ETH
Swiss Federal Institute of Technology Zurich

Seminar for Statistics

Department of Mathematics

Master Thesis Fall 2018

Christopher Salahub

Seen to be done:
A statistical investigation of peremptory challenge

Submission Date: March 3 2019

Co-Adviser	None
Adviser:	Prof. Dr. Marloes Maathuis

Preface

This work would be nowhere near as polished or complete without the effort of Prof. Dr. Marloes Maathuis to ensure I was performing analysis with a clear direction and purpose. I would like to thank her for finding time in her busy schedule to allow for weekly meetings. The group meetings organized by her Ph.D. student Marco Eigenmann were also critical in the development of more nuanced analysis and intuitive visualizations. I thank Marco Eigenmann for organizing them, and Jinzhou Li, Armin Fingerle, Sanzio Monti, and Qikun Xiang for attending my presentations and listening attentively. A special thanks is extended to Cédric Bleutler and Leonard Henckel, both of whom were especially engaged and participated in lengthy discussions both during and outside of the group meetings.

I would like to acknowledge in particular Prof. Dr. Tilman Altwicker for his detailed suggestions on where to look for more legal context on the topic and Prof. Dr. Samuel Baumgartner for his research suggestions. They were very important at providing the necessary information to begin a first investigation of the topic. Of course, without the cooperation of Dr. Ronald Wright, Dr. George Woodworth, Dr. Barbara O'Brien, and Dr. Catherine Grosso for generously providing me with the data from their investigations on the subject of peremptory challenge. Without this data, the visualizations and modelling presented here simply would not have been possible, and so I am exceptionally grateful that they were so enthusiastic to share the fruits of their labour to help cultivate mine.

Abstract

Short summary of my thesis.

Contents

List of Figures

List of Tables

Notation and Terms

In order to facilitate clarity despite brevity, a list of terms used in this paper is presented here.

Prosecution/State The legal representation which argues for conviction

Defence The legal reperesentation which argues against conviction

Court Reference to the judge, prosecution, and defence

Venire The population sample from which a jury is selected

Jury The final group of (usually) twelve chosen venire members which judge the guilt or innocence of the accused/defendant

Accused/Defendant The individual on trial for a crime

Voir dire From old French "to speak the truth", this is the questioning process used by the court to assess the suitability of a venire member to sit on the jury

Struck In the context of a venire member being rejected from the jury, struck indicates removal by peremptory challenge or challenge with cause

Litigants The accusor and the accused

Chapter 1

Introduction

The Gerald Stanley murder trial was noteworthy for all of the wrong reasons. The first reason was the crime itself. The rural region around Biggar, Saskatchewan (Quenneville (2018)) is not known for crime, indeed, the crime statistics collected by Statistics Canada suggest it is one of the safest in the province (Statistics Canada (2018)). Any murder at all would be worthy of attention and subject to plenty of drama. But beyond the damage this trial has done to the community, this trial is noteworthy because it led to a significant re-examination of the legal jurisprudence surrounding the jury selection process culminating in the proposition of Bill C-75 by the Canadian government in March of 2018 (42nd Parliament of Canada (2018a)), less than two months after the trial's verdict (Quenneville and Warick (2018)).

Bill C-75, in part, aims to ameliorate one of the critical points of contention about the Gerald Stanley case: the use of peremptory challenges in jury selection. The outsized impact of the case was due, in large part, to it's racial aspect. Gerald Stanley, a white man, was accused of second degree murder in the killing of Colten Boushie, a First Nations man. Given Canada's troubled history with First Nations groups, this alone would have been enough to make the trial a flash point for race issues, but that was not the worst aspect of the trial. Rather, it was the alleged use of peremptory challenges to strike five potential jurors who "appeared" to be First Nations, resulting in an all-white jury, that proved to be the most controversial and influential facet of the entire affair (Harris (2018), MacLean (2018)).

With Bill C-75 currently moving through the Canadian parliamentary system, having completed its second reading in June 2018 (42nd Parliament of Canada (2018b)), a close re-examination of the practice of peremptory challenge is warranted. A great deal of ink has already been spilled on both sides of the debate (see Hasan (2018), Zinchuk (2018), and Roach (2018)), but startlingly little of this discussion has been based on any hard evidence on the impact of peremptory challenge in jury selection. This paper aims to provide analysis and evidence to illuminate the topic further by analyzing three separate peremptory challenge data sets collected in the United States, namely Wright, Chavis, and Parks (2018), Grosso and O'Brien (2012), and Baldus, Woodworth, Zuckerman, and Weiner (2001). While this data cannot tell us if challenges were racially motivated in the Stanley trial, stepping back from this fraught legal episode to take a wider view of the practice of peremptory challenge provides a more sober place to start the discussion of its place in modern jury trials.

This paper will proceed in five parts. Chapter 2 provides a brief history of the practice of peremptory challenges in jury trials, in particular explaining their original motivation, past implementations, and how they have developed in the United States, the United Kingdom, and Canada. Chapter 3 proceeds to discuss the three data sets obtained, explaining the sources and collection methods before detailing the cleaning and preprocessing. Chapter **??** then provides the details and results of the analysis performed on the different data sets. It begins discussing the Jury Sunshine data set, which was used as a 'test' set of sorts, where analysis could be flexibly performed before the final analysis methods were turned to the other two data sets. The results of this analysis are compared to previous works in Chapter **??**. Finally, the results and findings are summarized in **??**, and recommendations based on the observations obtained here are provided.

Chapter 2

Peremptory Challenges

As the focus of this text is the legal practice of peremptory challenges, and these are a specific practice which may not be known in detail to the reader, a brief exploration of their history, motivation, and current use is presented here. It is not meant to be exhaustive, but rather to provide context and references for an interested and motivated reader to learn more. Indeed, many details have been omitted from the summary of the history here. Roughly, the presentation of the history of jury trials follows the comprehensive and exhaustively referenced description provided by Hoffman (1997), with additional context and opinion on certain details provided by von Moschzisker (1921), Forsyth (1994), and Brown (2000). Information regarding the history of the Canadian system was provided by Brown (2000) and Petersen (1993).

Before reviewing the history, it is best to give some context. The central and unchanging function of a jury in a jury trial system is to judge the innocence or guilt of an accused in light of evidence. As discussed in von Moschzisker (1921) and Forsyth (1994), the expectation of how this act is perform has varied throughout history. In the distant past, von Moschzisker (1921) and Hoffman (1997) report that the central function of the jury was to collect evidence, and so they assumed the role commonly performed by modern police detectives, and so the selection of the most "trustworthy" individuals of some reknown was paramount. This is contrasted with the modern jury, which performs no collection of the evidence, but instead merely judges the guilt of the accused, and is meant to be composed of a panel of peers or "equals" sampled at random from the population, an idea markedly different from, but motivated by, the Magna Carta (see Davis (1963) and Hoffman (1997)).

Peremptory challenges are a departure from this random selection. They are a privileged removal of a venire member - to be replaced by a new randomly selected venire member - by either the prosecution or defence without providing a justification to the court. The modern motivation for this was best described by Justice Byron R. White in Supreme Court of the United States (1965):

> The function of the challenge is not only to eliminate extremes of partiality on both sides, but to assure the parties that the jurors before whom they try the case will decide on the basis of the evidence placed before them, and not otherwise. In this way, the peremptory satisfies the rule that, "to perform its high function in the best way, justice must satisfy the appearance of justice."

This suggests two modern justifications for the peremptory challenge. The first is that of removing venire members with “extreme” bias, and the second is the creation of a jury which is composed of jurors mutually acceptable to both the defense and the prosecution. Those who defended the practice of peremptory challenges in Canada after the Gerald Stanley trial, including Hasan (2018) and Macnab (2018), seem to use this defence or some variant of it to argue in favour of keeping the practice.

2.1 Modern Practice

This broad overview, however, leaves a significant amount of detail unexplained. One might legitimately wonder how the venire is randomly chosen, what limits are placed on the use of peremptory challenges, and what the actual process looks like in a courtroom setting. Of course, these details are where there are significant differences between the implementation of the practice in court systems around the English-speaking world.

2.1.1 In American Law

The practice of peremptory challenges in the United States of America is a heavily restricted. As discussed by Page (2005), the Supreme Court of the United States (1986) decided in 1986 to

2.1.2 In Canadian Law

2.1.3 The Gerald Stanley Trial

Whatever justification for the modern and historical practice of peremptory challenges a proponent espouses, it should be clear from the fallout of the Gerald Stanley trial that peremptory challenges have failed in this case. Rather than guaranteeing the creation of a mutually acceptable jury, their use inspired an atagonistic response to the verdict of the case. In the eyes of many, justice was not done, and peremptory challenges are the specific reason cited.

While

Every time a prospective juror is peremptorily challenged we are telling that prospective juror that the foundation of this system is not evidence, but rather rumor, innuendo, and prejudice. - Morris B. Hoffman Hoffman (1997)

2.2 History

2.2.1 Pre-English History

Although precise timelines are hard to establish, there is evidence that jury trials have occurred in some form or another since antiquity. The concept, that of judgement by a group of peers, is so ancient that it is prevalent not only in historical records, but in myth. As Hoffman (1997) indicates, both Norse and Greek mythology feature groups of individuals assessing the guilt or collecting evidence about the actions of a peer.

Outside of the realm of myth, Hoffman (1997) reports on evidence of the use of juries in Ancient Egypt, Mycenae, Druid England, Greece, Rome, Viking Scandanavia, the Holy Roman Empire, and Saracen Jerusalem. It should be noted that in none of these areas was the jury trial the primary form of conflict resolution practiced. Nonetheless, it is clear the jury trial has a broad and long history of use.

Something similar to the modern peremptory challenge does not appear until Rome, however. The Roman *Judices* were groups of senators selected to judge the guilt of the accused in a legal case. Hoffman (1997) presents evidence of the selection of 81 Senators to sit on one of these *Judices*, after which the litigants were permitted to remove fifteen of these Senators each. This egalitarian reduction of the jury size seems analogous to the modern peremptory challenge system in placing the power of removal with the litigant and suggesting no justification is necessary for their removal.

2.2.2 In English Law (1066–1988)

Peremptory challenge did not reach is modern form, as outlined above, until it was established in the English legal system. It should be noted that despite some previous debate on the topic, the most modern historical evidence suggests that the basis of the English practice was not related to the system used in the selection of *Judices* in Rome. The English system appears to be its own beast entirely.

The dominant historical interpretation is presented by von Moschzisker (1921) and Hoffman (1997): that the jury system was introduced to England during the Norman conquest of 1066 by William the Conqueror. The practice, however, was not made official until the Assize of Clarendon in 1166 by Henry II, and it was not until the outlaw of trials by ordeal (the most common method of trial at that time) in 1215, that peremptory challenges began to appear in England in the late thirteenth century. The challenges were officially recognized in 1305 when Parliament outlawed their use by the Crown, only to replace them with an analogous system of so-called "standing-aside"[1].

It should be noted here that although the challenges issued between the Assize of Clarendon and this 1305 act are called "peremptory," they may not have served the same purpose, nor the same justification, as the modern challenges. Indeed, as Hoffman (1997) argues convincingly, these challenges may have been closer to modern challenges with cause. Two plausible explanations are provided: smaller communities in which the venire members would be familiar to all members of the court, and the paradigm of royal infallibility which was present at the time.

While the first of these, where both sides simply "know" why an individual is being challenged is difficult to verify, the second justification for the Crown's use of challenges seems quite reasonable. If the king cannot be wrong in his judgement and he has some reason to feel that a venire member cannot serve on the jury, then it would be highly disrespectful to ask him to justify his action. The Crown prosecutors, as representatives of the king, would be similarly shielded from criticism.

Additionally, this is supported by the abolition of their royal use in 1305, the language of which suggests that peremptory challenges were originally the privilege of the Crown, with none being granted to the defence. As royal infallibilty grew out of favour, peremptory

[1] For a detailed explanation of this system see Hoffman (1997) and Brown (2000)

challenges seem to have been granted to the defence as an act done out of a desire to limit the power and special privileges of the monarch.

Whatever the logic of the expansion of these challenges, their legal limits are recorded more precisely. From a maximum of 35 challenges allowed at their peak in the fourteenth century, the allowed number of challenges has only decreased. This culminated in the Cyprus spy case in the late 1970s, which led to a "sustained campaign in Parliament and in the press alleging that defence counsel were systematically abusing it" (see Hoffman (1997)). Ultimately this campaign was settled by the Criminal Justice Act of 1988, in which the Parliament of the United Kingdom abolished the practice. It did not, however, abolish the use of "standing-aside" by the Crown, although the practice has been heavily curtailed with strict guidelines to its use, which are limited to national security trials as outlined by Attorney General's Office of the United Kingdom (2012).

2.2.3 In American Law (ca. 1700–1986)

2.2.4 In Canadian Law (1867–1988)

Chapter 3

Data

3.1 Jury Sunshine Project

3.2 North Carolina Data

3.3 Philadelphia Data

3.4 Data Cleaning

3.4.1 Sunshine Data

The data collected in North Carolina proved invaluable to this project Wright et al. (2018).

Problem: some columns of the data contained only NA values Solution: `lapply` to remove these uninformative columns

Problem: relational database provided did not have all data in one joined table Solution: creation of `CleaningMerge` function: a wrapper for `merge` which provides information about the mismatches which may be present in the two merged tables

Problem: inconsistently coded levels, e.g. inconsistent case or "?" instead of "U" for unknowns Solution: forcing levels to be uppercase and the replacement of obvious misspecified levels

Problem: some columns seem to have swapped values, e.g. the gender column should be one of "M", "F", or "U" and the political affiliation column should be one of "D", "R", "I", or "U", but some individuals have the gender recorded as "R" and political affiliation as "M" Solution: the creation of the `IdentifySwap` function, which has two arguments: a data set and the acceptable or correct levels for the variables in the data set. It then identifies rows which have candidate swaps and presents them for review

Chapter 4

Analysis

With this data cleaned and processed, questions can now be posed and addressed through analysis. A few obvious questions come to mind, considering the previous work done on this subject. The first is whether the results found by previous analyses which did not use statistics are statistically significant. Additionally, we may wonder whether the most common arguments posed in favour of peremptory challenge are satisfied in this data.

4.1 Arguments for Peremptory Challenges

The primary argument stated in favour of the continued use of peremptory challenges is that of the 'levelling' of the bias of the jury. Cite Canadian news articles here The argument, essentially states that peremptory challenges are necesary to remove those jurors which are somehow abnormally biased but which are not eligible for removal by cause, or have not been removed by cause out of error.

While the argument of recourse for an incorrect judgement of a challenge with cause is certainly valid, it seems unnecessary in light of the appeals process which already exists. Regardless, a precise comment on the validity of this statement cannot be easily made. The second assertion, however, permits a precise and straightforward mathematical analysis.

In the Jury Sunshine data, for example, the proportion of venire members rejected by peremptory challenge is roughly 0.43. In what sense can such a large proportion of the venire be judged as extreme?

A secondary argument is that of the creation of a jury which is mutually acceptable by giving both sides the privilege of removing any jurors they do not want assessing their case. The multiple American supreme court cases which address peremptory challenges and the outcome of the Gerald Stanley murder trial demonstrate quite clearly that this noble goal is not executed in practice. Rather, the privilege is a point of weakness that allows the politicization cases by doing the precise opposite of what it purports to achieve. It creates juries which are unacceptable to one party and society at large.

4.2 Modelling

In order to create a single model to test the statistical significance of the differences observed for strike rates by race, defendant race, and party doing the striking, a saturated poisson regression model was fit to the data. Letting i denote the level of the venire member race, j the defendant race, and k the disposition, the numbers of observed venire members in each ijk combination, y_{ijk} were modelled as Poisson-distributed random variables with expectation λ_{ijk}. A saturated model was then fit to the data, that is a model described by the equation:

$$\log E[y_{ijk}] = \mathbf{x}_{ijk}\beta = \beta_o + \beta_R x_{i..} + \beta_D x_{.j.} + \beta_S x_{..k} + \beta_{R:D} x_{i..} x_{.j.} + \beta_{R:S} x_{i..} x_{..k} + \beta_{D:S} x_{.j.} x_{..k} + \beta_{R:D:S} x_{i..} x_{.j.} x_{..k} \quad (4.2.0.1)$$

Where $x_{i..}$ indicates the race level of the ijk cell, and $x_{.j.}, x_{..k}$ are defined analogously for the defendant race and disposition. The interaction terms then serve to answer questions about the racial pattern of strikes which is utilized by each party given the defendant race. Most interesting to this investigation is the third order interaction term. This term indicates a significant difference in racial strike patterns given the party striking and the defendant race. In other words, this term accounts for different patterns for the different parties which are not independant of the defendant race.

When this term is tested using a nested model without the third order interaction, the third order interaction is found to be significant. This suggests that not only do the patterns present in the different parties vary, but they vary differently for different defendant races. This dependence can be viewed using a novel graphic presented in Figure 4.1.

The conditional probability of a particular disposition given the racial combination of venire person and defendant is displayed on the y-axis, that is the count of individuals for a particular race, defendant race, and disposition combination divided by the number of individuals with the racial combination across all dispositions. The x-axis then displays the combinations, grouped by the venire member race to show the dominant pattern in the data.

The black line running across the plot is the mean, or expected, rejection probability that all parties would have if they acted identically. That is, the relative level of this line provides the relative strike rate on aggregate for a particular racial combination. The bars extending from this line at each point go from this line to the corresponding value of the party represented by the bar. Finally, the horizontal lines provide approximate confidence intervals for each combination[1].

The dominant pattern to these strikes is a tendency of the defense to preferentially reject white venire members and keep black venire members, and of the prosecution to do the opposite. It was already noted in the literatureWright et al. (2018), but the addition of defendant race allows us to make a stronger statement, as this pattern remains across defendant races. It also adds nuance, however, as the race of the defendant has a clear impact on the lengths of the bars for both the defense and prosecution. The prosecution

[1] Generated assuming a binomial distribution of struck (by any party) against kept, as when this data is modelled with a poisson distribution, the distribution of sub-processes given the overall count will be binomially distributed

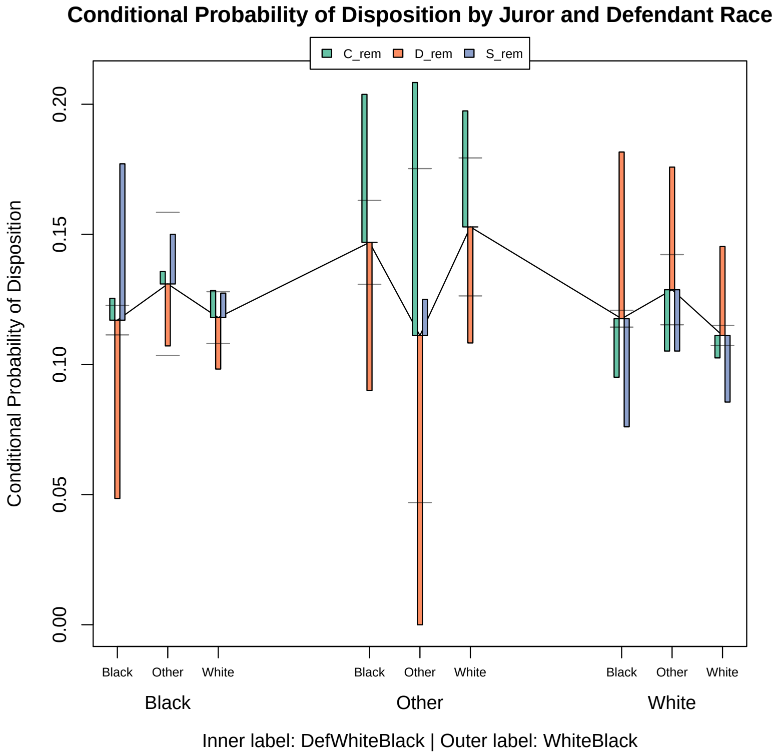

Figure 4.1: Parallel coordinate plot of racial strike tendencies

seems to favour a jury which does not match the race of the defendant, while the defense seems to favour a jury which does.

While this second tendency seems to have no justification beyond race, the dominant tendency may have other justification than simply skin colour. As was noted by "Ideological Imbalance and Peremptory Challenge", black individuals are more consistently aligned with the democratic party, and as a consequence a lawyer which suspects this political bias will impact the trial outcome would preferentially strike or keep black jurors in order to keep as many left wing individuals as possible. In this data, this political imbalance is incredibly prevalent, as can be seen in Figure 4.2 Add the plot of this effect here, elaborate on this pattern more based on the plot.

Perhaps more interestingly, the prosecution and judge seem to match in their tendency from the mean at every combination. This suggests that both challenges with cause and the prosecution tend to have the same effect on the jury composition, though the magnitudes can differ greatly for these two strikes. An immediate explanation to this is offered by Hans and Vidmar (1986), who outline, on pages 69-70, the skill and tact required to effectively propose challenges with cause. In order to determine an individual's bias, it is frequently the case that a direct question will fail to garner an honest reponse due to social pressures. As a consequence, the questions asked of venire members must be carefully presented.

Using this as a motivation, an obvious possible explanation for the challenges with cause is that the prosecution is simply more experienced on average than the defence. To determine the veracity of this claim, the year licensed for each lawyer was subtracted from the outcome date of each trial. The resulting distribution of years of experience was then

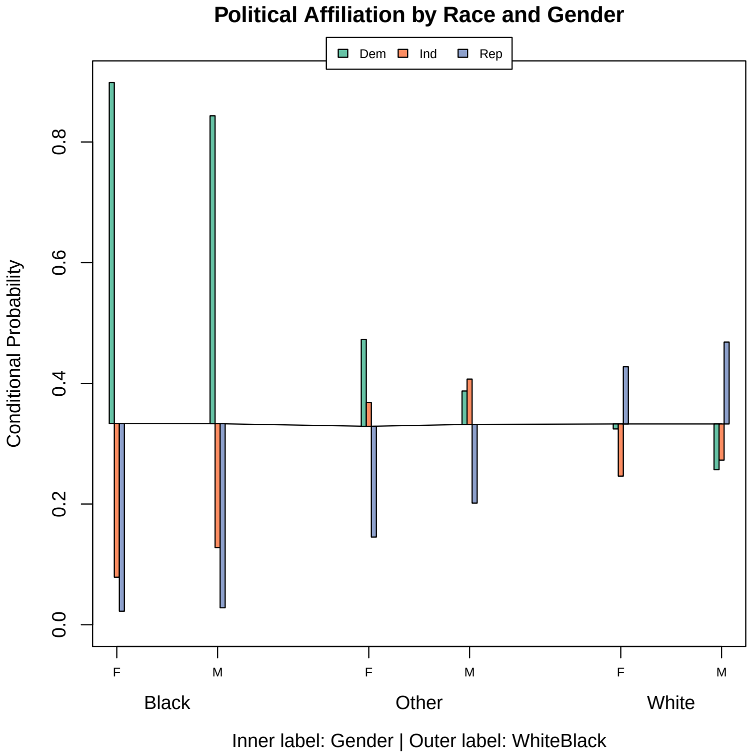

Figure 4.2: Conditional probabilities of political affiliation by race and gender

plotted in back-to-back histograms as shown in Figure 4.3.

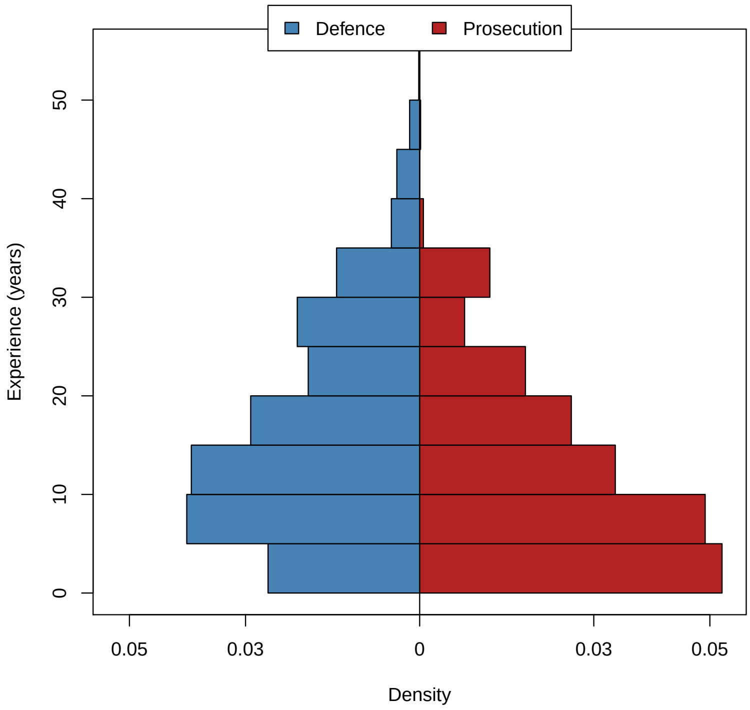

Figure 4.3: Distributions of lawyer experience for prosecutors and defence attorneys

4.3 To include a picture

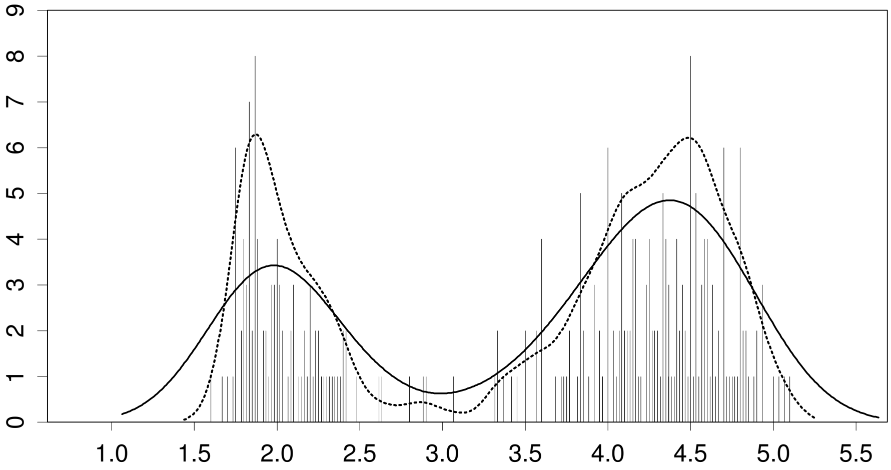

Figure 4.4: Old Faithful Geyser eruption lengths, $n = 272$; binned data and two (Gaussian) kernel density estimates ($\times 10$) with $h = h^* = .3348$ and $h = .1$ (dotted).

Or also with `includegraphics`:

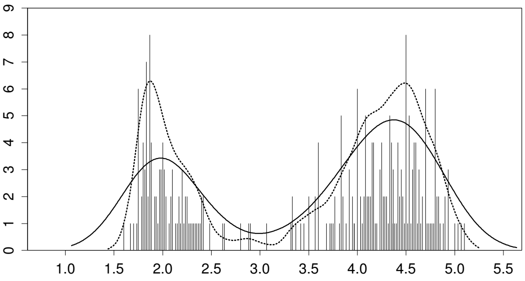

Figure 4.5: Old Faithful Geyser eruption lengths, $n = 272$; binned data and two (Gaussian) kernel density estimates ($\times 10$) with $h = h^* = .3348$ and $h = .1$ (dotted).

4.4 To make a proof

Proof. $1 + 1 = 2$ □

4.5 To include R code

See information in Appendix A.

4.6 Other information

Put a text between quotes: make sure to use nice quotes, such as “quote”.

Cite a document in the bibliography (an example here): Author and Author (tion). Or mention that Hampel (a person) or Stahel and Weisberg (two persons) have already done quite a bit work.

Referencing a different part of your work: please refer to Appendix A.

Chapter 5

Summary

Summarize the presented work. Why is it useful to the research field or institute?

5.1 Future Work

Possible ways to extend the work.

Bibliography

42nd Parliament of Canada (2018a, March). Bill C-75: An Act to Amend the Criminal Code, Youth Criminal Justice Act and other Acts and to make consequential amendments to other Acts. http://www.justice.gc.ca/eng/csj-sjc/pl/charter-charte/c75.html.

42nd Parliament of Canada (2018b, November). Bill C75. LEGISinfo. http://www.parl.ca/LegisInfo.

Attorney General's Office of the United Kingdom (2012, November). Jury vetting right of stand-by guidelines. https://www.gov.uk/guidance/jury-vetting-right-of-stand-by-guidelines–2.

Author, F. and S. Author (year of publication). Title of the article. *Journal where the article has been published volume of the journal*(issue number), firstpage–lastpage.

Baldus, D. C., G. Woodworth, D. Zuckerman, and N. A. Weiner (2001). The Use of Peremptory Challenges in Capital Murder Trials: A Legal and Empirical Analysis. *University of Pennsylvania Journal of Consitutional Law 3*(1).

Brown, R. B. (2000). Challenges for cause, stand-asides, and peremptory challenges in the nineteenth century. *Osgoode Hall Law Journal 38*(3), 453–494.

Davis, G. (1963). *Magna Carta*. London: British Museum.

Forsyth, W. (1994). *History of Trial by Jury* (2 ed.). Lawbook Exchange.

Grosso, C. M. and B. O'Brien (2012). A Stubborn Legacy: The Overwhelming Importance of Race in Jury Selection in 173 Post-Batson North Carolina Capital Trials. *Iowa Law Review 97*, 1531.

Hampel, F. R. (1985). The breakdown points of the mean combined with some rejection rules. *Technometrics 27*(2), 95–107.

Hans, V. P. and N. Vidmar (1986). *Judging the Jury* (1 ed.). Plenum Press.

Harris, K. (2018, February). Liberals review jury selection process after Boushie case uproar. CBC News. https://www.cbc.ca/news/politics/jury-selection-diversity-indigenous-1.4531792.

Hasan, N. R. (2018, April). Eliminating peremptory challenges makes trials less fair. The Star. https://www.thestar.com/opinion/contributors/2018/04/10/eliminating-peremptory-challenges-make-trials-less-fair.html.

Hoffman, M. B. (1997). Peremptory Challenges Should Be Abolished: A Trial Judge's Perspective. *The University of Chicago Law Review 64*(3), 809.

MacLean, C. (2018, February). Gerald Stanley acquittal renews calls for justice reform 27 years after Manitoba inquiry. CBC News. https://www.cbc.ca/news/canada/manitoba/aboriginal-justice-inquiry-colten-boushie-gerald-stanley-jury-1.4532394.

Macnab, A. (2018, February). Stanley acquittal should not lead to scrapping peremptory challenges, say criminal lawyers. Canadian Lawyer. https://www.canadianlawyermag.com/legalfeeds/author/aidan-macnab/stanley-acquittal-should-not-lead-to-scrapping-peremptory-challenges-say-criminal-lawyers-15332/.

Page, A. (2005). Batson's blind spot: Unconscious stereotyping and the peremptory challenge. *Boston University Law Review 85*, 155.

Petersen, C. (1993). Insitutionalized racism: The need for reform of the criminal jury selection process. *McGill Law Journal 38*(1).

Quenneville, G. (2018, February). What happened on Gerald Stanley's farm the day Colten Boushie was shot, as told by witnesses. CBC News. https://www.cbc.ca/news/canada/saskatoon/what-happened-stanley-farm-boushie-shot-witnesses-colten-gerald-1.4520214.

Quenneville, G. and J. Warick (2018, February). Shouts of 'murderer' in courtroom after Gerald Stanley acquitted in Colten Boushie shooting. CBC News. https://www.cbc.ca/news/canada/saskatoon/gerald-stanley-colten-boushie-verdict-1.4526313.

Roach, K. (2018, April). Ending peremptory challenges in jury selection is a good first step. The Ottawa Citizen. https://ottawacitizen.com/opinion/columnists/roach-ending-peremptory-challenges-in-jury-selection-is-a-good-first-step.

Stahel, W. and S. Weisberg (1991). *Directions in Robust Statistics and Diagnostics, 2 vol.* N. Y.: Springer-Verlag.

Statistics Canada (2018, November). Table 35-10-0061-01: Crime severity index and weighted clearance rates, police services in Saskatchewan.

Supreme Court of the United States (1965). Swain v. Alabama. Accessed: https://supreme.justia.com/cases/federal/us/380/202/.

Supreme Court of the United States (1986). Batson v. Kentucky. Accessed: https://www.law.cornell.edu/supremecourt/text/476/79.

von Moschzisker, R. (1921). The historic origin of trial by jury. *University of Pennsylvania Law Review 70*(1).

Wright, R. F., K. Chavis, and G. S. Parks (2018, October). The Jury Sunshine Project: Jury Selection Data as a Political Issue. *University of Illinois Law Review 2018*(4), 1407.

Zinchuk, B. (2018, March). Both sides wrong about Stanley trial. Prince George Citizen. https://www.princegeorgecitizen.com/opinion/editorial/both-sides-wrong-about-stanley-trial-1.23199321.

Appendix A

Complementary information

Additional material. For example long mathematical derivations could be given in the appendix. Or you could include part of your code that is needed in printed form. You can add several Appendices to your thesis (as you can include several chapters in the main part of your work).

A.1 Including R code with verbatim

A simple (rather too simple, see **??**) way to include code or *R* output is to use `verbatim`. It just prints the text however it is (including all spaces, “strange” symbols,...) in a slightly different font.

```
## loading packages
library(RBGL)
library(Rgraphviz)
library(boot)

## global variables
X_MAX <- 150

   This allows me to put as many s  p a   c es   as I want.
I can also use \ and ‘ and & and all the rest that is usually only
accepted in the math mode.

I can also make as
                  many
             line
    breaks as
I want... and
             where I want.
```

A.2 Data Processing Code

However, it is much nicer to use the *listings* package to include R code in your report. It allows you to number the lines, color the comments differently than the code, and so on.

```
########################################

## THESIS DATA PROCESSING SCRIPT
## Christopher Salahub
## Sept 26, 2018

########################################

## PACKAGES ###########################
library(readxl)
library(tm)
library(stringr)


## CONSTANTS ##########################

## start by defining file locations
ThesisDir ← "c:/Users/Chris/Documents/ETH Zurich/Thesis/Data"
SunshineFile ← paste0(ThesisDir, "/JurySunshineExcel.xlsx")
SunshineSheets ← excel_sheets(SunshineFile)

NorthCarFile ← paste0(ThesisDir,
                        "/Jury Study Data and Materials/NC Jury Selection Study Database6 Dec 2011.csv")

PhillyFile ← paste0(ThesisDir,
                      "/Voir Dire Data & Codebook/capital_venires.csv")

## next the factor level codes as given in the codebook and regularized here
## regularization: - political affiliation "N" replaced with "I" for all entries
LevRace ← sort(c("A","B","H","N","O","U","W"))
LevGen ← sort(c("F","M","U"))
LevPol ← sort(c("D","L","R","I","U"))

## create a charge tree with regex nodes to identify and clean charge text
chargeTree ← list("rape" = list("statutory", "first|1", "second|2"), "sex(?=.* offense)" = list("first|1", "second|2"),
                   "sex(?=.*offend)" = list("regis", "addr"), "murder" = list(" first|1" = list("att"), "second|2" = list("att")),
                   "arson", "firearm" = list("pos", "disch"), "stole" = list("pos "),
                   "mari" = list("pos", "sell|sale", "man", "pwimsd"), "coca" = list("pos", "sell|sale", "man", "pwimsd"),
                   "cs" = list("pos", "sell|sale", "man", "pwimsd"), "hero" = list("pos", "sell|sale", "man", "pwimsd"),
                   "meth" = list("pos", "sell|sale", "man", "pwimsd"),
                   "oxycod" = list("pos", "sell|sale", "man", "pwimsd"), "mass" = list("pos"), "break" = list("enter"),
                   "assa" = list("serious bodily", "female", "strangul", "deadly" , "official"),
                   "larceny" = list("motor", "felon", "merchant"), "false" = list ("pretense"),
                   "driving" = list("impaired"), "kidnap" = list("first|1", " second|2"),
                   "robb" = list("dang"), "burg" = list("first|1", "second|2"), " indec" = list("liber"),
                   "embez", "manslaughter" = list("inv"), "flee" = list("arrest") ,
                   "abuse|cruelty" = list("child", "anim"), "identity" = list(" theft"))

## create a list of variables which can sensibly be summarized by trial
TrialVars ← c("TrialNumberID", "DateOutcome", "JudgeID", "DefAttyType", "
    VictimName",
               "VictimRace", "VictimGender", "CrimeLocation", "PropertyType",
               "ZipCode.Trials", "StateTotalRemoved", "DefenseTotalRemoved",
               "CourtTotalRemoved", "JDistrict", "JName", "JRace", "JGender",
               "JPoliticalAff", "JVoterRegYr", "JYrApptd", "JResCity", "JResZip",
               "ChargeTxt", "Outcome", "Sentence.FullSunshine", "DefendantID.FullSunshine",
               "DefendantID.DefendantToTrial", "DefRace", "DefGender", "DefDOB", "DefAttyID",
               "DefAttyName", "DCRace", "DCGender", "DCPoliticalAff", "DCYrRegVote",
               "DCYrLicensed", "DCResideCity", "DCResideZip", "ProsecutorID", "ProsName",
               "ProsRace", "ProsGender", "ProsPoliticalAff", "PYrRegVote", "PYrLicensed",
               "PResideCity", "PResideZip", "Guilty", "CrimeType", "DefWhiteBlack")


## FUNCTIONS ###########################

## Loading and cleaning ################
## create a descriptive merge function for cleaning (essentially a 'merge' wrapper)
CleaningMerge ← function(x, y, ...) {
    ## start by creating the merge
    ## first match arguments
    MatchCall ← match.call(merge)
    MatchCall[[1]] ← quote(merge)
    ## get input names and ensure proper name structure
    xname ← MatchCall$x
    if (!is.symbol(xname)) xname ← as.symbol(paste0(xname[[2]],xname[[3]]))
    yname ← MatchCall$y
    if (!is.symbol(yname)) yname ← as.symbol(paste0(yname[[2]],yname[[3]]))
    ## use this to extract suffixes and fix MatchCall
    MatchCall$suffixes ← paste0(".", c(xname, yname))
    MatchCall$x ← xname
    MatchCall$y ← yname
    ## specify that the match should be an outer join
    MatchCall$all ← TRUE
    ## and use this to make a clean local assignment to modify
    assign(as.character(xname), cbind(x, Diag.x = 1), envir = environment())
    assign(as.character(yname), cbind(y, Diag.y = 1), envir = environment())
    ## now evaluate the call
    Merged ← eval(MatchCall, envir = environment())
    ## next perform some checks
    xExpInds ← is.na(Merged$Diag.x)
    yExpInds ← is.na(Merged$Diag.y)
    ## remove the diagnostic columns
    Merged$Diag.x ← NULL; Merged$Diag.y ← NULL
    ## summarize the diagnostic checks
    X_nexp ← sum(xExpInds)
    Y_nexp ← sum(yExpInds)
    X_missing ← Merged[xExpInds,]
    Y_missing ← Merged[yExpInds,]
    ## print the diagnostics
    cat("Joined ", paste(xname, yname, sep = " and "), " with ",
        X_nexp, " and ", Y_nexp, " failed matches respectively \n", sep = "")
    ## return the results, preferentially keeping the data which is present in x but missing from y
    if (X_nexp == 0 & Y_nexp == 0) {
        Merged
    } else list(Merge = Merged[!xExpInds,], Xfails = X_missing, Yfails = Y_missing)
}

## a function to identify and perform swaps with user input
SimpleSwapper ← function(data, CorrectLevs, auto = FALSE) {
    ## first match the data to the columns of interest
    colInds ← match(names(CorrectLevs), names(data))
    ## extract the levels of the columns of interest to check if there are any
        potential swaps
    swapCheck ← all(sapply(1:length(colInds),
                            function(ind) identical(sort(levels(as.factor(data[,
                                colInds[ind]]))),
                                                    sort(CorrectLevs[[ind]]))))
    ## if no swaps are present end this check
    if (swapCheck) {
        cat("No errors found, exiting.")
        return(data)
    }
    ## if errors are found, further investigate them
    ## identify potential rows
    ## first those which have elements out of place
    SwapPoss ← sapply(1:length(colInds),
                       function(ind) !(data[,colInds[ind]] %in% CorrectLevs[[ind
                           ]]))
    ## now rows containing unknown entries
    Unknown ← sapply(1:length(colInds),
                      function(ind) data[,colInds[ind]] == "U")
    ## identify potential swaps by row
    Swaps ← apply(SwapPoss, 1, function(row) sum(row) > 1)
    ## identify the potential errors
    PotErr ← apply(SwapPoss, 1, function(row) sum(row) == 1)
    ## use the unknowns to account for some errors
    UnkInd ← apply(Unknown, 1, any)
    FalErr ← PotErr & UnkInd
    ## identify the indices to investigate
    SwapInds ← which(Swaps|FalErr)
    ErrInds ← which(PotErr & !UnkInd)
    ## communicate to the user and ask for input
    cat("There are ", sum(Swaps|FalErr), " swaps to check\n", sep = "")
    cat("Additionally, it seems there are ", sum(PotErr & !UnkInd), " errors in
        entries\n", sep = "")
    ## unless automated
    if (auto) ErrorReturn ← TRUE else ErrorReturn ← as.logical(readline("Return
        the errors? (T/F): "))
    ## now, if there are possible swaps investigate them
    if (sum(Swaps|FalErr) != 0) {
        ## create a temporary storage structure
        tempRows ← data[SwapInds, colInds]
        tempRows ← as.data.frame(lapply(tempRows, function(var) levels(var)[as.
            numeric(var)]),
                                  stringsAsFactors = FALSE)
        ## loop through and populate this
        for (ii in 1:nrow(tempRows)) {
            ## inspect the row
            print(tempRows[ii,])
            ## suggest corrections, first generate matches
            candComb ← lapply(tempRows[ii,],
                               function(el) which(sapply(CorrectLevs,
                                                         function(levs) el %in%
                                                             levs)))
            reps ← unlist(lapply(candComb, length))
            ## now generate all swap combinations
            candComb[[1]] ← rep(candComb[[1]], each = max(reps[-1]))
            candComb ← as.data.frame(candComb, row.names = NULL)
            ## identify rows which contain all indices, in other words those
                valid as swaps
            compRows ← apply(candComb, 1, function(row) all(1:length(CorrectLevs)
                %in% row))
            goodComb ← candComb[compRows,]
            ## clean them up and print them
            colnames(goodComb) ← NULL
            rownames(goodComb) ← NULL
            cat("Potential combinations:\n")
            print(t(apply(goodComb,1,order)))
            ## take user input or automatically determine value
            if (auto) {
                if (!any(compRows)) acceptedComb ← 0 else acceptedComb ← 1
            } else acceptedComb ← as.numeric(readline("Enter a combination choice
                 (0 for error, <enter> to accept first): "))
            ## handle special cases, 0 if a true error has been identified
            if (identical(acceptedComb,0)) { ## 0 if a true error has been
                identified
                ErrInds ← c(ErrInds, SwapInds[ii])
                cat("True error identified, adding ", SwapInds[ii], " to error
                    list\n", sep = "")
            } else { ## the case where a swap has been correctly identified and
                selected, or enter has been pressed
                ## if enter has been pressed accept the first row
                if (is.na(acceptedComb)) acceptedComb ← 1
                ## print recombined row
                newRows ← tempRows[ii,order(as.matrix(goodComb[acceptedComb,]))]
                colnames(newRows) ← NULL
                rownames(newRows) ← NULL
                cat("Corrected row:")
                print(newRows)
                cat("------------------\n")
                ## correct entry
                tempRows[ii,] ← newRows
            }
        }
        ## fill the data
        ## first prevent factor level errors
        data[,colInds] ← lapply(colInds, function(ind) levels(data[,ind])[as.
            numeric(data[,ind])])
        ## now swap the data
        data[SwapInds,colInds] ← lapply(1:length(colInds), function(ind) tempRows
            [,ind])
        ## reconvert back to factors
        data[,colInds] ← lapply(colInds, function(ind) as.factor(data[,ind]))
    }
    ## in either case return the data and errors as specified
    if (ErrorReturn) {
        return(list(Data = data, Errors = ErrInds))
    } else {
        return(data)
    }
}

## now create a function to address the errors possibly identified in the above
    function automatically
SwapErrorFix ← function(errorData, CorrectLevs) {
    ## check if we are in the case without errors
    if (!identical(names(errorData), c("Data", "Errors"))) {
        cat("No errors\n")
        return(errorData)
    } else {
        ## extract the data and data in error
        fulldata ← errorData$Data
        ## get the relevant columns
        colInds ← match(names(CorrectLevs), names(fulldata))
        ## go through the specified variables and remove errors
        fixed ← lapply(1:length(colInds),
                        function(ind) {
                            var ← fulldata[,colInds[ind]]
                            var ← levels(var)[as.numeric(var)]
                            inds ← !(var %in% CorrectLevs[[ind]])
                            cat(names(CorrectLevs)[ind], ": ", sum(inds),
                                " errors\n", sep = "")
                            var[inds] ← "U"
                            as.factor(var)
                        })
        ## insert these fixed values
        fulldata[, colInds] ← fixed
        ## return this
        fulldata
    }
}
```

```

## write a wrapper to perform this swapping and error correction in one call
SwapandError ← function(data, CorrectLevs) {
    swapped ← SimpleSwapper(data = data, CorrectLevs = CorrectLevs, auto = TRUE)
    fixed ← SwapErrorFix(errorData = swapped, CorrectLevs = CorrectLevs)
    fixed
}

## Variable Synthesis #################
## Kullback-Leibler divergence function
kldiv ← function(samp, dist) {
    ## convert to matrices
    mat1 ← as.matrix(samp)
    mat2 ← as.matrix(dist)
    ## make into proper distributions
    mat1 ← mat1/rowSums(mat1)
    mat2 ← mat2/rowSums(mat2)
    ## take the log ratio
    logratio ← log(mat1/mat2)
    ## multiply by correct matrix
    vals ← mat1*logratio
    ## take the row sums
    rowSums(vals, na.rm = TRUE)
}

## make a text-mining regularization function
StringReg ← function(strs) {
    ## first set everything to lowercase
    strs ← tolower(strs)
    ## replace specific patterns (noticed during early tests)
    strs ← str_replace_all(strs, "b/e|break/enter|b&e|break or enter|b or e|b &/
        or e|b & e", "breaking and entering")
    strs ← str_replace_all(strs, "controlled substance", "cs")
    strs ← str_replace_all(strs, "dwi", "driving while impaired")
    strs ← str_replace_all(strs, "rwdw", "robbery with a deadly weapon")
    strs ← str_replace_all(strs, "pwisd|pwmsd|pwmsd|pwitd|pwid|pwmisd|pwosd", "
        pwimsd")
    strs ← str_replace_all(strs, "robery|rob ", "robbery")
    strs ← str_replace_all(strs, "bulgary", "burglary")
    strs ← str_replace_all(strs, "awdw", "assault with a deadly weapon")
    strs ← str_replace_all(strs, "(?<=[\\sa-z])[0-9]{2,}", "")
    strs ← str_replace_all(strs, "att ", "attempted ")
    strs ← str_replace_all(strs, "assult", "assault")
    strs ← str_replace_all(strs, "marj", "marijuana")
    ## replace punctuation
    strs ← gsub("[^[:alnum:][:space:]’]", "", strs)
    ## return these
    strs
}

## create a function to process such a tree structure given a list of strings
stringTree ← function(strs, regexTree, inds = 1:length(strs), includeOther = TRUE
    ) {
    ## identify the sublists, and divide the data
    sublists ← sapply(regexTree, is.list)
    ## iterate over unnamed items (leaf nodes)
    listdiv ← lapply(regexTree[!sublists], function(el) inds[grepl(el, strs, perl
         = TRUE)])
    names(listdiv) ← unlist(regexTree[!sublists])
    ## check if there are any sublists
    if (!any(sublists)) {
        if (includeOther) listdiv ← c(listdiv, other = list(inds[!(inds %in%
            unlist(listdiv))]))
        ## in the case of none, treat the object as a list to iterate through
        listdiv
    } else {
        ## otherwise recurse over the branches
        finlist ← c(listdiv, lapply(names(regexTree)[sublists],
                                    function(name) stringTree(strs[grepl(name,
                                        strs, perl = TRUE)],
                                                               regexTree[[name]],
                                                               inds[grepl(name,
                                                                   strs, perl =
                                                                   TRUE)],
                                                               includeOther)))
        names(finlist)[(length(listdiv) + 1):length(finlist)] ← names(regexTree)[
            sublists]
        c(finlist, other = list(inds[!(inds %in% unlist(finlist))]))
    }
}

## create a tree depth helper function
maxdepth ← function(tree, counter = 1) {
    max(sapply(tree, function(br) if (!is.list(br)) counter else maxdepth(br,
        counter + 1)))
}

## create a function to aggregate a tree as specified above at the desired depth
treeAgg ← function(tree, level = 1) {
    ## first check the max depth of the tree
    treedepth ← maxdepth(tree)
    ## compare this to requested aggregation level
    stopifnot(level <= treedepth)
    ## aggregate at desired level with a helper function
    agg ← function(tr, depth = 1) {
        if (depth == level) lapply(tr, function(el) setNames(unlist(el),NULL))
            else lapply(tr, function(br) agg(dr, depth + 1))
    }
    agg(tree)
}

## create a crime class aggregation function
CrimeClassify ← function(tree, regChar) {
    crimes ← list()
    crimes$Sex ← unique(c(unlist(tree[c("rape", "sex(?=.*offense)", "sex(?=.*
        offend)", "indec")]),
                           tree$other[grepl("sex", regChar[tree$other])]))
    crimes$Theft ← unique(unlist(tree[c("stole", "embez", "break", "larceny", "
        robb", "burg", "identity")]))
    crimes$Murder ← unique(unlist(tree[c("murder", "manslaughter")]))
    crimes$Drug ← unique(c(unlist(tree[c("mari", "coca", "cs", "hero", "meth", "
        oxycod")]),
                            tree$other[grepl("para|drug|substance|pwimsd",
                                regChar[tree$other])]))
    crimes$Violent ← unique(unlist(tree[c("arson", "assa", "abuse|cruelty")]))
    crimes$Driving ← unique(c(unlist(tree[c("driving")]),
                               tree$other[grepl("hit(?=.*run)|speeding", regChar[
                                   tree$other], perl = TRUE)]))
    crimes
}

## in order to make the process of pre-processing the data and adding desired
    columns, place the pre-processing into a
## flexible function and add operations as desired
SynCols ← function(data) {
    ## too busy, synthesize some variables to clearly indicate the results of
        defense and prosecution selection
    data$VisibleMinor ← data$Race != "White"
    data$PerempStruck ← grepl("S_rem|D_rem", data$Disposition)
    data$DefStruck ← data$Disposition == "D_rem"
    data$ProStruck ← data$Disposition == "S_rem"
    data$CauseRemoved ← data$Disposition == "C_rem"
    ## lets look at which race struck each juror
    data$StruckBy ← as.factor(sapply(1:nrow(data),
                                               function(ind) {
                                                   dis ← as.character(data$
                                                       Disposition[ind])
                                                   if (dis == "S_rem") {
                                                       as.character(data$ProsRace
                                                           [ind])
```

```
                                                  } else if (dis == "D_rem") {
                                                      as.character(data$DCRace[ind])
                                                  } else "Not Struck"
                                              }))
    ## create a white black other indicator
    data$WhiteBlack ← FactorReduce(data$Race, tokeep = c("Black", "White", "U"))
    data$DefWhiteBlack ← FactorReduce(data$DefRace, tokeep = c("Black", "White", "U"))
    data$VicWhiteBlack ← FactorReduce(data$VictimRace, tokeep = c("Black", "White", "U"))
    ## return the data with synthesized columns
    data
}

## write functions to process the sentences
SentenceProcess ← function(sentencing) {
    sents ← tolower(sentencing)
    ## identify sentences in months, years, and days
    monthsent ← str_extract(sents, "[0-9\\-]+\\s*(?=m)")
    daysent ← str_extract(sents, "[0-9\\-]+\\s*(?=d)")
    yearsent ← str_extract(sents, "[0-9\\-]+\\s*(?=y)")
    ## extract life without parole
    lwp ← str_extract(sents, "parol[e]*")
    ## and with parole
    life ← str_extract(sents, "life")
    life[!is.na(lwp)] ← NA
    ## get restitutions
    resti ← str_extract(sents, "[0-9,]+\\s*(?=restitu)|\\$[0-9,]+")
    ## get supervised probation
    suprob ← str_extract(sents, "sup.*pro")
}

## Summary Functions ##################
## make a function to summarize trial jury data
JurySummarize ← function(Varnames = c("Disposition", "Race", "Gender", "PoliticalAffiliation")) {
    ## check if a juror summary object exists already
    if (!("sun.juror" %in% ls(.GlobalEnv))) {
        ## first group the data for easy access
        Juries ← aggregate(sun.swap[, Varnames],
                            by = list(TrialNumberID = sun.swap$TrialNumberID,
                                JurorNumer = sun.swap$JurorNumber),
                            unique)
    } else Juries ← sun.juror
    ## in either case, perform aggregation by trial instance
    Juries ← aggregate(Juries[, Varnames],
                        by = list(TrialNumberID = Juries$TrialNumberID),
                        function(var) var)
    ## clean up the names
    names(Juries)[grepl("Polit", names(Juries))] ← "PolAff"
    Varnames[4] ← "PolAff"
    ## now summarize relevant features
    Summary ← apply(Juries[, Varnames], 1,
                     function(row) {
                         ## get final jury indices
                         disps ← unlist(row$Disposition)
                         foreman ← grepl("Foreman", disps)
                         finJur ← grepl("Foreman|Kept", disps)
                         defStruck ← grepl("D_rem", disps)
                         proStruck ← grepl("S_rem", disps)
                         ## process all variables
                         newrow ← sapply(row,
                                          function(el) {
                                              c(Jury = table(unlist(el)[finJur]),
                                                Venire = table(unlist(el)),
                                                DefRem = table(unlist(el)[defStruck]),
                                                ProRem = table(unlist(el)[proStruck]))
                                          })
                         newrow$Disposition ← NULL
                         newrow ← c(unlist(newrow), ForeRace = row$Race[foreman],
                                     ForeGender = row$Gender[foreman], ForePol =
                                         row$PolAff[foreman])
                         if (sum(foreman) > 1) {
                             names(newrow)[names(newrow) == "ForeRace1"] ← "
                                 ForeRace"
                             names(newrow)[names(newrow) == "ForeGender1"] ← "
                                 ForeGender"
                             names(newrow)[names(newrow) == "ForePol1"] ← "
                                 ForePol"
                         }
                         newrow
                    })
    ## perform some clean up
    longest ← sapply(Summary, length)
    longest ← which(longest == max(longest))[1]
    longNames ← names(Summary[[longest]])
    Summary ← lapply(names(Summary[[longest]]),
                      function(name) unname(sapply(Summary,
                                                   function(el) el[name])))
    names(Summary) ← longNames
    Summary ← lapply(longNames,
                      function(nm) {
                          if (grepl("ForeGender", nm)) {
                              Summary[[nm]] ← factor(Summary[[nm]], levels = 1:3,
                                   labels = LevGen)
                          } else if (grepl("ForePol", nm)) {
                              Summary[[nm]] ← factor(Summary[[nm]], levels = 1:5,
                                   labels = LevPol)
                          } else if (grepl("ForeRace", nm)) {
                              Summary[[nm]] ← factor(Summary[[nm]], levels = 1:7,
                                   labels = LevRace)
                          } else Summary[[nm]]
                      })
    names(Summary) ← longNames
    ## return these
    list(Juries = Juries, Summaries = as.data.frame(Summary))
}

## a generic simplification method to summarize a vector
Simplifier ← function(col, ...) {
    UseMethod("Simplifier")
}

## code up methods for the types to be seen
Simplifier.default ← function(col, collapse = "") paste0(col, collapse = collapse
    )
Simplifier.numeric ← function(col, na.rm = TRUE, trim = 0, ...) mean.default(col,
     trim = trim, na.rm = na.rm)
Simplifier.factor ← function(col, collapse = "", ...) paste0(sort(as.character(
    levels(col)[as.numeric(col)])),
                                                              collapse = collapse
                                                                  )
Simplifier.character ← function(col, collapse = "", ...) paste0(sort(col),
    collapse = collapse)

## create a grouping wrapper which does unique aggregation of a data set
UniqueAgg ← function(data, by, ...) {
    ## convert data to a data frame for regularity
    if (!is.data.frame(data)) data ← as.data.frame(data)
    ## identify the grouping column by in the data
    by.groups ← names(data) == by
    ## provide nice error handling
    stopifnot(sum(by.groups) > 0)
    ## first identify which rows are already unique
    groups ← as.numeric(as.factor(unlist(data[by.groups])))
    unqRows ← sapply(groups, function(el) sum(groups == el) == 1)
    ## consider grouping only the other rows using the unique function
    endata ← data[unqRows,]
    unqdata ← aggregate(data[!unqRows, !by.groups], by = list(data[!unqRows, by.groups]), unique)
    ## reorder to make sure everything is compatible
    names(unqdata)[1] ← by
    unqdata ← unqdata[,match(names(endata), names(unqdata))]
    ## now use the Simplifier helper defined above to process these results
    procdata ← lapply(unqdata, function(col) sapply(col, Simplifier, ...))
    ## append everything together
    endata ← lapply(1:length(endata),
                    function(n) c(if (is.factor(endata[[n]])) as.character(endata[[n]]) else endata[[n]],
                                  procdata[[n]]))
    names(endata) ← names(data)
    ## convert to a data frame
    as.data.frame(endata)
}

## a simple helper to convert multiple factor levels into a single 'other' level
FactorReduce ← function(vals, tokeep) {
    chars ← as.character(vals)
    ## simply replace elements
    chars[!grepl(paste0(tokeep, collapse = "|"), chars)] ← "Other"
    chars
}

## write a function to re-level factor variables to make mosaic plots cleaner
MatRelevel ← function(data) {
    temp ← lapply(data, function(el) if (is.factor(el)) as.factor(levels(el)[as.numeric(el)]) else el)
    temp ← as.data.frame(temp)
    names(temp) ← names(data)
    temp
}

## another simple processing function to correct NA's given some other identifier and data set
FillNAs ← function(dataNAs, filldata, identifier) {
    ## extract the relevant column indices in a flexible way
    if (is.null(colnames(filldata))) {
        relcol ← grepl(identifier, names(filldata))
    } else relcol ← grepl(identifier, colnames(filldata))
    ## first identify the relevant rows in the data NAs
    relRows ← is.na(dataNAs)
    ## take the relevant rows of the filldata
    filldata ← matrix(unlist(filldata[relcol]), ncol = sum(relcol))
    rowfiller ← rowSums(filldata[relRows,])
    ## return the filled data
    dataNAs[relRows] ← rowfiller
    dataNAs
}

## write a wrapper to estimate the values of total removed jurors
RemovedJurorEstimates ← function(tofill, data, ident, plot = TRUE) {
    temp ← FillNAs(tofill, filldata = data, identifier = ident)
    temp2 ← rowSums(data[,grepl(ident, names(data))])
    ## let's see how accurate this is if plotting is desired
    if (plot) {
        plot(temp, temp2, xlab = "Observed and Filled", ylab = "Juror Sums")
        abline(0,1)
    }
    cat("= : ", sum(temp == temp2)/length(temp2), "\n", "< : ", sum(temp2 < temp)/length(temp2), "\n", sep = "")
    ## replace the filled values less than the estimated, for consistency
    temp[temp < temp2] ← temp2[temp < temp2]
    temp
}


## LOADING AND PROCESSING DATA #########
```

```r

## load the data
SunshineData ← lapply(SunshineSheets, function(nm) as.data.frame(read_excel(SunshineFile, sheet = nm)))
names(SunshineData) ← SunshineSheets
NorthCarData ← read.csv(NorthCarFile)
PhillyData ← read.csv(PhillyFile)

## clean non-informative columns
CleanSunshine ← lapply(SunshineData, function(dat) dat[, !apply(dat,2,function(col) all(is.na(col)))])

## the Sunshine data needs to be restructured into one table, rather than a relational database structure
## see the IDMatch function, this was created specifically to perform ID-based table joins
## the most appropriate global target is the juror table, start by matching this to the trial
FullSunshine ← with(CleanSunshine, CleaningMerge(Jurors, Trials, by = "TrialNumberID"))
## remove extra ID column, fix a misleading name
FullSunshine$CountyName ← FullSunshine$CountyID
FullSunshine$CountyID ← NULL
## clean up two additional columns which had inconsistencies
FullSunshine$Disposition ← toupper(FullSunshine$Disposition)
FullSunshine$Race[FullSunshine$Race == "?"] ← "U"
## before appending everything to this table, perform some other joins
TrialsToCharge ← with(CleanSunshine, CleaningMerge(Charges, Junction, by = "ACISID", all = TRUE))
DefendantToTrial ← with(CleanSunshine, CleaningMerge(Defendants, DefendantTrial, by = "DefendantID", all = TRUE))
AttorneyToTrial ← with(CleanSunshine, CleaningMerge(Attorney, AttorneyTrial, by = "DefAttyID", all = TRUE))
ProsecutorToTrial ← with(CleanSunshine, CleaningMerge(Prosecutor, ProsecutorTrial, by = "ProsecutorID", all = TRUE))
## merge issues:
##    - trials to charge: one charge is missing a trial ID, hopefully not important
##    - prosecutors to trials: 26 prosecutors without trials, however all entries were entirely uninformative
## given the above outputs, rename the failed clean merges to make the next section cleaner
TrialsToCharge ← TrialsToCharge$Merge
ProsecutorToTrial ← ProsecutorToTrial$Merge

## now perform some additional merges to create one sheet/data.frame
## add the judge descriptions (no issues)
FullSunshine ← CleaningMerge(FullSunshine, CleanSunshine$Judges, by = "JudgeID", all = TRUE)
## the charges
FullSunshine ← CleaningMerge(FullSunshine, TrialsToCharge, by = "TrialNumberID", all = TRUE)
## this leads to 22 jurors in trials without charges and 29 charges without trials, inspecting these:
##    - the jurors without charges are all related to a trial with ID number "710-01", thankfully the other data
##      for this case is complete, and so it may still be useful for viewing jury behaviour
##    - the charges without trials are all of the form "710-0xx", suggesting the omission of entire trials of some
##      relation, hopefully these were not too similar, or this exclusion can be explained later
FullSunshine ← FullSunshine$Merge
## the defendants
FullSunshine ← CleaningMerge(FullSunshine, DefendantToTrial, by = "TrialNumberID", all = TRUE)
## the attorneys
FullSunshine ← CleaningMerge(FullSunshine, AttorneyToTrial, by = "TrialNumberID", all = TRUE)
## the prosecutors
```

```
FullSunshine ← CleaningMerge(FullSunshine, ProsecutorToTrial, by = "TrialNumberID", all = TRUE)
## 26 jurors appear to be lacking a prosecutor, these appear to be the uninformative prosecutors from earlier, included
## due to the preferential inclusion of the missing values in the first of the merged matrices
FullSunshine ← FullSunshine$Merge

## perform some cleanup
## start with some specific factor replacements
## replace the "N" with "I", as these factor levels are interchangeable in the codebook and prevent confusion with race
FullSunshine[,grepl("Pol", names(FullSunshine))] ← lapply(FullSunshine[,grepl("Pol", names(FullSunshine))],
                                                            function(var) {
                                                                var ← toupper(var)
                                                                var[var == "N"] ← "I"
                                                                var
                                                            })
## next save most variables as factors
FullSunshine ← lapply(FullSunshine,
                       function(el) if (is.character(el)) as.factor(el) else el)
## correct some overzealous assignment from above
FullSunshine[grepl("Notes", names(FullSunshine))] ← lapply(FullSunshine[grepl("Notes", names(FullSunshine))],
                                                             as.character)
## perform factor regularization according to the factor levels provided in the codebook
FullSunshine ← sapply(FullSunshine,
                       function(el) {
                           if (!is.factor(el)) {
                               el[el == 999] ← NA
                               el
                           } else {
                               el ← as.character(el)
                               el ← toupper(el)
                               el[is.na(el)] ← "U"
                               as.factor(el)
                           }
                       }, simplify = FALSE)
FullSunshine ← as.data.frame(FullSunshine)
## remove some unnecessary columns
FullSunshine$ID ← NULL
FullSunshine$TrialIDAuto ← NULL
## combine the name columns to produce more useful columns
FullSunshine$JName ← paste(FullSunshine$JFirstName, FullSunshine$JLastName)
FullSunshine$JName[FullSunshine$JName == "U U"] ← "U"
FullSunshine$DefAttyName ← paste(FullSunshine$DCFirstName, FullSunshine$DCLastName)
FullSunshine$DefAttyName[FullSunshine$DefAttyName == "U U"] ← "U"
FullSunshine$ProsName ← paste(FullSunshine$ProsecutorFirstName, FullSunshine$ProsecutorLastName)
FullSunshine$ProsName[FullSunshine$ProsName == "U U"] ← "U"

## Checkpoint 1: the clean data has been processed, none of the swaps, synthesis, or expansion has taken place
## save this
if (!("FullSunshine.csv" %in% list.files())) write.csv(FullSunshine, "FullSunshine.csv", row.names = FALSE)
## load if the desire is to start at checkpoint 1
if (!("FullSunshine" %in% ls())) FullSunshine ← read.csv("FullSunshine.csv")

## Note: the below swap functions have been set to auto as the function's performance in these cases has already
## been assessed, and so the swaps have already been inspected, it is critical for new data that "auto" be switched
## off to take full advantage of this functionality, and so the wrapper "SwapandError" should not be used
## in the juror data
```

```
sun.swapJuror ← SwapandError(FullSunshine, CorrectLevs = list(Race = LevRace,
                                                              Gender = LevGen,
                                                              PoliticalAffiliation
                                                                   = LevPol))
## in the judge data
sun.swap ← SimpleSwapper(sun.swapJuror, CorrectLevs = list(JRace = LevRace,
                                                           JGender =
                                                               LevGen,
                                                           JPoliticalAff
                                                                = LevPol
                                                               ))
## viewing the error report of these data, they are all related to one judge,
    Arnold O Jones II, who is verified
## as a male after a quick Google search
unique(sun.swap$Data[sun.swap$Errors, c("JFirstName", "JLastName")])
sun.swapJudge ← sun.swap$Data
sun.swapJudge$JGender[sun.swap$Errors] ← "M"
sun.swapJudge$JGender ← as.factor(levels(sun.swapJudge$JGender)[as.numeric(sun.
    swapJudge$JGender)])
## in the prosecutor data
sun.swap ← SimpleSwapper(sun.swapJudge, CorrectLevs = list(ProsRace = LevRace,
                                                           ProsGender =
                                                               LevGen,
                                                           ProsPoliticalAff
                                                                = LevPol
                                                               ))
## that found no errors
## a quick check of the levels of the defendant data finds only one error
levels(sun.swap$DefGender)
levels(sun.swap$DefRace)
sun.swap ← SwapandError(sun.swap, CorrectLevs = list(DefRace = LevRace,
                                                     DefGender = LevGen
                                                         ))
## next the attorney data
sun.swap ← SwapandError(sun.swap, CorrectLevs = list(DCRace = LevRace,
                                                     DCGender = LevGen,
                                                     DCPoliticalAff =
                                                         LevPol))
## finally the victim data
sun.swap ← SwapandError(sun.swap, CorrectLevs = list(VictimRace = LevRace,
                                                     VictimGender =
                                                         LevGen))
## this leaves the data error-free (in at least the race/gender/politics columns)

## fix the outcome data, which had some improper levels
sun.swap$Outcome[sun.swap$Outcome == "HC"] ← "U"
sun.swap$Outcome[sun.swap$Outcome == "G"] ← "GC"
sun.swap$Outcome ← as.factor(levels(sun.swap$Outcome)[as.numeric(sun.swap$Outcome
    )])

## lets make the levels more clear for some of the data (race, politics,
    disposition)
## start with the disposition
levels(sun.swap$Disposition) ← c("C_rem", "D_rem", "Foreman", "Kept", "U_rem",
                                     "S_rem", "Unknown")
## next the political affiliation
sun.swap ← lapply(sun.swap, function(el) {
    if (is.factor(el) & identical(levels(el), LevPol)) {
        levels(el) ← c("Dem", "Ind", "Lib", "Rep", "U")
        el
    } else el})
levels(sun.swap$JPoliticalAff) ← c("Dem", "Ind", "Rep", "U")
## now the race
sun.swap ← lapply(sun.swap, function(el) {
    if (is.factor(el) & identical(levels(el), LevRace)) {
        levels(el) ← c("Asian", "Black", "Hisp", "NatAm", "Other",
                        "U", "White")
        el
    } else el})
levels(sun.swap$VictimRace) ← c("Asian", "Black", "Hisp", "NatAm",
                                     "U", "White")
levels(sun.swap$JRace) ← c("Black", "Hisp", "NatAm", "U", "White")
levels(sun.swap$DCRace) ← c("Asian", "Black", "NatAm", "Other",
                                 "U", "White")
## now the outcome/verdict
levels(sun.swap$Outcome) ← c("Acquittal", "Guilty as Charged",
                                  "Guilty of Lesser", "Incomplete", "Mistrial",
                                  "U")
## the defense attorney type
levels(sun.swap$DefAttyType) ← c("App Priv", "Public", "Private",
                                  "Ret Priv", "U", "Waived")

## add a guilt indicator
sun.swap$Guilty ← grepl("Guilty", sun.swap$Outcome)

## add a simple indicator of defendant race matching juror race if they are both known
sun.swap$RaceMatch ← sun.swap$Race == sun.swap$DefRace
sun.swap$RaceMatch[sun.swap$Race == "U" | sun.swap$DefRace == "U"] ← NA

## now perform tree classification of crimes
## first cast sun.swap as a data frame
sun.swap ← as.data.frame(sun.swap)
## regularize the charges
chargFact ← as.factor(sun.swap$ChargeTxt)
regCharg ← StringReg(levels(chargFact))[as.numeric(chargFact)]
## classify these into a charge tree and aggregate this at the coarsest level
aggCharg ← treeAgg(stringTree(regCharg, chargeTree))
## these can be further classified into crime classes
crimes.trial ← CrimeClassify(aggCharg, regCharg)
## convert these classes into a factor for the data, start with a generic "other" vector
sun.swap$CrimeType ← rep("Other", nrow(sun.swap))
## now populate it
for (nm in sort(names(crimes.trial))) sun.swap$CrimeType[crimes.trial[[nm]]] ← nm
sun.swap$CrimeType ← as.factor(sun.swap$CrimeType)

## synthesize additional columns
sun.swap ← SynCols(sun.swap)

## now organize this on the juror scale
sun.juror ← UniqueAgg(sun.swap, by = "JurorNumber", collapse = ",")

## Checkpoint 2: the swapped data has been processed and summarized to be on the scale of individual jurors
## save the swapped data
write.csv(sun.swap, "FullSunshine_Swapped.csv", row.names = FALSE)
## and the juror summarized data
saveRDS(sun.juror, "JurorAggregated.Rds")

## summarize by trial, get the unique trials
Trials ← unique(sun.swap$TrialNumberID)
## extract information about these trials, note that grouping occurs on the trial ID, defendant ID, and charge ID levels,
## as the trials frequency involve multiple charges and defendants, which makes them less clean
sun.trial ← aggregate(sun.swap[,TrialVars],
                           by = list(sun.swap$TrialNumberID, sun.swap$DefendantID.DefendantToTrial,
                                     sun.swap$ID.Charges),
                           unique)
sun.trial$Group.1 ← NULL
sun.trial$Group.2 ← NULL
sun.trial$Group.3 ← NULL

## summarize the juries by trial as well
sun.jursum ← JurySummarize()

## merge the summaries to the trial sunshine data
sun.trialsum ← merge(cbind(TrialNumberID = sun.jursum$Juries$TrialNumberID, sun.
    jursum$Summaries),
                      sun.trial, all = TRUE)

## notice that the total removed variables are incomplete, try to correct this
    where possible using the jury
## summarized data above
sun.trialsum$DefRemEst ← RemovedJurorEstimates(sun.trialsum$DefenseTotalRemoved,
    data = sun.trialsum,
                                                ident = "Gender.DefRem", plot =
                                                    FALSE)
## perform this same procedure for the prosecution removals
sun.trialsum$ProRemEst ← RemovedJurorEstimates(sun.trialsum$StateTotalRemoved,
    data = sun.trialsum,
                                                ident = "Gender.ProRem", plot =
                                                    FALSE)
## synthesize some other variables, simple race indicators
sun.trialsum$DefWhiteBlack ← as.factor(FactorReduce(sun.trialsum$DefRace, tokeep
    = c("Black", "White", "U")))
sun.trialsum$DefWhiteOther ← as.factor(FactorReduce(sun.trialsum$DefWhiteBlack,
    tokeep = c("White", "U")))
## the Kullback-Leibler divergence
sun.trialsum$KLdiv ← kldiv(sun.trialsum[,grepl("Jury", names(sun.trialsum))],
                           sun.trialsum[,grepl("Venire", names(sun.trialsum))])

## Checkpoint 3: the data has been set to the trial level and summarized
## save this
saveRDS(sun.trialsum, "TrialAggregated.Rds")
saveRDS(sun.jursum, "AllJuries.Rds")
```

A.3 Using Sweave to include R code (and more) in your report

The easiest (and most elegant) way to include R code and its output (and have all your figures up to date with your report) is to use Sweave. You can find an introduction Sweave in /u/sfs/StatSoftDoc/Sweave/Sweave-tutorial.pdf.

Appendix B

Yet another appendix....

B.1 Description

Something details.

Something else other definition.

B.2 Tables

Refer to Table B.1 to see a left justified table with caption on top.

Table B.1: Results.

Student	**Grade**
Marie	6
Alain	5.5
Josette	4.5
Pierre	5

Epilogue

A few final words.

Declaration of Originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor .

Title of work (in block letters):

. . .

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Muster	Student

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the Citation etiquette information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work .
- I am aware that the work may be screened electronically for plagiarism.
- I have understood and followed the guidelines in the document *Scientific Works in Mathematics.*

Place, date:	**Signature(s):**
Zurich August 19th 2009	bla

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.