The Stewardship Model: An Inclusive Approach to Undergraduate Research

Online Appendix

Version: January 27, 2020

Megan Becker[1]
Benjamin A.T. Graham[2]
Kelebogile Zvobgo[3]

[1] Political Science and International Relations, University of Southern California, meganbec@usc.edu.
[2] Political Science and International Relations University of Southern California, benjamag@usc.edu
[3] Political Science and International Relations, University of Southern California; Global Research Institute, William & Mary, kzvobgo@wm.edu.

Table of Contents

APPENDIX A:
Course Syllabi and Training Materials

The following training materials include for-credit courses and non-credit-bearing workshops focused on research skill development and applied data science. The courses are meant to be taken before and during student research assistantships. They enable students to develop their research and data analysis skills while applying them to real research projects. Professionalization workshops coach students in how to apply, communicate and market their research skills both inside and outside of academia. Technical workshops, such as those listed below, are produced and led by graduate students or advanced undergraduates and give lab members hands-on training in technical skills like R, a statistical computing software package widely used in the social sciences. The data science workshops overlap in content with the credit-bearing classes, giving students different avenues to pursue the same skillsets.

Special thanks to Therese Anders and Miriam Barnum who designed a number of the trainings used by the SPEC Lab. Thanks also to Jihyun Shin and Alix Ziff who have helped revise those trainings over time. R training materials were originally created in R Markdown. **If you would like the R markdown code or the affiliated datasets, please don't hesitate to reach out by email** (benjamin.a.graham@usc.edu). For new trainings often being updated, we also recommend Therese Anders's Github page (https://github.com/thereseanders). Dr. Anders previously served as Director of Research and Training in the SPEC Lab.

Course: *Supervised Undergraduate Research Experience*

This course is open to upper division students interested in working as an RA on a collaborative research project under the supervision of a Faculty Research Mentor. Through workshops, training sessions and progress meetings, each student learns the skills necessary to contribute to a research project. Students work with other undergraduates, PhD students and professors on each aspect of the research process. Students produce a reflection paper on the status of their project and their own progress as well as a short article or blog post to clearly communicate their research to a broader audience. The goal of the course is to facilitate specialized skills development in the research process while enabling students to earn academic credit for their research experience. This course is meant to be taken in conjunction with a student's research assistantship.

Course: *Applied Data Science for International Relations I*

This course provides an introduction to statistical computing and data visualization in R (http://cran.r-project.org). This course is the first of a two-course sequence in applied data science. The focus of this course is on how to make sense of data and make proper use of graphics, rather than the specifics of statistical modeling. This course is strongly applied, and each student will complete a data visualization project related to the ongoing research of a School of International Relations faculty member. Students will be matched with faculty members. Most of these final projects will involve the graphical presentation of descriptive statistics, such as kernel density plots, heat maps, and scatter plots. Many of the final projects from this course will become part of published research by faculty members.

Course: *Applied Data Science for International Relations II*

This is the second part of a data science sequence. The goals of this course are to enable students to manage and clean data, including reshaping and merging datasets; to understand the intuition behind regression analysis; and create publication-grade figures in R that graphically display regression results. In two projects students apply their data science skills to existing research. For the first project, each student contributes a useful data source to an existing compilation dataset. This requires soliciting permission from the dataset authors, downloading the data, cleaning and merge-prepping the data, merging it into the data compilation, and writing a new entry for the codebook. In the second project, students continue working with faculty members. They will create a publication-grade figure that graphically presents the regression results from an actual working paper or book chapter written by that faculty member.

Workshop: *Introduction to R (Miriam Barnum)*

The introduction to R workshop familiarizes students with R as a language, its basic functions and operators. Students are taught how to use R for arithmetic, basic indexing, and summarizing a dataset. Students learn how to use R as a calculator and complete basic exercises prior to using operators across large amounts of data. They then learn differences in variable types, how to assign values to objects, and organize objects in a logical and accessible manner. By the end of the workshop students are familiar with key terms used in R, how to classify variables, basic arithmetic using those variables, and finding averages of those variables.

Workshops: *Introduction to Tidyr (Therese Anders)*

These workshops introduce students to packages in R which clean and organize data (i.e. the tidyverse). These workshops focus on data exploration and cleaning. Exercises take disorganized data, identify variables within them, and 'tidy' them into an accessible data frame. Students arrange variables and observations into rows and columns with descriptive headers.

The introduction to 'dplyr' teaches students how to select, filter, arrange, change and summarize large datasets. The 'dplyr' package gives students the commands to solve common challenges in data manipulation. Students learn to select portions of the data, group data, determine if there are missing values and change them. By the end of the 'dplyr' and 'tidyr' workshops, students are able to load, explore, clean and summarize large data frames in R.

Workshop: *Introduction to Data Visualization in R with ggplot2 (Therese Anders)*

The introduction to data visualization workshop, is designed as a crash course in how to use the 'ggplot2' package to create graphical summaries of data. Through several guided exercises students use real data to visualize data across multiple groups and create data visualizations with controlled parameters. Students create scatter and line plots, plot simple maps, and adjust their appearances to create aesthetic visualizations.

APPENDIX A.1: Syllabus for *Supervised Undergraduate Research Experience*
Professor: Megan Becker

Course Description and Objectives

By enrolling in the course, students earn credit for participating in collaborative research projects with other students (both UG and PhD) under the supervision of a Faculty Research Mentor. Only students who have been selected to serve as RAs may enroll in the course. In order to facilitate student development and enhance the benefits of the research experience, students are expected to attend regular skills training and professionalization workshops and complete associated assignments. Student progress will be supported and monitored, not only by their Faculty Research Mentor, but also by the professor.

Students are expected to have completed their introductory major courses before enrolling in this course and it is strongly recommended that they have completed an introductory statistics course. The research project undertaken for this course must be different from any honors thesis research. While students may take this course for a single term, it is designed in a manner that allows for students to enroll for a full school year (students may repeat the course once).

In this course, students will:

--gain a greater understanding of politics and the fundamentals of research, through direct experience with the process of research.

--develop skills that will make them more effective researchers and more marketable job candidates.

--be able to articulate both short- and long-term personal and professional goals and have concrete plans for achieving them.

Texts and Resources

All students are required to purchase a copy of Leanne Powner's *Empirical Research and Writing: A Political Science Student's Practical Guide.*

All other course readings will be posted on Blackboard. Individual Faculty Research Mentors will also assign students background reading for their project—please discuss with your Mentor.

Grading

Your grade in this course will be based on the following breakdown of assignments.:

Collaborative Research Project	*50%*
Training Sessions	*15%*
Participation	*5%*
Progress Meetings	*5%*
Weekly Reports	*5%*

Short Article/Blog Post	*5%*
End-of-Term Reflection Paper	*10%*

Collaborative Research Project: The main component of the course is working with an IR Faculty Research Mentor on one of their projects. Particular details of the work to be done by the student will be determined by the Mentor. The assignments undertaken as part of the course will also be advised by the Faculty Mentor, but must be in addition to other paid work.

Training Sessions: There will be two data science training sessions offered during the term. The purpose of these two-hour sessions is to help students learn and develop their research skills. Students in the course are expected to actively participate in all training sessions, completing any pre-session readings and associated assignments.

Weekly Reports: Research can frequently occur in spurts, making it easy to lose sight of the greater whole. This disconnectedness can also lead to procrastination or dissatisfaction. To keep students on track with their research activities and keep the focus on progress, however small, each student will be asked to produce a very brief weekly report, describing what they did during the past week and what they are committing to do for the following week. Certain Faculty Research Mentors may have slightly different requirements for how these reports are to be communicated, but all students taking the course must submit a weekly report to Blackboard.

Progress Meetings: As the purpose of this course is to support students in their development as researchers and encourage their personal career aspirations, faculty must be kept up to date as these matters evolve. To this end, all students will be required to meet with Prof. Becker, both at the beginning and end of the term, as well as have an official progress meeting with their Faculty Research Mentor at the end of the term. In order to facilitate these meetings, students will be required to prepare a self-reflection and may be asked to fill out pre- and post-meeting surveys. All Progress Meetings will be summarized using a common form and forms will be filed to track students who participate in the collaborative research experience for longer than a semester.

Short Article/Blog Post: The development of communications skills is of the utmost importance in any type of career, but particularly research. Sharing ideas and evidence is the foundation of the academic enterprise. As such, students are required to write one short article related to their research activities each term. These articles will be published, with permission on the Undergraduate Research Program website, but students are also encouraged to consult with the Director and their Faculty Research Mentor to discuss opportunities for further dissemination beyond USC, such as submitting an op-ed to a newspaper or a research brief to a politics blog like The Monkey Cage. This article can be submitted at any time during the term, but must completed by the last week of class.

***End-of-Term Reflection Paper*:** During finals week, students will be expected to turn in a paper reflecting on their progress during the term. This paper will include a discussion of the student's research activities during the semester, how they relate to their academic and personal goals, and an assessment of one's own progress. Details of this assignment will be distributed in class.

Americans with Disabilities Act

Students requesting accommodations based on disability are required to register with Disability Services and Programs each semester. A letter of verification for approved accommodations can be obtained from DSP when adequate documentation is filed. Please be sure the letter is delivered to me as early in the semester as possible. DSP is open Monday-Friday 8:30-5:00. The office is in Student Union 301 and their phone number is 213-740-0776.

Policy on Academic Ethics and Honesty

Academic honesty is fundamental to the activities and principles of a university. All members of the academic community must be confident that each person's work has been responsibly and honorably acquired, developed and presented. Any effort to gain an advantage not given to all students is dishonest whether or not the effort is successful. The academic community regards academic dishonesty as an extremely serious matter, with serious consequences. When in doubt about plagiarism, paraphrasing, quoting or collaboration, consult your instructor

Week 1 & 2: *Get situated with your Mentor and Project*
Week 3: *Introduction to the Class, Goal-setting*
Week 4: *Research Topic to RQ* (Powner Ch. 1)
Week 5: *RQ to Theory to Hypothesis* (Powner Ch. 2)
Week 6: *Doing Pre-Research* (Powner Ch. 4)
Week 7: *Annotated Bibliographies and Citation Management*
Week 8: *Choosing a Design that Fits Your Question* (Powner Ch. 4)
Week 9: *Quantitative Data Collection and Management* (Powner Ch. 7)
Week 10: *Training Session*--Intro to R
Week 11: *Training Session*--Intro to R
Week 12: *Case Selection and Study Design for Qualitative Research* (Powner Ch. 5)
Week 13: *Qualitative Data Collection and Management* (Powner Ch. 6)
Week 14: THANKSGIVING
Week 15: *Conferences* with your Mentor and the professor
Finals week: *Turn in Reflection*

Supervised Undergraduate Research Experience (Spring)

Course Description and Objectives

By enrolling in the course, students earn credit for participating in collaborative research projects with other students (both UG and PhD) under the supervision of a Faculty Research Mentor. Only students who have been selected to serve as RAs may enroll in the course. In order to facilitate student development and enhance the benefits of the research experience, students are expected to attend regular skills training and professionalization workshops and complete associated assignments. Student progress will be supported and monitored, not only by their Faculty Research Mentor, but also by the professor.

Students are expected to have completed their introductory major courses before enrolling in this course and it is strongly recommended that they have completed an introductory statistics course. The research project undertaken for this course must be different from any honors thesis research. While students may take this course for a single term, it is designed in a manner that allows for students to enroll for a full school year (students may repeat the course once).

In this course, students will:

--gain a greater understanding of politics and the fundamentals of research, through direct experience with the process of research.

--develop skills that will make them more effective researchers and more marketable job candidates.

--be able to articulate both short- and long-term personal and professional goals and have concrete plans for achieving them

Texts and Resources

All students are required to purchase a copy of two texts. The first is **David Allen's *Getting Things Done***, which we will discuss during our first few professionalization sessions. The text will serve as a primer for organizing one's research activities in a productive manner, as well as encouraging long-term goal-setting.

Our required text for this term's training sessions will be ***Data Wrangling with R* by Bradley C. Boehmke (2016)**. This text is incredibly helpful, as it provides an introduction to many of the data management tasks you may asked to be complete as a research assistant, but that are not always covered in general statistics courses.

All other course readings will be posted on Blackboard. Individual Faculty Research Mentors will also assign students background reading for their project—please discuss with your Mentor.

Grading

Your grade in this course will be based on following breakdown of assignments.

Collaborative Research Project	*50%*
Training Sessions	*15%*
Professionalization Workshops	*5%*
Progress Meetings	*5%*
Weekly Reports	*5%*
Short Article/Blog Post	*5%*
End-of-Term Reflection Paper	*10%*

Collaborative Research Project: The main component of the course is working with an IR Faculty Research Mentor on one of their projects. Particular details of the work to be done by the student will be determined by the Mentor. The assignments undertaken as part of the course will also be advised by the Faculty Mentor, but must be in addition to other paid work.

Training Sessions: There will be four data science training sessions offered during the term. The purpose of these two-hour sessions is to help students learn and develop their research skills. Students in the course are expected to actively participate in all training sessions (10% of grade), completing any pre-session readings and associated assignments (10% of grade).

Professionalization Workshops: These weekly hour-long sessions are meant to help students formulate and articulate goals and develop strategies for thriving both at USC and in their chosen career. Organization/time management, goal-setting, and developing effective communication skills will be emphasized. Sessions will frequently involve guest speakers who can share their experiences and connect students with both internal and external resources to support their academic and career development. The grade for this portion will be based completion of assignments and active participation during the sessions.

Weekly Reports: Research can frequently occur in spurts, making it easy to lose sight of the greater whole. This disconnectedness can also lead to procrastination or dissastisfaction. To keep students on track with their research activities and keep the focus on progress, however small, each student will be asked to produce a very brief weekly report, describing what they did during the past week and what they are committing to do for the following week. Certain Faculty Research Mentors may have slightly different requirements for how these reports are to be communicated, but all students taking the course must submit a weekly report to Blackboard.

Progress Meetings: As the purpose of this course is to support students in their development as researchers and encourage their personal career aspirations, faculty must be kept up to date as these matters evolve. To this end, all students will be required to meet with Prof. Becker, both at the beginning and end of the term, as well as have an official progress meeting with their Faculty Research Mentor at the end of the term. In order to facilitate these meetings, students will be required to prepare a self-reflection and may be asked to fill out pre- and post-meeting surveys. All Progress Meetings will be summarized using a common form and forms will be filed to track students who participate in the collaborative research experience for longer than a semester.

Short Article/Blog Post: The development of communications skills is of the utmost importance in any type of career, but particularly research. Sharing ideas and evidence is the foundation of the academic enterprise. As such, students are required to write one short article related to their research activities each term. These articles will be published on the Undergraduate Research Program website, but students are also encouraged to consult with the Director and their Faculty Research Mentor to discuss opportunities for further dissemination beyond USC, such as submitting an op-ed to a newspaper or a research brief to a politics blog like The Monkey Cage.

End-of-Term Reflection Paper**:** During finals week, students will be expected to turn in a paper reflecting on their progress during the term. This paper will include a discussion of the student's research activities during the semester, how they relate to their academic and personal goals, and an assessment of one's own progress. Details of this assignment will be distributed in class.

Americans with Disabilities Act

Students requesting accommodations based on disability are required to register with Disability Services and Programs each semester. A letter of verification for approved accommodations can be obtained from DSP when adequate documentation is filed. Please be sure the letter is delivered to me as early in the semester as possible. DSP is open Monday-Friday 8:30-5:00. The office is in Student Union 301 and their phone number is 213-740-0776.

Policy on Academic Ethics and Honesty

Academic honesty is fundamental to the activities and principles of a university. All members of the academic community must be confident that each person's work has been responsibly and honorably acquired, developed and presented. Any effort to gain an advantage not given to all students is dishonest whether or not the effort is successful. The academic community regards academic dishonesty as an extremely serious matter, with serious consequences. When in doubt about plagiarism, paraphrasing, quoting or collaboration, consult your instructor

Course Schedule:

Week 1
Introduction to the Course

Week 2
Professionalization: Goal-setting
<u>Reading</u>
Read excerpt of *Getting Things Done* posted on Blackboard
<u>Assignment</u>
Come to this session with your "Project List." Choose at least one project from the list (perhaps the one that is most important to you/highest priority) and write a "Next Action" list for that project.

Week 3
Professionalization: Marketing Yourself as a Researcher
<u>Reading</u>
Excerpt from *Developing Transferable Skills: Enhancing Your Research and Employment Potential* by Pam Denicolo and Julie Reeves
"Effective Resumes and Cover Letters" from USC Career Center
http://careers.usc.edu/docs/handouts/Resume_Booklet_Small.pdf
<u>Class activity</u>
Come to class with 2 copies of your resume for a Peer Workshop. We will focus on how to translate and sell your research skills to different audiences.

Week 4
Professionalization: Connecting Research and Policy 1
How to write a policy memo
<u>Class activity</u>
Come to class with 3 ideas for your Policy Memo topic. The topics should be related to your work as a research assistant.

Week 5
Professionalization: Connecting Research and Policy 2
<u>Class activity</u>
Come to class with 2 copies of your draft Policy Memo for a Peer Workshop.

Week 6
Professionalization: Academic Storytelling
<u>Reading</u>
Excerpt from *Segregation by Design* by Jessica Trounstine

Week7
Mid-term Check-In with your Mentor and the professor

SPRING BREAK

Week 9
Professionalization: Writing Op-Eds I
Reading
"How to Write an Op-Ed" (http://newsoffice.duke.edu/duke_resources/oped)
"How to Write Op-Ed Columns"
(http://www.earth.columbia.edu/sitefiles/file/pressroom/media_outreach/OpEdGuide.doc)
Class activity
Come to class with 3 ideas for your Op-Ed topic. The topics should be related to your work as a research assistant.

Week 10
Professionalization: Writing Op-Eds II
Class activity
Come to class with 2 copies of your draft Op-Ed for a Peer Workshop.

Week 11
Professionalization: Visualizing Data
Reading
Excerpt from *The Visual Display of Quantitative Information* by Edward R. Tufte
Class Activity
Come to class with a brainstorm on how you might present the data you are collecting in your research activities.

Week 12
Professionalization: What are Conferences All About?
Reading
"Posters, Presentations, and Publishing," excerpt from *Empirical Research and Writing: A Political Science Student's Practical Guide* by Leanne Powner.
Assignment
Visit the website for the 2019 Meeting of the International Studies Association and look around. Find a panel that interests you and try to download the papers.

Week 13
Professionalization: Self-Assessment
Reading pages 236-257 in *Getting Things Done*
Class activity
Come to class with the current version of your Projects List. We will talk about progress made, how projects change and the power of identifying Next Actions.

Week 14
End-of-Semester progress meetings with your Mentor and the professor.

Finals Week
Presentations

APPENDIX A.2: Syllabus for *Applied Data Science for International Relations I*

Professor: Benjamin Graham

Course Description

This course provides an introduction to statistical computing and data visualization in R (http://cran.r-project.org). This course is the first of a two-course sequence in applied data science. The focus of this course is on how to make sense of data and make proper use of graphics, rather than the specifics of statistical modeling.

Statistical computing and data visualization have increasingly become crucial tools for making sense of international politics and international economics. Over the course of the semester, students will learn the basic but important programming language concepts such as variables, vectors, assignments, functions, data structures, and input and output. Students will gain experience with the practice of data cleaning, reshaping of data, basic tabulations and aggregations, as well as high quality visualization in R.

This course is strongly applied, and each student will complete a data visualization project related to the ongoing research of a School of International Relations faculty member. Professor Graham will match students to projects in Week 6 of the course. Many of the final projects from this course will become part of published research by SIR faculty members. Most of these final projects will involve the graphical presentation of descriptive statistics, such as kernel density plots, heat maps, and scatter plots. Faculty members who submit requests for data visualizations are strongly encouraged to attend the final sessions of the course when students present their final projects.

Learning Objectives:

By the end of this course, students should be able to:

1. Load data into R
2. Select appropriate visualizations for a given data set and descriptive concept to be communicated
3. Create data visualizations of descriptive concepts using R
4. Interact with a client in an iterative process to meet the client's data visualization needs.

Prerequisites:

Grading

Three Homework Assignments – 15%

Assignments will consist mostly of problem sets working in R that are related to each module. Assignments will be handed out in class and will be due at the beginning of class on the due date. Late submission will be accepted for half credit up to one week after they are due.

Required Doing – 5%

For all "required doing" in swirl, students must hand in the log file at the beginning of class on the day for which the "doing" is assigned. Graded for completion only.

Three In-Class Lab Assignments – 15%
Assignments will consist mostly of problem sets working in R that are related to each module. Assignments will be handed out in class and will be due at the beginning of class on the due date. Late submission will be accepted for half credit up to one week after they are due.

Midterm Evaluation – 20%
This is a take-home midterm exam.

Final Project – 40%
Students are expected to design a final project in coordination with an IR faculty member, to whom they will be assigned. Most projects will involve a visual presentation of international relations data, but students have a great deal of flexibility to tailor their projects to their own interests. Matching between students and IR professors with proposed projects will occur in Week 6 of the course. For students proposing their own project not in collaboration with an IR faculty member, an independent project proposal is due in Week 6. Students will present their final projects in the last two weeks of class.

Final Presentation – 5%

Online Resoures
Swirl (Learn R, in R): http://swirlstats.com/
R Cheat Sheets: https://www.rstudio.com/resources/cheatsheets/
Data Camp:
https://www.datacamp.com/groups/35ff13b09ec1ed7f9ee5d14c7e518a2f01e28bf7/invite
Cookbook for R: http://www.cookbook-r.com/
Quick R: http://statmethods.net/
ggplot2. 2.1.0: http://docs.ggplot2.org/current/index.html
For demos: http://www.twotorials.com/

Recommended Books
Adler, J. (2010), R in a Nutshell, O'Reilly.
Wickham, H. (2016). Ggplot2: Elegant Graphics for Data Analysis (second edition)
Chang, W. (2013) R Graphics Cookbook

Lecture Topics, Assignment Dates, and Readings

Week 1
Lecture 1: Introducing R, Rstudio
Prior to class:
Download R from: http://www.cran.r-project.org

- Download Rstudio from: https://www.rstudio.com/products/rstudio/download/
- Viewing: "001 How to download and install r "(http://www.twotorials.com/)

Week 2 *(No Class)*

Week 3
Lab 1: Introducing R, Rstudio

- Prior to class:
- Required Viewing: 008; 010; 013; 014; 015 (http://www.twotorials.com/)
- Required Reading: "Creating new variables" (http://statmethods.net/management/variables.html)
- Required Reading: "Data Types" (http://statmethods.net/input/datatypes.html)
- Required Viewing/Doing: Install swirl (http://swirlstats.com/students.html) on R and complete "R Programming": lessons 1(Basic Building Blocks); 2(Workspace and Files); 3(Sequences of Numbers); 4(Vectors); 6(Subsetting Vectors); and 7(Matrices and Data Frames).

NOTE: Take notes on what confuses you when you work through these swirl sessions. Bring your questions in to lab for help.

- Optional Reading: "Factors" (http://www.cookbook-r.com/Manipulating_data/)

In lab, we will work on the following☹

1) Installing and Using R packages:
    - install.packages( )
    - library( )
    - getwd( )
    - setwd( )
2) Accessing Data in Rstudio
    - library(foreign)
    - read.dta / read.csv
3) Assigning Output to Objects
4) Vectors and Matrices
5) Recoding and Adding New Variables
    - $
6) Data Types in R
    - numeric; integer; character; factor

Homework 1 will be handed out at the end of class

Week 4
Lecture 2: Basic Statistical Routines
*** Homework 1 Due at the beginning of class***
Prior to class:

- Required Viewing: 022; 024(http://www.twotorials.com/)
- Required Reading: "Frequencies and Crosstabs" (excluding "Tests of Independence") http://statmethods.net/stats/frequencies.html
- Required Skimming: "HOW TO IMPROVE YOUR RELATIONSHIP WITH YOUR FUTURE SELF" by Bowers and Voors. http://www.jakebowers.org/PAPERS/11-BOWERS-RCP-363.pdf
- Optional Reading: "General" (http://www.cookbook-r.com/Manipulating_data/)

Week 5
Lab 2: Basic Statistical Routines
Prior to class:

- Required Doing: Swirl: Complete "R Programming": lessons 5(Missing Values); 8(Logic)
- Required Viewing: 019; 032; 033(http://www.twotorials.com/)
- Optional Reading: "Sequential Data" (http://www.cookbook-r.com/Manipulating_data/)

In lab we will work through:

1) summaries: summary( ); str( )
2) means: mean( )
3) variances: var( )
4) Logical expressions
    - equality ==
    - inequality ==
    - and &
    - or |
    - not !
5) Tables and Cross Tabulations: table( )
6) Missing Data
    - is.na

Homework 2 will be handed out at the end of class.

Week 6
Lecture 3: Data Visualization I
Homework 2 due at the beginning of class

- Required Reading: "geom_bar"; "geom_boxplot"; "geom_density"; "geom_errorbarh";"geom_dotplot" (http://docs.ggplot2.org/current/index.html)
- Required Viewing: "A Backstage Tour of ggplot2 with Hadley Wickham" (https://www.youtube.com/watch?v=RHu5vgBZ1yQ) (optional)

Week 7
Lab 3: Data Visiualization I

Data Visualization I

1) ggplot2 package
   - scatterplots; histograms;bar and line plots
   - density plots, boxplots
   - plotting means and error bars

- Reading: “Bar and line graphs”; “Plotting means and error bars”; “Plotting distributions”; “Scatterplots” (http://www.cookbook-r.com/Graphs/)
- Viewing: “Plotting with ggplot2:Part1” (https://www.youtube.com/watch?v=HeqHMM4ziXA)

Week 8
*** Students receive their final project assignments***

- Reading: “guide_legend”; “scale_colour_brewer”; “scale_manual”; “scale_x_discrete”(scale_y_discrete); “labs” (http://docs.ggplot2.org/current/index.html)
- Viewing: “Plotting with ggplot2:Part2” (https://www.youtube.com/watch?v=n8kYa9vu1l8)

Week 9
Lecture 4: Data Visualization II
Homework 3: Hand-drawn sketches and 1 paragraph description of proposed visualization(s) is due at the beginning of class.
Group activity to critique and develop proposed visualizations

Week 10
Lab 4: Data Visualization II
Midterm Exam will be handed out at the beginning of class

1) ggplot2 package
   - titles; axes; legends; facets; lines, and colors

- Reading: “Colors”; “Titles”; “Axes”; “Legends”; “Lines”; “Facets”; “Multiple graphs on one page” (http://www.cookbook-r.com/Graphs/)

Week 11
Lecture/Lab 5: Data Visualization III: Special Topics
Midterm Take Home Exam due at the beginning of class

- The topics for this session will be determined by the nature of students’ final projects

Week 12
Lecture/Lab 6: Data Visualization III: Special topics

- The topics for this session will be determined by the nature of students final projects

Week 13
Collaborative work session on final projects

Week 14
Collaborative work session on final projects

Week 15
Presentation and critique of final projects

Final Exam Week:
Revised final projects are due on the day of the final.

APPENDIX A.2: Syllabus for *Applied Data Science for International Relations II*

Professor: Benjamin Graham

This course provides an introduction to data analysis and the graphical presentation of statistical relationships using R (http://cran.r-project.org). This is the second part of a data science sequence following Applied Data Science for International Relations I. The goals of this course are to enable students to manage and clean data, including reshaping and merging datasets; to understand the intuition behind regression analysis; and create publication-grade figures in R that graphically display regression results.

This course centers around two large projects.

In the first, each student contributes a new dataset to the International Political Economy Data Resource, which is a compilation dataset maintained by the SPEC Lab. Each student identifies a useful country year data source that is not currently included in the resource, and adds that dataset to the resource. This requires soliciting permission from the dataset authors, downloading the data, cleaning and merge-prepping the data, merging it into the resource, and writing a new entry for the codebook.

In the second project, each student creates a publication-grade figure that graphically presents the regression results from an actual working paper or book chapter written by a social science faculty member. Your professor will match faculty and students, and faculty who solicit figures for their work are encouraged to attend the final sessions of the course when students present their work.

Pre-requisites:
Introduction to International Relations; either Applied Data Science for International Relations I OR Introduction to Data Analysis, OR equivalent.

Assignments & Grading
World Development Indicators Practice Assignment: 15%

Data Management Project: 30%
Each student must contribute a new country-year dataset to the IPE Data Resource. See above.

Graphical Presentation of Regression Results Project: 40%
Students will produce publication-grade figures that graphically present the results of the statistical analysis in an actual working paper or book chapter authored by a social science faculty member. Many of these figures will become part of actual published research. See above.

Required Doing in Swirl: 10%
Students are required to hand in a log file of the required doing at the beginning of class on the day for which the required doing is assigned.

Final Presentation: 5%

Online Resources

Swirl (Learn R, in R): http://swirlstats.com/
Cookbook for R: http://www.cookbook-r.com/
Quick R: http://statmethods.net/

Required Text:

Kosuke Imai, *Quantitative Social Science: An Introduction*

Recommended Texts:

Adler, J. (2010), R in a Nutshell, O'Reilly.
Wickham, H. (2012). Ggplot2: Elegant Graphics for Data Analysis
Chang, W. (2013) R Graphics Cookbook

Statement for Students with Disabilities

Statement on Academic Integrity

Lecture Topics, Assignment Dates, and Readings

Week 1:
Course Introduction and Planning

Week 2:
Lecture 1: Data Cleaning I
Prior to class:

- Required Viewing: 046; 047; 069; 078 (http://www.twotorials.com/)
- Required Reading: "Subsetting Data" (http://statmethods.net/management/subset.html)
- Required Reading: "Sorting Data" (http://statmethods.net/management/sorting.html)
- Recommended Reading: "Data Processing with dplyr & tidyr" (https://rpubs.com/bradleyboehmke/data_wrangling)
- Assignment 1 will be handed out (WDI Practice Exercise).

Week 3:
Lab 1: Data Cleaning I
Prior to class:

- Required viewing: 042; 028; 054; 079 (http://www.twotorials.com/)
- Required Doing: Swirl: Complete course "Getting and Cleaning Data": 1(Manipulating Data with dplyr); 2(Grouping and Chaining with dplyr)

In class exercise:

1) dplyr package:
    - select( ); filter( ); arrange( ); mutate( ); summarise( ); group_by( )

Week 4:
Lecture 2: Data Cleaning II
****** Assignment 1 Due (WDI Practice Exercise)*****
Prior to class:

- Required Reading: "Introducing tidyr" (http://blog.rstudio.org/2014/07/22/introducing-tidyr/)
- Optional Reading: "Restructuring Data" (http://www.cookbook-r.com/Manipulating_data/)

Week 5:
Lab 2: Data Cleaning II
Prior to class:

- Required Reading: "Converting data between wide and long format" (http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/)
- Required Reading: "Reshaping Data" (http://statmethods.net/management/reshape.html)
- Required Doing: Swirl: Complete course "Getting and Cleaning Data": 3(Tyding Data with tidyr)

In class exercise:

1) reshape2 package:
    - melt( ); cast( )
2) tidyr package:
    - gather( )
    - separate( )
    - spread( )

Week 6:
Lecture 3: Simple Linear Regression (intuition)

- Required Reading:

Week 7:
Lab 3: Simple Linear Regression (intuition)
******Project 1 is due at the beginning of class******

- In Class:
    - Swirl: Complete "Regression Models" course: Lesson 2(Residuals); 3(Least Squares Estimation)

Week 8:
Lecture 4: Multiple Regression (intuition)

- Required Reading:

Week 9:
Lab 4: Multiple Regression (intuition)
In Class:

Swirl: Complete the following lessons: 5 - Introduction to Multivariable Regression; 6 - MultiVar Examples; 7 - MultiVar Examples2; 7 - MultiVar Examples3

Week 10:
Lecture 5: Graphical presentation of regression results
Scatter plots with lines of best fit
Ladder plots
Varying the leads plots
Regression discontinuities & splines

******Project 2 assignments are made******

Week 11:
Lab 5: Graphical presentation of regression results
In Class: Ladder plot exercise

*****Hand drawn sketch of final projects is due******

Week 12:
Lab 6: Collaborative Work Session

Week 13:
Lab 7: Collaborative Work Session

Week 14:
Final Project Presentations & Critique

Week 15:
Final Project Presentations & Critique

Final Exam Week: *Revised final projects are due on the day of the final.*

Introduction to R

Miriam Barnum

August 31, 2018

1 Plan for this session

Today we'll get started with R, and cover working directories, installing packages, arithmetic, logical operators, basic indexing, data types, basic functions such as sum(), min(), max(), mean(), etc.

Thanks to Therese Anders, who orignially developed most of the material covered here!

2 Getting started in R

R is a programming language for statistical computing and data visualization. It provides an open source alternative to commercial statistical packages such as Stata or SPSS. R is maintained and developed by a vibrant community of programmers and statisticians and offers packages for just about any statistical task imaginable.

In this workshop, we will be using R together with the integrated development environment (IDE) **RStudio**. In addition to offering a 'cleaner' programming development than the basic R editor, RStudio offers a large number of added functionalities for integrating code into documents, built-in tools and web-development. To get started, please download the latest version of RStudio and R from this website:

https://www.rstudio.com/products/rstudio/download/

3 Getting Help

The key to learning R is: **Google**! This workshop will give you an overview over the basic functions of the language, but to really learn R you will have to actively use it yourself, struggle, ask questions, and google! The R mailing list server and other pages such as http://stackoverflow.com offer a rich archive of questions and answers by the R community. For example, if you google "recode data in r" you will find a variety of useful pages explaining how to do this on the first page of the search results. Also, don't be surprised if you find a variety of different ways to execute the same task.

RStudio also has a useful help menu. In addition, you can get information on any function or integrated data set in R through the console, for example:

```
?plot
```

Also, there are a lot of free R comprehensive guides, such as the Quick-R at http://www.statmethods.net or the R cookbook at http://www.cookbook-r.com.

4 Working Directories

R needs to know where to look for files if you want to read data and where to store files if you want to write data. This includes the code you will be typing in this session. The `getwd()` command returns the current working directory. We can change the working directory with `setwd()` (see below).

Think of your computer as a filing cabinet. As you work in R, you will be writing a number of R scripts, that are essentially text files with commands for R. In order to execute these files, we need to tell R where to look for the list of commands we want to execute. Setting a working directory is analogous to telling R in which file in the filing cabinet we stored our document (code or data) and into which file in the filing cabinet to put new documents (such as graphs, new data frames, new code).

```
getwd() # Prints the current working directory
setwd("/Users/miriam/Desktop")
getwd()
```

Important for Windows users: In R, the backslash is an escape character. Therefore, entering file paths is a little different in Windows than on a Mac. On a windows machine you would enter:

```
setwd("C:/Documents and Settings/Data")
```

OR

```
setwd("C:\\Documents and Settings\\Data")
```

5 Packages

R has many built-in functions, but the true power behind R lies in the large variety of external packages that are available to add functionality for tasks such as visualization, statistical computing, webscraping, data manipulation, and interaction with other programs. Today, we'll learn how to install different packages via the `install.packages()` function and how to load in the package during our session using the `library()` function.

Suppose we wish to load a data file produced by another statistical software such as Stata or SPSS. The `foreign` package is useful when dealing with files from other statistical software. If you work with colleagues who use Stata, you will find this package incredibly helpful. If your colleagues save their data files in the Stata `.dta` format, you can use the foreign package to read in the dataset in R.

You need the quotations around the name of the package when using the `install.packages()` function because R won't recognize the package otherwise. You don't need quotations around the package name when using the `library()` function because R recognizes the package name now. Every time you begin a new R session, you will need to load in the packge using `library()`. Otherwise, R won't recognize the `read.dta()` function.

```
install.packages("foreign")
library(foreign)
```

6 Arithmetic in R

You can use R as a calculator!

	Operator	Example
Addition	`+`	`2+4`
Subtraction	`-`	`2-4`
Multiplication	`*`	`2*4`
Division	`/`	`4/2`
Exponentiation	`^`	`2^4`
Square Root	`sqrt()`	`sqrt(144)`
Absolute Value	`abs()`	`abs(-4)`

```
4*9
```

```
## [1] 36
```

```
sqrt(144)
```

```
## [1] 12
```

```
4^5
```

```
## [1] 1024
```

```
4 ^ 5
```

```
## [1] 1024
```

Note that spaces between operators do not matter. So `4 + 5` is the same as `4+5`. We usually add spaces to make the code more legible. Note, however, that spaces do matter when defining character values. We will talk about character values in the next `R` lab.

Just like any regular calculator, you have to pay attention to the order of operations! Example:

```
6 * 8 - sqrt(7) + abs(-10) * (4/5)
```

```
## [1] 53.35425
```

```
6 * (8 - sqrt(7)) + abs(-10) * (4/5)
```

```
## [1] 40.12549
```

6.1 Logical operators

Arithmetic operations will yield a numerical output. (Correctly specified) logical operations yield one of two results: `TRUE` (or `T`) or `FALSE` (`F`). Logical operators are incredibly helpful for subsetting data, any type of exploratory analysis, data cleaning and/or visualization task.

	Operator
Less than	`<`
Less than or equal to	`<=`
Greater than	`>`
Greater than or equal to	`>=`
Exactly equal to	`==`
Not equal to	`!=`
Not `x`	`!x`
`x` or `y`	`x | y`
`x` and `y`	`x & y`
`%in%`	Testing whether a value is contained within a set.

Here are some simple examples:

```
4 > 2
```

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

```
4 <= 2
```

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

```r
5 == 2^2
```

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

The examples above are very straightforward because we already know the result. The true value of logical operators lies in their ability to be applied across large amounts of data.

Let;s take a set (or collection of numbers)–in `R`, the technical term is a vector–and use logical operators on them.

```r
2 == c(2, 3, 4)
```

```
## [1]  TRUE FALSE FALSE
```

```r
2 %in% c(2, 3, 4)
```

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

```r
5 %in% c(2, 3, 4)
```

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

We can use the 'not' operator `!` to check if two values are not equal, or to check if the opposite of an entire statement is true.

```r
5 != 4
```

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

```r
!(5 %in% c(2, 3, 4))
```

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

Exercise 1 What to you think is the output of the following operation?

`2 = 4`

And the following?

`2 == 4`

We can also use logical operators to compare text or `character variables`.

Exercise 2 What to you think is the output of the following two operations?

```r
"Apples" == "Oranges"

"USC" > "UCLA"
```

7 Objects in R

7.1 Assigning values to objects

`R` stores information as an *object*. You can name objects whatever you like. Just remember to not use names that are reserved for built-in functions or functions in the packages you use, such as `sum`, `mean`, or `abs`. Most of the time, `R` will let you use these as names, but it leads to confusion in your code.

A few things to remember

- Do not start object names with numbers: `result3` is fine, but `3result` wont work as a name.
- Do not use special characters such as `$` or `%`. Common symbols that are used in names include `.` or `_`.
- Remember that `R` is case sensitive.

- To assign values to objects, we use the assignment operator `<-`. Sometimes you will also see `=` as the assignment operator. This is a matter of preference and subject to debate among `R` programmers.
- The `#` symbol is used to include `comments` in the code. Nothing typed after `#` on the same line will be included in `R`'s calculations.

Below, `R` stores the result of the calculation in an object named `result`. We can access the value by referring to the object name.

```r
myresult <- 5/3
myresult # Implicitly printing the output
```

```
## [1] 1.666667
```

```r
print(myresult) # Explicitly printing the output
```

```
## [1] 1.666667
```

If we assign a different value to the object, the value of the object will be changed.

```r
myresult <- 5-3
myresult
```

```
## [1] 2
```

7.2 Vectors

`R` can deal with a variety of data types, including vectors, scalars, matrices, data frames, factors, and lists. Today, we will focus on vectors.

A **vector** is the simplest type of data you can work with in `R`. "A vector or a one-dimensional array simply represents a collection of information stored in a specific order."" (Imai 2016: 6) It is essentially a list of data of a single type (either numerical, character, or logical). To create a vector we use the function `c()` ("concatenate") to combine separate data points. The general format for creating a vector in R is as follows:

`name_of_vector <- c("what you want to put into the vector")`

Suppose, we have data on the population in millions for 5 the five most populous countries in 2016.

```r
pop1 <- c(1379, 1331, 325, 258, 207)
pop1
```

```
## [1] 1379 1331  325  258  207
```

We can use the function `c()` to combine two vectors. Suppose we had data on 5 additional countries.

```r
pop2 <- c(194, 187, 161, 142, 127)
pop <- c(pop1, pop2)
pop
```

```
##  [1] 1379 1331  325  258  207  194  187  161  142  127
```

7.3 Variable types

There are four main variable types that you should be familiar with:

- **Numerical**: Well, any number.
- **Character**: This is what data (and other programming languages such as Python) calls a string. We typically store any alphanumeric data that is not ordered as a character vector.
- **Logical**: A collection of `TRUE` and `FALSE` values.

- **Factor**: Think about it as a categorical variable. People might use a factor type instead of a numerical type if they want to imply that the categories are ordered, or if they want to restrict values of the variable to a set list of categories.

First, lets check which variable type was used to store our population data. The below output tells us that the object `pop` is of class numeric, and has the dimensions `[1:10]`, that is 10 elements in one dimension.

```
str(pop)
```

```
##  num [1:10] 1379 1331 325 258 207 ...
```

Suppose, we wanted to add data on the country names. We enter this data in character format. To save time, we will only do this for the five most populous countries.

```
cname <- c("CHN", "IND", "USA", "IDN", "BRA")
str(cname)
```

```
##  chr [1:5] "CHN" "IND" "USA" "IDN" "BRA"
```

Now, lets code a logical variable that shows whether the country is in Asia or not.

```
asia <- c(TRUE, TRUE, F, T, F)
str(asia)
```

```
##  logi [1:5] TRUE TRUE FALSE TRUE FALSE
```

Lastly, we define a factor variable for the regime type that can take one of three values: Full Democracy, Flawed Democracy, Autocracy.

```
regime <- c("Autocracy", "FlawedDem", "FullDem", "FlawedDem", "FlawedDem")
regime <- as.factor(regime)
str(regime)
```

```
##  Factor w/ 3 levels "Autocracy","FlawedDem",..: 1 2 3 2 2
```

Data types are important! `R` will not perform certain operations if you don't get the variable type right. The good news is that we can switch between data types. This can sometimes be tricky, especially when you are switching from a factor to a numerical type[1]. We won't go into this too much here, but just remember: Google is your friend!

Let's convert the factor variable `regime` into a character. Also, for practice, lets convert the `pop` variable to numeric.

```
regime <- as.character(regime)
str(regime)
```

```
##  chr [1:5] "Autocracy" "FlawedDem" "FullDem" "FlawedDem" ...
```

```
asia <- as.character(asia)
str(asia)
```

```
##  chr [1:5] "TRUE" "TRUE" "FALSE" "TRUE" "FALSE"
```

```
asia <- as.logical(asia)
str(asia)
```

```
##  logi [1:5] TRUE TRUE FALSE TRUE FALSE
```

Exercise3: Why won't `R` let us do the following?

```
no_good <- (a,b,c)
```

[1]Sometimes you have to do a work around, like switching to a character first, and then converting the character to numeric. You can concatenate these commands: `myvar <- as.numeric(as.character(myvar))`.

```
no_good_either <- c(one, two, three)
```

Exercise 4: What's the difference? (Bonus: What do you think is the type of the output vector?)

```
diff <-c(TRUE,"TRUE")
```

Exercise 5: What is the type of the following vector?

```
vec <- c("1", "2", "3")
```

7.4 Vector operations

You can do a variety of fun things like have R print out particular values or ranges of values in a vector, replace values, add additional values, etc. We will not get into that here, but be aware that (for all practical purposes) if you can think of a data operation, `R` can probably do it.

We can do arithmatic operations on vectors! Let's use the vector of population counts we created earlier.

```
pop1
```

```
## [1] 1379 1331  325  258  207
```

```
pop1_double <- pop1 * 2
pop1_double
```

```
## [1] 2758 2662  650  516  414
```

Exercise 6: What do you think this will do?

```
pop1 + pop2
```

Exercise 7: And this?

```
pop_c <- c(pop1, pop2)
```

7.4.1 Functions

There are a number of special functions that operate on vectors and allow us to compute measures of location and dispersion.

	Function
`min()`	Returns the minimum of the values or object.
`max()`	Returns the maximum of the values or object.
`sum()`	Returns the sum of the values or object.
`length()`	Returns the length of the values or object.
`mean()`	Returns the average of the values or object.
`median()`	Returns the median of the values or object.
`var()`	Returns the varianve of the values or object.
`sd()`	Returns the varianve of the values or object.

```
min(pop)
```

```
## [1] 127
```

```
max(pop)
```

```
## [1] 1379
```

```r
mean(pop)
```

```
## [1] 431.1
```

Exercise 8: What do you think is the output of the following operation?

```r
length(pop)
```

Exercise 9: Using functions in `R`, how else could we compute the mean population value?

```
## [1] 431.1
```

7.5 Accessing elements of vectors

There are many ways to access elements that are stored in an object. Here, we will focus on a method called *indexing*, using square brackets as an operator.

Accesing the first element of the top 5 population vector and the corresponding country name vector.

```r
pop1[1]
```

```
## [1] 1379
```

```r
cname[1]
```

```
## [1] "CHN"
```

Accessing the second and fifth element of the population and the country name vector.

```r
pop[c(2,5)]
```

```
## [1] 1331  207
```

```r
cname[c(2,5)]
```

```
## [1] "IND" "BRA"
```

Assinging the first element of the population vector to a new object called `first`.

```r
first <- pop[1]
```

Making a copy of the country names ratings vector and deleting the last element. Note, that we will use the `length()` function to achieve the highest level of *generalizability* in our code.

```r
cname_copy <- cname
cname_copy[-5]
```

```
## [1] "CHN" "IND" "USA" "IDN"
```

Exercise 10: Can you think of another way to do this? What if we didn't know how long the vector is?

```
## [1] 5
```

```
## [1] "CHN" "IND" "USA" "IDN"
```

Indexing can be used to alter values in a vector. Suppose, we notice that we wrongly entered the fifth element of the regime type vector (or the regime type changed).

```r
regime
```

```
## [1] "Autocracy" "FlawedDem" "FullDem"   "FlawedDem" "FlawedDem"
```

```r
regime[5] <- "FullDem"
regime
```

```
## [1] "Autocracy" "FlawedDem" "FullDem"   "FlawedDem" "FullDem"
```

Exercise 11: We made even more mistakes when entering the data! We want to subtract 10 from the third and fifth element of the top 5 population vector. *How would you do it*?

Workshop: Data Visualization with ggplot2 (Part I)

Therese Anders

April 7, 2017

1 Introduction

This workshop provides an introduction to data visualization in R using the ggplot2 package. The two part workshop is designed to give undergraduate and graduate research assistants a first course in producing publication ready graphs in R. The first part of the workshop introduces univariate graphical data summaries, parameters that control the appearance of the plot, the visualization of data across multiple groups, and how to save plots.

In this first workshop, we use data from the World Development Indicators. Specifically, we look at different indicators for the energy consumption of all countries in the WDI dataset for the past 25 years. For the sake of time, I downloaded and cleaned the data prior to the workshop. The code to clean the data (`clean_wdi.R`) is available on the GitHub repository.

1.1 Review: Reading the data into R

We start by a) setting the working directory, b) installing the ggplot2 package, and c) loading the ggplot2 package into the environment.

```
setwd("/Users/thereseanders/Documents/UNI/USC/Spring_2017/SPECRTraining/Part1")
#install.packages("ggplot2")
library(ggplot2)
```

Now, we can read the file `wdi_cleaned.csv`. Remember, that in order to read `.csv` files with the `read.csv()` function, you need to first load the `foreign` package.

```
library(foreign)
dat <- read.csv("./data/wdi_cleaned_part1.csv",
                stringsAsFactors = F)
```

1.2 Getting an overview of the data

The dataset contains 5425 observations on 5 variables. We can tell R to give us an overview of the data using the `str()` function. We can also take a look at the dataset in a spreadsheet format with the `View()` function.

```
str(dat)
```

```
## 'data.frame':    5425 obs. of  5 variables:
##  $ country           : chr  "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##  $ year              : int  1992 1992 1992 1992 1992 1992 1992 1992 1992 1992 ...
##  $ electricity_pop   : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ energyuse_pop     : num  NA 418 884 NA NA ...
##  $ renewable_energyuse: num  62.24 46.03 0.29 NA NA ...
```

The dataset contains the following variables:

- `year`: Variable coding the year of observation.
- `country`: Variable coding the country of observation.
- `electricity_pop`: Access to electricity (% of population).

- `energyuse_pop`: Energy use (kg of oil equivalent per capita).
- `renewable_energyuse`: Renewable energy consumption (% of total final energy consumption).

2 ggplot2 package

The `ggplot2` package was developed by Hadley Wickham based on Leland Wilkinson's "grammar of graphics" principles. According to the "grammar of graphics," you can create each graph from the following components: "a data set, a set of geoms–visual marks that represent data points, and a coordinate system" (Data Visualization with ggplot2 Cheat Sheet.

For most applications, the code to produce a graph in `ggplot2` is roughly structured as follows:

`ggplot(data = , aes(x = , y = , color = , linetype = )) +`

`geom()` +

`[other graphical parameters, e.g. title, color schemes, background]`

- `ggplot()`: Function to initiate a graph in `ggplot2`.
- `data`: Specifies the data frame from which the plot is produced.
- `aes()`: Specifies aesthetic mappings that describe how variables are mapped to the visual properties of the graph. The minimum value that needs to be specified (for univariate data visualization) is the `x` parameter, where `x` specifies the variable to be plotted on the x-axis. Analogously, the `y` parameter specifies the variable to be plotted on the y-axis. Other examples include the `color` parameter, which specifies the variable to be onto different colors, or the `linetype` parameter, which specifies the variable to be mapped onto different line types in case of line graphs.
- `geom()`: Specifies the type of plot to use. There are many different geoms ("geometric objects") to be specified with the `geom()` layer. Some of the most common ones include `geom_point()` for scatterplots, `geom_line()` for line graphs, `geom_boxplot()` for Boxplots, `geom_bar()` for bar plots for discrete data, and `geom_histogram()` for continuous data.

For an overview of the most important functions and geoms available through `ggplot2`, see the `ggplot2` cheat sheet.

3 Univariate visualizations

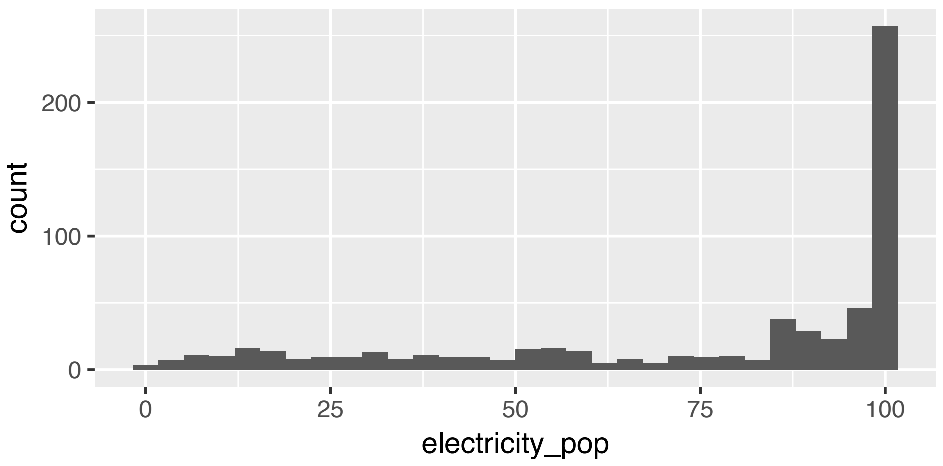
3.1 Histograms

Histograms graph the distribution of continuous variables. In this first example, we graph the distribution of the variable `electricity_pop`. Note that because `electricity_pop` specifies a percentage, its value is bound between 0 and 100.

```
summary(dat$electricity_pop)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00   53.44   92.68   75.74  100.00  100.00    4789
```

```
ggplot(dat, aes(electricity_pop)) +
  geom_histogram()
```



Question 1 Can you make sense of this graph? What is plotted on the x-axis? What is plotted on the y-axis? What specifies the width of each bar? What specifies the height of each bar?

A histogram plots the distribution of a variable. The x-axis specifies the values of the variable. The y-axis specifies the number of observations for each value (or group of values) of the variable. The width of the bar specifies which values of the variable are grouped into one bin. The height of the bar specifies the number of observations in each bin.

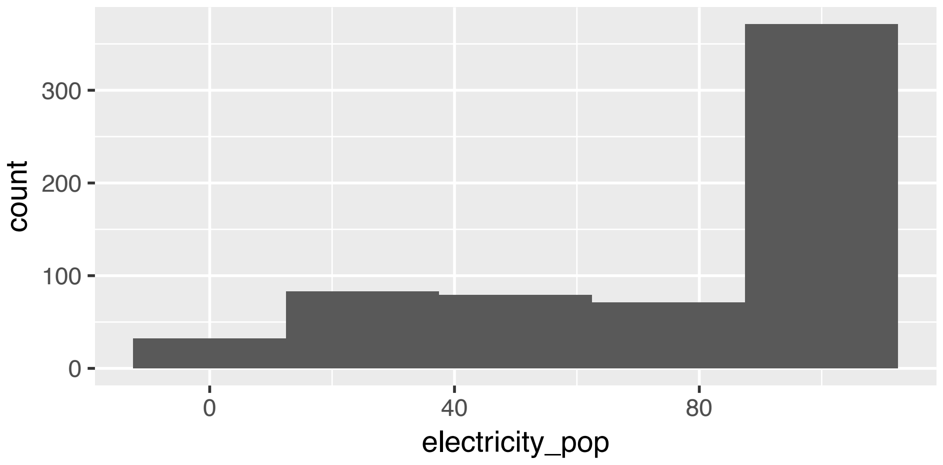
Question 2 Which conclusions do you draw from the histogram above about the distribution of the availability of electricity in the world?

The distribution is not normal (i.e. not a bell curve). It is skewed to the left. There are a lot more observations at the upper than the lower end of the scale, i.e. more country-years have high levels of availability of electricity than low levels.

3.1.1 Adjusting the number of bins

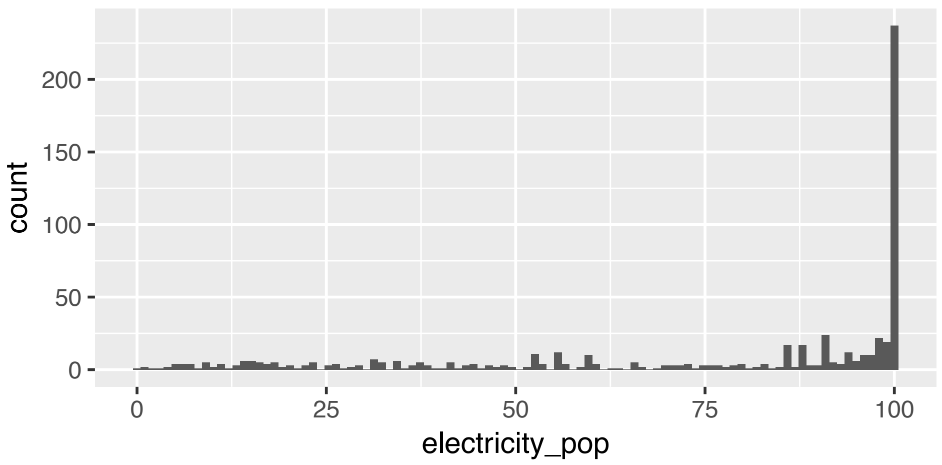
The default number of bins is 30, which means that the entire range of the variable (here 0 to 100) is split into 30 equally spaced bins. We can change the number of bins manually. In this example, since the variable is bound between 0 and 100, specifying `bins = 5` means that approximately values between 0-19 are grouped into one bin, values between 20-39 and so on.

```r
ggplot(dat, aes(electricity_pop)) +
  geom_histogram(bins = 5)
```



Question 3 How does the graph change if we specify `bins = 100`?

```r
ggplot(dat, aes(electricity_pop)) +
  geom_histogram(bins = 100)
```

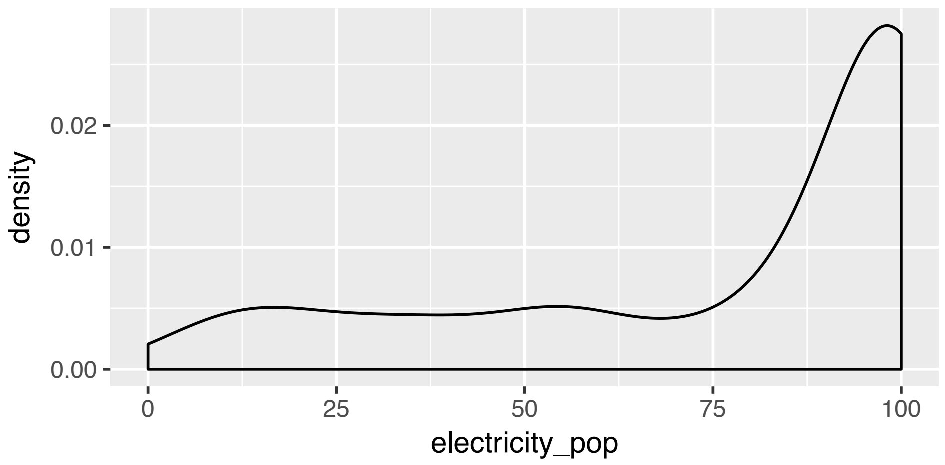


3.2 Density plots

We saw that the shape of the distribution is highly influenced by how many bins we specify. If we specify too few bins, we run the risk of masking a lot of variation within the bins. If we specify too many bins, we trade parsimony for detail–which might make it harder to draw conclusions about the overall distribution of the variable of interest from the graph.

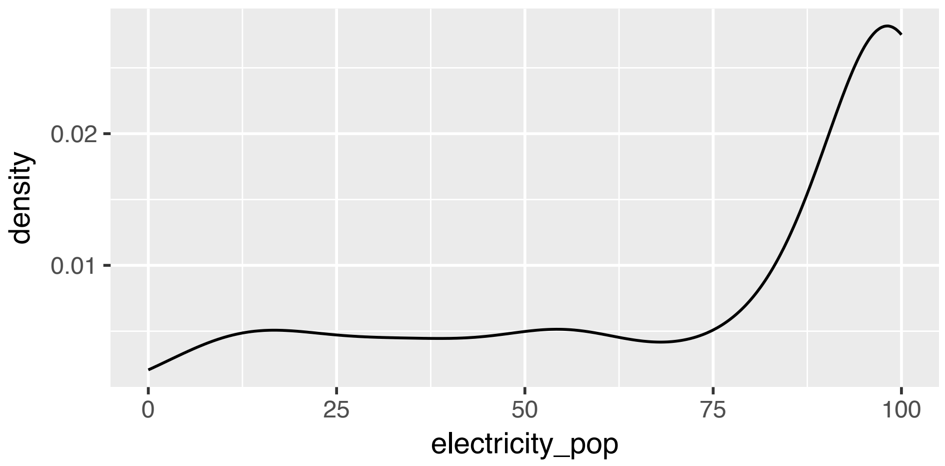
Density plots are continuous alternatives to histograms that do not rely on bins. We will cover details about the mechanics behind density plots and their estimation here. Just know that we can interpret the height of the density curve in a similar way that we interpreted the height of the bars in a histogram: The higher the curve, the more observations we have at that specific value of the variable of interest. In this first example, we use the `geom_density()` function to create the density plot.

```r
ggplot(dat, aes(electricity_pop)) +
  geom_density()
```



If you do not want the density graph to be plotted as a closed polygon, you can instead use the `geom_line()` geometric object function with the `stat = "density"` parameter.

```r
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density")
```



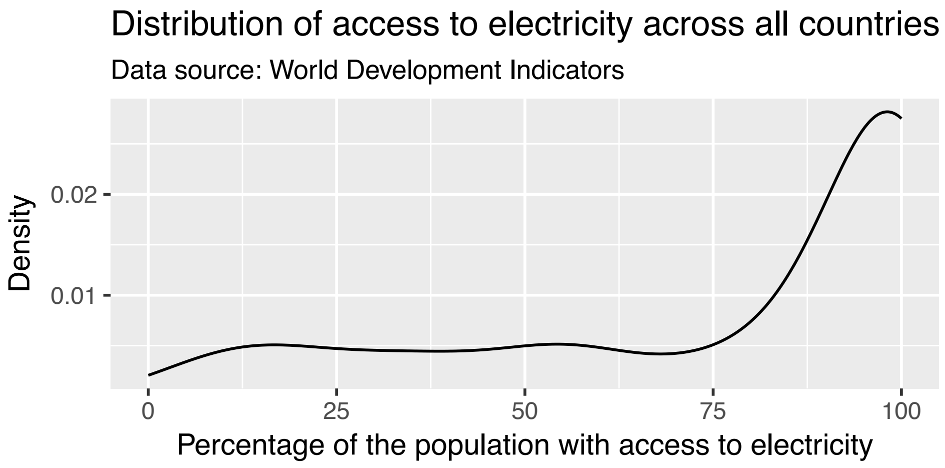
4 Controlling the appearance of graphs

The default graphs we have produced so far are not (yet) ready for publication. In particular, they lack informative labels. In addition, we might want to change the appearance of the graph in terms of size, color, linetype, etc.

4.1 Adding title, subtitle, and axes titles

We can specify titles and axes labels within the `labs()` argument.

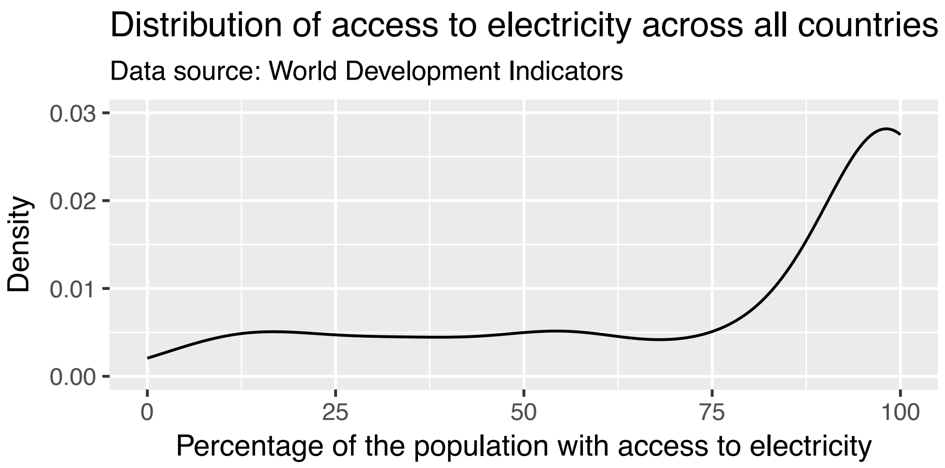
```r
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")
```



4.2 Adjusting the range of the axes

By default, `ggplot()` adjusted the y-axis to start not at zero but at approximately 0.2 to reduce the amount of empty space in the plot. This might confuse the viewer, as it visually underestimates the height of the density curve–in particular for lower values of the variable. We can manually adjust the range of the axes using the `coord_cartesian()` parameter.

```
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  coord_cartesian(ylim = c(0, 0.03))
```

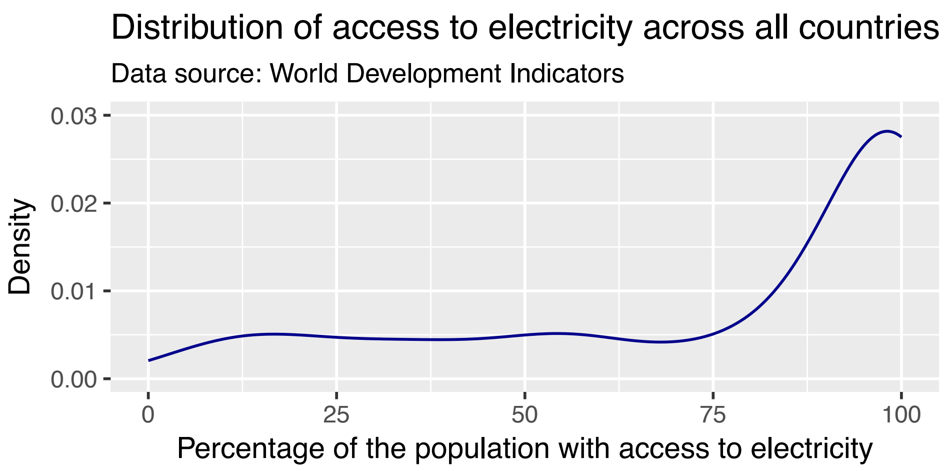


Caution!! You will sometimes see the command `scale_y_continuous(limits = c(0, 0.03))` instead of `coord_cartesian(ylim = c(0, 0.03))`. Note that these are not the same. `coord_cartesian()` only adjusts the range of the axes (it "zooms" in and out), while `scale_y_continuous(limits = c())` subsets the data. For density plots, this does not make a difference. But there are other examples where it alters the actual shape of the graph, rather than just the part of the graph that is visible.

4.3 Changing the color

Any changes to the appearance of the curve itself are made within the argument that specifies the geometric object to be plotted, here `geom_line()`. `R` knows many colors by name; for a great overview see http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf.

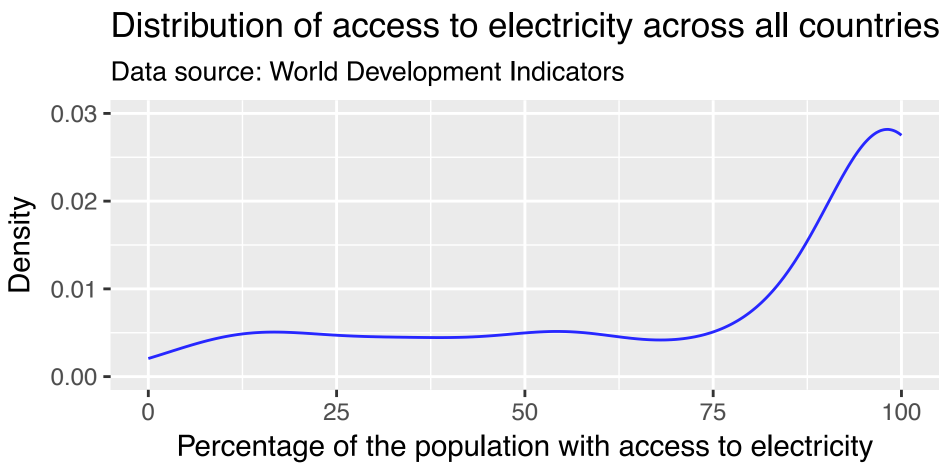
```
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "darkblue") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  coord_cartesian(ylim = c(0, 0.03))
```



We can also use hexadecimal or RGB (red, green, blue) strings to specify colors. There are plenty of online tools to pick colors and extract hexadecimal or RBG strings. One of my favorites is http://www.colorhexa.com. This online tool allows you to specify a color name, hexadecimal, or RGB string, and returns information on color schemes, complementary colors, as well as alternative shades, tints, and tones. It also offers a color blindness simulator.

Suppose, I like the general tone of the darkblue color above, but am worried that it is a bit too dark for my plot. I enter the color "darkblue" into the search field at http://www.colorhexa.com and look for a brighter alternative. Suppose I really like the color displayed in the second tile from the left on the tints scale. I can extract this color's hexadecimal value of `#2727ff` by hovering over the tile of that color.

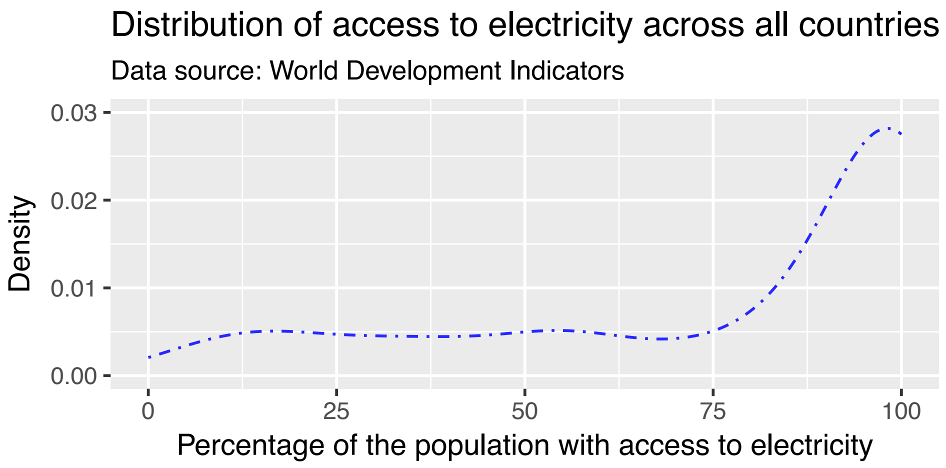
```r
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  coord_cartesian(ylim = c(0, 0.03))
```



4.4 Changing the line type

We can adjust the type of the line via the `linetype` parameter within `geom_line()`. For an overview of line types see http://sape.inf.usi.ch/quick-reference/ggplot2/linetype.

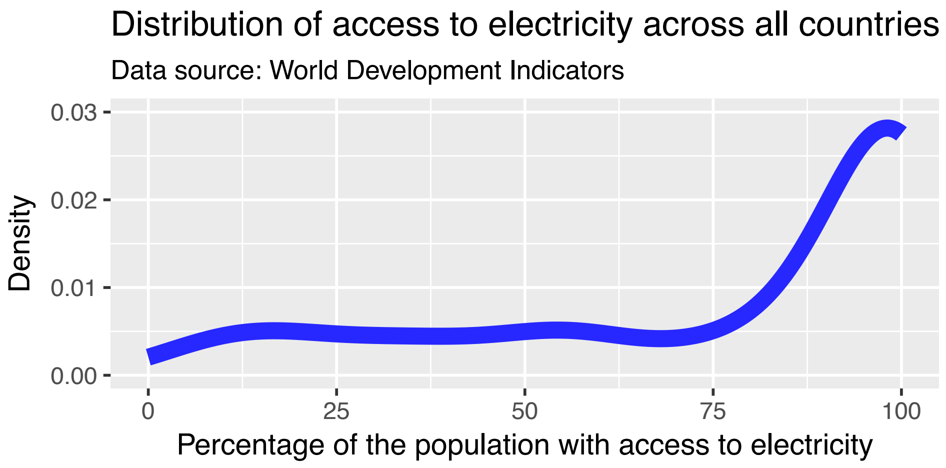
```r
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff", linetype = "dotdash") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  coord_cartesian(ylim = c(0, 0.03))
```



4.5 Changing the width of the line

We can adjust the width of the line via the `size` parameter within `geom_line()`. Note that the `size` parameter is universal in the way that it controls line width in line plots and point size in scatter plots.

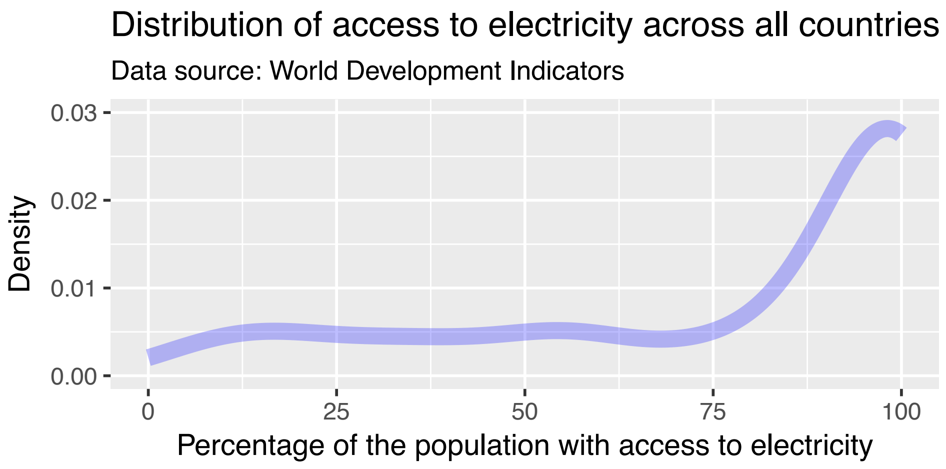
```r
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff", size = 3) +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  coord_cartesian(ylim = c(0, 0.03))
```



4.6 Changing the opacity of the line

We can adjust the opacity of the line via the `alpha` parameter within any geometric object. The `alpha` parameter ranges between zero and one. Adjusting the opacity of the geometric objects is especially important when plotting multiple lines (or objects) in the same graph to reduce overplotting.

```r
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff", size = 3, alpha = 0.3) +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  coord_cartesian(ylim = c(0, 0.03))
```



5 Graphing distributions across groups

5.1 Using different colors

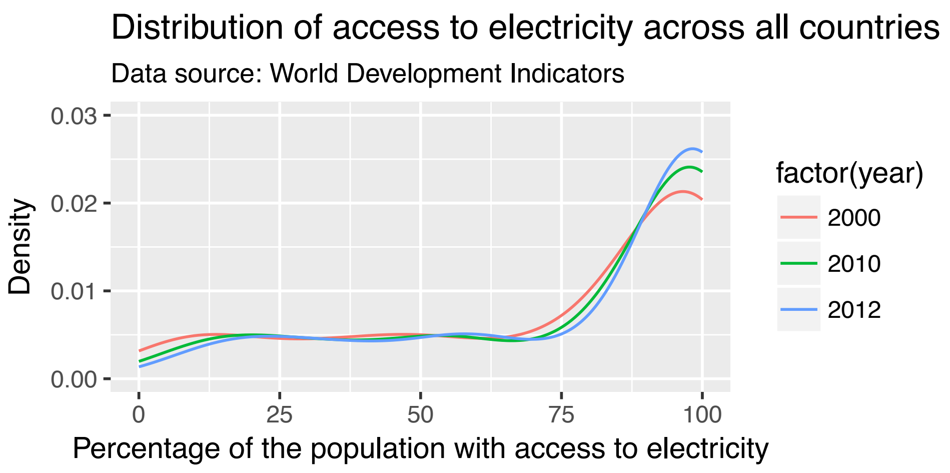
Sometimes, we want to compare distributions across different groups in our data set. Suppose, we wanted to assess how the distribution of the access to electricity changes over time. We have three years of observations for our variable: 2000, 2010, and 2012.

```r
table(dat$year[!is.na(dat$electricity_pop)])
```

```
##
## 2000 2010 2012
##  212  212  212
```

We pass a separate color to the distribution of the `electricity_pop` for each year by specifying the `color` parameter within the aesthetics. Since the year parameter is currently specified as a numerical variable, we also need to specify that `ggplot` should treat it as a `factor` (i.e. a categorical variable).

```r
ggplot(dat, aes(x = electricity_pop, color = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  coord_cartesian(ylim = c(0, 0.03))
```



Question 4 Review: What is the difference between specifying the `color` parameter outside the `aes()` argument versus within the `aes()` argument?

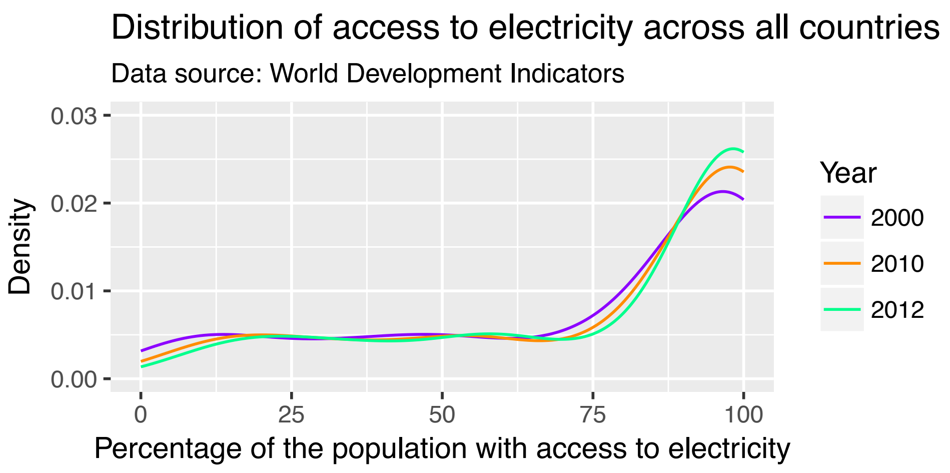
If the color parameter is specified outside the `aes()` argument, one color is passed all geometric objects of the same type. If the color parameter is specified within the `aes()` argument, different colors are passed to each value of the variable that is passed to the `color` parameter. A separate geometric object will be plotted for value–each in a different color.

Question 5 How would you interpret this plot? How did the access to electricity change over time?

Over time, the amount of country-years with high levels of access to electricity (90% to 100% of the population) has increased. In later years, there are fewer observations at very low levels of access to electricity and there are fewer country-years in which approximately 75% to 90% of the population had access to electricity.

We can use custom colors to distinguish between the three years using the `scale_color_manual()` function. This will change the colors in both the plot and the legend. Let us choose triadic colors to "darkorange" from http://www.colorhexa.com. Within the `scale_color_manual()` argument, we can also specify a name and labels for the legend.

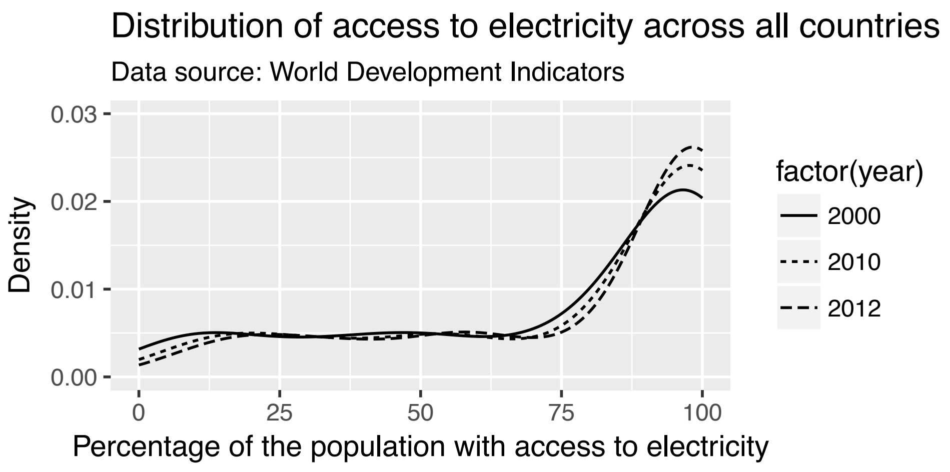
```r
ggplot(dat, aes(x = electricity_pop, color = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  coord_cartesian(ylim = c(0, 0.03)) +
  scale_color_manual(values = c("#8c00ff",
                                "#ff8c00",
                                "#00ff8c"),
                     name = "Year")
```



5.2 Using different linetypes

Many academic journals will only accept graphs on a gray scale. This means that color will not be enough to differentiate the three lines. We can use different line types instead by specifying the `linetype` parameter within the `aes()` argument.. This also makes the graph more color blind friendly.

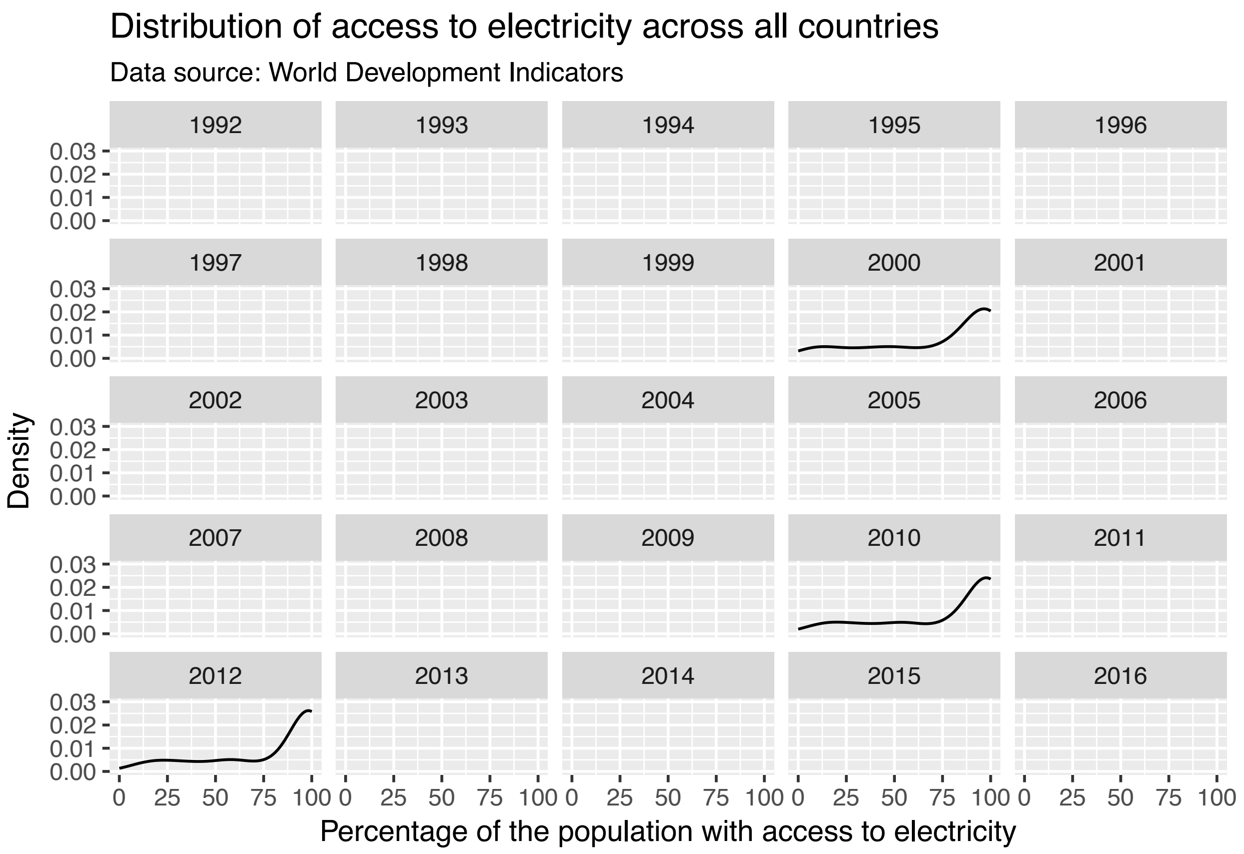
```
ggplot(dat, aes(x = electricity_pop, linetype = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  coord_cartesian(ylim = c(0, 0.03))
```



5.3 Faceting

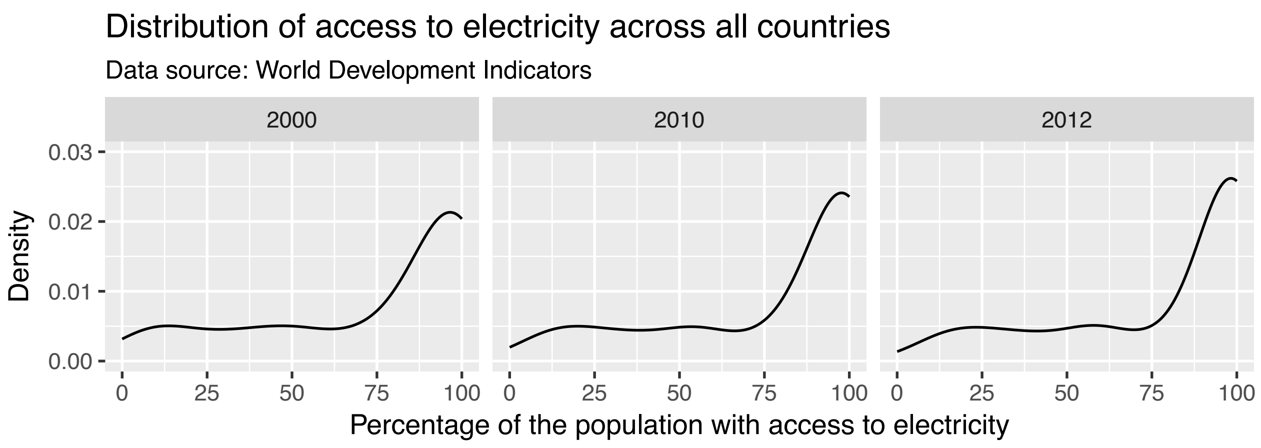
Another option to graph different groups is to use faceting. This means to plot each value of the variable upon which we facet in a different panel within the same plot. Here, we will use the `facet_wrap()` function. We could also use the `facet_grid()` which allows faceting across more than one variable.

```
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  coord_cartesian(ylim = c(0, 0.03)) +
  facet_wrap(~ year)
```



The `facet_wrap()` function draws a separate plot for each year in the data set. Since we only have data for 2000, 2010, and 2012, we subset the plot to omit empty panels. We can either create a new subsample data frame, or use the `subset()` command directly within `ggplot()`. Here, we explicitly choose the years 2000, 2010, and 2012. Alternatively, we could also subset to all non-missing observations on the variable `electricity_pop`.

```
ggplot(subset(dat, year %in% c(2000, 2010, 2012)), aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  coord_cartesian(ylim = c(0, 0.03)) +
  facet_wrap(~ year)
```



Question 6 For this particular application, which alternative to show the distribution of the variable across groups do you think is the most appropriate? Why?

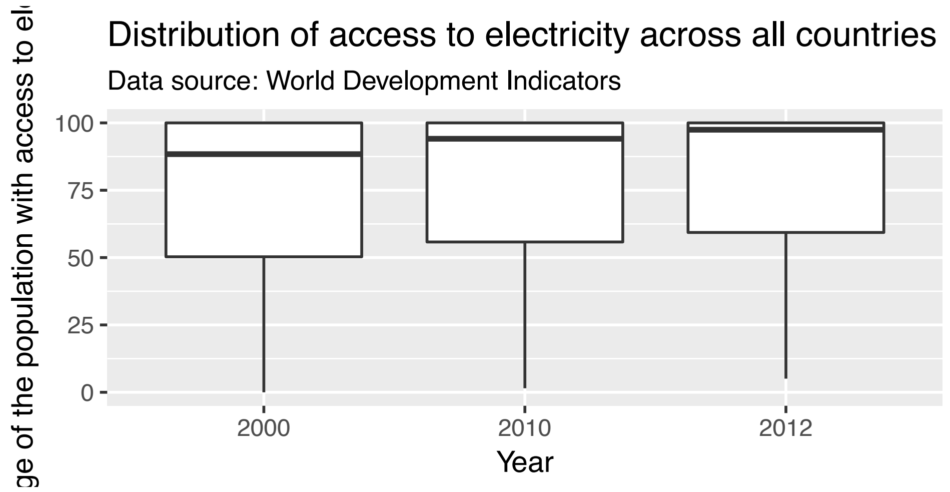
5.4 Boxplots

Another way to show the distribution of variables across groups are boxplots. Boxplots graph different properties of a distribution:

- The borders of the box denote the 25th and 75th percentile.
- The line within the box denotes the median.
- The position of the whiskers (vertical lines) denote the first quartile value minus 1.5 times the interquartile range and the third quartile value plus 1.5 times the interquartile range. We will not go into details here.
- Dots denote outliers (values that lie outside the whiskers), if applicable.

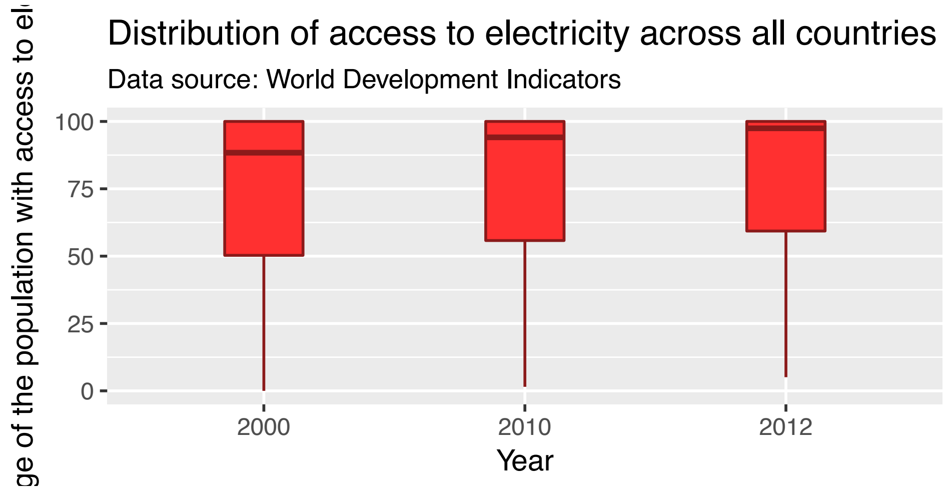
In `ggplot2` we can graph boxplots across multiple variables using the `geom_boxplot()` geometric object.

```r
ggplot(subset(dat, year %in% c(2000, 2010, 2012)),
       aes(x = factor(year), y = electricity_pop)) +
  geom_boxplot() +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Year",
       y = "Percentage of the population with access to electricity")
```



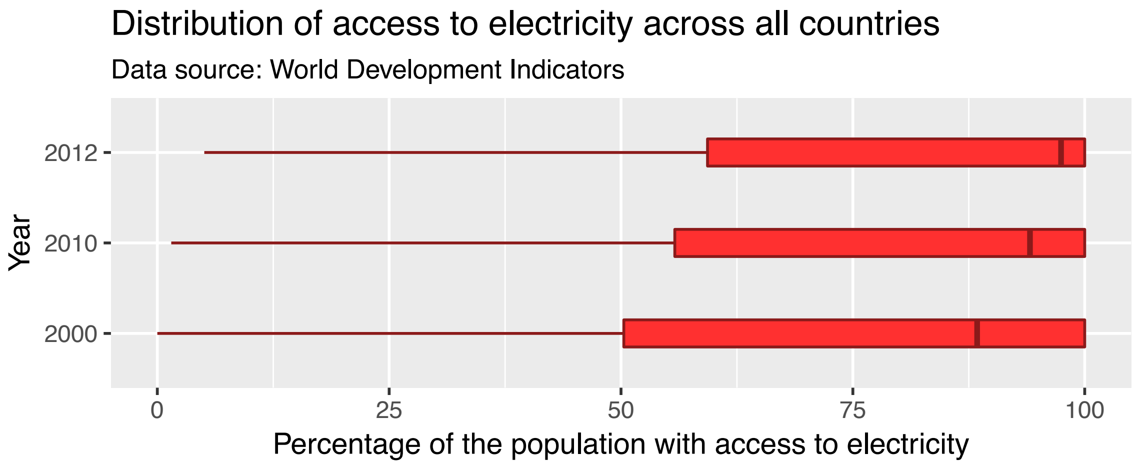
We can change the appearance of the boxplot using standard graphing parameters such as `color`, `fill`, and `width`.

```r
ggplot(subset(dat, year %in% c(2000, 2010, 2012)),
       aes(x = factor(year), y = electricity_pop)) +
  geom_boxplot(width = 0.3, color = "firebrick4", fill = "firebrick1") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Year",
       y = "Percentage of the population with access to electricity")
```



We can change the orientation of the plot with the `coord_flip()` command, which will flip the axes.

```r
ggplot(subset(dat, year %in% c(2000, 2010, 2012)),
       aes(x = factor(year), y = electricity_pop)) +
  geom_boxplot(width = 0.3, color = "firebrick4", fill = "firebrick1") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Year",
       y = "Percentage of the population with access to electricity") +
  coord_flip()
```



6 Saving plots

We can output your plots to many different format using the `ggsave()` function, including but not limited to `.pdf`, `.jpeg`, `.bmp`, `.tiff`, or `.eps`. Here, we output the graph as a Portable Network Graphics (.png) file. We can specify the size of the output graph as well as the resolution in dots per inch (dpi). If no graph is specified, `ggsave()` will save the last graph that was executed. For us, this is the boxplot in horizontal orientation. If we no not specify the complete file path, the plot will be saved to your working directory.

```r
ggsave("boxplot_horizontal.png", width = 6, height = 3, dpi = 400)
```

Workshop: Data visualization with `ggplot2` (part II)

Therese Anders

April 21, 2017

Introduction

In the second part of the workshop, we discuss the homework from the first part, introduce multivariate scatter and line plots, adding trend lines, and plotting simple maps.

As in the first part of this workshop, we use data from the World Development Indicators with a number of additional variables. Specifically, we look at different indicators for the energy consumption of all countries in the WDI dataset for the past 25 years. For the sake of time, I downloaded and cleaned the data prior to the workshop. The code to clean the data (`clean_wdi_.R`) is available on the GitHub repository.

- `year`: Variable coding the year of observation.
- `country`: Variable coding the country of observation.
- `electricity_pop`: Access to electricity (% of population).
- `gdppc`: GDP per capita, PPP (constant 2011 international $).
- `energyuse_pop`: Energy use (kg of oil equivalent per capita).
- `energyuse_gdp`: Energy use (kg of oil equivalent) per $1,000 GDP (constant 2011 PPP).
- `renewable_energyuse`: Renewable energy consumption (% of total final energy consumption).
- `time_electricity`: Time required to get permanent electricity connection in days.
- `rights_index`: Stength of legal rights (0 = weak, 12 = strong), that is "the degree to which collateral and bankruptcy laws protect the rights of borrowers and lenders" (World Development Indicators Metadata).

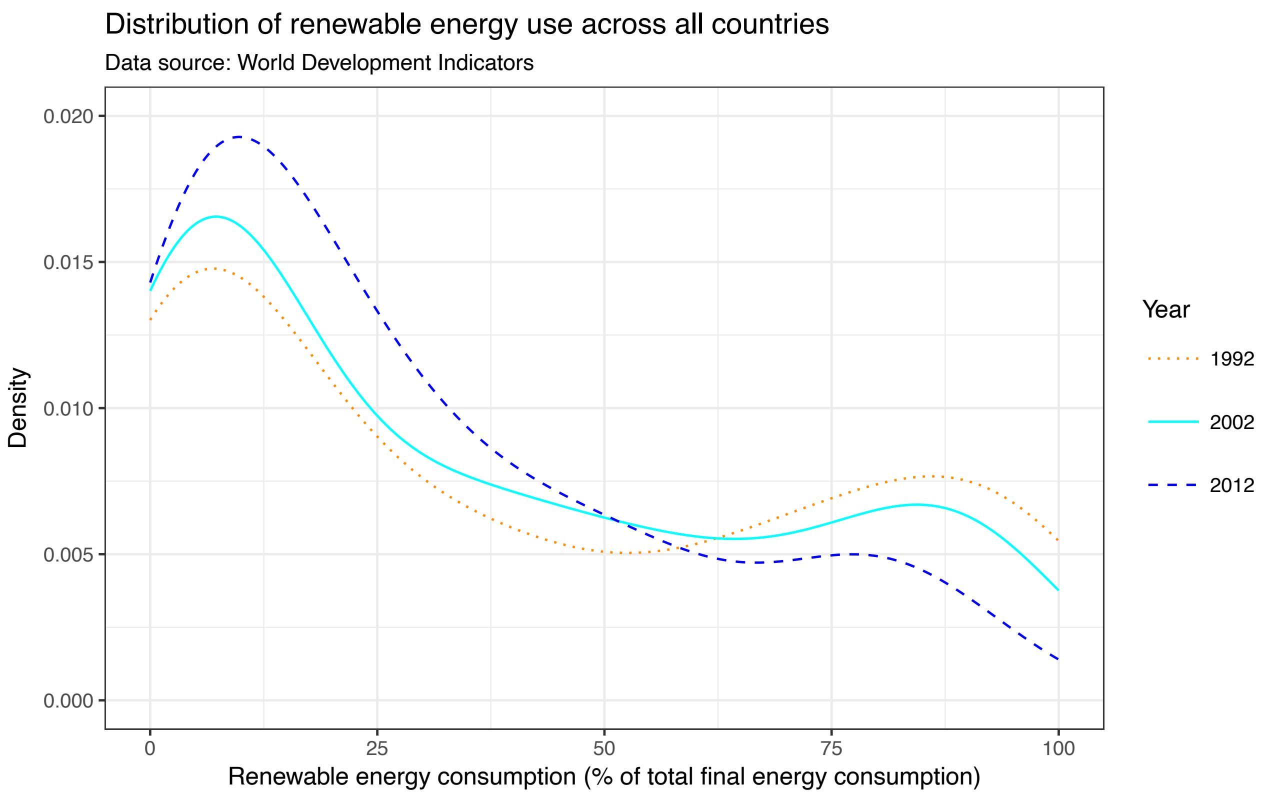
Homework from workshop 1

First, we set the working directory, load the `ggplot2` and `foreign` packages into the environment, and load the data. Note that the data set used in part I is a subset of the data set used in part II–we can therefore create the plots from the homework with the data for part II.

```
setwd("/Users/thereseanders/Documents/UNI/USC/Spring_2017/SPECRTraining/Part2")
library(ggplot2)
library(foreign)
dat <- read.csv("./data/wdi_cleaned_part2.csv",
                stringsAsFactors = F)
```

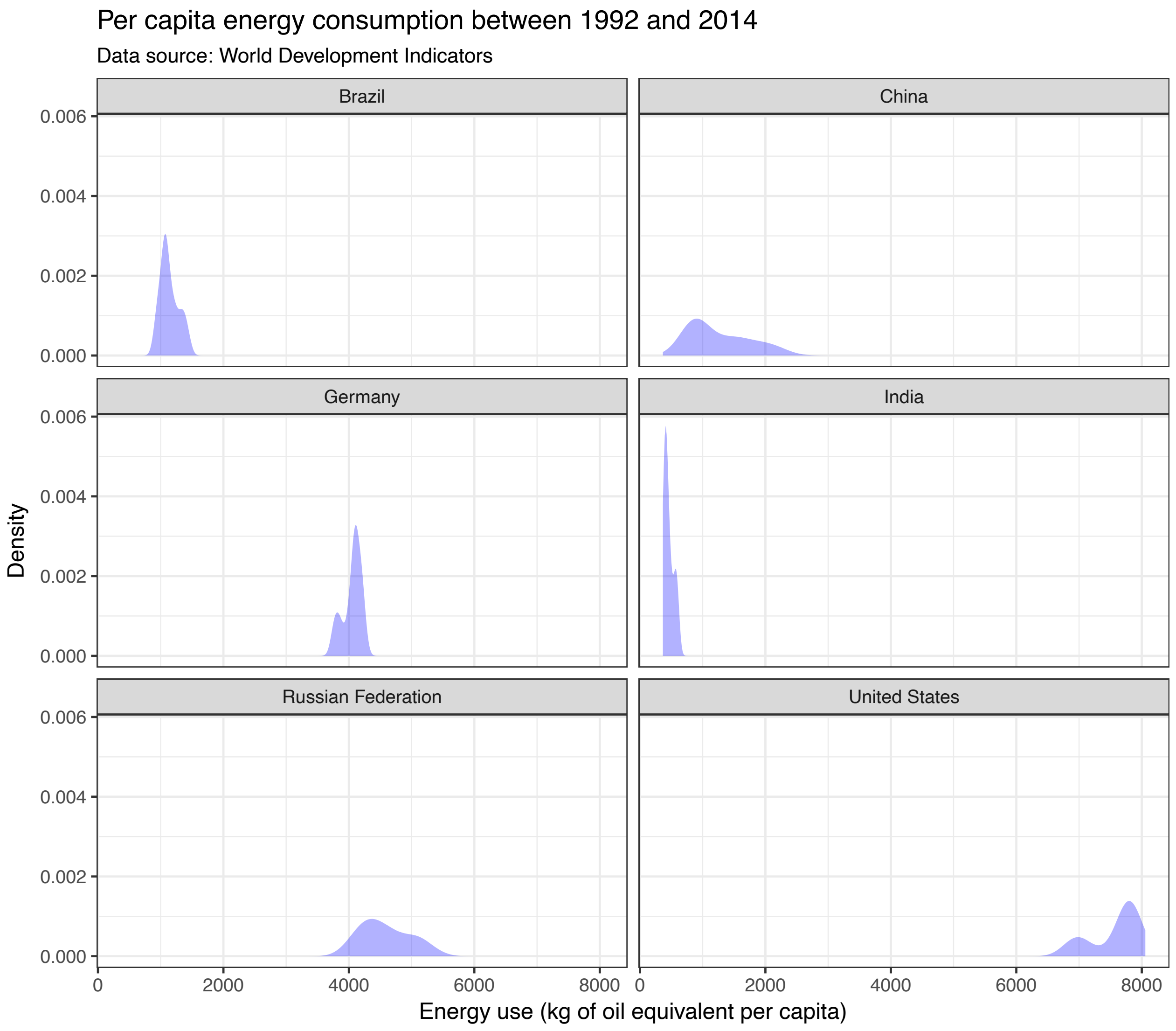
Graph 1

```
ggplot(subset(dat, year %in% c(1992, 2002, 2012)),
       aes(x = renewable_energyuse,
           color = factor(year),
           linetype = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of renewable energy use across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Renewable energy consumption (% of total final energy consumption)",
       y = "Density") +
  theme_bw() +
  scale_color_manual(name = "Year",
                     values = c("darkorange",
                                "cyan",
                                "blue")) +
  scale_linetype_manual(name = "Year",
                        values = c("dotted",
                                   "solid",
                                   "dashed")) +
  theme(legend.key.size = unit(1, "cm")) +
  coord_cartesian(ylim = c(0, 0.02))
```



Graph 2

```
ggplot(subset(dat, country %in% c("United States",
                                  "Germany",
                                  "Brazil",
                                  "Russian Federation",
                                  "India",
                                  "China")),
       aes(x = energyuse_pop)) +
  geom_density(fill = "blue", alpha = 0.3, color = NA) +
  facet_wrap(~ factor(country), ncol = 2) +
  theme_bw() +
  labs(title = "Per capita energy consumption between 1992 and 2014",
       subtitle = "Data source: World Development Indicators",
       x = "Energy use (kg of oil equivalent per capita)",
       y = "Density")
```



Multivariate graphs

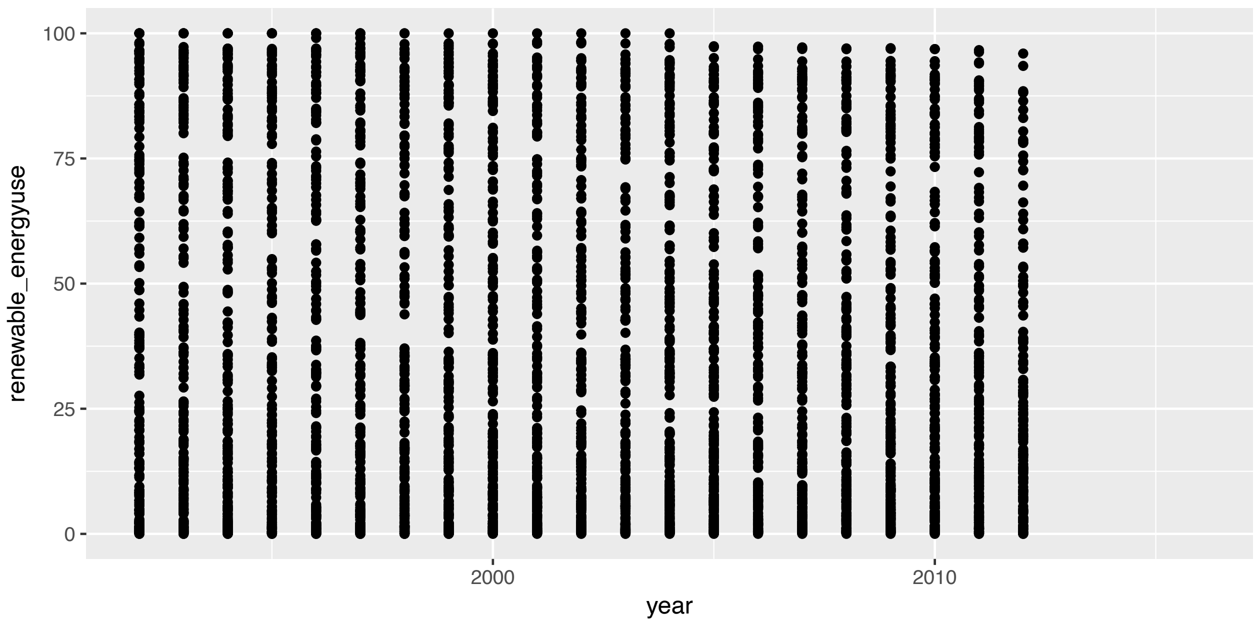
In the first session, we talked about univariate data summary graphs like density graphs or histograms in which we plot the range of values of a variable on the x-axis and the density or count of observations on the y-axis.

In this second session, we will plot two variables–one on the x- and one on the y-axis–against each other with the option of showing the value of additional variable(s) through color, shapes, or other aesthetics.

Scatterplots

Scatterplots show the relationship between two variables with the help of points (or other shapes). In `ggplot2` we use the `geom_point()` geometric object to create scatterplots. As an example, we plot the evolution of the worldwide renewable energy usage over time.

```r
ggplot(dat, aes(x = year, y = renewable_energyuse)) +
  geom_point()
```



```r
table(dat$year)
```

```
##
## 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
##  217  217  217  217  217  217  217  217  217  217  217  217  217  217  217
## 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016
##  217  217  217  217  217  217  217  217  217  217
```

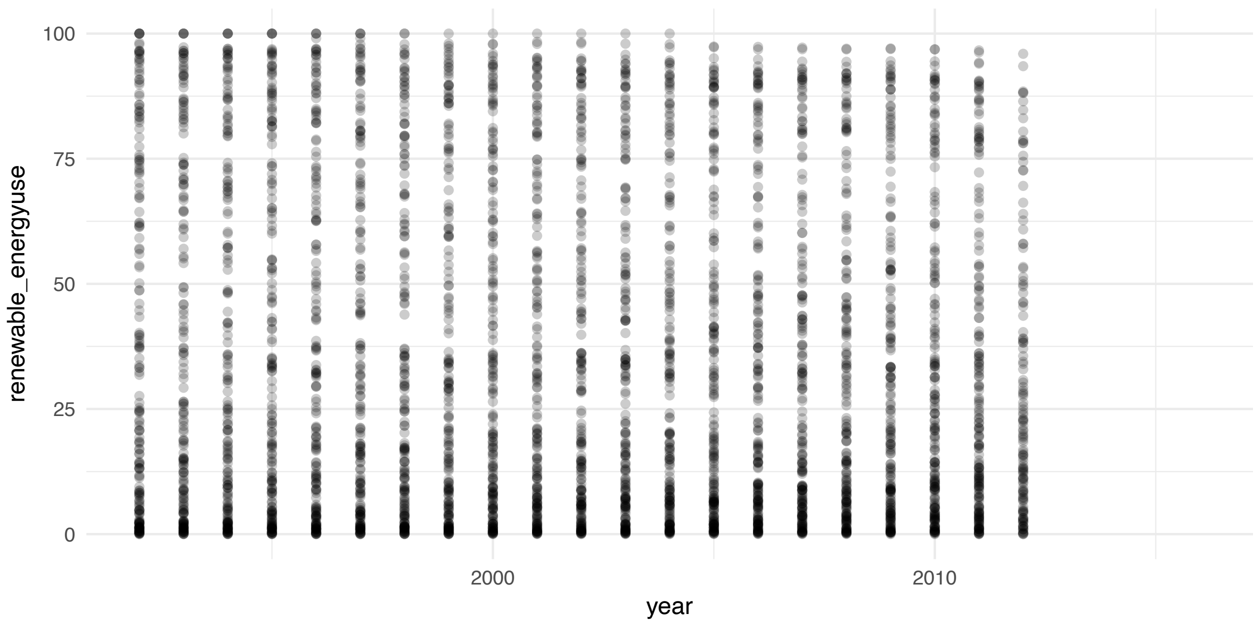
There are 217 observations per year. Due to overplotting, it is hard to draw conclusions from the plot. We have a number of ways in which we can adjust the appearance of the graph to highlight global trends in the data.

Adjusting the opacity of points

One way to use overplotting actively to highlight trends in the data is to reduce the opacity of points. Points will still be plotted on top of each other, but overlaying multiple transparent points will create clusters that signal an agglomeration of data points.

In addition, we use a theme with a white background to further increase the visibility of clusters in the data. In this example, while reducing the opacity of points does not significantly aid our understanding of over-time trends in the data, the plot shows that in the majority of countries between 0% to 10% of the energy used comes from renewables.

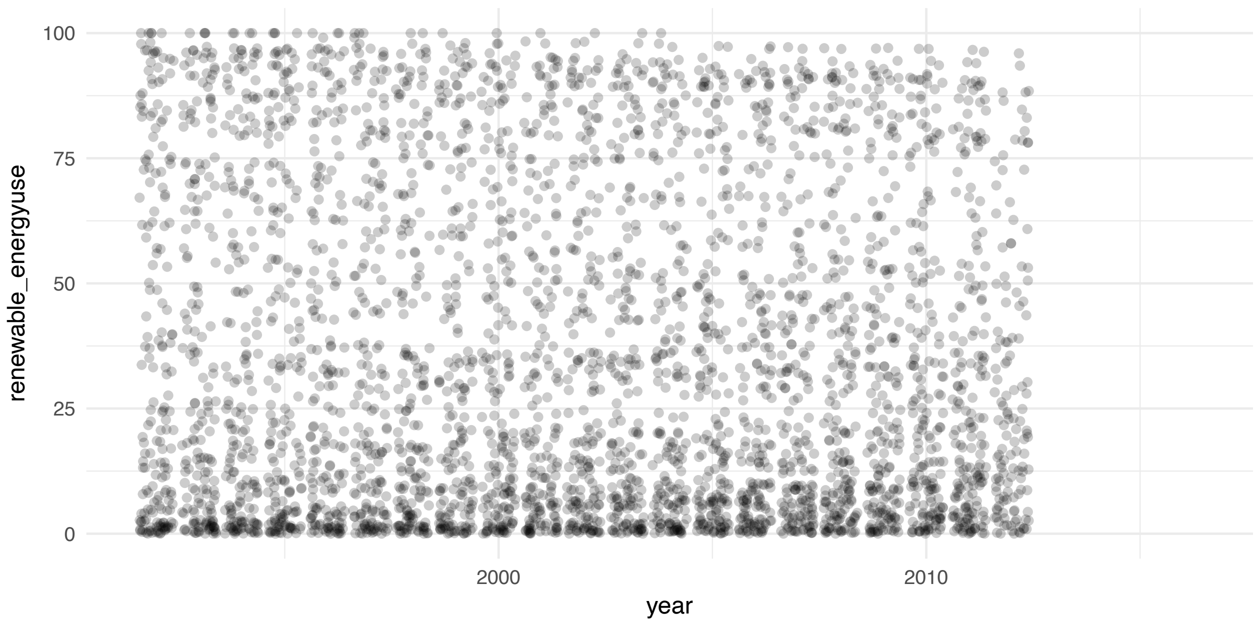
```r
ggplot(dat, aes(x = year, y = renewable_energyuse)) +
  geom_point(alpha = 0.2) +
  theme_minimal()
```



Jitter

Jittering points is another way to reduce the negative effects of overplotting. The `position_jitter()` argument randomly adds small values to each point. This way, each point is randomly shifted a tiny bit to the top, right, bottom, or left. Note that unless you set a seed to control the randomness, a graph with a jitter function will appear a little bit different every time you execute it.

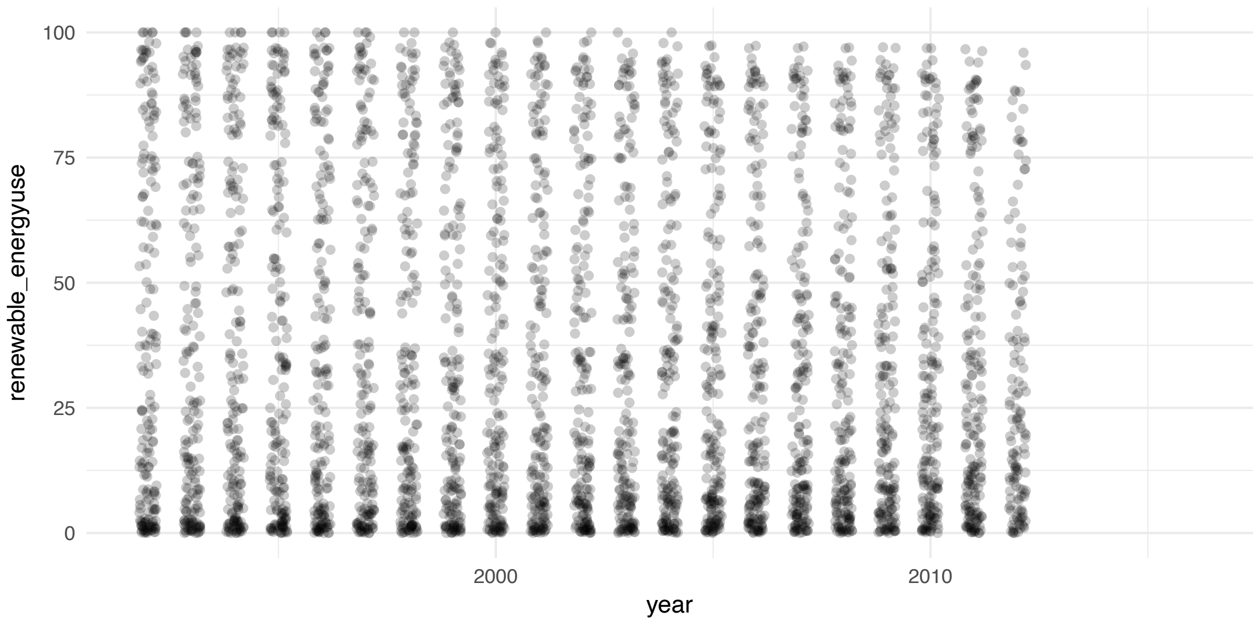
```r
ggplot(dat, aes(x = year, y = renewable_energyuse)) +
  geom_point(alpha = 0.2, position = position_jitter()) +
  theme_minimal()
```



We can change the default jitter value with the `width` and `height` parameters inside the `position_jitter()` argument. The default is for the data points to occupy 80% of the implied bins (see http://ggplot2.tidyverse.

org/reference/geom_jitter.html). In our case, by decreasing the jitter width, we can increase the visual separation between years.

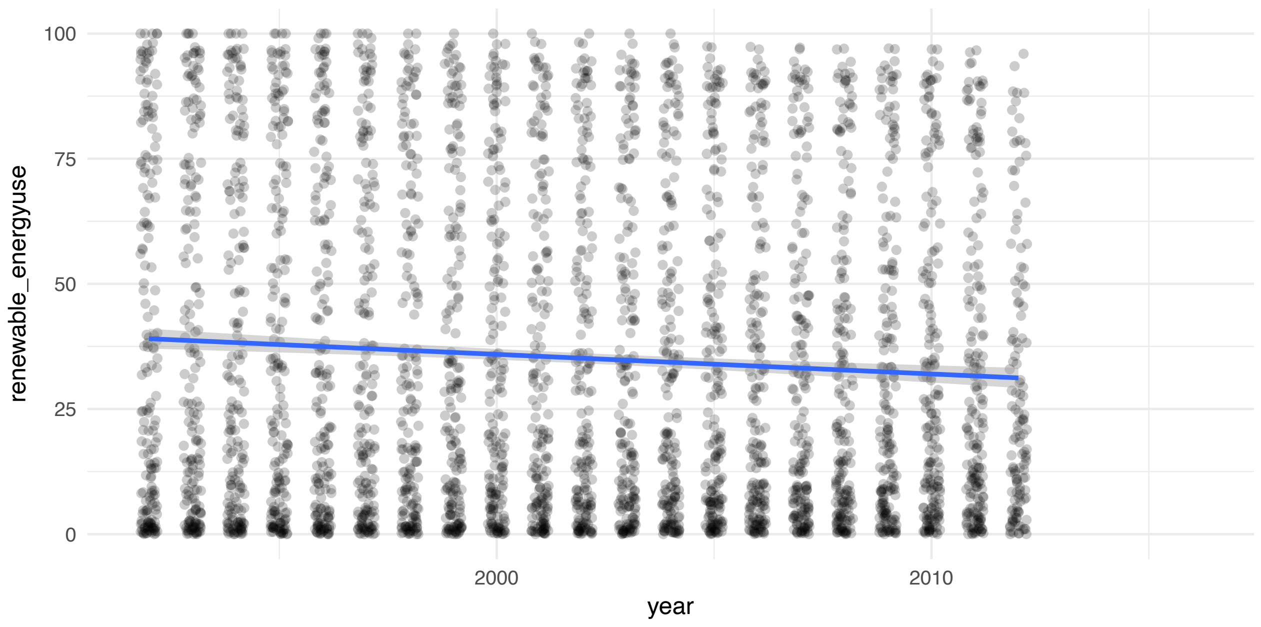
```r
ggplot(dat, aes(x = year, y = renewable_energyuse)) +
  geom_point(alpha = 0.2, position = position_jitter(width = 0.2)) +
  theme_minimal()
```



Adding trend lines

Another way to highlight trends in the data is to overlay a scatterplot with a trend line. In `ggplot2` this is implemented using the `stat_smooth()` function. For example, we could overlay the graph with the line of best fit of a linear model that regresses the proportion of renewable energy usage on the year using the `method = "lm"` argument. The resulting graph suggests that globally, the usage of renewables has decreased over time.

```r
ggplot(dat, aes(x = year, y = renewable_energyuse)) +
  geom_point(alpha = 0.2, position = position_jitter(width = 0.2)) +
  theme_minimal() +
  stat_smooth(method = "lm")
```



Using color to distinguish groups

Suppose we wanted to know whether the evolution in the usage of renewable energies differs between richer and poorer countries. We could do this by creating a binary variable that codes the wealth of countries and passing this variable to the color aesthetic. Here, we will use the median per capita GDP value as a cut point to split the sample into two roughly equally sized groups. Another alternative would be to use the average per capita GDP as a cut-off.

```
summary(dat$gdppc)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##    246.7   2791.0   8322.0  15330.0  20680.0 137200.0      958
```

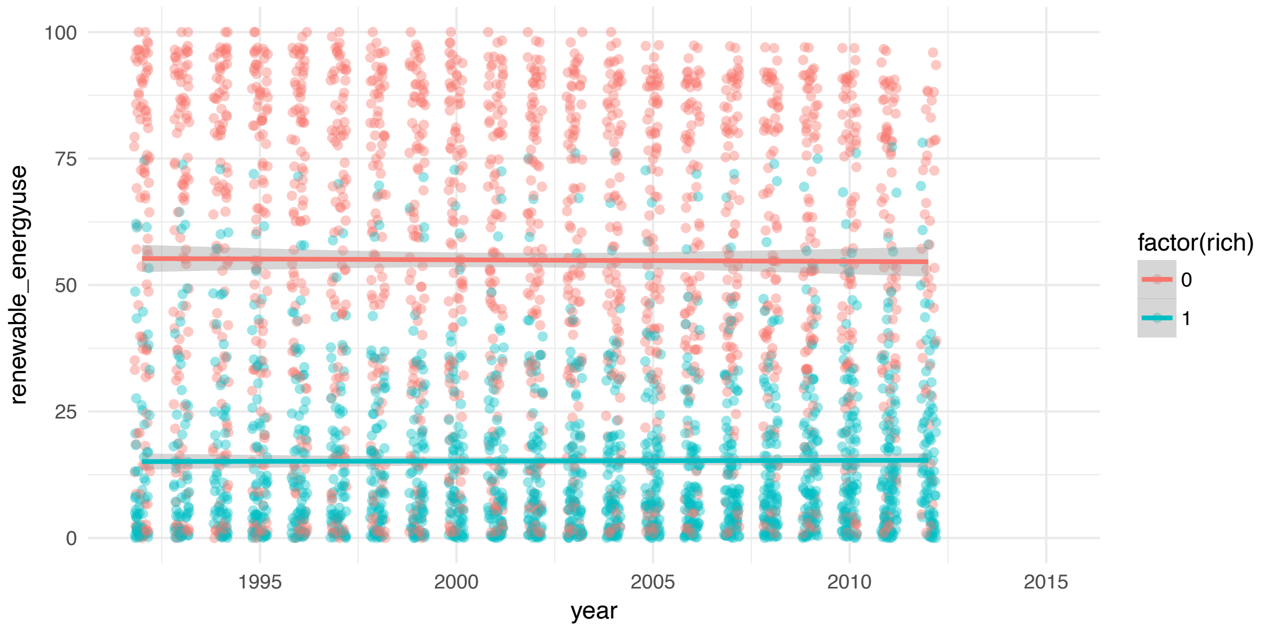
```
dat$rich <- ifelse(dat$gdppc >= median(dat$gdppc, na.rm = T), 1, 0)
table(dat$rich)
```

```
##
##    0    1
## 2233 2234
```

By passing the newly created `rich` variable to the color aesthetic, all following geometric objects will be plotted for both groups. This means that both the `geom_point()` objects and the `stat_smooth()` objects will be plotted for rich and poor countries. There are a number of missing values in our new variable `rich`. To avoid plotting these as a separate group, we need to subset the data to not include missing values on the variable `rich`.

The plot suggests that rich countries have lower levels of renewable energy usage. Interestingly, on average, the percentage of energy that comes from renewables has not changed much over time for neither the rich nor the poor countries.

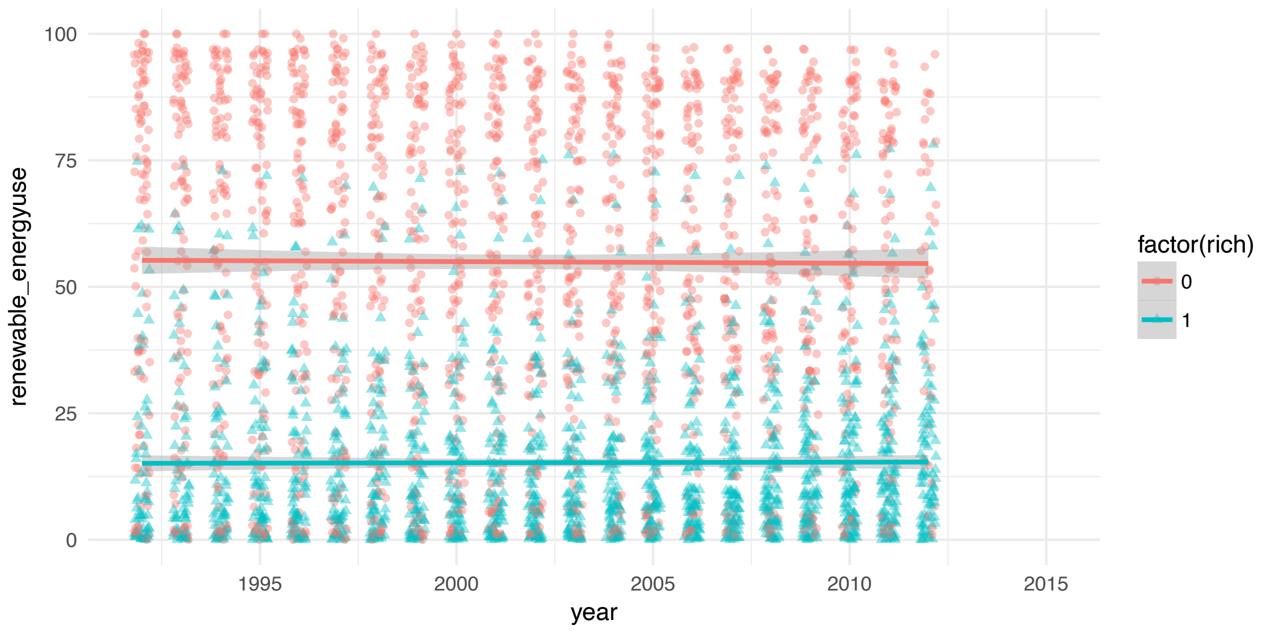
```
ggplot(subset(dat, !is.na(rich)),
       aes(x = year,
           y = renewable_energyuse,
           color = factor(rich))) +
  geom_point(alpha = 0.4, position = position_jitter(width = 0.2)) +
  theme_minimal() +
  stat_smooth(method = "lm")
```



Using shapes to distinguish groups

Sometimes, using color is not the optimal choice to distinguish groups, for example if we have to create plots on a grey scale or want to make sure our graphs are color blind save. We can instead use shapes to distinguish groups, or combine the distinction of shapes and color like in the example below.

```
ggplot(subset(dat, !is.na(rich)),
       aes(x = year,
           y = renewable_energyuse,
           color = factor(rich),
           shape = factor(rich))) +
  geom_point(alpha = 0.4, position = position_jitter(width = 0.2)) +
  theme_minimal() +
  stat_smooth(method = "lm")
```



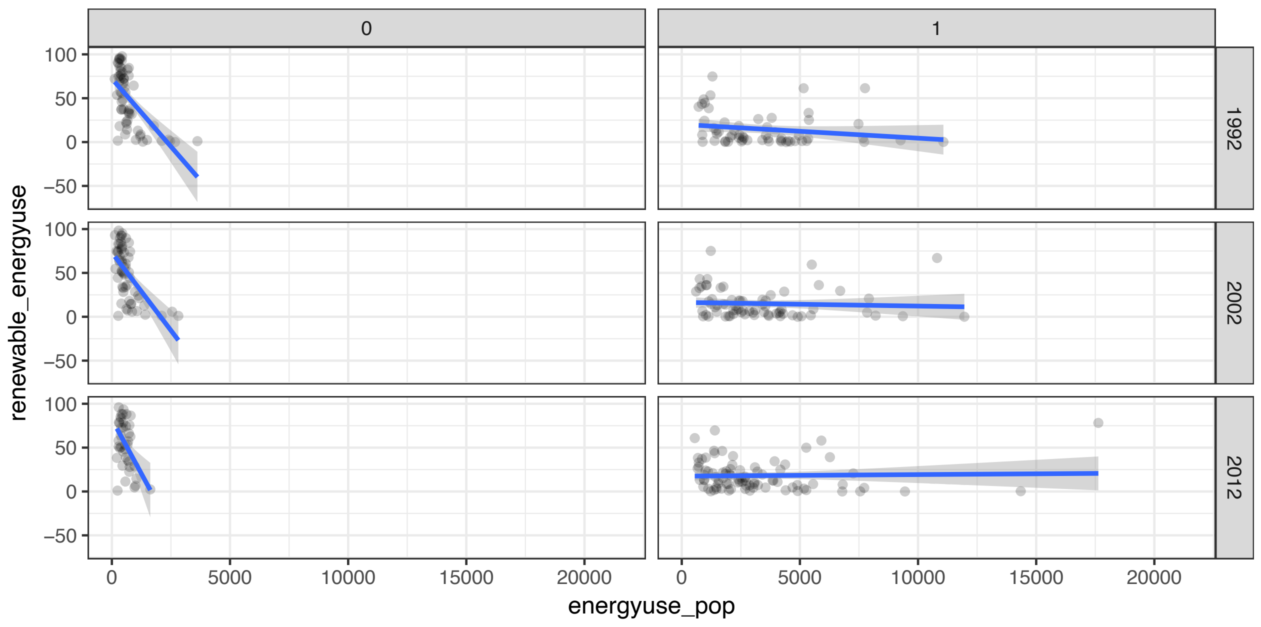
Using faceting to distinguish groups

We can also plot separate scatterplots for each group using faceting. In the first part of this workshop, we used the `facet_wrap()` function which allows us to plot relationships across groups contained in a single variable. Today, we will instead use the `facet_grid()` function which allows us to plot the relationship across two variables.

Suppose, we wanted to know whether there is a difference in the relationship between per capita energy consumption and percentage of renewables in poorer versus richer countries; and we also wanted to know how the relationship changes over time. We will only plot the relationship for years in which data is available on the per capita energy consumption: 2000, 2010, and 2012.

We can draw a number of conclusions from the plot below. First, the relationship between per capita energy consumption and the degree to which renewables are used to produce energy does not change much over time. What does influence the relationship between the two variables of interest is the wealth of a country. In richer countries, the amount of energy used and the prevalence of renewables are not strongly related. In poorer countries, the more energy is consumed on average, the less renewables contribute to the production of energy. Note, however, that the wealth of a country and the amount of energy used per inhabitant are positively correlated with a correlation coefficient of 0.83 (pooled across all available observations). This means that on average, the more energy a country consumes per capita, the more similar it will be to a richer country.

```
ggplot(subset(dat, !is.na(rich) & year %in% c(1992, 2002, 2012)),
       aes(x = energyuse_pop,
           y = renewable_energyuse)) +
  geom_point(alpha = 0.2) +
  theme_bw() +
  stat_smooth(method = "lm") +
  facet_grid(factor(year)~factor(rich))
```



```r
cor.test(dat$energyuse_pop, dat$gdppc)
```

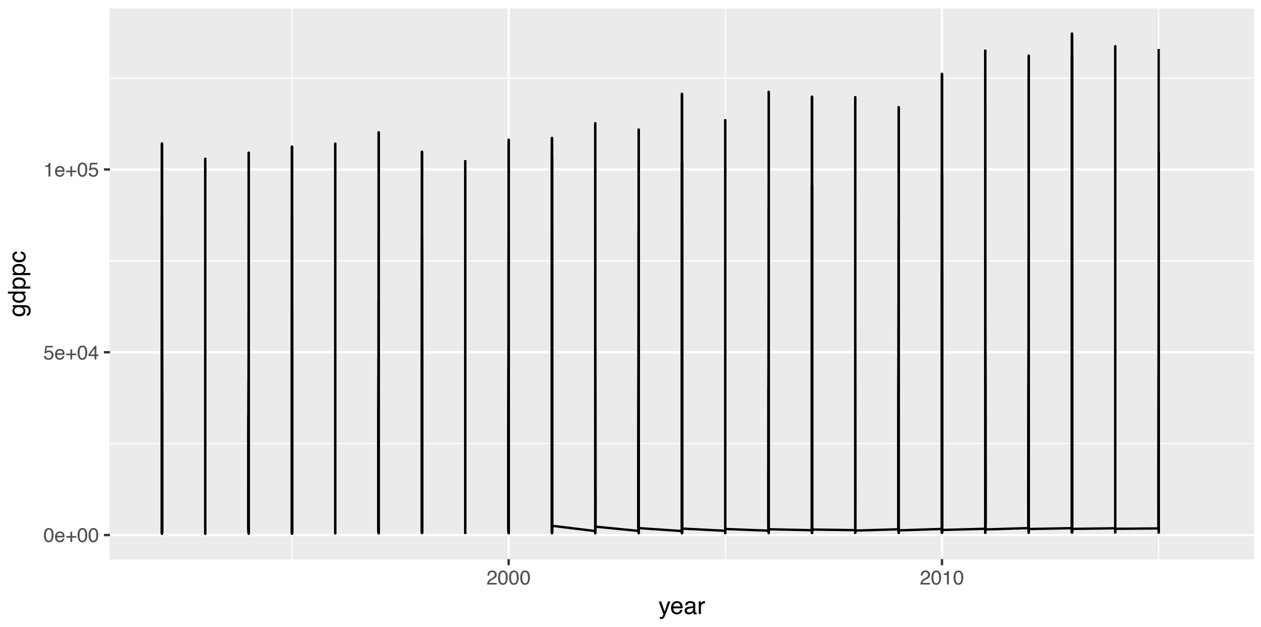
```
##
##  Pearson's product-moment correlation
##
## data:  dat$energyuse_pop and dat$gdppc
## t = 82.541, df = 3069, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8190053 0.8409938
## sample estimates:
##       cor
## 0.8303224
```

Line graphs

One of the most common uses for line graphs is the display of data over time. For example, we could be plotting the evolution of GDP per capita over time. Within the `aes()` command, we specify which variable to be plotted on the x- and y-axis. The geometric object we use for line graphs is `geom_line()`.

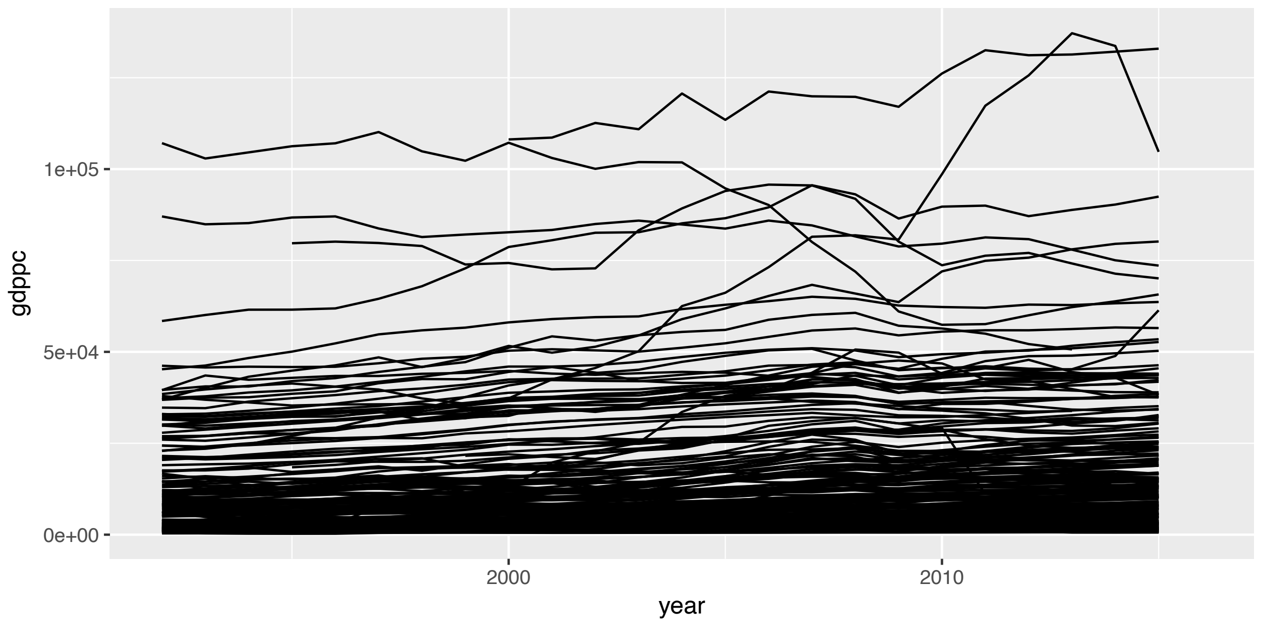
We are working with a panel data set. This means that we have multiple observations over time for each country. Displaying all this information without grouping or subsetting does not results in a plot that is useful. In the plot below, `geom_line()` is trying to draw a line between all non-missing country-year observations for the variable `gdppc`.

```r
ggplot(dat, aes(x = year, y = gdppc)) +
  geom_line()
```



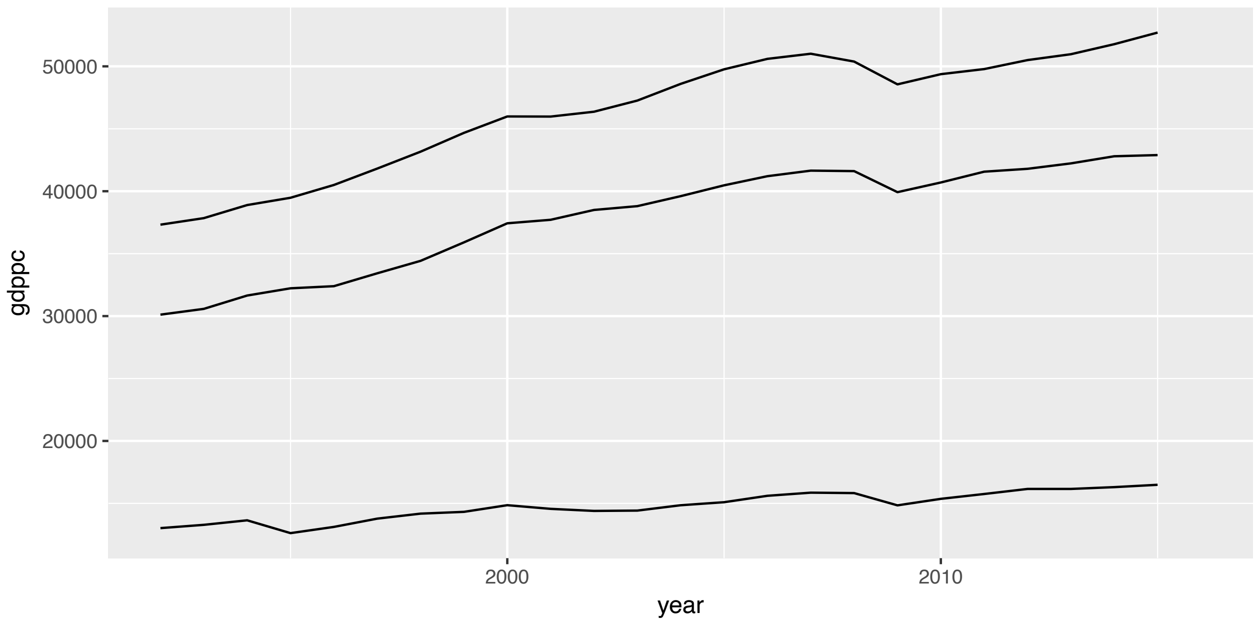
In order to plot one line per country, we can specify the `group` parameter inside the aesthetics argument. However, in this data set, even if we grouped the data by country, there are too many groups (countries) to be displayed in one graph.

```r
ggplot(dat, aes(x = year, y = gdppc, group = country)) +
  geom_line()
```



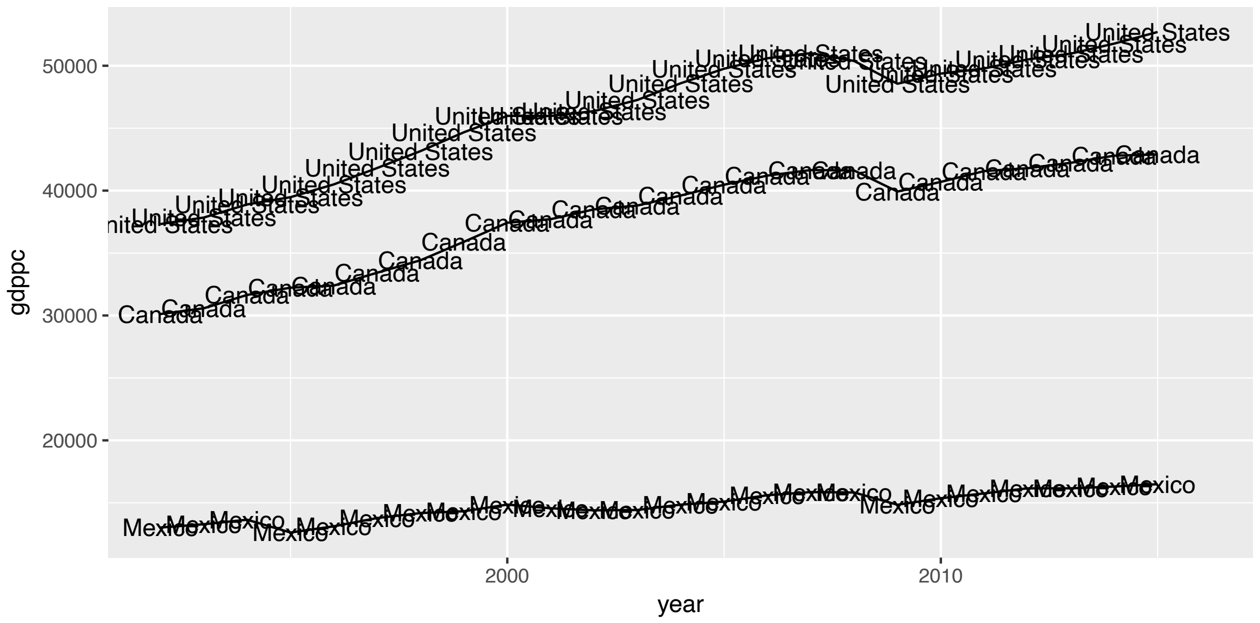
We will therefore use a subset of countries and plot the evolution of their per capita GDP over time. Suppose for example, we wanted to see how the per capita GDP has changed over time in member countries of the The North American Free Trade Agreement (NAFTA), that is Canada, Mexico, and the US. We can use subset inside the `ggplot()` function to plot a separate line for each of the countries.

```r
ggplot(subset(dat, country %in% c("Canada", "Mexico", "United States")),
       aes(x = year, y = gdppc, group = country)) +
  geom_line()
```



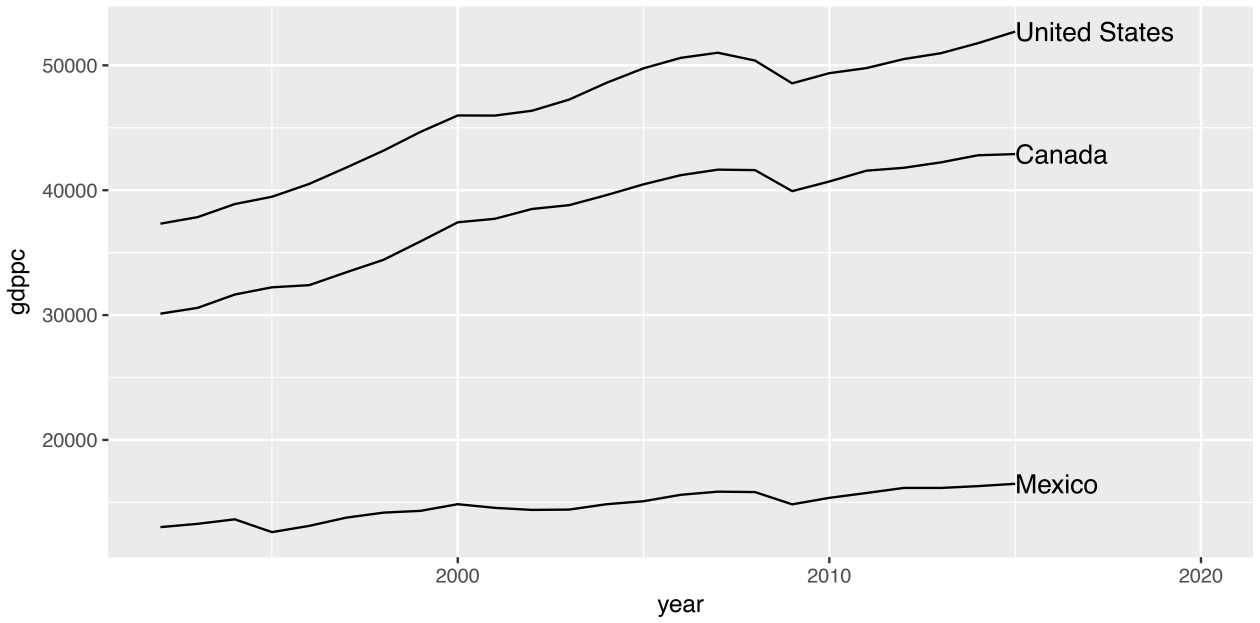
This graph does not tell us which line represents the evolution of per capita GDP in which country. We could add labels to each line to denote the country it represents. We could do this with the `geom_text()` function but this will add a label for each country-year observation.

```r
ggplot(subset(dat, country %in% c("Canada", "Mexico", "United States")),
       aes(x = year, y = gdppc, group = country, label = country)) +
  geom_line() +
  geom_text()
```



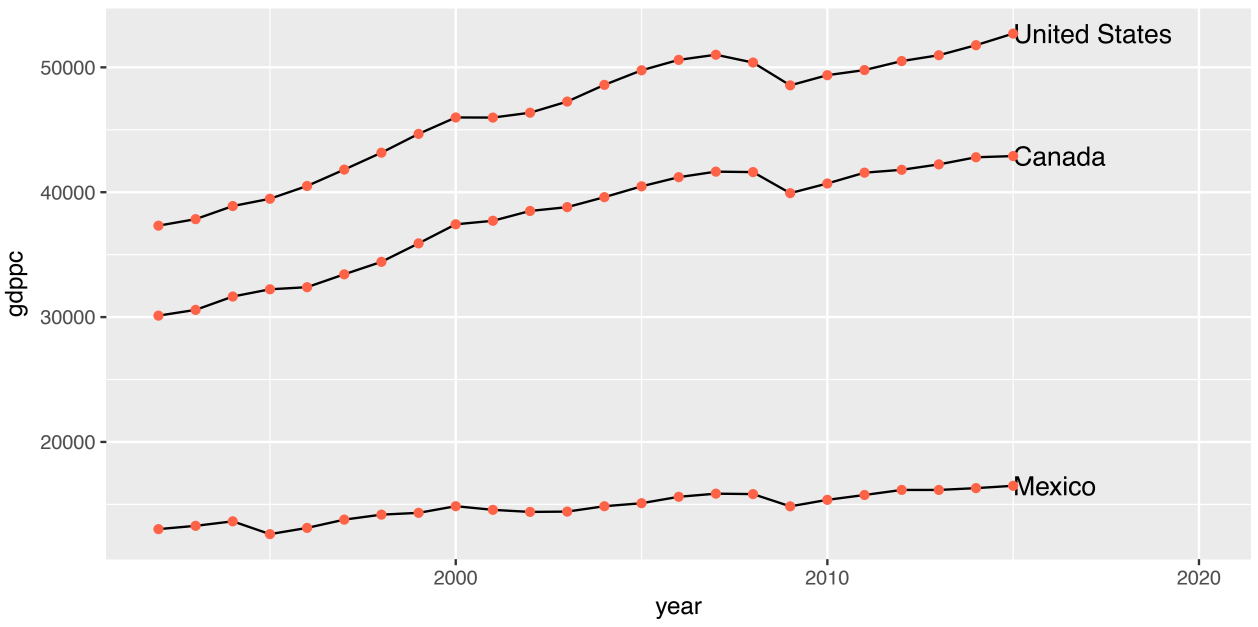
A better way to label each line (and not each country-year observation) is using the `directlabels` package and the `geom_dl()` geometric object in `ggplot()`. Note that you have to specify a method for the `geom_dl()` function to work (here: `"last.points"`). We can use the `cex` argument to control the size of the text labels. Here, I am also adjusting the range of the x-axis to ensure that all lines are properly labeled. The graph shows that the evolution of wealth in the three countries is intimately connected. Crises and upswings in show the same patterns for the US, Canada, and Mexico. However, the increase of per capita GDP appears to be a lot steeper in Canada and the US than in Mexico, despite similar trends.

```r
#install.packages("directlabels")
library(directlabels)
ggplot(subset(dat, country %in% c("Canada", "Mexico", "United States")),
       aes(x = year, y = gdppc, group = country)) +
  geom_line() +
  geom_dl(aes(label = country), method = ("last.points"), cex = 0.5) +
  coord_cartesian(xlim = c(1992, 2020))
```



It is not necessary in this example, but sometimes we would like to show which observations go into the computation of the line by adding points. Remember, that ggplot uses layers to graph multiple geometric objects in one plot. We can therefore just overlay the line plot with a `geom_point()` layer.

```r
ggplot(subset(dat, country %in% c("Canada", "Mexico", "United States")),
       aes(x = year, y = gdppc, group = country)) +
  geom_line() +
  geom_dl(aes(label = country), method = ("last.points"), cex = 0.5) +
  geom_point(color = "tomato") +
  coord_cartesian(xlim = c(1992, 2020))
```



Introduction to maps

We will use map data that is part of the maps package in R and does not require significant preprocessing before plotting. The `maps` package (https://cran.r-project.org/web/packages/maps/index.html) contains data on lines and polygons for a number of geographical units, including but not limited to, countries of the world, a database of large lakes, as well as United States federal states, counties, and cities. In this example, we retrieve data for a world map.

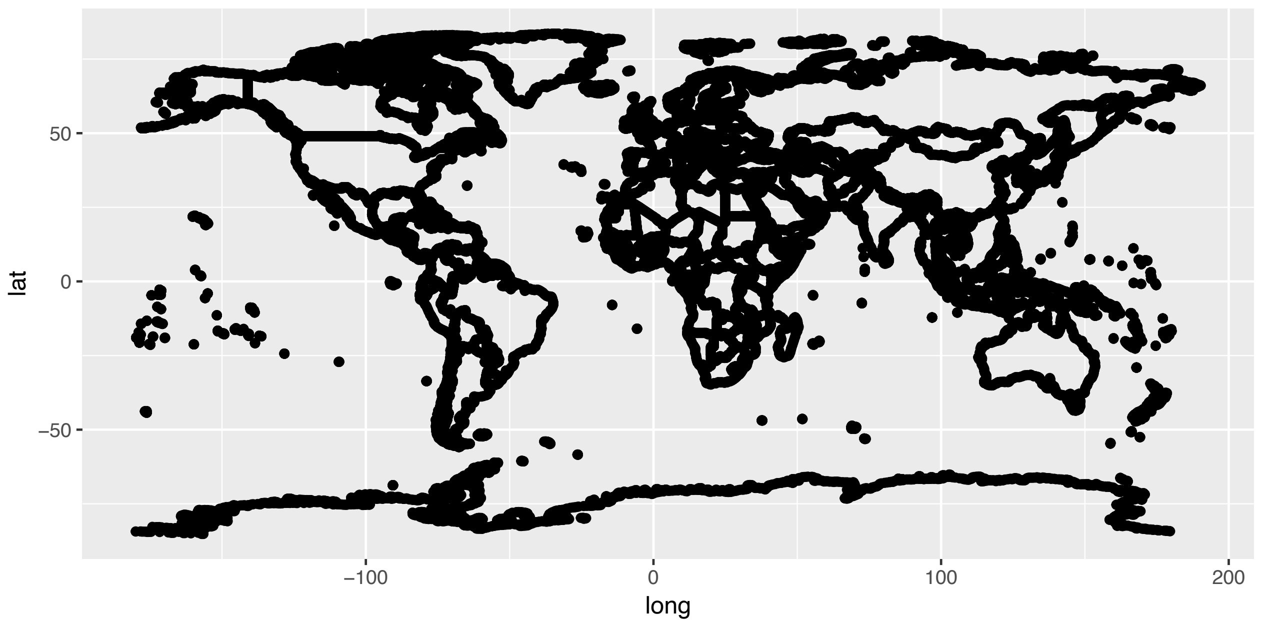
Before plotting the data, let us look at the structure of the data we drew from the `maps` package. The data is stored as a data frame and contains observations that are characterized by unique combinations of longitude and latitude values. In addition, each observation has the following attributes: group, order, region, and subregion if applicable.

```r
#install.packages("maps")
library(maps)
dat_map <- map_data("world")
head(dat_map)
```

```
##        long      lat group order region subregion
## 1 -69.89912 12.45200     1     1  Aruba      <NA>
## 2 -69.89571 12.42300     1     2  Aruba      <NA>
## 3 -69.94219 12.43853     1     3  Aruba      <NA>
## 4 -70.00415 12.50049     1     4  Aruba      <NA>
## 5 -70.06612 12.54697     1     5  Aruba      <NA>
## 6 -70.05088 12.59707     1     6  Aruba      <NA>
```

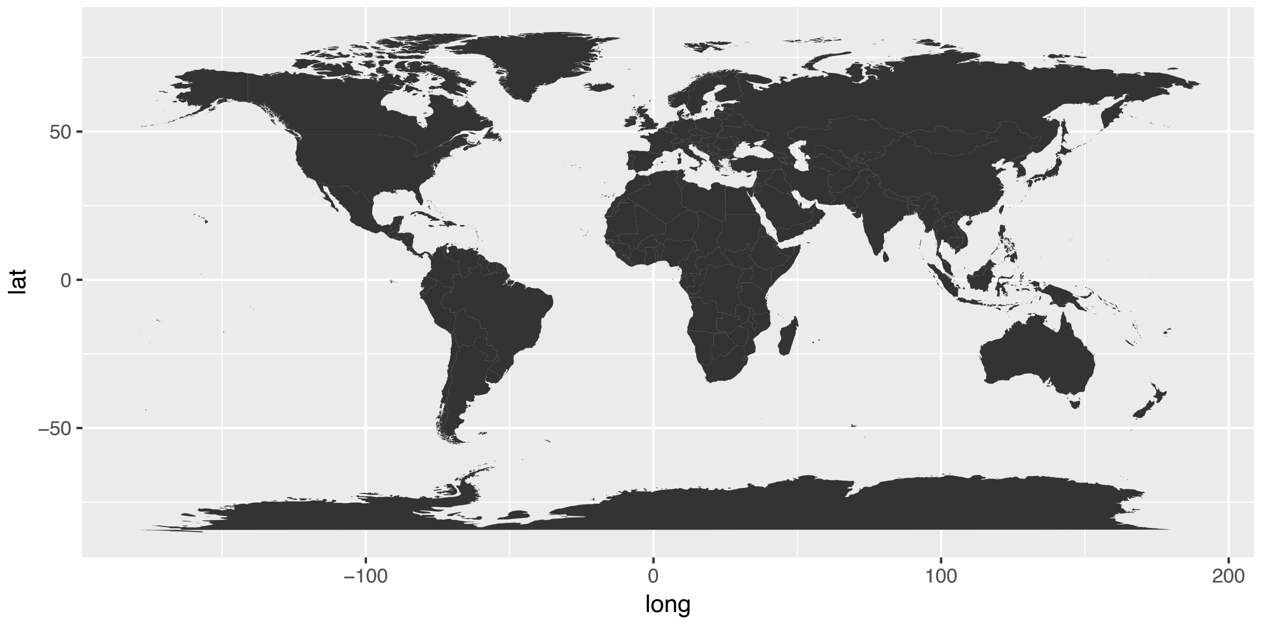
The longitude and latitude information denotes points along the borders of countries and geographical regions. We can represent them as points in the x-y-coordinate system, plotting the longitude along the x-axis and latitude along the y-axis.

```r
ggplot(dat_map, aes(x = long, y = lat)) +
  geom_point()
```



We can use ggplot2's geom_polygon() function to plot the observations as polygons, rather than points. In order to do that, we need to specify the grouping parameter. The geom_polygon() function connects the points for each group in the order of the data set (think of connecting dots on a sheet of paper without lifting your pen). This means that if the data is not already in the appropriate order, we need to sort it before plotting.

```r
ggplot(dat_map, aes(x = long, y = lat, group = group)) +
  geom_polygon()
```



Choropleth maps

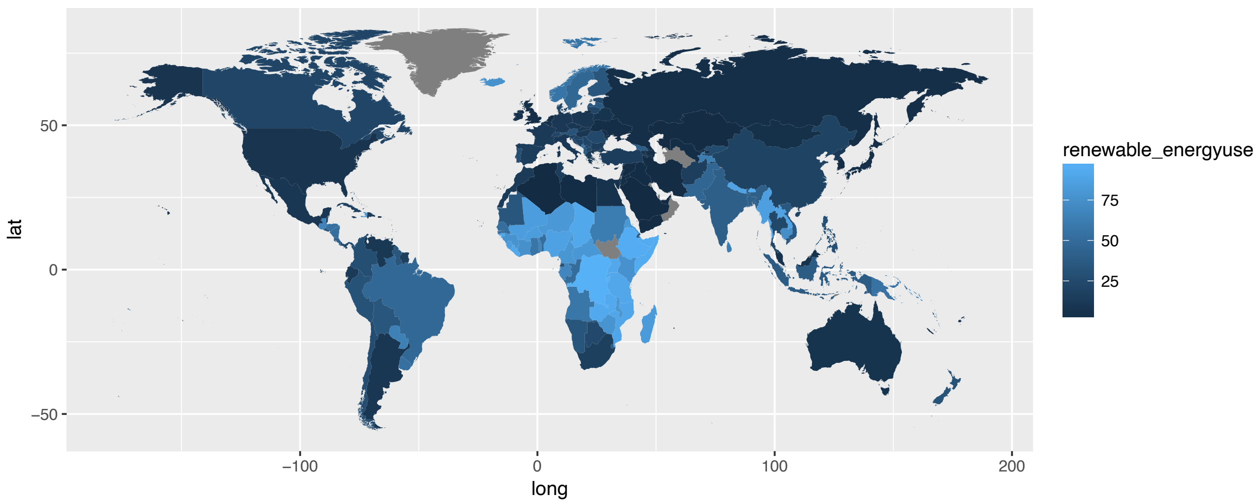
Choropleth maps use differences in shading of specific geographic regions to visualize data. Here, we will plot a map of the world where the shading denotes differences in the use of renewable energy across countries.

To plot the choropleth map, we will first need to **merge** our world map data with the data from the World Development Indicators. Unfortunately, country names differ between the two data sets. Therefore, we use the `countrycode()` function from the `countrycode` package to create vector of matching country codes for the two data sets before merging. We will use the `full_join()` function from the `dplyr` package for merging.

```
#install.packages("countrycode")
#install.packages("dplyr")
library(countrycode)
library(dplyr)
dat_map$ccode <- countrycode(dat_map$region,
                             origin = "country.name",
                             destination = "wb")

dat$ccode <- countrycode(dat$country,
                             origin = "country.name",
                             destination = "wb")

merged <- full_join(dat_map, dat, by = "ccode")
```

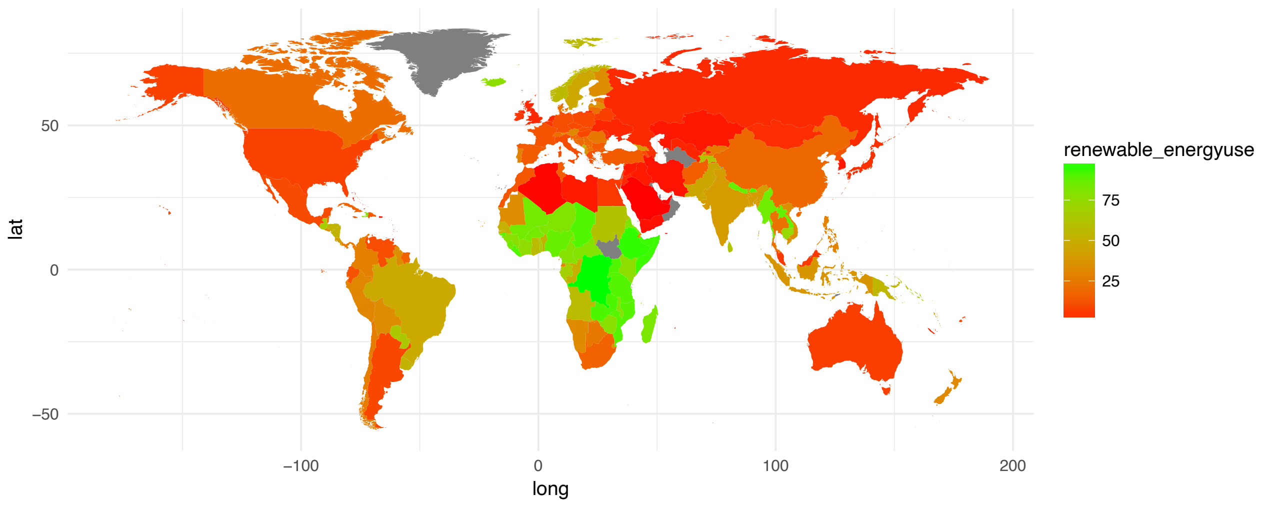
Now, we can plot our data. Suppose we wanted to illustrate the spatial variation of the usage of renewables in 2010. We plot the map as before, but pass the `renewable_energyuse` to the `fill` parameter inside the aesthetics argument. In addition, we specify that we only want to use data from 2010, using the `subset()` command when specifying the data frame. Note that we use our newly created `merged` data frame, rather than `dat_map`.

```
ggplot(subset(merged, year == 2010),
       aes(x = long, y = lat, group = group, fill = renewable_energyuse)) +
  geom_polygon()
```



The `ggplot2` default color scheme does not illustrate differences in the use of renewable energy particularly well. We can customize the color scheme to make differences between countries more apparent.

```
ggplot(subset(merged, year == 2010),
       aes(x = long, y = lat, group = group, fill = renewable_energyuse)) +
  geom_polygon() +
  scale_fill_gradient(low = "red", high = "green") +
  theme_minimal()
```


renewable_energyuse
75
50
25
lat
50
0
−50
long
−100
0
100
200

SPEC Lab R Workshop Series: Session 3

Therese Anders

1 Data wrangling with `dplyr`

1.1 Introduction

Data cleaning and reshaping is one of the tasks that we end up spending the most time on. Today, we will be introducing the `dplyr` library. Together with `stringr` (string operations using regular expressions) and `tidyr` these packages offer functionality for virtually any data cleaning and reshaping task in `R`.

For an overview of the most common functions inside `dplyr`, please refer to the RStudio data wrangling cheat sheet https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf.

1.2 Functions in `dplyr`

`dplyr` does not accept tables or vectors, just data frames (similar to `ggplot2`)! `dplyr` uses a strategy called "Split - Apply - Combine". Some of the key functions include:

- `select()`: Subset columns.
- `filter()`: Subset rows.
- `arrange()`: Reorders rows.
- `mutate()`: Add columns to existing data.
- `summarise()`: Summarizing data set.

First, lets dowload the package and call it using the `library()` function.

```
# install.packages("dplyr")
library(dplyr)
```

Today, we will be working with a data set from the `hflights` package. The data set contains all flights from the Houston IAH and HOU airports in 2011. Install the package `hflights`, load it into the library, extract the data frame into a new object called `raw` and inspect the data frame.

NOTE: The `::` operator specifies that we want to use the *object* `hflights` from the *package* `hflights`. In the case below, this explicit programming is not necessary. However, it is useful when functions or objects are contained in multiple packages to avoid confusion. A classic example is the `select()` function that is contained in a number of packages besides `dplyr`.

```
# install.packages("hflights")
library(hflights)
raw <- hflights::hflights
str(raw)
```

```
## 'data.frame':    227496 obs. of  21 variables:
##  $ Year             : int  2011 2011 2011 2011 2011 2011 2011 2011 2011 2011 ...
##  $ Month            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ DayofMonth       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ DayOfWeek        : int  6 7 1 2 3 4 5 6 7 1 ...
##  $ DepTime          : int  1400 1401 1352 1403 1405 1359 1359 1355 1443 1443 ...
##  $ ArrTime          : int  1500 1501 1502 1513 1507 1503 1509 1454 1554 1553 ...
##  $ UniqueCarrier    : chr  "AA" "AA" "AA" "AA" ...
##  $ FlightNum        : int  428 428 428 428 428 428 428 428 428 428 ...
##  $ TailNum          : chr  "N576AA" "N557AA" "N541AA" "N403AA" ...
##  $ ActualElapsedTime: int  60 60 70 70 62 64 70 59 71 70 ...
##  $ AirTime          : int  40 45 48 39 44 45 43 40 41 45 ...
##  $ ArrDelay         : int  -10 -9 -8 3 -3 -7 -1 -16 44 43 ...
##  $ DepDelay         : int  0 1 -8 3 5 -1 -1 -5 43 43 ...
##  $ Origin           : chr  "IAH" "IAH" "IAH" "IAH" ...
##  $ Dest             : chr  "DFW" "DFW" "DFW" "DFW" ...
##  $ Distance         : int  224 224 224 224 224 224 224 224 224 224 ...
##  $ TaxiIn           : int  7 6 5 9 9 6 12 7 8 6 ...
##  $ TaxiOut          : int  13 9 17 22 9 13 15 12 22 19 ...
##  $ Cancelled        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ CancellationCode : chr  "" "" "" "" ...
##  $ Diverted         : int  0 0 0 0 0 0 0 0 0 0 ...
```

1.3 Using `select()` and introducing the Piping Operator `%>%`

Using the so-called **piping operator** will make the `R` code faster and more legible, because we are not saving every output in a separate data frame, but passing it on to a new function. First, let's use only a subsample of variables in the data frame, specifically the year of the flight, the airline, as well as the origin airport, the destination, and the distance between the airports.

Notice a couple of things in the code below:

- We can assign the output to a new data set.
- We use the piping operator to connect commands and create a single flow of operations.
- We can use the select function to rename variables.
- Instead of typing each variable, we can select sequences of variables.
- Note that the `everything()` command inside `select()` will select all variables.

```r
data <-  raw %>%
  dplyr::select(Month,
              DayOfWeek,
              Airline = UniqueCarrier, #Renaming the variable
              Time = ActualElapsedTime, #Renaming the variable
              Origin:Cancelled) #Selecting a number of columns.
```

Suppose, we didn't really want to select the `Cancelled` variable. We can use `select()` to drop variables.

```r
data <- data %>%
  dplyr::select(-Cancelled)
```

1.4 Introducting `filter()`

There are a number of key operations when manipulating observations (rows).

- `x < y`
- `x <= y`
- `x == y`
- `x != y`
- `x >= y`
- `x > y`
- `x %in% c(a,b,c)` is `TRUE` if `x` is in the vector `c(a, b, c)`.

Suppose, we wanted to filter all the flights that have their destination in the greater Los Angeles area, specifically Los Angeles (LAX), Ontario (ONT), John Wayne (SNA), Bob Hope (BUR), and Long Beach

(LGB) airports.

```r
airports <- c("LAX", "ONT", "SNA", "BUR", "LGB")

la_flights <- data %>%
  filter(Dest %in% airports)
```

Caution: The following command does not return the flights to LAX or ONT!

```r
head(la_flights)
```

```
##   Month DayOfWeek Airline Time Origin Dest Distance TaxiIn TaxiOut
## 1     1         1      CO  227    IAH  LAX     1379      8      20
## 2     1         1      CO  229    IAH  LAX     1379     11      17
## 3     1         1      CO  236    IAH  LAX     1379     10      27
## 4     1         1      CO  211    IAH  ONT     1334      5      17
## 5     1         1      CO  243    IAH  SNA     1347      6      35
## 6     1         1      CO  226    IAH  LAX     1379     13      15
```

```r
la_flights_alt <- data %>%
  filter(Dest == c("LAX", "ONT"))
head(la_flights_alt)
```

```
##   Month DayOfWeek Airline Time Origin Dest Distance TaxiIn TaxiOut
## 1     1         1      CO  227    IAH  LAX     1379      8      20
## 2     1         1      CO  236    IAH  LAX     1379     10      27
## 3     1         1      CO  220    IAH  LAX     1379      7      12
## 4     1         1      CO  236    IAH  LAX     1379      8      33
## 5     1         1      CO  253    IAH  LAX     1379     11      30
## 6     1         1      CO  229    IAH  LAX     1379     15      14
```

Why? We are basically returning all values for which the following is `TRUE` (using the correct output of the `la_flights` data frame:

`Dest[1] == LAX`

`Dest[2] == ONT`

`Dest[3] == LAX`

`Dest[4] == ONT ...`

1.5 Helper functions

`dplyr` has a number of helper functions–and that is where the magic lies. These can be used with either `select()` or `filter()`. Here are some useful functions:

- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()`: Every name that matches "X", which can be a regular expression (we will talk about regular expressions in session 5).
- `one_of()`: Every name that appears in x, which should be a character vector.

For example, let's create a data frame with all variables that contain the word "Time".

```r
testframe <- raw %>%
  select(contains("Time"))
head(testframe)
```

```
##      DepTime ArrTime ActualElapsedTime AirTime
## 5424    1400    1500                60      40
## 5425    1401    1501                60      45
## 5426    1352    1502                70      48
## 5427    1403    1513                70      39
## 5428    1405    1507                62      44
## 5429    1359    1503                64      45
```

1.6 Introducting `mutate()`

Currently, we have two taxi time variables in our data set: `TaxiIn` and `TaxiOut`. I care about total taxi time, and want to add the two together. Also, people hate sitting in planes while it is not in the air. To see how much time is spent taxiing versus flying, we create a variable which measures the proportion of taxi time of total time of flight.

```
la_flights <- la_flights %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time)
```

Suppose, I only wanted to fly on weekends. Therefore, I create another variable that codes whether the flight is on a weekend or not and filter the data by this variable. Here, we introduce the `ifelse()` function that is tremendously helpful in data wrangling exercises. The syntax of `ifelse()` is as follows:

`ifelse(condition, if TRUE this, if FALSE this)`.

```
la_flights <- la_flights %>%
  mutate(Weekend = ifelse(DayOfWeek %in% c(1,7), 1, 0)) %>%
  filter(Weekend == 1)
```

1.7 Introducting `summarise()` and `arrange()`

One of the most powerful `dplyr` features is the `summarise()` function, especially in combination with `group_by()`.

First, in a simple example, lets compute the average flight time from Houston to Los Angeles by each day of the week. Also, I want to know what the maximum total taxi time is for each day of the weak. Note, that because there are missing values, we need to tell `R` what to do with them.

```
weekday_time <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(AverageTime = mean(Time, na.rm = T),
            MaxTaxiTotal = max(TaxiTotal, na.rm = T))
```

For legibility, lets reorder the output in ascending order using `arrange()`, with `MaxTaxiTotal` as a tie breaker.

```
weekday_time <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(AverageTime = mean(Time, na.rm = T),
            MaxTaxiTotal = max(TaxiTotal, na.rm = T)) %>%
  arrange(AverageTime, MaxTaxiTotal) #Default is ascending order
```

Now, suppose, I was flying from Houston to Los Angeles, and wanted to know which airline operates the most flights for this route before booking.

Here, we will be using the operator `n()` to tell dplyr to count all the observations for the groups specified in `group_by()`. After computing the result, I would like to arrange the output from highest number of flights, to lowest number.

```
carriers <- la_flights %>%
  group_by(Airline) %>%
  summarise(NoFlights = n()) %>%
  arrange(desc(NoFlights)) #desc() for descending order.
head(carriers)
```

```
## # A tibble: 3 x 2
##   Airline NoFlights
##   <chr>       <int>
## 1 CO           1943
## 2 WN            393
## 3 MQ            228
```

So if I want to have the highest selection of flights, I should book with Continental Airlines (at least back in 2011).

1.8 Putting it all together: The power of piping

In this example, I am starting all the way from the original `hflights` data set to demonstrate the power of the piping operator.

```
carriers_new <-  raw %>%
  select(Month,
         DayOfWeek,
         Airline = UniqueCarrier,
         Time = ActualElapsedTime,
         Origin:TaxiOut) %>%
  filter(Dest %in% c("LAX", "ONT", "SNA", "BUR", "LGB")) %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time,
         Weekend = ifelse(DayOfWeek %in% c(1,7), 1, 0)) %>%
  filter(Weekend == 1) %>%
  group_by(Airline) %>%
  summarise(NoFlights = n()) %>%
  arrange(desc(NoFlights))
 head(carriers_new)
```

```
## # A tibble: 3 x 2
##   Airline NoFlights
##   <chr>       <int>
## 1 CO           1943
## 2 WN            393
## 3 MQ            228
```

1.9 Saving files

Finally, lets save the output of our code. This time, we would like to save it in the .dta format. Remember to load the `foreign()` library before saving.

```
library(foreign)
write.dta(carriers_new, "carriers_houston_la.dta")
```

SPEC R Workshop Series: Session 4

Therese Anders

1 Merging data

1.1 The data

In today's workshop we will be using data on military expenditures from two publicly available sources: The World Bank Development Indicators (WDI), and the Correlates of War Project's (COW) National Material Capabilities Data Version 4.0 starting in 1914. The data has been cleaned after downloading it from the original sources.

The WDI data contains the following variables:

- `ccode`: COW country code (added using the `countrycode` package).
- `country`: Country name.
- `year`: Year.
- `milex_gdp`: Military expenditure as % of GDP.
- `gdp`: GDP in current USD.

The COW data contains the following variables:

- `ccode`: COW country code.
- `year`: Year.
- `milex`: Military expenditure in current USD.

First, let's load the data and look at the temporal coverage.

```
wdi <- read.csv("wdi_cleaned.csv")
summary(wdi$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1967    1979    1992    1992    2004    2016
```

```
cow <- read.csv("cow_cleaned.csv")
summary(cow$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1914    1956    1977    1972    1993    2007
```

Exercise 1 Suppose, we are interested in absolute military expenditures. Add a new variable to the WDI data set called `milex` that codes military expenditures in USD using functions from the `dplyr` package and piping.

```
##   ccode     country year milex_gdp        gdp milex
## 1   700 Afghanistan 1967        NA 1673333419    NA
## 2   700 Afghanistan 1968        NA 1373333367    NA
## 3   700 Afghanistan 1969        NA 1408888923    NA
## 4   700 Afghanistan 1970        NA 1748886596    NA
## 5   700 Afghanistan 1971        NA 1831108972    NA
## 6   700 Afghanistan 1972        NA 1595555476    NA
```

1.2 A primer on data merging using base R functions

In principle, we can merge data using the cbind() ("column bind") and rbind() ("row bind") functions, that we are already familiar with.

```
vec1 <- c(1, 2, 3)
vec2 <- c("a", "a", "b")
cbind(vec1, vec2)
```

```
##      vec1 vec2
## [1,] "1"  "a"
## [2,] "2"  "a"
## [3,] "3"  "b"
```

```
rbind(vec1, vec2)
```

```
##      [,1] [,2] [,3]
## vec1 "1"  "2"  "3"
## vec2 "a"  "a"  "b"
```

However, this requires the data to align in its dimensions. In addition, for cbind() the ordering of the rows has to be the exact same to achieve correct merging; for rbind() the ordering of the columns has to be equal to achieve the right output.

```
dat1 <- cbind(vec1, vec2)
vec3 <- c(T, F, T, T)
cbind(dat1, vec3) # The fourth observation in vec3 is omitted
```

```
## Warning in cbind(dat1, vec3): number of rows of result is not a multiple of
## vector length (arg 2)
```

```
##      vec1 vec2 vec3
## [1,] "1"  "a"  "TRUE"
## [2,] "2"  "a"  "FALSE"
## [3,] "3"  "b"  "TRUE"
```

```
obs <- c("c", 4)
rbind(dat1, obs)
```

```
##     vec1 vec2
##     "1"  "a"
##     "2"  "a"
##     "3"  "b"
## obs "c"  "4"
```

As a side note: The bind_rows() function in the dplyr package is the smart cousin of rbind(). Given that two dataframes have an equal number of columns, it can merge them based on the variable name; the ordering of the columns does not matter.

1.3 Using dplyr for merging

Luckily, the dplyr package in R contains a number of functions that allow us to merge data in "smarter" ways. Please refer to the Data Wrangling Cheat Sheet for an overview of the functions dplyr has available.

Suppose, we have two data frames: x (here WDI) and y (here COW). The basic syntax for data merging with dplyr is the following:

```
output <- join(A, B, by = "variable")
```

We will focus on the following four join functions:

- `full_join()`: Join data from `x` and `y` upon retaining all rows and values. This is the maximum join possible. Neither `x` nor `y` is the "master."
- `inner_join()`: Join only those rows that appear in both `x` and `y`. Neither `x` nor `y` is the "master."
- `left_join()`: Join only those rows from `y` that appear in `x`, retaining all data in `x`. Here, `x` is the "master."
- `right_join()`: Join only those tows from `x` that appear in `y`, retaining all data in `y`. Here, `y` is the "master."

1.3.1 full_join()

Suppose we wanted to keep the maximum amount of data in our new merged data set, we would use the `full_join()` function to merge the WDI and COW data.

```r
full_wrong <- full_join(wdi, cow, by = "ccode")
```

Take a look at the data frame! There are a couple of things to notice:

1. There are a number of duplicate column names in our data. `dplyr` automatically added a suffix to those instances; `x` for the WDI data (the first data set we passed to the function) and `y` for the COW data (the second data set passed to the function). We will rename these variables later on to keep track from which data set they originate.
2. We have over 500,000 observations even though our original datasets `wdi` and `cow` only have approximately 10,000 observations each. Why do you think this is the case? Do you notice something weird about the `year` variable(s)?

The reason for the high number of observations is that we asked `dplyr` to join the data sets based on the `ccode` variable only. This matches each observation in dataset `x` to each observation in dataset `y` as long as they have the same `ccode`. However, the rows in our data are not uniquely identified by the ccode variable. Since `wdi` and `cow` are panel data sets, each observation is uniquely identified by the combination of country code and year. We need to pass this information to the `full_join()` function to correctly match up the observations in our data.

In reality, `full_join()` and the other join functions from `dplyr` are smarter than that and will often identify which columns to match on based on duplicate column names. However, in our particular case, we run into problems because we have more duplicate column names that unique identifiers for the data; see the message that `R` outputs when running the command without specifying by which variables to merge.

```r
full_alsowrong <- full_join(wdi, cow)
```

```
## Joining, by = c("ccode", "year", "milex")
```

Now, before we do the correct full join, lets also rename the variables in our original data set to keep track of the data set from which the indicator originates. In the next session, we will learn how to do this with string operations, but for now, lets use dplyr's `select()` function.

```r
names(wdi)
```

```
## [1] "ccode"     "country"   "year"      "milex_gdp" "gdp"       "milex"
```

```r
wdi <- wdi %>%
  select(ccode:year,
         milex_gdp_wdi = milex_gdp,
         gdp_wdi = gdp,
         milex_wdi = milex,
         everything())
```

```r
names(cow)
```

```
## [1] "ccode" "year"  "milex"
```

```r
cow <- cow %>%
  select(ccode:year,
         milex_cow = milex)
```

We are ready to join! Notice that we use a vector to specify multiple inputs that define our unique observations.

```r
merged_full <- full_join(wdi, cow, by = c("ccode", "year")) %>%
  arrange(ccode, year)
head(merged_full)
```

```
##   ccode country year milex_gdp_wdi gdp_wdi milex_wdi    milex_cow
## 1     2    <NA> 1914            NA      NA        NA 2.532050e+10
## 2     2    <NA> 1915            NA      NA        NA 2.576480e+10
## 3     2    <NA> 1916            NA      NA        NA 2.774210e+10
## 4     2    <NA> 1917            NA      NA        NA 6.585840e+10
## 5     2    <NA> 1918            NA      NA        NA 7.014226e+11
## 6     2    <NA> 1919            NA      NA        NA 1.121780e+12
```

```r
nrow(merged_full)
```

```
## [1] 13698
```

1.3.2 inner_join()

Suppose, instead of retaining as much data as possible, we only wanted to keep data for which we have observations in both data sets. Note that this merging is only done based on the variables that we specify to uniquely identify each observation. `NA` values in all other columns are retained. Please refer to the *Data Wrangling Cheat Sheet* for methods that will select based on matching values beyond the unique identifiers (see for example `intersect()` and `setdiff()`).

```r
merged_inner <- inner_join(wdi, cow, by = c("ccode", "year")) %>%
  arrange(ccode, year)
head(merged_inner)
```

```
##   ccode       country year milex_gdp_wdi      gdp_wdi milex_wdi
## 1     2 United States 1967            NA 8.617000e+11        NA
## 2     2 United States 1968            NA 9.425000e+11        NA
## 3     2 United States 1969            NA 1.019900e+12        NA
## 4     2 United States 1970            NA 1.075884e+12        NA
## 5     2 United States 1971            NA 1.167770e+12        NA
## 6     2 United States 1972            NA 1.282449e+12        NA
##      milex_cow
## 1 7.544800e+12
## 2 8.073200e+12
## 3 8.144600e+12
## 4 7.782701e+12
## 5 7.486200e+12
## 6 7.763900e+12
```

```r
nrow(merged_inner)
```

```
## [1] 6614
```

1.3.3 `left_join()` and `right_join()`

Suppose, we had an existing master data set and wanted to add data to this master without adding new rows, just new columns. This could for example be the case if we had a spcific time frame that we wanted to study, and only want to merge data which matches this time frame.[1]

Here, suppose we wanted to make the WDI our master data set and merge data from the COW data onto this master. The WDI has a much smaller temporal coverage than the COW. For the sake of saving memory space, we only want the temporal coverage of the WDI reflected in the merged data. We can show that the WDI and the merged data frame have the same number of observations (rows).

```r
merged_left <- left_join(wdi, cow, by = c("ccode", "year"))
nrow(wdi) == nrow(merged_left)
```

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

Exercise 2 How would you achieve the exact same result using the `right_join()` function?

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

2 Data re-shaping with `tidyr`

Another important task in data management is data re-shaping. Often, data does not come in the format that we need for data merging, data visualization, statistical analysis, or vectorized programming. In general, we want data in the following format:

1. Each variable forms a column.
2. Each observation forms a row.[2]
3. For panel data, the unit (e.g. country) and time (e.g. year) identifier form columns.

The `tidyr` package offers two main functions for data reshaping:

- `gather()`: Shaping data from wide to long.
- `spread()`: Shaping data from long to wide.

2.1 Wide versus long data

For **wide** data formats, each unit's responses are in a single row. For example:

Country	Area	Pop1990	Pop1991
A	300	56	58
B	150	40	45

For **long** data formats, each row denotes the observation of a unit at a given point in time. For example:

Country	Year	Area	Pop
A	1990	300	56

[1]Another example is the case when we have a master data set that reflects the system membership of all countries which have COW country codes. We do not want to add observations from the other data sets that do not have COW country codes (for example data from Greenland or Puerto Rico). This example is for illustration only. When cleaning the WDI data, I have already removed the observations that do not have a COW code (such as Greenland or Puerto Rico).

[2]Hadley Wickham (2014, "Tidy Data" in *Journal of Statistical Analysis*) adds another condition, namely that "Each type of observational unit forms a table." We will not go into this here, but I can highly recommend you read Wickham's piece if you want to dive deeper into tidying data.

Country	Year	Area	Pop
A	1991	300	58
B	1990	150	40
B	1991	150	45

2.2 gather()

We use the `gather()` function to reshape data from wide to long. In general, the syntax of the data is as follows:

`new_df <- gather(old_df, key, value, columns to gather)`

We will be working with the `merged_left` data set. Suppose, we wanted to have a single column for military expenditures, and another column identifying the data set that the data comes from.

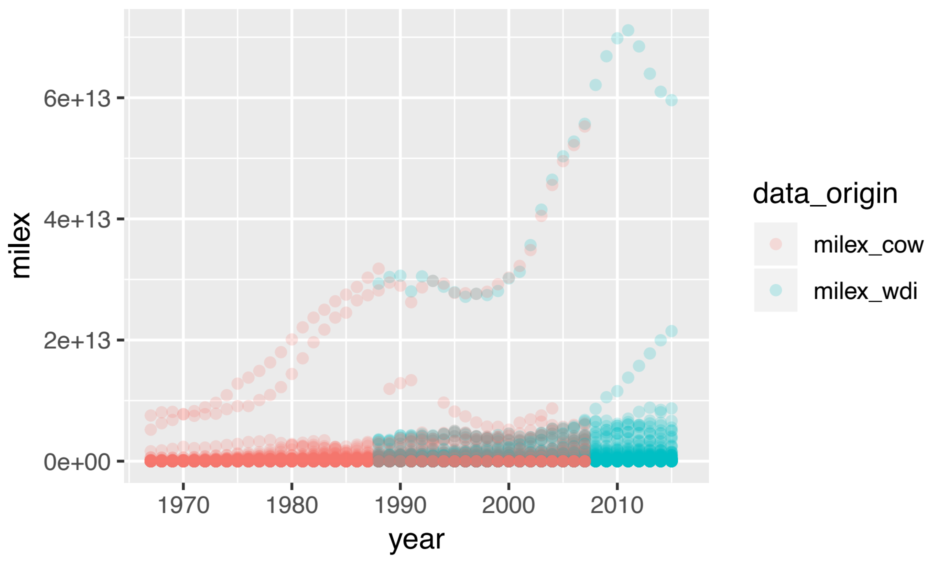
Gathering the military expenditure variables in this way will result in duplicate rows for each country-year. Whether or not this data format is desirable depends on the intended use for the data. Most statistical analysis methods require the data to have a single row per observation (for example country-year). However, some data visualization methods fare better when each concept (for example the variable `milex`) is captured in a single column with additional columns specifying supplementary attributes of the observation (see below).

Below, note that since `tidyr` and `dplyr` are sibling packages from the "tidyverse," we can use them seamlessly in the same pipe (here using `arrange()` from `dplyr` to illustrate the duplicate observations per country-year).

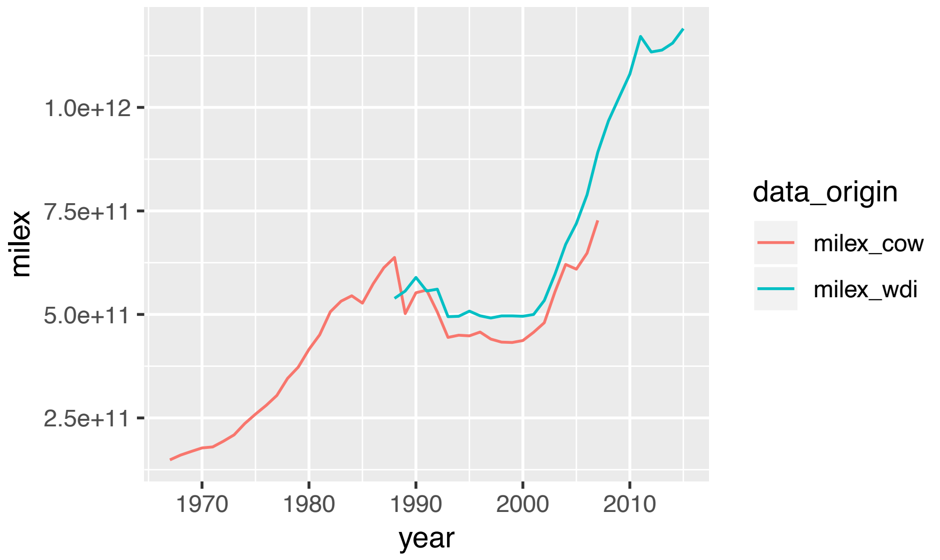
```r
library(tidyr)
merged_long <- merged_left %>%
  gather(data_origin, # name for categorical name indicator
         milex, # values
         milex_wdi:milex_cow) %>% # variables to be reshaped
  arrange(ccode, year)

# Creating averages by year
merged_long_averages <- merged_long %>%
  group_by(year, data_origin) %>%
  filter(!is.na(milex)) %>%
  summarize(milex = mean(milex))
```

We can use this format to plot the miliary expenditure for each of the data sets over time. Here, I use the `ggplot2` package (also from the "tidyverse") to show how to create over-time plots that use the properties of "tidy" data, namely the fact that a specific feature is represented with a single column (here `milex`), with additional columns specifying supplementary properties of this feature (here `data_origin`). Please note that these are both very basic graphs. For more customization of `ggplot2` graphs, see the introductory workshop on data visualization with `ggplot2`.

```r
library(ggplot2)
ggplot(merged_long,
       aes(x = year, y = milex, color = data_origin)) +
  geom_point(alpha = 0.2)
```



```r
ggplot(merged_long_averages,
       aes(x = year, y = milex, color = data_origin)) +
  geom_line()
```



2.3 spread()

Suppose we wanted to revert our operation (or generall shape data from a long to a wide format), we can use `tidyr`'s `spread()` function. The syntax is similar to `gather()`.

`new_df <- spread(old_df, key, value)`,

where `key` refers to the colum which contains the values that are to be converted to column names and `value` specifies the column that contains the value which is to be stored in the newly created columns.

```r
head(merged_long, 3)
```

```
##   ccode       country year milex_gdp_wdi   gdp_wdi data_origin      milex
## 1     2 United States 1967            NA 8.617e+11   milex_wdi         NA
## 2     2 United States 1967            NA 8.617e+11   milex_cow 7.5448e+12
## 3     2 United States 1968            NA 9.425e+11   milex_wdi         NA
```

```
merged_wide <- spread(merged_long, data_origin, milex)
```

Sources

Data Source

Correlates of War (COW) National Material Capabilities Data Version 4.0. http://www.correlatesofwar.org/data-sets/downloadable-files/national-material-capabilities-v4-0 (accessed Oct 2016).

The World Bank: World Development Indicators. https://datacatalog.worldbank.org/dataset/world-development-indicators (accessed Oct 2016). (accessed Oct 2016).

SPEC Lab R Workshop Series: Session 5

Therese Anders

1 Plan for today

1. String operations with the `stringr` library
2. Regular expressions
3. Homework exercise

2 String operations

String operations play a large role in the data cleaning component of data management, as well as in data gathering tasks such as web scraping. All string operations work on objects of the class **character**. Base `R` has a number of string operations, but they are often not user-friendly. The `stringr` package provides a comprehensive library of string operations. As part of the "tidyverse," we can use `stringr` functions together with other funtions from the `dplyr` and `tidyr` packages within the same pipe.

2.1 Basic string operations

The simplest string operation is base `R`'s `paste()` command. It simply concatenates two strings, by default with a space. We can also specify a custom expression to concatenate the characters with the `sep =` parameter.

```
paste("My", "string")
```

```
## [1] "My string"
```

```
paste("My", "string", sep = "_")
```

```
## [1] "My_string"
```

```
paste("My", "string", sep = "pretty")
```

```
## [1] "Myprettystring"
```

In data management, we most often use the `paste()` function when combining the values of two variables.

```
countries <- c("a", "b", "c")
years <- c(1990, 1990, 2000)
paste(countries, years, sep = "_")
```

```
## [1] "a_1990" "b_1990" "c_2000"
```

2.2 Pattern matching using `stringr`

Pattern matching is the work horse of string operations. Here, we focus on pattern matching functions that are most useful for the data management of data that is already organized in some kind of structure, such as vectors or data frames. There are many other string operation functions, such as `str_locate()` and `str_locate_all()` that are most useful for data cleaning and reorganization of unstructured data. Here, we will cover the following `stringr` functions:

- `str_sub()`: Extract substrings from a character vector by indices.

- `str_extract()`: Extract substrings from a character vector by matched pattern.
- `str_detect()`: Returns `TRUE` or `FALSE` if the pattern in matched in the input string.
- `str_replace()`: Replaces a matched pattern with a new pattern.

2.2.1 `str_sub()`

Since the `str_sub()` function operates with indices, is most useful when all elements of a vector (or some other data object of interest) have the same pattern. It extracts the information that matches the indices specified by the user. The general syntax is as follows.

`str_sub(pattern, start = , end = )`

```
library(stringr)
locations <- c("CA Los Angeles", "NV Las Vegas", "CA San Diego")
locations_state <- str_sub(locations, start = 1, end = 2)
locations_state
```

```
## [1] "CA" "NV" "CA"
```

We can also specify open intervals. For example, if we wanted to extract only the cities, we could extract all characters starting at the fourth position.

```
locations_city <- str_sub(locations, 4)
locations_city
```

```
## [1] "Los Angeles" "Las Vegas"   "San Diego"
```

2.2.2 `str_extract()`

Think of `str_extract()` as the smarter sibling of `str_sub()`. Rather than just extracting characters based on indices, it extracts characters based on matched patterns. Together with the regular expression operations we introduce below, this is very powerful in extracting information. Here is the syntax for `str_extract()`:

`str_extract(string, pattern)`

```
locations_state2 <- str_extract(locations, "CA")
locations_state2
```

```
## [1] "CA" NA   "CA"
```

2.2.3 `str_detect()`

Think of `str_detect()` as a logical operator. It returns `TRUE` if the pattern is matched and `FALSE` otherwise. The function has the following syntax:

`str_detect(string, pattern)`

```
str_detect(locations, "L")
```

```
## [1]  TRUE  TRUE FALSE
```

```
str_detect(locations, "Los")
```

```
## [1]  TRUE FALSE FALSE
```

2.2.4 `str_replace()`

`str_replace()` is very helpful in data management tasks, in particular when re-coding variables. As an example, suppose we wanted to replace the state abbreviations with the full state names. The syntax of the function is as follows:

`str_replace(string, pattern, replacement)`

```
locations_fullname <- str_replace(locations, "CA", "California") %>%
  str_replace("NV", "Nevada")
locations_fullname
```

```
## [1] "California Los Angeles" "Nevada Las Vegas"
## [3] "California San Diego"
```

2.3 Regular Expressions

String operations are useful, but their true potential is realized when used in combination with regular expressions. Regular expressions help us to define search patterns. This obliviates the need to type out entire character strings in pattern matching. Regular expressions are like a language that is understood by more than just one program. With small changes, you would be able to use the regular expressions discussed here in other programming languages, such as Python.

Suppose, our strings weren't as well organized as before. How would you extract the state from the elements of the following character vector?

```
locs <- c("CA Los Angeles", "Las NV Vegas", "San Diego CA")
```

Even though the elements of the character vector aren't as ordered any more, there is still a pattern! State abbreviations always have two letters and they are always capitalized. Below, we will learn a number of regular expression that will help us state this pattern ("two letters, capitalized") in a language that `R` understands.

Note that unless otherwise specified, regular expressions are case sensitive. The following patterns are examples for specifying the substantive content of the pattern.

- `^`: String **starts with** following pattern..
- `$`: String **ends with** preceding pattern.
- `[ ]`: or.
- `[a-z]`: Any letter.
- `[0-9]`: Any digit.

There is also a number of regular expressions that are used to specify the quantity of matches.

- `?`: Preceding pattern is optional (matched 0 or 1 times).
- `*`: Preceding pattern is matched 0 or more times.
- `+`: Preceding pattern is matched at least once.
- `{n}`: Preceding pattern is matched exactly`n` times.
- `{n, m}`: Preceding pattern is matched at least `n` times and up to `m` times.
- `{n,}`: Preceding pattern is matched at least `n` times.

So in our example above, we could use the following regular expression to extract the state names:

```
str_extract(locs, "[A-Z]{2}")
```

```
## [1] "CA" "NV" "CA"
```

2.3.1 Examples for regular expressions

```
fruits <- c("apple", "banana", "pineapple", "cherries4")
```

Exercise 1 What is the output of the following command?

```
str_detect(fruits, "a")
```

Exercise 2 What is the output of the following command?

```
str_detect(fruits, "^a")
```

Exercise 3 How would you ask `R` to print out `TRUE` for all elements of the `fruits` vector that end with the letter `e`?

```
## [1]  TRUE FALSE  TRUE FALSE
```

Exercise 4 What is the output of the following command?

```
str_detect(fruits, "[ie]")
```

Exercise 5 How would you extract the number from the last element of the `fruits` vector?

```
## [1] "4"
```

We can use regular expressions to create more complex queries. Suppose, we wanted to know which strings start with an `a` and end with an `e`.

```
str_detect(fruits, "^a[a-z]*e$")
```

```
## [1]  TRUE FALSE FALSE FALSE
```

Exercise 6 What do you think is the output of the following command?

```
str_detect(fruits, "a[a-z]*e$")
```

2.3.2 Escape character

Regular expressions can also be used to match special characters. Note, however, that since some special characters have a special meaning in `R`, we sometimes have to use a so-called **escape character**, i.e. a backslash, to specify these patterns within regular expressions. For example, since the open square bracket `[` is part of the regular expression for `OR`, in order to match a `[` in the text, you need to type `\\[` to get the desired output.

```
test <- "a []"
#str_detect(test, "[") # This throws and error.
str_detect(test, "\\[") # This correctly recognized the pattern.
```

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

2.3.3 Application: Phone Numbers

Suppose, we have data on contact information and wanted to extract only the values that match phone numbers. Unfortunately, phone numbers come in different formats, but there are a number of common patterns.

What are common patterns for phone numbers you guys can think of?

```r
phone <- c("123 456 7890",
           "(123) 456 7890",
           "123-456-7890")
```

We cannot use simple indices any longer to extract the phone numbers from this vector. The lengths of the elements differ!

```r
nchar(phone[1])
```

```
## [1] 12
```

```r
nchar(phone[2])
```

```
## [1] 14
```

Lets write a regular expression that outputs `TRUE` for all of these phone numbers.

```r
str_detect(phone, "[(]?[0-9]{3}[)]?[ -]{1}[0-9]{3}[ -]{1}[0-9]{3}")
```

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

Exercise 7 How would you alter this if I gave you the following number format and asked you to match it as well? `123.456.7890`

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

2.3.4 Application: Creating a data frame

Why do we need this? Suppose I gave you a long string of data and asked you to clean the data and give be a data set of phone numbers and ZIP codes. You could make your life **a lot** easier if you know regular expressions.

Load the file `data.RData` and implicitly print out at its content, a vector called `vec`.

```r
load("data.RData")
vec
```

```
## [1] "4: 12345"            "6: 123 456 7890"    "77: 90089"
## [4] "77: (234) 567 1234" "6: 90090"           "3: 90090"
## [7] "1: 90090"            "1: 789.345.6712"    "1000: 45123"
```

```r
# First, lets get the phone numbers (since we already have a parser)
phone <- str_extract(vec, "[()]?[0-9]{3}[)]?[ -.]{1}[0-9]{3}[ -.]{1}[0-9]{3}")
phone
```

```
## [1] NA               "123 456 789"    NA               "(234) 567 123"
## [5] NA               NA               NA               "789.345.671"
## [9] NA
```

```r
# Second, get ZIP codes
zip <- str_extract(vec, "[0-9]{5}")
zip
```

```
## [1] "12345" NA      "90089" NA      "90090" "90090" "90090" NA      "45123"
```

```r
# Lastly, lets get the IDs
id <- str_extract(vec, "[0-9]+:")
id
```

```
## [1] "4:"    "6:"    "77:"   "77:"   "6:"    "3:"    "1:"    "1:"    "1000:"
```

```r
id <- str_replace(id, ":", "")
id
```

```
## [1] "4"    "6"    "77"   "77"   "6"    "3"    "1"    "1"    "1000"
```

```r
# Putting it all into one data frame
df <- data.frame(id, phone, zip)
print(df)
```

```
##     id         phone   zip
## 1    4          <NA> 12345
## 2    6   123 456 789  <NA>
## 3   77          <NA> 90089
## 4   77 (234) 567 123  <NA>
## 5    6          <NA> 90090
## 6    3          <NA> 90090
## 7    1          <NA> 90090
## 8    1   789.345.671  <NA>
## 9 1000          <NA> 45123
```

```r
# Now cleaning it
# Assumptions:
# a) no one has two zip codes or phone numbers,
# b) No id is missing.
library(dplyr)
library(tidyr)
df_cleaned <- df %>%
  gather(indicator, value, phone:zip) %>%
  na.omit() %>%
  spread(indicator, value)
df_cleaned
```

```
##     id         phone   zip
## 1    1   789.345.671 90090
## 2 1000          <NA> 45123
## 3    3          <NA> 90090
## 4    4          <NA> 12345
## 5    6   123 456 789 90090
## 6   77 (234) 567 123 90089
```

3 Homework

In this homework, you will be working with an sample data that I scraped from Wikipedia, using the `rvest` package. In this workshop, we do not cover webscraping, but if you are interested in how to do it, you can find the code I used below. Note that since Wikipedia constantly changes, it is possible that the `xpath` specified below might not be accurate at a later point in time.

```r
library(rvest) #Webscraping package
library(stringr) #String methods, aka regular expressions etc.
url <- "https://en.wikipedia.org/wiki/List_of_U.S._states_by_population_density"
tbl <- url %>%
  read_html() %>% #This is from the rvest package
  html_nodes(xpath = '//*[@id="mw-content-text"]/div/table[1]') %>% #xpath from Wikipedia
  html_table() #Put it all into a table.
tbl <- tbl[[1]] #rvest saves it as a table inside list, we want only the table.
write.csv(tbl, "hw5_data.csv", row.names = F)
```

Tasks

1. Load the data called `hw5_data.csv`.
2. Use string operations to remove all `.` from the variable names. Pass the new names to the data frame (i.e. rename the variables).
3. Variables should not start with a letter. This is why `R` automatically attached a leading `X` to the variable `2015.population`. Use string operations to remove both the `X` and the `2015` from this variable name. Then, use the `paste()` function to attach `_2015` to the variable name.
4. Convert all variable names to lower case. Use the `tolower()` function.
5. Use the `str()` function to investigate the structure of the data set. Convert any factor variables to characters before using string operations on them! You can do this manually or google how to convert multiple columns at once.
6. Remove any of the "—" in the population density rank variable and replace it with a missing value.
7. Try converting the variables `population_2015`, `landareami2`, and `landareakm2` to numeric. Why do you think this fails? How can you fix this? Fix it and convert all variables, except the `state` variable to numeric!

Hints:

- Read up on the difference between `str_replace()` and `str_replace_all()` (for 2.).
- Note that `.` is a special character. It is a wild card. This means that it can be used as a place holder for anything, a number, letter, special character, or white space. Use escape characters to remove the `.` in 2.

Bonus If you get tired of manually converting variables from factor to character or from character to numeric, google how to make your life easier! There are multiple ways to do this, but none of them is straighforward. Don't worry if you cannot figure it out. We will go over a solution that uses loops in the next session.

You should get the following output when running `str(data)`:

```r
str(data)
```

```
## 'data.frame':    56 obs. of  10 variables:
##  $ state             : chr  "District of Columbia" "New Jersey" "Puerto Rico" "Rhode Island" ...
##  $ popdensrank       : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ popdensrank50states: num  NA 1 NA 2 3 NA NA 4 NA 5 ...
##  $ densitypopmi2     : num  11011 1218 1046 1021 871 ...
##  $ densitypopkm2     : num  4251 470 404 394 336 ...
##  $ poprank           : num  50 11 29 44 15 53 54 30 55 19 ...
##  $ population_2015   : num  672228 8958013 3680058 1056298 6794422 ...
##  $ landrank          : num  56 46 49 51 45 52 54 48 55 42 ...
##  $ landareami2        : num  61 7354 3515 1034 7800 ...
##  $ landareakm2        : num  158 19047 9104 2678 20202 ...
```

SPEC Lab R Workshop Series: Session 6

Therese Anders

1 for Loops in R

Loops are a staple of programming. Loops allow us to automate our code, and are particularily useful when you find yourself doing a task over and over again. While in practice, we try to vectorize our operations as much as possible (see the solution above where we convert factors to characters), being comfortable with loops is crucial for many programming tasks.

1.1 Basic loop structure

General syntax of a for loop:

`for(iterator in iterations){function/output}`

Lets write a loop that prints out the numbers 1 through 10.

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

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Suppose, the numbers we wanted to print out are part of a vector. We can use loops to iterate through this vector.

```
vec <- seq(11, 20, 1)
for(k in vec){
  print(k)
}
```

```
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
```

Exercise 1 What do you think does the following output do?

```
for(l in 5:length(vec)){
  print(l)
}
```

Of course, we can use loops to automate more complex tasks. For example, we could use it to change a batch of variables to `character()`. To try this, re-load the data from session 5.

```
data_new <- read.csv("hw5_data.csv")
str(data_new)
```

```
## 'data.frame':    56 obs. of  10 variables:
##  $ State             : Factor w/ 56 levels "Alabama","Alaska",..: 10 33 43 44 24 13 49 8 3 23 ...
##  $ Pop..dens..Rank        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Pop..dens.Rank50.states: Factor w/ 51 levels "-","1","10","11",..: 1 2 1 13 24 1 1 35 1 46 ...
##  $ Density.Pop...mi2.     : int  11011 1218 1046 1021 871 808 799 741 721 618 ...
##  $ Density.Pop...km2.     : int  4251 470 404 394 336 314 308 286 279 238 ...
##  $ Pop..Rank              : int  50 11 29 44 15 53 54 30 55 19 ...
##  $ X2015population      : Factor w/ 56 levels "1,032,949","1,056,298",..: 48 53 27 2 45 15 11 26 39 41 ...
##  $ Land.Rank              : int  56 46 49 51 45 52 54 48 55 42 ...
##  $ Landarea.mi2.       : Factor w/ 56 levels "1,034","1,949",..: 38 43 14 1 44 11 7 20 47 54 ...
##  $ Landarea.km2.       : Factor w/ 56 levels "1,477,953.4",..: 20 26 55 30 31 50 45 8 29 39 ...
```

```
test1 <- data_new
for(l in 1:ncol(test1)){
  test1[,l] <- as.character(test1[,l])
}
str(test1)
```

```
## 'data.frame':    56 obs. of  10 variables:
##  $ State             : chr  "District of Columbia" "New Jersey" "Puerto Rico" "Rhode Island" ...
##  $ Pop..dens..Rank        : chr  "1" "2" "3" "4" ...
##  $ Pop..dens.Rank50.states: chr  "-" "1" "-" "2" ...
##  $ Density.Pop...mi2.     : chr  "11011" "1218" "1046" "1021" ...
##  $ Density.Pop...km2.     : chr  "4251" "470" "404" "394" ...
##  $ Pop..Rank              : chr  "50" "11" "29" "44" ...
##  $ X2015population        : chr  "672,228" "8,958,013" "3,680,058" "1,056,298" ...
##  $ Land.Rank              : chr  "56" "46" "49" "51" ...
##  $ Landarea.mi2.          : chr  "61" "7,354" "3,515" "1,034" ...
##  $ Landarea.km2.          : chr  "158.0" "19,046.8" "9,103.8" "2,678.0" ...
```

As a more sophisticated version, we could convert only those variables that are factors (not numerical variables like `Density.Pop....km2..`) to characters, using an `if` statement.

```
test2 <- data_new
for(k in 1:ncol(test2)){
  if(is.factor(test2[,k])){
    test2[,k] <- as.character(test2[,k])
  }
}
str(test2)
```

```
## 'data.frame':    56 obs. of  10 variables:
##  $ State             : chr  "District of Columbia" "New Jersey" "Puerto Rico" "Rhode Island" ...
##  $ Pop..dens..Rank        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Pop..dens.Rank50.states: chr  "-" "1" "-" "2" ...
##  $ Density.Pop...mi2.     : int  11011 1218 1046 1021 871 808 799 741 721 618 ...
##  $ Density.Pop...km2.     : int  4251 470 404 394 336 314 308 286 279 238 ...
##  $ Pop..Rank              : int  50 11 29 44 15 53 54 30 55 19 ...
##  $ X2015population        : chr  "672,228" "8,958,013" "3,680,058" "1,056,298" ...
##  $ Land.Rank              : int  56 46 49 51 45 52 54 48 55 42 ...
##  $ Landarea.mi2.          : chr  "61" "7,354" "3,515" "1,034" ...
##  $ Landarea.km2.          : chr  "158.0" "19,046.8" "9,103.8" "2,678.0" ...
```

2 Data cleaning in R

In this example, we will use our new data management skills to clean a data set for inclusion in SPEC's IPE data resource. The data are figures on US Foreign Direct Investment (FDI) from the Bureau of Economic Analysis. Here are the data cleaning tasks that we are going to do:

1. Recode missing values.
2. Delete the header and footer.
3. Add COW country codes.
4. Drop observations from combined countries. Drop duplicated observations from Poland, Bahamas, Hungary, Czech Republic, Russia, Jamaica, Trinidad and Tobago, Guatemala, and conflicted observations from Serbia/Yugoslavia, Russia/USSR, Zaire/Congo, Timor-Leste, Micronesia, and Samoa.
5. Reshape data to long format.

To start, lets read the data and take a look at it using the `View()` in RStudio.

```r
library(foreign)
library(tidyverse)
fdi <- read.csv("fdidata.csv",
                stringsAsFactors = F)
```

2.1 Recoding missing values

The original data specifies a number of different ways of how missing values are coded, specifically `n.s.`, `(*)`, `--`, `---`, `----`, and `(D)`. In principle, we could recode the missing values with the following command.

`fdi[fdi == "(D)" | fdi == "n.s." | fdi == "(*)" | fdi == "----" | fdi == "---" | fdi == "--"] <- NA`

However, we could also make our life easier by specifying all possible values for `NA` when reading the data.

```r
fdi_nona <- read.csv("fdidata.csv",
                     stringsAsFactors = F,
                     na.strings = c("(D)", "n.s.", "(*)", "----", "---", "--"))
```

2.2 Dropping header and footer

There are a number of ways we could drop the header and footer. In principle, we could just inspect the data frame and manually delete all the rows that belong to the header or footer. However, there is a pattern here: The header and footer rows do not have entries in the second column. Note that when the value of the second column is `NA`, `R` would also drop that row. Therefore, we have to account for both, rows that have a non-empty value or `NA` in the second column, when selecting the rows to eliminate.

```r
empty <- fdi_nona[,2] == ""
head(empty, 10)
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE    NA
```

```r
fdi_new <- fdi_nona[empty %in% c(NA, F),]
```

Subsequently, we want to turn the second row into the column names and drop the first and second rows.

```r
names(fdi_new) <- unname(fdi_new[2,])
fdi_new <- fdi_new[-c(1,2),]
```

2.3 Stripping white space and dropping irrelevant observations

After renaming the first column of our new dataframe to `country`, lets look at the observations (i.e. countries that we have in this data set). Use the `View()` function or the Environment menu to inspect the values of the variable `country`.

```r
names(fdi_new)[1] <- "country"
head(fdi_new$country)
```

```
## [1] "Canada"            "Europe"            "  Austria"
## [4] "  Belgium"         "  Czech Republic" "  Denmark"
```

The first thing to notice is that many elements of the `country` variable have leading whitespace (i.e. spaces or tabs). This is a problem when trying to attach countrycodes later on. We will use operations from the `stringr` package to get rid of this leading whitespace. Note that we do **not** want to remove all white space, because country names such as `Czech Republic` would not be recognized by the `countrycode` package any longer if we stripped the character value from all whitespace. We will use `stringr`'s `str_trim()` function to remove leading (and trailing) whitespace from all elements of the variable `country`.

```r
fdi_new$country[5]
```

```
## [1] "  Czech Republic"
```

Second, there are a number of observations in the `country` variable that do not contain proper country names, such as `Other`, `Other Western Hemisphere`, or `Latin America and Other Western Hemisphere`. We can use regular expressions to drop these observations.

```r
nrow(fdi_new)
```

```
## [1] 242
```

```r
fdi_new_sub <- fdi_new %>%
  mutate(country = str_trim(country)) %>%
  filter(!str_detect(country, "[Oo]ther"))
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```r
nrow(fdi_new_sub)
```

```
## [1] 233
```

2.4 Adding COW country codes

Next, we will use the `countrycode` package to add COW country codes to the data. Note, that this does not work perfect here–some countrynames are not matched. For demonstration purposes we will simply drop observations that were not matched. In reality, we might have to add some country codes manually, or use a more sophisticated algorithm to attach country codes. Note that the countrycode package allows you to specify custom country dictionaries, for example if you needed to attach Gleditsch-Ward rather than COW countrycodes.

```r
# install.packages("countrycode")
library(countrycode)
fdi_new_sub <- fdi_new_sub %>%
  mutate(ccode = countrycode(country, "country.name", "cown"))
```

We can inspect the values that did not receive a COW code. The algorithm works as expected (Serbia does not have its own countrycode in the COW system). All the un-matched countries are either not fully sovereign regions (based on the COW system), continents, or smaller island nations that are not considered part of the international system, according to COW. We therefore drop all observations that did not receive a `ccode` coding.

```r
fdi_new_sub$country[is.na(fdi_new_sub$ccode)]
```

```
##  [1] "Europe"                           "Gibraltar"
##  [3] "Greenland"                        "Serbia"
##  [5] "South America"                    "French Guiana"
##  [7] "Central America"                  "Bermuda"
##  [9] "Netherlands Antilles"             "Anguilla"
## [11] "Aruba"                            "Curaao"
## [13] "French Islands, Caribbean"        "Netherlands Antilles"
## [15] "Netherlands Islands, Caribbean" "Sint Maarten"
## [17] "Africa"                           "Middle East"
## [19] "Iraq-Saudi Arabia Neutral Zone" "Asia and Pacific"
## [21] "Hong Kong"                        "French Islands, Indian Ocean"
## [23] "French Islands, Pacific"          "Macau"
## [25] "Micronesia"
```

```r
fdi_new_sub <- filter(fdi_new_sub, !is.na(ccode))
```

2.5 Handling duplicates

Before re-shaping our data, we need to check for duplicates. `R` provides a number of built-in functions to execute this task. Here, we will write a snipped of custom code using `dplyr` to show which observations (based on the `ccode` variable) have duplicates and how many.

```r
dupes <- fdi_new_sub %>%
  group_by(ccode) %>%
  summarise(count = n()) %>%
  filter(count > 1)
```

We have quite a few duplicate observations. In reality, we would want to go through each of these observations and inspect whether they are "true" duplicates, or whether they are produced in the process of attaching COW codes. Here, we will only go through two examples.

First, let us look at all the duplicates with COW code 200 (United Kingdom). If we inspect all the observations, we see that we have one "true" UK observations, and a number of other observations that contain the word "United Kingdom," but do not refere to the mainland. What to do with these duplicates is a substantive question that needs to be decided by the researcher. Here, for the purpose of demonstation, we will assume that all the observation belong to the UK and sum over them (taking into account the NA values).

```r
ccode200 <- fdi_new_sub %>%
  filter(ccode == 200)
```

Second, let us look at the cuplicates with COW code 365 (Russia). We can see that these are not "true" duplicates, that is no value is observed in two rows in the same year. We can therefore simply "melt" these three variables together to create one row for ccode 365.

```r
ccode365 <- fdi_new_sub %>%
  filter(ccode == 365)
```

Depending on the type of duplicate, we might want to apply different functions to each instance of duplication. For simplicity, here we simply sum over all duplicates, excluding missing values. This will achieve the desired transformation for both the UK and the Russia duplicates. Note that before summing, we will have to convert our values to class `numeric`. Note also that upon applying the `summarise_if()` command, we loose the information on the `country` variable that contains the name of the country. There are ways to retain this information (for example by subsequently attaching the countryname with the `countrycode` package, see below).

```r
for(i in 2:ncol(fdi_new_sub)){
  fdi_new_sub[,i] <- as.numeric(fdi_new_sub[,i])
}
str(fdi_new_sub)
```

```
## 'data.frame':    208 obs. of  35 variables:
##  $ country: chr  "Canada" "Austria" "Belgium" "Czech Republic" ...
##  $ 1982   : num  43511 562 5549 NA 1155 ...
##  $ 1983   : num  44779 548 5087 NA 1275 ...
##  $ 1984   : num  47498 534 5202 NA 1263 ...
##  $ 1985   : num  47934 509 5619 NA 1383 ...
##  $ 1986   : num  52006 736 5568 NA 1164 ...
##  $ 1987   : num  59145 711 7719 NA 1120 ...
##  $ 1988   : num  63900 697 7830 NA 1182 ...
##  $ 1989   : num  63948 962 7710 NA NA ...
##  $ 1990   : num  69508 1113 9464 NA 1726 ...
##  $ 1991   : num  70711 1268 10611 NA 1940 ...
##  $ 1992   : num  68690 1371 11381 NA 1676 ...
##  $ 1993   : num  69922 1312 11697 NA 1735 ...
##  $ 1994   : num  74221 2197 14714 NA 2030 ...
##  $ 1995   : num  83498 2829 18706 NA 2161 ...
##  $ 1996   : num  89592 2854 18740 NA 2554 ...
##  $ 1997   : num  96626 2646 17337 NA 2385 ...
##  $ 1998   : num  98200 3856 17899 NA 2764 ...
##  $ 1999   : num  119590 3848 21756 1038 3846 ...
##  $ 2000   : num  132472 2872 17973 1228 5270 ...
##  $ 2001   : num  152601 3964 22589 1179 5160 ...
##  $ 2002   : num  166473 4011 25727 1264 6184 ...
##  $ 2003   : num  187953 6366 27415 1668 5597 ...
##  $ 2004   : num  214931 9264 41840 2444 6815 ...
##  $ 2005   : num  231836 11236 49306 2729 6914 ...
##  $ 2006   : num  205134 14897 51862 3615 5849 ...
##  $ 2007   : num  250642 14646 62491 4066 8950 ...
##  $ 2008   : num  246483 13546 65279 5053 10481 ...
##  $ 2009   : num  274807 10954 46610 5372 13053 ...
##  $ 2010   : num  295206 11485 43975 5268 11802 ...
##  $ 2011   : num  330041 12556 50984 5840 14942 ...
##  $ 2012   : num  366709 14327 49144 6016 14306 ...
##  $ 2013   : num  390172 15641 51702 6990 13605 ...
##  $ 2014   : num  386121 15787 48128 7247 14108 ...
##  $ ccode  : num  20 305 211 316 390 375 220 255 350 310 ...
```

```r
fdi_new_sub_nodupes_alt <- fdi_new_sub %>%
  group_by(ccode) %>%
  summarise_if(is.numeric, funs(sum(., na.rm = T)))
```

Unfortunately, this last version returns 0 for columns that have all `NA` values. We will therefore write a custom `function` to pass to the `summarise_if()` wrapper. The general syntax of a function is the following.

`functionname <- function(operand){operation, return value}`.

Within the function we use an `if...else` statement. The `if...else` statement specifies a conditional operation with the following general syntax:

`if(condition is true){output} else {output}`.

```r
func_sum <- function(x){
  if(all(is.na(x))){
    return(NA)
  } else {
    return(sum(x, na.rm = T))
  }
}

fdi_new_sub_nodupes <- fdi_new_sub %>%
  group_by(ccode) %>%
  summarise_if(is.numeric, funs(func_sum))
```

2.6 Reshaping

The original data is in wide format. To make it compatible with the IPE data resource (and most other panel data sets) we need to reshape it into the long format, with one column indicating the country, another indicator accounting for the year, and lastly a column capturing the values of the respective indicator. In this case, the entire dataframe of the original data captures only one variable, namely "Outward FDI stocks from the US in USD (millions), all industries, BEA [OFS]." For simplicity, we will call this variable `outward_fdi_all`.

```r
fdi_long <- fdi_new_sub_nodupes %>%
  # re-shape to long format
  gather(year, outward_fdi_all, 2:ncol(fdi_new_sub_nodupes)) %>%
  # Re-attach country names using countrycode package
  mutate(countryname = countrycode(ccode, "cown", "country.name"))
```

The result is a clean data frame that can now be merged into the IPE data resource.

```r
head(fdi_long)
```

```
## # A tibble: 6 x 4
##   ccode year  outward_fdi_all countryname
##   <dbl> <chr>           <dbl> <chr>
## 1    20 1982            43511 Canada
## 2    31 1982             3121 Bahamas
## 3    40 1982               NA Cuba
## 4    41 1982               19 Haiti
## 5    42 1982              188 Dominican Republic
## 6    51 1982              386 Jamaica
```

Sources

U.S. Bureau of Economic Analysis, “U.S. Direct Investment Abroad, U.S. Direct Investment Position Abroad on a Historical-Cost Basis,” http://www.bea.gov/international/di1usdbal.htm (accessed Jun 21 2016).

APPENDIX B: Pitch Document for Recruiting Private-Sector Partners

Research Lab K Pipeline Partnerships: Building Diversity in Data Science

The field of data science lacks diversity. This is a problem that begins long before candidates are interviewed for positions at top companies. At the undergraduate level, too few students from under-represented groups are receiving the training and the mentoring necessary to prepare them to enter and thrive in data-intensive fields – there are not enough women, minority, and first-generation students entering the pipeline. The Pipeline Partnership program at the University K offers companies a collaborative opportunity to fill their data-science and social-science pipelines with talented minority, first-generation and female candidates. In these partnerships, firms share their training and recruitment needs, Research Lab K recruits and trains a diverse pool of students, called Pipeline Fellows, to meet those needs, and firms commit to interviewing some of those students each year for internships and post-graduation employment.

Pipeline Fellows gain their training and experience as members of the Research Lab K. Overseen by [REDACTED], Research Lab K is a proven vehicle for addressing the pipeline problem. The lab recruits promising female, minority and first-generation students early in their college careers and brings them into a close-knit, supportive research community in which they work for three years. We provide hands-on faculty mentoring, top-end technical training in research design and applied data science, and professionalization workshops covering everything from scientific communication to small-team leadership. Students gain direct experience conducting policy-relevant research in two inter-related issue areas: resolving violent political conflict and generating sustainable economic development. Students code data, clean, merge, and manipulate large datasets, conduct analysis, create data visualizations, co-author op-eds, and present at academic conferences. Students gain valuable applied experience solving complex applied problems in small teams and, in their junior year, leading those small teams. Our students work primarily in R, but many also develop skills in Python, Tableau, and Javascript (D3).

Many students from under-represented groups must work part-time jobs to fund their living expenses and defray tuition costs, and these jobs can prevent them from taking on unpaid internships or pursuing valuable extra-curricular opportunities. Support from Pipeline Partners allows the lab to support these students financially and provide training and recruitment services specific to firm needs. Research Lab K even conducts collaborative research with some of our partners, with students addressing research questions of joint interest to the lab and the firm.

Three Tiers of Partnership

Pipeline Partner (Tier 1): Firms share their recruitment needs, Research Lab K matches firms to qualified applicants, and the firm commits to interviewing some of these candidates each year.

Pipeline Partner (Tier 2): Firms provide financial support to fund student stipends. The Lab recruits fellows with interests matched to an individual firm and, over the course of the fellows' time in the lab, provides training specific to firm needs.

Research Partner: Firms provide funding that allows a team of Pipeline Fellows, under faculty and doctoral student supervision, to address a research question of joint interest to the lab and the firm. Fellows provide written deliverables as appropriate and a formal presentation of findings.

Sample Academic Calendar for a Data Science Fellow (Tier 2)		
Term	**Activity**	**Cost to Sponsor**
Sophomore Year	Training and Applied Research	$X
Sophomore Summer	Applied Research	$X
Junior Year	Applied Research (Team Leader)	$X
Junior Summer	Intern at Sponsoring Firm	At Company's Discretion
Senior Year	Honors Thesis	$X

APPENDIX C: Survey Instrument

Note: The following instrument is administered electronically via Qualtrix.

Undergraduate Research Impact Study

Please note that your answers will be kept anonymous and they will not be traced back to you in any way. In addition, the information below will not and cannot be used to evaluate your performance.

1. Data Literacy

- How comfortable are you with the following skills?

  - Finding good information
  - Finding good sources of data
  - Data management
  - Data analysis
  - Data visualization
  - Using data as a tool in support of an argument
  - Translating data and results into words

    - Options: Very comfortable/Comfortable/Somewhat uncomfortable/Not at all comfortable/NA - I don't know

- During the current school year, about how often have you done the following?

  - Asked questions or contributed to course discussions in other ways
  - Prepared two or more drafts of a paper or assignment before turning it in
  - Come to class without completing readings or assignments
  - Asked another student to help you understand course material
  - Given a course presentation

    - Options: *Very often, Often, Sometimes, Never*

- During the current school year, about how often have you done the following?

  - Combined ideas from different courses when completing assignments
  - Connected your learning to societal problems or issues
  - Examined the strengths and weaknesses of your own views on a topic or issue
  - Learned something that changed the way you understand an issue or concept
  - Connected ideas from your courses to your prior experiences and knowledge

  - Options: *Very often, Often, Sometimes, Never*

- During the current school year, about how often have you done the following?

  - Reached conclusions based on your own analysis of numerical information (numbers, graphs, statistics, etc.)
  - Used numerical information to examine a real-world problem or issue (unemployment, climate change, public health, etc.)
  - Evaluated what others have concluded from numerical information

    - Options: *Very often, Often, Sometimes, Never*

- During the current school year, about how often have you done the following?

  - Completed an assignment that used an information source (book, article, website, etc.) other than required course readings
  - Worked on a paper or project that had multiple smaller assignments such as an outline, annotated bibliography, rough draft, etc.
  - Received feedback from an instructor that improved your use of information resources (source selection, proper citation, etc.)
  - Completed an assignment that used the library's electronic collection of articles, books, and journals (JSTOR, EBSCO, LexisNexis, ProQuest, etc.)
  - Decided not to use an information source in a course assignment due to its questionable quality
  - Changed the focus of a paper or project based on information you found while researching the topic
  - Looked for a reference that was cited in something you read
  - Identified how a book, article, or creative work has contributed to a field of study

- How much has your time as an undergraduate research assistant contributed to your comfort with the following skills?

  - Finding good information
  - Finding good sources of data
  - Data management
  - Data analysis
  - Data visualization
  - Using data as a tool in support of an argument
  - Translating data and results into words

    - Options: A great deal/Somewhat/Not at all

- In what ways, if any, has your research experience increased your skills in any of the areas listed above?

[Open response]

- How likely are you to use data in research assignments in your future coursework?
    - Options: Very likely/Somewhat likely/Somewhat unlikely/Very Unlikely
- How likely are you to use data in research assignments in your future career?
    - Options: Very likely/Somewhat likely/Somewhat unlikely/Very Unlikely
- How well prepared do you feel to use data in your career?
    - Options: Very prepared/Somewhat prepared/Somewhat unprepared/Very unprepared

2. Student interaction with peer-, near-peer, and faculty mentors

- During the current term, how often have you:
    - Asked a fellow student to help you understand the steps for performing a task?
    - Explained the steps for performing a task to a fellow student?
    - Prepared for meetings with your research adviser by discussing or working through your task assignments with other students?
    - Worked with other students on research projects or assignments?
        - Options: Very often/Often/Sometimes/Never
- During the current term, how often have you:
    - Talked about career plans with a faculty member?
    - Worked with a faculty member on activities other than your current research project?
    - Discussed research topics, ideas, or concepts with a faculty member outside of your current research project?
    - Discussed your academic performance with a faculty member?
        - Options: Very often/Often/Sometimes/Never

- During the current term, how often have you:
  - Talked about career plans with a graduate student?
  - Worked with a graduate student on activities other than your current research project?
  - Discussed research topics, ideas, or concepts with a graduate student outside of your current research project?
  - Discussed your academic performance with a graduate student?

    - Options: Very often/Often/Sometimes/Never

- How would you rate the quality of faculty-student interaction at your college?

    - Options: Excellent/Very Good/Good/Fair/Poor/NA

- How would you rate the quality of faculty-student interaction working as an undergraduate research assistant?

    - Options: Excellent/Very Good/Good/Fair/Poor/NA

- How would you rate the quality of graduate-undergraduate student interaction at your college?

    - Options: Excellent/Very Good/Good/Fair/Poor/NA

- How would you rate the quality of graduate-undergraduate student interaction working as an undergraduate research assistant?
    - Options: Excellent/Very Good/Good/Fair/Poor/NA

3. Level of comfort with mentors

- How comfortable are you raising potentially sensitive issues with a faculty member (e.g., finances, performance in courses, etc.)?

    - Options: Very comfortable/Somewhat Comfortable/Not at all comfortable

- Have you raised sensitive issues with a faculty member?

    - Options: Yes/No

    b. Was the faculty member able to help you resolve the issue or connect you with the appropriate resources?

  - Options: Yes/No/Somewhat

- How comfortable are you raising potentially sensitive issues with a graduate student (e.g., finances, performance in courses, etc.)?

  - Options: Very comfortable/Somewhat Comfortable/Not at all comfortable

- Have you raised sensitive issues with a graduate student?

  - Options: Yes/No

  b. Was the graduate student able to help you resolve the issue or connect you with the appropriate resources?

  - Options: Yes/No/Somewhat

4. Sense of belonging to a learning community

- How often do you receive recognition or praise for doing good work?

  - Options: Very often/Often/Sometimes/Never

- Does your faculty research mentor seem to care about you as a person?

  - Options: Yes/No/Somewhat

- Rate the extent to which you agree or disagree with the following statements concerning your undergraduate research experience.

  My faculty research mentor:
  - Encourages me to use him or her as a sounding board to explain hopes, ideas, feelings
  - Expresses confidence in my abilities to succeed academically
  - Requires me to reflect on competencies needed in achieving my future goals
  - Assists me in mapping out realistic strategies to achieve my academic goals
  - Encourages me to consider educational opportunities beyond my current plans
  - Shares personal examples of difficulties they have had to overcome
  - Makes me feel that I belong in college
    - Options: Strongly Agree/Agree/Neither Agree nor Disagree/Disagree/Strongly Disagree

5. Interest in and preparation for graduate school and post-bac endeavors

- How interested were you, if at all, in attending graduate school or pursuing a research career prior to participating in undergraduate research?

    - Options: Very interested/Somewhat interested/Not at all interested

- How interested are you, if at all, in attending graduate school or pursuing a research career?

    - Options: Very interested/Somewhat interested/Not at all interested

    b. If you now have an increased interest in attending graduate school or pursuing a research career, what factors have influenced you?

    [Open response]

- Rate the extent to which you agree or disagree with the following statements concerning your undergraduate research experience.

  Participating in undergraduate research has helped me…
  - Enhance my professional or academic credentials
  - Clarify a potential career path
  - Understand the research process in my field
  - Study a topic in depth
  - Develop a continuing relationship with a faculty member
  - Learn to work independently
  - Overcome obstacles in the research process
  - Understand how scientists think
  - Understand how professionals work on real problems

    - Options: Strongly Agree/Agree/Neither Agree nor Disagree/Disagree/Strongly Disagree/NA

6. Motivation

- Students have many motivations for choosing to participate in undergraduate research. For each of the following reasons that students might have doing undergraduate research, please let us know how important they were to you when deciding to pursue this opportunity.
    - Learning more about a topic I'm interested in
    - Learning new skills
    - Getting to know a faculty member
    - Getting to know other students

- Earning money
- Earning academic credit
- Improving prospects for admission to graduate school
- Improving future job prospects

  - Options: Very important/Somewhat important/Not at all important

- How are you being compensated for your work as an undergraduate researcher?
  Options: hourly pay, stipend or fellowship, academic credit, other, I am not being compensated outside of gaining research experience.

7. Demographics

- Please select your year in college
  - First-year student
  - Sophomore
  - Junior
  - Senior

- Please select your primary academic discipline
  - Humanities
  - Social Sciences
  - Natural Sciences
  - Undecided

- Please list your major:

- What is the highest level of education you ever expect to complete?
  - Some college but less than a bachelor's degree
  - Bachelor's degree (B.A., B.S., etc.)
  - Master's degree (M.A., M.S., etc.)
  - Doctoral or professional degree (Ph.D., J.D., M.D., etc.)

- What is the highest level of education completed by either of your parents (or those who raised you)?
  - Did not finish high school
  - High school diploma or G.E.D.
  - Attended college but did not complete degree
  - Associate's degree (A.A., A.S., etc.)
  - Bachelor's degree (B.A., B.S., etc.)
  - Master's degree (M.A., M.S., etc.)
  - Doctoral or professional degree (Ph.D., J.D., M.D., etc.)

- Which racial group(s) do you identify with? These categories from the U.S. Census might not accurately describe you, but please let us know which is closest to your racial identity.
    - White
    - Black or African American
    - American Indian or Alaska Native
    - Asian
    - Native Hawaiian or Pacific Islander

- Which gender do you identify with?
    - Female
    - Male
    - Other, please specify:

- For how many semesters have you participated in undergraduate research?
    - Zero. This is my first semester.
    - One
    - Two
    - Three
    - Four
    - Five
    - Six
    - Seven

7. Improving the Undergraduate Research Experience

- Is there an area in which your undergraduate research experience has exceeded your expectations?

  [Open response]

- Is there an area in which your undergraduate research experience has fallen short? If you have any suggestions for how we can improve in this area, please let us know, but simply helping us diagnose areas in which we need to improve is also appreciated.

  [Open response]

- Is there anything else you'd like us to know?

  [Open response]