Big Data and Economics

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

Housekeeping

- If you notice something wrong with a problem set or lecture, please either:

1. Raise it as an issue in the relevant repository
2. Pull request the change you think is needed

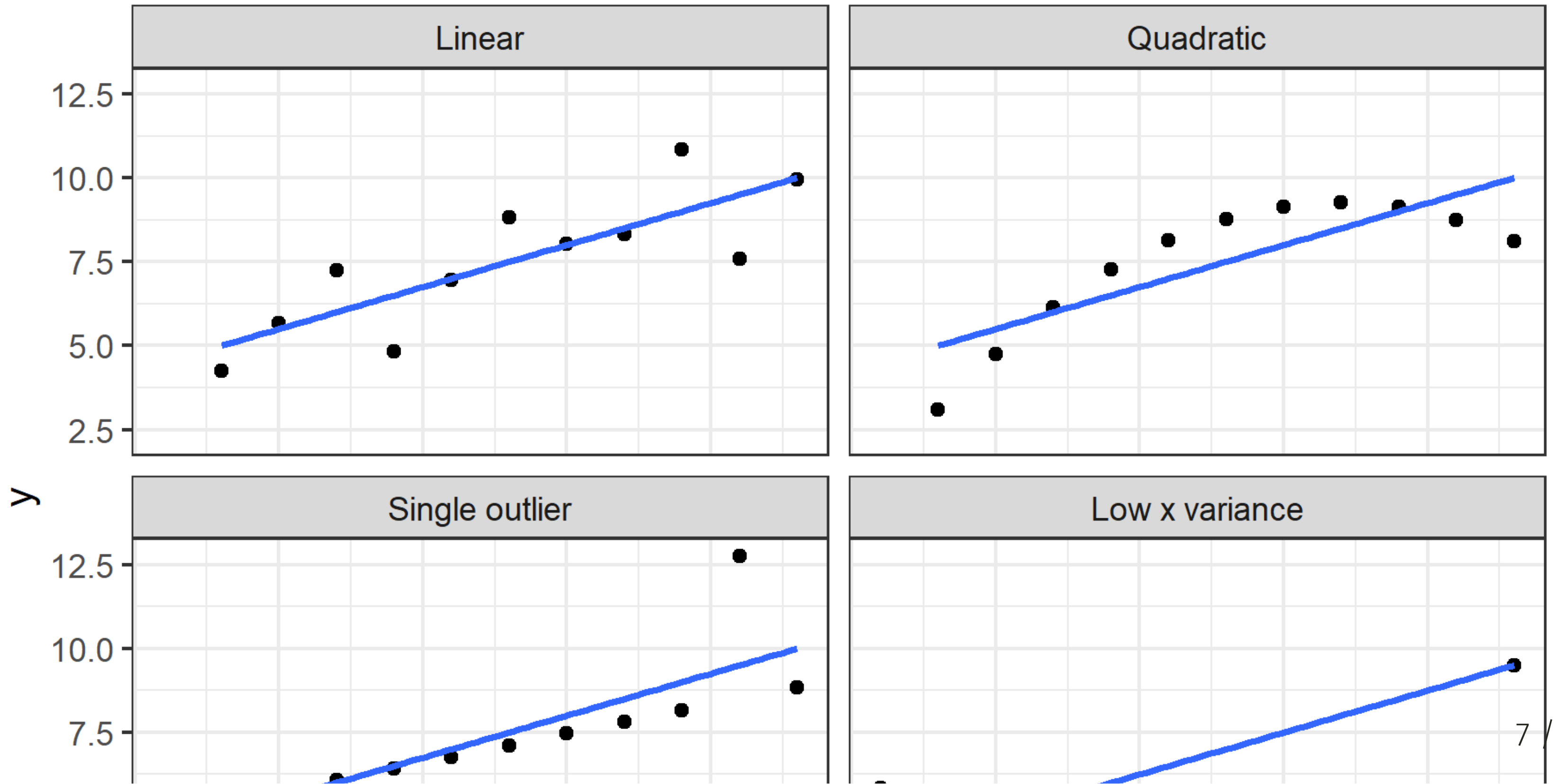
- Typo corrections, documentation improvement, etc. earn you extra credit (up to 10 points per problem set)
- I will ask you to pull request fixes to receive the point
- How do I pull request? It is in the git lecture, but I provide instructions on the class materials page
- I have to approve any fixes, but I will aim to do so within 24 hours

Student Presentation: GitHub Desktop

Why do we need to do this?

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

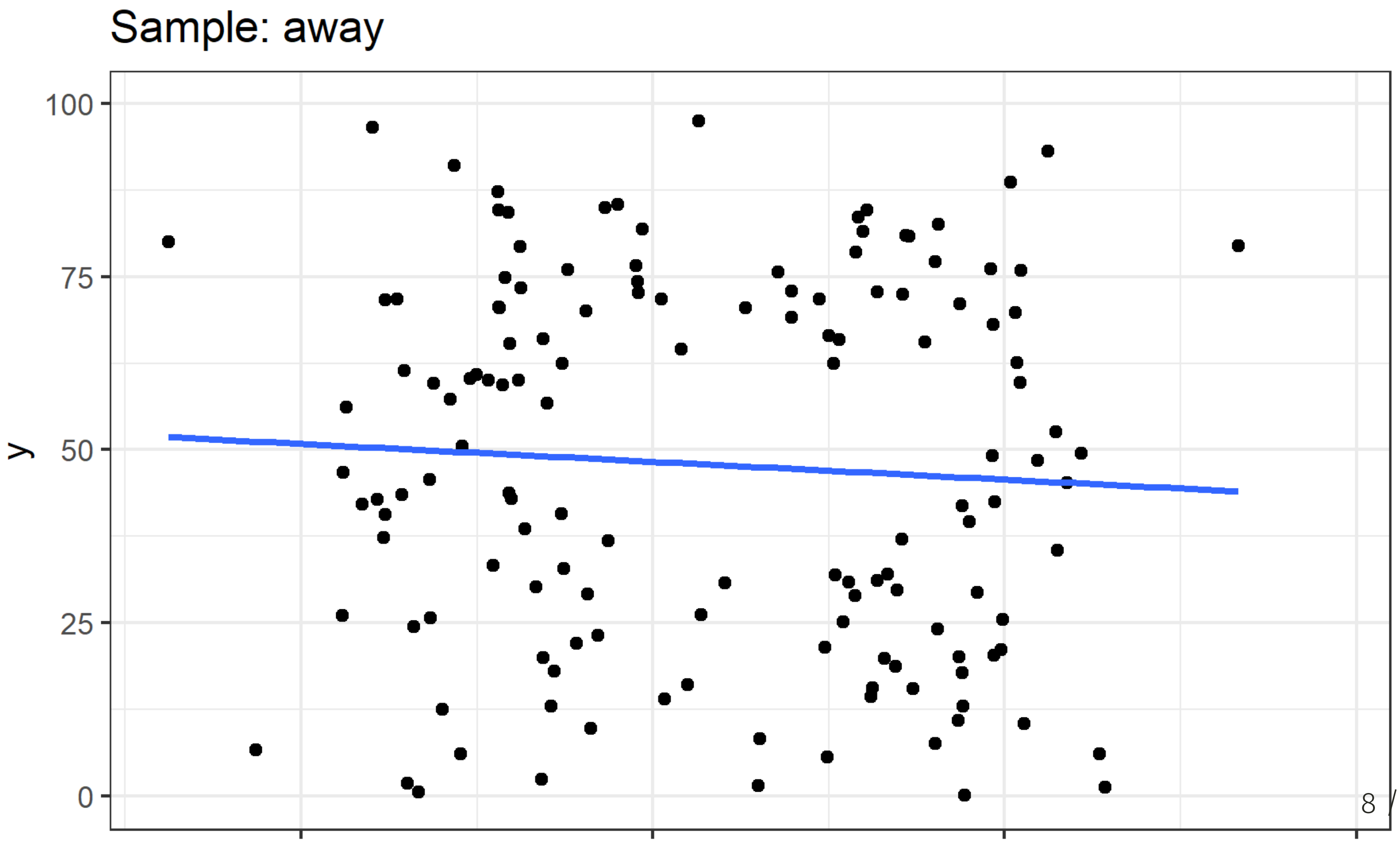
```
## geom_smooth() using formula = 'y ~ x'
```



DataSarus Dozen

- A more ambitious example is the Datasaurus Dozen dataset.

```
## geom_smooth() using formula = 'y ~ x'
```



Not every outlier is bad

- Sometimes bizarre looking outliers in data are real!
- But sometimes they're nonsense or an unusually coded "NA" value
- Today we're focusing on how to catch the latter so they don't cause you problems later
- Imagine writing an entire paper only to discover your dataset looks like a T-rex

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?

- I got engaged in July and am planning a wedding in Princeton for next July
- I also moved to New England in August and am still unpacking (in ME and MA)
- I am teaching two upper-level electives
- I am trying to submit several papers to conferences/journals this year
- Each of these gets a checklist:

- **Wedding:**

  ☐ Pick invitation design

  ☐ Send invitations
- **Unpacking:**

  ☐ Put books on shelves

  ☐ Buy dresser
- **AEJ: Policy submission**

  ☐ Write 5-page submission report

  ☐ Submit
- **ECON 368**

  ☐ Write Problem Set 1 solutions

  ☐ Rewrite data tips
- **Senior Thesis:**

  ☐ Create FAQs

  ☐ Develop data guidance
- **Census shocks and racial integration paper**

  ☐ Replicate census shocks results

  ☐ Acquire data on employment, racial wage gap

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

Play along at home

- Everyone fork the class materials repository
- Clone your fork of the class materials repository to your class folder
- Double-click the file `big-data-class-materials.Rproj` to open the project in RStudio at the correct working directory
- Type `setwd(lectures/03-data-tips/)` to make sure you are in the same directory as this `.Rmd`
- Package installation: Install the following functions using `install.packages()` and `library()`
  - Note: once you install a package, you don't need to do it again, just use `library()`

```r
install.packages(c('tidyverse','arrow','datasauRus','gganimate'))
library(tidyverse,arrow,datasauRus,gganimate)
```

- We're going to explore a mangled version of the dataset from the Bertrand and Mullainathan (2004) resume audit study
- Suspend some disbelief, please as I mangled the data to make it more interesting

Simple data checklist items

- Simple, yet non-negotiable, programming commands and exercises to check for data errors
- Let's work through a few using a messy dataset of blood pressure measurements with some demographic information
- First, above all else, read any documentation associated with the file
  - Codebooks, READMEs, etc.
  - They're not riveting, but they clarify tons of small things
  - Your first problem set doesn't come with one -- you're gonna build it!
- Open the raw data and look at it: "Real eyes realize real lies"[1]

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

```
## # A tibble: 6 × 7
##   id        ofjobs firstname        sex        race         call lmedhhinc
##   <chr>      <dbl> <chr>            <chr>      <chr>       <dbl>     <dbl>
## 1 id row 1       2 firstname row 3 sex row 4 race row 5      0      9.53
## 2 id row 1       3 firstname row 3 sex row 4 race row 5      0     10.4
## 3 id row 1       1 firstname row 3 sex row 4 race row 5      0     10.5
## 4 b              2 Allison          FEMALE     Caucasian       0      9.53
## 5 b              3 Kristen          FEMALE     Caucasian       0     10.4
## 6 b              1 Lakisha          FEMALE     BLACK           0     10.5
```

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

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

Look at summaries of variables

Do factor variables have multiple spellings?

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

```
##
##                 f FEMALE    m MALE  MLE WOMAN WOMN
##   b          1140    131  471   18    0   325    0
##   BLACK         0    290    0   60    0     0    0
##   Caucasian     0    193    0   55    1     0    1
##   w          1113    220  488   31    0   333    0
```

The `skimr` package is great!

```
skimr::skim(resumes)
```

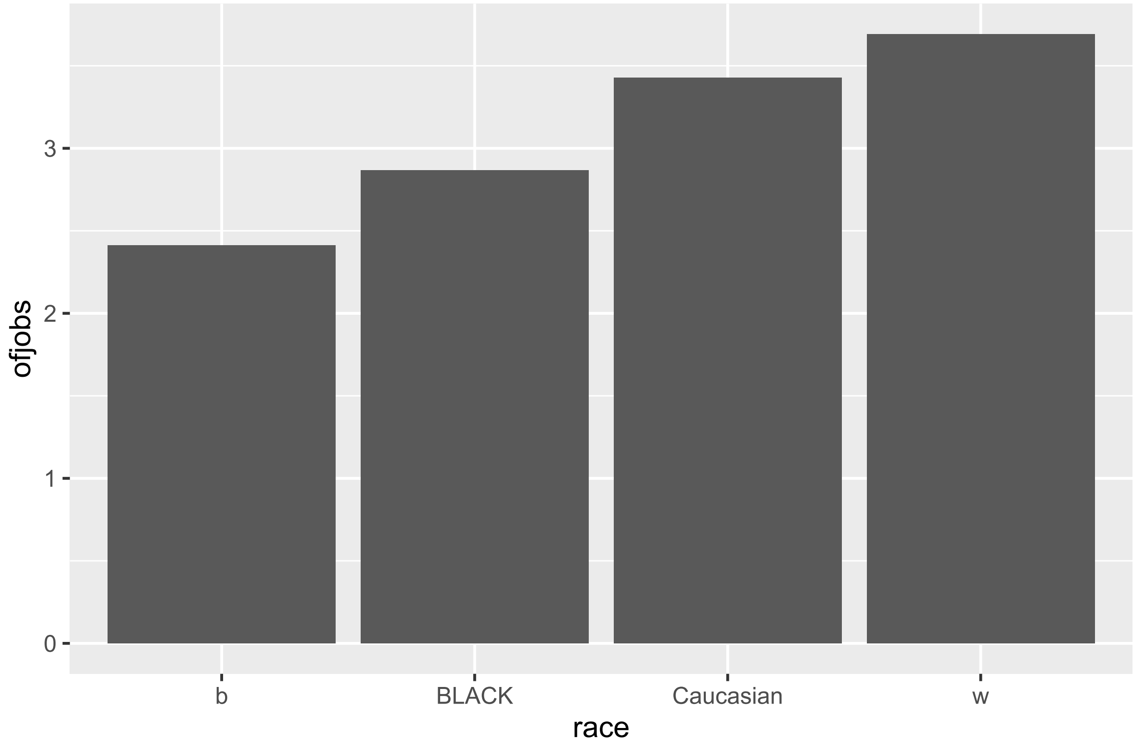
Table: Data summary

Name	resumes
Number of rows	4870
Number of columns	7

- What's 99?

Visualize key facets of the data

```r
# ggplot is a great tool for visualizing data
ggplot(data=resumes,aes(y=ofjobs,x=race)) +
  geom_bar(stat='summary',fun='mean')
```



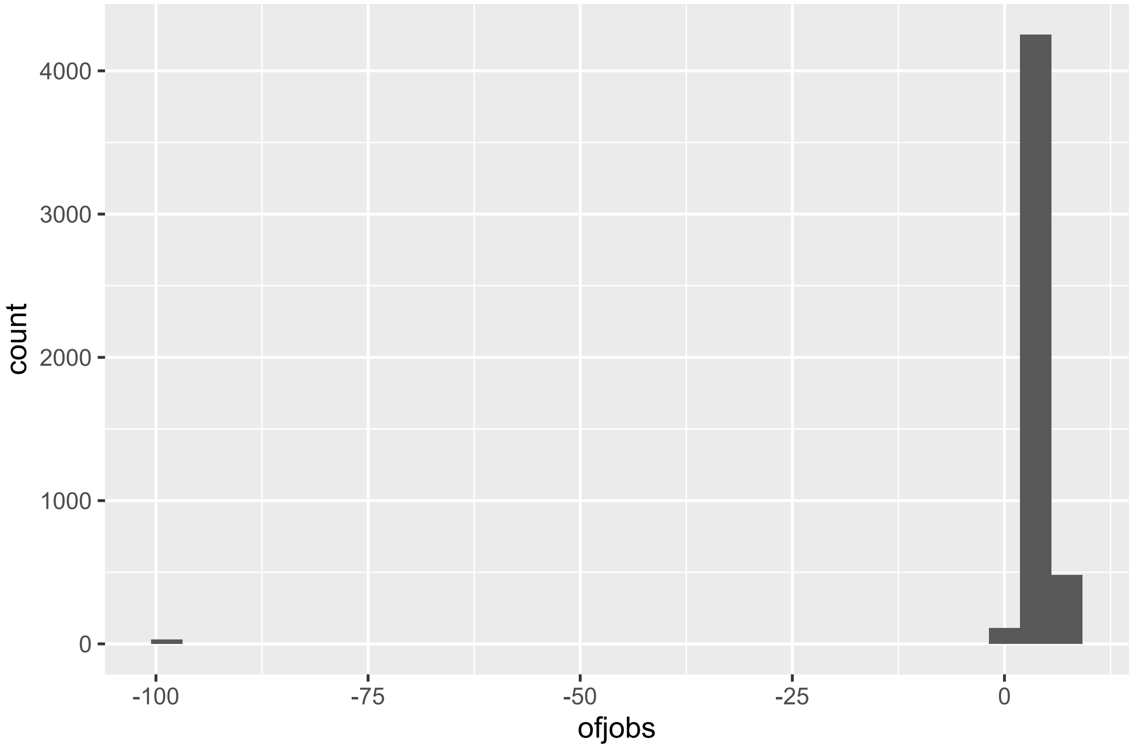
- Wait, this is an RCT, why are the previous job counts of black/white applicants different?

Visualize the raw data

- Go beyond the eyeball and graph the data

```
ggplot(data=resumes,aes(x=ofjobs))+
  geom_histogram()
```

```
## stat_bin() using bins = 30. Pick better value with binwidth.
```

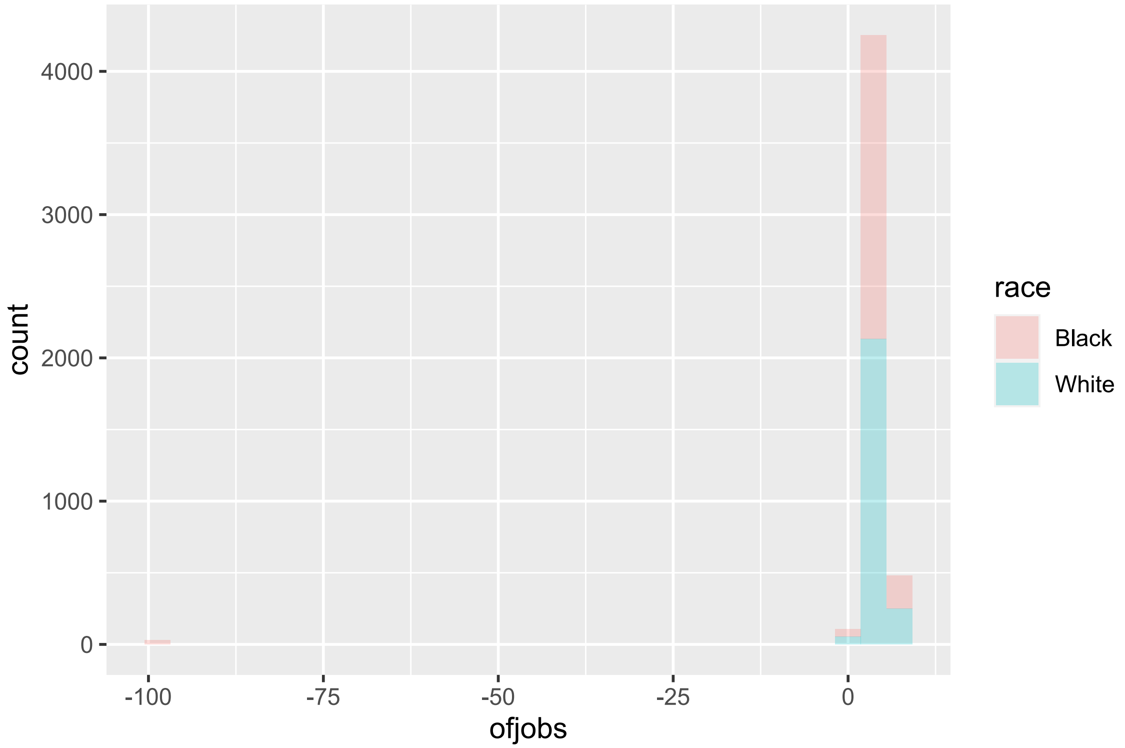


- Wait a minute! What's going on with the -99 values?

Visualize by group

```r
# Get the first three rows of the data frame (or as many rows as needed)
#Make a density of the heart rate on visit 1:
resumes %>% # pipes are a greate tool, we'll cover them in a bit
  mutate(race=ifelse(race=='w' | race=='Caucasian','White',
    ifelse(race=="BLACK" | race=="b", "Black",race))) %>% # change the data up
  ggplot(aes(x=ofjobs,fill=race))+
    geom_histogram(alpha=0.25) # alpha makes bars see through!
```

```
## stat_bin() using bins = 30. Pick better value with binwidth.
```



Other tricks:

- 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
- Search for outliers or oddities and work out possible explanations using:
  - Codebooks
  - Intuition
  - Emails to the source/creator of data

Wait, how does ggplot work?

See slides from Lecture 1 for a full explanations

- Basically, it is a useful data visualization tool that allows you to specify the data and the type of plot you want to make
- **Pro tip:** It works best when your are organized "long" instead of wide
- If you'd like to present on ggplot in class this month (instead of the topic you signed up for), let me know!

File formats

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

Common file extensions when working

- In the following table, I list some of the most common file extensions
- For a more complete list of almost every file extension imaginable (note: they missed Stata's `.do` and `.dta` formats), see here.
- Another great discussion about file formats is here on stackexchange

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)

Examples of CSV, TSV, XML, YAML, and

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"
    },
    {
```

Examples of CSV, TSV, XML, YAML, and

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>
```

Examples of CSV, TSV, XML, YAML, and

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
```

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

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

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('https://www2.census.gov/ces/opportunity/national_percentile_outcomes.csv')
df_fread   ← data.table::fread('https://www2.census.gov/ces/opportunity/national_percentile_outcomes.csv')
df_stata ← haven::read_dta('https://www2.census.gov/ces/opportunity/national_percentile_outcomes.dta')

head(df_csv)
```

```
## # A tibble: 6 × 3,914
##   par_pctile kfr_pooled_pooled kir_pooled_pooled jail_pooled_pooled
##        <dbl>             <dbl>             <dbl>              <dbl>
## 1          1             0.305             0.34              0.0559
## 2          2             0.317             0.352             0.0503
## 3          3             0.326             0.360             0.0463
## 4          4             0.333             0.367             0.0445
## 5          5             0.338             0.371             0.0417
## 6          6             0.342             0.375             0.0405
## # ℹ 3,910 more variables: married_asian_female_n <dbl>,
## #   married_asian_female <dbl>, s_married_asian_female <dbl>,
## #   imp_married_asian_female <dbl>, working_asian_female_n <dbl>,
## #   working_asian_female <dbl>, s_working_asian_female <dbl>,
## #   imp_working_asian_female <dbl>, has_dad_asian_female_n <dbl>,
## #   has_dad_asian_female <dbl>, s_has_dad_asian_female <dbl>,
## #   imp_has_dad_asian_female <dbl>, kir_top20_asian_female_n <dbl>, …
```

Help! This file froze my computer!

- Sometimes we'll be reading quite large files
  - These can be too big to fit in memory

Just read in a single row to see the column names:

```r
# I need to sete an environment variable to increase the size of the connection
# R will complain if you try to read in a file that's too big
# This will reset when I close this session.
Sys.setenv("VROOM_CONNECTION_SIZE"=1e6)
df <- read_csv('https://www2.census.gov/ces/opportunity/cz_outcomes.csv',n_max=1)
```

```
## Rows: 1 Columns: 10825
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr    (1): czname
## dbl (9658): cz, kir_natam_female_p1, kir_natam_female_p25, kir_natam_female_...
## lgl (1166): proginc_natam_female_p1, proginc_natam_female_p25, proginc_natam...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

- You can and should also consult the codebook (remember those?)

Help! This file froze my computer!

Once you know your columns, read those in:

```
Sys.setenv("VROOM_CONNECTION_SIZE"=1e6) # Telling R to read in 1 million bytes at a time
read_csv('https://www2.census.gov/ces/opportunity/cz_outcomes.csv',
  col_select=c('cz', 'kfr_pooled_pooled_p1', 'kfr_pooled_pooled_p25',
  'kfr_pooled_pooled_p50','kfr_pooled_pooled_p75','kfr_pooled_pooled_p100'))
```

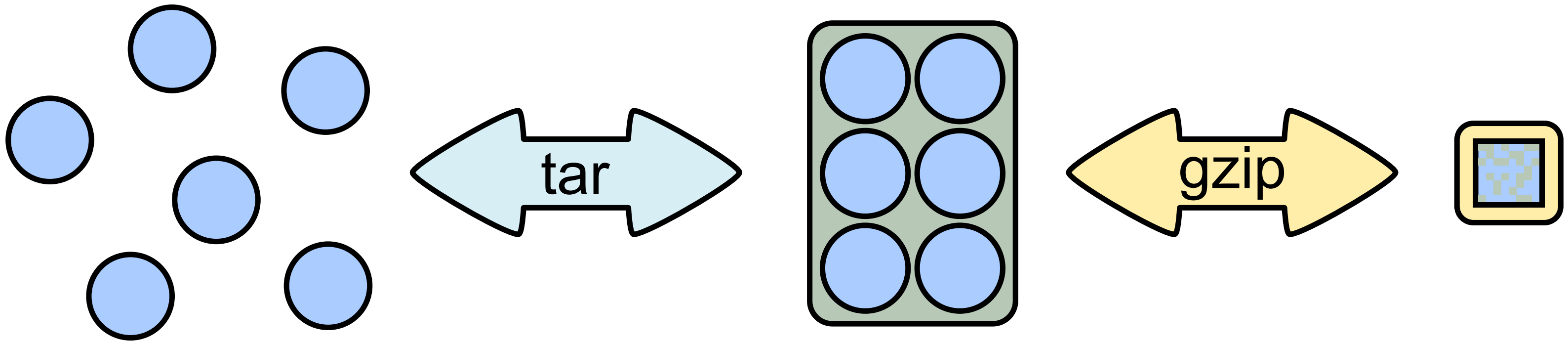
```
## # A tibble: 741 × 6
##       cz kfr_pooled_pooled_p1 kfr_pooled_pooled_p25 kfr_pooled_pooled_p50
##    <dbl>                <dbl>                 <dbl>                 <dbl>
##  1   100                0.260                 0.364                 0.461
##  2   200                0.264                 0.363                 0.456
##  3   301                0.286                 0.380                 0.467
##  4   302                0.262                 0.366                 0.462
##  5   401                0.251                 0.359                 0.459
##  6   402                0.249                 0.362                 0.466
##  7   500                0.241                 0.352                 0.454
##  8   601                0.283                 0.384                 0.478
##  9   602                0.267                 0.382                 0.489
## 10   700                0.240                 0.351                 0.454
## # ℹ 731 more rows
## # ℹ 2 more variables: kfr_pooled_pooled_p75 <dbl>, kfr_pooled_pooled_p100 <dbl>
```

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



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)

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

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

```
download.file('https://www2.census.gov/ces/opportunity/county_outcomes.zip','county_outcomes.zip')
read_csv(unz('county_outcomes.zip','county_outcomes.csv'),
  col_select=c('county','kfr_pooled_pooled_p1','kfr_pooled_pooled_p25',
  'kfr_pooled_pooled_p50','kfr_pooled_pooled_p75','kfr_pooled_pooled_p100'))
```

```
## # A tibble: 3,219 × 6
##    county kfr_pooled_pooled_p1 kfr_pooled_pooled_p25 kfr_pooled_pooled_p50
##     <dbl>                <dbl>                 <dbl>                 <dbl>
##  1      1                0.245                 0.362                 0.471
##  2      3                0.292                 0.389                 0.479
##  3      5                0.233                 0.349                 0.457
##  4      7                0.249                 0.363                 0.469
##  5      9                0.293                 0.392                 0.483
##  6     11                0.244                 0.346                 0.440
##  7     13                0.231                 0.357                 0.474
##  8     15                0.256                 0.362                 0.461
##  9     17                0.236                 0.341                 0.437
## 10     19                0.265                 0.365                 0.459
## # ℹ 3,209 more rows
## # ℹ 2 more variables: kfr_pooled_pooled_p75 <dbl>, kfr_pooled_pooled_p100 <dbl>
```

```
# Delete the file
file.remove('county_outcomes.zip')
```

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

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, ...

Big Data File Types

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)

Useful Links

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

Your challenge

- With time left, try to download each of the following files to a folder and read in a five columns of your choosing:
  - https://www2.census.gov/ces/opportunity/tract_covariates.csv
  - https://www2.census.gov/ces/opportunity/county_outcomes.zip
  - https://www2.census.gov/ces/opportunity/tract_outcomes.zip (challenge)

THese are all found on the webpage: https://www.census.gov/programs-surveys/ces/data/public-use-data/opportunity-atlas-data-tables.html

Next lecture: Coding in R