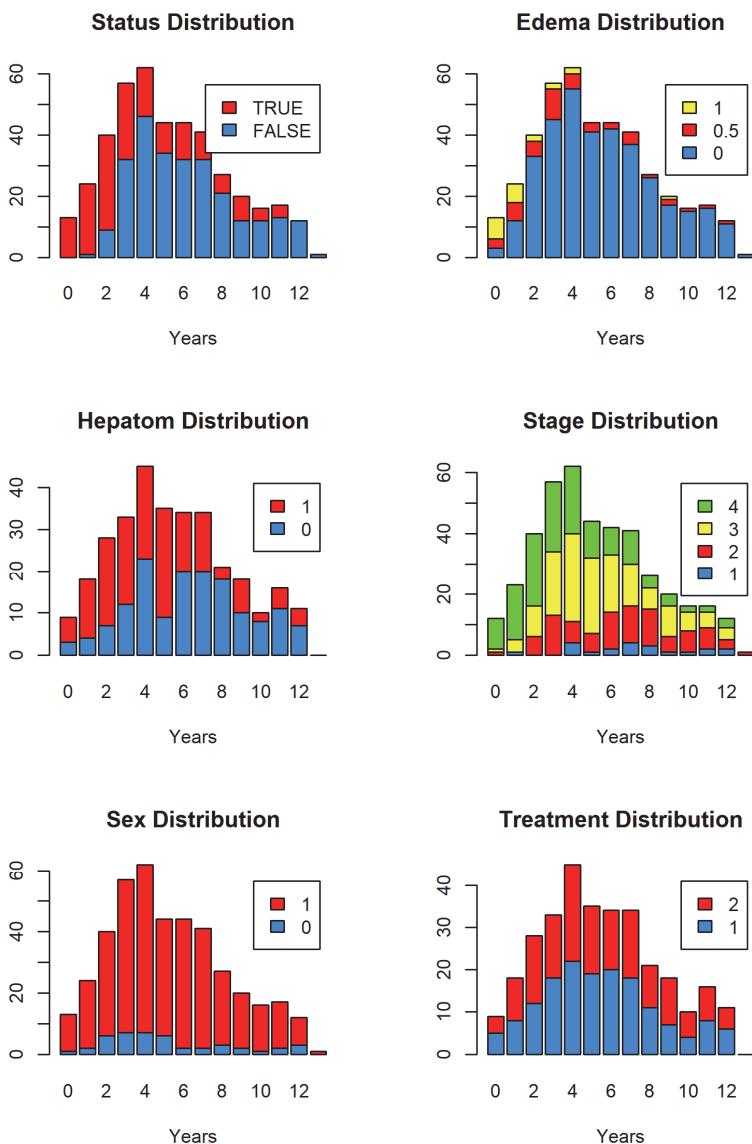


Figure 7-3. EDA plots for categorical variables (logicals and factors).

## 7.6.2 Plotting Continuous Variables

Next, we use scatter plots to observe the continuous variables of `pbc`. To construct these plots, we again use the `tidy::gather` function to

transform the data into long format. We perform this is manipulation of data in preparation of the ggplot plotting functions.

```
# Use tidyverse::gather to transform the data into long format.  
pbc2 <- pbc1[,2:20]  
dta <- gather(pbc2, variable, value, -status, -edema, -  
spiders, -sex, -ascites, -hepatom, -treatment, -stage, -  
trig, -years)  
head(dta)
```

	status	treatment	sex	ascites	hepatom	spiders	edema	trig	
1	TRUE		1	1	1	1	1	1.0	172
2	FALSE		1	1	0	1	1	0.0	88
3	TRUE		1	0	0	0	0	0.5	55
4	TRUE		1	1	0	1	1	0.5	92
5	FALSE		2	1	0	1	1	0.0	72
6	TRUE		2	1	0	1	0	0.0	63
	stage	years	variable	value					
1	4	1.095890	age	58.80548					
2	3	12.328767	age	56.48493					
3	4	2.772603	age	70.12055					
4	4	5.273973	age	54.77808					
5	3	4.120548	age	38.13151					
6	3	6.857534	age	66.30411					

### 7.6.3 Covariate Panel Plot

In continuous data EDA plots (see **Figure 7-4**), we are looking for missingness (rug marks) and extreme or non-physical values. For survival settings, we color and shape the points as red x's to indicate events, and blue circles to indicate censored observation.

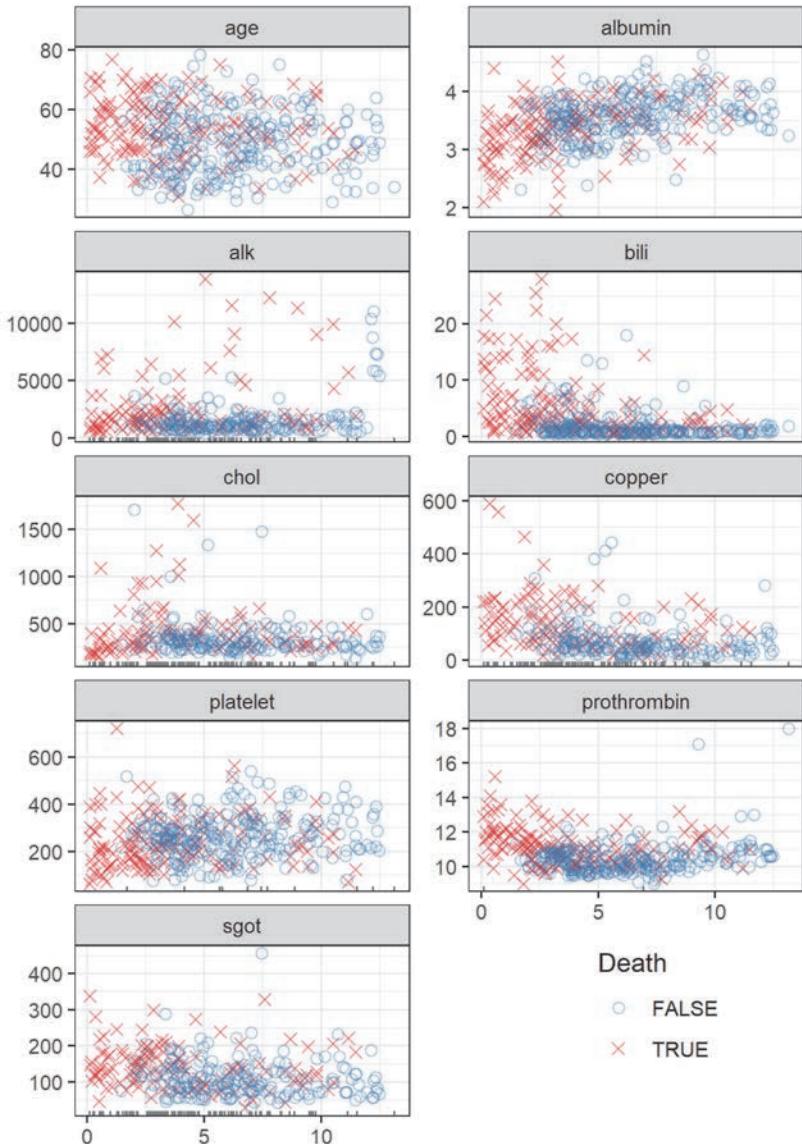
Extreme value examples are evident in a few of the variables in. We are typically looking for values that are outside of the biological range. This is often caused by measurements recorded in differing units, which can sometimes be corrected algorithmically. Since we cannot ask the original investigator to clarify these values in this particular study, we will continue without modifying the data.

```
ggplot(dta %>% filter(!is.na(value)), aes(x = years, y =  
value, color = status, shape = status)) +  
  geom_point(alpha = 0.4, size=2) +  
  geom_rug(data = dta[which(is.na(dta$value)),], color =  
"grey50") +  
  labs(y = "", x = st.labs["years"], color = "Death", shape  
= "Death") +
```

```

scale_color_manual(values = strCol) +
scale_shape_manual(values = event.marks) +
facet_wrap(~variable, scales = "free_y", ncol = 2) +
theme(legend.position = c(0.8, 0.1))

```

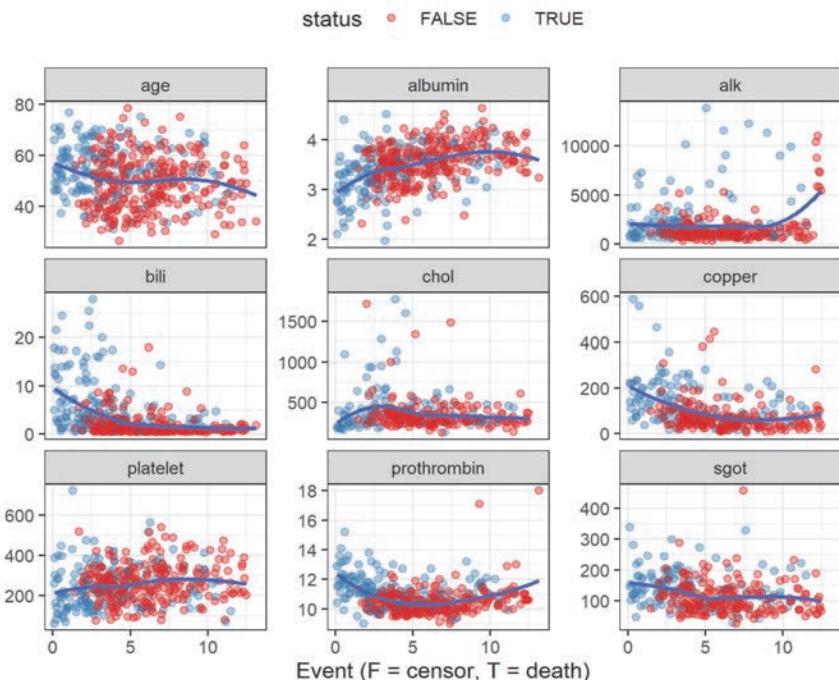


**Figure 7-4.** EDA plots for continuous variables. Symbols indicate observations with variable value on y-axis against follow up time in years. Symbols are colored and shaped according to the death event. Missing values are indicated by rug marks along the x-axis.

## 7.6.4 Fitted Scatter Plots

The next set of scatter plots in **Figure 7-5** uses the combination of points (`geom_point`) and fitted curves (`geom_smooth`).

```
ggplot(dta)+  
  geom_point(alpha=0.4, aes(x = years, y = value,  
    color = status)) +  
  geom_smooth(aes(x = years, y = value), se=FALSE) +  
  labs(y = "", x = st.labs["status"]) +  
  scale_color_brewer(palette="Set1") +  
  facet_wrap(~variable, scales = "free_y", ncol = 3) +  
  theme(legend.position = "top")
```



**Figure 7-5.** Scatter plots of continuous pbc variables by status over time (years).

This figure is loosely related to a pairs scatter plot (Becker, Chambers, & Wilks, 1988), but in this case we only examine the relation between the response variable against the remainder. Plotting the data against the response also gives us a “sanity check” when viewing our model results. It is pretty obvious from this figure that we should find a strong, non-linear relation between variable values and the patient status over years.

## 7.6.5 Ph model from Fleming and Harrington 1991

As a final part of our EDA, we reprint a table from the Fleming and Harrington study using knitr's `kable` function for generating a simple table.

```
fleming.table <- data.frame(matrix(ncol = 3, nrow = 5))
rownames(fleming.table) <- c("Age", "log(Albumin)",
"log(Bilirubin)", "Edema", "log(Prothrombin Time)")
colnames(fleming.table) <- c("Coef.", "Std. Err.", "Z
stat.")
fleming.table[,1] <- c(0.0333, -3.0553, 0.8792, 0.7847,
3.0157)
fleming.table[,2] <- c(0.00866,
0.72408, 0.09873, 0.29913, 1.02380)
fleming.table[,3] <- c(3.84, -4.22, 8.9, 2.62, 2.95)

kable(fleming.table, format="simple", digits = 3, caption =
"PBC proportional hazards model summary of 312 randomized
cases in pbc.trial data set. (Table 4.4.3c)")
```

*Table 7-3. PBC proportional hazards model summary of 312 randomized cases in pbc.trial data set. (Table 4.4.3c)*

	Coef.	Std. Err.	Z stat.
Age	0.033	0.009	3.84
log(Albumin)	-3.055	0.724	-4.22
log(Bilirubin)	0.879	0.099	8.90
Edema	0.785	0.299	2.62
log(Prothrombin Time)	3.016	1.024	2.95

## 7.7 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**.

```
pbc2$status <- as.numeric(pbc1$status)
pbc.trial <- pbc2[-which(is.na(pbc2$treatment)),]
pbc.test <- pbc2[which(is.na(pbc2$treatment)),]
pbc.trial
```

```

# Load the data, from the call:
rfsrc_pbc <- rfsrc(Surv(years, status) ~.,
  data = pbc.trial,
  nsplit = 10, na.action = "na.impute",
  tree.err = TRUE, importance = TRUE)

# print the forest summary
rfsrc_pbc

```

```

Sample size: 312
Number of deaths: 125
Was data imputed: yes
Number of trees: 500
Forest terminal node size: 15
Average no. of terminal nodes: 15.28
No. of variables tried at each split: 5
Total no. of variables: 17
Resampling used to grow trees: swor
Resample size used to grow trees: 197
Analysis: RSF
Family: surv
Splitting rule: logrank *random*
Number of random split points: 10
(00B) CRPS: 0.12224327
(00B) Requested performance error: 0.17066

```

The `randomForestSRC::print.rfsrc` summary details the parameters used for the `rfsrc` call described above and returns variance and generalization error estimate from the forest training set. The forest is built from 418 observations and eighteen independent variables. It was constructed for the continuous `status` variable using `ntree=1000` regression (`regr`) trees, randomly selecting five candidate variables at each node split, and terminating nodes with at least five observations.

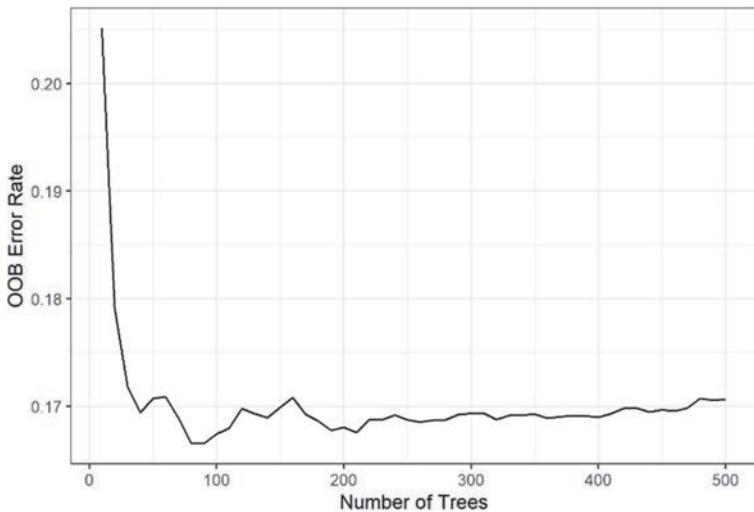
## 7.8 Generalization error estimates

One advantage of Random Forests is a built-in generalization error estimate. Each bootstrap sample selects approximately 63.2% of the population on average. The remaining 36.8% of observations, the **Out-of-Bag (OOB)** (Breiman, Bagging Predictors, 1996) sample, can be used as a holdout test set for each of the trees in the forest. We can calculate an OOB prediction error estimate for each observation by predicting the response over the set of trees, which were NOT trained with that

particular observation. The Out-of-Bag prediction error estimates have been shown to be nearly identical to  $n$ -fold cross validation estimates (James, Witten, Hastie, & Tibshirani, 2013). This feature of Random Forests allows us to obtain both model fit and validation in one pass of the algorithm.

The `gg_error` function operates on the `randomForestSRC::rfsrc` object to extract the error estimates as the forest is grown. The code block demonstrates part the `ggRandomForests` design philosophy, to create separate data objects and provide functions to operate on the data objects. For instance, the following code block first creates a `gg_error` object, then uses the `plot.gg_error` function to create a `ggplot` object for display in **Figure 7-6**.

```
# Plot the OOB errors against the growth of the forest.
gg_e <- gg_error(rfsrc_pbc)
gg_e <- gg_e %>% filter(!is.na(error))
class(gg_e) <- c("gg_error",class(gg_e))
plot(gg_e)
```



**Figure 7-6.** Shows a small number of trees needed for error estimates

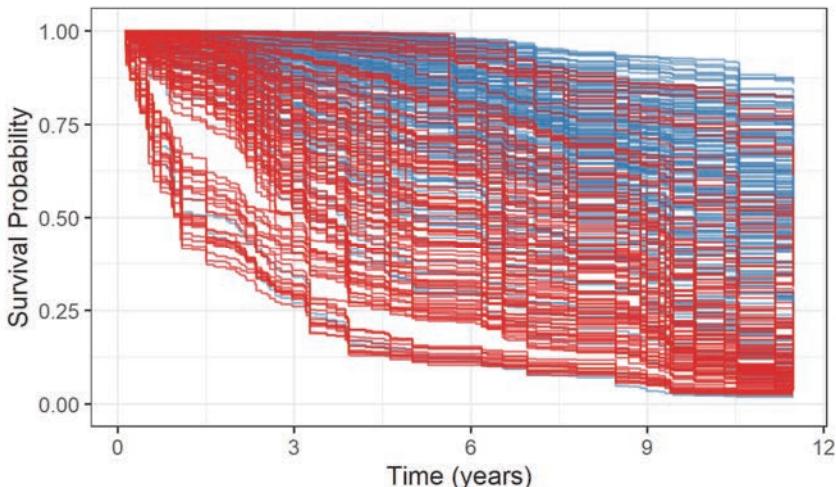
**Figure 7-6** demonstrates that it does not take a large number of trees to stabilize the forest prediction error estimate. However, to ensure that each variable has enough of a chance to be included in the forest prediction process, we do want to create a large random forest of trees.

## 7.9 Random Forest Prediction

### 7.9.1 Training Set Prediction

The `gg_rfsrc` function extracts the OOB prediction estimates from the random forest. This code block executes the data extraction and plotting in one line, since we are not interested in holding the prediction estimates for later reuse. Also note that we add in the additional `ggplot2` command (`coord_cartesian`) to modify the plot object (see *Figure 7-7*). Each of the `ggRandomForests` plot commands return `ggplot` objects, which we can also store for modification or reuse later in the analysis.

```
ggRFsrc <- plot(gg_rfsrc(rfsrc_pbc), alpha = 0.6) +  
  scale_color_manual(values = strCol) +  
  theme(legend.position = "none") +  
  labs(y = "Survival Probability", x = "Time (years)") +  
  coord_cartesian(ylim = c(-0.01, 1.01))  
show(ggRFsrc)
```



*Figure 7-7. Random forest OOB predicted survival. Blue curves correspond to censored observations, red curves correspond to observations experiencing death events.*

The `gg_rfsrc` plot shows the predicted survival from our RSF model. Each line represents a single patient in the training data set, where censored patients are colored blue, and patients who have experienced the event (death) are colored in red. We extend all predicted survival

curves to the longest follow up time (12 years), regardless of the actual length of a patient's follow up time.

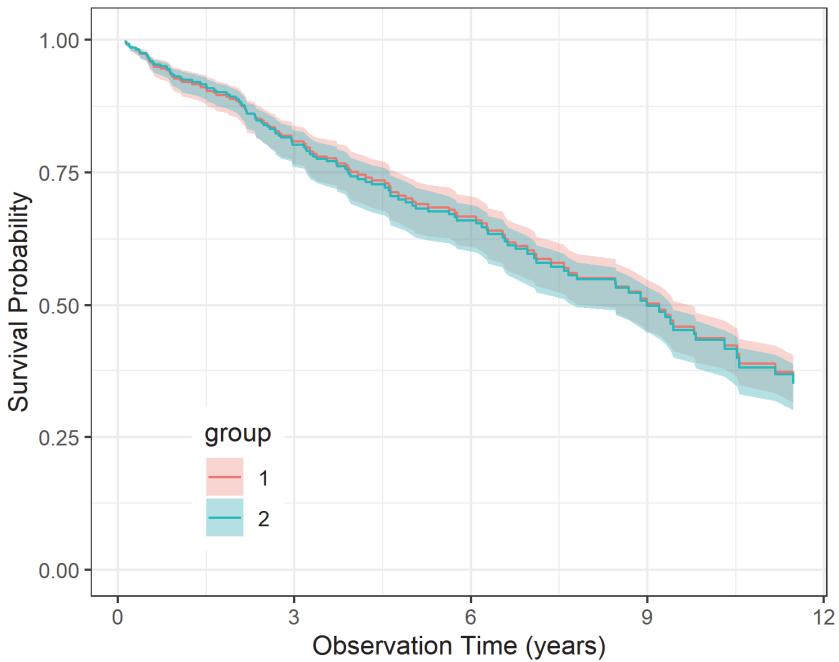
Interpretation of general survival properties from *Error! Reference source not found.* is difficult because of the number of curves displayed. To get more interpretable results, it is preferable to plot a summary of the survival results. The following code block compares the predicted survival between treatment groups, as we did in .

## 7.9.2 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.

```
gg_dta <- gg_survival(interval = "years",
                      censor = "status",
                      by = "treatment",
                      data = pbc.trial,
                      conf.int = .95)
plot(gg_rfsrc(rfsrc_pbc, by = "treatment")) +
  theme(legend.position = c(0.2, 0.2)) +
  labs(y = "Survival Probability", x = "Observation Time
(years)") +
  coord_cartesian(ylim = c(0, 1.01))
```

The `gg_rfsrc` plot shows the predicted status, one point for each observation in the training set. The points are jittered around a single point on the  $x$ -axis since we are only looking at predicted values from the forest. These estimates are Out of Bag, which are analogous to test set estimates. We show the boxplot to give an indication of the distribution of the prediction estimates. For this analysis, **Figure 7-8** is another model sanity check, as we are more interested in exploring the why questions for these predictions.



**Figure 7-8.** Random forest predicted survival stratified by treatment groups.  
DPCA group in red, placebo in blue with shaded 95% confidence bands.

## 7.10 Plot the cumulative hazard function

Now, we plot the cumulative hazard function for the treatment and placebo in **Figure 7-9**. We define the **hazard function** (also known as failure rate or hazard rate function) as the rate of failure of a component or system, given that the failure has not occurred prior to time  $t$ .

```
plot(gg_dta, type="cum_haz") +
  labs(y = "Cumulative Hazard",
       x = "Observation Time (Years)",
       color = "Treatment", fill = "Treatment") +
  theme(legend.position = c(.2,.8))
```

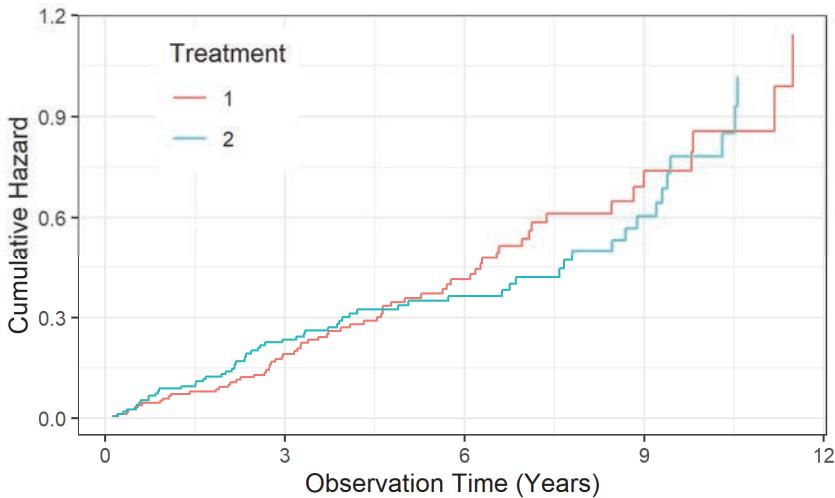


Figure 7-9. Cumulative hazard function for each treatment

### 7.10.1 Random forest imputation

There are two modeling issues when dealing with missing data values:

1. How does the algorithm build a model when values are missing from the training data?
2. How does the algorithm predict a response when values are missing from the test data?.

The standard procedure for linear models is to either remove or impute the missing data values before modeling. We remove the **missingness** by either removing the variable with missing values (column wise) or removing the observations (row wise). Removal is a simple solution but may bias results when either observations or variables are scarce.

The `randomForestSRC` package imputes missing values using **adaptive tree imputation** (Ishwaran H. , Kogalur, Chen, & J., 2011). Rather than impute missing values before growing the forest, the algorithm takes a *just-in-time* approach. At each node split, the algorithm checks the set of `mtry` candidate variables for missing values. Then the algorithm imputes missing values by randomly drawing values from non-missing data within the node. It then calculates the split-statistic on observations that were not missing values. It uses the imputed values to sort observations into the subsequent daughter nodes and then discarded before the next split occurs. The algorithm repeats the process until the

stopping criteria is reached and all observations are sorted into terminal nodes.

A final imputation step can be used to fill in missing values from within the terminal nodes. This step uses a process similar to the previous imputation but uses the OOB non-missing terminal node data for the random draws. The algorithm aggregates these values (averaging for continuous variables, voting for categorical variables) over the `ntree` trees in the forest to estimate an imputed data set. By default, it does not fill the missing values into the training data, but they are available within the forest object for later use if desired.

Adaptive tree imputation still requires the missing at random assumptions (Rubin, 1976). At each imputation step, the random forest assumes that similar observations are grouped together within each node. The random draws used to fill in missing data do not bias the split rule, but only sort observations similar in non-missing data into like nodes. An additional feature of this approach is the ability to predict on test set observations with missing values.

### 7.10.2 Test set predictions

The strength of adaptive tree imputation becomes clear when doing prediction on test set observations. If we want to predict survival for patients that did not participate in the trial using the model we created in Section , we need to somehow account for the missing values detailed in .

The `predict.rfsrc` call takes the forest object (`rfsrc_pbc`), and the test data set (`pbc_test`) and returns a predicted survival using the same forest imputation method for missing values within the test data set (`na.action="na.impute"`). Now, we predict the survival for 106 patients not in randomized trial:

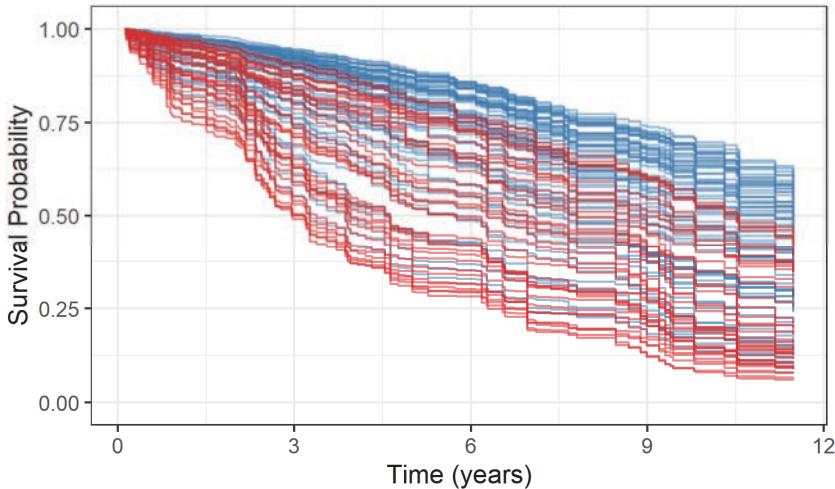
```
rfsrc_pbc_test <- predict(rfsrc_pbc, newdata = pbc.test,  
                           na.action = "na.impute",  
                           importance = TRUE)
```

The forest summary indicates there are 106 test set observations with thirty-six deaths and the predicted error rate is 19.1. In **Figure 7-10**, we plot the predicted survival just as we did the training set estimates.

```

plot(gg_rfsrc(rfsrc_pbc_test), alpha=.5) +
  scale_color_manual(values = strCol) +
  theme(legend.position = "none") +
  labs(y = "Survival Probability", x = "Time (years)") +
  coord_cartesian(ylim = c(-0.01, 1.01))

```



**Figure 7-10.** Random forest survival estimates for patients in the pbc.test data set. Blue curves correspond to censored patients, red curves correspond to patients experiencing a death event.

The `gg_rfsrc` plot of **Figure 7-10** shows the test set predictions, similar to the training set predictions in *Error! Reference source not found.*, though with fewer patients the survival curves do not cover the same area of the figure. It is important to note that because **Figure 7-10** is constructed with OOB estimates, the survival results are comparable as estimates from unseen observations in *Error! Reference source not found.*.

## 7.11 Variable selection

Random forest is not a parsimonious method but uses all variables available in the data set to construct the response predictor. Also, unlike parametric models, random forest does not require the explicit specification of the functional form of *covariates* to the response. Therefore there is no explicit *p*-value/significance test for variable selection with a random forest model. Instead, RF ascertains which variables contribute to the prediction through the split rule optimization, optimally choosing variables which separate observations.

The typical goal of a random forest analysis is to build a prediction model, in contrast to extracting information regarding the underlying process (Breiman, Statistical Modeling: The Two Cultures, 2001b). We usually do not care about how many variables we include in the training data set. Since the goal is prediction, investigators often include the “kitchen sink” if it can help.

In contrast, in survival settings we are typically also interested in how we can improve the outcome of interest. To achieve this, for understandable inference, it is important to avoid both duplication and transformations of variables whenever possible when building our data sets. Duplication of variables, including multiple measures of a similar covariate, can reduce or mask the importance of the covariate. Transformations can also mask importance as well as make interpretation of the inference results difficult to impossible.

In this Section, We explore two separate approaches to investigate the RF variable selection process. Variable Importance, a property related to **variable misspecification**, and **Minimal Depth**, a property derived from the construction of the trees within the forest.

### 7.11.1 Variable Importance

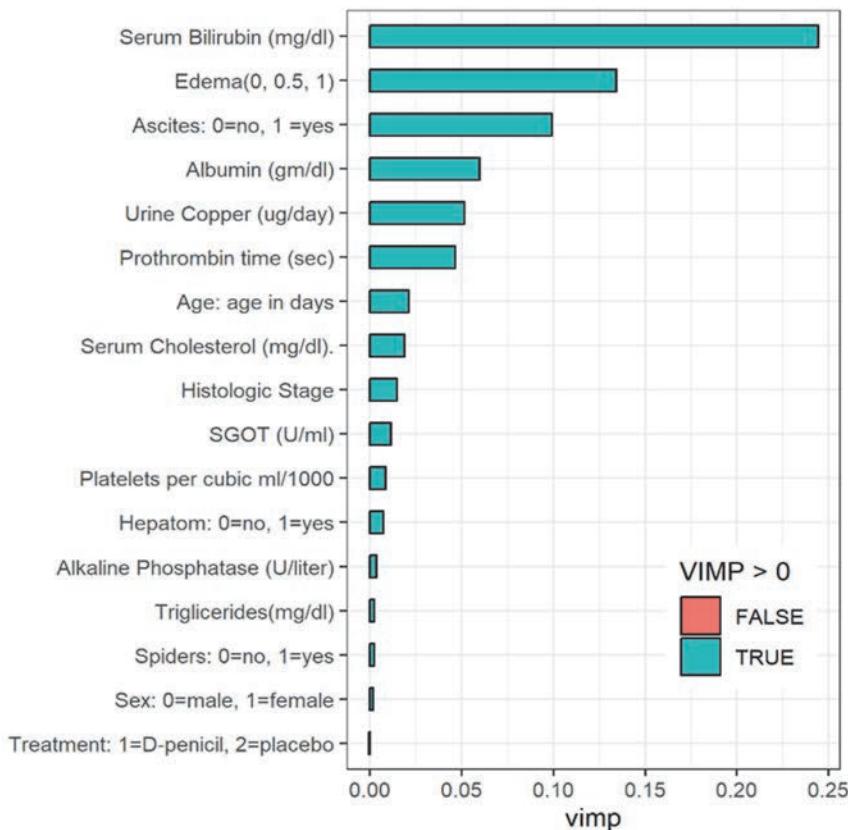
Variable importance (VIMP) was originally defined in CART using a measure involving surrogate variables (see (Breiman, Friedman, Olshen, & Stone, 1984) (Krzywinski & Altman, 2017)). The most popular VIMP method uses a prediction error approach involving *noising-up* each variable in turn. VIMP for a variable  $x_v$  is the difference between prediction error when  $x_v$  is randomly permuted, compared to prediction error under the observed values ( (Breiman, Random Forests, 2001a); (Liaw & Wiener, 2002); (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010) (Ishwaran H. , Kogalur, Chen, & J., 2011)).

Since VIMP is the difference in OOB prediction error before and after permutation, a large VIMP value indicates that misspecification detracts from the predictive accuracy in the forest. VIMP close to zero indicates the variable contributes nothing to predictive accuracy, and negative values indicate the predictive accuracy improves when the variable is misspecified. 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 shows the variables, in VIMP rank order, labeled with the named vector in the `lbls` argument as shown in **Figure 7-11**.

```
plot(gg_vimp(rfsrc_pbc), lbls=st.labs) +  
  theme(legend.position = c(0.8, 0.2)) +  
  labs(fill = "VIMP > 0")
```



**Figure 7-11. Random forest Variable Importance (VIMP).** Blue bars indicates positive VIMP, red indicates negative VIMP. Importance is relative to positive length of bars.

The `gg_vimp` plot of **Figure 7-11** details VIMP ranking for the `pbc.trial` baseline variables, from the largest at the top, to smallest at the bottom.

We use bars to show VIMP measures to compare the scale of the error increase under permutation. Colored bars (red for negative values) show the sign of the measure. Note that four of the five highest ranking variables by VIMP match those selected by the (Fleming & Harrington, 1991) model listed in (Fleming & Harrington, 1991), with urine copper (2) ranking higher than age (8). We will return to this in variable-selection-comparison.

## 7.12 Minimal Depth

In VIMP, prognostic risk factors are determined by evaluating the forest prediction under alternative data settings, ranking the most important variables according to their impact on predictive ability of the forest. An alternative 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, there are numbered node levels 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. The assumption in the metric is that smaller minimal depth values indicate the variable separates large groups of observations, and therefore has an 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 difference in VIMP measures. Given minimal depth is a quantitative property of the forest construction, (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010) we 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` package's `var.select` function uses the minimal depth methodology for variable selection, returning an object with both minimal depth and `vimp` measures. The `gg_minimal_depth` function from the `ggRandomForests` package is analogous to the

`gg_vimp` function. The algorithm ranks variables from most important at the top (minimal depth measure), to least at the bottom (maximal minimal depth).

```
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 : 312  
ntree : 500  
nsplit : 10  
mtry : 5  
nodesize : 15  
refitted forest : FALSE  
model size : 6  
depth threshold : 5.323  
PE (true OOB) : 17.066
```

Top variables:

```
-----  
          depth    vimp  
bili      2.108  0.244  
albumin   3.960  0.060  
copper     4.068  0.052  
prothrombin 4.412  0.047  
edema     4.608  0.134  
ascites    4.882  0.099  
-----
```

Here is another method for calculating minimum depth.

```
gg_md <- gg_minimal_depth(varsel_pbc, lbls = st.labs)  
print(gg_md, lbls=st.labs)
```

```
-----  
gg_minimal_depth  
model size      : 6  
depth threshold : 5.323  
PE : [1] 17.066  
-----
```

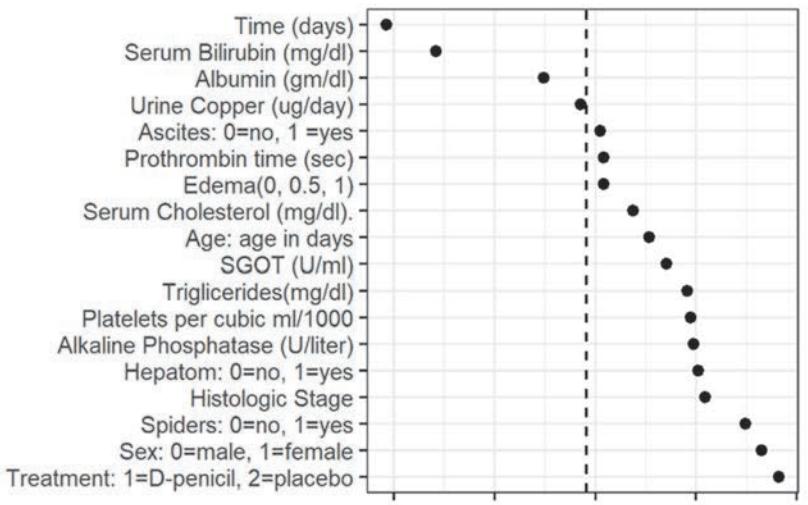
Top variables:

	depth	vimp
	<dbl>	<dbl>
bili	2.108	0.24422523
albumin	3.960	0.05982341
copper	4.068	0.05164650
prothrombin	4.412	0.04663953
edema	4.608	0.13431571
ascites	4.882	0.09915825

6 rows

The `gg_minimal_depth` summary mostly reproduces the output from the `var.select` function from the `randomForestSRC` package. We report the **minimal depth threshold** (threshold 5.323) and the number of variables with depth below that threshold (model size 6). We also list a table of the top (6) selected variables, in minimal depth rank order with the associated VIMP measures. The minimal depth numbers indicate that `bili` tends to split between the first and second node level, and the next three variables (albumin, copper, prothrombin) split between the second and third levels on average.

```
plot(gg_md, lbls = st.labs)
```



*Figure 7-12. Minimal Depth variable selection. Low minimal depth indicates important variables. The dashed line is the threshold of maximum value for variable selection.*

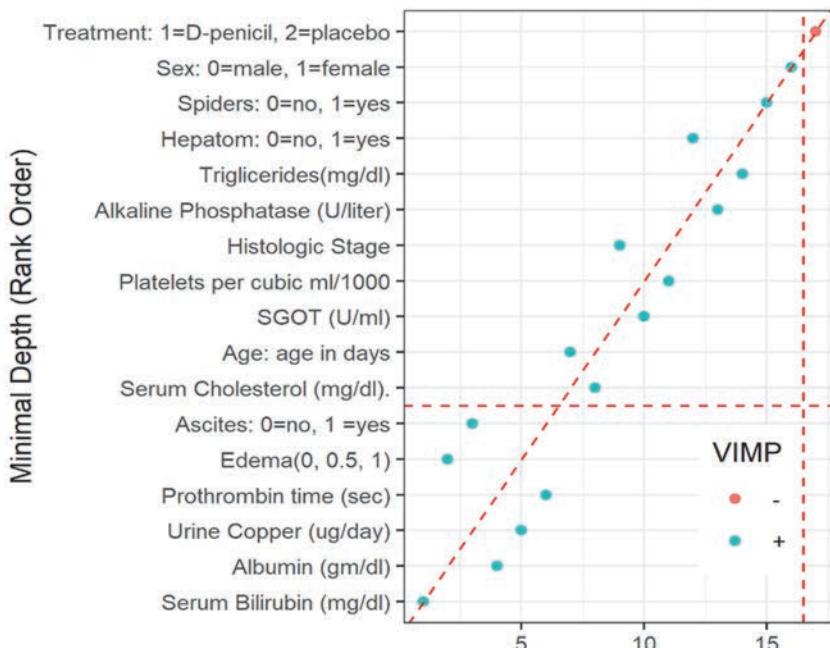
Coordinate system already present. Adding new coordinate system, which will replace the existing one.

The `gg_minimal_depth` plot of **Figure 7-12** is similar to the `gg_vimp` plot in **Figure 7-11**, ranking variables from most important at the top (minimal depth measure), to least at the bottom (maximal minimal depth). The *vertical dashed line* indicates the minimal depth threshold where smaller minimal depth values indicate higher importance and larger values indicate lower importance.

### 7.12.1 Variable selection comparison

Since the VIMP and Minimal Depth measures use different criteria, we expect the variable ranking to be somewhat different. We use `gg_minimal_vimp` function to compare rankings between minimal depth and VIMP in **Figure 7-13**.

```
plot(gg_minimal_vimp(gg_md), lbls = st.labs) +  
  theme(legend.position=c(0.8, 0.2))
```



**Figure 7-13. Comparing Minimal Depth and VIMP rankings. Points on the red dashed line are ranked equivalently, points above have higher VIMP ranking, those below have higher minimal depth ranking.**

The points along the red dashed line indicate where the measures are in agreement. 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.

```
fleming.table <- data.frame(matrix(ncol = 3, nrow = 5))
rownames(fleming.table) <- c("Age", "log(Albumin)",
"log(Bilirubin)", "Edema", "log(Prothrombin Time)")
colnames(fleming.table) <- c("Coef.", "Std. Err.", "Z
stat.")

fleming.table$nm <- c("age", "albumin", "bili", "edema",
"prothrombin")
fh.model <- data.frame(cbind(names = fleming.table$nm,
FH = order(abs(fleming.table$`Z stat.`)),
decreasing = TRUE),
Variable=rownames(fleming.table),
Coeff=fleming.table$Coef.
))

kable(fh.model,
      format="simple",
      caption = "Comparison of variable selection criteria.
Minimal depth ranking, VIMP ranking and (fleming,1991) (FH)
proportional hazards model ranked according to `abs(Z
stat)`",
      align=c("l", "r", "r", "r"),
      digits = 3,
      row.names = FALSE)
```

*Table 7-4. Comparison of variable selection criteria. Minimal depth ranking, VIMP ranking and (fleming,1991) (FH) proportional hazards model ranked according to abs(Z stat)*

names	FH	Variable	Coeff
age	1	Age	NA
albumin	2	log(Albumin)	NA
bili	3	log(Bilirubin)	NA
edema	4	Edema	NA
prothrombin	5	log(Prothrombin Time)	NA

We examine the ranking of the different variable selection methods further in . We can use the Z statistic from to rank variables selected in the (Fleming & Harrington, 1991) model to compare with variables selected by minimal depth and VIMP. The table is constructed by taking the top ranked minimal depth variables (below the selection threshold) and matching the VIMP ranking and model transforms. We see all three methods indicate a strong relation of serum bilirubin to survival, and overall, the minimal depth and VIMP rankings agree reasonably well with the (Fleming & Harrington, 1991) model.

The minimal depth selection process reduced the number of variables of interest from 17 to 6, which is still a rather large subset of interest. An obvious selection set is to examine the five variables selected by (Fleming & Harrington, 1991). Combining the Minimal Depth and model, there may be evidence to keep the top seven variables. Though minimal depth does not indicate the edema variable is very interesting, VIMP ranking does agree with the proportional hazards model, indicating we might not want to remove the edema variable. Both minimal depth and VIMP suggest including [copper](#), a measure associated with liver disease.

Regarding the [chol](#) variable, recall missing data summary of . In the trial data set, there were twenty-eight observations missing [chol](#) values. The forest imputation randomly sorts observations with missing values into daughter nodes when using the [chol](#) variable, which is also how calculates VIMP. We therefore expect low values for VIMP when a variable has a reasonable number of missing values.

Restricting our remaining analysis to the five variables (Fleming & Harrington, 1991), plus the copper retains the biological sense of these analysis. We will now examine how these six variables are related to survival using variable dependence methods to determine the direction of the effect and verify that the log transforms used by (Fleming & Harrington, 1991) are appropriate.

## 7.13 Response/Variable Dependence

As random forest is not parsimonious, we have used minimal depth and VIMP to reduce the number of variables to a manageable subset. Once we have an idea of which variables contribute most to the predictive accuracy of the forest, we would like to know how the response depends on these variables.

Although often characterized as a method, the forest predictor is a function of the predictor variables  $\hat{f}_{RF} = f(x)$ . We use graphical methods to examine the forest predicted response dependency on covariates. We again have two options, variable dependence plots are quick and easy to generate, and partial dependence plots are more computationally intensive but give us a risk adjusted look at variable dependence.

### 7.13.1 Variable dependence

The plots show the predicted response relative to a covariate of interest, with each training set observation represented by a point on the plot. Interpretation of variable dependence plots can only be in general terms, as point predictions are a function of all covariates in that particular observation.

Variable dependence is straight forward to calculate, involving only the getting the predicted response for each observation. In survival settings, we must account for the additional dimension of time. We plot the response at specific time points of interest, for example survival at one or three years.

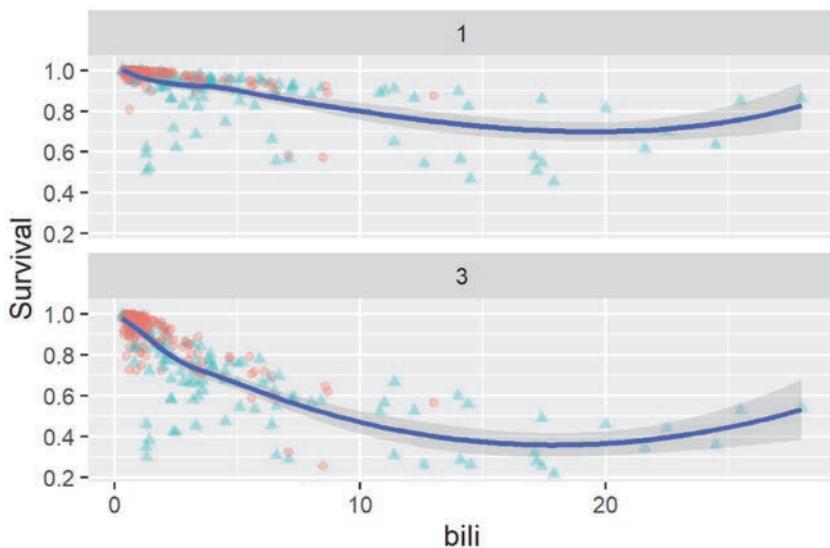
The `gg_rfsrc` is identical to (stored in the `ggRFSrc` variable) with the addition of a vertical dashed line at the 1- and 3- year survival time. A variable dependence plot is generated from the predicted response value of each survival curve at the intersecting time line plotted against covariate value for that observation.

The `gg_variable` function extracts the training set variables and the predicted OOB response from `rfsrc` and `predict` objects. In the following code block, we store the `gg_variable` data object for later use (`gg_v`), as all remaining variable dependence plots can be constructed from this object.

```
gg_v <- gg_variable(rfsrc_pbc, time = c(1, 3),
                     time.labels = c("1 Year", "3 Years"))
plot(gg_v, xvar = "bili", alpha = 0.4) +
  labs(y = "Survival", x = st.labs["bili"]) +
  theme(legend.position = "none") +
  scale_color_manual(values = strCol, labels=vent.labels) +
  scale_shape_manual(values = event.marks, labels =
    event.labels) +
  coord_cartesian(ylim = c(-0.01, 1.01))
```

The `gg_variable` plot of **Figure 7-14** shows variable dependence for the Serum Bilirubin (`bili`) variable. Again censored cases are shown as blue circles, events are indicated by the red `x` symbols. Each predicted point is dependent on the full combination of all other covariates, not only on the covariate displayed in the dependence plot. The smooth loess line (Cleveland W., 1981) indicates the trend of the prediction over the change in the variable.

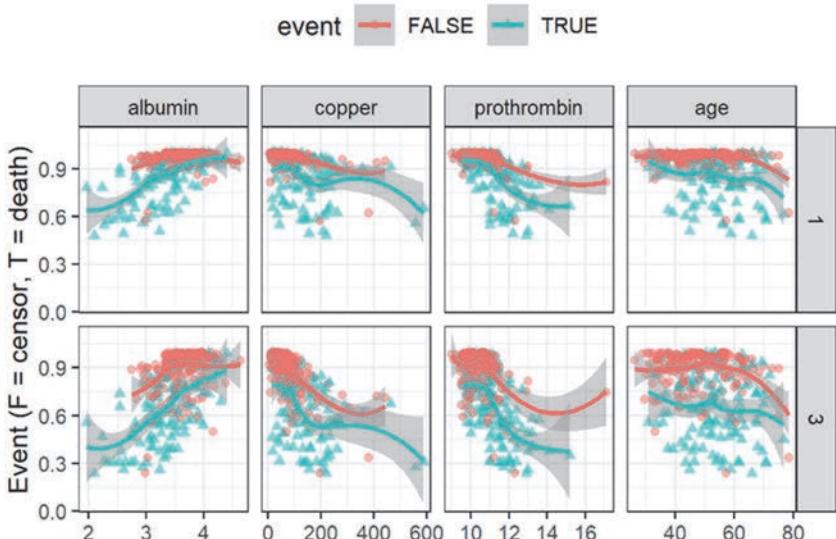
Examination of **Figure 7-14** indicates most of the cases are grouped in the lower end of `bili` values. We also see that most of the higher values experienced an event. The “normal” range of Bilirubin is from 0.3 to 1.9 mg/dL, indicating the distribution from our population is well outside the normal range. These values make biological sense considering Bilirubin is a pigment created in the liver, the organ effected by the PBC disease. The figure also shows that the risk of death increases as time progresses. The risk at three years is much greater than that at one year for patients with high Bilirubin values compared to those with values closer to the normal range.



**Figure 7-14.** Variable dependence of survival at 1 and 3 years on `bili` variable. Individual cases are marked with blue circles (alive or censored) and red `x` (dead). Loess smooth curve with shaded 95% confidence band indicates decreasing survival with increasing bilirubin.

With reference to the code below which generates **Figure 7-15**, the `plot.gg_variable` function call operates on the `gg_variable` object controlled by the list of variables of interest in the `xvar` argument. By default, the `plot.gg_variable` function returns a list of `ggplot` objects, one figure for each variable named in `xvar`. The remaining arguments are passed to internal functions controlling the display of the figure. The `se` argument is passed to the internal call to `geom_smooth` for fitting smooth lines to the data. The `alpha` argument lightens the coloring points in the `geom_point` call, making it easier to see point over plotting. We also demonstrate modification of the plot labels using the `labs` function and point attributes with the `scale_functions`.

```
# Create the variable dependence object from the random
forest
gg_v <- gg_variable(rfsrc_pbc)
# We want the top ranked minimal depth variables only
xvar <- gg_md$topvars
# plot the variable list in a single panel plot
plot(gg_v, xvar=xvar, panel=TRUE, alpha=.5) +
  labs(y=st.labs["medv"], x="") +
  theme(legend.position = "top")
```



**Figure 7-15.** Variable dependence of predicted survival at 1 and 3 years on continuous variables of interest. Individual cases are marked with blue circles for censored cases and red x for death events. Loess smooth curve indicates the survival trend with increasing values.

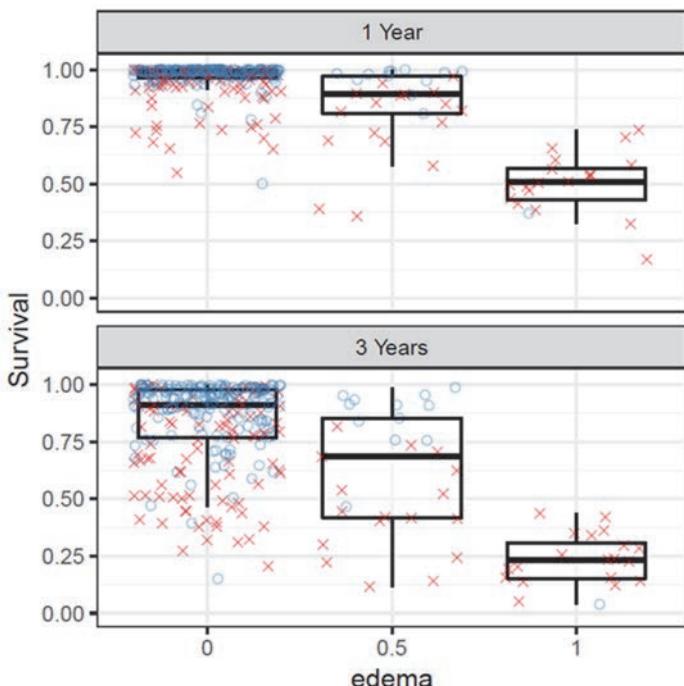
This figure looks similar to *Figure 7-5* although with a transposed axis as we plot the response variable on the y-axis. The closer the panels match, the better the RF prediction. The panels are sorted to match the order of variables in the `xvar` argument and include a smooth loess line (Cleveland W. , 1981), with 95% shaded confidence band, to indicates the trend of the prediction dependence over the covariate values. We show censored cases as teal triangles, events are indicated by the orange circles symbols.

The figures indicate that survival increases with albumin level, and decreases with copper, prothrombin, and age. Note the extreme value of prothrombin ( $> 16$ ) influences the loess curve more than other points, which would make it a candidate for further investigation.

We expect survival at three years to be lower than at one year. However, comparing the two time-plots for each variable does indicate a difference in response relation for `bili`, `copper` and `prothrombin`. The added risk for elevated levels of these variables at three years indicates a non-proportional hazards response. The similarity between the time curves for albumin and age indicates the effect of these variables is constant over the disease progression.

There is not a convenient method to panel scatter plots and box plots together, so we recommend creating panel plots for each variable type separately. We plot the categorical variable (`edema`) separately from the continuous variables in *Figure 7-16*.

```
plot(gg_v, xvar = xvar.cat, alpha = 0.4) +  
  labs(y = "Survival") +  
  theme(legend.position = "none") +  
  scale_color_manual(values = strCol,  
    labels = event.labels) +  
  scale_shape_manual(values = event.marks,  
    labels = event.labels) +  
  coord_cartesian(ylim = c(-0.01, 1.02))
```



**Figure 7-16. Variable dependence of survival 1 and 3 years on edema categorical variable. Symbols with blue circles indicate censored cases and red x indicate death events. Boxplots indicate distribution of predicted survival for all observations within each edema group.**

The `gg_variable` plot of for categorical variable dependence displays boxplots to examine the distribution of predicted values within each level of the variable. The points are plotted with a jitter to see the censored and event markers more clearly. The boxes are shown with horizontal bars indicating the median, 75th (top) and 25th (bottom) percentiles. Whiskers extend to 1.5 times the interquartile range. Points plotted beyond the whiskers are considered outliers.

When using categorical variables with linear models, we use Boolean dummy variables to indicate class membership. In the case of edema, we would create two logical variables for `edema = 0.5` (complex Edema presence indicator) and `edema = 1.0` (Edema with diuretics) contrasted with the `edema = 0` variable (no Edema). Random Forest can use factor variables directly, separating the populations into homogeneous groups of `edema` at nodes that split on that variable. indicates similar survival response distribution between 1- and 3-year when `edema = 1.0`. The distribution of predicted survival does seem

to spread out more than for the other values, again indicating a possible non-proportional hazards response.

## 7.14 Partial Dependence

plots are a risk adjusted alternative to variable dependence. We generated partial plots by integrating out the effects of variables beside the covariate of interest. We construct the figures by selecting points evenly spaced along the distribution of the variable of interest. For each of these points ( $X = x$ ), we calculate the average RF prediction over all remaining covariates in the training set by

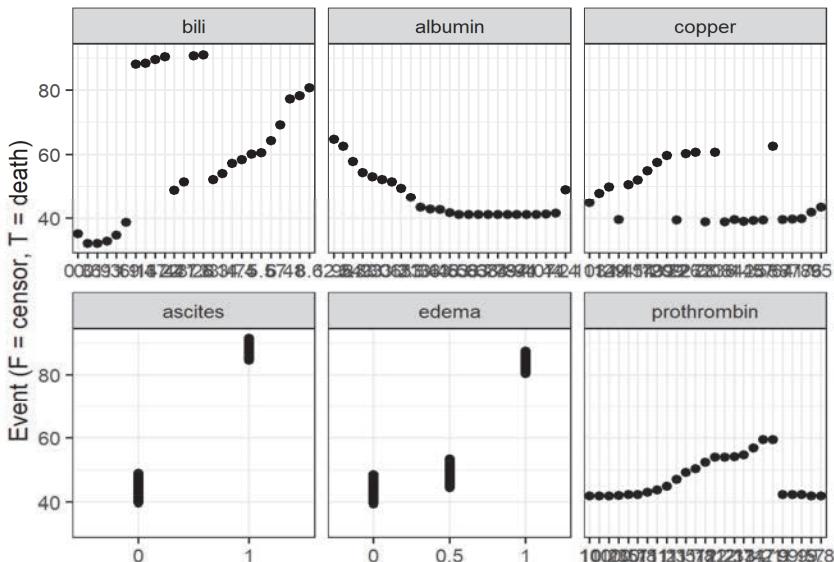
$$\tilde{f}(x) = \frac{1}{n} \sum_{i=1}^n \hat{f}(x, x_{i,o})$$

where  $\hat{f}$  is the predicted response from the random forest and  $x_{i,o}$  is the value for all other covariates other than  $X = x$  for observation  $i$  (Friedman, 2000).

Partial dependence plots (see **Figure 7-17**) are another computationally intensive analysis, especially when there are a large number of observations. We again turn to our data caching strategy here. The default parameters for the `randomForestSRC::plot.variable` function generate partial dependence estimates at `npts=25` points (the default value) along the variable of interest. For each point of interest, the `plot.variable` function averages  $n$  response predictions. This is repeated for each of the variables of interest and the results are returned for later analysis.

```
# Load the data, from the call:  
partial_pbc <- plot.variable(rfsrc_pbc,  
                                xvar=gg_md$topvars,  
                                partial=TRUE, sorted=FALSE,  
                                show.plots = FALSE )  
  
# generate a list of gg_partial objects, one per xvar.  
gg_p <- gg_partial(partial_pbc)  
  
# plot the variable list in a single panel plot  
plot(gg_p, panel=TRUE) +  
  labs(y=st.labs["status"], x="") +  
  geom_smooth(se=FALSE)
```

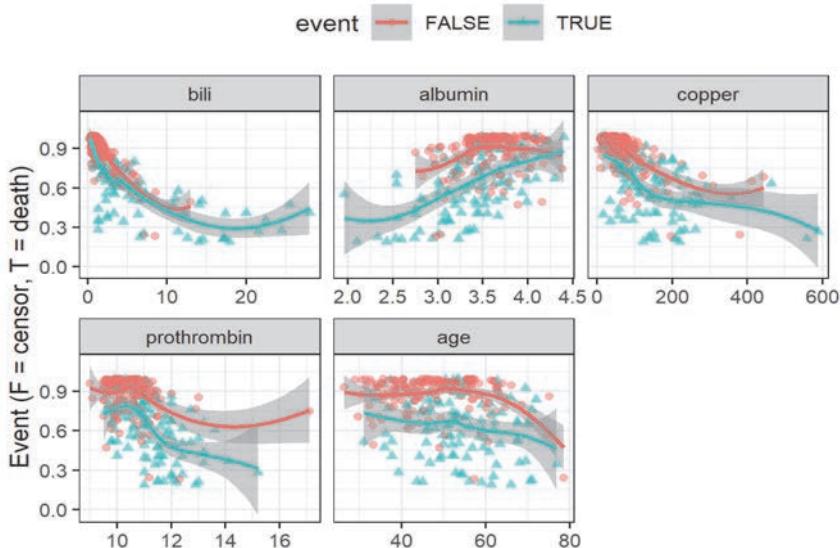
The `gg_partial` function creates a list of objects (one for each variable) using the result of the `randomForestSRC plot.variable` function, which we can then plot using `plot.gg_partial`.



*Figure 7-17. Partial dependence plots for the top six variables.*

```
plot(gg_v, xvar = xvar, panel = TRUE, alpha = .5) +
  labs(y = st.labs["status"], x = "") +
  theme(legend.position = "top")
```

We again order the panels by minimal depth ranking. We see again how the `bili` and `copper` variables are strongly related to the median value response, making the partial dependence of the remaining variables look flat. We also see strong nonlinearity of these two variables. The `bili` variable looks quadratic, while the copper shape is more complex.



*Figure 7-18. More partial dependence plots using alternative graphics*

We could stop here, indicating that the RF analysis has found these ten variables to be important in predicting PBC patient survival. That increasing **bili** (serum bilirubin in mg/dl) values are associated with decreasing median survivability values (status) and increasing **copper** > 6 (urine copper in ug/day > 6) are associated with increasing median survivability values. However, we may also be interested in investigating how these variables work together to help improve the random forest prediction of median survivability values.

```
xvar <- c(xvar, xvar.cat)

time_index <- c(which(rfsrc_pbc$time.interest > 1)[1]-1,
  which(rfsrc_pbc$time.interest > 3)[1]-1,
  which(rfsrc_pbc$time.interest > 5)[1]-1)
partial_pbc <- clapply(rfsrc_pbc$time.interest[time_index],
  function(tm){
  plot.variable(rfsrc_pbc, surv.type = "surv",
  time = tm, xvar.names = xvar,
  partial = TRUE ,
  show.plots = FALSE)
})

gg_dta <- mclapply(partial_pbc, gg_partial)
cbc_ggpart <- combine.gg_partial(gg_dta[[1]], gg_dta[[2]],
```

```

lbls = c("1 Year", "3 Years")

ggplot(pbc_ggpart[["edema"]], aes(y=yhat, x=edema,
  col=group))+  

  geom_boxplot(notch = TRUE,  

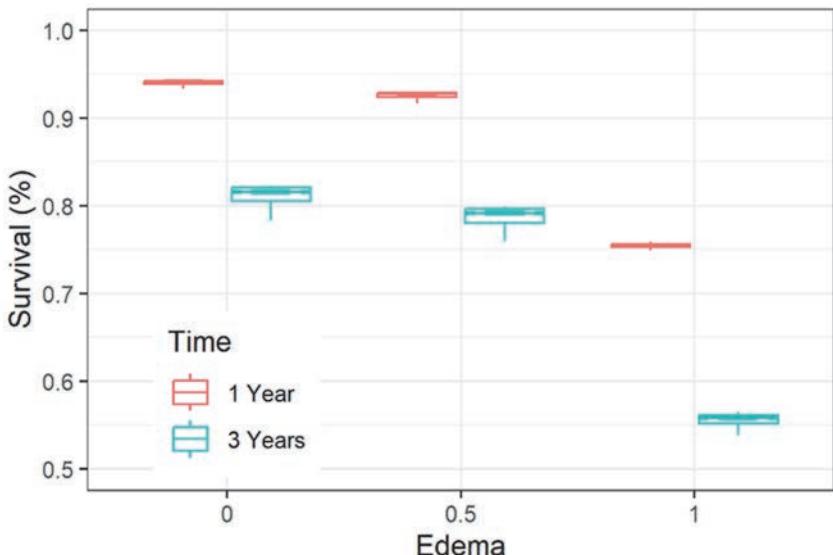
  outlier.shape = NA) + # panel=TRUE,  

  labs(x = "Edema", y = "Survival (%)", color="Time",  

shape="Time") +  

  theme(legend.position = c(0.2, 0.2))

```



*Figure 7-19. Partial dependence plot of predicted survival at 1 year (red) and 3 years (blue) as a function of edema groups (categorical variable). Boxplots indicate distribution within each group.*

## 7.15 Variable Interactions

Using the different variable dependence measures, it is also possible to calculate measures of pairwise interactions among variables. Recall that we define minimal depth measure by averaging the tree depth of variable  $i$  relative to the root node. To detect interactions, we can modify this calculation to measure the minimal depth of a variable  $j$  with respect to the maximal subtree for variable  $i$  (Ishwaran H., Kogalur, Gorodeski, & Minn, 2010; Ishwaran H., Kogalur, Chen, & J., 2011), Ishwaran, 2011).

The `randomForestSRC::find` interaction function traverses the forest, calculating all pairwise minimal depth interactions, and returns a

$p \times p$  matrix of interaction measures. For each row, the diagonal terms are related to the minimal depth relative to the root node, though normalized to minimize scaling issues. Each off diagonal minimal depth term is relative to the diagonal term on that row. Small values indicate that an off-diagonal term typically splits close to the diagonal term, indicating a forest split proximity of the two variables.

The `gg_interaction` function wraps the `find.interaction` matrix for use with the provided plot and print functions. The `xvar` argument indicates which variables we are interested in looking at. We again use the cache strategy and collect the figures together using the `panel=TRUE` option.

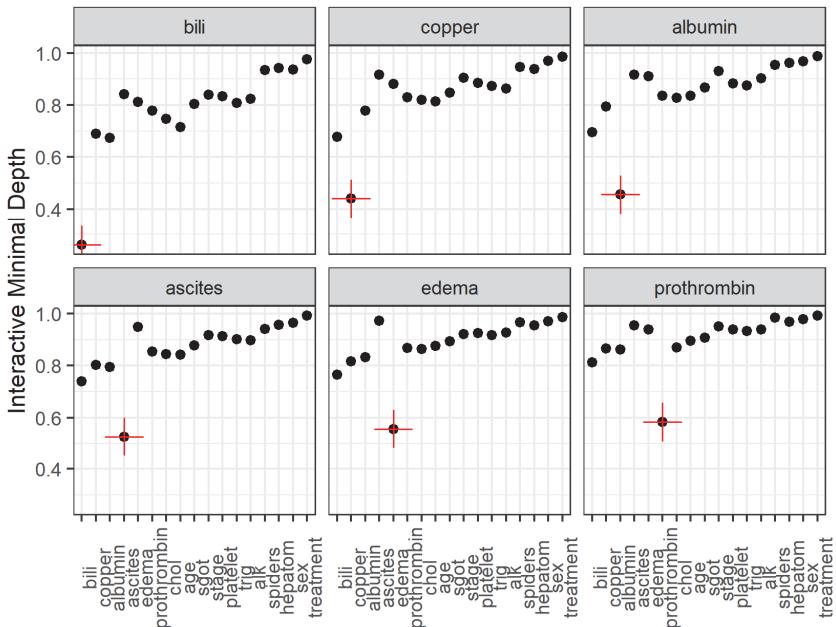
```
# Load the data, from the call:
interaction_pbc <- find.interaction(rfsrc_pbc)
```

	Method: maxsubtree							
	No. of variables: 17							
	Variables sorted by minimal depth?: TRUE							
bili	0.27	0.73	0.69	0.84	0.88	0.76		
copper	0.64	0.41	0.74	0.87	0.90	0.81		
albumin	0.73	0.81	0.48	0.92	0.93	0.86		
edema	0.75	0.81	0.81	0.54	0.97	0.88		
ascites	0.73	0.79	0.83	0.95	0.54	0.85		
prothrombin	0.80	0.86	0.87	0.95	0.95	0.59		
chol	0.82	0.89	0.86	0.95	0.97	0.90		
age	0.85	0.90	0.86	0.97	0.97	0.94		
sgot	0.87	0.92	0.92	0.98	0.99	0.94		
stage	0.90	0.92	0.93	0.96	0.99	0.96		
platelet	0.90	0.94	0.93	0.97	0.97	0.97		
alk	0.91	0.94	0.93	0.99	0.99	0.96		
trig	0.91	0.92	0.91	0.99	0.99	0.97		
hepatom	0.91	0.94	0.94	0.97	0.99	0.96		
spiders	0.92	0.94	0.94	0.99	0.99	0.97		
sex	0.95	0.97	0.96	1.00	1.00	0.98		
treatment	1.00	1.00	1.00	1.00	1.00	1.00		
	chol	age	sgot	stage	platelet	alk	trig	hepatom
bili	0.73	0.71	0.84	0.86	0.84	0.81	0.83	0.93
copper	0.80	0.81	0.83	0.91	0.86	0.87	0.87	0.93
albumin	0.81	0.84	0.87	0.92	0.91	0.90	0.87	0.96
edema	0.85	0.85	0.91	0.93	0.92	0.90	0.90	0.95
ascites	0.84	0.85	0.88	0.94	0.91	0.91	0.91	0.94
prothrombin	0.89	0.91	0.91	0.95	0.94	0.93	0.93	0.96

chol	0.59	0.90	0.92	0.95	0.92	0.93	0.94	0.98
age	0.90	0.66	0.94	0.97	0.93	0.94	0.93	0.98
sgot	0.94	0.93	0.73	0.97	0.95	0.95	0.95	0.98
stage	0.93	0.95	0.94	0.76	0.95	0.96	0.95	0.97
platelet	0.94	0.96	0.96	0.97	0.76	0.97	0.96	0.99
alk	0.94	0.94	0.95	0.98	0.98	0.78	0.95	0.99
trig	0.94	0.95	0.96	0.99	0.96	0.96	0.79	0.99
hepatom	0.93	0.94	0.95	0.97	0.98	0.98	0.97	0.84
spiders	0.95	0.96	0.96	0.98	0.96	0.97	0.97	0.98
sex	0.97	0.97	0.98	0.99	0.99	0.98	0.98	0.99
treatment	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
					spiders	sex	treatment	
bili					0.92	0.93	0.98	
copper					0.92	0.97	0.99	
albumin					0.94	0.95	0.99	
edema					0.95	0.96	0.99	
ascites					0.94	0.95	0.99	
prothrombin					0.97	0.97	1.00	
chol					0.98	0.97	0.99	
age					0.97	0.98	0.99	
sgot					0.98	0.99	1.00	
stage					0.99	0.98	1.00	
platelet					0.98	0.98	1.00	
alk					0.99	0.99	0.99	
trig					0.99	0.99	0.99	
hepatom					0.99	0.98	1.00	
spiders					0.84	0.99	1.00	
sex					0.99	0.91	1.00	
treatment					1.00	1.00	0.98	

We plot the results in **Figure 7-20**

```
# Plot the results in a single panel.
plot(gg_interaction(interaction_pbc),
      xvar=gg_md$topvars, panel=TRUE)
```



**Figure 7-20. Minimal depth variable interaction plot for six variables of interest. Higher values indicate lower interactivity with target variable marked in red.**

**Figure 7-20** plots the interactions for the target variable (shown in the red cross) with interaction scores for all remaining variables. We expect the covariate with lowest minimal depth (**bili**) to be associated with all other variables, as it typically splits close to the root node, so viewed alone it may not be as informative as looking at a collection of interactive depth plots. Scanning across the panels, we see each successive target depth increasing, as expected. We also see the interactive variables increasing with increasing target depth. Of interest here is the interaction of **bili** with the **copper** variable shown in the **copper** panel. Aside from these being the strongest variables by both measures, this interactive measure indicates the strongest connection between variables.

## 7.16 Coplots

Conditioning plots (coplots) (Chambers, 1992) are a powerful visualization tool to efficiently study how a response depends on two or more variables (Cleveland W. S., 1993). The method allows us to view data by grouping observations on some conditional membership. The

simplest example involves a categorical variable, where we plot our data conditional on class membership, for instance on the `status` logical variable. We can view a coplot as a stratified variable dependence plot, indicating trends in the RF prediction results within panels of group membership.

Conditional membership with a continuous variable requires stratification at some level. Often we can implement stratification along some feature of the variable. For instance we can stratify a variable with integer values, or 5- or 10-year age group cohorts. However in the variables of interest in our PBC example, we have no “logical” stratification indications. Therefore we will arbitrarily stratify our variables into six groups of equal population size using the `quantile_pts` function. We pass the break points located by `quantile_pts` to the `cut` function to create grouping intervals, which we can then add to the `gg_variable` object before plotting with the `plot.gg_variable` function. The simple modification to convert variable dependence plots into condition variable dependence plots is to use the `ggplot2::facet_wrap` command to generate a panel for each grouping interval (see **Figure 7-21**).

We start by examining the predicted survivability value as a function of `bili` conditional on membership within six groups of copper “intervals.”

```
ggvar <- gg_variable(rfsrc_pbc, time = 1)

# Find intervals with similar number of observations.
bili_cts <- quantile_pts(ggvar$bili, groups = 6, intervals = TRUE)

# We need to move the minimal value so we include that
observation
bili_cts[1] <- bili_cts[1] - 1.e-7
# Create the conditional groups and add to the
gg_variable object
bili_grp <- cut(ggvar$bili, breaks = bili_cts)
ggvar$bili_grp <- bili_grp

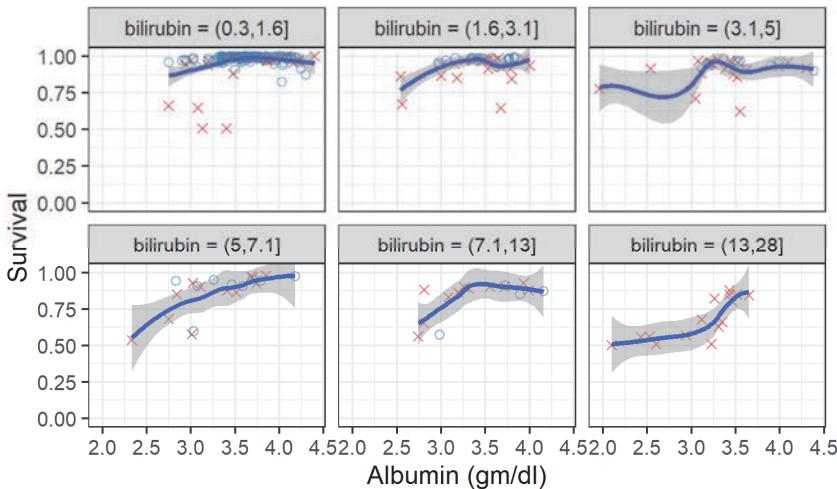
# Adjust naming for facets
levels(ggvar$bili_grp) <- paste("bilirubin =", 
levels(bili_grp))

# plot.gg_variable
plot(ggvar, xvar = "albumin", alpha = 0.5) +
```

```

# method = "glm", se = FALSE) +
  labs(y = "Survival", x = st.labs["albumin"]) +
  theme(legend.position = "none") +
  scale_color_manual(values = strCol, labels = event.labels)
+
  scale_shape_manual(values = event.marks, labels =
event.labels) +
  facet_wrap(~bili_grp) +
  coord_cartesian(ylim = c(-0.01,1.01))

```



*Figure 7-21. Variable dependence coplot of survival at 1 year against albumin, conditional on bili interval group membership.*

Each point in this figure is the predicted median survival value response plotted against albumin value conditional on `bili` being on the interval specified. The `gg_variable` coplot of Figure 24 indicates the probability of survival increases with increasing `albumin` and increases within groups of increasing `bili`.

Typically, conditional plots for continuous variables include overlapping intervals along the grouped variable (Cleveland 1993). We chose to use mutually exclusive continuous variable intervals for the following reasons:

- **Simplicity** - We can create the coplot figures directly from the `gg_variable` object by adding a conditional group column directly to the object.
- **Interpretability** -We find it easier to interpret and compare the panels if each observation is only in a single panel.

- **Clarity** - We prefer using more space for the data portion of the figures than typically displayed in the coplot function which requires the bar plot to present the overlapping segments.

It is still possible to augment the `gg_variable` to include overlapping conditional membership with continuous variables by duplicating rows of the training set data within the `rfsrc$xvar` object, and then setting the conditional group membership as described. We could use the `plot.gg_variable` function recipe above to generate the panel plot, with panels ordered according to the factor levels of the grouping variable. We leave this as an exercise for you to consider.

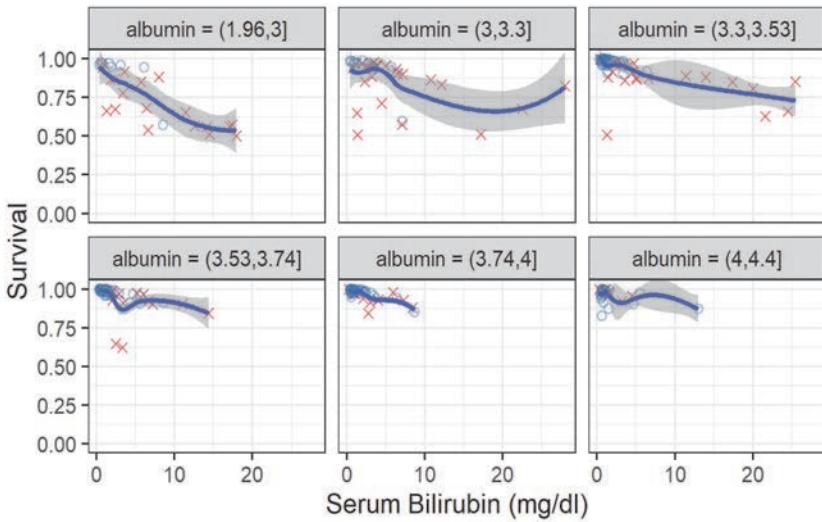
```
# Find intervals with similar number of observations.
albumin_cts <- quantile_pts(ggvar$albumin, groups = 6,
                             intervals = TRUE)

# We need to move the minimal value so we include that
# observation
albumin_cts[1] <- albumin_cts[1] - 1.e-7

# Create the conditional groups and add to the
# gg_variable object
albumin_grp <- cut(ggvar$albumin, breaks = albumin_cts)
ggvar$albumin_grp <- albumin_grp

# Adjust naming for facets
levels(ggvar$albumin_grp) <- paste("albumin =", 
                                      levels(albumin_grp))

# plot.gg_variable
plot(ggvar, xvar = "bili", alpha = 0.5) +
  # method = "glm", se = FALSE) +
  labs(y = "Survival", x = st.labs["bili"]) +
  theme(legend.position = "none") +
  scale_color_manual(values=strCol,
                     labels=event.labels) +
  scale_shape_manual(values = event.marks,
                     labels = event.labels) +
  facet_wrap(~albumin_grp) +
  coord_cartesian(ylim = c(-0.01,1.01))
```



*Figure 7-22. Variable dependence coplot of survival at 1 year against bili, conditional on albumin interval group membership.*

## 7.17 Partial Dependence Coplots

By characterizing conditional plots as stratified variable dependence plots, the next logical step would be to generate an analogous conditional partial dependence plot. The process is similar to variable dependence coplots, first determine conditional group membership, then calculate the partial dependence estimates on each subgroup using the `randomForestSRC::plot.variable` function with a subset argument for each grouped interval. The `ggRandomForests::gg_partial_coplot` function is a wrapper for generating a conditional partial dependence data object. Given a random forest (`randomForestSRC::rfsrc` object) and a groups vector for conditioning the training data set observations, `gg_partial_coplot` calls the `randomForestSRC::plot.variable` function for a set of training set observations conditional on groups membership. The function returns a `gg_partial_coplot` object, a sub class of the `gg_partial` object, which we can plot with the `plot.gg_partial` function.

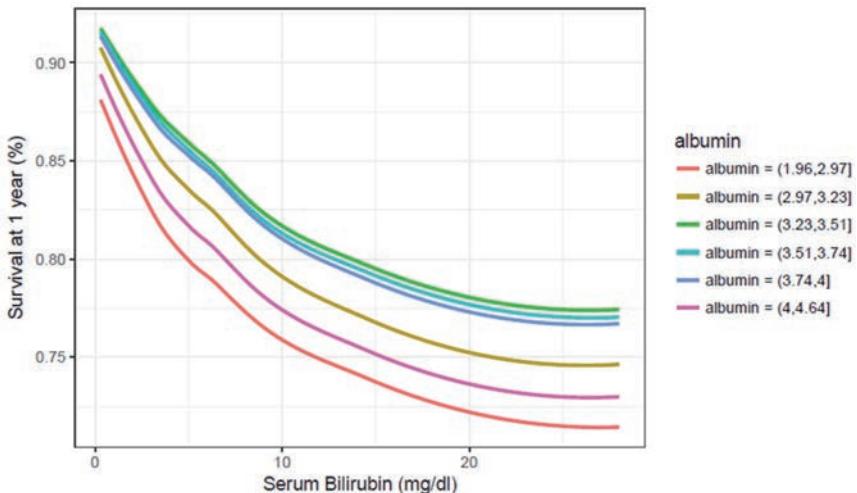
The following code block will generate the data object for creating partial dependence coplot of one year survival as a function of `bili` conditional on membership within the six groups of `albumin` intervals that we examined in *Figure 7-22*.

```

partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc,
  xvar = "bili",
  groups = ggvar$albumin_grp,
  surv_type = "surv",
  time= rfsrc_pbc$time.interest[time_index[1]],
  show.plots = FALSE)

ggplot(partial_coplot_pbc, aes(x=bili, y=yhat,
  col=group, shape=group)) +
  geom_smooth(se = FALSE) +
  labs(x = st.labs["bili"], y = "Survival at 1 year (%)",
  color = "albumin", shape = "albumin")

```



*Figure 7-23. Partial dependence coplot of survival at 1 year against bili, conditional on albumin interval group membership. Points estimates with loess smooth to indicate trend within each group.*

The `gg_partial_coplot` of **Figure 7-23** shows point estimates of the risk adjusted survival as a function of `bili` conditional on group membership defined by albumin intervals. The figure is slightly different than the `gg_partial` plot of **Figure 7-24** as each set of partial dependence estimates is calculated over a subset of the training data. We again connect the point estimates with a Loess curve.

For completeness, we construct the compliment coplot view of one year survival as a function of albumin conditional on `bili` interval group membership in **Figure 7-24**.

```

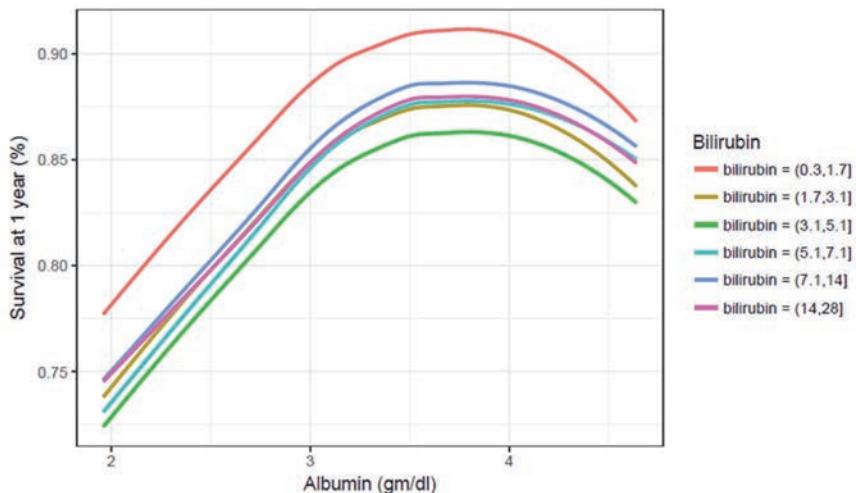
partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc,
```

```

xvar = "albumin", groups = ggvar$bili_grp,
surv_type = "surv",
time= rfsrc_pbc$time.interest[time_index[1]],
show.plots = FALSE)

ggplot(partial_coplot_pbc, aes(x=albumin, y=yhat,
col=group, shape=group)) +
  geom_smooth(se = FALSE) +
  labs(x = st.labs["albumin"], y = "Survival at 1 year
(%)",
color = "bili", shape = "bili")

```



*Figure 7-24. Partial dependence coplot of survival at 1 year against albumin, conditional on bilirubin interval group membership. Points estimates with loess smooth to indicate trend within each group.*

### 7.17.1 Partial Dependence Plot

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-year and 3-year partial dependence plots for each continuous variable in *Error! Reference source not found.* through *Error! Reference source not found..* We also plot our categorical variables in *Error! Reference source not found.* and **Figure 7-28**.

```

partial_alb <- plot.variable(rfsrc_pbc, xvar.names =
"albumin", partial=TRUE)

```

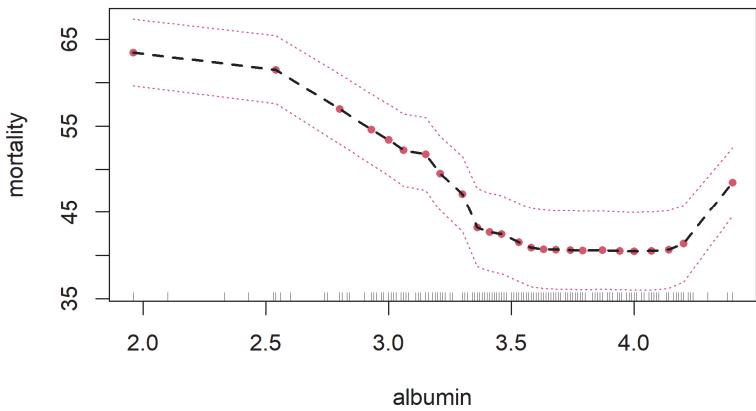


Figure 7-25. Partial dependence plot for albumin vs. mortality

```
partial_bili <- plot.variable(rfsrc_pbc, xvar.names = "bili", partial=TRUE)
```

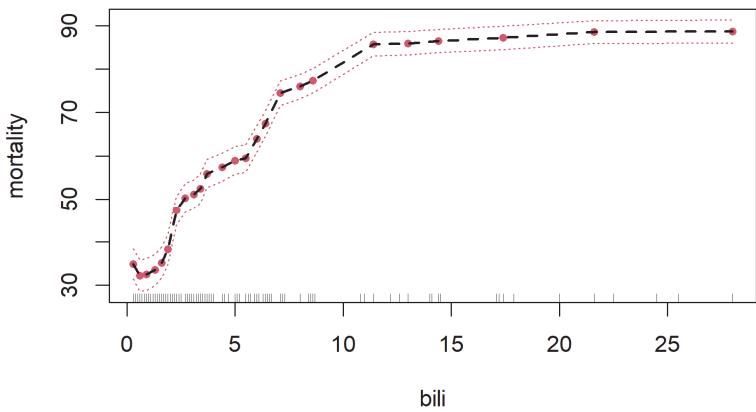
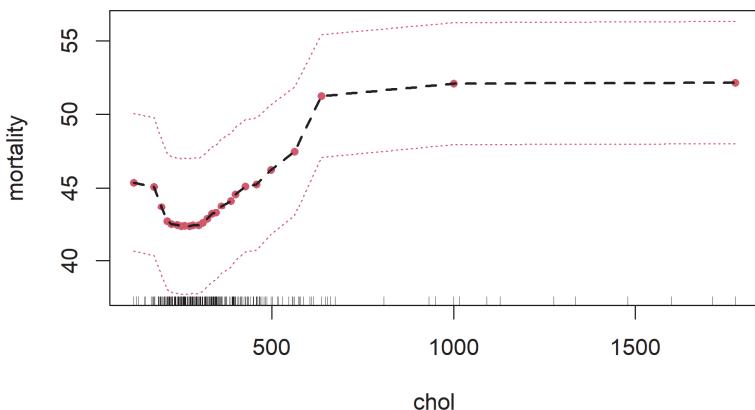


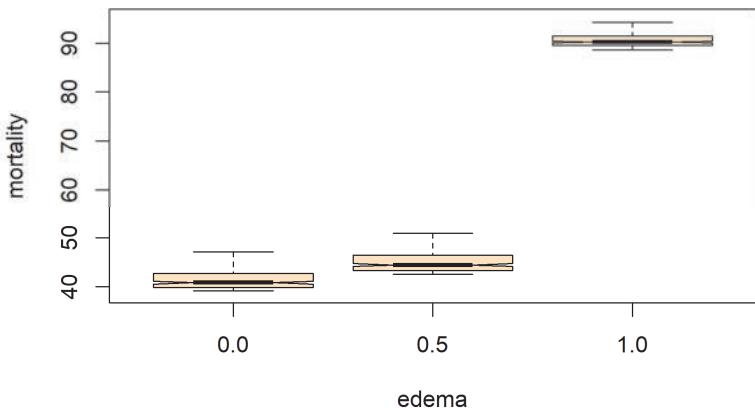
Figure 7-26. Partial dependence plot for bilirubin vs. mortality

```
partial_chol <- plot.variable(rfsrc_pbc, xvar.names = "chol", partial=TRUE)
```



*Figure 7-27. Partial dependence plot for cholesterol vs. mortality*

```
partial_edema <- plot.variable(rfsrc_pbc, xvar.names =
  "edema",
  partial=TRUE)
```



*Figure 7-28. Partial dependence plot for edema vs. mortality*

### 7.17.2 Conditional Dependence Plots

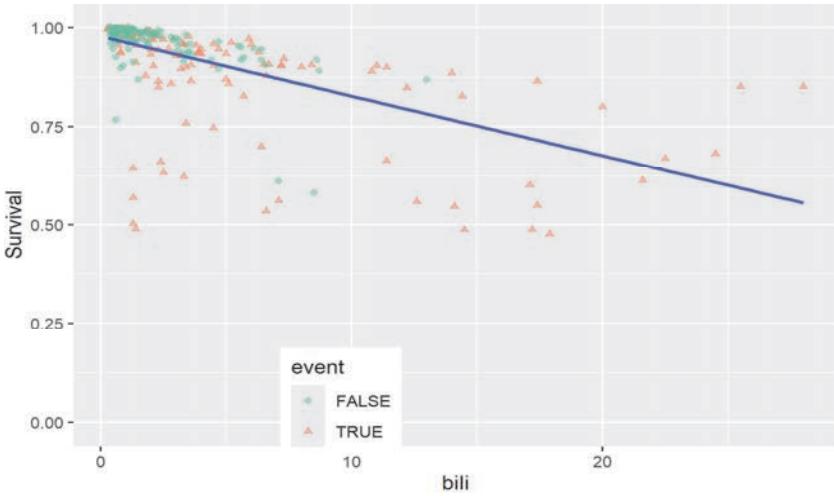
Now, we generate the conditional dependence plot for serum bilirubin with fitted line. We can view the conditional dependence of survival against bilirubin, conditional on edema group membership (categorical variable) in **Figure 7-29** by reusing the saved ggplot object (**var\_dep**) grid function.

```
ggvar <- gg_variable(rfsrc_pbc, time = 1)
ggvar$stage <- paste("stage = ", ggvar$stage, sep = "") 
var_dep <- plot(ggvar, xvar = "bili",
```

```

            method = "glm", alpha = .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

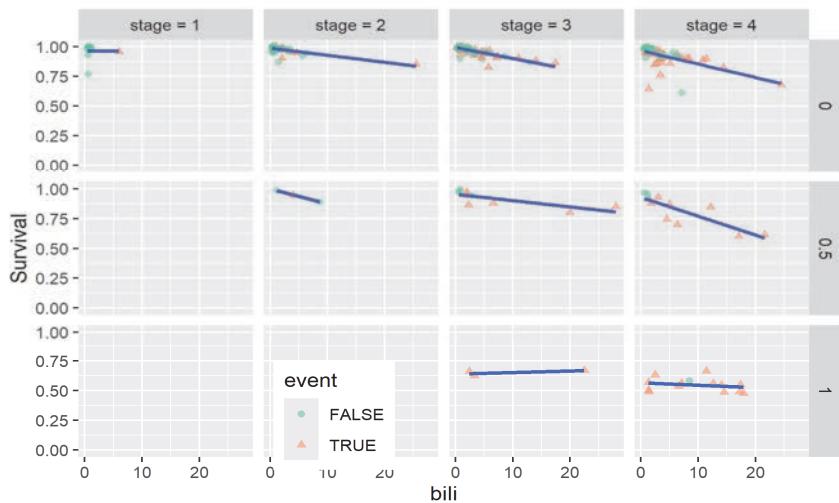
```



*Figure 7-29. 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.

```
var_dep + facet_grid(edema~stage)
```



**Figure 7-30. Conditional dependence of survival against bilirubin, versus other categorical covariates**

Furthermore, we find intervals with similar number of observations.

### 7.17.3 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)
```

### 7.17.4 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")
```

## 7.18 Summary: How random forest works

Each tree is grown as follows:

1. Random Record Selection : Each tree is trained on roughly 2/3rd of the total training data (exactly 63.2%) . Cases are drawn at random with replacement from the original data. This sample will be the training set for growing the tree.
2. Random Variable Selection : Some predictor variables (say,  $m$ ) are selected at random out of all the predictor variables and the best split on these  $m$  is used to split the node. By default,  $m$  is square root of the total number of all predictors for classification. For regression,  $m$  is the total number of all predictors divided by 3. The value of  $m$  is held constant during the forest growing.

Note : In a standard tree, each split is created after examining every variable and picking the best split from all the variables.

3. For each tree, using the leftover (36.8%) data, calculate the misclassification rate - out of bag (OOB) error rate. Aggregate error from all trees to determine overall OOB error rate for the classification. If we grow 200 trees then on average a record will be OOB for about  $.37*200=74$  trees.
4. Each tree gives a classification on leftover data (OOB), and we say the tree “votes” for that class. The forest chooses the classification having the most votes over all the trees in the forest. For a binary dependent variable, the vote will be YES or NO, count up the YES votes. This is the RF score and the percent YES votes received is the predicted probability. In regression case, it is average of dependent variable.

For example, suppose we fit 500 trees, and a case is out-of-bag in 200 of them: - 160 trees votes class 1 - 40 trees votes class 2

In this case, RF score is class1. Probability for that case would be 0.8 which is 160/200. Similarly, it would be an average of target variable for regression problem.

We have found that the variable importance measures produced by random forests can sometimes be useful for model variable reduction.



## 8 Cluster Models

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups (more often than not the data seems to be a cluster of grapes, for instance). We call each group a cluster and exploring the differences (and similarities when they overlap) is a focal point of cluster analysis. An essential task of exploratory data mining, and a common technique for statistical data analysis are in order. We use cluster analysis in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Rather than a specific algorithm, cluster analysis offers a general process for solving complex problems. We have numerous algorithms that differ in both their notion of what constitutes a cluster and how to efficiently find those clusters. Popular designs of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals, or particular statistical distributions. Therefore, we can formulate clustering as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings depend on the individual data set and intended use of the results. Parameters include values such as the distance function to use, a density threshold or the number of expected clusters. As such, cluster analysis is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and error. We often need to modify data preprocessing and model parameters until the result achieves the desired properties.

Cluster analysis was originated in anthropology by Driver and Kroeber in 1932 and introduced to psychology by Zubin in 1938 and Robert Tryon in 1939 (Bailey, 1994) (Tryon, 1939) and famously used by Cattell beginning in 1943 (Cattell, 1943) for trait theory classification in personality psychology.

### 8.1 Definition

According to Vladimir Estivill-Castro, the notion of a “cluster” cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms (Estivill-Castro, 2002). However, there is a common denominator: a group of data objects. Different researchers employ different cluster models, and for each of these cluster models again

different algorithms can be given. The notion of a cluster, as found by different algorithms, varies significantly in its properties. Understanding these “cluster models” is key to understanding the differences between the various algorithms. Typical cluster models include:

**Connectivity models.** The models are hierarchical, where clustering builds models based on distance connectivity.

**Centroid models.** These models use a single mean vector, like the  $k$ -means algorithm, to represent each cluster by a single mean vector.

**Distribution models.** We model the clusters using statistical distributions, such as multivariate normal distributions used by the Expectation-maximization algorithm.

**Density models.** Here we define clusters as connected dense regions in the data space, for example, DBSCAN and OPTICS.

**Subspace models.** In this case, we model clusters with both cluster members and relevant attributes. We call this **Biclustering**, co-clustering or two-mode-clustering.

**Group models.** Some algorithms do not provide any advanced model for their results and just provide the grouping information.

**Graph-based models.** Nodes in a graph (or their subsets, called cliques) are connected by an edge and we can consider them as a prototypical form of cluster. We know relaxations of the complete connectivity requirement (a fraction of the edges can be missing) as quasi-cliques.

A “clustering” is a set of such clusters, usually containing all objects in the data set. Additionally, a clustering may specify the relationship of the clusters to each other, for example a hierarchy of clusters embedded in each other. Clustering can be roughly distinguished as:

- **Hard clustering.** Here, each object either belongs to a cluster or it does not.
- **Soft clustering.** Here, each object belongs to each cluster to a certain degree (e.g. a probability of belonging to the cluster).

There are also finer distinctions possible, for example:

- **Strict partitioning clustering.** Here, each object belongs to exactly one cluster.
- **Strict partitioning clustering with outliers.** Here, objects can also belong to no cluster and are considered outliers.
- **Overlapping clustering.** In this case, while usually considered a hard clustering, objects may belong to more than one cluster. This is also known as multi-view clustering.
- **Hierarchical clustering.** Here, objects that belong to a child cluster also belong to the parent cluster.
- **Subspace clustering.** In this instance, we have an overlapping clustering, but within a uniquely defined subspace, clusters are not expected to overlap.

## 8.2 Algorithms

Based on the cluster models we briefly described above, we can categorize clustering algorithms. We will only discuss the most prominent examples of clustering algorithms, as there are over 100 published clustering algorithms. Some algorithms do not provide models for their clusters and can thus not easily be categorized.

There is not one objectively “correct” clustering algorithm, but as it was noted, “clustering is in the eye of the beholder.” (Estivill-Castro, 2002) We often choose the most appropriate clustering algorithm for a particular problem experimentally, unless there is a mathematical reason to prefer one cluster model over another. We should note that an algorithm that is designed for one kind of model has no chance on a data set that contains a radically different kind of model. For example,  $k$ -means cannot find non-convex clusters (Estivill-Castro, 2002).

Some algorithms that we will cover include:

- K-means clustering, a centroid-based cluster model
- Agglomerative clustering, a connectivity-based cluster model
- Divisive hierarchical clustering, also a connectivity-based cluster model
- *Gaussian Mixed Model* (GMM), a model-based cluster model
- DBSCAN, a density-based cluster model

I will begin this discussion on *k-means* clustering going step-by-step, using a data set that we design for the purpose of illustrating concepts. Next, we will cover *agglomerative clustering*, followed by *divisive clustering* g, *medoid clustering*, and a model-based cluster algorithm named *Gaussian Mixed Model*, using the same data and then perform a comparison of algorithms. Then we will take a closer look at hierarchical models. Finally, we will examine storm (severe weather) data using a clustering model approach.

## 8.3 k-Means Clustering

K-means is a popular unsupervised machine learning technique that allows the identification of clusters (similar groups of data points) within the data. We will go through an example of *k*-means clustering step-by-step, followed by storm (weather) model we mentioned earlier. So, here are the steps.

## 8.4 Steps

1. Install and import relevant libraries.
2. Create the data.
3. Use K-means and medoids to cluster data.
4. Use Hierarchical clustering.
5. Use Gaussian Mixed Models to cluster.
6. Compare the accuracy of each method.

### 8.4.1 Step 1. Install and import relevant libraries

In this step, we'll import the necessary R libraries.

```
library("factoextra")
library("mvtnorm")
library("dplyr")
library("ggplot2")
library("fpc")
library("caret")
library("mclust")
library("cluster")

for(pkg in c("mvtnorm", "dplyr", "ggplot2", "fpc", "caret", "
```

```

mclust", "factoextra", "cluster")){
  suppressPackageStartupMessages(library(pkg, character.only
= TRUE))
}

set.seed(42)

```

## 8.4.2 Step 2. Create the data

For this example, first we will generate data so that we can be sure of the underlying structure and have labels for our dataset to compare the efficacy of different methods. We generate three clusters using normal distributions. Each data point has three features: an  $x$  value, a  $y$  value, and a class that labels the point. We'll have three classes that we'd like our clustering algorithms to represent as clusters.

```

dataset <- {
  #create our 3 distributions
  cluster1 = data.frame(rmvnorm(40, c(0, 0),
    diag(2) * c(2, 1))) %>% mutate(class = factor(1))
  cluster2 = data.frame(rmvnorm(100, c(3, 3),
    diag(2) * c(1, 3))) %>% mutate(class = factor(2))
  cluster3 = data.frame(rmvnorm(60, c(6, 6),
    diag(2))) %>% mutate(class = factor(3))
  #bind them together
  data = bind_rows(cluster1, cluster2, cluster3)
  #set the column names
  names(data) = c("x", "y", "class")
  #return the data
  data
}

```

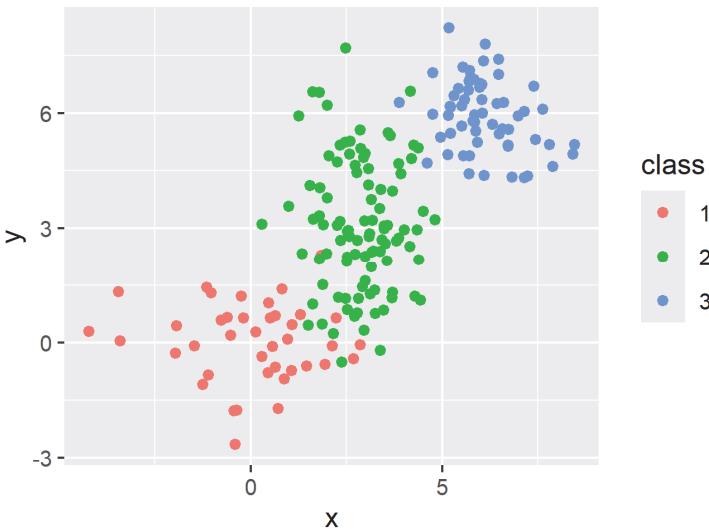
Since the data is being generated here we don't need to check for missing values or normalize our features. If our  $x$  feature had values between -1000 and 1000 and our  $y$  feature had values between -1 and 1, we would want to normalize those so that distance measures could be calculated accurately. In this case though, that's not necessary.

We can plot our data in a scatter plot in Figure 8-1 using `ggplot` to see the natural clustering that we'll try to replicate:

```

dataset %>% ggplot(aes(x = x, y = y, color = class)) +
  geom_point() +
  coord_fixed() +
  scale_shape_manual(values = c(0, 1, 2))

```



**Figure 8-1. Scatterplot of the RF data**

We can see in this visualization that the boundaries between clusters are not well defined, which will present an interesting challenge and basis for comparison with our clustering algorithms.

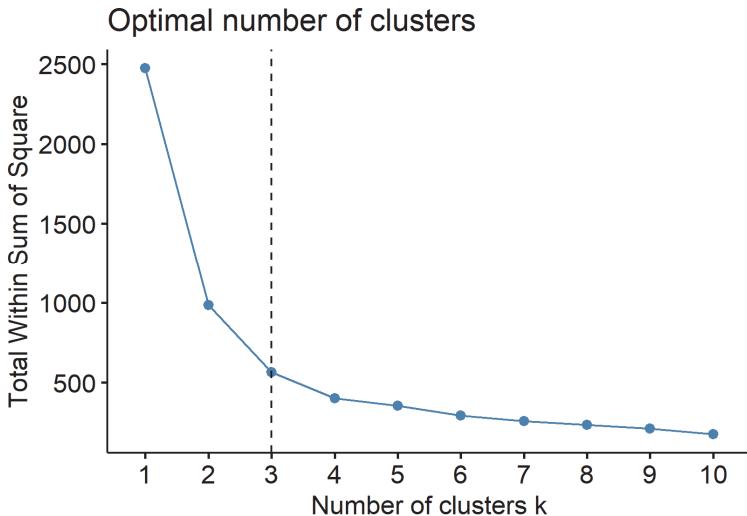
### 8.4.3 Step 3. Use K-means and medoids to cluster data

K-means clustering is one of the most commonly used unsupervised machine learning algorithms for partitioning a given data set into a set of  $k$  clusters, where  $k$  represents the number of groups pre-specified by the analyst. It classifies objects in multiple groups so that objects clustered together are as similar as possible, referred to as high intra-class similarity, and objects from different clusters are as dissimilar as possible, referred to as low inter-class similarity. In  $k$ -means, each cluster is represented by its centroid, which corresponds to the mean of points assigned to the cluster.

A fundamental question is how to determine the value of the parameter  $k$ . If we look at the percentage of variance explained as a function of the number of clusters, one should choose a number of clusters so that adding another cluster does not give much better modeling of the data. More precisely, if one plots the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variances), but at some point the marginal

gain will drop, giving an angle in the graph (see **Figure 8-2**). To check the optimal number of clusters, we can use the `fviz_nbclust` method.

```
fviz_nbclust(dataset, kmeans, method = "wss") +  
geom_vline(xintercept = 3, linetype = 2)
```



**Figure 8-2.** the percentage of variance explained by the clusters against the number of clusters

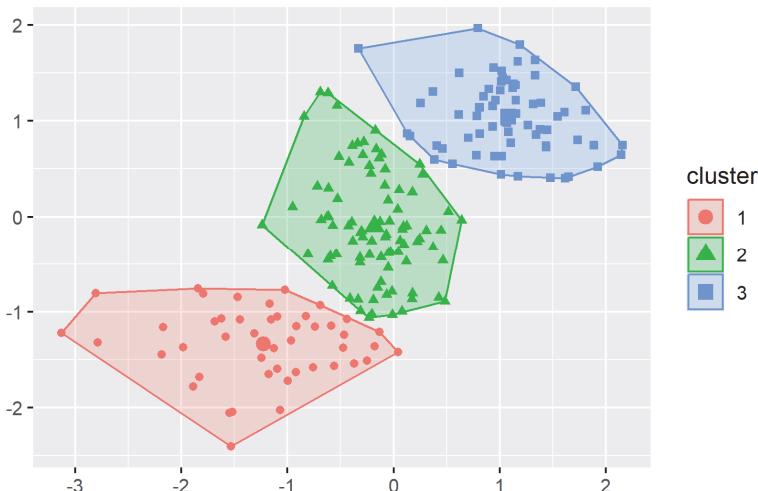
We want to look at the elbow of the plot where the within-cluster sum of squares drops significantly.

Now that we know the optimal number of clusters we can perform a k-means clustering. By default the call to `kmeans` will minimize the Euclidean distance between data points to define clusters.

```
k = 3  
kmeans_clustering <- kmeans(dataset[c("x", "y")],  
centers = k, nstart = 20)
```

We can use the `fviz_cluster()` method to view the results of our K-means clustering (see **Figure 8-3**).

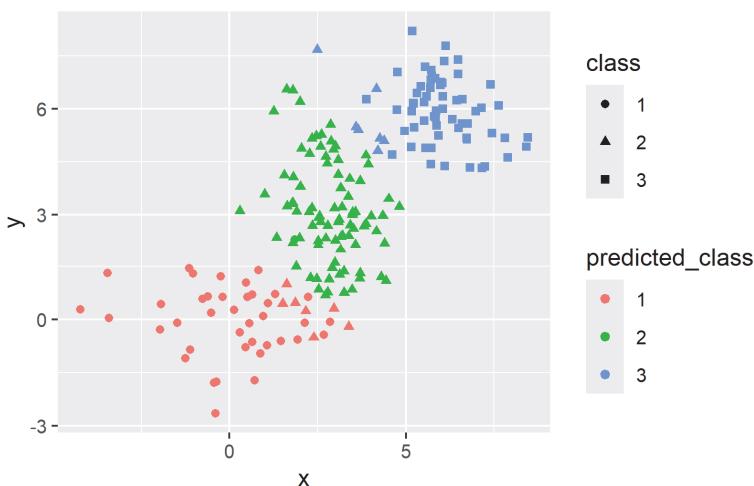
```
fviz_cluster(kmeans_clustering, dataset[c("x", "y")],  
xlab = FALSE, ylab = FALSE, geom = "point")
```



*Figure 8-3. Three distinct clusters, as expected by our “designed” data set.*

We can check the accuracy of our clustering with a plot in **Figure 8-4**:

```
kmeans_res <- dataset
kmeans_res['predicted_class'] =
  factor(kmeans_clustering$cluster)
kmeans_res %>% ggplot(aes(x = x, y = y, shape = class,
  color=predicted_class)) +
  geom_point() +
  coord_fixed() +
  scale_shape_manual(values = c(0, 1, 2)) +
  scale_shape(solid = TRUE)
```



*Figure 8-4. The figure shows good match, but we deigned dataset to do this*

We see that K-means clustering results match our data quite well, but this approach has some trouble with our clusters overlapping. This is to be expected since we generated a tricky dataset to use, and the cluster means don't fully capture the structure of the data. Since we know the labels, we can check the accuracy using a confusion matrix.

```
knn_conf_mat <-  
confusionMatrix(factor(kmeans_clustering$cluster),  
factor(dataset$class), mode = "everything", positive="1")
```

This gives us a confusion matrix and an accuracy score.

```
print(knn_conf_mat)
```

Confusion Matrix and Statistics

		Reference		
		1	2	3
Prediction	1	39	7	0
	2	1	86	0
	3	0	7	60

Overall Statistics

Accuracy : 0.925  
95% CI : (0.8793, 0.9574)  
No Information Rate : 0.5  
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8821

McNemar's Test P-Value : NA

Statistics by Class:

	Class: 1	Class: 2	Class: 3
Sensitivity	0.9750	0.8600	1.0000
Specificity	0.9563	0.9900	0.9500
Pos Pred Value	0.8478	0.9885	0.8955
Neg Pred Value	0.9935	0.8761	1.0000
Precision	0.8478	0.9885	0.8955
Recall	0.9750	0.8600	1.0000
F1	0.9070	0.9198	0.9449
Prevalence	0.2000	0.5000	0.3000
Detection Rate	0.1950	0.4300	0.3000
Detection Prevalence	0.2300	0.4350	0.3350
Balanced Accuracy	0.9656	0.9250	0.9750

Which returns:

**Accuracy** is perhaps the most basic evaluation metric for classification models, although it does not always offer a complete picture of model performance.

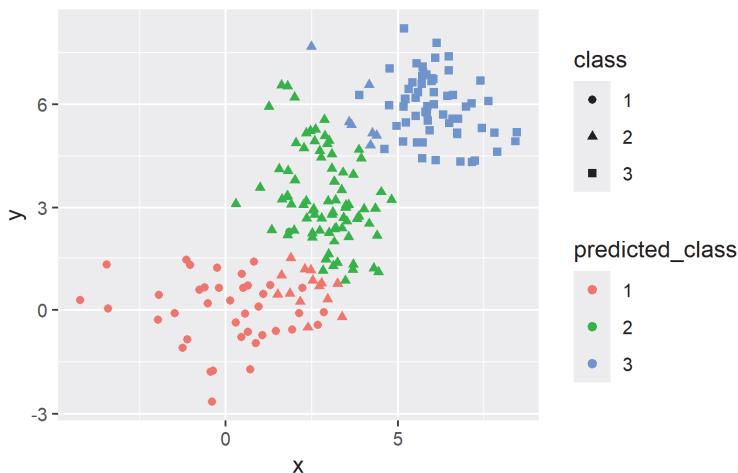
**Kappa** is also known as Cohen's kappa. This is another accuracy metric that indicates the level of agreement between the ground truth and predictions beyond the level of agreement resulting from chance.

For comparison now we can use medoids instead of using centroids with the `pam` clustering. Unlike the  $k$ -means algorithm, using medoids selects a median point to calculate the clusters rather than cluster centroids.

```
pam_k = 3  
pam_clustering <- pam(dataset[c("x", "y")], pam_k)
```

By inspection (*Figure 8-5*), we see that this performs slightly worse:

```
pam_res <- dataset  
pam_res['predicted_class'] <-  
  factor(pam_clustering$cluster)  
pam_res %>% ggplot(aes(x = x, y = y, shape = class,  
  color = predicted_class)) +  
  geom_point() +  
  coord_fixed() +  
  scale_shape_manual(values = c(0, 1, 2)) +  
  scale_shape(solid = TRUE)
```



*Figure 8-5. Medoids clustering does not fit as well as k-means*

Checking the accuracy with a confusion matrix gives us the following results:

```
pam_conf_mat <-  
confusionMatrix(factor(pam_clustering$cluster),  
factor(dataset$class), mode = "everything", positive="1")  
print(pam_conf_mat)
```

Confusion Matrix and Statistics

		Reference	
Prediction	1	2	3
1	39	14	0
2	1	79	0
3	0	7	60

Overall Statistics

Accuracy : 0.89  
95% CI : (0.8382, 0.9298)  
No Information Rate : 0.5  
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8299

McNemar's Test P-Value : NA

Statistics by Class:

	Class: 1	Class: 2	Class: 3
Sensitivity	0.9750	0.7900	1.0000
Specificity	0.9125	0.9900	0.9500
Pos Pred Value	0.7358	0.9875	0.8955
Neg Pred Value	0.9932	0.8250	1.0000
Precision	0.7358	0.9875	0.8955
Recall	0.9750	0.7900	1.0000
F1	0.8387	0.8778	0.9449
Prevalence	0.2000	0.5000	0.3000
Detection Rate	0.1950	0.3950	0.3000
Detection Prevalence	0.2650	0.4000	0.3350
Balanced Accuracy	0.9437	0.8900	0.9750

This shows that the medoids approach doesn't work quite as well as the centroids:

## 8.4.4 Step 4. Use Hierarchical clustering

We'll look at two types of hierarchical clustering:

Agglomerative clustering

Divisive clustering g

### 8.4.4.1 Agglomerative clustering

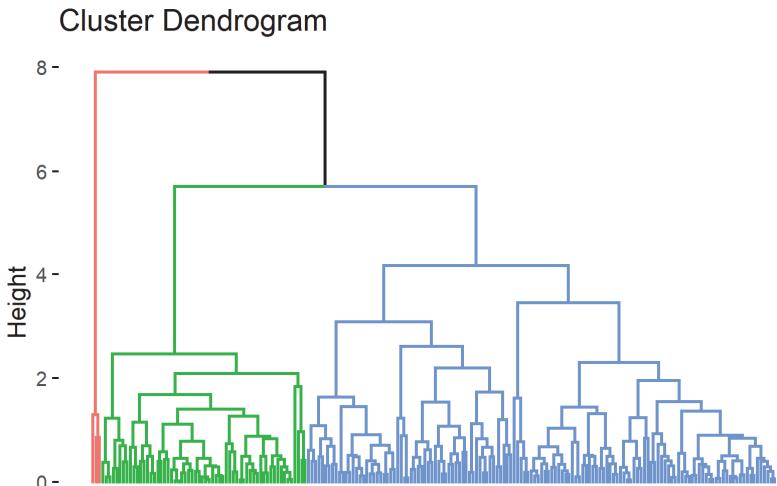
Agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on some similarity metric. This begins by treating each individual data point as a cluster and successively merging upwards until all clusters have been merged into one cluster at the base of the tree. The result is a tree-based representation of the objects called a dendrogram. Hierarchical clustering relies on maximizing the dissimilarity of each cluster but this can also make it more sensitive to outliers when performing segmentation.

One of the most commonly used is the `hclust()` method of the stats library. First we construct distance matrix of the distances for between each point using a Euclidean distance and then construct the hierarchical clustering using a call to `hclust`. Each datasets may perform differently with different distance measures but in this case a simple Euclidean distance will work as well as any other distance metric since the data has the same range along all features.

```
distances = dist(dataset[c("x", "y")], method =
  'euclidean')
agglomerative_clustering <- hclust(distances, method =
  'average')
```

Now we can visualize the resulting dendrogram in *Figure 8-6*:

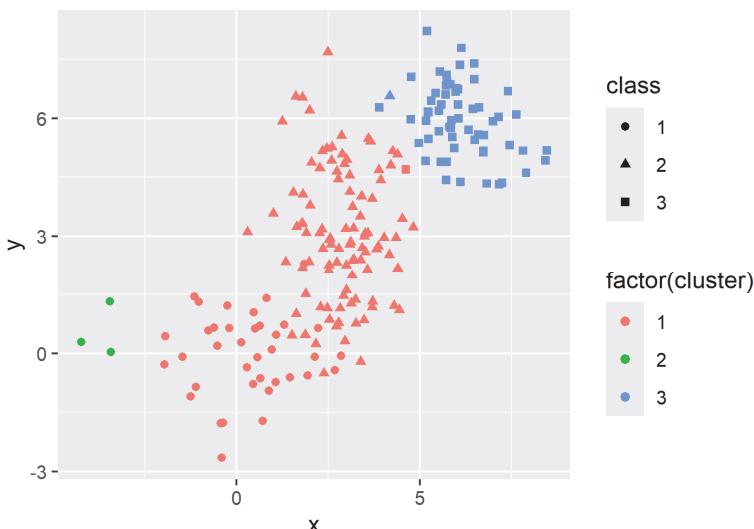
```
fviz_dend(agglomerative_clustering, cex = 0.5, k = 3,
color_labels_by_k = TRUE, show_labels = FALSE)
```



*Figure 8-6. Dendrogram of our designed data*

In this case though, the agglomerative clustering doesn't represent our data especially well as we can see in the dendrogram. We can compare the actual labels to the generated clusters in *Figure 8-7*:

```
cut_avg <- cutree(agglomerative_clustering, k = 3)
dataset_agg_cluster <- mutate(dataset, cluster = cut_avg)
ggplot(dataset_agg_cluster, aes(x=x, y = y, color = factor(cluster), shape=class)) + geom_point()
```



*Figure 8-7. Scatterplot of the cluster data grouped by the dendrogram groupings*

Again, we'll check the accuracy and Kappa coefficient with a confusion matrix:

```
ac_conf_mat <-  
confusionMatrix(factor(dataset_agg_cluster$cluster),  
factor(dataset_agg_cluster$class), mode = "everything",  
positive="1")  
print(ac_conf_mat)
```

#### Confusion Matrix and Statistics

		Reference		
Prediction		1	2	3
1	37	99	1	
2	3	0	0	
3	0	1	59	

#### Overall Statistics

Accuracy : 0.48  
95% CI : (0.409, 0.5516)  
No Information Rate : 0.5  
P-Value [Acc > NIR] : 0.7377

Kappa : 0.3207

McNemar's Test P-Value : <2e-16

#### Statistics by Class:

	Class: 1	Class: 2	Class: 3
Sensitivity	0.9250	0.0000	0.9833
Specificity	0.3750	0.9700	0.9929
Pos Pred Value	0.2701	0.0000	0.9833
Neg Pred Value	0.9524	0.4924	0.9929
Precision	0.2701	0.0000	0.9833
Recall	0.9250	0.0000	0.9833
F1	0.4181	NaN	0.9833
Prevalence	0.2000	0.5000	0.3000
Detection Rate	0.1850	0.0000	0.2950
Detection Prevalence	0.6850	0.0150	0.3000
Balanced Accuracy	0.6500	0.4850	0.9881

We can see that this has poor accuracy:

#### 8.4.4.2 Divisive clustering

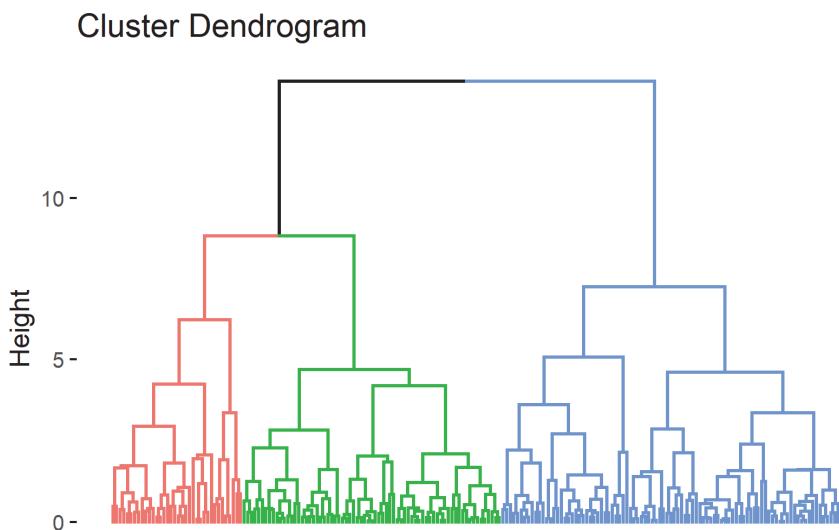
Now we'll look at a divisive clustering approach to our dataset. The inverse of agglomerative clustering is divisive clustering, which can be done with the `diana` function (**D**ivisive **A**Nalysis). This algorithm works in a top-down manner, beginning at the root representing all the data points in a single cluster. At each step of iteration, the most heterogeneous cluster is divided into two. The process is iterated until all objects are in their own cluster.

```
# compute divisive hierarchical clustering
divisive_clustering <- diana(dataset[c("x", "y")])
# Divide coefficient
print(divisive_clustering$dc)
```

```
[1] 0.973885
```

Now we can check the generated dendrogram in **Figure 8-8**:

```
#plot using a colors dendrogram to see our clusters with 3
groups
fviz_dend(divisive_clustering, cex = 0.5, k = 3,
color_labels_by_k = TRUE, show_labels = FALSE)
```



**Figure 8-8. Dendrogram for the designed data using divisive clustering**

When we check the confusion matrix, we can see that this structure represents our data much more accurately than the agglomerative clustering:

```
dc_conf_mat <-  
confusionMatrix(factor(cutree(divisive_clustering, k = 3)),  
factor(dataset$class), mode = "everything", positive="1")  
print(dc_conf_mat)
```

#### Confusion Matrix and Statistics

		Reference		
Prediction		1	2	3
1	1	35	1	0
	2	5	66	0
	3	0	33	60

#### Overall Statistics

Accuracy : 0.805  
95% CI : (0.7432, 0.8575)  
No Information Rate : 0.5  
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.6986

Mcnemar's Test P-Value : NA

#### Statistics by Class:

	Class: 1	Class: 2	Class: 3
Sensitivity	0.8750	0.6600	1.0000
Specificity	0.9938	0.9500	0.7643
Pos Pred Value	0.9722	0.9296	0.6452
Neg Pred Value	0.9695	0.7364	1.0000
Precision	0.9722	0.9296	0.6452
Recall	0.8750	0.6600	1.0000
F1	0.9211	0.7719	0.7843
Prevalence	0.2000	0.5000	0.3000
Detection Rate	0.1750	0.3300	0.3000
Detection Prevalence	0.1800	0.3550	0.4650
Balanced Accuracy	0.9344	0.8050	0.8821

The divisive approach performs much better than the agglomerative approach:

Next, we can look at an approach that should better match our underlying data process.

### 8.4.5 Step 5. Use Gaussian Mixed Models to cluster

Model-based clustering, which treats the data as though it is coming from a distribution which is a mixture of two or more clusters. Unlike k-means, the model-based clustering uses a soft assignment, where each data point has a probability of belonging to each cluster. Each cluster is modeled by the normal or Gaussian distribution which is described by the following parameters:

- $\mu_k$ : mean vector
- $\Sigma_k$ : covariance matrix
- An associated probability in the mixture. Each point has a probability of belonging to each cluster.

We can create a GMM clustering with the `Mclust` function in the `Mclust` package. Our data is roughly normalized but if the dimensions of our features are very different, we'll want to make sure that we normalize those features in our data preparation.

```
library(mclust)    # for fitting clustering algorithms  
  
# Create a GMM model  
dataset_mc <- Mclust(dataset[c('x', 'y')])  
summary(dataset_mc)
```

Gaussian finite mixture model fitted by EM algorithm

Mclust EVI (diagonal, equal volume, varying shape) model with 3 components:

log-likelihood	n	df	BIC	ICL
-805.8015	200	12	-1675.183	-1701.315

Clustering table:

1	2	3
97	39	64

After fitting the model we should look at a summary of the model using the `summary` method. We can see that the model inferred the correct number of clusters:

```
# Plot results
plot(dataset_mc, what = "density")
```

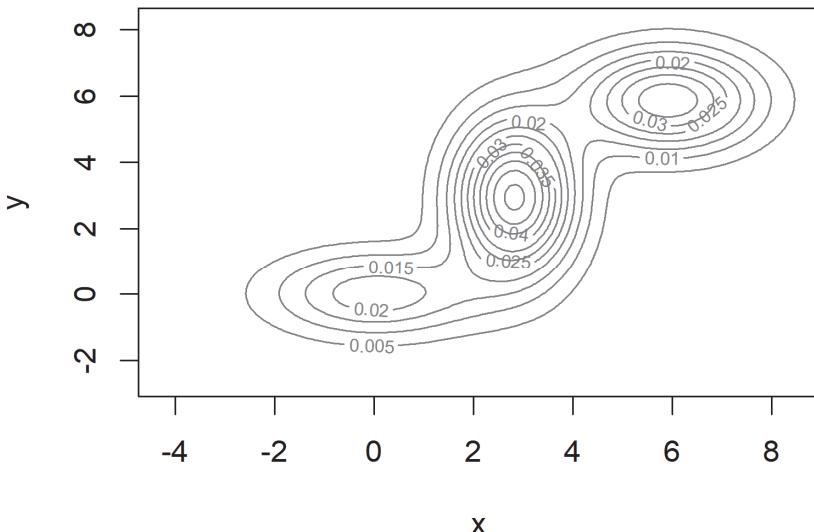


Figure 8-9. Contour plot showing the same three clusters

We can look at a confusion matrix to check how well our model fits the data. One aspect of this to note is that the cluster assignment generated by `mclust` may not match our original labels. To get a correct measure of accuracy, we'll recode the generated labels so that the cluster membership matches our original labels.

```
mc_recoded <- recode(dataset_mc$classification, '1' = '2',
                      '2' = '1', '3' = '3')
gmm_conf_mat <- confusionMatrix(factor(mc_recoded),
                                  factor(dataset$class), mode = "everything", positive="1")
print(gmm_conf_mat)
```

#### Confusion Matrix and Statistics

		Reference		
		1	2	3
Prediction	1	36	3	0
	2	4	93	0

```
3 0 4 60
```

#### Overall Statistics

```
Accuracy : 0.945
95% CI : (0.9037, 0.9722)
No Information Rate : 0.5
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9116
```

```
McNemar's Test P-Value : NA
```

#### Statistics by Class:

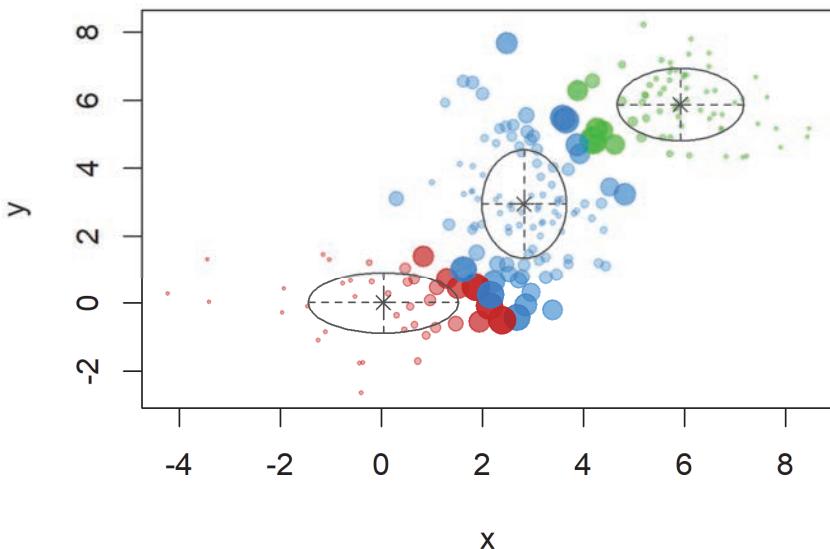
	Class: 1	Class: 2	Class: 3
Sensitivity	0.9000	0.9300	1.0000
Specificity	0.9812	0.9600	0.9714
Pos Pred Value	0.9231	0.9588	0.9375
Neg Pred Value	0.9752	0.9320	1.0000
Precision	0.9231	0.9588	0.9375
Recall	0.9000	0.9300	1.0000
F1	0.9114	0.9442	0.9677
Prevalence	0.2000	0.5000	0.3000
Detection Rate	0.1800	0.4650	0.3000
Detection Prevalence	0.1950	0.4850	0.3200
Balanced Accuracy	0.9406	0.9450	0.9857

The GMM approach shows fairly high accuracy and Kappa values:

Since our original data was constructed from three Gaussian mixtures, we shouldn't be surprised to see that this model-based approach reconstructs it quite well.

Our model does seem to have some trouble where the edges of the distributions representing our different clusters overlap with one another. One helpful feature of the GMM approach to clustering is that we can see the uncertainty of the model by plotting the uncertainty of the model, as shown in **Figure 8-10**.

```
plot(dataset_mc, what = "uncertainty")
```



*Figure 8-10. Pot of model uncertainty from the GMM approach*

This shows us which members of our discovered clusters the model is least certain about. In the graph, larger data points have more uncertainty, smaller have less. The GMM approach performs well with outlier points, but we can see that the overlapping areas of the distributions are difficult for our model to identify correctly, but this approach has the advantage of returning the uncertainty for each data point.

#### 8.4.6 Step 6. Compare the accuracy of each method

Now we can look at the results of each of our clustering methods:

```
comparison <- c(c(knn_conf_mat$overall[[1]],
  pam_conf_mat$overall[[1]],
  ac_conf_mat$overall[[1]],
  dc_conf_mat$overall[[1]],
  gmm_conf_mat$overall[[1]]),
  c(knn_conf_mat$overall[[2]],
  pam_conf_mat$overall[[2]],
  ac_conf_mat$overall[[2]],
  dc_conf_mat$overall[[2]],
  gmm_conf_mat$overall[[2]]))

comparison <- matrix(comparison, nrow = 2, byrow = TRUE)
```

```

colnames(comparison) =
  c('KNN', 'Medoids', 'Agglomerative', 'Divisive', 'GMM')
rownames(comparison) <- c('Accuracy', 'Kappa')

```

Now we can generate the table:

```
as.table(comparison)
```

	KNN	Medoids	Agglomerative	Divisive	GMM	
Accuracy	0.9250000	0.8900000		0.4800000	0.8050000	0.9450000
Kappa	0.8820755	0.8298531		0.3207054	0.6986090	0.9116466

We can see that the GMM approach performs slightly better than the KNN and that, with the data that we've generated, hierarchical approaches don't model the underlying data as well. That is much more a reflection of our data rather than a feature of hierarchical clustering. Finding the right approach to modeling your data will depend on your dataset and what you're looking to uncover by clustering.

## 8.5 Other Clustering Considerations

### 8.5.1 Choosing k for k-Means

A fundamental question is how to determine the value of the parameter  $k$ . If we look at the percentage of variance explained as a function of the number of clusters, one should choose a number of clusters so that adding another cluster does not give much better modeling of the data. More precisely, if one plots the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variance), but at some point the marginal gain will drop, giving an angle in the graph. At this point we choose the number of clusters, hence the “elbow criterion” (see **Figure 8-11**)

```

library(rattle)
data(wine)
head(wine)

```

Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	
1	14.23	1.71	2.43		15.6	127	2.80
2	13.20	1.78	2.14		11.2	100	2.65
3	13.16	2.36	2.67		18.6	101	2.80
4	14.37	1.95	2.50		16.8	113	3.85
5	13.24	2.59	2.87		21.0	118	2.80
6	14.20	1.76	2.45		15.2	112	3.27

	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue
1	3.06	0.28	2.29	5.64	1.04
2	2.76	0.26	1.28	4.38	1.05
3	3.24	0.30	2.81	5.68	1.03
4	3.49	0.24	2.18	7.80	0.86
5	2.69	0.39	1.82	4.32	1.04
6	3.39	0.34	1.97	6.75	1.05
	Dilution	Proline			
1	3.92	1065			
2	3.40	1050			
3	3.17	1185			
4	3.45	1480			
5	2.93	735			
6	2.85	1450			

The following code provides that plot weed need for the “optimal”  $k$  analysis.

```
df <- scale(wine[-1])
wssplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data,
      centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
    ylab="Within groups sum of squares")}
wssplot(df)
```

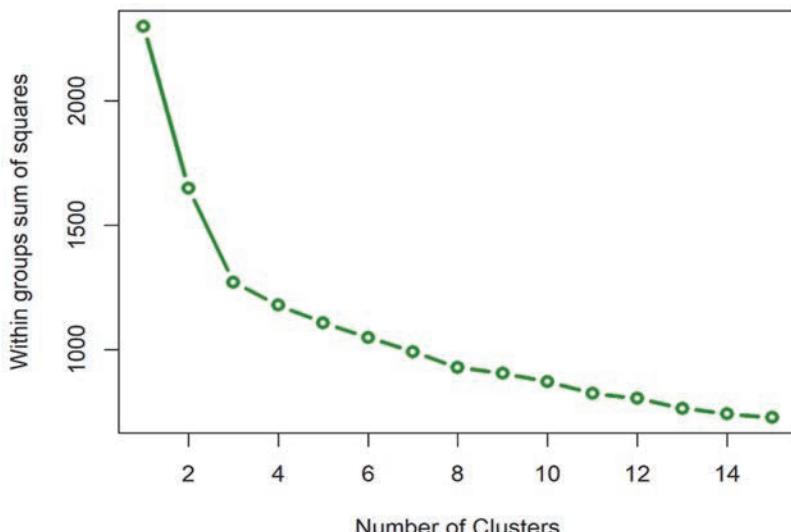
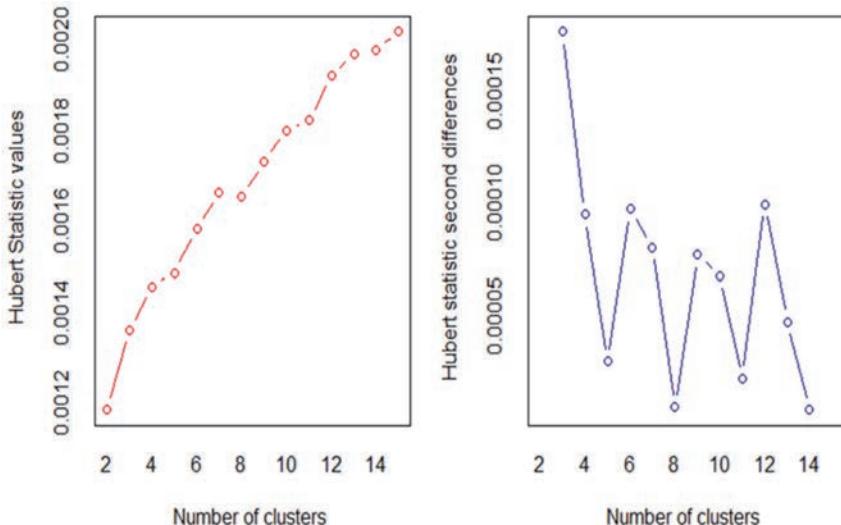


Figure 8-11. Number of clusters using the “elbow” method.

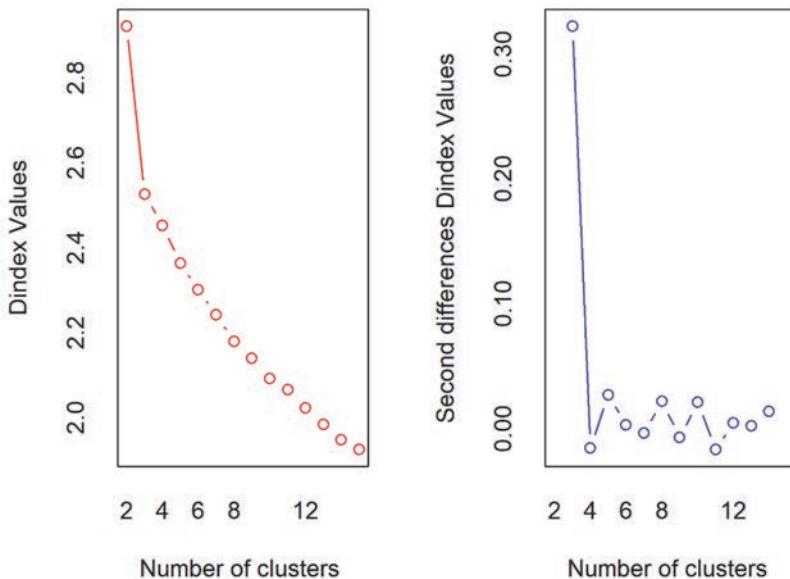
`NbClust` package provides 30 indices for determining the number of clusters and proposes to user the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering methods.

```
library(NbClust)
set.seed(1234)
nc <- NbClust(df, min.nc = 2, max.nc = 15,
method = "kmeans")
```



*Figure 8-12. The Hubert index is a graphical method of determining the number of clusters. In the plot of Hubert index, we seek a significant knee that corresponds to a significant increase of the value of the measure i.e. the significant peak in Hubert index second differences plot.*

The **Hubert index** is a graphical method of determining the number of clusters. In the plot of Hubert index, we seek a significant knee that corresponds to a significant increase of the value of the measure i.e. the significant peak in Hubert index second differences plot. The **D** index is a graphical method of determining the number of clusters. In the plot of **D** index, we seek a significant knee (the significant peak in **Dindex** second differences plot) that corresponds to a significant increase of the value of the measure. All 178 observations were used.



**Figure 8-13.** The  $D$  index is a graphical method of determining the number of clusters. In the plot of  $D$  index, we seek a significant knee (the significant peak in  $D$ index second differences plot) that corresponds to a significant increase of the value of the measure.

\* Among all indices:

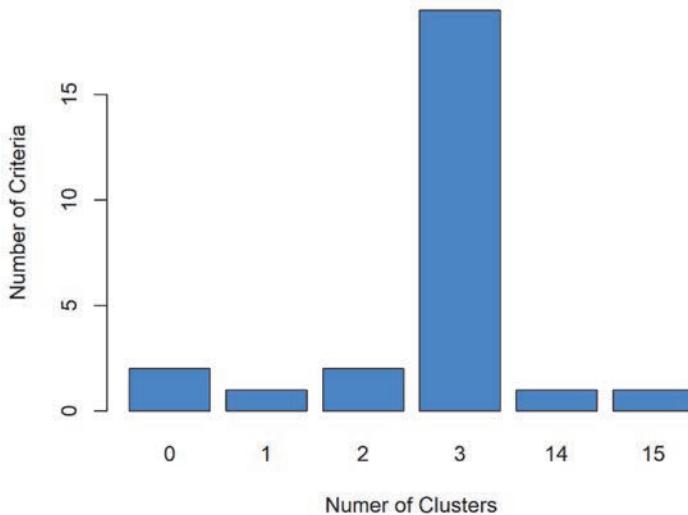
- \* 2 proposed 2 as the best number of clusters
- \* 19 proposed 3 as the best number of clusters
- \* 1 proposed 14 as the best number of clusters
- \* 1 proposed 15 as the best number of clusters

\* According to the majority rule, the best number of clusters is 3.

```
table(nc$Best.n[1,])
```

0	1	2	3	14	15
2	1	2	19	1	1

```
barplot(table(nc$Best.n[1,]), xlab = "Numer of Clusters",
ylab = "Number of Criteria", main = "Number of Clusters
Chosen by 26 Criteria", col = "dodgerblue")
```

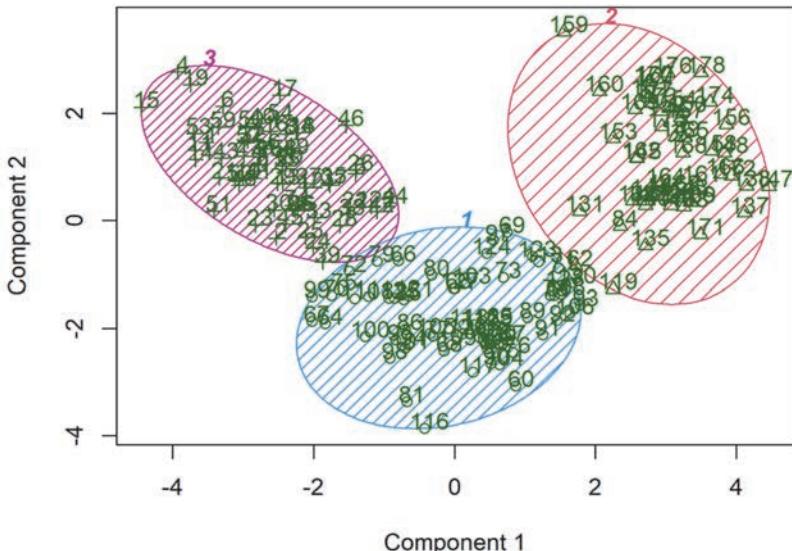


*Figure 8-14. Bar chart of number of clusters chosen by 26 criteria*

```
library(cluster)
set.seed(1234)
fit.km <- kmeans(df, 3, nstart=25)
fit.km$centers
```

	Alcohol	Malic	Ash	Proline
1	0.8328826	-0.3029551	0.3636801	1.1220202
2	-0.9234669	-0.3929331	-0.4931257	-0.7517257
3	0.1644436	0.8690954	0.1863726	-0.4059428
	Alcalinity	Magnesium	Phenols	Dilution
1	-0.6084749	0.57596208	0.88274724	0.7770551
2	0.1701220	-0.49032869	-0.07576891	0.2700025
3	0.5228924	-0.07526047	-0.97657548	-1.2887761
	Flavanoids	Nonflavanoids	Proanthocyanins	
1	0.97506900	-0.56050853	0.57865427	
2	0.02075402	-0.03343924	0.05810161	
3	-1.21182921	0.72402116	-0.77751312	
	Color	Hue		
1	0.1705823	0.4726504		
2	-0.8993770	0.4605046		
3	0.9388902	-1.1615122		

```
clusplot(wine, fit.km$cluster, color = TRUE, shade = TRUE,
labels = 2, lines = 0)
```



**Figure 8-15. Wine cluster plot – T These two components explain 57.38% of point variability**

## 8.6 Connectivity based clustering (hierarchical clustering)

Algorithms using **connectivity-based clustering** or hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away. These algorithms connect “items” to form “clusters” based on their distance from one another, using connectivity models. A cluster can be described largely by the maximum distance needed to connect parts of the cluster and at different distances, different clusters will form. We can represent these clusters using a **dendrogram**, hence the common name “hierarchical clustering”. These algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the  $y$ -axis marks the distance at which the clusters merge, while the objects are placed along the  $x$ -axis such that the clusters don’t mix.

We classify connectivity-based clustering by the way we compute distances. Apart from the usual choice of distance functions, we also need to decide on the linkage criterion. Since a cluster consists of multiple objects, there are multiple candidates to compute the distance. Popular choices include:

- Single-linkage clustering (the minimum of object distances)
- Complete linkage clustering (the maximum of object distances)
- Unweighted Pair Group Method with Arithmetic Mean (UPGMA).

There are two main types of strategies for hierarchical clustering:

- **Agglomerative** is a “bottom up” approach. Here each observation starts in its own cluster, and pairs of clusters merge as one moves up the hierarchy.
- **Divisive** is a “top down” approach. Here all observations start in one cluster, and we perform splits recursively as one moves down the hierarchy.

### 8.6.1 Metric

The choice we make for an appropriate metric will influence the shape of the clusters, as some elements may be close to one another according to one distance and farther away according to another. For example, in a 2-dimensional space, the distance between the point (1,0) and the origin (0,0) is always 1 according to the usual norms, but the distance between the point (1,1) and the origin (0,0) can be 2 under Manhattan distance,  $\sqrt{2}$  under Euclidean distance, or 1 under maximum distance. Some commonly used metrics for hierarchical clustering include (The DISTANCE Procedure: Proximity Measures, 2016a):

*Table 8-1. Formulas for distance computation in various cluster models*

Names	Formula
Euclidean distance	$\ a - b\ _2 = \sqrt{\sum_i (a_i - b_i)^2}$
Squared Euclidean distance	$\ a - b\ _2^2 = \sum_i (a_i - b_i)^2$
Manhattan distance	$\ a - b\ _1 = \sum_i  a_i - b_i $
maximum distance	$\ a - b\ _\infty = \max_i  a_i - b_i $
Mahalanobis distance	$\sqrt{(a - b)^\top S^{-1}(a - b)}$ where $S$ is the Covariance matrix

For text or other non-numeric data, we use metrics such as the **Hamming distance** or **Levenshtein distance**. A review of cluster analysis in health psychology research found that the most common distance measure in published studies in that research area is the Euclidean distance or the squared Euclidean distance

### 8.6.2 Linkage criteria

The linkage criterion determines the distance between sets of observations as a function of the pairwise distances between observations. Some commonly used linkage criteria between two sets of observations A and B are (The CLUSTER Procedure: Clustering Methods, 2016b):

*Table 8-2. Formulas or linkage types of cluster models*

Names	Formula
<b>Maximum or complete-linkage clustering</b>	$\max\{d(a, b) : a \in A, b \in B\}$
<b>Minimum or single-linkage clustering</b>	$\min\{d(a, b) : a \in A, b \in B\}$
<b>Mean or average linkage clustering, or UPGMA</b>	$\frac{1}{ A  B } \sum_{a \in A} \sum_{b \in B} d(a, b)$
<b>Centroid linkage clustering, or UPGMC</b>	$\ c_s - c_t\ $ where $c_s$ and $c_t$ are the centroids of clusters s and t, respectively.
<b>Minimum energy clustering</b>	$\begin{aligned} & \frac{2}{nm} \sum_{i,j=1}^{n,m} \ a_i - b_j\ _2 - \frac{1}{n^2} \sum_{i,j=1}^n \ a_i - a_j\ _2 \\ & - \frac{1}{m^2} \sum_{i,j=1}^m \ b_i - b_j\ _2 \end{aligned}$

where  $d$  is the chosen metric. Other linkage criteria include:

- The sum of all intra-cluster variance.
- The decrease in variance for the cluster being merged (Ward's criterion) (Ward, 1963).
- The probability that candidate clusters spawn from the same distribution function (V-linkage).
- The product of in-degree and out-degree on a k-nearest-neighbor graph (graph degree linkage) (Zhang, Graph degree linkage):

Agglomerative clustering on a directed graph, October 7–13, 2012).

- The increment of some cluster descriptor (i.e., a quantity defined for measuring the quality of a cluster) after merging two clusters (Zhang, 2013).

These methods will not produce a unique partitioning of the data set, but a hierarchy from which the user still needs to choose appropriate clusters. They are not very robust towards outliers, which will either show up as additional clusters or even cause other clusters to merge (known as “**chaining phenomenon**”, in particular with single-linkage clustering). In the general case, the complexity is  $\mathcal{O}(n^3)$ , which makes them too slow for large data sets. For some special cases, optimal efficient methods (of complexity  $\mathcal{O}(n^2)$ ) are known: **SLINK** (Sibson, 1973) for **single-linkage** and **CLINK** (Defays, 1977) for **complete-linkage** clustering. In the data mining community these methods are recognized as a theoretical foundation of cluster analysis, but often considered obsolete. They did, however, provide inspiration for many later methods such as density-based clustering.

### 8.6.3 Agglomerative Nesting (Hierarchical Clustering)

Here, we are using the `agnes` function computes agglomerative hierarchical clustering of the dataset. `votes.repub` is a data frame with the percent of votes given to the republican candidate in presidential elections from 1856 to 1976. Rows represent the 50 states, and columns the 31 elections (see **Figure 8-16**).

```
library(cluster)
data(votes.repub)
agn1 <- agnes(votes.repub, metric = "manhattan",
               stand = TRUE)
agn1
```

Call: `agnes(x = votes.repub, metric = "manhattan", stand = TRUE)`

Agglomerative coefficient: 0.7977555

Order of objects:

[1]	Alabama	Georgia	Arkansas	Louisiana
[5]	Mississippi	South Carolina	Alaska	Vermont
[9]	Arizona	Montana	Nevada	Colorado
[13]	Idaho	Wyoming	Utah	California
[17]	Oregon	Washington	Minnesota	Connecticut
[21]	New York	New Jersey	Illinois	Ohio

```
[25] Indiana      Michigan      Pennsylvania   New Hampshire
[29] Wisconsin    Delaware     Kentucky      Maryland
[33] Missouri     New Mexico   West Virginia Iowa
[37] South Dakota North Dakota Kansas       Nebraska
[41] Maine        Massachusetts Rhode Island Florida
[45] North Carolina Tennessee  Virginia     Oklahoma
[49] Hawaii       Texas
```

Height (summary):

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
8.382	12.804	18.528	23.118	28.411	87.455

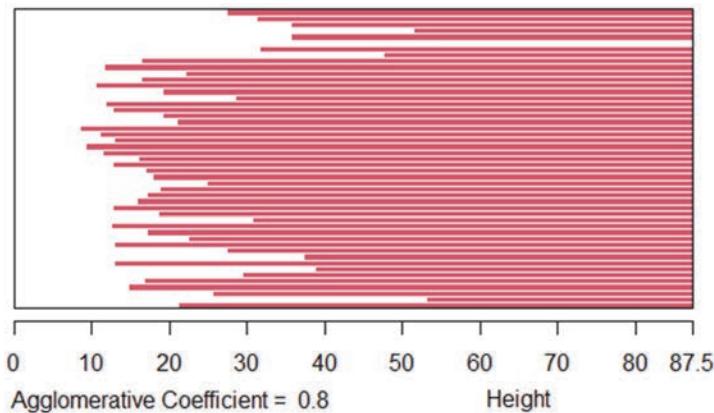
Available components:

```
[1] "order"      "height"      "ac"          "merge"      "diss"
[6] "call"       "method"     "order.lab"   "data"
```

#### 8.6.4 Plot with Parameters

We commonly refer to agglomerative clustering as AGNES (AGglomerative NESting), which uses a bottom-up approach. That is, the algorithm initially considers each observation as a single-element cluster (leaf). At each step, the algorithm combines the two clusters that are the most similar into a new bigger cluster (nodes). Here are some options for plotting parameters:  $\alpha = 0.625$  and  $\beta = -1/4$  is “recommended” by some experts. The code below yields clusters of Republican voters n presidential elections from 1856 to 1976.

```
agnS <- agnes(votes.repub, method="manhattan", stand = TRUE)
plot(agnS)
```



*Figure 8-16. Clusters the percent of votes given to the republican candidate in presidential elections from 1856 to 1976. Generated using the Manhattan metric.*

```

agn2 <- agnes(daisy(votes.repub), diss = TRUE, method =
"complete")
plot(agn2)

```

Dendrogram of `agnes(x = votes.repub, metric = "manhattan", stand = TRUE)`

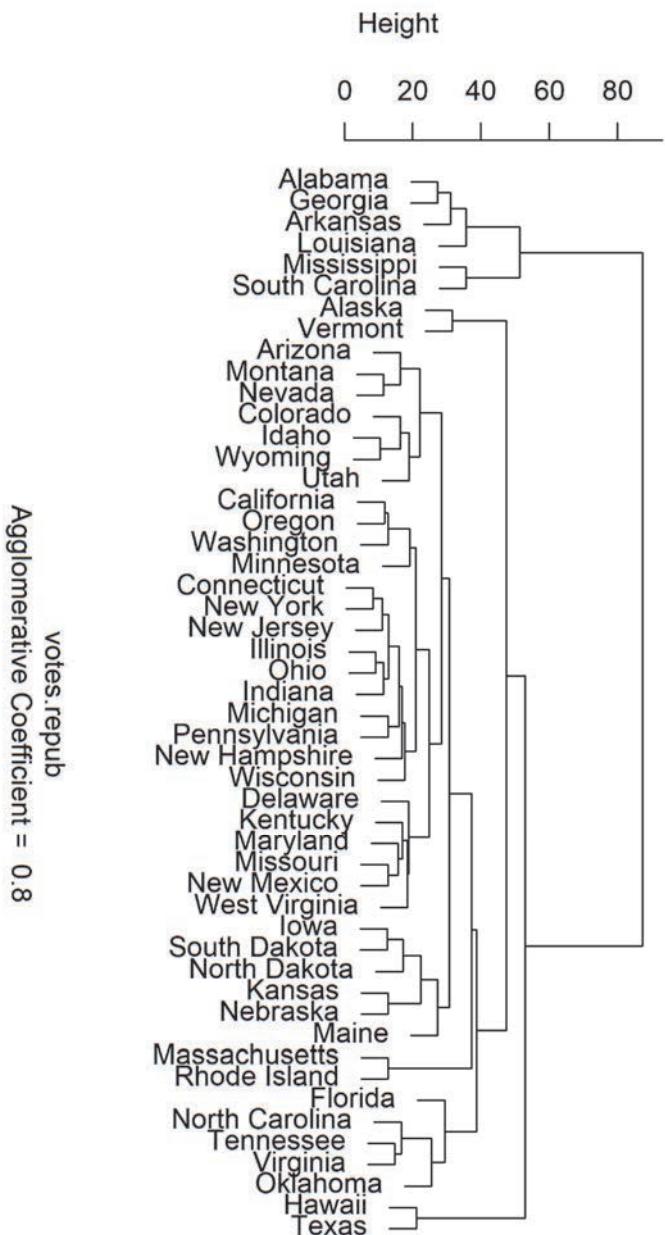


Figure 8-17. Dendrogram representation of clusters in Figure 8-16

Daisy computes all the pairwise dissimilarities (distances) between observations in the data set. The original variables may be of mixed types. In that case, or whenever metric = “gower” is set, a generalization of Gower’s formula is used.

```
d.vr <- daisy(votes.repub)
a.wgt <- agnes(d.vr, method = "weighted")
a.sing <- agnes(d.vr, method = "single")
a.comp <- agnes(d.vr, method = "complete")
iC <- -(6:7) # not using 'call' and 'method' for
comparisons
stopifnot(all.equal(a.wgt [iC], agnes(d.vr,
method="flexible",par.method = 0.5)[iC]),
all.equal(a.sing[iC], agnes(d.vr, method="flex",
par.method= c(.5,.5,0, -.5))[iC]),
all.equal(a.comp[iC], agnes(d.vr, method="flex",
par.method = c(.5,.5,0, +.5))[iC]))
```

The class “dendrogram” provides general functions for handling tree-like structures. It is intended as a replacement for similar functions in hierarchical clustering and classification/regression trees, such that all of these can use the same engine for plotting or cutting trees.

Two main branches “dendrogram” with 2 branches and 50 members total, at height 281.9508. The first branch ‘dendrogram’ with 2 branches and 8 members total, at height 116.7048. The 2nd one { 8 + 42 = 50 } ‘dendrogram’ with 2 branches and 42 members total, at height 178.4119. The first sub-branch of branch 1 .. and shorter form ‘dendrogram’ with 2 branches and 6 members total, at height 72.92212

```
(d2 <- as.dendrogram(agn2))
```

```
'dendrogram': 2 branches & 50 members total, at height 281.9508
d2[[1]] #
'dendrogram': 2 branches & 8 members total, at height 116.7048
d2[[2]] #
'dendrogram': 2 branches & 42 members total, at height 178.4119
d2[[1]][[1]]#
'dendrogram': 2 branches & 6 members total, at height 72.92212
```

```
identical(d2[[c(1,1)]], d2[[1]][[1]])
```

```
[1] TRUE
```

A “textual picture” of the dendrogram

```
str(d2)
```

```
--[dendrogram w/ 2 branches and 50 members at h = 282]
|--[dendrogram w/ 2 branches and 8 members at h = 117]
| |--[dendrogram w/ 2 branches and 6 members at h = 72.9]
| | |--[dendrogram w/ 2 branches and 3 members at h = 60.9]
| | | |--leaf "Alabama"
| | | |`--leaf "Georgia"
| | |`--leaf "Louisiana"
| |`-[dendrogram w/ 2 branches and 3 members at h = 58.8]
| | |--[dendrogram w/ 2 branches and 2 members at h = 56.1]
| | | |--leaf "Arkansas"
| | | |`--leaf "Florida"
| | |`--leaf "Texas"
| |`-[dendrogram w/ 2 branches and 2 members at h = 63.1]
| | |--leaf "Mississippi"
| | |`--leaf "South Carolina"
|`-[dendrogram w/ 2 branches and 42 members at h = 178]
| |--[dendrogram w/ 2 branches and 37 members at h = 121]
| | |--[dendrogram w/ 2 branches and 31 members at h = 80.5]
| | | |--[dendrogram w/ 2 branches and 17 members at h = 64.5]
| | | | |--[dendrogram w/ 2 branches and 13 members at h = 56.4]
| | | | | |--[dendrogram w/ 2 branches and 10 members at h = 47.2]
| | | | | |`-[dendrogram w/ 2 branches and 2 members at h = 28.1]
| | | | | | |--leaf "Alaska"
| | | | | | |`--leaf "Michigan"
| | | |`-[dendrogram w/ 2 branches and 8 members at h = 39.2]
| | | | |--[dendrogram w/ 2 branches and 5 members at h = 36.8]
| | | | | |--[dendrogram w/ 2 branches and 3 members at h = 32.9]
| | | | | |`-[dendrogram w/ 2 branches & 2 members at h = 19.4]
| | | | | | |--leaf "Connecticut"
| | | | | | |`--leaf "New York"
| | | | |`--leaf "New Hampshire"
| | | |`-[dendrogram w/ 2 branches and 2 members at h = 20.2]
| | | | |--leaf "Indiana"
| | | | |`--leaf "Ohio"
| | | |`-[dendrogram w/ 2 branches and 3 members at h = 25.3]
| | | | |--[dendrogram w/ 2 branches and 2 members at h = 20.9]
| | | | | |--leaf "Illinois"
| | | | | |`--leaf "New Jersey"
| | | | |`--leaf "Pennsylvania"
| | |`-[dendrogram w/ 2 branches and 3 members at h = 42.2]
| | | |--leaf "Minnesota"
| | | |`-[dendrogram w/ 2 branches and 2 members at h = 33.7]
| | | | |--leaf "North Dakota"
| | | | |`--leaf "Wisconsin"
| | |`-[dendrogram w/ 2 branches and 4 members at h = 37.5]
```

```
|--[dendrogram w/ 2 branches and 2 members at h = 26.2]
|  |--leaf "Iowa"
|  `--leaf "South Dakota"
`--[dendrogram w/ 2 branches and 2 members at h = 25.9]
|  |--leaf "Kansas"
`  `--leaf "Nebraska"
`-[dendrogram w/ 2 branches and 14 members at h = 70.5]
|  |--[dendrogram w/ 2 branches and 8 members at h = 48]
|  |  |--[dendrogram w/ 2 branches and 4 members at h = 43.4]
|  |  |  |--[dendrogram w/ 2 branches and 3 members at h = 27.8]
|  |  |  |  |--[dendrogram w/ 2 branches and 2 members at h = 23.4]
|  |  |  |  |  |--leaf "Arizona"
|  |  |  |  |  `--leaf "Nevada"
|  |  |  |  `--leaf "Montana"
|  |  `--leaf "Oklahoma"
`  |--[dendrogram w/ 2 branches and 4 members at h = 43.7]
|  |  |--leaf "Colorado"
`  |  `-[dendrogram w/ 2 branches and 3 members at h = 31.2]
|  |  |  |--[dendrogram w/ 2 branches and 2 members at h = 17.2]
|  |  |  |  |--leaf "Idaho"
|  |  |  |  `--leaf "Wyoming"
|  |  `--leaf "Utah"
`-[dendrogram w/ 2 branches and 6 members at h = 54.3]
|  |--[dendrogram w/ 2 branches and 3 members at h = 33.2]
|  |  |--leaf "California"
`  |  `-[dendrogram w/ 2 branches and 2 members at h = 22.2]
|  |  |  |--leaf "Oregon"
`  |  |  `--leaf "Washington"
`-[dendrogram w/ 2 branches and 3 members at h = 35.1]
|  |--[dendrogram w/ 2 branches and 2 members at h = 21.1]
|  |  |--leaf "Missouri"
`  |  `--leaf "New Mexico"
`  `--leaf "West Virginia"
`-[dendrogram w/ 2 branches and 6 members at h = 66.8]
|  |--[dendrogram w/ 2 branches and 3 members at h = 43.4]
|  |  |--leaf "Delaware"
`  |  `-[dendrogram w/ 2 branches and 2 members at h = 33.5]
|  |  |  |--leaf "Kentucky"
`  |  |  `--leaf "Maryland"
`-[dendrogram w/ 2 branches and 3 members at h = 30.2]
|  |--[dendrogram w/ 2 branches and 2 members at h = 29.5]
|  |  |--leaf "North Carolina"
`  |  `--leaf "Tennessee"
`  `--leaf "Virginia"
`-[dendrogram w/ 2 branches and 5 members at h = 83.1]
|  |--[dendrogram w/ 2 branches and 4 members at h = 55.4]
`  |--[dendrogram w/ 2 branches and 2 members at h = 32.8]
|  |  |--leaf "Hawaii"
```

```

|   |   '--leaf "Maine"
|   |   '--[dendrogram w/ 2 branches and 2 members at h = 22.6]
|   |       |--leaf "Massachusetts"
|   |       '--leaf "Rhode Island"
`--leaf "Vermont"

```

## 8.7 Density-based Clustering

Now, we look at a method and graphics, density-based clustering, that shows all the interactions of the iris variables (see **Figure 8-18**).

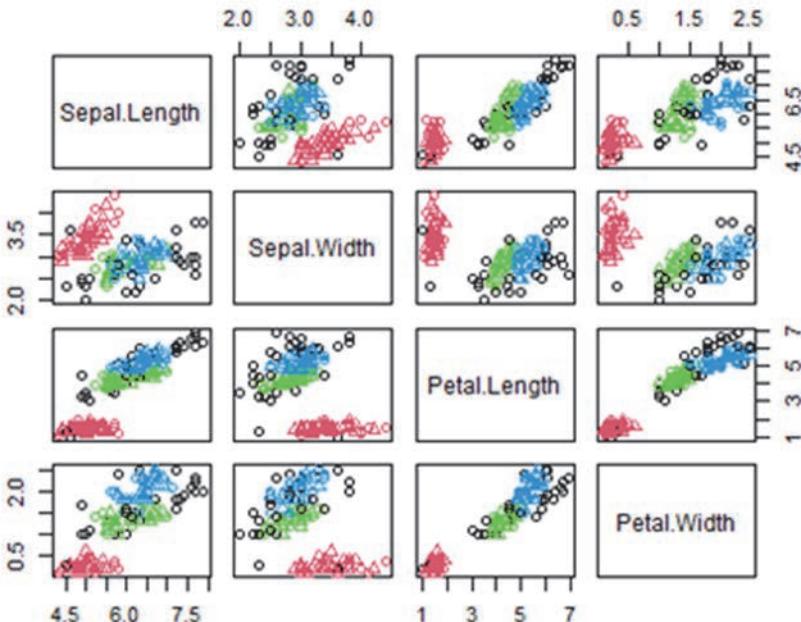
```

library(fpc)
iris2 <- iris[-5] # remove class tags
ds <- dbSCAN(iris2, eps = 0.42, MinPts = 5)
# compare clusters with original class labels
table(ds$cluster, iris$Species)

```

	setosa	versicolor	virginica
0	2	10	17
1	48	0	0
2	0	37	0
3	0	3	33

```
plot(ds, iris2)
```



*Figure 8-18. Iris correlation plot using DB clustering*

Here, we plot a simple scatter plot of the iris data.

```
plot(ds, iris2)
```

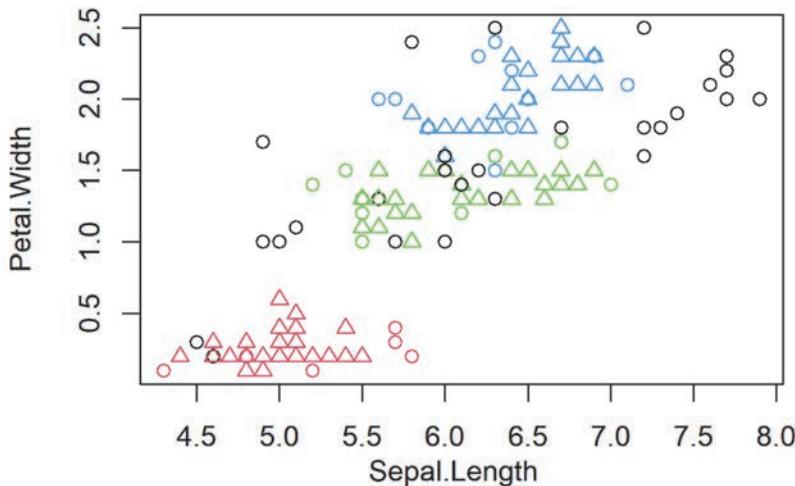


Figure 8-19 Scatterplot of sepal length vs. Sepal length

Next, we use the same data but with numbers appearing instead of shapes.

```
plotcluster(iris2, ds$cluster)
```

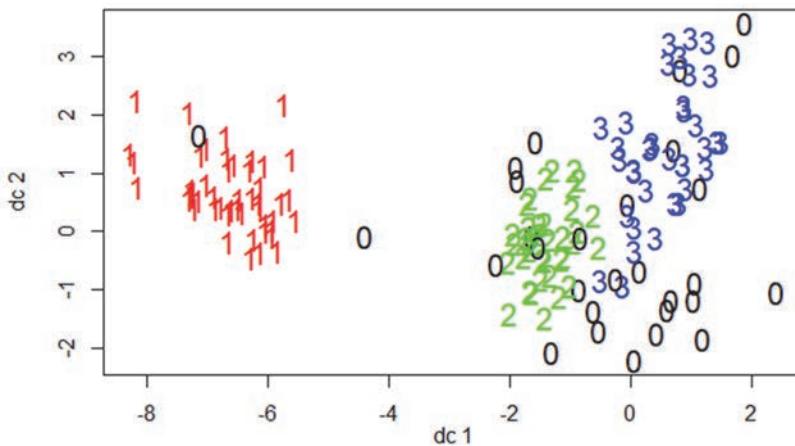


Figure 8-20. Scatterplot of sepal length vs. Sepal length using cluster group numbers

Now we will create a new dataset for labeling.

```
set.seed(435)
idx <- sample(1:nrow(iris), 10)
```

```

newData <- iris[idx, -5]
newData <- newData + matrix(runif(10*4, min = 0,
max = 0.2), nrow = 10, ncol = 4)

```

Next, we use model to make predictions.

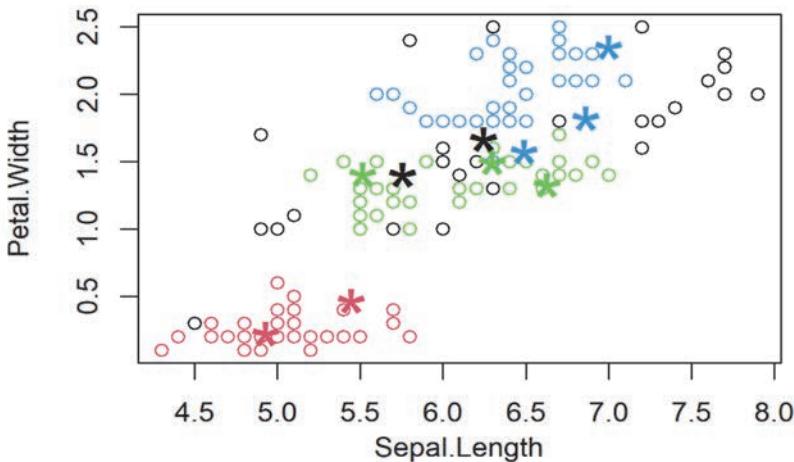
```
myPred <- predict(ds, iris2, newData)
```

Now, we plot results.

```

plot(iris2[c(1,4)], col = 1+ds$cluster)
points(newData[c(1,4)], pch = "*", col = 1+myPred, cex = 3)

```



*Figure 8-21. Iris 4-Cluster scatterplot with density-based clustering*

Finally, we check the cluster labels

```
table(myPred, iris$Species[idx])
```

myPred	setosa	versicolor	virginica
0	0	2	0
1	2	0	0
2	0	3	0
3	0	2	1

## 8.8 Cluster Model – Severe Weather Events and their Effects

### 8.8.1 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.

## 8.8.2 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.

## 8.8.3 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.

## 8.8.4 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`.

## 8.8.5 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.

```
my.path <- "C:/Users/jeff/Documents/Books/Prob&Stats/";
setwd(my.path);
fileUrl <-
"https://d396qusza40orc.cloudfront.net/reldata%2Fdata%2F
StormData.csv.bz2";
fileZip <- "./reldata-data-StormData.csv.bz2";
if (file.exists(fileZip) == F) {
  download.file(fileUrl, fileZip, mode = "wb")
}
#storm_data <-
read.csv("C:\\Users\\jeff\\Documents\\Data\\reldata_
StormData.csv.bz2", header = TRUE, sep=",")
storm_data <-
read.csv("C:\\Users\\jeff\\Documents\\Books\\Prob&Stats\\
\\reldata_data_StormData.csv.bz2", header = TRUE,
sep=",")
head(storm_data)
```

	STATE	BGN_DATE	BGN_TIME	TIME_ZONE	COUNTY			
1	1	4/18/1950	0:00:00	0130	CST	97		
2	1	4/18/1950	0:00:00	0145	CST	3		
3	1	2/20/1951	0:00:00	1600	CST	57		
4	1	6/8/1951	0:00:00	0900	CST	89		
5	1	11/15/1951	0:00:00	1500	CST	43		
6	1	11/15/1951	0:00:00	2000	CST	77		
	COUNTYNAME	STATE	EVTYPE	BGN_RANGE	BGN_AZI	BGN_LOCATI		
1	MOBILE	AL	TORNADO	0				
2	BALDWIN	AL	TORNADO	0				
3	FAYETTE	AL	TORNADO	0				
4	MADISON	AL	TORNADO	0				
5	CULLMAN	AL	TORNADO	0				
6	LAUDERDALE	AL	TORNADO	0				
	END_DATE	END_TIME	COUNTY_END	COUNTYENDN	END_RANGE	END_AZI		
1			0	NA	0			
2			0	NA	0			
3			0	NA	0			
4			0	NA	0			
5			0	NA	0			
6			0	NA	0			
	END_LOCATI	LENGTH	WIDTH	F	MAG	FATALITIES	INJURIES	PROPDMG
1		14.0	100	3	0	0	15	25.0
2		2.0	150	2	0	0	0	2.5

3		0.1	123	2	0	0	2	25.0
4		0.0	100	2	0	0	2	2.5
5		0.0	150	2	0	0	2	2.5
6		1.5	177	2	0	0	6	2.5
	PROPDMGEXP	CROPDMG	CROPDMGEXP	WFO	STATEOFFIC	ZONENAMES		
1	K	0						
2	K	0						
3	K	0						
4	K	0						
5	K	0						
6	K	0						
	LATITUDE	LONGITUDE	LATITUDE_E	LONGITUDE_	REMARKS	REFNUM		
1	3040	8812	3051	8806		1		
2	3042	8755	0	0		2		
3	3340	8742	0	0		3		
4	3458	8626	0	0		4		
5	3412	8642	0	0		5		
6	3450	8748	0	0		6		

## 8.8.6 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 reference (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"
[4] "TIME_ZONE"    "COUNTY"       "COUNTYNAME"
[7] "STATE"        "EVTYPE"       "BGN_RANGE"
[10] "BGN_AZI"      "BGN_LOCATI"   "END_DATE"
[13] "END_TIME"     "COUNTY_END"   "COUNTYENDN"
[16] "END_RANGE"    "END_AZI"      "END_LOCATI"
[19] "LENGTH"       "WIDTH"        "F"
[22] "MAG"          "FATALITIES"  "INJURIES"
[25] "PROPDMG"      "PROPDMGEXP"  "CROPDMG"
[28] "CROPDMGEXP"   "WFO"         "STATEOFFIC"
[31] "ZONENAMES"    "LATITUDE"    "LONGITUDE"
[34] "LATITUDE_E"   "LONGITUDE_" "REMARKS"
[37] "REFNUM"
```

## 8.8.7 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 Reference (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 example computational efficiency and removes detractors.

### 8.8.8 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.

```
storm1 <- storm_data[,c("EVTYPE", "FATALITIES",
"INJURIES")]
head(storm1,15)%>%
  kbl(col.names = c("Event Type", "Fatalities",
"Injuries"), align = "c",
  caption = "Table 1. Severe Weather Data Affecting
Population Health") %>%
  kable_classic(full_width = F,
html_font = "Cambria", "striped") %>%
  column_spec(2, width = "10em") %>%
  column_spec(3, width = "10em")
```

*Table 8-3. Severe Weather Data Affecting Population Health*

Event Type	Fatalities	Injuries
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 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 `storm2`. Again, we examine the structure and data summary.

```
storm2 <- storm_data[,c("EVTYPE", "PROPDMG",
"PROPDMGEXP",
"CROPDMG", "CROPDMGEXP")]
head(storm2,15)%>%
  kbl(col.names = c("Event Type", "Property Damage",
"Property Exponent", "Crop Damage", "Crop
Exponent"), align = "c", caption = "Table 2.
Severe Weather Data Affecting Population Health")
%>% kable_classic(full_width = F,
html_font = "Cambria", "striped")
```

Table 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
TORNADO	2.5	M	0
TORNADO	2.5	M	0
TORNADO	250.0	K	0
TORNADO	0.0	K	0
TORNADO	25.0	K	0

```
print(str(storm2))
```

```
'data.frame': 902297 obs. of  5 variables:
$ EVTYPE   : chr  "TORNADO" "TORNADO" "TORNADO" "TORNADO" ...
$ 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 0 ...
$ CROPDMGEXP: chr  "" "" "" "" ...
NULL
```

```
print(summary(storm2))
```

EVTYPE	PROPDMG	PROPDMGEXP
Length:902297	Min. : 0.00	Length:902297
Class :character	1st Qu.: 0.00	Class :character
Mode :character	Median : 0.00	Mode :character
	Mean : 12.06	
	3rd Qu.: 0.50	
	Max. :5000.00	
CROPDMG	CROPDMGEXP	
Min. : 0.000	Length:902297	
1st Qu.: 0.000	Class :character	
Median : 0.000	Mode :character	
Mean : 1.527		
3rd Qu.: 0.000		
Max. :990.000		

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.

## 8.8.9 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.

```
storm1$FATALITIES = as.numeric(storm1$FATALITIES)
fatalities<-aggregate(FATALITIES~EVTYPE,storm1,sum)
storm1$INJURIES = as.numeric(storm1$INJURIES)
injuries<-aggregate(INJURIES~EVTYPE,storm1,sum)
harm <- fatalities$FATALITIES + injuries$INJURIES
maxharm <- max(harm)
maxharm
```

```
[1] 96979
```

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,]
top_7_fatality
```

	EVTYPE	FATALITIES
1	TORNADO	5633
2	EXCESSIVE HEAT	1903
3	FLASH FLOOD	978
4	HEAT	937
5	LIGHTNING	816
6	TSTM WIND	504
7	FLOOD	470

	EVTYPE	INJURIES
1	TORNADO	91346
2	TSTM WIND	6957
3	FLOOD	6789
4	EXCESSIVE HEAT	6525
5	LIGHTNING	5230
6	HEAT	2100
7	ICE STORM	1975

### 8.8.10 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, PROPDGEXP, and crop damage field, CROPDMGEXP. 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 PROPDGEXP and CROPDMGEXP. field.

```
prop <- storm_data$PROPDG[  
  which(storm_data$PROPDGEXP == "B")]  
prop = as.numeric(prop)  
sum(prop)
```

```
[1] 275.85
```

```
max(prop)
```

```
[1] 115
```

```
crop <- storm_data$CROPDMG[  
  which(storm_data$CROPDMGEXP == "B")]  
crop = as.numeric(crop)  
sum(crop)
```

```
[1] 13.61
```

```
max(crop)
```

```
[1] 5
```

```
a <- max(prop)  
b <- max(crop)  
econ <- storm_data$EVTYPE[which(  
  (storm_data$PROPDG == a) &  
  (storm_data$PROPDGEXP == "B"))]  
econ
```

```
[1] "FLOOD"
```

```
storm2 <- storm_data[,c("EVTYPE", "PROPDG", "PROPDGEXP",  
  "CROPDMG", "CROPDMGEXP")]
```

```
#tail(storm2)
#unique(storm2$PROPDMGEXP)
#unique(storm2$CROPDMGEXP)
storm2
```

	EVTYPE	PROPDMG	PROPDMGEXP	CROPDMG	CROPDMGEXP
1	TORNADO	25.00		K	0
2	TORNADO	2.50		K	0
3	TORNADO	25.00		K	0
4	TORNADO	2.50		K	0
5	TORNADO	2.50		K	0
6	TORNADO	2.50		K	0
7	TORNADO	2.50		K	0
8	TORNADO	2.50		K	0
9	TORNADO	25.00		K	0
10	TORNADO	25.00		K	0
11	TORNADO	2.50		M	0
12	TORNADO	2.50		M	0
19989	TSTM WIND	0.00			0
19990	HAIL	0.00			0
19991	HAIL	0.00			0
19992	HAIL	0.00			0
19993	HAIL	0.00			0
19994	HAIL	0.00			0
19995	HAIL	0.00			0
19996	TORNADO	2.50		M	0
19997	TORNADO	25.00		K	0
19998	TORNADO	0.25		K	0
19999	TSTM WIND	0.00			0
					[ reached 'max' / getOption("max.print") -- omitted 882298 rows ]

```
unique(storm2$PROPDMGEXP)
```

```
[1] "K" "M" "" "B" "m" "+" "0" "5" "6" "?" "4" "2" "3" "h"
"7" "H" "-" "1" "8"
```

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.

### 8.8.11 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$PROPDGMGEXP))
# property damage levels
propMult <-
c(1,1,1,1,1,10,100,1000,10000,100000,1000000,10000000,
# property damage multipliers
100000000,1000000000,100,100,1000,1000000,10000000)
propLookup <- data.frame(cbind(PROPDMGEXP, propMult))
# propLookup table
propLookup$propMult <-
as.numeric(as.character(propLookup$propMult))
# convert to numeric
CROPDMGEXP <- sort(unique(storm2$CROPDMGEXP))
# crop damage levels
cropMult <-
c(1,1,1,100,100000000,1000,1000,1000000,10000000)
# crop damage multipliers
cropLookup <- data.frame(cbind(CROPDMGEXP, cropMult))
# cropLookup table
cropLookup$cropMult <-
as.numeric(as.character(cropLookup$cropMult))
# convert to numeric
```

## 8.8.12 Merge Lookup Tables into Storm Data

With the next code chunk, we merge property damage multiplier, `propMult`, and crop damage multiplier, `cropMult`, with sever weather (`storm2`) data set.

```
storm2 <- merge(storm2,propLookup)
# merge propMult into data set
storm2 <- merge(storm2,cropLookup)
# merge cMultiplier into data set
```

For reference, we generate `kable` tables 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, html_font = "Cambria",
"striped") %>%
  column_spec(2, width = "18em")
```

*Table 8-4. Property Damage Exponent Codes with Values*

EXONENT	VALUE
	1e+00
-	1e+00
?	1e+00
+	1e+00
0	1e+00
1	1e+01
2	1e+02
3	1e+03
4	1e+04
5	1e+05
6	1e+06
7	1e+07
8	1e+08

```
cropLookup%>%
  kbl(col.names = c("EXONENT", "VALUE"),
  caption = "Table 4. Crop Damage Exponent Codes with
Values") %>%
  kable_classic(full_width = F, html_font = "Cambria",
"striped") %>%
  column_spec(2, width = "18em")
```

*Table 8-5. Crop Damage Exponent Codes with Values*

EXONENT	VALUE
Z	1e+10
?	1e+10
0	1e+00
2	1e+20
B	1e+09
k	1e+03
K	1e+03
n	1e+06
M	1e+06

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, ]
```

## 8.9 Results

We are now ready to present and discuss the results of our analysis questions.

### 8.9.1 Public Health - Fatalities

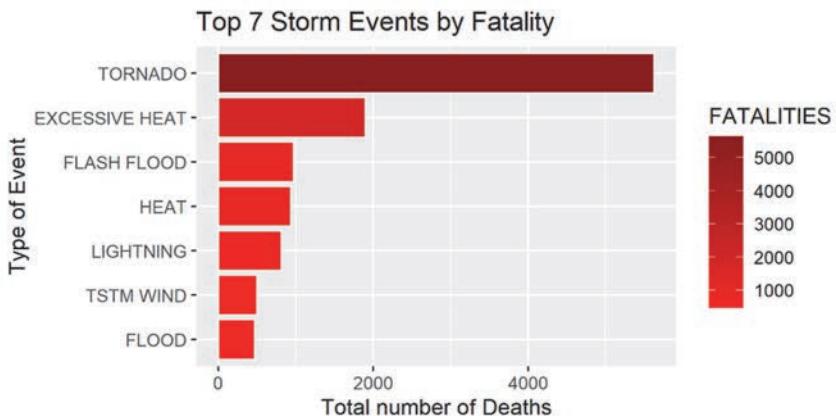
First, we generate a barplot with Fatality results, as seen in *Figure 8-22*.

```
f1<-ggplot(top_7_fatality,
aes(x=reorder(EVTYPE, FATALITIES),
```

```

    = FATALITIES, fill = FATALITIES))
f1 + geom_bar(stat = 'identity', color = 'white')+
  scale_fill_gradient(low = "red", high = "red4")+
  ggtitle('Top 7 Storm Events by Fatality')+
  xlab('Type of Event')+
  coord_flip()+
  ylab('Total number of Deaths')

```



*Figure 8-22. Top 7 Storm Events by 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,
  html_font = "Cambria", "striped") %>%
  column_spec(2, width = "18em")

```

*Table 8-6. Top Seven Storm Type Affecting Public Health (Fatalities)*

Event Type	Fatalities
TORNADO	5633
EXCESSIVE HEAT	1903
FLASH FLOOD	978
HEAT	937
LIGHTNING	816
TSTM WIND	504
FLOOD	470

**Figure 8-22** 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 Reference (NWS, 2021), the big distinction is “well above normal” versus “above normal” heat and high humidity:

**“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 (NWS, 2021): 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. The results are summarized in **Table 8-7**.

```
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, html_font = "Cambria",
    "striped") %>%
  column_spec(2, width = "18em")
```

*Table 8-7. Top Seven Storm Type Affecting Public Health (Fatalities)*

Event Type	Fatalities
TORNADO	5633
EXCESSIVE HEAT	1903
FLASH FLOOD	978
HEAT	937
LIGHTNING	816
TSTM WIND	504
FLOOD	470

## 8.9.2 Public Health - Injuries

Next, we deal with injuries caused by severe weather events. As before, we summarize the effects with a table, followed by a horizontal barplot.

```
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, html_font = "Cambria",
    "striped") %>%
  column_spec(2, width = "20em")
```

*Table 8-8. 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

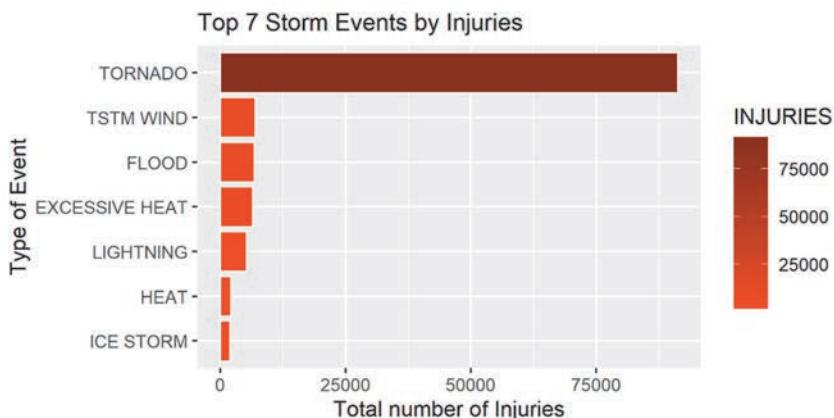
The outcome is very similar to fatality results. One exception is the sheer scale of tornado related injuries. There are more than 75,000 more injuries caused by tornado than any other event.

Added to the injury results are ice storms, defined as:

**"Ice Storm (Z).** Ice accretion meeting or exceeding local/regional defined warning criteria (typical value is 1/4 or 1/2 inch or more)."

The barplot summarizes these results visually.

```
f2 <- ggplot(top_7_injury,
  aes(x=reorder(EVENTTYPE, INJURIES),
      y=INJURIES, fill=INJURIES))
f2 + geom_bar(stat='identity',color='white')+
  scale_fill_gradient(low = "orangered",
    high = "orangered4")+
  ggtitle('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 8-23. Top 7 Storm Events by Personal Injuries*

### 8.9.3 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 bar chart.

```
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, html_font = "Cambria",
"striped") %>%
  column_spec(2, width = "20em")
```

*Table 8-9. Top Seven Storm Type Affecting Property Damage*

Num.	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 than tornado. It is more common to hear that the Mississippi River went over its banks (and 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 (NWS, 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.

```
# 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")+
  ggtitle('Top 7 Storm Events by property damage')+ 
  xlab('Type of Event')+ 
  coord_flip()+
  ylab('Total Property Damage Cost(USD)')

# Crop Damage
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")+
  ggtitle('Top 7 Storm Events by crop damage')+ 
  xlab('Type of Event')+ 
  coord_flip()+
  ylab('Total CROP Damage Cost(USD)')

grid.arrange(d1, d2)
```

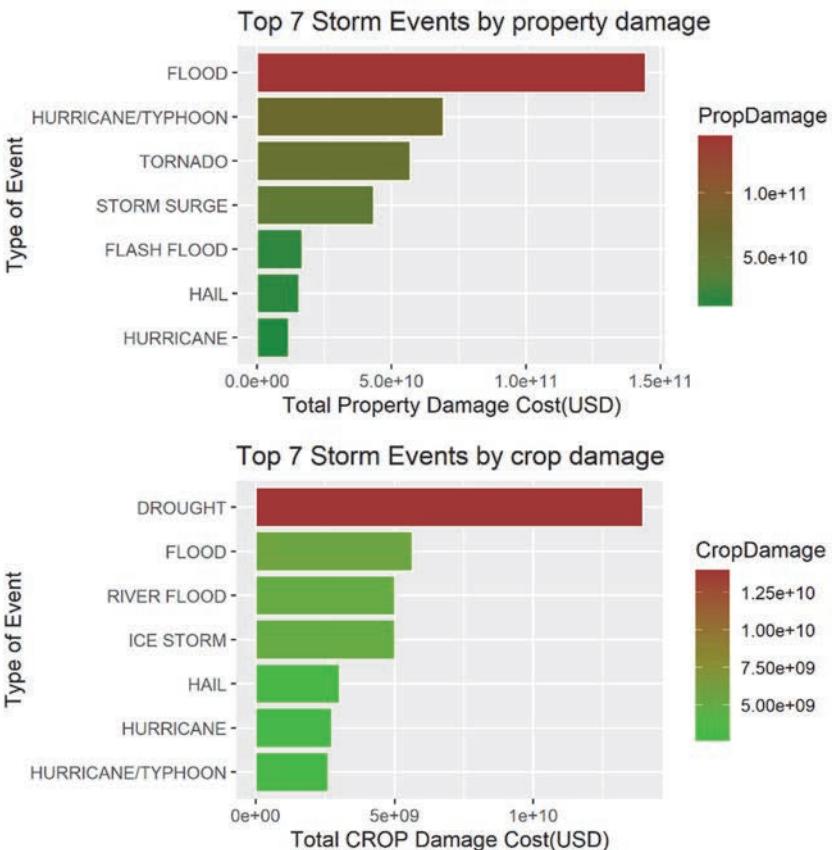


Figure 8-24. Top 7 storm events by property damage (top) and crop damage (bottom)

#### 8.9.4 Crop Damage

Here again, we generate a similar table with the crop damage resulting from severe weather, with an accompanying horizontal bar chart. The bottom barplot in **Figure 8-24** conveys this information visually.

```
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, html_font = "Cambria",
"striped") %>%
  column_spec(2, width = "20em")
```

Table 8. Top Seven Storm Type Affecting Crop Damage

Num	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

For this category, drought rules the day as is intuitive. The difference between flash floods and floods is clear, but what about river flooding. Reference (NOAA, 2021) has no separate category for river floods and states:

“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.

## 8.10 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.



## References

- Bailey, K. (1994). Numerical Taxonomy and Cluster Analysis. In *Typologies and Taxonomies* (p. 34).
- Becker, G., & Barro, R. (1988). A Reformulation of the Economic Theory of Fertility. *The Quarterly Journal of Economics*, 103(1), 1-25.
- Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.
- Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24, 123-140.
- Breiman, L. (2001a). Random Forests. *Machine Learning*, 45, 5–32.  
doi:<https://doi.org/10.1023/A:1010933404324>
- Breiman, L. (2001b). Statistical Modeling: The Two Cultures. *Statistical Science*, 16(3), 199-231.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
- Cattell, R. B. (1943). The description of personality: Basic traits resolved into clusters. *Journal of Abnormal and Social Psychology*, 38, 476–506. doi:[doi:10.1037/h0054116](https://doi.org/10.1037/h0054116)
- Chambers, J. M. (1992). Data for models. In J. M. Chambers, & T. J. Hastie, *Statistical Models in S*. Wadsworth & Brooks/Cole.
- Cleveland, W. (1981). LOWESS: A program for smoothing scatter plots by robust locally weighted regression. *The American Statistician*, 35(54). doi:<http://dx.doi.org/10.2307/2683591>
- Cleveland, W. S. (1993). *Visualizing Data*. New Jersey: Summit Press.
- Defays, D. (1977). An efficient algorithm for a complete link method. *The Computer Journal (British Computer Society)*, 20(4), 364–366.  
doi:[doi:10.1093/comjnl/20.4.364](https://doi.org/10.1093/comjnl/20.4.364)
- Estivill-Castro, V. (2002, June 20). Why so many clustering algorithms — A Position Paper. *ACM SIGKDD Explorations Newsletter*, 4(1), 65–75. doi:[doi:10.1145/568574.568575](https://doi.org/10.1145/568574.568575)

- Fleming, T. R., & Harrington, D. P. (1991). *Counting Processes and Survival Analysis*. Hoboken, New Jersey: John Wiley & Sons, Inc.
- Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29, 1189-1232.
- Friedrich-Rust, M., W., R., J., P., E., H., S., Z., & C., S. (2017). *Clinical Review Report: Indication - For the treatment of primary biliary cholangitis (PBC) in combination with ursodeoxycholic acid (UDCA) in adults with an inadequate response to UDCA*. Retrieved from The National Institutes of Health: <https://www.ncbi.nlm.nih.gov/books/NBK534940/table/cl.app.4.table11/>
- Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5, 299-314.
- Ika, S. B. (2017, January 17). CSAF talks future attributes of warfare during 2017 WEPTAC. *Nellis Air Force News*, p. Online. Retrieved from <https://www.nellis.af.mil/News/Article-Display/Article/1051279/csaf-talks-future-attributes-of-warfare-during-2017-weptac/>
- 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: with Applications in R*. New York: Springer.
- Krzywinski, M., & Altman, N. (2017). Classification and regression trees. *Nature Methods*, 14, 757–758. doi:<https://doi.org/10.1038/nmeth.4370>
- Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. *R News*, 2(3), 18-22.

- Mayo Clinic. (2023, November 14). *Primary biliary cholangitis*. Retrieved from Mayo Clinic: <https://www.mayoclinic.org/diseases-conditions/primary-biliary-cholangitis/symptoms-causes/syc-20376874>
- NHS. (2024, February 2). *PBC Treatments*. Retrieved from National Health Service (NHS): <https://www.nhs.uk/conditions/primary-biliary-cirrhosis-pbc/treatment/>
- NHS. (2024, February 2). *Treatment - Primary biliary cholangitis (primary biliary cirrhosis)*. Retrieved from National Health Service (NHS): <https://www.nhs.uk/conditions/primary-biliary-cholangitis-pbc/treatment/>
- NIH. (2017). *Ludwig's Classification (PBC Histological Stage)*. Retrieved from National Center for Biotechnology Information: [https://www.ncbi.nlm.nih.gov/books/NBK534940/table/cl.app\\_4.table11/](https://www.ncbi.nlm.nih.gov/books/NBK534940/table/cl.app_4.table11/)
- 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>
- NWS. (2021, July 26). *National Weather Service Instruction 10-1605: Storm Data Preparation*. Retrieved from National Weather Service (NWS): <http://www.nws.noaa.gov/directives/>
- Pohl, J. (2016). 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/>
- Rubin, D. B. (1976). Inference and missing data. *Biometrika*, 63(3), 581-592.
- Samenow, J., Feuerstein, J., & Livingston, I. (2021, Dec 11). How Friday night's rare and deadly December tornado outbreak unfolded. Retrieved from <https://www.washingtonpost.com/weather/2021/12/11/tornado-path-mayfield-kentucky-deaths/>

- Sibson, R. (1973). SLINK: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal (British Computer Society)*, 16(1), 30–34. doi:doi:10.1093/comjnl/16.1.30
- 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>
- Strickland, J. (2020). *Data Science Applications using R*. Lulu.com. Retrieved from [https://www.lulu.com/spotlight/strickland\\_jeffrey](https://www.lulu.com/spotlight/strickland_jeffrey)
- Strickland, J. S. (2023). *Regression totum modum*. Colorado Springs: Lulu, Inc. Retrieved from <https://www.lulu.com/shop/jeffrey-strickland/regression-totum-modum/hardcover/product-4kzged.html>
- The CLUSTER Procedure: Clustering Methods. (2016b). In *SAS/STAT 9.2 Users Guide*. SAS Institute. Retrieved Retrieved 2014-12-26
- The DISTANCE Procedure: Proximity Measures. (2016a). In *SAS/STAT 9.2 Users Guide*. SAS Institute. Retrieved Retrieved 2014-12-26
- Tryon, R. C. (1939). *Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality*. Edwards Brothers.
- Tukey, J. (1977). *Exploratory Data Analysis*. New York: Pearson.
- Ward, J. H. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 50(301), 236–244. doi:doi:10.2307/2282967
- Wickman, H. (2010). *ggplot2: Elegant Graphics for Data Analysis (Using R!)*. New York: Springer.
- Zhang, e. a. (2013). Agglomerative clustering via maximum incremental path integral. In *Pattern Recognition*.
- Zhang, e. a. (October 7–13, 2012). Graph degree linkage: Agglomerative clustering on a directed graph. *12th European Conference on*

*Computer Vision.* Florence, Italy. Retrieved from  
<http://arxiv.org/abs/1208.5092>



# Index

## A

accuracy .....	217, 218, 219, 222, 224, 227, 229
adaptive tree imputation.....	174, 175
adjusted R-squared.....	ii, 69, 74, 77, 85, 106, 109, 119
agglomerative clustering.....	211, 220, 221
agglomerative coefficient.....	237
agglomerative hierarchical clustering .....	237
AGglomerative NESting .....	238
aggregated data.....	46
<b>agnes</b> function.....	237
alternative hypothesis.....	22, 23, 24
analysis of variance .....	<i>See</i> ANOVA model
ANOVA	
table.....	3, 14
<b>anova</b> function.....	3
aov.....	4
ANOVA model.....	2, 4, 5, 6, 8
one-way .....	2
two-way .....	6
<b>anova(fit)</b> object.....	15
Aspin-Welch Unequal-Variance test .....	17
asymmetry.....	95
autocorrelation.....	79, 110
average linkage.....	236

## B

bagging .....	152
Baghdad i, 33, 34, 37, 41, 42, 43, 45, 46, 47, 48, 50, 51, 52, 54, 55, 57, 58, 59, 60, 61, 62, 63	
balanced accuracy .....	219, 227
bar chart .....	159, 263, 265
bar graph.....	43
<b>barplot</b> function 75, 76, 78, 85, 86, 100, 109, 130, 143, 163, 232, 258, 261, 262, 265	
bias .....	103, 131, 132, 134, 174, 175
bias-variance tradeoff.....	131
bilirubin .....	181, 186
Boolean dummy variables .....	189
bootstrap .....	152, 169
bootstrap aggregating .....	152

bow plot .....	97
box plot.....	126, 188
boxplot.....	1, 19, 20, 21, 32, 97, 172
<b>C</b>	
categorical values .....	84
categorical variables.....	160, 164, 175, 189, 202
Centennial Olympic Park bombing .....	41
Central Limit Theorem .....	i, 27, 29
central tendency.....	97, 127
centroid linkage.....	236
centroid model.....	210, 218
chaining phenomenon .....	237
cholangitis .....	153
Clarity.....	199
classification tree .....	152, 153
cluster analysis .....	209, 236, 237
cluster model .....	209, 210, 211, 235, 236
clustering.....	33, 209, 210, 211, 235, 236, 237
clustering algorithms .....	209, 211
Cohen's kappa.....	218, 222
combatant commands .....	33
comma delimited data .....	71, 103
common variance.....	7
complete linkage clustering .....	235
complete-linkage.....	236, 237
compliment coplot.....	201
concatenate.....	38
conditional dependence.....	205
conditioning plots.....	196
confidence band .....	186, 188
confidence interval....	3, 5, 11, 12, 15, 22, 23, 24, 25, 79, 87, 89, 97, 111, 112, 126, 127
<b>confint</b> function.....	3
confusion matrix .....	217, 219, 222, 224, 226
connectivity model.....	210, 211
connectivity-based clustering.....	234
constant variance.....	67, 79, 88, 90, 112, 114
continuous variable .....	160, 197, 198, 202
control group .....	1, 2
<i>Cook's Distance</i> .....	81, 89, 92, 113, 122
<b>coord_flip</b> function .....	12
coplots.....	<i>See</i> conditioning plots
correlation.....	52
correlation coefficient.....	52

<code>cost</code> function .....	132, 136
covariance matrix.....	225
covariate of interest .....	185, 190
covariates.....	176, 185, 186, 190, 205, 206
crime series.....	42, 45
<i>crimes series</i> .....	41
crop damage.....	254, 256, 258
cross validation set.....	142
cross-correlation.....	79, 110
cross-validation.....	108, 148
cross-validation set.....	73, 74, 77, 84, 85, 105, 138
curvilinear.....	52, 98, 99, 126
<code>cv.glmnet</code> function.....	139

## D

<i>D</i> index.....	231, 232
Daisy.....	240
data frame.....	1, 36
data mining.....	154, 209, 237
data preprocessing .....	71, 104, 138, 209
<code>datatable</code> function.....	35
DBSCAN.....	210, 211
dendrogram .....	220, 221, 223, 234, 240, 241
density model.....	210, 211, 243
density threshold.....	209
dependent variable.....	72, 73, 74, 104, 106, 207
diagnostic plot.....	4, 9
diagnostic plots .....	81
<code>diana</code> function.....	223
<code>Dindex</code> function.....	231
distance measure.....	213, 220, 231
distribution model .....	210
divisive hierarchical clustering .....	211, 212, 220, 224, 235
dot plot .....	6
D-penicillamine .....	153, 155
drifting.....	90, 91, 114, 115, 116
<code>DT</code> package .....	35

## E

EDA .....	<i>See</i> exploratory data analysis
<code>equatiomatic</code> package .....	74, 77, 85, 106
error variance .....	78, 80, 90, 91, 92, 114, 115, 116, 122
estimated mean response.....	68
Euclidean distance .....	215, 220, 235, 236
Excessive Heat .....	260

exploratory data analysis .....	17, 157, 159, 248
exponential distribution .....	27, 29
exponential random variate .....	27
<code>extract_eq</code> function .....	74, 77, 85, 106, 108, 117

## F

F test .....	22, 23
<i>F</i> values .....	14
<code>facet_wrap</code> function .....	8, 159
factor variable .....	1
fatalities .....	246, 253, 260, 263, 266
feature selection .....	133
features.....	32, 36, 72, 73, 76, 77, 83, 100, 101, 104, 105, 108, 109, 116, 117, 119, 130, 133, 134, 137, 138, 143, 213, 220, 225
field hospital .....	5, 6, 7, 8
fitted model .....	7, 79, 100, 130
fitted values .....	3, 7, 8, 9, 80, 87, 89, 113, 120
flash flood .....	260, 266
forest predicted response dependency .....	185
F-paired test .....	23
frequency distribution .....	161
frequency-domain .....	78, 86, 110
F-statistic .....	74, 77, 85, 106, 109, 119
<code>function</code> .....	178
<code>fviz_nbclust</code> function .....	214, 215

## G

Gaussian distribution .....	<i>See</i> normal distribution
Gaussian mixed model .....	211, 212, 225, 227, 229
generalization .....	131, 169, 240
<code>geom</code> function .....	18
<code>geom_boxplot</code> .....	1, 2, 20, 193
<code>geom_point</code> function .....	187
<code>geom_pointrange</code> function .....	11
<code>geom_smooth</code> function .....	187
<code>gg_error</code> function .....	170
<code>gg_partial</code> function .....	191
<code>gg_partial</code> plot .....	201
<code>gg_partial_coplot</code> .....	201
<code>gg_partial_coplot</code> function .....	200
<code>gg_rfsrc</code> function .....	171, 185
<code>gg_variable</code> function .....	185, 187, 197, 199
<code>gg_variable</code> plot .....	186, 189
<code>ggplot</code> .....	1, 8, 11, 18, 152
<code>ggplot2</code>	

<code>facet_wrap</code>	197
<code>ggplot2</code> package	151, 159
<code>ggRandomForests</code> package	151, 152, 170, 179
<code>glmnet</code> function	139
goodness of fit	2
Gower's formula	240
graph-based model	210
<code>grid.arrange</code> function	161
group model	210
group variables	1

## H

Hamming distance	236
hard clustering	210
hazard function	v, 173, 174
<code>heatmap</code>	47, 48, 54
heterogeneous cluster	223
heteroskedasticity	79, 88, 89, 112, 113
hierarchical clustering	210, 211, 220, 234, 240
histogram	28, 30, 31, 43, 160, 161
holdout set	135
HTML	35, 36
HTML <code>table</code> widget	42
Hubert index	231
hurricane	264
hyperparameter tuning	139
hyperparameters	139
hyperplane	68
hypothesis test	17, 19, 22

## I

ice storm	262
IED	<i>See</i> improvised explosive devices
time-detonated	44
improvised explosive devices	i, ii, 33, 34, 36, 39, 41, 42, 43, 44, 45, 46, 49, 50, 51, 52, 53, 54, 55, 56, 57, 59, 60, 61, 63
imputation	72, 95, 104, 169, 174, 175
independence	7, 17, 27, 72, 74, 90, 104, 106, 114
independent variable	90, 169
interaction	
two-way	6
intercept	67, 68, 73, 76, 110
Interpretability	198
iris data	244

## K

<code>kable</code> table.....	46
Kappa .....	<i>See Cohen's kappa</i>
k-means.....	210, 211
<i>k-means</i> clustering.....	212, 214, 215, 217
<code>kmeans</code> function .....	215, 216, 217, 230, 231, 233
<code>knitr</code> -package.....	46
knowledge discovery .....	209

## L

L1 regularization.....	136
L2 regularization.....	136
<code>labs</code> function.....	187
Lasso regression.....	134, 135, 136, 139, 140, 149
<code>leaflet</code> function .....	36
Leaflet map .....	36, 37
<code>leaflet</code> package .....	37
Least Absolute Shrinkage Selector Operator.....	<i>See Lasso regression</i>
Levenshtein distance.....	236
leverage values.....	79, 88, 112
<code>lightning</code> .....	260
linear model .....	89, 90, 105, 113
linear regression .....	105
linear regression model .....	67, 69, 75, 97, 106, 116, 127
linkage.....	234, 236, 237
linkage criteria .....	236
<code>lm</code> function .....	2
log transform .....	184
logistic regression.....	xii
lowest minimal depth .....	196

## M

map widget .....	36
maximal minimal depth .....	180
<code>mclust</code> function .....	225, 226
Mcnemar's Test.....	217, 219, 222, 224, 227
mean response .....	68
mean square error.....	iv, 68, 74, 75, 77, 99, 106, 107, 109, 129, 134
mean vector.....	210, 225
medoid model.....	218
minimal depth .....	177, 179, 180, 181, 182, 183, 184, 187, 191, 193, 194
minimal depth measure .....	180
minimal depth threshold.....	181
minimum energy clustering.....	236
missing values .....	72, 83, 95, 104, 137, 157, 159, 174, 175, 184, 213, 252

<i>missingness</i> .....	160, 165, 174
misspecification .....	177, 183
mixed feature .....	84, 101
model matrix .....	138
model-based cluster .....	211, 225
multicollinearity.....	134, 135, 149
multi-objective optimization.....	209
multiple linear regression .....	67, 69, 70
multiple R-squared .....	74, 77, 85, 106, 109, 119
<code>mutate</code> function.....	40

## N

National Weather Service .....	266
nation-state threats .....	71, 103
<code>NbClust</code> package.....	231
neural network.....	139
NOAA storm database .....	246
node split.....	174
non-linear transformation.....	67
normal distribution .....	11, 27, 29, 30, 32, 67
normal probability distribution.....	17
normal probability plot.....	9, 68
normality.....	10, 31, 79, 80
normalization.....	56, 57, 61, 62, 63, 194, 225
null hypothesis .....	22, 23, 24, 25, 70

## O

observed value.....	75, 107
OOB.....	<i>See Out-of-Bag</i>
optimal lambda.....	139, 140
outlier ..10, 32, 79, 81, 87, 88, 89, 91, 112, 113, 121, 159, 189, 211, 220, 237	
Out-of-Bag.....	169, 185
overfitting .....	131
overlapping clustering .....	211

## P

paired t-test.....	17
pairwise comparisons .....	5, 11
pairwise dissimilarities.....	240
panel plot.....	159, 188
partial dependence estimates.....	190
partial dependence plot.....	185, 190, 192, 202
partial likelihood model .....	153, 155
partitioning .....	234, 237
PBC.....	<i>See primary biliary cirrhosis, See primary biliary cirrhosis</i>

<code>pbc data</code> ..	153, 154, 156, 157, 158, 160, 163, 164, 167, 168, 169, 170, 171, 172, 175, 176, 178, 180, 185, 187, 190, 192, 193, 194, 195, 197, 201, 202, 203, 204, 206
<code>pbc data.trial</code> .....	178
penalty.....	132, 133, 136, 137, 149
performance metrics.....	74, 77, 85, 99, 148
pie chart .....	ii, 44
<code>pivot_longer function</code> .....	159
placebo-controlled .....	153, 155
<code>plot.gg_variable function</code> .....	187
<code>plot.gg_vimp function</code> .....	178
<code>plot.variable function</code> .....	200
polynomial .....	131, 132, 133
power spectrum .....	79, 110
precision .....	217, 219, 222, 224, 227
<code>predict</code> function .....	106, 142
predicted survival curve.....	172
predicted value .....	68, 75, 107
predictive accuracy.....	151, 177, 184
predictor variable.....	67, 68, 69, 70
predictor variables.....	67, 185, 207
primary biliary cirrhosis .....	v, 152, 153, 154, 155
probability distribution .....	95
<code>probability</code> function.....	172
property damage.....	254, 256, 258
p-value.....	22, 23, 24, 70, 74, 77, 85, 106, 109, 119
Python programming.....	xi

## Q

Q-Q plot .....	10, 80, 89, 113
<code>quantile_pts</code> function.....	197

## R

R function	
<code>glmnet</code> .....	149
<code>mean</code> .....	149
<code>R.sq</code> function.....	109
random decision forests .....	<i>See</i> random forests
random forest.....	151, 177
random forests.....	152
random sample .....	17, 27, 152
Random Survivability Function .....	171
Random Survival Forest .....	151
random variable .....	95
random variates .....	29, 32

<code>randomForestSRC</code> package .....	151, 152, 153, 174, 179, 181
<code>read.csv</code> function .....	103
<code>read_csv</code> function .....	35
<code>readr</code> package .....	35
regression analysis .....	75, 107
regression parameters .....	67
regression tree .....	152
regularization .....	101, 103, 131, 132, 133, 134, 149
resident space object .....	71, 72, 93, 95, 103, 106, 122, 123, 125, 130, 136
residual standard error .....	68, 74, 77, 85, 106, 109, 119
residuals ...	3, 4, 7, 8, 9, 10, 68, 78, 79, 80, 81, 86, 87, 88, 89, 90, 91, 110, 112, 113, 114, 116, 119, 120, 121
response variable .....	1, 67, 69, 71, 74, 103, 106, 167, 188, 249
reverse inverse method .....	34
<code>rexp</code> function .....	27, 28, 29
Ridge regression .....	136
risk adjusted survival .....	201
root mean square error .....	68, 74, 75, 77, 85, 106, 107, 109, 116, 119, 142, 143, 147, 148
<code>R-sq</code> function .....	85
<b>S</b>	
sample mean .....	27, 28, 29, 30, 32
sample size .....	68, 75, 107, 180
sample statistics .....	31
sample variance .....	30
sampling distribution .....	29
satellite .....	135, 137
<code>scale_fill_gradient</code> function .....	48
<code>scale</code> functions .....	187
scale-location plot .....	80
scatterplot .....	18, 97, 99, 126
sensitivity .....	217, 219, 222, 224, 227
severe weather .....	212, 245, 263, 265
shape ....	18, 27, 79, 88, 89, 112, 113, 160, 165, 185, 188, 191, 193, 198, 199, 201, 202, 205, 213, 216, 218, 221, 225, 235
shrinkage .....	149
Simplicity .....	198
simulation .....	71, 103, 115
Simulation	
discrete event .....	xi, xii
single-linkage .....	236, 237
single-linkage clustering .....	235
skewed .....	126
skewness .....	95, 97, 126, 127

smooth loess line.....	186, 188
soft clustering.....	210
specificity.....	217, 219, 222, 224, 227
split datasets .....	73
stacked bar plot .....	160
standard deviation .....	27, 29, 137
standardization.....	137
standardized coefficients .....	133
standardized residuals .....	80
<b><code>stat_qq</code> function.....</b>	9
statistical significance.....	22, 52
<b><code>stats</code>-package.....</b>	27
stem-leaf diagram.....	13
step values .....	138
storm surge.....	264
storm type.....	253
strict partitioning clustering.....	211
subspace clustering.....	211
subspace model .....	210
<b>sum of squared errors.....</b>	67, 68
Sum of Squares	
Treatment.....	15
<b><code>summary</code> function.....</b>	226
survivability	71, 72, 75, 83, 92, 93, 94, 95, 96, 97, 98, 99, 100, 103, 104, 106, 107, 108, 114, 115, 116, 124, 125, 126, 127, 128, 129, 130, 135, 136, 137, 143, 192, 197
survivability score.....	72, 119
survival probability function .....	168
<b>T</b>	
terrorism.....	41
thunderstorm winds.....	260
<b><code>tidyverse::gather</code> function .....</b>	161
time-domain.....	78, 79, 86, 110, 111
time-domain data.....	79, 86, 111
time-domain validation data .....	79
ToothGrowth.....	18
tornado .....	246, 249, 260, 263
training set.....	73, 105, 152, 185
transfer-function amplitude .....	79, 110
treatment.....	1, 2, 3, 4, 14, 151, 154, 155, 156, 157, 158, 159, 163, 165, 168, 172, 173, 174, 194, 195
TSTM Winds .....	<i>See</i> thunderstorm winds
t-test.....	23
<i>Tukey HSD test</i> .....	4

TukeyHSD function.....	11
two sample t-test.....	22, 24
<b><i>U</i></b>	
uncertainty .....	172, 227, 228
underfitting .....	131
Uniform distribution .....	34
Uniform[0,1] random number .....	34
<b><i>V</i></b>	
V linkage .....	236
validation data .....	78, 79, 86, 110, 142
<code>var.select</code> function .....	151
variable dependence .....	184, 185, 186, 187, 189, 190, 193, 197, 200
variable dependence plots.....	185, 197, 200
variable importance .....	144, 177, 178, 179, 181, 182, 183, 184
variable misspecification .....	177
variable reduction .....	144, 207
variable transformation.....	158
variance reduction .....	134, 135
VIMP .....	<i>See</i> variable importance
<b><i>W</i></b>	
Ward's criterion .....	236
web safe colors .....	47

