Big Data and Economics

Lecture 3: Data tips

Kyle Coombs
Bates College | ECON/DCS 368

Table of contents

Prologue

Prologue

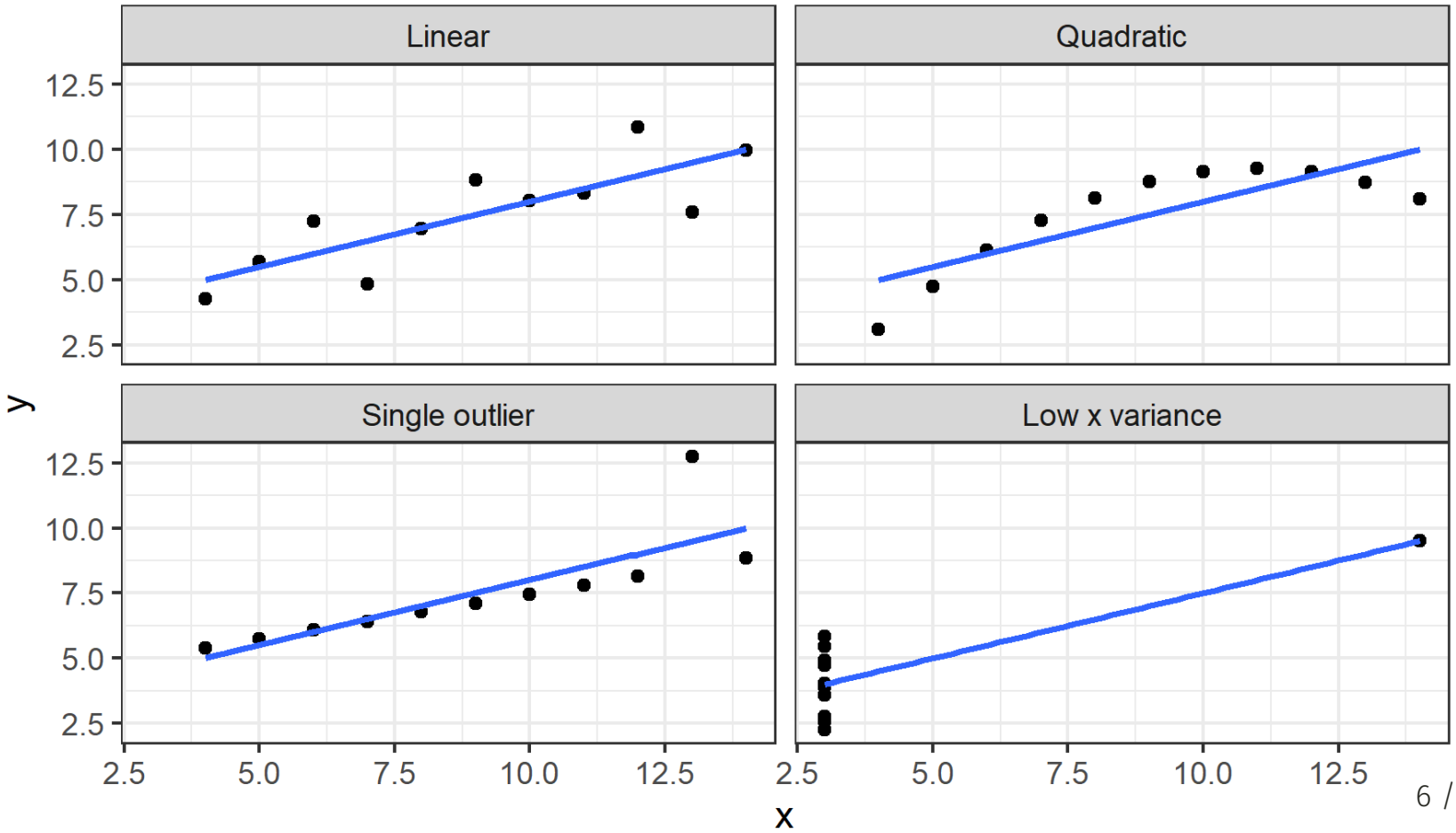
Today we'll focus on grappling with data

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

Student Presentation

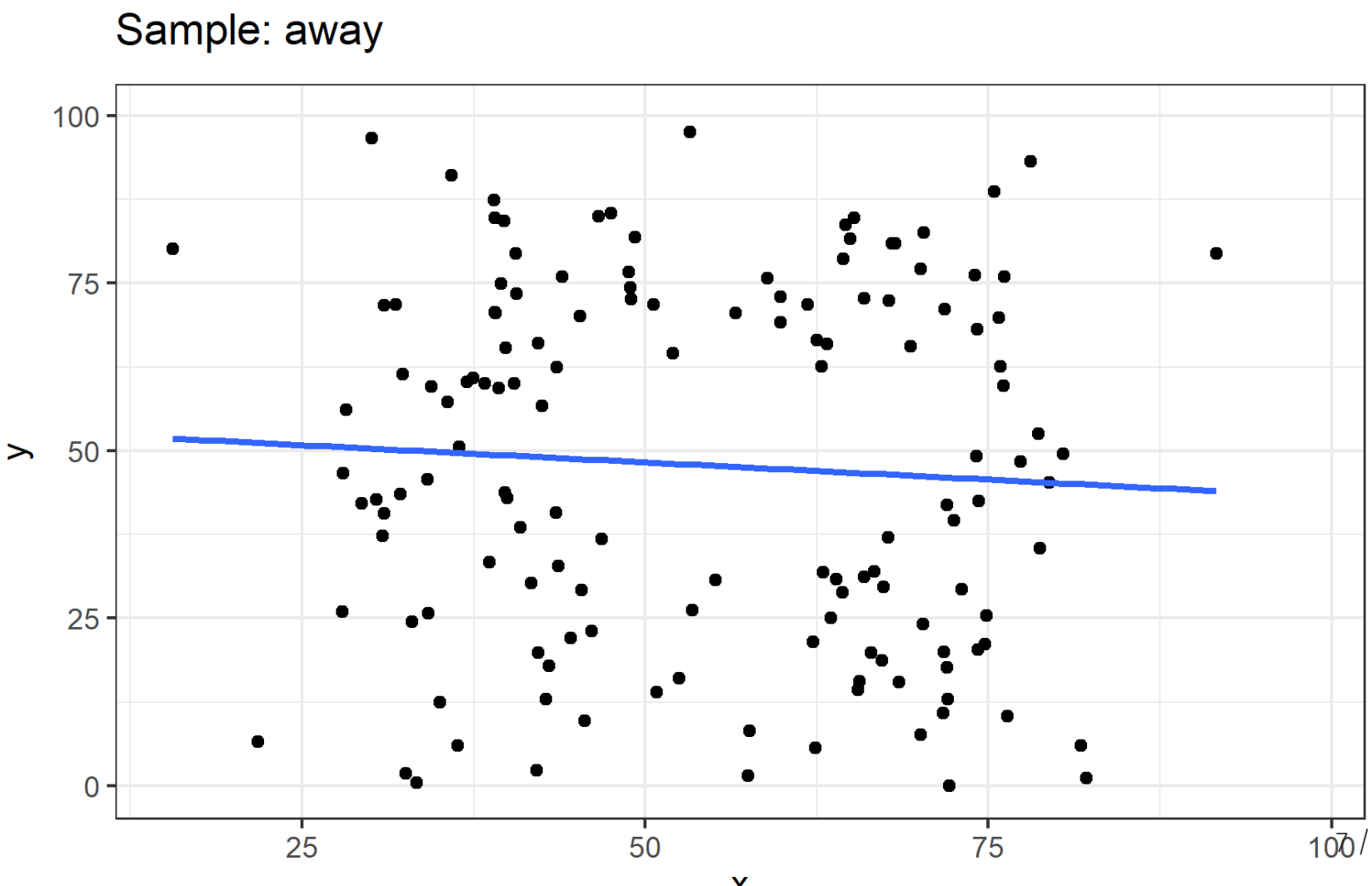
Why do we need to do this?

- 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

- A more ambitious example is the Datasaurus Dozen dataset.



Not every odd dataset is wrong

- Sometimes bizarre looking dataaset 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

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 routine usually involves driving from Melrose, MA to Lewiston, ME by 11am
    - Last year it was 9:30am -- the checklist was extra important then
- 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 fiance 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

Play along at home

On your computer

- Everyone sync your fork of the class exercises repository[1]
- Double-click `exercises.Rproj` on your computer to open RStudio at the correct working directory
- Run the housekeeping file to install the relevant packages
- Alternatively, you can open a codespace in your `exercises` repo
- We will tackle the class exercise after class, but you'll take what we do in class to fill it in
- 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

[1] **Note**: I put data files on Github. This is bad practice for large files, but fine for small ones.

Simple data checklist items

- Simple, yet non-negotiable, programming commands and exercises to check for data errors

1. Read the documentation
2. Look at the data ("Real eyes realize real lies"[1])
3. Look at summaries and frequency tables of variables
4. Plot histograms of key variables
5. Visualize by key groups
6. Check sum stats by key groups
7. 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
  - Your first problem set doesn't come with one -- you're gonna build it!

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

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
```

3a. Skim again

The `skimr` package is great!

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

Table: Data summary

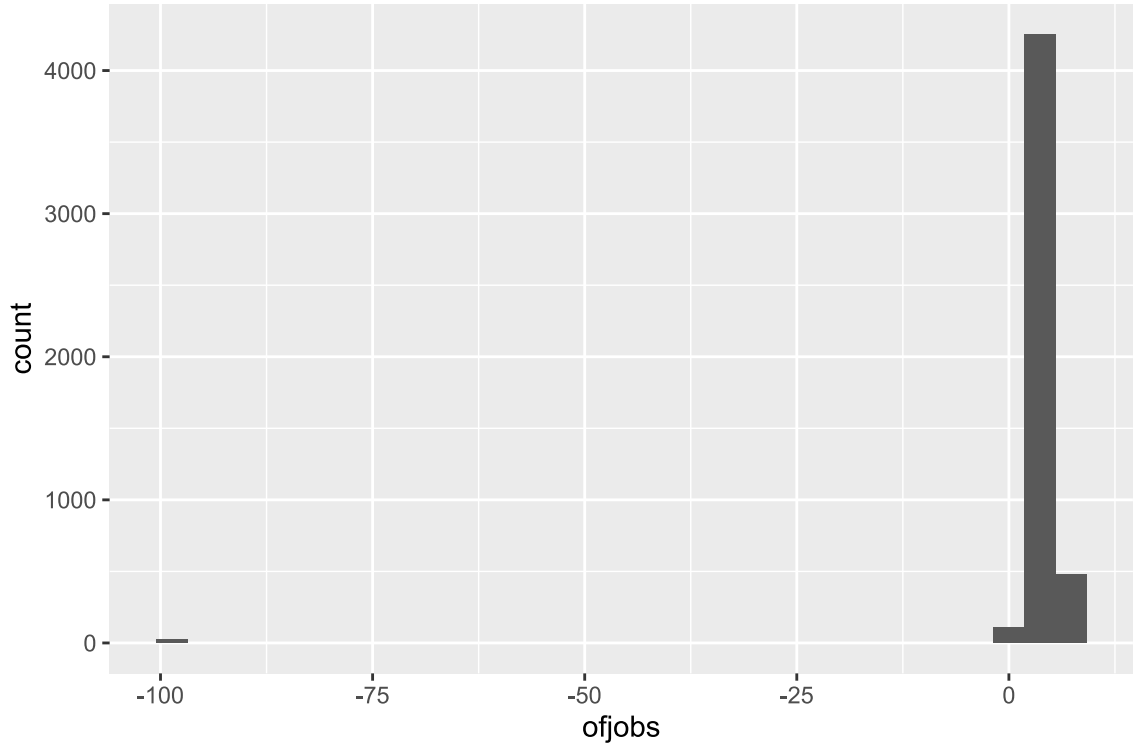
Name	resumes
Number of rows	4870
Number of columns	5
–	
Column type frequency:	
character	4
numeric	1

- What's -99?

4. Visualize the raw data

- Go beyond the eyeball and graph the data

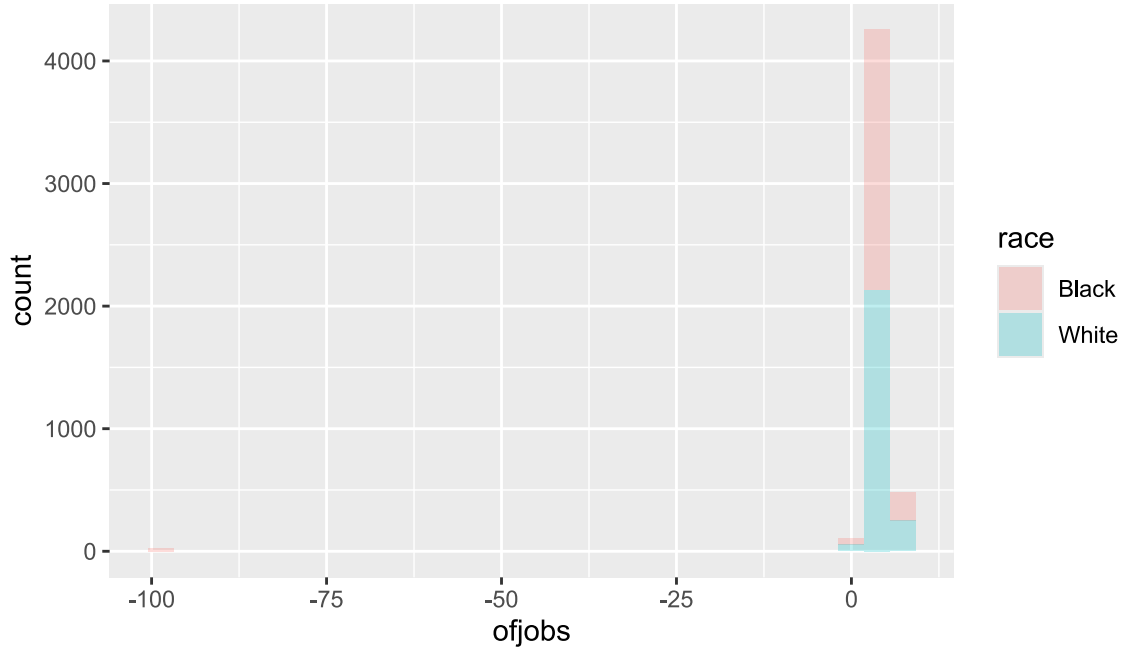
```
# ggplot is a great tool for visualizing data
ggplot(data=resumes,mapping=aes(x=ofjobs))+
  geom_histogram()
```



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

5. Visualize by group

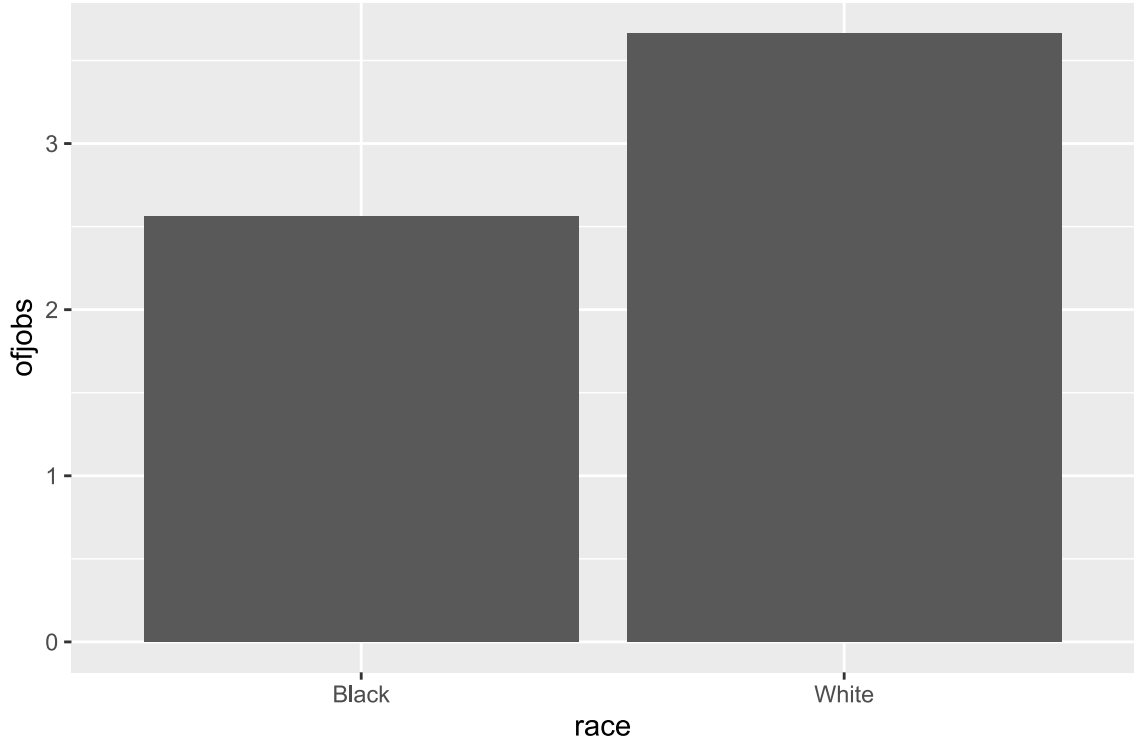
```
resumes ←  mutate(resumes,race=ifelse(race=='cauc' | race=='Caucasian','White',
    ifelse(race=="BLACK" | race=="afam", "Black",race))) # change the data up
ggplot(data=resumes,aes(x=ofjobs,fill=race))+
    geom_histogram(alpha=0.25) # alpha makes bars see through!
```



- 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

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!
- Charlie will do a short session on it as well

Downloading data

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.csv')
```

- 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 × 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
##  7 b             4 Latonya         FEMALE    BLACK          0     10.4
##  8 b             3 Carrie          FEMALE    Caucasian      0      9.88
##  9 b             2 Jay             MALE      Caucasian      0     10.4
## 10 b             2 Jill            FEMALE    Caucasian      0     10.5
```

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

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

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

```
library(jsonlite)

# Read in the JSON file
read_json('data/lakisha_aer.json',
  simplifyVector = TRUE # Simplify the data to a dataframe
  ) %>%
  head(5)
```

```
##         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
```

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('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

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

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

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

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'))
```

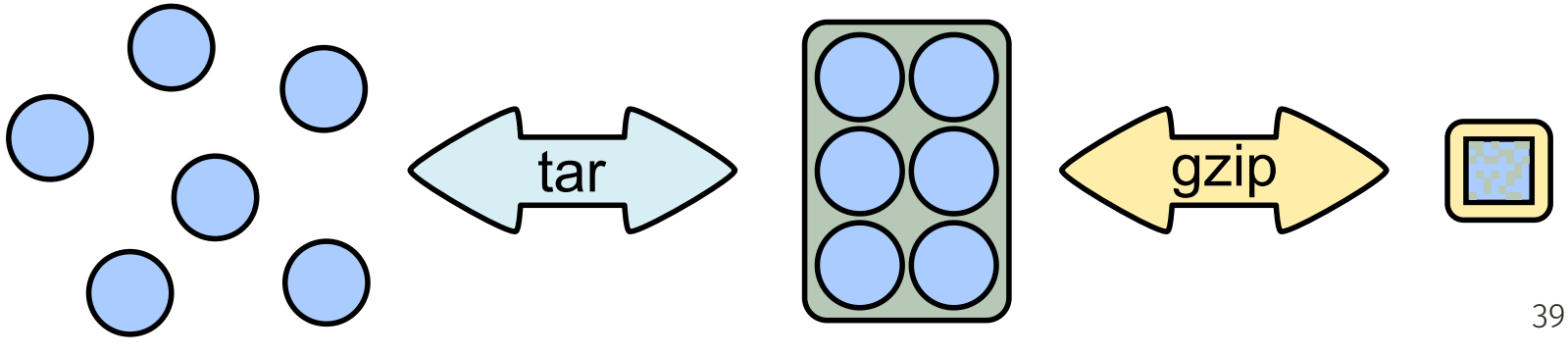
```
## # A tibble: 4,873 × 5
##    firstname       race       gender       call       ofjobs
##    <chr>           <chr>      <chr>        <chr>       <dbl>
##  1 firstname row 1 race row 1 gender row 1 call row 1     NA
##  2 firstname row 2 race row 2 gender row 2 call row 2     NA
##  3 firstname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



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

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

```
## # 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
```

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

Minor aside: what's `%>%`?

- This may be the first time you're seeing `%>%` called a pipe
- This comes from the `tidyverse`, which is a group of R functions that are designed to work well together and follow the **tidy data** principles
- Basically a pipe takes the output of the function before it and sends it to the first argument of the function after it
- It means you can do things in order of how you think about them

1. unzip first (returns the file name)
2. then feed the file name to the `read_csv` function

- You can do this with lots of functions, not just `unz()` and `read_csv()`
- We'll cover the tidyverse more in the next lecture

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 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

Appendix

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