**Data manipulation operations**

**DPLYR operations**, done with library DPLYR, DPLYR::

select(dataframe, 1, 3 , 5) returns a **dataframe** with the 1, 3, 5th columns of the table passed in.

Trying to return the same col multiple times doesn’t result in duplicates

select(dataframe, 1:5) returns  cols 1 to 5

select(dataframe, starts\_with(“a”)) returns cols that start with a

select(dataframe, -age) **the - sign DROPS ATTRIBUTES**

Can rename using **rename** or just **select**(arres\_tab, arrest\_date=arrestDate)

**Slice chooses specific entities/rows**

slice(arrest\_tab, c(1,3,5)) Unlike select, you need c() “concatonate” for the rows

slice(arrest\_Tab, 1:5) gets rows 1 to 5, no c() needed

**Filter selects entities based on attribute properties**

filter(Arrest\_tab, age > 20 & age < 20) \*notice only 1 and. This isn’t a bitwise op. Same with or |

Equaility still has != and ==

filter(arrest\_Tab, age < 20, age > 20) Filter takes multiple comparisons. So technically, don’t need &

**sample\_n**(arrest\_tab, 10) grabs 10 random entities

**sample\_frac(arrest\_tab, .1)** selects a fraction of entities at random

Pipelines**:**

%>% takes the **value** of its **left** and inserts it as the first argument of the function call to its right.

LHS %>% f(other\_arg) is the same as f(LHS, other\_arg)

Example pipeline that filters the dataset between ages 18 and 25, selects sex, district, and arrestDate renamed,  and sample 50% at random

analysis\_tab <- arrest\_tab %>%  
  **filter**(age >= 18, age <= 25) %>%  
  **select**(sex, district, arrest\_date=arrestDate) %>%  
  **sample\_frac**(.5)

JOINS default join, in this case is natural, full outer join, etc

**anti**-**join** returns one copy of each row in the first table for which no match is found

Joins are done like

Flights %>% left\_join(airlines, by=”carrier”)

Flights %>% left\_join(airlines, by = c(“carrier” = “name”)

**SQL and Database Systems-----------**

SQL is a **declarative** language, we don’t write a procedure to compute a relation, we declare what the relation we want to compute looks like.

The join operation combines rows from two tables to create a new single table.

SFW --- select, from, where

**JOINS**

Left join includes all the left and the matched right table.

Right join is the opposite of a left join.

Inner join only results in matches in both tables

Full join joins matches and unmatched stuff from both side.

**Joins with dplyr with no conditions specified are natural joins by default.**

Transactions

A transaction is a sequence of queries and update statements executed as a single unit

Transferring money frmo one account to another. **both** deduction from and account and credit to the other should happen, or **neither** should

Triggers

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.

Integretity constraints

Predicates on the database that must always hold

key constraints specify something is a primary key or unique

**Tidy Data**

Structure: assume that data is organized in rectangular data structures (tables)

Semantics: Values, attributes, and entities

Data models: a collection of concepts that describes how data is represented and accessed.

1. Modeling constructs
2. Integrity constraints
3. Manipulation languages

Relational, entity-relationship model, XML, OOP, JSON, AVRO, Thrift

**Physical models** concern itself with how data is physically stored

**Logical/Conceptual models** concern itself with the type on info stored, entities, attributes, and relationships among them.

**Data independence:** The idea that you can change the representation of data w.e changing programs that operate on it.

**Physical data independence:** can change the layout of data on disk and the programs wont change. This includes indexing, partitioning, compressing and sorting data

Extra relationships added are usually combined primary keys of the different related tables. When going backwards, if the relationship is only primary keys of other tables, it isn’t needed.

One to One: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One to Many: An entity in A is associated with any number 0 or more entities in B. Any entity in B however can be associated with at most one entity in A.

Many to One: An entity in A is associated with at most one entity in B. An entity in B however, can be associated with any number (zero or more entities) with A

K is a super key of R if the values in K are sufficient to identify a unique tuple of each possible relation r(R)

Super key K is a **candidate key** if K is minimal

One **candidate key** is selected to be a **primary key**

**Foreign key** is a primary key of a relation that appears in an other relation.

**Foreign key constraint:** the tuple corresponding to that primary key must exist. You can have multiple foreign keys from different tables in a single relationship/table. Occurs when adding a new relationship for an ER diagram.

#prelim stuff

data\_tab <- read\_csv("bpd-arrests-1.csv")

summary(data\_tab$Sex)

summary(factor(data\_tab$Sex))

# factoring

data\_tab$Sex <- factor(data\_tab$Sex)

levels(data\_tab$Sex)

arrange(data\_tab, desc(age))

# 15th oldest subject

data\_tab %>%

arrange(desc(Age)) %>%

slice(1) # slice(c(1,23,199)) or slice(1:5) or slice(seq(2, nrow(arrest\_tab), by=2))

#from 2 to nrow(arrest\_tab), increase by 2

data\_tab %>%

mutate(age\_month = 12\*Age) %>% #mutate create new attribute

select(data\_tab, Age, age\_months)

data\_tab %>%

filter(Age != 0) %>%

summarize(min\_age = min(Age), mean\_age = mean(Age), max\_age=max(Age))

data\_tab %>%

sample\_n(10) # or sample\_frac(.1)

data\_tab %>%

arrange(desc(age)) #sort by

# group by

data\_tab %>%

filter(Age != 0) %>%

group\_by(District) %>%

summarize(mean\_age = mean(Age)) # functions: mean, median, min, max, n or n\_distinct, any or all (only for True/False), sd for standard deviation

data\_tab %>%

filter(Age != 0) %>%

group\_by(District, Sex) %>%

summarize(mean\_age = mean(Age), median\_age = median(Age))

analysis\_df <- data\_tab %>%

filter(Age != 0) %>%

group\_by(District) %>%

summarize(mean\_age = mean(Age))

data\_tab %>%

pull(age) %>% # pull creates a vector of [v1, v2, v3 ... vn]

mean()

# function

summarize\_district <- function(df) {

df %>%

filter(age >= 21) %>%

group\_by(district, sex) %>%

summarize(mean\_age=mean(age))

}

# graphing

<data\_frame> %>%

ggplot(mapping=aes(<graphical\_characteristic>=<attribute>)) +

geom\_<representation>()

geom\_text(aes(lable = District))

# scatter points

# Used to visualize the relationship between two attributes

mpg %>%

ggplot(mapping=aes(x=displ, y=hwy)) +

geom\_point(mapping=aes(color=cyl))

# bar graph

# visualize the relationship between a continuous variable to a categorical (or discrete) attribute

mpg %>%

group\_by(cyl) %>%

summarize(mean\_mpg=mean(hwy)) %>%

ggplot(mapping=aes(x=cyl, y=mean\_mpg)) +

geom\_bar(stat="identity")

# histogram

# Used to visualize the distribution of the values of a numeric attribute

mpg %>%

ggplot(mapping=aes(x=hwy)) +

geom\_histogram()

# boxplot

# Used to visualize the distribution of a numeric attribute based on a categorical attribute

mpg %>%

ggplot(mapping=aes(x=class, y=hwy)) +

geom\_boxplot()

**Data cleaning------------**

**-**use tidyr and dplyr

**Tidy Data**

1. Each attribute (or variable) forms a column
2. Each entity (or observation) forms a row
3. Each type of entity (observational unit) forms a table

**Common problems \*know 2 gather and seperate**

1. Column headers are values, not variables (gather)

* Surveys are usually in this format
* Suppose the column headers are qaurter1 -> quarter4. Those are values, for something like time. I can use gather to **gather** those headers into a single column called time. The data under is still preserved because the columns were transposed into the table.
* gather(dframe, key=”col name for the headers gathered into this column”, the headers/column names.

1. Multiple variables stored in one column (seperate \*90% sure this was suppose to be split and not seperate)
2. Variables stored in both rows and column(rotate)
3. Multiple types of observational units are stored in the same table(normalize)
4. Single observational unit store in multiple tables(join)

#data scraping notes

#html node take a url and the selector to search for'

library(rvest)

url <- "https://en.wikipedia.org/wiki/Lists\_of\_100\_best\_books"

table\_node <- read\_html(url) %>%

 html\_node(".wikitable") #the class of the table is wikitable. this gets us some rough xlm

#every row in this table has a class headerSort, so were using it

Booknames <- table\_node %>% html\_nodes("[href]") %>% html\_text()

#in this example, all elements in the passed in html\_node with the attribute scope are added

"figure out attribute and then review and print out sheet"

'tidy data practice'

library(tidyverse)

library(foreign)

library(stringr)

library(plyr)

library(reshape2)

'Gather example billboard is piped in as the data frame,

week is what were calling the gathered headers in the new column

rank is what were calling the new corresponding values previously under the headers

the : just means starting from some col and ending at some col

look up r gather for info on : and c'

billboard <- read\_csv("C:/Users/andre/Documents/CMSC320/hadley\_data/tidy-data/data/billboard.csv")

view(billboard)

tidy\_billboard <- billboard %>%

 gather(week, rank, x1st.week:x76th.week, na.rm=TRUE)

tidy\_billboard

**'Seperate is the opposite of gather. It separate multiple values in the same column**

Lets say a column demo had values such as m04 representing male age 4.

we can use seperate(dataf, c("sex, age"), sep=1) to split demo into 2 columns where sep is the

character in the value to actually seperate on. in this case, 2 columns sex and age

would be made and sex woul have the value male and age would have 04'

tb <- read\_csv("C:/Users/andre/Documents/CMSC320/hadley\_data/tidy-data/data/tb.csv")

'first gather the headers into a column. each header had multiple values. yikes'

almostTidyTB <- tb %>%

 gather(SexandAge, occurence, new\_sp\_m04:new\_sp\_f65, na.rm=TRUE)

almostTidyTB

'now that the trouble columns are gathered in SexandAge, seperate them into sex and age'

tidyGoodEnoughTB <- separate(almostTidyTB, SexandAge, c("sex", "age"),sep = 8)

tidyGoodEnoughTB

---

Text and dates

String length

str\_length(c(short\_string, long\_string))

**String ops**

Str\_c(“str1”,”str2”, sep=”. “) combine strings and seperate them with a .

Str\_sub(“”, int start, int end) both ends included and counting starts at 1

str\_trim(“”, side=”both”) removes whitespace from the sides of a string

str\_split(words, “ “) splits sentence based on the 2nd arg

**Regex**. NOTE, r escapes the backslash, so 2 are required when normally one is fine. Example, to check for a . use \\.

Anchors are ^ and $

strs <- **c**("apple", "banana", "pear")  
**str\_view**(strs, "an")

**str\_view**(**c**("t867nine", "gray9"), "[aeiou]|[0-9]")

**str\_view**(**c**("color", "colour"), "colou?r")

str\_detect(words), given a vector of strings,return true for those that match and false otherwise’

**data\_frame**(word=words, result=**str\_detect**(words, "^[aeiou]")) %>% **sample\_n**(30)

Str\_count(words, regex) returns number of matches instead of TRUE or FALSE

Str\_extract(words, regex) returns extracts the string matches

----- extra

**Text datasets can be thought of in terms of**

1. Documents: the instances offree text in our dataset, nad
2. Terms: the specific, e.g, words, they contain

WE can think of **documents** as **entities**, described by attributes based on documents. To tidy text data, we tend to create **one-token-per-row** data frames that list the instances of terms in documents in a dataset.

**Best Practices----**

Reproducibility: can you reproduce your results

Open Data is the idea that some datra should be fgreely available to everyone to use and republish as tehy wish, without restrictions from copyright, patents or other mechanisms of control.

Tools that help:

* Version control such as git, svn
* Unit testing: RUnit, testrthat
* Share and publish: github, bitbucket

Ethics

**-DO NOT** let an algorithm look at protected attributes.

-These can be race, gender, sex, religion, etc

- Membership in a protected class should have no correlation with the final decision

- Demographic parity allows classifiers that select qualified candidates in the “majority” demographic and unqualified candidates in the “minority” demographic, within a protected attribute, so long as the expected percentages work out.

**- F.A.T.M.L** Fairness, Accountability, and Transparency in Machine Learning

**Fairness** can be viewed as a measure of diversity in the combinatorial space of sensitive features

**Accountability** is the ability to justify algorithmic decision-making and mitigate negative social impacts or potential harms

**Transparency** should be understandable, meaningful, accessible and measurable

FAT should be asked in data collection, modeling, and deployment

Think like an experimentalist

Plan you experiment, gather raw data, gather tools, execute, analyze, communicate

Don’t forget reproducibility and to separate the steps above

**Preliminaries----**

**The data analysis cycle:**

**Aquisition:**

Data Wrangling includes

Data aquasition, cleaning,

**General work flow, ddmmpd**

1. **Define the goal.**

What is the question, who wants to answer it, how well can we expect to answer it, how well should our result be?

1. **Data collection and management**

What data is available

Is it good enough, is it enough?

What can we derive from the data?

1. **Modeling**

What kind of problem is it? Classification, regression, etc

What model to use? Linear regression?

Is there enough data

Does it answer the question

1. **Model evaluation**

Did the model work, how well?

1. **Presentation**

How can I describe and present the data visually?

Do the measurements actually tell a story?

1. **Deployment**

Who is using it, maintaining, hosting it.

**R environment info**

***Dataframes*** basically tables

***Expressions*** are text that R can evaluate into a value like View(swiss)

***Values*** are numbers, strings, dataframes, etc.

View(swiss) produces nothing, which is also a value

***Names*** refer to values. Using Swiss tells R we want the value referenced by the name swiss.

***Functions*** are a series of isntructions that take some input value and produces a different value. The name view refers to a function. View(swiss) is a function that takes a name

**Measurement types--------------**

Categorical Data: Like heads or tails, numbers in a dice

* **Unordered** would be sex, race
* **Ordered** would be class grades

**Discrete numerical Data**

“These are attributes that can take specific values from elements of **ordered, discrete (possibly infinite) sets.** Age in years is an example.

**Continuos numerical data**

These attributes can take any value in an continuos set. Continuity is not a property of the specifiv dataset you have in hand, but rather of the process you are measuring. For example, the number arrest in a neighborhood could never be fractional. However, a **person’s height** can be measured with infinite precision.

Discrete data can be as the result of counting.

Continuous data can be the result of some physical measurement.

**Text**

Arbitrary strings that do not encode a categorical attribute

**Datetime**

Date and time of some event or observation

Units are important and can be converted to one another. In a given dataset, they will be recorded in some specific unit.

# scraping data

# same css selector type = "table" ".class" "[attribute="value"]" "#id"

# "p:first-of-type"

# "td:nth-of-type(2)"

html\_table()

# messy data

scrape\_billboard <- function(year, baseurl="https://en.wikipedia.org/wiki/List\_of\_Billboard\_Hot\_100\_number-one\_singles\_of\_") {

url <- paste0(baseurl, year)

# find table node

singles\_tab\_node <- read\_html(url) %>%

html\_node(".plainrowheaders")

# extract dates

dates <- singles\_tab\_node %>% html\_nodes("[scope]") %>% html\_text()

# extract titles and spans

title\_nodes <- singles\_tab\_node %>% html\_nodes("tr") %>% html\_node("td:first-of-type") %>% magrittr::extract(-1)

song\_titles <- title\_nodes %>% html\_text()

title\_spans <- title\_nodes %>% html\_attr("rowspan")

# make data frame

data\_frame(month\_day=dates, year=year, song\_title\_raw=song\_titles, title\_span=title\_spans,

artist\_raw=artists)

}

scrape\_billboard("2016")

# maping function to table

billboard\_tab <- as.character(2010:2017) %>%

purrr::map\_df(scrape\_billboard)

billboard\_df <- data\_frame(month\_day=dates, year="2017", song\_title\_raw=song\_titles, title\_span=title\_spans,

artist\_raw=artists)

# gather(table, type, value, -whatever stays the same)

# gather into key-value column

tidy\_pew <- gather(pew, income, frequency, -religion)

# multiple catigory in columns

tidy\_tb <- tb %>%

gather(demo, n, -iso2, -year) %>%

separate(demo, c("sex", "age"), sep=1)

# variables stored in both rows and columns

# gather by day and value in cols d1:d31, rm na

# them spread by element and value.

weather %>%

gather(day, value, d1:d31, na.rm=TRUE) %>%

spread(element, value)

# multiple relation/data in one table

# seperate into 2

song <- tidy\_billboard %>%

select(artist, track, year, time, date.entered) %>%

unique() %>%

mutate(song\_id = row\_number())

rank <- tidy\_billboard %>%

left\_join(song, c("artist", "year", "track", "time", "date.entered")) %>%

select(song\_id, week, rank)

# stringr

str\_length()

str\_c(a, b, sep=". ")

str\_sub(a, 2, 5) # starts 1

str\_trim(" I am padded ", side="both") # "both", "left", "right"

# regex

\\. for dot

start (^), end ($)

\d digit

\s spaces

[^abc] anything except this set

| match any of one or more patterns

?: zero or one

+: one or more

\*: zero or more

mathching groups:

str\_view(fruit, "(..)\\1")

str\_detect(vectorString, regex) => T/F

str\_count(vectorString, regex) => num

str\_subset(vectorString, regexe) => subset

str\_extract(vectorString, regexe) => matched content

str\_match=>First column: complete match, one column for each capture group.

str\_split

# sql commands

# create table w/ integrity constraints

CREATE TABLE customer (

ssn CHAR(9) PRIMARY KEY,

cname CHAR(15), address CHAR(30), city CHAR(10),

UNIQUE (cname, address, city));

CREATE TABLE <name> ( <field> <domain>, ... )

INSERT INTO <name> (<field names>) VALUES (<field values>)

DELETE FROM <name> WHERE <condition>

UPDATE <name> SET <field name> = <value> WHERE <condition>

SELECT <fields> FROM <name> WHERE <condition>

# missing chapter 10, 11

# joins

library(nycflights13)

data(flights)

data(airlines)

# all observations on left operand (LHS) are retained

flights %>%

left\_join(airlines)

flights %>%

left\_join(airlines, by="carrier")

flights %>%

left\_join(airlines, by=c("carrier" = "name"))

# all observations on right operand (RHS) are retained

flights %>%

right\_join(airlines, by="carrier")

# only observations matching on both tables are retained

flights %>%

inner\_join(airlines, by="carrier")

# all observations are retained, regardless of matching condition

flights %>%

full\_join(airlines, by="carrier")

# This filters the flights table to only include flights from airlines that are not included in the airlines table.

flights %>% anti\_join(airlines, by="carrier")

# samething in sql but outer

# reading problematic datas

df <- read\_csv(readr\_example("challenge.csv"))

problems(df)

df <- read\_csv(readr\_example("challenge.csv"), col\_types = cols(

x = col\_double(), y = col\_date()))

df <- read\_csv(readr\_example("challenge.csv"), col\_types = cols(.default=col\_character()))

df <- read\_lines(readr\_example("challenge.csv"))