Data Science for Economists

Lecture 3: Data tips

Kyle Coombs
Bates College | ECON/DCS 368

Table of contents

Prologue

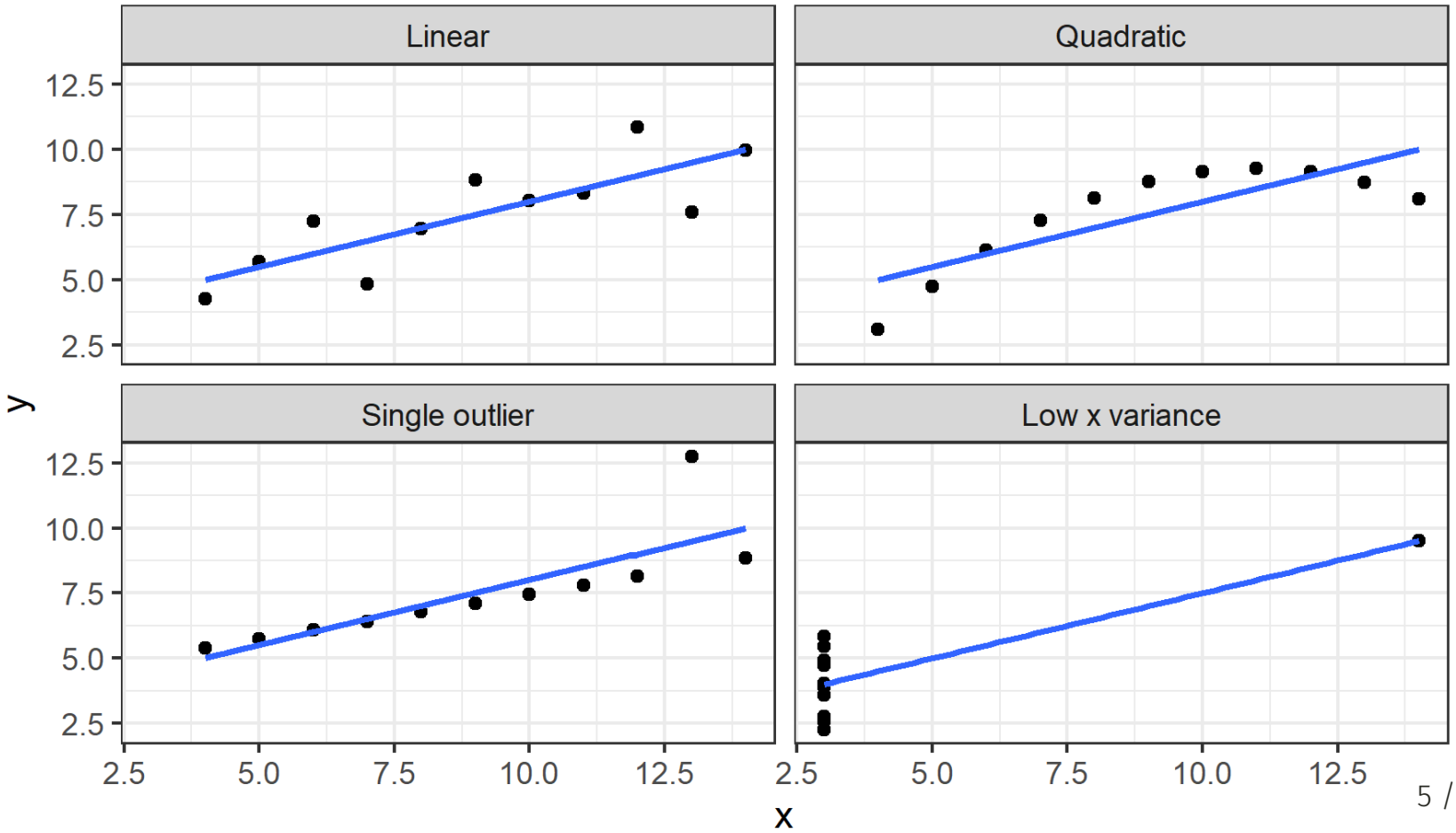
Prologue

Today we'll focus on grappling with data

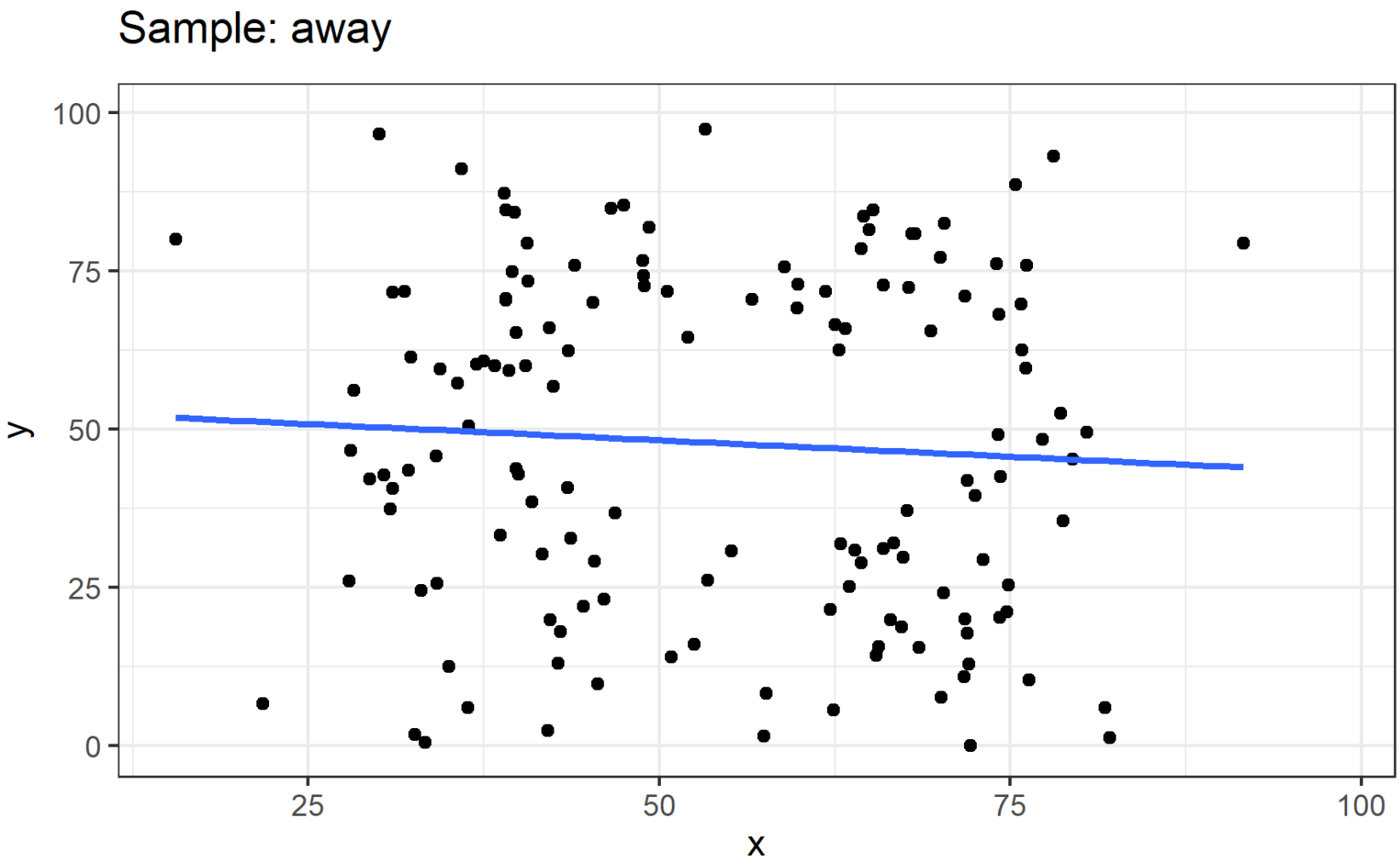
- Checklist to ensure data quality
- File formats and extensions
- Archiving & file compression
- If time:
    - Dictionaries (hash tables)
    - Big Data file types

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



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.)

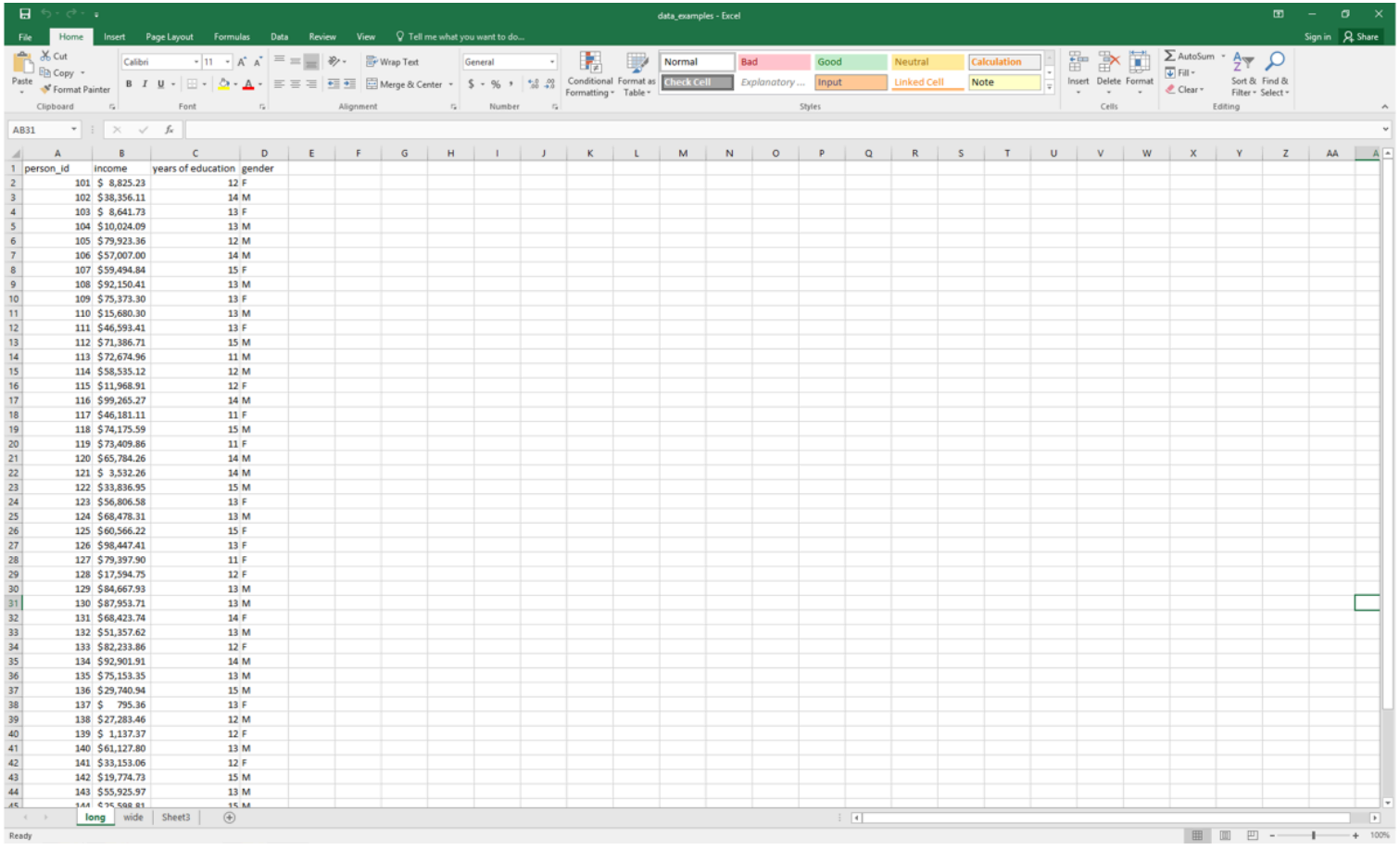
2. "Wide" data (a.k.a. "Big-K" data because $K > N$, customer data sets where each click is a variable)

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

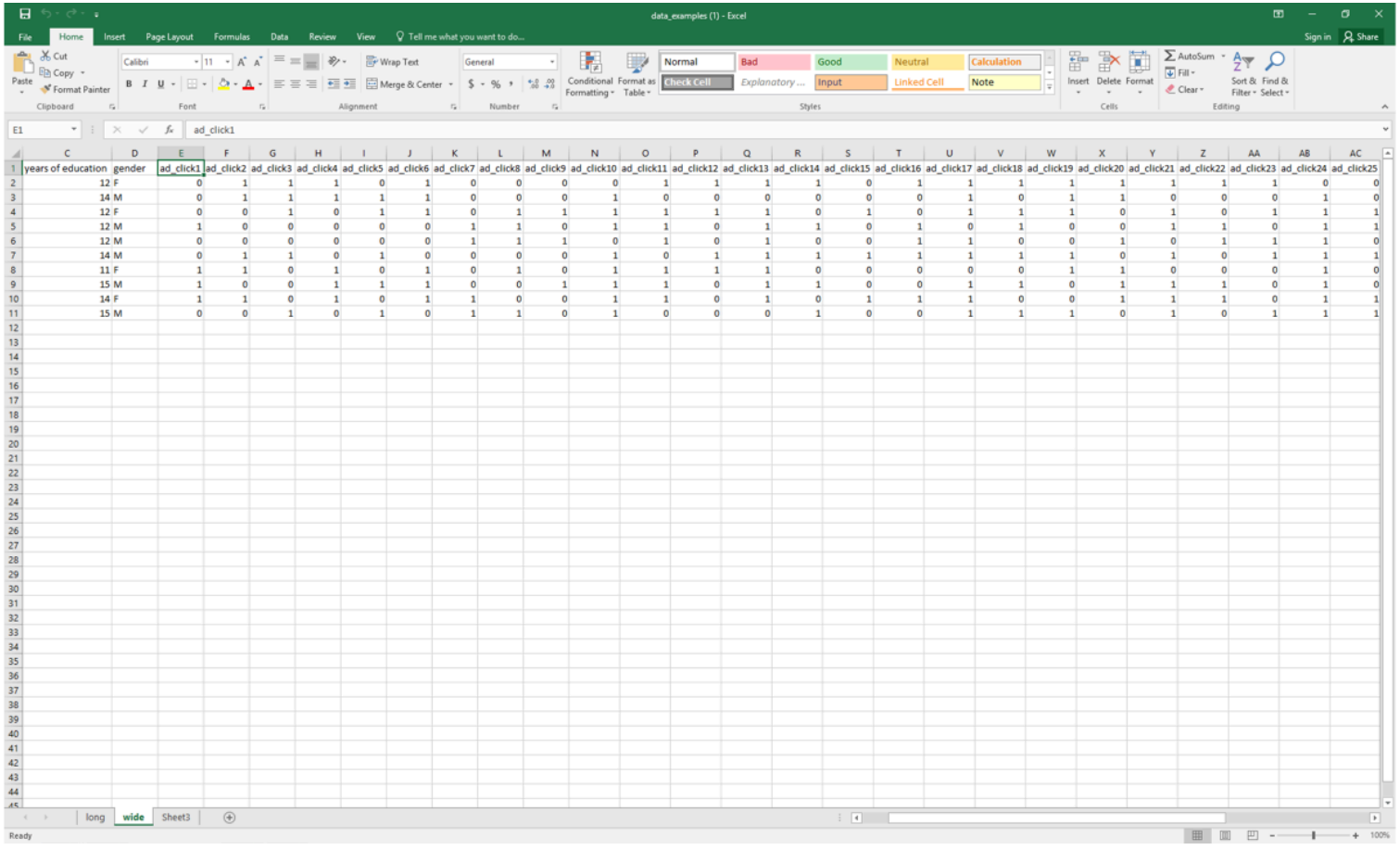
Long data



person_id	income	years of education	gender
101	$ 8,825.23	12	F
102	$38,356.11	14	M
103	$ 8,641.73	13	F
104	$10,024.09	13	M
105	$79,923.36	12	M
106	$57,007.00	14	M
107	$59,494.84	15	F
108	$92,150.41	13	M
109	$75,373.30	13	F
110	$15,680.30	13	M
111	$46,593.41	13	F
112	$71,386.71	15	M
113	$72,674.96	11	M
114	$58,535.12	12	M
115	$11,968.91	12	F
116	$99,265.27	14	M
117	$46,181.11	11	F
118	$74,175.59	15	M
119	$73,409.86	11	F
120	$65,784.26	14	M
121	$ 3,532.26	14	M
122	$33,836.95	15	M
123	$56,806.58	13	F
124	$68,478.31	13	M
125	$60,566.22	15	F
126	$98,447.41	13	F
127	$79,397.90	11	F
128	$17,594.75	12	F
129	$84,667.93	13	M
130	$87,953.71	13	M
131	$68,423.74	14	F
132	$51,357.62	13	M
133	$82,233.86	12	F
134	$92,901.91	14	M
135	$75,153.35	13	M
136	$29,740.94	15	M
137	$ 795.36	13	F
138	$27,283.46	12	M
139	$ 1,137.37	12	F
140	$61,127.80	13	M
141	$33,153.06	12	F
142	$19,774.73	15	M
143	$55,925.97	13	M

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

Wide data



years of education	gender	ad_click1	ad_click2	ad_click3	ad_click4	ad_click5	ad_click6	ad_click7	ad_click8	ad_click9	ad_click10	ad_click11	ad_click12	ad_click13	ad_click14	ad_click15	ad_click16	ad_click17	ad_click18	ad_click19	ad_click20	ad_click21	ad_click22	ad_click23	ad_click24	ad_click25
12	F	0	1	1	1	0	1	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	0	0
14	M	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0
12	F	0	0	1	0	1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1
12	M	1	0	0	0	0	0	1	1	0	1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1
12	M	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	1	0
14	M	0	1	1	0	1	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1
11	F	1	1	0	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1	0
15	M	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0	0	1	1	0	1	1	1	0	1	0
14	F	1	1	0	1	0	1	1	0	0	1	1	0	1	0	1	1	1	0	0	1	1	1	0	1	1
15	M	0	0	1	0	1	0	1	1	0	1	0	0	0	1	0	0	1	1	1	0	1	0	1	1	1

- 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>
<tbody><tr>
<th scope="col" width="12%">Event
</th>
<th class="unsortable" width="5%">Record
</th>
<th scope="col" width="10%">Athlete(s)
</th>
<th scope="col" width="15%">Nation
</th>
<th scope="col" width="10%">Games
</th>
<th scope="col" width="5%">Date
</th>
<th scope="col" class="unsortable" width="3%">Ref(s)
</th></tr>
<tr>
<th scope="row"><span data-sort-value="00100 !"><a href="/wiki/100_metres" title="100 metres">100
</th>
<td align="right">9.63 
</td>
<td><span data-sort-value="Bolt, Usain"><span class="vcard"><span class="fn"><a href="/wiki/Usain_Bolt"
</td>
<td><span class="mw-image-border" typeof="mw:File"><span><img alt="" src="//upload.wikimedia.org/wikipe
</td>
<td><span data-sort-value="2012 !"><a href="/wiki/Athletics_at_the_2012_Summer_Olympics_%E2%80%93_
</td>
```

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

The data you need depend on the

- 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?

`ggplot` wants long data

The ggplot2 aesthetics expect "long" data. This is distinct from the "wide" format, where each series has its own column.

`ggplot` wants long data

The ggplot2 aesthetics expect "long" data. This is distinct from the "wide" format, where each series has its own column.

Long will work

```
## # A tibble: 6 × 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Wide would not work

```
## # A tibble: 142 × 26
##    country     continent lifeExp_1952 lifeExp_1957 lifeExp_1962 lifeExp_1967
##    <fct>       <fct>            <dbl>        <dbl>        <dbl>        <dbl>
##  1 Afghanistan Asia              28.8         30.3         32.0         34.0
##  2 Albania     Europe            55.2         59.3         64.8         66.2
##  3 Algeria     Africa            43.1         45.7         48.3         51.4
##  4 Angola      Africa            30.0         32.0         34           36.0
##  5 Argentina   Americas          62.5         64.4         65.1         65.6
##  6 Australia   Oceania           69.1         70.3         70.9         71.1
##  7 Austria     Europe            66.8         67.5         69.5         70.1
```

Empirical 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

☐ 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"[1])
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 items: you'll develop your own with experience
- First, above all else, read any documentation associated with the file
    - Codebooks, READMEs, etc. -- check out `data/README.txt.` for this dataset
    - They're not riveting, but they clarify tons of small things

[1] Attributed to Ray Charles, Woody Guthrie, Tupac Shakur, Machine Head, and others

Downloading data

1. 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>       <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
```

Reading file formats

2. 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'))
```

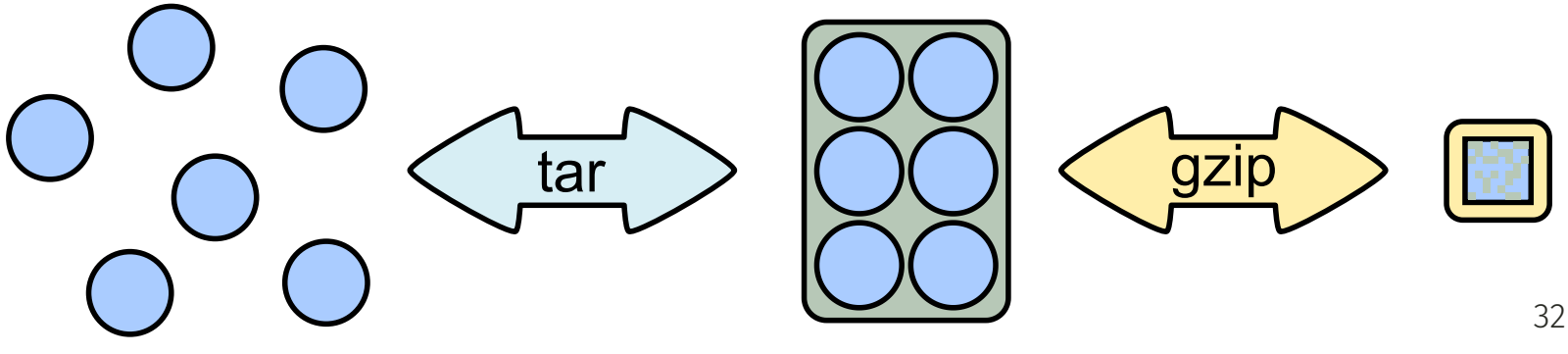
```
## # A tibble: 4,873 × 5
##    firstname       race       gender       call       ofjobs
##    <chr>           <chr>      <chr>        <chr>       <dbl>
##  1 firstname row 1 race row 1 gender row 1 call row 1     NA
##  2 firstname row 2 race row 2 gender row 2 call row 2     NA
##  3 firstname 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 `gzip`
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`)



Can I just read a zip directly in?

- Yes, but it's a little more complicated
- And you can may still want to read in a few rows or columns like before

```
read_csv(unz('data/lakisha_aer.zip','lakisha_aer.csv'))  # Unzip the file
```

```
## # A tibble: 4,873 × 5
##    firstname       gender       race       call       ofjobs
##    <chr>           <chr>        <chr>      <chr>       <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
```

```
  head(5) # pipe to a read csv
```

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

Check appendix for what happens with "bigger" files when you attempt this.

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

```
zip(zipfile='data/lakisha_aer_one.zip',
  files='lakisha_aer.csv',
  exdir='data') # extract to a directory

zip(zipfile='data/lakisha_aer.zip',
  files=c('lakisha_aer.rds','lakisha_aer.csv',
    'lakisha_aer.tsv','lakisha_aer.xlsx',
    'lakisha_aer.dta','lakisha_aer.json'))
```

2. 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>       <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>  <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
```

3. Look at summaries of variables

Do factor variables have multiple spellings?

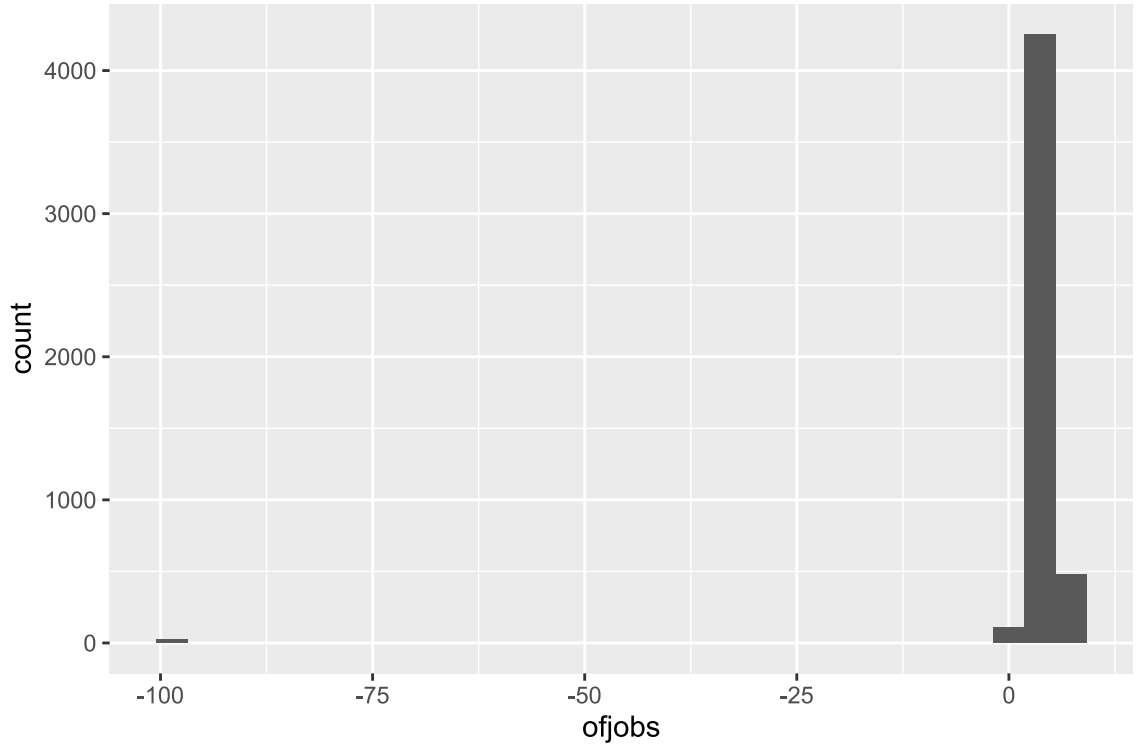
```
table(resumes$race,resumes$gender)
```

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

4. Visualize the raw data

- Go beyond the eyeball and graph the data

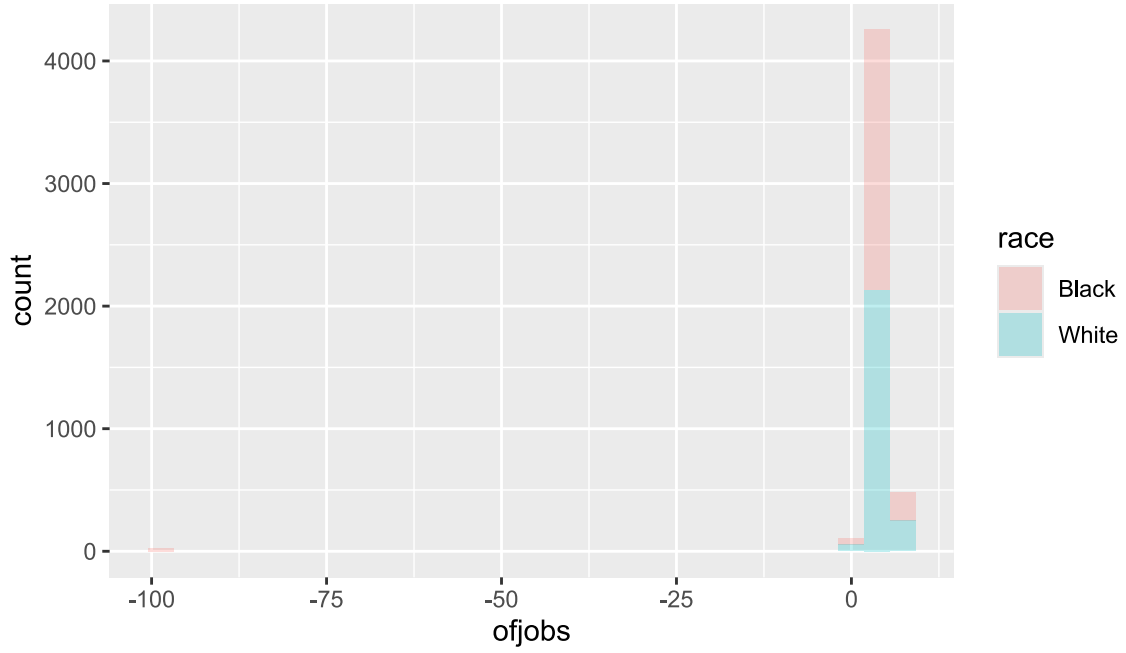
```
# ggplot is a great tool for visualizing data
ggplot(data=resumes,mapping=aes(x=ofjobs))+
  geom_histogram()
```



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

5. Visualize by group

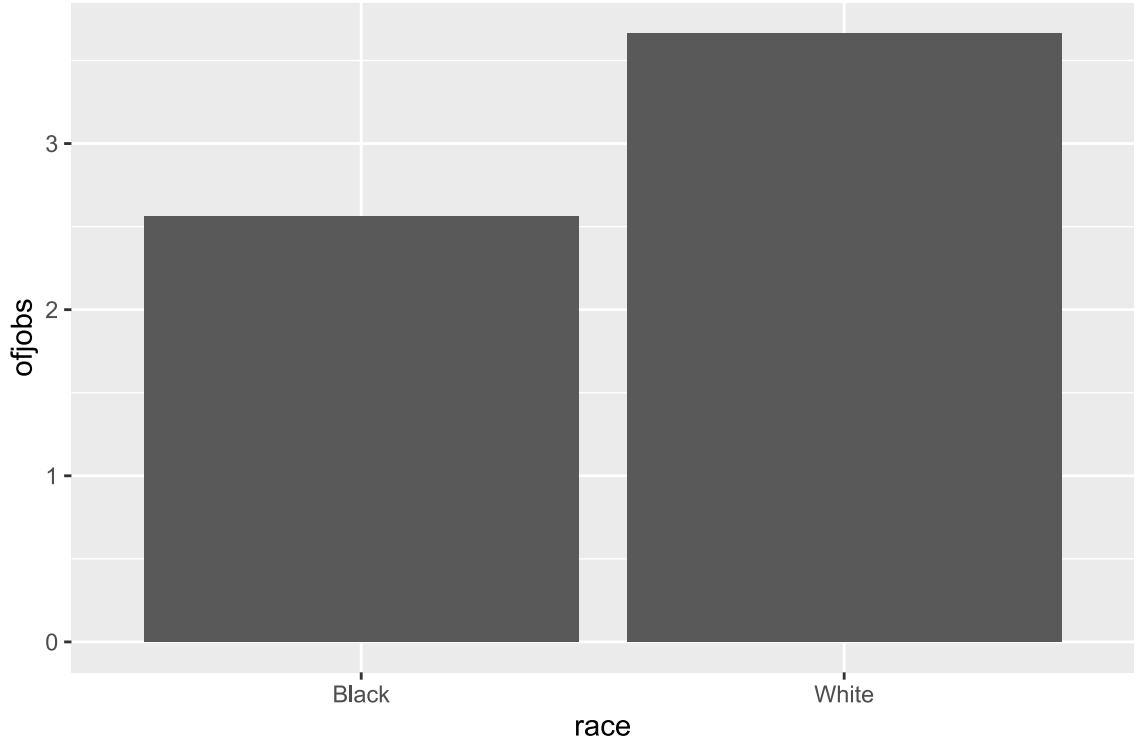
```
resumes ←  mutate(resumes,race=ifelse(race=='cauc' | race=='Caucasian','White',
    ifelse(race=="BLACK" | race=="afam", "Black",race))) # change the data up
ggplot(data=resumes,aes(x=ofjobs,fill=race))+
    geom_histogram(alpha=0.25) # alpha makes bars see through!
```



- 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...

6. 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.

7. 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)

```xml
<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)

```yaml
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

```r
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)
```

```
##         firstname       gender       race       call 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         Allison       female       cauc         no      2
## 5         Kristen       female       cauc         no      3
## 6         Lakisha       female       afam         no      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

```
phone_numbers_list ← list('Jenny'='1 (623) 867-5309',
  'Rejection Hotline'='1 (518) 935-4012',
  'Santa'='1 (951) 262-3062')

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!
    - 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>       <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