#### Data Science for Economists

Lecture 3: Data tips

Kyle Coombs Bates College | ECON/DCS 368

#### Table of contents

- Prologue
- Empirical Workflow
  - Downloading data
  - File formats
  - Archiving & file compression
  - Data checks
- Big Data file types (if time)
- Dictionaries (if time)

Prologue

## Prologue

Today we'll focus on grappling with data

- Checklist to ensure data quality
- File formats and extensions
- Archiving & file compression
- If time:
  - Dictionaries (hash tables)
  - Big Data file types

#### Why do we need to do this?

- We summarize data because we can't look at every data point and see a pattern
- But lots of data are messy or frankly bogus, but you wouldn't know it from sum stats
- Meet Anscombe's Quartet (Anscombe, 1973)



#### DataSarus Dozen

Sample: away



Data from the Datasaurus Dozen. Each dataset has the same summary statistics, but looks very different.

#### Not every odd dataset is wrong

- Sometimes bizarre looking dataset is real (see the Japan Phillips curve)
- For example, sometimes there are real outliers in the data
  - You'll need to decide what to do about them
- But sometimes they're a sign of nonsense or "NA"/missing values
- If you find an oddity in your data, you still have to decide what it is and how to handle it
  - Maybe that odd outlier is real, so you shouldn't drop it
  - Are the data missing? Due to randomness or a systematic error?
- Throughout this course we'll think about how to deal with these issues
- Today we're making sure you have safeguards in place so you don't write an entire paper only to discover your dataset looks like a T-rex

### Types of data

- 1. "Long" data (a.k.a. "Big-N" data because *N* very, very large [and may not all fit onto a single hard drive!], government tax records, Medicare claims data, etc.)
- 2. "Wide" data (a.k.a. "Big-K" data because K > N, customer data sets where each click is a variable)
- 3. "Wild" data (unstructured; happenstance; collected without a particular intention; e.g. twitter, contrast with Census surveys)
- 4. "Big Data" is a catch-all for any combination of the above data types that is hard to analyze with classical methods like OLS regression
  - Too many variables
  - Too many observations
  - Needs special wrangling or analysis

#### Long data

E 5- d	ne Insert	Page Layout Formulas	Data Rev		Γiew Ω⊺e						data_examp	les - Excel													– CI X Sign in A Share
🚔 🔏 Cut	08		= = =	Dr.	Wran Text		General			<b>.</b>	Normal		lad	Good		Neutral	Ca	kulation	-	F. 7	< 🖬	∑ AutoSu	m * As	0	
Paste Copy	*	x u - 1 - 1 - A - A -			E map rea		e or e	±0.00	[≢] Conditional	Format a	Check Ce		Svelanatory	Inout		Linked Co		te		Insert Dela	te Format	🕹 Fill -	Z 1 Sort	R Find R	
🗸 ؇ Forma	at Painter	I U I I I I I I A I		20 20	🖽 Merge 80	Center *	3 - % '	100 -010	Formatting *	Table *	chiedx ce		Aplanatory	mpor		Ciliked de		ne -	- Ŧ	* *	*	🧶 Clear 🔻	Filter	* Select *	
Clipboard	9	Font 5		Align	ment	G.	Numbe	r 5						Styles						Cel	5		Editing		
AB31 •	I X V	√ fx																							
				1		1	1 .		1		1		1				-	-			1		1	1 -	1
A	B	C U	) E		6	н		J	ĸ	L	M	N	0	P	Q	к	5	1	0	V	w	X	Ŷ	2	AA A
1 person_id	income	years of education gende	er																						
2	101 \$ 8,825.2	3 12 F																							
5	102 \$38,336.1	1 14 M																							
4	103 \$ 8,641.7	3 13 F																							
5	104 \$10,024.0	9 13 M																							
0	105 \$79,923.3	6 12 M																							
/	106 \$57,007.0	0 14 M																							
0	107 339,494.8	1 15 1																							
9	108 \$92,150.4	1 13 M																							
10	109 \$75,373.3	0 13 F																							
1	110 \$15,680.3	0 13 M																							
12	111 \$46,593.4	1 13 F																							
13	112 \$71,386.7	1 15 M																							
14	113 \$72,674.9	6 11 M																							
15	114 \$58,535.1	2 12 M																							
16	115 \$11,968.9	1 12 F																							
17	116 \$99,265.2	7 14 M																							
18	117 \$46,181.1	1 11 F																							
19	118 \$74,175.5	9 15 M																							
80	119 \$73,409.8	6 11 F																							
21	120 \$65,784.2	6 14 M																							
22	121 \$ 3,532.2	6 14 M																							
23	122 \$33,836.9	5 15 M																							
24	123 \$56,806.5	8 13 F																							
25	124 \$68,478.3	1 13 M																							
26	125 \$60,566.2	2 15 F																							
27	126 \$98,447.4	1 13 F																							
28	127 \$79,397.9	0 11 F																							
29	128 \$17,594.7	5 12 F																							
30	129 \$84,667.9	3 13 M																							
81	130 \$87,953.7	1 13 M																							
12	131 \$68,423.7	4 14 F																							
13	132 \$51,357.6	2 13 M																							
14	133 \$82,233.8	6 12 F																							
15	134 \$92,901.9	1 14 M																							
16	135 \$75,153.3	5 13 M																							
37	136 \$29,740.9	4 15 M																							
38	137 \$ 795.3	6 13 F																							
19	138 \$27,283.4	6 12 M																							
40	139 \$ 1,137.3	7 12 F																							
41	140 \$61,127.8	0 13 M																							
42	141 \$33,153.0	6 12 F																							
43	142 \$19,774.7	3 15 M																							
44	143 \$55,925.9	7 13 M																							
15	144 \$ 25 598 8	1 15 M			_																	_		_	
	long wide	e   Sheet3   (+)													4										Þ
leady																								U -	+ 10

- Main application: *identifying causal effects*
- Example: effects of improving schools on income

#### Wide data

Ele Home Ins	vent Pa	ae Lavout	Formula	s Data	Review	View	Ø Tell me	what you wa	nt to do			đ	ata_examples (	l) - Excel												œ	— Sian in	07 X Q.Share
Paste Vormat Painter	Calibri B I	• =   • <u>u</u>	11 - A	= = = =	* = * •	₩	rap Text erge & Centr	Gener	al %	•.0 .00 F	Conditional cormatting *	Format as Table -	Normal Check Cell	Bad Expla	natory	Good Input		Neutral Linked Cell	Calcula	tion -	insert	Delete F	Format	∑ AutoSur ↓ Fill + ℓ Clear +	n * A Z Sort Filter	8. Find 8. • Select •		
Clipboard 5	1	Font		6	Aliq	gnment		6	Number	G.					Stj	les						Cells			Editing			^
E1 * i i	X V	f <sub>x</sub> ad	_click1																									v
	D	E	F	G	н	1	J	κ	L	м	N	0	P	Q	R	S		т U	v	w		x	Y	z		AA	AB	AC
1 years of education	gender	ad_click1	ad_click2	ad_click3 a	ad_click4 ad	d_click5 a	ad_click6 a	d_click7 ad	d_click8 a	ad_click9	ad_click10	ad_click	11 ad_click1	2 ad_click1	3 ad_click	14 ad_click1	15 ad_c	click16 ad_clic	k17 ad_cli	ck18 ad_cli	k19 ad	click20	ad_click	k21 ad_clic	k22 ad_0	lick23 ad	click24 ad	d_click25
2 12	F M	0	1	1	1	0	1	0	0	0	1	0	1	1	1	1	0	1	1	1	1	1		1	1	1	0	0
4 12	F	0	0	1	0	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1	1	0		1	0	1	1	1
5 12	м	1	0	0	0	0	0	1	1	0	1	1	1	0	1	1	0	1	0	1	0	0		1	1	0	1	1
6 12 7 14	M	0	0	0	0	0	0	1	1	1	1	0	1	0	1	0	0	1	1	0	0	1		0	1	1	1	0
8 11	F	1	1	0	1	0	1	0	1	0	1	1	1	1	1	0	0	ō	0	0	1	1		0	0	0	1	ō
9 15	М	1	0	0	1	1	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0	1		1	1	0	1	0
10 14	F	1	1	0	1	0	1	1	0	0	1	1	1	0	1	0	1	1	1	0	0	1		1	1	0	1	1
12				1	J	-	0	-	-				-			1	-		-	-	1			1		-	-	
13																												
14																												
16																												
17																												
18																												
20																												
21																												
22																												
23																												
25																												
26																												
27																												
29																												
30																												
31																												
33																												
34																												
35																												
36																												
38																												
39																												
40																												
41																												
43																												
44																												
< → long	wide	Sheet3	(+)										1				L C	1	-	1	-						-	•
Ready																									000 B	IJ	-	+ 100%

- Main application: *prediction*
- Example: predicting income to target ads from tons of information like location, links clicked, etc.

#### Wild Data

```
<caption>List of men's Olympic records in athletics
</caption>
Event
Record
Athlete(s)
Nation
Games
Date
Ref(s)
\langle th \rangle \langle tr \rangle
<span data-sort-value="00100&#160;!"><a href="/wiki/100 metres" title="100 metres">100
9.63 
<span data-sort-value="Bolt, Usain"><span class="vcard"><span class="fn"><a href="/wiki/Usain Bolt"
```

#### Why does data type matter?

#### Why does data type matter?

- 1. Data type determines how much memory is required to store information
- 2. Data type determines what method you can use to read and analyze the data

#### Why does data type matter?

- 1. Data type determines how much memory is required to store information
- 2. Data type determines what method you can use to read and analyze the data
  - A difference-in-difference model requires a different data shape than a regression discontinuity model
  - You cannot have a wide data set with one row per unit and one column per year

## What question are you asking?

- Any dataset, no matter how big, has simplified the world in some way
- You want the simplification to match the question
- How do you record where a person is?
  - County? Lots of people have same location.
  - IP address? Changes frequently
  - GPS coordinates? Too precise, and changes every second!
- Your question and theory should guide your data collection
  - Are you curious about the effect of local government policies or firms on people?
  - Are you looking to measure the effect of air pollution on health?
  - Do you want to see how people change their commute patterns over time? When there is a road closure?

# Empirical Workflow

#### Workflow workflow workflow

#### The Cunningham Empirical Workflow Conjecture

- The cause of most of your empirical coding errors is **not** due to insufficient knowledge of syntax in your chosen programming language
- The cause of most of your errors is due to a poorly designed **Empirical Workflow**
- **Empirical Workflow**: A fixed set of routines you always follow to identify the most common errors
  - Think of it as your morning routine: alarm goes off, go to wash up, make your coffee/tea, put pop tart in toaster, contemplate your existence in the universe until **ding**, eat pop tart repeat *ad infinitum*
- Finding weird errors is a different task; empirical workflows catch typical and common errors
- Empirical workflows follow a checklist

#### Why do we use checklists?

- My weekly Tuesday routine involves driving from Melrose, MA to Lewiston, ME by class start
- I need to make sure I have everything I need for the next two days (minimum)
- I have a checklist of things I need to do before I leave the house

□ Wake up by 7am, ideally 6am

□ Start coffee

 $\square$  Boil water for tea

□ Prep breakfast

□ Bring my spouse coffee in bed (bonus item)

□ Pour tea into travel mug

□ Make sure laptop, charger, lunch, and phone are in bag

□ Eat breakfast

🗆 etc

#### To remember the obvious stuff

- When I stop to think, I know I need to do everything on my checklists
- But then I forget when I move onto the next task
- Programming is the same, except you have an **empirical checklist**:
- The empirical checklist:
  - Covers the intermediate step between "getting the data" and "analyzing the data"
  - It largely focuses on ensuring data quality for the most common, easy to identify problems
  - It'll make you a better coauthor

#### Simple data checklist items

- Simple, yet non-negotiable, programming commands and exercises to check for data errors
- 1. Read the documentation
- 2. Download the data and documentation
- 3. Open the data
- 4. Look at the data ("Real eyes realize real lies"<sup>1</sup>)
- 5. Look at summaries and frequency tables of variables
- 6. Plot histograms of key variables
- 7. Visualize by key groups
- 8. Check sum stats by key groups
- 9. Check if the data are the right "size"
  - There are many more potential checklist: you'll develop your own with experience
  - First, above all else, read any documentation associated with the file
    - Codebooks, READMEs, etc.: not riveting, but they clarify little things and save hours of work

#### Downloading data

#### 2. Downloading data

- Often it is good practice to have a script that downloads the data for you
- This way, you can rerun the script and get the latest version of the data
- Of course, there is a tradeoff to this -- your results may not replicate exactly with newer data
- But it's a good practice to get into for reproducibility
- Also, if you have to downloading thousands of files manually, you'll go insane
- Automate it

#### R makes it easy to download

- The R function to download is download.file()
- Here is an example that downloads my copy of the Bertrand and Mullainathan (2004) data off of GitHub

```
download.file(
```

```
url='https://raw.githubusercontent.com/big-data-and-economics/big-data-class-materials/main/lectures,
destfile='data/lakisha_aer.zip')
```

• Several R packages will read files right off of the internet

```
readr::read_csv(
```

file='https://raw.githubusercontent.com/big-data-and-economics/big-data-class-materials/main/lectures

```
## # A tibble: 4,873 × 5
```

##		firstname			gender			race			call			ofjobs
##		<chr></chr>			<chr></chr>			<chr:< td=""><td>&gt;</td><td></td><td><chr:< td=""><td>&gt;</td><td></td><td><dbl></dbl></td></chr:<></td></chr:<>	>		<chr:< td=""><td>&gt;</td><td></td><td><dbl></dbl></td></chr:<>	>		<dbl></dbl>
##	1	firstname	row	1	gender	row	1	race	row	1	call	row	1	NA
##	2	firstname	row	2	gender	row	2	race	row	2	call	row	2	NA
##	3	firstname	row	3	gender	row	3	race	row	3	call	row	3	NA
##	4	Allison			female			cauc			no			2
##	5	Kristen			female			cauc			no			3
##	6	Lakisha			female			afam			no			1
##	7	Latonya			female			afam			no			4
##	8	Carrie			female			cauc			no			3
##	9	Jay			male			cauc			no			2
<del>11 11</del>	10	1;11			female			cauc			no			2

#### Reading file formats

### 3. Reading file extensions

- A file extension is the part of the file name after the period .dta, .csv, .tab, etc.
- Often, if you download a file, you will immediately understand what type of a file it is by its extension
- File extensions in and of themselves don't serve any particular purpose other than convenience
- File extensions were created so that humans could keep track of which files on their workspace are scripts, which are binaries, etc.

#### Why is the file format important?

- File formats matter because they may need to match the coding tools you're using
- If you use the wrong file format, it may cause your computations to run slower than otherwise
- To the extent that the tools you're using require a specific file format, then using the correct format is essential

#### Open-format file extensions

The following file extensions are not tied to a specific software program

• In this sense they are "raw" and can be viewed in any sort of text editor

File extension	Description
CSV	Comma separated values; data is in tabular form with column breaks marked by commas
TSV	Tab separated values; data is in tabular form with column breaks marked by tabs
DAT	Tab-delimited tabular data (ASCII file)
TXT	Plain text; not organized in any specific manner (though usually columns are delimited with tabs or commas)
TEX	LaTeX; markup-style typesetting system used in scientific writing
XML	eXtensible Markup Language; data is in text form with tags marking different fields
HTML	HyperText Markup Language; similar to XML; used for almost every webpage you view
YAML	YAML Ain't Markup Language: human readable version of XML

- Here's a more **complete list** of almost every file extension (note: missed Stata's .do and .dta formats).
- Another great discussion about file formats is here on stackexchange

### Proprietary file extensions

The following file extensions typically require additional software to read, edit, or convert to another format

File extension	Description
DB	A common file extension for tabular data for SQLite
SQLITE	Another common file extension for tabular data for SQLite
XLS, XLSX	Tab-delimited tabular data for Microsoft Excel
RDA, RDATA	Tabular file format for R
MAT	for Matlab
SAS7BDAT	for SAS
SAV	for SPSS
DTA	for Stata

#### Other file types that aren't data

- There are many file types that don't correspond to readable data. For example, script files (e.g. .R, .py, .jl, .sql, .do, .cpp, .f90, ...) are text files with convenient extensions to help the user remember which programming language the code is in
- As a rule of thumb, if you don't recognize the extension of a file, it's best to inspect the file in a text editor (though pay attention to the size of the file as this can also help you discern whether it's code or data)

# Tips for opening files with r

- If you're working with tabular data, you can use the read\_csv() function from the readr (tidyverse) package or fread from data.table
- If you're working with a proprietary file format, you can use the read\_\*() functions from the haven package
- If you're reading in any table format, read\_table() might work!
- If you're working with a JSON file, you can use the **jsonlite** package
- When in doubt, Google/ChatGPT "How do I open file .XXX in R?"
  - I bet you someone has already needed to solve this problem

```
df_csv ← read_csv('data/lakisha_aer.csv')
df_fread ← data.table::fread('data/lakisha_aer.tab')
df_stata ← haven::read_dta('data/lakisha_aer.dta')
df_xlsx ← readxl::read_xlsx('data/lakisha_aer.xlsx')
```

### Help! This file froze my computer!

- Sometimes we'll be reading quite large files
  - These can be too big to fit in memory
- Consult the codebook for the necessary columns
- Or read in a single row to see the column names:

```
df ← read_csv('data/lakisha_aer.csv',n_max=1)
print(names(df))
```

## [1] "firstname" "gender" "race" "call" "ofjobs"

#### Help! This file froze my computer!

Once you know your columns, read those in:

```
read_csv('data/lakisha_aer.csv',
    col_select=c('firstname', 'race', 'gender','call','ofjobs'))
```

## # A tibble: 4,873 × 5

##		firstname			race			gender			call			ofjobs
##		<chr></chr>			<chr></chr>	>		<chr></chr>			<chr></chr>	>		<dbl></dbl>
##	1	firstname	row	1	race	row	1	gender	row	1	call	row	1	NA
##	2	firstname	row	2	race	row	2	gender	row	2	call	row	2	NA
##	3	firstname	row	3	race	row	3	gender	row	3	call	row	3	NA
##	4	Allison			cauc			female			no			2
##	5	Kristen			cauc			female			no			3
##	6	Lakisha			afam			female			no			1
##	7	Latonya			afam			female			no			4
##	8	Carrie			cauc			female			no			3
##	9	Jay			cauc			male			no			2
##	10	Jill			cauc			female			no			2

## # i 4,863 more rows

#### Archiving & file compression

# Archiving & file compression

Because data can be big and bulky, it is often easier to store and share the data in compressed form

File extension	Description
ZIP	The most common format for file compression
Z	Alternative to ZIP; uses a slightly different format for compression
7Z	Alternative to ZIP; uses 7-Zip software for compression
GZ	Another alternative to ZIP (primarily used in Linux systems), using what's called <code>gzip</code>
TAR	So-called "tarball" which is a way to collect many files into one archive file. TAR stands for "Tape ARchive"
TAR.GZ; TGZ	A compressed version of a tarball (compression via gzip)
TAR.BZ2; .TB2; .TBZ; .TBZ2	Compressed tarball (via bzip2)





### Unzipping in two-steps

```
# unzip('data/lakisha_aer.zip',exdir='data') # already done
read_csv('data/lakisha_aer.csv') %>% head(5)
```

#### ## # A tibble: 5 × 5

##		firstname			gender			race			call			ofjobs
##		<chr></chr>			<chr></chr>			<chr:< td=""><td>&gt;</td><td></td><td><chr></chr></td><td>&gt;</td><td></td><td><dbl></dbl></td></chr:<>	>		<chr></chr>	>		<dbl></dbl>
##	1	firstname	row	1	gender	row	1	race	row	1	call	row	1	NA
##	2	firstname	row	2	gender	row	2	race	row	2	call	row	2	NA
##	3	firstname	row	3	gender	row	3	race	row	3	call	row	3	NA
##	4	Allison			female			cauc			no			2
##	5	Kristen			female			cauc			no			3

#### Make that a one-step process?

• Hairier syntax, but yes<sup>1</sup>

```
read_csv(unz('data/lakisha_aer.zip','lakisha_aer.csv')) %>% # Unzip the file
head(5) # pipe to a read csv
```

```
## # A tibble: 5 × 5
```

##		firstname			gender			race			call			ofjobs
##		<chr></chr>			<chr></chr>			<chr:< td=""><td>&gt;</td><td></td><td><chr></chr></td><td>&gt;</td><td></td><td><dbl></dbl></td></chr:<>	>		<chr></chr>	>		<dbl></dbl>
##	1	firstname	row	1	gender	row	1	race	row	1	call	row	1	NA
##	2	firstname	row	2	gender	row	2	race	row	2	call	row	2	NA
##	3	firstname	row	3	gender	row	3	race	row	3	call	row	3	NA
## Cl	4 1e0	Allison Kappendix f	For wł	nat	female	s with	ו" ו	cauc	files	wł	no Jen vo	u atte	-m	2 pt this.
##`	5	Kristen	0		female		. ,	cauc			no	0. 0.000		3

#### How do I zip a file?

- You can zip a file using the zip command in the R terminal
- It allows you to zip one or many files

#### 4. Look at the data

• Open the raw data and look at it:

```
resumes ← read_csv('data/lakisha_aer.csv',
    show_col_types= FALSE) # Don't tell me the column types
head(resumes,10)
```

```
## # A tibble: 10 × 5
```

##		firstname			gender			race			call			ofjobs
##		<chr></chr>			<chr></chr>			<chr:< td=""><td>&gt;</td><td></td><td><chr:< td=""><td>&gt;</td><td></td><td><dbl></dbl></td></chr:<></td></chr:<>	>		<chr:< td=""><td>&gt;</td><td></td><td><dbl></dbl></td></chr:<>	>		<dbl></dbl>
##	1	firstname	row	1	gender	row	1	race	row	1	call	row	1	NA
##	2	firstname	row	2	gender	row	2	race	row	2	call	row	2	NA
##	3	firstname	row	3	gender	row	3	race	row	3	call	row	3	NA
##	4	Allison			female			cauc			no			2
##	5	Kristen			female			cauc			no			3
##	6	Lakisha			female			afam			no			1
##	7	Latonya			female			afam			no			4
##	8	Carrie			female			cauc			no			3
##	9	Jay			male			cauc			no			2
##	10	Jill			female			cauc			no			2

### Drop junk rows

• Oh weird, the first few rows are junk, let's skip them and give more informative names

```
resumes ← read_csv('data/lakisha_aer.csv',
    skip=4, # skip some rows
    col_names=c('firstname','gender','race','call','ofjobs'), # Column names
    show_col_types = FALSE) # Don't tell me the column types
head(resumes)
```

```
## # A tibble: 6 × 5
   firstname gender race call ofjobs
###
    <chr> <chr> <chr> <chr> <chr> <chr> <dbl>
###
## 1 Allison female cauc no
                                      2
## 2 Kristen female cauc no
                                      3
## 3 Lakisha female afam no
                                     1
## 4 Latonya female afam no
                                      4
## 5 Carrie
            female cauc no
                                      3
## 6 Jay
              male cauc no
                                      2
```

#### 5. Look at summaries of variables

Do factor variables have multiple spellings?

table(resumes\$race,resumes\$gender)

##							
##		female	FEMALE	male	MALE	MLE	WOMN
##	afam	1855	15	548	0	0	14
##	BLACK	Θ	Θ	Θ	1	0	2
##	cauc	1761	13	541	0	2	0
##	Caucasian	73	0	32	Θ	0	13

#### 6. Visualize the raw data

• Go beyond the eyeball and graph the data

# ggplot is a great tool for visualizing data
ggplot(data=resumes,mapping=aes(x=ofjobs))+
geom\_histogram()



• Wait a minute! What's going on with the -99 values?

# 7. Visualize by group



- Oh! I bet -99 means NA or missing
- This only occurs for fake black name applicant profiles
- That could be bad news for an RCT...

### 8. Visualize summaries by group

ggplot(data=resumes,aes(y=ofjobs,x=race)) +
geom\_bar(stat='summary',fun='mean')



• Yep, the job counts differ meaningfully by race -- uh oh.

#### 9. Are the data the right-size?

- Check if the data are the right-size
- If you have a panel dataset is 50 states over 20 years, check if there are 1000 observations
- If not, find out why!
  - Maybe there are 1020 because DC is (rightfully) included
  - Alternatively, data are missing for a few states in a few years and dropped entirely
- Search for outliers or oddities and work out possible explanations using:
  - Codebooks
  - Intuition
  - Emails to the source/creator of data

#### Big Data File Types

#### General Types of Data

- When you think of data, you probably think of rows and columns, like a matrix or a spreadsheet
- But it turns out there are other ways to store data, and you should know their similarities and differences to tabular data

#### Examples JSON

A possible JSON representation describing a person (source)

```
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
  "postalCode": "10021-3100"
},
"phoneNumbers": [
    "type": "home",
    "number": "212 555-1234"
  },
    "type": "office",
    "number": "646 555-4567"
  },
    "type": "mobile",
    "number": "123 456-7890"
],
"children": [],
```

#### Examples: XML

The same example as previously, but in XML: (source)

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021/postalCode>
  </address>
  <phoneNumber>
    <type>home</type>
    <number>212 555-1234</number>
  </phoneNumber>
  <phoneNumber>
    <type>fax</type>
    <number>646 555-4567</number>
  </phoneNumber>
  <gender>
    <type>male</type>
  </gender>
</person>
```

#### Examples: YAML

The same example, but in YAML: (source)

```
firstName: John
lastName: Smith
age: 25
address:
   streetAddress: 21 2nd Street
   city: New York
   state: NY
   postalCode: '10021'
phoneNumber:
   type: home
   number: 212 555-1234
   type: fax
   number: 646 555-4567
gender:
   type: male
```

Note that the JSON code above is also valid YAML; YAML simply has an alternative syntax that makes it more human-readable

#### Read a JSON

#### library(jsonlite)

```
# Read in the JSON file
resumes_json ← read_json('data/lakisha_aer.json',
   simplifyVector = TRUE # Simplify the data to a dataframe
   )
head(resumes_json)
```

##		firs	stnam	ne	ŝ	gende	er		rad	ce		cal	.1	ofjobs
##	1	firstname	row	1	gender	row	1	race	row	1	call	row	1	NA
##	2	firstname	row	2	gender	row	2	race	row	2	call	row	2	NA
##	3	firstname	row	3	gender	row	3	race	row	3	call	row	3	NA
##	4	Al	lliso	on	1	femal	le		cau	JC		r	10	2
##	5	Kı	ciste	en	1	femal	le		cau	JC		r	סו	3
##	6	Lā	akisł	na	1	femal	le		afa	am		r	10	1

# Big Data file types

- Big Data file systems like Hadoop and Spark often use the same file types as R, SQL, Python, and Julia
- That is, CSV and TSV files are the workhorse
- Because of the nature of distributed file systems (which we will discuss in much greater detail next time), it is often the case that JSON and XML are not good choices because they can't be broken up across machines
- Note: there is a distinction between JSON files and JSON records; see the second link at the end of this document for further details

## Big Data File Types

#### Sequence

- Sequence files are dictionaries that have been optimized for Hadoop and friends
- The advantage to taking the dictionary approach is that the files can easily be coupled and decoupled

#### Avro

• Avro is an evolved version of Sequence---it contains more capability to store complex objects natively

#### Parquet

- Parquet is a format that allows Hadoop and friends to partition the data column-wise (rather than row-wise)
- Other formats in this vein are RC (Record Columnar) and ORC (Optimized Record Columnar)

### Dictionaries

# Dictionaries (a.k.a. Hash tables)

- A dictionary is a list that contains keys and values
- Each key points to one value
- While this may seem like an odd way to store data, it turns out that there are many, many applications in which this is the most efficient way to store things
- We won't get into the nitty gritty details of dictionaries, but they are the workhorse of computer science, and you should at least know what they are and how they differ from tabular data
- In fact, dictionaries are often used to store multiple arrays in one file (e.g. Matlab .mat files, R
   .RData files, etc.)

### Dictionaries (a.k.a Hash tables) in R

- Dictionraies are a little clunky in R
- You'll mainly use them as lists or vectors

```
print(phone_numbers_list)
```

```
## $Jenny
## [1] "1 (623) 867-5309"
##
## $Rejection Hotline
## [1] "1 (518) 935-4012"
##
## $Santa
## [1] "1 (951) 262-3062"
```

### Why are dictionaries useful?

- You might look at the previous example and think a vector would be a better way to store phone numbers
- The power of dictionaries is in their **lookup speed**
- Looking up an index in a dictionary takes the same amount of time no matter how long the dictionary is!
  - $\circ$  Computer scientists call this O(1) access time
- Moreover, dictionaries can index **objects**, not just scalars
- So I could have a dictionary of data frames, a dictionary of arrays, ...

Appendix

## **Useful Links**

- A beginner's guide to Hadoop storage formats
- Hadoop File Formats: It's not just CSV anymore

### What if there is only one file in the zip?

Turns out, you can read the file directly from the zip file with read\_csv():

```
read_csv(unz('data/lakisha_aer.zip','lakisha_aer.csv'),show_col_type=FALSE)
```

## # A tibble: 4,873 × 5

##		firstname			gender			race			call			ofjobs
##		<chr></chr>			<chr></chr>			<chr></chr>			<chr></chr>			<dbl></dbl>
##	1	firstname	row	1	gender	row	1	race	row	1	call	row	1	NA
##	2	firstname	row	2	gender	row	2	race	row	2	call	row	2	NA
##	3	firstname	row	3	gender	row	3	race	row	3	call	row	3	NA
##	4	Allison			female			cauc			no			2
##	5	Kristen			female			cauc			no			3
##	6	Lakisha			female			afam			no			1
##	7	Latonya			female			afam			no			4
##	8	Carrie			female			cauc			no			3
##	9	Jay			male			cauc			no			2
##	10	Jill			female			cauc			no			2

## # i 4,863 more rows

#### What is VROOM\_CONNECTION\_SIZE?

• You'll often hit an error when reading zipped files

read\_csv('data/county\_outcomes.zip',show\_col\_type=FALSE)

## Error: The size of the connection buffer (131072) was not large enough
## to fit a complete line:

## \* Increase it by setting Sys.setenv("VROOM\_CONNECTION\_SIZE")

#### What is VROOM\_CONNECTION\_SIZE?

VROOM\_CONNECTION\_SIZE is an environment variable that tells R how much data to read in at a time

- It's a way to read in large files without crashing your computer
- It basically tells R to read in a certain number of bytes at a time
- When R unzips and reads simultaneously, it needs more memory than usual while it decompresses
- Think of it like having too narrow a space to squeeze the data through
- If the data are wide, this can be a problem because of how read\_csv() works
  - It reads in the entire file and then tries to figure out the column types

#### **Two Fixes**

#### Fix 1: Increase VROOM\_CONNECTION\_SIZE

Sys.setenv("VROOM\_CONNECTION\_SIZE"=1e6) # Telling R to read in 1 million bytes at a time read\_csv(unz('data/county\_outcomes.zip','county\_outcomes.csv'),show\_col\_type=FALSE) Sys.setenv("VROOM\_CONNECTION\_SIZE"=131072) # Returning to default

#### Fix 2: Unzip then read

```
unz('data/county_outcomes.zip','county_outcomes.csv') %>% # Unzip the file
read_csv(show_col_types = FALSE) # pipe to a read csv
rm('county_outcomes.csv') # remove the file
```

I don't evaluate the code because it will make knitting take awhile, but try it yourself