

# **Data Science Applications using R**

by

Jeffrey Strickland



Data Science Applications using R

Jeffrey Strickland

Copyright © 2017, Jeffrey S. Strickland

ISBN 978-0-359-81042-0

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

Printed in the United States of America





## Acknowledgements

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

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

— Mark Twain



<b>CHAPTER 1 – DECISION TREES .....</b>	<b>1</b>
TYPES.....	2
METRICS.....	4
GINI IMPURITY .....	4
INFORMATION GAIN .....	4
<i>General definition.....</i>	5
<i>Formal Definition.....</i>	5
DRAWBACKS.....	5
<i>Decision tree advantages.....</i>	6
<i>Limitations.....</i>	7
EXTENSIONS .....	7
<i>Decision graphs .....</i>	7
<i>Alternative search methods .....</i>	8
EXAMPLE 1: CLASSIFICATION TREE WITH RPART.....	8
<i>Grow tree .....</i>	8
<i>Display the results .....</i>	8
<i>Visualize cross-validation results.....</i>	9
<i>Detailed summary of splits.....</i>	9
<i>Plot tree.....</i>	11
<i>Prune the tree.....</i>	13
<i>Plot the pruned tree .....</i>	13
EXAMPLE 2: REGRESSION TREES.....	13
<i>Deviance means here the mean squared error. ....</i>	15
<i>Prune model 3 .....</i>	20
EXAMPLE 3: REGRESSION TREE EXAMPLE.....	20
<i>Grow tree .....</i>	21
<i>Create additional plots.....</i>	24
<i>plot tree.....</i>	25
<i>Create attractive postscript plot of tree .....</i>	26
<i>Prune the tree.....</i>	26
<i>Plot the pruned tree .....</i>	26
EXAMPLE 4: CLASSIFICATION TREES USING TREE .....	26
<i>Distributional prediction.....</i>	27
<i>Point prediction .....</i>	28
<i>Another way to show the data:.....</i>	28
PURITY AND ENTROPY .....	32
<i>The entropy is a measure of the purity of a dataset. ....</i>	32
<i>Information Gain .....</i>	33
ID3 ALGORITHM .....	34
<i>Training with data.....</i>	35
<i>Predict Function.....</i>	35
WORLD POPULATIONTREEMAP .....	35
<i>Convert from data.frame .....</i>	36

<i>Aggregate and Cumulate</i> .....	38
<i>Prune</i> .....	39
<i>Plotting the treemap</i> .....	40
<i>Plot as dendrogram</i> .....	41
PORTFOLIO BREAKDOWN (FINANCE) .....	43
<i>Aggregate for calculations</i> .....	43
<i>Formatters</i> .....	44
<i>Print results</i> .....	45
JENNY LIND (DECISION TREE, PLOTTING) .....	45
<i>Load YAML file</i> .....	45
<i>Calculate</i> .....	47
<i>Plot</i> .....	47
FILE EXPLORER (SYSTEM UTILITIES) .....	49
LISTVIEWER HTML WIDGET .....	51
BUBBLE CHART (VISUALISATION) .....	51
<i>Load JSON file</i> .....	52
<i>Plot</i> .....	54
EXERCISES .....	56
<b>CHAPTER 2 – RANDOM FORESTS</b> .....	<b>57</b>
HISTORY .....	57
ALGORITHM .....	58
BOOTSTRAP AGGREGATING .....	59
<i>Description of the technique</i> .....	59
<i>Example: Ozone data</i> .....	59
<i>Bagging for nearest neighbor classifiers</i> .....	61
<i>History</i> .....	62
FROM BAGGING TO RANDOM FORESTS .....	62
RANDOM SUBSPACE METHOD .....	62
<i>Algorithm</i> .....	63
RELATIONSHIP TO NEAREST NEIGHBORS .....	63
VARIABLE IMPORTANCE .....	64
VARIANTS .....	65
RANDOM FOREST USING R .....	65
<i>Load libraries</i> .....	66
<i>Medical longevity Study of primary biliary cirrhosis (PBC)</i> .....	66
<i>Transform variable values: years to days; 0-1 to T-F</i> .....	68
<i>Get transformed data</i> .....	69
<i>Plot continuous variables</i> .....	69
<i>Show multiple plots in a window</i> .....	70
<i>Include only the randomized patients</i> .....	71
<i>Create a test set from the remaining patients</i> .....	71
<i>Create the gg_survival object</i> .....	72
<i>Plot the survival probability function</i> .....	72

<i>Plot the cumulative hazard function</i> .....	73
<i>Duplicate the trial data</i> .....	74
<i>Group by bilirubin values</i> .....	74
<i>Plot the gg_survival object directly</i> .....	74
<i>Using shiny GUI for colorspace</i> .....	75
<i>Analog to: choose_palette(gui = "shiny")</i> .....	75
<i>Grow and store the random survival forest</i> .....	77
<i>Print the forest summary</i> .....	77
<i>Predict Patient Survival</i> .....	78
<i>Print prediction summary</i> .....	79
<b>STRENGTH OF PREDICTORS</b> .....	80
<i>Variable Importance</i> .....	80
<i>Minial Depth</i> .....	82
<i>Return an object with both minimal depth and vimp measures</i> .....	83
<i>Both minimal depth and VIMP</i> .....	85
<i>Get the minimal depth selected variables</i> .....	85
<i>Data generation</i> .....	87
<i>Plot the bilirubin variable dependence plot</i> .....	87
<i>Pull the categorical variables</i> .....	88
<i>plot the next 5 continuous variable dependence plots.</i> .....	88
<i>Partial Dependence Plot</i> .....	89
<i>Calculate the 1- and 3-year partial dependence</i> .....	91
<i>Categorical features</i> .....	92
<i>Conditional dependence plots</i> .....	100
<i>Plot.gg_variable</i> .....	101
<i>Partial dependence coplots</i> .....	102
<i>Partial coplot</i> .....	103
<b>BUSINESS APPLICATION</b> .....	105
<i>Scenario and dataset</i> .....	105
<i>Read and Explore data</i> .....	105
<i>Make Formula</i> .....	107
<i>Building Random Forest using R</i> .....	107
<i>Variable Importance Plot</i> .....	108
<i>Variable Importance Table</i> .....	108
<i>Make row names as columns</i> .....	108
<i>Predict Response Variable Value using Random Forest</i> .....	109
<i>Predicting response variable</i> .....	110
<i>Confusion Matrix</i> .....	110
<i>Load Library or packages</i> .....	110
<i>Create Confusion Matrix</i> .....	110
<i>Predicting response variable</i> .....	111
<i>Create Confusion Matrix</i> .....	111
<b>FIT A RANDOM FOREST TO THE FGL DATA AND COMPARE WITH SVM</b> .....	111
<b>EXERCISES</b> .....	114

<b>CHAPTER 3 – CLUSTER MODELS I.....</b>	<b>116</b>
DEFINITION.....	116
ALGORITHMS .....	118
<i>Connectivity based clustering (hierarchical clustering) .....</i>	118
<i>Metric .....</i>	119
<i>Linkage criteria .....</i>	120
HIERARCHICAL CLUSTERING USING R .....	122
<i>First load the package .....</i>	122
<i>Agglomerative Nesting (Hierarchical Clustering) .....</i>	122
<i>Plot the Data.....</i>	125
<i>Plot with Parameters.....</i>	125
<i>"show" equivalence of three "flexible" special cases .....</i>	126
<i>A "textual picture" of the dendrogram.....</i>	127
<i>Multiple Plots.....</i>	129
K-MEANS CLUSTERING.....	130
<i>Load and View the Data .....</i>	131
<i>NbClust Package.....</i>	132
<i>Social Network Clustering Analysis.....</i>	141
<i>Customer Segmentation.....</i>	146
<i>2D representation of the Segmentation:.....</i>	151
<i>Load necessary libraries.....</i>	153
<i>Inspect data structure.....</i>	153
<i>Summarise data.....</i>	153
<i>Subset and Plot the Attitude Data.....</i>	154
<i>K-Means Model .....</i>	155
<i>Iris K-Means #1 .....</i>	158
<i>Iris K-Means #2 .....</i>	160
<i>Iris K-Means #3 .....</i>	163
HIERARCHICAL CLUSTERING .....	164
DENSITY-BASED CLUSTERING.....	165
ADVANCED CLUSTER MODELS IN R.....	168
<i>Example 1: h-cluster .....</i>	169
<i>Example 2 .....</i>	173
<i>Example 3 .....</i>	174
EXERCISES .....	198
<b>CHAPTER 4 – CRIME LINKAGE AND UNSOLVED CRIMES .....</b>	<b>201</b>
INTRODUCTION TO CRIME SERIES LINKAGE.....	201
TRIAL RUN WITH CRIMELINKAGE .....	201
<i>Make IDs: Criminal 1 committed crimes 1-4, etc.....</i>	202
<i>Spatial locations of the crimes:.....</i>	202
<i>Same distribution for all criminals .....</i>	202
<i>Times of the crimes.....</i>	202

COMMITTED BY THE SAME CRIMINAL: .....	203
PRELIMINARIES .....	204
<i>Using crimeLinkage for Analyzing Unsolved Crimes.....</i>	204
<i>Load Libraries .....</i>	204
<i>Load the Data.....</i>	204
DATA DESCRIPTION.....	205
SETUP THE DATA FOR ANALYSIS.....	205
<i>Make a Crime Series.....</i>	205
<i>Make Crime Pairs for Case Linkage .....</i>	205
<i>Make Evidence Variables for Case Linkage .....</i>	205
<i>Build Training and Testing Data.....</i>	206
FIT LOGISTIC REGRESSION MODEL AND MAKE ESTIMATEBF() FUNCTION .....	206
<i>Plot the Results.....</i>	207
<i>Examine the Factors.....</i>	212
<i>Prediction using the Testing Data .....</i>	213
FIT NAIVE BAYES MODEL AND MAKE ESTIMATEBF() FUNCTION.....	213
<i>Plot Results.....</i>	214
<i>Print Factors .....</i>	215
<i>Agglomerative Hierarchical Crime Series Clustering .....</i>	216
GET UNSOLVED CRIMES.....	217
RUN AGGLOMERATIVE HIERARCHICAL CRIME CLUSTERING .....	217
<i>Plot results in dendrogram using plot_hcc().....</i>	217
<i>Examine crimes C:431 and C:460 .....</i>	219
<i>Find path info for crime C:429.....</i>	219
<i>Hierarchical Based Crime Series Linkage.....</i>	219
<i>Example.....</i>	219
<i>Using Crime C:394 (the 4th unsolved crime) .....</i>	221
<i>Results from hierarchical clustering .....</i>	222
BAYESIAN MODEL-BASED APPROACHES.....	222
<i>Make the crime group labels for each crime (NA for unsolved crimes) .....</i>	223
RUN MARKOV CHAIN MONTE CARLO (MCMC) .....	223
<i>Extract pairwise probabilities.....</i>	227
<i>Get index of unsolved crimes.....</i>	228
<i>Image plot of linkage probabilities.....</i>	228
<i>Find strongest linkages.....</i>	229
EXERCISES .....	231
<b>CHAPTER 5 – INTRODUCTION TO TEXT MINING .....</b>	<b>235</b>
INTRODUCTION .....	235
LOAD THE R PACKAGES .....	235
LOAD THE DATA .....	236
CORPUS PREPROCESSING .....	237
CREATE DOCUMENT MATRICES .....	239
VISUALIZING THE RESULTS .....	241

FIND CORRELATIONS .....	242
USING WORDCLOUDS TO VISUALIZE RESULTS.....	243
TIDY TEXT ANALYTICS I.....	244
<i>Tidy Format.....</i>	244
<i>Tidy Text Format.....</i>	244
<i>Contrasting tidy text with other data structures .....</i>	245
<i>The unnest_tokens function .....</i>	246
<i>Tibbles.....</i>	247
<i>Tokenization .....</i>	247
<i>Example - Indian Philosophy.....</i>	248
<i>Filter for negative words.....</i>	253
<i>Build a cloud chart.....</i>	254
<i>END OF SCRIPT.....</i>	255
TIDY TEXT ANALYTICS II.....	256
<i>Load necessary libraries.....</i>	256
<i>Tidy Text .....</i>	256
<i>Quantum Words .....</i>	257
<i>Quantum Tokens .....</i>	258
<i>Quantum Cloud.....</i>	258
<i>Lexicon Exploration.....</i>	259
<i>NRC Lexicon .....</i>	260
<i>AFINN Lexicon.....</i>	260
<i>Bing Lexicon.....</i>	261
<i>Getting Sentiments with NRC .....</i>	262
<i>Analyzing word and document frequency: tf-idf .....</i>	263
<i>Term frequency.....</i>	264
<i>Get Term Frequencies.....</i>	265
<i>Zipf's law.....</i>	266
<i>The bind_tf_idf function .....</i>	271
SUMMARY.....	274
EXERCISES .....	275
<b>CHAPTER 6 – TWEET SENTIMENT ANALYSIS .....</b>	<b>277</b>
SENTIMENT LEXICONS.....	277
INSTALL REQUIRED LIBRARIES .....	278
LEXICON EXAMPLE .....	278
EXAMPLE 1: THE INNER JOIN .....	280
POSITIVE & NEGATIVE WORDS OVER TIME.....	286
MOST COMMON POSITIVE AND NEGATIVE WORDS.....	288
WORDCLOUDS .....	290
WORD FREQUENCIES.....	292
COMPARING WORD USAGE .....	296
CHANGES IN WORD USE .....	298
EXAMPLE 2: ISISFANBOY .....	303

<i>Loading the Data</i> .....	304
<i>Fixing the dates</i> .....	305
<i>Tokenization</i> .....	305
<i>Removing common words</i> .....	306
<i>Word frequencies and word clouds</i> .....	307
PLOTTING THE MOST OCCURRING POSITIVE AND NEGATIVE WORDS.....	309
NEW BIGRAM COUNTS .....	310
<i>Word clouds on positive and negative sentiment</i> .....	310
WORDCLOUD ON POSITIVE AND NEGATIVE WORDS.....	311
TOP 10 WORDS CONTRIBUTING TO DIFFERENT SENTIMENTS .....	312
POSITIVE & NEGATIVE WORDS OVER TIME.....	314
<i>Sentiment proportion analysis</i> .....	315
N-GRAMS .....	316
GET AFINN SETIMENTS.....	317
GET SENTIMENT SCORES.....	317
PLOT BIGRAMS .....	318
NETWORK OF BIGRAMS.....	319
EXERCISES .....	321
<b>CHAPTER 7 - LATENT DIRICHLET ALLOCATION (LDA) .....</b>	<b>323</b>
PART I: BLOG TOPIC ANALYSIS USING LDA .....	323
SET WORKING DIRECTORY .....	323
READ FILES INTO A CHARACTER VECTOR.....	323
CREATE CORPUS FROM VECTOR.....	324
START PREPROCESSING .....	325
STEM DOCUMENT .....	326
CREATE DOCUMENT-TERM MATRIX .....	327
LOAD TOPIC MODELS LIBRARY .....	328
SET PARAMETERS FOR GIBBS SAMPLING .....	328
NUMBER OF TOPICS .....	328
RUN LDA USING GIBBS SAMPLING .....	328
WRITE OUT RESULTS .....	329
TOP 6 TERMS PER TOPIC.....	329
TOPIC PROBABILITIES.....	329
PART II: 2012 USA PRESIDENTIAL DEBATE USING LDA .....	330
PRELIMINARIES .....	330
GET EXTERNAL SCRIPTS.....	330
LOAD THE DATA .....	330
DETERMINE OPTIMAL NUMBER OF TOPICS .....	331
RUN THE MODEL.....	332
PLOT THE TOPICS PER PERSON & TIME .....	333
PLOT THE TOPICS MATRIX AS A HEATMAP .....	334
HEATMAP OF TOPICS.....	335
NETWORK OF THE WORD DISTRIBUTIONS OVER TOPICS.....	335

WORD DISTRIBUTIONS OVER TOPICS .....	337
LDAVIS OF MODEL .....	338
FITTING NEW DATA.....	340
RUN THE MODEL FOR NEW DATA .....	340
PLOT THE TOPICS PER PERSON & LOCATION FOR NEW DATA .....	341
APPENDIX – OPTIMAL_K .....	344
TOPICMODELS2LDavis.R.....	349
EXERCISES .....	350
<b>CHAPTER 8 – SPATIAL DATA ANALYSIS.....</b>	<b>352</b>
INTRODUCTION.....	352
PRELIMINARIES.....	352
<i>Key Libraries for Spatial Analysis .....</i>	352
<i>Loading Libraries .....</i>	353
<i>Import India shape file (.SHP) .....</i>	353
<i>Mandatory files .....</i>	353
<i>Import India Population Grid Files (.GRD).....</i>	354
<i>Comments on Imported Data .....</i>	354
<i>Examining the Output.....</i>	354
<i>Some Comments about Spatial Objects.....</i>	355
<i>Shapefile Structure .....</i>	355
THE DATA SLOTS .....	355
<i>First Slot for Data Object .....</i>	355
<i>Second Slot for Polygons Object .....</i>	356
<i>Third slot for Plotting Order.....</i>	357
<i>Fourth Slot for Bounding Box Object .....</i>	357
<i>Fifth Slot for Class CRS .....</i>	357
<i>Headers.....</i>	358
<i>Classes .....</i>	358
EXPLORE SPATIAL STRUCTURE .....	358
PLOTTING LAYERS .....	358
PLOTTING MULTIPLE SHAPE DATA .....	361
APPLY PLOTTING ENHANCEMENTS .....	362
PLOT MAP WITH ENHANCEMENTS.....	363
SETUP BOUNDING BOX FOR INDIAN MAP .....	364
SETUP BOUNDING BOX PLOT .....	364
PLOT BOUNDING BOX MAP .....	365
PLOT INDIA MAP WITH GREY FILL .....	365
FIND GEOGRAPHIC CENTROIDS FOR INDIAN STATES .....	366
INDIA.SHP MAP RESCALED WITH LAT/LONG GRIDS .....	366
CREATE C-SHAPE DATAFRAME .....	366
<i>India.shp Map with States .....</i>	366
<i>India.shp Map with Cities .....</i>	369
SELECTING QUADRANTS .....	369

TEST FOR NE QUADRANT INCLUSION.....	370
TEST FOR SE QUADRANT INCLUSION .....	370
TEST FOR NW QUADRANT INCLUSION .....	370
TEST FOR SW QUADRANT INCLUSION.....	370
PLOT QUADRANTS WITH COLORS.....	370
CREATING NEW R OBJECT.....	372
<i>Check the class of these new objects using class():.....</i>	372
CREATING NEW SPATIAL DATA.....	373
<i>Comment on Creating Spatial Data.....</i>	373
<i>Add Data to an Existing Spatial Data File.....</i>	373
PROJECTIONS: SETTING AND TRANSFORMING CRS IN R .....	373
<i>Comments on CRS Changes.....</i>	374
<i>Find EPSG Codes .....</i>	374
<i>Convert the Coordinates.....</i>	374
<i>Change the EPSG .....</i>	374
<i>Saving CRS Formats.....</i>	375
<i>Reset INDIA Shapefile .....</i>	375
JOINING NON-SPATIAL AND SPATIAL DATA WITH SAME KEYS .....	376
<i>India 2011 Population .....</i>	376
<i>Joining Tables .....</i>	377
<i>Basic Scatterplots .....</i>	378
<i>India.shp Map with Cities .....</i>	379
<i>India Map with Sates Setup.....</i>	380
<i>Plot India Map with States (colored).....</i>	380
<i>India Population Mask (colored) .....</i>	381
<i>Define Popups for Leaflet Map.....</i>	382
<i>India Leaflet Population Map .....</i>	383
JOINING NON-SPATIAL DATA WITH DIFFERENT KEYS .....	385
<i>Import India shape file (.SHP).....</i>	385
<i>Import the Crime Data .....</i>	385
<i>Exploration Comments .....</i>	386
<i>Indian States.....</i>	386
<i>Comments on the Code .....</i>	386
<i>Joining Spatial and Non-Spatial Tables .....</i>	387
<i>Explore the New Dataset.....</i>	388
MAKING MAPS WITH TMAP .....	388
<i>Plot a Basic Map.....</i>	388
<i>Plot and Enhanced Map .....</i>	389
<i>Making Maps with ggmap .....</i>	393
<i>Add Layers to a Plot.....</i>	393
<i>Comments about Layers.....</i>	394
SUMMARY .....	394
EXERCISES .....	395

<b>CHAPTER 9 – USING R FOR CRIME ANALYSIS.....</b>	<b>398</b>
INTRODUCTION.....	398
AUTHOR’S NOTE.....	398
INSTALL R LIBRARIES .....	398
THE CRIME DATA.....	399
<i>About the Data</i> .....	399
<i>Read the data</i> .....	400
<i>Display Data</i> .....	400
<i>Preprocess Data</i> .....	402
<i>Visualize Data</i> .....	403
INTEACTIVE MAPS.....	405
CRIME SERIES – DAILY CRIMES PLOT WITH VARIANCE .....	409
<i>Aggregate Data</i> .....	410
<i>Create a Bar Chart</i> .....	411
<i>Create a pie chart based on the incident category.</i> .....	412
<i>Author’s Note</i> .....	412
THEFT OVER TIME.....	414
<i>Theft Time Heatmap</i> .....	415
<i>Reorder and format Factors</i> .....	416
<i>Create Time Heatmap</i> .....	416
<i>Arrest Over Time</i> .....	417
<i>Daily Arrests</i> .....	418
CORRELATION ANALYSIS.....	421
<i>Factor by Crime Category</i> .....	421
<i>Number of Arrests by Category and time of Arrest</i> .....	422
<i>Normalizled Gradients</i> .....	423
<i>Factor by Police District</i> .....	426
<i>Factor by Police District</i> .....	427
<i>Factor by Month</i> .....	428
<i>Factor by Year</i> .....	430
POLICE ARREST NORMALIZED BY YEAR .....	431
EXERCISES .....	433
<b>CHAPTER 10 – HYPERPARAMETER TUNING .....</b>	<b>436</b>
SEARCH METHODS FOR HYPERPARAMETER TUNING .....	436
INSTALL AND LOAD PACKAGES .....	436
IMPORTING THE DATA .....	437
<i>Rename variables</i> .....	437
EXPLORATORY DATA ANALYSIS .....	437
<i>Recalculate composition as proportions ranging from 0 to 1</i> .....	438
<i>Correlations</i> .....	440
<i>Plot correlation heatmap</i> .....	441
<i>Visualizing Feature relationships with a scatterplot matrix</i> .....	441

<b>MODELING.....</b>	<b>443</b>
<i>Create training and test sets .....</i>	443
<i>Training Parameters.....</i>	444
<i>Training Settings .....</i>	445
<i>Model Training .....</i>	445
<i>Grid Search .....</i>	446
<i>Define MAE.....</i>	449
<i>Random Search .....</i>	452
<i>Genetic Algorithm .....</i>	454
<i>Differential Evolution .....</i>	459
<b>MODEL PERFORMANCE .....</b>	<b>467</b>
<i>Performance on Training Set.....</i>	467
<i>Create summary table.....</i>	467
<i>Print table.....</i>	467
<i>Performance on Test Set .....</i>	468
<i>Function to plot observed vs predicted values .....</i>	468
<i>Function to plot residuals.....</i>	469
<i>Grid Search (GS) .....</i>	470
<i>Random Search .....</i>	470
<i>Genetic Algorithm (GA) .....</i>	471
<i>Differential evolution (DE).....</i>	472
<i>Create summary table.....</i>	473
<i>Test Set Statistics of XGBoost Models. ....</i>	473
<b>SUMMARY .....</b>	<b>473</b>
<b>EXERCISES.....</b>	<b>476</b>
<b>REFERENCES.....</b>	<b>478</b>



## Preface

To write a single book about data science, at least as I view the discipline, would result in several volumes. I have come to view Data Science kind of like Engineering. We have all sorts of engineers: mechanical, electrical, civil, aeronautical, industrial, and so on. We still have them, but when we talk about them, we tend to use the general term “engineers” and their field as “engineering.” I have thought about this for a few years, and I have concluded that we do something similar with the terms “data scientists” and “data science.”

*Data really powers everything that we do.*

— Jeff Weiner, LinkedIn

### What Comprises Data Science?

**Data Engineering** (Data Engineer). Data engineers convert data into a valuable format for investigation, inquiry, or analysis. This data conversion involves an extraction, transformation, and loading (ETL) effort to place data into data warehouses. One of the data engineer’s extremely valuable and sought-after skills is the capability to design, build, and maintain data warehouses. Along with ETL, these three conceptual steps are the basis for the design and structure of most data pipelines. They serve as an outline or blueprint the way raw data are transformed to data that is ready for consumption.

**Database Architecture** (Database Architect). Many people who write about data science place architecture in the field of information technology (IT), and indeed IT seems an appropriate. However, I am associating it data architecture with data science, as it relates to the composition of models, policies, rules or standards that govern data collection, storage, arrangement, integration, and utilization in data systems and organizations. The data models (e.g., logical data models), definitions and data flows on various levels are referred as Data Architecture artifacts. The artifacts allow for a common vocabulary that articulates the integrated data requirements. This articulation ensures that data assets are arranged, stored, managed, and used in systems that support of an organization’s strategy.

**Database Management** (Database Manager). Database management task is any task that protects an organization's data, ensures legality and compliance, and keeps data-driven applications performing optimally. This includes configuration management, storage and capacity planning, performance monitoring and modification, redundancy management, as well as recovery, archiving, partitioning, masking, and retirement. If we control data throughout its entire lifecycle, from requirements generation through retirement, organizations can prevent degradation of efficiency and enhance data integration for superior business intelligence, thereby increasing or sustaining revenue.

**Data Mining** (Data Analyst, Statistician). Data mining is the semi-automatic or automatic process of taking large quantities of data and extracting data that is pertinent to an organization's data operations. Such mining pulls out interesting patterns such as groups of data records (cluster analysis); identifies items, events or observations which do not conform to an expected pattern (anomaly detection or outlier detection); finds frequent co-occurring associations among a collection of items (association rule mining or market basket analysis); and disc

overs statistically relevant patterns between data examples where the values are delivered in a sequence (sequential pattern mining). These mining tasks usually involve using database techniques to find patterns, which may be used in further analysis or, for example, in machine learning and predictive analytics. However, the data collection, data preparation, as well as the result interpretation and reporting is not part of the data mining step.

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

— W. Edwards Deming, statistician

**Business Intelligence** (BI Engineer, BI Analyst, BI Developer). Business intelligence (BI), often referred to a business analytics, pursues the transform of data into actionable intelligence that informs an organization's strategic and tactical business decisions. In BI, analysts use software tools, such as Tableau and SAS, to access and analyze pertinent business data and present the results, in a variety of "story telling" forms, to provide stakeholders with detailed intelligence about the state of the business. Some experts distinguish between BI and business analytics, but I treat them as the same, with the additional idea that they are a specialized area of data analytics. Consequently, BI may

describe a past or current state of the business and it also analyzes data to predict what will happen or what could happen by taking a certain approach.

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

— Amod Malviya, CTO at flipkart

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

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

— Alan Turing, Computing machinery and intelligence

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

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

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

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

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

— Carly Fiorina, former CEO, Hewlett-Packard

**Statistics** (Statisticians). In my assessment, this is the foundation of data science. It is the science that deals with the collection, classification, analysis, and interpretation of numerical facts or data. Supported directly by use of mathematical theories of probability, statistics imposes order and regularity on collections of dissimilar elements. People generally do not know how to read, interpret, and use statistics, which must be governed by the following inquiries:

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

## **Interdisciplinary**

In data analytics, we use statistics, which entails the application of mathematical and computational aspect of data science and suggests decision options to take advantage of the results of foundational

descriptive and predictive analytics, where statistical methods and models prevail. We also use machine learning. I have also performed data mining while doing analytics. It like bringing a combined arms force to bear on a stubborn, defending enemy to drive them from their stronghold and reveal their vulnerabilities.

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

– Ronald Coase, winner of the Nobel Prize in Economics

## About R and R-Studio

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

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

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

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

## Pricing Philosophy

Those that follow me on LinkedIn know that my goal when I starting publishing was to minimize the cost of books, especially for parents with college student. I had to buy a book last semester for Phlebotomy—a story for another time—that was 239 pages from cover to cover (including front-matter, TOC and Index), and was not in full color, that cost \$131.00. The same book, if self-published through Lulu.com (the publisher I use) the manufacturing cost would be \$6.03. Lulu need a little profit, so I could sell this book for around \$10.00. Okay, less compare: big publishing house for \$131.00 or Lulu for \$10.00—I might be off a dime or two, but I attribute that to weak batteries in my calculator.



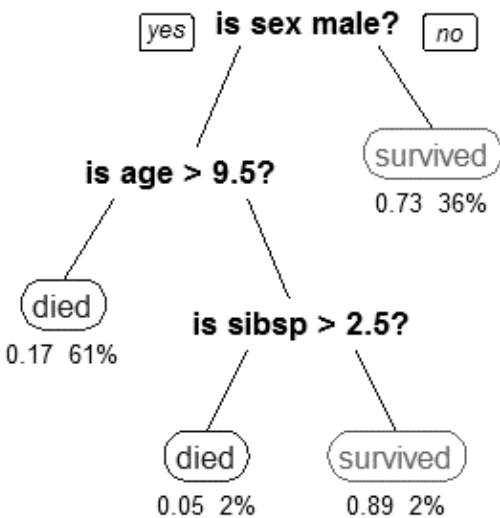
## CHAPTER 1 – Decision Trees

Decision tree learning is a method commonly used in data mining (Rokach & Maimon, 2008). The goal is to create a model that predicts the value of a target variable based on several input variables. An example is shown on the right. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

A decision tree is a simple representation for classifying examples. Decision tree learning is one of the most successful techniques for supervised classification learning. For this section, assume that all of the features have finite discrete domains, and there is a single target feature called the classification. Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature (see Figure 1-1). The arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.

A tree showing survival of passengers on the *Titanic* (“sibsp” is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

A tree can be “learned” by splitting the source *set* into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The *recursion* is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees (*TDIDT*) (Quinlan, 1986) is an example of a “greedy” algorithm, and it is by far the most common strategy for learning decision trees from data.



**Figure 1-1.** Example of a Decision Tree

In data mining, decision trees can be described also as the combination of mathematical and computational techniques to aid the description, categorization and generalization of a given set of data.

Data comes in records of the form:

$$(x, Y) = (x_1, x_2, x_3, \dots, x_k, Y).$$

The dependent variable,  $Y$ , is the target variable that we are trying to understand, classify or generalize. The vector  $x$  is composed of the input variables,  $x_1, x_2, x_3$ , etc., that are used for that task.

## Types

Decision trees used in *data mining* are of two main types:

- **Classification tree analysis** is when the predicted outcome is the class to which the data belongs.
- **Regression tree analysis** is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

The term **Classification and Regression Tree** (CART) analysis is an *umbrella term* used to refer to both of the above procedures, first introduced by Breiman et al. (Breiman, Friedman, Olshen, & Stone, 1984). Trees used for regression and trees used for classification have

some similarities—but also some differences, such as the procedure used to determine where to split.

Some techniques, often called ensemble methods, construct more than one decision tree:

Bagging decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction (Breiman L. , Bagging predictors, 1996).

A Random Forest classifier uses several decision trees, in order to improve the classification rate.

Boosted Trees can be used for regression-type and classification-type problems (Friedman, Stochastic Gradient Boosting, 1999) (Hastie, Tibshirani, & Friedman, The elements of statistical learning : Data mining, inference, and prediction, 2001).

Rotation forest, in which every decision tree is trained by first applying *principal component analysis* (PCA) on a random subset of the input features (Rodriguez, Kuncheva, & Alonso, 2006).

Decision tree learning is the construction of a decision tree from class-labeled training tuples. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.

There are many specific decision-tree algorithms. Notable ones include:

- **ID3** (Iterative Dichotomiser 3)
- **C4.5** (successor of ID3)
- **CART** (Classification and Regression Tree)
- **CHAID** (CHi-squared Automatic Interaction Detector): Performs multi-level splits when computing classification trees (Kass, 1980).
- **MARS**: extends decision trees to handle numerical data better.
- **Conditional Inference Trees**: Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning (Hothorn, Hornik, & Zeileis, 2006).

ID3 and CART were invented independently at around same time (between 1970-1980) yet follow a similar approach for learning decision tree from training tuples.

## Metrics

Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items (Rokach & Maimon, Top-down induction of decision trees classifiers-a survey, 2005). Different algorithms use different metrics for measuring “best”. These generally measure the homogeneity of the target variable within the subsets. Some examples are given below. These metrics are applied to each candidate subset, and the resulting values are combined (e.g., averaged) to provide a measure of the quality of the split.

### Gini impurity

Used by the CART (classification and regression tree) algorithm, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled, if it were randomly labeled according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category. This should not be confused with Gini coefficient.

To compute Gini impurity for a set of items, suppose  $i$  takes on values in  $\{1, 2, \dots, m\}$ , and let  $f_i$  be the fraction of items labeled with value  $i$  in the set.

$$I_G(f) = \sum_{i=1}^m f_i(1 - f_i) = \sum_{i=1}^m (f_i - f_i^2) = \sum_{i=1}^m f_i - \sum_{i=1}^m f_i^2 = 1 - \sum_{i=1}^m f_i^2$$

### Information gain

In information theory and machine learning, information gain is a synonym for Kullback–Leibler divergence. However, in the context of decision trees, the term is sometimes used synonymously with mutual information, which is the expectation value of the Kullback–Leibler divergence of a conditional probability distribution.

In particular, the information gain about a random variable  $X$  obtained from an observation that a random variable  $A$  takes the value  $A = a$  is

the Kullback-Leibler divergence  $D_{KL}(p(x|a)||p(x|I))$  of the prior distribution  $p(x|I)$  for  $x$  from the posterior distribution  $p(x|a)$  for  $x$  given  $a$ .

The expected value of the information gain is the mutual information  $I(X; A)$  of  $X$  and  $A$ , i.e., the reduction in the entropy of  $X$  achieved by learning the state of the random variable  $A$ .

In machine learning, this concept can be used to define a preferred sequence of attributes to investigate to most rapidly narrow down the state of  $X$ . Such a sequence (which depends on the outcome of the investigation of previous attributes at each stage) is called a decision tree. Usually an attribute with high mutual information should be preferred to other attributes.

### **General definition**

In general terms, the expected information gain is the change in information entropy  $H$  from a prior state to a state that takes some information as given:

$$IG(T, a) = H(T) - H(T|a).$$

### **Formal Definition**

Let  $T$  denote a set of training examples, each of the form  $(x, y) = (x_1, x_2, x_3, \dots, x_k, y)$  where  $x_a \in vals(a)$  is the value of the  $a$ th attribute of example  $x$  and  $y$  is the corresponding class label. The information gain for an attribute  $a$  is defined in terms of entropy  $H()$  as follows:

$$IG(T, a) = H(T) - \sum_{v \in vals(a)} \frac{|\{x \in T | x_a = v\}|}{|T|} \cdot H(\{x \in T | x_a = v\}).$$

The mutual information is equal to the total entropy for an attribute if for each of the attribute values a unique classification can be made for the result attribute. In this case, the relative entropies subtracted from the total entropy are 0.

### **Drawbacks**

Although information gain is usually a good measure for deciding the relevance of an attribute, it is not perfect. A notable problem occurs when information gain is applied to attributes that can take on a large number of distinct values. For example, suppose that one is building a

decision tree for some data describing the customers of a business. Information gain is often used to decide which of the attributes are the most relevant, so they can be tested near the root of the tree. One of the input attributes might be the customer's credit card number. This attribute has a high mutual information, because it uniquely identifies each customer, but we do not want to include it in the decision tree: deciding how to treat a customer based on their credit card number is unlikely to generalize to customers we haven't seen before (overfitting).

Information gain ratio is sometimes used instead. This biases the decision tree against considering attributes with a large number of distinct values. However, attributes with very low information values then appeared to receive an unfair advantage. In addition, methods such as permutation tests have been proposed to correct the bias (Deng, Runger, & Tuv, 2011).

### **Decision tree advantages**

Amongst other data mining methods, decision trees have various advantages:

- Simple to understand and interpret. People can understand decision tree models after a brief explanation.
- Requires little data preparation. Other techniques often require data normalization, *dummy variables* need to be created and blank values to be removed.
- Able to handle both numerical and *categorical* data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. (For example, relation rules can be used only with nominal variables while neural networks can be used only with numerical variables.)
- Uses a *white box* model. If a given situation is observable in a model the explanation for the condition is easily explained by Boolean logic. (An example of a black box model is an artificial neural network since the explanation for the results is difficult to understand.)
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- **Robust**. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
- Performs well with large datasets. Large amounts of data can be analyzed using standard computing resources in reasonable time.

## **Limitations**

The problem of learning an optimal decision tree is known to be *NP-complete* under several aspects of optimality and even for simple concepts (Hyafil & Rivest, 1976). Consequently, practical decision-tree learning algorithms are based on heuristics such as the *greedy algorithm* where locally-optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally-optimal decision tree. To reduce the greedy effect of local-optimality some methods such as the dual information distance (*DID*) tree was proposed (I., A., N., & Singer, 2014).

Decision-tree learners can create over-complex trees that do not generalize well from the training data. (This is known as *overfitting* (Bramer, 2007).) Mechanisms such as *pruning* are necessary to avoid this problem (with the exception of some algorithms such as the Conditional Inference approach, which does not require pruning (Strobl, Malley, & Tutz, 2009) (Hothorn, Hornik, & Zeileis, 2006)).

There are concepts that are hard to learn because decision trees do not express them easily, such as *XOR*, *parity* or *multiplexer* problems. In such cases, the decision tree becomes prohibitively large. Approaches to solve the problem involve either changing the representation of the problem domain (known as propositionalization) (Horváth & Yamamoto, 2003) or using learning algorithms based on more expressive representations (such as *statistical relational learning* or *inductive logic programming*).

For data including categorical variables with different numbers of levels, *information gain in decision trees* is biased in favor of those attributes with more levels (Deng, Runger, & Tuv, 2011). However, the issue of biased predictor selection is avoided by the Conditional Inference approach.

## **Extensions**

### **Decision graphs**

In a decision tree, all paths from the root node to the leaf node proceed by way of conjunction, or AND. In a decision graph, it is possible to use disjunctions (ORs) to join two more paths together using *Minimum message length* (MML) (Tan & Dowe, 2004). Decision graphs have been further extended to allow for previously unstated new attributes to be

learnt dynamically and used at different places within the graph. The more general coding scheme results in better predictive accuracy and log-loss probabilistic scoring. In general, decision graphs infer models with fewer leaves than decision trees.

### **Alternative search methods**

Evolutionary algorithms have been used to avoid local optimal decisions and search the decision tree space with little *a priori* bias (Papagelis, 2001) (Barros, Basgalupp, Carvalho, & Freitas, 2011).

It is also possible for a tree to be sampled using MCMC in a Bayesian paradigm (Chipman, George, & McCulloch, 1998).

The tree can be searched for in a bottom-up fashion (Barros, Cerri, Jaskowiak, & Carvalho, 2011).

### **Example 1: Classification Tree with rpart**

Let's use the data frame `kyphosis` to predict a type of deformation (`kyphosis`) after surgery, from age in months (`Age`), number of vertebrae involved (`Number`), and the highest vertebrae operated on (`Start`). In R, call the `rpart` library (**Recursive Partitioning and Regression Trees**).

```
library(rpart)
```

#### **Grow tree**

```
fit <- rpart(Kyphosis ~ Age + Number + Start, method= "class", data=kyphosis)
```

where `kyphosis` is the response, with variables `Age`, `Number`, and `Start`. `Class` is the method and `kyphosis` is the data set. Next, we display the results.

#### **Display the results**

```
printcp(fit)
##
## Classification tree:
## rpart(formula = Kyphosis ~ Age + Number + Start, data =
## kyphosis,
##       method = "class")
##
## Variables actually used in tree construction:
## [1] Age   Start
```

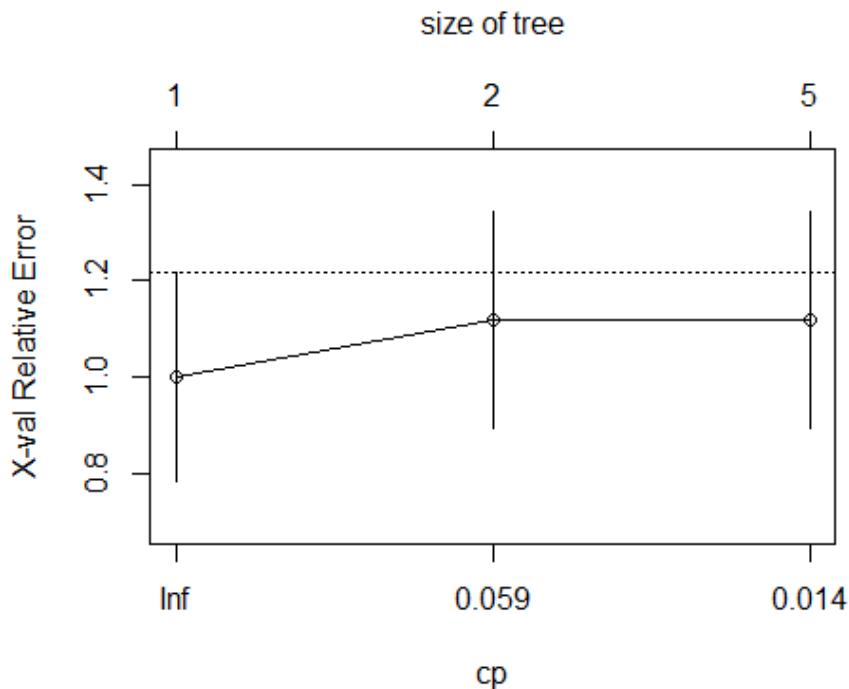
```

## 
## Root node error: 17/81 = 0.20988
## 
## n= 81
## 
##          CP nsplit rel error xerror      xstd
## 1 0.176471     0    1.00000 1.0000 0.21559
## 2 0.019608     1    0.82353 1.1176 0.22433
## 3 0.010000     4    0.76471 1.1176 0.22433

```

### *Visualize cross-validation results*

```
plotcp(fit)
```



### *Detailed summary of splits*

```

summary(fit)
## Call:
## rpart(formula = Kyphosis ~ Age + Number + Start, data = kyphosis,
##       method = "class")
##   n= 81
## 
##          CP nsplit rel error xerror      xstd
## 1 0.17647059     0 1.000000 1.00000 0.2155872

```

```

## 2 0.01960784      1 0.8235294 1.117647 0.2243268
## 3 0.01000000      4 0.7647059 1.117647 0.2243268
##
## Variable importance
## Start    Age Number
##     64      24      12
##
## Node number 1: 81 observations,    complexity param=0.1764706
##   predicted class=absent  expected loss=0.2098765  P(node) =1
##   class counts:  64    17
##   probabilities: 0.790 0.210
##   left son=2 (62 obs) right son=3 (19 obs)
## Primary splits:
##   Start < 8.5 to the right, improve=6.762330, (0 missing)
##   Number < 5.5 to the left,  improve=2.866795, (0 missing)
##   Age    < 39.5 to the left,  improve=2.250212, (0 missing)
## Surrogate splits:
##   Number < 6.5 to left,  agree=0.802, adj=0.158, (0 split)
##
## Node number 2: 62 observations,    complexity param=0.01960784
##   predicted class=absent
##   expected loss=0.09677419  P(node) =0.7654321
##   class counts:  56    6
##   probabilities: 0.903 0.097
##   left son=4 (29 obs) right son=5 (33 obs)
## Primary splits:
##   Start < 14.5 to the right, improve=1.0205280, (0 missing)
##   Age    < 55   to the left,  improve=0.6848635, (0 missing)
##   Number < 4.5   to the left,  improve=0.2975332, (0 missing)
## Surrogate splits:
##   Number < 3.5 to left,  agree=0.645, adj=0.241, (0 split)
##   Age < 16   to the left,  agree=0.597, adj=0.138, (0 split)
##
## Node number 3: 19 observations
##   predicted class=present
##   expected loss=0.4210526  P(node) =0.2345679
##   class counts:  8    11
##   probabilities: 0.421 0.579
##
## Node number 4: 29 observations
##   predicted class=absent  expected loss=0  P(node) =0.3580247
##   class counts:  29    0
##   probabilities: 1.000 0.000
##
## Node number 5: 33 observations,    complexity param=0.01960784
##   predicted class=absent  expected loss=0.1818182  P(node) =0.4074074
##   class counts:  27    6
##   probabilities: 0.818 0.182
##   left son=10 (12 obs) right son=11 (21 obs)
## Primary splits:

```

```

##   Age < 55 to the left, improve=1.2467530, (0 missing)
##   Start < 12.5 to the right, improve=0.2887701, (0 missing)
##   Number < 3.5 to the right, improve=0.1753247, (0 missing)
## Surrogate splits:
##   Start < 9.5 to the left, agree=0.758, adj=0.333, (0 split)
##   Number < 5.5 to the right, agree=0.697, adj=0.167, (0 split)
##
## Node number 10: 12 observations
## predicted class=absent expected loss=0 P(node) =0.1481481
##   class counts: 12 0
##   probabilities: 1.000 0.000
##
## Node number 11: 21 observations, complexity param=0.01960784
## predicted class=absent expected loss=0.2857143 P(node) =0.2592593
##   class counts: 15 6
##   probabilities: 0.714 0.286
##   left son=22 (14 obs) right son=23 (7 obs)
## Primary splits:
##   Age < 111 to the right, improve=1.71428600, (0 missing)
##   Start < 12.5 to the right, improve=0.79365080, (0 missing)
##   Number < 3.5 to the right, improve=0.07142857, (0 missing)
##
## Node number 22: 14 observations
## predicted class=absent expected loss=0.1428571 P(node) =0.1728395
##   class counts: 12 2
##   probabilities: 0.857 0.143
##
## Node number 23: 7 observations
## predicted class=present expected loss=0.4285714 P(node) =0.0864197
##   class counts: 3 4
##   probabilities: 0.429 0.571

```

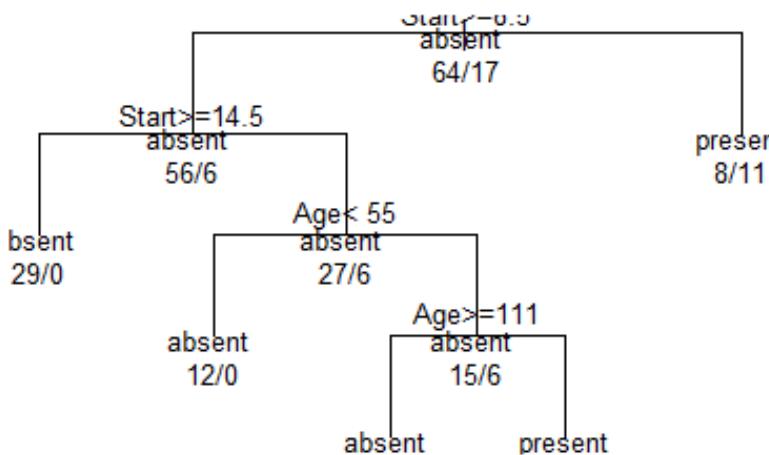
### Plot tree

```

plot(fit, uniform=TRUE, main= "Classification Tree for Kyp
hosis")
text(fit, use.n=TRUE, all=TRUE, cex=.8)

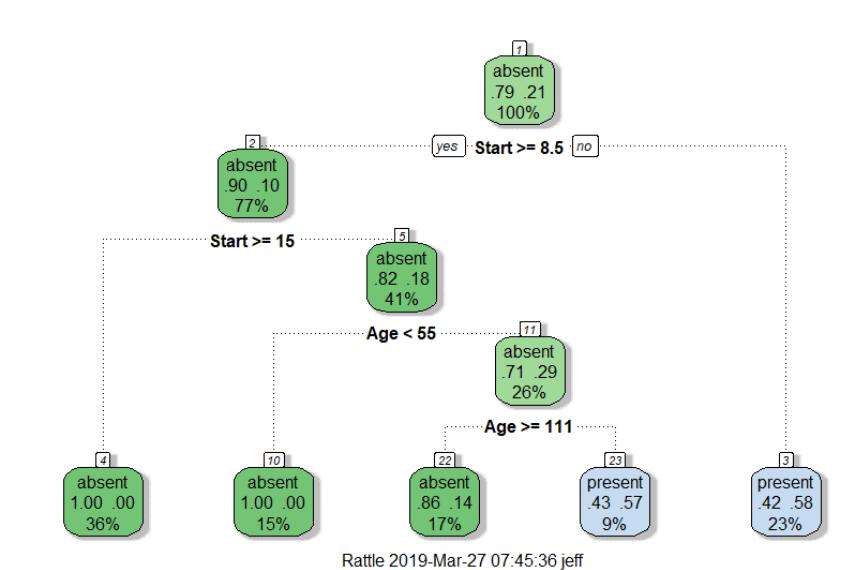
```

## Classification Tree for Kyphosis



Three additional packages, rpartplot, rattle and fancyRpartPlot can be downloaded to improve the graphic limitation we are exercising.

`fancyRpartPlot(fit)`

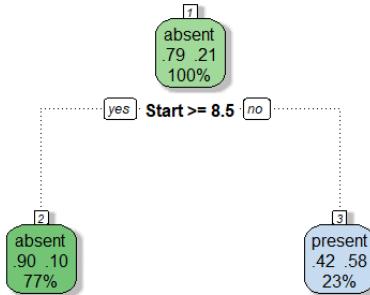


## Prune the tree

```
pfit <- prune(fit, cp=fit$cptable[which.min(fit$cptable[, "x  
error"])), "CP")
```

## Plot the pruned tree

```
pfit<-prune(fit, cp=0.1)  
plot(pfit, uniform=TRUE, main="Pruned Regression Tree for  
Mileage")  
text(pfit, use.n=TRUE, all=TRUE, cex=.8)  
fancyRpartPlot(pfit)
```

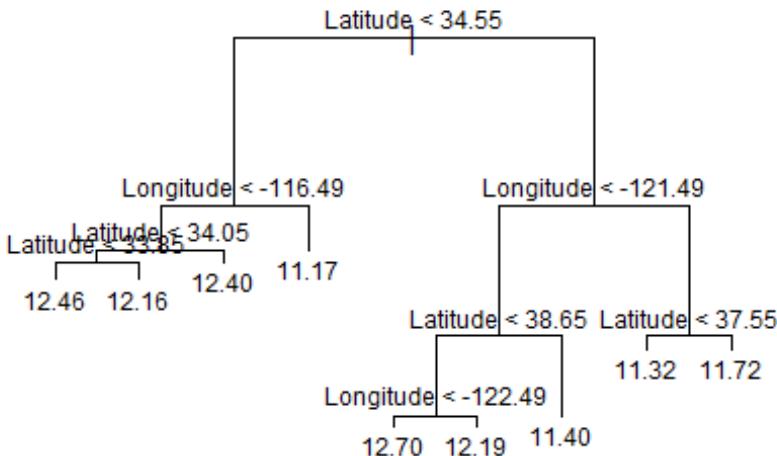


Rattle 2019-Mar-27 08:57:44 jeff

## Example 2: Regression Trees

Regression Trees like linear regression, outputs an expected value given a certain output. For demonstration purpose, we will train a regression model based on the California housing prices data from the 1990 Census.

```
library(tree)  
real.estate <- read.csv("C:/Users/Jeff/Documents/VIT_University/data/cadata.csv", header=TRUE)  
tree.model <- tree(log(HomeValue) ~ Longitude + Latitude,  
data=real.estate)  
plot(tree.model)  
text(tree.model, cex=.75)
```

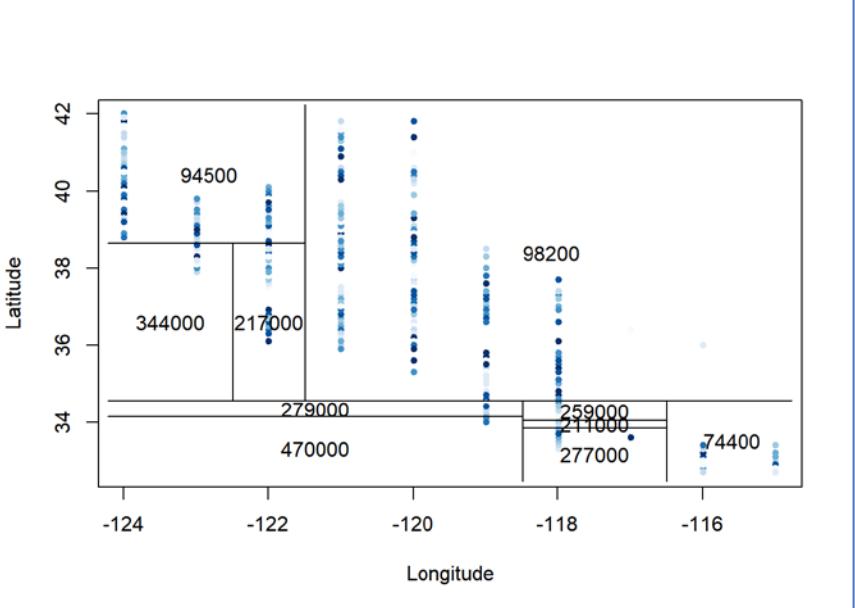


Notice that the leaf values represent the log of the price, since that was the way we represented the formula in the `tree()` function. (note: Knitr seems to output the wrong values above, check the results yourself in R). We can compare the predictions with the dataset (darker is more expensive) which seem to capture the global price trend:

```

tree.model <- tree(HomeValue ~ Longitude + Latitude, data=real.estate)
price.deciles <- quantile(real.estate$HomeValue, 0:10/10)
cut.prices   <- cut(real.estate$HomeValue, price.deciles,
include.lowest=TRUE)
plot(real.estate$Longitude, real.estate$Latitude, col=grey(10:2/11)[cut.prices], pch=20, xlab="Longitude", ylab="Latitude")
partition.tree(tree.model, ordvars=c("Longitude", "Latitude"),
add=TRUE)

```



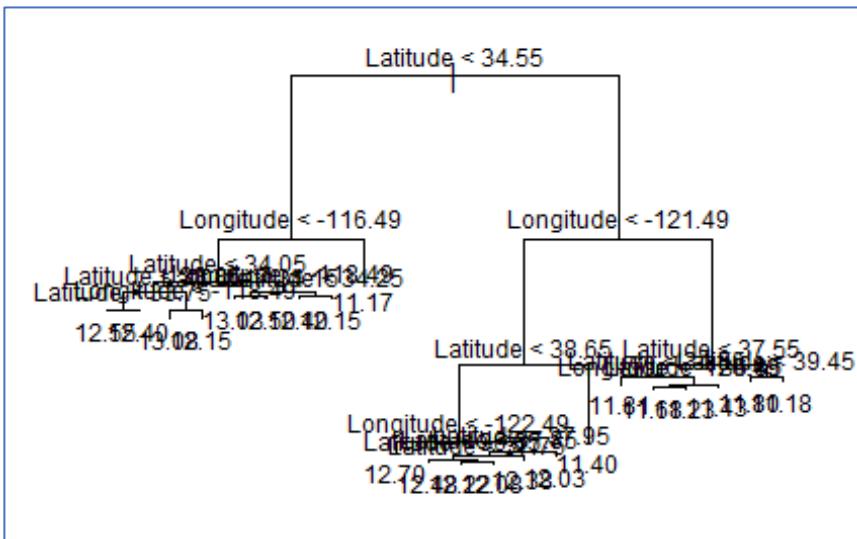
```
summary(tree.model)
##
## Regression tree:
## tree(formula = HomeValue ~ Longitude + Latitude, data =
real.estate)
## Number of terminal nodes: 10
## Residual mean deviance: 8.935e+09 = 9.495e+13 / 10630
## Distribution of residuals:
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## -276400 -61170 -22170       0   38760  401800
```

*Deviance means here the mean squared error.*

The flexibility of a tree is basically controlled by how many leaves they have, since that's how many cells they partition things into. The tree fitting function has a number of controls settings which limit how much it will grow | each node has to contain a certain number of points, and adding a node has to reduce the error by at least a certain amount. The default for the latter, min.dev, is 0.01; let's turn it down and see what happens:

```
tree.model2 <- tree(log(HomeValue) ~ Longitude + Latitude,
data=real.estate, mindev=0.001)
```

```
plot(tree.model2)
text(tree.model2, cex=.75)
```



Our tree has become illegible at this point. One remedy for this is to display the list of rules for the tree, using `asRules` from the rattle package.

```
asRules(tree.model2)
Rule number: 28 [log(HomeValue)=12.1639283212717]
cover=2894 (27%]
  Latitude< 34.55
  Longitude< -116.5
  Latitude< 34.05
  Latitude>=33.85

Rule number: 15 [log(HomeValue)=12.3991396359214]
cover=2492 (23%]
  Latitude< 34.55
  Longitude< -116.5
  Latitude>=34.05

Rule number: 22 [log(HomeValue)=12.1893649873663]
cover=1909 (18%]
  Latitude>=34.55
  Longitude< -121.5
  Latitude< 38.65
  Latitude>=-122.5

Rule number: 8 [log(HomeValue)=11.3240829661932]
cover=1337 (13%]
  Latitude>=34.55
```

```

Longitude>=- 121. 5
Latitude< 37. 55

Rule number: 29 [log(HomeValue)=12. 4577553098258
cover=893 (8%)]
    Latitude< 34. 55
    Longitude< -116. 5
    Latitude< 34. 05
    Latitude< 33. 85

Rule number: 10 [log(HomeValue)=11. 4027422571251
cover=496 (5%)]
    Latitude>=34. 55
    Longitude< -121. 5
    Latitude>=38. 65

Rule number: 9 [log(HomeValue)=11. 7191619424249
cover=350 (3%)]
    Latitude>=34. 55
    Longitude>=- 121. 5
    Latitude>=37. 55

Rule number: 23 [log(HomeValue)=12. 7034156567086
cover=148 (1%)]
    Latitude>=34. 55
    Longitude< -121. 5
    Latitude< 38. 65
    Longitude< -122. 5

Rule number: 6 [log(HomeValue)=11. 1691229504255
cover=118 (1%)]
    Latitude< 34. 55
    Longitude>=- 116. 5

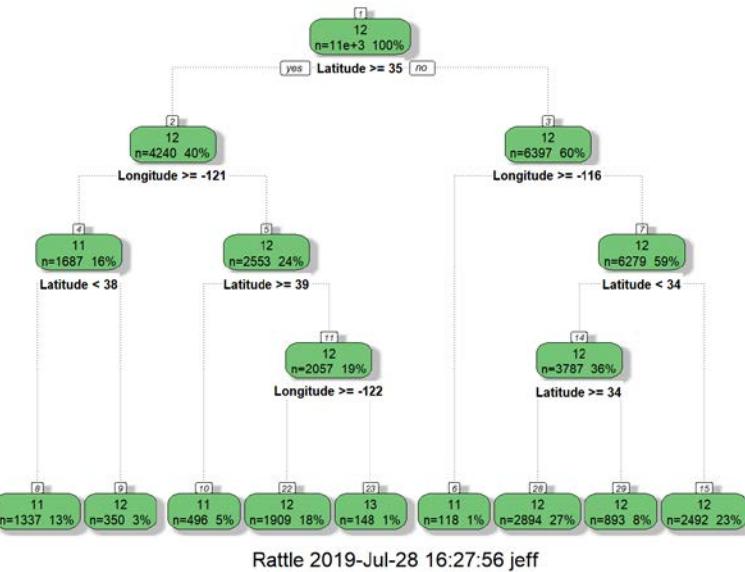
```

Another remedy is to model the problem using rpart and then applying FancyRpartPlot.

```

tree.model.2 <- rpart(log(HomeValue) ~ Longitude + Latitude,
                      data=real.estate)
fancyRpartPlot(tree.model.2)

```



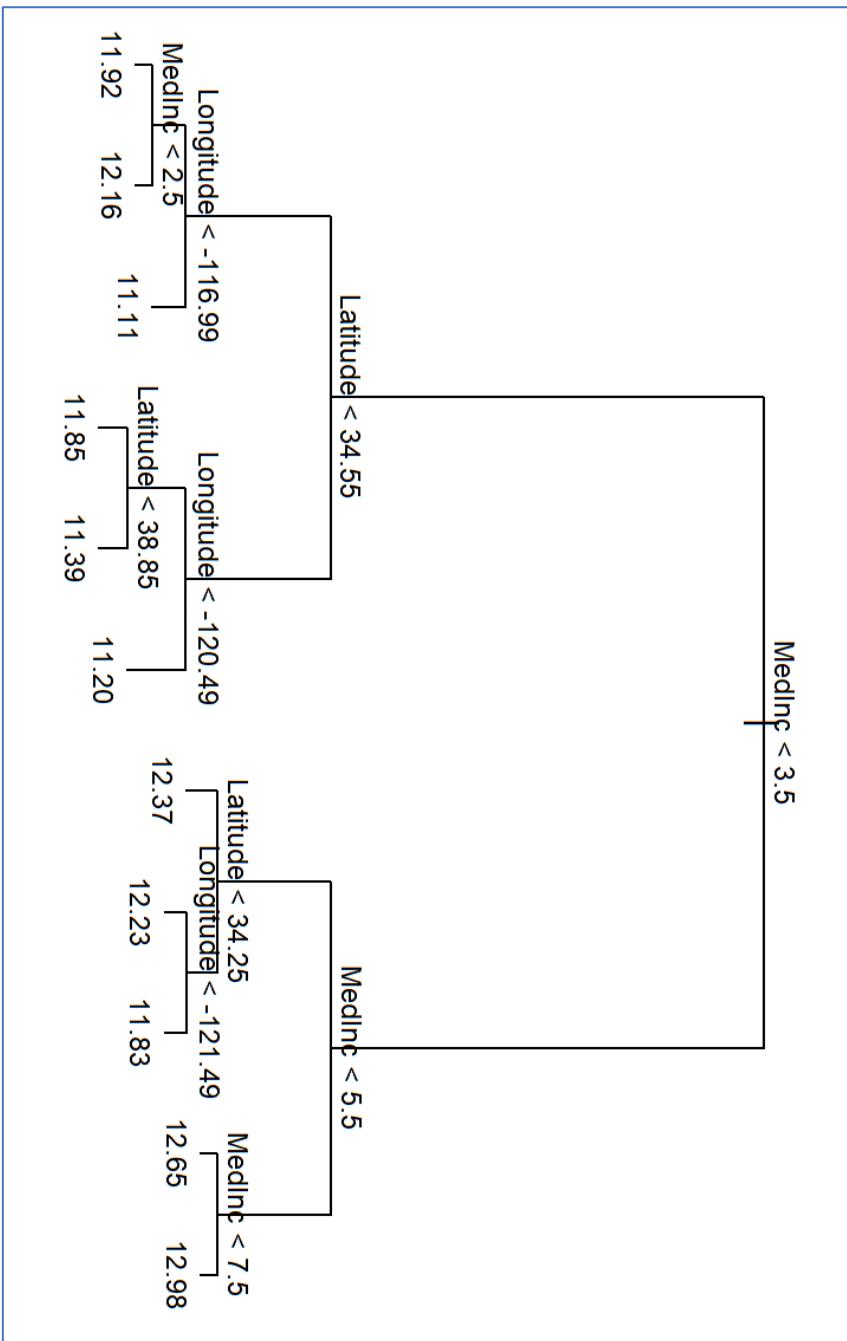
Rattle 2019-Jul-28 16:27:56 jeff

```
summary(tree.model2)

##
## Regression tree:
## tree(formula = log(HomeValue) ~ Longitude + Latitude, d
ata = real.estate,
##      mindev = 0.001)
## Number of terminal nodes: 22
## Residual mean deviance: 0.1614 = 1713 / 10620
## Distribution of residuals:
##      Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.78300 -0.26500 -0.04515  0.00000  0.25190  1.32800
```

It's obviously much finer-grained than the previous example (68 leafs against 12) and does a better job of matching the actual prices (lower error). Also, we can include all the variables, not only the latitude and longitude:

```
tree.model3 <- tree(log(HomeValue) ~ ., data=real.estate)
plot(tree.model3)
text(tree.model3, cex=.75)
```



```
summary(tree.model3)
```

```

## 
## Regression tree:
## tree(formula = log(HomeValue) ~ ., data = real.estate)
## Variables actually used in tree construction:
## [1] "MedInc"    "Latitude"   "Longitude"
## Number of terminal nodes: 11
## Residual mean deviance: 0.1218 = 1294 / 10630
## Distribution of residuals:
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## -2.75800 -0.23590 -0.01926 0.00000 0.19900 1.91800

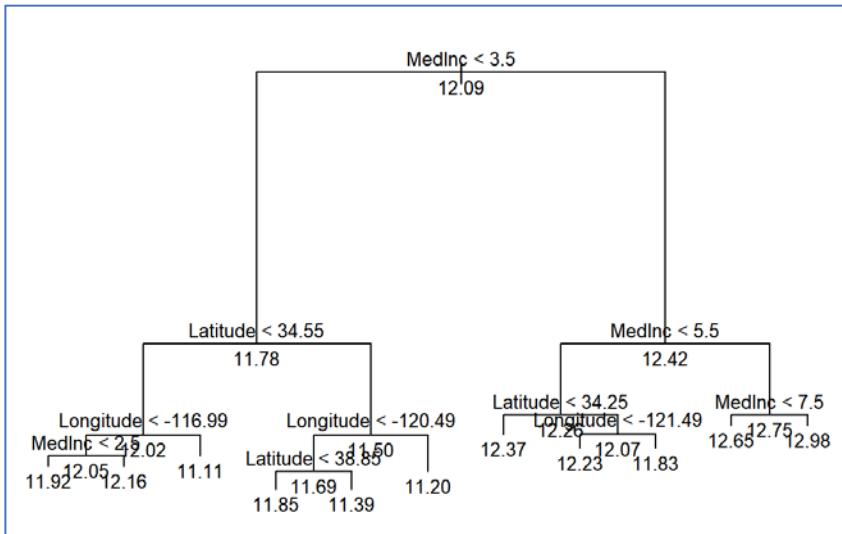
```

### Prune model 3

```

library(tree)
pfit3<-prune.tree(tree.model3, k = 0.1, best = 10, method
= "deviance", eps = 1e-3)
plot(pfit3, type = "proportional", main="Pruned Regression
Tree for Mileage")
text(pfit3, all=TRUE, cex=.8)

```



### Example 3: Regression Tree Example

In this example we will predict car mileage from price, country, reliability, and car type. We will continue to use rpart but will do so with a regression tree. The data frame is cu.summary.

```

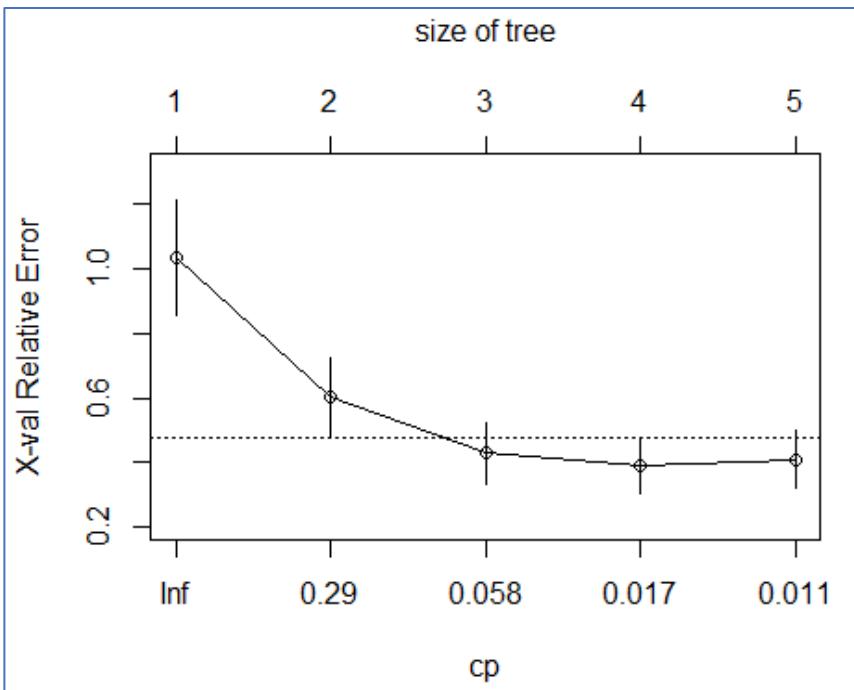
post(pfit, file = "C:/Users/Jeff/Documents/VIT_University/
ptree2.ps", title = "Pruned Regression Tree for Mileage")

```

```
library(rpart)
```

### **Grow tree**

```
fit2 <- rpart(Mileage~Price + Country + Reliability + Type  
, method="anova", data=cu.summary)  
printcp(fit2) # display the results  
##  
## Regression tree:  
## rpart(formula = Mileage ~ Price + Country + Reliability  
+ Type,  
##        data = cu.summary, method = "anova")  
##  
## Variables actually used in tree construction:  
## [1] Price Type  
##  
## Root node error: 1354.6/60 = 22.576  
##  
## n=60 (57 observations deleted due to missingness)  
##  
##          CP  nsplit rel_error  xerror      xstd  
## 1 0.622885      0  1.00000 1.03499 0.177516  
## 2 0.132061      1  0.37711 0.60315 0.123447  
## 3 0.025441      2  0.24505 0.42857 0.094780  
## 4 0.011604      3  0.21961 0.39018 0.084156  
## 5 0.010000      4  0.20801 0.40955 0.088301  
plotcp(fit2) # visualize cross-validation results
```



```

summary(fit2) # detailed summary of splits
## Call:
## rpart(formula = Mileage ~ Price + Country + Reliability + Type,
##       data = cu.summary, method = "anova")
## n=60 (57 observations deleted due to missingness)
##
##          CP nsplit rel.error      xerror      xstd
## 1 0.62288527      0 1.0000000 1.0349940 0.17751634
## 2 0.13206061      1 0.3771147 0.6031476 0.12344696
## 3 0.02544094      2 0.2450541 0.4285658 0.09477958
## 4 0.01160389      3 0.2196132 0.3901755 0.08415608
## 5 0.01000000      4 0.2080093 0.4095496 0.08830053
##
## Variable importance
##   Price     Type Country
##   48       42      10
##
## Node number 1: 60 observations,    complexity param=0.6228853
##   mean=24.58333, MSE=22.57639
##   left son=2 (48 obs) right son=3 (12 obs)
## Primary splits:
##   Price < 9446.5 to the right, improve=0.6228853, (0 missing)

```

```

## Type splits as LLLRLL, improve=0.5044405, (0 missing)
## Reliability splits as LLLRR, improve=0.1263005, (11 missing)
## Country splits as --LRLRRRLL, improve=0.1243525, (0 missing)
## Surrogate splits:
##   Type splits as LLLRLL, agree=0.950, adj=0.750, (0 split)
##   Country splits as --LLLLRRL, agree=0.833, adj=0.167, (0 split)
##
## Node number 2: 48 observations, complexity param=0.1320606
## mean=22.70833, MSE=8.498264
## left son=4 (23 obs) right son=5 (25 obs)
## Primary splits:
##   Type splits as RLLRRL, improve=0.43853830, (0 missing)
##   Price < 12154.5 to the right, improve=0.25748500, (0 missing)
##   Country splits as --RRLRL-LL, improve=0.13345700, (0 missing)
##   Reliability splits as LLLRR, improve=0.01637086, (10 missing)
##   Surrogate splits:
##     Price < 12215.5 to the right, agree=0.812, adj=0.609, (0 split)
##     Country splits as --RRLRL-RL, agree=0.646, adj=0.261, (0 split)
##
## Node number 3: 12 observations
## mean=32.08333, MSE=8.576389
##
## Node number 4: 23 observations, complexity param=0.02544094
## mean=20.69565, MSE=2.907372
## left son=8 (10 obs) right son=9 (13 obs)
## Primary splits:
##   Type splits as -LR--L, improve=0.515359600, (0 missing)
##   Price < 14962 to the left, improve=0.131259400, (0 missing)
##   Country splits as ----L-R--R, improve=0.007022107, (0 missing)
##   Surrogate splits:
##     Price < 13572 to the right, agree=0.609, adj=0.1, (0 split)
##
## Node number 5: 25 observations, complexity param=0.01160389
## mean=24.56, MSE=6.4864
## left son=10 (14 obs) right son=11 (11 obs)
## Primary splits:
##   Price < 11484.5 to the right, improve=0.09693168, (0 missing)
##   Reliability splits as LLRRR, improve=0.07767167, (4 missing)
##   Type splits as L--RR-, improve=0.04209834, (0 missing)
##   Country splits as --LRRR--LL, improve=0.02201687, (0 missing)
##   Surrogate splits:
##     Country splits as --LLLL--LR, agree=0.80, adj=0.545, (0 split)
##     Type splits as L--RL-, agree=0.64, adj=0.182, (0 split)
##
## Node number 8: 10 observations
## mean=19.3, MSE=2.21

```

```

## 
## Node number 9: 13 observations
##   mean=21.76923, MSE=0.7928994
##
## Node number 10: 14 observations
##   mean=23.85714, MSE=7.693878
##
## Node number 11: 11 observations
##   mean=25.45455, MSE=3.520661

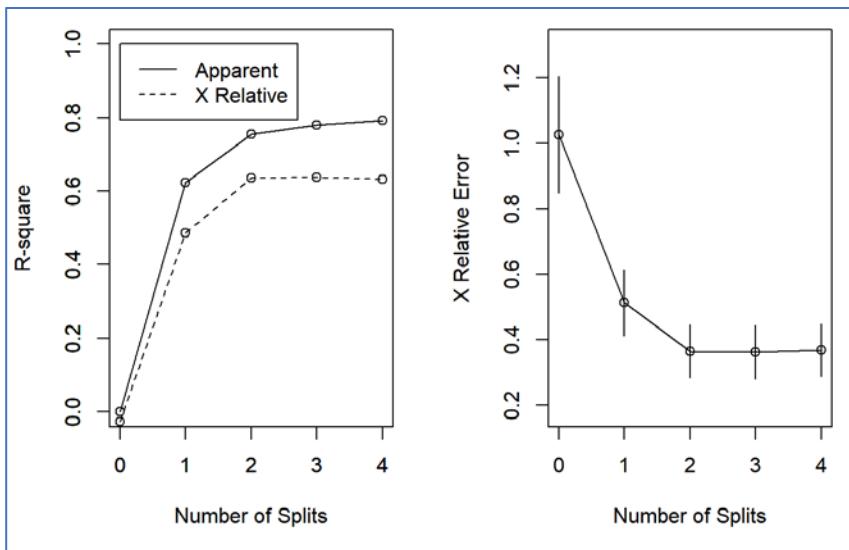
```

### **Create additional plots**

```

par(mfrow=c(1,2)) # two plots on one page
rsq.rpart(fit2) # visualize cross-validation results
##
## Regression tree:
## rpart(formula = Mileage ~ Price + Country + Reliability
+ Type,
##       data = cu.summary, method = "anova")
##
## Variables actually used in tree construction:
## [1] Price Type
##
## Root node error: 1354.6/60 = 22.576
##
## n=60 (57 observations deleted due to missingness)
##
##          CP nsplit rel.error xerror      xstd
## 1 0.622885     0  1.00000 1.03499 0.177516
## 2 0.132061     1  0.37711 0.60315 0.123447
## 3 0.025441     2  0.24505 0.42857 0.094780
## 4 0.011604     3  0.21961 0.39018 0.084156
## 5 0.010000     4  0.20801 0.40955 0.088301

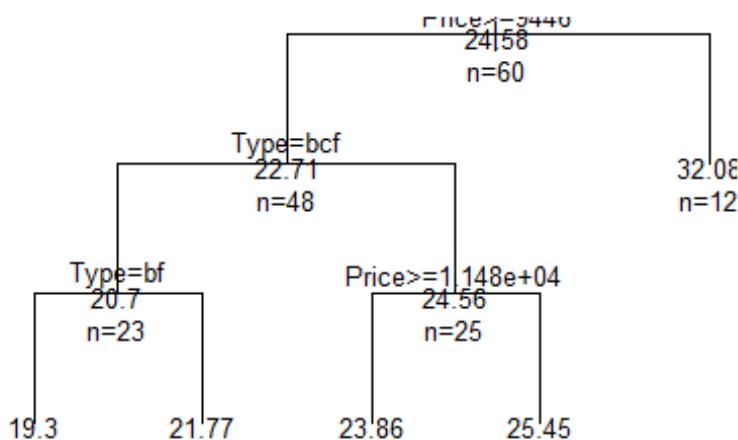
```



### plot tree

```
plot(fit2, uniform=TRUE, main= "Regression Tree for Mileage")
text(fit2, use.n=TRUE, all=TRUE, cex=.8)
```

## Regression Tree for Mileage



### Create attractive postscript plot of tree

```
post(fit2, file = "C:/Users/Jeff/Documents/VIT_University/  
ptree4.ps", title = "Regression Tree for Mileage")
```

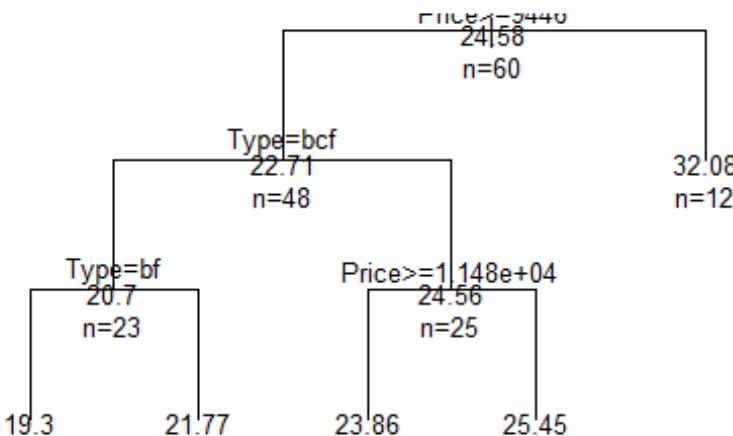
### Prune the tree

```
pfit2<- prune(fit2, cp=0.01160389) # from cptable
```

### Plot the pruned tree

```
plot(pfit2, uniform=TRUE, main= "Pruned Regression Tree fo  
r Mileage")  
text(pfit2, use.n=TRUE, all=TRUE, cex=.8)
```

## Pruned Regression Tree for Mileage



```
post(pfit2, file = "C:/Users/Jeff/Documents/VIT_University/  
ptree5.ps", title = "Pruned Regression Tree for Mileage")
```

### Example 4: Classification Trees using tree

Classification trees output the predicted class for a given sample. We will use the R package `tree`: Classification and Regression Trees Let's use here the iris dataset (and split it into train and test sets):

```
set.seed(101)  
alpha      <- 0.7 # percentage of training set  
inTrain   <- sample(1:nrow(iris), alpha * nrow(iris))
```

```

train.set <- iris[inTrain,]
test.set <- iris[-inTrain,]

```

There are two options for the output:

- + Point prediction: simply gives the predicted class
- + Distributional prediction: gives a probability for each class

```

library(tree)
tree.model <- tree(Species ~ Sepal.Width + Petal.Width, data=train.set)
tree.model

## node), split, n, deviance, yval, (yprob)
##      * denotes terminal node
## 1) root 105 230.200 versicolor( 0.33333 0.36190 0.30476 )
## 2) Petal.Width < 0.8 35  0.000 setosa(1.00 0.00 0.00) *
## 3) Petal.Width > 0.8 70  96.530 versicolor(0.0 0.543 0.457)
## 6) Petal.Width < 1.7 40  21.310 versicolor(0.0 0.925 0.075)
## 2) Petal.Width < 1.35 20  0.000 versicolor(0.0 1.00 0.00)*
## 13) Petal.Width > 1.35 20  16.91 versicolor(0.0 0.85 0.150)
## 26) Sepal.Width < 3.05 14  14.55 versicolor(0.0 0.79 0.2144)*
## 27) Sepal.Width > 3.05 6  0.000 versicolor(0.00 1.00 0.00)*
## 7) Petal.Width > 1.7 30  8.769 virginica(0.00 0.033 0.967)
## 14) Petal.Width < 1.85 8  6.028 virginica(0.00 0.125 0.875)*
## 15) Petal.Width > 1.85 22  0.000 virginica(0.00 0.00 1.00)*

summary(tree.model)

##
## Classification tree:
## tree(formula = Species ~ Sepal.Width + Petal.Width, data = train.set)
## Number of terminal nodes:  6
## Residual mean deviance:  0.2078 = 20.58 / 99
## Misclassification error rate: 0.0381 = 4 / 105

```

### *Distributional prediction*

```

my.prediction <- predict(tree.model, test.set) # gives the
probability for each class
head(my.prediction)

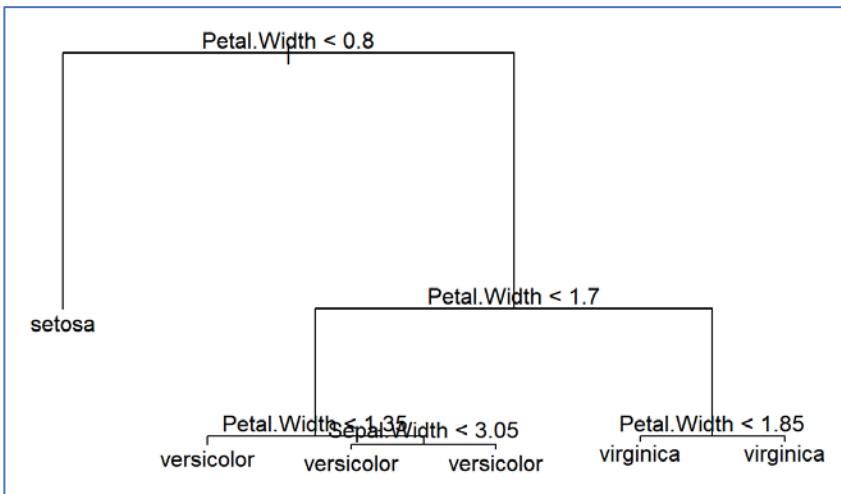
##      setosa versicolor virginica
## 5          1          0          0
## 10         1          0          0
## 12         1          0          0
## 15         1          0          0
## 16         1          0          0
## 18         1          0          0

```

## Point prediction

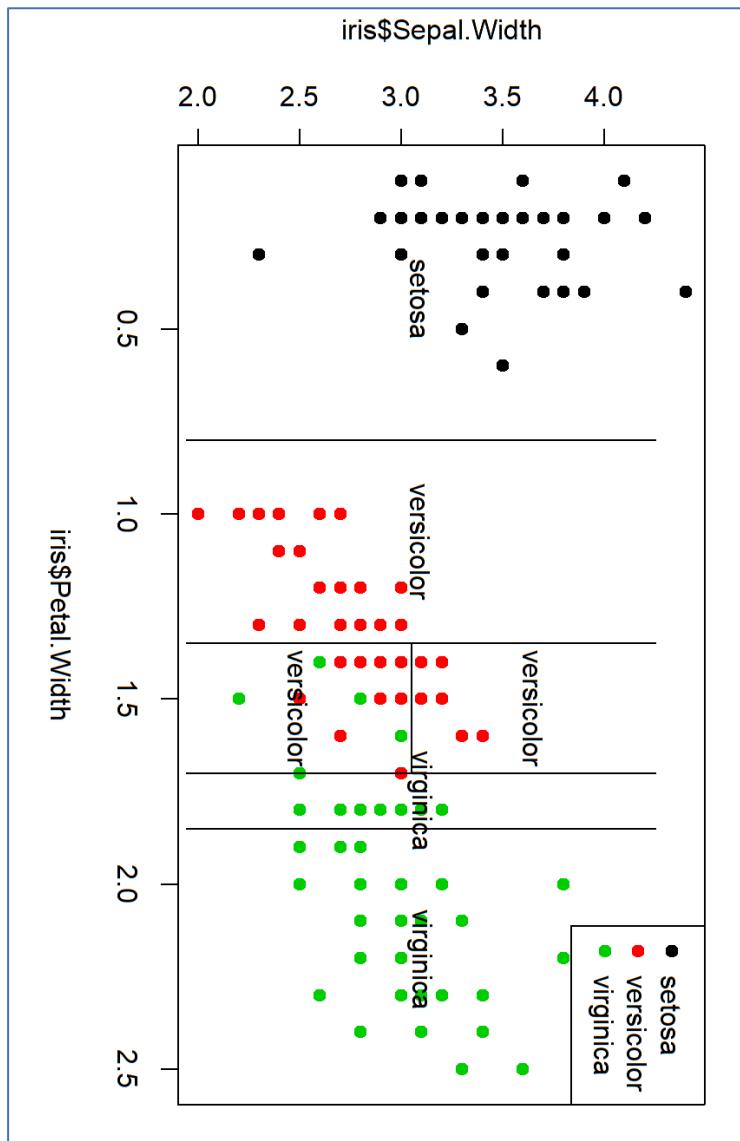
Let's translate the probability output to categorical output

```
maxidx <- function(arr) {  
  return(which(arr == max(arr)))  
}  
idx <- apply(my.prediction, c(1), maxidx)  
prediction <- c('setosa', 'versicolor', 'virginica')[idx]  
table(prediction, test.set$Species)  
##  
## prediction      setosa    versicolor   virginica  
##   setosa          15           0           0  
##   versicolor       0           11          1  
##   virginica        0           1          17  
plot(tree.model)  
text(tree.model)
```



## Another way to show the data:

```
plot(iris$Petal.Width, iris$Sepal.Width, pch=19, col=as.numeric(iris$Species))  
partition.tree(tree.model, label="Species", add=TRUE)  
legend("topright", legend=unique(iris$Species), col=unique(as.numeric(iris$Species)), pch=19)
```



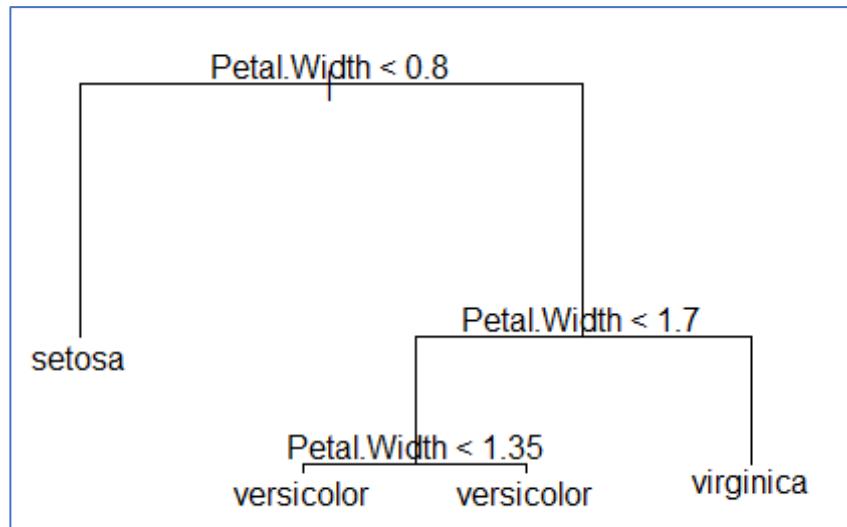
```

summary(tree.model)
##
## Classification tree:
## tree(formula = Species ~ Sepal.Width + Petal.Width, dat
a = train.set)
## Number of terminal nodes: 6
## Residual mean deviance: 0.2078 = 20.58 / 99
## Misclassification error rate: 0.0381 = 4 / 105

```

We can prune the tree to prevent overfitting. The next function `prune.tree()` allows us to choose how many leafs we want the tree to have, and it returns the best tree with that size. The argument `newdata` accepts new input for making the prune decision. If new data is not given, the method uses the original dataset from which the tree model was built. For classification trees we can also use argument `method="misclass"` so that the pruning measure should be the number of misclassifications.

```
pruned.tree <- prune.tree(tree.model, best=4)  
plot(pruned.tree)  
text(pruned.tree)
```



```
pruned.prediction <- predict(pruned.tree, test.set, type="class") # give the predicted class  
table(pruned.prediction, test.set$Species)  
##  
## pruned.prediction setosa versicolor virginica  
##      setosa        15         0         0  
##      versicolor     0        11         1  
##      virginica      0         1        17
```

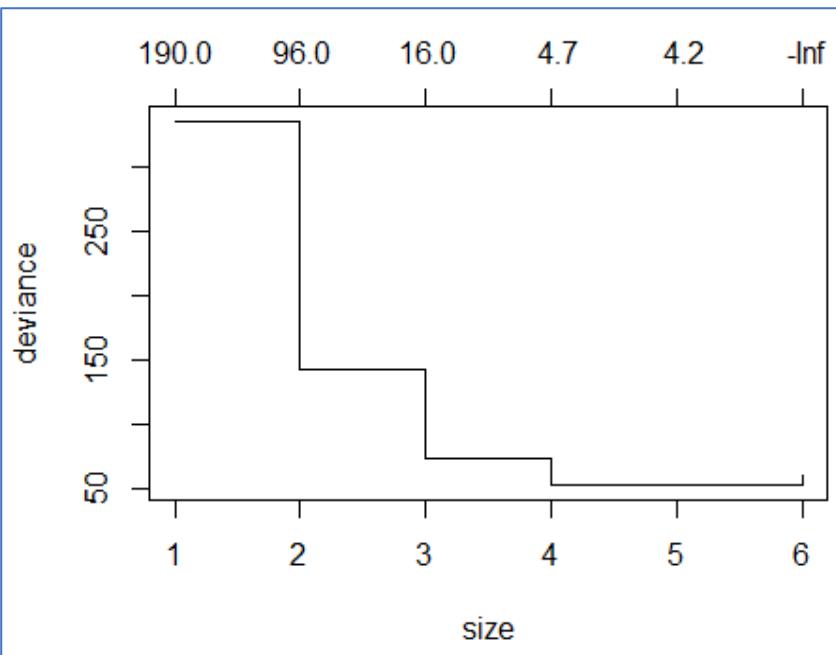
This package can also do K-fold cross-validation using `cv.tree()` to find the best tree: here, let's use all the variables and all the samples

```
tree.model <- tree(Species ~ ., data=iris)  
summary(tree.model)
```

```

## Classification tree:
## tree(formula = Species ~ ., data = iris)
## Variables actually used in tree construction:
## [1] "Petal.Length" "Petal.Width"  "Sepal.Length"
## Number of terminal nodes:  6
## Residual mean deviance:  0.1253 = 18.05 / 144
## Misclassification error rate: 0.02667 = 4 / 150
cv.model <- cv.tree(tree.model)
plot(cv.model)

```



```

cv.model$dev # gives the deviance for each K (small is better)
## [1] 59.08966 52.62454 52.58414 73.33613 142.82350 3
36.21863
best.size <- cv.model$size[which(cv.model$dev==min(cv.mode
l$dev))] # which size is better?
best.size
## [1] 4

```

Let's refit the tree model (the number of leafs will be no more than best.size)

```

cv.model.pruned <- prune.misclass(tree.model, best=best.size)
summary(cv.model.pruned)
##
## Classification tree:
## snip.tree(tree = tree.model, nodes = c(7L, 12L))
## Variables actually used in tree construction:
## [1] "Petal.Length" "Petal.Width"
## Number of terminal nodes: 4
## Residual mean deviance: 0.1849 = 26.99 / 146
## Misclassification error rate: 0.02667 = 4 / 150

```

The misclassification rate has just slightly increased with the pruning of the tree.

## Purity and Entropy

Let's define a subset to be completely pure if it contains only a single class. For example, if a subset contains only poisonous mushrooms, it is completely pure. In R, assuming the last column contains the class (i.e. the category to be predicted), this can be written as:

```

IsPure <- function(data) {
  length(unique(data[,ncol(data)])) == 1
}

```

*The entropy is a measure of the purity of a dataset.*

```

Entropy <- function( vls ) {
  res <- vls/sum(vls) * log2(vls/sum(vls))
  res[vls == 0] <- 0
  -sum(res)
}

```

If a dataset is completely pure, then it has entropy 0:

```

Entropy(c(10, 0))
## [1] 0
Entropy(c(0, 10))
## [1] 0

```

If a set contains 5 poisonous and 5 edible mushrooms, the entropy becomes 1, as the purity is at its lowest:

```

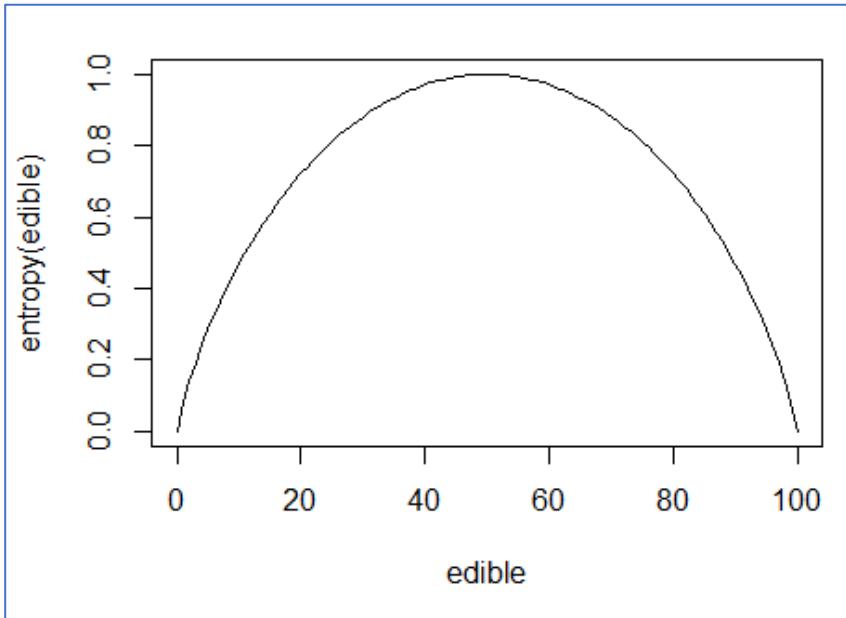
Entropy(c(5, 5))

```

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

We can plot the entropy as a function of the number of edible mushrooms in a set of, say, 100 mushrooms:

```
entropy <- function(edible) Entropy(c(edible, 100 - edible))
entropy <- Vectorize(entropy)
curve(entropy, from = 0, to = 100, xname = 'edible')
```



## Information Gain

Mathematically, the information gain IG is defined as:  
$$IG(T,a) = H(T) - \sum_{v \in \text{vals}(a)} \left( \frac{\text{abs}(\{x \in T | x_a = v\})}{\text{abs}(T)} \right) H(\{x \in T | x_a = v\})$$
 In words, the information gain measures the difference between the entropy before the split, and the weighted sum of the entropies after the split: So, let's rewrite that in R:

```
InformationGain <- function(tbl) {
  tbl <- as.data.frame.matrix(tbl)
  entropyBefore <- Entropy(colSums(tbl))
  s <- rowSums(tbl)
  entropyAfter <- sum(s / sum(s) * apply(tbl, MARGIN = 1,
    FUN = Entropy))
  informationGain <- entropyBefore - entropyAfter
```

```

    return (informationGain)
}

```

For example, using the mushroom data set:

```

library(data.tree)
data(mushroom)
tble <- table(mushroom[,c('color', 'edibility')])
tble
##          edibility
## color   edible toxic
## brown      2      0
## green      1      0
## red       1      1
InformationGain(tble)
## [1] 0.3219281
InformationGain(table(mushroom[,c('size', 'edibility')]))
## [1] 0.1709506
InformationGain(table(mushroom[,c('points', 'edibility')]))
)
## [1] 0.3219281

```

## ID3 Algorithm

```

TrainID3 <- function(node, data) {
  node$obsCount <- nrow(data)
  #if the data-set is pure (e.g. all toxic), then
  if (IsPure(data)) {
    #construct a Leaf having the name of the pure feature
    (e.g. 'toxic')
    child <- node$AddChild(unique(data[,ncol(data)]))
    node$feature <- tail(names(data), 1)
    child$obsCount <- nrow(data)
    child$feature <- ''
  } else {
    #choose the feature with the highest information gain (
    (e.g. 'color')
    ig <- sapply(colnames(data)[-ncol(data)],
                 function(x) InformationGain(
                   table(data[,x], data[,ncol(data)])
                 )
               )
    feature <- names(ig)[ig == max(ig)][1]
    childObs <- split(data[,!names(data) %in% feature]),
    data[,feature], drop = TRUE)
  }
}

```

```

for(i in 1:length(childObs)) {
  #construct a child having the name of that feature value (e.g. 'red')
  child <- node$AddChild(names(childObs)[i])

  #call the algorithm recursively on the child and the subset
  TrainID3(child, childObs[[i]])
}
}
}

```

### Training with data

```

library(data.tree)
data(mushroom)
tree <- Node$new("mushroom")
TrainID3(tree, mushroom)
print(tree, "feature", "obsCount")
##               levelName   feature obsCount
## 1   mushroom                         5
## 2     |--brown           edibility    2
## 3       |   |--edible
## 4       |   |--green          edibility    1
## 5       |       |   |--edible
## 6       |       |   |--red
## 7         |       |   |--large          edibility    1
## 8         |       |       |   |--edible
## 9         |       |   |--small          edibility    1
## 10        |       |       |   |--toxic

```

### Predict Function

```

Predict <- function(tree, features) {
  if (tree$children[[1]]$isLeaf) return (tree$children[[1]]$name)
  child <- tree$children[[features[[tree$feature]]]]
  return (Predict(child, features))
}

```

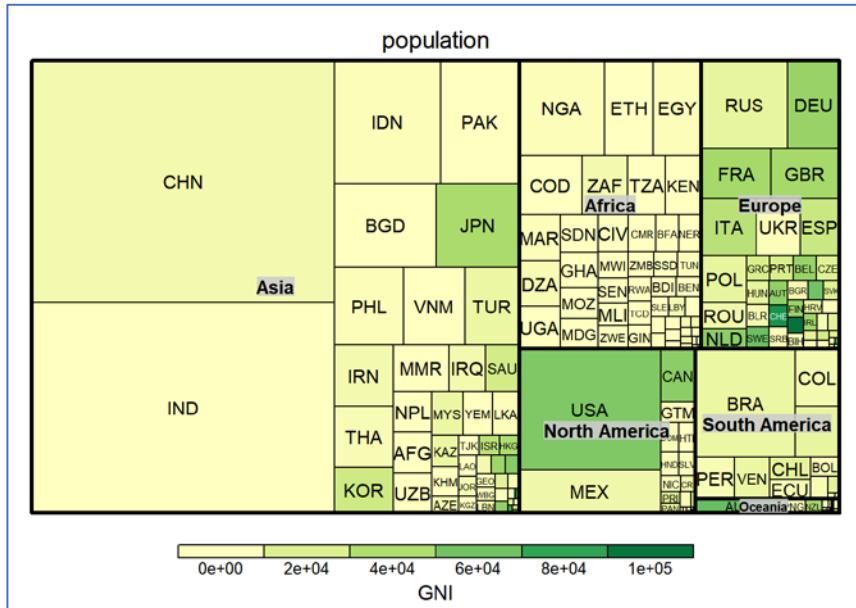
```
#Predicting some classes Predict(tree, c(color = 'red', size = 'large',
points = 'yes'))
```

### World PopulationTreeMap

- convert a data.frame to a data.tree structure
- navigate a tree and locate specific nodes

- use Aggregate and Cumulate
- manipulate an existing tree, e.g. by using the Prune method

```
library(treemap)
data(GNI2014)
treemap(GNI2014,
        index=c("continent", "iso3"),
        vSize="population",
        vColor="GNI",
        type="value")
```



## Convert from data.frame

First, let's convert the population data into a data.tree structure:

```
library(data.tree)
GNI2014$continent <- as.character(GNI2014$continent)
GNI2014$pathString <- paste("world", GNI2014$continent, GNI2014$country, sep = "/")
tree <- as.Node(GNI2014[,])
print(tree, pruneMethod = "dist", limit = 20)
##                                     levelName
## 1  world
## 2    |--North America
## 3      |--Bermuda
```

```

## 4      |--United States
## 5          |--... 22 nodes w/ 0 sub
## 6      --Europe
## 7          |--Norway
## 8          |--Switzerland
## 9              |--... 39 nodes w/ 0 sub
## 10     --Asia
## 11         |--Qatar
## 12         |--Macao SAR, China
## 13             |--... 45 nodes w/ 0 sub
## 14     --Oceania
## 15         |--Australia
## 16         |--New Zealand
## 17             |--... 11 nodes w/ 0 sub
## 18     --South America
## 19         |--Uruguay
## 20         |--Chile
## 21             |--... 10 nodes w/ 0 sub
## 22     --Seven seas (open ocean)
## 23         |--Seychelles
## 24         |--Mauritius
## 25             |--... 1 nodes w/ 0 sub
## 26     --Africa
## 27         |--... 48 nodes w/ 0 sub

```

We can also navigate the tree to find the population of a specific country. Luckily, RStudio is quite helpful with its code completion (use CTRL + SPACE):

```

tree$Europe$Switzerland$population
## [1] 7604467

```

Or, we can look at a sub-tree:

```

northAm <- tree`North America`
northAm$Sort("GNI", decreasing = TRUE)
## Warning: 'northAm$Sort' is deprecated.
## Use 'Sort(node, ...)' instead.
## See help("Deprecated")
print(northAm, "iso3", "population", "GNI", limit = 12)
##                                     levelName iso3 population      GNI
## 1  North America                         NA        NA
## 2      |--Bermuda                      BMU    67837 106140
## 3      |--United States                  USA 313973000  55200
## 4      |--Canada                       CAN 33487208  51630

```

```

## 5  |--Bahamas, The          BHS    309156  20980
## 6  |--Trinidad and Tobago  TTO    1310000 20070
## 7  |--Puerto Rico           PRI    3971020 19310
## 8  |--Barbados              BRB    284589  15310
## 9  |--St. Kitts and Nevis   KNA    40131   14920
## 10 |--Antigua and Barbuda  ATG    85632   13300
## 11 |--Panama                PAN    3360474 11130
## 12 °--... 14 nodes w/ 0 sub      NA

```

Or, we can find out what is the country with the largest GNI:

```

maxGNI <- Aggregate(tree, "GNI", max)
#same thing, in a more traditional way:
maxGNI <- max(sapply(tree$leaves, function(x) x$GNI))
tree$Get("name", filterFun = function(x) x$isLeaf && x$GNI
== maxGNI)
##   Bermuda
## "Bermuda"

```

### *Aggregate and Cumulate*

We aggregate the population. For non-leaves, this will recursively iterate through children, and cache the result in the population field.

```

tree$Do(function(x) {
  x$population <- Aggregate(node = x,
    attribute = "population",
    aggFun = sum)
},
traversal = "post-order")

```

Next, we sort each node by population:

```

tree$Sort(attribute = "population", decreasing = TRUE, recursive = TRUE)
## Warning: 'tree$Sort' is deprecated.
## Use 'Sort(node, ...)' instead.
## See help("Deprecated")

```

Finally, we cumulate among siblings, and store the running sum in an attribute called cumPop:

```

tree$Do(function(x) x$cumPop <- Cumulate(x, "population",
sum))

```

The tree now looks like this:

```

print(tree, "population", "cumPop", pruneMethod = "dist",
      limit = 20)
##                                     levelName population      cumPop
## 1   world                           6683146875 6683146875
## 2   |--Asia                          4033277009 4033277009
## 3   |---China                        1338612970 1338612970
## 4   |---India                         1166079220 2504692190
## 5   |---.... 45 nodes w/ 0 sub       NA          NA
## 6   |--Africa                         962382035 4995659044
## 7   |---Nigeria                      149229090 149229090
## 8   |---Ethiopia                      85237338 234466428
## 9   |---.... 46 nodes w/ 0 sub       NA          NA
## 10  |--Europe                        728669949 5724328993
## 11  |---Russian Federation           140041247 140041247
## 12  |---Germany                      82329758 222371005
## 13  |---.... 39 nodes w/ 0 sub       NA          NA
## 14  |--North America                 528748158 6253077151
## 15  |---United States                313973000 313973000
## 16  |---Mexico                        111211789 425184789
## 17  |---.... 22 nodes w/ 0 sub       NA          NA
## 18  |--South America                 394352338 6647429489
## 19  |---Brazil                        198739269 198739269
## 20  |---Colombia                     45644023 244383292
## 21  |---.... 10 nodes w/ 0 sub       NA          NA
## 22  |--Oceania                       33949312 6681378801
## 23  |---Australia                    21262641 21262641
## 24  |---Papua New Guinea             6057263 27319904
## 25  |---.... 11 nodes w/ 0 sub       NA          NA
## 26  |---Seven seas (open ocean)     1768074 6683146875
## 27  |---.... 3 nodes w/ 0 sub       NA          NA

```

## Prune

The previous steps were done to define our threshold: big countries should be displayed, while small ones should be grouped together. This lets us define a pruning function that will allow a maximum of 7 countries per continent, and that will prune all countries making up less than 90% of a continent's population. We would like to store the original number of countries for further use:

```
tree$Do(function(x) x$origCount <- x$count)
```

We are now ready to prune. This is done by defining a pruning function, returning 'FALSE' for all countries that should be combined:

```

myPruneFun <- function(x, cutoff = 0.9, maxCountries = 7)
{
  if (isNotLeaf(x)) return (TRUE)
  if (x$position > maxCountries) return (FALSE)
  return (x$cumPop < (x$parent$population * cutoff))
}

```

We clone the tree, because we might want to play around with different parameters:

```

treeClone <- Clone(tree, pruneFun = myPruneFun)
print(treeClone$Oceania, "population", pruneMethod = "simple", limit = 20)
##           levelName population
## 1 Oceania            33949312
## 2   |--Australia      21262641
## 3     |--Papua New Guinea 6057263

```

Finally, we need to sum countries that we pruned away into a new “Other” node:

```

treeClone$Do(function(x) {
  missing <- x$population - sum(sapply(x$children, function(x) x$population))
  other <- x$AddChild("Other")
  other$iso3 <- paste0("OTH(", x$origCount, ")")
  other$country <- "Other"
  other$continent <- x$name
  other$GNI <- 0
  other$population <- missing
},
filterFun = function(x) x$level == 2
)
print(treeClone$Oceania, "population", pruneMethod = "simple", limit = 20)
##           levelName population
## 1 Oceania            33949312
## 2   |--Australia      21262641
## 3     |--Papua New Guinea 6057263
## 4       |--Other        6629408

```

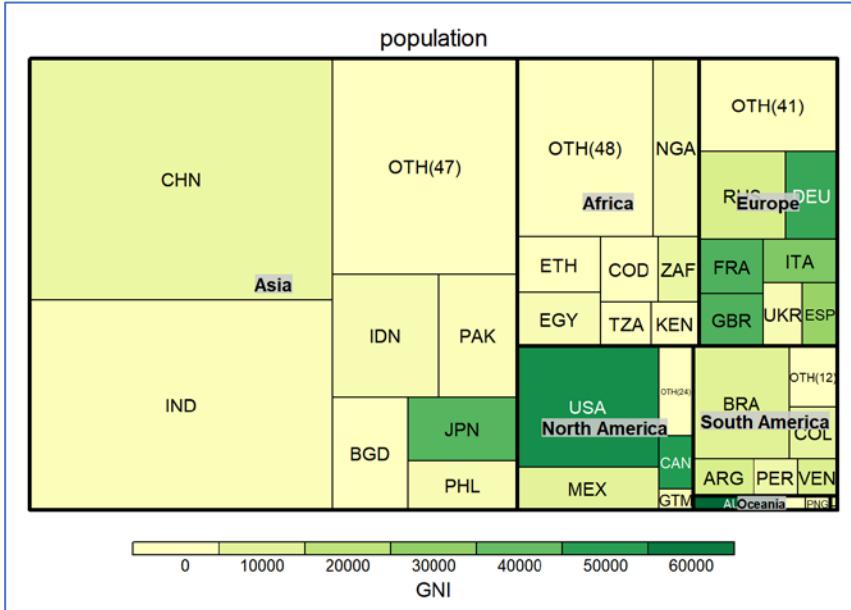
## *Plotting the treemap*

In order to plot the treemap, we need to convert the data.tree structure back to a data.frame:

```

df <- ToDataFrameTable(treeClone, "iso3", "country", "continent",
                       "population", "GNI")
treemap(df,
        index=c("continent", "iso3"),
        vSize="population",
        vColor="GNI",
        type="value")

```



### Plot as dendrogram

Just for fun, and for no reason other than to demonstrate conversion to dendrogram, we can plot this in a very unusual way:

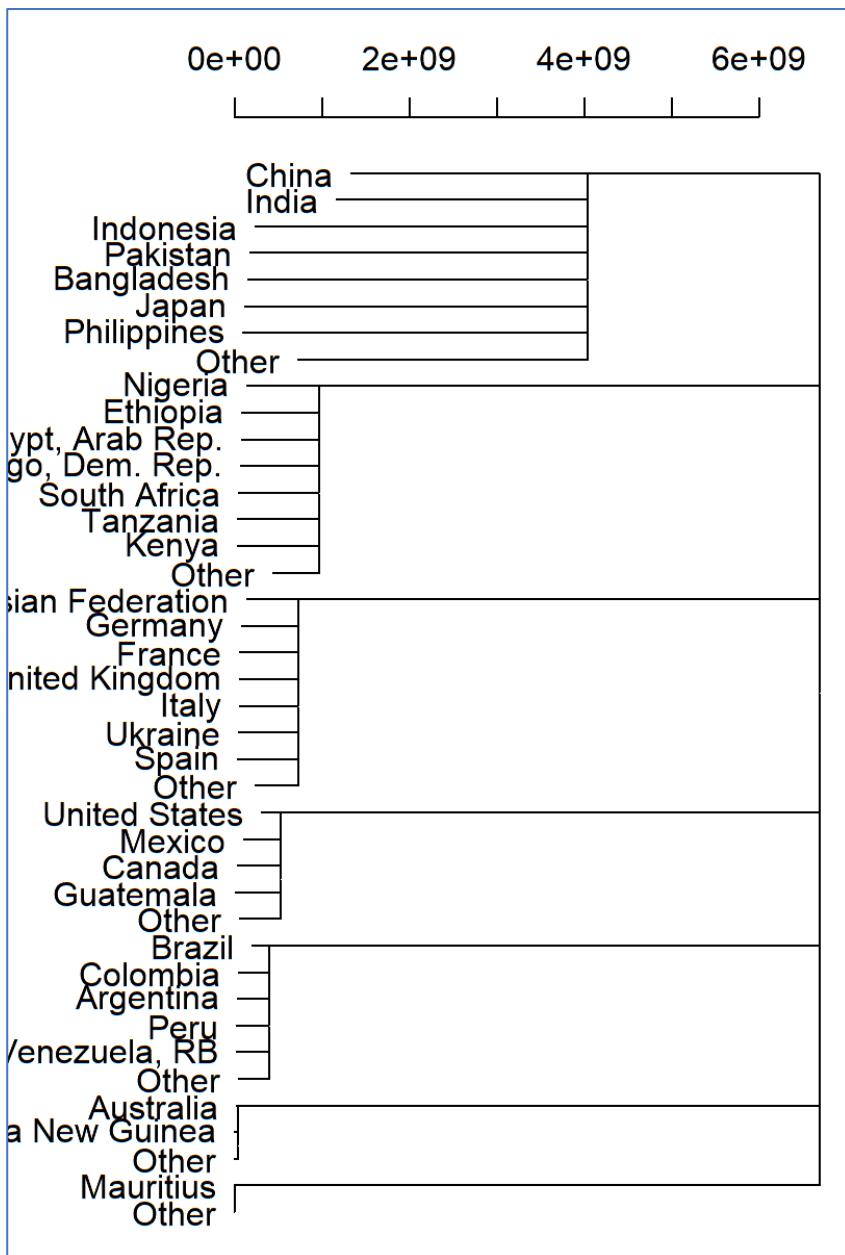
```

plot(as.dendrogram(treeClone, heightAttribute = "population"))

```

The output appears on the next page.

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



## Portfolio Breakdown (finance)

In this example, we show how to display an investment portfolio as a hierachic breakdown into asset classes. You'll see:

- \* how you can re-use a traversal
- \* advanced use of Aggregate
- \* how to add default attribute formatters to your tree

Convert from data.frame

```
fileName <- system.file("extdata", "portfolio.csv", package="data.tree")
pfodf <- read.csv(fileName, stringsAsFactors = FALSE)
head(pfodf)

##           ISIN          Name Ccy Type Dur.
## 1 LI0015327682    LGT Money Market Fund (CHF) - B CHF Fund     NA
## 2 LI0214880598      CS (Lie) Money Market Fund EUR EB EUR Fund     NA
## 3 LI0214880689      CS (Lie) Money Market Fund USD EB USD Fund     NA
## 4 LU0243957825 Invesco Euro Corporate Bond A EUR Acc EUR Fund 5.10
## 5 LU0408877412 JPM Euro Gov Sh. Duration Bd A (acc)-EUR Fund 2.45
## 6 LU0376989207 Aberdeen Global Sel Emerg Mkt Bd HEUR EUR Fund 6.80
##   Weight AssetCategory AssetClass       SubAssetClass
## 1  0.030      Cash      CHF
## 2  0.060      Cash      EUR
## 3  0.020      Cash      USD
## 4  0.120  Fixed Income EUR Sov. and Corp. Bonds
## 5  0.065  Fixed Income EUR Sov. and Corp. Bonds
## 6  0.030  Fixed Income      EUR      Em. Mkts Bonds
```

Let us convert the data.frame to a data.tree structure. Here, we use again the path string method. For other options, see `?as.Node.data.frame`.

```
pfodf$pathString <- paste("portfolio",
                           pfodf$AssetCategory,
                           pfodf$AssetClass,
                           pfodf$SubAssetClass,
                           pfodf$ISIN,
                           sep = "/")
```

pfo <- as.Node(pfodf)

### Aggregate for calculations

To calculate the weight per asset class, we use the Aggregate method:

```
t <- Traverse(pfo, traversal = "post-order")
Do(t, function(x) x$Weight <- Aggregate(node = x, attribute = "Weight", aggFun = sum))
```

We now calculate the WeightOfParent,

```
Do(t, function(x) x$WeightOfParent <- x$Weight / x$parent$Weight)
```

Duration is a bit more complicated, as this is a concept that applies only to the fixed income asset class. Note that, in the second statement, we are reusing the traversal from above.

```
pfo$Do(function(x) x$Duration <- ifelse(is.null(x$Duration), 0, x$Duration), filterFun = isLeaf)
Do(t, function(x) x$Duration <- Aggregate(x, function(x) x$WeightOfParent * x$Duration, sum))
```

## Formatters

We can add default formatters to our `data.tree` structure. Here, we add them to the root, but we might as well add them to any Node in the tree.

```
SetFormat(pfo, "WeightOfParent", function(x) FormatPercent(x, digits = 1))
SetFormat(pfo, "Weight", FormatPercent)
FormatDuration <- function(x) {
  if (x != 0) res <- FormatFixedDecimal(x, digits = 1)
  else res <- ""
  return (res)
}
SetFormat(pfo, "Duration", FormatDuration)
```

These formatter functions will be used when printing a `data.tree` structure. Below, we print the hierarchical structure of the portfolio we defined as “`pfo`” or the nodes of the portfolio dataframe.

## Print results

```
print(pfo,
      "Weight",
      "WeightOfParent",
      "Duration",
      filterFun = function(x) !x$isLeaf)

##                                     levelName  Weight WeightOfParent Duration
## 1 portfolio                           100.00 %
## 2 |--Cash                            11.00 %   11.0 %
## 3   |--CHF                            3.00 %   27.3 %
## 4   |--EUR                            6.00 %   54.5 %
## 5     |--USD                           2.00 %   18.2 %
## 6   |--Fixed Income                  28.50 %   28.5 %   3.0
## 7     |--EUR                           26.00 %   91.2 %   3.1
## 8       |--Sov. and Corp. Bonds    18.50 %   71.2 %   2.4
## 9       |--Em. Mkts Bonds          3.00 %   11.5 %   6.8
## 10      |--High Yield Bonds        4.50 %   17.3 %   3.4
## 11      |--USD                           2.50 %   8.8 %   1.6
## 12      |--High Yield Bonds        2.50 %   100.0 %  1.6
## 13   |--Equities                      40.00 %   40.0 %
## 14     |--Switzerland                 6.00 %   15.0 %
## 15     |--Euroland                   14.50 %   36.2 %
## 16     |--US                           8.10 %   20.2 %
## 17     |--UK                           0.90 %   2.2 %
## 18     |--Japan                        3.00 %   7.5 %
## 19     |--Australia                  2.00 %   5.0 %
## 20     |--Emerging Markets           5.50 %   13.7 %
## 21   |--Alternative Investments    20.50 %   20.5 %
## 22     |--Real Estate                 5.50 %   26.8 %
## 23     |--Eurozone                   5.50 %   100.0 %
## 24     |--Hedge Funds                10.50 %  51.2 %
## 25     |--Commodities                4.50 %   22.0 %
```

## Jenny Lind (decision tree, plotting)

This demo calculates and plots a simple decision tree. It demonstrates the following:

- \* how to read a yaml file into a data.tree structure
- \* how to calculate a decision tree
- \* how to plot a data.tree with the data.tree plotting facility

### Load YAML file

YAML is similar to JSON but targeted towards humans (as opposed to computers). It's concise and easy to read. YAML can be a neat format to

store your data.tree structures, as you can use it across different software and systems, you can edit it with any text editor, and you can even send it as an email. This is how our YAML file looks:

```
fileName <- system.file("extdata", "jennylin.yaml", package="data.tree")
cat(readChar(fileName, file.info(fileName)$size))

## name: Jenny Lind
## type: decision
## Sign with Movie Company:
##   type: chance
##   Small Box Office:
##     type: terminal
##     p: 0.3
##     payoff: 200000
##   Medium Box Office:
##     type: terminal
##     p: 0.6
##     payoff: 1000000
##   Large Box Office:
##     type: terminal
##     p: 0.1
##     payoff: 3000000
## Sign with TV Network:
##   type: chance
##   Small Box Office:
##     type: terminal
##     p: 0.3
##     payoff: 900000
##   Medium Box Office:
##     type: terminal
##     p: 0.6
##     payoff: 900000
##   Large Box Office:
##     type: terminal
##     p: 0.1
##     payoff: 900000
```

Let's convert the YAML into a data.tree structure. First, we load it with the yaml package into a list of lists. Then we use as.Node to convert the list into a data.tree structure:

```
library(data.tree)
library(yaml)
lol <- yaml.load_file(fileName)
```

```

jl <- as.Node(lol)
print(jl, "type", "payoff", "p")
##                                     levelName      type  payoff   p
## 1 Jenny Lind             decision     NA  NA
## 2 |--Sign with Movie Company    chance     NA  NA
## 3 |  |--Small Box Office    terminal 200000 0.3
## 4 |  |--Medium Box Office   terminal 1000000 0.6
## 5 |  |  |--Large Box Office  terminal 3000000 0.1
## 6 |  |--Sign with TV Network   chance     NA  NA
## 7 |  |  |--Small Box Office  terminal 900000 0.3
## 8 |  |  |--Medium Box Office terminal 900000 0.6
## 9 |  |  |  |--Large Box Office terminal 900000 0.1

```

## Calculate

Next, we define our payoff function, and apply it to the tree. Note that we use post-order traversal, meaning that we calculate the tree from leaf to root:

```

payoff <- function(node) {
  if (node$type == 'chance') node$payoff <- sum(sapply(node$children, function(child) child$payoff * child$p))
  else if (node$type == 'decision') node$payoff <- max(sapply(node$children, function(child) child$payoff)))
}
jl$Do(payoff, traversal = "post-order", filterFun = isNotLeaf)

```

The decision function is the next step. Note that we filter on decision nodes:

```

decision <- function(x) {
  po <- sapply(x$children, function(child) child$payoff)
  x$decision <- names(po[po == x$payoff])
}
jl$Do(decision, filterFun = function(x) x$type == 'decision')

```

## Plot

Plot with the `data.tree` plotting facility. The `data.tree` plotting facility uses GraphViz / Diagrammer. You can provide a function as a style:

```

GetNodeLabel <- function(node) switch(node$type,
                                       terminal = paste0('
$ ', format(node$payoff, scientific = FALSE, big.mark = ",",
"")))

```

```

          paste0('ER\n', '$ ',
format(node$payoff, scientific = FALSE, big.mark = ","))
}

GetEdgeLabel <- function(node) {
  if (!node$isRoot && node$parent$type == 'chance') {
    label = paste0(node$name, " (", node$p, ")")
  } else {
    label = node$name
  }
  return (label)
}
GetNodeShape <- function(node) switch(node$type, decision
= "box", chance = "circle", terminal = "none")
SetEdgeStyle(jl, fontname = 'helvetica', label = GetEdgeLabel)
SetNodeStyle(jl, fontname = 'helvetica', label = GetNodeLabel, shape = GetNodeShape)

```

Note that the fontname is inherited as is by all children, whereas e.g. the label argument is a function, it's called on each inheriting child node.

Another alternative is to set the style per node:

```

jl$Do(function(x) SetEdgeStyle(x, color = "red", inherit =
FALSE),
      filterFun = function(x) !x$isRoot && x$parent$type =
= "decision" && x$parent$decision == x$name)

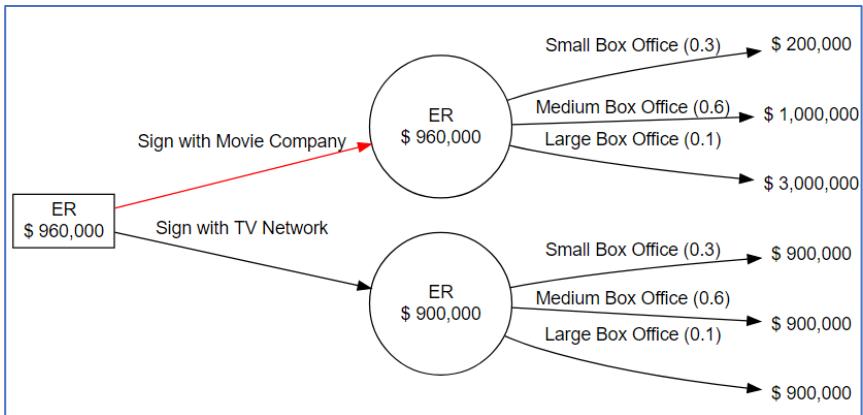
```

Finally, we direct our plot from left-to-right, and use the plot function to display:

```

SetGraphStyle(jl, rankdir = "LR")
plot(jl)

```



## File Explorer (system utilities)

In this example, we print the files that exist in the folder structure of the file system. As a special goodie, we'll show code that lets you build your own R File Explorer, an interactive tree / list widget that lets you expand folders and browse through your file system.

First, let's read the files in a directory tree into R. In this example, the root path “..” is the parent of the vignettes folder, i.e. the data.tree package folder itself:

```

path <- ".."
files <- list.files(path = path,
                     recursive = TRUE,
                     include.dirs = FALSE)

df <- data.frame(
  filename = sapply(files,
                     function(f1) paste0("data.tree","/",
                                         f1)),
  file.info(paste(path, files, sep = "/")),
  stringsAsFactors = FALSE
)
print(head(df)[-c(3,5,6)], row.names = FALSE)
## 
## filename size
## data.tree/2018 Data Analytics Schedule - Rev4.xlsx 165
## data.tree/2018 Data Analytics Schedule - Rev2.xlsx 1426
## data.tree/2018 Data Analytics Schedule - Rev1.xlsx 1667
## data.tree/2018 Data Analytics Schedule - Rev3.xlsx 1437
  
```

```

## data.tree/code/10 More Signs that might be a Data you
Scientist.txt 2002
## data.tree/code/10 Signs that might be a Data you Scientist.txt 2300
## mode atime exe
## 666 2018-09-05 19:14:05 no
## 666 2018-08-31 17:50:45 no
## 666 2018-08-31 12:41:24 no
## 666 2018-09-17 18:05:17 no
## 666 2018-09-12 04:43:30 no
## 666 2018-09-12 04:43:30 no

```

We now convert this into a data.tree:

```

fileStructure <- as.Node(df, pathName = "filename")
fileStructure$leafCount / (fileStructure$totalCount - fileStructure$leafCount)
## [1] 26.3
print(fileStructure, "mode", "size", limit = 25)
##   levelName mode
## 1 data.tree NA
## 2   |--~$2018 Data Analytics Schedule - Revision03.xlsx 438
## 3   |--2018 Data Analytics Schedule - Rev2.xlsx 438
## 4   |--2018 Data Analytics Schedule - Revised.xlsx 438
## 5   |--2018 Data Analytics Schedule - Revision03.xlsx 438
## 6   |--code NA
## 7   |  |--10 More Signs that might be a Data you Scientist.txt 438
## 8   |  |--10 Signs that might be a Data you Scientist.txt 438
## 9   |  |--4 Types of Analytics Users & How to Sell to Them.txt 438
## 10  |  |--A one-eyed man in the kingdom of the blind.txt 438
## 11  |  |--All Things Data.txt 438
## 12  |  |--Analytics and Statistics.txt 438
## 13  |  |--Analytics is it more than a buzzword.txt 438
## 14  |  |--Bayesian networks.txt 438
## 15  |  |--Big Data Analytics and Human Resources.txt 438
## 16  |  |--Big Data The Good the Bad and the Ugly.txt 438
## 17  |  |--blog_corpus NA
## 18  |  |  |--10 More Signs that might be a Data you Scientist. 438
## 19  |  |  |--10 Signs that might be a Data you Scientist.txt 438
## 20  |  |  |--4 Types of Analytics Users and How to Sell to Them.txt 438
## 21  |  |  |--A one-eyed man in the kingdom of the blind.txt 438
## 22  |  |  |--All Things Data.txt 438
## 23  |  |  |--Analytics and Statistics.txt 438
## 24  |  |  |--Analytics is it more than a buzzword.txt 438

```

```

## 25 |   |   °---... 19 nodes w/ 0 sub          NA
## 26 |   |   °---... 104 nodes w/ 102 sub       NA
## 27 °---... 11 nodes w/ 511 sub              NA
##     size
[Output omitted]

```

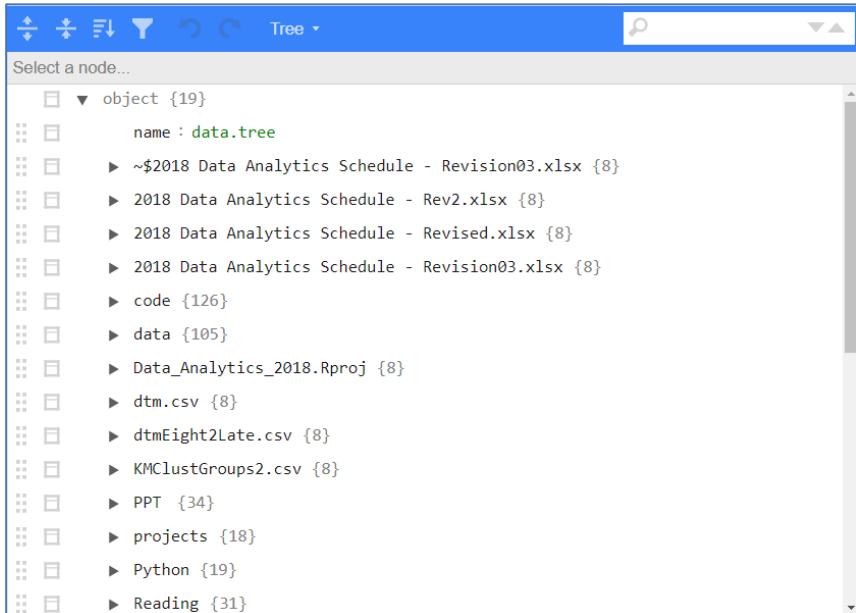
## Listviewer html widget

Finally, we can display the files by timelyportfolio's listviewer. As it's not on CRAN, we only display a screenshot of the widget in this vignette. This is not half as fun as the interactive widget, of course. So please try it out for yourself to see it in action. This requires listviewer, which is available only on github.

```

#devtools::install_github("timelyportfolio/Listviewer")
library(listviewer)
l <- ToListSimple(fileStructure)
jsonedit(l)

```



## Bubble Chart (visualisation)

In this example, we will replicate Mike Bostock's bubble example. See here for details: <http://bl.ocks.org/mbostock/4063269>. We use Joe Cheng's bubbles package. All of this is inspired by Timelyportfolio, the

king of htmlwidgets. You'll learn how to convert a complex JSON into a data.frame, and how to use this to plot hierachic visualizations.

## Load JSON file

The data represents the Flare class hierarchy, which is a code library for creating visualizations. The JSON is long, deeply nested, and complicated.

```
require(Rcpp)
require(data.tree)
require(jsonlite)
require(bubbles)
require(RColorBrewer)
fileName <- system.file("extdata", "flare.json", package="data.tree")
flareJSON <- readChar(fileName, file.info(fileName)$size)
cat(substr(flareJSON, 1, 300))
## {
##   "name": "flare",
##   "children": [
##     {
##       "name": "analytics",
##       "children": [
##         {
##           "name": "cluster",
##           "children": [
##             {"name": "AgglomerativeCluster", "size": 3938},
##             {"name": "CommunityStructure", "size": 3812},
##             {"name": "HierarchicalCluster", "size": 6714},
##             {"name": "Modularity", "size": 1337}
##           ]
##         }
##       ]
##     }
##   ]
## }
```

So, let's convert it into a data.tree structure:

```
library(jsonlite)
flareLoL <- fromJSON(file(fileName),
                      simplifyDataFrame = FALSE
                     )

flareTree <- as.Node(flareLoL, mode = "explicit")
## Warning in CheckNameReservedWord(name, check): Name 'count' is a reserved
## word as defined in NODE_RESERVED_NAMES_CONST. Using 'count2' instead.
```

```

## Warning in CheckNameReservedWord(name, check): Name 'So
rt' is a reserved
## word as defined in NODE_RESERVED_NAMES_CONST. Using 'So
rt2' instead.

flareTree$fieldsAll
## [1] "size"
print(flareTree, "size", limit = 30)
##                                     levelName    size
## 1   flare                               NA
## 2   |--analytics                         NA
## 3   |--cluster                           NA
## 4   |   |--AgglomerativeCluster        3938
## 5   |   |--CommunityStructure          3812
## 6   |   |--HierarchicalCluster         6714
## 7   |       |--MergeEdge                743
## 8   |   |--graph                            NA
## 9   |       |--BetweennessCentrality     3534
## 10  |       |--LinkDistance              5731
## 11  |       |--MaxFlowMinCut            7840
## 12  |       |--ShortestPaths             5914
## 13  |           |--SpanningTree          3416
## 14  |       |--optimization                  NA
## 15  |           |--AspectRatioBanker      7074
## 16  |   |--animate                          NA
## 17  |       |--Easing                   17010
## 18  |       |--FunctionSequence          5842
## 19  |       |--interpolate                  NA
## 20  |           |--ArrayInterpolator     1983
## 21  |           |--ColorInterpolator      2047
## 22  |           |--DateInterpolator       1375
## 23  |           |--Interpolator             8746
## 24  |           |--MatrixInterpolator      2202
## 25  |           |--NumberInterpolator       1382
## 26  |           |--ObjectInterpolator       1629
## 27  |           |--PointInterpolator        1675
## 28  |               |--RectangleInterpolator 2042
## 29  |           |--ISchedulable            1041
## 30  |               |--... 8 nodes w/ 0 sub      NA
## 31  |               |--... 8 nodes w/ 215 sub     NA

```

Finally, we can convert it into a data.frame. The `ToDataTable` only converts leafs, but inherits attributes from ancestors:

```

flare_df <- ToDataTable(flareTree,
                        className = function(x) x$parent$name,

```

```

packageName = "name", "size")
head(flare_df)
##   className      packageName size
## 1 cluster    AgglomerativeCluster 3938
## 2 cluster    CommunityStructure 3812
## 3 cluster HierarchicalCluster 6714
## 4 cluster       MergeEdge  743
## 5 graph BetweennessCentrality 3534
## 6 graph     LinkDistance 5731

```

This does not look spectacular. But take a look at this stack overflow question to see how people struggle to do this type of operation.

Here, it was particularly simple, because the underlying JSON structure is regular. If it were not (e.g. some nodes contain different attributes than others), the conversion from JSON to data.tree would still work. And then, as a second step, we could modify the data.tree structure before converting it into a data.frame. For example, we could use Prune and Remove to remove unwanted nodes, use Set to remove or add default values, etc.

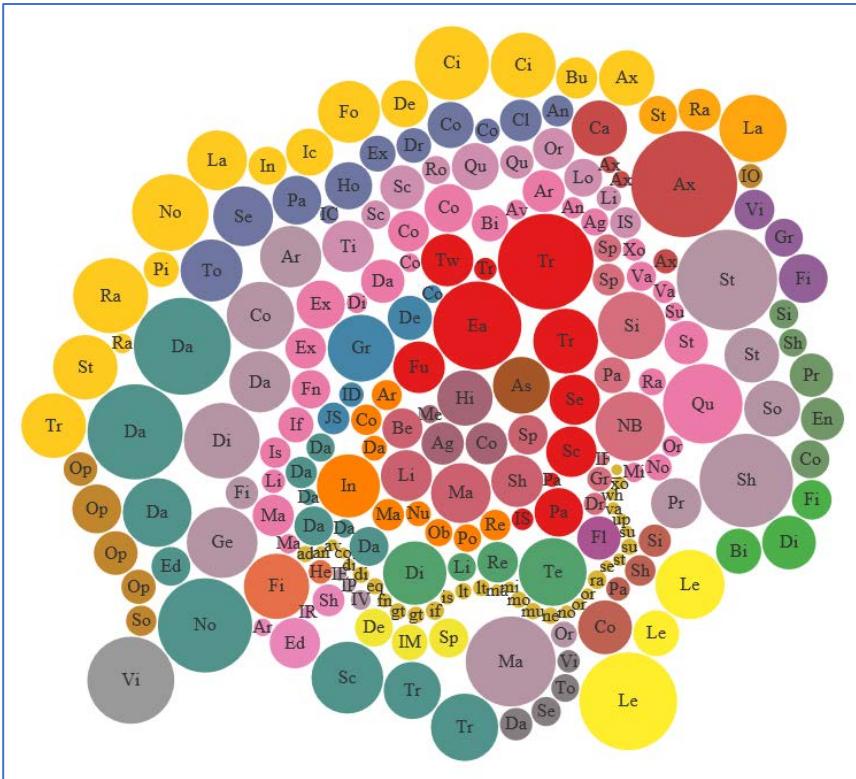
## Plot

What follows has nothing to do with data.tree anymore, except as it reflects the flareTree. We simply provide the bubble chart printing for your enjoyment. In order to run it yourself, you need to install the bubbles package from github.

```

#devtools::install_github("jcheng5/bubbles@6724e43f5e")
library(scales)
library(bubbles)
library(RColorBrewer)
bubbles(
  flare_df$size,
  substr(flare_df$packageName, 1, 2),
  tooltip = flare_df$packageName,
  color = col_factor(
    brewer.pal(9,"Set1"),
    factor(flare_df$className)
  )(flare_df$className),
  height = 800,
  width = 800
)

```



## Exercises

1. Download the data set Lahman Baseball Database – lahman591-csv.zip, <[http://seanlahman.com/files/database/baseballdatabank-master\\_2016-03-02.zip](http://seanlahman.com/files/database/baseballdatabank-master_2016-03-02.zip)>. We have several files of baseball statistics and we are going to bring them into R-Studio and do some simple computing with them. We are going to find the player with the highest runs for each year. This file has all the statistics from 1871–2011 and contains more than 90,000 rows. Once we have the highest runs, we will extend the script to translate a player id field into the first and last names of the players.
  - a. Using R, write a script that generates player ID, year, and total runs by player ID. Sort the query by year. [Hint: build a new dataframe.]
  - b. Using R, write a series of queries that will generate a file with player ID, year, and total runs by player ID. Sort the query by year.
  - c. Using your Hadoop environment, write a script that generates regular season pitcher (player ID), year, and ERA (earned run average) by pitcher player ID. Sort the list by ERA
  - d. Construct a classification tree model that predicts the NLCS (National League Championship Series) Winner from 1965 to 2005, based on regular season Wins/Loss ratio, Runs/Run-Against ratio, and Earned Run Average (ERA).
  - e. Based on actual NLCS winners, how accurate is your model based on NLCS winners from 2006 to 2015?
  - f. Try several different tree presentation options and choose the best one for explaining your results.

## Chapter 2 – Random Forests

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman (Breiman L. , Random Forests, 2001) and Adele Cutler (Liaw, 2012), and “Random Forests” is their trademark. The term came from random decision forests that was first proposed by Tin Kam Ho of Bell Labs in 1995. The method combines Breiman’s “bagging” idea and the random selection of features, introduced independently by Ho (Ho T. , 1995) (Ho T. , 1998) and Amit and Geman (Amit & Geman, 1997) in order to construct a collection of decision trees with controlled variance.

The selection of a random subset of features is an example of the random subspace method, which, in Ho’s formulation, is a way to implement classification proposed by Eugene Kleinberg (Kleinberg, 1996).

### History

The early development of random forests was influenced by the work of Amit and Geman (Amit & Geman, 1997) which introduced the idea of searching over a random subset of the available decisions when splitting a node, in the context of growing a single tree. The idea of random subspace selection from Ho (Ho T. , 1998) was also influential in the design of random forests. In this method a forest of trees is grown, and variation among the trees is introduced by projecting the training data into a randomly chosen subspace before fitting each tree. Finally, the idea of randomized node optimization, where the decision at each node is selected by a randomized procedure, rather than a deterministic optimization was first introduced by Dietterich (Dietterich T. , 2000).

The introduction of random forests proper was first made in a paper by Leo Breiman (Breiman L. , Random Forests, 2001). This paper describes a method of building a forest of uncorrelated trees using a CART like procedure, combined with randomized node optimization and bagging. In addition, this paper combines several ingredients, some previously known and some novel, which form the basis of the modern practice of random forests, in particular:

1. Using out-of-bag error as an estimate of the generalization error.
2. Measuring variable importance through permutation.

The report also offers the first theoretical result for random forests in the form of a bound on the generalization error which depends on the strength of the trees in the forest and their correlation.

More recently several major advances in this area have come from Microsoft Research (Criminisi, Shotton, & Konukoglu, 2011), which incorporate and extend the earlier work from Breiman.

## Algorithm

The training algorithm for random forests applies the general technique of bootstrap aggregating (see Bootstrap aggregating), or bagging, to tree learners. Given a training set  $X = x_1, \dots, x_n$  with responses  $Y = y_1$  through  $y_n$ , bagging repeatedly selects a bootstrap sample of the training set and fits trees to these samples:

For  $b = 1$  through  $B$ :

1. Sample, with replacement,  $n$  training examples from  $X, Y$ ; call these  $X_b, Y_b$ .
2. Train a decision or regression tree  $f_b$  on  $X_b, Y_b$ .

After training, predictions for unseen samples  $x'$  can be made by averaging the predictions from all the individual regression trees on  $x'$ :

$$\hat{f} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x')$$

or by taking the majority vote in the case of decision trees.

In the above algorithm,  $B$  is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. Increasing the number of trees tends to decrease the variance of the model, without increasing the bias. As a result, the training and test error tend to level off after some number of trees have been fit. An optimal number of trees  $B$  can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample  $x_i$ , using only the trees that did not have  $x_i$  in their bootstrap sample (James, Witten, Hastie, & Tibshirani, 2013).

## Bootstrap aggregating

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

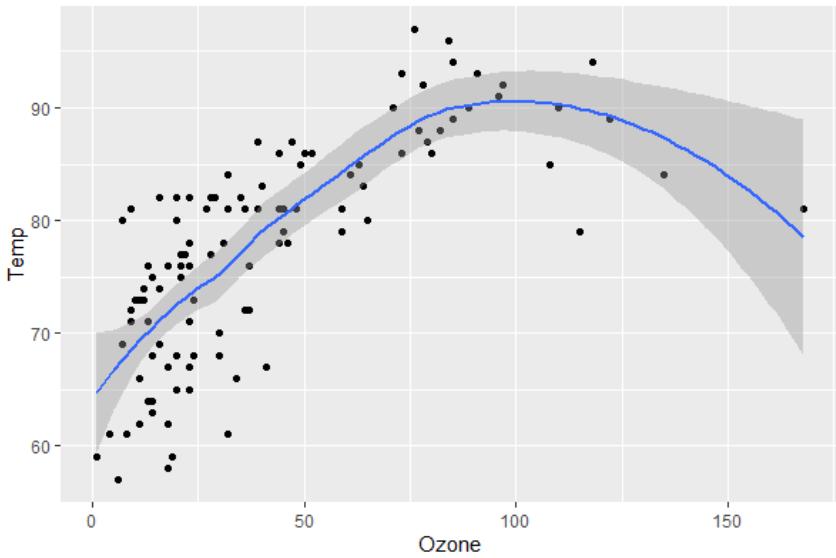
### Description of the technique

Given a standard training set  $D$  of size  $n$ , bagging generates  $m$  new training sets  $D_i$ , each of size  $n'$ , by sampling from  $D$  uniformly and with replacement. By sampling with replacement, some observations may be repeated in each. If  $n' = n$ , then for large  $n$  the set is expected to have the fraction  $(1 - 1/e)$  ( $\approx 63.2\%$ ) of the unique examples of  $D$ , the rest being duplicates (Aslam, Popa, & Rivest, 2007). This kind of sample is known as a bootstrap sample. The  $m$  models are fitted using the above  $m$  bootstrap samples and combined by averaging the output (for regression) or voting (for classification). Bagging leads to “improvements for unstable procedures” (Breiman L. , Random Forests, 2001), which include, for example, neural nets, classification and regression trees, and subset selection in linear regression (Breiman L. , 1996). An interesting application of bagging showing improvement in preimage learning is provided here (Sahu, Runger, & Apley, 2011) (Shinde, Sahu, Apley, & Runger, 2014). On the other hand, it can mildly degrade the performance of stable methods such as  $K$ -nearest neighbors (Breiman, 1996).

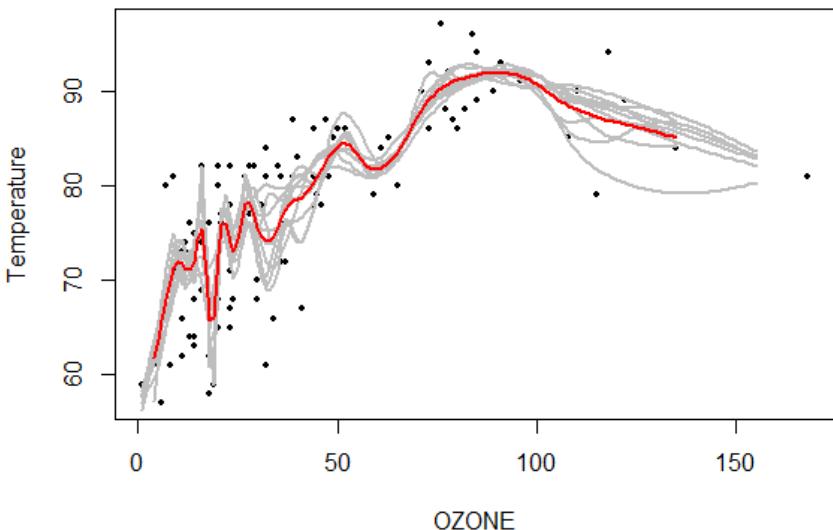
### Example: Ozone data

To show the basic principles of bagging, we use an analysis of the relationship between temperature and ozone (data from Rousseeuw and Leroy (Rousseeuw & Leroy, 2003), available at classic data sets, analysis done in R).

The relationship between temperature and ozone in the airquality data set is seemingly non-linear, based on the scatter plot. To mathematically describe this relationship, LOESS smoothers (with span 0.5) are used. First, we look at a simple smoother.



Rather than constructing a single smoother from the complete data set, we drew 100 bootstrap samples of the data. Each sample is different from the original data set yet is similar to it in distribution and variability. A LOESS smoother was fit for each bootstrap sample. We then made Predictions from these 100 smoothers across the range of the data. The first 10 predicted smooth fits appear as grey lines in the figure below. The lines are clearly very wiggly and they overfit the data—a result of the span being too low. But taking the average of 100 smoothers, each fitted to a subset of the original data set, we arrive at one bagged predictor (red line). Clearly, the mean is more stable and there is less overfit.



### *Bagging for nearest neighbor classifiers*

It is well known that the risk of a 1 nearest neighbor (1NN) classifier is at most twice the risk of the Bayes classifier, but there are no guarantees that this classifier will be consistent. By careful choice of the size of the resamples, bagging can lead to substantial improvements of the performance of the 1NN classifier. By taking a large number of resamples of the data of size  $n'$ , the bagged nearest neighbor classifier will be consistent provided  $n' \rightarrow \infty$  diverges but  $n'/n \rightarrow 0$  as the sample size  $n \rightarrow \infty$ .

Under infinite simulation, the bagged nearest neighbor classifier can be viewed as a weighted nearest neighbor classifier. Suppose that the feature space is  $d$  dimensional and denote by  $C_{n,n'}^{bnn}$  the bagged nearest neighbor classifier based on a training set of size  $n$ , with resamples of size  $n'$ . In the infinite sampling case, under certain regularity conditions on the class distributions, the excess risk has the following asymptotic expansion

$$\mathcal{R}_{\mathcal{R}}(C_{n,n'}^{bnn}) - \mathcal{R}_{\mathcal{R}}(C^{bayes}) = \left( B_1 \frac{n'}{n} + B_2 \frac{1}{(n')^{4/d}} \right) \{1 + o(1)\},$$

for some constants  $B_1$  and  $B_2$ . The optimal choice of  $n'$ , that balances the two terms in the asymptotic expansion, is given by  $n' = Bn^{d/(d+4)}$  for some constant  $B$ .

## History

Bagging (Bootstrap aggregating) was proposed by Leo Breiman in 1994 to improve the classification by combining classifications of randomly generated training sets. See Breiman (Breiman L. , Bagging Predictors, 1994).

## From bagging to random forests

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the  $B$  trees, causing them to become correlated.

Typically, for a dataset with  $p$  features,  $\sqrt{p}$  features are used in each split.

## Random subspace method

Random subspace method (or attribute bagging (Bryll, 2003)) is an ensemble classifier that consists of several classifiers and outputs the class based on the outputs of these individual classifiers. Random subspace method is a generalization of the random forest algorithm (Ho T. , 1998). Whereas random forests are composed of decision trees, a random subspace classifier can be composed from any underlying classifiers. Random subspace method has been used for linear classifiers (Skurichina, 2002), support vector machines (Tao, 2006), nearest neighbors (Tremblay, 2004) and other types of classifiers. This method is also applicable to one-class classifiers.

The algorithm is an attractive choice for classification problems where the number of features is much larger than the number of training objects, such as fMRI data or gene expression data (Kuncheva, Rodríguez, Plumpton, Linden, & Johnston, 2010).

## Algorithm

The ensemble classifier is constructed using the following algorithm:

1. Let the number of training objects be  $N$  and the number of features in the training data be  $D$ .
2. Choose  $L$  to be the number of individual classifiers in the ensemble.
3. For each individual classifier,  $l$ , Choose  $d_l$  ( $d_l < D$ ) to be the number of input variables for  $l$ . It is common to have only one value of  $d_l$  for all the individual classifiers
4. For each individual classifier,  $l$ , create a training set by choosing  $d_l$  features from  $D$  without replacement and train the classifier.
5. For classifying a new object, combine the outputs of the  $L$  individual classifiers by majority voting or by combining the posterior probabilities.

## Relationship to Nearest Neighbors

Given a set of training data

$$\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$$

a weighted neighborhood scheme makes a prediction for a query point  $X$ , by computing

$$\hat{Y} = \sum_{i=1}^n W_i(X) Y_i,$$

for some set of non-negative weights  $\{W_i(X)\}_{i=1}^n$  which sum to 1. The set of points  $X_i$  where  $W_i(X) > 0$  are called the neighbors of  $X$ . A common example of a weighted neighborhood scheme is the  $k$ -NN algorithm which sets  $W_i(X) = 1/k$  if  $X_i$  is among the  $k$  closest points to  $X$  in  $\mathcal{D}_n$  and 0 otherwise.

Random forests with constant leaf predictors can be interpreted as a weighted neighborhood scheme in the following way. Given a forest of  $M$  trees, the prediction that the  $m$ -th tree makes for  $X$  can be written as

$$T_m(X) = \sum_{i=1}^n W_{im}(X) Y_i,$$

where  $W_{im}(X)$  is equal to  $1/k_m$  if  $X$  and  $X_i$  are in the same leaf in the  $m$ -th tree and 0 otherwise, and  $k_m$  is the number of training data which

fall in the same leaf as  $X$  in the  $m$ -th tree. The prediction of the whole forest is

$$F(X) = \sum_{i=1}^n T_m(X) = \frac{1}{M} \sum_{m=1}^M \sum_{i=1}^n W_{im}(X) Y_i = \sum_{i=1}^n \left( \frac{1}{M} \sum_{m=1}^M W_{im}(X) \right) Y_i,$$

which shows that the random forest prediction is a weighted average of the  $Y_i$ 's, with weights

$$W_i(X) = \frac{1}{M} \sum_{m=1}^M W_{im}(X).$$

The neighbors of  $X$  in this interpretation are the points  $X_i$  which fall in the same leaf as  $X$  in at least one tree of the forest. In this way, the neighborhood of  $X$  depends in a complex way on the structure of the trees, and thus on the structure of the training set.

This connection was first described by Lin and Jeon in a technical report from 2001 where they show that the shape of the neighborhood used by a random forest adapts to the local importance of each feature (Lin & Jeon, 2001).

## Variable importance

Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. The following technique was described in Breiman's original paper (Breiman L., Random Forests, 2001) and is implemented in the R package *randomForest* (Liaw, 2012).

The first step in measuring the variable importance in a data set  $\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$  is to fit a random forest to the data. During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest (errors on an independent test set can be substituted if bagging is not used during training).

To measure the importance of the  $j$ -th feature after training, the values of the  $j$ -th feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set. The importance score for the  $j$ -th feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees. The score is normalized by the standard deviation of these differences.

Features which produce large values for this score are ranked as more important than features which produce small values.

This method of determining variable importance has some drawbacks. For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Methods such as partial permutations can be used to solve the problem (Deng, Runger, & Tuv, 2011) (Altmann, Tolosi, Sander, & Lengauer, 2010). If the data contain groups of correlated features of similar relevance for the output, then smaller groups are favored over larger groups (Tolosi & Lengauer, 2011).

## Variants

Instead of decision trees, linear models have been proposed and evaluated as base estimators in random forests, in particular multinomial logistic regression and Naïve Bayes classifiers (Prinzie & Van den Poel, 2008).

## Random Forest using R

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set  $X = x_1, \dots, x_n$  with responses  $Y = y_1, \dots, y_n$ , bagging repeatedly (B times) selects random samples with replacement of the training set and fits trees to these samples:

1. Each sample may be comprised of different mixes members of the population, i.e., sample will have some members in common and some will be different
2. For  $n = 1, \dots, N$ , and  $m = 1, \dots, M$ , there will be  $N$  training sets comprised of  $m$  members sampled with replacement
3. Each sample may be comprised of a different mix of variables from the original set  $X$

4. Each tree (regression or classification) will train using one of the  $X_i$  training sets
5. After training, predictions are made by either averaging of the prediction from individual trees or by majority vote in the case of classification trees

Random Forest algorithm is built in randomForest package of R and same name function allows us to use the Random Forest in R.

### ***Load libraries***

```
if(!require(randomForest)) install.packages("randomForest")
if(!require(colorspace)) install.packages("colorspace")
if(!require(reshape)) install.packages("reshape")
if(!require(ggplot2)) install.packages("ggplot2")
library(randomForest)
library(colorspace)
library(reshape)
library(ggplot2)
```

Some of the commonly used parameters of randomForest functions are

- x: Random Forest Formula
- data: Input data frame
- ntree: Number of decision trees to be grown
- replace: Takes True and False and indicates whether to take sample with/without replacement
- sampsize: Sample size to be drawn from the input data for growing decision tree
- importance: Whether independent variable importance in random forest be assessed
- proximity: Whether to calculate proximity measures between rows of a data frame

### ***Medical longevity Study of primary biliary cirrhosis (PBC)***

Data was obtained from a Mayo Clinic randomized trial in primary biliary cirrhosis (PBC) of the liver conducted between 1974 and 1984. A total of

424 PBC patients, referred to Mayo Clinic during that ten-year interval met eligibility criteria for the randomized placebo-controlled trial of the drug D-penicillamine (DPCA). The data and partial likelihood model is described in Fleming and Harrington (1991). The variables in the data set are:

- case number
- number of days between registration and the earlier of death, transplantation, or study analysis time in July, 1986
- status: 0=alive, 1=liver transplant, 2=dead
- drug: 1=D-penicillamine, 2=placebo
- age in days
- sex: 0=male, 1=female
- presence of ascites: 0=no 1=yes
- presence of hepatomegaly 0=no 1=yes
- presence of spiders 0=no 1=yes
- presence of edema 0=no edema and no diuretic therapy for edema; .5 = edema present without diuretics, or edema resolved by diuretics; 1 = edema despite diuretic therapy
- serum bilirubin in mg/dl
- serum cholesterol in mg/dl
- albumin in gm/dl
- urine copper in ug/day
- alkaline phosphatase in U/liter
- SGOT in U/ml
- triglycerides in mg/dl
- platelets per cubic ml / 1000
- prothrombin time in seconds
- histologic stage of disease

```
#if(!require(ggRandomForest)) install.packages("ggRandomForest")
#if(!require(randomForestSRC)) install.packages("randomForestSRC")
library(ggRandomForests)
library(randomForestSRC)
data(pbc)
summary(pbc)

##      days          status         treatment        age
##  Min.   : 41   Min.   :0.0000   Min.   :1.000   Min.   : 9598
##  1st Qu.:1093  1st Qu.:0.0000  1st Qu.:1.000   1st Qu.:15644
```

```

## Median :1730  Median :0.0000  Median :1.000  Median :18628
## Mean   :1918  Mean   :0.3852  Mean   :1.494  Mean   :18533
## 3rd Qu.:2614 3rd Qu.:1.0000  3rd Qu.:2.000  3rd Qu.:21273
## Max.   :4795  Max.   :1.0000  Max.   :2.000  Max.   :28650
##
## NA's   :106
##          sex      ascites      hepatom      spiders
## Min.   :0.0000  Min.   :0.00000  Min.   :0.0000  Min.   :0.0000
## 1st Qu.:1.0000 1st Qu.:0.00000  1st Qu.:0.0000  1st Qu.:0.0000
## Median :1.0000  Median :0.00000  Median :1.0000  Median :0.0000
## Mean   :0.8947  Mean   :0.07692  Mean   :0.5128  Mean   :0.2885
## 3rd Qu.:1.0000 3rd Qu.:0.00000  3rd Qu.:1.0000  3rd Qu.:1.0000
## Max.   :1.0000  Max.   :1.00000  Max.   :1.0000  Max.   :1.0000
## NA's   :106    NA's   :106    NA's   :106    NA's   :106
##          edema      bili       chol      albumin
## Min.   :0.0000  Min.   : 0.300  Min.   :120.0  Min.   :1.960
## 1st Qu.:0.0000 1st Qu.: 0.800  1st Qu.:249.5 1st Qu.:3.243
## Median :0.0000  Median : 1.400  Median :309.5  Median :3.530
## Mean   :0.1005  Mean   : 3.221  Mean   :369.5  Mean   :3.497
## 3rd Qu.:0.0000 3rd Qu.: 3.400  3rd Qu.:400.0  3rd Qu.:3.770
## Max.   :1.0000  Max.   :28.000  Max.   :1775.0  Max.   :4.640
## NA's   :134
##          copper      alk      sgot      trig
## Min.   : 4.00  Min.   : 289.0  Min.   : 26.35  Min.   :33.00
## 1st Qu.:41.25 1st Qu.: 871.5  1st Qu.: 80.60  1st Qu.:84.25
## Median :73.00  Median :1259.0  Median :114.70  Median :108.00
## Mean   :97.65  Mean   :1982.7  Mean   :122.56  Mean   :124.70
## 3rd Qu.:123.00 3rd Qu.:1980.0  3rd Qu.:151.90  3rd Qu.:151.00
## Max.   :588.00  Max.   :13862.4  Max.   :457.25  Max.   :598.00
## NA's   :108    NA's   :106    NA's   :106    NA's   :136
##          platelet      prothrombin      stage
## Min.   : 62.0  Min.   :  9.00  Min.   :1.000
## 1st Qu.:188.5 1st Qu.:10.00  1st Qu.:2.000
## Median :251.0  Median :10.60  Median :3.000
## Mean   :257.0  Mean   :10.73  Mean   :3.024
## 3rd Qu.:318.0 3rd Qu.:11.10  3rd Qu.:4.000
## Max.   :721.0  Max.   :18.00  Max.   :4.000
## NA's   :11     NA's   : 2     NA's   : 6

```

### Transform variable values: years to days; 0-1 to T-F

```

pbc1<-pbc
pbc1$Years<-pbc$days/365
pbc1$age<-pbc$age/365
pbc1>Status <- NULL
pbc1$status
##
## OUTPUT OMITTED

```

## Get transformed data

```
pb2<-pb1[,2:20]
head(pb2)

##   status treatment      age sex ascites hepatom spiders edema bili c
##   hol
## 1     1          1 58.80548   1     1     1     1    1.0 14.5 261
## 2     0          1 56.48493   1     0     1     1    0.0  1.1 302
## 3     1          1 70.12055   0     0     0     0    0.5  1.4 176
## 4     1          1 54.77808   1     0     1     1    0.5  1.8 244
## 5     0          2 38.13151   1     0     1     1    0.0  3.4 279
## 6     1          2 66.30411   1     0     1     0    0.0  0.8 248
##   albumin copper    alk   sgot trig platelet prothrombin stage     Y
##   ears
## 1   2.60   156 1718.0 137.95  172     190    12.2   4 1.095890
## 2   4.14    54 7394.8 113.52   88     221    10.6   3 12.328767
## 3   3.48   210 516.0  96.10   55     151    12.0   4 2.77260
## 4   2.54   64 6121.8  60.63   92     183    10.3   4 5.273973
## 5   3.53   143 671.0 113.15   72     136    10.9   3 4.120548
## 6   3.98    50 944.0  93.00   63      NA    11.0   3 6.857534
```

Reshape continuous variable data for exploratory analysis

```
dtb1<- melt(pbc2, id.vars=c("age", "Years", "status"))
dtb2<- melt(pbc2, id.vars=c("bili", "Years", "status"))
dtb3<- melt(pbc2, id.vars=c("albumin", "Years", "status"))
dtb4<- melt(pbc2, id.vars=c("alk", "Years", "status"))
dtb5<- melt(pbc2, id.vars=c("sgot", "Years", "status"))
dtb6<- melt(pbc2, id.vars=c("prothrombin", "Years", "status"))
)
dtb7<- melt(pbc2, id.vars=c("chol", "Years", "status"))
dtb8<- melt(pbc2, id.vars=c("copper", "Years", "status"))
dtb9<- melt(pbc2, id.vars=c("trig", "Years", "status"))
dtb10<- melt(pbc2, id.vars=c("platelet", "Years", "status"))
```

## Plot continuous variables

```
gg1<-ggplot(data=dtb1, aes(x=Years, y=age)) +
  geom_point(aes(x=Years, color=status)) +
  scale_fill_brewer(type="seq", palette = "Set1")
gg2<-ggplot(data=dtb2, aes(x=Years, y=bili)) +
  geom_point(aes(x=Years, color=status)) +
  scale_fill_brewer(type="seq", palette = "Set1")
gg3<-ggplot(data=dtb3, aes(x=Years, y=albumin)) +
  geom_point(aes(x=Years, color=status)) +
```

```

scale_fill_brewer(type="seq", palette = "Set1")
gg4<-ggplot(data=dtb4, aes(x=Years, y=alk)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")
gg5<-ggplot(data=dtb5, aes(x=Years, y=sgot)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")
gg6<-ggplot(data=dtb6, aes(x=Years, y=prothrombin)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")
gg7<-ggplot(data=dtb7, aes(x=Years, y=chol)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")
gg8<-ggplot(data=dtb8, aes(x=Years, y=copper)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")
gg9<-ggplot(data=dtb9, aes(x=Years, y=trig)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")
gg10<-ggplot(data=dtb10, aes(x=Years, y=platelet)) +
  geom_point(aes(x=Years, color=status)) +
scale_fill_brewer(type="seq", palette = "Set1")

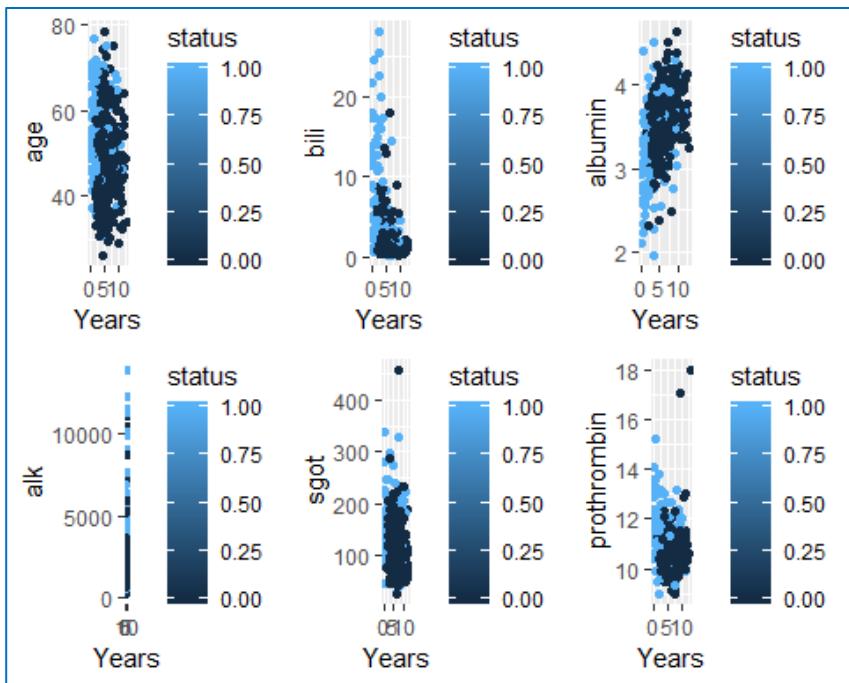
```

### *Show multiple plots in a window*

```

if(require(!gridExtra)) install.packages("gridExtra")
library(gridExtra)
grid.arrange(gg1,gg2,gg3,gg4,gg5,gg6,nrow=2)

```



***Include only the randomized patients.***

```
pbctrial <- pbctrial[which(is.na(pbctrial$treatment)),]
```

***Create a test set from the remaining patients***

Now, we create a test set for the remaining patients, i.e., those that were not randomized. We will also create a survival object and plot the survival probability function.

```
pbctest <- pbctrial[which(is.na(pbctrial$treatment)),]
head(pbctrial)

##   status treatment age sex ascites hepatom spiders edema bili chol
## 1     1        1 58.80548  1     1     1     1 1.0 14.5 261
## 2     0        1 56.48493  1     0     1     1 0.0 1.1 302
## 3     1        1 70.12055  0     0     0     0 0.5 1.4 176
## 4     1        1 54.77808  1     0     1     1 0.5 1.8 244
## 5     0        2 38.13151  1     0     1     1 0.0 3.4 279
## 6     1        2 66.30411  1     0     1     0 0.0 0.8 248
##   albumin copper alk sgot trig platelet prothrombin stage Years
## 1    2.60 156 1718.0 137.95 172      190    12.2    4 1.095890
## 2    4.14  54 7394.8 113.52  88      221    10.6    3 12.328767
## 3    3.48 210 516.0  96.10  55      151    12.0    4 2.772603
## 4    2.54  64 6121.8  60.63  92      183    10.3    4 5.273973
```

```
## 5   3.53   143  671.0 113.15   72     136   10.9    3 4.120548
## 6   3.98    50  944.0  93.00   63      NA   11.0    3 6.857534
```

### Create the gg\_survival object

```
gg_dta <- gg_survival(interval = "Years",
                       censor = "status",
                       by = "treatment",
                       data = pbc.trial,
                       conf.int = .95)
```

### Plot the survival probability function

```
plot(gg_dta) +
  labs(y = "Survival Probability",
       x = "Observation Time (Years)",
       color = "Treatment", fill = "Treatment") +
  theme(legend.position = c(.2,.2)) +
  coord_cartesian(y = c(0,1.01))
```

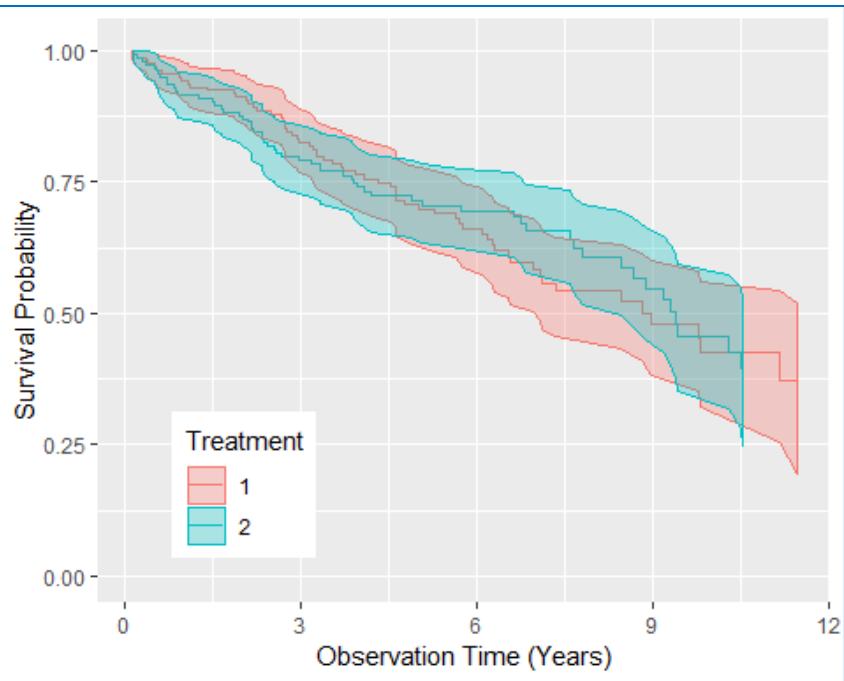
### Markdown Note.

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

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

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

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



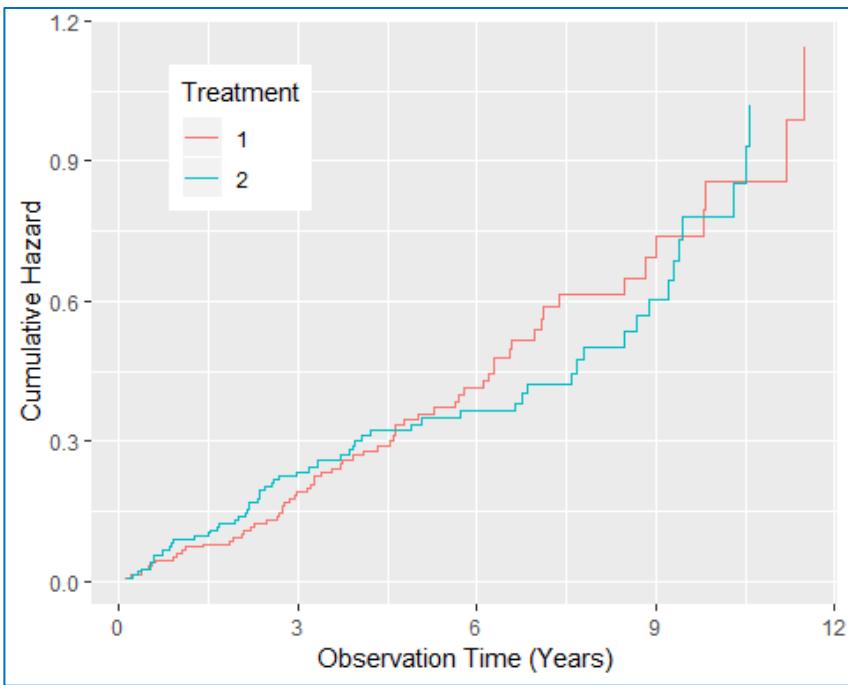
### *Plot the cumulative hazard function*

```
plot(gg_dta, type="cum_haz") +
  labs(y = "Cumulative Hazard",
       x = "Observation Time (Years)",
       color = "Treatment", fill = "Treatment") +
  theme(legend.position = c(.2,.8))
```

#### **Markdown Note.**

This is a paragraph in an R Markdown document. Below is a code chunk that produces the output that follows:

```
```{r echo=TRUE}
plot(gg_dta, type="cum_haz") +
  labs(y = "Cumulative Hazard",
       x = "Observation Time (Years)",
       color = "Treatment", fill = "Treatment") +
  theme(legend.position = c(.2,.8))
```
```



### *Duplicate the trial data*

```
pbc.bili <- pbc.trial
```

### *Group by bilirubin values*

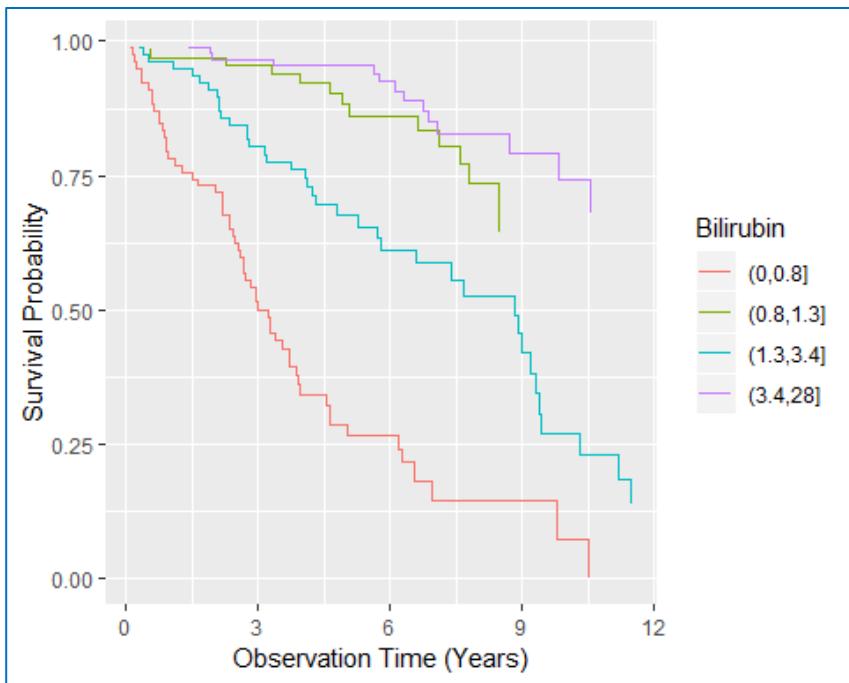
```
pbc.bili$bili_grp <- cut(pbc.trial$bili,
                           breaks = c(0, .8, 1.3, 3.4,
                                     max(pbc.trial$bili)))
```

### *Plot the gg\_survival object directly*

```
plot(gg_survival(interval = "Years", censor = "status",
                  by = "bili_grp", data = pbc.bili),
     error = "none") +
  labs(y = "Survival Probability",
       x = "Observation Time (Years)",
       color = "Bilirubin")
```

**Markdown Note.** In ````{r echo=TRUE}````

If `echo = FALSE`, knitr will not display the code in the code chunk above it's results in the final document.



```
#if(!require(shape2)) install.packages(shape2)
library(reshape2)

dta <- melt(pbc2, id.vars=c("bili","Years"))
dtb <- melt(pbc2, id.vars=c("Years","status"))
head(dtb)

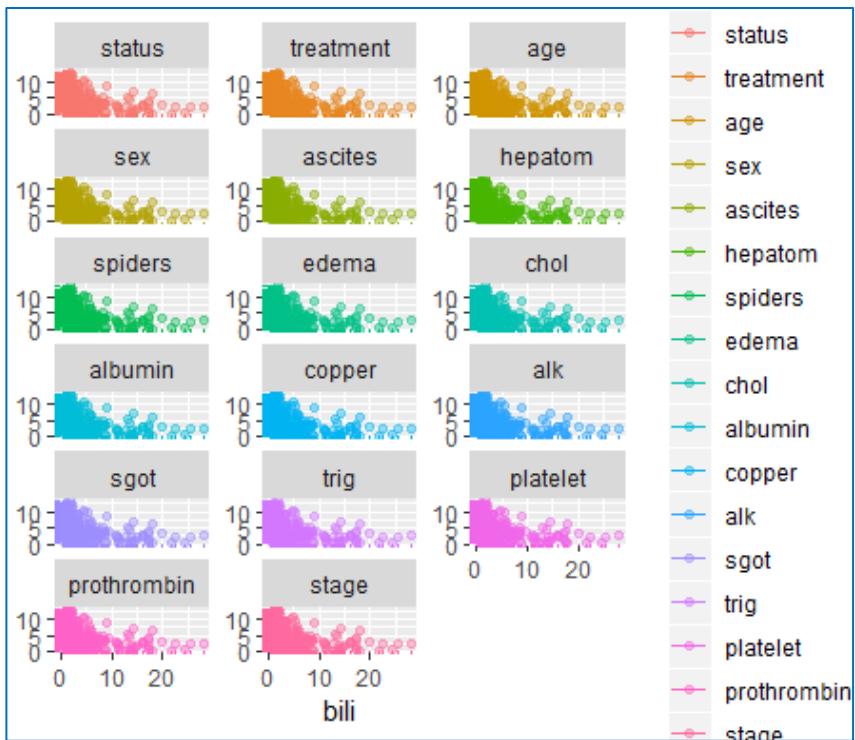
##      Years status variable value
## 1  1.095890     1 treatment    1
## 2 12.328767     0 treatment    1
## 3  2.772603     1 treatment    1
## 4  5.273973     1 treatment    1
## 5  4.120548     0 treatment    2
## 6  6.857534     1 treatment    2
```

### *Using shiny GUI for colorspace*

```
choose_palette("tcltk")
```

*Analog to: choose\_palette(gui = "shiny")*

```
ggplot(dta, aes(x=bili, y=Years, color=variable)) + geom_point(alpha=.4) + geom_rug(data=dta) +
  labs(y="", x="bili") + scale_fill_gradientn(colours =
colorspace::rainbow_hcl(17)) +
  facet_wrap(~variable, scales="free_y", ncol=3)
```

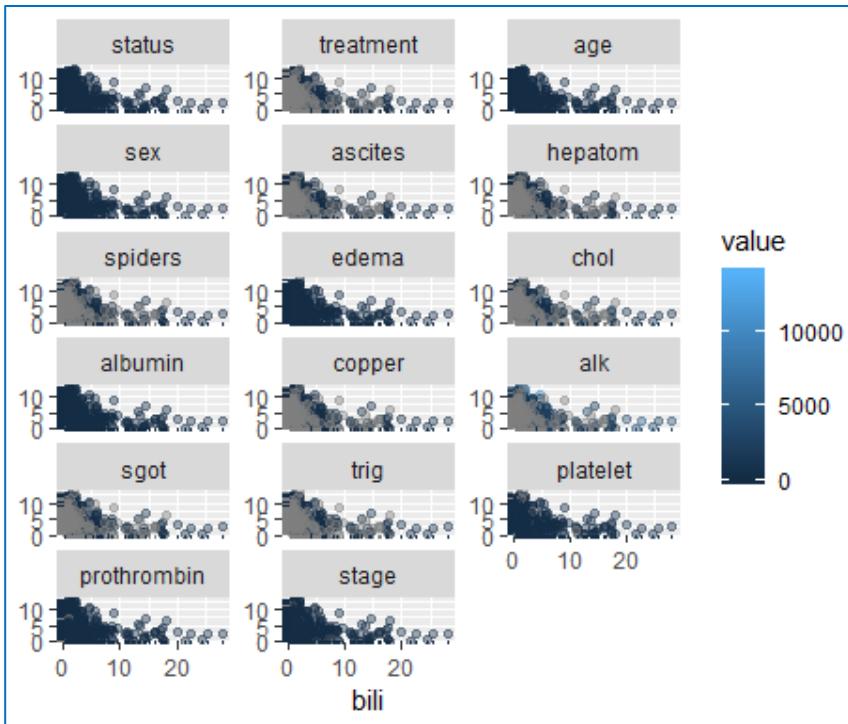


```
ggplot(dta, aes(x=bili, y=Years, color=value)) + geom_point(alpha=.4) + geom_rug(data=dta) +
  labs(y="", x="bili") + scale_fill_brewer(type = "seq",
  palette = "Set2") +
  facet_wrap(~variable, scales="free_y", ncol=3)
```

### Markdown Note.

```
```{r results = 'markup'}
```

If `'hide'`, knitr will not display the code's results in the final document. If `'hold'`, knitr will delay displaying all output pieces until the end of the chunk. If `'asis'`, knitr will pass through results without reformatting them (useful if results return raw HTML, etc.)



### *Grow and store the random survival forest*

```
rfsrc_pbc <- rfsrc(Surv(Years, status) ~ .,
                      data = pbc.trial)
```

Use random splitting (nsplit = 10) and impute missing values (na.action = “na.impute”)

```
rfsrc_pbc2 <- rfsrc(Surv(Years, status) ~ .,
                      data = pbc.trial,
                      nsplit = 10,
                      na.action = "na.impute")
```

### *Print the forest summary*

```
rfsrc_pbc
##                                     Sample size: 276
##                                     Number of deaths: 111
##                                     Number of trees: 1000
##          Forest terminal node size: 3
##          Average no. of terminal nodes: 66.925
## No. of variables tried at each split: 5
```

```

##          Total no. of variables: 17
##                               Analysis: RSF
##                               Family: surv
##          Splitting rule: logrank
##          Error rate: 17.46%
rfsrc_pbc2
##          Sample size: 312
##          Number of deaths: 125
##          Was data imputed: yes
##          Number of trees: 1000
##          Forest terminal node size: 3
##          Average no. of terminal nodes: 74.26
## No. of variables tried at each split: 5
##          Total no. of variables: 17
##                               Analysis: RSF
##                               Family: surv
##          Splitting rule: logrank *random*
##          Number of random split points: 10
##          Error rate: 16.49%

```

The print.rfsrc function returns information on how the random forest was grown. Here the family = "surv" forest has ntree = 1000 trees (the default ntree argument). We used nsplitt = 10 random split points to select random split rule, instead of an optimization on each variable at each split for performance reasons.

### Predict Patient Survival

Now, we predict the survival for 106 patients not in randomized trial:

```

pbc.test$status<-ifelse(pbc.test$status == "T",1,0)
head(pbc.test)
  status treatment      age sex ascites hepatom spiders
313     0        NA 60.04110    1      NA      NA      NA
314     0        NA 65.04384    1      NA      NA      NA
315     0        NA 54.03836    1      NA      NA      NA
316     0        NA 75.05205    1      NA      NA      NA
317     0        NA 62.04384    1      NA      NA      NA
318     0        NA 43.03014    1      NA      NA      NA
  edema bili chol albumin copper alk sgot trig platelet
313   0.0   0.7   NA    3.65    NA    NA    NA    NA    378
314   0.5   1.4   NA    3.04    NA    NA    NA    NA    331
315   0.0   0.7   NA    4.03    NA    NA    NA    NA    226

```

```

316  0.5  0.7  NA   3.96    NA  NA  NA  NA  NA
317  0.0  0.8  NA   2.48    NA  NA  NA  NA  273
318  0.0  0.7  NA   3.68    NA  NA  NA  NA  306
      prothrombin stage    Years
313       11.0     NA 11.128767
314       12.1      4  9.756164
315       9.8      4  7.791781
316      11.3      4  5.673973
317      10.0     NA  8.301370
318       9.5      2  4.602740
rfsrc_pbc_test <- predict(rfsrc_pbc,
                           newdata = pbc.test,
                           na.action = "na.impute")

```

### *Print prediction summary*

```

rfsrc_pbc_test
##   Sample size of test (predict) data: 106
##           Was test data imputed: yes
##           Number of grow trees: 1000
##   Average no. of grow terminal nodes: 66.925
##           Total no. of grow variables: 17
##           Analysis: RSF
##           Family: surv

```

Next, we setup the subsequent prediction summary:

```

rfsrc_pbc_test2 <- predict(rfsrc_pbc2,
                            newdata = pbc.test,
                            na.action = "na.impute")

```

Then, we print the subsequent prediction summary:

```

rfsrc_pbc_test2
##   Sample size of test (predict) data: 106
##           Was test data imputed: yes
##           Number of grow trees: 1000
##   Average no. of grow terminal nodes: 74.26
##           Total no. of grow variables: 17
##           Analysis: RSF
##           Family: surv

```

## Strength of Predictors

### Variable Importance

Variable importance (VIMP) was originally defined in CART using a measure involving surrogate variables (Breiman, H., Olshen, & Stone, 1984). The most popular VIMP method uses a prediction error approach involving "noising-up" each variable in turn. VIMP for a variable  $xv$  is the difference between prediction error when  $xv$  is noised up by randomly permuting its values, compared to prediction error under the observed values (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010).

Since VIMP is the difference in Out-of-Bag prediction error before and after permutation, a large VIMP value indicates that misspecification detracts from the predictive accuracy in the forest. If VIMP is close to zero the variable contributes nothing to predictive accuracy, and negative values indicate the predictive accuracy improves when the variable is misspecified. In the later 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.

Now, we extract VIMP measures for each of the variables used to grow the forest.

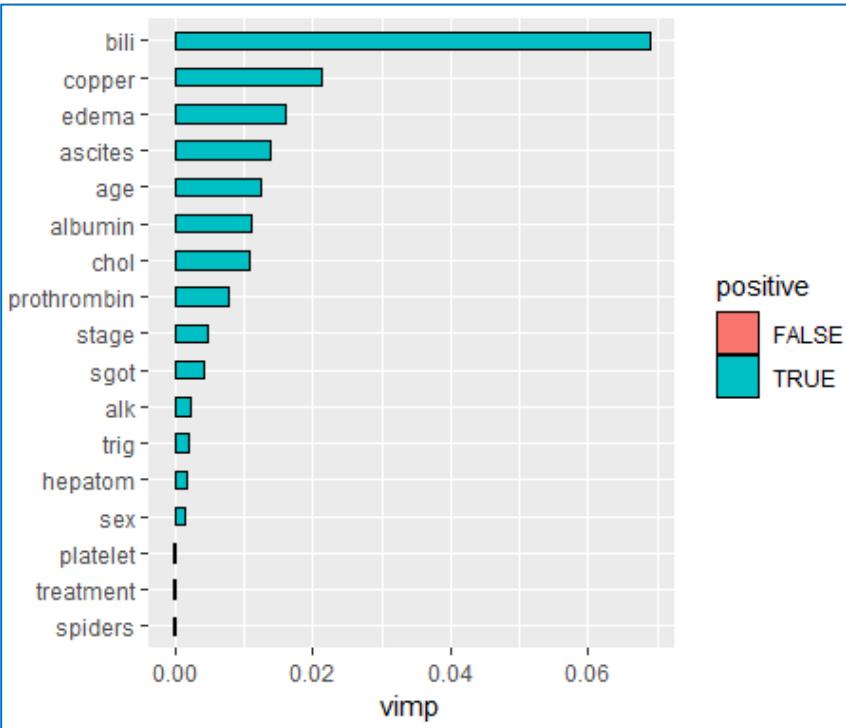
```
gg_variable(rfsrc_pbc)
gg_dta<-gg_vimp(rfsrc_pbc)
gg_dta

##           vars   set          vimp positive
## 1        bili VIMP  0.0689719956    TRUE
## 2      copper VIMP  0.0213892597    TRUE
## 3       edema VIMP  0.0161389117    TRUE
## 4     ascites VIMP  0.0138731402    TRUE
## 5        age VIMP  0.0124575963    TRUE
## 6     albumin VIMP  0.0112094109    TRUE
## 7       chol VIMP  0.0109248146    TRUE
## 8 prothrombin VIMP  0.0077009750    TRUE
## 9      stage VIMP  0.0047462763    TRUE
## 10      sgot VIMP  0.0040700859    TRUE
## 11       alk VIMP  0.0021514165    TRUE
```

```

## 12      trig VIMP  0.0019009172    TRUE
## 13   hepatom VIMP  0.0016792764    TRUE
## 14       sex VIMP  0.0013872642    TRUE
## 15 platelet VIMP -0.0001474261   FALSE
## 16 treatment VIMP -0.0002531301   FALSE
## 17   spiders VIMP -0.0003258470   FALSE
plot(gg_dta)

```



```

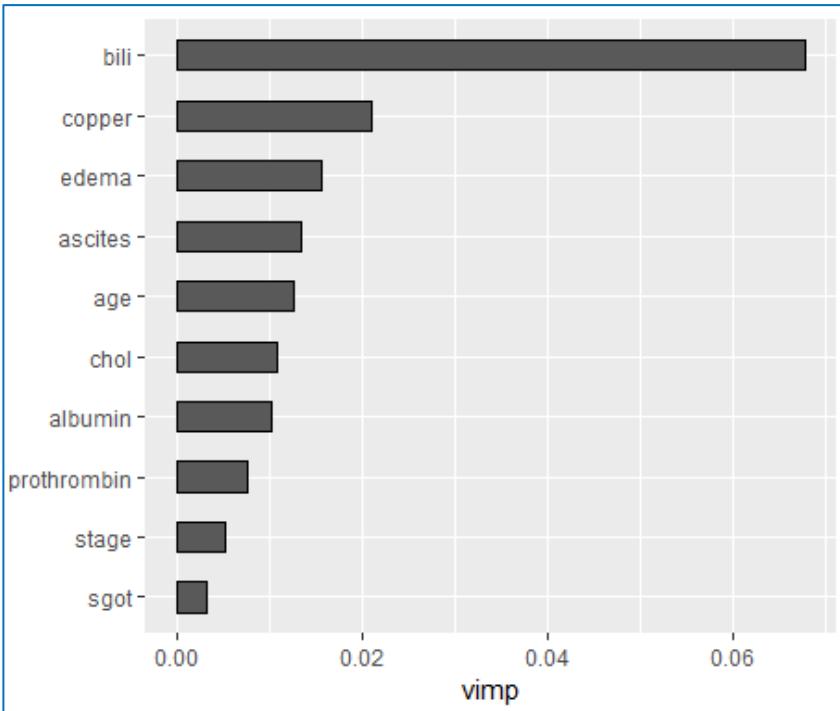
gg_dta_10<-gg_vimp(rfsrc_pbc, nvar=10)
## Warning in gg_vimp.rfsrc(rfsrc_pbc, nvar = 10): rfsrc object does not
## contain VIMP information. Calculating...
gg_dta_10
##          vars  set      vimp positive
## 1        bili VIMP  0.067855241    TRUE
## 2    copper VIMP  0.021075369    TRUE
## 3     edema VIMP  0.015652838    TRUE
## 4   ascites VIMP  0.013474512    TRUE
## 5       age VIMP  0.012696169    TRUE
## 6       chol VIMP  0.010789392    TRUE

```

```

## 7      albumin VIMP 0.010124282    TRUE
## 8  prothrombin VIMP 0.007659104    TRUE
## 9       stage VIMP 0.005156435    TRUE
## 10      sgot VIMP 0.003290305    TRUE
plot(gg_dta_10)

```



```

plot(rfsrc_pbc, lbls = st.labs) +
  theme(legend.position = c(0.8,0.2)) +
  labs(fill = "VIMP > 0") +
  scale_fill_brewer(palette = "Set1")

```

### *Minial Depth*

In VIMP, predictive risk factors are determined by testing the forest model prediction under alternative data settings, ranking the most important variables according to their impact on predictive ability of the forest. Another method uses inspection of the forest construction to rank variables. Minimal depth (Ishwaran, Kogalur, Chen, & J., 2011) (Ishwaran H., Kogalur, Gorodeski, & Minn, 2010) assumes that variables with high impact on the prediction are those that most frequently split

nodes nearest to the root node, where they partition the largest samples of the population.

Within each tree, node levels are numbered based on their relative distance to the root of the tree (with the root at 0). Minimal depth measures important risk factors by averaging the depth of the first split for each variable over all trees within the forest. We assume that in the metric smaller minimal depth values indicate the variable separates large groups of observations, and therefore has a large impact on the forest prediction.

In general, to select variables according to VIMP, we examine the VIMP values, looking for some point along the ranking where there is a large difference in VIMP measures. Given minimal depth is a quantitative property of the forest construction, Ishwaran et al. (2010) also derive an analytic threshold for evidence of variable impact. A simple optimistic threshold rule uses the mean of the minimal depth distribution, classifying variables with minimal depth lower than this threshold as important in forest prediction.

The randomForestSRC var.select function uses the minimal depth methodology for variable selection, returning an object with both minimal depth and vimp measures. The ggRandomForests gg\_minimal\_depth function is analogous to the gg\_vimp function. Variables are ranked from most important at the top (minimal depth measure), to least at the bottom (maximal minimal depth).

### ***Return an object with both minimal depth and vimp measures***

```
varsel_pbc <- var.select(rfsrc_pbc)
## minimal depth variable selection ...
## -----
## family          : surv
## var. selection  : Minimal Depth
## conservativeness : medium
## x-weighting used? : TRUE
## dimension       : 17
## sample size      : 276
## ntree            : 1000
## nsplit           : 0
## mtry              : 5
## nodesize         : 3
## refitted forest   : FALSE
```

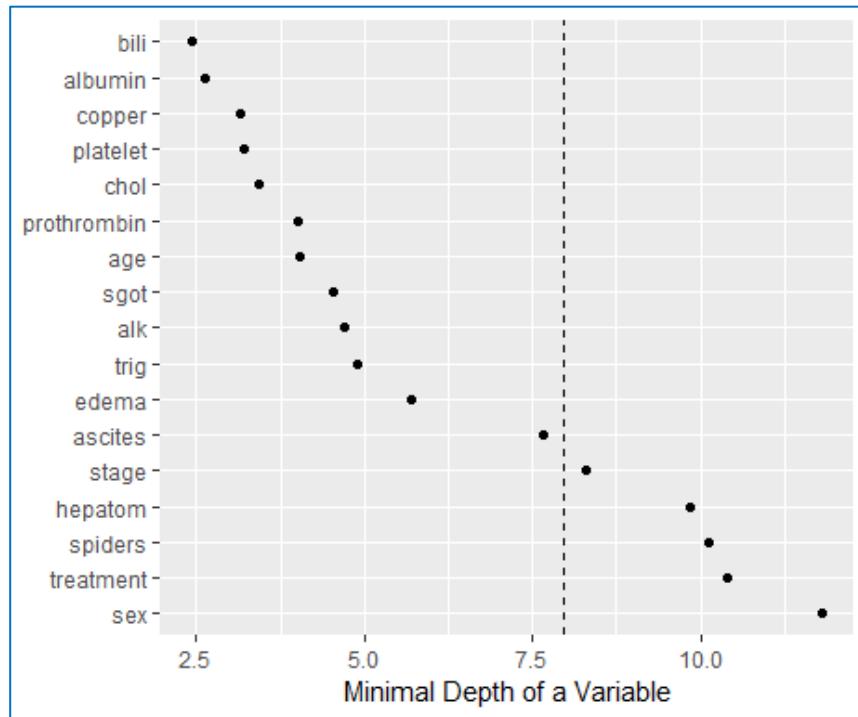
```

## model size      : 12
## depth threshold : 7.9681
## PE (true OOB)   : 17.4626
##
##
## Top variables:
##              depth vimp
## bili        2.431  NA
## albumin     2.617  NA
## copper      3.139  NA
## platelet    3.193  NA
## chol         3.434  NA
## prothrombin 3.995  NA
## age          4.029  NA
## sgot         4.531  NA
## alk          4.708  NA
## trig         4.893  NA
## edema        5.706  NA
## ascites     7.658  NA
## -----
ggMindepth <- gg_minimal_depth(varsel_pbc, lbls = Years)
print(ggMindepth)

## -----
## gg_minimal_depth
## model size      : 12
## depth threshold : 7.9681
##
## PE :[1] 17.463
## -----
##
## Top variables:
##              depth vimp
## bili        2.43  NA
## albumin     2.62  NA
## copper      3.14  NA
## platelet    3.19  NA
## chol         3.43  NA
## prothrombin 4.00  NA
## age          4.03  NA
## sgot         4.53  NA
## alk          4.71  NA
## trig         4.89  NA
## edema        5.71  NA
## ascites     7.66  NA
## -----

```

```
plot(ggMindepth)
## Coordinate system already present. Adding new coordinate system, which will replace the existing one.
```



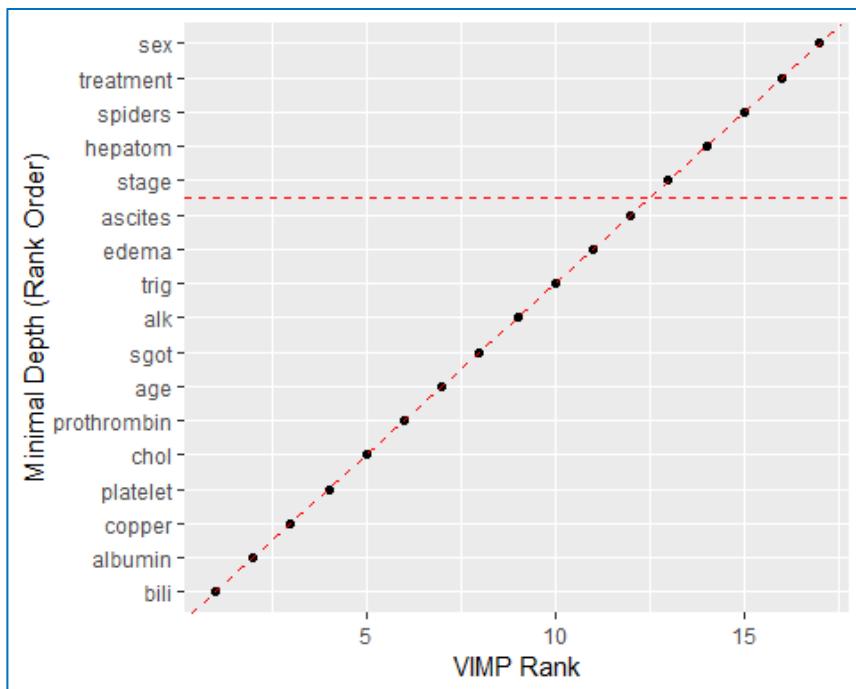
### Both minimal depth and VIMP

VIMP and Minimal Depth measures use different criteria, so we expect the variable ranking to be slightly different. We use gg\_minimal\_vimp function to compare rankings between minimal depth and VIMP. In this call, we plot the stored gg\_minimal\_depth object (gg\_md), which would be equivalent to calling `plot.gg_minimal_vimp(ggMindepth)` or `plot(gg_minimal_vimp(ggMindepth))`.

```
plot(gg_minimal_vimp(ggMindepth))
```

### Get the minimal depth selected variables

```
xvar <- varsel_pbc$topvars
xvar
## [1] "bili"   "albumin" "copper"  "platelet" "chol"
## [6] "prothrombin" "age"    "sgot"    "alk"      "trig"
## [11] "edema"   "ascites"
```



The points along the red dashed line indicate where the measures agree. Points above the red dashed line are ranked higher by VIMP than by minimal depth, indicating the variables are more sensitive to misspecification. Those below the line have a higher minimal depth ranking, indicating they are better at dividing large portions of the population. The further the points are from the line, the more the discrepancy between measures. The table below compares them.

Variable	Min depth	VIMP
age	1	1
albumin	2	5
alk	3	2
bili	4	4
chol	5	10
copper	6	3

Variable	Min depth	VIMP
edema	7	8
platelet	8	12
prothrombin	9	9
sgot	10	13
stage	11	16
trig	12	7

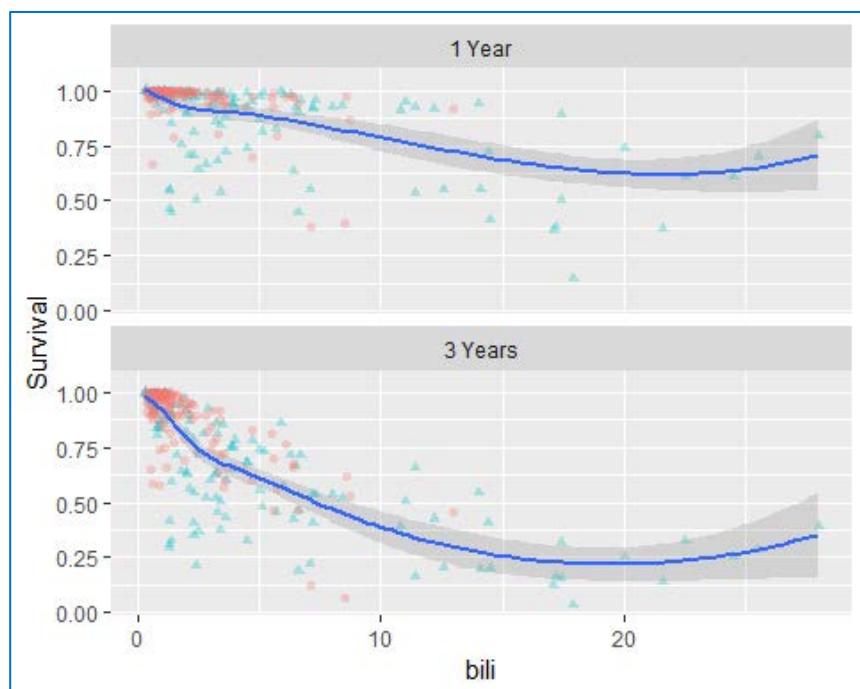
## Data generation

```
ggrf <- gg_variable(rfsrc_pbc, time = c(1, 3),  
                     time.labels = c("1 Year", "3 Years"))
```

## Plot the bilirubin variable dependence plot

Here we generate variable dependence plots of survival at 1 and 3 years on bilirubin variable. Individual cases are marked with red circles (alive or censored) and green triangles (dead). Loess smooth curve with shaded 95% confidence band indicates decreasing survival with increasing bilirubin.

```
plot(ggrf, xvar = "bili", se = .95, alpha = .3) +  
  labs(y = "Survival", x = "bili") +  
  theme(legend.position = "none")  
  
## Warning: Ignoring unknown parameters: se  
## `geom_smooth()` using method = 'loess' and formula 'y ~  
x'
```



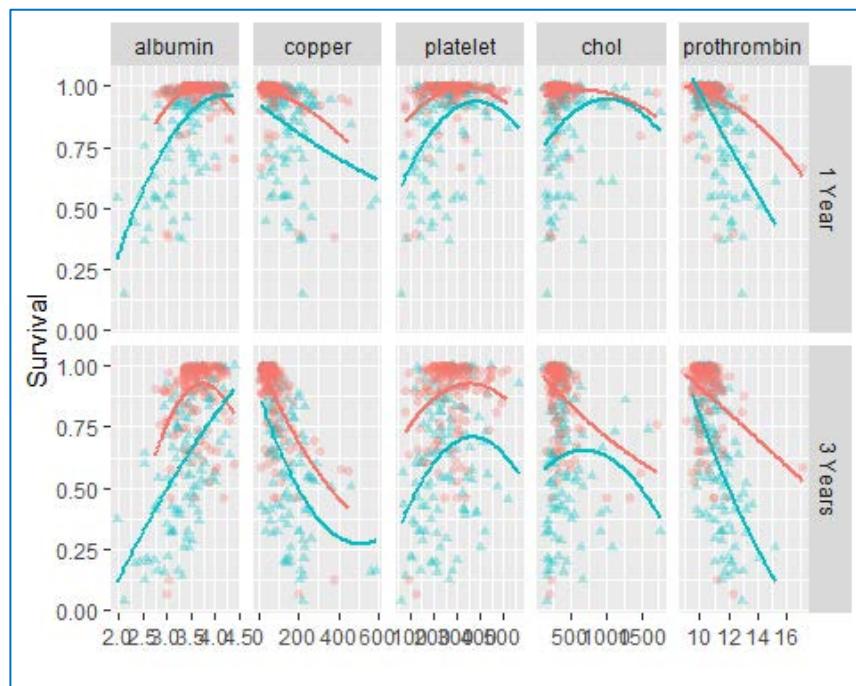
## Pull the categorical variables

```
xvar.cat <- c("edema", "stage")
xvar <- xvar[-which(xvar %in% xvar.cat)]
```

## plot the next 5 continuous variable dependence plots.

We now generate variable dependence plots of predicted survival at 1 and 3 years on continuous variables of interest. Individual cases are marked with red circles for censored cases and green triangles for death events. Loess smooth curve indicates the survival trend with increasing values.

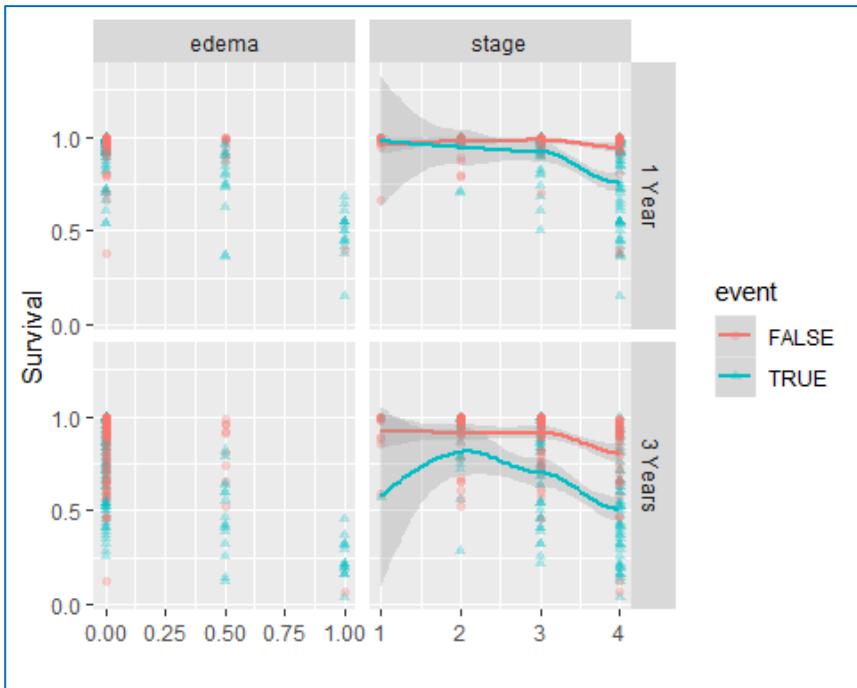
```
plot(grfr, xvar = xvar[2:6], panel = TRUE,
      se = FALSE, alpha = .3,
      method = "glm", formula = y~poly(x,2)) +
  labs(y = "Survival") +
  theme(legend.position = "none") #optional
```



Variable dependence plots for categorical variables are constructed using boxplots to show the distribution of the predictions within each category. Here we demonstrate variable dependence of survival 1 and 3 years on the edema categorical variable. Symbols with red circles

indicate censored cases and green triangles indicate death events. Boxplots indicate distribution of predicted survival for all observations within each edema group.

```
plot(ggrf, xvar = xvar.cat, panel = TRUE, notch = TRUE, alpha = .3) +
  labs(y = "Survival") + scale_fill_gradientn(colours = colorspace::rainbow_hcl(17))
```



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

### Partial Dependence Plot

The partial dependence plot (PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a machine learning model (J. H. Friedman 2001). A partial dependence function is

the partial distribution of a multi-variable distribution function and its plot can show whether the relationship between the target and a feature is linear, monotonic or more complex. For example, when applied to a linear regression model, partial dependence plots always show a linear relationship.

The partial dependence function for multi-variable regression is defined as:

$$\hat{f}_{x_s}(x_s) = E_{x_c}[\hat{f}(x_s, x_c)] = \int \hat{f}(x_s, x_c) d\mathbb{P}(x_c)$$

The  $x_s$  are the features for which the partial dependence function should be plotted and  $x_c$  are the other features used in the machine learning model  $\hat{f}$ . Usually, there are only one or two features in the set  $S$ . The feature(s) in  $S$  are those for which we want to know the effect on the prediction. The feature vectors  $x_s$  and  $x_c$  combined make up the total feature space  $x$ . Partial dependence works by marginalizing the machine learning model output over the distribution of the features in set  $C$ , so that the function shows the relationship between the features in set  $S$  we are interested in and the predicted outcome. By marginalizing over the other features, we get a function that depends only on features in  $S$ , interactions with other features included.

The partial function  $\hat{f}_{x_s}$  is estimated by calculating averages in the training data, also known as Monte Carlo method:

$$\hat{f}_{x_s}(x_s) = \frac{1}{n} \sum_{i=1}^n \hat{f}(x_s, x_c^{(i)})$$

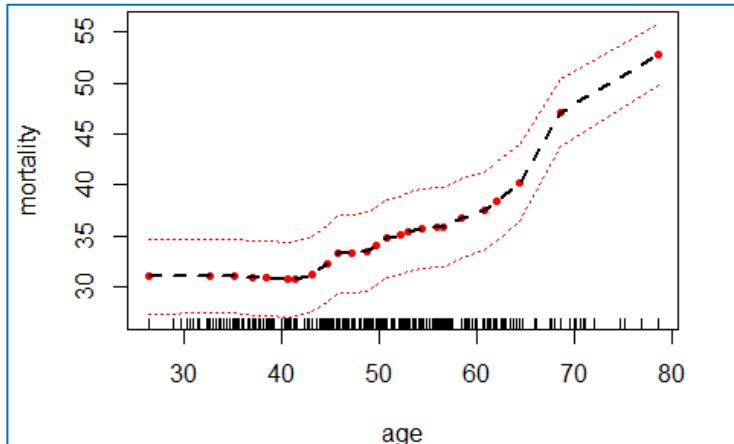
The partial function tells us for given value(s) of features  $S$  what the average marginal effect on the prediction is. In this formula,  $x_c^{(i)}$  are actual feature values from the dataset for the features in which we are not interested, and  $n$  is the number of instances in the dataset. An assumption of the PDP is that the features in  $C$  are not correlated with the features in  $S$ . If this assumption is violated, the averages calculated for the partial dependence plot will include data points that are very unlikely or even impossible.

For classification where the machine learning model outputs probabilities, the partial dependence plot displays the probability for a certain class given different values for feature(s) in  $S$ . An easy way to deal with multiple classes is to draw one line or plot per class.

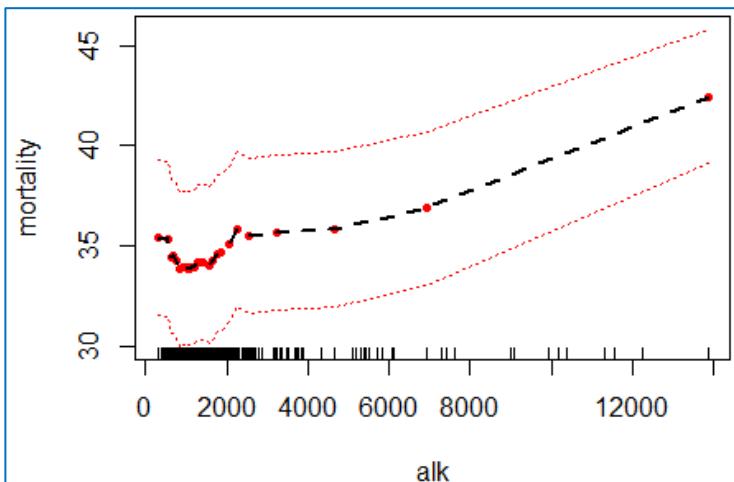
The partial dependence plot is a global method: The method considers all instances and gives a statement about the global relationship of a feature with the predicted outcome.

### ***Calculate the 1- and 3-year partial dependence***

```
partial_age <- plot.variable(rfsrc_pbc, xvar.names = "age",  
", partial=TRUE)
```



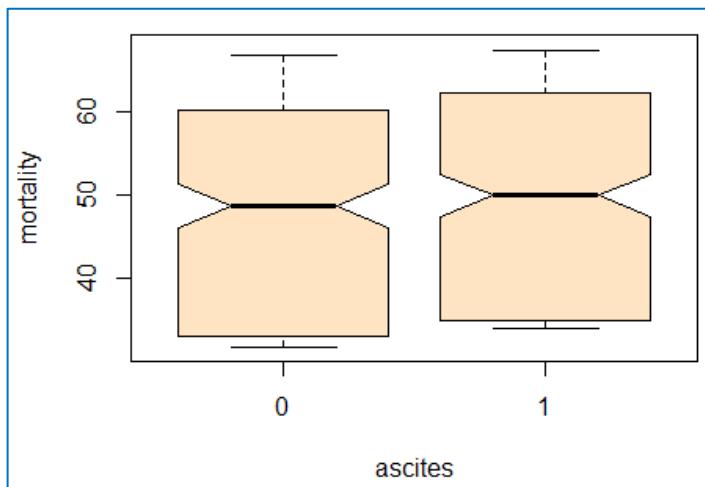
```
partial_alk <- plot.variable(rfsrc_pbc, xvar.names = "alk",  
", partial=TRUE)
```



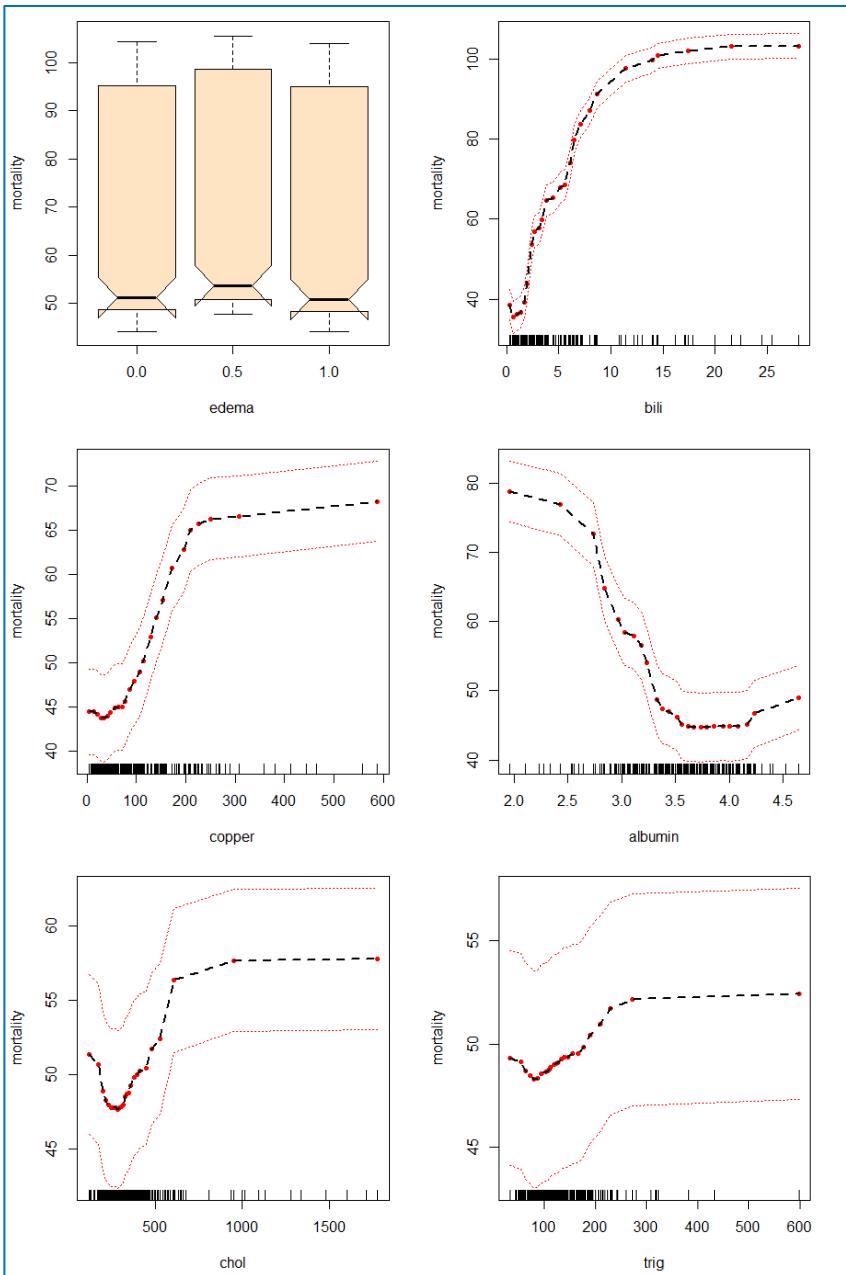
Rather than plot all of the PDPs here, we will plot the most interesting, after an explantion of the PDP for categorical variables.

### Categorical features

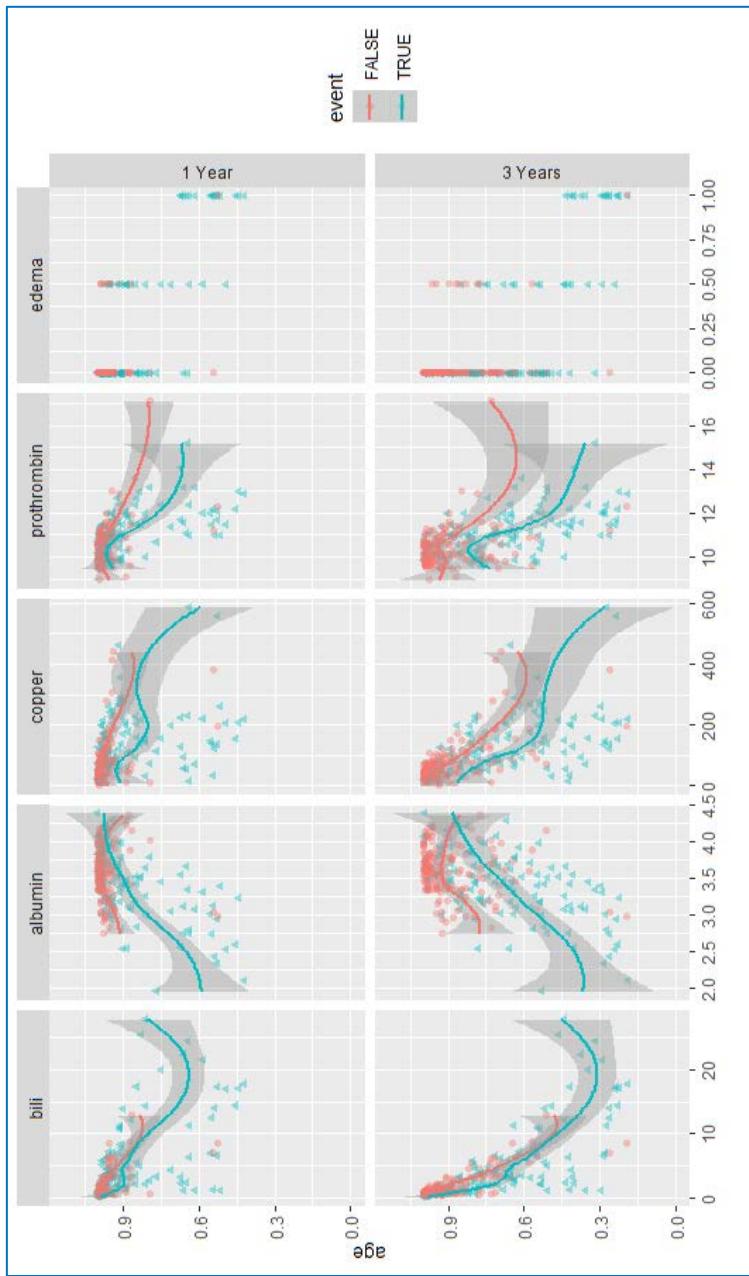
So far, we have only thought about numerical features. For categorical features, the partial dependence is very easy to compute. For each of the categories, we get a PDP estimate by forcing all data instances to have the same category. For example, if we look at the PBC dataset and are interested in the partial dependence plot for the edema, we get 3 numbers, one for each category. To compute the value for “no edema and no diuretic therapy for edema”, we replace the edema of all data instances with “no edema and no diuretic therapy for edema” and average the predictions.



```
part_var01 <- plot.variable(rfsrc_pbc, xvar.names =
  c("edema","bili"), partial = TRUE)
part_var02 <- plot.variable(rfsrc_pbc, xvar.names =
  c("copper","albumin"), partial = TRUE)
part_var03 <- plot.variable(rfsrc_pbc, xvar.names =
  c("chol","trig"), partial = TRUE)
```



```
xvar <- ggMindepth$topvars
plot(gggrf, xvar=xvar, panel=TRUE, alpha=.4) + labs(y="age",
, x="`")
```



Recall that presence of edema has three values representing categorical flags:

- 0.0 = no edema and no diuretic therapy for edema;

- 0.5 = edema present without diuretics, or edema resolved by diuretics;
- 1.0 = edema despite diuretic therapy

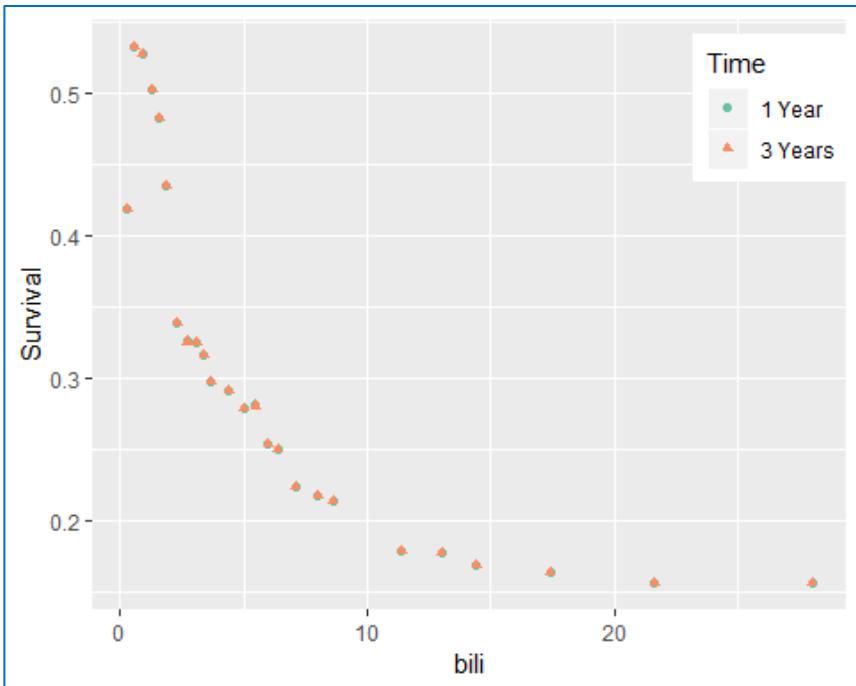
The distributions for these categories is surprisingly similar and we might need to investigate the physiological and pathophysiological aspect.

Next, we convert all partial plots to gg\_partial objects.

```
gg_dta <- lapply(partial_pbc, gg_partial)
```

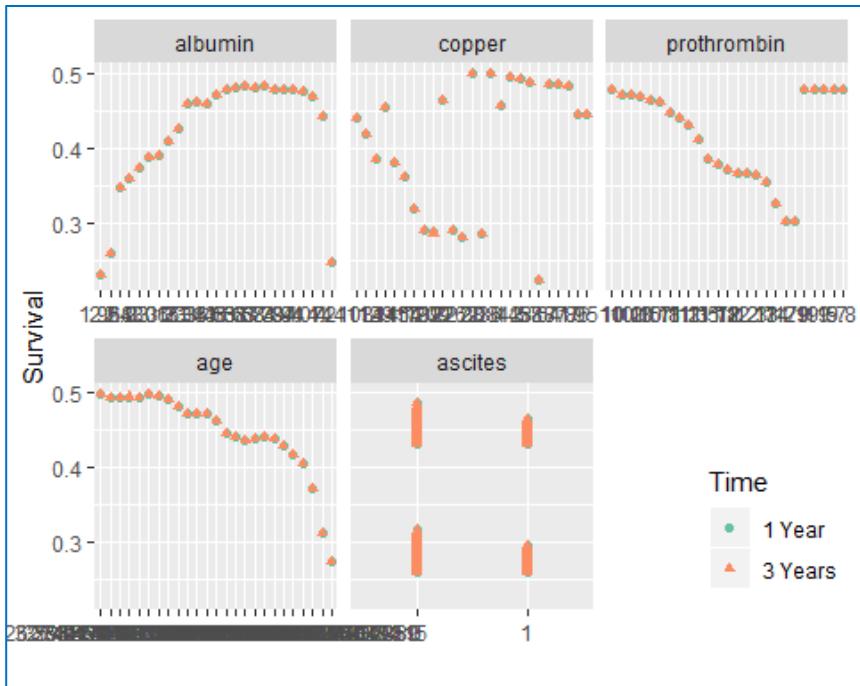
Now, we combine the objects to get multiple time curves along variables on a single figure.

```
gg_dta <- lapply(partial_pbc, gg_partial)
pbc_ggpart <- combine.gg_partial(gg_dta[[1]], gg_dta[[2]],
lbls = c("1 Year", "3 Years"))
#pbc_ggpart2 <- combine.gg_partial(ggRandomForests::gg_partial(gg_dta[[1]], gg_dta[[2]]), lbls = c("1 Year", "3 Years"))
plot(pbc_ggpart[["bili"]]) +
  theme(legend.position = c(.9, .85)) +
  labs(y = "Survival",
       x = "bili",
       color = "Time", shape = "Time") +
  scale_color_brewer(palette = "Set2")
```



Let's create a temporary holder and remove the stage and edema data.

```
ggpart <- pbc_ggpart
ggpart$edema <- ggpart$stage <- NULL
ggpart$bili <- ggpart$sgot <- ggpart$chol <- NULL
ggpart$platelet <- ggpart$trig <- ggpart$alk <- NULL
# Panel plot the remainder.
plot(ggpart, panel = TRUE) +
  labs(x = "", y = "Survival", color = "Time", shape = "Time") +
  scale_color_brewer(palette = "Set2") +
  theme(legend.position = c(.9, .15))
```



```

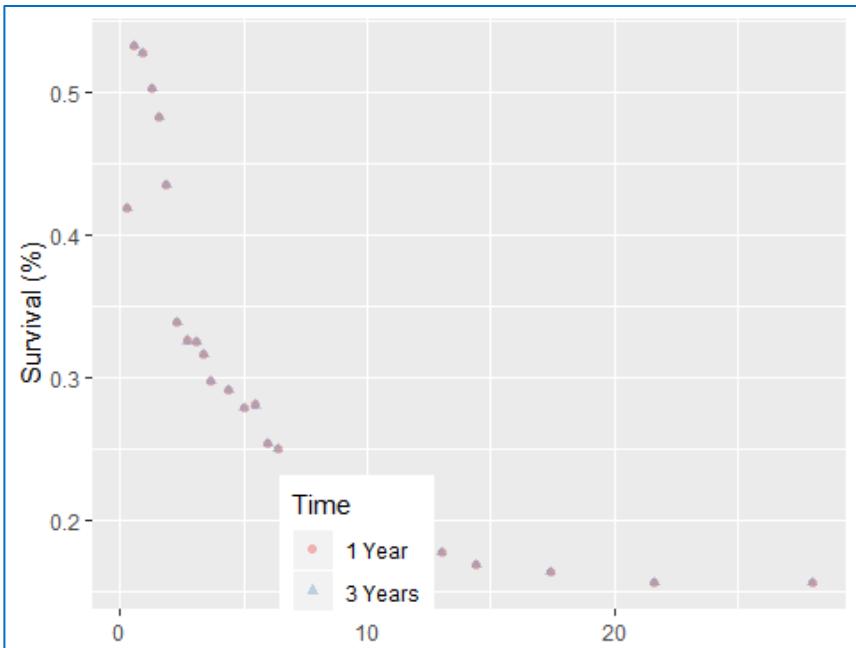
ggpart <- pbc_ggpart
head(ggpart$edema)

##          yhat edema   group
## 1    0.4341768     0 1 Year
## 2    0.4317110     0 1 Year
## 3    0.2670310     0 1 Year
## 4    0.4519583     0 1 Year
## 5    0.4299843     0 1 Year

ggpart$stage
## NULL

names(ggpart) <- c("edema", "stage")
class(ggpart) <- c("gg_partial_list", class(ggpart))
plot(ggpart$edema, panel=TRUE, notch = TRUE, alpha = .3 +
  labs(x = "", y = "Survival (%)", color="Time", shape="Ti
me") +
  scale_color_brewer(palette = "Set1") +
  theme(legend.position = c(.35, .1)))
## Warning: Ignoring unknown parameters: panel, notch

```



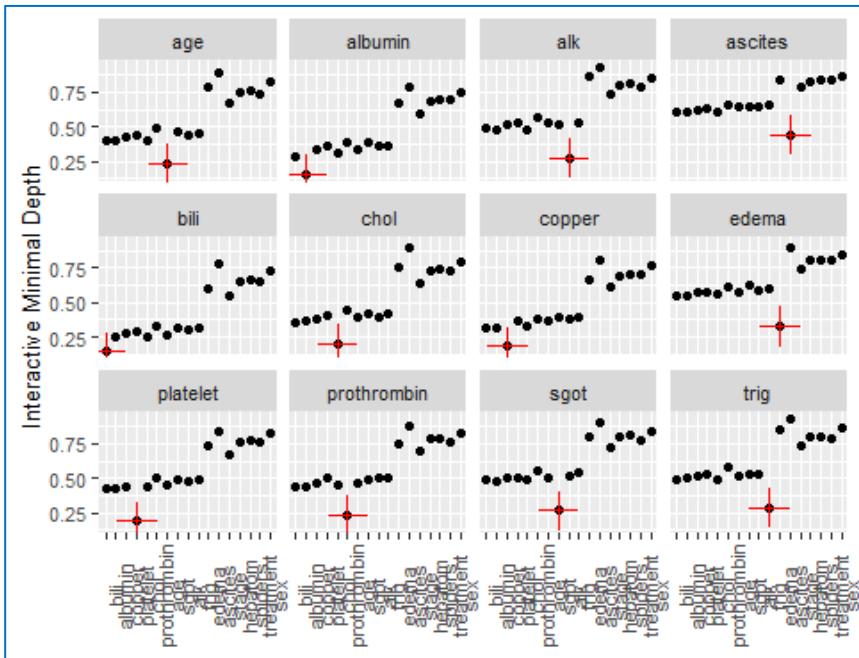
The `gg_interaction` function wraps the `find.interaction` matrix for use with the provided S3 plot and print functions.

```
interaction_pbc <- find.interaction(rfsrc_pbc)
##                               No. of variables: 17
##   Variables sorted by minimal depth?: TRUE
##      bili albumin copper platelet chol prothrombin age sgot alk
## bili    0.14    0.25    0.28    0.29  0.25      0.33 0.26 0.32 0.30
## albumin 0.29    0.16    0.33    0.36  0.31      0.39 0.33 0.38 0.37
## copper  0.32    0.31    0.18    0.37  0.33      0.39 0.36 0.39 0.38
## platele 0.43    0.43    0.44    0.19  0.44      0.50 0.46 0.49 0.48
## chol    0.36    0.36    0.38    0.41  0.20      0.44 0.39 0.41 0.40
## prothro 0.44    0.44    0.46    0.50  0.46      0.23 0.46 0.49 0.50
## age     0.40    0.40    0.43    0.44  0.40      0.49 0.24 0.47 0.44
## sgot    0.49    0.48    0.51    0.50  0.49      0.55 0.50 0.27 0.51
## alk     0.48    0.47    0.51    0.53  0.48      0.57 0.53 0.52 0.27
## trig    0.49    0.50    0.52    0.53  0.49      0.58 0.52 0.53 0.53
## edema   0.55    0.55    0.57    0.58  0.56      0.62 0.58 0.62 0.58
## ascites 0.61    0.61    0.62    0.63  0.61      0.66 0.65 0.65 0.64
## stage   0.71    0.73    0.73    0.75  0.72      0.75 0.73 0.74 0.73
## hepatom 0.82    0.83    0.82    0.83  0.81      0.83 0.82 0.83 0.83
## spiders 0.85    0.85    0.85    0.85  0.83      0.85 0.85 0.85 0.85
## treat   0.91    0.91    0.91    0.90  0.91      0.91 0.91 0.90 0.90
## sex     0.89    0.89    0.91    0.90  0.89      0.90 0.90 0.90 0.89
```

```

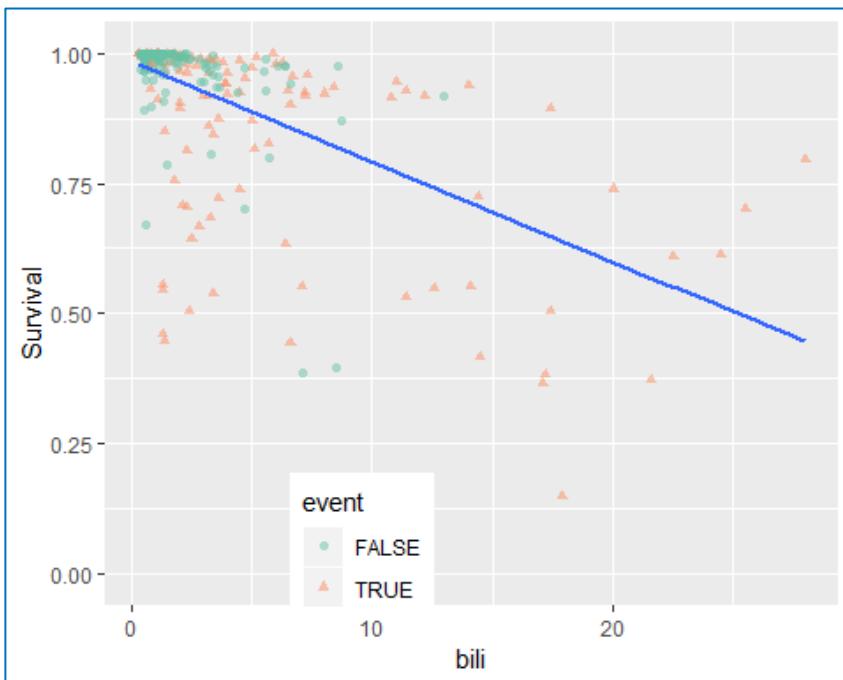
##          trig edema ascites stage hepatom spiders treatment sex
## bili      0.32  0.59   0.78  0.55   0.65  0.66   0.65  0.73
## albumin   0.36  0.67   0.79  0.60   0.69  0.70   0.69  0.75
## copper    0.39  0.67   0.81  0.62   0.70  0.71   0.70  0.77
## platelet   0.49  0.74   0.84  0.68   0.76  0.78   0.76  0.82
## chol       0.42  0.75   0.89  0.63   0.72  0.74   0.73  0.79
## prothrombin 0.50  0.75   0.88  0.70   0.78  0.78   0.76  0.83
## age        0.45  0.79   0.89  0.67   0.74  0.76   0.74  0.83
## sgot       0.54  0.80   0.91  0.72   0.80  0.81   0.77  0.84
## alk         0.53  0.86   0.93  0.74   0.81  0.81   0.79  0.85
## trig       0.29  0.85   0.93  0.74   0.80  0.80   0.79  0.87
## edema      0.60  0.33   0.90  0.75   0.80  0.81   0.81  0.84
## ascites    0.66  0.84   0.44  0.79   0.83  0.84   0.83  0.87
## stage       0.75  0.90   0.97  0.48   0.88  0.90   0.88  0.92
## hepatom    0.84  0.95   0.98  0.90   0.57  0.93   0.93  0.95
## spiders    0.86  0.95   0.98  0.90   0.94  0.59   0.92  0.96
## treatment   0.91  0.98   0.99  0.95   0.96  0.96   0.61  0.97
## sex         0.90  0.97   0.99  0.95   0.96  0.96   0.95  0.68
ggint <- gg_interaction(interaction_pbc)
plot(ggint, xvar = xvar) +
  labs(y = "Interactive Minimal Depth") +
  theme(legend.position = "none")

```



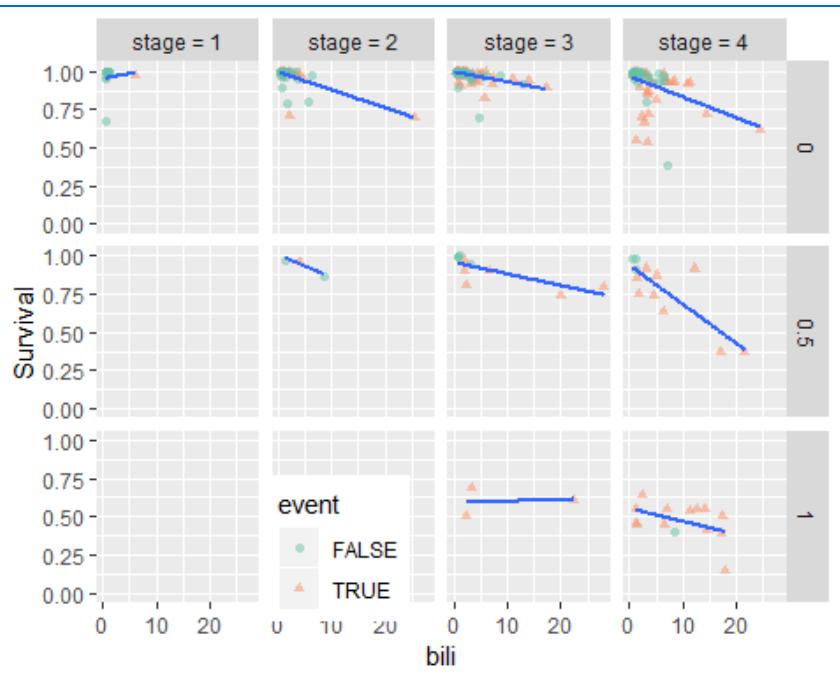
## Conditional dependence plots

```
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))
## Warning: Ignoring unknown parameters: method, se
var_dep
```



We now show the conditional dependence of survival against bilirubin, versus other categorical covariates, say edema and stage.

```
var_dep +
  facet_grid(edema~stage)
```



Furthermore, we find intervals with similar number of observations.

```
copper_cts <- quantile_pts(ggvar$copper, groups = 6, intervals = TRUE)
```

Next, we create the conditional groups and add to the gg\_variable object

```
copper_grp <- cut(ggvar$copper, breaks = copper_cts)
ggvar$copper_grp <- copper_grp
```

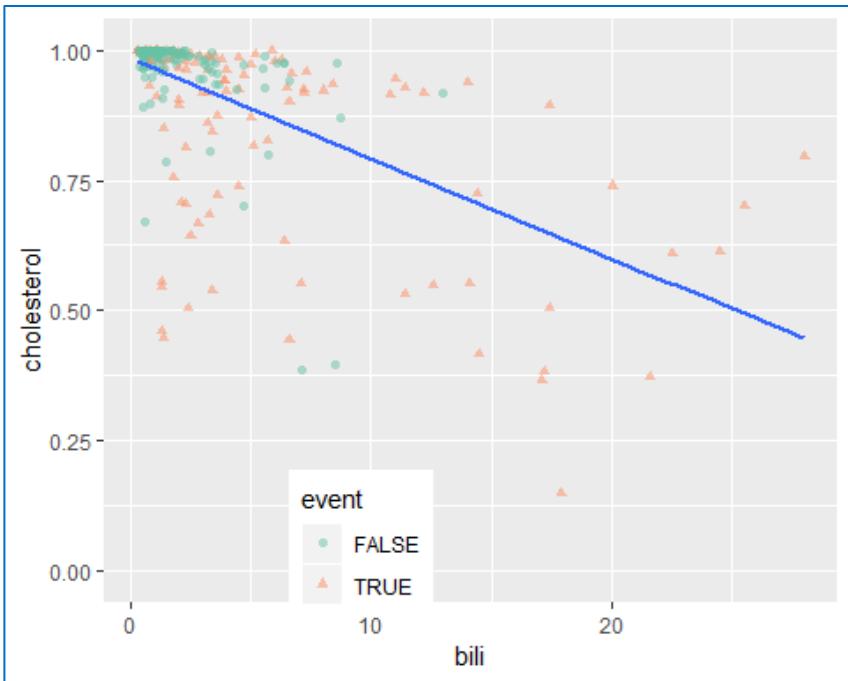
Finally, we adjust naming for facets before plotting.

```
levels(ggvar$copper_grp) <- paste("copper = ", levels(copper_grp), sep = "")
```

### **Plot.gg\_variable**

```
var_dep <- plot(ggvar, xvar = "bili", method = "glm", alpha = .5, se = FALSE) +
  labs(y = "cholesterol", x = "bili") +
  theme(legend.position = c(.35, .1)) +
  scale_color_brewer(palette = "Set2") +
  scale_shape(solid=TRUE) +
  coord_cartesian(y = c(-.01,1.01))

## Warning: Ignoring unknown parameters: method, se
var_dep
```



### Partial dependence coplots

```
data(rfsrc_pbc, package="ggRandomForests")
## Warning in data(rfsrc_pbc, package = "ggRandomForests"):
: data set
## 'rfsrc_pbc' not found
```

For the partial dependence coplots, we create the variable plot.

```
ggvar <- gg_variable(rfsrc_pbc, time = 1)
```

Then, we find intervals with similar number of observations.

```
copper_cts <- quantile_pts(ggvar$copper, groups = 6, intervals = TRUE)
```

Next, we create the conditional groups and add to the gg\_variable object.

```
copper_grp <- cut(ggvar$copper, breaks = copper_cts)
partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc, xvar =
"bili",
   groups = copper_grp,
   surv_type = "surv",
   time = 1,
   show.plots = FALSE)
```

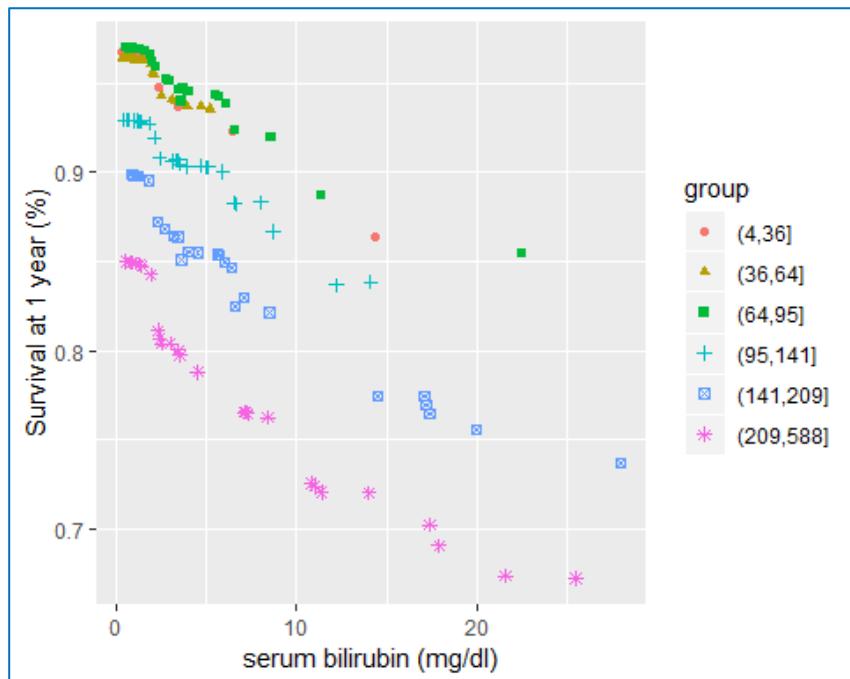
Now, we load the cached set.

```
data(partial_coplot_pbc, package="ggRandomForests")
## Warning in data(partial_coplot_pbc, package = "ggRandomForests"): data set
## 'partial_coplot_pbc' not found
```

### Partial coplot

Finally, we display the partial dependence coplots.

```
plot(partial_coplot_pbc) +
  labs(x = "serum bilirubin (mg/dl)", y = "Survival at 1 year (%)")
```

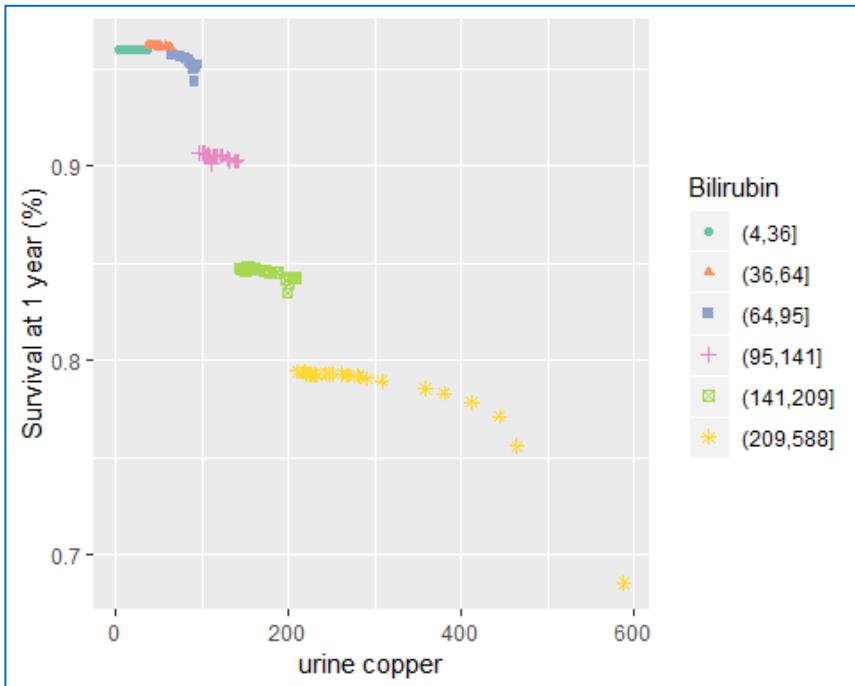


```
partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc, xvar =
  "copper",
   groups = copper_grp,
   surv_type = "surv",
   time = 1,
   show.plots = FALSE)
data(partial_coplot_pbc, package = "ggRandomForests")
```

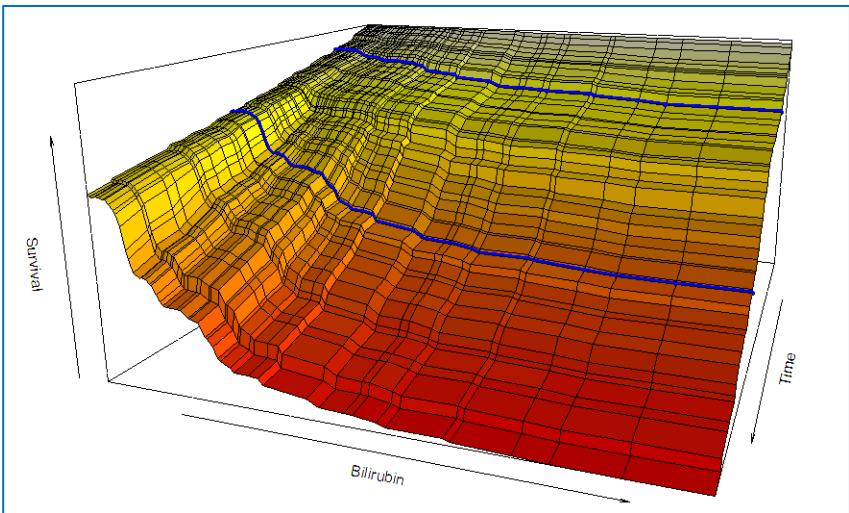
```

## Warning in data(partial_coplot_pbc, package = "ggRandomForests"): data set
## 'partial_coplot_pbc' not found
plot(partial_coplot_pbc) +
  labs(x = "urine copper",
       y = "Survival at 1 year (%)",
       color = "Bilirubin", shape = "Bilirubin") +
  scale_color_brewer(palette = "Set2")

```



The figure below shows partial dependence of survival (Z-axis) as a function of bilirubin over a five year follow up time period. The lines perpendicular to the bilirubin axis are distributed along the bilirubin variable. Lines parallel to the bilirubin axis are taken at 50 training set event times, the first event after  $t = 0$  at the back to last event before  $t = 5$  years at the front. The The code is provided in (Ehrlinger, Rajeswaran, & Blackstone, 2019, pp. 38-40)



## Business Application

### *Scenario and dataset*

A marketing department of a bank runs various marketing campaigns for cross-selling products, improving customer retention and customer services. In this example, the bank wanted to cross-sell term deposit product to its customers. Contacting all customers is costly and does not create good customer experience. So, the bank wanted to build a predictive model which will identify customers who are more likely to respond to term deposit cross sell campaign. We will use sample Marketing Data sample for building Random Forest based model using R.

### *Read and Explore data*

```
bank<-read.csv(file="C:/Users/Jeff/Documents/VIT_University/data/Banking.csv",header = T)
names(bank)
## [1] "job"      "marital"    "education"   "age"       "balance"
## [6] "homeowner" "loans"      "default"     "contact"    "length"
## [11] "campaign"  "pdays"      "previous"    "poutcome"   "RESP"
```

Input dataset has 20 independent variables and a target variable. The target variable y is binary.

```



```

Eleven % of the observations has target variable “yes” and remaining 89% observations take value “no”.

Now, we will split the data sample into development and validation samples.

```

sample.ind <- sample(2,
                     nrow(bank),
                     replace = T,
                     prob = c(0.5,0.5))
#sample.ind <- sample(2, 10000, replace=F)
cross.sell.dev <- bank[sample.ind==1,]
cross.sell.val <- bank[sample.ind==2,]
table(cross.sell.dev$RESP)/nrow(cross.sell.dev)
##
##      NO      YES
## 0.8824438 0.1175562
table(cross.sell.val$RESP)/nrow(cross.sell.val)
##
##      NO      YES
## 0.8835941 0.1164059
cross.sell.val$RESP
## [1] NO NO
## [18] NO NO NO YES NO YES NO
## [35] NO NO
## [22628] YES YES YES YES YES YES NO YES NO NO YES NO NO NO NO NO NO
## [22645] NO NO YES YES NO YES YES YES YES YES
## Levels: NO YES

```

Both development and validation samples have similar target variable distribution. This is just a sample validation.

If target variable is factor, classification decision tree is built. We can check the type of response variable.

```

class(cross.sell.dev$RESP)
## [1] "factor"

```

```
class(cross.sell.val$RESP)
## [1] "factor"
```

Class of target or response variable is factor, so a classification Random Forest will be built. The current data frame has a list of independent variables, so we can make it formula and then pass as a parameter value for randomForest.

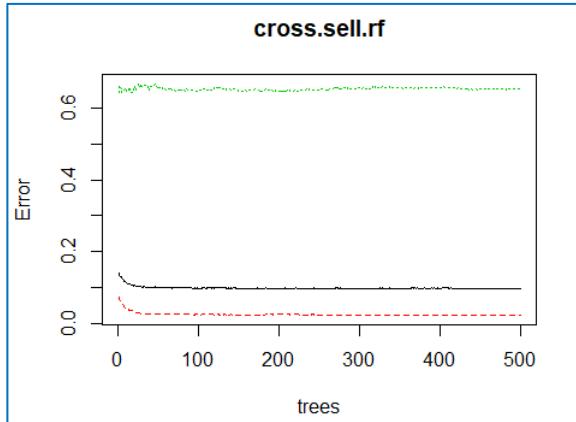
### Make Formula

```
varNames <- names(cross.sell.dev)
# Exclude ID or Response variable
varNames <- varNames[!varNames %in% c("RESP")]
# add + sign between exploratory variables
varNames1 <- paste(varNames, collapse = "+")
# Add response variable and convert to a formula object
rf.form <- as.formula(paste("RESP", varNames1, sep=" ~ "))
```

### Building Random Forest using R

Now, we have a sample data and formula for building Random Forest model. Let's build 500 decision trees using Random Forest.

```
cross.sell.rf <- randomForest(rf.form, cross.sell.dev, ntree=500, importance=T)
plot(cross.sell.rf)
```



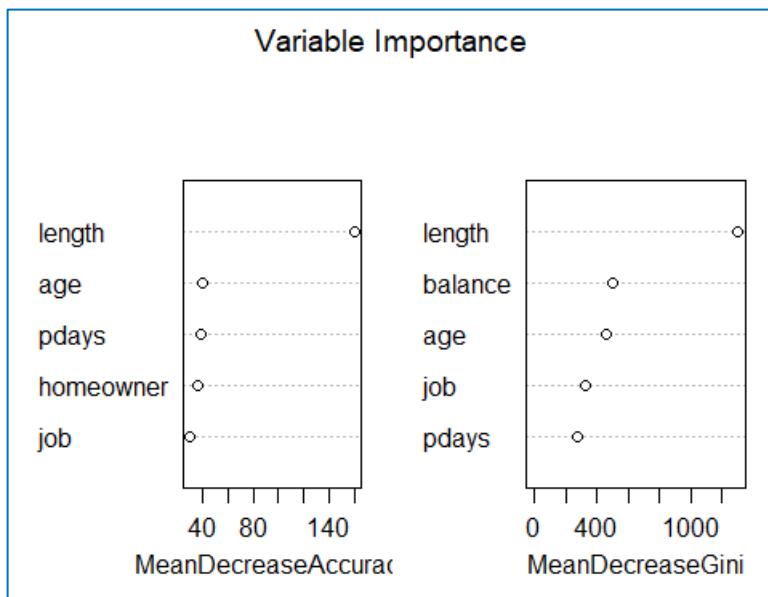
500 decision trees or a forest has been built using the Random Forest algorithm-based learning. We can plot the error rate across decision

trees. The plot seems to indicate that after 100 decision trees, there is not a significant reduction in error rate.

### Variable Importance Plot

Variable importance plot is also a useful tool and can be plotted using varImpPlot function. Top 5 variables are selected and plotted based on Model Accuracy and Gini value. We can also get a table with decreasing order of importance based on a measure (1 for model accuracy and 2 node impurity)

```
varImpPlot(cross.sell.rf,
            sort = T,
            main="Variable Importance",
            n.var=5)
```



### Variable Importance Table

```
var.imp <- data.frame(importance(cross.sell.rf,
                                    type=2))
```

### Make row names as columns

```
var.imp$Variables <- row.names(var.imp)
var.imp[order(var.imp,decreasing = T),]
```

```

##               MeanDecreaseGini Variables
## NA                      NA      <NA>
## NA.1                     NA      <NA>
## NA.2                     NA      <NA>
## NA.3                     NA      <NA>
## NA.4                     NA      <NA>
## NA.5                     NA      <NA>
## NA.6                     NA      <NA>
## NA.7                     NA      <NA>
## NA.8                     NA      <NA>
## NA.9                     NA      <NA>
## NA.10                    NA      <NA>
## NA.11                    NA      <NA>
## NA.12                    NA      <NA>
## NA.13                    NA      <NA>
## length                  1300.789733  length
## balance                 497.901416  balance
## age                      456.717286  age
## job                      320.024358  job
## pdays                   276.070140  pdays
## poutcome                217.999505  poutcome
## campaign                174.409370  campaign
## homeowner              125.707781  homeowner
## education               123.510237  education
## previous                119.876718  previous
## marital                 99.500687  marital
## contact                 98.514174  contact
## loans                   47.061799  loans
## default                 9.970018   default

```

Based on Random Forest variable importance, the variables could be selected for any other predictive modelling techniques or machine learning.

Now, we want to measure the accuracy of the Random Forest model. Some of the other model performance statistics are: - KS - Lift Chart - ROC curve

### **Predict Response Variable Value using Random Forest**

Generic predict function can be used for predicting response variable using Random Forest object.

## Predicting response variable

```
cross.sell.dev$predicted.response <- predict(cross.sell.rf, cross.sell.dev)
```

## Confusion Matrix

confusionMatrix *function* from caret package can be used for creating confusion matrix based on actual response variable and predicted value.

### Load Library or packages

```
#if(!require(e1071)) install.packages("e1071")
#if(!require(caret)) install.packages("caret")
library(e1071)
library(caret)
```

### Create Confusion Matrix

```
confusionMatrix(data=cross.sell.dev$predicted.response,
                 reference=cross.sell.dev$RESP)

## Confusion Matrix and Statistics
##
##          Reference
## Prediction    NO     YES
##           NO 19604   1188
##           YES   235   1473
##
##                  Accuracy : 0.9368
##                  95% CI  : (0.9335, 0.9399)
##      No Information Rate : 0.8817
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                  Kappa : 0.6411
##  Mcnemar's Test P-Value : < 2.2e-16
##
##                  Sensitivity : 0.9882
##                  Specificity  : 0.5536
##      Pos Pred Value : 0.9429
##      Neg Pred Value : 0.8624
##      Prevalence    : 0.8817
##      Detection Rate : 0.8713
##  Detection Prevalence : 0.9241
##      Balanced Accuracy : 0.7709
##
##      'Positive' Class : NO
```

It has accuracy of 99.81%, which is fantastic. Now we can predict response for the validation sample and calculate model accuracy for the sample.

### Predicting response variable

```
cross.sell.val$predicted.response <- predict(cross.sell.rf  
, cross.sell.val)
```

### Create Confusion Matrix

```
confusionMatrix(data=cross.sell.val$predicted.response,  
                 reference=cross.sell.val$RESP)  
  
## Confusion Matrix and Statistics  
##  
##             Reference  
## Prediction      NO     YES  
##           NO 19820   1156  
##           YES   264   1472  
##  
##                  Accuracy : 0.9375  
##                  95% CI  : (0.9343, 0.9406)  
##      No Information Rate : 0.8843  
##      P-Value [Acc > NIR] : < 2.2e-16  
##  
##                  Kappa : 0.6416  
##  Mcnemar's Test P-Value : < 2.2e-16  
##  
##                  Sensitivity : 0.9869  
##                  Specificity  : 0.5601  
##      Pos Pred Value : 0.9449  
##      Neg Pred Value : 0.8479  
##                  Prevalence : 0.8843  
##      Detection Rate  : 0.8727  
##  Detection Prevalence : 0.9236  
##      Balanced Accuracy : 0.7735  
##  
##      'Positive' Class : NO
```

Accuracy level has dropped to 91.4% but still significantly higher.

### Fit a Random Forest to the fgl data and compare with SVM

The fgl data frame has 214 rows and 10 columns. It was collected by B. German on fragments of glass collected in forensic work.

```

# if(!require(MASS)) install.packages("MASS")
library(MASS)
data(fgl)
set.seed(17)
fgl.rf <- randomForest(type ~ ., data = fgl, mtry = 2, importance = TRUE, do.trace = 100)
## ntree      OOB      1      2      3      4      5      6
##   100: 20.09% 14.29% 18.42% 58.82% 23.08% 22.22% 13.79%
##   200: 21.03% 10.00% 22.37% 64.71% 23.08% 33.33% 13.79%
##   300: 18.69% 10.00% 18.42% 64.71% 23.08% 11.11% 13.79%
##   400: 19.16% 11.43% 17.11% 64.71% 23.08% 22.22% 13.79%
##   500: 19.63% 11.43% 19.74% 58.82% 23.08% 22.22% 13.79%
print(fgl.rf)
##
## Call:
##   randomForest(formula = type ~ ., data = fgl, mtry = 2,
## importance = TRUE,      do.trace = 100)
##           Type of random forest: classification
##                   Number of trees: 500
## No. of variables tried at each split: 2
##
##           OOB estimate of error rate: 19.63%
## Confusion matrix:
##             WinF WinNF Veh Con Tabl Head class.error
## WinF     62     6    2    0     0     0    0.1142857
## WinNF    11    61    1    1     1     1    0.1973684
## Veh      7     3    7    0     0     0    0.5882353
## Con      0     2    0   10     0     1    0.2307692
## Tabl     0     2    0    0     7     0    0.2222222
## Head     1     3    0    0     0    25    0.1379310

```

We can compare random forests with support vector machines by doing ten repetitions of 10-fold cross-validation, using the errorest functions in the ipred package. #errorest performs resampling based estimates of prediction error: misclassification error, root mean squared error or Brier score for survival data.

```

# if(!require(ipred)) install.packages("ipred")
library(ipred)
set.seed(131)
error.RF <- numeric(10)
for(i in 1:10) error.RF[i] <- errorest(type ~ ., data = fgl,
model = randomForest, mtry = 2)$error
summary(error.RF)

```

```

##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.
##  0.1822  0.1928  0.2009  0.2033  0.2173  0.2243
set.seed(563)
error.SVM <- numeric(10)
for (i in 1:10) error.SVM[i] <- errorest(type ~ ., data =
fgl, model = svm, cost = 10, gamma = 1.5)$error
summary(error.SVM)

##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.
##  0.3645  0.3785  0.3808  0.3794  0.3832  0.3879

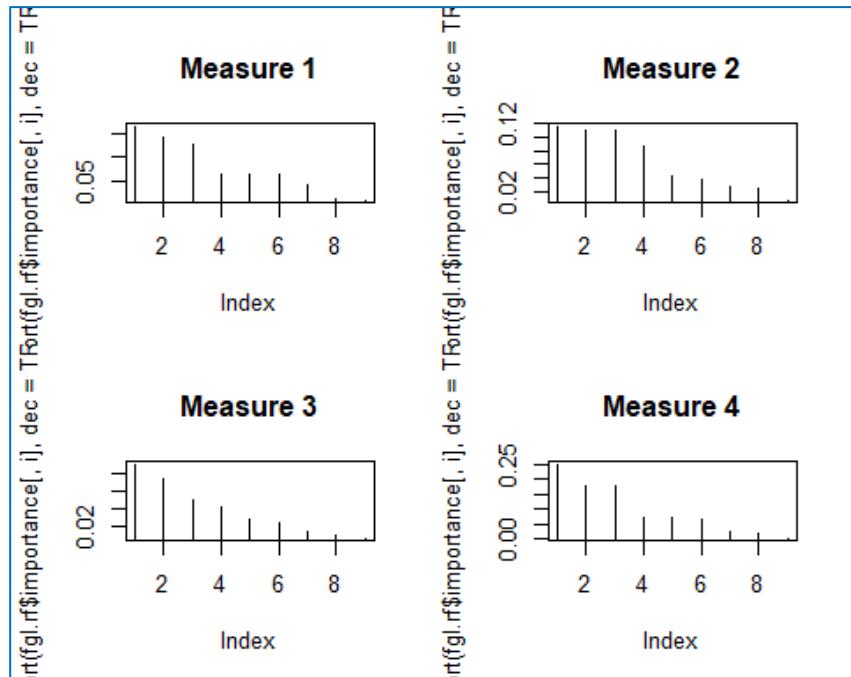
```

We see that the random forest compares quite favorably with SVM. We have found that the variable importance measures produced by random forests can sometimes be useful for model reduction

```

par(mfrow = c(2, 2))
for (i in 1:4) plot(sort(fgl.rf$importance[, i], dec = TRUE
), type = "h", main = paste("Measure", i))

```



## Exercises

- a. Random Forest in R with Low Birth Weight
  - a. Download The Low Birth Weight coman seaparte file CSV, “birthwt,” at  
<https://vincentarelbundock.github.io/Rdatasets/datasets.html>.
  - b. Describe the contexts of “birthwt.”
  - c. Use Random Forest to determine how well low brith rate (lwt) can be predicted by other factors.
  - d. Determine VIMP
  - e. Measure model fit
  - f. Plot the errors, e.g., MSE, R-SQ, and summarize them.
  - g. Use Markdown to render a report of your findings.
- b. Random Forest in R with IRIS Data
  - a. Split iris data to Training data and testing data
  - b. Generate Random Forest learning treee
  - c. Print Random Forest model and see the importance features
  - d. Plot Random Forest model and see the importance features
  - e. Determine variable importance.
  - f. Build random forest for testing data
  - g. Examine the margin, positive or negative, if positif it means correct classification
  - h. Try to tune Random Forest
  - i. Determine the OOB error
  - j. Use Markdown to render a report

**Markdown Note.** There are numerous things you can do in a code chunk: you can generate text output, tables, and graphics. This give you precise control over all these output via chunk options, which you can place inside curly braces (between ``{r and }). For example, suppose you want to hide the output of library(randomForest), which consist of warning that may be important to you but not so for your reader. You can choose to hide the text output using the chunk option `results = 'hide'` or set the figure height to 4 inches by `fig.height = 4`. Chunk options are separated by commas, like

```
```{r, chunk-label, results='hide', fig.height=4}
```

1. This script trains a Random Forest model based on the data, saves a sample submission, and plots the relative importance of the variables in making predictions

- a. Download 1\_random\_forest\_r\_submission.csv through  
<https://www.kaggle.com/c/titanic-gettingStarted/submissions/attach>
- b. Use rgw function ‘extraFeature’ to construct a random forest.

```
extractFeatures <- function(data) {  
  features <- c("Pclass",  
             "Age",  
             "Sex",  
             "Parch",  
             "SibSp",  
             "Fare",  
             "Embarked")  
  fea <- data[,features]  
  fea$Age[is.na(fea$Age)] <- -1  
  fea$Fare[is.na(fea$Fare)] <- median(fea$Fare, na.rm=TRUE)  
  fea$Embarked[fea$Embarked==""] = "S"  
  fea$Sex      <- as.factor(fea$Sex)  
  fea$Embarked <- as.factor(fea$Embarked)  
  return(fea)  
}
```

- c. Determine VIMP and plot them

## CHAPTER 3 – Cluster Models I

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). Each group is called a cluster and exploring the differences (and similarities when they overlap) is a focal point of cluster analysis, which is an essential task of exploratory data mining, and a common technique for statistical data analysis. Cluster analysis is used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Rather than a specific algorithm, cluster analysis is general task to be solved. It can be achieved by 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, clustering can be formulated 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 K. , 1994) (Tryon, 1939) and famously used by Cattell beginning in 1943 (Cattell, 1943) for trait theory classification in personality psychology.

### 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. This is also called 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 essentially 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. Clusterings 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.

## 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 possibly over 100 published clustering algorithms. Not all algorithms provide models for their clusters and can thus not easily be categorized.

There is no 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. It should be noted 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).

### *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 the connectivity models we have already mentioned. 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.

Strategies for hierarchical clustering generally fall into two types:

**Agglomerative** is a “bottom up” approach. Here each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

**Divisive** is a “top down” approach. Here all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Connectivity-based clustering is a whole family of methods that differ by the way distances are computed. 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) (“also known as average linkage clustering).

Furthermore, hierarchical clustering can be agglomerative or divisive.

### **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 are (The DISTANCE Procedure: Proximity Measures):

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	$\sqrt{(a - b)^T S^{-1} (a - b)}$ where $S$ is the Covariance matrix

For text or other non-numeric data, metrics such as the Hamming distance or Levenshtein distance are often used. 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

### **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):

Names	Formula
Maximum or complete linkage clustering	$\max\{d(a, b) : a \in A, b \in B\}$
Minimum or single-linkage clustering	$\min\{d(a, b) : a \in A, b \in B\}$
Mean or average linkage clustering, or UPGMA	$\frac{1}{ A  B } \sum_{a \in A} \sum_{b \in B} d(a, b)$
Centroid linkage clustering, or UPGMC	$\ c_s - c_t\ $ where $c_s$ and $c_t$ are the centroids of clusters $s$ and $t$ , respectively.

Names	Formula
Minimum energy clustering	$\frac{2}{nm} \sum_{i,j=1}^{n,m} \ a_i - b_j\ _2 - \frac{1}{n^2} \sum_{i,j=1}^n \ a_i - a_j\ _2$ $- \frac{1}{m^2} \sum_{i,j=1}^m \ b_i - b_j\ _2$

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 e. a., 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 e. a., Agglomerative clustering via maximum incremental path integral, 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.

## Hierarchical Clustering using R

### *First load the package.*

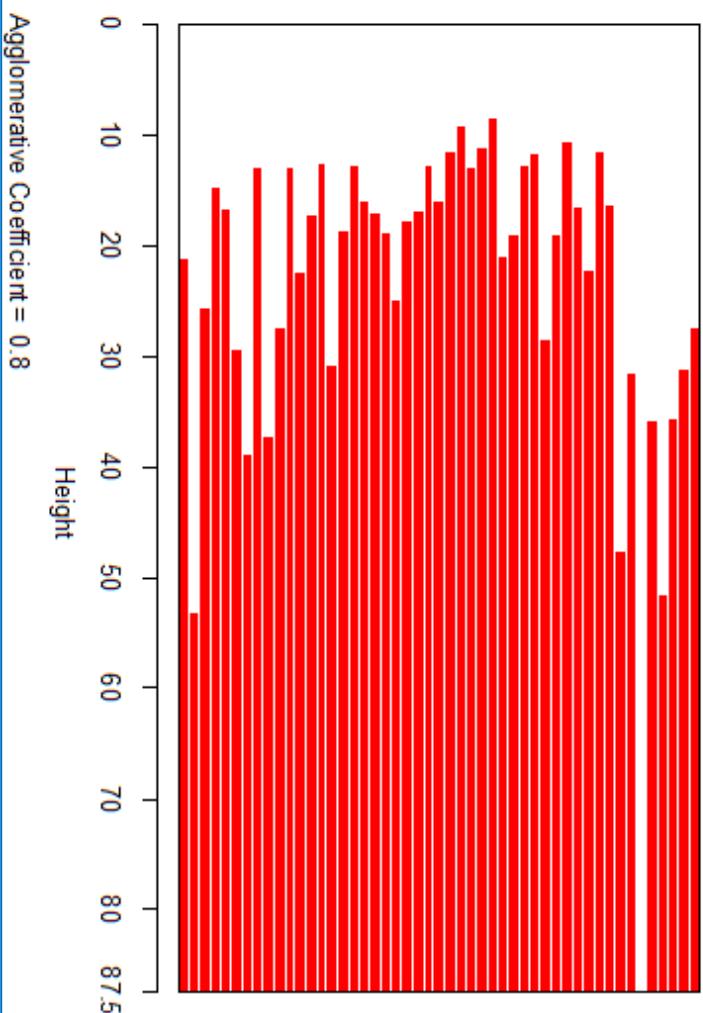
votes.repub is a data frame with the percent of votes given to the republican candidate in presidential elections from 1856 to 1976. Rows represent the 50 states, and columns the 31 elections.

### *Agglomerative Nesting (Hierarchical Clustering)*

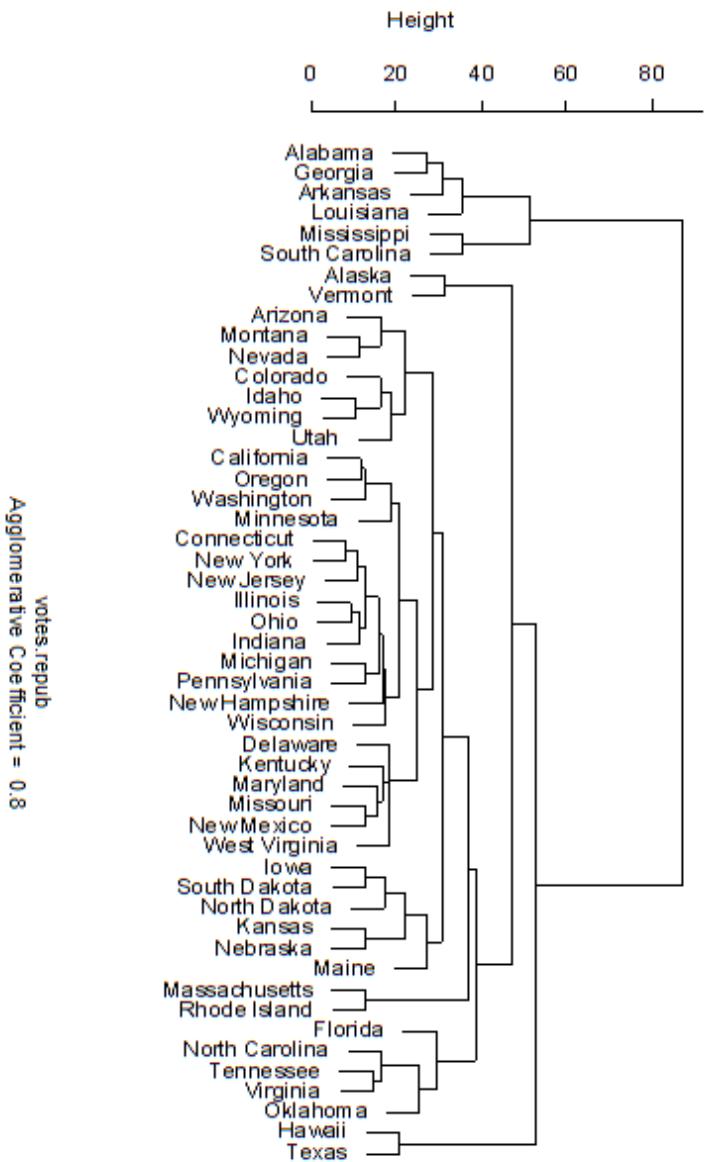
Here, we are using the agnes function computes agglomerative hierarchical clustering of the dataset.

```
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"
## "call"
## [7] "method"     "order.lab"  "data"
plot(agn1)
```

`Banner of agnes(x = votes.repub, metric = "manhattan", stand = TRUE)`

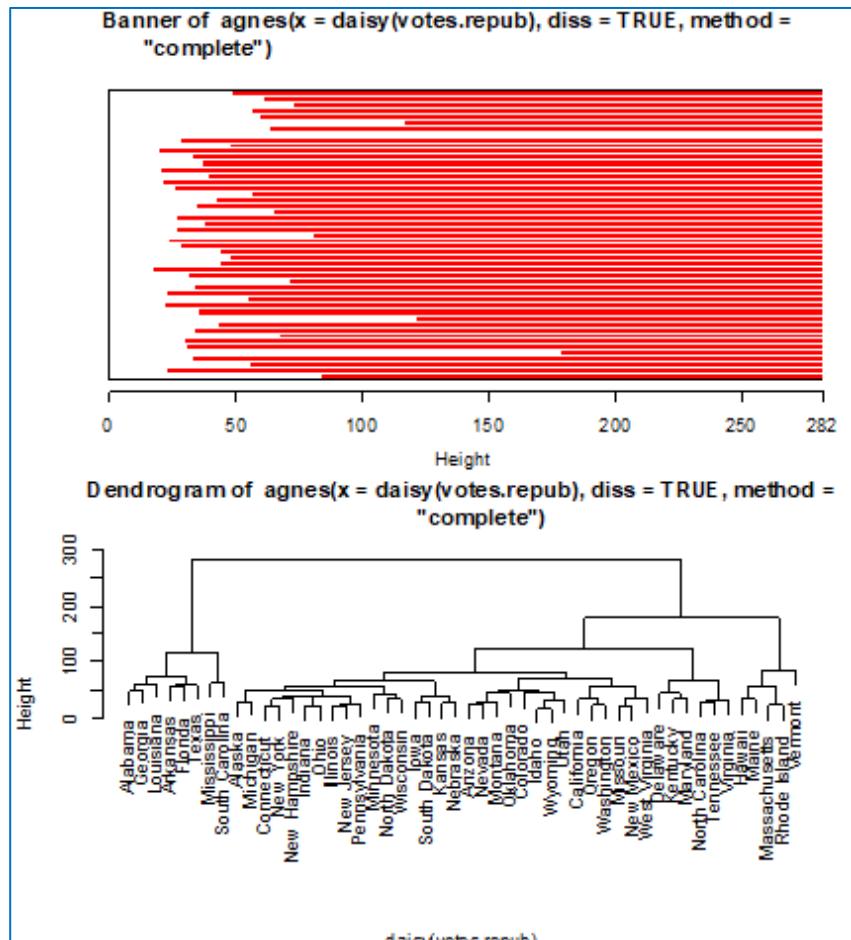


Dendrogram of agnes(x = votes.repub, metric = "manhattan", stand = TRUE)



## Plot the Data

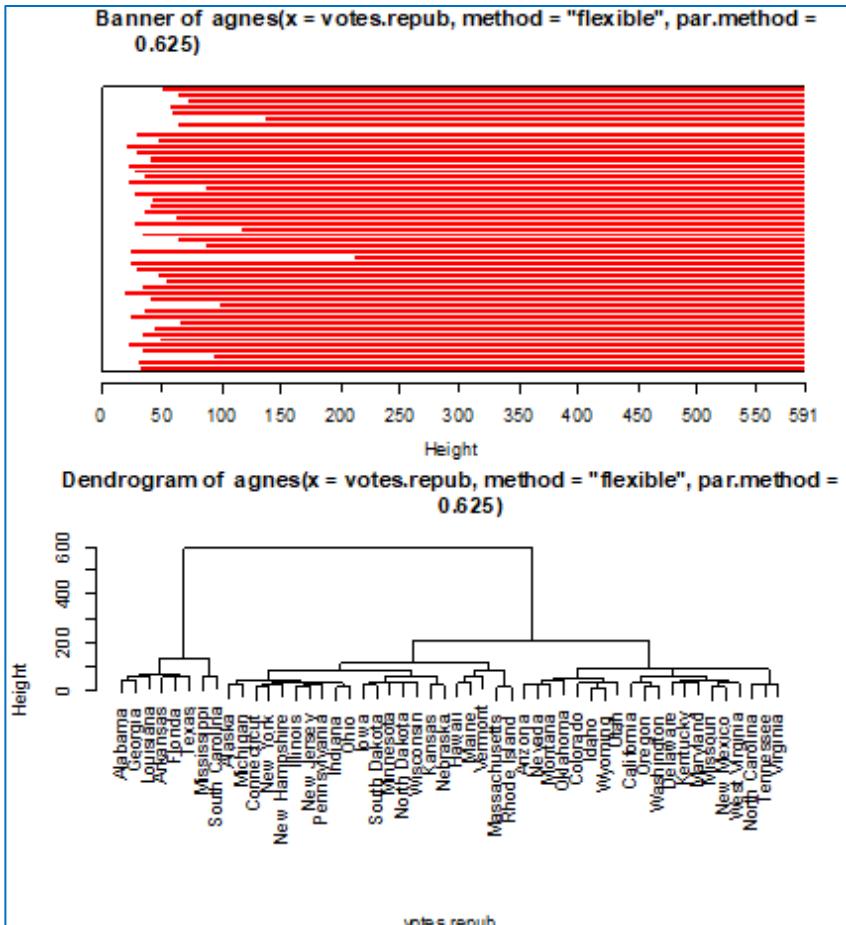
```
op <- par(mfrow=c(2,2))
agn2 <- agnes(daisy(votes.repub), diss = TRUE, method = "complete")
plot(agn2)
```



## Plot with Parameters

`alpha = 0.625 ==beta = -1/4` is “recommended” by some

```
agnS <- agnes(votes.repub, method = "flexible", par.meth = 0.625)
plot(agnS)
```



**“show” equivalence of three “flexible” special cases**

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]
## | | | |--[dendrogram w/ 2 branches and 2 members at h = 48.2]
## | | | | |--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 memb at h = 56.4]
##           |--[dendrogram w/ 2 bchs and 10 memb at h = 47.2]
##             |--[dendrogram w/ 2 bchs and 2 memb at h = 28.1]
##               |--leaf "Alaska"
##               `--leaf "Michigan"
##             `--[dendrogram w/ 2 bchs and 8 memb at h = 39.2]
##               |--[dendrog w/ 2 bchs and 5 memb at h = 36.8]
##                 |--[dend w/ 2 bchs and 3 memb at h = 32.9]
##                   |--[dend w/ 2 bch and 2 memb at h=19.4]
##                     |--leaf "Connecticut"
##                     |--leaf "New York"
##                     `--leaf "New Hampshire"
##                   `--[dend w/ 2 bchs and 2 memb at h = 20.2]
##                     |--leaf "Indiana"
##                     `--leaf "Ohio"
##                   `--[dendrog w/ 2 bchs and 3 memb at h = 25.3]
##                     |--[dend w/ 2 bchs and 2 memb at h = 20.9]
##                       |--leaf "Illinois"
##                         |--leaf "New Jersey"
##                         `--leaf "Pennsylvania"
##           `--[dendrogram w/ 2 bchs and 3 memb at h = 42.2]
##             |--leaf "Minnesota"
##             `--[dendrogram w/ 2 bchs and 2 memb at h = 33.7]
##               |--leaf "North Dakota"
##               `--leaf "Wisconsin"
##             `--[dendrogram w/ 2 branches and 4 memb at h = 37.5]
##               |--[dendrogram w/ 2 bchs and 2 memb at h = 26.2]
##                 |--leaf "Iowa"
##                 `--leaf "South Dakota"
##               `-[dendrogram w/ 2 bchs and 2 memb 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 bchs and 4 memb at h = 43.4]
##                   |--[dendrogram w/ 2 bchs and 3 memb at h = 27.8]
##                     |--[dendrog w/ 2 bchs and 2 memb at h = 23.4]
##                       |--leaf "Arizona"
##                         |--leaf "Nevada"
##                           `--leaf "Montana"
##                         `--leaf "Oklahoma"
##                     `-[dendrogram w/ 2 bchs and 4 memb at h = 43.7]
##                       |--leaf "Colorado"
##                         `-[dendrogram w/ 2 bchs and 3 memb at h = 31.2]
##                           |--[dendrog w/ 2 bchs and 2 memb at h = 17.2]
##                             |--leaf "Idaho"

```

```

##          | `--leaf "Wyoming"
##          | `--leaf "Utah"
##          '--[dendrogram w/ 2 branches and 6 memb at h = 54.3]
##              |--[dendrogram w/ 2 bchs and 3 memb at h = 33.2]
##                  |--leaf "California"
##                  '--[dendrogram w/ 2 bchs and 2 memb at h = 22.2]
##                      |--leaf "Oregon"
##                      '--leaf "Washington"
##              '--[dendrogram w/ 2 bchs and 3 members at h = 35.1]
##                  |--[dendrogram w/ 2 bchs and 2 memb 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 memb 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 memb 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"

```

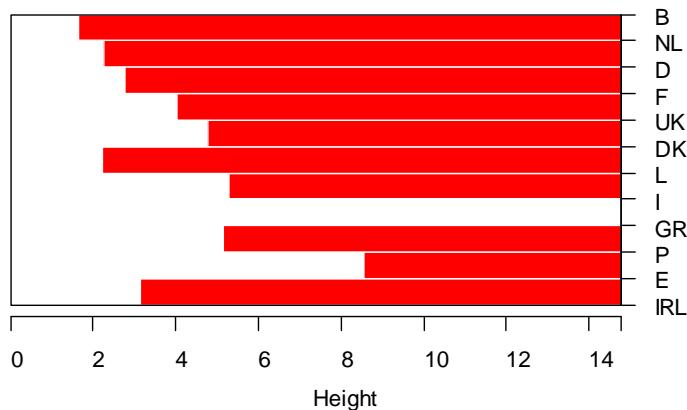
### Multiple Plots

```

plot(agnes(agriculture), ask = TRUE)
data(animals)
aa.a <- agnes(animals) # default method = "average"
aa.ga <- agnes(animals, method = "gaverage")
op <- par(mfcol=1:2, mgp=c(1.5, 0.6, 0), mar=c(.1+ c(4,3,2
,1)),cex.main=0.8)
plot(aa.a, which.plot = 2)

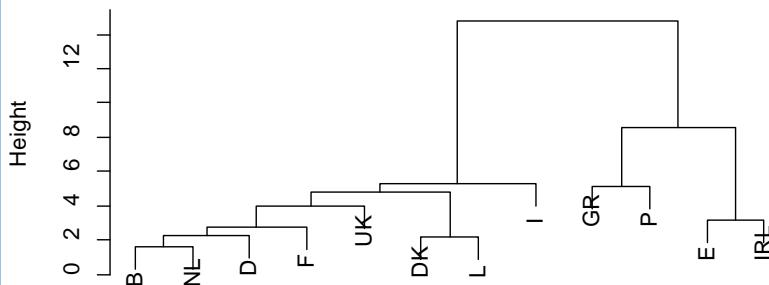
```

**Banner of agnes(x = agriculture)**



Agglomerative Coefficient = 0.78

**Dendrogram of agnes(x = agri)**



agriculture  
Agglomerative Coefficient = 0.78

## k-Means Clustering

The wine dataset contains the results of a chemical analysis of wines grown in a specific area of Italy. Three types of wine are represented in the 178 samples, with the results of 13 chemical analyses recorded for each sample. The Type variable has been transformed into a categoric variable.

## Load and View the Data

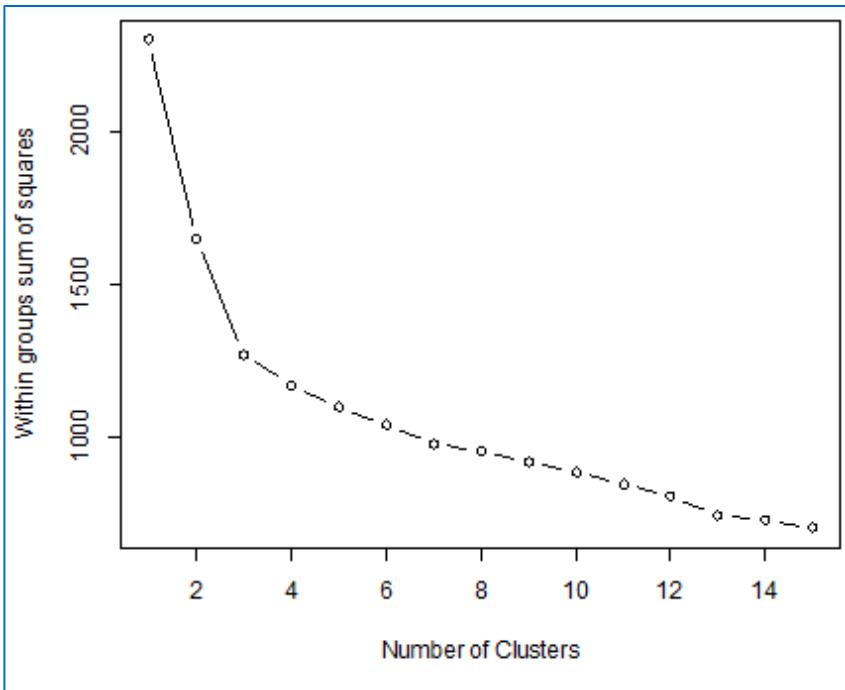
```
library(rattle)
## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

library(rattle.data)
data(wine)
head(wine)

##   Type Alcohol Malic Acid Alkalinity Magnesium Phenols Flavanoids
## 1    1    14.23  1.71  2.43      15.6     127    2.80     3.06
## 2    1    13.20  1.78  2.14      11.2     100    2.65     2.76
## 3    1    13.16  2.36  2.67      18.6     101    2.80     3.24
## 4    1    14.37  1.95  2.50      16.8     113    3.85     3.49
## 5    1    13.24  2.59  2.87      21.0     118    2.80     2.69
## 6    1    14.20  1.76  2.45      15.2     112    3.27     3.39
##   Nonflavanoids Proanthocyanins Color Hue Dilution Proline
## 1          0.28           2.29  5.64 1.04     3.92    1065
## 2          0.26           1.28  4.38 1.05     3.40    1050
## 3          0.30           2.81  5.68 1.03     3.17    1185
## 4          0.24           2.18  7.80 0.86     3.45    1480
## 5          0.39           1.82  4.32 1.04     2.93     735
## 6          0.34           1.97  6.75 1.05     2.85    1450
```

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 doesn't 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. The number of clusters is chosen at this point, hence the "elbow criterion".

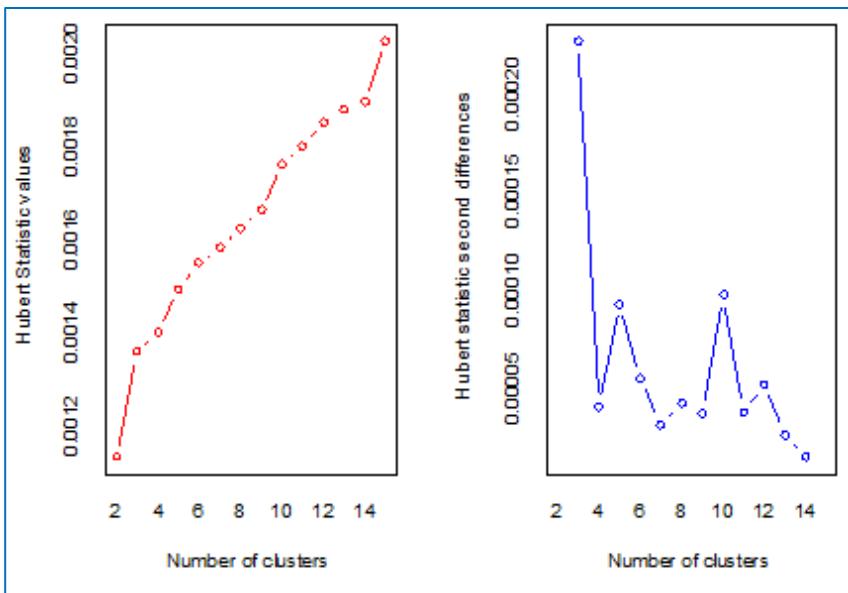
```
df <- scale(wine[-1])
wssplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data, 2, var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")}
wssplot(df)
```



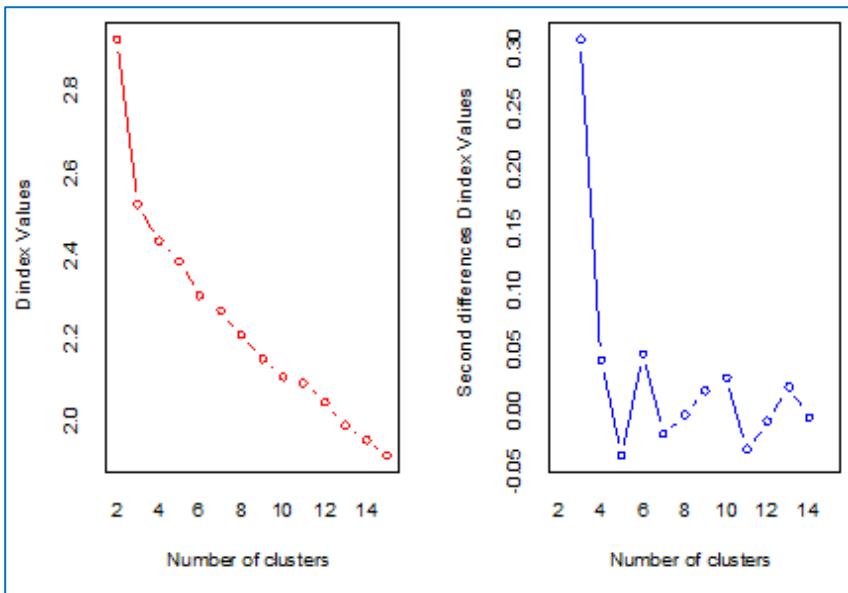
### *NbClust Package*

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")
```



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.

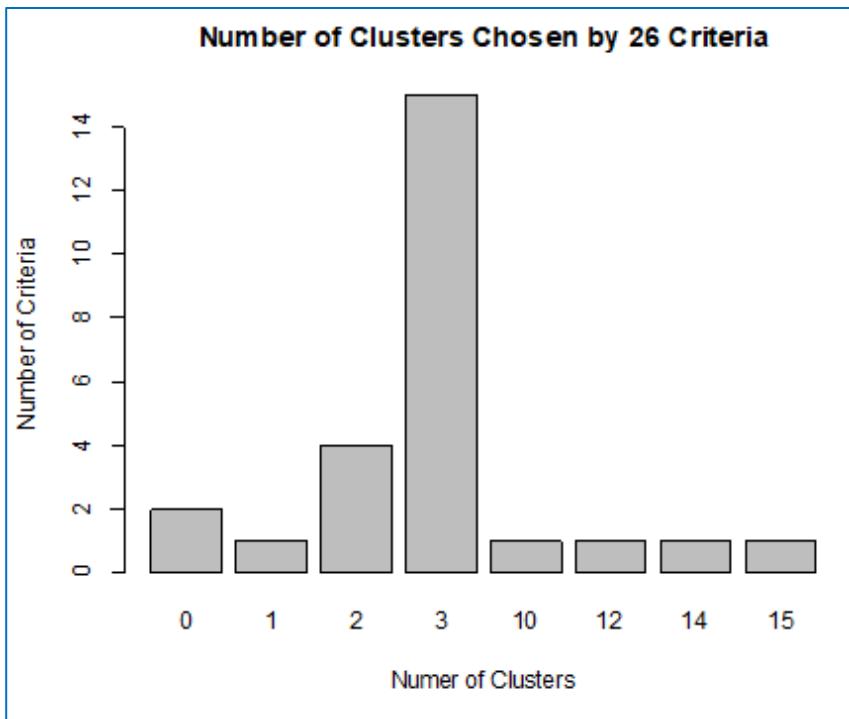


The D index is a graphical method of determining the number of clusters. In the plot of D index, we seek a significant knee (the significant peak in D index second differences plot) that corresponds to a significant increase of the value of the measure.

```
##  
## ****  
## * Among all indices:  
## * 4 proposed 2 as the best number of clusters  
## * 15 proposed 3 as the best number of clusters  
## * 1 proposed 10 as the best number of clusters  
## * 1 proposed 12 as the best number of clusters  
## * 1 proposed 14 as the best number of clusters  
## * 1 proposed 15 as the best number of clusters  
##  
##           ***** Conclusion *****  
##  
## * According to the majority rule, the best number of clusters is 3  
##  
##  
## ****
```

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.

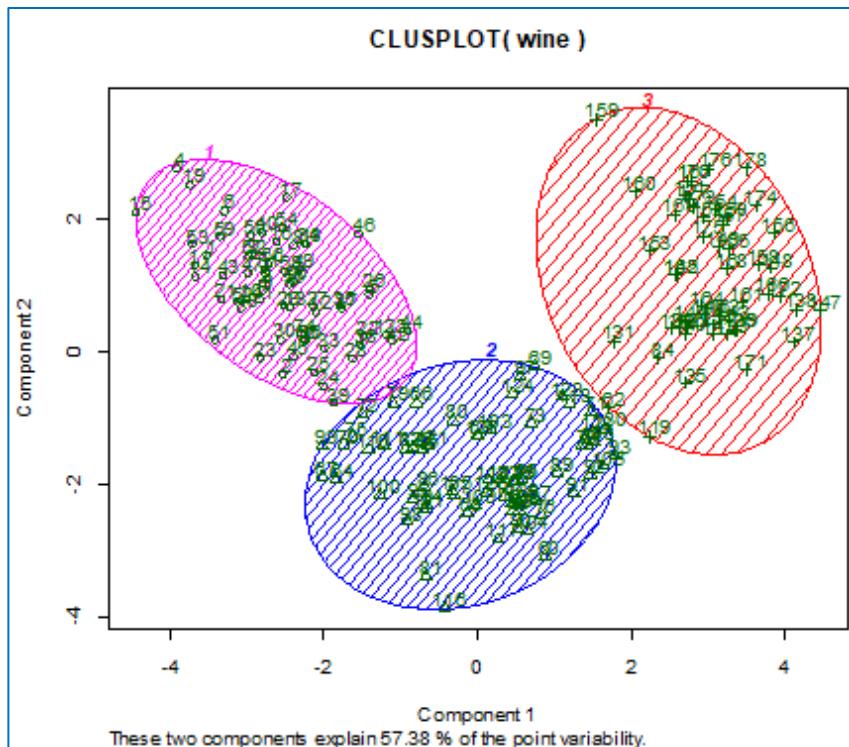
```
table(nc$Best.n[1,])  
##  
##  0  1  2  3 10 12 14 15  
##  2  1  4 15  1  1  1  1  
barplot(table(nc$Best.n[1,]), xlab="Numer of Clusters", yl  
ab="Number of Criteria", main="Number of Clusters Chosen b  
y 26 Criteria")
```



```

library(cluster)
set.seed(1234)
fit.km <- kmeans(df, 3, nstart=25)
fit.km$size
## [1] 62 65 51
fit.km$centers
##      Alcohol      Malic      Ash Alcalinity Magnesium   Phenols
## 1 0.8328826 -0.3029551 0.3636801 -0.6084749 0.57596208 0.8827472
## 2 -0.9234669 -0.3929331 -0.4931257 0.1701220 -0.49032869 -0.0757689
## 3 0.1644436  0.8690954 0.1863726 0.5228924 -0.07526047 -0.9765754
##      Flavanoids Nonflavanoids Proanthocyanins      Color      Hue
## 1 0.97506900 -0.56050853 0.57865427 0.1705823 0.4726504
## 2 0.02075402 -0.03343924 0.05810161 -0.8993770 0.4605046
## 3 -1.21182921 0.72402116 -0.77751312 0.9388902 -1.1615122
##      Dilution     Proline
## 1 0.7770551 1.1220202
## 2 0.2700025 -0.7517257
## 3 -1.2887761 -0.4059428
clusplot(wine, fit.km$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)

```



Hierarchical methods use a distance matrix as an input for the clustering algorithm. The choice of 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.

```
d <- dist(wine, method = "euclidean") # Euclidean distance matrix.
```

We use the Euclidean distance as an input for the clustering algorithm (Ward's minimum variance criterion minimizes the total within-cluster variance):

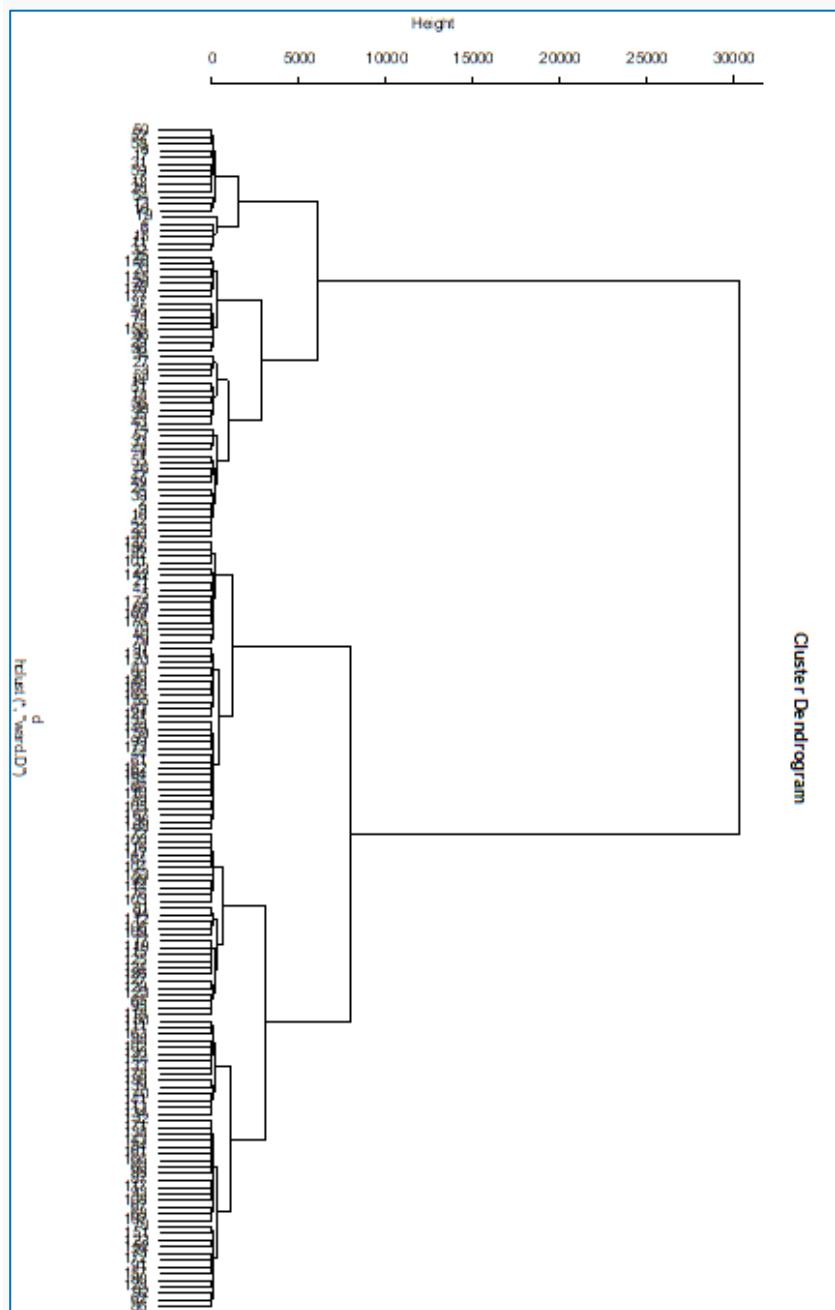
```
H.fit <- hclust(d, method="ward.D")
H2.fit<- hclust(d, method="ward.D2")
```

The clustering output can be displayed in a dendrogram.

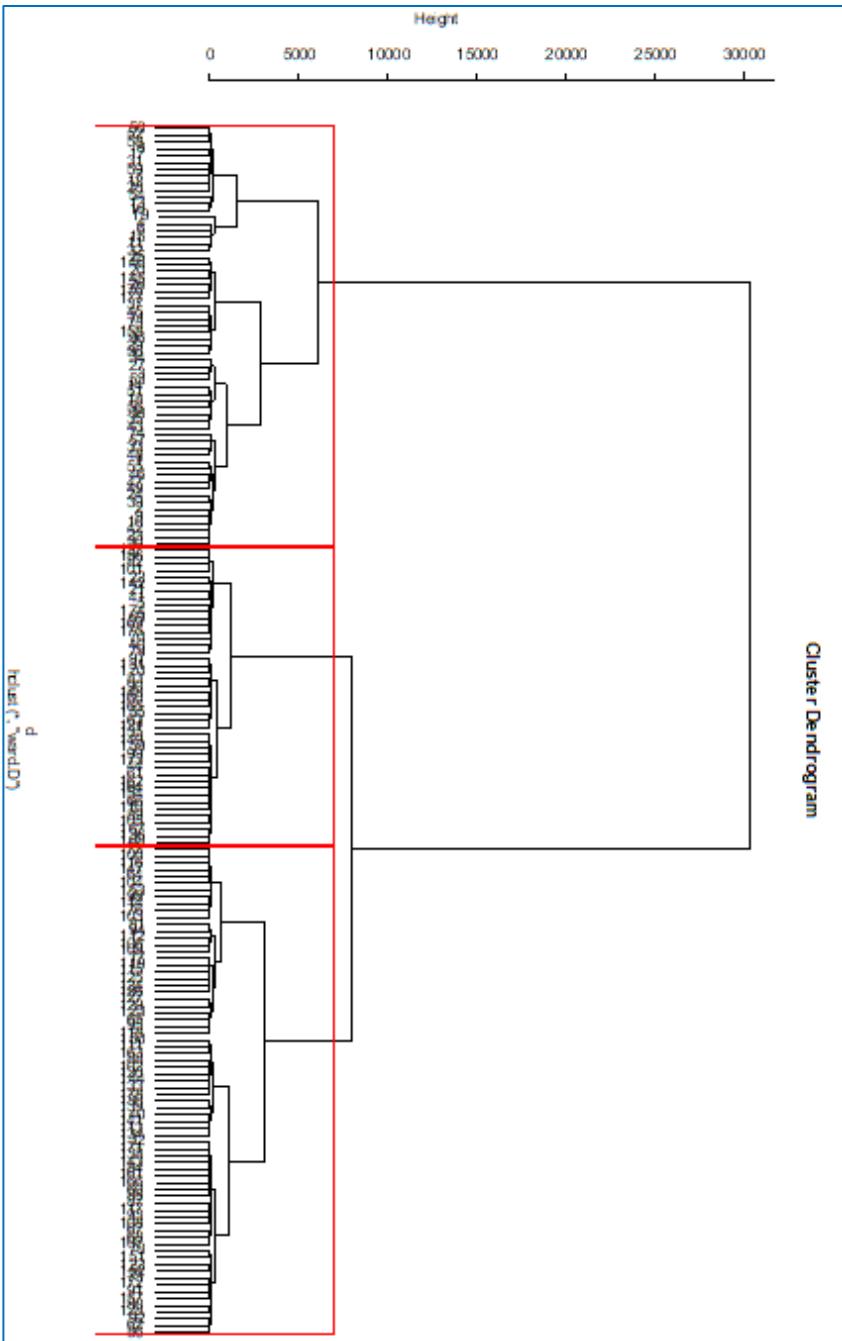
```
plot(H.fit)
```

First, we draw dendrogram with red borders around the 5 clusters.

```
plot(H.fit)
groups <- cutree(H.fit, k=3) # cut tree into 5 clusters
rect.hclust(H.fit, k=3, border="red")
```

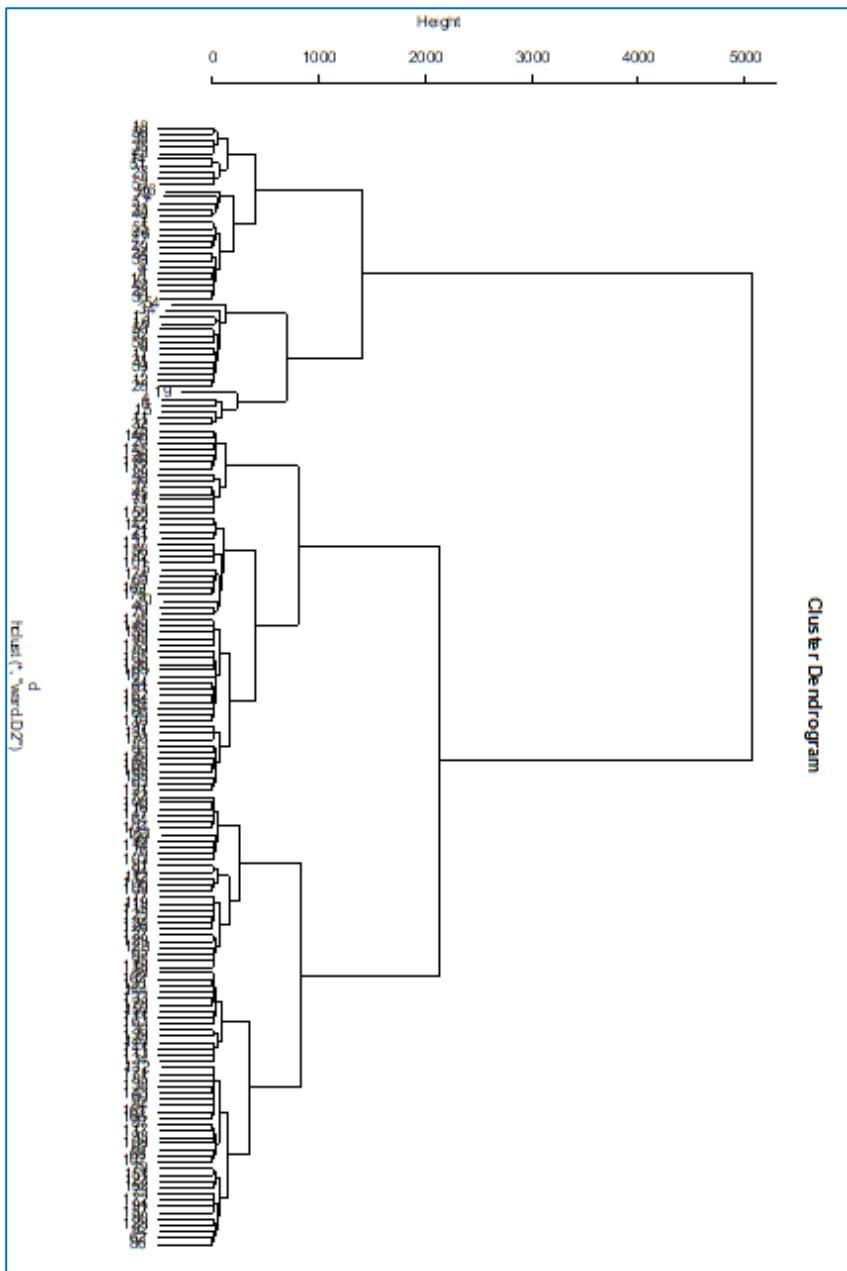


Cluster Dendrogram

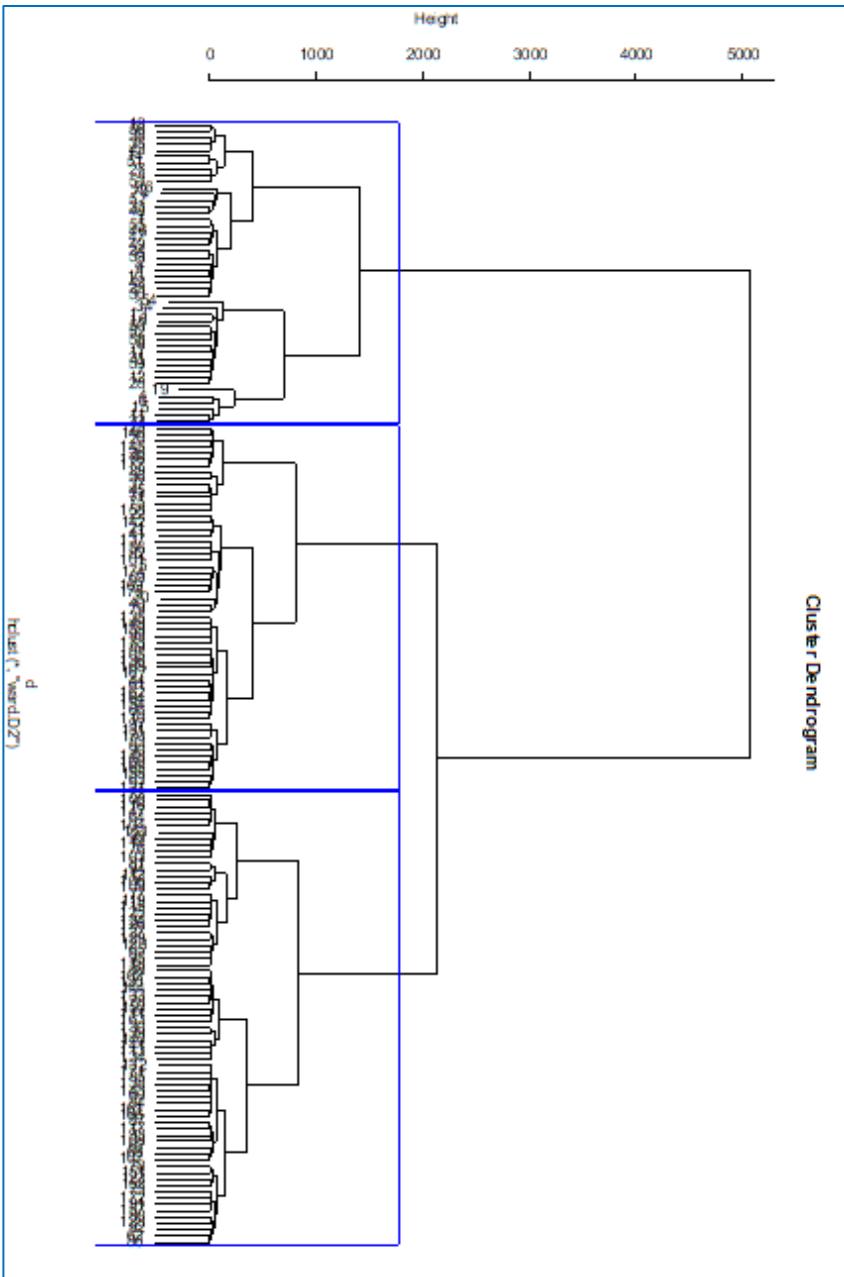


We now draw dendrogram with blue borders around the 5 clusters.

```
plot(H.fit)
rect.hclust(H2.fit, k=3, border="blue")
```



Cluster Dendrogram



```
groups <- cutree(H2.fit, k=3) # cut tree into 5 clusters
```

## Social Network Clustering Analysis

For this analysis, we will be using a dataset representing a random sample of 30.000 U.S. high school students who had profiles on a well-known Social Network in from 2006 to 2009. From the top 500 words appearing across all pages, 36 words were chosen to represent five categories of interests, namely extracurricular activities, fashion, religion, romance, and antisocial behavior. The 36 words include terms such as football, sexy, kissed, bible, shopping, death, and drugs. The final dataset indicates, for each person, how many times each word appeared in the person's SNS profile.

```
social<-read.csv(file="C:/Users/Jeff/Documents/VIT_University/data/social_net.csv",header = T)
head(social,3)

##   gradyear gender    age friends basketball football soccer softball
## 1      2006      M 18.982       7        0        0        0        0
## 2      2006      F 18.801       0        0        1        0        0
## 3      2006      M 18.335      69        0        1        0        0
##   volleyball swimming cheerleading baseball tennis sports cute sex sexy
## 1          0         0           0        0        0        0        0        0        0
## 2          0         0           0        0        0        0        1        0        0
## 3          0         0           0        0        0        0        0        0        0
##   hot kissed dance band marching music rock god church jesus bible hair
## 1     0     0     1     0       0     0     0     0     0     0     0     0
## 2     0     0     0     0       0     2     2     1     0     0     0     6
## 3     0     0     0     2       0     1     0     0     0     0     0     0
##   dress blonde mall shopping clothes hollister abercromb die death drunk
## 1     0     0     0       0     0       0     0     0     0     0     0
## 2     4     0     1       0     0       0     0     0     0     0     0
## 3     0     0     0       0     0       0     0     0     0     1     0
##   drugs
## 1     0
## 2     0
## 3     0
dim(social)
## [1] 10599    40
```

Let's also take a quick look at the specifics of the data. The first several lines of the str() output are as follows:

```
str(social)
## 'data.frame': 10599 obs. of 40 variables:
## $ gradyear : int 2006 2006 2006 2006 2006 2006 200
```

```

6 2006 2006 2006 ...
## $ gender      : Factor w/ 2 levels "F","M": 2 1 2 1 NA
1 1 2 1 1 ...
## $ age         : num  19 18.8 18.3 18.9 19 ...
## $ friends     : int  7 0 69 0 10 142 72 17 52 39 ...
## $ basketball  : int  0 0 0 0 0 0 0 0 0 0 ...
## $ football    : int  0 1 1 0 0 0 0 0 0 0 ...
## $ soccer      : int  0 0 0 0 0 0 0 0 0 0 ...
## $ softball    : int  0 0 0 0 0 0 0 1 0 0 ...
## $ volleyball   : int  0 0 0 0 0 0 0 0 0 0 ...
## $ swimming    : int  0 0 0 0 0 0 0 0 0 0 ...
## $ cheerleading: int  0 0 0 0 0 0 0 0 0 0 ...
## $ baseball    : int  0 0 0 0 0 0 0 0 0 0 ...
## $ tennis       : int  0 0 0 0 0 0 0 0 0 0 ...
## $ sports       : int  0 0 0 0 0 0 0 0 0 0 ...
## $ cute         : int  0 1 0 1 0 0 0 0 0 1 ...
## $ sex          : int  0 0 0 0 1 1 0 2 0 0 ...
## $ sexy         : int  0 0 0 0 0 0 0 1 0 0 ...
## $ hot          : int  0 0 0 0 0 0 0 0 0 1 ...
## $ kissed        : int  0 0 0 0 5 0 0 0 0 0 ...
## $ dance        : int  1 0 0 0 1 0 0 0 0 0 ...
## $ band         : int  0 0 2 0 1 0 1 0 0 0 ...
## $ marching     : int  0 0 0 0 0 1 1 0 0 0 ...
## $ music        : int  0 2 1 0 3 2 0 1 0 1 ...
## $ rock         : int  0 2 0 1 0 0 0 1 0 1 ...
## $ god          : int  0 1 0 0 1 0 0 0 0 6 ...
## $ church       : int  0 0 0 0 0 0 0 0 0 0 ...
## $ jesus        : int  0 0 0 0 0 0 0 0 0 2 ...
## $ bible        : int  0 0 0 0 0 0 0 0 0 0 ...
## $ hair          : int  0 6 0 0 1 0 0 0 0 1 ...
## $ dress         : int  0 4 0 0 0 1 0 0 0 0 ...
## $ blonde        : int  0 0 0 0 0 0 0 0 0 0 ...
## $ mall          : int  0 1 0 0 0 0 2 0 0 0 ...
## $ shopping      : int  0 0 0 0 2 1 0 0 0 1 ...
## $ clothes       : int  0 0 0 0 0 0 0 0 0 0 ...
## $ hollister     : int  0 0 0 0 0 0 2 0 0 0 ...
## $ abercrombie  : int  0 0 0 0 0 0 0 0 0 0 ...
## $ die           : int  0 0 0 0 0 0 0 0 0 0 ...
## $ death         : int  0 0 1 0 0 0 0 0 0 0 ...
## $ drunk         : int  0 0 0 0 1 1 0 0 0 0 ...
## $ drugs         : int  0 0 0 0 1 0 0 0 0 0 ...

```

As we had expected, the data include 30,000 teenagers with four variables indicating personal characteristics and 36 words indicating interests. Note that there are some NA's in the variable gender.

```
summary(social$age)
##      Min. 1st Qu. Median    Mean 3rd Qu.    Max.    NA's
## 4.928   18.010  18.453  18.907  18.867 106.927 1690
```

We will skip all the data with missing values:

```
social = na.omit(social)
dim(social)
## [1] 8586   40
```

We'll start our cluster analysis by considering only the 36 features that represent the number of times various interests appeared on the SNS profiles of teens. For convenience, let's make a data frame containing only these features:

```
interests <- social[5:40]
```

To apply z-score standardization to the interests data frame, we can use the scale() function with lapply(), as follows:

```
interests_z <- as.data.frame(lapply(interests, scale))
```

To divide teens into five clusters, we can use the following command:

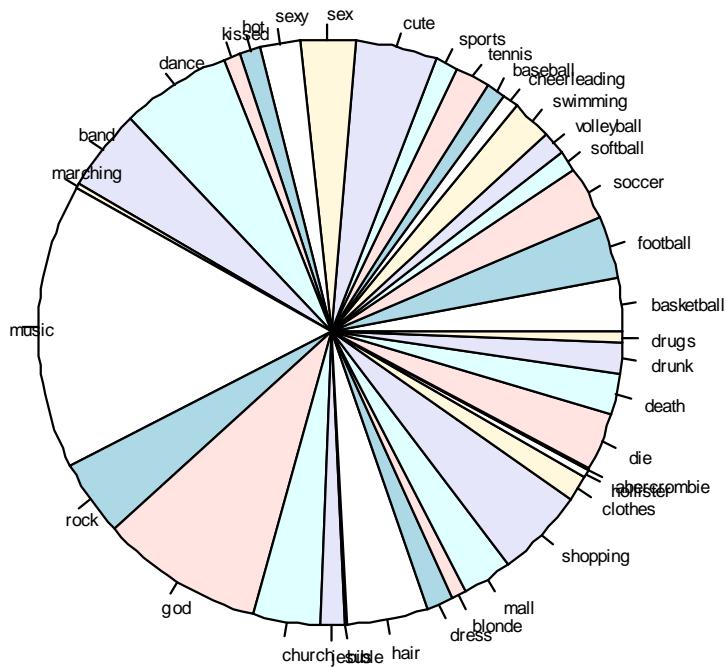
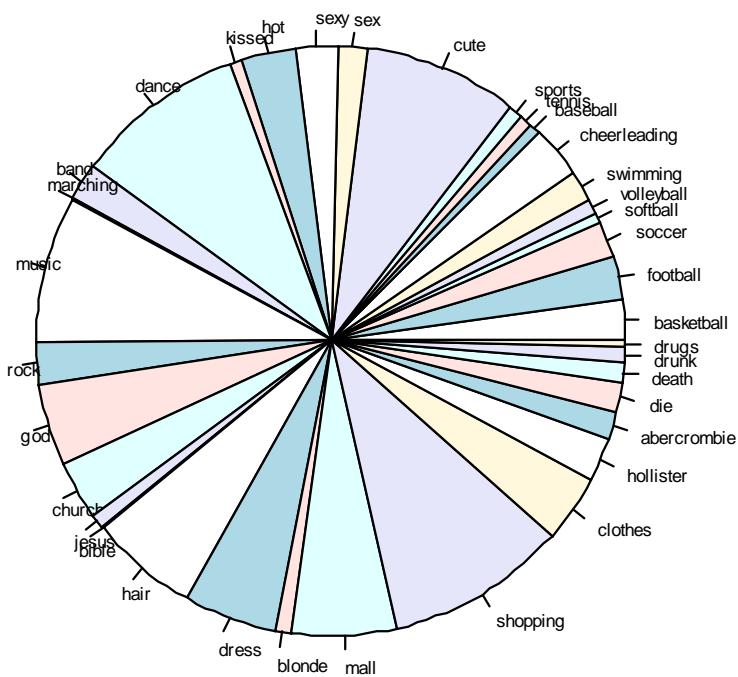
```
social_clusters <- kmeans(interests_z, 5)
```

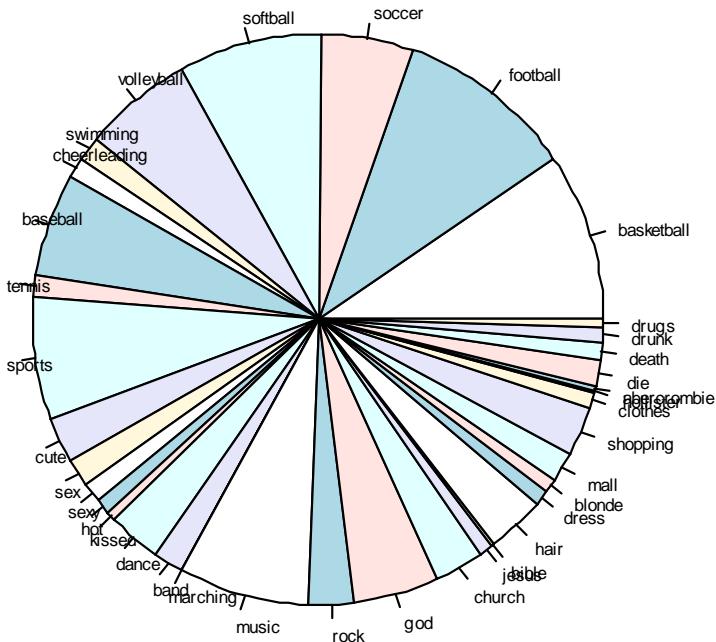
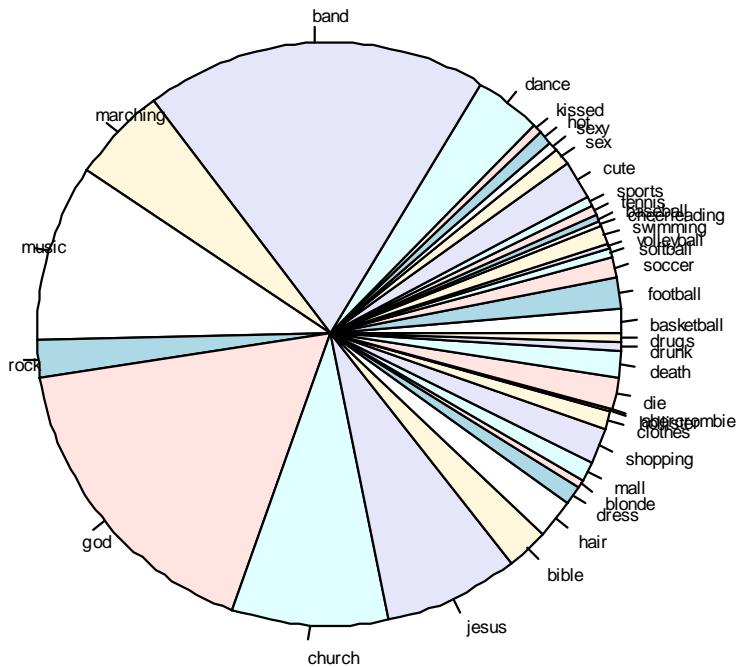
The number of examples falling in each of the groups is provided below. If the groups are too large or too small, then they are not likely to be very useful. To obtain the size of the kmeans() clusters, use the teen\_clusters\$size component as follows:

```
social_clusters$size
## [1] 251 6320 1578 176 261
```

The cluster characterization can be obtained with pie charts:

```
par(mfrow=c(2,2))
pie(colSums(interests[social_clusters$cluster==1,]),cex=0.5)
pie(colSums(interests[social_clusters$cluster==2,]),cex=0.5)
pie(colSums(interests[social_clusters$cluster==3,]),cex=0.5)
pie(colSums(interests[social_clusters$cluster==4,]),cex=0.5)
```





## *Customer Segmentation*

Customer segmentation is as simple as it sounds: grouping customers by their characteristics - and why would you want to do that? To better serve their needs! Our example is to do with e-mail marketing.

```
offers<-read.table("C:/Users/Jeff/Documents/VIT_University  
/data/offers.csv", sep = ',', header=T)  
head(offers)  
##   OfferID Campaign Varietal MinimumQt Discount Origin  
## 1      1 January  Malbec       72      56 France  
## 2      2 January Pinot Noir     72      17 France  
## 3      3 February Espumante    144       2 Oregon  
## 4      4 February Champagne    72       8 France  
## 5      5 February Cabernet    144      4 New Zealand  
                                Sauvignon  
## 6      6 March     Prosecco    144      86 Chile  
##   PastPeak  
## 1 FALSE  
## 2 FALSE  
## 3 TRUE  
## 4 TRUE  
## 5 TRUE  
## 6 FALSE  
transactions<-read.table("C:/Users/Jeff/Documents/VIT_University  
/data/transactions.csv", sep = ',', header=T)  
head(transactions)  
##   CustomerLastName OfferID  
## 1           Smith        2  
## 2           Smith       24  
## 3        Johnson       17  
## 4        Johnson       24  
## 5        Johnson       26  
## 6      Williams       18
```

### *Step 1: Organizing the information*

We have two data sets: one for the offers and the other for the transactions. First what we need to do is create a transaction matrix. That means, we need to put the offers we mailed out next to the transaction history of each customer. This is easily achieved with a pivot table.

```

library(reshape)
pivot<-melt(transactions[1:2])#Using CustomerLastName as id variables
## Using CustomerLastName as id variables
pivot<-(cast(pivot,value~CustomerLastName,fill=0,fun.aggregate=function(x) length(x)))
pivot<-cbind(offers,pivot[-1])
# write.csv(file="pivot.csv",pivot) # to save your data
cluster.data<-pivot[,8:length(pivot)]
cluster.data<-t(cluster.data)
head(cluster.data)

##          1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 1
9 20 21 22 23 24 25
## Adams      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
## Allen      0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
## Anderson   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0
## Bailey     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
## Baker      0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
## Barnes     0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0
##          26 27 28 29 30 31 32
## Adams      0 0 0 1 1 0 0
## Allen      0 1 0 0 0 0 0
## Anderson   1 0 0 0 0 0 0
## Bailey     0 0 0 0 1 0 0
## Baker      0 0 0 0 0 1 0
## Barnes     0 0 0 0 0 1 0

```

In the clustering data set, rows represent customers and columns are different wine brands/types.

## Step 2: Distances and Clusters

We will use k=4 indicating that we will use 4 clusters. This is somewhat arbitrary, but the number you pick should be representative of the number of segments you can handle as a business. So, 100 segments does not make sense for an e-mail marketing campaign. We need to calculate how far away each customer is from the cluster's mean. To do

this we could use many distances/dissimilarity indices, one of which is the Gower dissimilarity.

```
library(cluster)
D=daisy(cluster.data, metric='gower')
## Warning in daisy(cluster.data, metric = "gower"): binary variable(s) 1, 2,
## 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
## , 19, 20, 21, 22,
## 23, 24, 25, 26, 27, 28, 29, 30, 31, 32 treated as interval scaled
```

After the creation of a distance matrix, we implement a Ward's hierarchical clustering procedure. That is, we will use an agglomeration method known as `ward.D2`, created by Joe H. Ward, Jr. As we continue through the discussion of hierarchical clustering techniques, we will use a number of different agglomeration methods, including "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC).

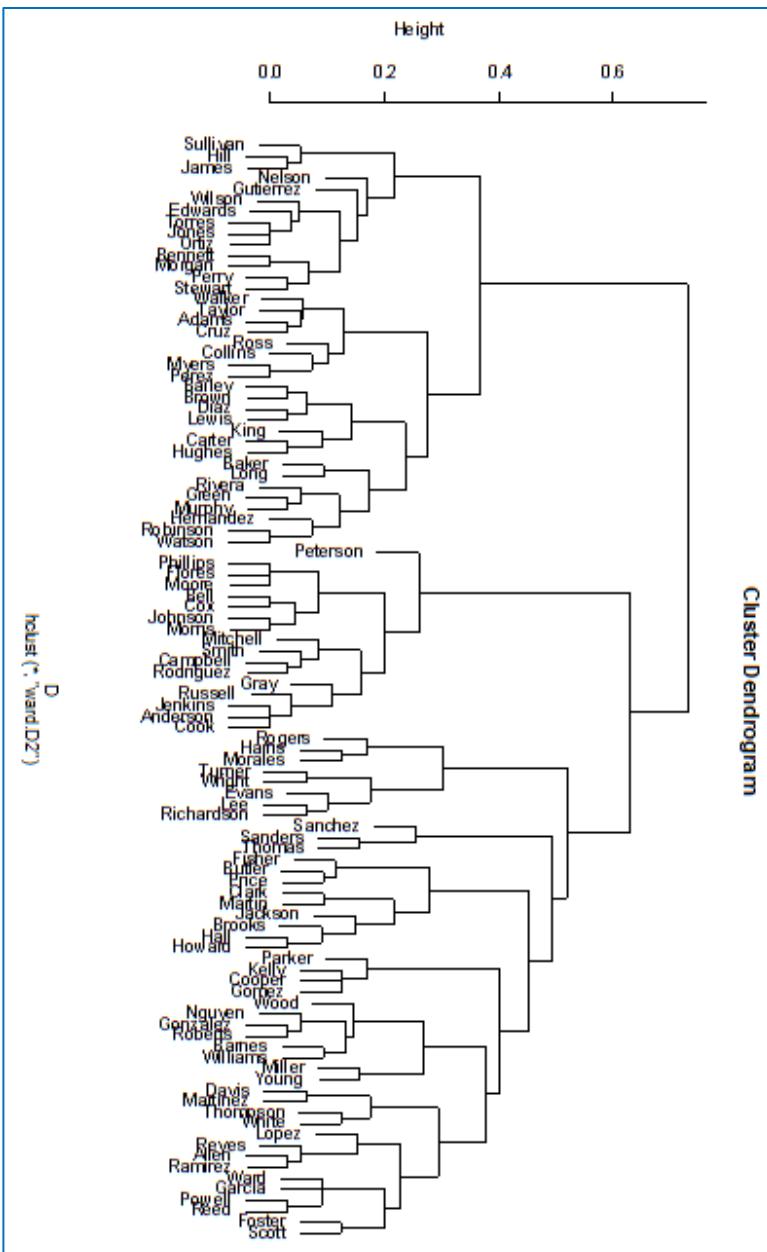
```
H.fit <- hclust(D, method="ward.D2")
plot(H.fit) # display dendrogram
# dendrogram appears on page 149
```

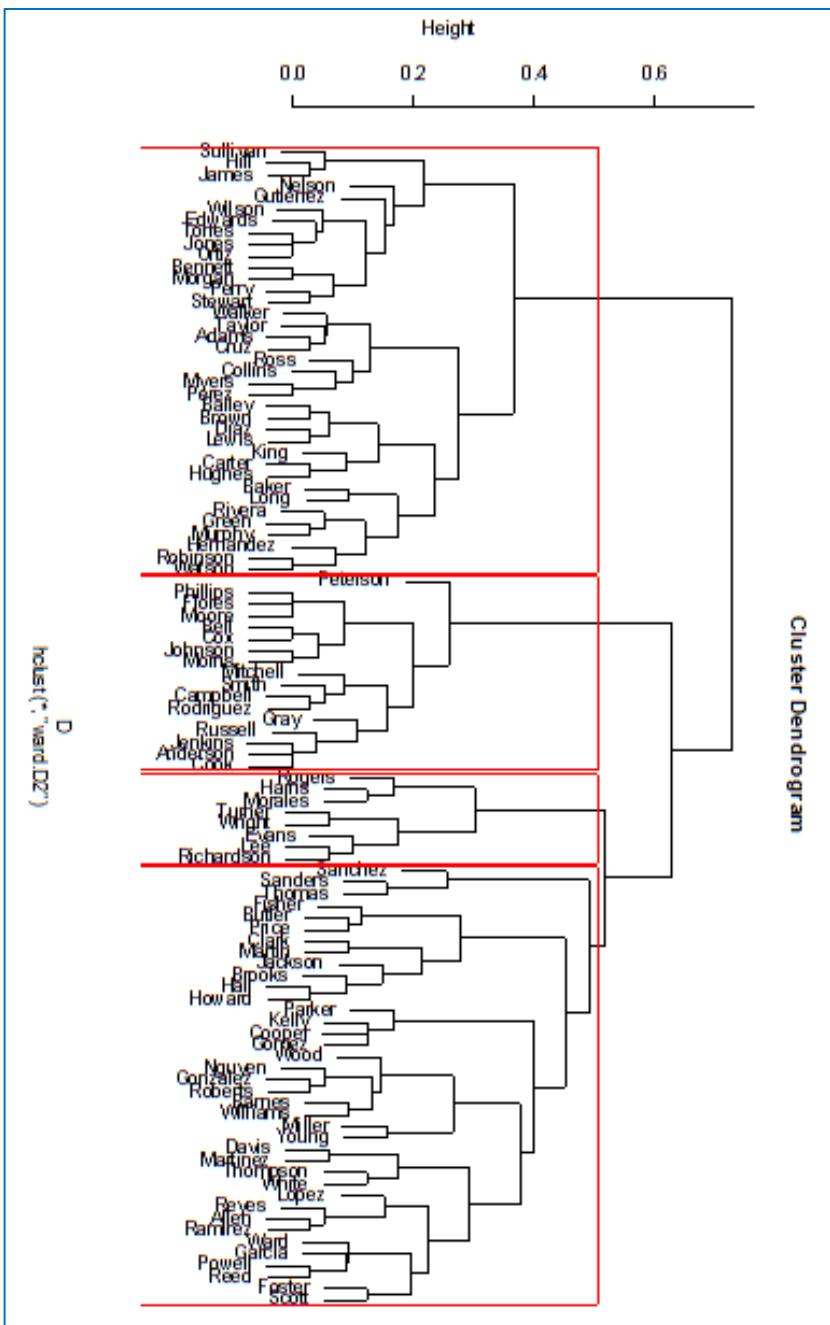
Now that we have a tree to visualize the clusters, it is obvious that pruning could produce a more feasible result. Visually, it looks like there are least four or five clusters and at most seven. Let's start with four clusters and see where that leads us. Using `cutree` and setting  $k = 4$ , our R code is as follows:

```
groups <- cutree(H.fit, k=4) # cut tree into 4 clusters
```

Now we draw dendrogram with red borders around the 4 clusters

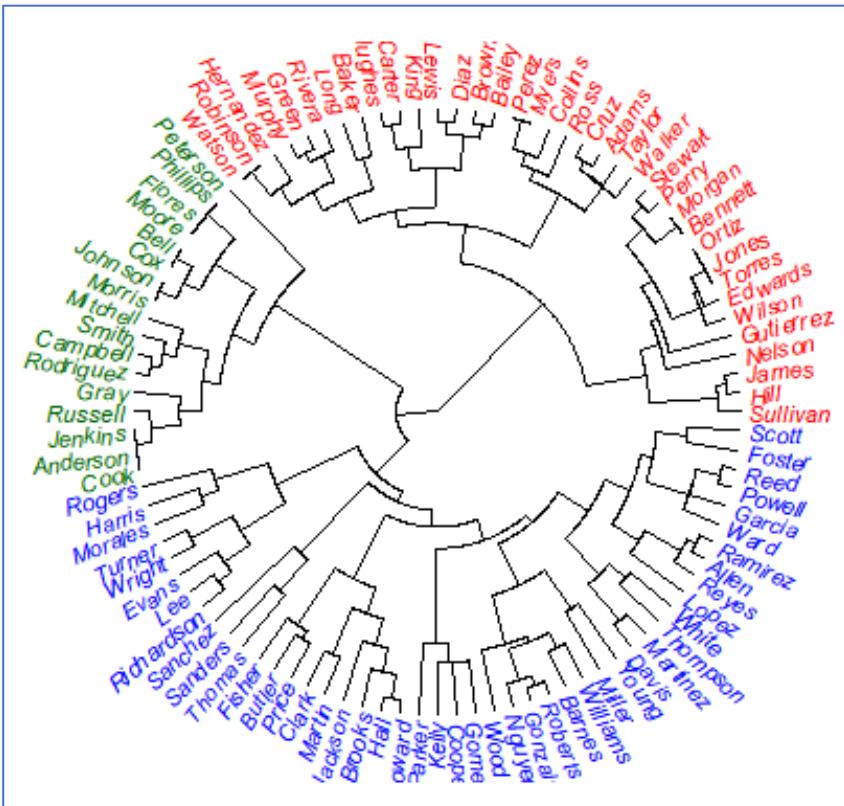
```
plot(H.fit) # display dendrogram
groups <- cutree(H.fit, k=4)
rect.hclust(H.fit, k=4, border="red")
```





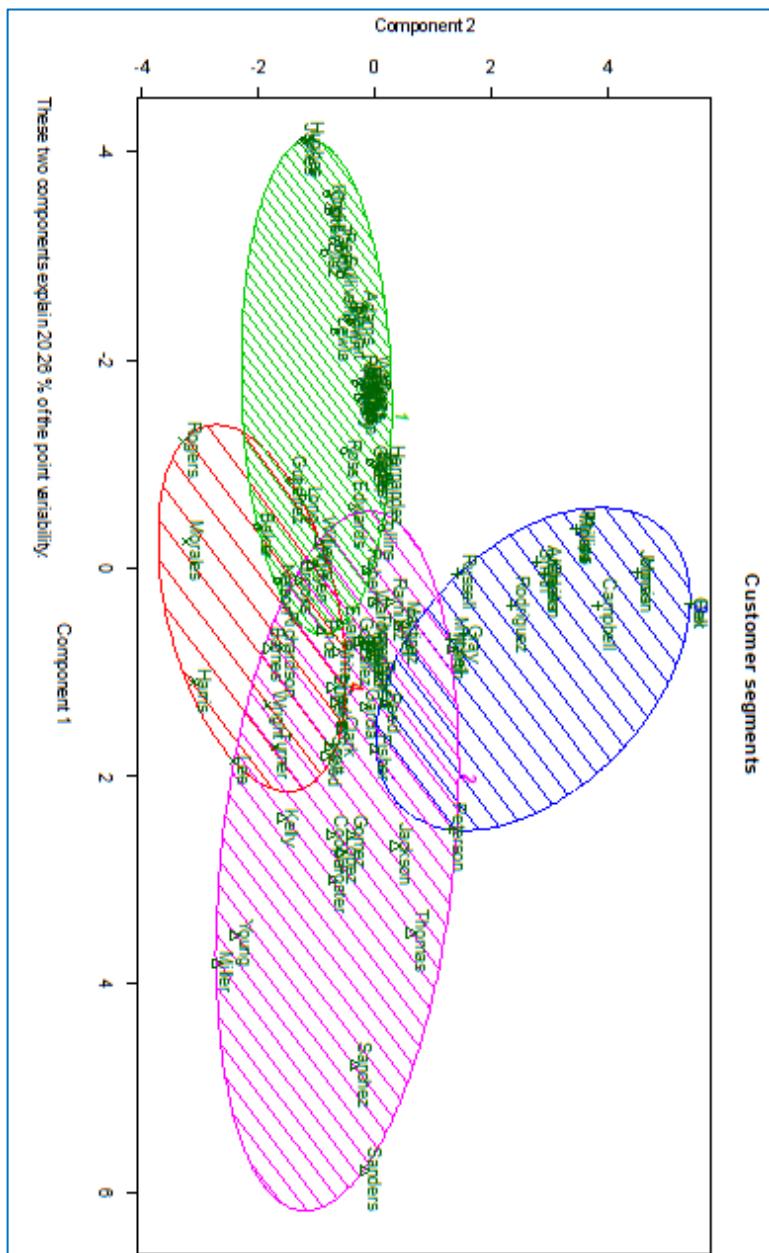
For a little different view, we use the ape-package and present the dendrogram as a fan. The ape package provides several other dendrogram representations that you should try.

```
# install.packages("ape")
library("ape")
# Change the appearance
# Plot as a fan with edge and label (tip)
colors = c("red", "blue", "darkgreen")
clus3 = cutree(H.fit, 3)
plot(as.phylo(H.fit), type = "fan",
     tip.color = colors[clus3],
     label.offset = 0.01, cex = 0.7)
```



## 2D representation of the Segmentation:

```
clusplot(cluster.data, groups, color=TRUE, shade=TRUE,  
        labels=2, lines=0, main= 'Customer segments')
```



Generally, the way K-Means algorithms work is via an iterative refinement process:

- Each data point is randomly assigned to a cluster (number of clusters is given before hand).
- Each cluster's centroid (mean within cluster) is calculated.
- Each data point is assigned to its nearest centroid (iteratively to minimise the within-cluster variation) until no major differences are found.

Let's have a look at an example in R using the Chatterjee-Price Attitude Data from the library(datasets) package. The dataset is a survey of clerical employees of a large financial organization. The data are aggregated from questionnaires of approximately 35 employees for each of 30 (randomly selected) departments. The numbers give the percent proportion of favourable responses to seven questions in each department. For more details, see ?attitude.

### **Load necessary libraries**

```
library(datasets)
```

### **Inspect data structure**

```
str(attitude)
## 'data.frame': 30 obs. of 7 variables:
## $ rating   : num  43 63 71 61 81 43 58 71 72 67 ...
## $ complaints: num  51 64 70 63 78 55 67 75 82 61 ...
## $ privileges: num  30 51 68 45 56 49 42 50 72 45 ...
## $ learning  : num  39 54 69 47 66 44 56 55 67 47 ...
## $ raises    : num  61 63 76 54 71 54 66 70 71 62 ...
## $ critical  : num  92 73 86 84 83 49 68 66 83 80 ...
## $ advance   : num  45 47 48 35 47 34 35 41 31 41 ...
```

### **Summarise data**

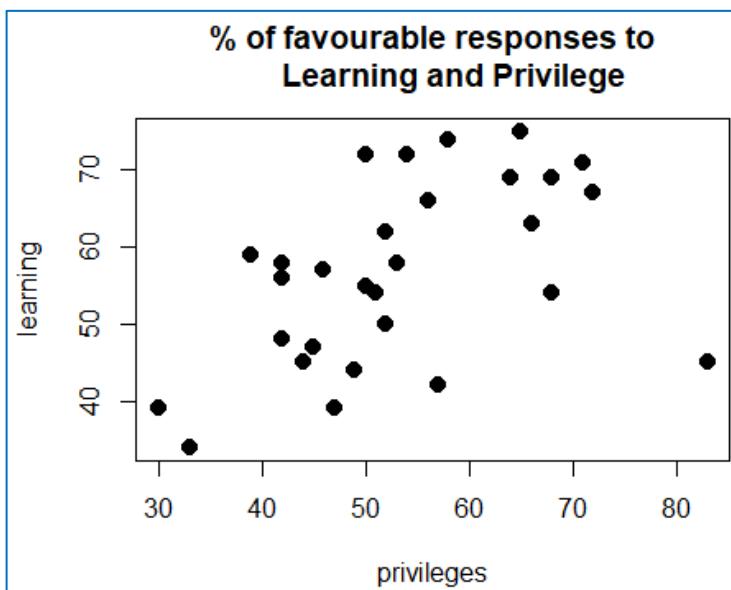
```
summary(attitude)
##      rating      complaints      privileges      learning
## Min.   :40.00   Min.   :37.0   Min.   :30.00   Min.   :34.00
## 1st Qu.:58.75  1st Qu.:58.5  1st Qu.:45.00  1st Qu.:47.00
## Median :65.50  Median :65.0  Median :51.50  Median :56.50
## Mean   :64.63  Mean   :66.6  Mean   :53.13  Mean   :56.37
## 3rd Qu.:71.75  3rd Qu.:77.0  3rd Qu.:62.50  3rd Qu.:66.75
## Max.   :85.00  Max.   :90.0  Max.   :83.00  Max.   :75.00
##      raises      critical      advance
## Min.   :43.00  Min.   :49.00  Min.   :25.00
## 1st Qu.:58.25  1st Qu.:69.25  1st Qu.:35.00
## Median :63.50  Median :77.50  Median :41.00
## Mean   :64.63  Mean   :74.77  Mean   :42.93
## 3rd Qu.:71.00  3rd Qu.:80.00  3rd Qu.:47.75
## Max.   :88.00  Max.   :92.00  Max.   :72.00
```

As we've seen, this data gives the percent of favourable responses for each department. For example, in the summary output above we can see that for the variable privileges among all 30 departments the minimum percent of favourable responses was 30 and the maximum was 83. In other words, one department had only 30% of responses favourable when it came to assessing 'privileges' and one department had 83% of favorable responses when it came to assessing 'privileges', and a lot of other favourable response levels in between.

In light of the example, we'll take a subset of the attitude dataset and consider only two variables in our K-Means clustering exercise. So imagine that we would like to cluster the attitude dataset with the responses from all 30 departments when it comes to 'privileges' and 'learning' and we would like to understand whether there are commonalities among certain departments when it comes to these two variables.

### **Subset and Plot the Attitude Data**

```
dat = attitude[,c(3,4)]  
plot(dat, main = "% of favourable responses to  
Learning and Privilege", pch =20, cex =2)
```

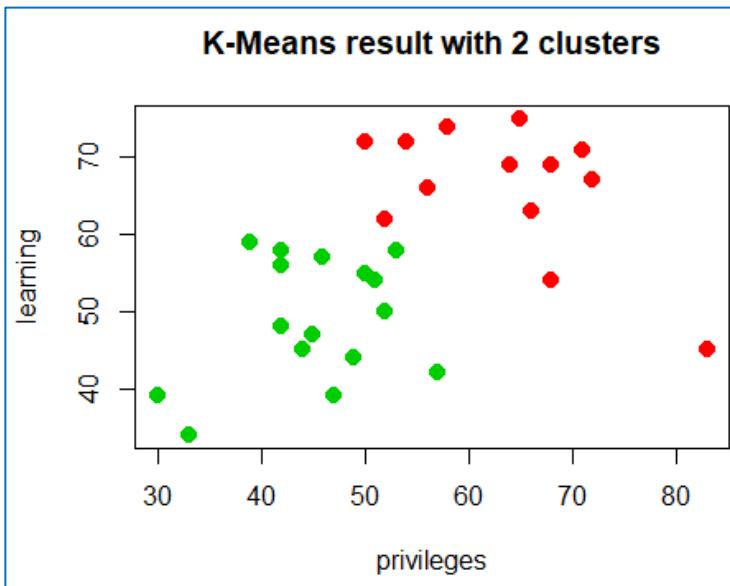


With the data subset and the plot above we can see how each department's score behave across Privilege and Learning compare to each other. In the most simplistic sense, we can apply K-Means clustering to this data set and try to assign each department to a specific number of clusters that are “similar”.

### K-Means Model

Let's use the kmeans function from R base stats package to Perform K-Means with 2 clusters

```
set.seed(7)
km1 = kmeans(dat, 2, nstart=100)
plot(dat, col =(km1$cluster +1) , main="K-Means result with 2 clusters", pch=20, cex=2)
```



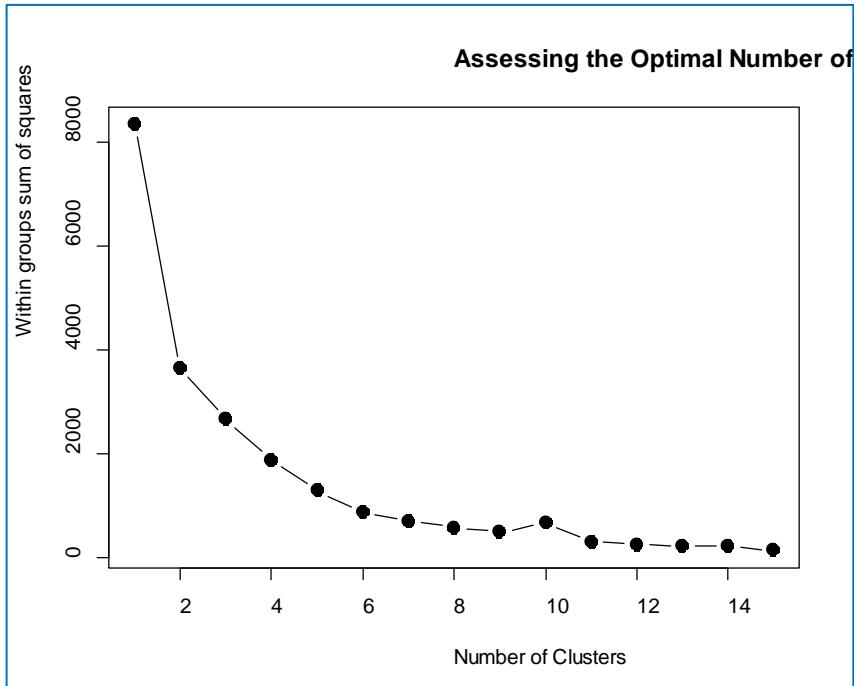
As mentioned before, one of the key decisions to be made when performing K-Means clustering is to decide on the numbers of clusters to use. In practice, there is no easy answer and it's important to try different ways and numbers of clusters to decide which options is the most useful, applicable or interpretable solution.

In the plot above, we randomly chose the number of clusters to be 2 for illustration purposes only.

However, one solution often used to identify the optimal number of clusters is called the Elbow method and it involves observing a set of possible numbers of clusters relative to how they minimise the within-cluster sum of squares. In other words, the Elbow method examines the within-cluster dissimilarity as a function of the number of clusters. Below is a visual representation of the method:

Check for the optimal number of clusters given the data.

```
mydata <- dat
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
  for (i in 2:15) wss[i] <- sum(kmeans(mydata,
                                         centers=i)$withinss
)
plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares",
     main="Assessing the Optimal Number of Clusters with t
he Elbow Method",
     pch=20, cex=2)
```



We can say that after 6 clusters the observed difference in the within-cluster dissimilarity is not substantial. Consequently, we can say with

some reasonable confidence that the optimal number of clusters to be used is 6.

Perform K-Means with the optimal number of clusters identified from the Elbow method

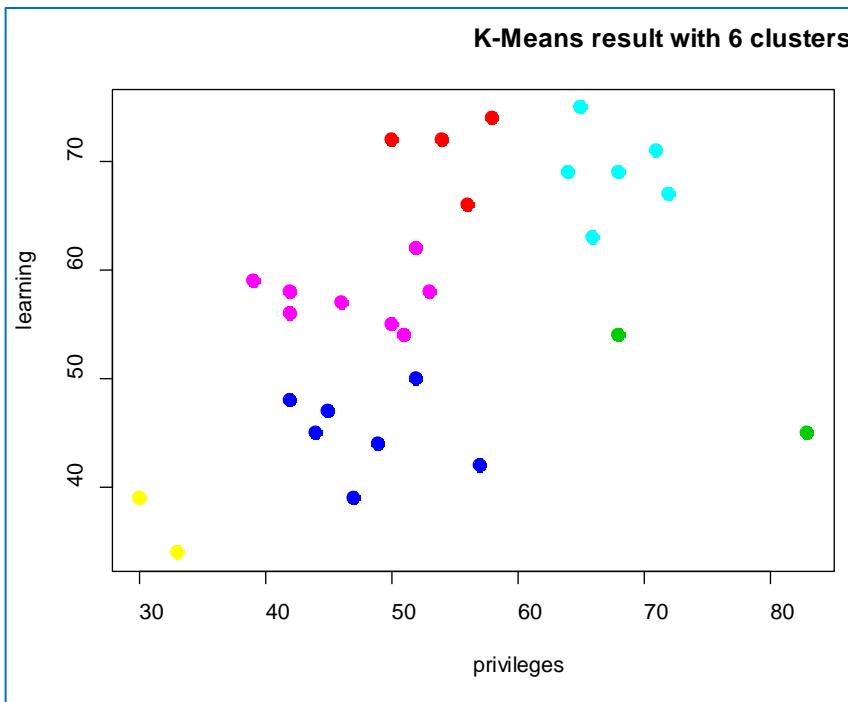
```
set.seed(7)
km2 = kmeans(dat, 6, nstart=100)
```

Examine the result of the clustering algorithm

```
km2
## K-means clustering with 6 clusters of sizes 4, 2, 8, 6,
8, 2
##
## Cluster means:
##   privileges learning
## 1 54.50000 71.000
## 2 75.50000 49.500
## 3 47.62500 45.250
## 4 67.66667 69.000
## 5 46.87500 57.375
## 6 31.50000 36.500
##
## Clustering vector:
##  [1] 6 5 4 3 1 3 5 5 4 3 5 3 3 2 1 1 4 4 5 2 6 5 3 5 3
4 1 3 4 5
##
## Within cluster sum of squares by cluster:
## [1] 71.0000 153.0000 255.3750 133.3333 244.7500 17.00
00
## (between_SS / total_SS =  89.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"       "totss"        "withi
nss"
## [5] "tot.withinss" "betweenss"     "size"         "iter"
## [9] "ifault"
```

Now, we plot the results:

```
plot(dat, col =(km2$cluster +1) , main="K-Means result wit
h 6 clusters", pch=20, cex=2)
```



From the results above we can see that there is a relatively well-defined set of groups of departments that are relatively distinct when it comes to answering favourably around Privileges and Learning in the survey. It is only natural to think the next steps from this sort of output. One could start to devise strategies to understand why certain departments rate these two different measures the way they do and what to do about it. But we will leave this to another exercise.

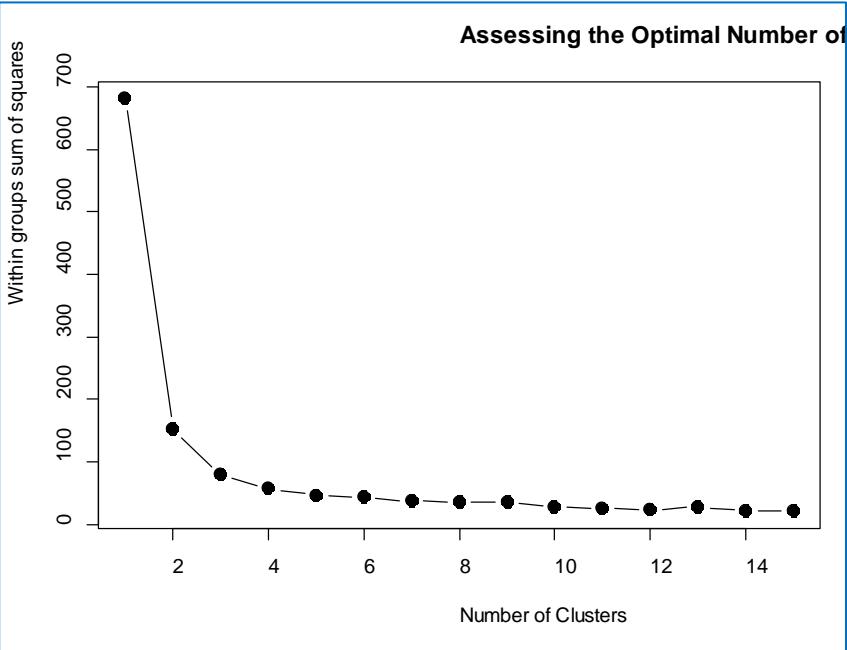
### Iris K-Means #1

```
iris2 <- iris
iris2$Species <- NULL
mydata <- iris2
```

How many clusters?

```
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
  for (i in 2:15) wss[i] <- sum(kmeans(mydata,
                                         centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares",
     main="Assessing the Optimal Number of Clusters with t")
```

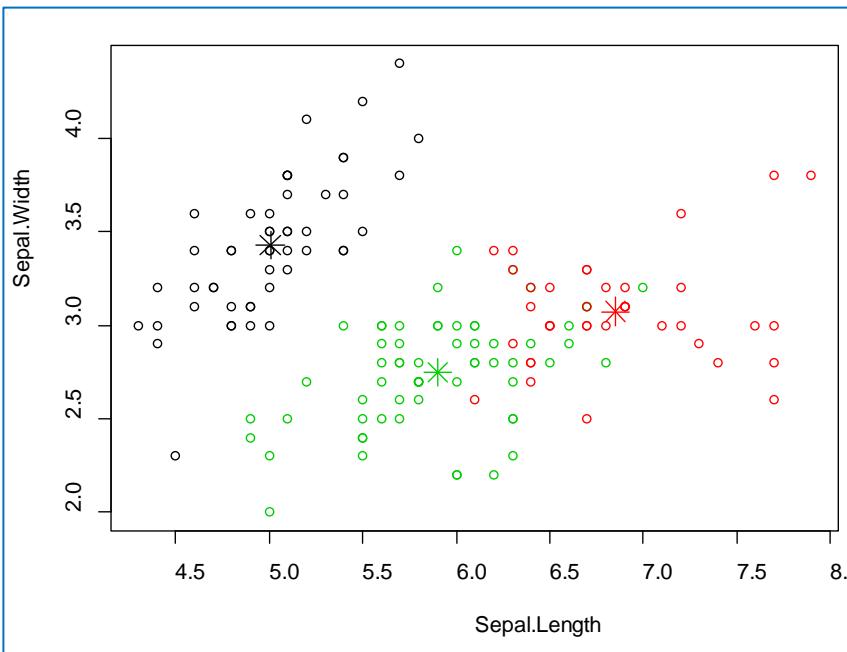
he Elbow Method”,  
pch=20, cex=2)



```

## [5] "tot.withinss" "betweenss"      "size"           "iter"
## [9] "ifault"


```

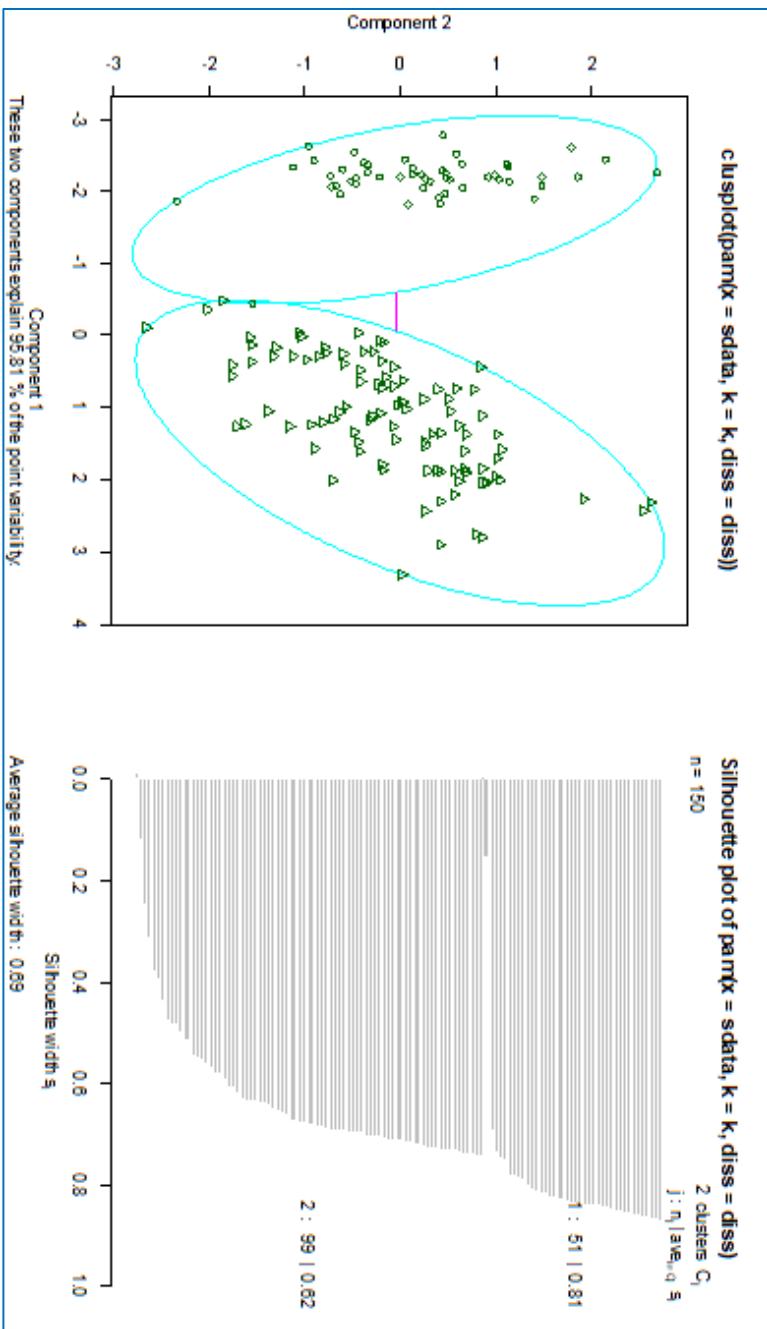


## Iris K-Means #2

```

library(fpc)
pamk.result <- pamk(iris2)
pamk.result$nc


```

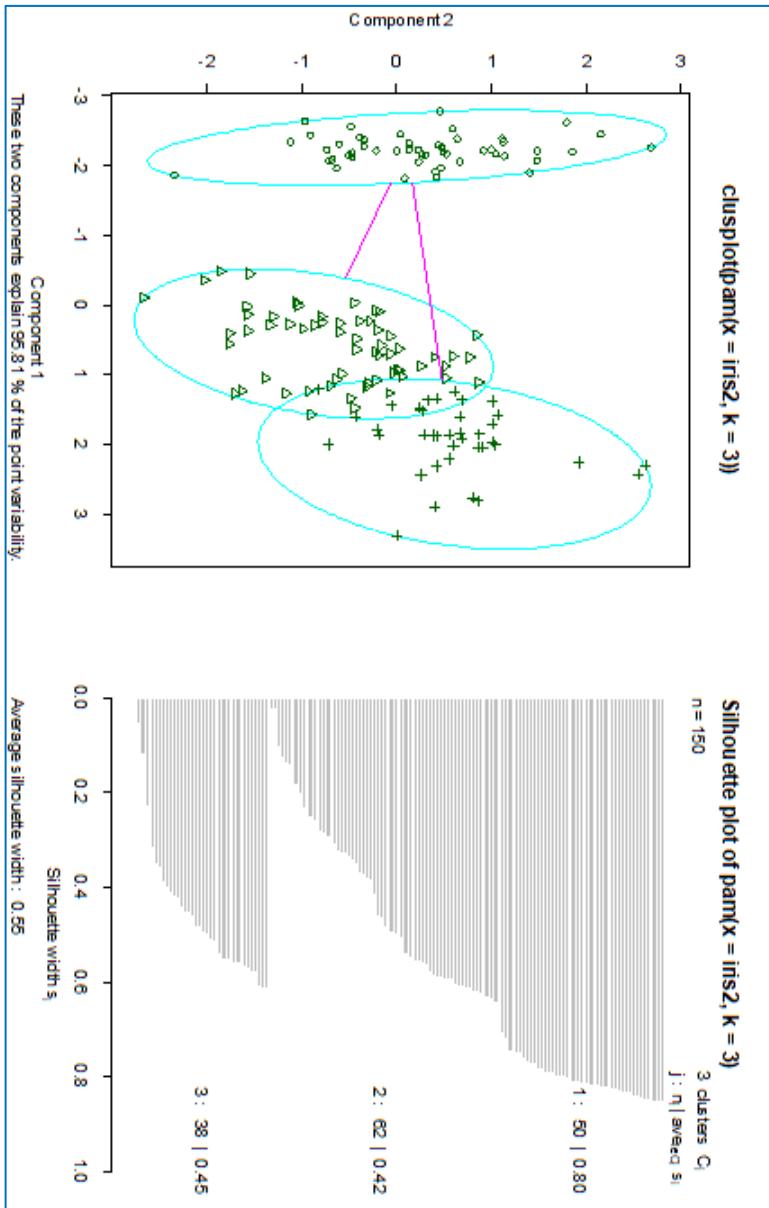


```
layout(matrix(1)) # change back to one graph per page
```

```

library(fpc)
pam.result <- pam(iris2, 3)
layout(matrix(c(1,2),1,2)) # 2 graphs per page
plot(pam.result)

```



```

table(pam.result$clustering, iris$Species)

```

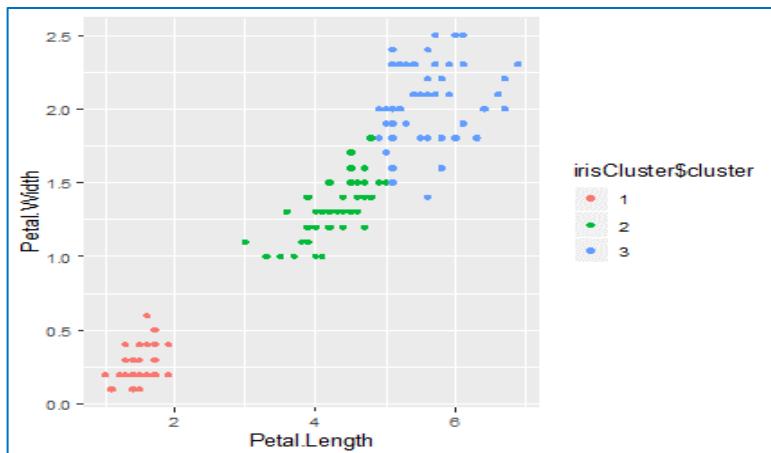
```

##      setosa versicolor virginica
## 1      50          0         0
## 2      0          48        14
## 3      0          2         36

layout(matrix(1)) # change back to one graph per page

```

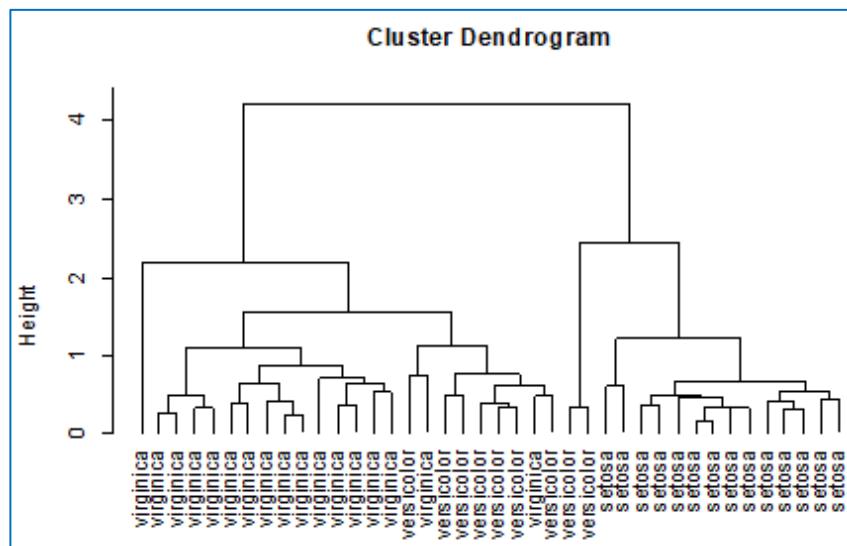
## Iris K-Means #3



## Hierarchical Clustering

Here, we take the same data (iris) and examine the relationships between the various features using hierarchical clustering. We will use the `hclust` function for doing this.

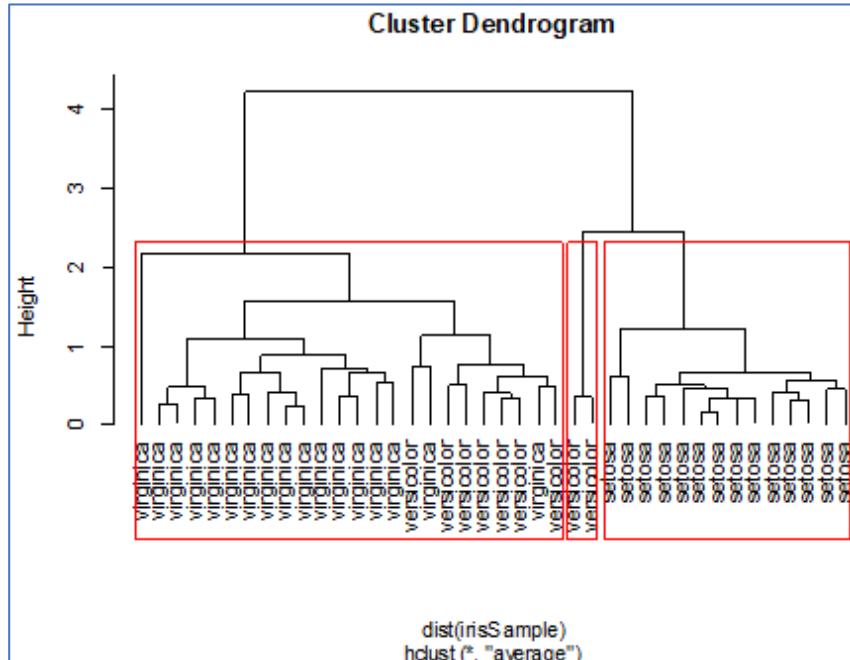
```
idx <- sample(1:dim(iris)[1], 40)
irisSample <- iris[idx,]
irisSample$Species <- NULL #fill empty records
hc <- hclust(dist(irisSample), method="ave")
plot(hc, hang = -1, labels=iris$Species[idx])
```



```
dist(irisSample)
hclust(*,"average")
```

Now, we cut tree into 3 clusters

```
plot(hc, hang = -1, labels=iris$Species[idx])
rect.hclust(hc, k=3)
```



```
dist(irisSample)
hclust(*, "average")
```

```
groups <- cutree(hc, k=3)
```

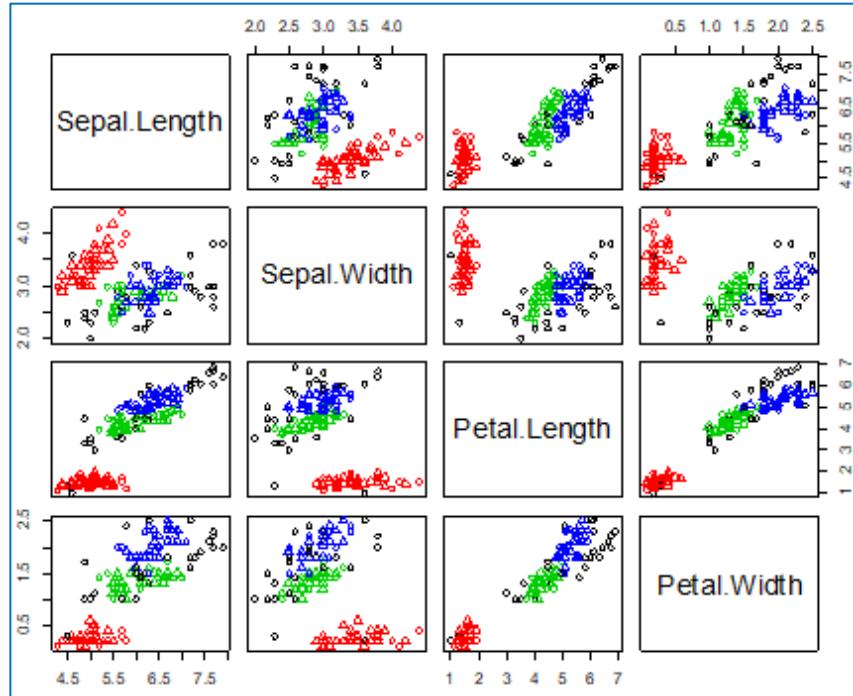
## Density-based Clustering

Next, we examine the iris dataset using density-base clustering and the fpc package:

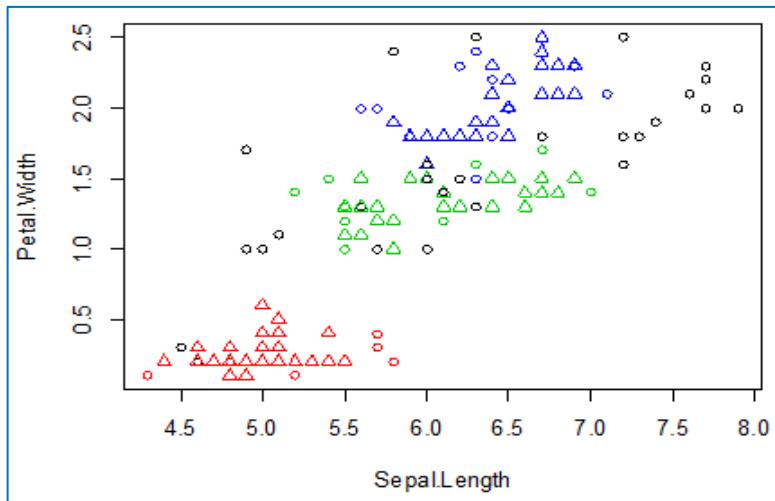
```
library(fpc)
iris2 <- iris[-5] # remove class tags
ds <- dbSCAN(iris2, eps=0.42, MinPts=5)
# compare clusters with original class labels
table(ds$cluster, iris$Species)

##
##      setosa versicolor virginica
##    0        2       10      17
##    1       48        0       0
```

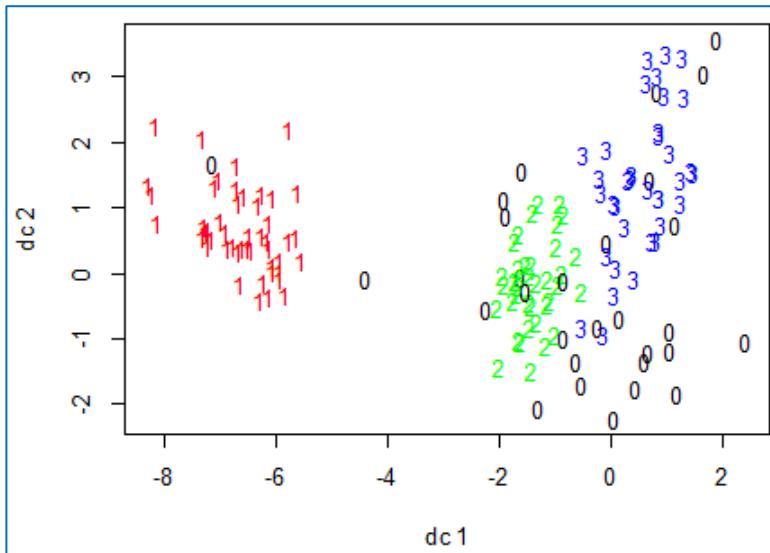
```
##    2      0      37      0
##    3      0       3     33
plot(ds, iris2)
```



```
plot(ds, iris2[c(1,4)])
```



```
plotcluster(iris2, ds$cluster)
```



Now we 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)
myPred <- predict(ds, iris2, newData)
```

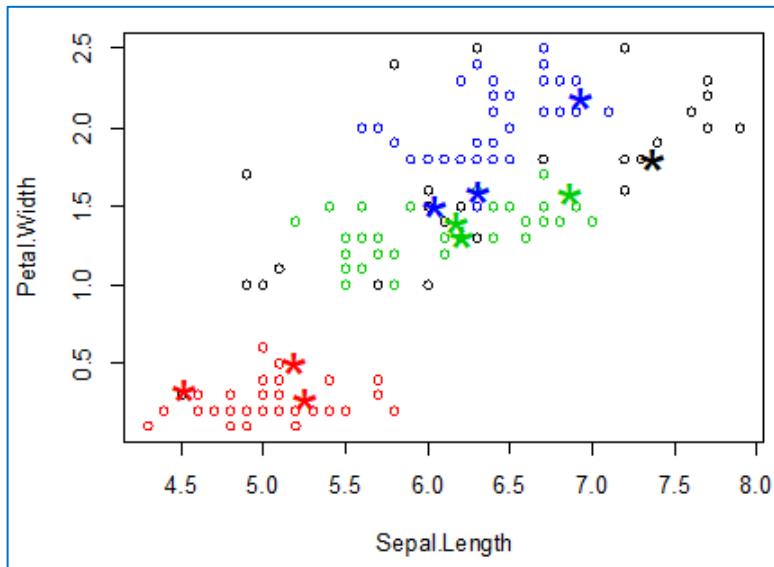
Now, we combine the plots and show the result:

```
plot(iris2[c(1,4)], col=1+ds$cluster)
points(newData[c(1,4)], pch="*", col=1+myPred, cex=3)
```

### Chaining

The chaining operator (`%>%`) turns  $x \%>\% f(y)$  into  $f(x, y)$  so you can use it to rewrite multiple operations such that they can be read from left-to-right, top-to-bottom. For instance, the the `hclust` on page 164 is:

```
idx <- sample(1:dim(iris)[1], 40)
dend <- iris[idx,] %>%
  dist %>%
  hclust(method="ave") %>%
  as.dendrogram
plot(dend)
```



Finally, we check the cluster labels:

```
table(myPred, iris$Species[idx])
##
## myPred setosa versicolor virginica
##    0      0        0       1
##    1      3        0       0
##    2      0        3       0
##    3      0        1       2
```

## Advanced Cluster Models in R

The functions in the vegan package contain tools for diversity analysis, ordination methods and tools for the analysis of dissimilarities. Together with the labdsv package, the vegan package provides most standard tools of descriptive community analysis. The dune meadow vegetation data, dune, has cover class values of 30 species on 20 sites. The corresponding environmental data frame dune.env has following entries: - A1: a numeric vector of thickness of soil A1 horizon. - Moisture: an ordered factor with levels: 1 < 2 < 4 < 5. - Management: a factor with levels: BF (Biological farming), HF (Hobby farming), NM (Nature

Conservation Management), and SF (Standard Farming). - Use: an ordered factor of land-use with levels: Hayfield < Haypasture < Pasture. - Manure: an ordered factor with levels: 0 < 1 < 2 < 3 < 4.

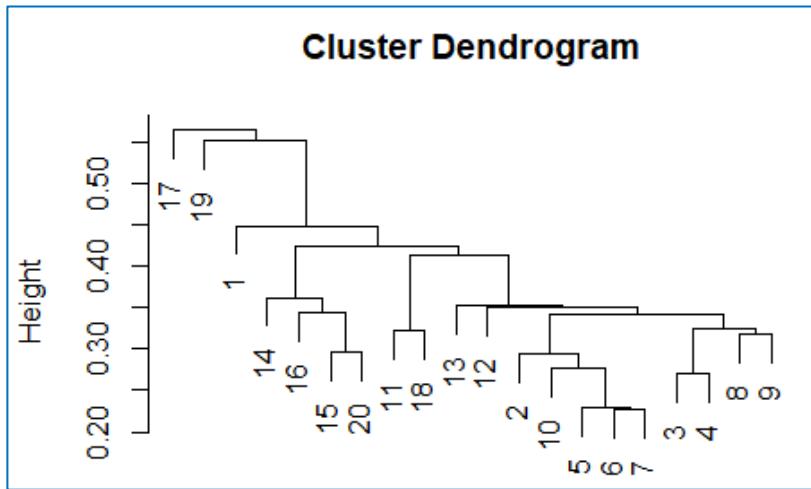
```
if(!require(vegan)) install.packages("vegan")
if(!require(vegdata)) install.packages("vegdata")
library(vegan)
library(vegdata)
```

### Example 1: h-cluster

```
data(dune)
dis <- vegdist(dune)
```

Function hclust provides several alternative clustering strategies. In community ecology, most popular are single linkage a.k.a. nearest neighbour, complete linkage a.k.a. furthest neighbour, and various brands of average linkage methods.

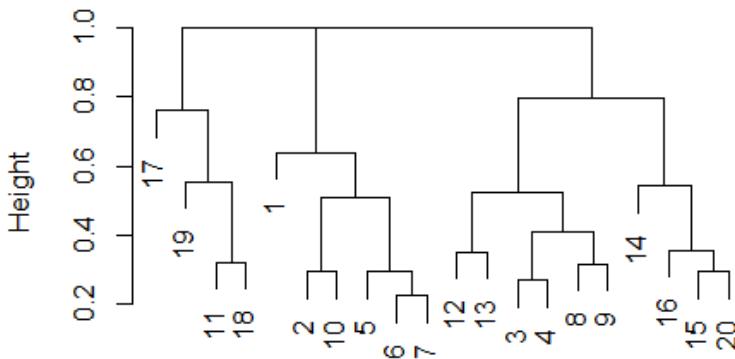
```
clus <- hclust(dis, "single")
plot(clus)
```



```
dis
hclust(*,"complete")
```

```
cluc <- hclust(dis, "complete")
plot(cluc)
```

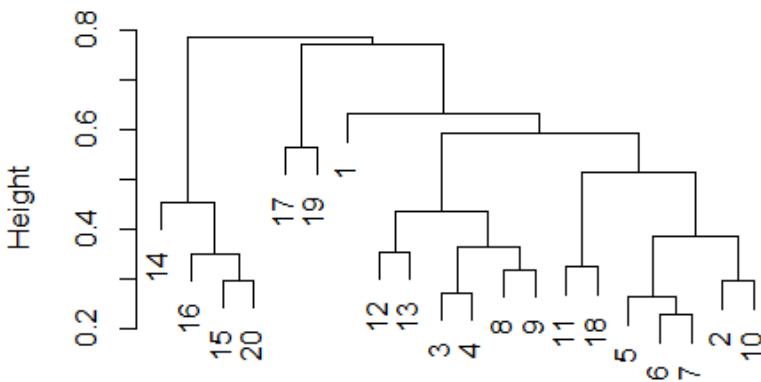
### Cluster Dendrogram



```
dis  
hclust(*,"complete")
```

```
clua <- hclust(dis, "average")  
plot(clua)
```

### Cluster Dendrogram

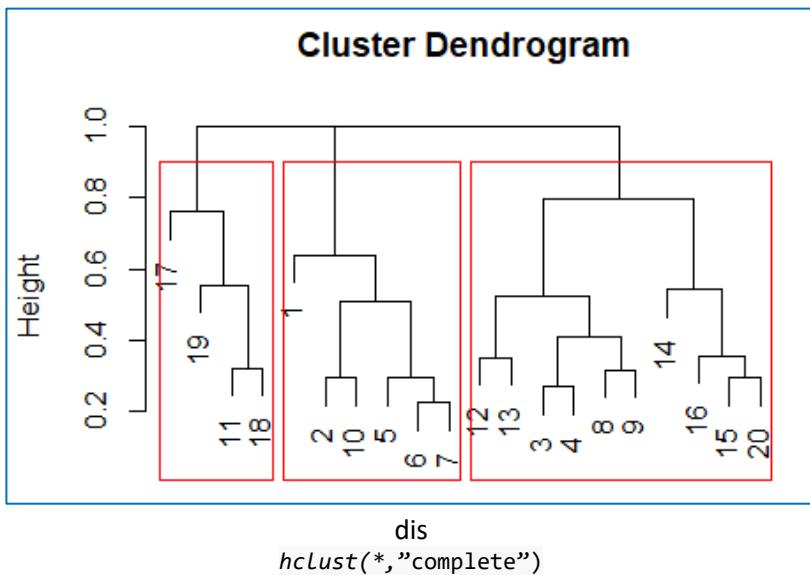


```
dis  
hclust(*,"complete")
```

```
range(dis)  
## [1] 0.2272727 1.0000000
```

**Cophenetic correlation** measures the similarity between original dissimilarities and dissimilarities estimated from the tree

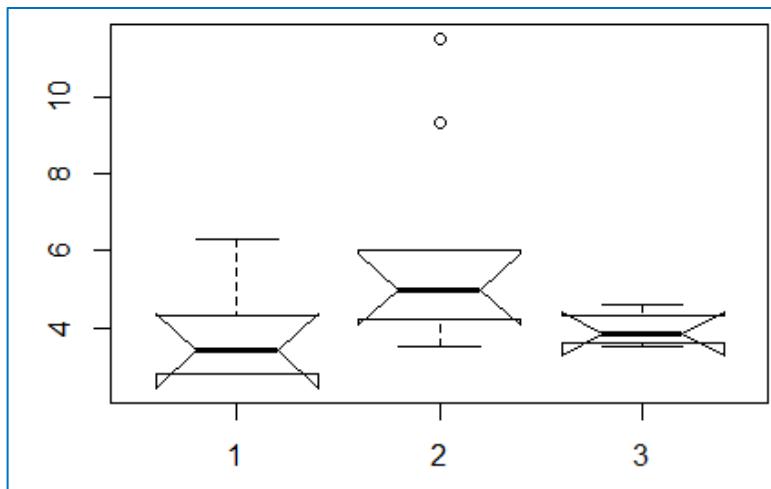
```
cor(dis, cophenetic(clus))
## [1] 0.6601692
cor(dis, cophenetic(cluc))
## [1] 0.6707479
cor(dis, cophenetic(clua))
## [1] 0.8168825
# Cutting
plot(cluc)
rect.hclust(cluc, 3)
```



```
grp <- cutree(cluc, 3)
```

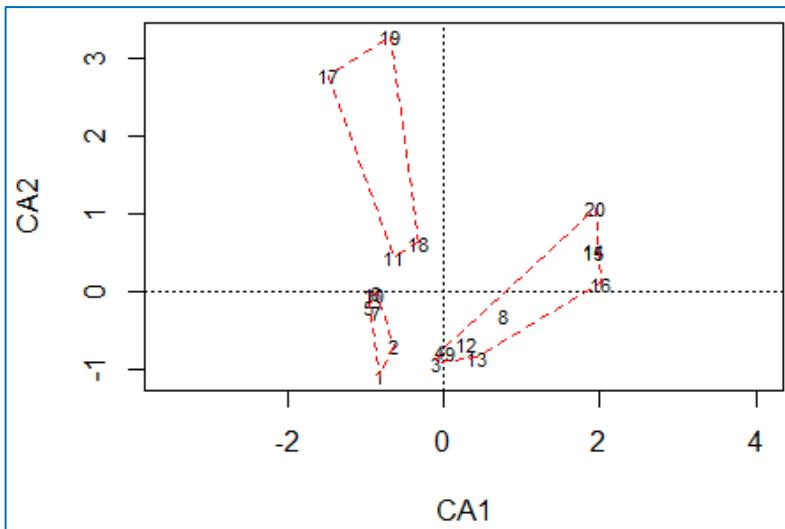
The only continuous variable in the Dune data is the thickness of the A1 horizon

```
data(dune.env)
boxplot(A1 ~ grp, data=dune.env, notch = TRUE)
## Warning in bxp(list(stats = structure(c(2.8, 2.8, 3.4,
4.3, 6.3, 3.5,
## 4.2, : some notches went outside hinges ('box'): maybe
set notch=FALSE
```



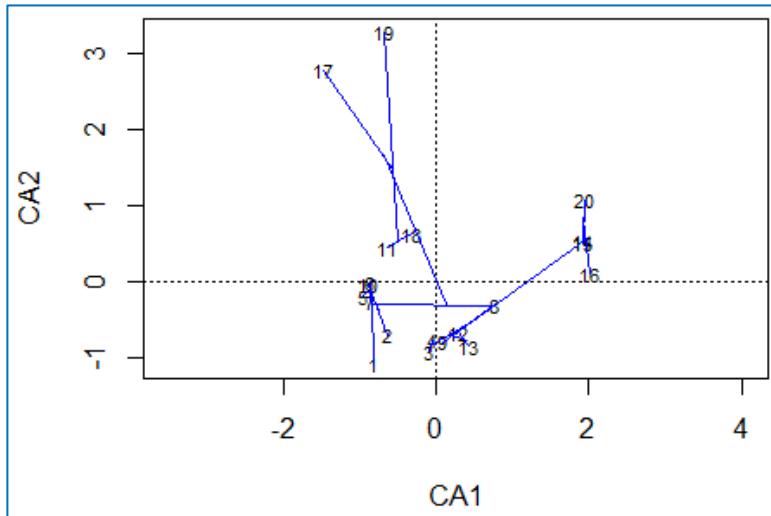
The clustering results can be displayed in ordination diagrams. All usual vegan functions for factors can be used: **ordihull**, **ordispider**, and **ordiellipse**.

```
ord <- cca(dune)
plot(ord, display = "sites")
ordihull(ord, grp, lty = 2, col = "red")
```



The vegan package has the function **ordicluster** to overlay hclust tree in an ordination.

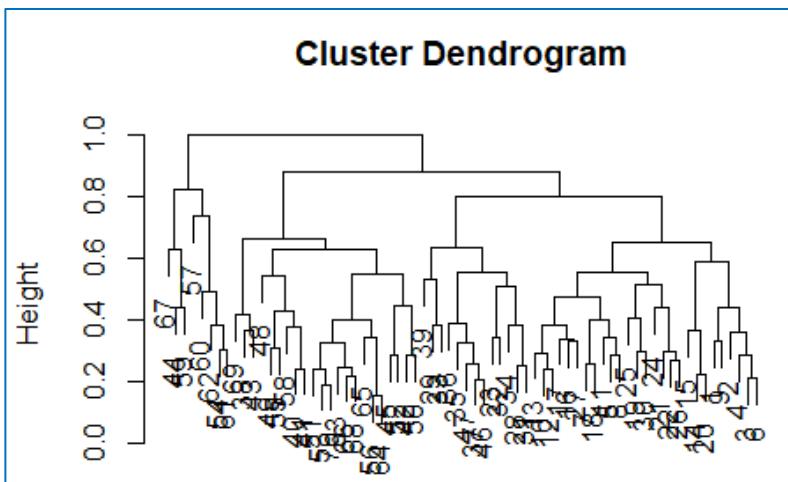
```
plot(ord, display="sites")
ordicluster(ord, cluc, col="blue")
```



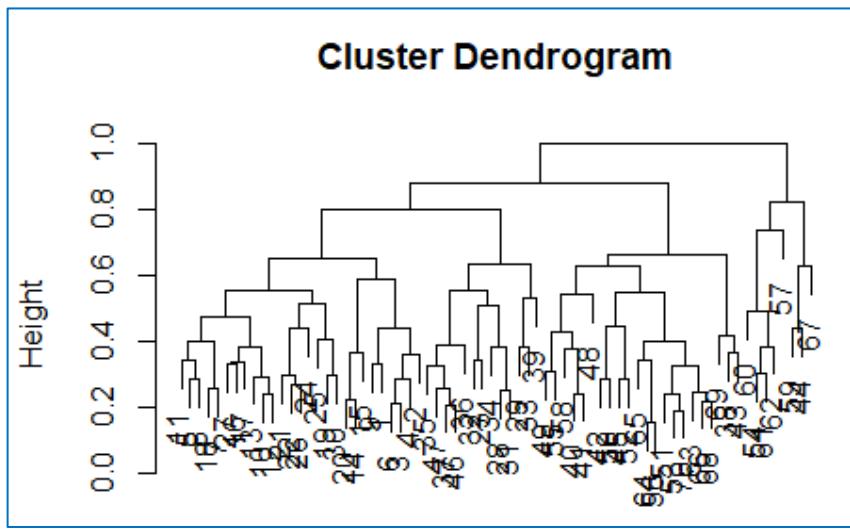
### Example 2

Now, we reorder by water content of soil.

```
data(mite, mite.env)
hc <- hclust(vegdist(wisconsin(sqrt(mite))))
plot(hc)
```



```
ohc <- with(mite.env, reorder(hc, WatrCont))
plot(ohc)
```



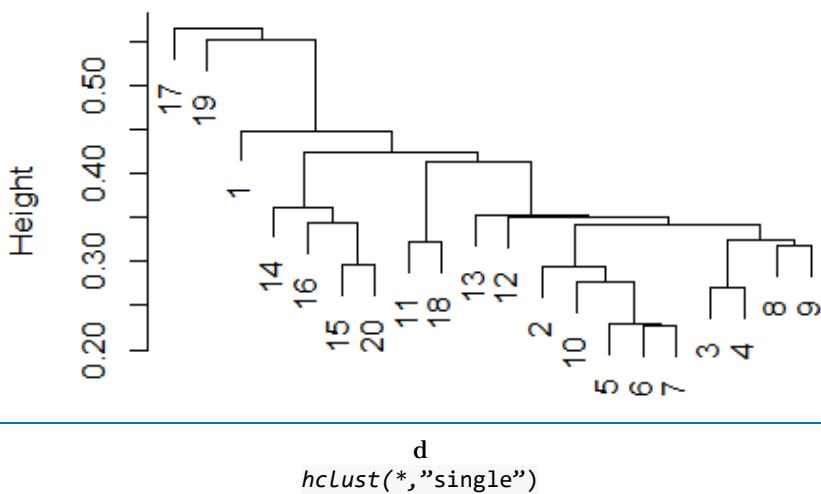
```
vegdist(wisconsin(sqrt(mite)))
hclust(*,"complete")
```

### Example 3

Vegdist computes dissimilarity indices. Here we use the Bray-Curtis which are good in detecting underlying ecological gradients.

```
data(dune)
d <- vegdist(dune)
par(mfrow=c(1,3))
par(mfrow=c(3,1))
par(mfrow=c(1,1))
par(mar=c(3,4,1,1)+.1)
csin <- hclust(d, method="single")
csin
##
## Call:
## hclust(d = d, method = "single")
##
## Cluster method : single
## Distance       : bray
## Number of objects: 20
plot(csin)
```

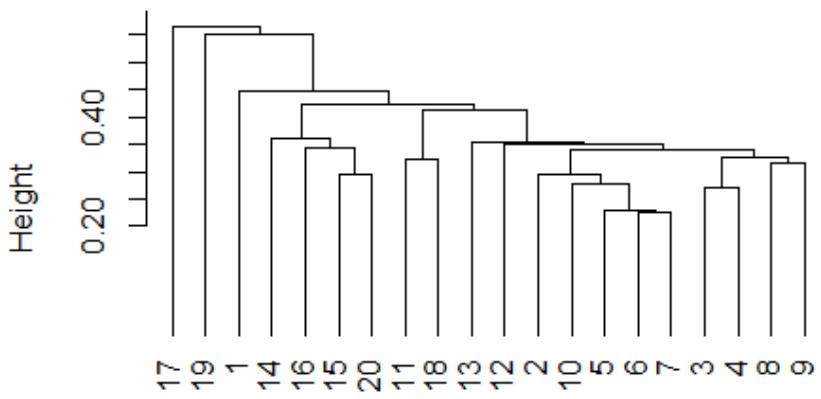
### Cluster Dendrogram



d  
hclust(\*,"single")

```
plot(csin, hang=-1)
```

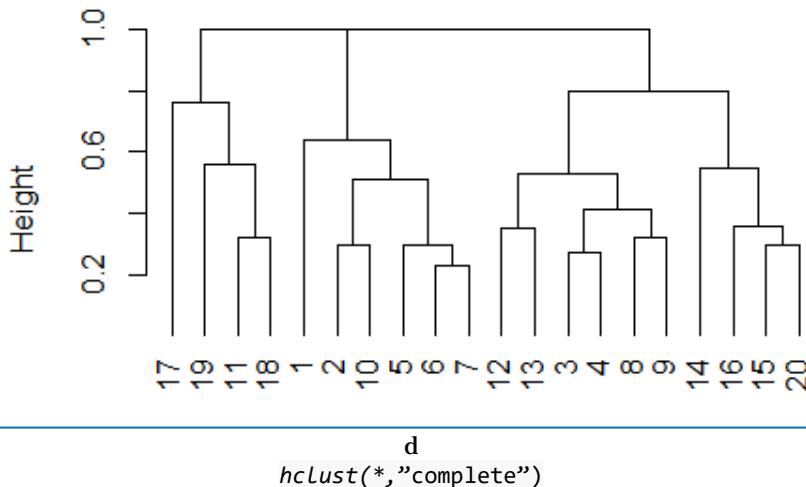
### Cluster Dendrogram



d  
hclust(\*,"single")

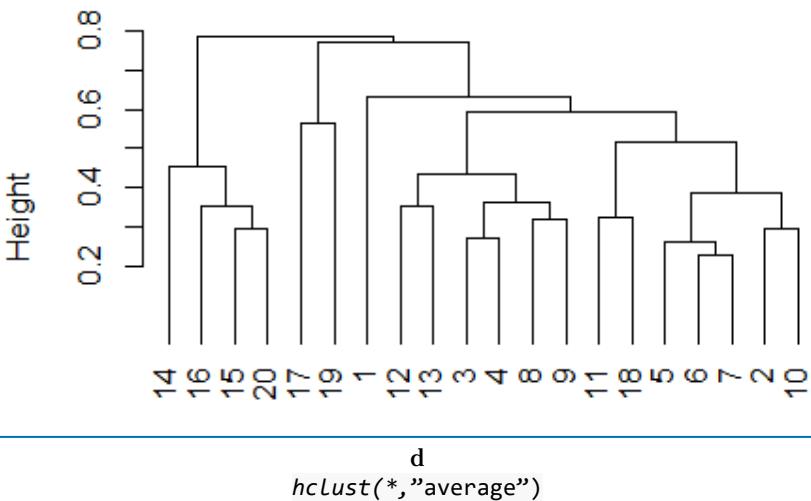
```
ccom <- hclust(d, method="complete")
plot(ccom, hang=-1)
```

### Cluster Dendrogram



```
caver <- hclust(d, method="aver")
plot(caver, hang=-1)
```

### Cluster Dendrogram



```

vegemite(dune, caver)
##
##          111211 11    11    1
##          46507912334891856720
## Comapalu 2.2.....
## Callcusp 43.3.....
## Eleopalu 4854.....4.....
## Ranuflam 2224....2..2.....
## Airaprae ....23.....
## Empenigr .....2.....
## Agrostol 4745...454843.....
## Juncarti .334.....44.....
## Salirepe ...5.3.....3.....
## Hyporadi ....25.....2.....
## Chenalbu .....1.....
## Alopgeni .4.....857253....2.
## Sagiproc ....3.42.5222.....
## Bracruta .444.3.4.222246262.2
## Cirsarve .....2.....
## Juncbufo .....43...4....2..
## Scorautu 2.2226.2222325533353
## Elymrepe .....4..44.6..4..4.
## Trifrepe 6.1..2.3221233225256
## Anthodor ....44.....432.4
## Poatriv .2....2496545..64574
## Poaprat ....1.4.254444323444
## Lolipere .....7..65427226656
## Rumeacet .....2....2..563..
## Bellpere .....22...22..32
## Vicilath .....21....1
## Planlanc ....2.....33555.3
## Achimill ....2.1.....22234
## Trifprat .....252..
## Bromhord .....3....2.244
##   sites species
##      20      30

```

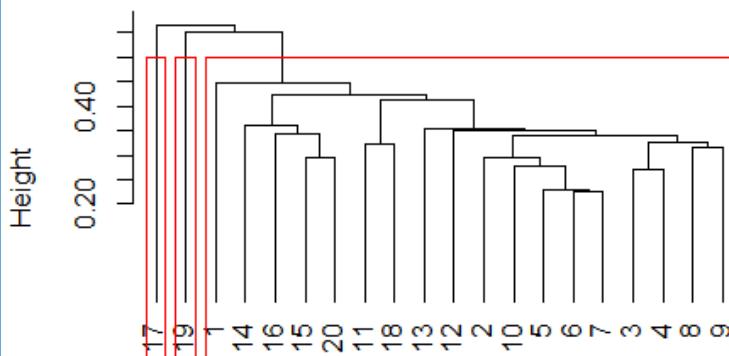
Here we will plot our dendograms using the agglomeration methods “single,” “complete”, and “average.” Notice the difference in sclusters.

```

plot(csin, hang=-1)
rect.hclust(csin, 3)

```

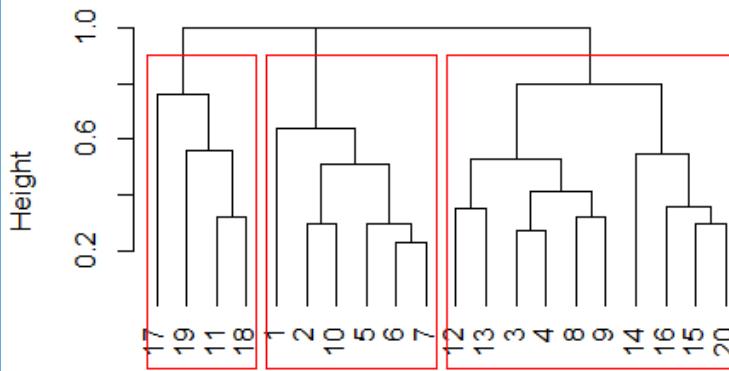
### Cluster Dendrogram



$d$   
`hclust(*, "single")`

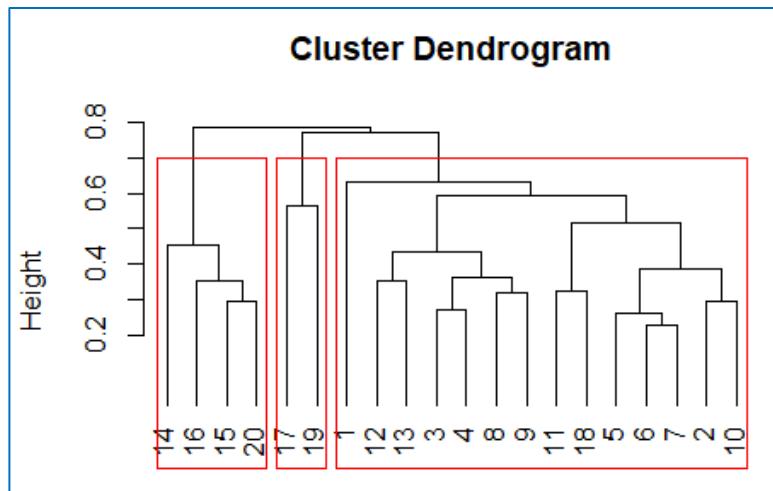
```
plot(ccom, hang=-1)
rect.hclust(ccom, 3)
```

### Cluster Dendrogram



$d$   
`hclust(*, "complete")`

```
plot(caver, hang=-1)
rect.hclust(caver, 3)
```



*d*  
*hclust(\*, "average")*

We can use the function **ordiplot** and **orditorp** to add text to the plot in place of points to make some sense of this rather non-intuitive mess. We can also plot “spider graphs” using the function **orderspider**, ellipses using the function **ordiellipse**, or a minimum spanning tree (MST) using **ordicluster** which connects similar communities (useful to see if treatments are effective in controlling community structure).

```

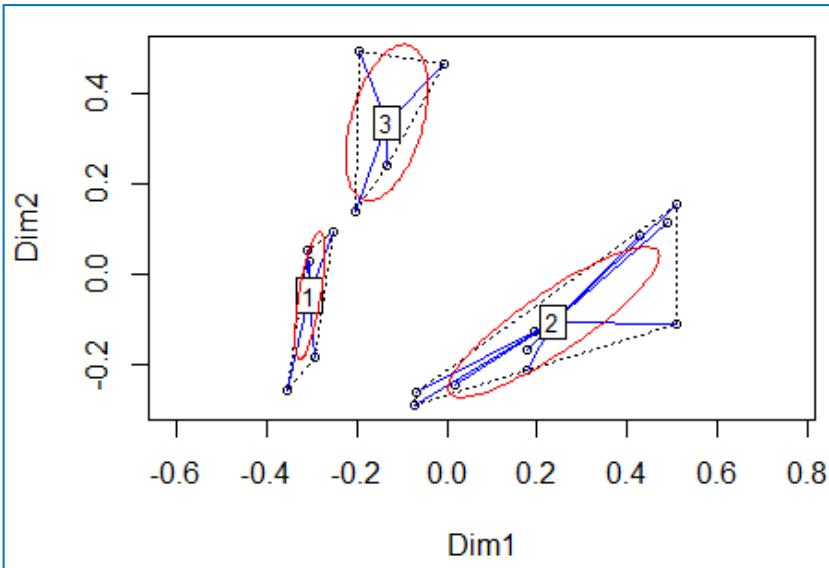
cl <- cutree(ccom, 3)
cl
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  1  1  2  2  1  1  1  2  2  1  3  2  2  2  2  2  3  3  3  2
table(cl)
## cl
##  1  2  3
##  6 10  4
table(cl, cutree(csin, 3))
##
## cl   1   2   3
##   1   6   0   0
##   2 10   0   0
##   3   2   1   1
table(cl, cutree(caver, 3)) # Classical tree pruning
##
## cl  1 2 3
##   1 6 0 0
##   2 6 4 0
##   3 2 0 2

```

```

ord <- cmdscale(d) # Classical Multidimensional Scaling
ordiplot(ord) ## species scores not available
ordihull(ord, cl, lty=3)
ordispider(ord, cl, col="blue", label=TRUE)
ordielipse(ord, cl, col="red")

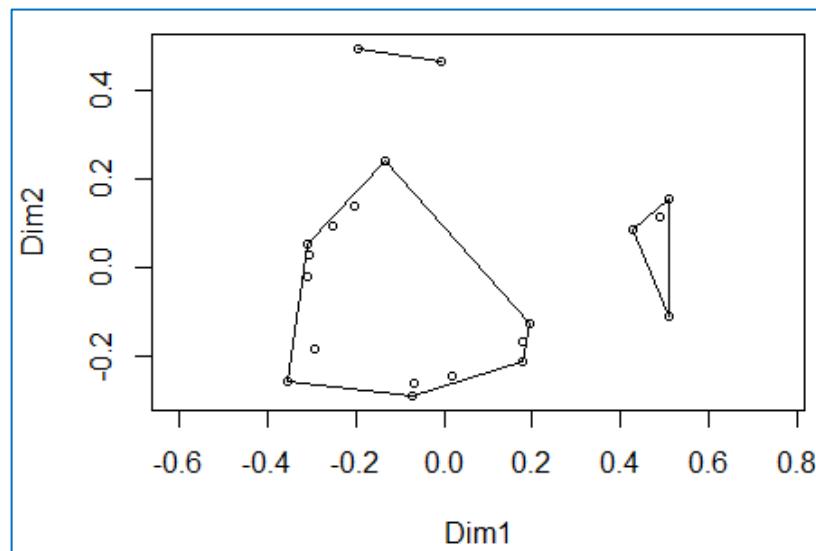
```



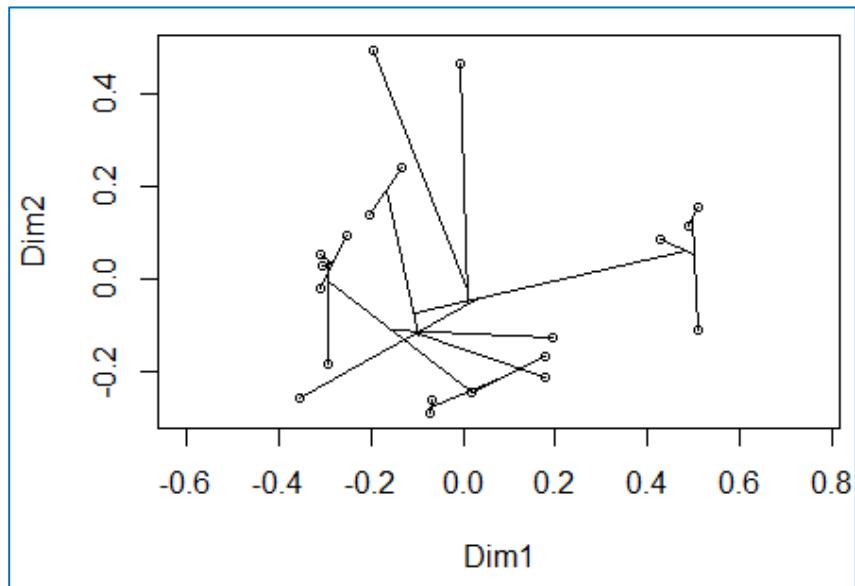
```

ordiplot(ord, dis="si")
ordihull(ord, cutree(cavertree, 3))

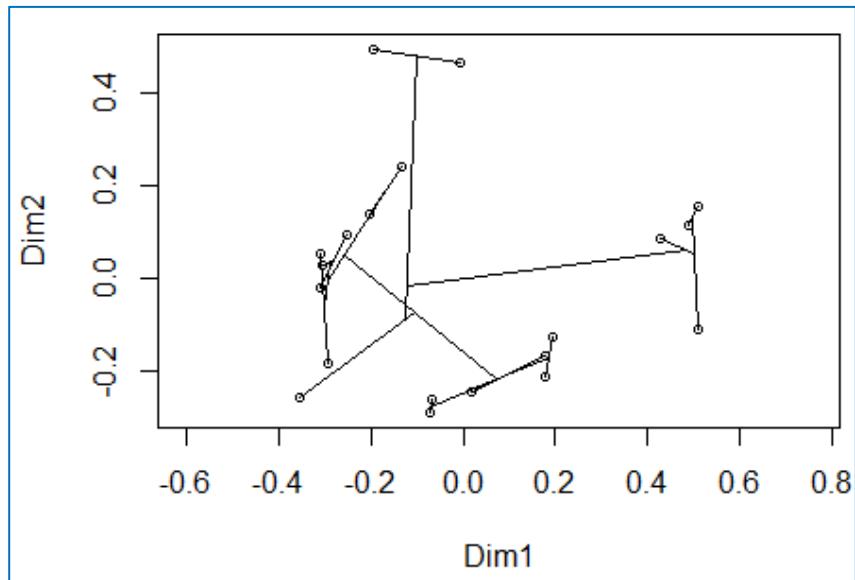
```



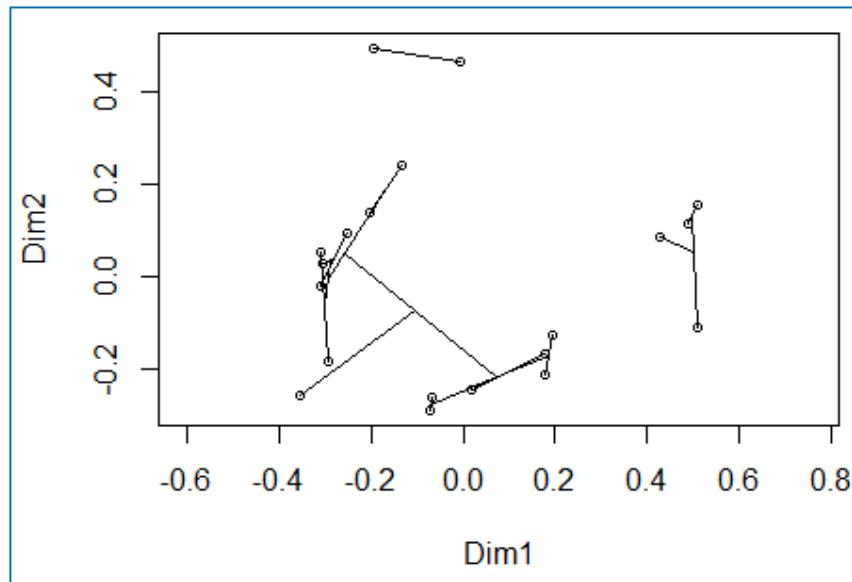
```
ordiplot(ord, dis="si")
ordicluster(ord, csin)
```



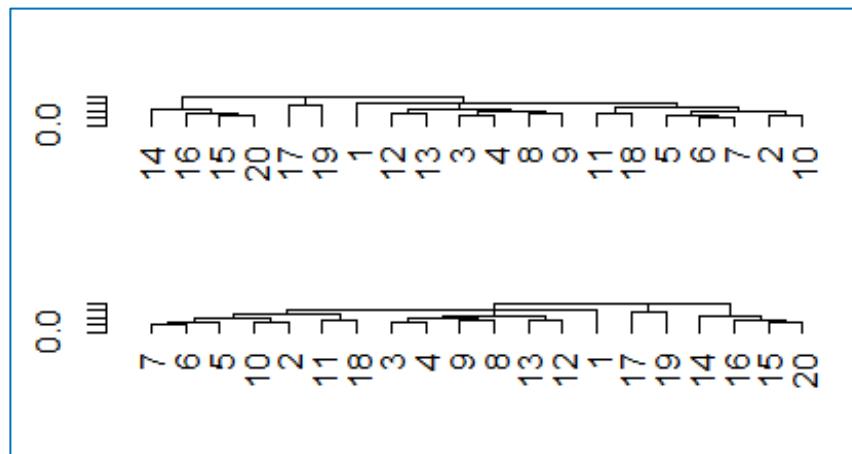
```
ordiplot(ord, dis="si")
ordicluster(ord, caver)
```



```
ordiplot(ord, dis="si")
ordicluster(ord, caver, =2)
```



```
den <- as.dendrogram(caver)
x <- scores(ord, display = "sites", choices = 1)
oden <- reorder(den, x)
par(mfrow=c(2,1))
plot(den)
plot(often)
```



```

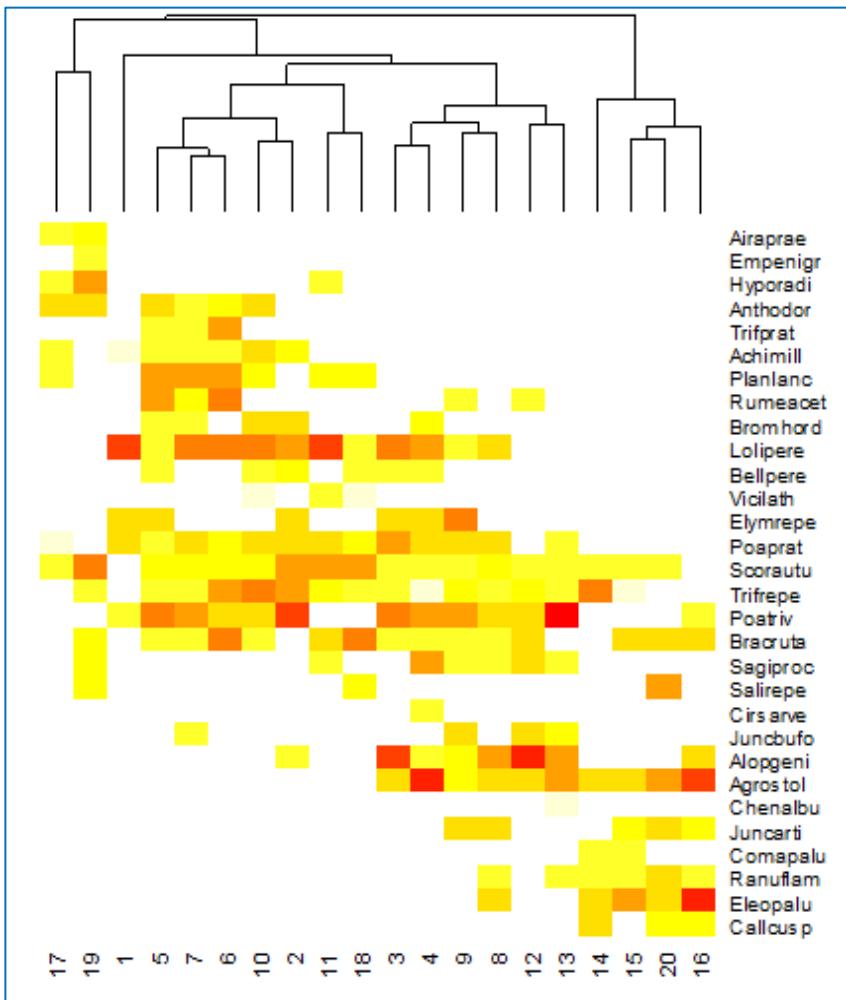
par(mfrow=c(1,1))
vegemite(dune, oden)
##
##                                     1   11      11  111112
##                                     76502183498321794650
##          Trifprat           252. .....
##          Rumeacet           365.....2..2.....
##          PlanLanc            5553.33.....2.....
##          Bromhord            2.244...3.....
##          Achimill             22243.....12.....
##          Vicilath             ...1.21.....
##          Bellpere              ..223.222.....
##          Lolipere              66265726524..7.....
##          Poaprat               432444354442.41.....
##          Anthodor              2344.....44.....
##          Poatriv                54647..6554942..2..
##          Elymrepe              ..4.4..446...4.....
##          Trifrepe              2526532213223..26.1.
##          Cirsarve              .....2.....
##          Scorautu              3333555222322.262.22
##          Juncbufo              2.....4.34.....
##          Bracruta              2622.462222.4..3.444
##          Sagiproc              .....2..52224..3....
##          Alopogeni             ....2..723558....4..
##          Chenalbu              .....1.....
##          Hyporadi              .....2.....25.....
##          Agrostol              .....483454...4745
##          Juncarti              .....44.....334
##          Salirepe              .....3.....3...5
##          Airaprae              .....23.....
##          Empenigr              .....2.....
##          Ranuflam              .....22....2224
##          ELeopalu              .....4.....4854
##          Comapalu              .....2.2.
##          CaLLcusp              .....43.3
##                                     sites       species
##          20     30

```

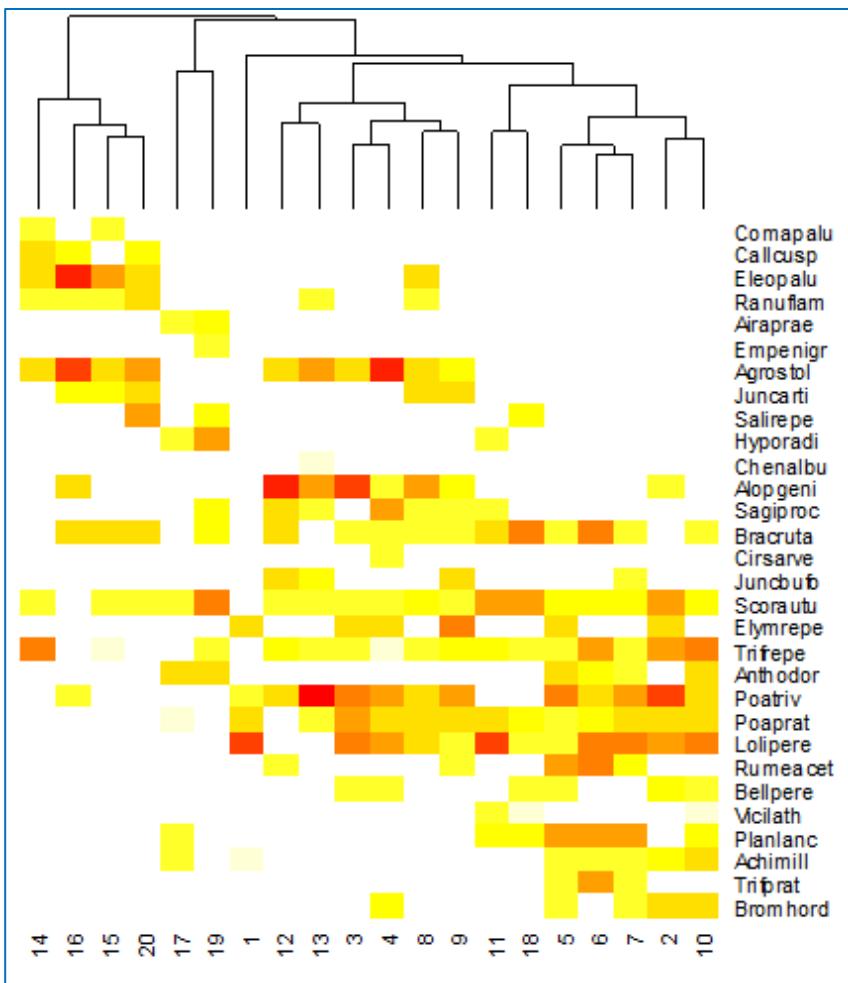
The vegemite function has a graphical sister function tabasco that can also display the dendrogram. Moreover, it defaults to rearrange the dendrogram by the first axis of Correspondence Analysis. Technically it is an interface to R heatmap, but its use is closer to vegan function vegemite. The function can reorder the community data matrix similarly as vegemite, for instance, by ordination results. Unlike heatmap, it only

displays dendrograms if supplied by the user, and it defaults to re-order the dendrograms by correspondence analysis.

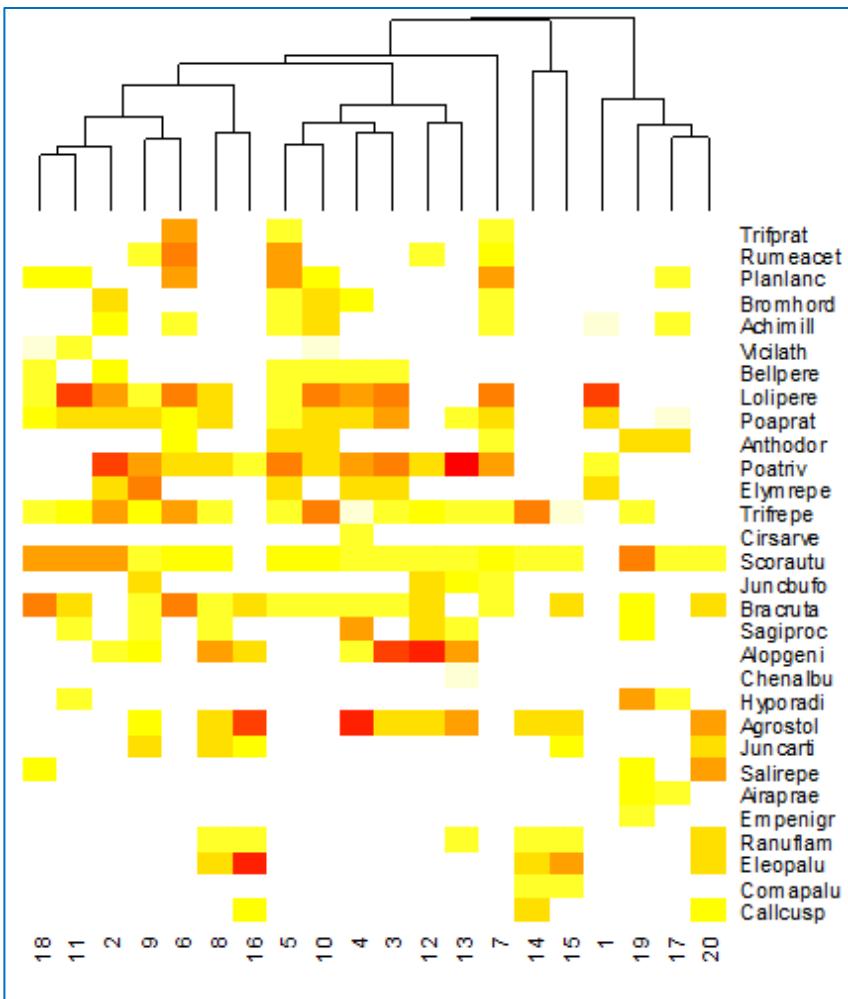
**tabasco(dune, caver)**



**tabasco(dune, caver, Rowv = FALSE)**

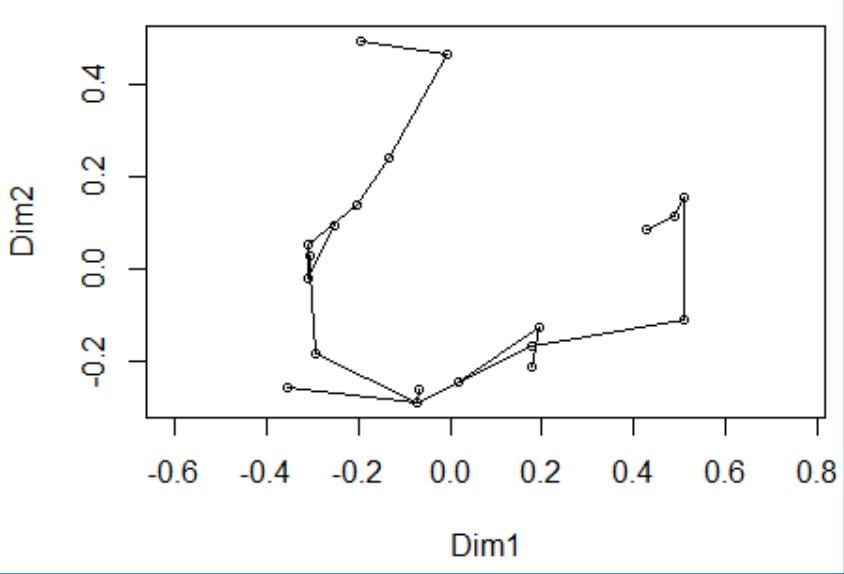


**tabasco(dune, oden, Rowv = FALSE)**



We talked about ordiplot on page 179, but here is some additional information. Function `ordiplot` is an alternative plotting function which can be worked with any vegan ordination result and many non-vegan results. In addition, plot functions for vegan ordinations return invisibly an “ordiplot” result object, and this allows using `ordiplot` support functions with this result: `identify` can be used to add labels to selected site, species or constraint points, and `points` and `text` can add elements to the plot.

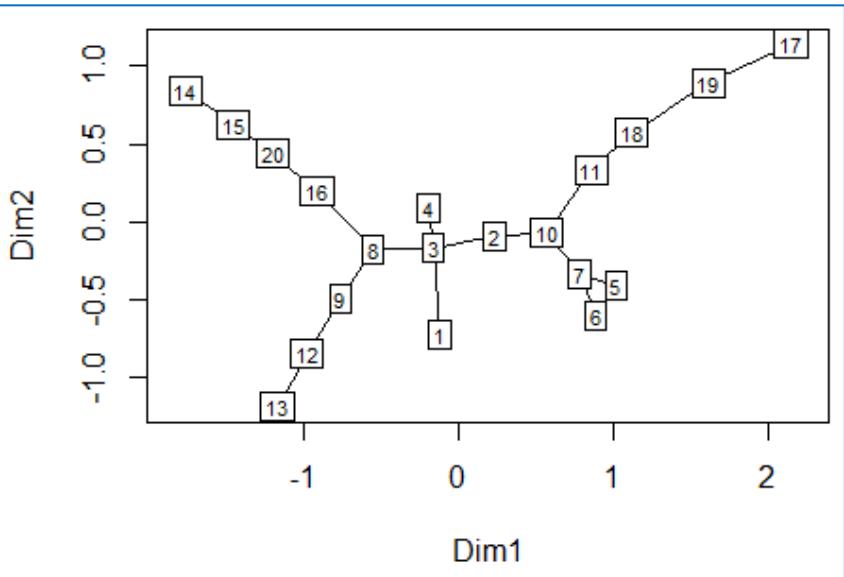
```
mst <- spantree(d)
ordiplot(ord, dis="si")
lines(mst, ord)
```



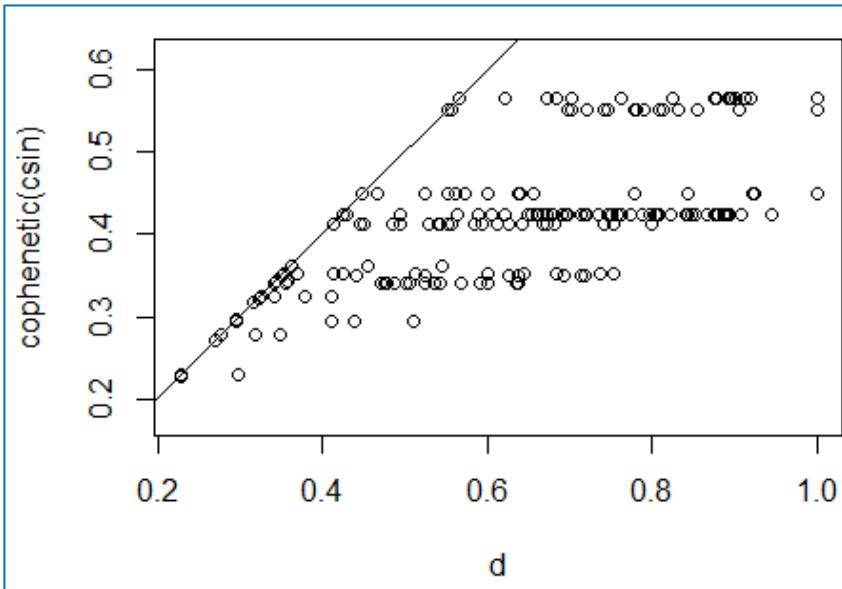
```

plot(mst, type="t")
## Initial stress      : 0.03111
## stress after 10 iters: 0.01302, magic = 0.500
## stress after 20 iters: 0.01139, magic = 0.500
## stress after 30 iters: 0.01118, magic = 0.500
## stress after 40 iters: 0.01114, magic = 0.500

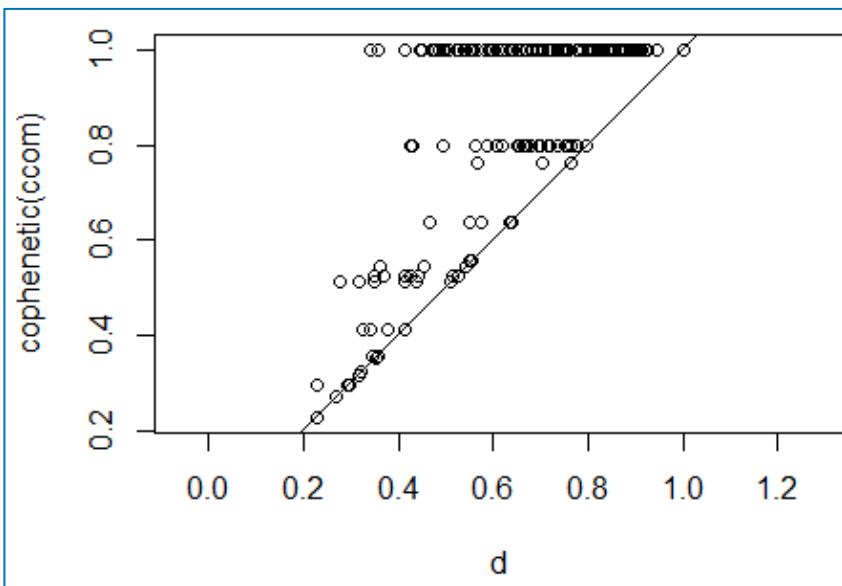
```



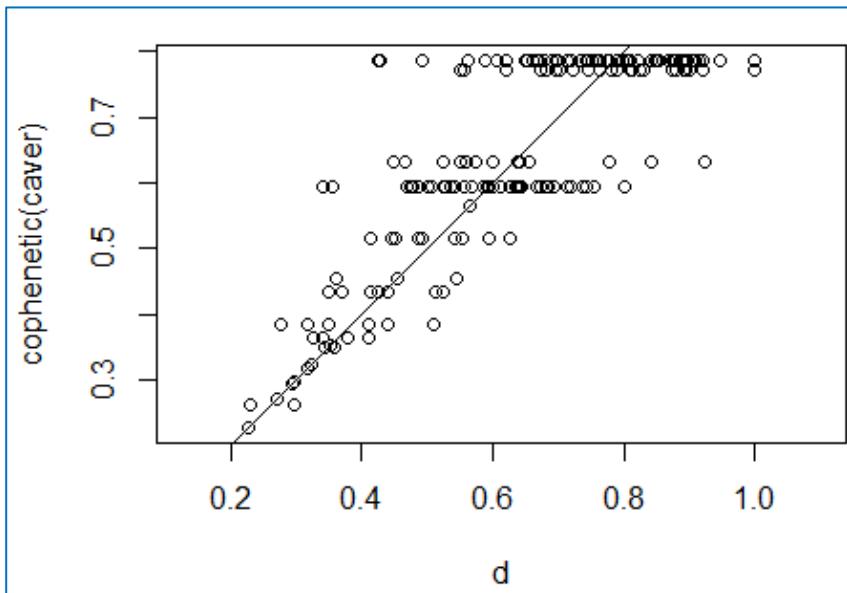
```
plot(d, cophenetic(csin), asp=1)
abline(0, 1)
```



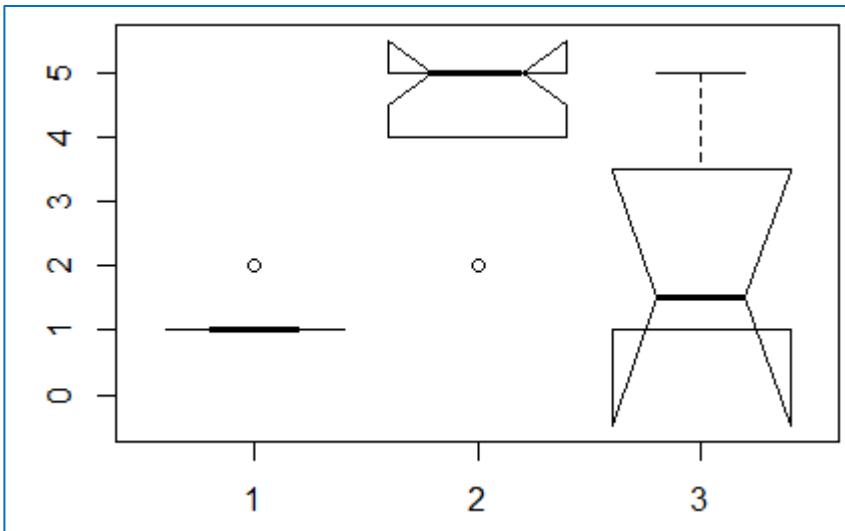
```
plot(d, cophenetic(ccom), asp=1)
abline(0, 1)
```



```
plot(d, cophenetic(caver), asp=1)
abline(0, 1)
```



```
cor(d, cophenetic(csin))
## [1] 0.6601692
cor(d, cophenetic(ccom))
## [1] 0.6707479
cor(d, cophenetic(caver))
## [1] 0.8168825
cl <- factor(cl)
Moist <- with(dune.env, as.numeric(as.character(Moisture)))
)
data(dune.env)
Moist <- with(dune.env, as.numeric(as.character(Moisture)))
)
with(dune.env, as.numeric(Moisture))
## [1] 1 1 2 2 1 1 4 3 2 1 3 4 4 4 4 2 1 4 4
boxplot(Moist ~ cl, notch=TRUE)
## Warning in bxp(list(stats = structure(c(1, 1, 1, 1, 1, 4, 4, 5, 5, 5, 1, :
## some notches went outside hinges ('box'): maybe set notch=FALSE
```



```

anova(lm(Moist ~ cl))
## Analysis of Variance Table
##
## Response: Moist
##             Df Sum Sq Mean Sq F value    Pr(>F)
## cl          2 36.617 18.3083 12.359 0.0004854 ***
## Residuals 17 25.183  1.4814
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.
## 1 ' ' 1
anova(rda(Moist ~ cl))
## Permutation test for rda under reduced model
## Permutation: free
## Number of permutations: 999
##
## Model: rda(formula = Moist ~ cl)
##             Df Variance      F Pr(>F)
## Model      2   1.9272 12.359  0.002 **
## Residual 17   1.3254
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.
## 1 ' ' 1
with(dune.env, table(cl, Management))
##     Management
## cl  BF HF NM SF
## 1   2   3   0   1

```

```

##   2  0  2  3  5
##   3  1  0  3  0
if(!require(labdsv)) install.packages("labdsv")
## Loading required package: labdsv
## Loading required package: mgcv
## Loading required package: nlme
## This is mgcv 1.8-24. For overview type 'help("mgcv-package")'.
## Loading required package: MASS
## Loading required package: cluster
##
## Attaching package: 'labdsv'
## The following object is masked from 'package:stats':
##
##      density
library(labdsv)
const(dune, cl)

##           1   2   3
## Achimill 1.00 0.0 0.25
## Agrostol 0.00 1.0 0.00
## Airaprae 0.00 0.0 0.50
## Allopogeni 0.17 0.7 0.00
## Anthodor 0.67 0.0 0.50
## Bellpere 0.50 0.2 0.25
## Bromhord 0.67 0.1 0.00
## Chenalbu 0.00 0.1 0.00
## Cirsarve 0.00 0.1 0.00
## Comapalu 0.00 0.2 0.00
## Eleopalu 0.00 0.5 0.00
## Elymrepe 0.50 0.3 0.00
## Empenigr 0.00 0.0 0.25
## Hyporadi 0.00 0.0 0.75
## Juncarti 0.00 0.5 0.00
## Juncbufo 0.17 0.3 0.00
## Lolipere 1.00 0.4 0.50
## Planlanc 0.67 0.0 0.75
## Poaprat  1.00 0.5 0.75
## Poatriv  1.00 0.7 0.00
## Ranuflam 0.00 0.6 0.00
## Rumeacet 0.50 0.2 0.00
## Sagiproc 0.00 0.5 0.50
## Salirepe 0.00 0.1 0.50
## Scorautu 0.83 0.9 1.00

```

```

## Trifprat 0.50 0.0 0.00
## Trifrepe 0.83 0.8 0.75
## Viciafaba 0.17 0.0 0.50
## Bracruta 0.67 0.8 0.75
## Callcusp 0.00 0.3 0.00

importance(dune, cl)
##          1     2     3
## Achimill 2.33 0.00 2.00
## Agrostol 0.00 4.80 0.00
## Airaprae 0.00 0.00 2.50
## Alopogeni 2.00 4.86 0.00
## Anthodor 3.25 0.00 4.00
## Bellpere 2.33 2.00 2.00
## Bromhord 3.00 3.00 0.00
## Chenalbu 0.00 1.00 0.00
## Cirsarve 0.00 2.00 0.00
## Comapalu 0.00 2.00 0.00
## Eleopalu 0.00 5.00 0.00
## Elymrepe 4.00 4.67 0.00
## Empenigr 0.00 0.00 2.00
## Hyporadi 0.00 0.00 3.00
## Juncarti 0.00 3.60 0.00
## Juncbufo 2.00 3.67 0.00
## Lolipere 5.33 4.25 4.50
## Planlanc 4.50 0.00 2.67
## Poaprat 3.50 3.80 2.67
## Poatriv 4.67 5.00 0.00
## Ranuflam 0.00 2.33 0.00
## Rumeacet 4.67 2.00 0.00
## Sagiproc 0.00 3.00 2.50
## Salirepe 0.00 5.00 3.00
## Scorautu 3.40 2.11 4.50
## Trifprat 3.00 0.00 0.00
## Trifrepe 4.00 2.50 2.33
## Viciafaba 1.00 0.00 1.50
## Bracruta 3.00 3.00 4.33
## Callcusp 0.00 3.33 0.00

mod <- indval(dune, as.numeric(cl))
names(mod)
## [1] "relfrq" "relabu" "indval" "maxcls" "indcls" "pval"
## "error"

summary(mod)
##           cluster indicator_value probability
## Achimill           1             0.8235      0.001

```

```

## Bromhord      1      0.5797      0.021
## Lollipop     1      0.5745      0.027
## Poatriv      1      0.5714      0.014
## Trifprat     1      0.5000      0.032
## Agrostol      2      1.0000      0.001
## Alopogeni     2      0.6375      0.014
## Ranuflam      2      0.6000      0.013
## Hyporadi       3      0.7500      0.003
## Airaprae       3      0.5000      0.033
## Scorautu      3      0.4874      0.027
##
## Sum of probabilities =  6.199
##
## Sum of Indicator Values = 13.19
##
## Sum of Significant Indicator Values = 7.02
##
## Number of Significant Indicators = 11
##
## Significant Indicator Distribution
##
## 1 2 3
## 5 3 3
summary(mod, type = "long")
##          1    2    3
## Achimill 0.82  .
## Agrostol  .    1.00  .
## Airaprae  .    .    0.50
## Alopogeni .    0.64  .
## Anthodor 0.35  .    0.24
## Bellpere 0.28  .    0.06
## Bromhord 0.58  .    .
## Chenalbu   .    0.10  .
## Cirsarve  .    0.10  .
## Comapalu  .    0.20  .
## Eleopalu  .    0.50  .
## Elymrepe  0.29  0.12  .
## Empenigr  .    .    0.25
## Hyporadi  .    .    0.75
## Juncarti  .    0.50  .
## Juncbufo  .    0.23  .
## Lollipop  0.57  0.07  0.12
## Planlanc  0.40  .    0.30
## Poaprat   0.47  0.13  0.20
## Poatriv   0.57  0.30  .

```

```

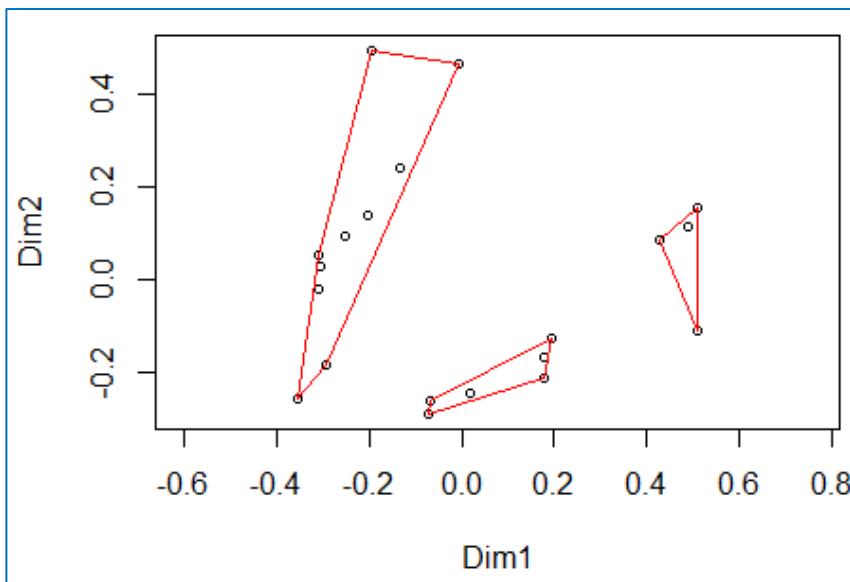
## Ranuflam   .  0.60  .
## Rumeacet  0.43  .
## Sagiproc   .  0.27  0.23
## Salirepe   .  .  0.38
## Scorautu  0.26  0.19  0.49
## Trifprat  0.50  .
## Trifrepe   0.39  0.23  0.19
## Vicilath   .  .  0.41
## Bracruta   0.17  0.25  0.32
## Callcusp   .  0.30  .

ckm <- kmeans(decostand(dune, "hell")), 3)
ckm$cluster

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 1
9 20
## 1 1 3 3 1 1 1 3 3 1 1 3 3 2 2 2 1 1
1 2

ordiplot(ord, dis="si")
ordihull(ord, ckm$cluster, col="red")

```

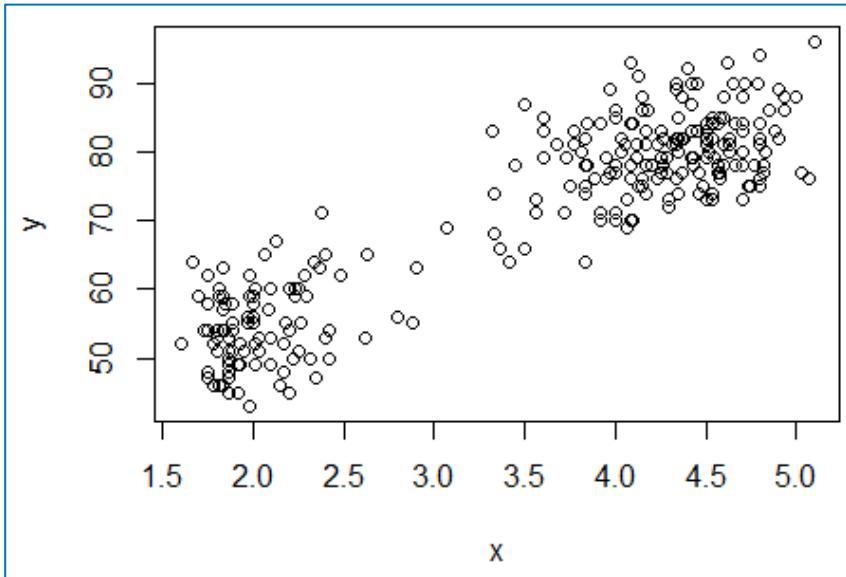


Mclust is a function in the mclust-package. It is used for model-based clustering based on parameterized finite Gaussian mixture models. Models are estimated by Expectation-Maximization (EM) algorithm initialized by hierarchical model-based agglomerative clustering. The optimal model is then selected according to BIC.

```

if(!require(mclust)) install.packages("mclust")
## Loading required package: mclust
## Package 'mclust' version 5.4.1
## Type 'citation("mclust")' for citing this R package in
publications.
##
## Attaching package: 'mclust'
## The following object is masked from 'package:mgcv':
##
##      mvn
library(mclust)           # Load mclust library
x = faithful[,1]           # get the first column of the fa
ithful data set
y = faithful[,2]           # get the second column of the f
aithful data set
plot(x,y)                 # plot the spread points before
the clustering

```

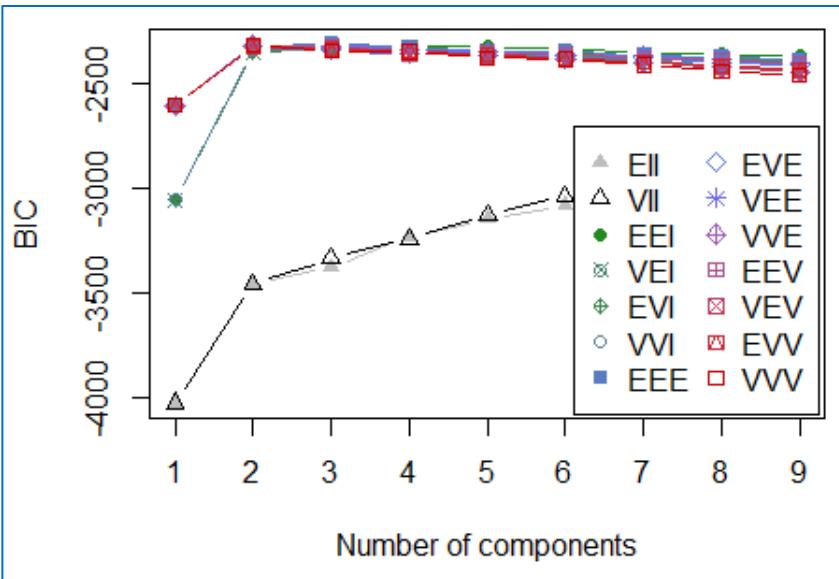


Here, we estimate the number of cluster (BIC), initialize (HC) and clusterize (EM)

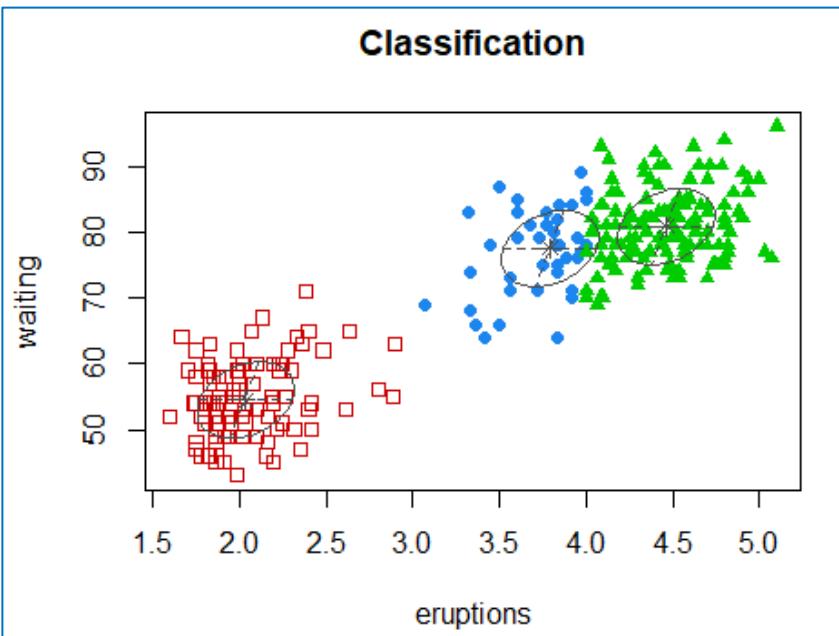
```

model <- Mclust(faithful)
data = faithful                      # get the data set
plot(model)                          # plot the clustering results

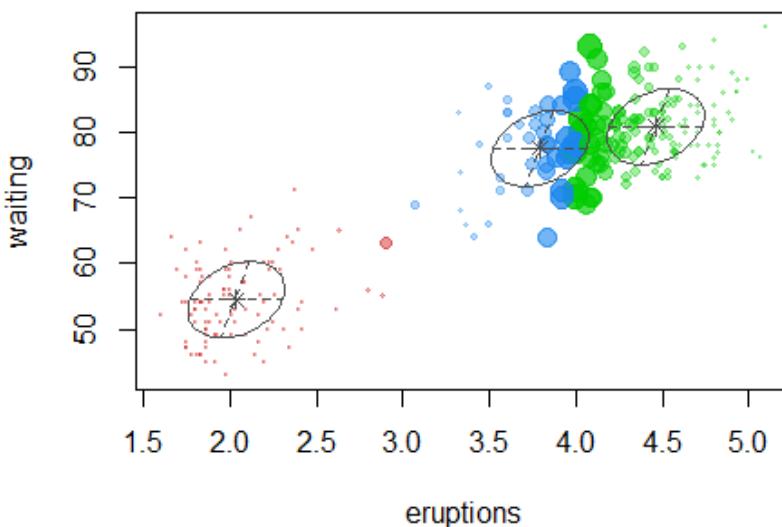
```



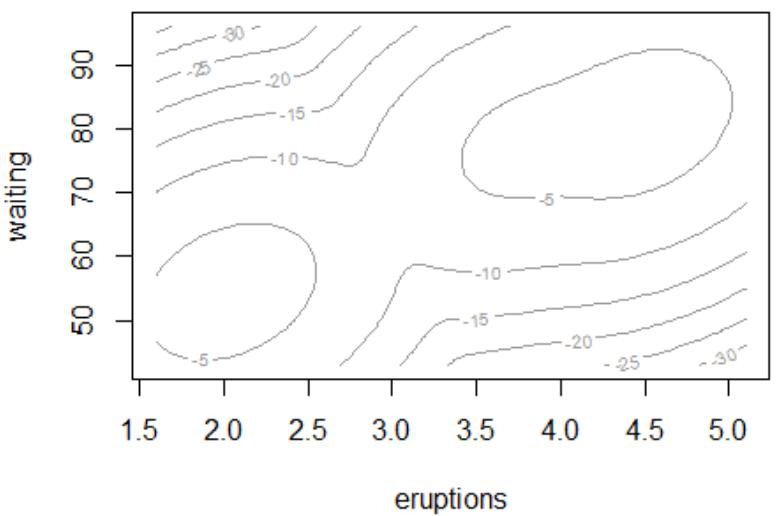
We start seeing diminishing gains after 3 components, so we will see three clusters in our plot.



## Uncertainty



## log Density Contour Plot



## Exercises

1. Download USAarrest.csv from Kaggle at <https://www.kaggle.com/deepakg/usarrests>
  - a. Scale the data.
  - b. Construct a distance matrix.
  - c. Perform a hierarchical cluster using hclust and the Ward.D method.
  - d. Display the dendrogram.
  - e. How many clusters do you see?
  - f. Draw dendrogram with red borders around the clusters.
  - g. Use the elbow method to determine the number of clusters.
2. Perform a k-means cluster analysis of USAarrest
  - a. Use optimal number of clusters in k-means and get cluster means
  - b. Create a Cluster Plot against first 2 principal components and vary parameters for most readable graph
3. Perform Model-Based clustering on the USAarrest data.
  - a. Perform your fit using Mclust and print the summary.
  - b. Show the Bayesian Information Criterion (BIC)
  - c. Make a cluster plot using clustplot.
  - d. Get the cluster means.
4. Download OfferInformation.csv and transactions.csv from Kaggle at <https://www.kaggle.com/alex243/dataset>.
  - a. Read offers and transaction data into R and view the datasets.
  - b. Using the ClusterR, cluster, reshape and/or reshape2, construct a pivot table using CustomerLastName as the ID variable.
  - c. Compute all the pairwise dissimilarities (distances) between observations in the data set [Hint: use daisy].
  - d. Develop a hierarchical clustering, i.e., using hclust.
  - e. Cut the tree into four clusters.

- f. Print a dendrogram and a cluster plot.
5. In the clustering data set from Exercise 4, rows represents costumers and columns are different wine brands/types.
- a. Use 4 clusters to build a hierarchical clustering. (This is somewhat arbitrary, but the number you pick should be representative of the number of segments you can handle as a business.)
  - b. Calculate how far away each customer is from the cluster mean. [Hint: use many distances/dissimilarity indices, one of which is the Gower dissimilarity.]
  - c. Draw dendrogram with red borders around the 4 clusters.
  - d. After the creation of a distance matrix, implement a Wards hierarchical clustering procedure (The “ward” method has been renamed to “ward.D”; note new “ward.D2”).
  - e. Render a 2D representation of the Segmentation, e.g., using clusplot.
6. Using your work in Exercise 5, get the top deals which require a little bit of data manipulation.
- a. Combine our clusters and transactions. [Hint: the lengths of the ‘tables’ holding transactions and clusters are different.]
  - b. Use the merge() function and give our columns sensible names:
  - c. Get top deals by cluster
  - d. Repeat the pivoting process to get Offers in rows and clusters in columns counting the total number of transactions for each cluster.
  - e. Once you have the pivot table, merge it with the offers data table like you did before. [Hint: use melt and cast.]
  - f. Plot the top-deal clusters.
  - g. What are the top deals by offer and name?

7. Perform the following step using the iris dataset.
  - a. Build a dendrogram for the irid dataset using hclust.
  - b. Transform a dendrogram into a ggdend object using as.ggdend() function. [Hint: see Beautiful dendrogram visualizations in R, at <http://www.sthda.com/english/wiki/beautiful-dendrogram-visualizations-in-r-5-must-known-methods-unsupervised-machine-learning>.]
  - c. Make the plot using the function ggplot().

# CHAPTER 4 – Crime Linkage and Unsolved Crimes

## Introduction to Crime Series Linkage

Statistical linkage and clustering of criminal events can be used by crime analysts to create lists of potential suspects for an unsolved crime, identify groups of crimes that may have been committed by the same individuals or group of individuals, for offender profiling, and for predicting future events. Pairwise case linkage attempts to establish if a pair of crimes share a common offender. In practice, there is more interest in **crime series linkage**, which attempts to identify the set of crimes committed by a common offender. For example, a crime analyst may need to identify the additional crimes that are part of a crime series (crime series identification) or discover all the crime series in a criminal incident database (crime series clustering).

**Crime series clustering** is fundamentally a clustering problem where we seek to group the crimes into clusters that correspond to a serial offender (or group of offenders). Instead of simultaneously clustering all crimes in a criminal database, **Crime series identification** is focused on identifying the additional unsolved crimes that are part of an existing crime series. Crime series identification can be useful, for example, in interrogations [ @Adderly-Musgrove-2003; @Adderly-2004] by providing investigators a list of additional crimes (like the known crimes) that a suspect may be responsible for. A crime analyst could also use crime series identification when investigating a crime series or suspected crime series to generate a list of additional crimes to jointly investigate.

The **crimelinkage** package provides several tools for crime series identification and clustering based on methods from hierarchical and model-based clustering.

## Trial Run with **crimeLinkage**

It is a good idea to open the help file for fitting with Bayesian model-based partially-supervised clustering for reference, so we do it in the code, lest we forget. `help("crimeClust_bayes")` Now, we will run the `crimeLinkage` documentation example for fitting with Bayesian model-based partially-supervised clustering. This will familiarize you with the function and its parameters.

**Make IDs: Criminal 1 committed crimes 1-4, etc.**

```
id <- c(1,1,1,1,  
       2,2,2,2,  
       3,3,3,3)
```

**Spatial locations of the crimes:**

```
s <- c(0.8,0.9,1.1,1.2,  
      1.8,1.9,2.1,2.2,  
      2.8,2.9,3.1,3.2)  
s <- cbind(0,s)
```

We can categorical crime features, say mode of entry (1=door, 2=other) and type of residence (1=apartment, 2=other). # Different distribution by criminal

```
Mode <- c(1,1,1,1,  
        1,2,1,2,  
        2,2,2,2)
```

**Same distribution for all criminals**

```
Type <- c(1,2,1,2,  
        1,2,1,2,  
        1,2,1,2)  
Xcat <- cbind(Mode,Type)
```

**Times of the crimes**

```
t <- c(1,2,3,4,  
      2,3,4,5,  
      3,4,5,6)
```

Now let's pretend we don't know the criminal for crimes 1, 4, 6, 8, and 12.

```
id <- c(NA,1,1,NA,2,NA,2,NA,3,3,3,NA)
```

Then, we fit the model (naïve Bayes, NB, use much larger iterations and burn on a real problem)

```
fit1 <- crimeClust_bayes(crimeID=id, spatial=s, t1=t,t2=t,  
Xcat=Xcat,maxcriminals=12,iters=500,burn=100,update=100)  
# Plots are omitted
```

```

summary(fit1)
##          Length Class  Mode
## p.equal     144   -none- numeric
## D           500   -none- numeric
## df          1000  -none- numeric
## sd1         500   -none- numeric
## sd2         500   -none- numeric
## sds         1000  -none- numeric
## theta        500   -none- numeric
## s.miss       0    -none- NULL
## t.censored   0    -none- NULL
## missing_s    0    -none- numeric
## missing_t    0    -none- numeric
## crimeID      12   -none- numeric

```

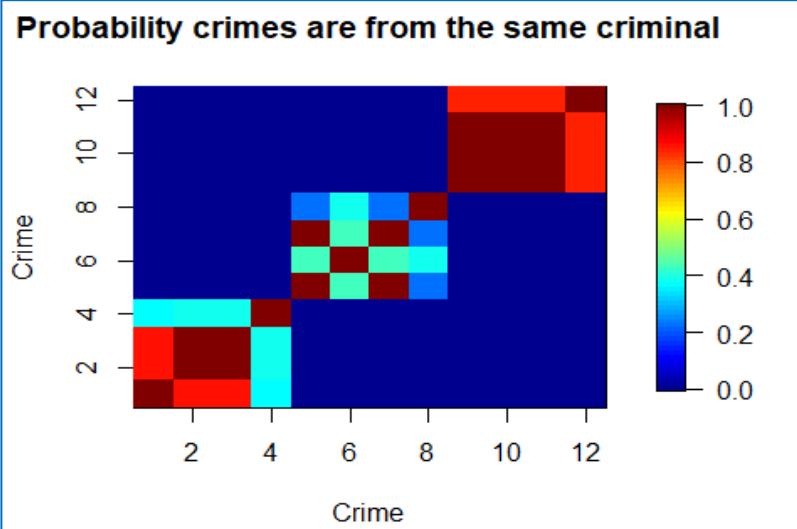
Now we plot the posterior probability matrix in which each pair of crimes was contained and view the model summary.

### Committed by the same criminal:

```

if(require(fields,quietly=TRUE)) {
  fields::image.plot(1:12,1:12,fit1$p.equal,
  xlab="Crime", ylab="Crime",
  main="Probability crimes are from the same criminal")
}

```



The “heatmap” plot above shows the probability that there is a linkage between unsolved and solved crime. The “hot” spots (shades of red) have a very high chance of linkage and can likely be solved.

## Preliminaries

### *Using crimeLinkage for Analyzing Unsolved Crimes*

We should first check to see if the package is installed using:

```
installed.packages("crimeLinkage")
```

```
Package    LibPath    Version    Priority    Depends    Imports
LinkingTo  Suggests    Enhances    License
License_is_FOSS  License_restricts_use  OS_type    Archs
MD5sum    NeedsCompilation  Built
```

The output above shows use that `crimeLinkage` is installed. If it were not, we would install the package using:

```
install.packages("crimeLinkage")
```

Next, we need to load the package and get the example crime data.

### *Load Libraries*

```
library(crimeLinkage)
library(fields)
```

### *Load the Data*

```
data(crimes)
data(offenders)
head(crimes)

##   crimeID      X      Y M01 M02 M03           DT.FROM
## 1 C:1 13661.1 -4659.3   6    a    J 1993-01-06 23:55:00
## 2 C:2  6758.8 -10092.4  25    a    E 1993-01-06 00:00:00
## 3 C:3 10077.6 -8810.9  25    a    E 1993-01-08 07:00:00
## 4 C:4 12080.3 -4011.1  25    a    E 1993-01-10 20:27:00
## 5 C:5 10862.5 -8091.0  19    b    C 1993-01-11 07:45:00
## 6 C:6 14032.9 -1945.4   7 <NA> <NA> 1993-01-04 15:30:00
##
##           DT.TO
## 1 1993-01-07 04:39:00
## 2 1993-01-06 00:00:00
## 3 1993-01-08 17:30:00
## 4 1993-01-10 20:27:00
## 5 1993-01-11 07:45:00
## 6 1993-01-04 15:30:00
head(offenders)

##   offenderID crimeID
## 1          0:1      C:6
```

```

## 2      0:2    C:2
## 3      0:3    C:29
## 4      0:4    C:19
## 5      0:5    C:18
## 6      0:6    C:18

```

## Data Description

- crimeID: The crime ID number
- X,Y: Spatial
- Coordinates MO1 A categorical
- MO1: A variable that takes values 1,...,31
- MO2: A categorical MO variable that takes values a,...,h
- MO3: A categorical MO variable that takes values A,...,O
- DT.FROM: The earliest possible Date-time of the crime.
- DT.TO: The latest possible Date-time of the crime

## Setup the Data for Analysis

### *Make a Crime Series*

```

seriesData = makeSeriesData(crimesdata=crimes,offenderTable
=offenders)
head(seriesData)

##   crimeID Index    CS  offenderID             TIME
## 1          C:6     6      1  0:1 1993-01-04 15:30:00
## 2          C:9     9      2  0:10 1993-01-11 13:00:00
## 3         C:121    121     3  0:100 1993-06-02 03:14:00
## 4         C:127    127     4  0:101 1993-06-04 16:32:00
## 5         C:100    100     5  0:102 1993-05-17 00:00:00
## 6         C:94     94     6  0:103 1993-05-10 13:50:00

```

### *Make Crime Pairs for Case Linkage*

```

set.seed(1)           # set random seed for replication
allPairs = makePairs(seriesData,thres=365,m=40)

```

### *Make Evidence Variables for Case Linkage*

```

set.seed(1)           # set random seed for replication
allPairs = makePairs(seriesData,thres=365,m=40)
varnames = list(spatial = c("X", "Y"), temporal = c("DT.FR
OM","DT.TO")), categorical = c("MO1", "MO2", "MO3"))
X = compareCrimes(allPairs, crimes, varnames, binary=TRUE)
# Evidence data

```

```

Y = ifelse(X$type=='linked',1,0)      # Linkage indicator.
1=linkage, 0=unlinked

```

## Build Training and Testing Data

```

set.seed(3) # set random seed for replication
train = sample(c(TRUE,FALSE),nrow(X),replace=TRUE,prob=c(.7,.3)) # assign pairs to training set
test = !train
D.train = data.frame(X[train,],Y=Y[train]) # training data
D.test = data.frame(X[test,],Y=Y[test]) # training data

```

## Fit Logistic Regression model and make estimateBF() function

```

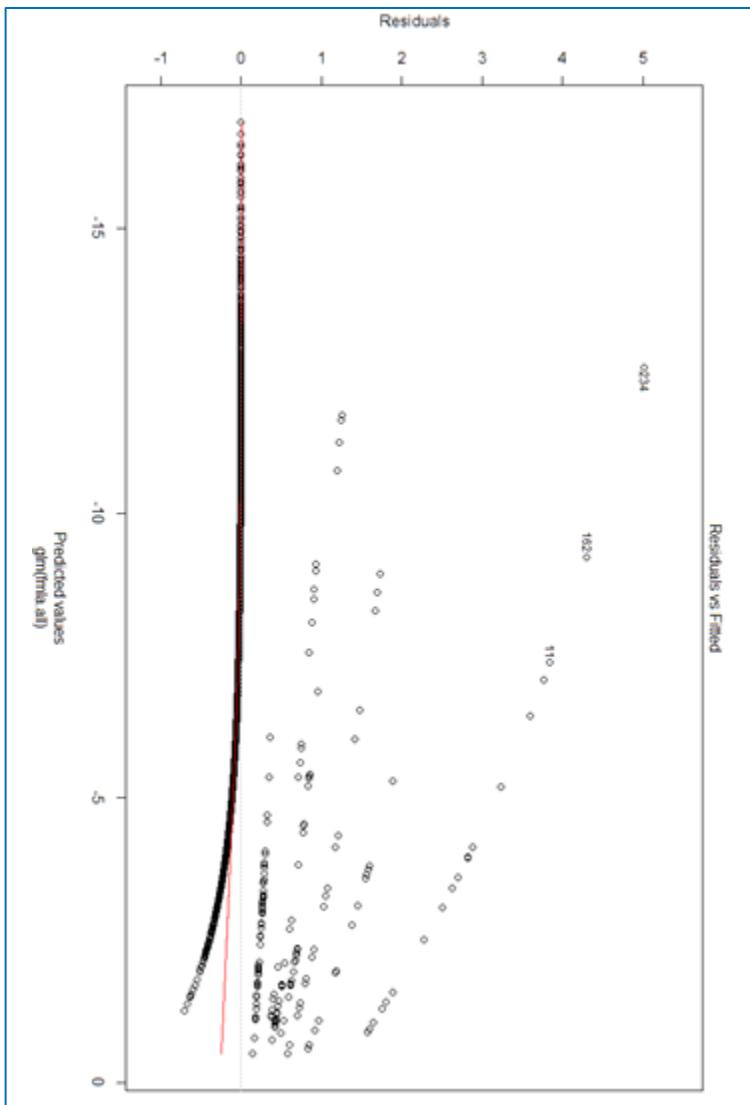
spatial = c("X", "Y")
temporal = c("DT.FROM","DT.T0")
categorical = c("M01", "M02", "M03")
vars = c("spatial","temporal","tod","dow","M01","M02","M03")
fmla.all = as.formula(paste("Y ~ ", paste(vars, collapse="+")))
fmla.all
## Y ~ spatial + temporal + tod + dow + M01 + M02 + M03
fit.logistic = glm(fmla.all,data=D.train,family=binomial,weights=weight)
## glm!
summary(fit.logistic)
##
## Call:
## glm(formula = fmla.all, family = binomial, data = D.train, weights = weight)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max 
## -0.7063   -0.0676   -0.0300   -0.0117    5.0136 
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -2.013335  0.605822 -3.323  0.00089 ***
## spatial     -0.450665  0.086470 -5.212 1.87e-07 ***
## temporal    -0.016284  0.003922 -4.152 3.29e-05 ***
## tod          -0.128988  0.065646 -1.965  0.04943 *  
## dow          -0.149256  0.183872 -0.812  0.41694    
## M01          1.507261  0.370704  4.066 4.78e-05 ***
```

```
## M021      0.278068  0.364385  0.763  0.44539
## M031      0.081398  0.412758  0.197  0.84367
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.
##   ' ' 1
##
## (Dispersion param for binomial family taken to be 1)
##
## Null deviance: 424.84  on 7900  degrees of freedom
## Residual deviance: 308.86  on 7893  degrees of freedom
## AIC: 242.59
##
## Number of Fisher Scoring iterations: 10
estimateBF <- function(X){ # estimateBF() returns the estimated log Bayes factor
  predict(fit.logistic,X)
}
est <- estimateBF(D.test)
```

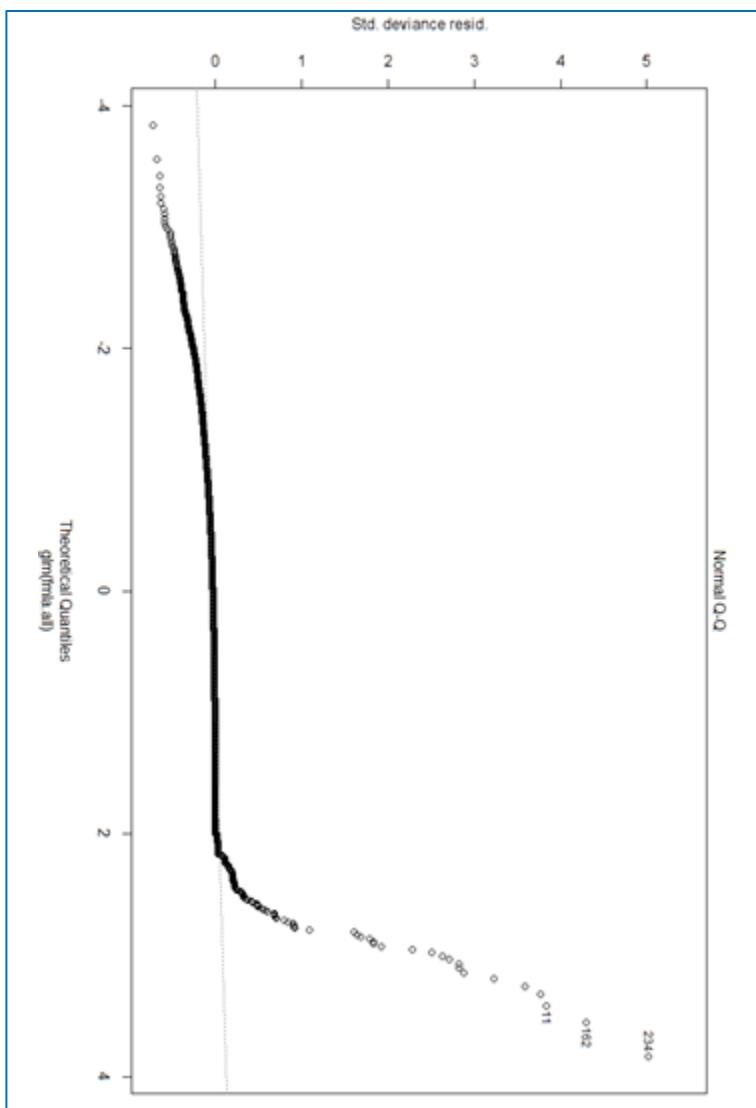
### **Plot the Results**

```
plot(fit.logistic)
plot(est)
```

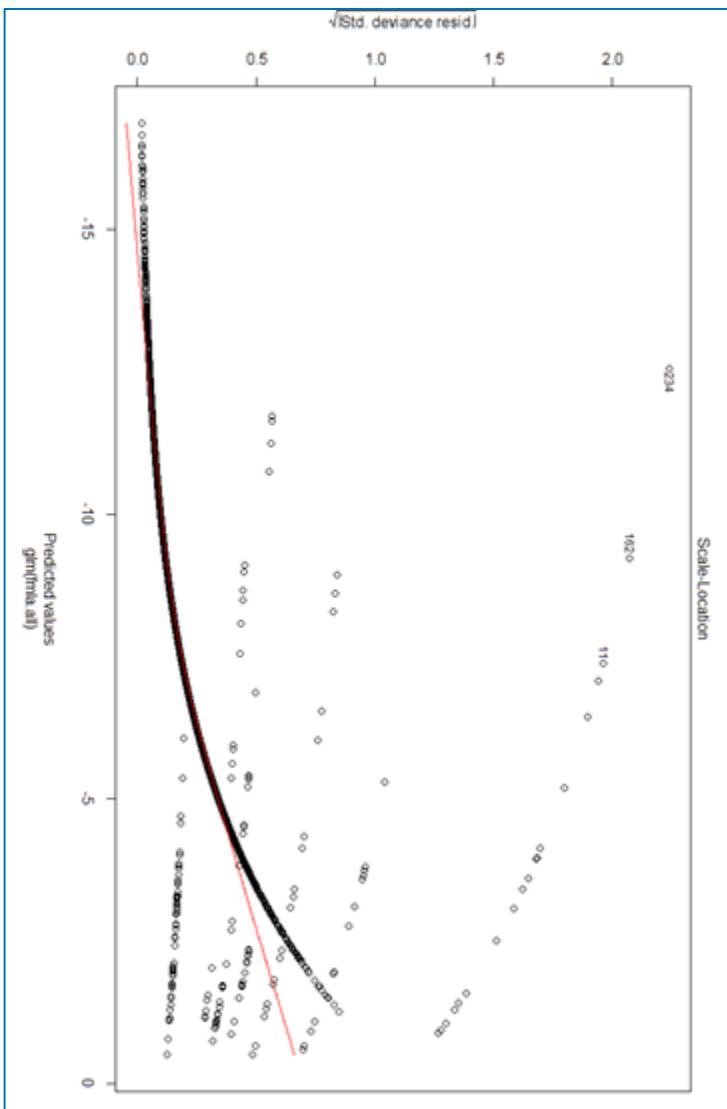
This is the plot of residual versus fitted predicted values for the logistic regression model.



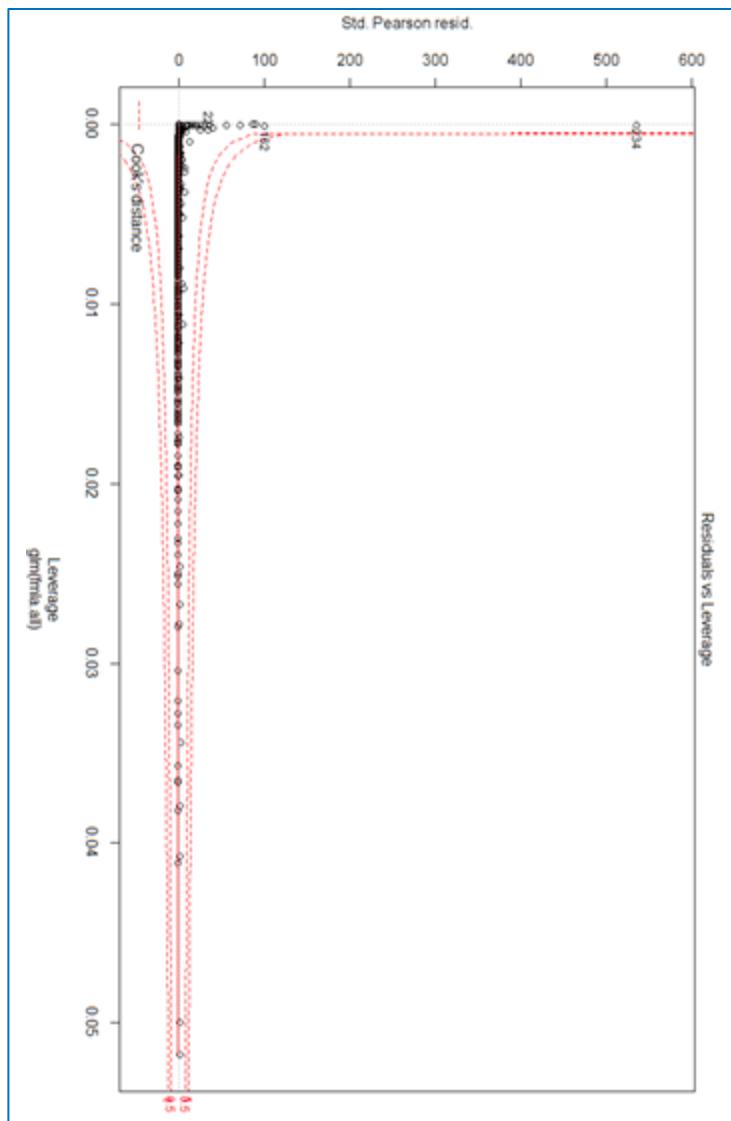
The next plot is the Normal QQ plot of theoretical quantiles for the logistic regression model.



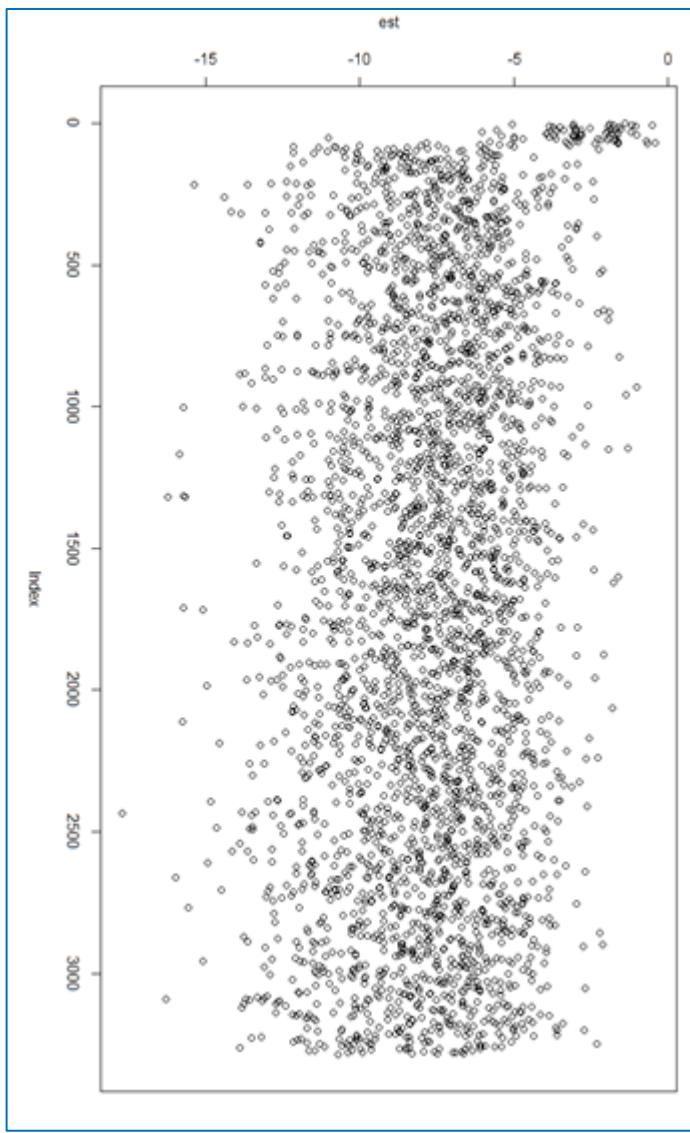
This is the Scale-location plot of predicted values for the logistic regression model.



This next graph is the Residuals versus leverage plot for the logistic regression model.



The next graph is the plot of estimated values for the logistic regression model.



### Examine the Factors

```
fit.logistic$R
##           (Intercept)    spatial    temporal
tod          dow
## (Intercept) -5.55762 -14.04891 -256.02661 -26.226672
781 -8.97039066
## spatial      0.00000  11.70624   11.51314   0.609282
777  0.21928296
```

```

## temporal      0.00000  0.00000 255.52427 -0.004565
678  0.08918358
## tod          0.00000  0.00000  0.00000 15.310088
738  0.17031564
## dow          0.00000  0.00000  0.00000 0.000000
000 -5.44614620
## M011         0.00000  0.00000  0.00000 0.000000
000  0.00000000
## M021         0.00000  0.00000  0.00000 0.000000
000  0.00000000
## M031         0.00000  0.00000  0.00000 0.000000
000  0.00000000
##                  M011           M021           M031
## (Intercept) -3.00844646 -2.859329604 -1.46245700
## spatial      -0.39530917  0.008579316 -0.02068826
## temporal     -0.15428202  0.040069342 -0.04939764
## tod          -0.20798744  0.086261012  0.09196271
## dow          0.04021707 -0.110916385  0.06374548
## M011         -2.72838851 -0.329903354 -0.24952803
## M021         0.00000000 -2.754095684 -0.20432278
## M031         0.00000000  0.000000000  2.42272556

```

### *Prediction using the Testing Data*

```

pred <- predict.glm(fit.logistic, D.test)

summary(pred)
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## -17.7122 -9.3407 -7.5364 -7.6950 -5.9541 -0.4169

```

### **Fit naive Bayes model and make estimateBF() function**

```

vars = c("spatial","temporal","tod","dow","M01","M02","M03")
fmla.all = as.formula(paste("Y ~ ", paste(vars, collapse=
"")))
fmla.all

## Y ~ spatial + temporal + tod + dow + M01 + M02 + M03

NB = naiveBayes(fmla.all,data=D.train,weights=weight,df=10
,nbins=15,partition='quantile')

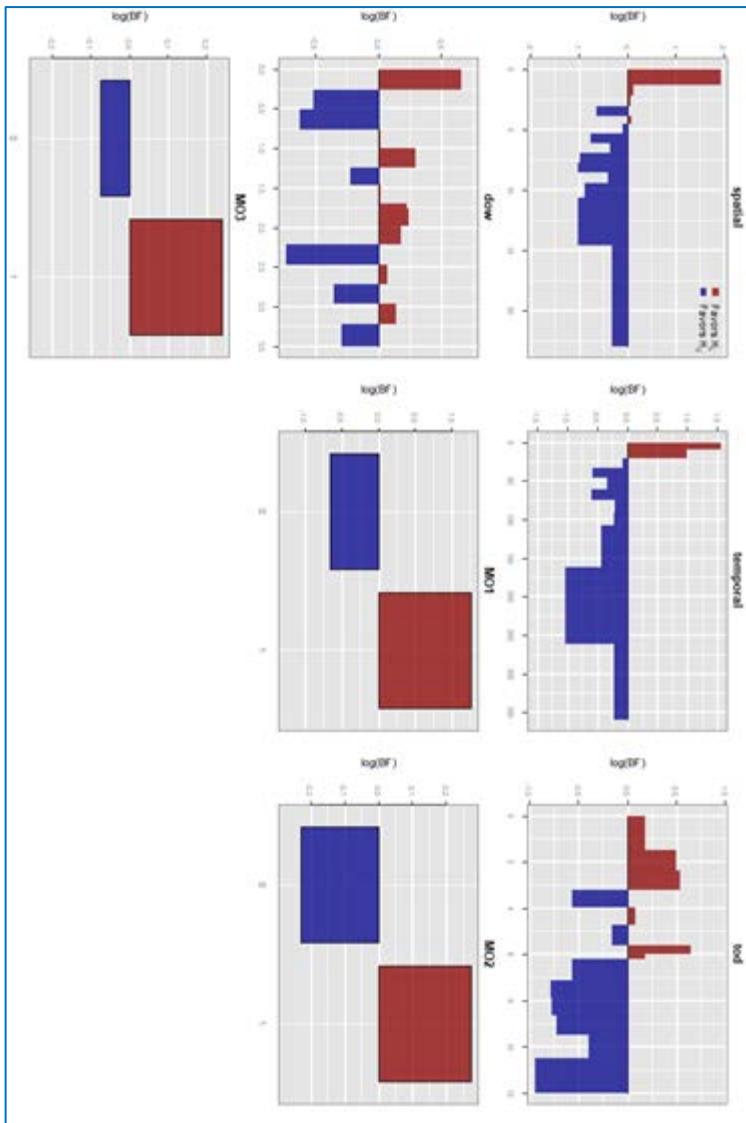
estimateBF <- function(X){# estimateBF() returns the estimated Log Bayes factor
  predict(NB,newdata=X)
}

```

## Plot Results

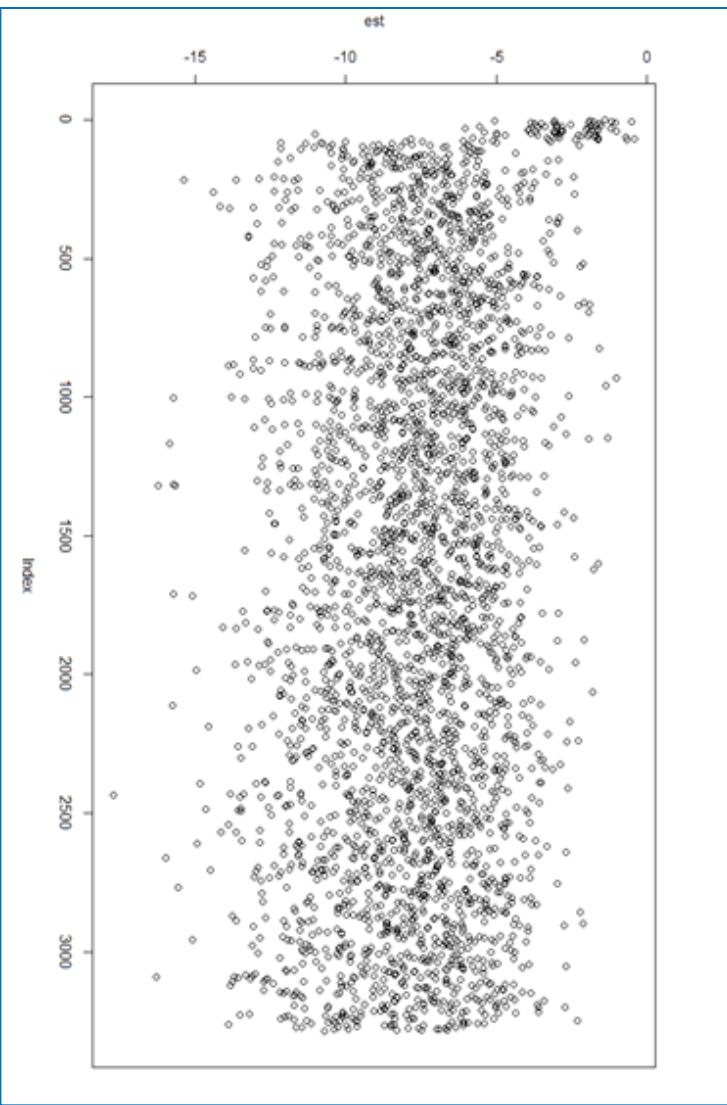
The log-scale plots of factors for the naïve Bayes model

```
plot(NB)
```



The plots of estimated values for the naïve Bayes model

```
plot(est)
```



### Print Factors

```
NB$spatial(head,6)
##           from         to      value N.linked N.unlinked
## 1  0.000000  1.330002 [0,1.33] 18.92490842     408
## 2  1.330002  2.236050 (1.33,2.24]  2.58791209     515
## 3  2.236050  3.111460 (2.24,3.11]  2.35531136     519
## 4  3.111460  3.854694 (3.11,3.85]  0.57765568     522
```

```

## 5 3.854694 4.568613 (3.85,4.57] 2.47692308 516
## 6 4.568613 5.365327 (4.57,5.37] 1.87765568 520
## p.linked p.unlinked BF
## 1 0.39386082 0.05777996 6.8165648
## 2 0.07441269 0.06668887 1.1158187
## 3 0.06986449 0.06702191 1.0424127
## 4 0.03510481 0.06727170 0.5218362
## 5 0.07224245 0.06677213 1.0819251
## 6 0.06052457 0.06710517 0.9019360

NB$temporal(head,6)

##          from          to      value N.linked
## 1 0.01052083 9.936227 [0.0105,9.94] 13.3608059
## 2 9.93622685 21.243403 (9.94,21.2] 7.5739927
## 3 21.24340278 33.065972 (21.2,33.1] 1.8457875
## 4 33.06597222 46.466667 (33.1,46.5] 0.6692308
## 5 46.46666667 60.875926 (46.5,60.9] 1.2172161
## 6 60.87592593 75.062500 (60.9,75.1] 0.6428571

## N.unlinked p.linked p.unlinked BF
## 1 451 0.28506223 0.06136017 4.6457207
## 2 486 0.17190883 0.06427430 2.6746121
## 3 504 0.05990143 0.06577300 0.9107298
## 4 515 0.03689544 0.06668887 0.5532473
## 5 521 0.04761056 0.06718843 0.7086124
## 6 518 0.03637974 0.06693865 0.5434788

NB$M01

##      value N.linked N.unlinked p.linked p.unlinked
## 1 0 14.37253       6482 0.4371664 0.8395286
## 2 1 18.50403       1239 0.5628336 0.1604714
## BF
## 1 0.5207285
## 2 3.507375

1NB$M02$BF
## [1] 0.797155 1.312511

```

### *Agglomerative Hierarchical Crime Series Clustering*

Hierarchical clustering is an algorithmic approach to crime series cluster analysis that sequentially forms a hierarchy of cluster solutions. The

agglomerative approach starts with every observation (e.g., crime incident) in its own cluster. Then it sequentially merges the two closest clusters to form a new larger cluster. This process is repeated until all observations are in the same cluster or a stopping criterion is met.

This algorithm requires two similarity measures to be specified: the pairwise similarity between two observations and the similarity between two groups of observations. There are three primary approaches to measuring the similarity between groups of observations. *Single linkage*, or nearest neighbor, uses the most similar pair between the two groups as the group similarity measure. In contrast, *complete linkage* uses the least similar pair between two groups as the measure of group similarity. *Average linkage* uses the average similarity between all pairs in the two groups.

The `crimeLinkage` package provides the function `crimeClust_hier()` for agglomerative hierarchical crime clustering. It uses the log Bayes factor as the pairwise similarity measure. That is, the similarity between crimes  $i$  and  $j$  is  $S(i,j) = \log BF(i,j)$ , where  $BF(i,j)$  is the estimated Bayes factor for linkage. Then one of: *average*, *single*, or *complete* linkage is used for the similarity between groups.

In this example, we will cluster all of the unsolved crimes from `crimes` data and display dendrogram of results with `plot_hcc()` function.

## Get unsolved crimes

```
unsolved = subset(crimes, !crimeID %in% seriesData$crimeID )
```

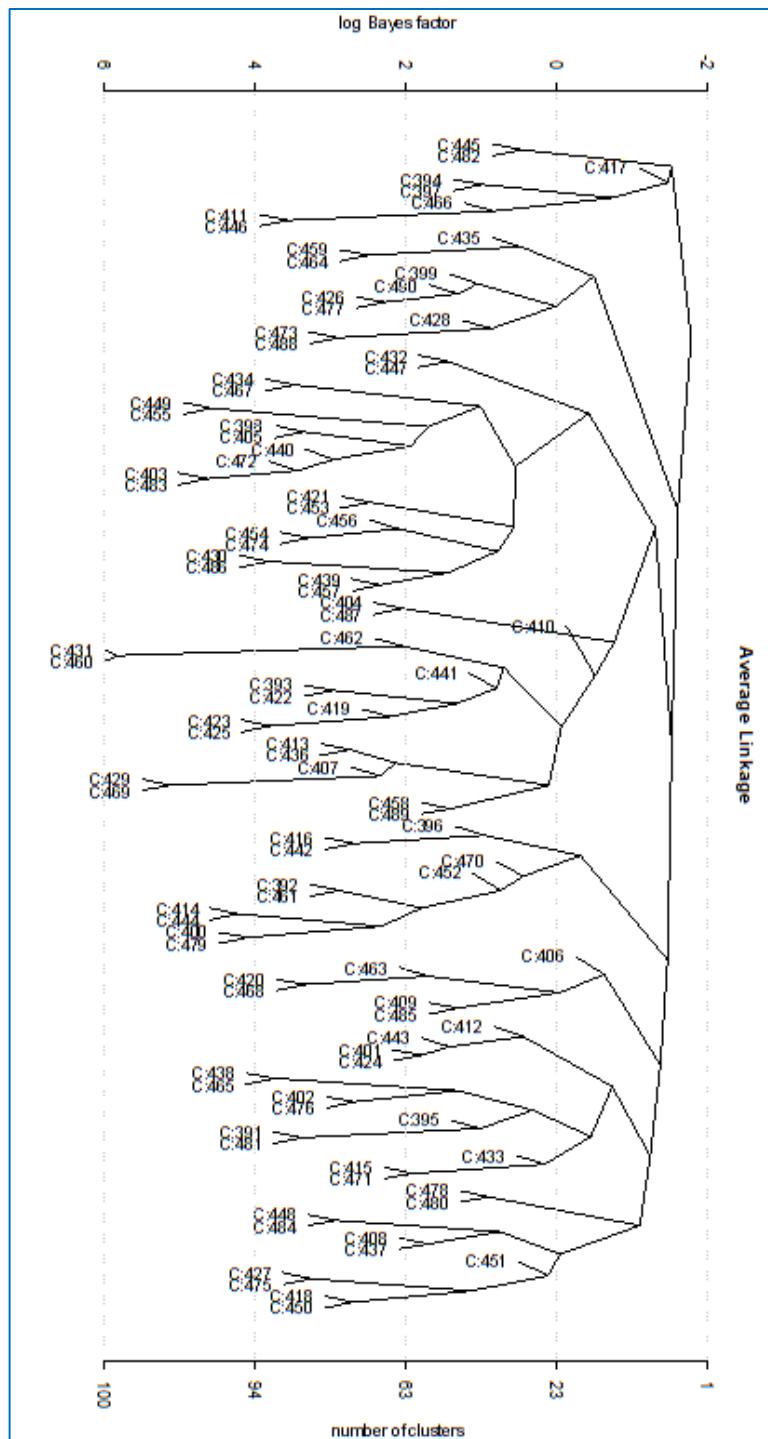
## Run agglomerative hierarchical crime clustering

```
tree = crimeClust_hier(unsolved, varlist, estimateBF, linkage = 'average', binary=TRUE)
```

### Plot results in dendrogram using `plot_hcc()`

```
plot_hcc(tree, yticks=seq(-2,6,by=2), type="triangle", hang=.05, main="Average Linkage")
```

The dendrogram created using `plot_hcc()`. The scale of the bottom (or right if in landscape view) represents the number of clusters as you move left (or down). The top (or left scale) represents the log Bayes factors for the different clusters.



## Examine crimes C:431 and C:460

```
subset(crimes, crimeID %in% c('C:431', 'C:460'))  
##   crimeID      X          Y M01 M02 M03      DT.FROM  
## 431 C:431 7097.0 -10256.8 26  a  D 1993-10-23 22:00:00  
## 460 C:460 7195.3 -10624.8 26  a  D 1993-11-07 03:10:00  
##                                DT.TO  
## 431 1993-10-24 06:00:00  
## 460 1993-11-07 03:10:00
```

## Find path info for crime C:429

```
cp = clusterPath('C:429', tree)  
cp[cp$logBF > 0,] # only return path for scores > 0  
##      logBF      crimes  
## 1 5.1074735      C:469  
## 2 2.3400814      C:407  
## 3 2.1364088 C:413, C:436  
## 4 0.1009954 C:458, C:489
```

## Hierarchical Based Crime Series Linkage

This approach to crime series identification compares an unsolved crime to every crime in criminal incident database and calculates its similarity as the log Bayes factor (according to the model developed for case linkage). Then it aggregates the similarity scores over the crime groups using single, complete, or average linkage. Single linkage uses the largest score (most similar crime) from each group, complete linkage uses the smallest score (least similar crime) from each group, and average linkage uses the average score as the group score.

## Example

To give an example, extract the solved and unsolved crimes from the crimes data.

```
solved = subset(crimes, crimeID %in% seriesData$crimeID)  
unsolved = subset(crimes, !crimeID %in% seriesData$crimeID )
```

The function seriesID() can be used to find the most similar crime series to the unsolved crime.

```
crime = unsolved[2,] # use the 2nd unsolved crime C:392  
crime
```

```

##   crimeID      X      Y M01 M02 M03          DT.FROM
## 392 C:392 12793.2 -3386.5 25  a  E 1993-06-19 07:00:00
##                               DT.TO
## 392 1993-06-19 07:00:00
results = seriesID(crime,solved,seriesData,varlist,estimateBF)
head(results$score)

##   group average single complete
## 1    12  3.739680 3.739680 3.739680
## 2    154  3.637851 3.637851 3.637851
## 3    160  3.617055 3.617055 3.617055
## 4     8  3.494182 3.494182 3.494182
## 5     9  3.224453 3.924757 2.760956
## 6    10  3.224453 3.924757 2.760956

```

This shows that the unsolved crime is most similar to the crime(s) in crime group 12 with an average linkage log Bayes factor of 4.06. To get the crimes and offenders associated with these groups, just use the subset() function with the groups object:

```

subset(results$groups,group=='12')      # most similar crime series
##   crimeID Index CS offenderID          TIME group
## 27 C:148    148 12      0:109 1993-06-25 01:20:00  12
subset(results$groups,group=='154')      # 2nd most similar series
##   crimeID Index CS offenderID          TIME group
## 205 C:304   304 154     0:237 1993-10-25 07:00:00 154
subset(results$groups,group=='9')        # a series with multiple crimes
##   crimeID Index CS offenderID          TIME group
## 9   C:144    144  9      0:106 1993-06-20 03:27:00   9
## 10  C:163    163  9      0:106 1993-06-20 13:15:00   9
## 11  C:145    145  9      0:106 1993-06-20 01:30:00   9
## 12  C:164    164  9      0:106 1993-06-20 13:15:00   9
## 13  C:165    165  9      0:106 1993-06-20 12:45:00   9
## 14  C:166    166  9      0:106 1993-06-20 12:45:00   9

```

We can do this for another unsolved crime

```

crime4 = unsolved[4,] # use the 4th unsolved crime
results4 = seriesID(crime4,solved,seriesData,varlist,estimateBF)
head(results4$score)

```

```

##   group    average    single   complete
## 1 136 1.5283543 1.5283543 1.5283543
## 2 316 1.1363833 1.1363833 1.1363833
## 3 37 0.8748010 0.8748010 0.8748010
## 4 48 0.8748010 0.8748010 0.8748010
## 5 206 0.5522861 0.5522861 0.5522861
## 6 219 0.4631613 0.4631613 0.4631613

```

Because the scores are so low (log Bayes factors around 1), this unsolved crime is not very similar to any other solved crimes in the crime database. Perhaps this is the start of a new crime series? It is also possible to compare a crime to all unsolved crimes to detect potential unsolved crime series.

### *Using Crime C:394 (the 4th unsolved crime)*

```

pairs = data.frame(i1=unsolved$crimeID[4],i2=unique(unsolved$crimeID[-4]))
X = compareCrimes(pairs,unsolved,varlist,binary=TRUE)
# Evidence data
score = data.frame(pairs,logBF=estimateBF(X))
head(score[order(-score$logBF),])
##      i1     i2      logBF
## 6 C:394 C:397  0.980062532
## 79 C:394 C:470  0.752385770
## 5 C:394 C:396  0.339529086
## 60 C:394 C:451 -0.007776621
## 69 C:394 C:460 -0.186420669
## 47 C:394 C:438 -0.206162572

```

There are no unsolved crimes that are very similar to this one - probably not enough evidence to link this crime to any others. This approach also gives similar results to what was obtained from the hierarchical clustering path approach:

```

C429 = which(unsolved$crimeID %in% 'C:429')           # now us
e crime C:429
pairs = data.frame(i1=unsolved$crimeID[C429],i2=unique(unsolved$crimeID[-C429]))
X = compareCrimes(pairs,unsolved,varlist,binary=TRUE)
# Evidence data
score = data.frame(pairs,logBF=estimateBF(X))
head(score[order(-score$logBF),])
##      i1     i2      logBF
## 78 C:429 C:469  5.107474

```

```
## 17 C:429 C:407 2.735772
## 23 C:429 C:413 2.371051
## 45 C:429 C:436 2.287908
## 13 C:429 C:403 1.305843
## 3 C:429 C:393 1.261425
```

### Results from hierarchical clustering

```
cp = clusterPath('C:429',tree)
cp[cp$logBF>0,]
##          logBF      crimes
## 1 5.1074735      C:469
## 2 2.3400814      C:407
## 3 2.1364088 C:413, C:436
## 4 0.1009954 C:458, C:489
```

The function `crimeClust_bayes()` is used for the Bayesian model-based clustering approach. Because it uses both solved and unsolved crimes, the labels (crime group) for the solved crimes is also passed into the function. (Note: this function will take 20+ minutes to run.)

## Bayesian Model-Based Approaches

This section illustrates the partially-supervised Bayesian model-based clustering approach to crime series linkage of `[@CrimeClust]`. This approach is partially-supervised because the offender is known for a subset of the events and utilizes spatiotemporal crime locations as well as crime features describing the offender's modus operandi.

The hierarchical model naturally handles complex features often seen in crime data, including missing data, interval censored event times, and a mix of discrete and continuous variables. It can also provide uncertainty assessments for all model parameters, including the relative influence of each feature (space, time, method of entry, etc.) in the model. In addition, the model produces posterior clustering probabilities which allow analysts to act on model output only as warranted.

The function `crimeClust_bayes()` is used for the Bayesian model-based clustering approach. Because it uses both solved and unsolved crimes, the labels (crime group) for the solved crimes is also passed into the function. (Note: this function will take 20+ minutes to run.)

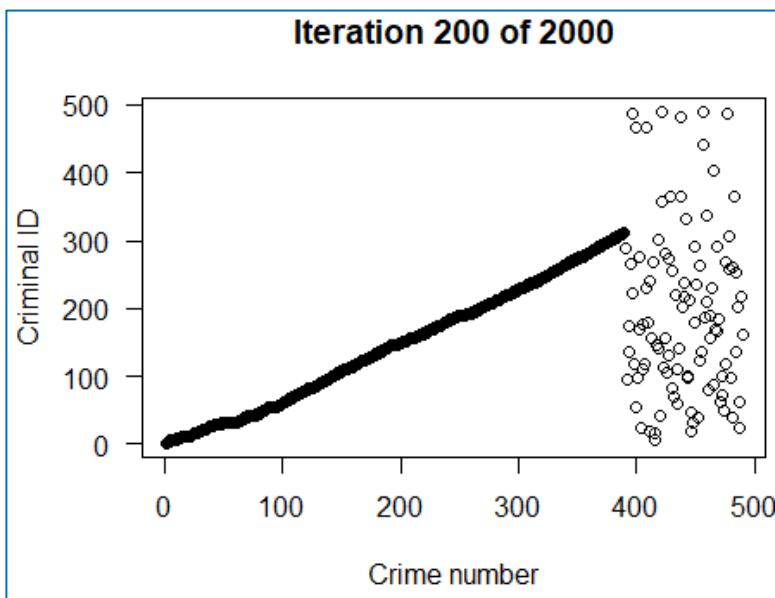
**Make the crime group labels for each crime (NA for unsolved crimes)**

```
seriesData$CG = makeGroups(seriesData,method=2)      # method=2 uses unique co-offenders
group_info = unique(seriesData[,c('crimeID','CG')])  # extract the group info
A = merge(crimes,group_info,by="crimeID",all.x=TRUE) # add group info to crimes
A = A[order(A$CG),]                                     # order by crime group
```

### Run Markov chain Monte Carlo (MCMC)

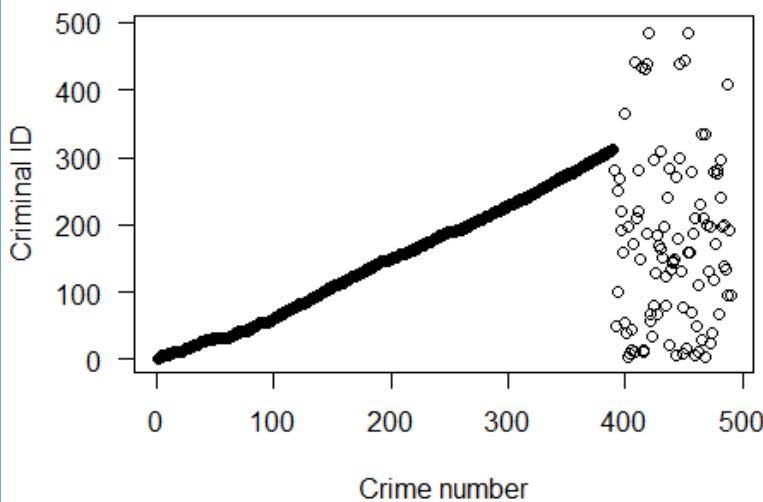
This is one of second solution plots from 2000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.

```
fit = crimeClust_bayes(A$CG, spatial=A[,c('X','Y')],t1=A$DT.FROM,
                        t2=A$DT.TO,
                        Xcat=A[,c("MO1","MO2","MO3")],maxcriminals=1000,
                        iters=2000,burn=500,update=100,seed=5)
```

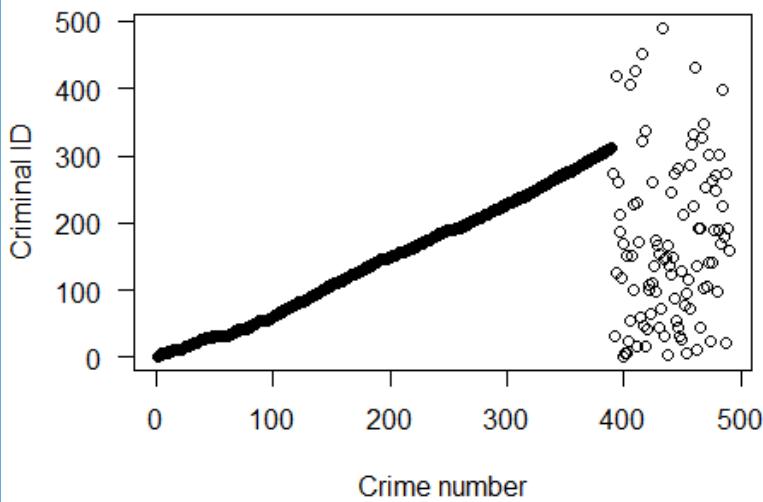


This is one of fourth solution plots from 2000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.

**Iteration 400 of 2000**

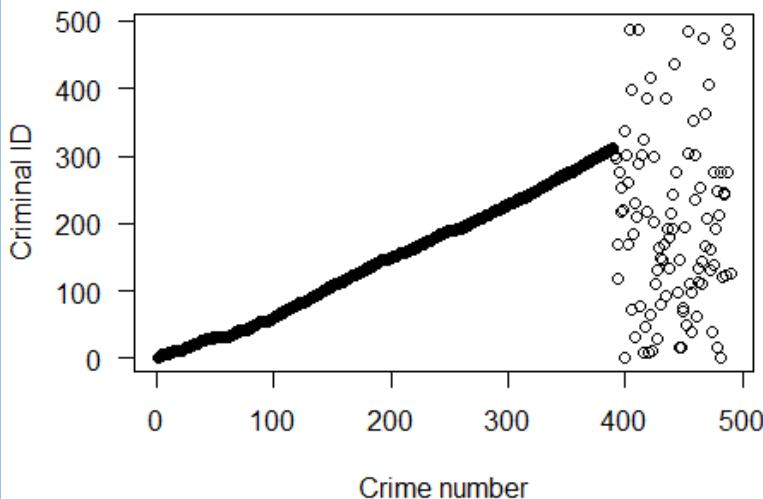


**Iteration 1000 of 2000**



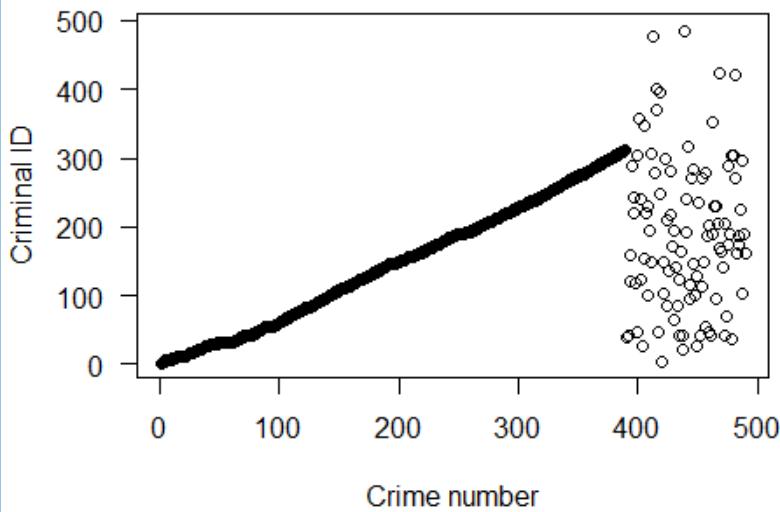
This is one of tenth solution plots from 2000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.

**Iteration 1200 of 2000**

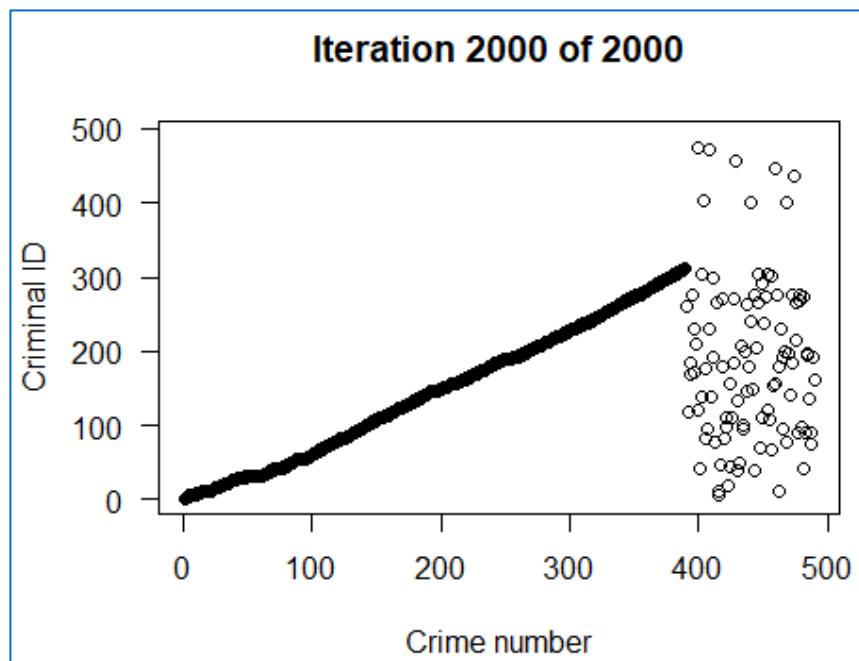


This is one of twelfth solution plots from 2000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.

**Iteration 1800 of 2000**



This is one of eighteenth solution plots from 2000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.



This is one of twentieth solution plots from 2000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.

```
summary(fit)
##          Length Class  Mode
## p.equal    240100 -none- numeric
## D           2000 -none- numeric
## df          6000 -none- numeric
## sd1         2000 -none- numeric
## sd2         2000 -none- numeric
## sds          4000 -none- numeric
## theta        2000 -none- numeric
## s.miss       0 -none- NULL
## t.censored   0 -none- NULL
## missing_s    0 -none- numeric
## missing_t   319 -none- numeric
## crimeID     490 -none- numeric
```

## Extract pairwise probabilities

```
pp = fit$p.equal # probability that crime i is linked to
# crime j
diag(pp) = NA
summary(pp)

##          V1              V2              V3
##  Min.   :0.0000000   Min.   :0.0000000   Min.   :0.00000
##  1st Qu.:0.0000000  1st Qu.:0.0000000  1st Qu.:0.00000
##  Median :0.0000000  Median :0.0000000  Median:0.00000
##  Mean   :0.0003476  Mean   :0.0003476  Mean   :0.00035
##  3rd Qu.:0.0000000  3rd Qu.:0.0000000  3rd Qu.:0.00000
##  Max.   :0.0766667  Max.   :0.0926667  Max.   :0.03400
##  NA's    :1           NA's    :1           NA's    :1
##          V4              V5              V6
##  Min.   :0.0000000   Min.   :0.0000000   Min.   :0.00000
##  1st Qu.:0.0000000  1st Qu.:0.0000000  1st Qu.:0.00000
##  Median :0.0000000  Median :0.0000000  Median :0.00000
##  Mean   :0.0006885  Mean   :0.001114   Mean   :0.00684
##  3rd Qu.:0.0000000  3rd Qu.:0.0000000  3rd Qu.:0.00000
##  Max.   :0.0640000  Max.   :0.052000   Max.   :1.00000
##  NA's    :1           NA's    :1           NA's    :1
##  .
##  .
##  .

##          V488             V489             V490
##  Min.   :0.0000000   Min.   :0.0000000   Min.   :0.000000
##  1st Qu.:0.0000000  1st Qu.:0.0000000  1st Qu.:0.000000
##  Median :0.0000000  Median :0.0000000  Median :0.000000
##  Mean   :0.003264   Mean   :0.003847   Mean   :0.002363
##  3rd Qu.:0.002000   3rd Qu.:0.000000   3rd Qu.:0.000000
##  Max.   :0.034000   Max.   :0.101333   Max.   :0.131333
##  NA's    :1           NA's    :1           NA's    :1
```

The matrix pp contains the pairwise estimated probability that two crime are linked (share a common offender). We can use this information for crime series identification. Using the image.plot() from the fields package, we can see how strongly the unsolved crimes are linked to the existing (solved) crime series.

```
library(fields) # if not installed, type: install.packages
("fields")
```

### **Get index of unsolved crimes**

```
ind.unsolved = which(is.na(A$CG)) # index of unsolved crimes  
n = nrow(A) # number of crimes
```

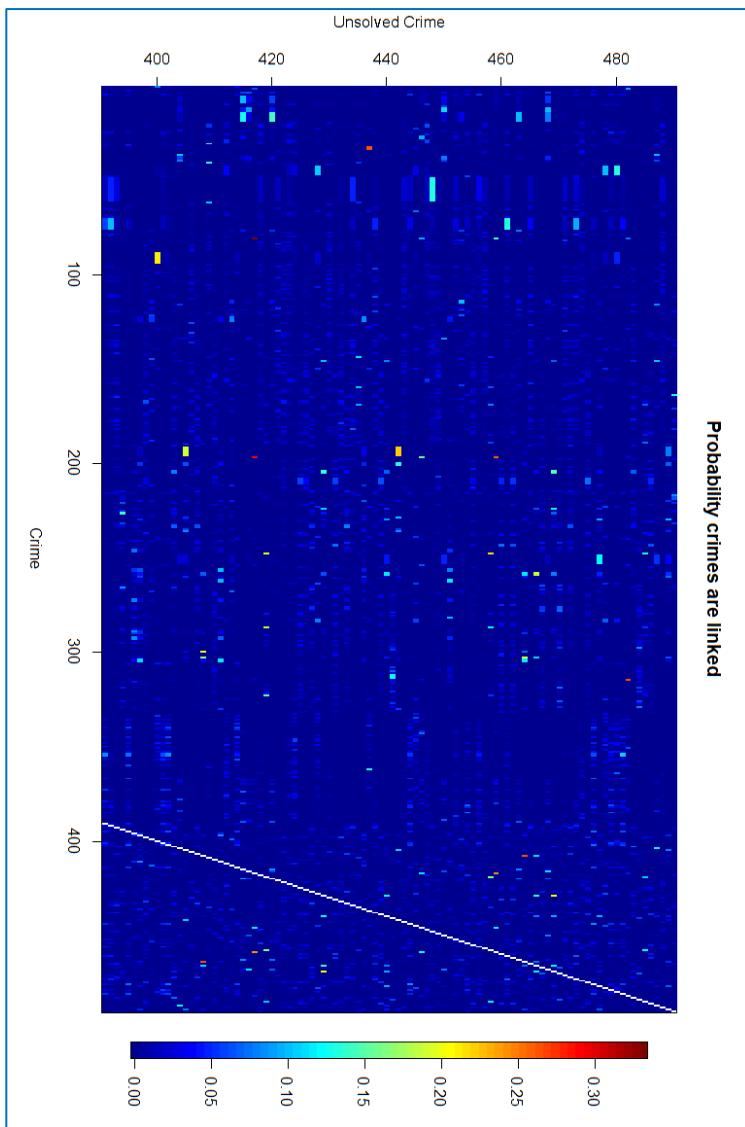
### **Image plot of linkage probabilities**

This plot shows the probability that unsolved crimes are related to crimes that are solved and clustered with. For most of the crimes shown, there is very little chance they are linked.

We see that some unsolved crimes are linked to solved crimes with a posterior probability above 0.25. These crimes may be worth further investigations. Here we plot the maximum posterior probability that an unsolved crime is linked to another crime (solved or unsolved).

```
fields::image.plot(1:n,ind.unsolved,pp[1:n,ind.unsolved],  
                  xlab="Crime",ylab="Unsolved Crime",  
                  main="Probability crimes are linked")
```

Probability crimes are linked

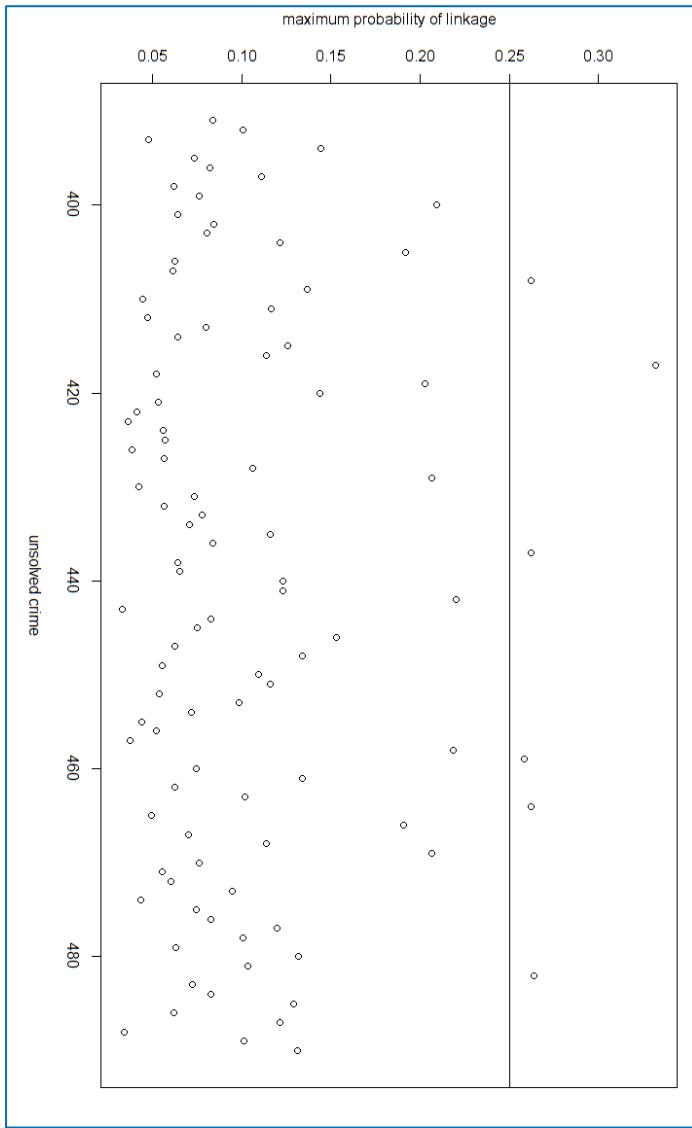


### Find strongest linkages

The plot also shows the probability that unsolved crimes are related to crimes that are solved and clustered with. There are only five unsolved crime that have a better than 25 percent chance of linkage.

```
unsolved.probs = apply(pp[ind.unsolved,],1,max,na.rm=TRUE)  
# maximum probability
```

```
plot(ind.unsolved,unsolved.probs,xlab="unsolved crime",yla  
b='maximum probability of linkage')  
abline(h=0.25)
```



```
ind = ind.unsolved[unsolved.probs > 0.25]  
investigate = as.character(A$crimeID[ind]) # crimeIDs for  
# crimes with strongest linkage  
investigate  
## [1] "C:408" "C:417" "C:437" "C:459" "C:464" "C:482"
```

This shows that C:408, C:417, C:437, C:459, C:464, and C:482 are the crimes with the strongest linkages (posterior probabilities greater than 0.25). A particular crime can be investigated in more detail with the function bayesProb():

```
bp = bayesProb(pp[A$crimeID %in% "C:417"])
bp$crimeID = A$crimeID[bp$index]
bp$CG = A$CG[bp$index]
head(bp)
##   index      prob crimeID  CG
## 1    81 0.33200000   C:15  46
## 2   197 0.28800000   C:26 146
## 3   459 0.24400000   C:459 NA
## 4   446 0.11400000   C:446 NA
## 5     2 0.06066667   C:10   2
## 6   482 0.04066667   C:482 NA
```

For this example, our model provides a list of the most likely crimes associated with the unsolved crime C:417. The first two crimes (C:15 and C:26) are solved crimes indicating that the offender(s) responsible for these crimes may also be responsible for C:417. The next two crimes, C:459 and C:446 do not have a group ID. This means that they are unsolved crimes. By providing the posterior probabilities, crime analysts may choose to investigate further only if the linkage is strong enough.

## Exercises

1. Using the logic of section “Trial Run with crimeLinkage,” build a dataset that 4 criminals (id 1 – 4), 16 spatial locations (4 x 4 matrix), a mode of robbery (armed, or unarmed) and type of business (pub, convenient store) and times of the crimes committed. Now let’s pretend we don’t know the criminal for crimes 1, 4, 6, 8, 12 and 14.
  - a. Fit the data using a naïve Bayes, NB, model and summarize the fit.
  - b. Plot the posterior probability matrix in which each pair of crimes was contained and view the model summary.
  - c. Use crimeLinkage for analyzing the unsolved crimes for the dataset you just created

2. Download crimes\_set.csv and offenders\_set.csv from [https://github.com/stricje1/VIT\\_University/tree/master/Predictive\\_Modeling/data](https://github.com/stricje1/VIT_University/tree/master/Predictive_Modeling/data). The data is organized like the data we used in this chapter. The offenders\_set file has 161 data points with fields “offenderID” and “crimeID”. The crimes\_set has 125 data points with he fields:
- crimeID: The crime ID number (4 criminal)
  - X,Y: Spatial (16 locations)
  - Coordinates MO1 A categorical (assign 16 pairs of coordinates.  
Hint: use Google Maps)
  - MO1: variable that takes values 1,...,31
  - MO2: A categorical MO variable that takes values “Armed” and “Unarmed”
  - MO3: A categorical MO variable that takes values “Male” and “Female”
  - DT.FROM: The earliest possible Date-time of the crime.
  - DT.TO: The latest possible Date-time of the crime
- a. Load the data and set up the data for analysis, making Crime Pairs for Case Linkage, and making Evidence Variables for Case Linkage.
  - b. Build training and testing data sets. [Hint: use 60%-70% of the data to train the model.]
  - c. Fit a logistic regression model to the training data and summarize your fit
  - d. Make an estimateBF() function to return the estimated log Bayes factor and use the testing set to make a prediction.
  - e. Create a plot of residual versus fitted predicted values for the logistic regression model.
  - f. Create a Normal QQ plot of theoretical quantiles for the logistic regression model.
  - g. Create a scale-location plot of predicted values for the logistic regression model.
  - h. Create a residual versus leverage plot for the logistic regression model.
  - i. Create a plot of estimated values for the logistic regression model.

- j. Fit a naive Bayes model and make estimateBF() function to make predictions.
  - k. Plot the log-scale plots of factors for the naïve Bayes model.
  - l. Plot the of estimated values for the naïve Bayes model.
3. Use the `crimelinkage` package and the function `crimeClust_hier()` for agglomerative hierarchical crime clustering.
- a. Get the unsolved crimes subset from the data in Exercise 2.
  - b. Run agglomerative hierarchical crime clustering on the data.
  - c. Plot results in dendrogram using `plot_hcc()` and interpret the results.
  - d. Use the Hierarchical Based Crime Series Linkage approach and the crimes and unsolved crimes data to score the crimes.
  - e. Interpret your results
4. Use a partially-supervised Bayesian model-based clustering approach to crime series linkage of [@CrimeClust] using the data in Exercise 2.
- a. Use the function `crimeClust_bayes()` for the Bayesian model-based clustering approach.
  - b. Run MCMC using 1000 iterations of the `crimeClust_bayes()` model, recalling that we update every 100 iterations.
  - c. Extract pairwise probabilities from the model.
  - d. Get index of unsolved crimes and create an image plot of linkage probabilities.
  - e. Find strongest crime linkages.



# CHAPTER 5 – Introduction to Text Mining

## Introduction

This is our first lesson on text analytics, so we will do some necessary but basic preprocessing to prepare for our analysis. This includes converting the text to lower case, removing numbers and stop-words, combining words that need to stay together (like “data science”), and putting our text into a dataframe.

The text we will use is a collect of texts, specifically a few blogs I have written. The complete set of blogs comprise what we call a “corpus,” which is Latin for “body” or “body of texts” in this instance. You may have heard “corpus” used in city names like Corpus Christi, which literally means “the body of Christ.”

## Load the R packages

Load the following R packages for text mining and then load your texts into R.

```
library(tm)
## Loading required package: NLP
library(wordcloud2)
library(yaml)
library(NLP)
library(tm)
library(SnowballC)
library(ggplot2)
## Attaching package: 'ggplot2'
## The following object is masked from 'package:NLP':
##     annotate
```

On your PC, create a folder “Data\_Analytics” on your C: drive or use a folder you already have to download the “text.zip” from [https://github.com/stricje1/VIT\\_University/tree/master/Predictive\\_Modeling/data](https://github.com/stricje1/VIT_University/tree/master/Predictive_Modeling/data). Then use the following code chunk to load your data into R Studio (the path you should use has been commented out):

## Load the Data

```
cname <- file.path("C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/data", "text")
#cname <- file.path("C:/Users/username/Documents/Data_Analytics/data", "text")
cname
## [1] "C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/data/text"
dir(cname)
## [1] "A one-eyed man in the kingdom of the blind.txt"
## [2] "All Things Data.txt"
## [3] "Analytics and Statistics.txt"
## [4] "Analytics is it more than a buzzword.txt"
## [5] "Bayesian networks.txt"
## [6] "Big Data Analytics and Human Resources.txt"
## [7] "Big Data The Good the Bad and the Ugly.txt"
## [8] "Call Center Analytics.txt"
## [9] "Classification Trees using R.txt"
## [10] "Clouds clouds and more clouds.txt"
## [11] "Cluster Models.txt"
## [12] "Cyber-Threat Risk Assessment using R.txt"
## [13] "Data Scientist are Dead Long Live Data Science.txt"
## [14] "Do you like my Ensemble.txt"
## [15] "Free SAS.txt"
## [16] "Getting the Question Right.txt"
## [17] "What are Association Rules in Analytics.txt"
## [18] "Where_did_all_the_Teaching_Go.txt"
## [19] "Where_did_all_the_Thinking_Go.txt"
## [20] "Why_Stand_Many_Have_Fallen.txt"
docs <- Corpus(DirSource(cname))
```

Now examine the data you loaded using

```
summary(docs)
##
  Length Class Mode
## [1] 2     PlainTextDocument list
## [2] 2     PlainTextDocument list
## [3] 2     PlainTextDocument list
## [4] 2     PlainTextDocument list
## [5] 2     PlainTextDocument list
## [6] 2     PlainTextDocument list
```

```

## [7]    2 PlainTextDocument  list
## [8]    2 PlainTextDocument  list
## [9]    2 PlainTextDocument  list
## [10]   2 PlainTextDocument  list
## [11]   2 PlainTextDocument  list
## [12]   2 PlainTextDocument  list
## [13]   2 PlainTextDocument  list
## [14]   2 PlainTextDocument  list
## [15]   2 PlainTextDocument  list
## [16]   2 PlainTextDocument  list
## [17]   2 PlainTextDocument  list
## [18]   2 PlainTextDocument  list
## [19]   2 PlainTextDocument  list
## [20]   2 PlainTextDocument  list
##
inspect(docs)
## <<SimpleCorpus>>
## Metadata: corpus specific: 1, document level (indexed)
: 0
## Content: documents: 20
##
A one-eyed man in the kingdom of the blind.txt
##
A one-eyed man in the kingdom of the blind:\nPredicting the Unpredictable\nâ\200Almost nobodyâ\200\231s competent, Paul. Itâ\200\231s enough to make you cry to see how bad most people are at their jobs. If you can do a half-assed job of anything, youâ\200\231re a one-eyed man in the kingdom of the blind.â\200\235 â\200"Kurt Vonnegut, Player Piano\nAbstract\nThis article is about Predictive Modeling. It explores the appropriateness of modeling in general and predictive modeling in particular, as well as examining some pitfalls. Modeling is the process of formulating and abstracting a representation of a real problem, based on simplifying assumptions. Thus, no model is an exact representation of reality. Said a different way, a model cannot fully represent a complex problem, but can provide some insight into the problem and assist decision makers with applying solutions.

```

## Corpus Preprocessing

Next, convert the text to lowercase and inspect your work:

```

docs <- tm_map(docs, tolower)
inspect(docs[1])
## <<SimpleCorpus>>
## Metadata: corpus specific: 1, document level (indexed)
: 0
## Content: documents: 1
##
A one-eyed man in the kingdom of the blind.txt
## a one-eyed man in the kingdom of the blind:\npredicting
the unpredictable\nâ\200\u200ealmost nobodyâ\200\u231s competent
, paul. itâ\200\u231s enough to make you cry to see how bad
most people are at their jobs. if you can do a half-assed
job of anything, youâ\200\u231re a one-eyed man in the king
dom of the blind.â\200\u235 â\200”kurt vonnegut, player pia
no\nabstract\nthis article is about predictive modeling. i
t explores the appropriateness of modeling in general and
predictive modeling in particular, as well as examining so
me pitfalls. modeling is the process of formulating and ab
stracting a representation of a real problem, based on sim
plifying assumptions. thus, no model is an exact represent
ation of reality. said a different way, a model cannot ful
ly represent a complex problem, but can provide some insig
ht into the problem and assist decision makers with applyi
ng solutions.

```

Next, remove unnecessary words from the text:

```

docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removeWords, c("can", "should", "woul
d", "figure", "using", "will", "use", "now", "see", "may",
"given", "since", "want", "next", "like", "new", "one", "m
ight", "without"))

```

Now, combine words that should stay together

```

for (j in seq(docs))
{
  docs[[j]] <- gsub("data analytics", "data_analytics", docs
[[j]])
  docs[[j]] <- gsub("predictive models", "predictive_models"
, docs[[j]])
  docs[[j]] <- gsub("predictive analytics", "predictive_anal
ytics", docs[[j]])

```

```
docs[[j]] <- gsub("data science", "data_science", docs[[j]])
docs[[j]] <- gsub("operations research", "operations_resea
rch", docs[[j]])
docs[[j]] <- gsub("chi-square", "chi_square", docs[[j]])
}
```

## Create Document Matrices

In these setps we will prepare the documents for analysis. First we will put the text into a term-document matrix and view it:

```
tdm <- TermDocumentMatrix(docs)
tdm
## <<TermDocumentMatrix (terms: 3971, documents: 20)>>
## Non-/sparse entries: 7178/72242
## Sparsity          : 91%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Second, create document-term matrix and view it:

```
dtm <- DocumentTermMatrix(docs)
dtm
## <<DocumentTermMatrix (documents: 20, terms: 3971)>>
## Non-/sparse entries: 7178/72242
## Sparsity          : 91%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Next, organize the terms by their frequency:

```
freq <- colSums(as.matrix(dtm))
length(freq)
## [1] 3971
ord <- order(freq)
```

Now, put it into a matrix and save it to your working directory:

```
m <- as.matrix(dtm)
dim(m)
## [1] 20 3971
write.csv(m, file="dtm.csv")
```

Remove sparse terms. This makes a matrix that is a maximum of 10% empty space.

```
dtms <- removeSparseTerms(dtm, 0.1)
inspect(dtms)

## <<DocumentTermMatrix (documents: 20, terms: 0)>>
## Non-/sparse entries: 0/0
## Sparsity           : 100%
## Maximal term length: 0
## Weighting          : term frequency (tf)
## Sample             :
##                                         Terms
## Docs
##   A one-eyed man in the kingdom of the blind.txt
##   All Things Data.txt
##   Analytics and Statistics.txt
##   Analytics is it more than a buzzword.txt
##   Bayesian networks.txt
##   Big Data Analytics and Human Resources.txt
##   Big Data The Good the Bad and the Ugly.txt
##   Call Center Analytics.txt
##   Classification Trees using R.txt
##   Clouds clouds and more clouds.txt
##   Cluster Models.txt
##   Cyber-Threat Risk Assessment using R.txt
##   Data Scientist are Dead Long Live Data Science.txt
##   Do you like my Ensemble.txt
##   Free SAS.txt
##   Getting the Question Right.txt
##   What are Association Rules in Analytics.txt
##   Where_did_all_the_Teaching_Go.txt
##   Where_did_all_the_Thinking_Go.txt
##   Why_Stand_Many_Have_Fallen.txt
```

Next, we check some of the frequency counts. There are a lot of terms, so for now, we just check out some of the most and least frequently occurring words, as well as check out the frequency of frequencies.

```
knitr::kable(findFreqTerms(dtm, lowfreq=60), caption = 'Frequent Words')
```

*Frequent Words*

x
analytics

data
model
true
classification
tree
dendrogram

```
knitr:::kable(freq[1:10], caption = 'Frequency Counts')
```

*Frequency Counts*

	x
data	198
analytics	104
true	68
model	67
classification	66
dendrogram	64
tree	62
branches	59
clustering	59
models	57

```
findFreqTerms(dtm, lowfreq=150)
```

```
## [1] "data"
```

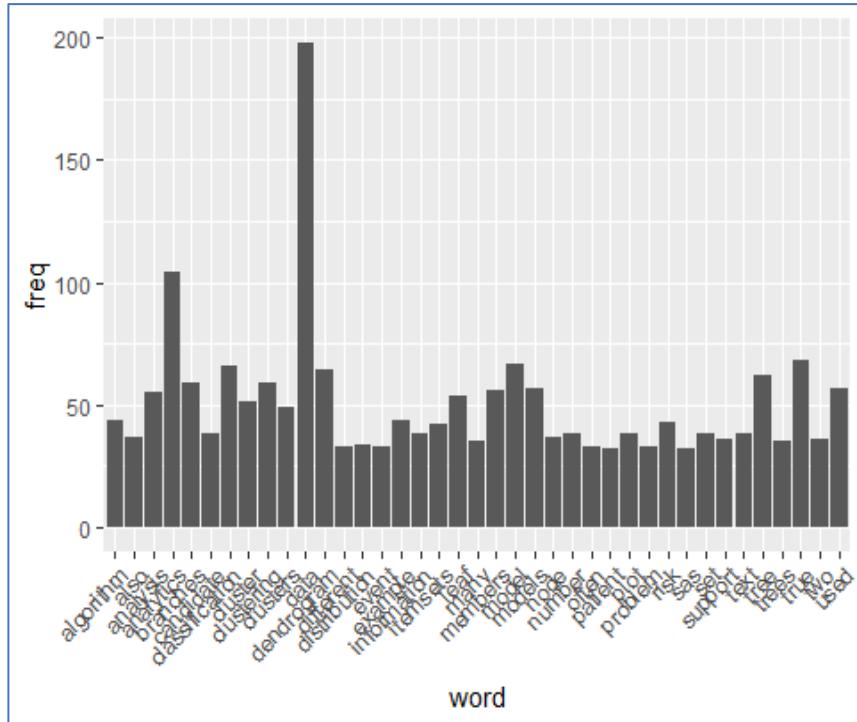
## Visualizing the Results

Now, we plot words that appear at least 50 times.

```
wf <- data.frame(word=names(freq), freq=freq)
datatable(wf, options = list(pageLength = 5,scrollX='400px'))
```

Show 5 entries		Search:
	word	freq
data	data	198
analytics	analytics	104
true	true	68
model	model	67
classification	classification	66

```
p <- ggplot(subset(wf, freq>30), aes(word, freq))
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```



## Find correlations

Now, we find correlations in the text.

```
findAssocs(dtm, c("question", "analysis"), corlimit=0.98)
# specifying a correlation limit of 0.98
$`question`
```

achieve	behaviors	brainer	cart
0.99	0.99	0.99	0.99
currency	dave	detect	deviceÃ
0.99	0.99	0.99	0.99
dialogue	director	downstream	expertise
0.99	0.99	0.99	0.99
failures	forever	horse	investment

```

    0.99      0.99      0.99      0.99
john      keys      knowing     mention
    0.99      0.99      0.99      0.99
months   phenomenon   realize     recipe
    0.99      0.99      0.99      0.99
reiterated      ret      robinson    rolls
    0.99      0.99      0.99      0.99
roske      roskeÃ     seen       slides
    0.99      0.99      0.99      0.99
staffs      stake      stems      temporal
    0.99      0.99      0.99      0.99
timing      twice      vince      wallet
    0.99      0.99      0.99      0.99

##  

## $analysis  

## numeric(0)  

findAssocs(dtms, "contrast", corlimit=0.90) # specifying a  

correlation limit of 0.95  

## $contrast  

## numeric(0)

```

## Using Wordclouds to Visualize Results

Plot words using a wordcloud that occur at least 50 times.

```
wordcloud2(subset(wf, freq>10))
```



# Tidy Text Analytics I

## *Tidy Format*

The tidy format is described by Hadley Wickham (Wickham 2014), with the following specific structure:

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table

## **Tidy Text Format**

Applying this to text, we can say that the tidy text format is table with one-token-per-row. A token is usually taken to be a word and is a meaningful unit of text that we are interested in using for analysis. The process of splitting text into tokens is called tokenization. In addition to words, the token that is stored in each row can be an n-gram, sentence, or paragraph. In the `tidytext` package, functionality is provided to tokenize by commonly used units of text like these and convert to a one-term-per-row format. However, note that the one-token-per-row structure of tidy text is in contrast to the usual methods of storing text for analyses, like strings or in a document-term matrix.

Tidy text data sets can be manipulated with a standard set of “tidy” tools, including popular packages such as `tidyverse` (Wickham 2016), `dplyr`,

(Wickham and Francois 2016), broom (Robinson 2017), and ggplot2 (Wickham 2009).

### **Contrasting tidy text with other data structures**

Given the unique approach of tidy text, it is worth contrasting with the ways text is often stored in text mining approaches. - String: Text can, of course, be stored as strings, i.e., character vectors, within R, and often text data is first read into memory in this form. - Corpus: These types of objects typically contain raw strings annotated with additional metadata and details. - Document-term matrix: This is a sparse matrix describing a collection (i.e., a corpus) of documents with one row for each document and one column for each term.

#Installing Required Libraries

```
if(!require(tidytext)) install.packages("tidytext")
## Loading required package: tidytext
if(!require(tidyverse)) install.packages("tidyverse")
## Loading required package: tidyverse
## -- Attaching packages -----
----- tidyverse 1.2.1 --
## v ggplot2 3.0.0      v purrr   0.2.5
## v tibble   1.4.2      v dplyr    0.7.6
## v tidyverse 0.8.1     v stringr  1.3.1
## v readr    1.1.1      vforcats  0.3.0
## -- Conflicts -----
----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
if(!require(dplyr)) install.packages("dplyr")
if(!require(tidyr)) install.packages("tidyr")
if(!require(tibble)) install.packages("tibble")
if(!require(readr)) install.packages("readr")
if(!require(broom)) install.packages("broom")
## Loading required package: broom
if(!require(ggplot2)) install.packages("ggplot2")
```

## *The unnest\_tokens function*

We will use an example to look at the basic operations required for “tidying” text. Here we have the first stanza of the song “Hallelujah,” released by Leonard Cohen (1984) and we place it into an array.

```
text_sample <- c("Now I've heard there was a secret chord"
,
  "That David played, and it pleased the Lord",
  "But you don't really care for music, do you?",
  "It goes like this",
  "The fourth, the fifth",
  "The minor fall, the major lift",
  "The baffled king composing Hallelujah"
)
text_sample
## [1] "Now I've heard there was a secret chord"
## [2] "That David played, and it pleased the Lord"
## [3] "But you don't really care for music, do you?"
## [4] "It goes like this"
## [5] "The fourth, the fifth"
## [6] "The minor fall, the major lift"
## [7] "The baffled king composing Hallelujah"
```

This is a typical character vector that we might want to analyze. In order to turn it into a tidy text dataset, we first need to put it into a data frame. We will use the `data-frame` function from the `dplyr` package to accomplish this.

```
library(dplyr)
text_sample_df <- data_frame(line = 1:7, text = text_sample)
text_sample_df
## # A tibble: 7 x 2
##   line    text
##   <int> <chr>
## 1     1 Now I've heard there was a secret chord
## 2     2 That David played, and it pleased the Lord
## 3     3 But you don't really care for music, do you?
## 4     4 It goes like this
## 5     5 The fourth, the fifth
## 6     6 The minor fall, the major lift
## 7     7 The baffled king composing Hallelujah
```

## Tibbles

Notice that the data frame printed out as a “tibble.” Within R, a tibble is a modern class of data frame available in the dplyr and tibble packages with the following features: - has a convenient print method - will not convert strings to factors - does not use row names. Tibbles are good to use with tidy tools.

However, observe that this text data frame is not yet compatible with tidy text analysis. Since each row is made up of multiple words, we cannot filter out words or count which occur most frequently. We need to convert this so that it has one-token-per-row. Later, when we work with multiple documents, we will need one-token-per-document-per-row.

## Tokenization

As we have stated, token is a meaningful unit of text that we are interested in using for further analysis. Tokenization is the process of splitting text into tokens. To obtain the tidy text framework, we need to break the text into individual tokens and transform the text to a tidy data structure. To do this, we use tidytext’s unnest\_tokens() function.

```
library(tidytext)
text_sample_df %>% unnest_tokens(word, text)
## # A tibble: 44 x 2
##       line word
##   <int> <chr>
## 1     1 now
## 2     1 i've
## 3     1 heard
## 4     1 there
## 5     1 was
## 6     1 a
## 7     1 secret
## 8     1 chord
## 9     2 that
## 10    2 david
## # ... with 34 more rows
```

The two basic arguments to unnest\_tokens are the column names, “word” and “text.” First we have the output column name “word” that will be created as the text is unnested into it. Second, we have the input

column “text” that the text comes from (Recall that `text_sample_df` above has a column called `text` that contains the data of interest).

After using `unnest_tokens`, we split each row so that there is one token (word) in each row of the new data frame, which is the default tokenization in `unnest_tokens()`. Also observe that the line number each word came from are retained and punctuation has been stripped. Moreover, `unnest_tokens()` converts the tokens to lowercase by default, which makes them easier to combine or compare with other datasets.

### ***Example - Indian Philosophy***

After teaching data analytics at the Vellore Institute of Technology (VIT), at Vellore India, in 2016, my friend and Dean of the School for Information Technology Engineering (SITE) gave me a two-volume set of books on “Indian Philosophy.” In order to enhance my understanding of what I had read, I performed text analytics on them, and repeat that here as an example.

First, we put ind texts to a data frame.

```
ind_words <- tibble(file = paste0("~/VIT_University/tidy_text/",  
  c("Indian_Philosophy_Part_I.txt", "Indian_Philosophy_P  
art_II.txt"))) %>%  
  mutate(text = map(file, read_lines))  
ind_words  
## # A tibble: 2 × 2  
##   file  
##     <chr>  
<list>  
## 1 ~/VIT_University/tidy_text/Indian_Philosophy_Part_I.t  
xt <chr [13,715]>  
## 2 ~/VIT_University/tidy_text/Indian_Philosophy_Part_II.  
txt <chr [19,984]>
```

The resulting tibble has a variable `file` that is the name of the file that created that row and a list-column of the text of that file.

We want to `unnest()` that tibble, remove the lines that are LaTeX crude and compute a line number.

```

ind_words <- ind_words %>%
  unnest() %>%
  filter(text != "%!TEX root = ind.tex") %>%
  filter(!str_detect(text, "^(\\\\\\\\[A-Z,a-z])"),
         text != "") %>%
  mutate(line_number = 1:n(),
        file = str_sub(basename(file), 1, -5))

```

Now we have a tibble with file giving us the chapter, text giving us the line of text from the text files and line\_number giving a counter of the number of lines since the start of the ind texts.

Now we want to tokenize (strip each word of any formatting and reduce down to the root word, if possible). This is easy with unnest\_tokens(). I played around with the results and came up with some other words that needed to be deleted (stats terms like ci or p, LaTeX terms like \_i or tabular and references/numbers).

```

ind_words <- ind_words %>%
  unnest_tokens(word, text) %>%
  filter(!str_detect(word, "[0-9]"),
         word != "fismanreview",
         word != "multicolumn",
         word != "p",
         word != "_i",
         word != "c",
         word != "ci",
         word != "al",
         word != "dowellsars",
         word != "h",
         word != "tabular",
         word != "t",
         word != "ref",
         word != "cite",
         !str_detect(word, "[a-z]_"),
         !str_detect(word, ":"),
         word != "bar",
         word != "emph",
         !str_detect(word, "textless"))

```

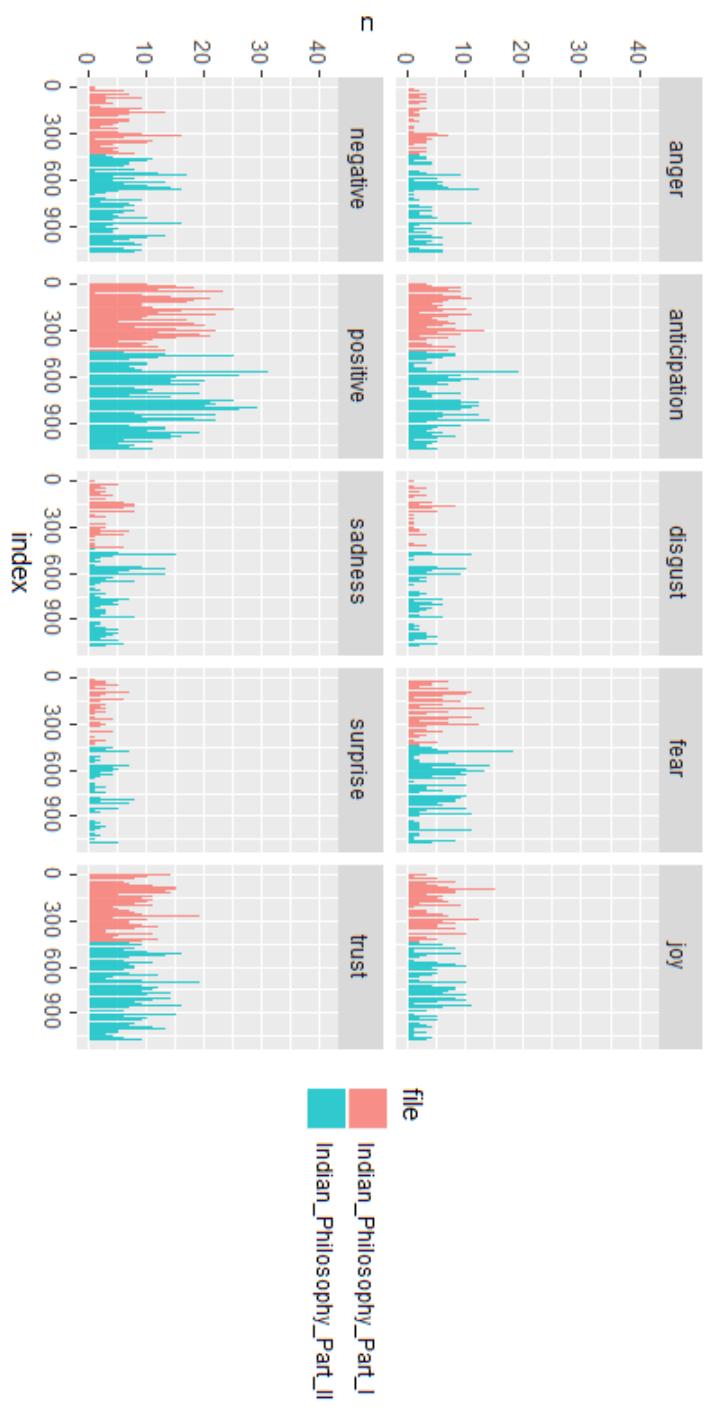
file	line_number	word
Indian_Philosophy_Part_I	1	indian
Indian_Philosophy_Part_I	1	philosophy

```
## 3 Indian_Philosophy_Part_I          1 part
## 4 Indian_Philosophy_Part_I          1 i
## 5 Indian_Philosophy_Part_I          2 chapter
## 6 Indian_Philosophy_Part_I          2 i
## 7 Indian_Philosophy_Part_I          3 introduction
## 8 Indian_Philosophy_Part_I          4 general
## 9 Indian_Philosophy_Part_I          4 characteristics
## 10 Indian_Philosophy_Part_I         4 of
## # ... with 250,154 more rows
```

Now to compute the sentiment using the words written per line in the ind texts. tidytext comes with three sentiment lexicons, affin, bing and nrc. affin provides a score ranging from -5 (very negative) to +5 (very positive) for 2,476 words. bing provides a label of “negative” or “positive” for 6,788 words. nrc provides a label (anger, anticipation, disgust, fear, joy, negative, positive, sadness, surprise or trust) for 13,901 words. None of these account for negation (“I’m not sad,” is a negative sentiment, not a positive one).

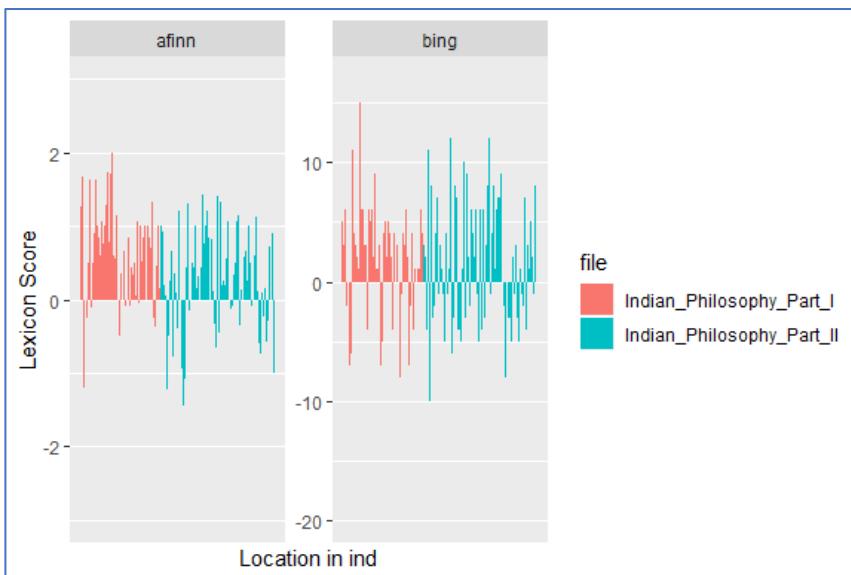
Using the nrc lexicon, let’s see how the emotions of my words change over the three ind texts.

```
ind_words %>%
  inner_join(get_sentiments("nrc")) %>%
  group_by(index = line_number %/%
  25, file, sentiment) %>
%
  summarize(n = n()) %>%
  ggplot(aes(x = index, y = n, fill = file)) +
  geom_bar(stat = "identity", alpha = 0.8) +
  facet_wrap(~ sentiment, ncol = 5)
## Joining, by = "word"
```



Both texts are more positive than negative and represent trust fairly well. It looks like “disgust” and “sadness” are minimized. We can use the bing and afinn lexicons to look at how the sentiment of the words changed over the course of the thesis.

```
ind_words %>%
  left_join(get_sentiments("bing")) %>%
  left_join(get_sentiments("afinn")) %>%
  group_by(index = line_number %% 25, file) %>%
  summarize(afinn = mean(score, na.rm = TRUE),
            bing = sum(sentiment == "positive", na.rm = TRUE) -
            sum(sentiment == "negative", na.rm = TRUE)) %>%
  gather(lexicon, lexicon_score, afinn, bing) %>%
  ggplot(aes(x = index, y = lexicon_score, fill = file)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ lexicon, scale = "free_y") +
  scale_x_continuous("Location in ind", breaks = NULL) +
  scale_y_continuous("Lexicon Score")
## Joining, by = "word"
## Joining, by = "word"
## Warning: Removed 2 rows containing missing values (position_stack).
```



```
ind_words
```

```

## # A tibble: 250,164 x 3
##   file           line_number word
##   <chr>          <int> <chr>
## 1 Indian_Philosophy_Part_I      1 indian
## 2 Indian_Philosophy_Part_I      1 philosophy
## 3 Indian_Philosophy_Part_I      1 part
## 4 Indian_Philosophy_Part_I      1 i
## 5 Indian_Philosophy_Part_I      2 chapter
## 6 Indian_Philosophy_Part_I      2 i
## 7 Indian_Philosophy_Part_I      3 introduction
## 8 Indian_Philosophy_Part_I      4 general
## 9 Indian_Philosophy_Part_I      4 characteristics
## 10 Indian_Philosophy_Part_I     4 of
## # ... with 250,154 more rows

```

Looking at the two lexicon's scoring of my books, the affin lexicon seems a little more stable if we assume local correlation of sentiments is likely. The scores show that all three text are much more positive than negative.

### *Filter for negative words*

Next, we apply a filter for negative words in order to distinguish between positive and negative in a wordcloud, for example.

```

bingnegative <- get_sentiments("bing") %>%
  filter(sentiment == "negative")

```

The, we get a word count.

```

wordcounts <- ind_words %>%
  group_by(index = line_number %% 25, file) %>%
  summarize(words = n())
wordcounts

## # A tibble: 1,081 x 3
## # Groups:   index [?]
##   index file           words
##   <dbl> <chr>          <int>
## 1 0 Indian_Philosophy_Part_I 187
## 2 1 Indian_Philosophy_Part_I 240
## 3 2 Indian_Philosophy_Part_I 251
## 4 3 Indian_Philosophy_Part_I 227
## 5 4 Indian_Philosophy_Part_I 215
## 6 5 Indian_Philosophy_Part_I 239
## 7 6 Indian_Philosophy_Part_I 241

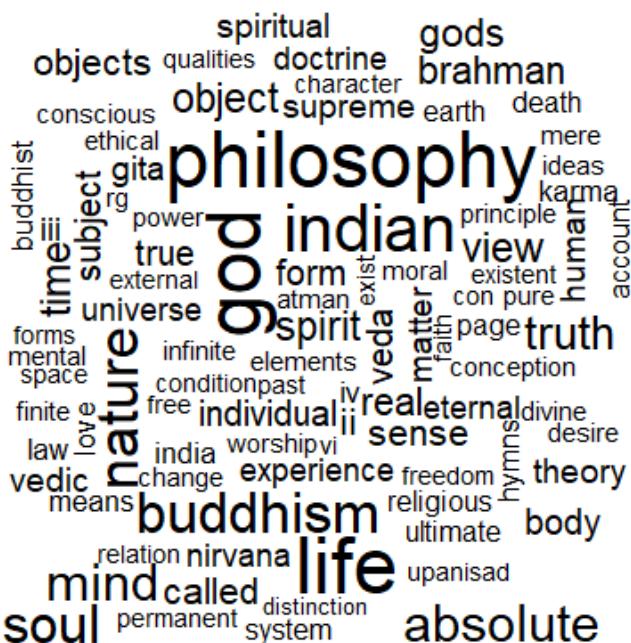
```

```
## 8      7 Indian_Philosophy_Part_I  239
## 9      8 Indian_Philosophy_Part_I  228
## 10     9 Indian_Philosophy_Part_I  221
## # ... with 1,071 more rows
```

## *Build a cloud chart*

We use the previous code chunk results to build a wordcloud.

```
library(wordcloud)
## Loading required package: RColorBrewer
ind_words %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
## Joining, by = "word"
```



Finally, we build a contrasting cloud chart.

```
library(reshape2)
## 
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyverse':
##
##     smiths
ind_words %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "blue"),
                    max.words = 100)
## Joining, by = "word"
```



**END OF SCRIPT**

## Tidy Text Analytics II

### *Load necessary libraries*

```
if(!require(tm)) install.packages("tm")
if(!require(tidytext)) install.packages("tidytext")
if(!require(tidyverse))
install.packages("tidyverse")
if(!require(stringr)) install.packages("stringr")
if(!require(dplyr)) install.packages("dplyr")
if(!require(tidyr)) install.packages("tidyr")
if(!require(janeaustenr))
install.packages("janeaustenr")
if(!require(purrr)) install.packages("purrr")
if(!require(sentimentr))
install.packages("sentimentr")
if(!require(readr)) install.packages("readr")
if(!require(wordcloud))
install.packages("wordcloud")
if(!require(lubridate))
install.packages("lubridate")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(ggraph)) install.packages("ggraph")
if(!require(igraph)) install.packages("igraph")
if(!require(plotrix)) install.packages("plotrix")
```

### *Tidy Text*

As already covered, tidy text is a format useful for several type of text mining problems, including sentiment and topic analysis. Some advantages of the tidy text format are:

- keeps one token (typically a word) in each row
- keeps each variable (such as a document or chapter) in a column.
- when your data is tidy, you can use a common set of tools for exploring and visualizing them

This frees you from struggling to get your data into the right format for each task and lets you focus on the questions you want to ask.

#### **Step 1:** put the text into a data frame

- the `c()` function returns a vector (a one dimensional array)

- `Paste0()` concatenates strings without spaces
- the `data_frame()` function is used for storing data tables
- The `map()` function transforms the text by applying a function to each element and returning a vector the same length
- `read_lines()` reads up to `n_max` lines from a file. ...  
`read_lines_raw()` produces a list of raw vectors
- the `Mutate` function adds new variables and preserves existing

```
quantum_words<-
  data_frame(file =
    paste0("C:\\Users\\jeff\\Documents\\VIT_Course_Material
  \\Data_Analytics_2018\\data\\",
    c("quantum_phaith.txt",
      "quantum_hope.txt",
      "quantum_love.txt")) ) %>%
  mutate(text = map(file, read_lines))
```

### Step 2: Unnest the tibble

Tibbles are new data frames that keep the features we like and drops the features that are now frustrating (i.e. converting character vectors to factors). - unnest() the tibble - remove the lines that are LaTeX crude  
- compute a line number with the mutate function

```
quantum_words <- quantum_words %>%
  unnest() %>%
  mutate(line_number = 1:n(),
        file =
          str_sub(basename(file), 1, -5))
```

### Step 3: Delete words and LaTex

- `str_detect()` detects the presence or absence of a pattern and returns a logical vector
- `!` is the logical operator for negation

### *Quantum Words*

```
quantum_words <- quantum_words %>%
  unnest() %>%
  filter(text != "%!TEX root = ind.tex") %>%
  filter(!str_detect(text, "\\\\([A-Z,a-z]\\\\)"), text != "")
```

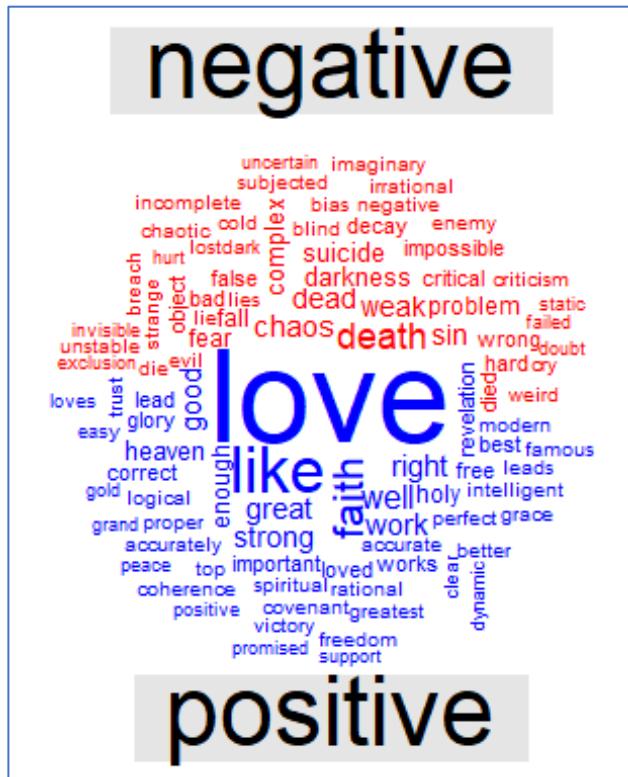
## Quantum Tokens

```
quantum_words <- quantum_words %>%
  unnest_tokens(word, text) %>%
  filter(!str_detect(word, "[0-9"]),
         word != "fismanreview",
         word != "multicolumn",
         word != "p",
         word != "_i",
         word != "al",
         word != "tabular",
         word != "ref",
         word != "cite",
         !str_detect(word, "[a-z]_"),
         !str_detect(word, ":")),
  word != "bar",
  word != "emph",
  !str_detect(word, "textless"))
quantum_words
## # A tibble: 158,482 x 3
##   file           line_number word
##   <chr>          <int>     <chr>
## 1 quantum_phaith      1 quantum
## 2 quantum_phaith      1 phaith
## 3 quantum_phaith      2 preface
## 4 quantum_phaith      3 obviously
## 5 quantum_phaith      3 either
## 6 quantum_phaith      3 i
## 7 quantum_phaith      3 do
## 8 quantum_phaith      3 not
## 9 quantum_phaith      3 know
## 10 quantum_phaith     3 how
## # ... with 158,472 more rows
```

## Quantum Cloud

```
library(reshape2)
##
## Attaching package: 'reshape2'
## The following object is masked from 'package:tidyverse':
## 
##   smiths
quantum_words %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
```

```
comparison.cloud(colors = c("red", "blue"),
                  max.words = 100)
## Joining, by = "word"
```



### Lexicon Exploration

The tidytext package contains several sentiment lexicons in the sentiments dataset. The three general-purpose lexicons are:

- AFINN from Finn Årup Nielsen
- bing from Bing Liu and collaborators
- nrc from Saif Mohammad and Peter Turney

All three of these lexicons are based on unigrams, i.e., single words.

```
library(tidytext)
sentiments
```

```

## # A tibble: 27,314 x 4
##   word      sentiment lexicon score
##   <chr>     <chr>    <chr>    <int>
## 1 abacus    trust     nrc      NA
## 2 abandon    fear      nrc      NA
## 3 abandon    negative  nrc      NA
## 4 abandon    sadness   nrc      NA
## 5 abandoned  anger    nrc      NA
## 6 abandoned  fear     nrc      NA
## 7 abandoned  negative  nrc      NA
## 8 abandoned  sadness   nrc      NA
## 9 abandonment anger   nrc      NA
## 10 abandonment fear    nrc      NA
## # ... with 27,304 more rows

```

### NRC Lexicon

These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. The nrc lexicon:

- Includes 13,901 words
- Categorizes words in a binary fashion ("yes"/"no") into categories of:

  - Positive
  - Negative
  - Anger
  - Fear
  - Joy
  - Sadness
  - Surprise
  - Trust

```

get_sentiments("nrc")
## # A tibble: 13,901 x 2
##   word      sentiment
##   <chr>
## 1 abacus    trust
## 2 abandon    fear
## 3 abandon    negative
## 4 abandon    sadness
## 5 abandoned  anger
## 6 abandoned  fear
## 7 abandoned  negative
## 8 abandoned  sadness
## 9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows

```

### AFINN Lexicon

- Includes 2,476 words
- The AFINN lexicon assigns words with a score that runs between -5 and 5 with

- negative scores indicating negative sentiment
- positive scores indicating positive sentiment.

```
get_sentiments("afinn")
## # A tibble: 2,476 x 2
##   word      score
##   <chr>    <int>
## 1 abandon     -2
## 2 abandoned    -2
## 3 abandons     -2
## 4 abducted     -2
## 5 abduction     -2
## 6 abductions    -2
## 7 abhor       -3
## 8 abhorred     -3
## 9 abhorrent     -3
## 10 abhors      -3
## # ... with 2,466 more rows
```

### Bing Lexicon

- Includes 13,901 words
- The nrc lexicon categorizes words in a binary fashion ("yes"/"no") into categories of:
  - Positive
  - Negative
  - Anger
  - Fear
  - Joy
  - Sadness
  - Surprise
  - Trust

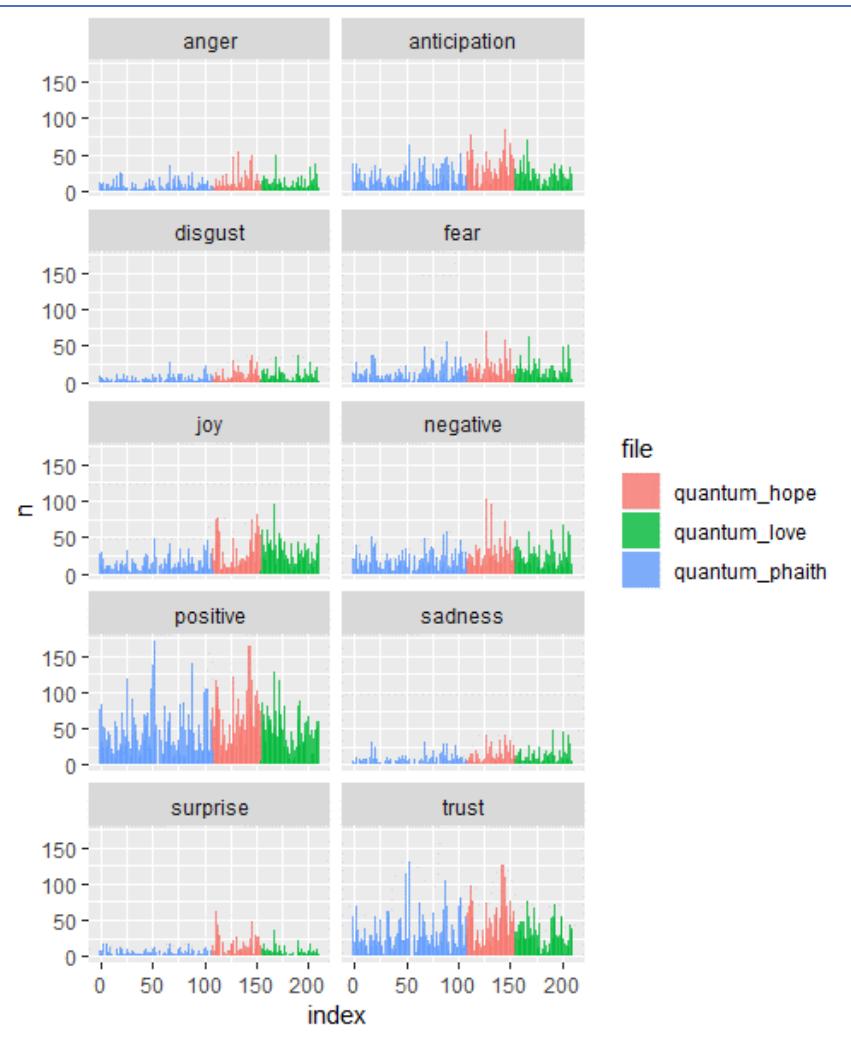
```
get_sentiments("bing")
## # A tibble: 6,788 x 2
##   word      sentiment
##   <chr>    <chr>
## 1 2-faced    negative
## 2 2-faces    negative
## 3 a+          positive
## 4 abnormal    negative
## 5 abolish    negative
## 6 abominable  negative
## 7 abominably  negative
```

```
## 8 abominate negative
## 9 abomination negative
## 10 abort negative
## # ... with 6,778 more rows
```

## Getting Sentiments with NRC

Using the nrc lexicon, let's see how the emotions of my words change between Quantum Phaith and Quantum Love.

```
quantum_words %>%
  inner_join(get_sentiments("nrc")) %>%
  group_by(index = line_number %/% 25, file, sentiment) %>
%
  summarize(n = n()) %>%
  ggplot(aes(x = index, y = n, fill = file)) +
  geom_bar(stat = "identity", alpha = 0.8) +
  facet_wrap(~ sentiment, ncol = 5)
## Joining, by = "word"
```



Based on the plot, it looks like Quantum Love is more “sentimental, as the sentiment scores are a little higher than for Quantum Phaith. Not that Quantum Phaith is much more scientific.

### *Analyzing word and document frequency: tf-idf*

A central question in text mining and natural language processing is how to quantify what a document is about. Can we do this by looking at the words that make up the document? One measure of how important a word may be is its term frequency (tf), how frequently a word occurs in a document, as we examined in Chapter 1. There are words in a

document, however, that occur many times but may not be important; in English, these are probably words like “the”, “is”, “of”, and so forth. We might take the approach of adding words like these to a list of stop words and removing them before analysis, but it is possible that some of these words might be more important in some documents than others. A list of stop words is not a very sophisticated approach to adjusting term frequency for commonly used words.

Another approach is to look at a term’s inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term’s tf-idf (the two quantities multiplied together), the frequency of a term adjusted for how rarely it is used.

The statistic tf-idf is intended to measure how important a word is to a document in a collection (or corpus) of documents, for example, to one novel in a collection of novels or to one website in a collection of websites.

### **Term frequency**

Let’s start by looking at a corpus of four books and examine first term frequency, then tf-idf. We can start just by using dplyr verbs such as group\_by() and join(). What are the most commonly used words in our book corpus? (Let’s also calculate the total words in each novel here, for later use.)

First, we load and tokenize the book corpus called :readings,” as well as remove stopwords.

```
readings <- read.csv("C:\\Users\\jeff\\Documents\\VIT_Course_Material\\Data_Analytics_2018\\data\\readings.csv",stringsAsFactor=FALSE)

readings<-readings %>% group_by(book) %>% mutate(ln=row_number())%>% unnest_tokens(word,text) %>% count(book, word, sort = TRUE) %>%ungroup()

my_stops <- c("https", "http", stopwords("en"))
readings <- readings %>%
  filter(!word %in% stop_words$word,
         !word %in% my_stops,
```

```

!word %in% str_remove_all(stop_words$word, "''"))
head(readings,5)
## # A tibble: 5 x 3
##   book           word     n
##   <chr>          <chr> <int>
## 1 Indian Philosophy world  1073
## 2 Indian Philosophy life   693
## 3 Indian Philosophy god    660
## 4 Quantum Love      love   604
## 5 Indian Philosophy buddha 566

```

### Get Term Frequencies

```

total_words <- readings %>%
  group_by(book) %>%
  summarize(total = sum(n))

book_words <- left_join(readings, total_words)
## Joining, by = "book"
book_words
## # A tibble: 31,228 x 4
##   book           word     n total
##   <chr>          <chr> <int> <int>
## 1 Indian Philosophy world  1073 93145
## 2 Indian Philosophy life   693 93145
## 3 Indian Philosophy god    660 93145
## 4 Quantum Love      love   604 12077
## 5 Indian Philosophy buddha 566 93145
## 6 Indian Philosophy reality 538 93145
## 7 Indian Philosophy nature 451 93145
## 8 Indian Philosophy soul   443 93145
## 9 Indian Philosophy upanisads 439 93145
## 10 Indian Philosophy existence 437 93145
## # ... with 31,218 more rows

```

There is one row in this `book_words` data frame for each word-book combination; `n` is the number of times that word is used in that book and `total` is the total words in that book. The usual suspects are here with the highest `n`, “the”, “and”, “to”, and so forth. The plot shows the distribution of  $n/\text{total}$  for each novel, the number of times a word appears in a novel divided by the total number of terms (words) in that book. This is exactly what “term frequency” is.

```

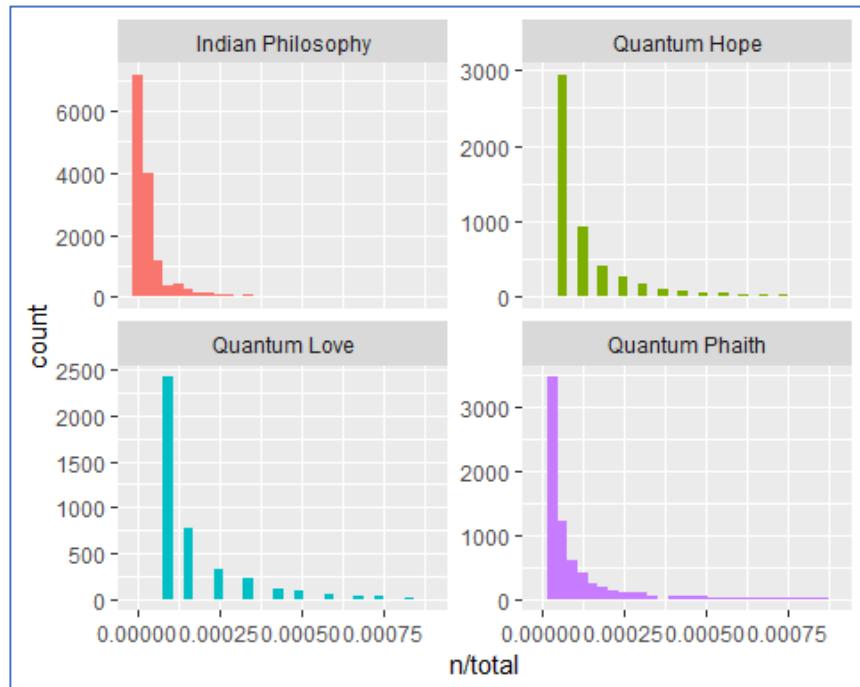
library(ggplot2)
ggplot(book_words, aes(n/total, fill = book)) +

```

```

geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, ncol = 2, scales = "free_y")
## `stat_bin()` using `bins = 30`. Pick better value with
`binwidth`.
## Warning: Removed 584 rows containing non-finite values
(stat_bin).

```



### *Zipf's law*

Distributions like those shown in the previous plot are typical in language. In fact, those types of long-tailed distributions are so common in any given corpus of natural language (like a book, or a lot of text from a website, or spoken words) that the relationship between the frequency that a word is used and its rank has been the subject of study; a classic version of this relationship is called Zipf's law, after George Zipf, a 20th century American linguist.

Zipf's law states that the frequency that a word appears is inversely proportional to its rank.

Since we have the data frame we used to plot term frequency, we can examine Zipf's law for our book collect with just a few lines of dplyr functions.

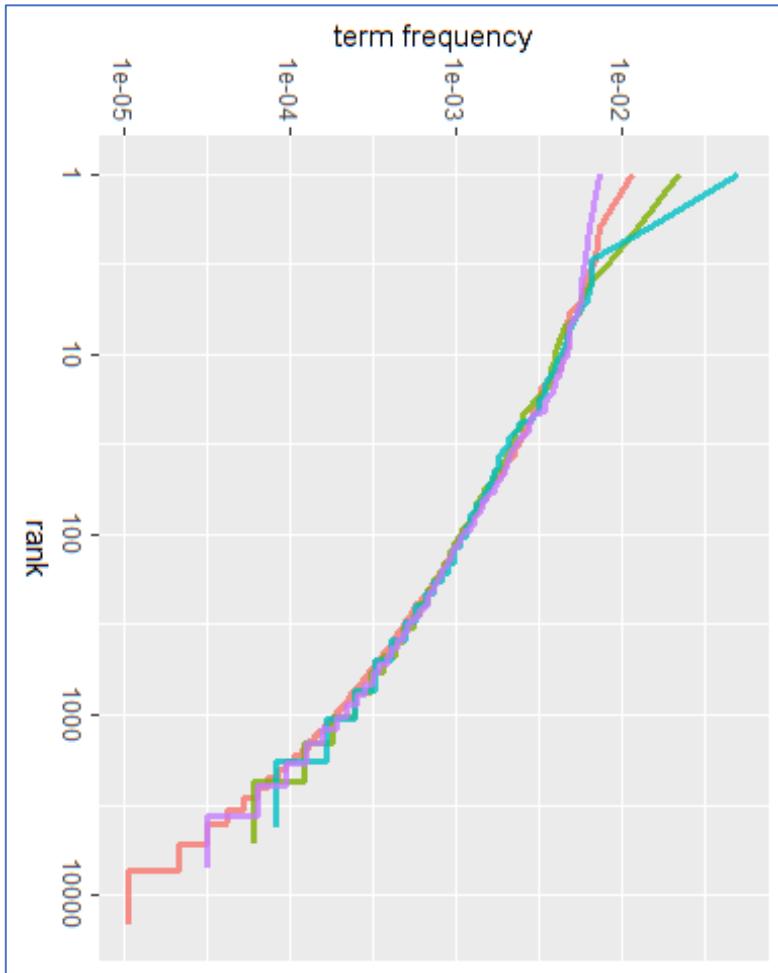
```
freq_by_rank <- book_words %>%
  group_by(book) %>%
  mutate(rank = row_number(),
        `term frequency` = n/total)
freq_by_rank
## # A tibble: 31,228 x 6
## # Groups:   book [4]
##   book           word      n total rank `term
##   <chr>          <chr>    <int> <int> <int>
## 1 Indian Philosophy world    1073 93145     1
## 2 Indian Philosophy life    693  93145     2
## 3 Indian Philosophy god     660  93145     3
## 4 Quantum Love             love    604 12077     1
## 5 Indian Philosophy buddha   566  93145     4
## 6 Indian Philosophy reality  538  93145     5
## 7 Indian Philosophy nature  451  93145     6
## 8 Indian Philosophy soul    443  93145     7
## 9 Indian Philosophy upanisads 439  93145     8
## 10 Indian Philosophy existence 437  93145    9
## # ... with 31,218 more rows
```

The rank column here tells us the rank of each word within the frequency table; the table was already ordered by n so we could use row\_number() to find the rank. Then, we can calculate the term frequency in the same way we did before. Zipf's law is often visualized by plotting rank on the x-axis and term frequency on the y-axis, on logarithmic scales. Plotting this way, an inversely proportional relationship will have a constant, negative slope.

```

freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()

```



Notice that the plot is in log-log coordinates. We see that all four of our books are similar to each other, and that the relationship between rank and frequency does have negative slope. It is not quite constant, though; perhaps we could view this as a broken power law with, say, three

sections. Let's see what the exponent of the power law is for the middle section of the rank range.

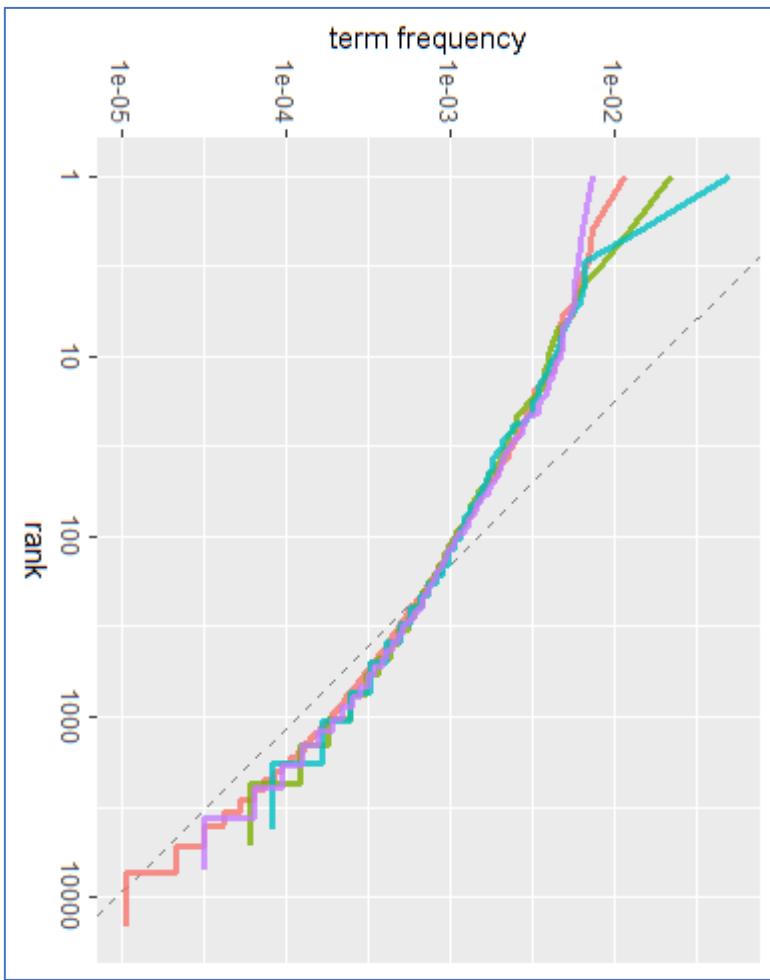
```
rank_subset <- freq_by_rank %>%
  filter(rank < 500,
         rank > 10)
lm(log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##           -1.6528      -0.6521
```

Classic versions of Zipf's law have

$$frequency = \propto \frac{1}{rank}$$

and we have in fact gotten a slope close to -1 here. Let's plot this fitted power law with the data in figure below to see how it looks

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_abline(intercept = -0.62, slope = -1.1, color = "gray50", linetype = 2) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE)
+
  scale_x_log10() +
  scale_y_log10()
```



We have found a result close to the classic version of Zipf's law for the corpus of four "philosophy" books. The deviations we see here at high rank are not uncommon for many kinds of language; a corpus of language often contains fewer rare words than predicted by a single power law. The deviations at low rank are more unusual. Our books use a lower percentage of the most common words than many collections of language. This kind of analysis could be extended to compare authors, or to compare any other collections of text; it can be implemented simply using tidy data principles.

## The bind\_tf\_idf function

The idea of tf-idf is to find the important words for the content of each document by decreasing the weight for commonly used words and increasing the weight for words that are not used very much in a collection or corpus of documents, in this case, the group of books as a whole. Calculating tf-idf attempts to find the words that are important (i.e., common) in a text, but not too common. Let's do that now.

The bind\_tf\_idf function in the tidytext package takes a tidy text dataset as input with one row per token (term), per document. One column (word here) contains the terms/tokens, one column contains the documents (book in this case), and the last necessary column contains the counts, how many times each document contains each term (n in this example). We calculated a total for each book for our explorations in previous sections, but it is not necessary for the bind\_tf\_idf function; the table only needs to contain all the words in each document.

```
book_words <- book_words %>%  
  bind_tf_idf(word, book, n)  
book_words  
## # A tibble: 31,228 x 7  
##   book          word      n total      tf     id  
##   <chr>        <chr>    <int> <int>    <dbl> <dbl>  
## 1 Indian Philosophy world    1073 93145  0.0115  0  
## 2 Indian Philosophy life     693 93145  0.00744 0  
## 3 Indian Philosophy god      660 93145  0.00709 0  
## 4 Quantum Love       love    604 12077  0.0500  0  
## 5 Indian Philosophy buddha   566 93145  0.00608 1.3  
## 6 Indian Philosophy reality   538 93145  0.00578 0  
## 7 Indian Philosophy nature   451 93145  0.00484 0  
## 8 Indian Philosophy soul      443 93145  0.00476 0  
## 9 Indian Philosophy upanisads 439 93145  0.00471 1.3  
## 10 Indian Philosophy .         439 93145  0.00653 0
```

```

## 10 Indian Philosophy existence 437 93145 0.00469 0
0
## # ... with 31,218 more rows

```

Notice that idf and thus tf-idf are zero for these extremely common words. These are all words that appear in all four books, so the idf term (which will then be the natural log of 1) is zero. The inverse document frequency (and thus tf-idf) is very low (near zero) for words that occur in many of the documents in a collection; this is how this approach decreases the weight for common words. The inverse document frequency will be a higher number for words that occur in fewer of the documents in the collection.

Let's look at terms with high tf-idf in book collection.

```

book_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))

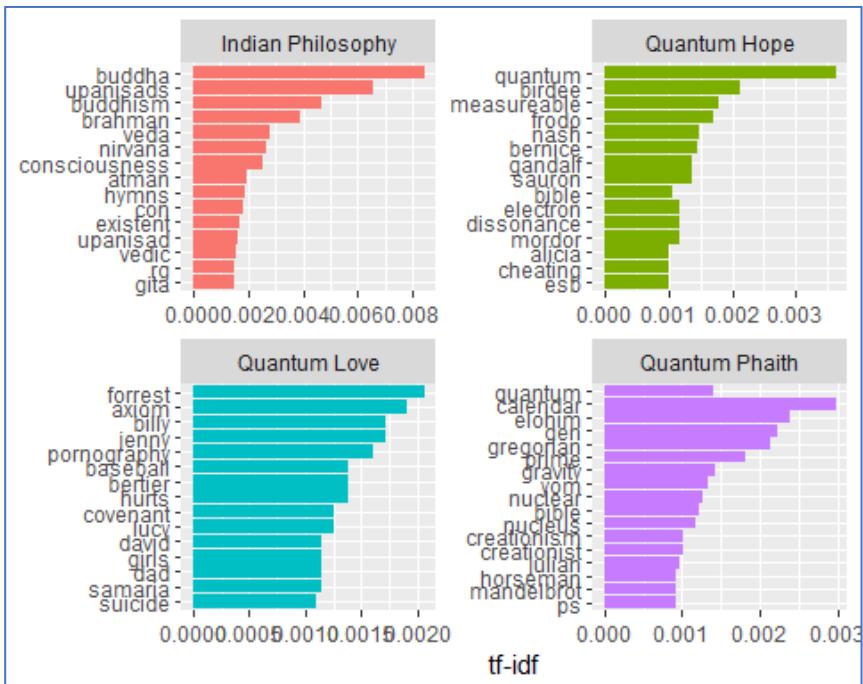
## # A tibble: 31,228 x 6
##   book           word      n     tf     idf
##   <chr>         <chr>    <int>  <dbl>  <dbl>
## 1 Indian Philosophy buddha      566  0.00608 1.39
## 2 Indian Philosophy upanisads   439  0.00471 1.39
## 3 Indian Philosophy buddhism   313  0.00336 1.39
## 4 Indian Philosophy brahman   260  0.00279 1.39
## 5 Quantum Hope      quantum  205  0.0126   0.288
## 6 Quantum Phaith     calendar 134  0.00429 0.693
## 7 Indian Philosophy veda      186  0.00200 1.39
## 8 Indian Philosophy nirvana  180  0.00193 1.39
## 9 Indian Philosophy consciousness 344  0.00369 0.693
## 10 Quantum Phaith    elohim   54   0.00173 1.39
## # ... with 31,218 more rows

```

Here we see all proper nouns, names that are in fact important in these novels. None of them occur in all of novels, and they are important, characteristic words for each text within the corpus of books.

We will examine a visualization for these high tf-idf words in the next plot.

```
book_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))
%>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()
## Selecting by tf_idf
```



We still see all proper nouns! These words are, as measured by tf-idf, the most important to each novel and most readers would likely agree. What measuring tf-idf has done here is show us that our authors used similar language across their four books. This is the point of tf-idf; it identifies words that are important to one document within a collection of documents.

## Summary

Using term frequency and inverse document frequency allows us to find words that are characteristic for one document within a collection of documents, whether that document is a novel or physics text or webpage. Exploring term frequency on its own can give us insight into how language is used in a collection of natural language, and dplyr verbs like `count()` and `rank()` give us tools to reason about term frequency. The tidytext package uses an implementation of tf-idf consistent with tidy data principles that enables us to see how different words are important in documents within a collection or corpus of documents.

## Exercises

5. Choose and download five text files from  
[https://github.com/stricje1/VIT\\_University/tree/master/Predictive\\_Modeling/tidy\\_text](https://github.com/stricje1/VIT_University/tree/master/Predictive_Modeling/tidy_text). And put I into a corpus (directory) of your local drive.
  - a. Assign a file path to the variable cname, using your local drive corpus directory file.path("C:/Users/<your name>/<subdirectory name>/.../data", "text\_files").
  - b. Create a corpus, i.e. `docs <- Corpus(DirSource(cname))`
  - c. Inspect the corpus and record the metadata.
  - d. Convert the text to lowercase and inspect your work.
  - e. Remove unnecessary words from the text.
  - f. If appropriate, combine words that should stay together.
  - g. Put the text into a *term-document matrix* and inspect it.
  - h. Create *document-term matrix* and inspect it.
  - i. Organize the terms by their frequency and put it into a matrix and save it to your working directory.
  - j. Remove sparse terms.
  - k. Check out some of the most and least frequently occurring words.
  - l. Build a frequency table.
  - m. Plot a frequency distribution of the most frequent terms.
  - n. Find correlations in the text.
  - o. Plot words using a wordcloud that occur at least X (determine X from your frequency table) times.
6. Pick your favorite song and put it into an array, `c()`, like we did earlier in the chapter.
  - a. Print out the tibble in R using the dplyr and tibble packages.
  - b. Break the text into individual tokens using tidytext's `unnest_tokens()` function.
  - c. What are the resulting word frequencies?
7. Use the collection of Associated Press newspaper articles included in the topicmodels package and analyze the data with tidy tools.

[Hint: use `data("AssociatedPress", package = "topicmodels")`.]

- a. How many documents and terms are in the data set?
- b. What the sparsity of the data?
- c. Turn the text into a data frame with one-token-per-document-per-row.
- d. Perform sentiment analysis on these newspaper articles with the approach described in this chapter.
- e. Use tables and graphs to present your results.

## CHAPTER 6 – Tweet Sentiment Analysis

### Sentiment Lexicons

One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. This is often performed using lexicons. Linguistic theories generally regard human languages as consisting of two parts: a lexicon, essentially a catalogue of a language's words (its wordstock); and a grammar, a system of rules which allow for the combination of those words into meaningful sentences. Lexicons used of sentiment analysis contain sentiment words, like “trust” and “disgust,” and all the words associated with a particular sentiment. These lexicons are “lined” together with the text under analysis by performing an inner-join with the text, the details of which we will discuss later.

Sentiment lexicons constructed via either crowdsourcing (using, for example, Amazon Mechanical Turk) or by the labor of one of the authors, and are validated using some combination of crowdsourcing again, restaurant or movie reviews, or Twitter data. Given this information, we may hesitate to apply these sentiment lexicons to styles of text dramatically different from what they were validated on, such as narrative fiction from 200 years ago. While it is true that using these sentiment lexicons with “Indian Philosophy,” for example, may give us less accurate results than with tweets sent by a contemporary writer, we still can measure the sentiment content for words that are shared across the lexicon and the text.

The three general-purpose lexicons are - AFINN from Finn Årup Nielsen, - bing from Bing Liu and collaborators, and - nrc from Saif Mohammad and Peter Turney.

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. The nrc lexicon categorizes words in a binary fashion (“yes”/“no”) into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. The bing lexicon categorizes words in a binary fashion into positive and negative

categories. The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. All of this information is tabulated in the sentiments dataset, and tidytext provides a function `get_sentiments()` to get specific sentiment lexicons without the columns that are not used in that lexicon.

Not every English word is in the lexicons because many English words are pretty neutral. It is important to keep in mind that these methods do not consider qualifiers before a word, such as in “no good” or “not true”; a lexicon-based method like this is based on unigrams only.

## Install Required Libraries

```
if(!require(tidytext)) install.packages("tidytext")
if(!require(tidyverse))
install.packages("tidyverse")
if(!require(dplyr)) install.packages("dplyr")
if(!require(tidyr)) install.packages("tidyr")
if(!require(sentimentr))
install.packages("sentimentr")
if(!require(tm)) install.packages("tm")
if(!require(readr)) install.packages("readr")
if(!require(wordcloud))
install.packages("wordcloud")
if(!require(lubridate))
install.packages("lubridate")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(ggraph)) install.packages("ggraph")
if(!require(igraph)) install.packages("igraph")
if(!require(plotrix)) install.packages("plotrix")
```

## Lexicon Example

The tidytext package contains several sentiment lexicons in the sentiments dataset.

```
library(tidytext)
sentiment
```

```

## function (text.var, polarity_dt = lexicon::hash_sentiment_jockers_rinker,
##           valence_shifters_dt = lexicon::hash_valence_shifters, hyphen = "",
##           amplifier.weight = 0.8, n.before = 5, n.after = 2,
##           question.weight = 1,
##           adversative.weight = 0.85, neutral.nonverb.like = FALSE,
##           missing_value = 0, ...)
## {
##   UseMethod("sentiment")
## }
## <bytecode: 0x000000002113b470>
## <environment: namespace:sentimentr>
```

We can look at the way specific lexicons score sentiment. The bing lexicon categorizes words in a binary fashion into positive and negative categories.

```

get_sentiments("bing")
## # A tibble: 6,788 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 2-faced   negative
## 2 2-faces   negative
## 3 a+        positive
## 4 abnormal  negative
## 5 abolish   negative
## 6 abominable negative
## 7 abominably negative
## 8 abominate  negative
## 9 abomination negative
## 10 abort    negative
## # ... with 6,778 more rows
```

The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment.

```

get_sentiments("afinn")
## # A tibble: 2,476 x 2
##   word      score
##   <chr>     <int>
## 1 abandon    -2
## 2 abandoned  -2
```

```
## 3 abandons -2
## 4 abducted -2
## 5 abduction -2
## 6 abductions -2
## 7 abhor -3
## 8 abhorred -3
## 9 abhorrent -3
## 10 abhors -3
## # ... with 2,466 more rows
```

The nrc lexicon categorizes words in a binary fashion (“yes”/“no”) into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.

```
get_sentiments("nrc")
## # A tibble: 13,901 x 2
##   word      sentiment
##   <chr>    <chr>
## 1 abacus   trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
## 6 abandoned fear
## 7 abandoned negative
## 8 abandoned sadness
## 9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

### Example 1: The Inner Join

With data in a tidy format, sentiment analysis can be done as an inner-join. This is another of the great successes of viewing text mining as a tidy data analysis task; much as removing stop words is an anti-join operation, performing sentiment analysis is an inner-join operation. For this example, we will use a collection of tweets from President Donald Trump, as the lexicons are geared more toward the analysis of tweets than the analysis of ancient philosophies.

```
setwd("C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/data/")
trump_tweets<-read.csv("trump_tweets.csv",stringsAsFactor=FALSE)
str(trump_tweets)
```

```
## 'data.frame': 1050 obs. of 2 variables:  
## $ time : chr "8/13/2018 8:57" "8/13/2018 9:21" "8/11  
/2018 1:28" "8/8/2018 10:14" ...  
## $ tweets: chr " ....such wonderful and powerful things about me - a true Champion of Civil Rights - until she got fired. Omarosa" | __truncated__ " While I know it's "not presidential" to take on a lowlife like Omarosa, and while I would rather not be doing " | __truncated__ " The big story that the Fake News Media refuses to report is lowlife Christopher Steele's many meetings with De" | __truncated__ " The Republicans have now won 8 out of 9 House Seats, yet if you listen to the Fake News Media you would think " | __truncated__ ..."
```

Observe that `trump_tweets` is already a data frame, and that there are 1050 tweets in the dataset, with line one as a header. We will look at the words with a “trust” score from the NRC lexicon. What are the most common “trust” words in `Trump_Tweets`? First, we need to take the text of the novels and convert the text to the tidy format using `unnest_tokens()`.

The next code chunk converts the time to date in the dataset, which will be ordered by date.

```
trump_tweets$date<-as.Date(trump_tweets$time, "%m/%d/%Y %H  
:%M")  
head(trump_tweets)  
##           time  
## 1 8/13/2018 8:57  
## 2 8/13/2018 9:21  
## 3 8/11/2018 1:28  
## 4 8/8/2018 10:14  
## 5 8/5/2018 7:49  
## 6 8/5/2018 7:35  
##  
tweets  
## 1 ....such wonderful and powerful things about me - a true Champion of Civil Rights - until she got fired. Omarosa had Zero credibility with the Media (they didn't want interviews) when she worked in the White House. Now that she says bad about me, they will talk to her. Fake News! [Twitter for iPhone] link  
## 2 While I know it's "not presidential" to take on a lowlife like Omarosa, and while I would rather not be doing so, this is a modern day form of communication and I k
```

```

now the Fake News Media will be working overtime to make even Wacky Omarosa look legitimate as possible. Sorry! [Twitter for iPhone] link
## 3 The big story that the Fake News Media refuses to report is lowlife Christopher Steele's many meetings with Deputy A.G. Bruce Ohr and his beautiful wife, Nelly. It was Fusion GPS that hired Steele to write the phony & discredited Dossier, paid for by Crooked Hillary & the DNC.... [Twitter for iPhone] link
## 4 The Republicans have now won 8 out of 9 House Seats, yet if you listen to the Fake News Media you would think we are being clobbered. Why can't they play it straight, so unfair to the Republican Party and in particular, your favorite President! [Twitter for iPhone] link
## 5
Too bad a large portion of the Media refuses to report the lies and corruption having to do with the Rigged Witch Hunt - but that is why we call them FAKE NEWS! [Twitter for iPhone] link
## 6 Fake News reporting, a complete fabrication, that I am concerned about the meeting my wonderful son, Donald, had in Trump Tower. This was a meeting to get information on an opponent, totally legal and done all the time in politics - and it went nowhere. I did not know about it! [Twitter for iPhone] link
## date
## 1 2018-08-13
## 2 2018-08-13
## 3 2018-08-11
## 4 2018-08-08
## 5 2018-08-05
## 6 2018-08-05

```

Next, we will tokenize the text.

```

trump_tweets<-trump_tweets %>% group_by(date) %>%
  mutate(ln=row_number())%>%
  unnest_tokens(word,tweets) %>% ungroup()
head(trump_tweets,5)
## # A tibble: 5 x 4
##   time           date      ln word
##   <chr>         <date>    <int> <chr>
## 1 8/13/2018 8:57 2018-08-13     1 such
## 2 8/13/2018 8:57 2018-08-13     1 wonderful

```

```
## 3 8/13/2018 8:57 2018-08-13      1 and
## 4 8/13/2018 8:57 2018-08-13      1 powerful
## 5 8/13/2018 8:57 2018-08-13      1 things
```

Notice that we chose the name word for the output column from unnest\_tokens(). This makes performing inner joins and anti-joins is thus easier because the sentiment lexicons and stop word datasets have columns named word.

Now that the text is in a tidy format with one word per row, we are ready to do the sentiment analysis. First, let's use the NRC lexicon and filter() for the “trust” words. Next, we will filter() the data frame with the text from the trump\_tweets for the words and then use inner\_join() to perform the sentiment analysis. What are the most common “trust” words in trump\_tweets? We'll use count() from dplyr get this answer.

```
nrc_trust <- get_sentiments("nrc") %>%
  filter(sentiment == "trust")
trump_tweets %>%
  inner_join(nrc_trust) %>%
  count(word, sort = TRUE)
## Joining, by = "word"
## # A tibble: 189 x 2
##       word         n
##   <chr>     <int>
## 1 president    63
## 2 show        31
## 3 enjoy        25
## 4 good         20
## 5 vote         20
## 6 real          19
## 7 credibility  18
## 8 money         18
## 9 trade         17
## 10 white        17
## # ... with 179 more rows
```

We see mostly positive words associated with “trust”. Now we'll look an see what “disgust” the President has.

```
nrc_disgust <- get_sentiments("nrc") %>%
  filter(sentiment == "disgust")
trump_tweets %>%
```

```

inner_join(nrc_disgust) %>%
  count(word, sort = TRUE)

## Joining, by = "word"
## # A tibble: 126 x 2
##   word      n
##   <chr>    <int>
## 1 bad        66
## 2 dishonest  49
## 3 phony      25
## 4 witch      21
## 5 dying      19
## 6 collusion  18
## 7 enemy      17
## 8 john       16
## 9 terrible   12
## 10 hate      11
## # ... with 116 more rows

```

We can also examine how sentiment changes throughout the time period (10/19/2011 to 8/13/2018). We can do this with just a handful of lines that are mostly dplyr functions. First, we find a sentiment score for each word using the Bing lexicon and `inner_join()`. Next, we count up how many positive and negative words there are in defined in each tweet. We then use `spread()` so that we have negative and positive sentiment in separate columns, and lastly calculate a net sentiment (positive - negative).

```

library(tidyr)
trump_sentiment <- trump_tweets %>% inner_join(get_sentiments("bing"))

## Joining, by = "word"

```

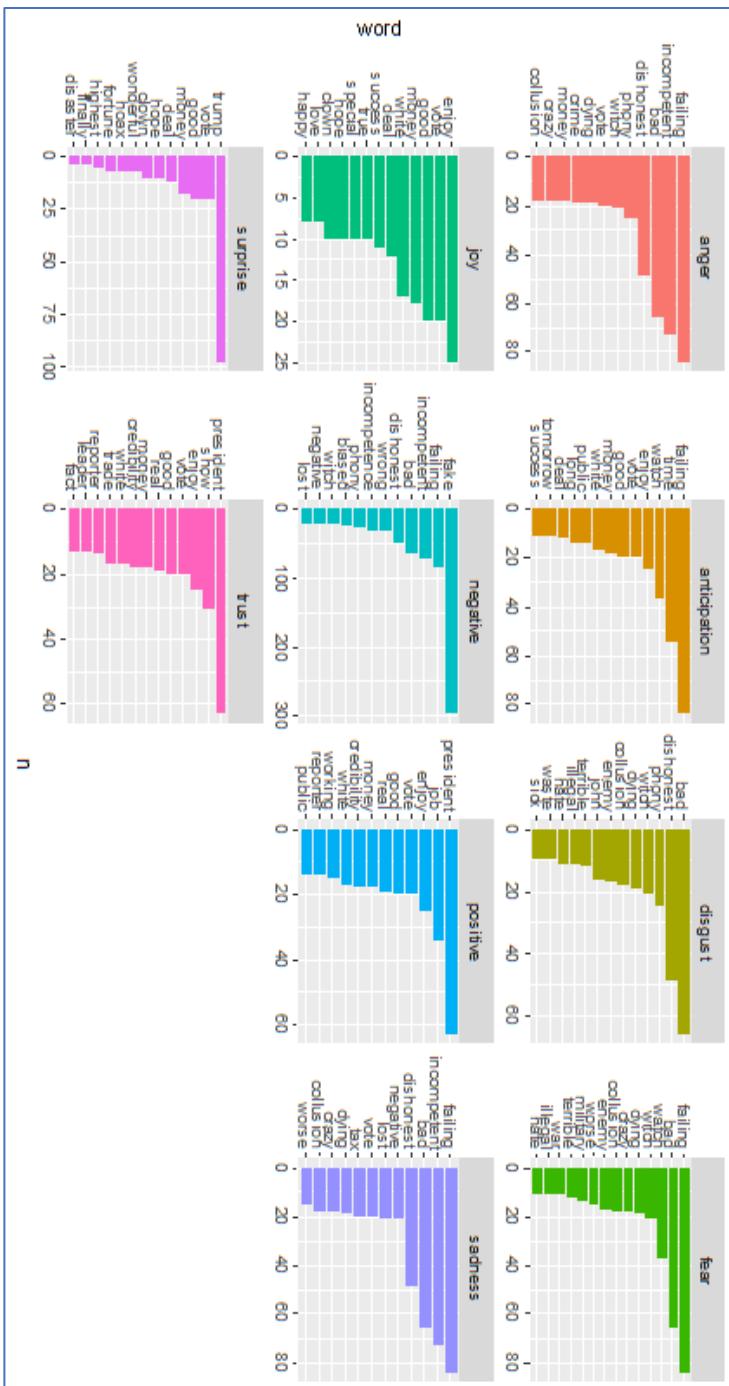
Notice that we are plotting against the index on the x-axis that keeps track of narrative time in sections of text.

```

library(ggplot2)
trump_tweets %>% inner_join(get_sentiments("nrc")) %>% count(
  word, sentiment) %>% group_by(sentiment) %>% top_n(10) %>%
  ungroup() %>% mutate(word=reorder(word,n)) %>% ggplot(aes(x=
  word, y=n, fill=sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ sentiment, scales = "free") +
  coord_flip()

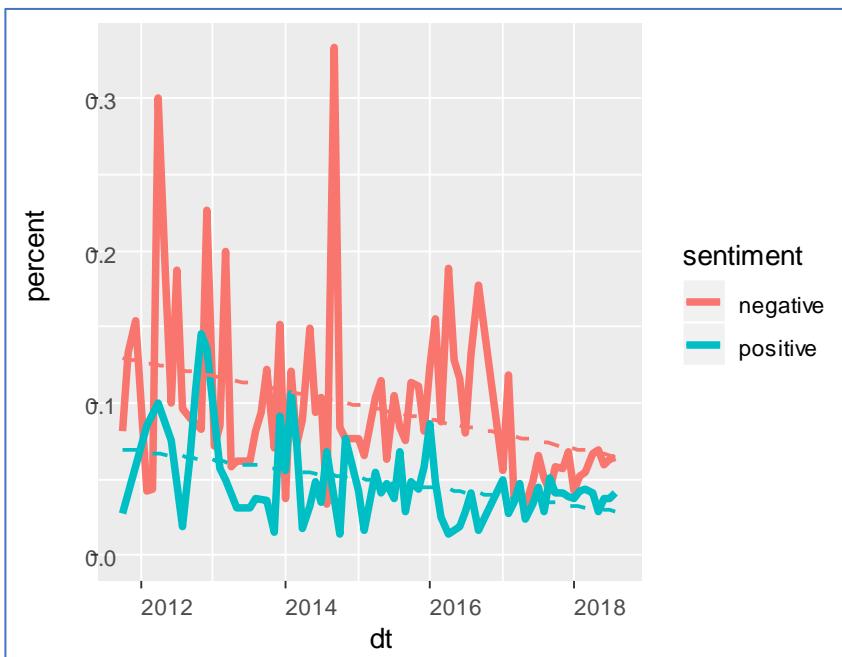
## Joining, by = "word" Selecting by n

```



## Positive & Negative Words over time

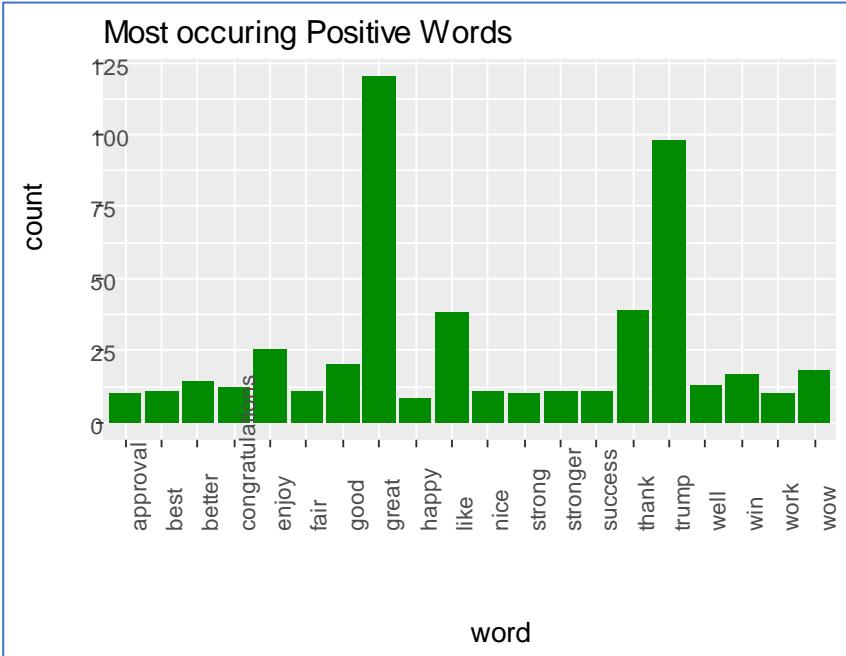
```
sentiment_by_time <- trump_tweets %>%
  mutate(dt = floor_date(date, unit = "month")) %>%
  group_by(dt) %>%
  mutate(total_words = n()) %>%
  ungroup() %>%
  inner_join(get_sentiments("nrc"))
## Joining, by = "word"
sentiment_by_time %>%
  filter(sentiment %in% c('positive','negative')) %>%
  count(dt,sentiment,total_words) %>%
  ungroup() %>%
  mutate(percent = n / total_words) %>%
  ggplot(aes(x=dt,y=percent,col=sentiment,group=sentiment))
) +
  geom_line(size = 1.5) +
  geom_smooth(method = "lm", se = FALSE, lty = 2) +
  expand_limits(y = 0)
```



```

pos_neg<-trump_sentiment %>%count(word,sentiment,sort=TRUE)
)
pos_neg %>% filter(sentiment=='positive')%>%head(20) %>%gg
plot(aes(x=word,y=n))+geom_bar(stat="identity",fill="green
4")+
  theme(axis.text.x=element_text(angle=90))+labs(title="Mo
st occurring Positive Words",y="count")

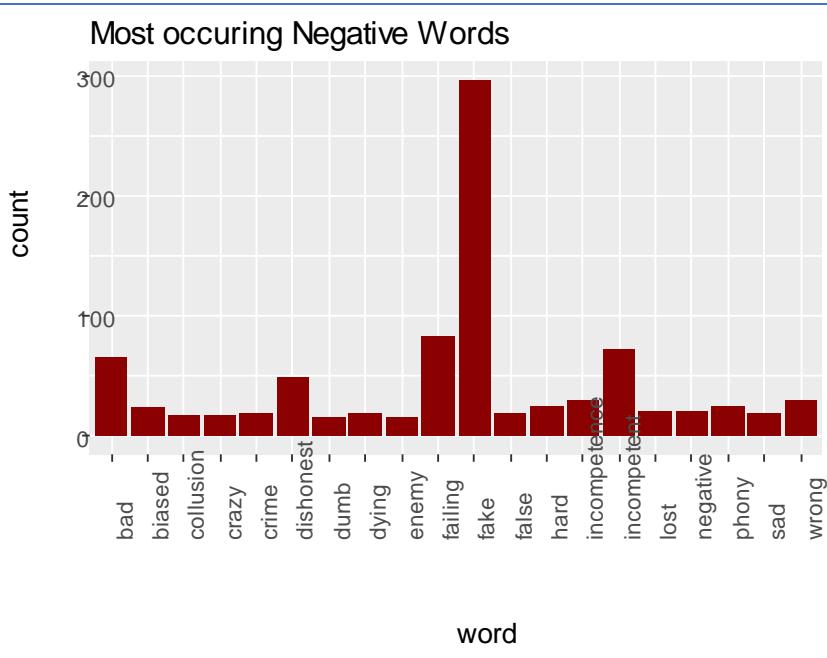
```



```

pos_neg %>% filter(sentiment=='negative')%>%head(20) %>%gg
plot(aes(x=word,y=n))+geom_bar(stat="identity",fill="red4"
)+_
  theme(axis.text.x=element_text(angle=90))+labs(title="Mo
st occurring Negative Words",y="count")

```



## Most common positive and negative words

One advantage of having the data frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment. By implementing `count()` here with arguments of both word and sentiment, we find out how much each word contributed to each sentiment.

```
bing_word_counts <- trump_tweets %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

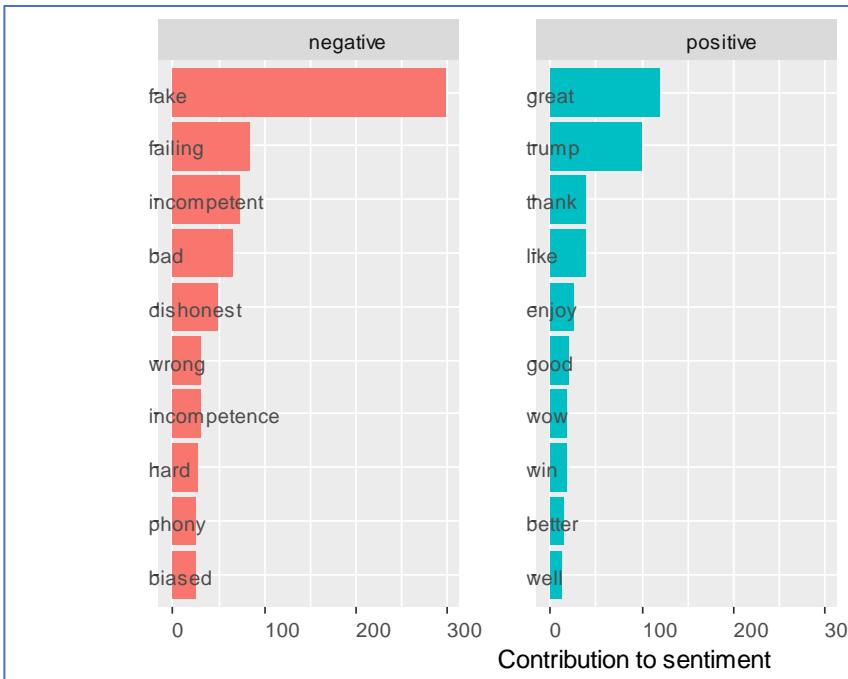
## Joining, by = "word"
bing_word_counts

## # A tibble: 543 x 3
##   word      sentiment     n
##   <chr>      <chr>    <int>
## 1 fake      negative    298
## 2 great     positive    120
## 3 trump     positive    98
## 4 failing   negative    84
## 5 incompetent negative   73
```

```
## 6 bad negative 66
## 7 dishonest negative 49
## 8 thank positive 39
## 9 like positive 38
## 10 incompetence negative 31
## # ... with 533 more rows
```

This can be shown visually, and we can pipe straight into ggplot2, if we like, because of the way we are consistently using tools built for handling tidy data frames.

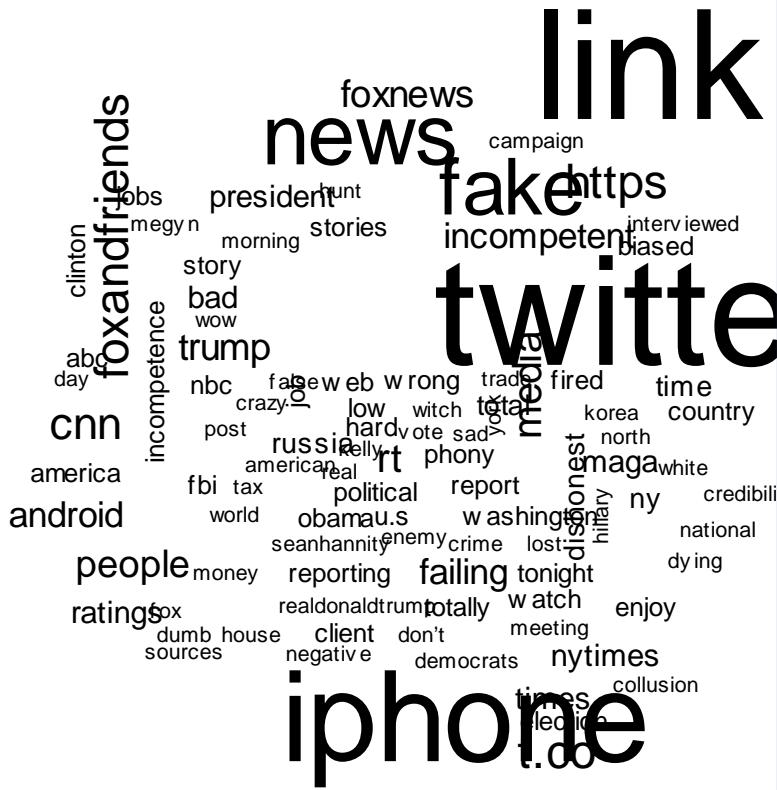
```
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(y = "Contribution to sentiment",
       x = NULL) +
  coord_flip()
## Selecting by n
```



## Wordclouds

We've seen that this tidy text mining approach works well with ggplot2 but having our data in a tidy format is useful for other plots as well. For example, consider the wordcloud package, which uses base R graphics. Let's look at the most common words in trump\_tweets, but this time as a wordcloud.

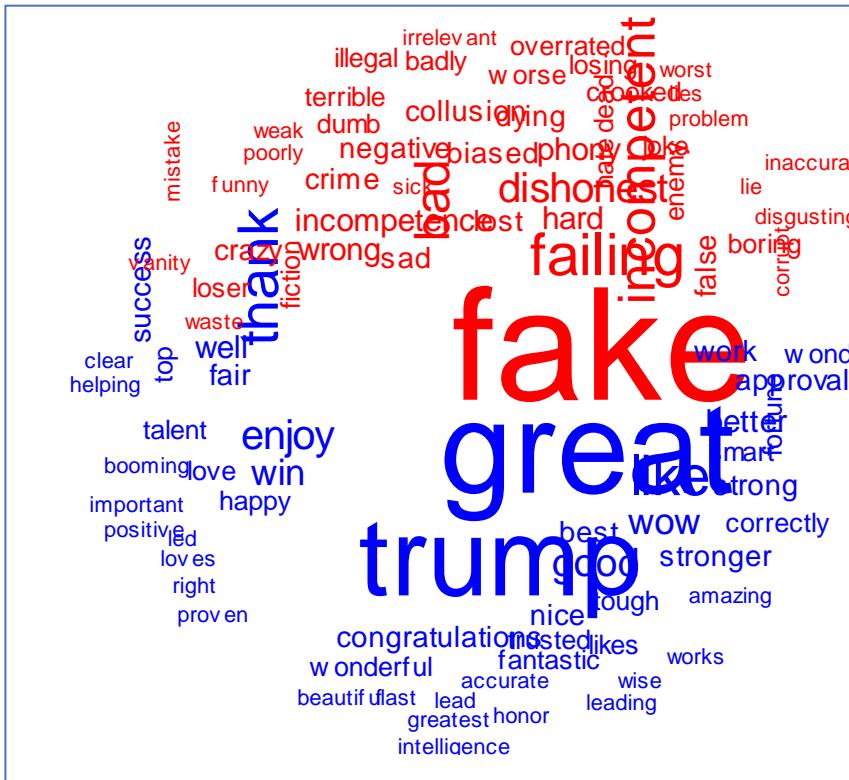
```
library(wordcloud)
trump_tweets %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```



In other functions, such as `comparison.cloud()`, you may need to turn the data frame into a matrix with `reshape2`'s `acast()`. Let's do the sentiment analysis to tag positive and negative words using an inner join, then find the most common positive and negative words. Until the step where we need to send the data to `comparison.cloud()`, this can all be done with joins, piping, and `dplyr` because our data is in tidy format.

```
library(reshape2)
##
## Attaching package: 'reshape2'
## The following object is masked from 'package:tidyverse':
##     smiths
trump_tweets %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
```

```
comparison.cloud(colors = c("red", "blue"),
                  max.words = 100)
## Joining, by = "word"
```



## Word Frequencies

In this next section, we want to look at `trump_tweets` from the perspective of Mr. Trump's different roles, assuming that might be some sentiment differences between Candidate Trump and President Trump. The column "role" contains a flag to delineate Mr. Trump's different roles. After January 21, 2017, his role is "president" and prior to that, his role is "candidate."

The next code chunk removes user-made and standard English stopwords.

```
library(tm)
library(tidyr)
my_stops <- c("https", "a", "rt", "t.co", "for", "they", s
```

```

topwords("en"))
trump_tweets <- trump_tweets %>%
  filter(!word %in% stop_words$word,
         !word %in% my_stops,
         !word %in% str_remove_all(stop_words$word, "’"))

```

Now we can calculate word frequencies for each Donald Trump role. First, we group by person and count how many times each role used each word. Then we use `left_join()` to add a column of the total number of words used by each role.

```

frequency <- trump_tweets %>%
  group_by(role) %>%
  count(word, sort = TRUE) %>%
  left_join(trump_tweets %>%
    group_by(role) %>%
    summarise(total = n())) %>%
  mutate(freq = n/total)
## Joining, by = "role"
frequency
## # A tibble: 3,719 x 5
## # Groups:   role [2]
##   role     word       n total     freq
##   <chr>    <chr>     <int> <int>     <dbl>
## 1 president twitter      584  9947  0.0587
## 2 president link        575  9947  0.0578
## 3 president iphone      541  9947  0.0544
## 4 president news        320  9947  0.0322
## 5 president fake        298  9947  0.0300
## 6 president foxandfriends 143  9947  0.0144
## 7 president media       109  9947  0.0110
## 8 president cnn          106  9947  0.0107
## 9 candidate link         91   3178  0.0286
## 10 president foxnews      89  9947  0.00895
## # ... with 3,709 more rows

```

This is a nice and tidy data frame but we would actually like to plot those frequencies on the x- and y-axes of a plot, so we will need to use `spread()` from `tidyverse` to make a differently shaped data frame.

```

if(!require(tidyverse)) install.packages("tidyverse")
library(tidyverse)
frequency <- frequency %>%
  select(role, word, freq) %>%
  spread(role, freq) %>%

```

```

arrange(president, candidate)
frequency

## # A tibble: 3,210 x 3
##   word      candidate president
##   <chr>     <dbl>     <dbl>
## 1 18        0.000315  0.000101
## 2 admitted  0.000315  0.000101
## 3 agreed    0.000315  0.000101
## 4 allowing  0.000315  0.000101
## 5 articles  0.000315  0.000101
## 6 attempt   0.000315  0.000101
## 7 average   0.000315  0.000101
## 8 basic     0.000315  0.000101
## 9 bob       0.000315  0.000101
## 10 buy      0.000315  0.000101
## # ... with 3,200 more rows

```

Now this is ready for us to plot. We will use `geom_jitter()` so that we don't see the discreteness at the low end of frequency as much, and `check_overlap = TRUE` so the text labels don't all print out on top of each other (only some will print).

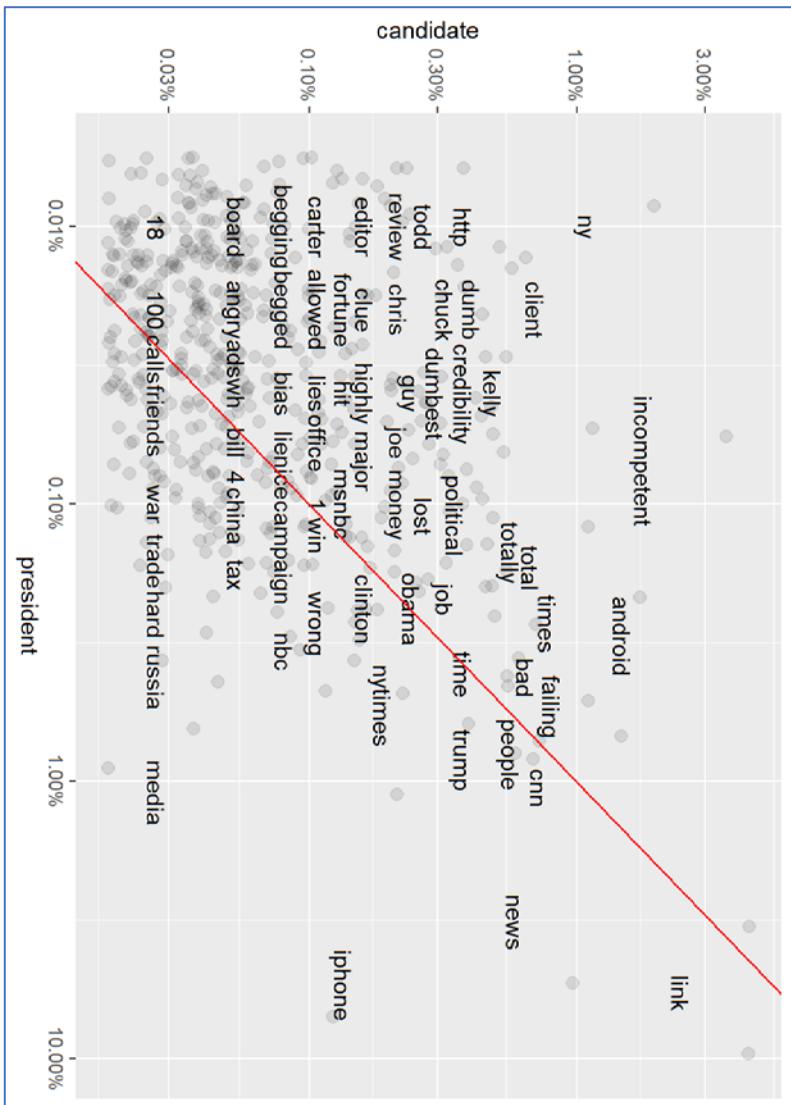
```

if(!require(scales)) install.packages("scales")
## Loading required package: scales
##
## Attaching package: 'scales'
## The following object is masked from 'package:plotrix':
## 
##     rescale
## The following object is masked from 'package:purrr':
## 
##     discard
## The following object is masked from 'package:readr':
## 
##     col_factor
library(scales)
ggplot(frequency, aes(president, candidate)) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  geom_abline(color = "red")

```

```
## Warning: Removed 2701 rows containing missing values (geom_point).
```

```
## Warning: Removed 2701 rows containing missing values (geom_text).
```



Words near the line are used with about equal frequencies by President Trump and Candidate Trump, while words far away from the line are

used much more by one person compared to the other. Words, hashtags, and usernames that appear in this plot are ones that we have both used at least once in tweets.

## Comparing word usage

We just made a plot comparing raw word frequencies over our whole Twitter histories; now let's find which words are more or less likely to come from each Mr. Trump's roles, using the log odds ratio.

Here, we count how many times each role uses each word and keep only the words used more than 10 times. After a `spread()` operation, we can calculate the log odds ratio for each word, using log odd ratio =

$$\ln((n+1/(total+1))president) / ((n+1)/(total+1))candidate)$$

where n is the number of times the word in question is used by each role and the total indicates the total words for each role.

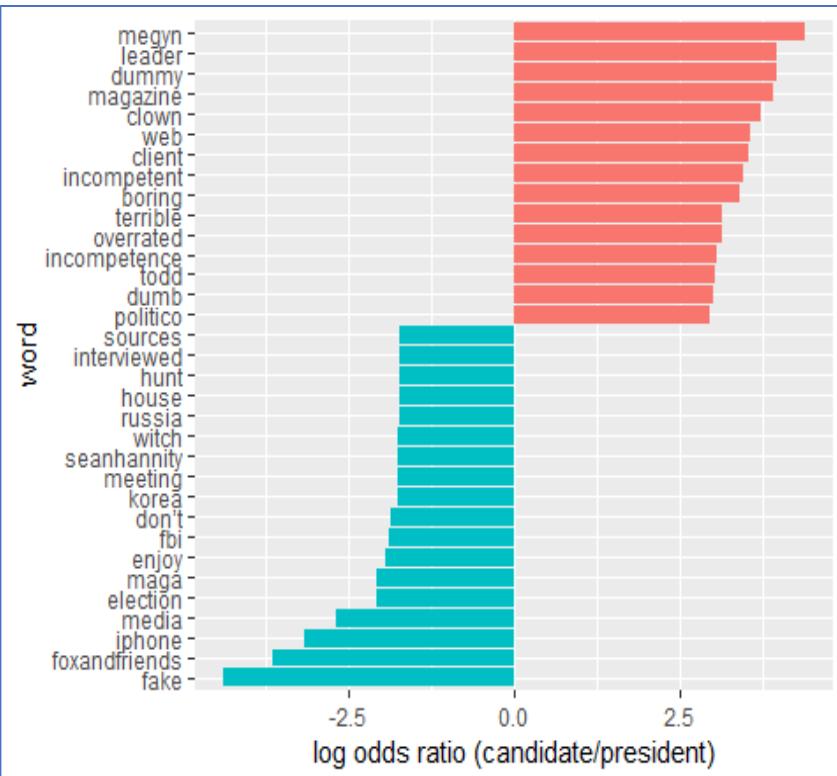
```
my_stops <- c("https", "http", "a", "rt", "t.co", "for", "they", "north", "ny", stopwords("en"))
trump_tweets <- trump_tweets %>%
  filter(!word %in% stop_words$word,
         !word %in% my_stops,
         !word %in% str_remove_all(stop_words$word, "'"))
word_ratios <- trump_tweets %>%
  filter(!str_detect(word, "^@")) %>%
  count(word, role) %>%
  group_by(word) %>%
  filter(sum(n) >= 10) %>%
  ungroup() %>%
  spread(role, n, fill = 0) %>%
  mutate_if(is.numeric, funs(. + 1) / (sum(.) + 1))) %>%
  mutate(logratio = log(candidate / president)) %>%
  arrange(desc(logratio))
word_ratios %>%
  arrange(abs(logratio))

## # A tibble: 203 x 4
##   word     candidate president logratio
##   <chr>      <dbl>     <dbl>     <dbl>
## 1 cnn       0.0191    0.0192   -0.00422
## 2 people    0.0150    0.0145    0.0330
## 3 polls     0.00205   0.00197   0.0371
## 4 china     0.00205   0.00215   -0.0499
## 5 coverage   0.00205   0.00215   -0.0499
```

```
## 6 million    0.00205   0.00215 -0.0499
## 7 campaign   0.00273   0.00287 -0.0499
## 8 world      0.00273   0.00287 -0.0499
## 9 clinton    0.00478   0.00449  0.0634
## 10 u.s       0.00478   0.00521 -0.0850
## # ... with 193 more rows
```

What are some words that have been about equally likely to come from Candidate Trump or President Trump?

```
word_ratios %>%
  group_by(logratio < 0) %>%
  top_n(15, abs(logratio)) %>%
  ungroup() %>%
  mutate(word = reorder(word, logratio)) %>%
  ggplot(aes(word, logratio, fill = logratio < 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  ylab("log odds ratio (candidate/president)") +
  scale_fill_discrete(name = "", labels = c("candidate", "president"))
```



## Changes in word use

The section above looked at overall word use, but now let's ask a different question. Which words' frequencies have changed the fastest in Mr. Trump's Twitter feeds? Or to state this another way, which words has he tweeted about at a higher or lower rate as time has passed in his different roles? To do this, we will define a new time variable in the data frame that defines which unit of time each tweet was posted in. We can use `floor_date()` from lubridate to do this, with a unit of our choosing; using 1 month seems to work well for set of tweets.

After we have the time bins defined, we count how many times each of us used each word in each time bin. After that, we add columns to the data frame for the total number of words used in each time bin by each role and the total number of times each word was used by each role. We can then `filter()` to only keep words used at least some minimum number of times (30, in this case).

```

words_by_time <- trump_tweets %>%
  filter(!str_detect(word, "^\@")) %>%
  mutate(time_floor = floor_date(date, unit = "1 month"))
%>%
  count(time_floor, role, word) %>%
  group_by(role, time_floor) %>%
  mutate(time_total = sum(n)) %>%
  group_by(role, word) %>%
  mutate(word_total = sum(n)) %>%
  ungroup() %>%
  rename(count = n) %>%
  filter(word_total > 30)
words_by_time
# A tibble: 482 x 6
  time_floor role   word     count time_total word_total
  <date>      <chr> <chr>    <int>      <int>      <int>
1 2011-10-01 candi~ incom~     2          24         66
2 2011-10-01 candi~ link      2          24         91
3 2011-11-01 candi~ incom~     1          9          66
4 2011-11-01 candi~ link      1          9          91
5 2012-06-01 candi~ link      1          20         91
6 2012-06-01 candi~ twitt~     1          20         86
7 2012-08-01 candi~ incom~     1          28         66
8 2012-08-01 candi~ link      1          28         91
9 2012-09-01 candi~ incom~     4          49         66
10 2012-09-01 candi~ link     4          49         91
# ... with 472 more rows

```

Each row in this data frame corresponds to one role using one word in a given time bin. The count column tells us how many times that role used that word in that time bin, the time\_total column tells us how many words that role used during that time bin, and the word\_total column tells us how many times that person used that word over the whole year. This is the data set we can use for modeling.

We can use nest() from tidyverse to make a data frame with a list column that contains little miniature data frames for each word. Let's do that now and take a look at the resulting structure.

```

nested_data <- words_by_time %>%
  nest(-word, -role)
nested_data

## # A tibble: 25 x 3
##   role      word       data
##   <chr>    <chr>     <list>
## 1 candidate incompetent <tibble [31 x 4]>
## 2 candidate link       <tibble [39 x 4]>
## 3 candidate twitter    <tibble [36 x 4]>
## 4 candidate android    <tibble [29 x 4]>
## 5 president cnn        <tibble [18 x 4]>
## 6 president failing    <tibble [15 x 4]>
## 7 president fake       <tibble [19 x 4]>
## 8 president foxnews    <tibble [19 x 4]>
## 9 president iphone     <tibble [19 x 4]>
## 10 president link      <tibble [20 x 4]>
## # ... with 15 more rows

```

This data frame has one row for each role-word combination; the data column is a list column that contains data frames, one for each combination of role and word. Let's use `map()` from `purrr` (Henry and Wickham 2018) to apply our modeling procedure to each of those little data frames inside our big data frame. This is count data so let's use `glm()` with `family = "binomial"` for modeling.

```

if(!require(purrr)) install.packages("purrr")
library(purrr)
nested_models <- nested_data %>%
  mutate(models = map(data, ~ glm(cbind(count, time_total)
~ time_floor, .,
                           family = "binomial")))
nested_models

## # A tibble: 25 x 4
##   role      word       data           models
##   <chr>    <chr>     <list>         <list>
## 1 candidate incompetent <tibble [31 x 4]> <S3: glm>
## 2 candidate link       <tibble [39 x 4]> <S3: glm>
## 3 candidate twitter    <tibble [36 x 4]> <S3: glm>
## 4 candidate android    <tibble [29 x 4]> <S3: glm>
## 5 president cnn        <tibble [18 x 4]> <S3: glm>
## 6 president failing    <tibble [15 x 4]> <S3: glm>
## 7 president fake       <tibble [19 x 4]> <S3: glm>
## 8 president foxnews    <tibble [19 x 4]> <S3: glm>
## 9 president iphone     <tibble [19 x 4]> <S3: glm>

```

```
## 10 president link      <tibble [20 x 4]> <S3: glm>
## # ... with 15 more rows
```

Now notice that we have a new column for the modeling results; it is another list column and contains `glm` objects. The next step is to use `map()` and `tidy()` from the `broom` package to pull out the slopes for each of these models and find the important ones. We are comparing many slopes here and some of them are not statistically significant, so let's apply an adjustment to the p-values for multiple comparisons.

```
if(!require(broom)) install.packages("broom")
## Loading required package: broom
library(broom)
library(tidyr)
slopes <- nested_models %>%
  unnest(map(models, tidy)) %>%
  filter(term == "time_floor") %>%
  mutate(adjusted.p.value = p.adjust(p.value))
```

Now let's find the most important slopes. Which words have changed in frequency at a moderately significant level in our tweets?

```
top_slopes <- slopes %>%
  filter(adjusted.p.value < 0.05)
top_slopes
```

role	word	term	estimate	std.error
<chr>	<chr>	<chr>	<dbl>	<dbl>
candidate	incompetent	time_floor	-0.0008457277	0.0002855362
candidate	link	time_floor	-0.0008922026	0.0002349048
candidate	twitter	time_floor	-0.0009304188	0.0002706097
candidate	android	time_floor	-0.0014821153	0.0004195812
president	failing	time_floor	-0.0042022246	0.0008896979
president	iphone	time_floor	-0.0010930502	0.0003032900
president	link	time_floor	-0.0013127521	0.0002774002
president	nytimes	time_floor	-0.0058712637	0.0010400612
president	twitter	time_floor	-0.0011893092	0.0002756080
president	foxandfriends	time_floor	-0.0034668078	0.0005628461

1-10 of 10 rows | 1-5 of 8 columns

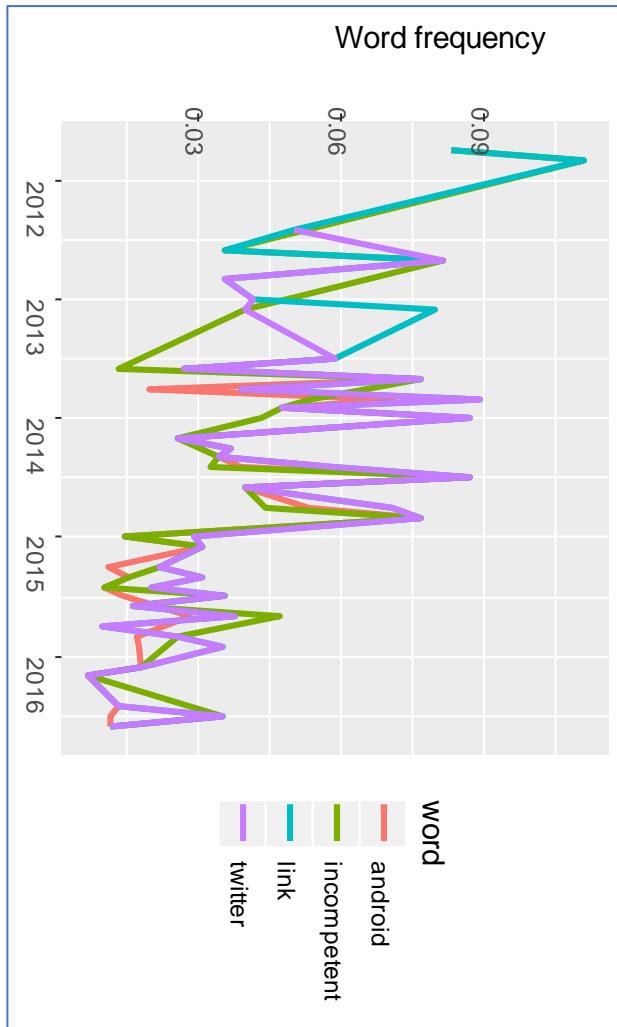
To visualize our results, we can plot these words' use for both Candidate Trump and President Trump over tweet-period.

```
words_by_time %>%
  inner_join(top_slopes, by = c("word", "role")) %>%
  filter(role == "candidate") %>%
```

```

ggplot(aes(time_floor, count/time_total, color = word))
+
  geom_line(size = 1.3) +
  labs(x = NULL, y = "Word frequency")

```



Now let's plot words that have changed frequency in President Trump's tweets.

```

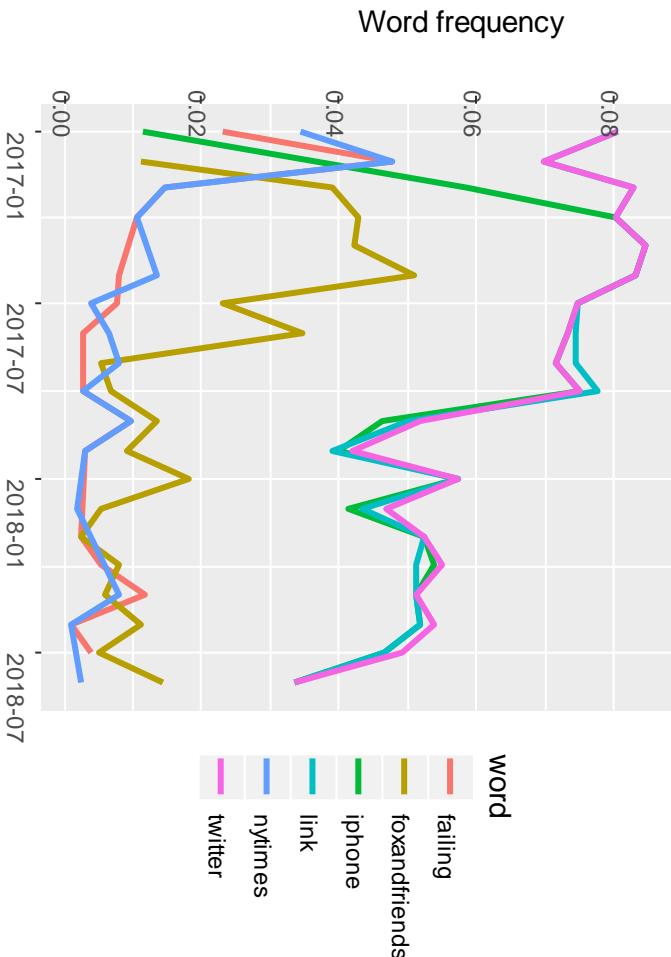
words_by_time %>%
  inner_join(top_slopes, by = c("word", "role")) %>%
  filter(role == "president") %>%

```

```

ggplot(aes(time_floor, count/time_total, color = word))
+
  geom_line(size = 1.3) +
  labs(x = NULL, y = "Word frequency")

```



## Example 2: IsisFanboy

So-called ISIS Fanboys have been actively tweeting all over the world. A fanboy is a boy or man who ardently supports a single hobby, ideology, movement, etc. Isis released TweetMovie from twitter, a “normal” day when two Isis operatives murdered a priest saying mass in a French church. The two attackers, who said they were from the so-called Islamic

State (IS), slit Fr Jacques Hamel's throat during a morning Mass, according to French officials. (BBC, 2016)

Fanboy tweets obviously would suggest certain sentiments. However, there are also many tweets that perhaps counterpoise the ISIS fanboy tweets. Kaggle release a selection of data from the site and made it available to Kaggle users. You can find the dataset in my GitHub repository, with the filename: ISISFanboy. This data set is intended to be a counterpoise to the *How Isis Uses Twitter* data set. That data set contains 17k tweets alleged to originate with “100+ pro-ISIS fanboys”. This is not a perfect counterpoise as it almost surely contains a small number of pro-Isis fanboy tweets.

**Our goal** in this chapter is to perform a counterpoise analysis to pro-ISIS tweets. A counterpoise provides a balance or backdrop against which to measure a primary object, in this case the original pro-Isis data. So, if anyone wants to discriminate between pro-Isis tweets and other tweets concerning Isis we will need to model the original pro-Isis data or signal against the counterpoise which is signal + noise.

### **Loading the Data**

It is always a good idea to make sure you place data in the R working directory. This makes it easy to load data and locate your results—it keeps things organized. To check the working directory type:

```
getwd()
```

```
[1] "C:/Users/jeff/Documents/VIT_University/IsisFanboy"
```

If you are in the directory you want to be in, you do not need to do anything. However, if you want to be in a different directory, the setwd() command, like:

```
setwd("C:/Users/jeff/Documents/VIT_University/IsisFanboy/")
```

Note that if you begin your analysis by setting up a project as described on pages XX-YY. By doing this, the working directory will be set to your project directory by default.

Now we load the data using the read.csv() function:

```
tweets<-read.csv("AboutIsis.csv",stringsAsFactor=FALSE)  
str(tweets)
```

Next, we compactly display the internal structure of our data (an R object), as a diagnostic function and an alternative to summary():

```
str(tweets) # output not shown
```

Ideally, only one line for each ‘basic’ structure is displayed. It is especially well suited to compactly display the (abbreviated) contents of nested lists. The idea is to give reasonable output for any R object.

### **Fixing the dates**

We now use a Functions to convert between character representations and objects of class “Date” representing calendar dates. In other words, the dates in the CSV file are entered as character strings and we must convert them to date data, using day/month/year and hour/minute/second format.

```
tweets$date<-as.Date(tweets$time, "%d/%m/%Y %H:%M:%S")
head(tweets)
[partial output]
##           date
## 1 2016-11-07
## 2 2016-11-07
## 3 2016-11-07
## 4 2016-11-07
## 5 2016-11-07
## 6 2016-11-07
```

### **Tokenization**

Tokenization is the process of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens. Tokens can be individual words, phrases or even whole sentences. In the process of tokenization, some characters like punctuation marks are discarded. Consequently, it appears that we are “undoing” what we did in the previous step with the dates. This is not the case, however. Rather, we are taking the date strings, converting them to dates, breaking them into dates and times, and then creating one string for dates and another for times. This is not entirely necessary since all the dates are the same—we could have just pulled out the times. Note that the sequence of tweets could be important.

```
tweets$date<-as.Date(tweets$time, "%d/%m/%Y %H:%M:%S")
tidy_tweets<-tweets %>%group_by(name,username,tweetid)%>%
  mutate(ln=row_number())%>%
  unnest_tokens(word,tweets)%>%
```

```
ungroup()  
head(tidy_tweets, 5)
```

The result from this process yields dates like:

```
1 7/11/2016 8:45:39 AM  
2 7/11/2016 8:45:39 AM  
3 7/11/2016 8:45:38 AM  
4 7/11/2016 8:45:38 AM  
5 7/11/2016 8:45:37 AM  
6 7/11/2016 8:45:36 AM
```

Now, we split the dates and times. Note that times include the date, like 7/11/2016 8:45:39 AM, else we would not know what day the time refers to.

```
t_wrd<-tidy_tweets %>%count(word, sort=TRUE)  
head(t_wrd, 5)
```

From `head(t_wrd, 5)`, the output shows what we intended:

```
# A tibble: 5 x 7  
  name    username tweetid      time       date  
  <chr>   <chr>     <dbl>      <chr>      <date>  
1 Sean Feri~ ferigan 7.52e17 7/11/2016 8:4~ 2016-11-07  
2 Sean Feri~ ferigan 7.52e17 7/11/2016 8:4~ 2016-11-07  
3 Sean Feri~ ferigan 7.52e17 7/11/2016 8:4~ 2016-11-07  
4 Sean Feri~ ferigan 7.52e17 7/11/2016 8:4~ 2016-11-07  
5 Sean Feri~ ferigan 7.52e17 7/11/2016 8:4~ 2016-11-07
```

Note that `tibbles` are a modern take on data frames. They keep the features that have stood the test of time and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors). `tidy_tweets` contains all words like 'the', 'is', 'are' etc. So, we'll take only the sentimental words by joining with lexicons.

### Removing common words

The dataframe, `tidy_tweets`, contains all words like 'the', 'is', 'are' etc. So, let's get only the sentimental words by joining with lexicons. One of the readily available lexicons in R is Bing Lexicon. The Bing lexicon categorizes words in a binary fashion into positive and negative categories.

```
tweets_sentiment<-tidy_tweets%>% inner_join(get_sentiments  
("bing"))
```

Some of the words in tidy\_tweets include:

```
tidy_tweets$word  
## [2] "influence"  
## [3] "on"  
## [4] "the"  
## [5] "decline"
```

When we applied the step giving us t\_wrd, the output looked like this:

```
t_wrd  
## # A tibble: 90,942 x 2  
##   word      n  
##   <chr>    <int>  
## 1 isis    113117  
## 2 ã        93958  
## 3 rt      86464  
## 4 the     67765  
## 5 à        58118
```

Note that these are some of the words that will be stripped out by the inner join with the lexicon.

When we apply the inner join, we get:

```
tweets_sentiment$word  
[1] "truthful"       "easy"          "promise"  
[4] "victory"        "dislike"        "attack"  
[7] "available"      "breaking"       "supports"  
[10] "best"           "drastically"    "support"  
[13] "killing"        "love"          "holy"
```

Notice that words like “on”, “the”, ã, rt, and “as” no longer appear.

### **Word frequencies and word clouds**

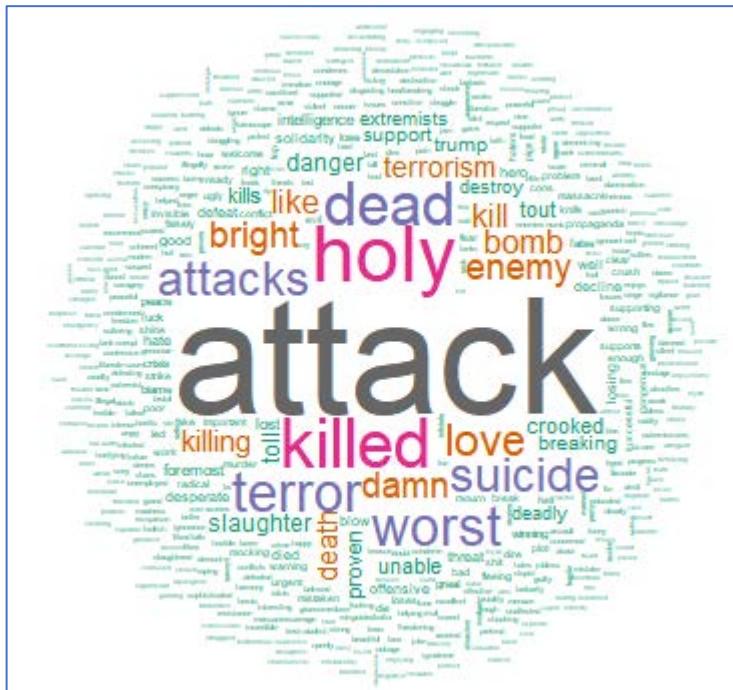
Now we want to start analyzing our data. We start by examining word frequencies. A good graphical representation of word frequencies is the “word cloud.” We begin by getting a count of the words that remain in our dataset.

```
tot_wrkd<-tweets_sentiment%>%count(word,sort=TRUE)

## A tibble: 1,609 x 2
##   word      n
##   <chr>    <int>
## 1 killed    1322
## 2 breaking   558
## 3 attack     539
## 4 like       316
## 5 support    236
```

Now we can generate a word cloud. Word clouds show all the words (unless you restrict the number or words) and represent the frequencies by the sizes of the words within the cloud. Notice the relative sized of “killed”, “attack”, and “death.” See if you can find “peace” and “destroy.”

```
wordcloud(tot_wrd$word,tot_wrd$n, min.freq =25,  
          scale=c(5, .2), random.order = FALSE,  
          random.color = FALSE,  
          colors = brewer.pal(8, "Dark2"))
```

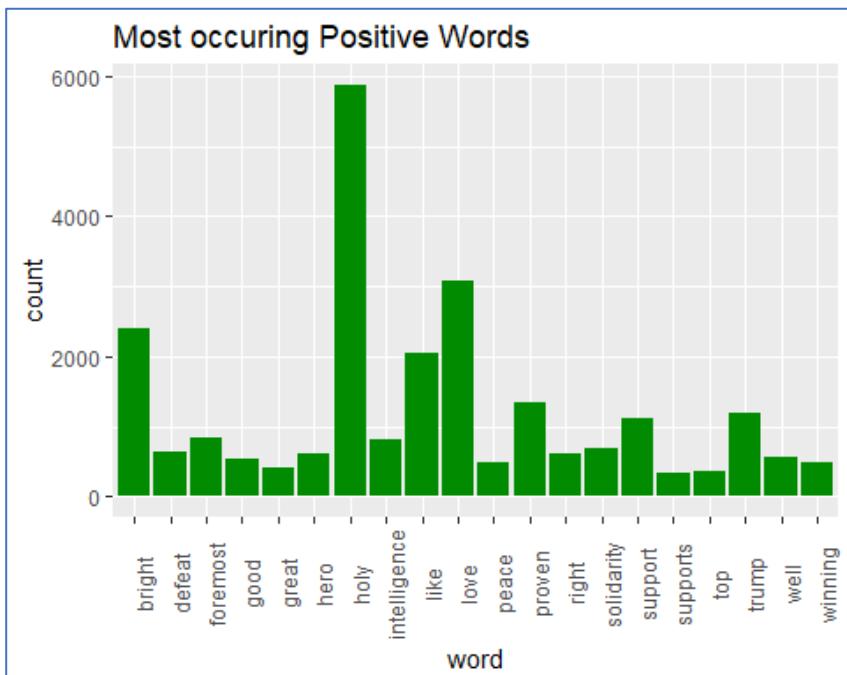


```
pos_neg<-tweets_sentiment %>%count(word,sentiment,sort=TRUE)
```

## Plotting the most occuring positive and negative words

Next, we will construct bar charts depicting the most frequent words in the corpus of ISIS related text messages. Here we use ggplot, which allows us to aesthetics, specify plt type, designate plot theme, and designate labels. For instance, we use `labs()` to apply the label “Most occurring Positive Words.” Also, we specify the number of words to include on the chart using the `head(n)` function, where n = 15.

```
pos_neg %>% filter(sentiment=='positive')%>%head(20)  
%>%ggplot(aes(x=word,y=n))+  
  geom_bar(stat="identity",fill="green4") +  
  theme(axis.text.x=element_text(angle=90)) +  
  labs(title="Most occuring Positive Words",  
       y="count")
```



```
pos_neg %>% filter(sentiment=='negative')%>%head(20)
```

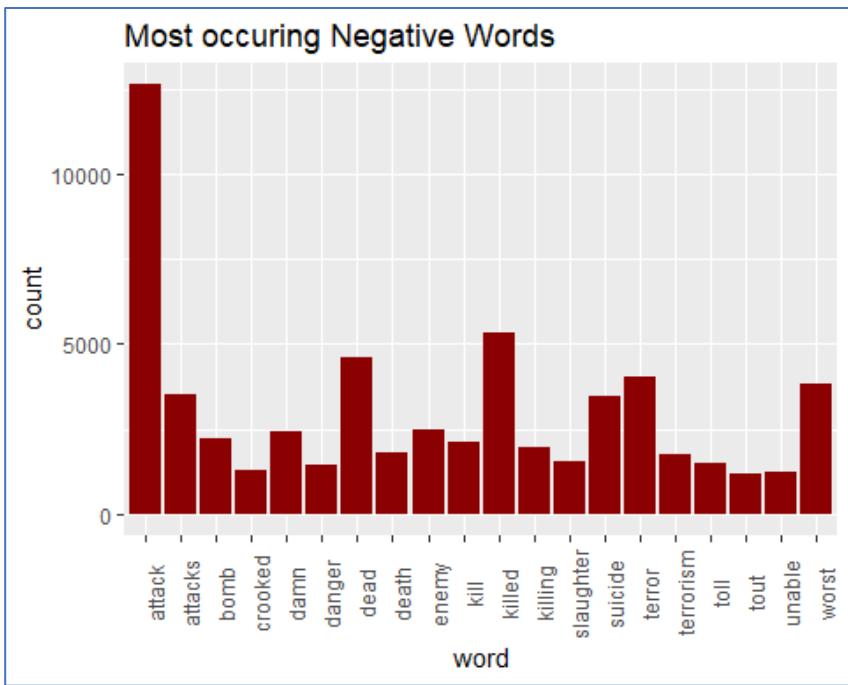
```
%>%ggplot(aes(x=word,y=n)) +
  geom_bar(stat="identity",fill="red4") +
  theme(axis.text.x=element_text(angle=90)) +
  labs(title="Most occurring Negative Words",
       y="count")
```

## New Bigram Counts

```
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)
bigram_counts
## # A tibble: 206,674 x 3
##   word1 word2     n
##   <chr> <chr> <int>
## 1 à     à     49079
## 2 ã     ã     43140
## 3 2016  07    10569
## 4 ø     ø     8012
## 5 ã     ãf    7980
## 6 ø     ù     7240
## 7 https t.co  6973
## 8 ù     ø     6658
## 9 ãf    ã     5950
## 10 å    ã     5617
## # ... with 206,664 more rows
```

## *Word clouds on positive and negative sentiment*

We constructed word clouds for positive and negative sentiment as we did for pro ISIS words. For plotting, we wanted them presented for side by side and used the `par()` function with two columns and one row.



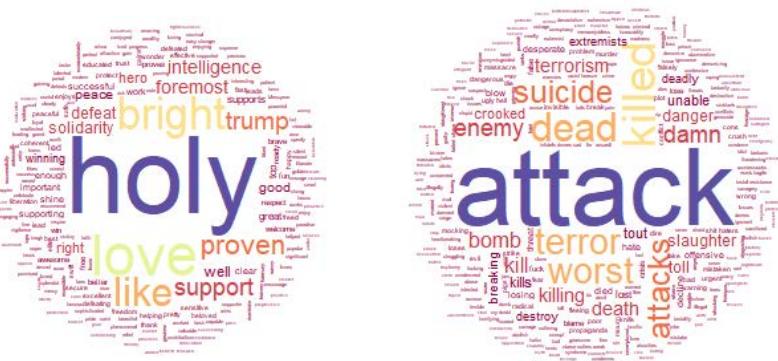
## Wordcloud on Positive and Negative words

```
par(mfrow=c(1,2))

pos<-pos_neg %>% filter(sentiment=='positive')
neg<-pos_neg %>% filter(sentiment=='negative')
wordcloud(pos$word,pos$n, min.freq =15, scale=c(5, .2),
          random.order = FALSE, random.color = FALSE,
          colors = brewer.pal(9,"Spectral"))


```

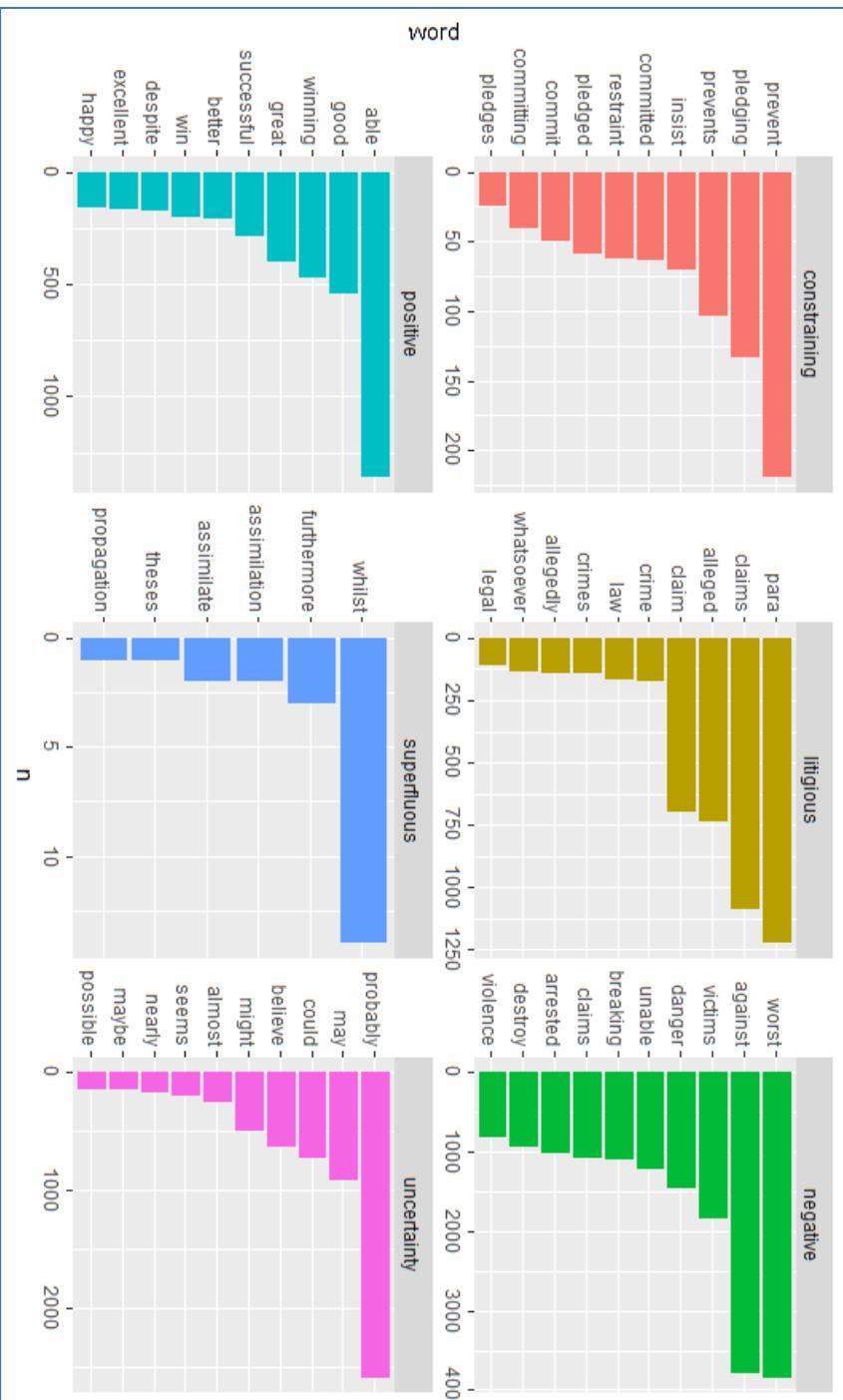
```
wordcloud(neg$word,neg$n, min.freq =30, scale=c(5, .2), ra
ndom.order = FALSE, random.color = FALSE,colors = brewer.p
al(9, “Spectral”))
```



## Top 10 words contributing to different sentiments

The code below produces a plot of ten sentiments and the top five words contributing to them. We are show six sentiments in plot that follows, including “constraining,” “litigious,” “positive,” “negative,” “superfluous,” and “uncertainty.” From a pure sentiment perspective, this does not seem too threatening.

```
tidy_tweets %>% inner_join(get_sentiments("loughran")) %>%
  count(word, sentiment) %>%
  group_by(sentiment) %>% top_n(10) %>%
  ungroup() %>% mutate(word=reorder(word,n)) %>%
  ggplot(aes(x=word, y=n, fill=sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ sentiment, scales = "free") +
  coord_flip()
## Joining, by = "word"
## Selecting by n
```



## Positive & Negative Words over Time

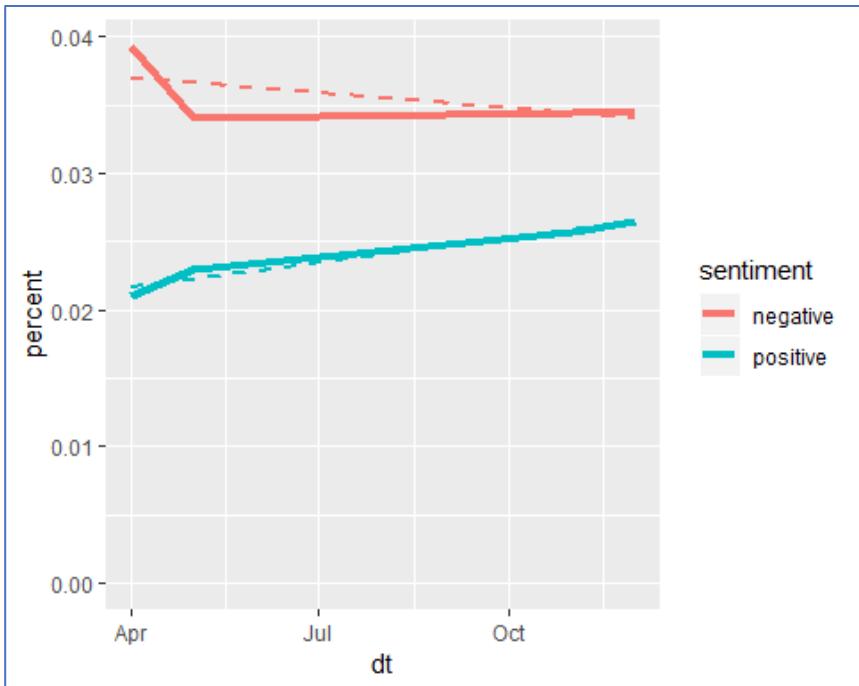
In this section we will look at the trend of positive and negative words over time on an expanded (by date) dataset, using the `bing` lexicon. The `bing` lexicon categorizes words in a binary fashion ("yes"/"no") into categories of positive and negative. The `mutate()` function adds new variables and preserves existing; `transmute()` drops existing variables. We will use `mutate()` several times throughout this section, and ggplot to present the results.

```
sentiment_by_time <- tidy_tweets %>%
  mutate(dt = floor_date(date, unit = "month")) %>%
  group_by(dt) %>%
  mutate(total_words = n()) %>%
  ungroup() %>%
  inner_join(get_sentiments("bing"))
## Joining, by = "word"
```

The `geom_smooth()` function attempts to fit a linear model (`lm`) to both positive and negative sentiment.

```
sentiment_by_time %>%
  filter(sentiment %in% c('positive','negative')) %>%
  count(dt,sentiment,total_words) %>%
  ungroup() %>%
  mutate(percent = n / total_words) %>%
  ggplot(aes(x=dt,y=percent,col=sentiment,group=sentiment))
) +
  geom_line(size = 1.5) +
  geom_smooth(method = "lm", se = FALSE, lty = 2) +
  expand_limits(y = 0)
```

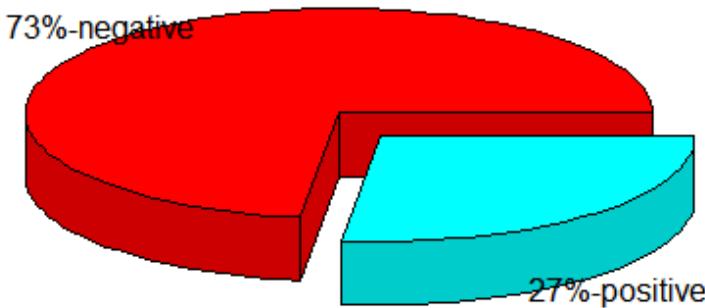
The next plot shows that the negative sentiment slightly decreases over time as a linear model, but it is not fit well and essentially remains constant after an initial drop in April. The positive sentiment is fit well by a linear model and slightly increases over time.



### *Sentiment proportion analysis*

Percentages of positive and negative words in the tweets are generated by the code below. The pie chart (below) shows the percentages of positive sentiment and negative sentiment, with negative sentiment “winning” by a long shot.

```
library(plotrix)
perc<-tweets_sentiment%>%count(sentiment)
      %>%mutate(total=sum(n))%>%group_by(sentiment)
      %>%mutate(percent=round(n/total,2)*100)%>%ungroup()
label <- c( paste(perc$percent[1], '%', '-',
      perc$sentiment[1], sep=''),
      paste(perc$percent[2], '%', '-',
      perc$sentiment[2], sep=''))
pie3D(perc$percent,labels=label,labelcex=1.1,
      explode=0.1      )
```



## N-grams

An n-gram is a sequence of n “words” taken, in order, from a body of text. If we n = 2, then we get a bigram.

```
demo_bigrams <- unnest_tokens(tweets, input = tweets, output = bigram, token = "ngrams", n=2)
demo_bigrams %>%
  count(bigram, sort = TRUE)
## # A tibble: 336,681 x 2
##   bigram          n
##   <chr>     <int>
## 1 à         49079
## 2 ã         43140
## 3 islamic state 10743
## 4 2016 07    10569
## 5 isis is    9489
## 6 Ø Ø        8012
## 7 ã af      7980
## 8 Ø ù        7240
## 9 https t.co 6973
## 10 ù Ø       6658
## # ... with 336,671 more rows
bigrams_separated <- demo_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")
bigrams_filtered <- bigrams_separated %>%
```

```
filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)
```

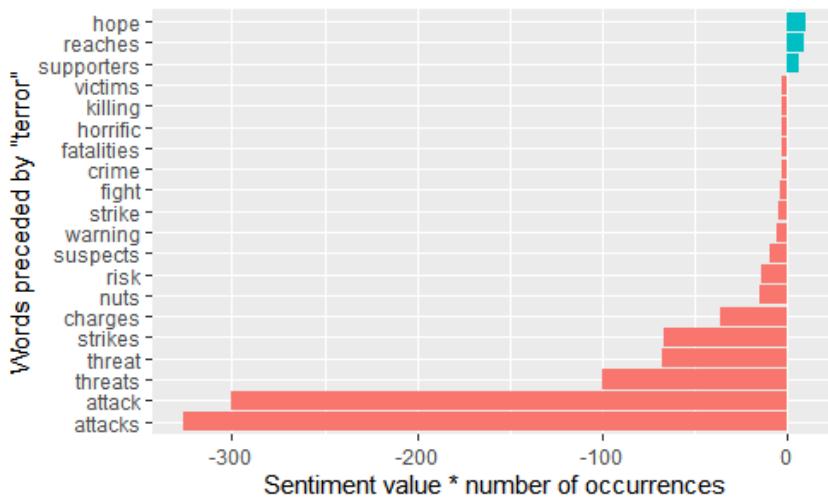
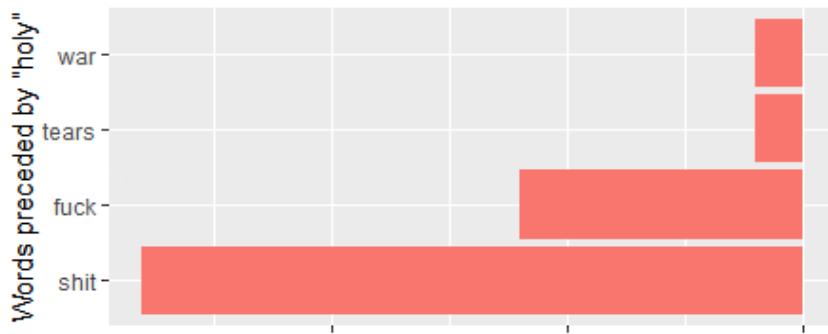
## Get AFINN Setiments

```
AFINN <- get_sentiments("afinn")
not_words <- bigrams_separated %>%
  filter(word1 == "holy") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, value, sort = TRUE) %>%
  ungroup()
```

## Get Sentiment Scores

The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. Here we are looking for words proceeded by “holy.” We also use `ggplot()` to show the words following “holy war.” We use AFINN to score the word pairs. A high negative score, like -166, would express a very strong negative sentiment, while a strong positive sentiment score might be 600. The score for “holy”-word pairs are not high in the negative direction and are 0 in the positive direction. You can see the results in graph below.

```
not_words %>%
  mutate(contribution = n * value) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * value, fill = n * value > 0)) +
  geom_col(show.legend = FALSE) +
  xlab("Words preceded by \"holy\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```



## Plot Bigrams

```
bigram_graph <- bigram_counts %>%
  filter(n > 10) %>%
  graph_from_data_frame()
## Warning in graph_from_data_frame(.): In `d` `NA` elements were replaced
## with string "NA"
bigram_graph
## IGRAPH 147a33c DN-- 8600 17052 --
## + attr: name (v/c), n (e/n)
## + edges from 147a33c (vertex names):
## [1] à      ->à          ã      ->ã
## [3] 2016   ->07        ø      ->ø
```

```
## [5] ã      ->ãf          ø      ->ù
## [7] https ->t.co        ù      ->ø
## [9] ãf     ->ã           å      ->ã
## [11] https ->twitter.com  ù      ->ù
## [13] holy   ->month       ãf     ->ãf
## [15] ã      ->å           https  ->â
## + ... omitted several edges
```

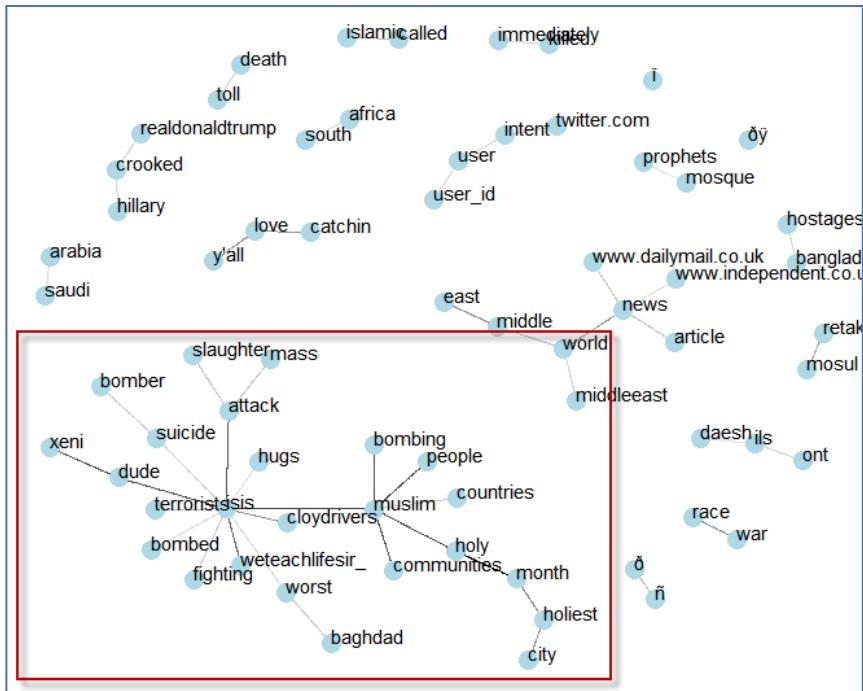
## Network of bigrams

```
set.seed(2017)
ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 0, hjust = 0)
```

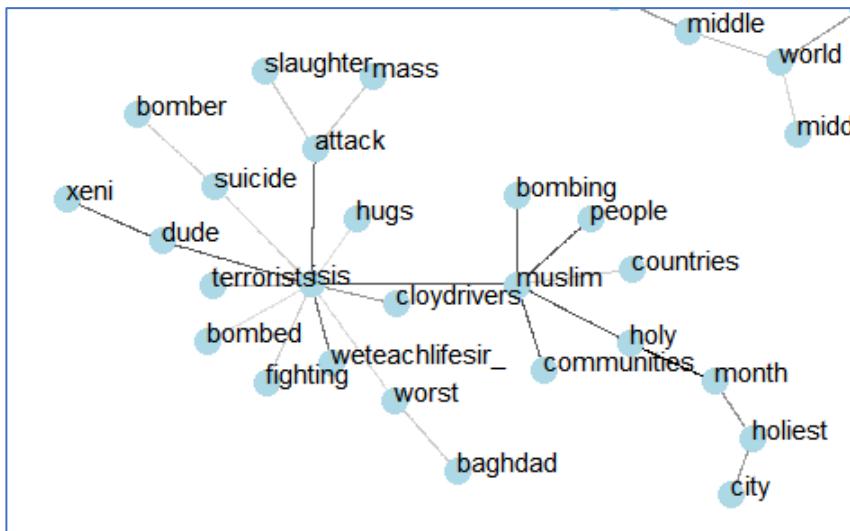
The output is a little messy, but we can make out the following bigrams:

- isis attack
- isis terrorists
- isis suicide (as as a trigram: isis suicide bomber)
- muslim countries
- muslim people
- muslim bombing

These are words that occur 1000 times or more as pairs. With tweets, it is often difficult to remove every erroneous symbol, like é, à, ä, and ÿ, but can go back and remove them as we see them appear.



Below is the zoom-in view of the red rectangle above.



## Exercises

1. Download 250000\_Plus\_Tweets from [https://github.com/stricje1/VIT\\_University/upload/master/Data\\_Analytics\\_2018/data](https://github.com/stricje1/VIT_University/upload/master/Data_Analytics_2018/data) and perform Tweet Sentiment Analysis.
  - a. Chart positive and negative Words over time.
  - b. Chart Most common positive and negative words.
  - c. Create a positive and negative Wordcloud.
  - d. Compare word usage.
2. Visit this blog by Shahin Ashkiani, Bi-grams, Tri-grams, and word network, at <https://rpubs.com/Shaahin/anxiety-bigram>.
  - a. Work through the example. The code for it is at the end of the blog post.
  - b. Render the results as a Word document using R Markdown.



# CHAPTER 7 - Latent Dirichlet Allocation (LDA)

## PART I: BLOG TOPIC ANALYSIS USING LDA

Topic modeling focuses on the problem of classifying sets of documents into themes, either manually or automatically (preferred). It is a way of identifying patterns in a corpus-technically, this is text mining. We take our corpus and group words across the corpus into ‘topics,’ using a text mining algorithm or tool.

In natural language processing, latent Dirichlet allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. A topic model which discovers underlying topics in a collection of documents and infers word probabilities in topics

We will use a collection of 20 posts from my LinkedIn blog as an example corpus, call “blog\_corpus.” The corpus can be downloaded at [https://github.com/stricje1/VIT\\_University/tree/master/Data\\_Analytics\\_2018/data](https://github.com/stricje1/VIT_University/tree/master/Data_Analytics_2018/data).

```
#Load text mining library  
library(tm)  
## Loading required package: NLP
```

### Set working Directory

see your working dierct(modify path as needed)

```
#setwd("C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/code/blog_corpus/")
```

Here, we get a listing of .txt files in directory.

```
filenames <- list.files(getwd(),pattern="*.txt")
```

### Read files into a character vector

```
files <- lapply(filenames,readLines)
```

## create corpus from vector

```
docs <- Corpus(VectorSource(files))

#inspect a particular document in corpus

writeLines(as.character(docs[1]))

## c("10 More Signs that you might be a Data Scientist", "Why are people so down on Data Scientists? Probably because they don't know what one looks like. There are a lot of folks out there calling themselves Data Scientist just because they had some introductory statistic courses, and maybe they can make really cool charts in Excel. All that makes you is someone who had some introductory statistic courses and can make some really cool charts in Excel. So what might a real Data Scientist look like? Here are some more signs that you might \"really\" be a Data Scientist (see 10 Signs that you might be a Data Scientist, October 7, 2014).",
## "1.\tYou have a title of Operation Research Analyst, Principal Business Analyst, Senior Risk Analyst, or Statistician, Data Architect, Senior Data Analyst, and so on.", "2.\tYou have a graduate degree or graduate course work in a related field, i.e., Analytics, Predictive Modeling, Risk Analysis, Applied Statistics, Database Architecture, etc.",
## "3.\tYou have a professionally taken greyscale photo on your LinkedIn profile. Who does that?", "4.\tIn addition to Data Science, you include some or all of the following skills on your profile: SPSS Modeler, SAS Programming, Regression Analysis, Statistical Modeling, Logistics Regression, Machine Learning, SQL, Python, etc.",
## "5.\tYour boss or customer listens to you and takes action when you present your analysis and recommendations.", "6.\tNot only can you crunch numbers with the best of them, you can logically present the results of your analysis in plain language, or at least your customerâ\200\231s language.", "7.\tYou write incredibly Geek-like post on LinkedIn, whether there are many viewers or not. Who does that?",
## "8.\tProfit was made or money was saved (or for non-profits some measure of performance was achieved) based on your work.",
## "9.\tYou belong to a LinkedIn group for predictive modeling, prescriptive modeling, analytics, business analytics, etc.", "10.\tOne of your hobbies is data analysis and modeling. Who would do that?")
```

```
## list(language = "en")
## list()
```

## Start Preprocessing

The next preprocessing steps include converting to lower case, removing special characters/symbols, removing punctuation, stripping digits, removing stop words, and removing white spaces.

First, we transform the corpus to lower case.

```
docs <- tm_map(docs, content_transformer(tolower))
```

Second, we remove punctuation.

```
docs <- tm_map(docs, removePunctuation)
```

Third, we strip digits from the corpus.

```
docs <- tm_map(docs, removeNumbers)
```

Fourth, we remove stopwords.

```
docs <- tm_map(docs, removeWords, stopwords("english"))
```

Finally, we remove whitespaces.

```
docs <- tm_map(docs, stripWhitespace)
```

It is good practice to check the document preprocessing every now and then.

```
writeLines(as.character(docs[2]))
## cten signs might data scientist nailed really good definition data scientist argue need entity poorly defined perform one following task might data scientist tyou build database models predictive descriptive prescriptive tyou perform data mining analysis tyou use data perform analytics business otherwise tyou can spell data scientist using binary numbers tyou regularly use sas enterprise guide sas enterprise miner spss modeler spss statistics tyou can program r use perform data analysis modeling tyou really hardcore use matlab octave data analysis tyou can spell data scientist latin ready little sas book tyou entire sas library ipad iphone etc tyou use data provide added value company customer ten signs might data scientist tyou title operation research analyst principal business analyst senior risk
```

```
analyst statistician tyou graduate degree graduate course  
work  
## list(language = "en")  
## list()
```

## Stem document

(fix up 1) differences between us and aussie english 2) general errors

```
docs <- tm_map(docs,stemDocument)  
docs <- tm_map(docs, content_transformer(gsub),  
                pattern = "organiz", replacement = "organ")  
docs <- tm_map(docs, content_transformer(gsub),  
                pattern = "organis", replacement = "organ")  
docs <- tm_map(docs, content_transformer(gsub),  
                pattern = "andgovern", replacement = "gover  
n")  
docs <- tm_map(docs, content_transformer(gsub),  
                pattern = "inenterpris", replacement = "ent  
erpris")  
docs <- tm_map(docs, content_transformer(gsub),  
                pattern = "team-", replacement = "team")  
docs <- tm_map(docs, content_transformer(gsub),  
                pattern = "henry", replacement = "henry ")  
#define and eliminate all custom stopwords  
myStopwords <- c("can", "say", "one", "way", "use",  
                 "also", "howev", "tell", "will",  
                 "much", "need", "take", "tend", "even",  
                 "like", "particular", "rather", "said",  
                 "get", "well", "make", "ask", "come", "end",  
                 "first", "two", "help", "often", "may",  
                 "might", "see", "someth", "thing", "point",  
                 "post", "look", "right", "now", "think",  
                 "anoth", "put", "set", "new", "good",  
                 "want", "sure", "kind", "yes", "day", "etc",  
                 "quit", "sinc", "attempt", "lack", "seen",  
                 "littl", "ever", "moreov", "though", "abl",  
                 "enough", "earli", "away", "achiev", "draw",  
                 "last", "never", "brief", "entir", "brief",  
                 "great", "lot")
```

```
docs <- tm_map(docs, removeWords, myStopwords)
```

At this point, it is a good idea to inspect a document as a check.

```
writeLines(as.character(docs[3]))  
## c type analyt user sell thursday afternoon analyt user  
meet becom standard mayb youâr familiar similar meet key m  
etric everyon follow present present deck everi number sli  
ce dice bar chart pie chart dozen page differ version pro  
cess whatâ reallí happen peopl question most peopl just  
took meet peopl thank present taken time number togeth  
hung accomplish declar meet wast time theyâr wast time an  
alyt mean differ differ peopl â” differ analyt user donâ  
t understand peopl consum analyt wonât absorb lesson crit  
ic analyt solut provid youâr sell differ user mean solut  
adapt differ user messag adapt analyt user user analyt br  
eak dimens purpos analyt versus skill user axe form b  
ox diagram across bottom userâ purpos analyt either explor  
atori decisionmak along side userâ technic skill level low  
er technic skill higher technic skill lower technic skill  
user someon understand metric graph chart produc advanc a  
nalyt output higher technic skill user understand genera  
l advanc statist output twodimens breakdown creat four maj  
or type analyt user analyst scientist expert execut type a  
nalyt user break categori type analyt user differ type so  
ftwar differ type support requir differ analyst letâ st  
art explor analyst user analyt user statist savvi someon u  
nderstand databas queri simpli possess sophist advanc tec  
hnic skill statist model construct role primarili queri d  
ata number differ understand relationship  
## list(language = "en")  
## list()
```

## Create document-term matrix

This code chunk generates the document-term matrix for our corpus, and the next several code chucks set up the document-term matrix for our analysis.

```
dtm <- DocumentTermMatrix(docs)
```

This code chunk converts rownames to filenames

```
rownames(dtm) <- filenames
```

This code chunk collapses matrix by summing over columns

```
freq <- colSums(as.matrix(dtm))
```

This code chunk determines the length, which should be total number of terms.

```
length(freq)  
## [1] 2833
```

This code chunk creates the sort order (descending by frequency) for our terms and writes it to our disk.

```
ord <- order(freq,decreasing=TRUE)  
write.csv(freq[ord],"word_freq.csv")
```

## Load Topic Models Library

Here we load the R-package “topicmodels” that we will be using in our analysis.

```
library(topicmodels)
```

## Set Parameters for Gibbs Sampling

In this code chunk we set the parameters for the LDA using Gibbs sampling that we will implement below. This is just a matter of convenience.

```
burnin <- 4000  
iter <- 2000  
thin <- 500  
seed <- list(2003,5,63,100001,765)  
nstart <- 5  
best <- TRUE
```

## Number of Topics

Now we set a rough guess for the number of topics as five. Later, we will use an user function to make a better guess at the optimal number of topics, k.

```
k <- 5
```

## Run LDA using Gibbs sampling

Using the parameter, we set above, we now implment the LDA model using Gibbs sampling.

```
library(topicmodels)
control=list(nstart=5, seed = list(2003,5,63,100001,765),
best=TRUE, burnin = 4000, iter = 2000, thin=500)
ldaOut <- LDA(dtm, 5, method="Gibbs", control=control)
```

## Write out Results

Now we write the documents to topics:

```
ldaOut.topics <- as.matrix(topics(ldaOut))
write.csv(ldaOut.topics,file=paste("LDAGibbs",k,"DocsToTopics.csv"))
```

## Top 6 Terms per Topic

The next code chunk provides the top six terms for each topic.

```
ldaOut.terms <- as.matrix(terms(ldaOut,6))
write.csv(ldaOut.terms,file=paste("LDAGibbs",k,"TopicsToTerms.csv"))
```

## Topic Probabilities

Here, we calculate the probabilities associated with each topic assignment:

```
topicProbabilities <- as.data.frame(ldaOut@gamma)
write.csv(topicProbabilities,file=paste("LDAGibbs",k,"TopicProbabilities.csv"))
```

Then, we find relative importance of top 2 topics:

```
topic1ToTopic2 <- lapply(1:nrow(dtm),function(x)
sort(topicProbabilities[x,])[k]/sort(topicProbabilities[x,
])[k-1])
```

Next, we find relative importance of second and third most important topics:

```
topic2ToTopic3 <- lapply(1:nrow(dtm),function(x)
sort(topicProbabilities[x,])[k-1]/sort(topicProbabilities[x,
])[k-2])
```

Finally, we write the output to a file:

```
write.csv(topic1ToTopic2,file=paste("LDAGibbs",k,"Topic1ToTopic2.csv"))
```

```
write.csv(topic2ToTopic3,file=paste("LDAGibbs",k,"Topic2ToTopic3.csv"))
```

## PART II: 2012 USA PRESIDENTIAL DEBATE USING LDA

[*source files available on GitHub*]

### PRELIMINARIES

Here, we install and load libraries needed for data processing and plotting:

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load_gh("trinker/gofastr")
pacman::p_load(tm, topicmodels, dplyr, tidyverse, igraph, devtools, LDavis, ggplot2, sentimentr)
library(scales)
library(Rcpp)
library(tm)
library(topicmodels)
library(dplyr)
library(tidyverse)
library(igraph)
library(devtools)
library(LDavis)
library(ggplot2)
library(sentimentr)
```

### Get External Scripts

Next, we call external script containing my own optimal\_k functions definitions, using the source function call.

```
source("C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/code/optimal_k.R")
```

The content of this external file is included in the Appendix at the end of this document.

### Load the Data

Now, we load the data.

```
data(presidential_debates_2012)
head(presidential_debates_2012,5)
```

```
## # A tibble: 5 x 5
##   person tot    time   role dialogue
##   <fct>  <chr> <fct>  <fct>  <chr>
## 1 LEHRER 1.1    time 1 modera~ We'll talk about specific
##   ally about health ~
## 2 LEHRER 1.2    time 1 modera~ But what do you support t
##   he voucher system,~
## 3 ROMNEY 2.1    time 1 candid~ What I support is no chan
##   ge for current ret~
## 4 ROMNEY 2.2    time 1 candid~ And the president supports
##   taking dollar se~
## 5 LEHRER 3.1    time 1 modera~ And what about the vouche
##   rs?
```

In this code chunk we generate user-defined stopwords that are relevant to our case.

```
stops <- c(
  tm::stopwords("english"),
  tm::stopwords("SMART"), "governor",
  "president", "mister", "obama", "romney"
) %>
gofastr::prep_stopwords()
```

The next code chunk creates the document-term matrix.

```
doc_term_mat <- presidential_debates_2012 %>%
  with(gofastr::q_dtm_stem(dialogue, paste(person, time,
sep = "_")))) %>%
  gofastr::remove_stopwords(stops, stem=TRUE) %>%
  gofastr::filter_tf_idf()
#gofastr::filter_documentheads()
```

This code chunk established the control list we will use to find optimum\_k.

```
control <- list(burnin = 500, iter = 1000, keep = 100)
```

## Determine Optimal Number of Topics

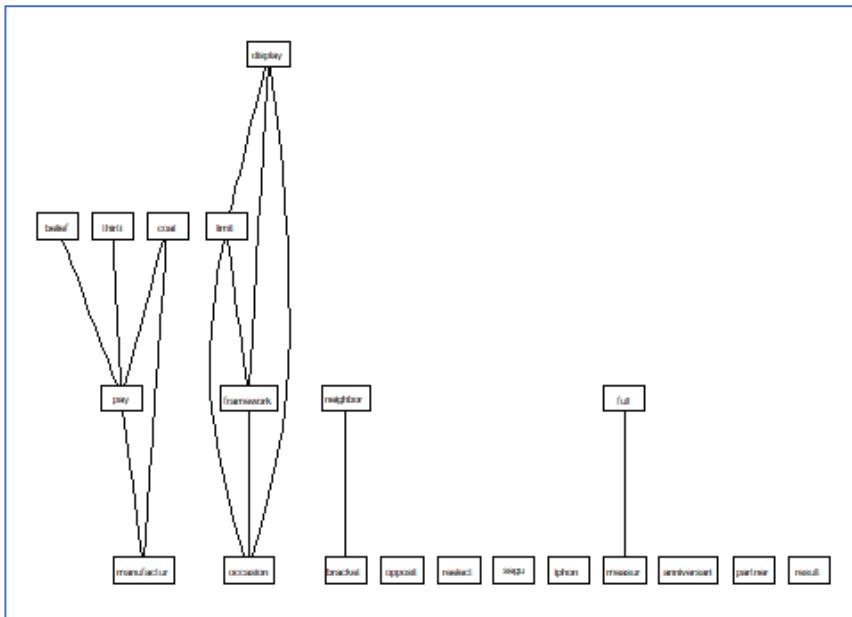
Using our external optimal\_k function code, we determine the optimal number of topics, which is a little better than a random guess.

```
k <- optimal_k(doc_term_mat, 40, control = control)
## Grab a cup of coffee this could take a while...
```

```

## 10 of 40 iterations (Current: 05:32:19; Elapsed: .1 min
s)
## 20 of 40 iterations (Current: 05:32:30; Elapsed: .3 min
s; Remaining: ~.7 mins)
## 30 of 40 iterations (Current: 05:32:47; Elapsed: .5 min
s; Remaining: ~.4 mins)
## 40 of 40 iterations (Current: 05:33:12; Elapsed: 1 mins
; Remaining: ~0 mins)
## Optimal number of topics = 14
print.optimal_k(doc_term_mat, 40, control = control)

```



```

## [1] "A graph with 20 nodes."

```

This gives us the number of topics—the Harmonic Mean of Log Likelihood

## Run the Model

Now, we implement the LDA model using Gibbs sampling and the parameters we have already established.

```

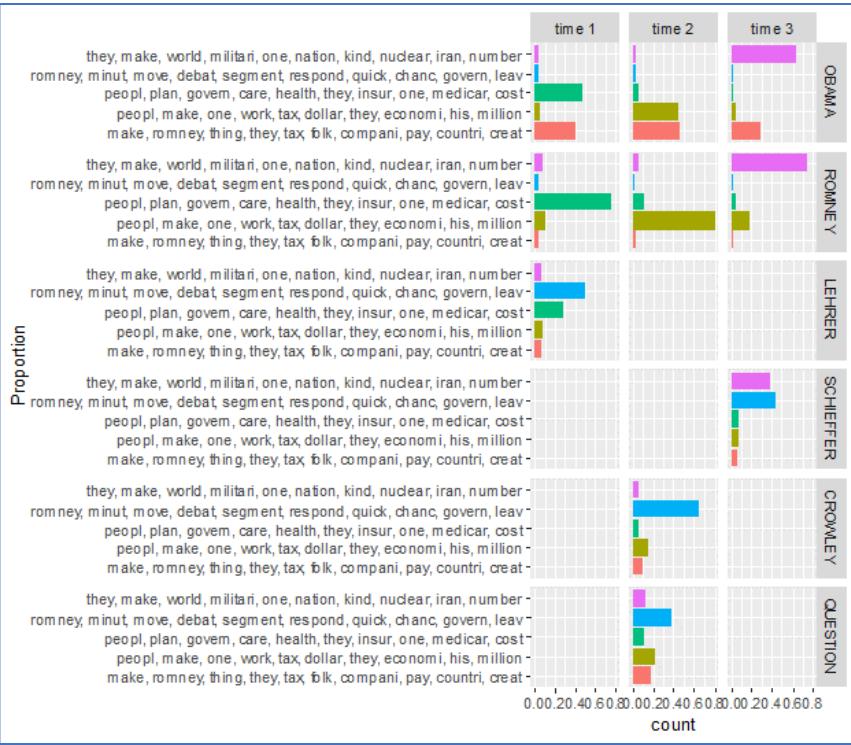
control[["seed"]] <- 100
lda_model <- topicmodels::LDA(doc_term_mat, k=as.numeric(k),
), method = "Gibbs",
control = control)

```

## Plot the Topics Per Person & Time

This code chunk is used to generate a plot of the topics per person and time.

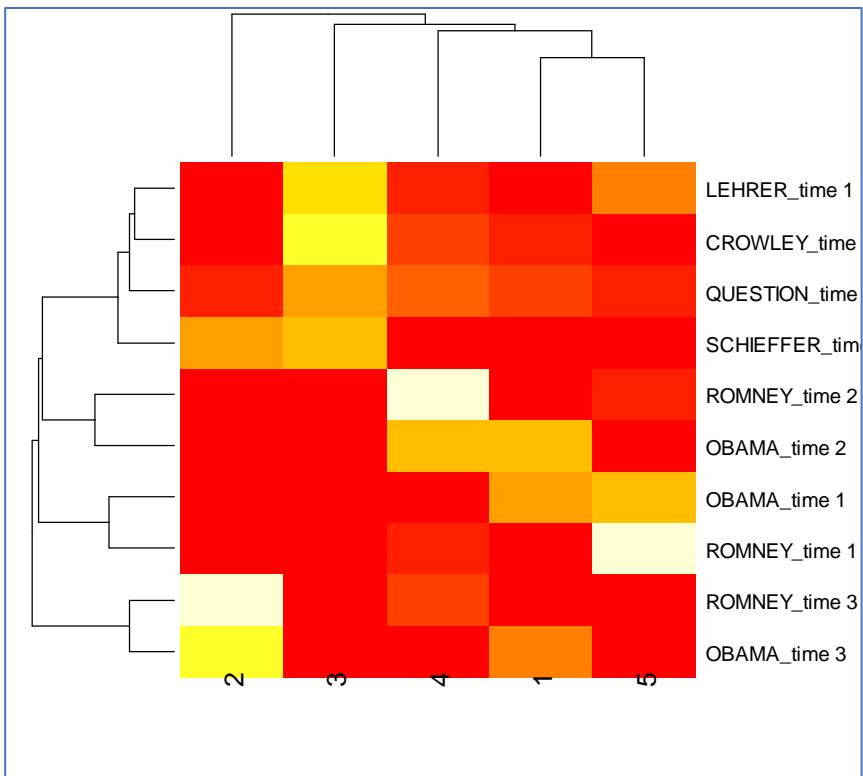
```
topics <- topicmodels::posterior(lda_model, doc_term_mat)[[“topics”]]  
topic_dat <- dplyr::add_rownames(as.data.frame(topics), “Person_Time”)  
## Warning: Deprecated, use tibble::rownames_to_column() instead.  
colnames(topic_dat)[-1] <- apply(terms(lda_model, 10), 2,  
paste, collapse = “, ”)  
  
tidyr::gather(topic_dat, Topic, Proportion, -c(Person_Time)) %>%  
  tidyr::separate(Person_Time, c(“Person”, “Time”), sep  
= “_”) %>%  
  dplyr::mutate(Person = factor(Person,  
    levels = c(“OBAMA”, “ROMNEY”, “LEHRER”, “SCHIEFFER”,  
    “CROWLEY”, “QUESTION” ))  
  ) %>%  
  ggplot2::ggplot(ggplot2::aes(weight=Proportion, x=Topic,  
    fill=Topic)) +  
    ggplot2::geom_bar() +  
    ggplot2::coord_flip() +  
    ggplot2::facet_grid(Person~Time) +  
    ggplot2::guides(fill=FALSE) +  
    ggplot2::xlab(“Proportion”)
```



# Plot the Topics Matrix as a Heatmap

Now, we make a heatmap of our results.

```
heatmap(topics, scale = "none")
```



## Heatmap of Topics

Topics by Candidate

## Network of the Word Distributions Over Topics

Next, we generate a network of word distributions over topics.

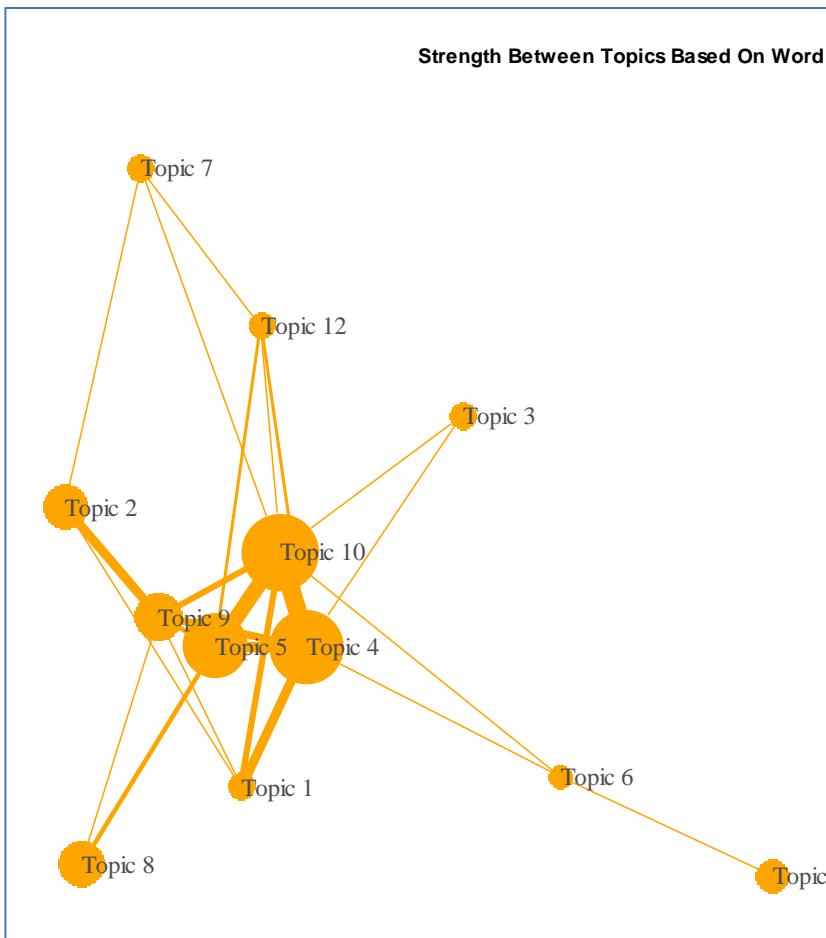
```
post <- topicmodels::posterior(lda_model)
cor_mat <- cor(t(post[["terms"]]))
cor_mat[ cor_mat < .05 ] <- 0
diag(cor_mat) <- 0
```

Now, we get the strength between Topics based on word probabilities:

```
graph <- graph.adjacency(cor_mat, weighted=TRUE, mode="lower")
graph <- delete.edges(graph, E(graph)[ weight < 0.05 ])

E(graph)$edge.width <- E(graph)$weight*20
V(graph)$label <- paste("Topic", V(graph))
V(graph)$size <- colSums(post[["topics"]]) * 15
```

```
par(mar=c(0, 0, 3, 0))
set.seed(110)
plot.igraph(graph, edge.width = E(graph)$edge.width,
            edge.color = "orange", vertex.color = "orange",
            vertex.frame.color = NA, vertex.label.color = "grey30"
)
title("Strength Between Topics Based On Word Probabilities",
      cex.main=.8)
```



## Word Distributions over Topics

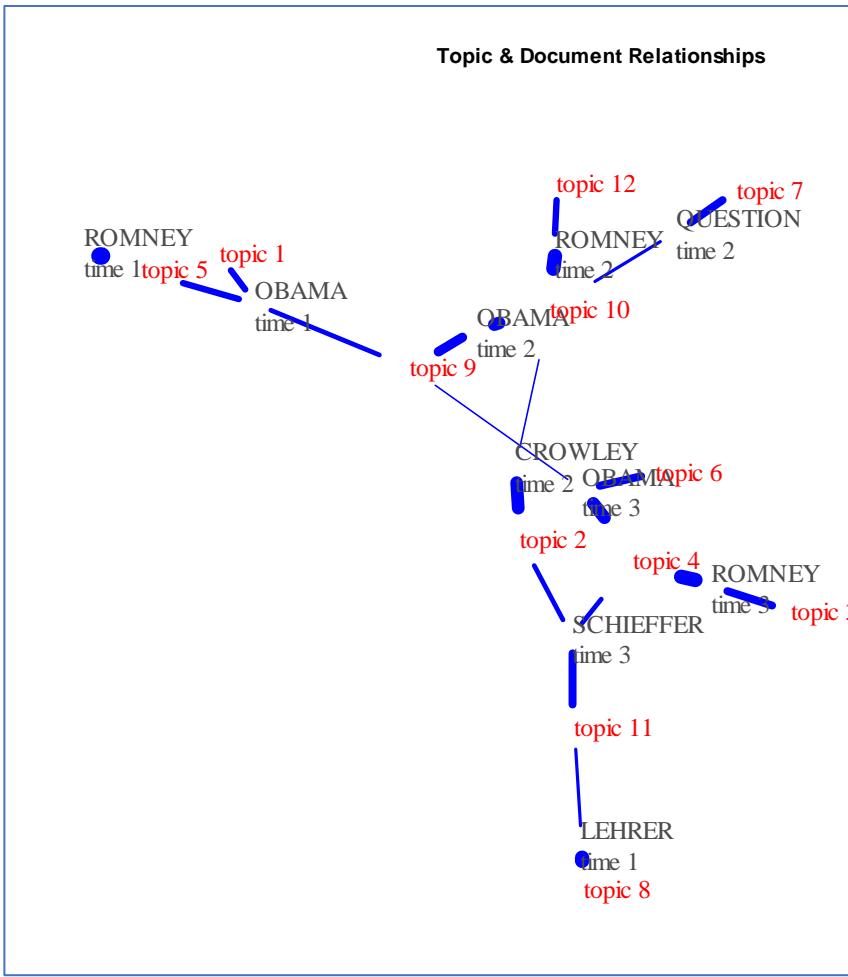
This code chunk generates a network of the topics over documents.

```
minval <- .1
topic_mat <- topicmodels::posterior(lda_model)[["topics"]]
]

graph <- graph_from_incidence_matrix(topic_mat, weighted =TRUE)
graph <- delete.edges(graph, E(graph)[ weight < minval])

E(graph)$edge.width <- E(graph)$weight*17
E(graph)$color <- "blue"
V(graph)$color <- ifelse(grep1("^\d+$", V(graph)$name),
"grey75", "orange")
V(graph)$frame.color <- NA
V(graph)$label <- ifelse(grep1("^\d+$", V(graph)$name),
paste("topic", V(graph)$name), gsub("_", "\n", V(graph)$name))
V(graph)$size <- c(rep(10, nrow(topic_mat)), colSums(topic_mat) * 20)
V(graph)$label.color <- ifelse(grep1("^\d+$", V(graph)$name),
"red", "grey30")

par(mar=c(0, 0, 3, 0))
set.seed(365)
plot.igraph(graph, edge.width = E(graph)$edge.width,
vertex.color = adjustcolor(V(graph)$color, alpha.f = .4))
title("Topic & Document Relationships", cex.main=.8)
```



The topic and document relationships provide us with Topics by Candidate.

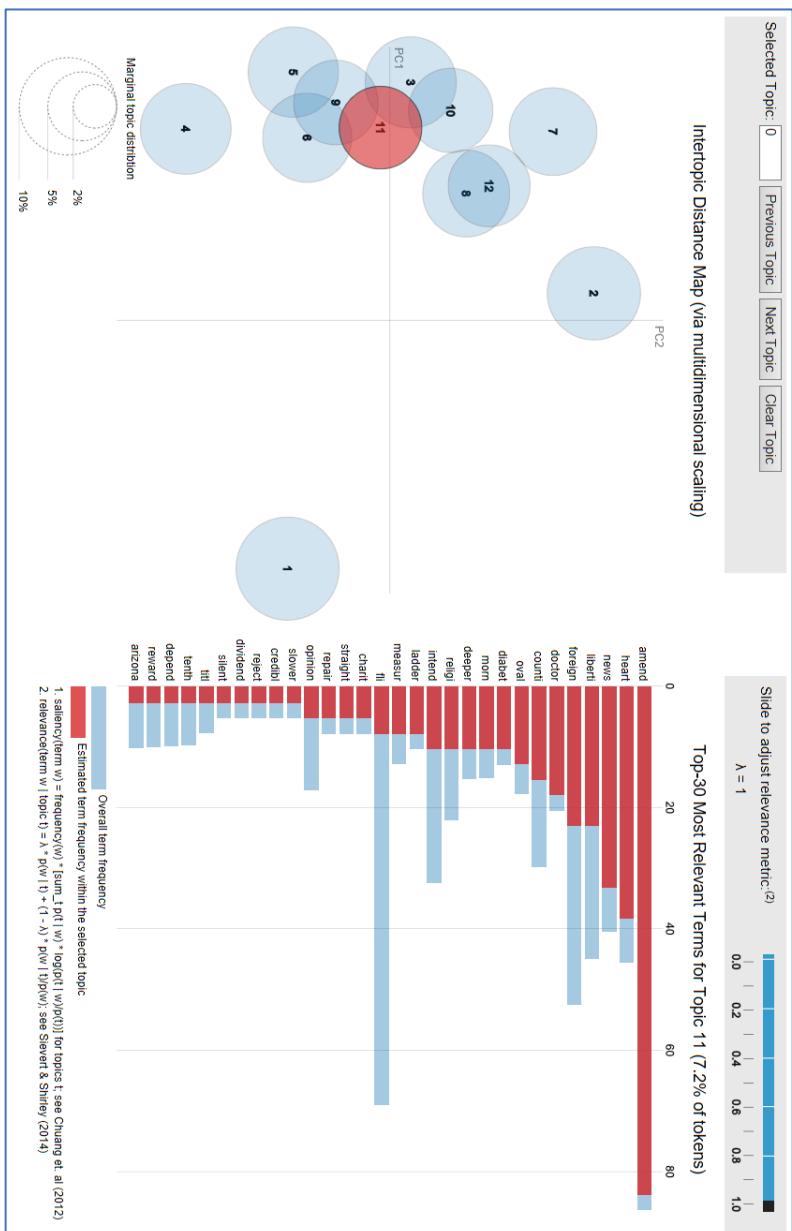
## LDAvis of Model

The LDAvis provides an interactive view of the results by topic number. Note that topic number 11 has been selected on the html dashboard.

```

source("C:/Users/jeff/Documents/VIT_Course_Material/Data_Analytics_2018/code/topicmodels2LDAvis.R")
lda_model %>%
  topicmodels2LDAvis() %>%
  LDAvis::serVis()
  
```

```
## Loading required namespace: servr
```

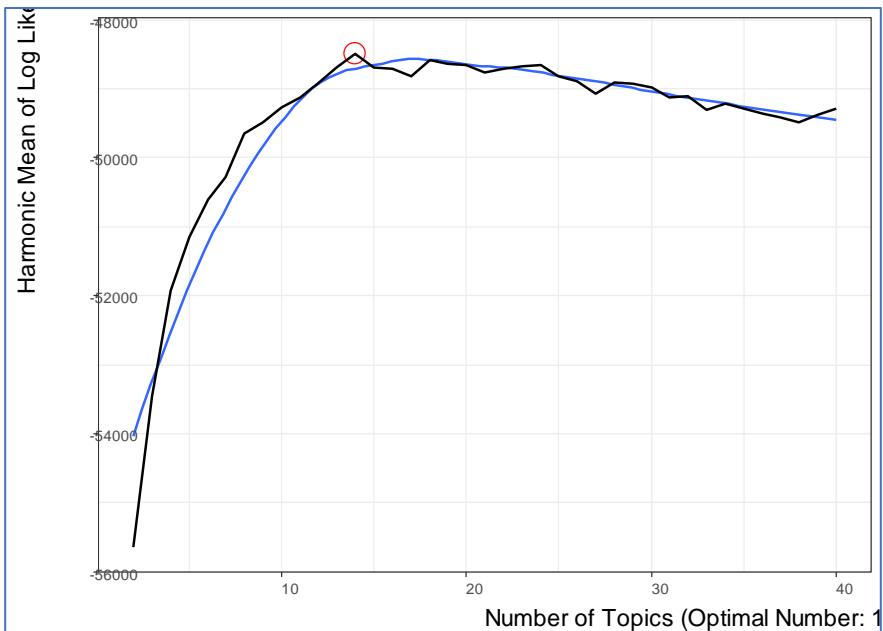


## Fitting New Data

```
library(gofastr)
## Create the DocumentTermMatrix for New Data
doc_term_mat2 <- partial_republican_debates_2015 %>%
  with(gofastr::q_dtm_stem(dialogue, paste(person, location, sep = " _")))) %>%
  gofastr::remove_stopwords(stops, stem=TRUE) %>%
  gofastr::filter_tf_idf() %>%
  gofastr::filter_documents()
```

## Run the Model for New Data

```
control = list(nstart=5, seed = list(2003,5,63,100001,765),
, best=TRUE, burnin = 4000, iter = 2000, thin=500)
control2 = list(burnin = 500, iter = 1000, keep = 100)
#control2[["estimate.beta"]] <- FALSE
k <- optimal_k1(doc_term_mat2,40,control=control2)
##
## Grab a cup of coffee this could take a while...
## 10 of 40 iterations (Current: 05:33:21; Elapsed: .1 mins)
## 20 of 40 iterations (Current: 05:33:32; Elapsed: .3 mins;
## Remaining: ~.7 mins)
## 30 of 40 iterations (Current: 05:33:50; Elapsed: .6 mins;
## Remaining: ~.4 mins)
## 40 of 40 iterations (Current: 05:34:15; Elapsed: 1 mins;
## Remaining: ~0 mins)
## Optimal number of topics = 16
plot.optimal_k1(optimal_k1(doc_term_mat2,40,control=control2))
##
## Grab a cup of coffee this could take a while...
## 10 of 40 iterations (Current: 05:34:23; Elapsed: .1 mins)
## 20 of 40 iterations (Current: 05:34:34; Elapsed: .3 mins;
## Remaining: ~.7 mins)
## 30 of 40 iterations (Current: 05:34:52; Elapsed: .6 mins;
## Remaining: ~.4 mins)
## 40 of 40 iterations (Current: 05:35:18; Elapsed: 1 mins;
## Remaining: ~0 mins)
## Optimal number of topics = 23
```



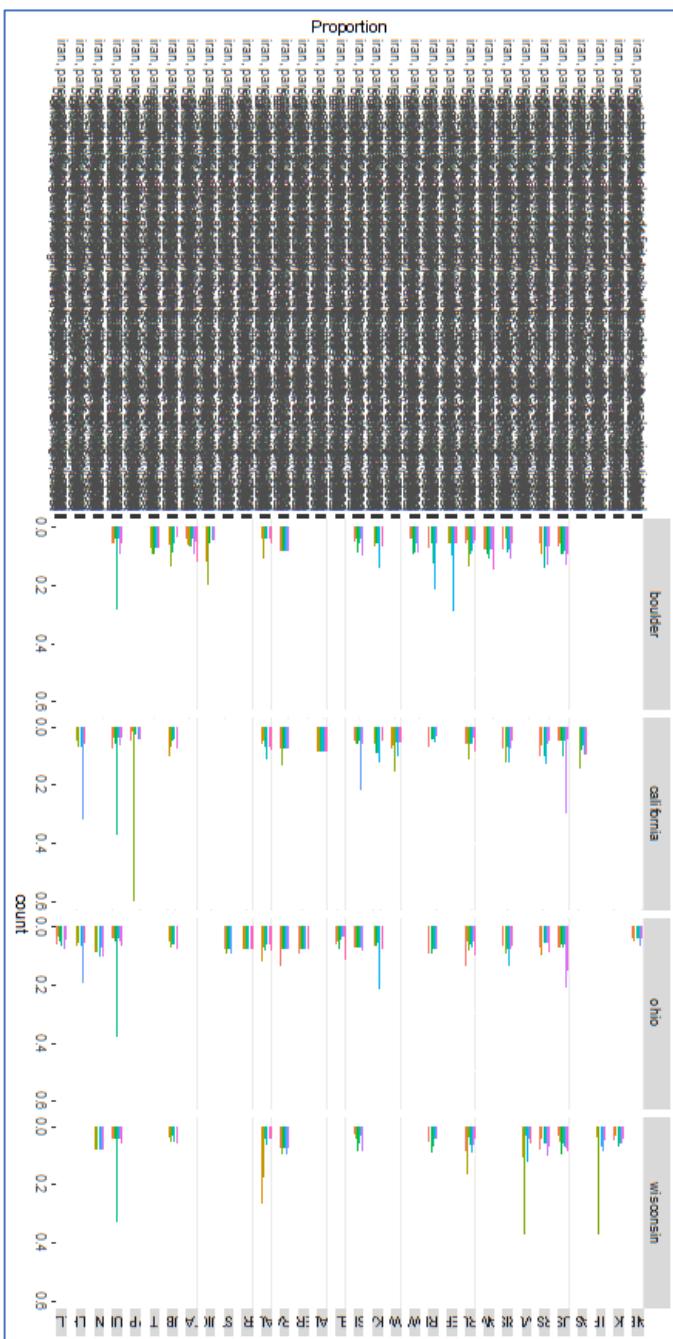
```
lda_model2 <- topicmodels::LDA(doc_term_mat2, k = as.numeric
ic(k), model = lda_model,
control = control2)
```

## Plot the Topics Per Person & Location for New Data

```
topics2 <- topicmodels::posterior(lda_model2, doc_term_mat2)[[“topics”]]
topic_dat2 <- dplyr::add_rownames(as.data.frame(topics2),
“Person_Location”)
## Warning: Deprecated, use tibble::rownames_to_column() instead.
colnames(topic_dat2)[-1] <- apply(terms(lda_model2, 10), 2
, paste, collapse = “, ”)

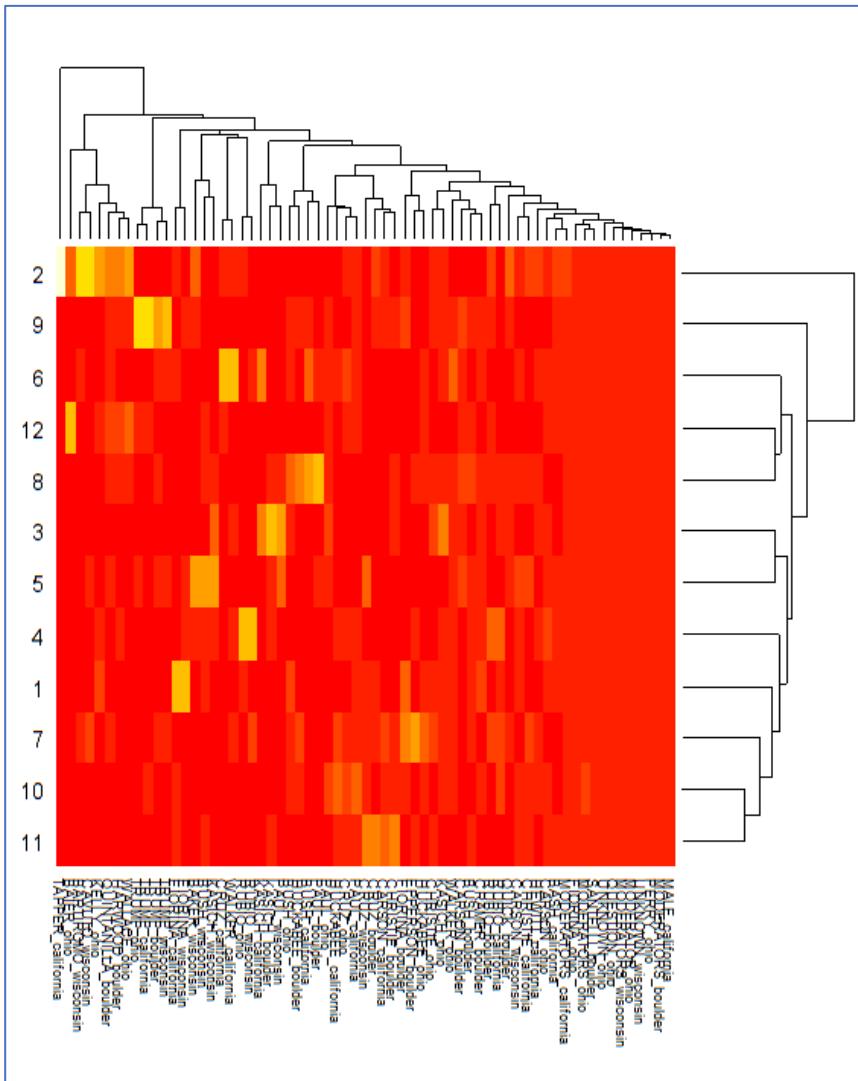
tidy::gather(topic_dat2, Topic, Proportion, -c(Person_Location)) %>%
  tidy::separate(Person_Location, c(“Person”, “Location”),
sep = “_”) %>%
  ggplot2::ggplot(ggplot2::aes(weight=Proportion, x=Topic,
fill=Topic)) +
  ggplot2::geom_bar() +
  ggplot2::coord_flip() +
  ggplot2::facet_grid(Person~Location) +
```

```
ggplot2::guides(fill=FALSE) +  
ggplot2::xlab("Proportion")
```



The figure below is the heatmap for the preceding hierarchical model. It makes the interpretation more intuitive and is easier to read.

```
heatmap(topics2, scale = "none")
```



## Appendix – optimal\_k

```
' Find Optimal Number of Topics
#
#' Iteratively produces models and then compares the harmonic mean of the log
#' Likelihoods in a graphical output.
#
#' @param x A \code{\link[tm]\{DocumentTermMatrix\}}.
#' @param max.k Maximum number of topics to fit (start small [i.e., default of
#' 30] and add as necessary).
#' @param harmonic.mean Logical. If \code{TRUE} the harmonic means of the
#' Log Likelihoods are used to determine k (see
#' \url{http://stackoverflow.com/a/21394092/1000343}). Otherwise just the log
#' Likelihoods are graphed against k (see
#' \url{http://stats.stackexchange.com/a/25128/7482}).
#' @param burnin Object of class \code{"integer"}; number of omitted Gibbs
#' iterations at beginning, by default equals 0.
#' @param iter Object of class \code{"integer"}; number of Gibbs iterations, by
#' default equals 2000.
#' @param keep Object of class \code{"integer"}; if a positive integer, the
#' Log Likelihood is saved every keep iterations.
#' @param method The method to be used for fitting; currently
#' \code{method = "VEM"} or \code{method= "Gibbs"} are supported
#'.
#' @param drop.seed Logical. If \code{TRUE} \code{seed} argument is dropped from
#' \code{control}.
#' @param ... Other arguments passed to \code{??LDAcontrol}.
#' @return Returns the \code{\link[base]\{data.frame\}} of k (number of topics) and
#' the associated Log Likelihood.
#' @references \url{http://stackoverflow.com/a/21394092/1000343}
\cr
#' \url{http://stats.stackexchange.com/a/25128/7482} \cr
#' Ponweiser, M. (2012). Latent Dirichlet Allocation in R (Diploma Thesis).
#' Vienna University of Economics and Business, Vienna.
#' http://epub.wu.ac.at/3558/1/main.pdf \cr\cr
#' Griffiths, T.L., and Steyvers, M. (2004). Finding scientific topics.
#' Proceedings of the National Academy of Sciences of the United States of America
```

```

#' 101(Suppl 1), 5228 - 5235. \url{http://www.pnas.org/content/101/suppl_1/5228.full.pdf}
#' @keywords k topicmodel
#' @export
#' @author Ben Marwick and Tyler Rinker <tyler.rinker@gmail.com>.
#' @examples
#' ## Install/Load Tools & Data
#' if (!require("pacman")) install.packages("pacman")
#' pacman::p_load_github("trinker/gofastr")
#' pacman::p_load(tm, topicmodels, dplyr, tidyverse, devtools, LDAvis, ggplot2)
#'
#'
#' ## Source topicmodels2LDAvis function
#' devtools::source_url("https://gist.githubusercontent.com/trinker/477d7ae65ff6ca73cace/raw/79dbc9d64b17c3c8befde2436fdeb8ec2124b07b/topicmodels2LDAvis")
#'
#' data(presidential_debates_2012)
#'
#' ## Generate Stopwords
#' stops <- c(
#'   tm::stopwords("english"),
#'   "governor", "president", "mister", "obama", "romney"
#' ) %>%
#'   gofastr::prep_stopwords()
#'
#' ## Create the DocumentTermMatrix
#' doc_term_mat <- presidential_debates_2012 %>%
#'   with(gofastr::q_dtm_stem(dialogue, paste(person, time, sep = "_"))) %>%
#'   gofastr::remove_stopwords(stops) %>%
#'   gofastr::filter_tf_idf() %>%
#'   gofastr::filter_documents()
#'
#' opti_k1 <- optimal_k(doc_term_mat)
#' opti_k1
#'
#' opti_k2 <- optimal_k(doc_term_mat, harmonic.mean = FALSE)
#' opti_k2

optimal_k <- function(x, max.k = 30, harmonic.mean = TRUE,
                      control = if (harmonic.mean) list(burnin = 500, iter = 1000, keep = 100) else NULL,
                      method = if (harmonic.mean) "Gibbs" else "VEM",
                      verbose = TRUE, drop.seed = TRUE, ...){
  if (isTRUE(drop.seed)){
    control[["seed"]] <- NULL
  }
  if (isTRUE(harmonic.mean)) {

```

```

    optimal_k1(x, max.k = max.k, control = control, method = met
    hod, verbose = verbose, ...)
} else {
    optimal_k2(x, max.k = max.k, control = control, method = met
    hod, ...)
}
}

optimal_k1 <- function(x, max.k = 30,
                      control = list(burnin = 500, iter = 1000,
keep = 100), method = "Gibbs",
                      verbose = TRUE, ...){
    if (max.k > 20) {
        message("\nGrab a cup of coffee this could take a while...\n")
        flush.console()
    }
    tic <- Sys.time()
    v <- rep(NA, floor(max.k/10))
    dat <- data.frame(k = v, time = v)
    end <- data.frame(k = max.k^2)
    hm_many <- sapply(2:max.k, function(k){
        if (k %% 10 == 0){
            time <- as.numeric(difftime(Sys.time(), tic, units = "mins
"))
            dat[k/10, 1:2] <- c(k^2, time)
            if (k/10 > 1) {
                fit <- with(dat, lm(time~k))
                pred <- predict(fit, end) - time
                if (pred < 0) pred <- 0
                est <- paste0("; Remaining: ~", time2char(pred), " mins"
)
            } else {
                est <- ""
            }
            cur <- format(Sys.time(), format="%I:%M:%S")
            elapsed <- time2char(time)
            #gsub("^0+", "", as.character(round(as.numeric(difftime(Sy
s.time(), tic, units = "mins")), 1)))
            cat(sprintf("%s of %s iterations (Current: %s; Elapsed: %s
mins%s)\n", k, max.k, cur, elapsed, est)); flush.console()
        }
        burnin <- control[["burnin"]]
        keep <- control[["keep"]]
        if (is.null(burnin) | is.null(keep)) stop("Supply burnin &
keep to control")
        fitted <- topicmodels::LDA(x, k = k, method = method, contro
l = control)
        logLik <- fitted@logLik[-c(1:(burnin/keep))]
        harmonicMean(logLik)
    })
}

```

```

out <- c(2:max.k)[which.max(hm_many)]
if (which.max(hm_many) == max.k) warning("Optimal K is last value; suggest increasing `max.k`")
class(out) <- c("optimal_k", "optimal_k1", class(out))
attributes(out)[["k_dataframe"]] <- data.frame(
  k = 2:max.k,
  harmonic_mean = hm_many
)
if (isTRUE(verbose)) cat(sprintf("Optimal number of topics = %s\n", as.numeric(out)))
out
}

optimal_k2 <- function(x, max.k = 30, control = NULL, method = "VEM", ...){
  if (max.k > 20) {
    message("\nGrab a cup of coffee this could take a while...\n")
  }
  flush.console()
}
tic <- Sys.time()
v <- rep(NA, floor(max.k/10))
dat <- data.frame(k = v, time = v)
end <- data.frame(k = max.k^2)
best_model <- lapply(seq(2, max.k, by=1), function(k){
  if (k %% 10 == 0){
    time <- as.numeric(difftime(Sys.time(), tic, units = "mins"))
  }
  dat[k/10, 1:2] <- c(k^2, time)
  if (k/10 > 1) {
    fit <- with(dat, lm(time~k))
    est <- paste0("; Remaining: ~", time2char(predict(fit, end) - time), " mins")
  } else {
    est <- ""
  }
  cur <- format(Sys.time(), format="%I:%M:%S")
  elapsed <- time2char(time)
  #gsub("^0+", "", as.character(round(as.numeric(difftime(Sys.time(), tic, units = "mins")), 1)))
  cat(sprintf("%s of %s iterations (Current: %s; Elapsed: %s mins)\n", k, max.k, cur, elapsed, est)); flush.console()
}
topicmodels::LDA(x, k = k, method = method, control = control, ...)
})
out <- data.frame(
  k = c(2:max.k),
  logLik = sapply(best_model, logLik)
)
class(out) <- c("optimal_k", "optimal_k2", "data.frame")

```

```

    out
}

#' Plots a plot.optimal_k1 Object
#'
#' Plots a plot.optimal_k1 object
#'
#' @param x A \code{optimal_k1} object.
#' @param ... Ignored.
#' @method plot plot.optimal_k1
#' @export

plot.optimal_k1 <- function(x, ...){
  y <- attributes(x)[["k_dataframe"]]
  y <- y[y[["k"]]] == as.numeric(x), ]
  ggplot2::ggplot(attributes(x)[["k_dataframe"]], ggplot2::aes_string(x="k", y="harmonic_mean")) +
    ggplot2::xlab(sprintf("Number of Topics (Optimal Number: %s)", as.numeric(x))) +
    ggplot2::ylab("Harmonic Mean of Log Likelihood") +
    ggplot2::geom_smooth(method = "loess", fill=NA) +
    geom_point(data=y, color="red", fill=NA, size = 6, shape = 2
  1) +
    ggplot2::geom_line(size=1) +
    ggplot2::theme_bw() +
    ggplot2::theme(
      axis.title.x = ggplot2::element_text(vjust = -0.25, size =
  14),
      axis.title.y = ggplot2::element_text(size = 14, angle=90)
    )
}

#' Plots a plot.optimal_k2 Object
#' @param x A \code{optimal_k2} object.
#' @param ... Ignored.
#' @method plot plot.optimal_k2
#' @export

plot.optimal_k2 <- function(x, ...){
  ggplot2::ggplot(x, ggplot2::aes_string(x="k", y="logLik")) +
    ggplot2::xlab("Number of Topics") +
    ggplot2::ylab("Log Likelihood") +
    ggplot2::geom_smooth(size=.8, se=FALSE, method="loess") +
    ggplot2::geom_line(size=1) +
    ggplot2::theme_bw() +
    ggplot2::theme(
      axis.title.x = ggplot2::element_text(vjust = -0.25, size =
  14),
      axis.title.y = ggplot2::element_text(size = 14, angle=90)
    )
}

```

```

#' Prints a optimal_k object
#'
#' @param x A \code{optimal_k} object.
#' @param \ldots Ignored.
#' @method print optimal_k
#' @export

print.optimal_k <- function(x, ...){
  print(graphics::plot(x))
}

time2char <- function(x){
  x <- as.character(round(x, 1))
  if (identical("0", x)) return(x)
  gsub("^0+", "", x)
}

harmonicMean <- function(logLikelihoods, precision=2000L) {
  llMed <- Rmpfr::median(logLikelihoods)
  as.double(llMed - log(Rmpfr::mean(exp(-Rmpfr::mpfr(logLikelihoods, prec = precision) + llMed))))
}

if (!require("pacman")) install.packages("pacman"); library(pacman)
## Loading required package: pacman
pacman::p_load(ggplot2, topicmodels, Rmpfr)

```

## topicmodels2LDavis.R

```

#' Transform Model Output for Use with the LDavis Package
#'
#' Convert a \code{topicmodels} output into the JSON form required by the \code{LDavis} package.
#'
#' @param model A \code{\link[]\{topicmodel\}} object.
#' @param \ldots Currently ignored.
#' @seealso \code{\link[LDavis]\{createJSON\}}
#' @export
#' @examples
#'
#' data("AssociatedPress", package = "topicmodels")
#' model <- LDA(AssociatedPress[1:20,], control = list(alpha = 0.1), k = 3)
#' LDavis::serVis(topicmodels2LDavis(model))
#'

topicmodels2LDavis <- function(x, ...){
  post <- topicmodels::posterior(x)
  if (ncol(post[["topics"]]) < 3) stop("The model must contain >
```

```

2 topics")
mat <- x@wordassignments
LDAvis::createJSON(
  phi = post[["terms"]],
  theta = post[["topics"]],
  vocab = colnames(post[["terms"]]),
  doc.length = slam::row_sums(mat, na.rm = TRUE),
  term.frequency = slam::col_sums(mat, na.rm = TRUE)
)
}

```

## Exercises

For the purpose of this exercise, we will be using the collection of Harry Potter novels. This package holds the collection of Harry Potter books 1-7 by J.K. Rowling. Install these using:

```

if (packageVersion("devtools") < 1.6) {
  install.packages("devtools")
}
devtools::install_github("bradleyboehmke/harrypotter")
library(harrypotter)

```

We will also be using the following packages:

```

library(topicmodels) #topic modeling functions
library(stringr) #common string functions
library(tidytext) #tidy text analysis
library(tidyverse) #data manipulation and visualization
library(scales) #used for percent scale on confusion table

```

1. Perform preprocessing tasks
  - a. Create a tibble of the document, which is a combined column of title and chapter, and a text column which contains the corpus. [Hint: The harrypotter package makes indexing the text by chapter easy, as each book is already separated into its chapters.]
  - b. Use unnest\_tokens to take each individual word from the corpus.

- c. Remove stop\_words using anti\_join from the tidytext package.
  - d. Remove any proper name and other proper nouns that not add value to the meaning we are trying to extract from our data [Hint: look at the top words using top\_n(word\_counts, 10)].
  - e. Transform our term frequencies into a document-term matrix (dtm).
2. Perform Latent Dirichlet Allocation (LDA)
- a. Use to find the mixture of words that make up each topic and the mixture of topics that make up each document.
  - b. Plot the top terms
  - c. how much each document is associated with each topic.
  - d. Inspect visually how well our unsupervised learning was able to distinguish between the topics for each of the titles.
3. Unsupervised Classification to assign topics
- a. Use augment function to add the count of each term next to the topic the term has been assigned to. [Hint: This allows us to know how much each word weighs in on the topic assignment of the chapter, and ultimately title, as a whole.]
  - b. Investigate the proportion of assignments from one titles were assigned to another title using the confusion table.
  - c. Which terms were the were most frequent in the mis-assigned topics?

# CHAPTER 8 – Spatial Data Analysis

## Introduction

“I knew not what wild beast we were about to hunt down in the dark jungle of criminal London, but I was well assured from the bearing of this master huntsman that the adventure was a most grave one, while the sardonic smile which occasionally broken through his ascetic gloom boded little good for the object of our quest.”

—Dr. John Watson, The Adventure of the Speckled Band

Many problems faced by businesses, healthcare, education, and government involve populations and their geographic dispositions. Today’s nonhomogeneous society make data-based studies more complex than they were in less recent years. Examples include studies of epidemics, crime analysis, political polling, community service, and so on.

## Preliminaries

R offers a variety of functionality to perform mapping and population studies. Some of these packages are used in this module and are described below.

### *Key Libraries for Spatial Analysis*

- `ggmap`: extends the plotting package `ggplot2` for maps
- `rgdal`: R’s interface to the popular C/C++ spatial data processing library `gdal`
- `rgeos`: R’s interface to the powerful vector processing library `geos`
- `maptools`: provides various mapping functions
- `dplyr` and `tidyR`: fast and concise data manipulation packages
- `tmap`: a new package for rapidly creating beautiful maps
- `install.packages(x)`
- `install.packages(c("rgdal", "maptools", "dplyr", "tidyR", "tmap", "tmaptools", "rgeos"))`

## ***Loading Libraries***

```
library(maptools)
library(dplyr)
library(tidyr)
library(tmap)
library(rgdal)
library(rgeos)
library(ggplot2)
library(ggmap)
library(raster)
library(DT)
```

## ***Import India shape file (.SHP)***

The shapefile format is a popular geospatial vector data format for geographic information system (GIS) software. It is developed and regulated by Esri as a (mostly) open specification for data interoperability among Esri and other GIS software products. The shapefile format is a digital vector storage format for storing geometric location and associated attribute information. This format lacks the capacity to store topological information. However, due to its simplicity, it is now possible to read and write geographical datasets using the shapefile format with a wide variety of software, including R.

The term “shapefile” is quite common but is misleading since the format consists of a collection of files with a common filename prefix, stored in the same directory. The three mandatory files have filename extensions .shp, .shx, and .dbf. The actual shapefile relates specifically to the .shp file, but alone is incomplete for distribution as the other supporting files are required.

## ***Mandatory files***

.shp — shape format; the feature geometry itself

.shx — shape index format; a positional index of the feature geometry to allow seeking forwards and backwards quickly

.dbf — attribute format; columnar attributes for each shape, in dBase IV format

The files necessary for performing the operations contained in this paper are located at [https://github.com/stricje1/VIT\\_University/blob/master/Data\\_Analytics\\_2018/data/INDIA\\_geo.zip](https://github.com/stricje1/VIT_University/blob/master/Data_Analytics_2018/data/INDIA_geo.zip). It is best to download these files to your local machine and then configure your path in R to your computer.

```
mydir<-"C:/Users/jeff/Documents/Crime Analysis/India_data"
ind <- readOGR(dsn = mydir, layer = "INDIA")
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\jeff\Documents\Crime Analysis\India_d
ata", layer: "INDIA"
## with 35 features
## It has 1 fields
```

### **Import India Population Grid Files (.GRD)**

The first file we are going to load into R Studio is the “INDIA” shapefile located in the “data” folder of the project. It is worth looking at this input dataset in your file browser before opening it in R.

```
ind_pop<- raster("C:/Users/jeff/Documents/Crime Analysis/I
ndia_data/ind_pop.grd")
ind_msk_pop<- raster("C:/Users/jeff/Documents/Crime Analys
is/India_data/ind_msk_pop.grd")
```

### **Comments on Imported Data**

*In the second line of code above the readOGR function is used to load a shapefile and assign it to a new spatial object called “ind”; short for London. readOGR is a function which accepts two arguments: dsn which stands for “data source name” and specifies the directory in which the file is stored, and layer which specifies the file name (note that there is no need to include the file extension .shp). The file we assigned to the ind object contains the population of London Boroughs in 2001 and the percentage of the population participating in sporting activities.*

### **Examining the Output**

*Look at the output created (note the table format of the data and the column names). There are two important symbols at work in the above block of code: the @ symbol in the first line of code is used to refer to the data slot of the ind object. The \$ symbol refers to the Partic\_Per column (a variable within the table) in the data slot, which was i*

*dentified from the result of running the first line of code.*

### **Some Comments about Spatial Objects**

Spatial objects like the `ind` object are made up of several different slots, the key slots being `@data` (non-geographic attribute data) and `@polygons` (or `@lines` for line data). The data slot can be thought of as an attribute table and the geometry slot is the polygons that make up the physical boundaries. Specific slots are accessed using the `@ symbol`. We analyze the `INDIA` object with some basic commands.

#### **Shapefile Structure**

It is a data frame, `SpatialPolygonsDataFrame`, bearing this structure:

```
structure(ind)
## class      : SpatialPolygonsDataFrame
## features   : 35
## extent     : 68.1202, 97.41516, 6.754256, 37.13564
##           (xmin, xmax, ymin,
##                    ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +no_defs
## variables  : 1
## names      : ST_NAME
## min values : ANDAMAN AND NICOBAR ISLANDS
## max values : West Bengal
```

*All shapefiles have at least two slots, a bounding box and coordinate system (CRS). Using `str(ind@polygons)`, we get a stream of polygon slots, which show five slots (there are about 800 of these or 4000 slots).*

## **The Data Slots**

### **First Slot for Data Object**

*In the shape file structure, `data` occupies the first slot, which consist of 35 factors of State Names for India.*

`str(ind@data)` yields:

```
## 'data.frame':      35 obs. of  1 variable:
## $ ST_NAME: Factor w/ 35 levels "ANDAMAN AND NICOBAR
## ISLANDS",...: 1 2 3 4
##                   5 6 7 8 9 10 ...
```

## Second Slot for Polygons Object

The second slot is for `polygons`. Things to note are: (1) `labpt` or the centroid of the polygon, given by Longitude and Latitude; (2) the area of the polygon; (3) a logical indicating a hole or ring as the outer boundary; (4) a ring direction (+ or -); and (5) a numeric array with two columns of ordered coordinates. The bulk of the India shapefile is comprised of the polygon data.

`str(ind@polygons)` prints out more data than we could hope to read, so we included on piece below.

```
$ :Formal class 'Polygons' [package "sp"] with 5 slots
... . . . @ Polygons :List of 8
... . . . . $ :Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 76.1 19.4
... . . . . . @ area    : num 26.4
... . . . . . @ hole    : logi FALSE
... . . . . . @ ringDir: int 1
... . . . . . @ coords  : num [1: 32323, 1:2] 80.4 80.4 80.4
80.4 80.4 ...
... . . . . $ :Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.8 19.2
... . . . . . @ area    : num 1.11e-10
... . . . . . @ hole    : logi TRUE
... . . . . . @ ringDir: int -1
... . . . . . @ coords  : num [1:4, 1:2] 72.8 72.8 72.8
72.8 19.2 ...
... . . . . $ :Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.8 19.3
... . . . . . @ area    : num 2.83e-10
... . . . . . @ hole    : logi TRUE
... . . . . . @ ringDir: int -1
... . . . . . @ coords  : num [1:4, 1:2] 72.8 72.8 72.8
72.8 19.3 ...
... . . . . $ :Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.8 19.2
... . . . . . @ area    : num 4.35e-10
... . . . . . @ hole    : logi TRUE
... . . . . . @ ringDir: int -1
... . . . . . @ coords  : num [1:4, 1:2] 72.8 72.8 72.8
72.8 19.2 ...
... . . . . $ :Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.9 19
... . . . . . @ area    : num 5.45e-10
... . . . . . @ hole    : logi TRUE
... . . . . . @ ringDir: int -1
... . . . . . @ coords  : num [1:4, 1:2] 72.9 72.9 72.9
72.9 19 ...
... . . . . $ :Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.8 19.6
```

```

... . . . . . @ area    : num 1.45e-09
... . . . . . @ hole    : logi TRUE
... . . . . . @ ringDir: int -1
... . . . . . @ coords   : num [1:4, 1:2] 72.8 72.8 72.8
72.8 19.6 . .
... . . . . . $ : Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.8 19.2
... . . . . . @ area    : num 1.02e-07
... . . . . . @ hole    : logi TRUE
... . . . . . @ ringDir: int -1
... . . . . . @ coords   : num [1:6, 1:2] 72.8 72.8 72.8
72.8 72.8 . .
... . . . . . $ : Formal class 'Polygon' [package "sp"] with 5
slots
... . . . . . @ labpt   : num [1:2] 72.9 19.3
... . . . . . @ area    : num 0.000204
... . . . . . @ hole    : logi FALSE
... . . . . . @ ringDir: int 1
... . . . . . @ coords   : num [1:19, 1:2] 72.9 72.9 72.9
72.9 72.9 . .
... . . @ plotOrder: int [1:8] 1 8 7 6 5 4 3 2
... . . @ labpt    : num [1:2] 76.1 19.4
... . . @ ID        : chr "19"
... . . @ area      : num 26.4

```

### Third slot for Plotting Order

This is an integer vector of `plotOrder` or the plotting order of the 800 polygons. `str(ind@plotOrder)` yields:

```
## int [1:35] 29 19 20 2 33 14 11 16 26 7 ...
```

### Fourth Slot for Bounding Box Object

The fourth slot is for a `bbox` or bounding box or a 2x2 matrix with named dimensions of the max & min of the X and Y coordinates. The bounding box of the map produced by the INDIA shapefile is key slot, called by

`str(ind@box):`

```
##           min       max
## x 68.120198 97.41516
## y  6.754256 37.13564
```

### Fifth Slot for Class CRS

The fifth slot is for the proj4string of class CRS. `str(ind@proj4string)` yields:

```
## CRS arguments: +proj=longlat +ellps=WGS84  
+no_defs
```

## Headers

The head function in the first line of the code above simply means “show the first few lines of data” (try entering `head(ind@data)`, see `?head` for more details).

```
head(ind@data, n = 2)  
## ST_NAME  
## 0 ANDAMAN AND NICOBAR ISLANDS  
## 1 Andhra Pradesh
```

## Classes

To check the classes of all the variables in a spatial dataset, you can use the following command:

```
sapply(ind@data, class)  
## ST_NAME  
## "factor"
```

## Explore Spatial Structure

- To explore `ind` object further, try typing

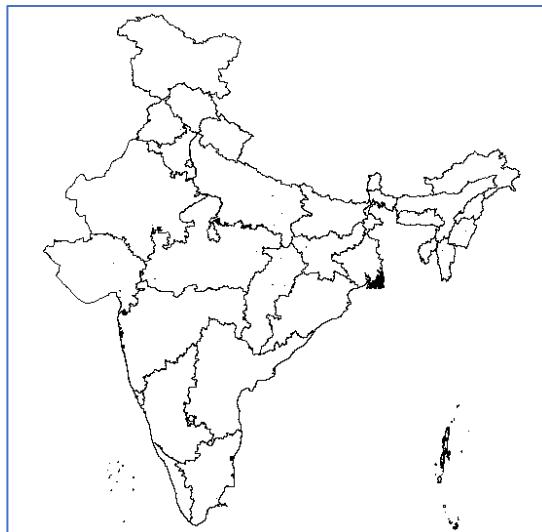
```
nrow(ind)  
## [1] 35  
ncol(ind)  
## [1] 1  
##  
ind@proj4string  
## CRS arguments: +proj=longlat +ellps=WGS84 +no_defs
```

The last part of the code chunk provides information about the map data. Its projection is given as latitude-longitude using the world grid system (WGS) 84 as the coordinate reference system (CRS). We will have more to say about this later.

## Plotting Layers

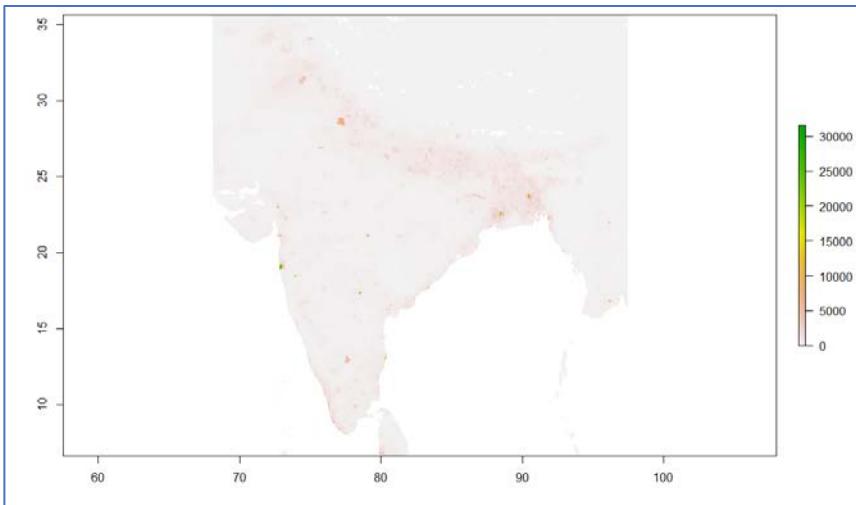
Now we have seen something of the structure of spatial objects in R, let us look at plotting them. Note, that plots use the geometry data, contained primarily in the `@polygons` slot.

```
plot(ind)
```



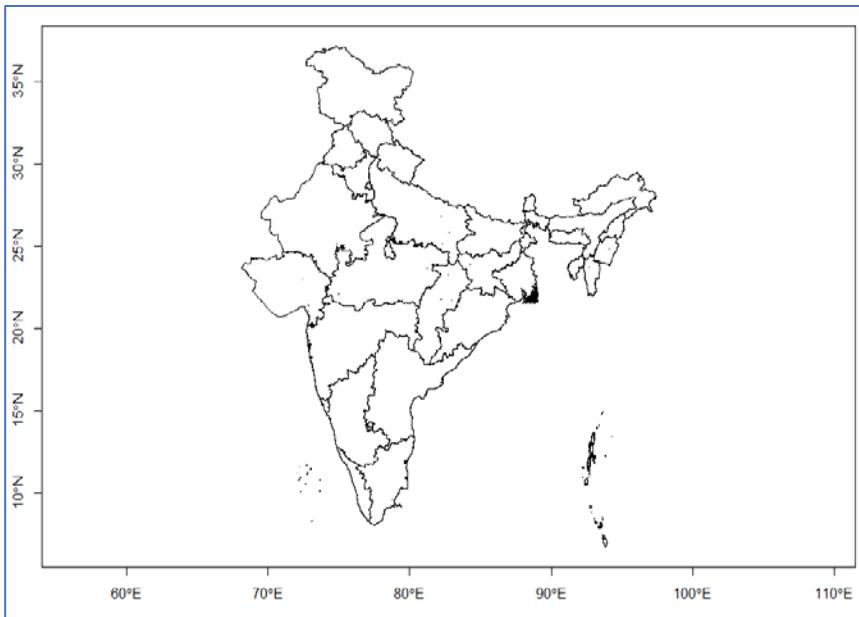
All layers do not come in shape files. `In_pop`, as we saw, comes as a `grd` (`grid`) file and we had to read it using the `raster` function. While we will demonstrate this kind of data, we will handle population in a different manner later on. So, for demonstration purposes, `plot(ind_pop)` prints the population mask layer for India. Perhaps you notice several “hot spots” on the map, representing cities with large populations. Mumbai (formerly called Bombay) is India’s most densely populated city. Located on the southwest coast of India, Mumbai appears as a small light-green dot on the population mask, indicating a population size of approximately  $20,000 \times 1000 = 20,000,000$  or 20 million. In 2011, the Indian census record its population as 18.41 million.

Notice that the hotspot in the plot below are not large in relation to the other data in the layer. The hotspots appear as relatively small orange spots, but rest assured the data underlying the plot really does represent the actual populated areas.



Proceeding on, we want to determine the center-mass of India, as our next several steps depend on this information. We do this using `raster::extent` and then plot the India map using the `xmin` and `xmax` from the data from the extent object. Notice that we now have a box with the lat-long grids with our India map.

```
raster::extent(ind)
## class       : Extent
## xmin        : 68.1202
## xmax        : 97.41516
## ymin        : 6.754256
## ymax        : 37.13564
plot(ind, xlim = c(68.1202, 97.41516), axes = TRUE)
```

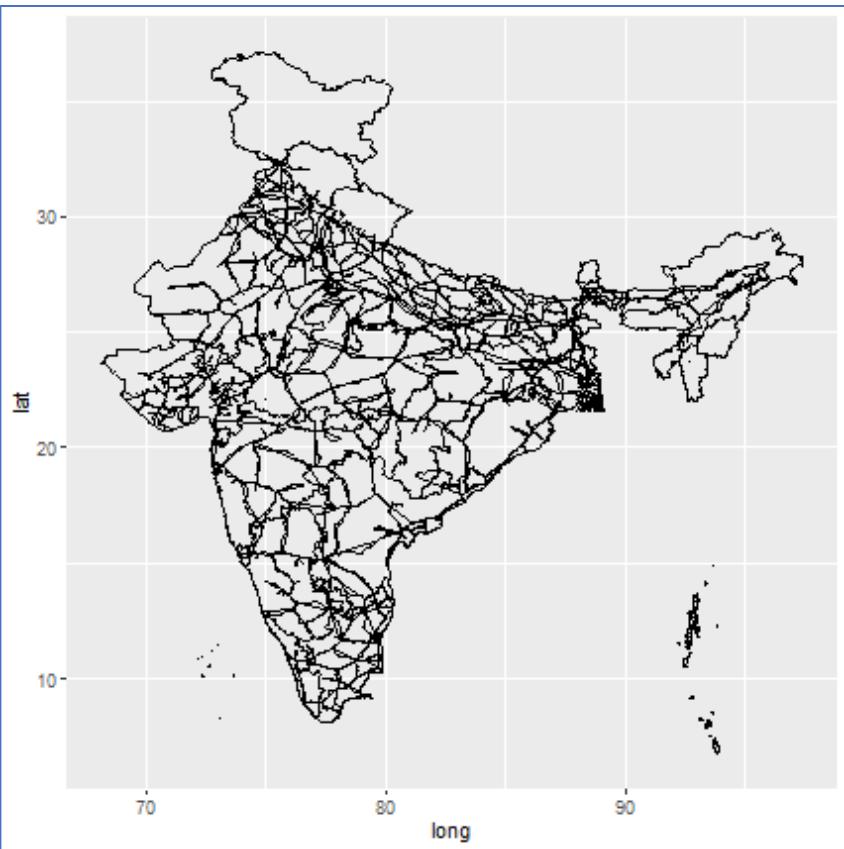


## Plotting Multiple Shape Data

We can easily overlay data from one shape file to another using `ggplot()`. The way to do this is creating a base map and then add an additional polygon data slot to the map. In this case, we want to use the `ind` polygon data as the base map and overlay the railroad system using `IND_rails.shp`.

```
map <- ggplot() + geom_polygon(data = ind, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
```

```
map + geom_polygon(data = ind_rails, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
```



## Apply Plotting Enhancements

We will consider this as our base map.

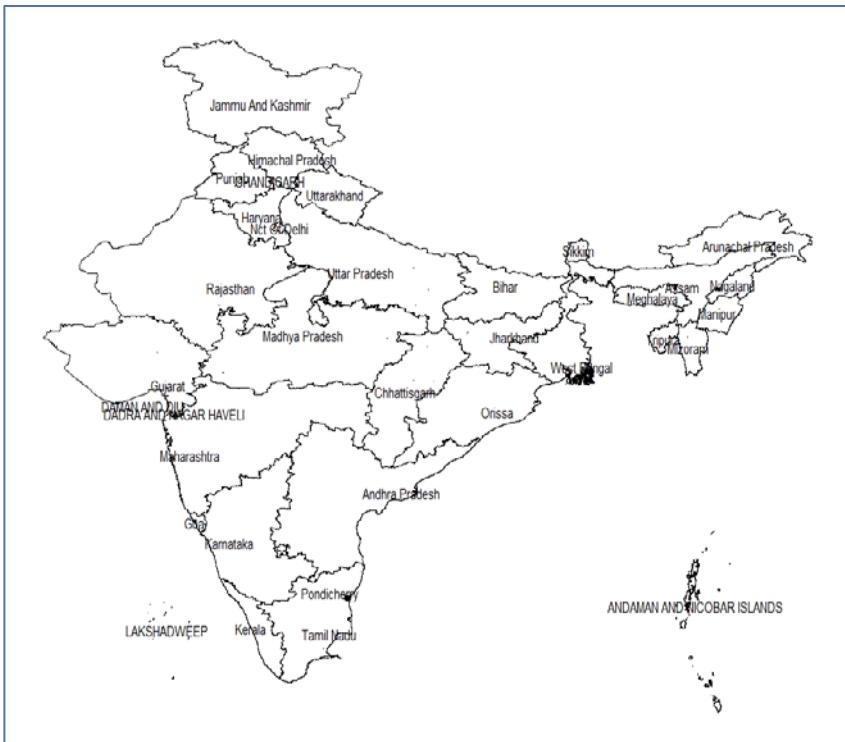
```
map <- ggplot() + geom_polygon(data = ind, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
## Regions defined for each Polygons
map + theme_void()
```



## Plot Map with Enhancements

Now, we will begin enhancing our base map.

```
map <- ggplot() + geom_polygon(data = ind, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
## Regions defined for each Polygons
cnames <- aggregate(cbind(long, lat) ~ id, data=shp_df, FUN=mean)
map + geom_text(data = cnames, aes(x = long, y = lat, label = id),
size = 4) + theme_void()
```



## Setup Bounding Box for Indian Map

```
subscr<-data.frame(lon=c(79.1,79.15,79.2),lat=c(12.85,12.9,12.95),
, pop=c(58,12,150))
coordinates(subscr)<-~lon+lat
proj4string(subscr)<-CRS("+init=epsg:4326")
lon <- c(78.9,79.3)
lat <- c(12.7,13.2)
map1<-get_map(location = c(lon[1], lat[2], lon[2], lat[1]),
, zoom=11)
## Map from URL : http://maps.googleapis.com/maps/api/stat
icmap?center=12.95,79.1&zoom=11&size=640x640&scale=2&mapty
pe=terrain&language=en-EN&sensor=false
```

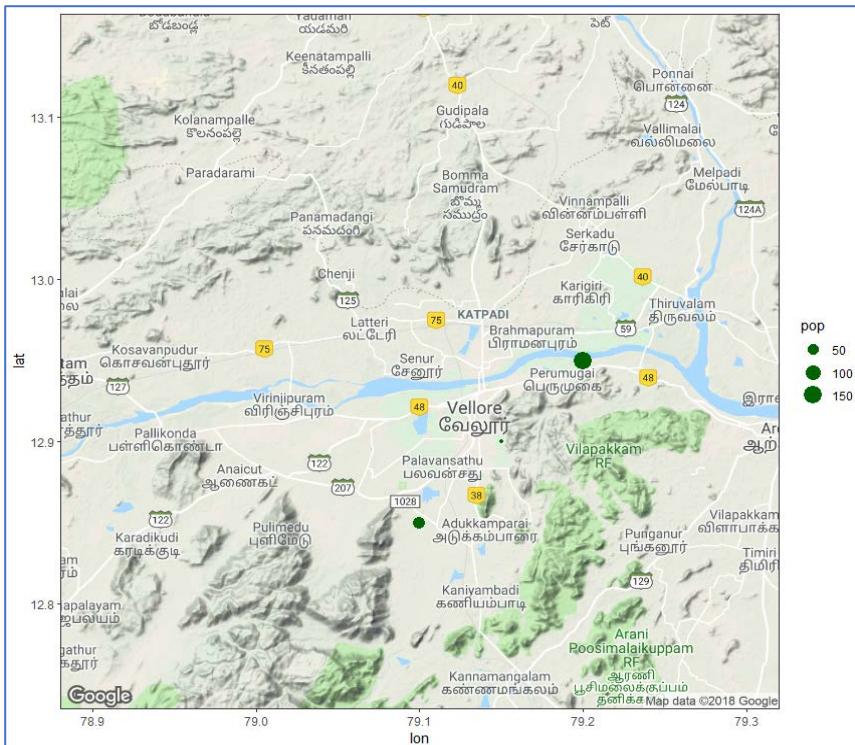
## Setup Bounding Box Plot

```
p <- ggmap(map1) +
  geom_point(data = as.data.frame(subscr),
```

```
aes(x = lon, y = lat, size=pop),
      colour = "darkgreen") +
theme_bw()
```

## Plot Bounding Box Map

```
print(p)
```



## Plot India Map with Grey Fill

This requires `rgeos` and will be a base-map for this section of work.

```
ind <- readOGR(dsn = mydir, layer = "INDIA")
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\jeff\Documents\Crime Analysis\India_data",
## layer: "INDIA"
## with 35 features
## It has 1 fields
plot(ind, col = "grey")
```

## Find Geographic centroids for Indian States

We will calculate centroids for India and each of its states. This will allow us to perform several tasks including dividing the country into quadrants and color-coding states.

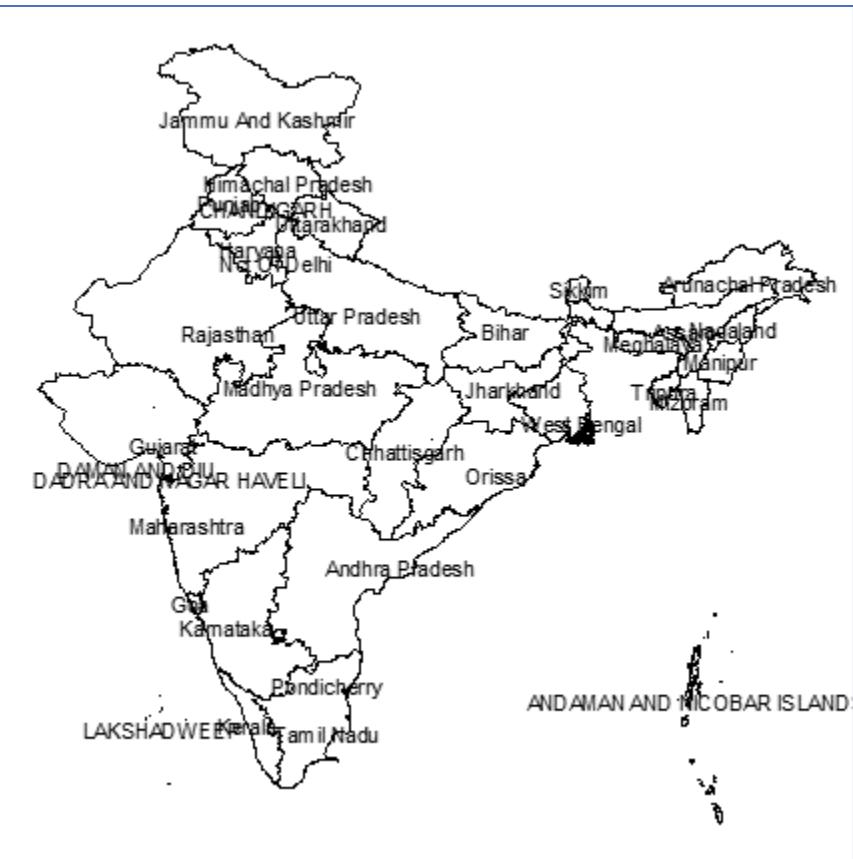
### India.shp Map Rescaled with lat/long Grids

```
require(ggplot2)
map <- ggplot() + geom_polygon(data = ind, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
# We printed this one earlier and will not show it here
## Regions defined for each Polygons
map + theme_void()
```

### Create C-Shape Dataframe

#### India.shp Map with States

```
map <- ggplot() + geom_polygon(data = ind, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
## Regions defined for each Polygons
cnames <- aggregate(cbind(long, lat) ~ id, data=shp_df, FUN=mean)
map + geom_text(data = cnames, aes(x = long, y = lat, label = id), size = 4) + theme_void()
```



Now, we will plot our India map with grey fill and calculate the center of mass or geographic centroid for each Indian state, using two different methods. Note that this requires the `rgeos` package. The centroids will appear on the map.

Using method 1, we find the centroid for Tamil Nau with a relatively large buffer zone of 10km.

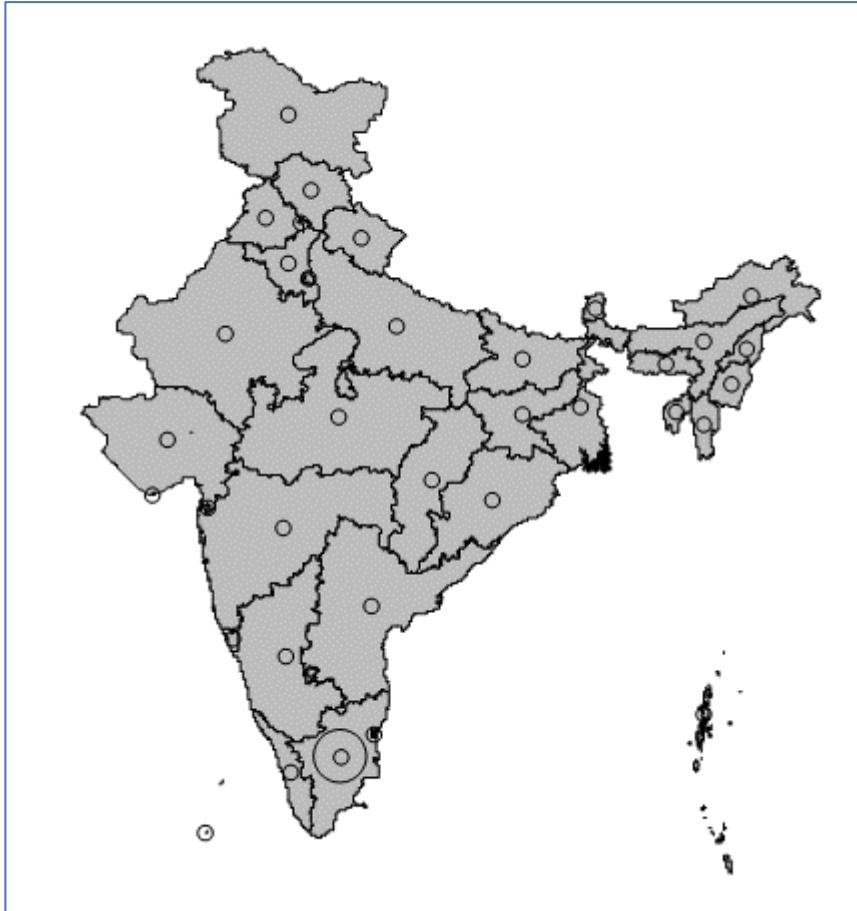
```
cent_ind <- gCentroid(ind[ind$ST_NAME == "Tamil Nadu",])

points(cent_ind, cex = 3)
ind_buffer <- gBuffer(spgeom = cent_ind, width = 10000)
```

Using method 2 of subsetting, selects only points within the buffer, and we do this for the remaining Indian states.

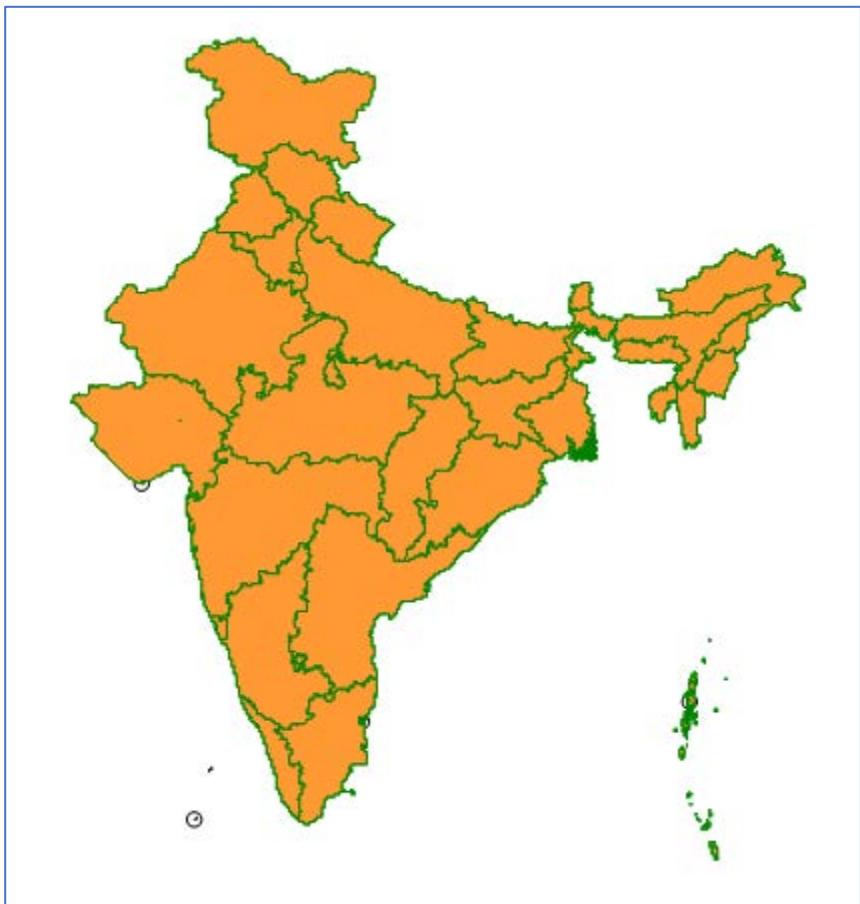
```
ind_cents <- SpatialPoints(coordinates(ind),
```

```
proj4string = CRS(proj4string(ind)) # create spatialpoints  
sel <- ind_cents[ind_buffer,] # select points inside buffer  
points(sel) # show where the points are located  
ind_central <- ind[sel,] # select zones intersecting w. sel
```



Now, we will recolor the India map with centroids. What do you say to saffron and green, the colors of India?

```
plot(ind_central, add = T, col = "#ff9933",  
      border = "#008000")  
plot(ind_buffer, add = T, border = "red", lwd = 2)
```



### *India.shp Map with Cities*

```
ggplot(india_pop, aes(X,Y)) + geom_point() + geom_text(aes(label=Centre)) +
  geom_polygon(data = ind, aes(x = long, y = lat, group = group), colour = "black", fill = NA)
## Regions defined for each Polygons
```

### **Selecting quadrants**

The code below should help understand the way spatial data work in R. It is used to find the geographical centroid of India.

```
lat <- coordinates(gCentroid(ind))[[1]]  
lng <- coordinates(gCentroid(ind))[[2]]
```

## Test for NE Quadrant Inclusion

The following comprise arguments to test whether or not a coordinate is east or north of the center.

```
east<-sapply(coordinates(ind)[,1],function(x) x>lat)  
north<-sapply(coordinates(ind)[,2],function(x) x>lng)
```

- test if the coordinate is east and north of the center

```
ind@data$quadrant[east & north] <- "northeast"
```

## Test for SE Quadrant Inclusion

```
east<-sapply(coordinates(ind)[,1],function(x) x>lat)  
south<-sapply(coordinates(ind)[,2],function(x) x<lng)
```

```
ind@data$quadrant[east & south] <- "southeast"
```

## Test for NW Quadrant Inclusion

```
west<-sapply(coordinates(ind)[,1],function(x) x<lat)  
north<-sapply(coordinates(ind)[,2],function(x) x>lng)
```

```
ind@data$quadrant[west & north] <- "northwest"
```

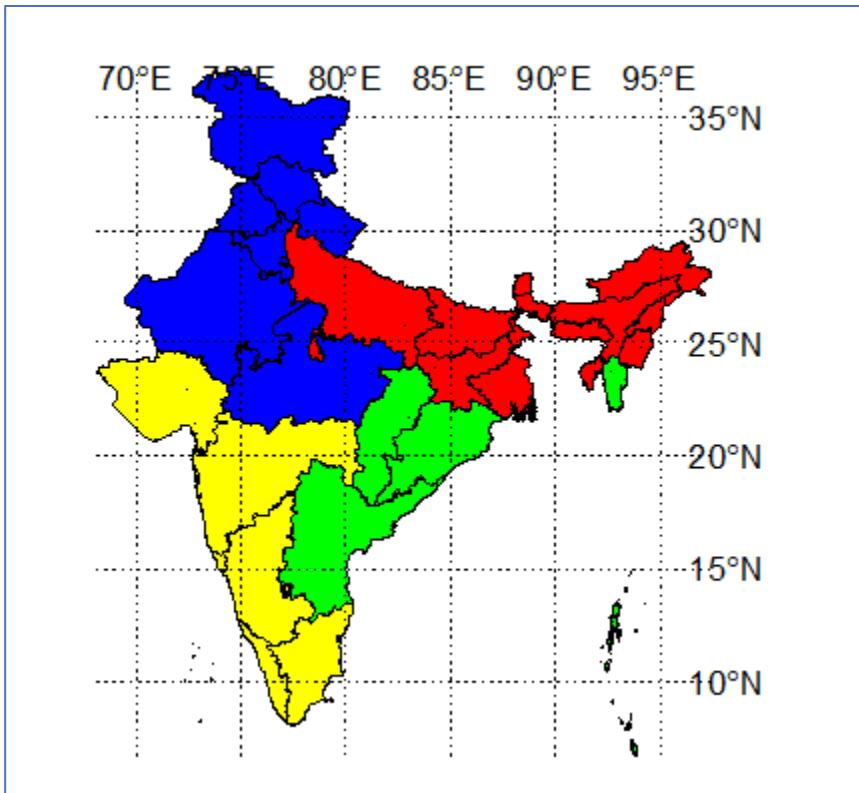
## Test for SW Quadrant Inclusion

```
west<-sapply(coordinates(ind)[,1],function(x) x<lat)  
south<-sapply(coordinates(ind)[,2],function(x) x<lng)
```

```
ind@data$quadrant[west & south] <- "southwest"
```

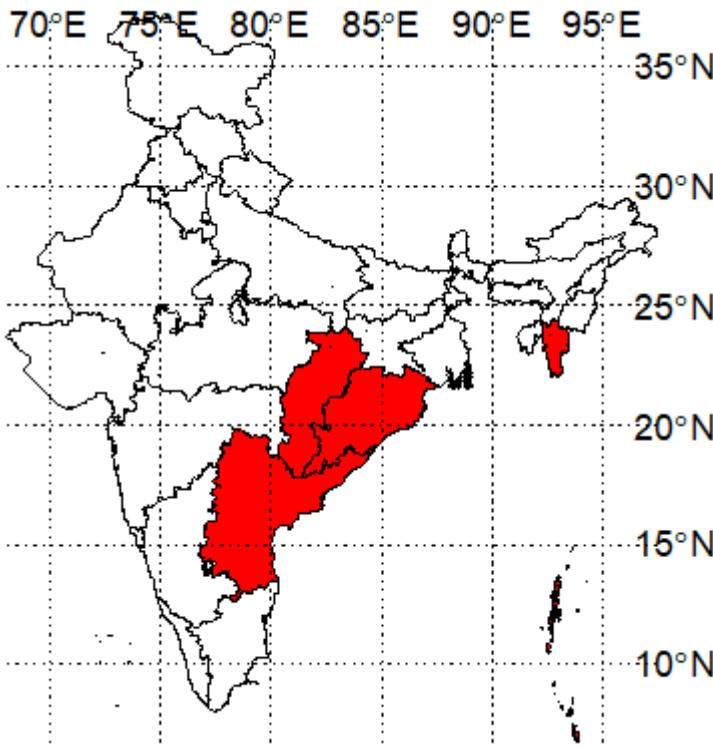
## Plot Quadrants with Colors

```
plot(ind)  
plot(ind[east & north,], col = "red", add = TRUE)  
plot(ind[east & south,], col = "green", add = TRUE)  
plot(ind[west & north,], col = "blue", add = TRUE)  
plot(ind[west & south,], col = "yellow", add = TRUE)
```



Now, we plot SE Quadrants with a red fill color.

```
ind$quadrant[east & north,] <- "northeast"
ind$quadrant[!east & !north] <- "southwest"
ind$quadrant[!east & north] <- "Northwest"
plot(ind)
plot(ind[east & !north,], add = TRUE, col = "red")
llgridlines(ind, lty= 3, side ="EN", offset = -0.5)
```



## Creating New R Object

Alongside visualisation and interrogation, a GIS must also be able to create and modify spatial data. R's spatial packages provide a very wide and powerful suite of functionality for processing and creating spatial data. R objects can be created by entering the name of the class we want to make. Vector and *data.frame* objects for example, can be created as follows:

```
vec <- vector(mode = "numeric", length = 3)
df <- data.frame(x = 1:3, y = c(1/2, 2/3, 3/4))
```

*Check the class of these new objects using class():*

```
class(vec)
## [1] "numeric"
class(df)
## [1] "data.frame"
```

## Creating New Spatial Data

The same logic applies to spatial data. The input must be a numeric matrix or data frame:

```
sp1 <- SpatialPoints(coords = df)
```

### *Comment on Creating Spatial Data*

We have just created a spatial points object, one of the fundamental data types for spatial data. (The others are lines, polygons and pixels, which can be created by `SpatialLines`, `SpatialPolygons` and `SpatialPixels`, respectively.) Each type of spatial data has a corollary that can accept non-spatial data, created by adding `DataFrame`. `SpatialPointsDataFrame()`, for example, creates points with an associated `data.frame`. The number of rows in this dataset must equal the number of features in the spatial object, which in the case of `sp1` is 3.

### *Add Data to an Existing Spatial Data File*

The code below extends the pre-existing object `sp1` by adding data from `df`. To see how strict spatial classes are, try replacing `df` with `mat` in the above code: it causes an error. All spatial data classes can be created in a similar way, although `SpatialLines` and `SpatialPolygons` are much more complicated (Bivand et al. 2013). More frequently your spatial data will be read-in from an externally-created file, e.g. using `readOGR()`. Unlike the spatial objects we created above, most spatial data comes with an associate ‘CRS’.

```
class(sp1)
## [1] "SpatialPoints"
## attr(,"package")
## [1] "sp"
spdf <- SpatialPointsDataFrame(sp1, data = df)
class(spdf)
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

## Projections: setting and transforming CRS in R

The Coordinate Reference System (CRS) of spatial objects defines where they are placed on the Earth’s surface. You may have noticed

`proj4string` in the summary of `ind` above: the information that follows represents its CRS. Spatial data should always have a CRS.

If no CRS information is provided, and the correct CRS is known, it can be set as follow:

```
proj4string(ind) <- NA_character_ # remove CRS information  
from ind proj4string(ind) <- CRS("+init=epsg:27700") #  
assign a new CRS
```

### Comments on CRS Changes

R issues a warning when the CRS is changed. This is so the user knows that they are simply changing the CRS, not reprojecting the data. An easy way to refer to different projections is via EPSG codes. Under this system 27700 represents the British N ‘WGS84’ (epsg:4326) is a very commonly used CRS worldwide.

### Find EPSG Codes

The following code shows how to search the list of available EPSG codes and create a new version of `ind` in WGS84:3

```
proj4string(ind)  
## [1] "+proj=longlat +ellps=WGS84 +no_defs"
```

### Convert the Coordinates

`spTransform` converts the coordinates of `ind` into the widely used WGS84CRS.

```
CRS("+init=epsg:4326")  
## CRS arguments:  
## +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +  
## ellps=WGS84  
## +towgs84=0,0,0  
CRS("+init=epsg:3785")  
## CRS arguments:  
## +init=epsg:3785 +proj=merc +a=6378137 +b=6378137 +lat_  
## ts=0.0  
## +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@n  
## ull  
## +no_defs
```

### Change the EPSG

```
EPSG <- make_EPSG() # create data frame of available EPSG  
codes
```

```

EPSG[grep1("WGS 84$", EPSG$note), ] # search for WGS 84 code
##      code      note
prj4
## 249 4326 # WGS 84           +proj=longlat +datum=WGS84
+no_defs
## 5311 4978 # WGS 84 +proj=geocent +datum=WGS84 +units=m
+no_defs
ind84 <- spTransform(ind, CRS("+init=epsg:3785")) # reproject
ind84
## class       : SpatialPolygonsDataFrame
## features    : 35
## extent      : 7583106, 10844206, 753627.8, 4458031  (xm
in, xmax, ymin, ymax)
## coord. ref. : +init=epsg:3785 +proj=merc +a=6378137 +b=
6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +uni
ts=m +nadgrids=@null +no_defs
## variables   : 2
## names       : ST_NAME, quadrant
## min values  : ANDAMAN AND NICOBAR ISLANDS, northeast
## max values  : West Bengal, southwest

```

### Saving CRS Formats

Now we've transformed `ind` into a more widely used CRS, it is worth saving it. R stores data efficiently in RData or Rds formats. The former is more restrictive and maintains the object's name, so we use the latter

```
saveRDS(object = ind84, file = "C:/Users/jeff/Documents/Crime Analysis/Creating-maps-in-R-master/data/ind84.Rds")
```

### Reset INDIA Shapefile

To reaffirm our starting point, let's re-load the "INDIA" shapefile as a new object and plot it: Create new object called "ind" from "INDIA" shapefile

```

ind <- readOGR(dsn = mydir, "INDIA")
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\jeff\Documents\Crime Analysis\India_data", layer: "INDIA"
## with 35 features
## It has 1 fields
plot(ind) # Not shown

```

We are not showing the plot, but we can check that the 35 rows of data are still in the file.

```
nrow(ind)
## [1] 35
```

## Joining Non-Spatial and Spatial Data with same Keys

The non-spatial data we are going to join to the `ind` object contains records of population in India. This is stored in a comma separated values (.csv) file called “[India\\_pop\\_2011](#)”.

If you open the file in a separate spreadsheet application first, we can see each row represents a single city’s population. We are going to use a function called `aggregate` to aggregate the population at the city, ready to join to our spatial `ind` dataset. A new object called `india_pop` is created to store this data.

### [India 2011 Population](#)

The 15th Indian Census was conducted in two phases, house listing and population enumeration. House listing phase began on 1 April 2010 and involved collection of information about all buildings. Information for National Population Register was also collected in the first phase, which will be used to issue a 12-digit unique identification number to all registered Indian residents by Unique Identification Authority of India (UIDAI).

The second population enumeration phase was conducted between 9 and 28 February 2011. Census has been conducted in India since 1872 and 2011 marks the first-time biometric information was collected. According to the provisional reports released on 31 March 2011, the Indian population increased to 1.21 billion with a decadal growth of 17.64%.

```
india_pop <- read.csv("C:/Users/jeff/Documents/Crime Analysis/India_data/India_pop_2011.csv", stringsAsFactors = FALSE)
datatable(india_pop, options = list(scrollX='400px')) # retain price information about key commodities by city
```

Show 10 entries

Rank Date Centre ST\_NAME Region Country X Y Pop\_2011 X1

Search:

Rank	Date	Centre	ST_NAME	Region	Country	X	Y	Pop_2011	X1	
1	64	2011	Port Blair	ANDAMAN AND NICOBAR ISLANDS	INDIA	92.4416	11.4006	100186		
2	13	2011	VISAKHAPATNAM	Andhra Pradesh	SOUTH INDIA	83.218482	17.686816	2035922		
3	28	2011	VILAYWADA	Andhra Pradesh	SOUTH INDIA	80.648015	16.506174	1034558		
4	66	2011	Itanagar	Arunachal Pradesh	EAST INDIA	93.3712	27.06	59490		
5	32	2011	GUWAHATI	Assam	NORTH EAST INDIA	91.736237	26.144518	963429		
6	17	2011	PATNA	Bihar	EAST INDIA	85.137565	25.594095	1683200		
7	45	2011	BHAGALPUR	Bihar	EAST INDIA	86.98243	25.347799	398138		
8	52	2011	PURNIA	Bihar	EAST INDIA	87.475255	25.777139	280547		
9	34	2011	CHANDIGARH	CHANDIGARH	NORTH INDIA	76.779419	30.733315	960787		
10	30	2011	RAIPUR	Chhattisgarh	WEST INDIA	81.629641	21.251384	1010087		

< >

Showing 1 to 10 of 69 entries

Previous [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) Next

## Joining Tables

We use `left_join` because we want the length of the data frame to remain unchanged, with variables from new data appended in new columns (see `?left_join`). The `*join` commands (including `inner_join` and `anti_join`) assume, by default, that matching variables have the same name. Here we will specify the association between variables in the two data sets:

```

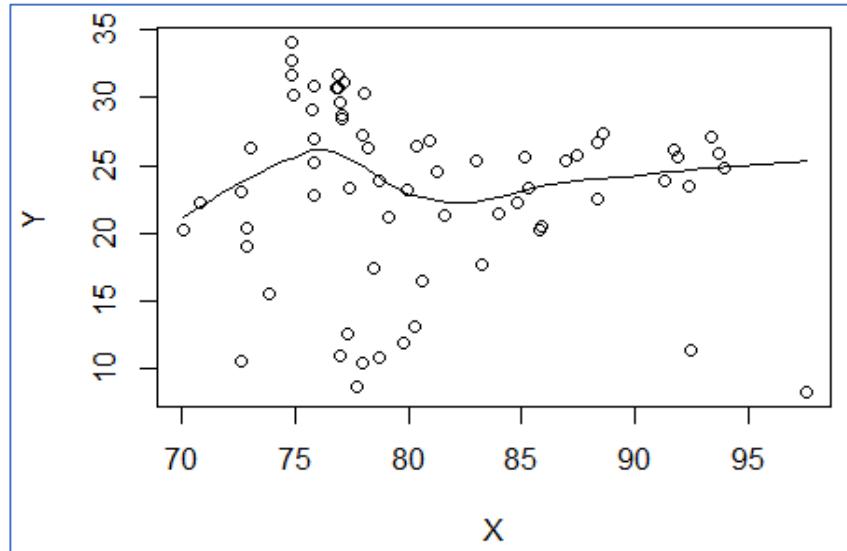
head(ind$ST_NAME) # dataset to add to (results not shown)
## [1] ANDAMAN AND NICOBAR ISLANDS Andhra Pradesh
## [3] Arunachal Pradesh Assam
## [5] Bihar CHANDIGARH
## 35 Levels: ANDAMAN AND NICOBAR ISLANDS ... West Bengal
head(india_pop$ST_NAME) # the variables to join
## [1] "Andaman & Nicobar Islands" "Andhra Pradesh"
## [3] "Andhra Pradesh"           "Arunachal Pradesh"
## [5] "Assam"                  "Bihar"
ind@data <- left_join(ind@data, india_pop, by = c('ST_NAME' 
  ' = 'ST_NAME'))
## Warning: Column `ST_NAME` joining factor and character
vector, coercing
## into character vector

```

### Basic Scatterplots

Recall that we plot x versus Y using scatterplots to visualize two-dimensional data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis.

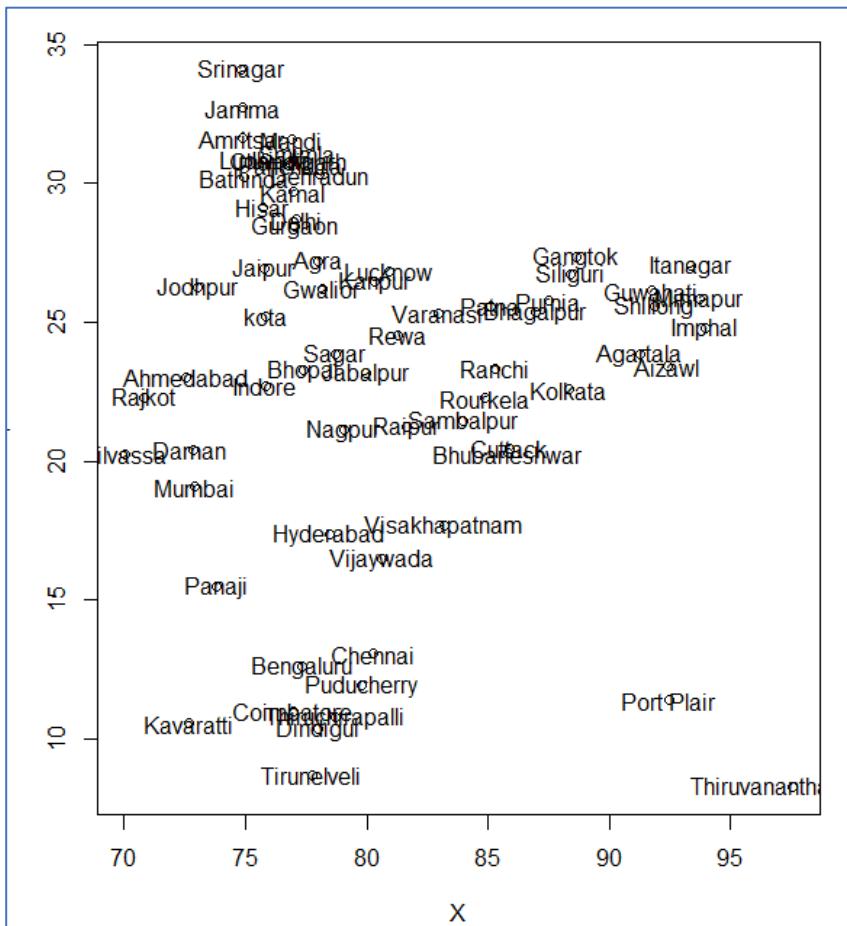
```
with(india_pop, scatter.smooth(X, Y))
```



The basic scatterplot in Figure shows the cities contained in the population data (with no labels) and a smooth fitted function. The fitted

function is passing through the center of India, dividing north and south, but is not necessary for performing our analysis, but demonstrates this feature for latter chapters.

```
attach(india_pop)
plot(X,Y) + text(X,Y, labels=india_pop$Centre)
```

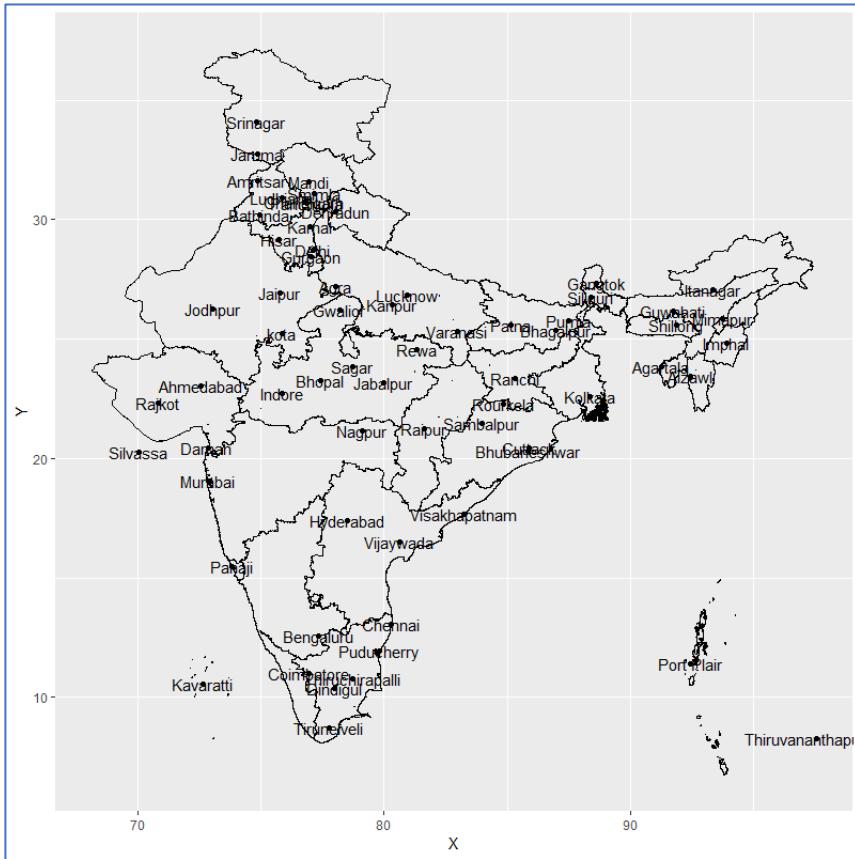


```
attach(india_pop)
#ggplot(X, Y, labels=india_pop$Centre)
```

### India.shp Map with Cities

```
ggplot(india_pop, aes(X,Y)) + geom_point() + geom_text(aes(label=Centre)) +
```

```
geom_polygon(data = ind, aes(x = long, y = lat, group =  
group), colour = "black", fill = NA)  
## Regions defined for each Polygons
```



## *India Map with States Setup*

```
library(tmaptools)
ind_wgs = spTransform(ind, CRS("+init=epsg:4326"))
osm_tiles = read_osm(bbox(ind_wgs))
ind_wgs$ST_NAME <- ind$ST_NAME
```

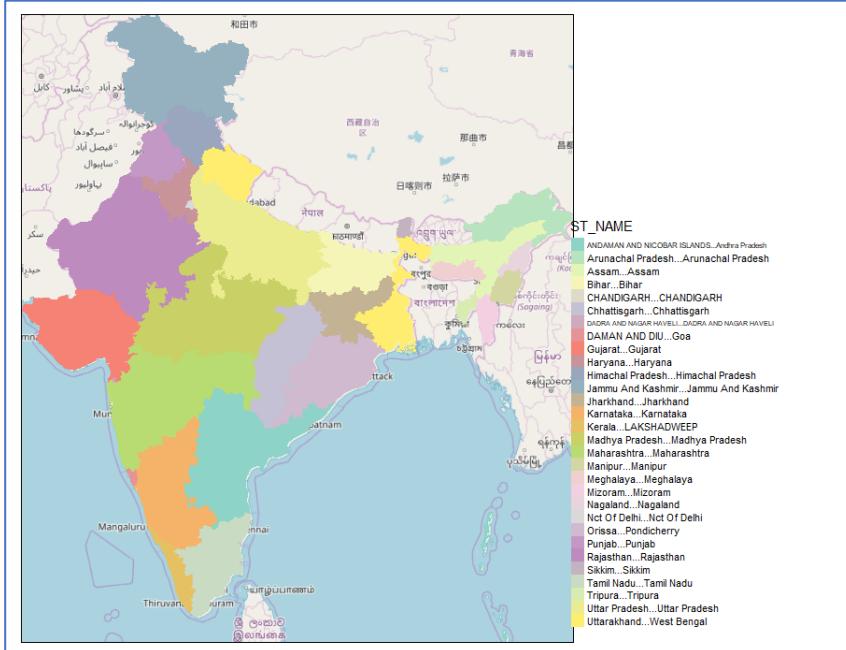
## **Plot India Map with States (colored)**

```
tm1<-tm_shape(osm_tiles) +  
  tm_raster() +  
  tm_shape(ind_wgs) +
```

```

tm_fill("ST_NAME", legend.show=TRUE) +
tm_layout(legend.position = c(0.99,0.02))
tm1

```



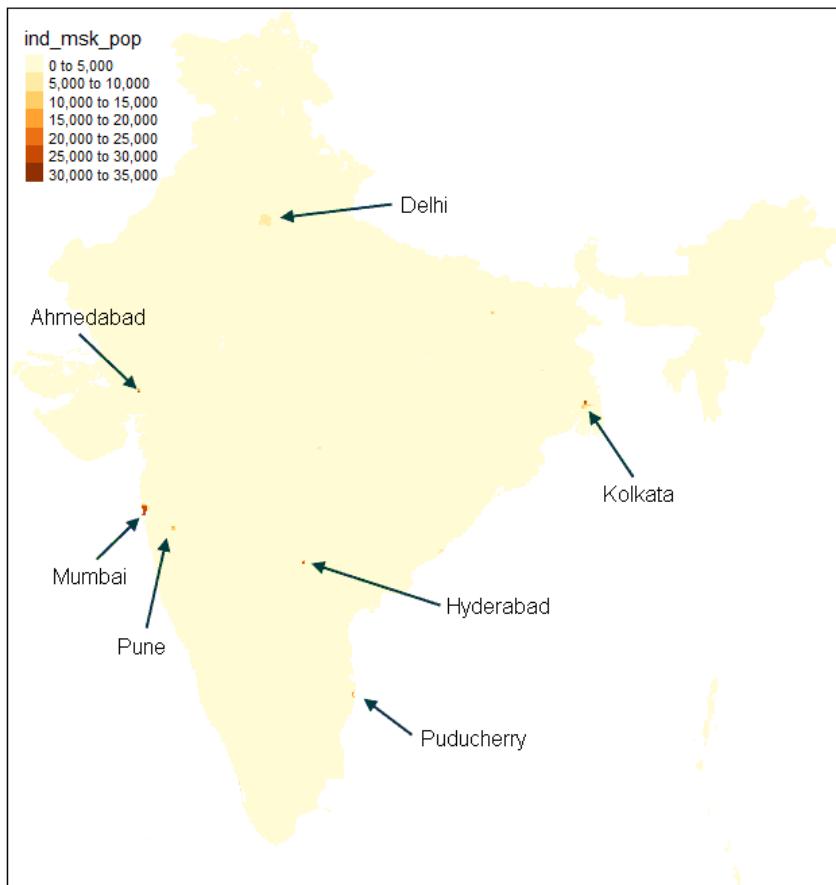
### India Population Mask (colored)

Mumbai appears more vividly on this population mask as a hot-read spot. Kolkata, formerly Calcutta, appears on the northeast coast and Chennai on the southeast cost as lighter shades of red. Delhi appears in the north as a larger orange area. Between Mumbai and Chennai is the red dot of Hyderabad, India's sixth largest city. North of Mumbai is Ahmedabad, another red dot. According to the scale, the hotter the dot, the more populous the city.

```

tm2<-tm_shape(ind_msk_pop) +
  tm_raster()
tm2

```



### Define Popups for Leaflet Map

The following code defines the “popups” we will use in a `leaflet` generated map.

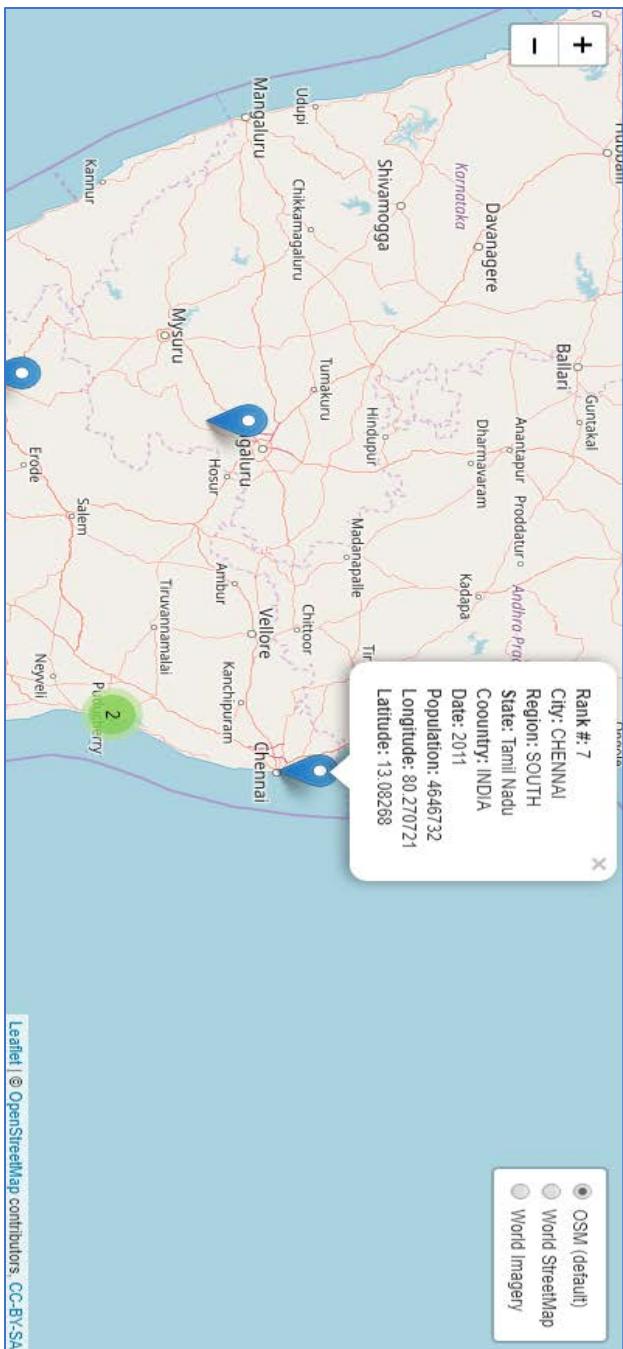
```
india_pop$popup <- paste("Rank #: <b>", india_pop$Rank
,
"<br>", "<b>City: </b>", india_pop$Centre,
"<br>", "<b>Region: </b>", india_pop$Region,
"<br>", "<b>State: </b>", india_pop$ST_NAME,
"<br>", "<b>Country: </b>", india_pop$Country,
"<br>", "<b>Date: </b>", india_pop>Date,
"<br>", "<b>Population: </b>", india_pop$Pop_2011,
"<br>", "<b>Longitude: </b>", india_pop$X,
"<br>", "<b>Latitude: </b>", india_pop$Y)
```

## India Leaflet Population Map

Using `leaflet`, we now generate a map of India with popups defined above. The figure is zoomed-in to southern India and show the popup for Chennai.

```
library(leaflet)

leaflet(india_pop, 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 = india_pop$popup,
  clusterOptions =
    markerClusterOptions()) %>%
  addLayersControl(
    baseGroups = c("OSM (default)", "World StreetMap", "World Imagery"),
    options = layersControlOptions(collapsed = FALSE)
  )
```



## Joining Non-Spatial Data with different Keys

The non-spatial data we are going to join to the `ind` object contains records of crimes in India. This is stored in a comma separated values (.csv) file called “`ind-recordedcrime-state`”. If you open the file in a separate spreadsheet application first, we can see each row represents a single reported crime type. We are going to use a function called `aggregate()` to aggregate the crimes at the state-level, ready to join to our spatial `ind` dataset. We create a new object called `crime_data` to store this data. Before we create and look at new `crime_data` object, we want to “refresh” the `ind` dataset by reloading it.

### Import India shape file (.SHP)

```
mydir<-“C:/Users/jeff/Documents/Crime Analysis/India_data”
ind <- readOGR(dsn = mydir, layer = “INDIA”)
## OGR data source with driver: ESRI Shapefile
## Source: “C:\Users\jeff\Documents\Crime Analysis\India_d
ata”, layer: “INDIA”
## with 35 features
## It has 1 fields
```

### Import the Crime Data

```
crime_data <- read.csv(“C:/Users/jeff/Documents/VIT_Course
_Material/Data_Analytics_2018/data/india-recordedcrime-sta
te.csv”, stringsAsFactors = FALSE)
head(crime_data$CrimeType) # information about crime type
## [1] “Murder”    “Rape”       “Kidnapping Abduction”
## [4] “Dacoity”   “Robbery”    “Arson”
```

We extract “Total” Crimes:

```
crime_theft <- crime_data[crime_data$CrimeType == “Total”,
]
head(crime_theft, 2) # take a Look at the result (replace
2 with 10 to see more rows)
##      X Year CrimeType CrimeDetails CrimeCount
State
## 11 11  201    Total    Total   805 Andhra Pradesh
## 22 22 2011    Total    Total    34 Arunachal Pradesh
```

We calculate the Sum of the Crime Count by District:

```
crime_ag <- aggregate(CrimeCount ~ State, FUN = sum, data
= crime_theft)
```

Now, we show the First Two Rows of Aggregated Data:

```
head(crime_ag, 2)
##                                     State CrimeCount
## 1 ANDAMAN AND NICOBAR ISLANDS          7
## 2 Andhra Pradesh                      805
```

### Exploration Comments

You should not expect to understand all of this upon first try: simply typing the commands and thinking briefly about the outputs is all that is needed at this stage. Here are a few things that you may not have seen before that will likely be useful in the future: In the first line of code when we read in the file we specify its location (check in your file browser to be sure). The == function is used to select only those observations that meet a specific condition i.e. where it is equal to, in this case all crimes involving “Total”. The ~ symbol means “by”: we aggregated the CrimeCount variable by the district name.

### Indian States

Now, that we have crime data at the state level, the challenge is to join it to the `ind` object. We will base our join on the State variable from the `crime_ag` object and the name variable from the `ind` object. It is not always straight-forward to join objects based on names as the names do not always match. Now, we will see which names in the `crime_ag` object matches the spatial data object, `ind`:

```
ind$ST_NAME %in% crime_ag$State
## [1] TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [8] TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [15] TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [22] TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [29] TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
ind$name[ !ind$ST_NAME %in% crime_ag$State]
## NULL
```

### Comments on the Code

The first line of code above uses the `%in%` command to identify which values in `ind$ST_NAME` are also contained in the State names of the

aggregated crime data. The results indicate that all but one of the state names matches. The second line of code tells us that it is not in ‘India’.

The state name in the crime data does not match `ind$ST_NAME` is ‘NULL’. Check this by typing

```
crime_ag$State[ !crime_ag$State %in% ind$ST_NAME ]  
## character(0)
```

This tells us that there is an “extra” state in the crime data.

### ***Joining Spatial and Non-Spatial Tables***

Having checked the data found that one state does not match, we are now ready to join the spatial and non-spatial datasets. We recommend using the `left_join` function from the `dplyr` package but you could also use the `merge` function.

We use `left_join` because we want the length of the data frame to remain unchanged, with variables from new data appended in new columns (see `?left_join`). The join commands include `inner_join` and `anti_join` statements. These assume, by default, that matching variables have the same name. Here we will specify the association between variables in the two data sets, namely `ST_NAME` and `State`:

```
library(dplyr) # Load dplyr  
head(ind$ST_NAME) # dataset to add to (results not shown)  
## [1] ANDAMAN AND NICOBAR ISLANDS Andhra Pradesh  
## [3] Arunachal Pradesh Assam  
## [5] Bihar CHANDIGARH  
## 35 Levels: ANDAMAN AND NICOBAR ISLANDS ... West Bengal  
head(crime_ag$State) # the variables to join  
## [1] "ANDAMAN AND NICOBAR ISLANDS" "Andhra Pradesh"  
## [3] "Arunachal Pradesh" "Assam"  
## [5] "Bihar" "CHANDIGARH"  
ind@data <- left_join(ind@data, crime_ag, by = c('ST_NAME' = 'State'))  
## Warning: Column `ST_NAME` / `State` joining factor and character vector,  
## coercing into character vector
```

## **Explore the New Dataset**

Look at the new `ind@data` object. You should see new variables added, meaning the attribute join was successful. You can now plot the rate of theft crimes in India by state.

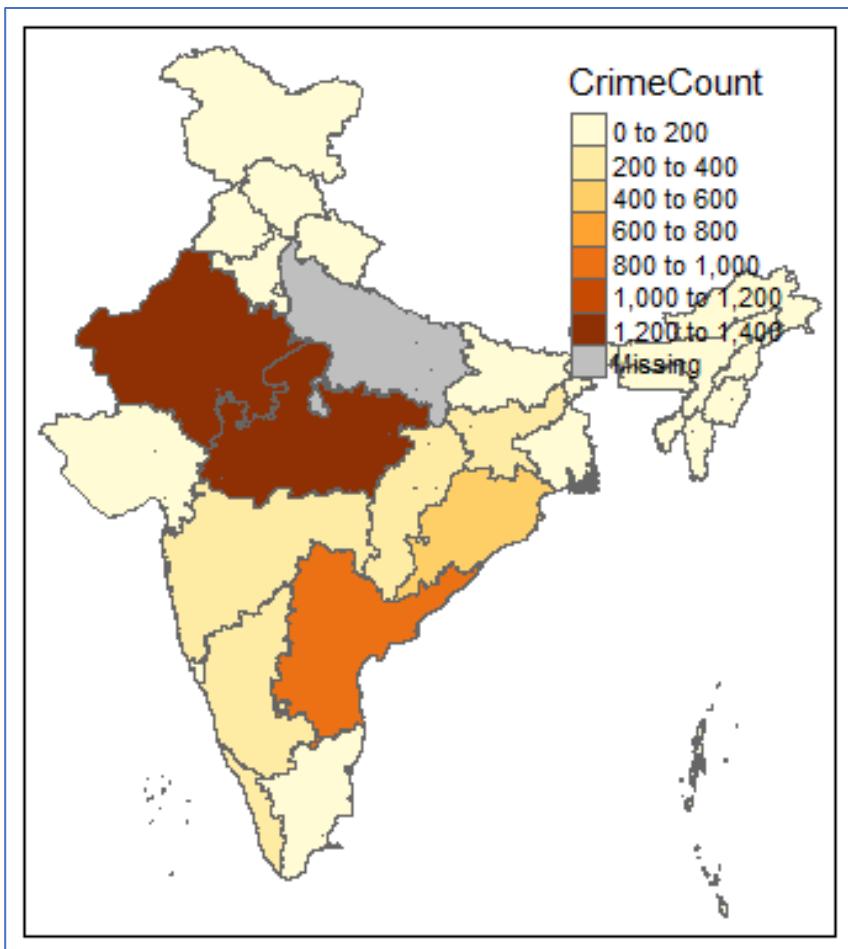
## **Making maps with tmap**

tmap was created to overcome some of the limitations of base graphics and ggmap. A concise introduction to tmap can be accessed (after the package is installed) by using the vignette function:

```
library(tmap)
vignette("tmap-getstarted")
## starting httpd help server ... done
```

### **Plot a Basic Map**

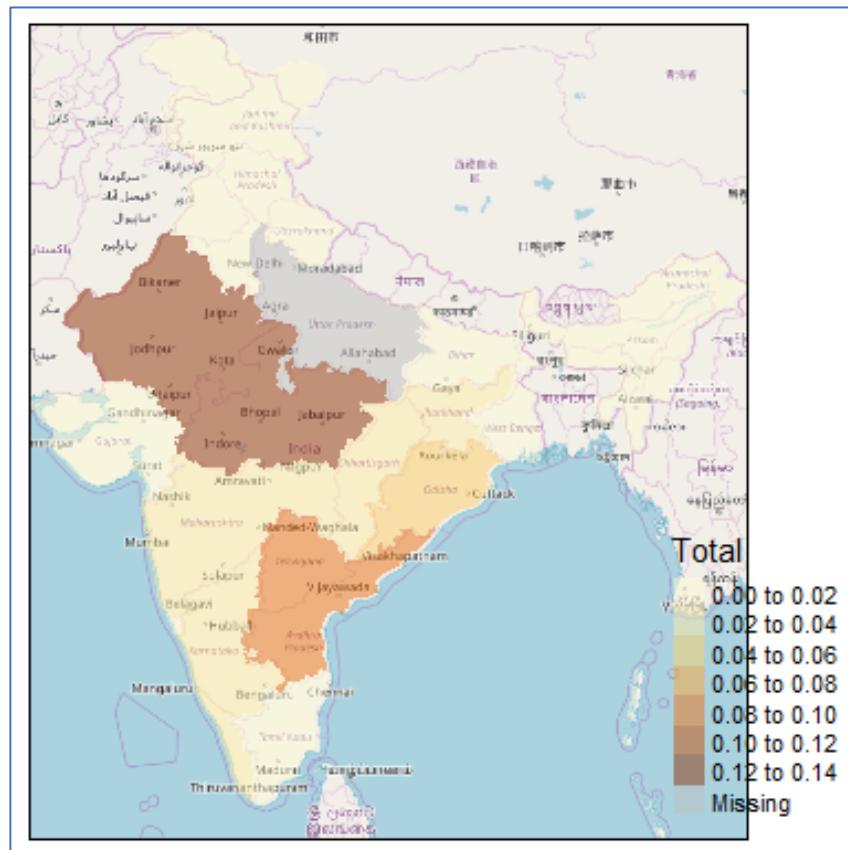
```
library(tmap) # Load tmap package
qtm(ind, "CrimeCount") # plot the basic map
## Linking to GEOS 3.6.1, GDAL 2.2.3, proj.4 4.9.3
```



### Plot and Enhanced Map

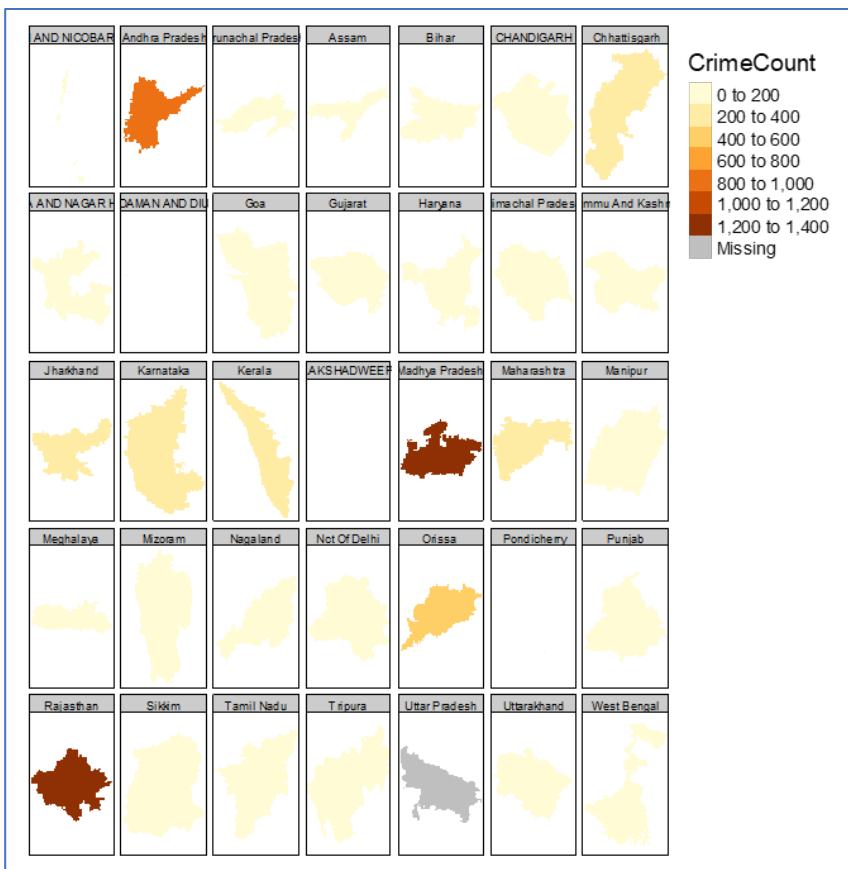
```
library(tmap)
library(tmaptools)
ind_wgs = spTransform(ind, CRS("+init=epsg:4326"))
osm_tiles = read_osm(bbox(ind_wgs))
## Warning: Current projection unknown. Long lat coordinates (wgs84) assumed.
ind_wgs$Total <- ind$CrimeCount / 10000
tm_shape(osm_tiles) +
  tm_raster() +
  tm_shape(ind_wgs) +
  tm_fill("Total", fill.title = "TotalCrimes\n(10000)", sc
```

```
ale = 0.8, alpha = 0.5) +  
  tm_layout(legend.position = c(0.89,0.02))
```



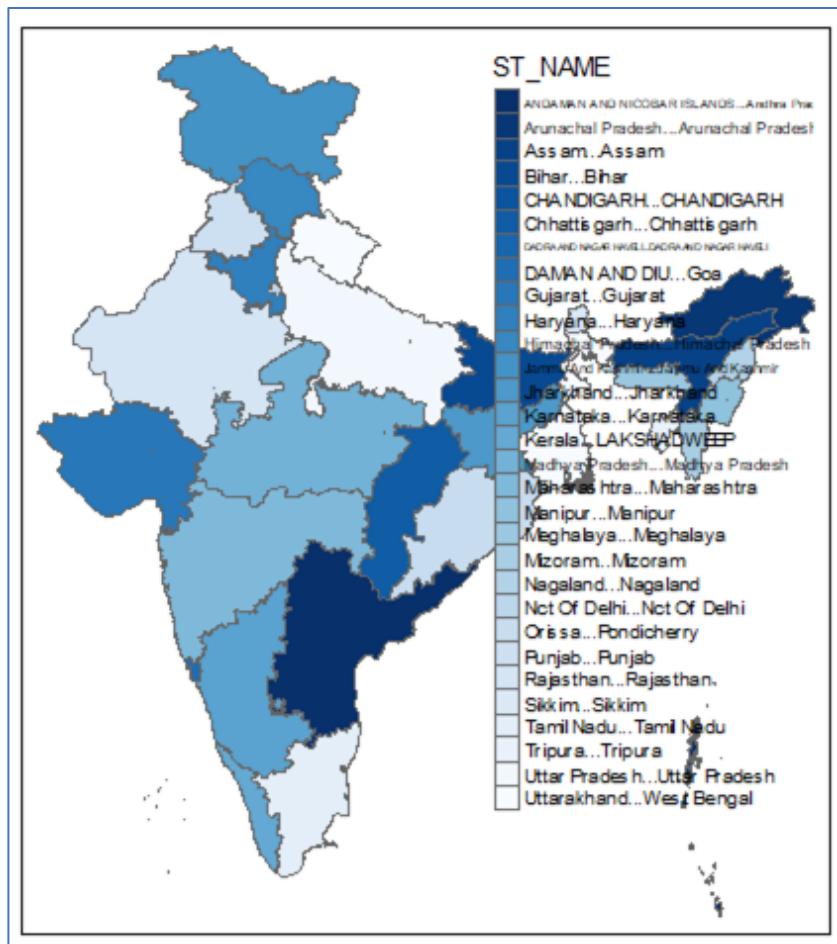
Now, we can compile a map that shows each state in a matrix, color filled for its crime count, using `tm_facets()`.

```
tm_shape(ind) +  
  tm_fill("CrimeCount", thres.poly = 0) +  
  tm_facets("ST_NAME", free.coords = TRUE, drop.units = TRUE)
```



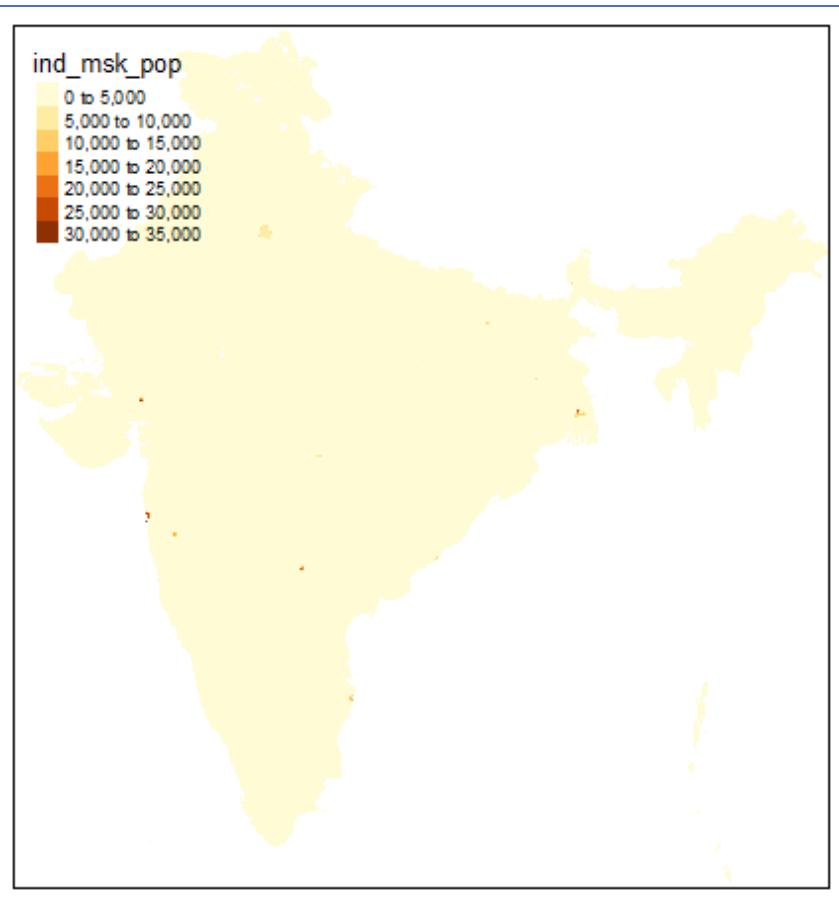
A couple of basic plots show the package intuitive syntax and attractive default parameters.

```
qtm(shp = ind, fill = "ST_NAME", fill.palette = "-Blues")
# not shown
```



Previously, when we plotted the population mask, we use `tm_shape()`. Compare the plot from `tm_shape()` with this one from `tmap()`.

```
qtm(shp = ind_msk_pop, fill = c("F_CODE_DES", "FID_rail_d"),
  fill.palette = "Blues", nrow = 1)
```



## Making Maps with ggmap

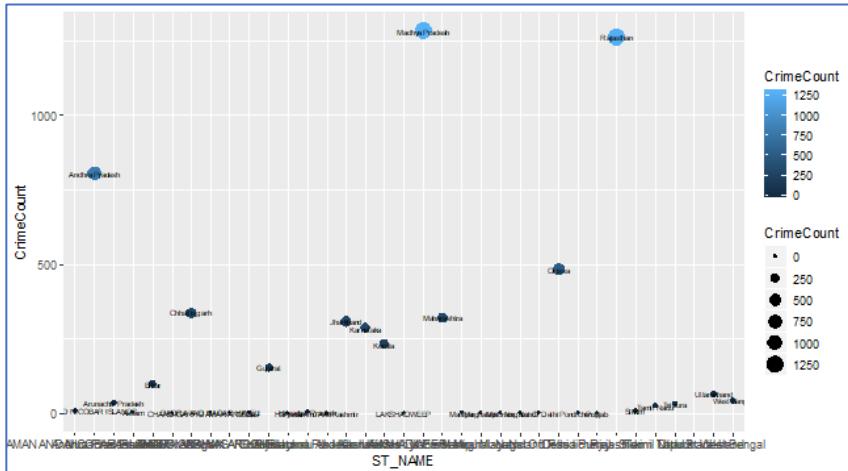
ggmap is based on the ggplot2 package, an implementation of the Grammar of Graphics (Wilkinson 2005). ggplot2 can replace the base graphics in R (the functions you have been plotting with so far). It contains default options that match good visualisation practice and is well-documented: <http://docs.ggplot2.org/current/>.

```
p <- ggplot(ind@data, aes(ST_NAME, CrimeCount))
```

### Add Layers to a Plot.

The real power of ggplot2 lies in its ability to add layers to a plot. In this case we can add text to the plot.

```
p + geom_point(aes(colour = CrimeCount, size = CrimeCount)
) +
  geom_text(size = 2, aes(label = ST_NAME))
```



### *Comments about Layers.*

This idea of layers (or geoms) is quite different from the standard plot functions in R. You will find that each of the functions does a lot of clever stuff to make plotting much easier (see the documentation for a full list).

## Summary

This has been a concise overview of spatial mapping in R. We have only touched the tip of the iceberg and I am sure you already see the applications.

## Exercises

The data used for this exercises can be downloaded entering this url in your browser:

[https://github.com/stricje1/VIT\\_University/tree/master/Predictive\\_Modeling/data/map\\_data.zip](https://github.com/stricje1/VIT_University/tree/master/Predictive_Modeling/data/map_data.zip), the “Download ZIP” button and once downloaded, unzip this to a new folder on your computer.

The code for this exercise can be downloaded from GitHub (just to check the code you write 😊) in the file at:

[https://github.com/stricje1/VIT\\_University/blob/master/Predictive\\_Modeling/code/spatial\\_mapping.r](https://github.com/stricje1/VIT_University/blob/master/Predictive_Modeling/code/spatial_mapping.r).

The code for the animation part (at the end) can be downloaded from GitHub (just to check the code you write) in the file at:

[https://github.com/stricje1/VIT\\_University/blob/master/Predictive\\_Modeling/code/faceted\\_plot\\_animation.R](https://github.com/stricje1/VIT_University/blob/master/Predictive_Modeling/code/faceted_plot_animation.R).

The first file we are going to load into R Studio is the “london\_sport” shapefile located in the ‘data’ folder of the project. It is worth looking at this input dataset in your file browser before opening it in R. You will notice that there are several files named “london\_sport”, all with different file extensions. This is because a shapefile is actually made up of a number of different files, such as .prj, .dbf and .shp.

You could also try opening the file “london\_sport.shp” file in a conventional GIS such as QGIS to see what a shapefile contains.

You should also open “london\_sport.dbf” in a spreadsheet program such as LibreOffice Calc. to see what this file contains. Once you think you understand the input data, it’s time to open it in R. There are several ways to do this, the most commonly used and versatile of which is `readOGR`. This function, from the `rgdal` package, automatically extracts the information regarding the data.

`rgdal` is R’s interface to the “Geospatial Abstraction Library (GDAL)” which is used by other open source GIS packages such as QGIS and enables R to handle a broader range of spatial data formats. If you’ve not already *installed* and loaded the `rgdal` package (see the ‘prerequisites and packages’ section) do so now:

```
> library(rgdal)  
> mydir <- "C:/Users/jeff/Documents/"
```

```
Crime Analysis/Creating-maps-in-R-master/data"
> lnd <- readOGR(dsn = mydir, layer = "london_sport")
```

In the second line of code above the `readOGR` function is used to load a shapefile and assign it to a new spatial object called `lnd`, short for London.

`readOGR` is a *function* of the `rgdal` package, the first *argument* of which `dsn`: “data source name”, the file or directory of the geographic data to be loaded. Thanks to recent developments in `rgdal`, the layer no longer has to be specified (as it is in the 3 line of code which has been *commented out*).

`lnd` is now an object representing the population of London Boroughs in 2001 and the percentage of the population participating in sporting activities according to the Active People Survey. The boundary data is from the Ordnance Survey.

For information about how to load different types of spatial data, see the help documentation for `readOGR`. This can be accessed by typing `?readOGR`. For another worked example, in which a GPS trace is loaded, please see Cheshire and Lovelace (2014).

1. Now that the data is loaded, complete the following tasks:
  - a. Inspect the structure of the spatial data. What is the coordinate reference system (CRS)?
  - b. Plot the spatial data
  - c. Simple plot of London with areas of high sports participation highlighted in blue
  - d. Now create zones in London whose centroid lie within 10 km of the geographic centroid of the City of London. Select all zones whose geographic centroid lies within 10 km of the geographic centroid of inner London.
    - Find London's geographic centroid [Hint: use `gCentroid`]
    - Set 10 km buffer [Hint: use `gBuffer`]
    - Create spatial points [Hint: use `SpatialPoints`]
    - Select points inside buffer
    - Show where the points are located [Hint: use `points`]
    - Select zones intersecting
    - Plot the resulting map

- e. Select the four quadrants of the London map and plot them using a different color for each zone.
2. Attribute joins are used to link additional pieces of information to our polygons. In the `lnd` object, for example, we have 4 attribute variables — that can be found by typing `names(lnd)`. But what happens when we want to add more variables from an external source? We will use the example of recorded crimes by London boroughs to demonstrate this.

The non-spatial data we are going to join to the `lnd` object contains records of crimes in London. This is stored in a comma separated values (.csv) file called “mps- recordedcrime- borough”. If you open the file in a separate spreadsheet application first, we can see each row represents a single reported crime. We are going to use a function called `aggregate()` to aggregate the crimes at the borough level, ready to join to our spatial `lnd` dataset. A new object called `crime_data` is created to store this data.

Now that we have crime data at the borough level, the challenge is to join it to the `lnd` object. We will base our join on the `Borough` variable from the `crime_ag` object and the `name` variable from the `lnd` object. (see `?left_join`).

- a. Identify the number of crimes taking place in borough `NULL`, less than 4,000.
- b. Create a map plot showing the number of thefts per borough

# CHAPTER 9 – Using R for Crime Analysis

“Sherlock Holmes was a man, however, who, when he had an unsolved problem upon his mind, would go for days, and even for a week, without rest, turning it over, rearranging his facts, looking at it from every point of view until he had either fathomed it or convinced himself that his data were insufficient.”

-Dr. John Watson, The Man with the Twisted Lip

## Introduction

Sherlock’s data may have been insufficient, but at present we cannot make the same complaint. 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 criminal activity based on data, (2) analyzing factors that may affect unlawful behavior based upon the data. Taken together, we are really using R programming to cluster and describe data that the police departments and/or city halls have already collected and turned over to the skillful hands of the analyst.

## Author’s Note

I rendered this chapter from R-Studio using HTML Markdown instead of Word, in order to show you some different output option. Obviously, you do not get the “ah-ha” moment that a dynamic webpage would give you (in this static book) but it does highlight some of the feature that you can use your imagination to put into play. Ecalled that I emphasized in the Preface, that in data scoience not only are we concerned with make accurate conclusions regarding our studies, we are also concerned about presenting those results.

## 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")
```

## The Crime Data

### About the Data

The crime data was simulated using distributions derived from an analysis of major crimes in Atlanta, Georgia USA, San Francisco, California USA, and Boston, Massachusetts USA. For instance, Larceny/theft was approximately uniformly distributed  $\text{Uniform} \sim [120000, 160000]$  with mean 140000 and standard deviation of approximately 11547.

The distributions were then applied to Chennai India using the inverse transform method. To simulate larceny/theft in Chennai, we used the reverse inverse method, which for the Uniform distribution is  $X = a + (b - a)U$ , where  $U$  is the  $\text{Uniform}[0,1]$  random number. In Excel, this is the `RAND()` function, so the inverse transform would be  $X = a + (b - a) * \text{RAND}()$ . The crimes are then geographically distributed in zones in a similar fashion.

The data was simulated due to the constraint that crime data was only available at the state-level in India. The data include the following factors. The resulting set is `atlanta_crime_4yr.xls`, with 515807 records and thirteen variables

- IncidntNum (T) Incident number
- Category (T) Crime category, i.e., larceny/theft
- Descript (T)
- DayOfWeek (T)
- Date (D Date: DD/MM/YYYY)
- Time (T) Time: 24-hour system
- PdDistrict (T) Police district where incident occurred
- Resolution (T) Resolution of the crime
- Address (T) Address of the crime
- X (N) Longitude
- Y (N) Latitude
- Location (T) Lat/long
- PdId (N) Police Department ID

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

## **Read the data**

Load the data using `readr` and `read_csv()`.

```
library(readr)
wwwpath <- "https://github.com/stricje1/VIT_University/blob/master/Crime_Analysis_Mapping/data/atlanta_crime_4yr.zip"
path <- "c:\\\\Users\\\\jeff\\\\Documents\\\\VIT_University\\\\data\\\\atlanta_crime_4yr.csv"
df <- read_csv(path)
## Parsed with column specification:
## cols(
##   IncidntNum = col_double(),
##   Category = col_character(),
##   Descript = col_character(),
##   DayOfWeek = col_character(),
##   Date = col_character(),
##   Time = col_time(format = ""),
##   PdDistrict = col_character(),
##   Resolution = col_character(),
##   Address = col_character(),
##   X = col_double(),
##   Y = col_double(),
##   Location = col_character(),
##   PdId = col_double()
## )
```

## **Display Data**

Now, we display the data using `DT` and `datatable()`. The table below shows a partial view of the Chennai crime data

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

Note that the HTML table would scroll left and right and you would interactively be able to select the number of rows to display.

Show <input type="text" value="5"/> entries									
	IncidentNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution	Address
1	150098210	ROBBERY	ROBBERY, BODILY FORCE	Sunday	2/1/2015	15:45:00	Zone4	NONE	None 80.26669397 13.09199072
2	150098210	AGG_ASSAULT	AGGRAVATED ASSAULT WITH BODILY FORCE	Sunday	2/1/2015	15:45:00	Zone4	NONE	None 80.26669397 13.09199072
3	150098260	LARCENY/THEFT	PETTY THEFT SHOPLIFTING	Saturday	1/31/2015	17:00:00	Zone3	ARREST	None 80.26380468 13.09405785
4	150098345	LARCENY/THEFT	PETTY THEFT SHOPLIFTING	Sunday	2/1/2015	14:00:00	Zone5	ARREST	None 80.26774581 13.0768748
5	150098367	ROBBERY	ROBBERY, ARMED WITH A KNIFE	Sunday	2/1/2015	16:20:00	Zone5	ARREST	None 80.24890198 13.07291073

Showing 1 to 5 of 100 entries

Previous  2 3 4 5 ... 20 Next

Here we apply `sprintf`, a wrapper for the C function `sprintf`, that returns a character vector containing a formatted combination of text and variable values.

```
 sprintf("Number of Rows in Dataframe: %s", format(nrow(df),  
 ,big.mark = ","))  
## [1] "Number of Rows in Dataframe: 515,807"  
str(df_sub)  
## Classes 'tbl_df', 'tbl' and 'data.frame': 515807 obs. of 24  
variables:  
## $ IncidntNum: int 150098210 150098210 150098210 150098226  
150098248 150098248 150098254 150098260 150098345 150098345 ...  
## $ Category : chr "ROBBERY" "ASSAULT" "SECONDARY CODES"  
"VANDALISM" ...  
## $ Descript : chr "ROBBERY, BODILY FORCE" "AGGRAVATED ASSAULT  
WITH BODILY FORCE" "DOMESTIC VIOLENCE" "MALICIOUS MISCHIEF,  
VANDALISM OF VEHICLES" ...  
## $ DayOfWeek : chr "Sunday" "Sunday" "Sunday" "Tuesday" ...  
## $ Date : chr "2/1/2015" "2/1/2015" "2/1/2015" "1/27/2015"  
...  
## $ Time : 'hms' num 15:45:00 15:45:00 15:45:00 15:45:00 19:00:00  
...  
## ... attr(*, "units")= chr "secs"  
## $ PdDistrict: chr "ZONE10" "ZONE10" "ZONE10" "ZONE05" ...  
## $ Resolution: chr "NONE" "NONE" "NONE" "NONE" ...  
## $ Address : chr "300 Block of LEAVENWORTH ST" "300 Block of  
LEAVENWORTH ST" "300 Block of LEAVENWORTH ST" "LOMBARD ST / LAGUNA  
ST" ...  
## $ X : num -84.4 -84.4 -84.4 -84.4 -84.3 ...  
## $ Y : num 33.8 33.8 33.8 33.8 33.7 ...  
## $ Location : chr "(33.80414172,-84.35398903)" "(33.80414172,-  
84.35398903)" "(33.80414172,-84.35398903)" "(33.8204197,-  
84.37070154)" ...  
## $ PdId : num 1.5e+13 1.5e+13 1.5e+13 1.5e+13 1.5e+13 ...
```

## Preprocess Data

The All-Caps text is difficult to read. Let's force the text in the appropriate columns into proper case.

Note: For the following code chunk, output not shown.

```
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'))

```

## Visualize Data

### 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 addCondtroll (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 a large number of markers on a map as in our case with crimes, we can cluster them together. Now, we define crime incident locations on the map using leaflet, which we use for our popups.

```

library(leaflet)
data <- df[1:10000,] # display the first 10,000 rows
data$popup <- paste("<b>Incident #: </b>", data$IncindntNum,
                    "<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)

```

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 here:

```

data$popup[1]
## [1] “<b>Incident #: </b> 150098210 <br> <b>Category: </b>
ROBBERY <br> <b>Description: </b> ROBBERY, BODILY FORCE
<br> <b>Day of week: </b> Sunday <br> <b>Date: </b> 2/1/20
15 <br> <b>Time: </b> 15:45:00 <br> <b>PD district: </b> Z
ONE10 <br> <b>Resolution: </b> NONE <br> <b>Address: </b>
300 Block of LEAVENWORTH ST <br> <b>Longitude: </b> -84.35
398903 <br> <b>Latitude: </b> 33.80414172”

```

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.

```

leaflet(data, width = “100%”) %>% addTiles() %>%
  addTiles(group = “OSM (default)”) %>%
  addProviderTiles(provider = “Esri.WorldStreetMap”,
    group = “World StreetMap”) %>%
  addProviderTiles(provider = “Esri.WorldImagery”,
    group = “World Imagery”) %>%
#  addProviderTiles(provider = “NASAGIBS.ViirsEarthAtNight
2012”, group = “Nighttime Imagery”) %>%
  addMarkers(lng = ~X, lat = ~Y, popup = data$popup,
    clusterOptions = markerClusterOptions()) %>%
  addLayersControl(
    baseGroups = c(“OSM (default)”, “World StreetMap”,
      “World Imagery”), 
    options = layersControlOptions(collapsed = FALSE)
  )

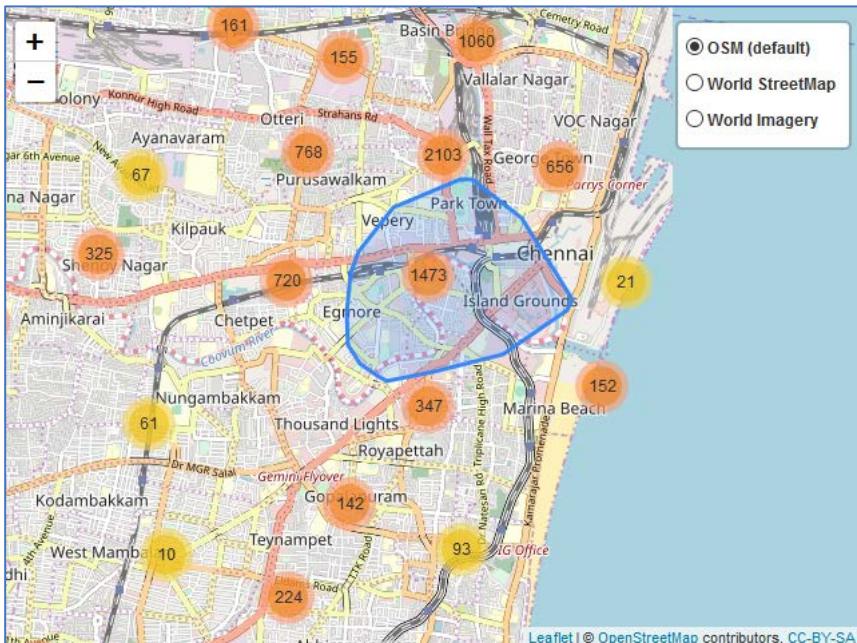
```

## Crime Over Time

That was not meant to rhyme, but I like it. In this section, we will manipulate the data using the `dplyr::mutate` function. `mutate` adds new variables while preserving existing variables. Below, we used “shades of blue” in the code for our plot, with a dark blue line that smooths the data.

```
library(dplyr)
```

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

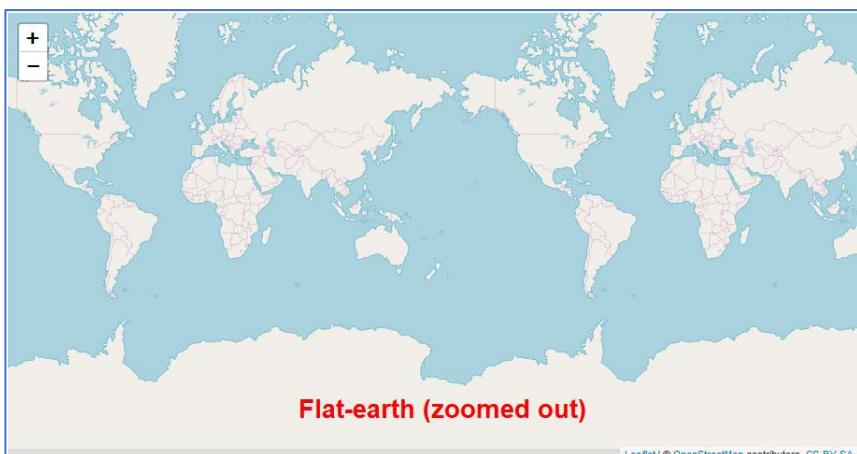
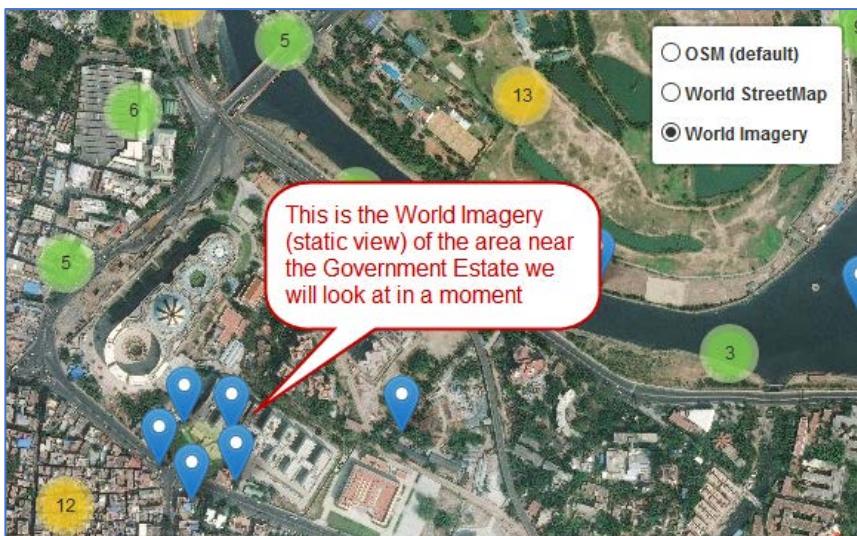


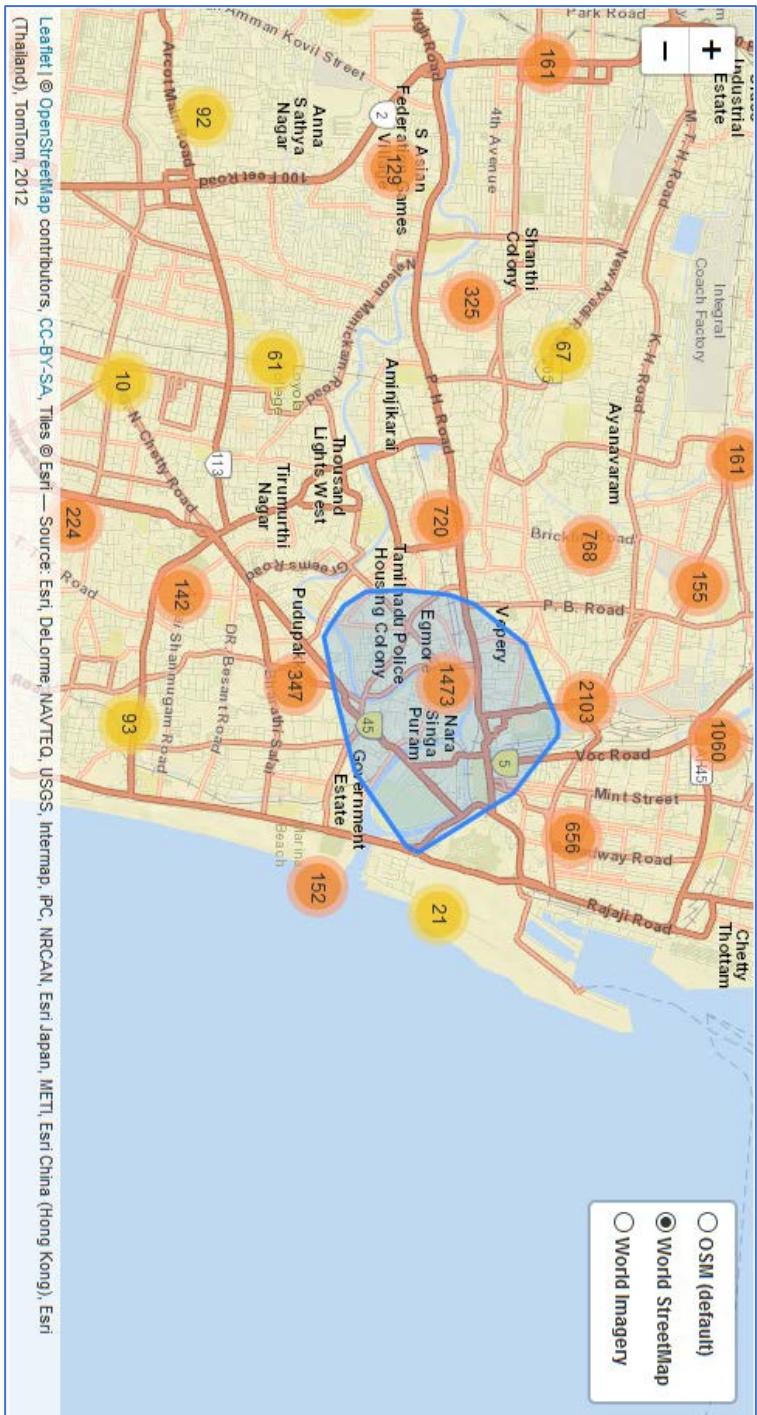
The figure above shows Chennai, India, Crime Map by Police Department (PD) Zones (simulated)

## Interactive Maps

The leaflet app maps are interactive. We can zoom out to a world view (flat-earth) or zoom in on areas that appear when we hover over the map. If we zoom in on the area near the Government Estate (next page)

where there are 1473 crimes, and then further zoomin on the 5 crimes in the vicinity of the Government Estate, we will see five markers. I selected one to view the popup. Notice that it is an Assault and is described as "Threat or force to resist executive officer." It occurred on Febraury 17, 2015, at 3:33 PM (11.5 hours ahead of my time in Colorado), and resulted in an arrest. Note that thre are also three map views OSM (the default), a layering supported, vector data map with prerendered raster map tiles from OpenStreetMap, a World StreetMap view, and a World Imagery view (flat-earth).





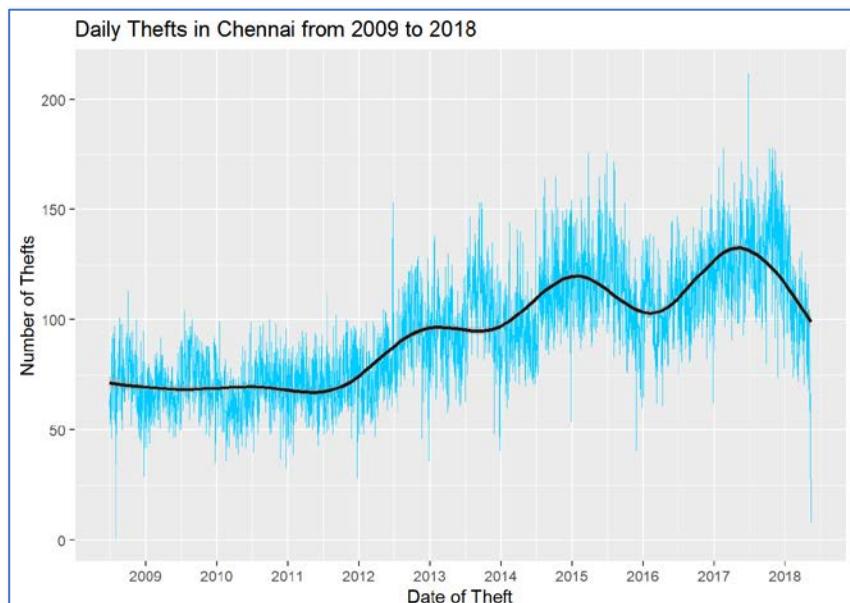


## Crime Series – Daily Crimes Plot with Variance

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 crimes may increase at night and fall off during the day. Another scenario might involve increased crime rate during certain events, like parades, rodeos, fairs, and so on. The 1996 Summer Olympics (not in the dataset) brought an increase in crime to Atlanta, including the Centennial Olympic Park bombing on July 27, attributed to domestic terrorism.

```
library(ggplot2)
library(scales)

plot <- ggplot(df_crime_daily, aes(x = Date, y = count)) +
  geom_line(color = "#50a8ff", size = 0.1) +
  geom_smooth(color = "#00008b") +
  # fte_theme() +
  scale_x_date(breaks = date_breaks("1 year"), labels = date_format("%Y")) +
  labs(x = "Date of Crime", y = "Number of Crimes", title =
  "Daily Crimes in Atlanta from 2014 – 2018")
plot
## `geom_smooth()` using method = 'gam' and formula 'y ~ s
(x, bs = "cs")'
```



The figure above is the *crime series* of number of crimes by date of crime with smoothing

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

**Markdown Note.** This chapter introduces an additional markdown feature, tables using the DT-package (sata tables). The function, datatable, creates an HTML widget to display rectangular data (a matrix or data frame) using the JavaScript library DataTables. See <https://rstudio.github.io/DT> for the full documentation.

## Aggregate 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)
datatable(df_category)
```

**Markdown Note.** For HTML output, tables can be given more complex formatting. For instance, we can alternate row color, emphasize headings, insert scroll bars for long tables, and so on. The actual HTML table self-adjusts size when you reduce your browser size. Size Word doesn't, I took a screenshot.

This table shows the Crime category frequencies (10,000 or more over 10 years):

Show 10 entries

Search:

	Category	Frequency	Percentage
1	Larceny/Theft	338418	0.677696674776967
2	Assault	97483	0.195213921680534
3	Robbery	35481	0.071052236340152
4	Agg_ASSAULT	23040	0.0461385960169415
5	Manslaughter	3711	0.00743143792616623
6	Murder	1232	0.00246713325923923

Showing 1 to 6 of 6 entries

Previous 1 Next

The table shows the frequencies (counts) of the reported crimes with 10,000 or more occurrences, indicating that the most frequent crime is Larceny/Theft (non-vehicular and from the vehicle). With all the hype regarding the dangers of Chennai, there were only 886 murders in the 10-year period, or about 88 per year, and only 121 reported rapes for about 12 per year or 1 per month.

### Create a Bar Chart

Now that we can aggregate the data, we will show the data with a bar graph. The bar graph (or histogram) shows the frequencies of the crimes recorded in Table 9-3. It makes it easy to see the vast difference between Larceny/Theft and the other reported crimes.

```
library(ggplot2)
library(ggrepel)
bp<-ggplot(df_category, aes(x=Category, y=Frequency,
```

```

fill=Category)) + geom_bar(stat="identity") +
theme(axis.text.x=element_blank()) +
geom_text_repel(data=df_category, aes(Label=Category))

```

### Create a pie chart based on the incident category.

To further illustrate the crime incident data, a subsequent pie chart is plotted. The chart illustrates the same data as does the bar chart, but it may be more understandable in this instance. It shows that Larceny/Theft occurs more than twice as much as all other reported crimes taken together. The chart is also ascetically pleasing.

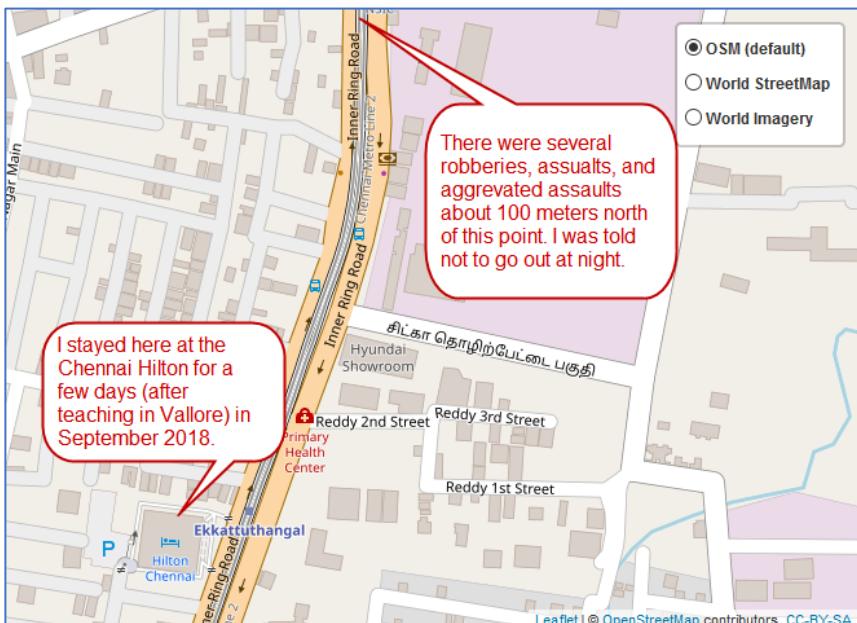
```

bp<-ggplot(df_category, aes(x="", y=Percentage, fill=Category)) + geom_bar(stat="identity")
pie <- bp + coord_polar("y")
pie

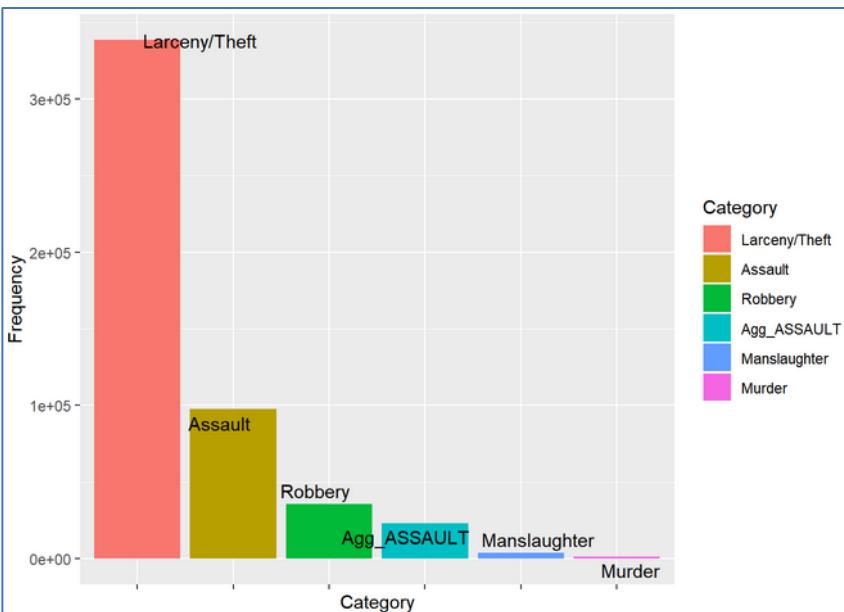
```

### Author's Note

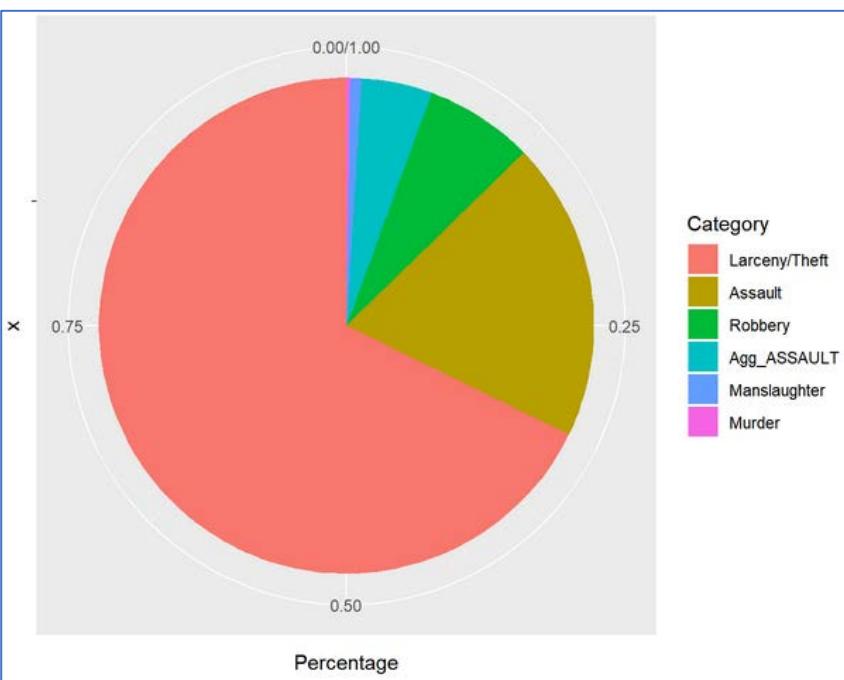
I was last in Chennai in September 2018. It is the port nearest Vellore, where I occasionally teach data science.



Here is the bar chart of crimes in Chennai by category:



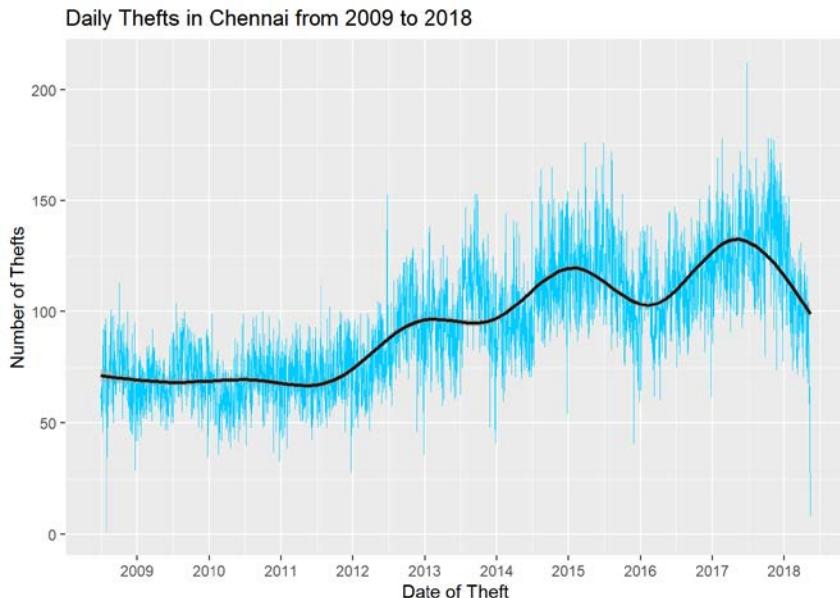
Here is the pie chart of crimes in Chennai by category



## Theft Over Time

In this section, we create a chart of crimes (Larceny/Theft) over time. And for aesthetic effect as well as clarity, we make use color in depicting the series shown below. This will provide us with a crime series for thefts, which we will smooth as we did before.

```
df_theft <- df %>% filter(grepl("Larceny/Theft", Category))  
df_theft_daily <- df_theft %>%  
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%  
  group_by(Date) %>%  
  summarize(count = n()) %>%  
  arrange(Date)  
plot <- ggplot(df_theft_daily, aes(x = Date, y = count)) +  
  geom_line(color = "#00cc00", size = 0.1) +  
  geom_smooth(color = "#008000") +  
  # fte_theme() +  
  scale_x_date(breaks = date_breaks("1 year"), labels = date_format("%Y")) +  
  labs(x = "Date of Theft", y = "Number of Thefts", title = "Daily Thefts in Chennai from 2009 - 2018")  
plot
```



## Theft Time Heatmap

Now, we aggregate counts of thefts 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 atlanta\_crime\_10yr.csv, so that we can use an approximate arrest time with day of the week. But R does not have one, or one I can find. So, we build the function below, using the colon delimiter to separate hours from minutes. Then we build 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_theft_time <- df_theft %>%  
  mutate(Hour = sapply(DayOfWeek, get_hour)) %>%  
  group_by(DayOfWeek) %>%  
  summarize(count = n())  
# df_theft_time %>% head(10)  
datatable(df_theft_time)
```

	DayOfWeek	Hour	count
1	Friday	0	1932
2	Friday	1	1126
3	Friday	2	711
4	Friday	3	488
5	Friday	4	316
6	Friday	5	317
7	Friday	6	524
8	Friday	7	835
9	Friday	8	1331
10	Friday	9	1637

## **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.

```
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_theft_time$DayOfWeek <- factor(df_theft_time$DayOfWeek,
                                     level = rev(dow_format))
df_theft_time$Hour <- factor(df_theft_time$Hour,
                               level = 0:23, label = hour_format)
# df_theft_time %>% head(10)
datatable(df_theft_time)
```

The theft frequency table by day-of-week and time of day appears on the netxt page.

## **Create Time Heatmap**

Using our previous results, we build a “heatmap” and plot it with a green color scheme 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.

Note that are several steps we need to take before plotting the heatmap that follows on page 421.

	DayOfWeek	Hour	count
1	Friday	12 AM	1932
2	Friday	1 AM	1126
3	Friday	2 AM	711
4	Friday	3 AM	488
5	Friday	4 AM	316
6	Friday	5 AM	317
7	Friday	6 AM	524
8	Friday	7 AM	835
9	Friday	8 AM	1331
10	Friday	9 AM	1637

The graph brings up a question: why is there a surge at 6-7 PM on weekdays? Note that Saturday and Sunday are in the middle running and the time axis is in 24-hour time. Law enforcement crime experts would probably be able to explain the “heat,” but without seeing the information provided by the data, they may not realize that 6-7 PM on weekdays is an issue.

### ***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.

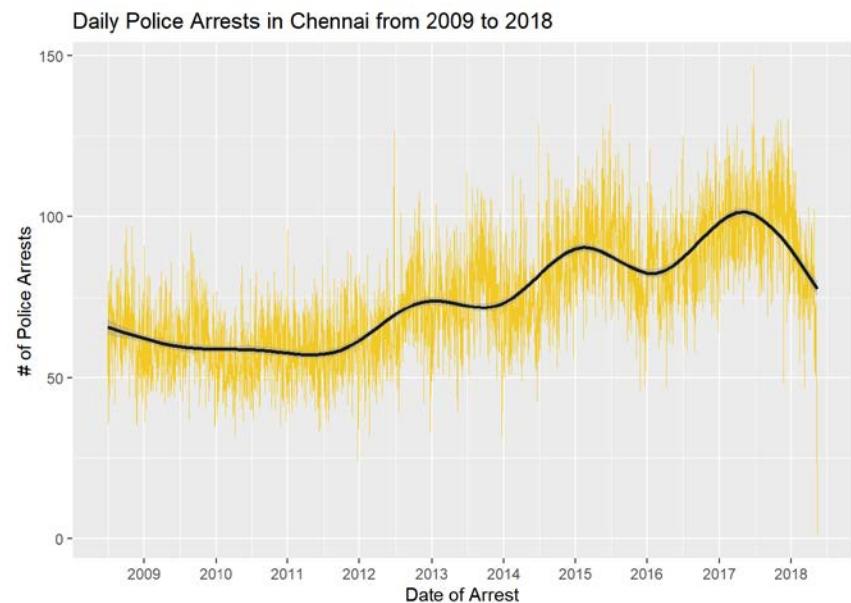
```
df_arrest <- df %>% filter(grepl("Arrest", Resolution))
df_arrest_daily <- df_arrest %>%
  mutate(Date = as.Date(Date, "%m/%d/%Y")) %>%
  group_by(Date) %>%
  summarize(count = n()) %>%
  arrange(Date)
```

## Daily Arrests

Next, we build the plot of daily theft arrests by police or theft arrests over time. This will provide us with another crime series, and we will smooth it as usual, noticing a downward trend in arrests over time until a sharp increase starting around 2017.

```
library(ggplot2)
library(scales)
plot <- ggplot(df_arrest_daily, aes(x =Date, y =count)) +
  geom_line(color = "#F2CA27", size = 0.1) +
  geom_smooth(color = "#1A1A1A") +
  # fte_theme() +
  scale_x_date(breaks = date_breaks("1 year"), labels =
    date_format("%Y")) +
  labs(x = "Date of Arrest", y = "# of Police Arrests",
       title ="Daily Police Arrests in Chennai from 2009 to
2018")
plot

## `geom_smooth()` using method = 'gam' and formula 'y ~ s
(x, bs = "cs")'
```



```

plot <- ggplot(df_theft_time, aes(x = Hour,
  y = DayOfWeek, fill = count)) +
  geom_tile() +
  # fte_theme() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.
6),
    legend.title = element_blank(),
    legend.position="top",
    legend.direction="horizontal",
    legend.key.width=unit(2, "cm"),
    legend.key.height=unit(0.25, "cm"),
    legend.margin=unit(-0.5,"cm"),
    panel.margin=element_blank()) +
  labs(x = "Hour of Theft (Local Time)", y = "Day of Week
of Theft",
    title = "Number of Thefts in Chennai from 2009 - 20
18, by Time of Theft") +
  scale_fill_gradient(low = "white", high = "#008000", la
bels = comma)
plot

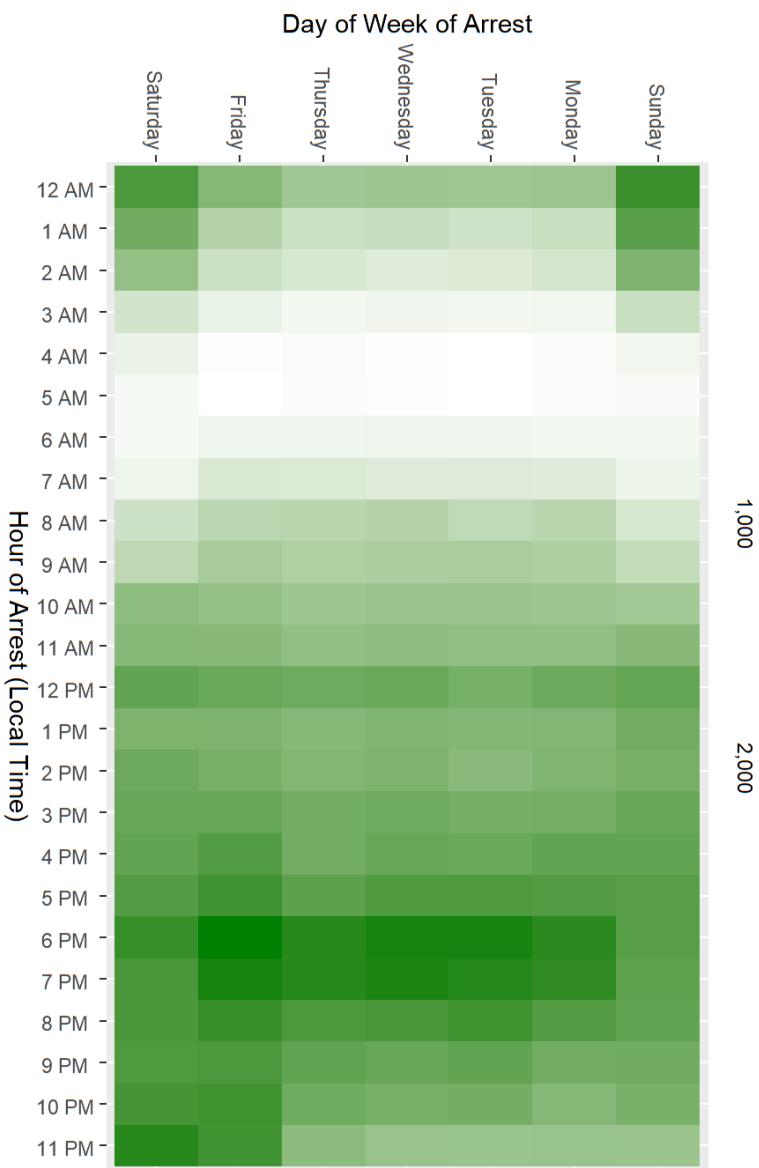
```

Note that most of the code is for the legend and its formatting. The line of code that gives the heatmap its “heat” is the `scale_fill_gradient()` function with its low and high intensity fill colors.

The graph brings up a question: why is there a surge at 6-7 PM on weekdays? Note that Saturday and Sunday are in the middle running and the time axis is in 24-hour time. Law enforcement crime experts would probably be able to explain the “heat,” but without seeing the information provided by the data, they may not realize that 6-7 PM on weekdays is an issue.

Now we show the heatmap of theft by day of week and hour of theft (local time):

## Number of Police Arrests in Chennai from 2009 - 2018, by Time of Arrest



## Correlation Analysis

### Factor by Crime Category

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

```
df_top_crimes <- df_arrest %>%
  group_by(Category) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
datatable(df_top_crimes, , options = list(pageLength = 10,
scrollX='400px'))
```

Top Crimes Table

Category	count
Larceny/Theft	171594
Assault	59451
Robbery	20226
Agg_ASSAULT	14810
Manslaughter	2286
Murder	757

```
df_arrest_time_crime <- df_arrest %>%
  filter(Category %in% df_top_crimes$Category[2:19]) %>%
  mutate(Hour = sapply(DayOfWeek, get_hour)) %>%
  group_by(Category, DayOfWeek) %>%
  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)
datatable(df_arrest_time_crime, options = list(pageLength = 10, scrollX='400px'))
```

Arrest frequency table by day-of-week and hour (time)

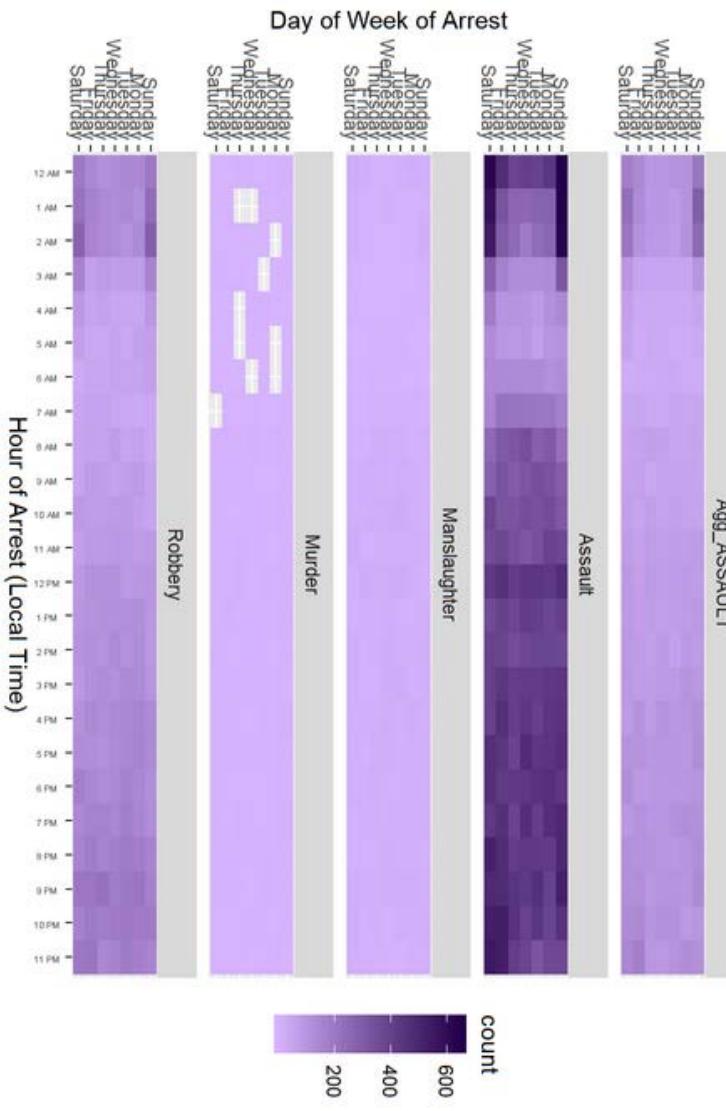
Category	DayOfWeek	Hour	count
Agg_ASSAULT	Friday	12 AM	143
Agg_ASSAULT	Friday	1 AM	135
Agg_ASSAULT	Friday	2 AM	110
Agg_ASSAULT	Friday	3 AM	52
Agg_ASSAULT	Friday	4 AM	35
Agg_ASSAULT	Friday	5 AM	28
...			
Robbery	Wednesday	6 PM	143
Robbery	Wednesday	7 PM	153
Robbery	Wednesday	8 PM	178
Robbery	Wednesday	9 PM	184
Robbery	Wednesday	10 PM	193
Robbery	Wednesday	11 PM	163

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

In this section, we plot the number of arrests by category and time of arrest. This leads us to use a chart type that you may not have seen very often. We take the heat map application from the previous section and plot all the crime heat maps in one aggregated chart as seen in the figure below. For each heat map in the chart, the horizontal axes are “hours of arrest (local time),” and the vertical axes are “days of the week.”

```
plot <- ggplot(df_arrest_time_crime,
  aes(x = Hour, y = DayOfWeek, fill = count)) +
  geom_tile() + 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 San Francisco
from 2014 - 2018, by Category and Time of Arrest") +
  scale_fill_gradient(low = "white", high = "#2980B9") +
  facet_wrap(~ Category, nrow = 6)
plot
```

## Number of Police Arrests in Chennai from 2014 to 2018, by Category and Time



This looks good, but the gradients aren't helpful because they are not normalized. We need to normalize the range on each facet. (unfortunately, this makes the value of the gradient unhelpful)

### Normalized Gradients

```
df_arrest_time_crime <- df_arrest_time_crime %>%  
  group_by(Category) %>%
```

```
    mutate(norm = count/sum(count))
datatable(df_arrest_time_crime, options = list(pageLength = 10,scrollX='400px'))
```

Show 10 entries

Search:

	Category	DayOfWeek	Hour	count	
1	Agg_ASSAULT	Friday	12 AM	143	0.009655638
2	Agg_ASSAULT	Friday	1 AM	135	0.009115462
3	Agg_ASSAULT	Friday	2 AM	110	0.007427413
4	Agg_ASSAULT	Friday	3 AM	52	0.003511141
5	Agg_ASSAULT	Friday	4 AM	35	0.002363268
6	Agg_ASSAULT	Friday	5 AM	28	0.001890614
7	Agg_ASSAULT	Friday	6 AM	33	0.002228224
8	Agg_ASSAULT	Friday	7 AM	49	0.003308575
9	Agg_ASSAULT	Friday	8 AM	58	0.003916272
10	Agg_ASSAULT	Friday	9 AM	64	0.004321404

< >

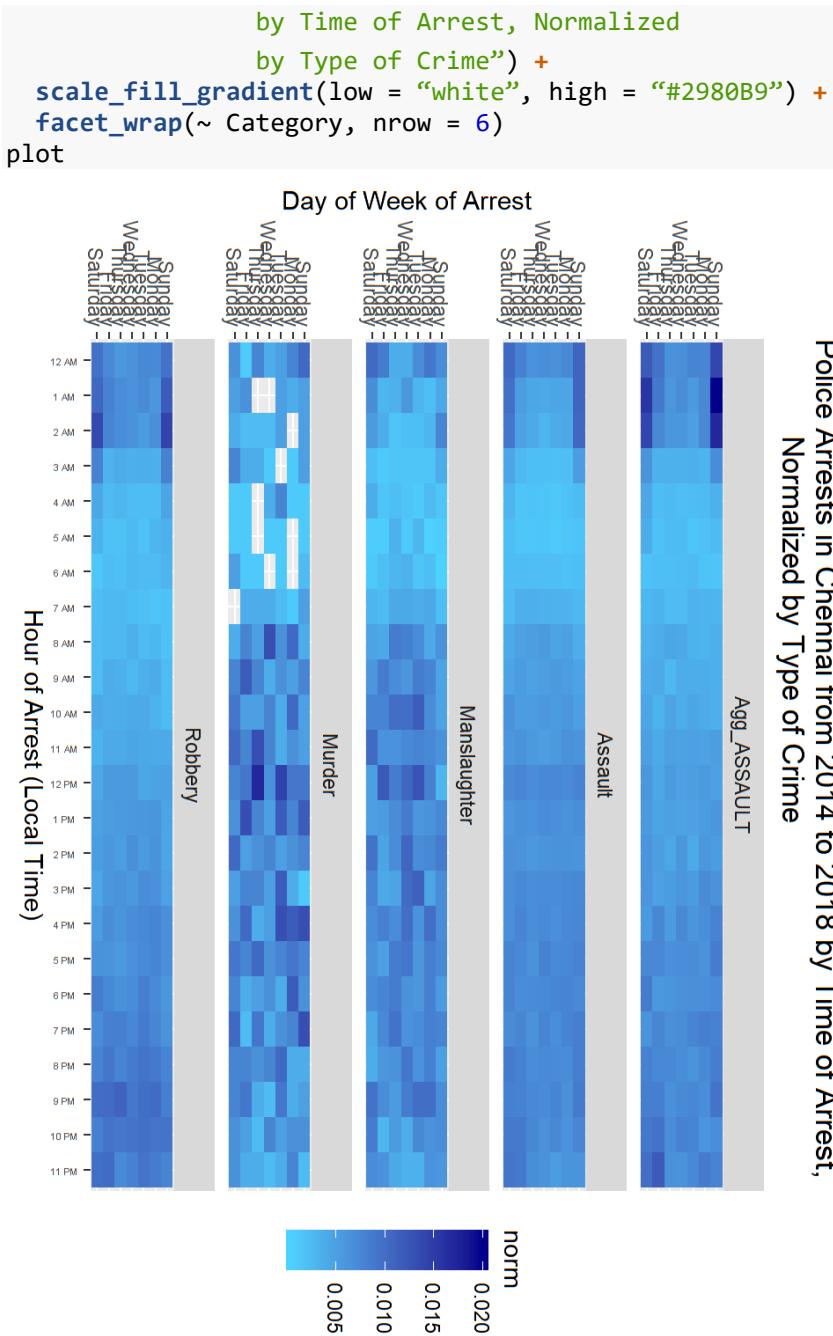
Showing 1 to 10 of 830 entries

Previous 1 2 3 4 5 ... 83 Next

Now, we normalized the number of arrests by the category and the time of arrest:

```
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 Atlanta from 2014- 2018")
```

## Police Arrests in Chennai from 2014 to 2018 by Time of Arrest, Normalized by Type of Crime



## Factor by Police District

In this section, we plot like we did for the preceding heatmap, but with a different scope. In following table and its corresponding plot, 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(DayOfWeek, get_hour)) %>%
  group_by(PdDistrict, DayOfWeek) %>%
  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)
datatable(df_arrest_time_district,
  options = list(pageLength = 10,scrollX='400px'))
```

	PdDistrict	DayOfWeek	Hour	count	
1	Zone1	Friday	12 AM	184	0.008252231
2	Zone1	Friday	1 AM	90	0.0040364174
3	Zone1	Friday	2 AM	78	0.0034982284
4	Zone1	Friday	3 AM	49	0.0021976050
5	Zone1	Friday	4 AM	30	0.0013454724
6	Zone1	Friday	5 AM	40	0.0017939633
7	Zone1	Friday	6 AM	45	0.0020182087
8	Zone1	Friday	7 AM	79	0.0035430775
9	Zone1	Friday	8 AM	83	0.0037224738
10	Zone1	Friday	9 AM	136	0.0060994752

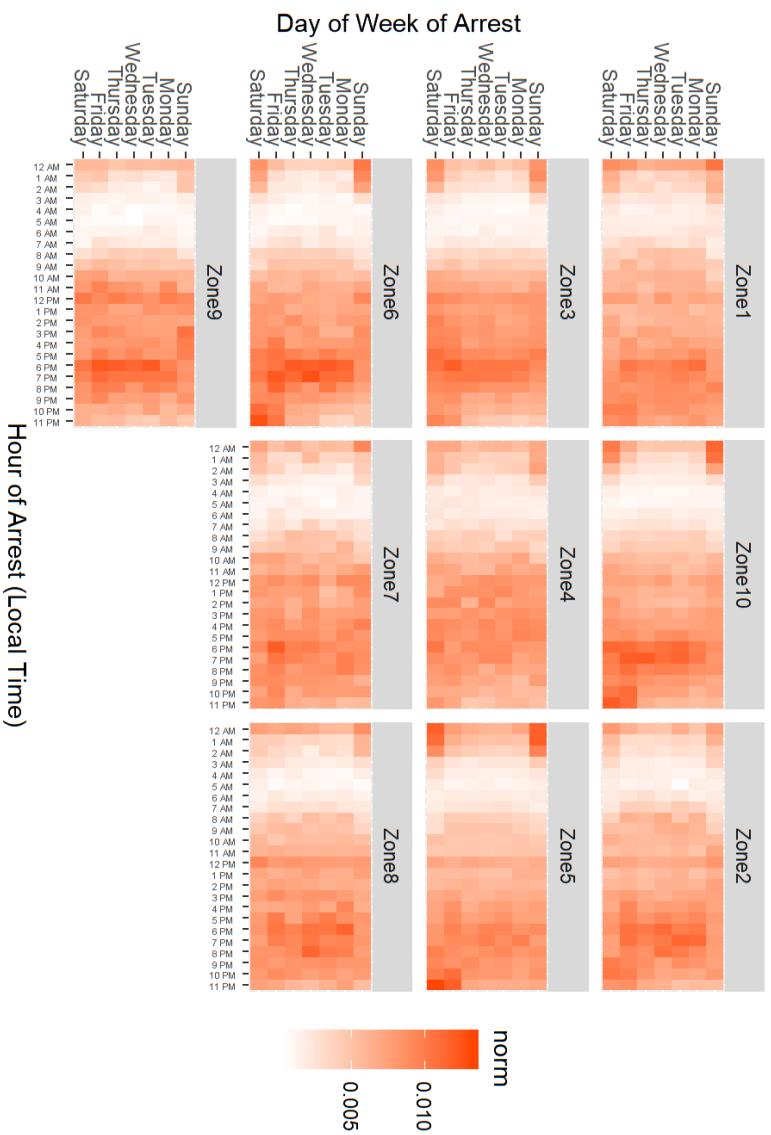
Showing 1 to 10 of 1,680 entries

Previous 1 2 3 4 5 ... 168 Next

### Factor by Police District

```
plot <- ggplot(df_arrest_time_district, aes(x = Hour, y = DayOfWeek, fill = norm)) + 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 = "Police Arrests in Atlanta from 2014 - 2018 by Time of Arrest, Normalized by Station") +
  scale_fill_gradient(low = "white", high = "#8E44AD") +
  facet_wrap(~ PdDistrict, nrow = 5)
plot
```

## Police Arrests in Chennai from 2014 to 2018 by Time of Arrest, Normalized b



### Factor by Month

We now look at factor by month. If crime is tied to activities, the period at which activies end may impact.

```
df_arrest_time_month <- df_arrest %>%
  mutate(Month = format(as.Date(Date, "%m/%d/%Y"), "%B"),
  Hour = sapply(Time, get_hour)) %>%
```

```

group_by(Month, DayOfWeek) %>%
summarize(count = n()) %>%
group_by(Month) %>%
mutate(norm = count/sum(count))

```

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

```

df_arrest_time_month$DayOfWeek <- factor(
df_arrest_time_month$DayOfWeek, level = rev(dow_format))
df_arrest_time_month$Hour <- factor(
  df_arrest_time_month$Hour, level = 0:23,
  label = hour_format)
df_arrest_time_month$Month <- factor(
  df_arrest_time_month$Month,
  level = c("January", "February", "March", "April", "May",
            "June", "July", "August", "September",
            "October", "November", "December"))
# Plot of Factor by Month

```

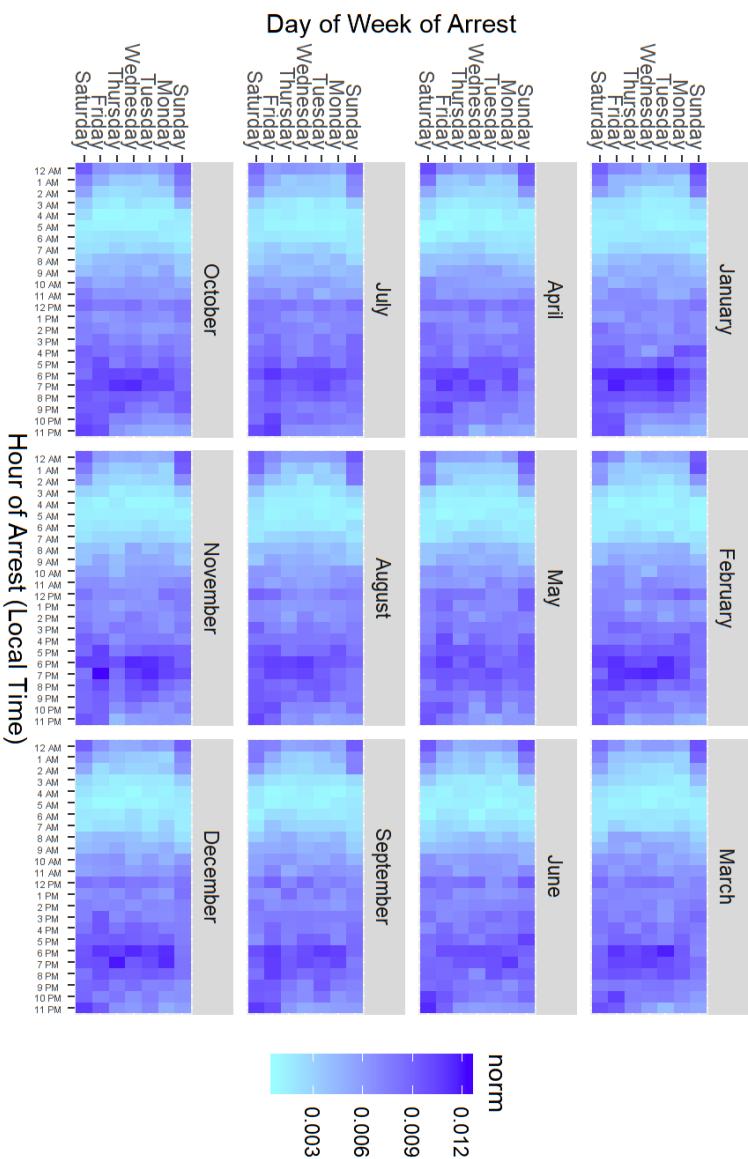
Now we plot the data as shown below and seen in the figure below.

```

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 Chennai from 2009 to
                 2018 by Time of Arrest,
                 Normalized by Month") +
  scale_fill_gradient(low = "#9bfdff",
                      high = "#4401ff") +
  facet_wrap(~ Month, nrow = 4)
plot

```

## Police Arrests in Chennai from 2009 to 2018 by Time of Arrest, Normalized by Month



### Factor by Year

What if things changed overtime?

```
df_arrest_time_year <- df_arrest %>%
  mutate(Year = format(as.Date(Date, "%m/%d/%Y"), "%Y"),
        hour = sapply(Time, get_hour)) %>%
```

```

group_by(Year, DayOfWeek) %>%
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)
## Warning: Unknown or uninitialized column: 'Hour'.

```

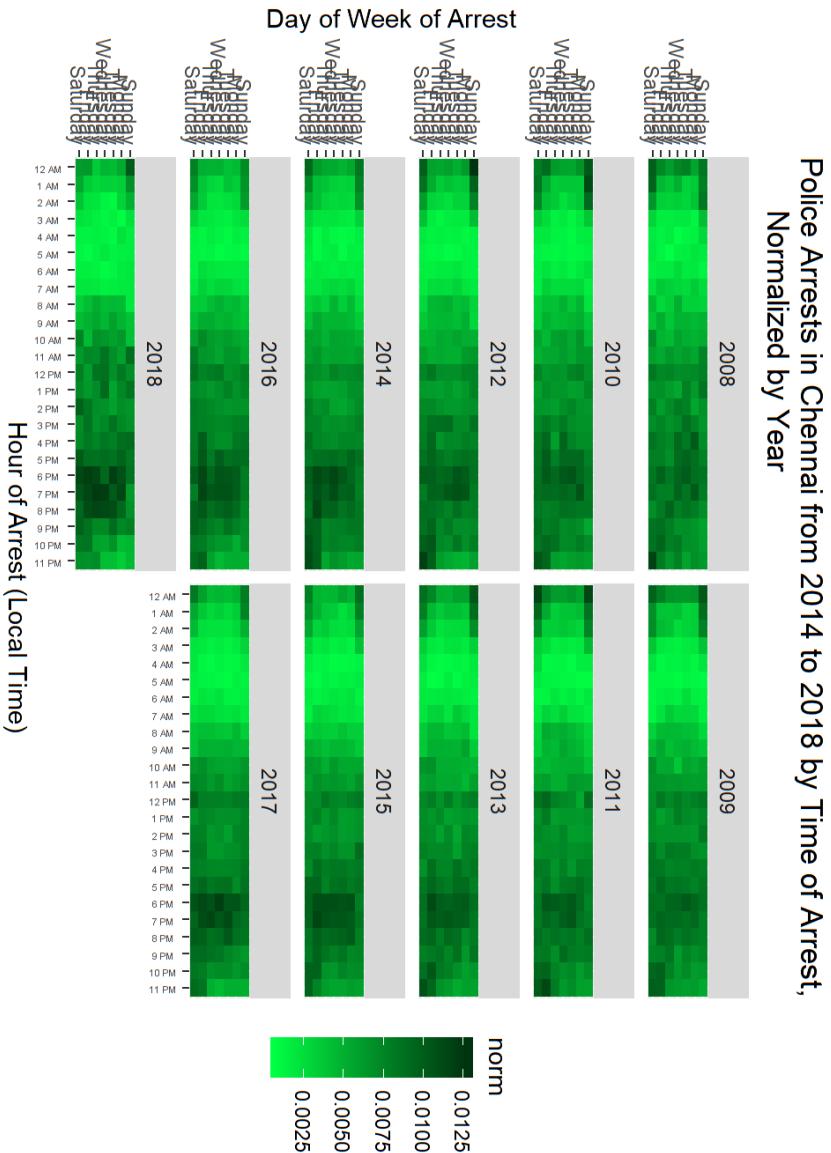
## Police Arrest Normalized by Year

In similar fashion, we can look at the arrests by year aggregated over day-of-week and time-of-day, as seen in the figure.

```

plot <- ggplot(df_arrest_time_year,
aes(x = Hour, y = DayOfWeek, fill = norm)) +
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 = "Police Arrests in Atlanta from 2014 - 2018
by Time of Arrest, Normalized by Year") +
scale_fill_gradient(low = "white", high = "#E67E22") +
facet_wrap(~ Year, nrow = 6)
plot

```



## Exercises

1. The data we use for this exercise can be downloaded from the Atlanta Police Department (PD) website ([www.atlantapd.org](http://www.atlantapd.org)). The data is contained in ten ZIP files, one for each year from 2009 to 2018 (partial).

- IncidntNum (T) Incident number
- Category (T) Crime category, i.e., larceny/theft
- Descript (T)
- DayOfWeek (T)
- Date (D Date: DD/MM/YYYY)
- Time (T) Time: 24-hour system
- PdDistrict (T) Police district where incident occurred
- Resolution (T) Resolution of the crime
- Address (T) Address of the crime
- X (N) Longitude
- Y (N) Latitude
- Location (T) Lat/long
- PdId (N) Police Department ID

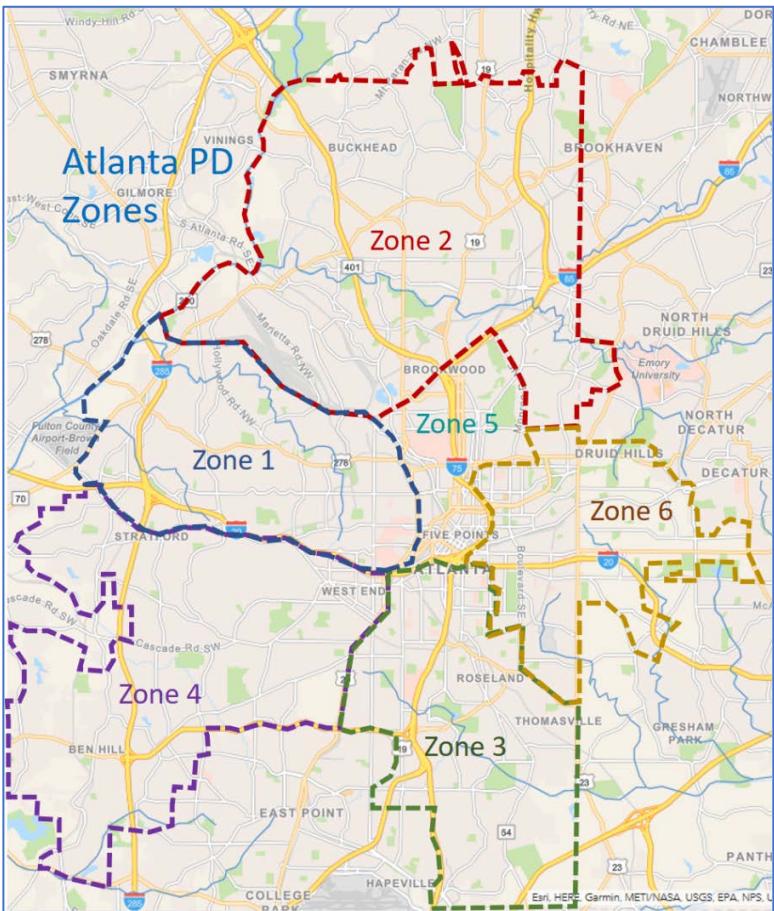
(N = Numeric, T = Time, D = Date, C = Class)

To orient you toward our geographic area, the map below shows Atlanta with the general border defined by the Perimeter (I-285). Zones are generally divided by (1) I-75/85 running north and south to the split just north of the downtown area (I did not collect the data for the Airport zone, defined by the I-75/85 split south of the downtown area); and (2) I-20 divided Atlanta into north and south. The map was generated using ArcGIS Online with the Esri World Navigation Map overlay (layer). I added the layer of for PD zones.

The seven PD zones are:

- Zone 1, encompassing northwest Atlanta
- Zone 2, encompassing north Atlanta
- Zone 3, encompassing southeast Atlanta
- Zone 4, encompassing southwest Atlanta
- Zone 5, encompassing downtown Atlanta

- Zone 6, encompassing east Atlanta
- Homicide, multi-jurisdictional



- a. Read the data into R and display the first 10 rows.
- b. Preprocess the data as may be applicable.
- c. Use leaflet to display the map
- d. Create an interactive popup that contains Category, Description, Day of Week, Date, Time, PD District, Resolution, Address, Longitude and Latitude.
- e. Plot a crime series plot
- f. Aggregate the data and create a frequency distribution plot for the top six crimes
- g. Create a Theft time heatmap
- h. Create a heatmap of theft by day of week and hour of theft

- i. Perform correlation analysis
- j. Create a heatmap for the number of arrests by category and time of arrest
- k. Normalize the gradients and plot the number of arrests by category and time of arrest
- l. Create a plot of arrests by police district
- m. Create a plot of arrests by month
- n. Create a plot of arrests by year

# CHAPTER 10 – Hyperparameter Tuning

## Markdown Note

I generated this chapter exclusively as an R Markdown document in Word format (except for the exercises at the end).

## Search Methods for Hyperparameter Tuning

The objective of this document is to assess the use of various search methods in finding the optimal values of hyperparameters of a machine learning model. The population-based search methods to be tested are genetic algorithms (GA), differential evolution (DE) and particle swarm optimization (PSO). Grid and random search will also be performed and used as benchmarks. XGBoost with generalized linear models as learners will be used to predict the compressive strength of concrete based on its composition. The compressive strength of concrete is determined by its age and composition.

The dataset used comes from the research paper Modeling of strength of high-performance concrete using artificial neural networks by I-Cheng Yeh (1998). The dataset can be downloaded through the UCI Machine learning Repository. The dataset contains 1030 examples and the following features:

Input Variable: Cement (kg in a m<sup>3</sup> mixture) Input Variable: Blast Furnace Slag (kg in a m<sup>3</sup> mixture) Input Variable: Fly Ash (kg in a m<sup>3</sup> mixture) Input Variable: Water (kg in a m<sup>3</sup> mixture) Input Variable: Superplasticizer (kg in a m<sup>3</sup> mixture) Input Variable: Coarse Aggregate (kg in a m<sup>3</sup> mixture) Input Variable: Fine Aggregate (kg in a m<sup>3</sup> mixture) Input Variable: Age (days) Output Variable: Concrete compressive strength (MPa)

## Install and Load Packages

The pacman package is used to install and load all necessary packages.

```
setwd("C:/Users/jeff/R_data")
suppressWarnings(pacman::p_load(caret, DEoptim, doParallel,
dplyr, foreach, GA, gbm, GGally, ggplot2, gridExtra,
kableExtra, knitr, leaflet, pacman, parallel, plyr, pso,
```

```
read, tibble, tidyverse, xgboost, install = T)
#devtools::install_github("rstats-db/RSQlite")
```

## Importing the Data

The data was downloaded from the UCI Machine Learning Data Repository:

```
# download.file(url = "http://archive.ics.uci.edu/ml/machine-Learning-databases/concrete/compressive/Concrete_Data.xls", destfile = "Concrete_Data.xls", method = "curl", quiet = TRUE)
data_path=(“C:\\\\Users\\\\jeff\\\\R_data\\\\”)
concrete_data <- read_csv(“Concrete_Data.csv”)
## Parsed with column specification:
## cols(
##   `Cement (component 1)(kg in a m^3 mixture)` = col_double(),
##   `Blast Furnace Slag (component 2)(kg in a m^3 mixture)` = col_double(),
##   `Fly Ash (component 3)(kg in a m^3 mixture)` = col_double(),
##   `Water (component 4)(kg in a m^3 mixture)` = col_double(),
##   `Superplasticizer (component 5)(kg in a m^3 mixture)` = col_double(),
##   `Coarse Aggregate (component 6)(kg in a m^3 mixture)` = col_double(),
##   `Fine Aggregate (component 7)(kg in a m^3 mixture)` = col_double(),
##   `Age (day)` = col_double(),
##   `Concrete compressive strength(MPa, megapascals)` = col_double()
## )
```

## Rename variables

```
colnames(concrete_data) <- c("Cement", "Slag", "Ash", "Water",
"Superplasticizer", "Coarse_Aggregate", "Fine_Aggregate",
"Age", "Strength")
```

## Exploratory Data Analysis

```
glimpse(concrete_data)
## Observations: 1,030
## Variables: 9
## $ Cement           <dbl> 540.0, 540.0, 332.5, 332.5, 198.6, 2
66.0, 380...
## $ Slag            <dbl> 0.0, 0.0, 142.5, 142.5, 132.4, 114.0
, 95.0, 9...
## $ Ash             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, ...
```

```

## $ Water <dbl> 162, 162, 228, 228, 192, 228, 228, 2
28, 228, ...
## $ Superplasticizer <dbl> 2.5, 2.5, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, ...
## $ Coarse_Aggregate <dbl> 1040.0, 1055.0, 932.0, 932.0, 978.4,
932.0, 9...
## $ Fine_Aggregate <dbl> 676.0, 676.0, 594.0, 594.0, 825.5, 6
70.0, 594...
## $ Age <dbl> 28, 28, 270, 365, 360, 90, 365, 28,
28, 28, 9...
## $ Strength <dbl> 79.99, 61.89, 40.27, 41.05, 44.30, 4
7.03, 43....

```

Here, we recalculate the composition as proportions ranging from 0 to 1

```

concrete_data[, 1:7] <- t(apply(X = concrete_data[, 1:7],
MARGIN = 1, FUN = function(x) {x/sum(x)}))
# Print summary statistics
glimpse(concrete_data)
## Observations: 1,030
## Variables: 9
## $ Cement <dbl> 0.22309440, 0.22172039, 0.14917003,
0.1491700...
## $ Slag <dbl> 0.00000000, 0.00000000, 0.06393001,
0.0639300...
## $ Ash <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, ...
## $ Water <dbl> 0.06692832, 0.06651612, 0.10228802,
0.1022880...
## $ Superplasticizer <dbl> 0.001032844, 0.001026483, 0.00000000
0, 0.0000...
## $ Coarse_Aggregate <dbl> 0.4296633, 0.4331759, 0.4181247, 0.4
181247, 0...
## $ Fine_Aggregate <dbl> 0.2792811, 0.2775611, 0.2664872, 0.2
664872, 0...
## $ Age <dbl> 28, 28, 270, 365, 360, 90, 365, 28,
28, 28, 9...
## $ Strength <dbl> 79.99, 61.89, 40.27, 41.05, 44.30, 4
7.03, 43....
summary(concrete_data)
##      Cement             Slag              Ash              W
ater
##  Min.   :0.04482   Min.   :0.000000   Min.   :0.00000   Min.
:0.05139
##  1st Qu.:0.08206   1st Qu.:0.000000   1st Qu.:0.00000   1st Q
u.:0.06954
##  Median :0.11528   Median :0.009455   Median :0.00000   Media
n :0.07862
##  Mean:0.11955   Mean:0.031643   Mean:0.02317   Mean:0.07773
##  3rd Qu.:0.14917   3rd Qu.:0.061972   3rd Qu.:0.05034   3rd Qu.:

```

```

0.08386
## Max.:0.22541  Max.:0.150339  Max.:0.08884  Max.:0.11222
## Superplasticizer  Coarse_Aggregate Fine_Aggregate   Age
## Min. :0.0000  Min. :0.3459  Min. :0.2480  Min. : 1.00
## 1st Qu.:0.0000  1st Qu.:0.3923 1st Qu.:0.3112 1st Qu.: 7.00
## Median:0.002727 Median:0.4205  Median:0.3305  Median: 28.00
## Mean:0.002621  Mean:0.4152  Mean:0.3301  Mean: 45.66
## 3rd Qu.:0.004351 3rd Qu.:0.4376 3rd Qu.:0.3541 3rd Qu.: 5
6.00
## Max.:0.013149  Max.:0.4798  Max.:0.4141  Max.:365.00
## Strength
## Min. : 2.33
## 1st Qu.:23.71
## Median :34.45
## Mean :35.82
## 3rd Qu.:46.13
## Max. :82.60

```

Now, we print summary statistics for all variables and check for missing (NA) values:

```

## Check number of NA values in each column
colSums(is.na(concrete_data), na.rm = F)
##          Cement           Slag           Ash           Water
##                 0              0              0              0
## Superplasticizer Coarse_Aggregate Fine_Aggregate   Age
##                 0                  0                  0              0
##          Strength
##                 0

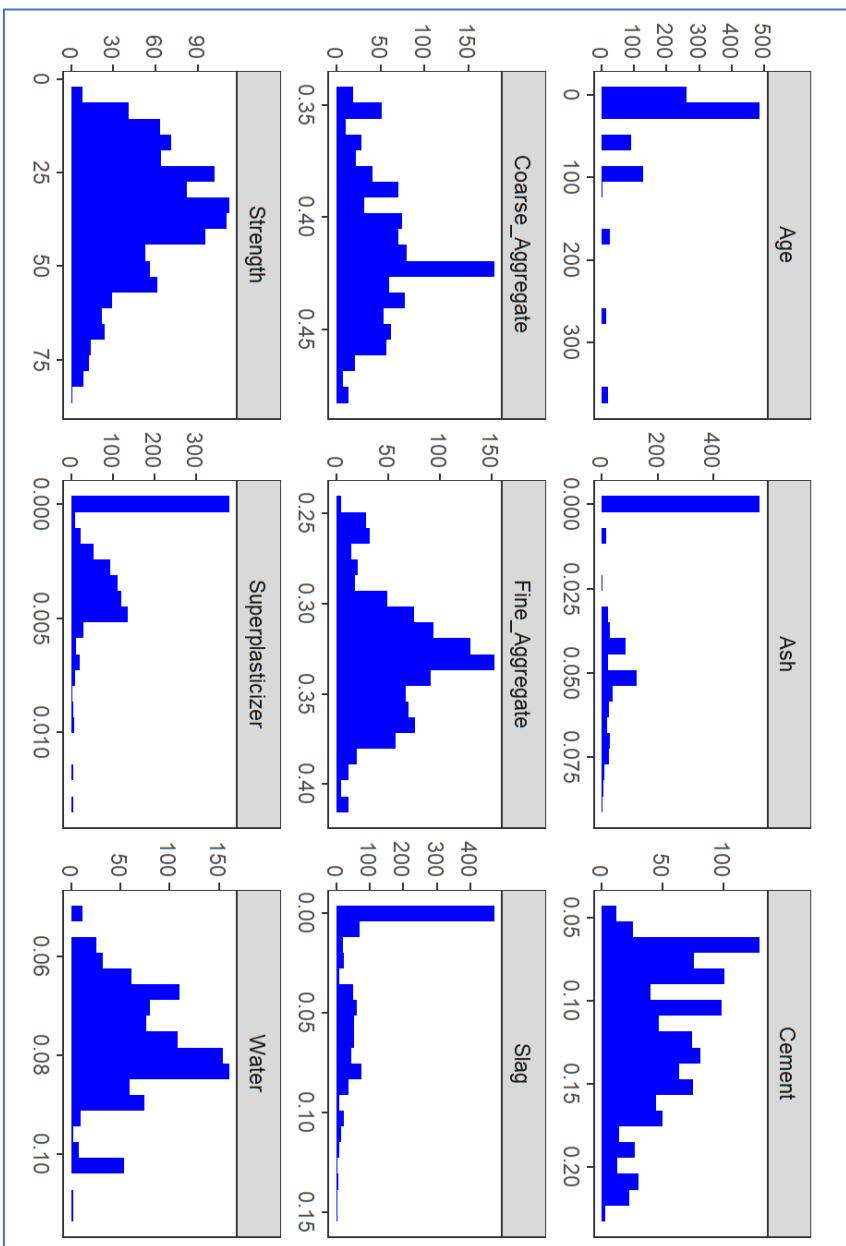
```

There are no NA values in the dataset however input variables have different ranges of values.

```

concrete_data %>%
  gather(key = Variable, value = Value) %>%
  ggplot() +
  geom_histogram(aes(x = Value),
                 bins = 20, fill = "blue") +
  facet_wrap(~Variable, scales='free') +
  theme_bw() + \theme(aspect.ratio = 0.5,
  axis.title = element_blank(),
  panel.grid = element_blank())

```

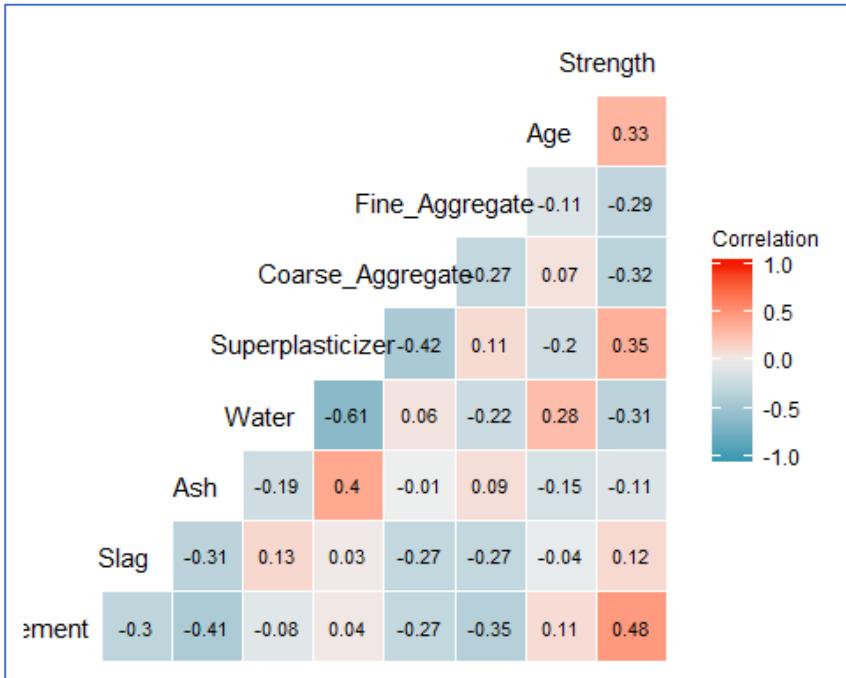


### *Correlations*

Plot correlation heatmap using the `ggcorr()` function from GGally package.

## Plot correlation heatmap

```
ggcorr(concrete_data, label = TRUE, palette = "RdBu",
       name = "Correlation", hjust = 0.75, label_size = 3,
       label_round = 2)
```

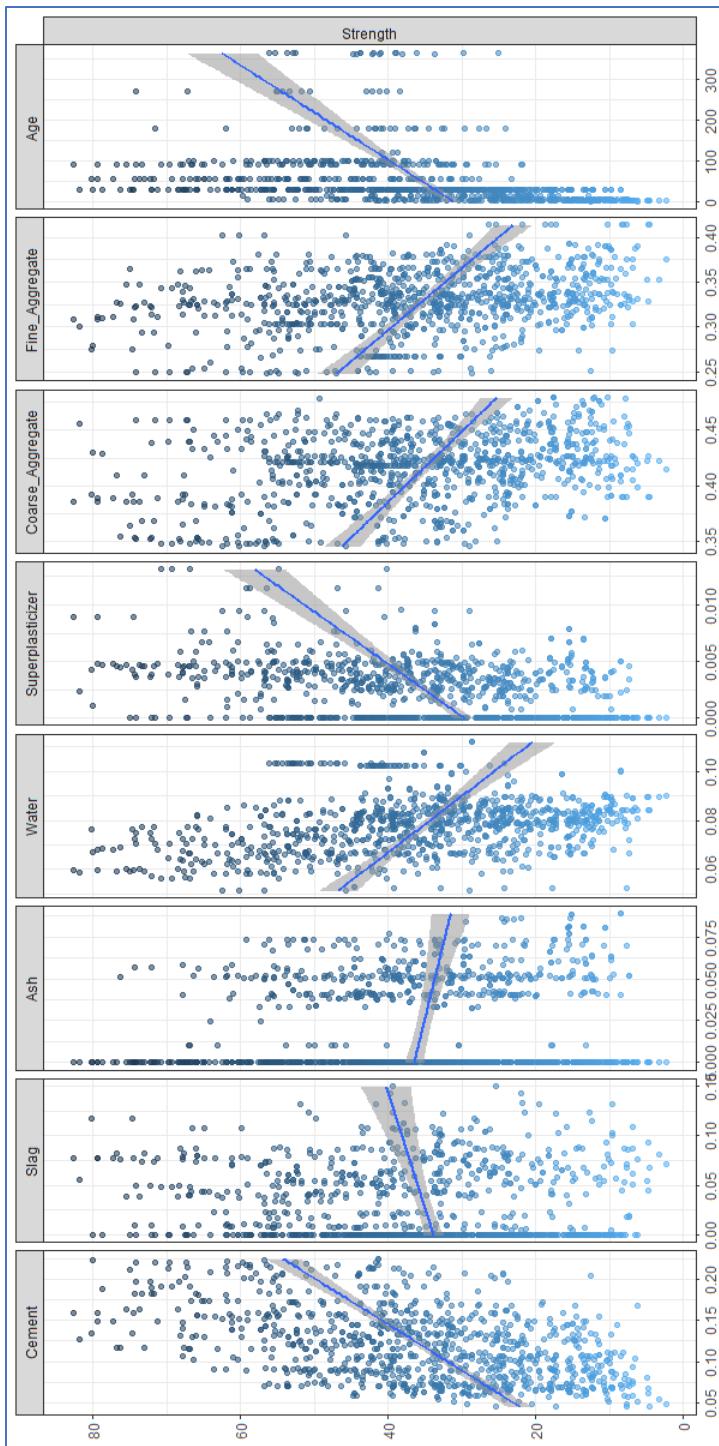


There seems to be a considerable positive correlation between the amount of cement and superplasticizer used and cement strength. Age is also positively correlation with concrete compressive strength. Most pairwise correlations between predictors are generally low.

## Visualizing Feature relationships with a scatterplot matrix

We can use a scatterplot matrix (a collection of scatterplots organized in a grid) to understand the relationship between each predictor and the target feature.

```
ggduo(data = concrete_data, columnsX = 1:8, columnsY = 9,
       types = list(continuous = "smooth_1m"), mapping = ggplot2::aes(color = -Strength, alpha = 0.3)) + theme_bw()
```



## Modeling

### Create training and test sets

80% of the samples from the dataset are randomly selected for the training set, the remaining 20% are allocated for testing the models. Stratified partitioning on the target feature will be applied for the creation of these subsets.

Here we remove any observations with missing values:

```
concrete_data <- concrete_data[complete.cases(concrete_data), ]
```

Then, we average the values of compressive strength of replicate experiments:

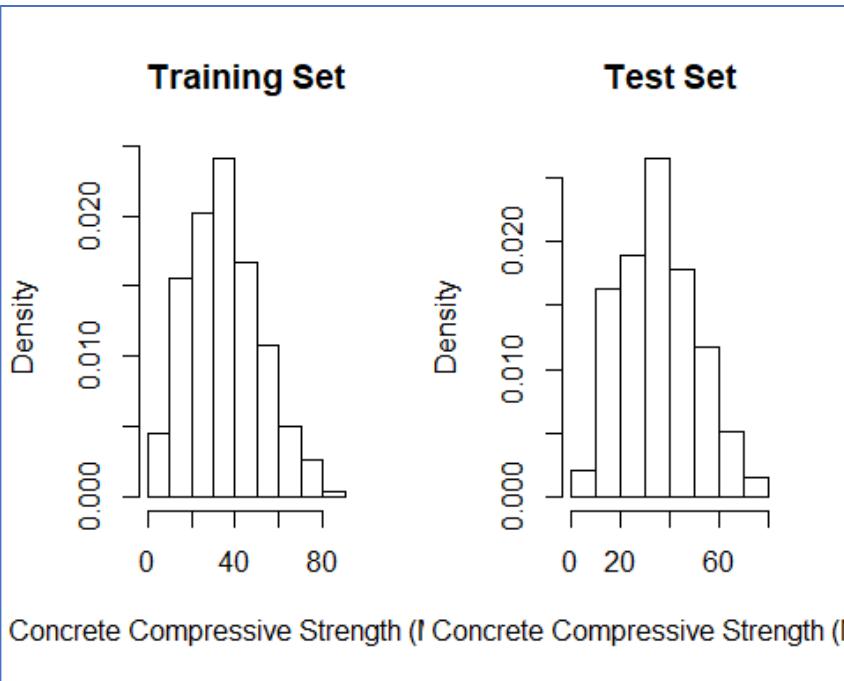
```
concrete_data <- ddply(.data = concrete_data,
  .variables = .(Cement, Slag, Ash, Water, Superplasticizer,
    `Coarse_Aggregate`, `Fine_Aggregate`, Age),
  .fun = function(x) c(Strength = mean(x$Strength)))
```

Next, we create training and test sets using stratified portioning:

```
set.seed(1)
training_index <- createDataPartition(y = concrete_data$Strength, p = 0.80)[[1]]
training_set <- concrete_data[training_index, ]
test_set <- concrete_data[-training_index, ]
```

Finally, we check the distribution of strength on training set and test sets:

```
par(mfrow = c(1, 2))
hist(training_set$Strength, main = "Training Set", xlab =
  "Concrete Compressive Strength (MPa)", freq = FALSE)
hist(test_set$Strength, main = "Test Set", xlab = "Concrete
  Compressive Strength (MPa)", freq = FALSE)
```



```
## Print summary statistics for training and test set
```

```
dplyr::bind_rows(summary(training_set$Strength),
  summary(test_set$Strength)) %>% as_tibble() %>%
  add_column(Subset = c("Training", "Test"), .before = 1)
## # A tibble: 2 x 7
##   Subset   Min. `1st Qu.` Median Mean `3rd Qu.` Max.
##   <chr>     <dbl>    <dbl>  <dbl> <dbl>    <dbl> <dbl>
## 1 Training  2.33    23.5   33.7  35.1    44.6   82.6
## 2 Test      6.94    23.4   34.1  35.3    44.8   76.8
```

The distribution of values of the target feature on both the training and test set are similar.

### Training Parameters

```
CV_folds <- 5 # number of folds
CV_repeats <- 3 # number of repeats
minimum_resampling <- 5 # minimum number of resamples
```

Parameter tuning and selection will be done using repeated 5-fold cross-validation. Each round of cross-validation will be repeated 3 times. Adaptive resampling will be used for model training with grid and random search.

The caret package will be used for model training, tuning and evaluation.

### Training Settings

Here we use trainControl object for repeated cross-validation with random search. This includes finding the minimum number of resamples tested before the model is executed, using a generalized least squares method, indicated by “gls” in the argument. The confidence level is used to exclude parameter settings.

```
adapt_control_random <- caret::trainControl(method = "adaptive_cv",
                                              number = CV_folds, repeats = CV_repeats,
                                              adaptive = list(min = minimum_resampling,
                                                              alpha = 0.05, # confidence Level
                                                              method = "gls", # generalized least squares
                                                              complete = TRUE),
                                              search = "random", # execute random search
                                              verboseIter = FALSE, returnData = FALSE)
```

Now, we setup the trainControl object for standard repeated cross-validation (“repeatedcv”). The parameter verboseIter suppresses the training log when set to false.

```
set.seed(1234)
train_control <- caret::trainControl(method = "repeatedcv",
                                       number = CV_folds, repeats = CV_repeats, verboseIter = FALSE, returnData = FALSE)
```

Next, we setup the trainControl object for repeated cross-validation with grid search, using minimum number of resamples tested before model is excluded, and set the confidence level used to exclude parameter settings.

```
adapt_control_grid <- caret::trainControl(method = "adaptive_cv",
                                             number = CV_folds, repeats = CV_repeats, adaptive = list(min = minimum_resampling,
                                                             alpha = 0.05, # confidence Level
                                                             method = "gls", # generalized least squares
                                                             complete = TRUE),
                                             search = "grid", # execute grid search
                                             verboseIter = FALSE, returnData = FALSE)
```

### Model Training

Extreme Gradient Boosting (XGBoost) will be used to create the regression models. XGBoost is similar to gradient boosting but has the capacity to do parallel computation on a single machine and perform

regularization to avoid overfitting. It was developed by Tianqi Chen. Additional advantages of the XGBoost algorithm include its internal cross-validation function, its ability to handle missing values, its flexibility and its ability to prune the tree until the improvement in loss function is below a threshold.

Similar to GBM, XGBoost uses the errors of previous models to reduce them on the next iteration. The final model is a weighted combination of the models obtained on previous iterations. XGBoost has several tuning parameters some of which depend on the type of booster used (CART or generalized linear model) while others are general, regularization and learning task parameters.

Models will be created using generalized linear models as learners.

When using the `xgbLinear` method in `caret`, there are four hyperparameters to optimize:

- `nrounds`: number of boosting iterations
- `eta`: step size shrinkage
- `lambda`: L2 Regularization (Ridge Regression)
- `alpha`: L1 Regularization (Lasso Regression)

The method above requires the `xgboost` package.

### ***Grid Search***

First, we create the grid for the search. The argument “`nrounds`” sets the number of boosting iterations, and “`eta`” sets the learning rate (a low value means model is more robust to overfitting). The arguments “`lambda`” and “`alpha`” set the methos of regularization as Bayesian Ridge Regression and Lasso Regression, respectively.

```
XGBoost_Linear_grid <- expand.grid(  
  nrounds = c(50, 100, 250, 500), # boosting iterations  
  eta = c(0.01, 0.1, 1), # Learning rate  
  lambda = c(0.1, 0.5, 1), # L2 Regularization  
  alpha = c(0.1, 0.5, 1) # L1 Regularization  
)
```

Train XGBoost models with grid search. 108 different combinations of hyperparameter values are tested. Training will be done using adaptive resampling with a minimum resampling number of 5.

```

GS_T0 <- Sys.time()
cluster <- parallel::makeCluster(detectCores() - 1) # number of cores, convention to leave 1 core for OS
doParallel::registerDoParallel(cluster) # register the parallel processing

```

## Train the Model

Next, we train the model with grid search, using “xgbLinear” as the method for gradient boosting, with the trControl object setting the type of cross validation, “adaptive grid search.” Setting “verbose” to false suppresses the output to be printed when the model fits. The “silent” mode is activated and set to 1, i.e. no running messages will be printed. You can set silent to equal 0, as the messages might help in understanding the model. The argument tuneGrid can take a data frame with columns for each tuning parameter. The column names should be the same as the fitting function’s arguments. The function registerDoSEQ() forces R to return to single threaded processing.

```

set.seed(1);
GS_XGBoost_Linear_model <- caret::train(Strength ~.,
                                         data = training_set,
                                         method = "xgbLinear",
                                         trControl = adapt_control_grid,
                                         verbose = FALSE,
                                         silent = 1,
                                         # tuneLength = 20
                                         tuneGrid = XGBoost_Linear_grid
)

```

```

stopCluster(cluster) # shut down the cluster
registerDoSEQ(); # return to single threaded processing
GS_T1 <- Sys.time()
GS_T1-GS_T0
## Time difference of 6.822978 mins

```

If you decide to print the output afterward (since we set verbose to false), then type SG\_XGBoost\_Linear\_Model.

```

GS_XGBoost_Linear_model
## eXtreme Gradient Boosting
##
## No pre-processing
## Resampling: Adaptively Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 637, 637, 637, 637, 636, 636, ...

```

```

## Resampling results across tuning parameters:
##
##   nrounds  eta  lambda  alpha  RMSE    Rsquared  MAE
##   50       0.01  0.1     0.1    5.049536  0.9047208 3.330799
##   50       0.01  0.1     0.5    5.194508  0.8991922 3.412664
##   50       0.01  0.1     1.0    5.012276  0.9063535 3.309096
##   50       0.01  0.5     0.1    4.998170  0.9064545 3.306838
##   50       0.01  0.5     0.5    5.024338  0.9054681 3.309119
...
##   100      0.01  0.1     0.1    5.043471  0.9048086 3.313304
##   100      0.01  0.1     0.5    5.156669  0.9003716 3.347023
##   100      0.01  0.1     1.0    4.972845  0.9075382 3.250918
##   100      0.01  0.5     0.1    5.015873  0.9074391 3.287405
##   100      0.01  0.5     0.5    4.973409  0.9072377 3.217236
...
##   250      0.01  0.1     0.1    5.035747  0.9050662 3.300938
##   250      0.01  0.1     0.5    5.154856  0.9004296 3.343376
##   250      0.01  0.1     1.0    4.973259  0.9075212 3.251353
##   250      0.01  0.5     0.1    4.995654  0.9081349 3.259006
##   250      0.01  0.5     0.5    5.053146  0.9063806 3.284016
...
##   500      0.01  0.1     0.1    5.035747  0.9050662 3.300938
##   500      0.01  0.1     0.5    5.154856  0.9004296 3.343376
##   500      0.01  0.1     1.0    4.973259  0.9075212 3.251353
##   500      0.01  0.5     0.1    4.995654  0.9081349 3.259006
##   500      0.01  0.5     0.5    5.053146  0.9063806 3.284016
...
##
## RMSE was used to select the optimal model using the smallest
## value.
## The final values used for the model were nrounds = 500, lambd
## a = 1,
## alpha = 0.5 and eta = 0.1.

```

We then use `saveRDS` to save the model, giving it a path either to our machine's directory or to the environment we are working in (i.e., Hadoop, Domino, etc.)

```

saveRDS(object = GS_XGBoost_Linear_model, file = "C:/Users
/jeff/GS_XGBoost_Linear_model.rds")
saveRDS(object = GS_XGBoost_Linear_model$finalModel, file
= paste0("C:/Users/jeff/GS_XGBoost_Linear_model_", class(G
S_XGBoost_Linear_model$finalModel)[1], ".rds"))

```

The best model from the grid search has an average cross-validated root-mean-squared-error (RMSE) of 4.683. This model has 500 iterations and an alpha and lambda of 1 and 0.5, respectively. The step size shrinkage (eta) is 0.01.

The effect of the hyperparameters values tested on the cross-validation statistics can be seen on the charts below.

```
RMSE_plot_XGBoost_Linear <-  
  GS_XGBoost_Linear_model$results %>%  
    dplyr::select(nrounds, lambda, alpha, eta, RMSE) %>%  
    gather(key = Parameter,  
           value = Value, -nrounds, -lambda, -alpha, -eta) %>%  
    filter(Parameter == "RMSE") %>%  
    ggplot(mapping = aes(x = nrounds, y = Value,  
                          col = factor(alpha), shape = factor(lambda))) +  
      geom_line(size = 1) +  
      geom_point(size = 2) +  
      facet_wrap(~eta) +  
      labs(title = "Average Cross-validated RMSE of XGBoost  
Linear Model",  
            subtitle = "Horizontal tiles representative of step  
size shrinkage",  
            x = "# Boosting Iterations",  
            y = "RMSE",  
            col = "alpha",  
            shape = "lambda")  
  ) +  
  theme_bw() +  
  theme(  
    # panel.grid = element_blank(),  
    legend.title = element_text(size = 10, face="bold"),  
    legend.text = element_text(size = 9),  
    plot.title = element_text(size=16),  
    axis.title=element_text(size=10, face="bold"),  
    axis.text.x = element_text(angle = 0)) +  
    scale_x_continuous(breaks = unique(  
      GS_XGBoost_Linear_model$results$nrounds)) +  
    scale_color_brewer(type = "qual", palette = "Set1")
```

Now we define the mean absolute error (MAE) for the model plot of average cross validation, using the model results given by GS\_XGBoost\_Linear\_model\$results.

```
MAE_plot_XGBoost_Linear <-  
  GS_XGBoost_Linear_model$results %>%  
    dplyr::select(nrounds, lambda, alpha, eta, MAE) %>%  
    gather(key = Parameter,
```

```

    value = Value, -nrounds, -lambda, -alpha, -eta) %>%
filter(Parameter == "MAE") %>%
ggplot(mapping = aes(x = nrounds, y = Value,
  col = factor(alpha), shape = factor(lambda))) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  facet_wrap(~eta) +
  labs(
    title =
      "Average Cross-validated MAE of XGBoost Linear Model",
    subtitle =
      "Horizontal tiles representative of step size shrinkage",
    x = "# Boosting Iterations",
    y = "MAE",
    col = "alpha",
    shape = "lambda"
  ) +
  theme_bw() +
  theme(
    # panel.grid = element_blank(),
    legend.title = element_text(size = 10, face="bold"),
    legend.text = element_text(size = 9),
    plot.title = element_text(size=16),
    axis.title=element_text(size=10, face="bold"),
    axis.text.x = element_text(angle = 0)) +
  scale_x_continuous(breaks =
    unique(GS_XGBoost_Linear_model$results$nrounds)) +
  scale_color_brewer(type = "qual", palette = "Set1")

```

Finally, we define R-squared for the model plot of average cross validation, again using the model results.

```

Rsquared_plot_XGBoost_Linear <-
  GS_XGBoost_Linear_model$results %>%
  dplyr::select(nrrounds, lambda, alpha, eta, Rsquared) %>%
  gather(key = Parameter, value = Value, -nrrounds,
  -lambda, -alpha, -eta) %>%
  filter(Parameter == "Rsquared") %>%
  ggplot(mapping = aes(x = nrrounds, y = Value,
  col = factor(alpha), shape = (lambda))) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  facet_wrap(~eta) +
  labs(title =

```

```

“Average Cross-validated R^2 of XGBoost Linear Model”,
subtitle =
“Horizontal tiles illustrative of step size shrinkage”,
x = “# Boosting Iterations”,
y = “R-squared”,
col = “alpha”,
shape = “lambda”
) +
theme_bw() +
theme(
# panel.grid = element_blank(),
legend.title = element_text(size = 10, face=“bold”),
legend.text = element_text(size = 9),
plot.title = element_text(size=16),
axis.title=element_text(size=10, face=“bold”),
axis.text.x = element_text(angle = 0)) +
scale_x_continuous(breaks =
unique(GS_XGBoost_Linear_model$results$nrounds)) +
# Optional parameter: scale_y_continuous(limits =
c(0,1), expand = c(0,0), breaks = seq(0,1,0.1)) +
scale_color_brewer(type = “qual”, palette = “Set1”)

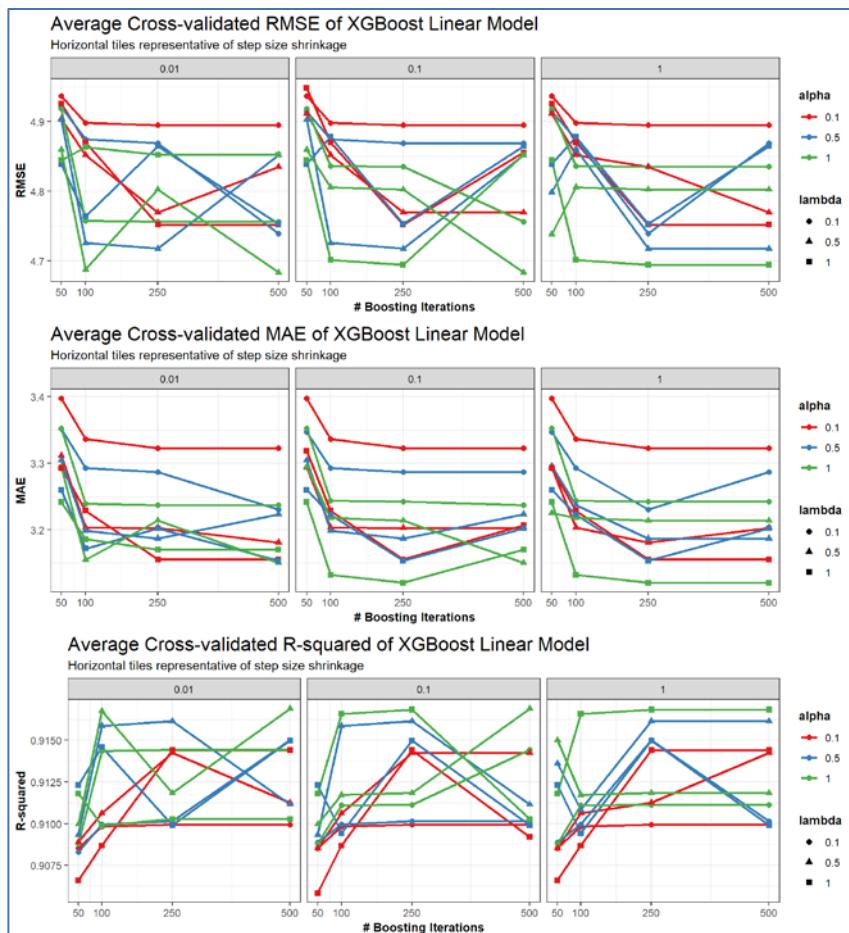
```

We are now ready to plot the results using grid.arrange.

```

grid.arrange(grobs = list(RMSE_plot_XGBoost_Linear,
MAE_plot_XGBoost_Linear,
Rsquared_plot_XGBoost_Linear), ncol = 1, nrow = 3)

```



## Random Search

```
n_combinations <- nrow(XGBoost_Linear_grid)
```

In this section, we train XGBoost models with random search. The same number of different combinations of hyperparameters used in the grid search (108) will be tested. Training will be done using adaptive resampling with a minimum resampling of 5.

```
RS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 1) # number of cores, convention to leave 1 core for OS
registerDoParallel(cluster) # register the parallel processing
```

```

set.seed(1);
RS_XGBoost_Linear_model <- caret::train(Strength ~.,
                                         data = training_set,
                                         method = "xgbLinear",
                                         trControl = adapt_control_random,
                                         verbose = FALSE,
                                         silent = 1,
                                         tuneLength = n_combinations
)
stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #force R to return to single threaded processing
RS_T1 <- Sys.time()
RS_T1-RS_T0
## Time difference of 5.588873 mins
RS_XGBoost_Linear_model
## eXtreme Gradient Boosting
##
## No pre-processing
## Resampling: Adaptively Cross-Validated
## (5 fold, repeated 3 times)
## Summary of sample sizes: 637, 637, 637, 637, 636, ...
## Resampling results across tuning parameters:
##
##     lambda      alpha      nrnds      eta      RMSE      Rsquare
##     1.167e-05  4.397e-01    92      1.4585   5.0456   0.9053
##     1.308e-05  3.178e-03    28      1.3108   5.1265   0.9037
##     1.971e-05  8.208e-02    82      1.4275   5.1288   0.9020
##     2.038e-05  3.280e-05    52      2.4364   5.1228   0.9027
##     2.256e-05  1.703e-03     7      0.5316   6.5213   0.8796
##     2.638e-05  1.666e-02     5      2.7757   8.7760   0.8670
##     3.143e-05  7.942e-02    11      1.4790   5.4589   0.8926
##     3.465e-05  1.832e-03    61      0.9998   5.1007   0.9036
##     4.059e-05  8.972e-03    47      1.0722   5.1663   0.9025
##     4.244e-05  2.428e-05    46      0.5154   5.1312   0.9020
##     4.435e-05  1.239e-01    41      2.8071   5.1313   2.3787
##     7.635e-05  1.587e-02     5      1.0919   8.7760   0.8670
##     8.533e-05  1.018e-02    95      0.2467   5.1406   0.9019
##     1.020e-04  6.076e-04    27      0.8036   5.1963   0.9006
##     1.032e-04  4.330e-01    10      2.8209   5.5750   0.8902
##
## Truncated after 15 observations
##

```

```

## Minimizing RMSE was used to select the optimal.
## The final values used for the model were nrounds = 93,
## lambda = 0.0139820, alpha = 0.037830 and eta = 1.19805.
saveRDS(object = RS_XGBoost_Linear_model, file = "C:/Users/jeff/R_data/RS_XGBoost_Linear_model.rds")
saveRDS(object = RS_XGBoost_Linear_model$finalModel, file = paste0("C:/Users/jeff/R_data/RS_XGBoost_Linear_model_",
class(RS_XGBoost_Linear_model$finalModel)[1]," .rds"))

```

The best model from the grid search has an average cross-validated RMSE of 4.717. This model has 64 iterations and a alpha and lambda of 0.0018598 and 0.9110254, respectively. The step size shrinkage (eta) is 0.5293561.

### **Genetic Algorithm**

First, we define the parameter settings for search algorithm:

```

max_iter <- 10 # maximum number of iterations
pop_size <- 10 # population size

```

The GA package will be used to optimize the hyperparameters of the XGBoost model using a genetic algorithm. Prior to run the search method, an objective function needs to be created. Similar to the grid and random search, the average cross-validated root-mean-square error (RMSE) will be used as the parameter to be optimized.

Next, we create a custom function for assessing solutions:

```

eval_function_XGBoost_Linear <- function(x1, x2, x3, x4, data, train_settings) {
  suppressWarnings(
    XGBoost_Linear_model <- caret::train(Strength ~.,
      data = data,
      method = "xgbLinear",
      trControl = train_settings,
      verbose = FALSE,
      silent = 1,
      tuneGrid = expand.grid(
        nrounds = round(x1), # number of boosting iterations
        eta = 10^x2, # Learning rate
        alpha = 10^x3, # L1 Regularization on weights
        lambda = 10^x4 # L2 Regularization on weights
      )
  )
}

```

```

    )
  return(-XGBoost_Linear_model$results$RMSE) # min RMSE
}

```

Now, we define the minimum and maximum values for each input:

```

nrounds_min_max <- c(10,10^3)
eta_min_max <- c(-5,3)
alpha_min_max <- c(-3,1)
lambda_min_max <- c(-3,1)

```

The population size was set to 10 with a maximum of 10 round of iterations. Minimum and maximum values for each parameter was defined above.

```

set.seed(1)
n_cores <- detectCores()-1
GA_T0 <- Sys.time()

```

Now, we run genetic algorithm.

```

GA_model_XGBoost_Linear <- GA:::ga(type = "real-valued",
                                       fitness = function(x) eval_function_XGBoost_Linear(x[1],x[2],x[3],x[4]),
                                       data = training_set,
                                       train_settings = train_control),
                                       lower = c(nrounds_min_max[1], eta_min_max[1], alpha_min_max[1], lambda_min_max[1]), # minimum value
                                       upper = c(nrounds_min_max[2], eta_min_max[2], alpha_min_max[2], lambda_min_max[2]), # maximum value
                                       popSize = pop_size, # population size
                                       maxiter = max_iter, # number of iterations
                                       pmutation = 0.5, # probability of mutation
                                       elitism = 0.3, # percentage of elitism (fraction of best current solutions used on next round)
                                       # suggestions = starting_point,
                                       parallel = n_cores,
                                       optim = F,
                                       keepBest = T,
                                       seed = 1
)
GA_T1 <- Sys.time()
GA_T1-GA_T0
## Time difference of 19.84363 mins

```

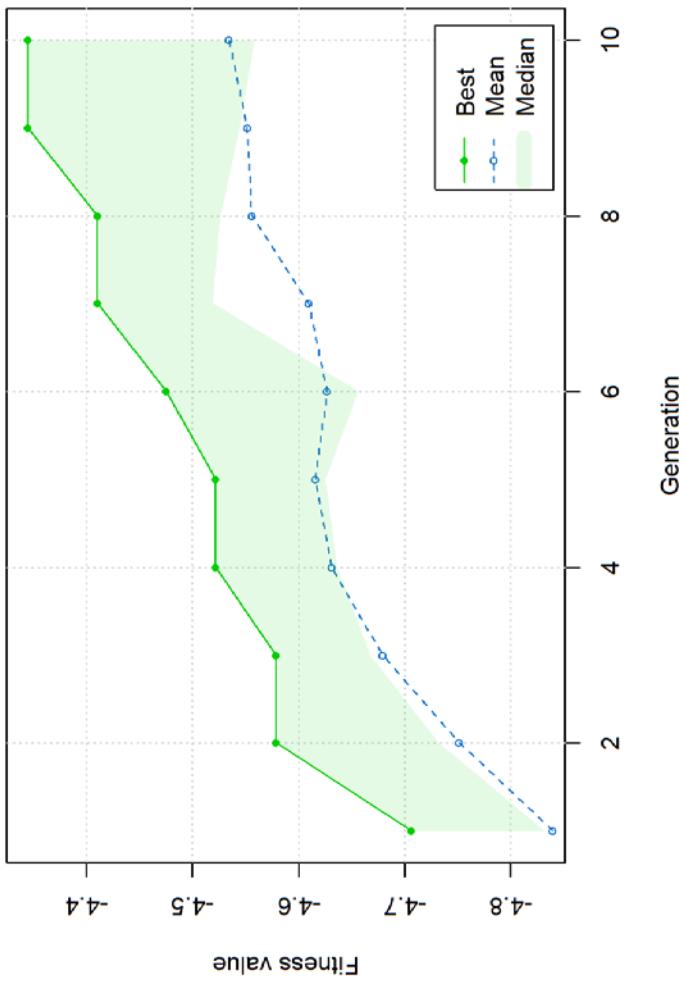
Next, using “summary”, we print summary of search method

```
summary(GA_model_XGBoost_Linear)
## -- Genetic Algorithm -----
## 
## GA settings:
## Type                  = real-valued
## Population size       = 10
## Number of generations = 10
## Elitism                = 0.3
## Crossover probability = 0.8
## Mutation probability  = 0.5
## Search domain =
##           x1 x2 x3 x4
## lower    10 -5 -3 -3
## upper   1000  3  1  1
##
## GA results:
## Iterations            = 10
## Fitness function value = -4.613893
## Solution =
##           x1          x2          x3          x4
## [1,] 220.0211 -0.2034734 -0.4117592 0.4448379
```

Finally, we pplot the outcome.

```
par(mfrow = c(1, 1))
GA::plot.ga(GA_model_XGBoost_Linear, main = "Genetic Algorithm: RMSE values at each iteration")
```

**Genetic Algorithm: RMSE values at each iteration**



The search was completed and found the optimal RMSE value to be 4.344.

Once the optimal values are found for each hyperparameter, the final model is trained with these values. For the learning rate, low value means model is more robust to overfitting.

```
GA_XGBoost_Linear_grid <- expand.grid(  
  nrounds = round(GA_model_XGBoost_Linear@solution[1]), #  
  Learning rate  
  eta = 10^GA_model_XGBoost_Linear@solution[2], # number o  
f boosting iterations
```

```

# L2 Regularization
alpha = 10^GA_model_XGBoost_Linear@solution[3],
# L1 Regularization
lambda = 10^GA_model_XGBoost_Linear@solution[4])
T0 <- Sys.time()
# number of cores, convention to Leave 1 core for OS
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster) # register parallel processing

```

Now, we establish the grid of optimal hyperparameter values:

```

set.seed(1)
GA_XGBoost_Linear_model <- caret::train(Strength ~.,
                                         data = training_set,
                                         method = "xgbLinear",
                                         trControl = train_control,
                                         verbose = F, metric = "RMSE", max
                                         imize = FALSE,
                                         silent = 1,
                                         # tuneLength = 1
                                         tuneGrid = GA_XGBoost_Linear_grid
)

stopCluster(cluster) # shut down the cluster
registerDoSEQ() # force R to return to single threaded pr
ocessing
T1 <- Sys.time()
T1-T0
## Time difference of 23.42039 secs
GA_XGBoost_Linear_model
## eXtreme Gradient Boosting
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 637, 637, 637, 637, 636, 636,
...
## Resampling results:
##
##    RMSE      Rsquared   MAE
##    4.707331  0.91719   3.145469
##
## Tuning parameter 'nrounds' was held constant at a value
## of 220
##
## Tuning parameter 'alpha' was held constant at a value o

```

```
f 0.3874724
##
## Tuning parameter 'eta' was held constant at a value of
0.6259312
```

Now, we save the model as we did previously.

```
saveRDS(object = GA_XGBoost_Linear_model, file = paste0("C
:/Users/jeff/R_data/GA_XGBoost_Linear_model_PSO_",
stringr
::str_remove_all(Sys.time(), pattern = ":"), ".rds"))
saverDS(object = GA_XGBoost_Linear_model$finalModel, file
= paste0("C:/Users/jeff/R_data/GA_XGBoost_Linear_model_PSO
_", class(GA_XGBoost_Linear_model$finalModel)[1], "_",
str
ingr::str_remove_all(Sys.time(), pattern = ":"), ".rds"))
```

The best model from the search with the genetic algorithm has an average cross-validated RMSE of 4.833.

This model has 420 iterations and a alpha and lambda of 0.0001534 and 0.0016644, respectively. The step size shrinkage (eta) is .

### *Differential Evolution*

Set parameter settings for search algorithm

```
max_iter <- 10 # maximum number of iterations
pop_size <- 10 # population size
```

The DEoptim package will be used to optimize the hyperparameters of the XGBoost model using differential evolution. Prior to run the search method, a custom objective function needs to be created. The average cross-validated root-mean-square error (RMSE) will be used as the parameter to be optimized.

Again, we create a custom function for assessing solutions:

```
eval_function_XGBoost_Linear <- function(x, data, train_se
ttings) {
  x1 <- x[1]; x2 <- x[2]; x3 <- x[3]; x4 <- x[4]

  suppressWarnings(
    XGBoost_Linear_model <- caret::train(Strength ~.,
      data = data,
      method = "xgbLinear",
      trControl = train_settings,
```

```

    verbose = FALSE,
    silent = 1,
    tuneGrid = expand.grid(
      nrounds = round(x1), # number of boosting iterations
      eta = 10^x2, # Learning rate
      alpha = 10^x3, # L1 Regularization on weights
      lambda = 10^x4 # L2 Regularization on weights
    )
  )
}

return(XGBoost_Linear_model$results$RMSE) # minimize RMSE
}

```

Now, we define the minimum and maximum values for each input:

```

nrounds_min_max <- c(10,10^3)
eta_min_max <- c(-5,3)
alpha_min_max <- c(-3,1)
lambda_min_max <- c(-3,1)

set.seed(1)
n_cores <- detectCores()-1

DE_T0 <- Sys.time()

```

Next, we run differential evolution algorithm.

```

DE_model_XGBoost_Linear <- DEoptim::DEoptim(
  fn = eval_function_XGBoost_Linear,
  lower = c(nrounds_min_max[1], eta_min_max[1],
            alpha_min_max[1], lambda_min_max[1]),
  upper = c(nrounds_min_max[2], eta_min_max[2],
            alpha_min_max[2], lambda_min_max[2]),
  control = DEoptim.control(
    NP = pop_size, # population size
    itermax = max_iter, # maximum number of iterations
    CR = 0.5, # probability of crossover
    storepopfreq = 1, # store every population
    parallelType = 1 # run parallel processing
  ),
  data = training_set,
  train_settings = train_control
)
## Warning in DEoptim::DEoptim(fn = eval_function_XGBoost_
Linear, lower = c(nrounds_min_max[1], : For many problems

```

```

it is best to set 'NP' (in 'control') to be at least ten times the length of the parameter vector.

## Iteration: 1 bestvalit: 4.670648 bestmemit: 209.665112
1.158731 -1.931117 0.309493
## Iteration: 2 bestvalit: 4.670648 bestmemit: 209.665112
1.158731 -1.931117 0.309493
## Iteration: 3 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 4 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 5 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 6 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 7 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 8 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 9 bestvalit: 4.622383 bestmemit: 614.602559
0.740948 0.271932 0.176959
## Iteration: 10 bestvalit: 4.622383 bestmemit: 614.602559
9 0.740948 0.271932 0.176959
DE_T1 <- Sys.time()
DE_T1-DE_T0
## Time difference of 21.55292 mins

```

Now, we print search results as we did previously.

```

summary(DE_model_XGBoost_Linear)
##
## ***** summary of DEoptim object *****
## best member : 614.6026 0.74095 0.27193 0.17696
## best value : 4.62238
## after : 10 generations
## fn evaluated : 22 times
## ****
DE_solutions <- DE_model_XGBoost_Linear$optim$bestmem

```

Finally, we plot the results.

```

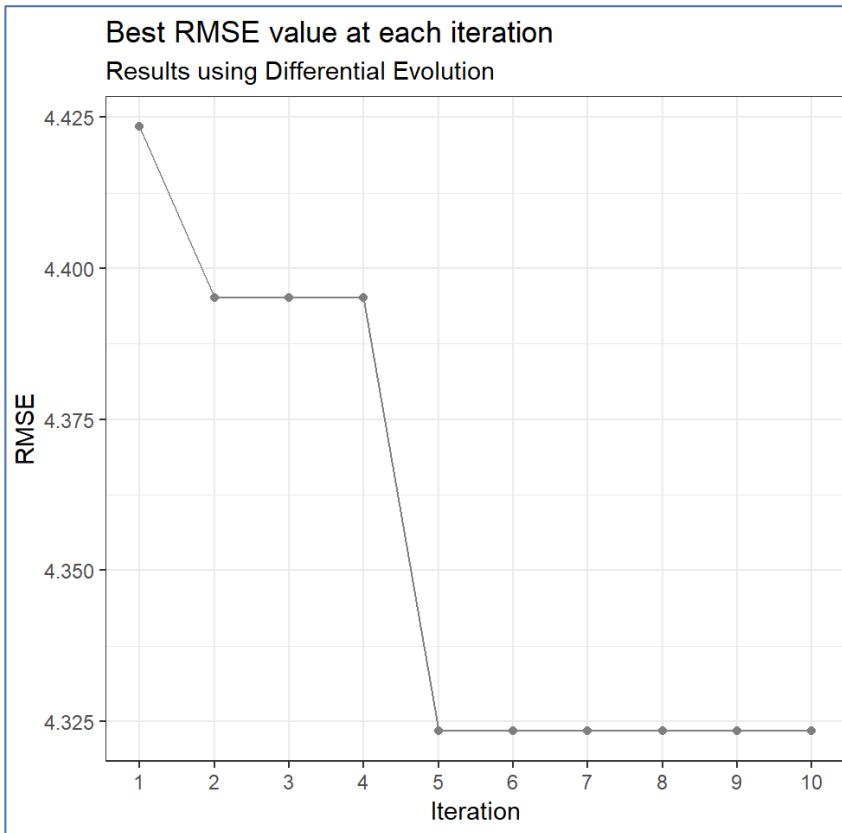
ggplot(mapping = aes(x = 1:length(DE_model_XGBoost_Linear$member$bestvalit), y = DE_model_XGBoost_Linear$member$bestvalit)) +

```

```

geom_line(col = "grey50") +
geom_point(col = "grey50") +
theme_bw() +
theme(aspect.ratio = 0.9) +
labs(x = "Iteration", y = "RMSE",
title = "Best RMSE value at each iteration",
subtitle = "Results using Differential Evolution") +
scale_x_continuous(breaks = 1:
DE_model_XGBoost_Linear$optim$iter,
minor_breaks = NULL)

```



The search was completed and found the optimal RMSE value to be 4.323 after 10 iterations and 22 function evaluations.

The final model is trained with the optimal values found for each hyperparameter.

Next, we establish a grid of optimal hyperparameter values:

```
DE_XGBoost_Linear_grid <- expand.grid(
  nrounds = round(DE_solutions[1]), # Learning rate
  eta = 10^DE_solutions[2], # number of boosting iteration
  s
  alpha = 10^DE_solutions[3], # L2 Regularization
  lambda = 10^DE_solutions[4] # L1 Regularization
)

T0 <- Sys.time()
# number of cores, convention to leave 1 core for OS
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster) # register parallel processing
```

Finally, we train the model with optimal values:

```
set.seed(1)

DE_XGBoost_Linear_model <- caret::train(Strength ~.,
  data = training_set,
  method = "xgbLinear",
  trControl = train_control,
  verbose = F, metric = "RMSE", maximize = FALSE,
  silent = 1,
  # tuneLength = 1
  tuneGrid = DE_XGBoost_Linear_grid
)

stopCluster(cluster) #shut down the cluster
registerDoSEQ() #force R to use single-threaded processing
T1 <- Sys.time()
T1-T0
## Time difference of 16.52981 secs
```

We can then print the model output if needed as we have done previously.

```
DE_XGBoost_Linear_model
## eXtreme Gradient Boosting
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 637, 637, 637, 637, 636, 636,
...
```

```
## Resampling results:  
##  
##    RMSE      Rsquared     MAE  
##    4.713553  0.9175857  3.114525  
##  
## Tuning parameter 'nrounds' was held constant at a value  
of 615  
##  
## Tuning parameter 'alpha' was held constant at a value o  
f 1.870389  
##  
## Tuning parameter 'eta' was held constant at a value of  
5.507418
```

We then save the model as we have done previously.

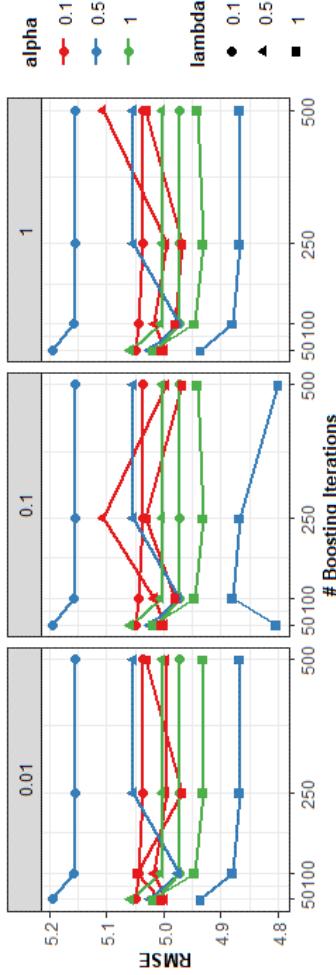
```
saveRDS(object = DE_XGBoost_Linear_model, file = paste0("C  
:/Users/jeff/R_data/DE_XGBoost_Linear_model_PSO_", stringr  
::str_remove_all(Sys.time(), pattern = ":"), ".rds"))  
saveRDS(object = DE_XGBoost_Linear_model$finalModel, file  
= paste0("C:/Users/jeff/R_data/DE_XGBoost_Linear_model_PSO  
_", class(DE_XGBoost_Linear_model$finalModel)[1], "_", str  
ingr::str_remove_all(Sys.time(), pattern = ":"), ".rds"))
```

The best model from the search with the differential evolution algorithm has an average cross-validated RMSE of 4.589.

This model has 577 iterations and a alpha and lambda of 0.0015413 and 7.0589095, respectively. The step size shrinkage (eta) is 0.2028573.

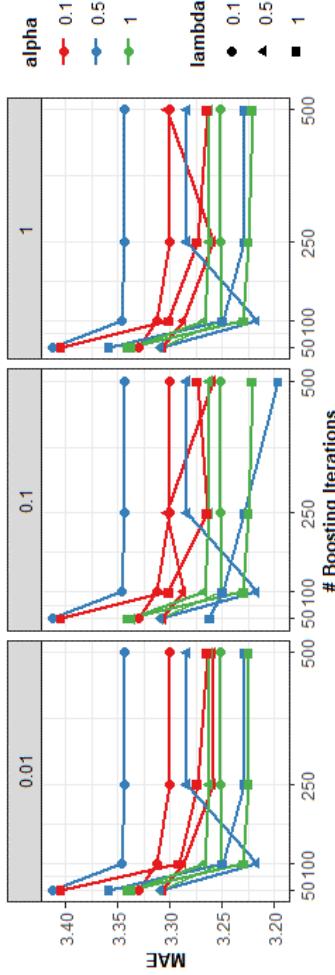
### Average Cross-validated RMSE of XGBoost Linear Model

Horizontal tiles representative of step size shrinkage



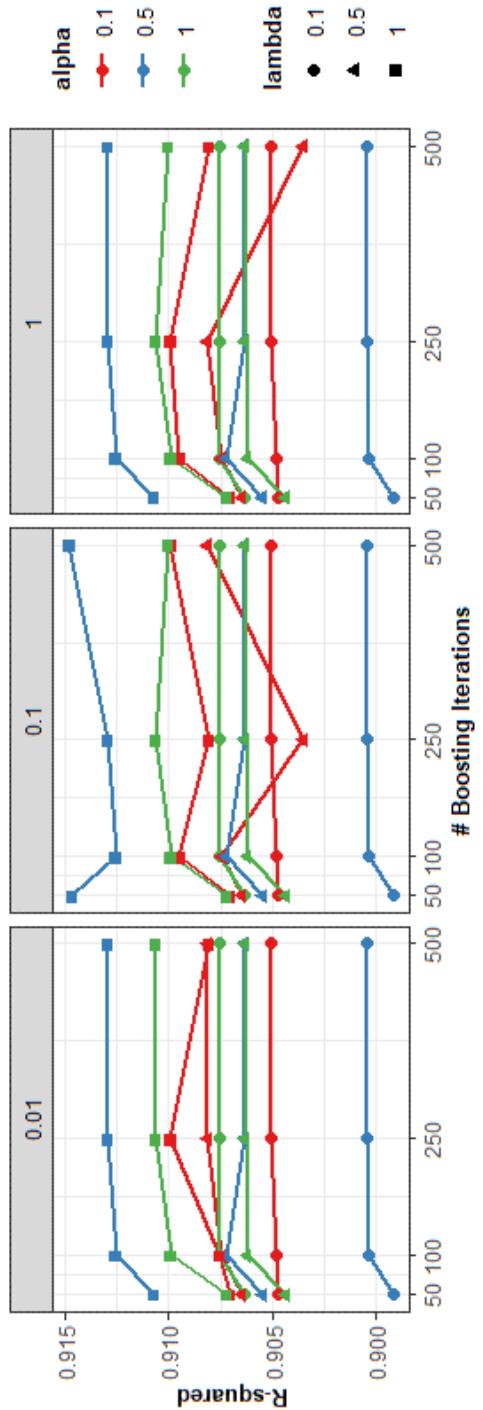
### Average Cross-validated MAE of XGBoost Linear Model

Horizontal tiles representative of step size shrinkage



## Average Cross-validated R-squared of XGBoost Linear Model

Horizontal tiles representative of step size shrinkage



## Model Performance

### Performance on Training Set

The training set statistics and hyperparameters values for each search method tested are summarised on the table below.

#### Create summary table

```
Summary_Table_Training <- bind_rows(  
  GS_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  RS_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  GA_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  DE_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  DE_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5))  
  
Summary_Table_Training <- Summary_Table_Training %>%  
  add_column(Method = c("Grid Search", "Random Search", "G  
enetic Algorithm", "Differential Evolution", "Particle Swa  
rm Optimization"), .before = 1) %>%  
  add_column(`Processing Time` = round(c(GS_T1-GS_T0, RS_T  
1-RS_T0, GA_T1-GA_T0, DE_T1-GS_T0, DE_T1-GS_T0),0))
```

#### Print table

```
Summary_Table_Training %>%  
  kable(align = "c", caption = "Training Set Statistics an  
d Hyperparameter Values of XGBoost Models.") %>%  
  kableExtra::kable_styling(bootstrap_options = c("striped"  
", "hover", "condensed", "responsive"), full_width = T, po  
sition = "center") %>%  
  kableExtra::footnote(general = paste0("Note:\nSummary st  
atistics obtained using ", CV_folds, "-fold cross validati  
on repeated ", CV_repeats, " times.", "\nGrid and Random Se  
arch performed with adaptive resampling."), general_title  
= "\n ")
```

### *Training Set Statistics and Hyperparameter Values of XGBoost Models.*

Method	RMSE	MAE	R <sup>2</sup>	n-rds	η	λ	α	Prcg Time
GS	4.7996	3.1966	0.91479	500	0.1000	1.0000	0.5000	7 m
RS	5.0317	3.3256	0.90610	93	1.1981	0.0140	0.0378	6 m
GA	4.7073	3.1455	0.91719	220	0.6259	2.7851	0.3875	20 m
DE	4.5893	3.1145	0.91759	615	5.5074	1.5030	1.8704	54 m
PSO	4.7136	3.1145	0.91759	615	5.5074	1.5030	1.8704	54 m

### *Training Set Statistics and Hyperparameter Values of XGBoost Models.*

Note: Summary statistics obtained using 5-fold cross validation repeated 3 times. Grid and Random Search performed with adaptive resampling.

Although there are some considerable differences between the values of hyperparameters obtained from each search method, the performance metrics are relatively similar.

Differential Evolution obtained the lowest RMSE value (4.5893).

### *Performance on Test Set*

Custom functions to plot observed, predicted and residual values for each method

#### *Function to plot observed vs predicted values*

```
predicted_observed_plot <- function(predicted_val, observed_val, residual_val, model_name = "", R_squared, ...) {  
  
  plot <- ggplot(mapping = aes(x = predicted_val, y = observed_val, col = abs(residual_val))) +  
    geom_point(alpha = 0.9, size = 2) +  
    geom_abline(intercept = 0, slope = 1) +  
    # facet_wrap(~) +  
    labs(title = paste0(model_name, "\nPredicted vs Observed: Test Set"),  
         subtitle = paste0("R-squared: ", R_squared),  
         x = "Predicted",  
         y = "Observed",  
         col = "Absolute Deviation") +  
    theme_bw() +
```

```

    theme(aspect.ratio = 0.9, panel.grid.minor.x = element_
_blank(), legend.title = element_text(size = 10, face="bol
d"), legend.text = element_text(size = 9), plot.title = el
ement_text(size=12, face="bold"), axis.title=element_text(
size=10, face="bold"), axis.text.x = element_text(angle =
0), legend.position = "none") +
    # scale_x_continuous(expand = c(0,0)) +
    # scale_y_continuous(expand = c(0,0)) +
    coord_equal() + scale_color_viridis_c(direction = -1)

    return (plot)
}

```

### **Function to plot residuals**

```

residuals_plot <- function(predicted_val, residual_val, mo
del_name = "", MAE, RMSE, ...) {

  plot <- ggplot(mapping = aes(x = predicted_val, y = resi
dual_val, col = abs(residual_val))) +
    geom_point(alpha = 0.9, size = 2) +
    geom_abline(intercept = 0, slope = 0) +
    # facet_wrap(~) +
    labs(
      title = paste0(model_name, "\nResiduals: Test Set"),
      subtitle = paste0("RMSE: ", RMSE, ", MAE: ", round(M
AE, 3)),
      x = "Predicted",
      y = "Residual",
      col = "Absolute Deviation"
    ) +
    theme_bw() +
    theme(aspect.ratio = 0.9, panel.grid.minor.x = element_
_blank(), legend.title = element_text(size = 10, face="bol
d"), legend.text = element_text(size = 9), plot.title = el
ement_text(size=12, face="bold"), axis.title=element_text(
size=10, face="bold"), axis.text.x = element_text(angle =
0), legend.position = "none") +
    # scale_x_continuous(expand = c(0,0)) +
    # scale_y_continuous(expand = c(0,0)) +
    coord_equal() + scale_color_viridis_c(direction = -1)

  return (plot)
}

```

## Grid Search (GS)

1. Make predictions on test set

```
test_set$XGBoost_Linear_GS <- predict(GS_XGBoost_Linear_model, test_set)
# Calculate Residuals on test set
test_set$XGBoost_Linear_GS_residual <- test_set$Strength - test_set$XGBoost_Linear_GS
```

2. Calculate test set R-squared, RMSE, MAE

```
R_squared <- round(cor(test_set$XGBoost_Linear_GS,
    test_set$Strength), 4)
RMSE <- signif(RMSE(pred = test_set$XGBoost_Linear_GS,
    obs = test_set$Strength, na.rm = T), 6)
MAE <- signif(MAE(pred = test_set$XGBoost_Linear_GS,
    obs = test_set$Strength), 6)
GS_Test_Set_Statistics <- c(RMSE, MAE, R_squared)
```

3. Plot predicted vs observed values and residuals

```
XGBoost_Linear_GS_pred_obs <- predicted_observed_plot(
    predicted_val = test_set$XGBoost_Linear_GS,
    observed_val = test_set$Strength,
    residual_val = test_set$XGBoost_Linear_GS_residual,
    R_squared = R_squared, model_name = "Grid Search")
XGBoost_Linear_GS_residuals <- residuals_plot(
    predicted_val = test_set$XGBoost_Linear_GS,
    observed_val = test_set$Strength,
    residual_val = test_set$XGBoost_Linear_GS_residual,
    MAE = MAE, RMSE = RMSE, model_name = "Grid Search")
```

## Random Search

1. Make predictions on test set

```
test_set$XGBoost_Linear_RS <- predict(RS_XGBoost_Linear_model, test_set)
```

2. Calculate Residuals on test set

```
test_set$XGBoost_Linear_RS_residual <- test_set$Strength - test_set$XGBoost_Linear_RS
```

3. Calculate test set R-squared, RMSE, MAE

```
R_squared <- round(cor(test_set$XGBoost_Linear_RS,
```

```

    test_set$Strength), 4)
RMSE <- signif(RMSE(pred = test_set$XGBoost_Linear_RS,
          obs = test_set$Strength, na.rm = T), 6)
MAE <- signif(MAE(pred = test_set$XGBoost_Linear_RS,
          obs = test_set$Strength), 6)
RS_Test_Set_Statistics <- c(RMSE, MAE, R_squared)

```

4. Plot predicted vs observed values and residuals

```

XGBoost_Linear_RS_pred_obs <- predicted_observed_plot(
  predicted_val = test_set$XGBoost_Linear_RS,
  observed_val = test_set$Strength,
  residual_val = test_set$XGBoost_Linear_RS_residual,
  R_squared = R_squared, model_name = "Random Seach")
XGBoost_Linear_RS_residuals <- residuals_plot(
  predicted_val = test_set$XGBoost_Linear_RS,
  observed_val = test_set$Strength,
  residual_val = test_set$XGBoost_Linear_RS_residual,
  MAE = MAE, RMSE = RMSE, model_name = "Random Seach")

```

### **Genetic Algorithm (GA)**

1. Make predictions on test set

```

test_set$XGBoost_Linear_GA <- predict(GA_XGBoost_Linear_mo
del, test_set)

```

2. Calculate Residuals on test set

```

test_set$XGBoost_Linear_GA_residual <- test_set$Strength -
test_set$XGBoost_Linear_GA

```

3. Calculate test set R-squared, RMSE, MAE

```

R_squared <- round(cor(test_set$XGBoost_Linear_GA,
  test_set$Strength), 4)
RMSE <- signif(RMSE(pred = test_set$XGBoost_Linear_GA,
  obs = test_set$Strength, na.rm = T), 6)
MAE <- signif(MAE(pred = test_set$XGBoost_Linear_GA,
  obs = test_set$Strength), 6)

```

```

GA_Test_Set_Statistics <- c(RMSE, MAE, R_squared)

```

4. Plot predicted vs observed values and residuals

```

XGBoost_Linear_GA_pred_obs <- predicted_observed_plot(
  predicted_val = test_set$XGBoost_Linear_GA,

```

```

observed_val = test_set$Strength,
residual_val = test_set$XGBoost_Linear_GA_residual,
R_squared = R_squared,
model_name = "Genetic Algorithm")
XGBoost_Linear_GA_residuals <- residuals_plot(
  predicted_val = test_set$XGBoost_Linear_GA,
  observed_val = test_set$Strength,
  residual_val = test_set$XGBoost_Linear_GA_residual,
  MAE = MAE, RMSE = RMSE,
  model_name = "Genetic Algorithm")

```

## *Differential evolution (DE)*

1. Make predictions on test set

```

test_set$XGBoost_Linear_DE <- predict(
  DE_XGBoost_Linear_model, test_set)
# Calculate Residuals on test set
test_set$XGBoost_Linear_DE_residual <- test_set$Strength -
test_set$XGBoost_Linear_DE

```

2. Calculate test set R-squared, RMSE, MAE

```

R_squared <- round(cor(test_set$XGBoost_Linear_DE,
  test_set$Strength), 4)
RMSE <- signif(RMSE(pred = test_set$XGBoost_Linear_DE,
  obs = test_set$Strength, na.rm = T), 6)
MAE <- signif(MAE(pred = test_set$XGBoost_Linear_DE,
  obs = test_set$Strength), 6)
DE_Test_Set_Statistics <- c(RMSE, MAE, R_squared)

```

3. Plot predicted vs observed values and residuals

```

XGBoost_Linear_DE_pred_obs <- predicted_observed_plot(
  predicted_val = test_set$XGBoost_Linear_DE,
  observed_val = test_set$Strength,
  residual_val = test_set$XGBoost_Linear_DE_residual,
  R_squared = R_squared, model_name = "Differential Evolution")
XGBoost_Linear_DE_residuals <- residuals_plot(
  predicted_val = test_set$XGBoost_Linear_DE,
  observed_val = test_set$Strength,
  residual_val = test_set$XGBoost_Linear_DE_residual,
  MAE = MAE, RMSE = RMSE,

```

```
model_name = "Differential Evolution")
```

### Create summary table

```
Summary_Table_Test <- rbind(  
  GS_Test_Set_Statistics,  
  RS_Test_Set_Statistics,  
  GA_Test_Set_Statistics,  
  DE_Test_Set_Statistics,  
  DE_Test_Set_Statistics, deparse.level = 0  
) %>% data.frame()  
  
colnames(Summary_Table_Test) <- c("RMSE", "MAE",  
  "R-squared")  
Summary_Table_Test <- Summary_Table_Test %>% add_column(  
  Method = c("Grid Search", "Random Search",  
  "Genetic Algorithm", "Differential Evolution",  
  "Particle Swarm Optimization"), .before = 1)
```

Print table

### Test Set Statistics of XGBoost Models.

Based on test set statistics, the Differential Evolution obtained the lowest RMSE value (4.936).

```
g1 <- gridExtra::grid.arrange (XGBoost_Linear_GS_pred_obs,  
XGBoost_Linear_GS_residuals)  
g2 <- gridExtra::grid.arrange (XGBoost_Linear_RS_pred_obs,  
XGBoost_Linear_RS_residuals)  
g3 <- gridExtra::grid.arrange (XGBoost_Linear_GA_pred_obs,  
XGBoost_Linear_GA_residuals)  
g4 <- gridExtra::grid.arrange (XGBoost_Linear_DE_pred_obs,  
XGBoost_Linear_DE_residuals)
```

### Summary

The use of population-based search methods such as genetic algorithms, differential evolution and particle swarm optimization show some potential for finding the optimal values of hyperparameters of regression models.

The use of Differential Evolution resulted on the lowest test set RMSE value of all the methods tested.

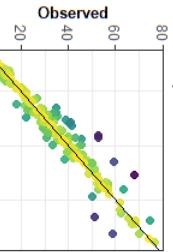
Due to the high computational demands, the use of these methods may be more suitable for models with several hyperparameters used on relatively small datasets and/or models that are easily trained. Performing a grid search across a wide range of values for beforehand may also help to narrow the limits of the constraints and make the overall process more efficient.

```
Summary_Table_Training <- bind_rows(  
  GS_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  RS_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  GA_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  DE_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5),  
  DE_XGBoost_Linear_model$results %>% arrange(RMSE) %>% .[  
 1,] %>% select(RMSE, MAE, Rsquared, nrounds, eta, lambda,  
 alpha) %>% round(5))
```

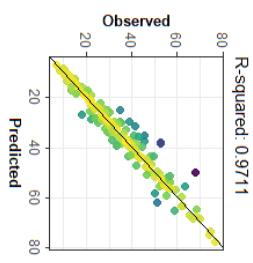
#### Test Set Statistics of XGBoost Models.

Method	RMSE	MAE	R-squared
Grid Search	3.31480	2.34397	0.9771
Random Search	3.94559	2.68917	0.9671
Genetic Algorithm	3.48790	2.34368	0.9647
Differential Evolution	3.87529	2.28620	0.9685
Particle Swarm Optimization	5.74515	2.52736	0.9685

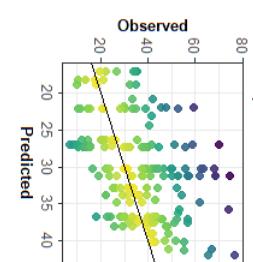
**Grid Search**  
Predicted vs Observed: Test Set  
R-squared: 0.9677



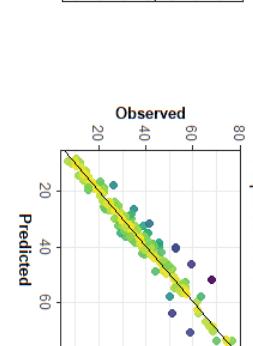
**Random Search**  
Predicted vs Observed: Test Set  
R-squared: 0.9711



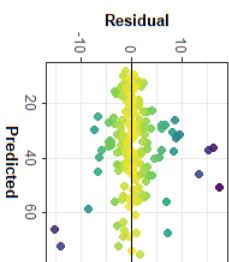
**Genetic Algorithm**  
Predicted vs Observed: Test Set  
R-squared: 0.4256



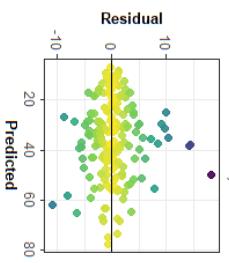
**Differential Evolution**  
Predicted vs Observed: Test Set  
R-squared: 0.9751



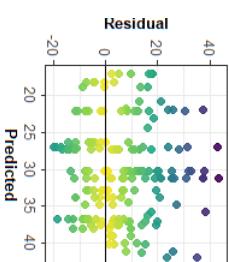
**Grid Search**  
Residuals: Test Set  
RMSE: 3.9252, MAE: 2.441



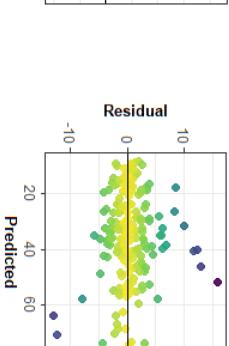
**Random Search**  
Residuals: Test Set  
RMSE: 3.70236, MAE: 2.458



**Genetic Algorithm**  
Residuals: Test Set  
RMSE: 14.9622, MAE: 11.73



**Differential Evolution**  
Residuals: Test Set  
RMSE: 3.46486, MAE: 2.286



## Exercises

In this chapter, we have used caret to perform hyperparameter tuning. We will introduce another package for doing this in Exercise 2.

1. Download the data ZIP files, application\_train.csv and application\_test.csv, from:  
[https://github.com/stricje1/VIT\\_University/tree/master/Predictive\\_Modeling/data/](https://github.com/stricje1/VIT_University/tree/master/Predictive_Modeling/data/)
  - a. Load application data (train, test) into R.
  - b. Convert categorical variables to numeric
  - c. Impute missing values with the median
  - d. Remove low variance variables
  - e. Split back to train/test
  - f. Perform grid search with caret
  - g. Generate predictions on best model
2. Import a sample binary outcome dataset into H2O

```
data <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
```

```
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")
```

- a. Open the H2O help using ??H2O
- b. Identify predictors and response
- c. For binary classification, response should be a factor
- d. Split data into train & validation [Hint: use h2o.splitFrame]
- e. Generate Gradient Boosting Machine (GBM) hyperparameters
- f. Train and validate a cartesian grid of GBMs [Hint: use h2o.grid]
- g. Get the grid results, sorted by validation AUC [Hint: use h2o.getGrid]

- h. Grab the top GBM model, chosen by validation AUC [Hint: use `h2o.getModel`]
- i. Now let's evaluate the model performance on a test set so we get an honest estimate of top model performance [Hint: use `h2o.performance`]
- j. Look at the hyperparamters for the best model

## References

- Altmann, A., Tolosi, L., Sander, O., & Lengauer, T. (2010). Permutation importance:a corrected feature importance measure. *Bioinformatics*. doi:10.1093/bioinformatics/btq134
- Amit, Y., & Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7), 1545–1588. doi:10.1162/neco.1997.9.7.1545
- Aslam, J. A., Popa, R. A., & Rivest, R. (2007). On Estimating the Size and Confidence of a Statistical Audit. *Proceedings of the Electronic Voting Technology Workshop (EVT '07)*. Boston, MA. Retrieved from <http://people.csail.mit.edu/rivest/pubs/APR07.pdf>
- Bailey, K. (1994). Numerical Taxonomy and Cluster Analysis. In *Typologies and Taxonomies* (p. 34).
- Barros, R. C., Cerri, R., Jaskowiak, P. A., & Carvalho, A. C. (2011). A bottom-up oblique decision tree induction algorithm. *Proceedings of the 11th International Conference on Intelligent Systems Design and Applications (ISDA 2011)* (pp. 450 - 456). IEEE. doi:10.1109/ISDA.2011.6121697
- Barros, R., Basgalupp, M., Carvalho, A., & Freitas, A. (2011). A Survey of Evolutionary Algorithms for Decision-Tree Induction. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 42(3), 291-312.
- BBC. (2016, July 26). France church attack: Priest killed by two 'IS militants'. *BBC News Online*. Retrieved from <https://www.bbc.com/news/world-europe-36892785>
- Bramer, M. (2007). *Principles of Data Mining*. London: Springer London. doi:10.1007/978-1-84628-766-4
- Breiman, L. (1994). *Bagging Predictors*. Technical Report No. 421, University of California, Department of Statistics, Berkeley. Retrieved from <http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf>
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123-140. Retrieved from

<http://link.springer.com/article/10.1023%2FA%3A1018054314>  
350

- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.  
doi:10.1023/A:1010933404324
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
- Breiman, L., H., F. J., Olshen, R., & Stone, C. (1984). *Classification and Regression*. Monterey, CA.: Wadsworth and Brooks. Retrieved from <ftp://ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps.Z>.
- Bryll, R. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 20(6), 1291–1302.
- 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
- Chipman, H., George, E., & McCulloch, R. (1998). Bayesian CART model search. *Journal of the American Statistical Association*, 93(443), 935-948.
- Criminisi, A., Shotton, J., & Konukoglu, E. (2011). Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Foundations and Trends in Computer Vision*, 7, 81–227.  
doi:10.1561/0600000035
- 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
- Deng, H., Runger, G., & Tuv, E. (2011). Bias of importance measures for multi-valued attributes and solutions. *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN)*, (pp. 293–300).
- Dietterich, T. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 139–157.

- Ehrlinger, J., Rajeswaran, J., & Blackstone, E. (2019, January 1). ggRandomForests: Exploring Random Forest Survival. *Statistical Methods in Medical Research*, 27(1), 126–141.
- 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
- Friedman, J. H. (1999). Stochastic Gradient Boosting. Retrieved from <https://statweb.stanford.edu/~jhf/ftp/stobst.pdf>
- Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning : Data mining, inference, and prediction*. New York: Springer Verlag.
- Ho, T. (1995). Random Decision Forest. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, (pp. 278–282). Montreal, QC. Retrieved from <http://cm.bell-labs.com/cm/cs/who/tkh/papers/odt.pdf>
- Ho, T. (1998). The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 832–844. doi:10.1109/34.709601
- Horváth, T., & Yamamoto, A. (2003). Inductive Logic Programming. In *Lecture Notes in Computer Science* (Vol. 2835). doi:10.1007/b13700
- Hothorn, T., Hornik, A., & Zeileis. (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. *Journal of Computational and Graphical Statistics*, 15(3), 651–674. doi:10.1198/106186006X133933
- Hyafil, L., & Rivest, R. (1976). Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, 5(1), 15–17. doi:10.1016/0020-0190(76)90095-8
- I., B.-G., A., D., N., S., & Singer, G. (2014). Efficient Construction of Decision Trees by the Dual Information Distance Method. *Quality Technology & Quantitative Management (QTQM)*, 11(1), 133-147. Retrieved from <http://www.eng.tau.ac.il/~bengal/DID.pdf>

- Ishwaran, H., Kogalur, U. B., Chen, X., & J., M. A. (2011). Random Survival Forests for High– Dimensional Data. *Statist. Anal. Data Mining*, 4, 115–132.
- Ishwaran, H., Kogalur, U., Gorodeski, E. Z., & Minn, A. J. (2010). High– Dimensional Variable Selection for Survival Data. *J. Amer. Statist. Assoc.*, 105, 205–217.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. New York: Springer.
- Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2), 119–127. doi:10.2307/2986296
- Kleinberg, E. (1996). An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition. *Annals of Statistics*, 24(6), 2319–2349. doi:10.1214/aos/1032181157
- Kuncheva, L., Rodríguez, J., Plumpton, C., Linden, D., & Johnston, S. (2010). Random subspace ensembles for fMRI classification. *IEEE Transactions on Medical Imaging*, 29(2), 531–542. Retrieved from <http://pages.bangor.ac.uk/~mas00a/papers/lkjrcpdlsjtm10.pdf>
- Liaw, A. (2012, October 16). *Documentation for R package randomForest*. Retrieved from The Comprehensive R Archive Network: <http://cran.r-project.org/web/packages/randomForest/randomForest.pdf>
- Lin, Y., & Jeon, Y. (2001). *Random forests and adaptive nearest neighbors*. Technical Report No. 1055, University of Wisconsin. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.9168>
- Papageorgiou, A. D. (2001). Breeding Decision Trees Using Evolutionary Techniques. *Proceedings of the Eighteenth International Conference on Machine Learning*, (pp. 393-400).
- Prinzie, A., & Van den Poel, D. (2008). Random Forests for multiclass classification: Random MultiNomial Logit". *Expert Systems with Applications*, 34(3), 1721–1732.

- Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1, 81-106.
- Rodriguez, J., Kuncheva, L., & Alonso, C. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10), 1619-1630.
- Rokach, L., & Maimon, O. (2005). Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 35(4), 476–487. doi:10.1109/TSMCC.2004.843247
- Rokach, L., & Maimon, O. (2008). *Data mining with decision trees: theory and applications*. World Scientific Pub Co Inc.
- Rousseeuw, P. J., & Leroy, A. M. (2003). *Robust Regression and Outlier Detection*. New York: Wiley.
- Sahu, A., Runger, G., & Apley, D. (2011). Image denoising with a multi-phase kernel principal component approach and an ensemble version. *IEEE Applied Imagery Pattern Recognition Workshop*, (pp. 1-7).
- Shinde, A., Sahu, A., Apley, D., & Runger, G. (2014). Preimages for Variation Patterns from Kernel PCA and Bagging. *IIE Transactions*, 46(5).
- 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
- Skurichina, M. (2002). Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis and Applications*, 5(2), 121–135. doi:10.1007/s100440200011
- Strobl, C., Malley, G., & Tutz. (2009). An Introduction to Recursive Partitioning: Rationale, Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests. *Psychological Methods*, 14(4), 323–348. doi:10.1037/a0016973
- Tan, J., & Dowe, D. (2004). MML Inference of Decision Graphs with Multi-Way Joins and Dynamic Attributes. Retrieved from [http://www.csse.monash.edu.au/~dld/Publications/2003/Tan+Dowe2003\\_MMLDecisionGraphs.pdf](http://www.csse.monash.edu.au/~dld/Publications/2003/Tan+Dowe2003_MMLDecisionGraphs.pdf)

- Tao, D. (2006). Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- The CLUSTER Procedure: Clustering Methods. (n.d.). In *SAS/STAT 9.2 Users Guide*. SAS Institute. Retrieved Retrieved 2014-12-26
- The DISTANCE Procedure: Proximity Measures. (n.d.). In *SAS/STAT 9.2 Users Guide*. SAS Institute. Retrieved Retrieved 2014-12-26
- Tolosi, L., & Lengauer, T. (2011). Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*. doi:10.1093/bioinformatics/btr300
- Tremblay, G. (2004). Optimizing Nearest Neighbour in Random Subspaces using a Multi-Objective Genetic Algorithm. *17th International Conference on Pattern Recognition*, (pp. 208–211).
- Tryon, R. C. (1939). *Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality*. Edwards Brothers.
- 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
- Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12), 1797-1808.
- 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>