Data Transfer, Management, and Manipulation
Week 3

PH 700A, Spring 2025

Rick Calvo

Table of contents

0.1 Outline

- Import and Export of Data
- Relational Data
- Data Frame Augmentation
  - Joining
  - Binding
  - Pivoting
  - Subsetting
- Feature Manipulation
  - Variable Structure
  - Variable Generation
  - Transformation

0.2 Data Origination

Data can exist as a file on your physical system or a file on another system.

R must be directed to the location of the file

The file containing your data must be imported into R before manipulation or analysis.

To access data on *your physical system*, either:

1. **Upload** the file to the directory of interest using the **Files** tab in the **Viewer** pane
2. Provide R with the full directory path to the location of the file on your system

To access data on *another system*, either:

1. Download the file and then upload it using option 1 above
2. Provide R with the full directory path to the location of the file on the other system (assuming permission to access is properly set)
3. Provide R with the URL to the file (if online)

0.3 Packages/Commands for File Import

data type	package	command
csv	base R	read.csv()
delimited	base R	read.table()
excel	readxl	read_excel()
sas	haven	read_sas()
spss	haven	read_sav()
stata	haven	read_dta()
sql	DBI	dbConnect()
odbc	RODBC	sqlQuery()

haven is part of the tidyverse

0.4 Import Code

`dataFrameName <- command("directoryPathToFile", options)`

Options vary and are specific to each command. View the command documentation for valid options: `help(command)`

```
install.packages('haven')   # installs the haven package
library(haven)              # loads the package to the R instance
help(read_sas)              # opens the documentation on the "read_sas" command in the help i

# imports the sas data and assigns it to a data frame object called, "df"
df <- read_sas("c:/users/rick/data/CurrentData.sas7bdat", .name_repair = "unique")
```

- ***CurrentData.sas7bdat*** is loaded as an R object called **df**
- `name_repair` is an optional parameter
- `"unique"` is the argument supplied to it
- `name_repair = "unique"` makes sure that all columns get a unique non-missing name

0.5 Exporting Data

Data can be exported to *(almost)* any format.

file type	package	command
csv	base R	write.csv()

file type	package	command
delimited	readr	write_delim()
excel	xlsx	write.xlsx()
sas	haven	write_xpt()
spss	haven	write_sav()
stata	haven	write_dta()
compressed	data.table	fread()

readr and **haven** are part of tidyverse

SAS outputs as Transport File format (.xpt)

Stata data can only be read up to version 15

0.6 Export Code

`command(df_object, "directoryPathToSave", options)`

Options are specific to each command. View the command documentation for valid options: `help(command)`

```
# loads the package to the R instance
library(readr)

# opens the documentation on the "write_delim" command in the help window
help(write_tsv)

# exports data frame to a tab-separated value file
write_tsv(df, "c:/users/rick/data/export/primaryData.tsv", col_names = TRUE, append = FALSE)
```

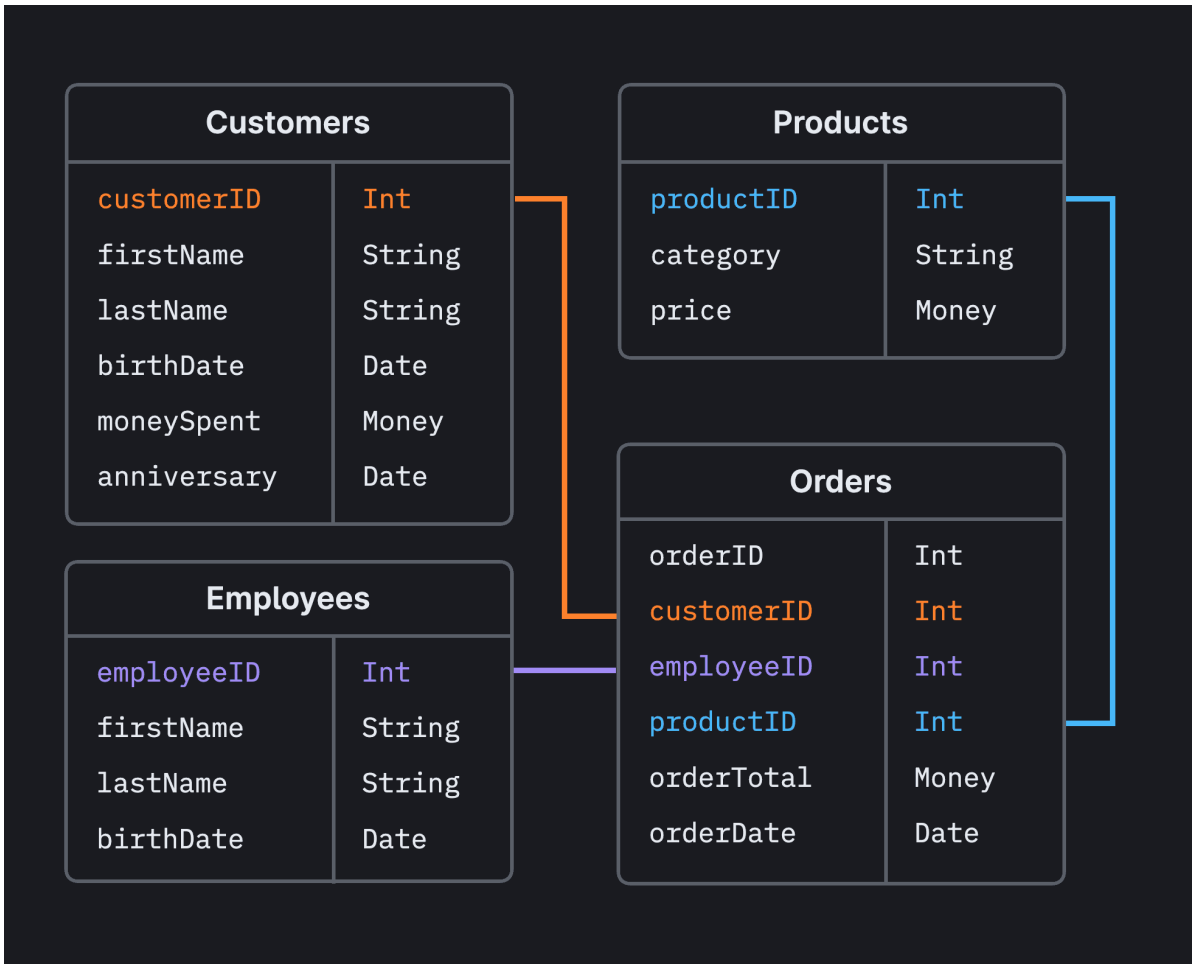
- `col_names = TRUE` will add column names to the first row
- `append = FALSE` will overwrite the file if it exists, if `TRUE`, it adds rows to the existing file

0.7 Relational Data

- Database that fragments data into smaller tables based on information type or category
- Each table has at least one **key** to allow for relations to be made
- Often used when *lots* of information needs to be organized
- Great for data storage, bad for analysis

Our objective is to assemble a dataset by correctly referencing tables of interest and merging columns we need using the keys provided.

0.8 Table Relations



Keys:

`customerID`

`employeeID`

`productID`

You determine which data frame is the *"master"* that will serve as the primary source of observations.

0.9 Example - MIMIC-IV-ED Open Demo

https://physionet.org/content/mimic-iv-ed-demo/2.2/

- 6 Tables
  - Emergency Department (ED) Visits: `edstays.csv`
  - Discharge Diagnoses (Billed): `diagnosis.csv`
  - Current Medications Reported by Patient: `medrecon.csv`
  - Medications Dispensed By Hospital: `pyxis.csv`
  - Vitals Used For Triage: `triage.csv`
  - Routine Vital Signs in ED: `vitalsign.csv`

Each table contains keys to relate variables in each frame to the correct observations.

Every sheet contains keys that can be used to join data together.

0.10 MIMIC-IV Data Import

Data are in CSV format, but are compressed as `.gz` files.

Start by importing the patient-level tables to begin development.

```
library(data.table)

df.ed <- fread("https://github.com/core-academics/sdsu.ph700a/raw/refs/heads/main/data/mimic

df.dx <- fread("https://github.com/core-academics/sdsu.ph700a/raw/refs/heads/main/data/mimic

df.triage <- fread("https://github.com/core-academics/sdsu.ph700a/raw/refs/heads/main/data/m
```

0.11 Data Preparation

Data preparation takes the majority of time when performing an analysis.

Data form, composition, and integrity often set the analytic options.

Effective data preparation reduces the overall analysis time.

0.12 Concepts Discussed

Data Construction	Variable Transformation
Joining	Type Conversion
Binding	Mutating
Pivoting	Renaming
	Deletion

0.13 Data Construction

Most of the time, we want each person to be represented in only one row

Joining - Bringing multiple data frames together to create a *wide* data frame containing all relevant variables.

Binding - Appending additional observations to extend the *length* of the data frame without substantially changing the number of variables.

Pivoting - Converting a *long* data frame containing multiple observations per person to a *wide* data frame containing multiple variables per person

0.14 Joining

When data is spread across multiple sheets, joining can be performed to consolidate them.

Also known as "merging" or "data concatenation".

Data can be matched by key in a variety of ways:

`left_join(x, y, by = "key")` - Designates `x` as the master and keeps all observations plus matches from `y`

`inner_join(x, y, by = "key")` - Only keeps matches between `x` and `y`

`right_join(x, y, by = "key")` - Designates `y` as the master and keeps all observations plus matches from `x`

`full_join(x, y, by = "key")` - Retains all observations from both `x` and `y` regardless of match

`semi_join(x, y, by = "key")` - Same as `inner_join` but will not add columns from `y`. Uses `x` variables as master.

`anti_join(x, y, by = "key")` - Retains only observations *without* a match between `x` and `y`

0.15 Binding

Also known as "appending".

Adds observations from another data frame to a master data frame.

Assumes that all the columns in both data frames are the same.

```
newdata <- rbindlist(data1, data2, ...)
```

You can have any number of objects in `rbindlist()` as long as they have the same columns and type.

0.16 Pivoting

Also known as "reshaping".

"Long" format

country	year	metric
x	1960	10
x	1970	13
x	2010	15
y	1960	20
y	1970	23
y	2010	25
z	1960	30
z	1970	33
z	2010	35

"Wide" format

country	yr1960	yr1970	yr2010
x	10	13	15
y	20	23	25
z	30	33	35

```
library(tidyverse)

longdata <- widedata %>%
```

```
  pivot_longer(
    cols = c("var1", "var2", ...),
    names_to = "old_Variable_Names",
    values_to = "value_contents_variable"
  )

widedata <- longdata %>%
  pivot_wider(
    id_cols = keyvariable,
    names_from = variable_names_Column,
    values_from = value_contents_variable
  )
```

0.17 Subsetting Data

The process of isolating data into separate data frames based on a condition(s)

```
library(data.table)

df.dogs <- subset(x = df.allpets, animal == "Dog")

df.cats <- subset(x = df.allpets, animal == "Cat")

df.turtle <- subset(x = df.allpets, animal == "Turtle")
```

This will create three separate objects each containing ONLY the respective animal.

0.18 SPECIAL CONDITIONS FOR MISSING DATA

Missing data is treated in a special way with R.

Typically takes the forms of:

- NA
- NaN

This is DIFFERENT than blank spaces and zero!

Missing data can be referenced using the `is.na()` or `anyNA()` functions

0.19 Variable Transformation

Once we have the data in our environment, we have to identify any potential issues

Variables often need to be converted, changed, or otherwise modified before they can be used for analyses.

`glimpse()` from `library(dplyr)` can be used to evaluate variables in each data frame

`describe()` from `library(explore)` could be similarly used

0.20 Data Types

Data can come in various forms:

- Numbers
- Text Strings
- Ordinal Factors
- Nominal Factors

Analyses can only be performed with *numeric data*.

Variable types can be viewed using `str()`

0.21 Conversion

`df$numeric <- as.numeric(df$factor)`

`df$factor <- as.factor(df$var)` where `var` is either a string or number

`df$ordinal <- as.ordered(df$var, levels = c("small", "medium", "large"))` where `var` is a string

Many packages can auto-detect the correct analysis to perform if the types are specified.

0.22 Data Transformation

We will use `dplyr` of the `tidyverse` for transformation.

0.22.1 Mutating

Creating or recoding a variable based on *conditions*

- `mutate()`

0.22.2 Commands to Specify Conditions

- `if_else()`
- `coalesce()`
- `case_when()`

0.23 Simple Mutate

Mutate alone can handle standard arithmetic operators.

```
df <- df %>%
  mutate(difference = endingTime - startingTime,
         totalLoad = load1 + load2 + load3)
```

A new column `difference` is generated in `df` that is equal to `endingTime` minus `startingTime`.

Another column `totalLoad` is the sum of `load1`, `load2`, and `load3`.

Any number of columns can be made with a single `mutate()` and are delimited with a comma.

0.24 if_else Example

I want to dichotomize `sbp` based on a cutpoint of 100. If `sbp < 100`, then the new variable `bloodpressure` will be equal to `hypotensive`, otherwise it will be equal to `okay`. Missing values will be kept as missing.

```
df <- df %>%
  mutate(bloodpressure = if_else(sbp < 100, "hypotensive", "okay"), missing = NULL)
```

0.25 case_when Example

I have two variables that I need to combine into one.

- *interfacilitytransfer* (String)
- *transfer* (String)

I want to generate a new variable from these two.

- *xferBinary* (Factor)

```
dfUpper <- dfUpper %>%
    mutate(xferBinary = case_when(
      interfacilitytransfer == "Yes" ~ 1,
      interfacilitytransfer == "1" ~ 1,
      transfer == "Yes" ~ 1,
      interfacilitytransfer == "No" ~ 0,
      interfacilitytransfer == "2" ~ 0,
      transfer == "No" ~ 0,
    .default = NA
    )
)
dfUpper$interfacilitytransfer <- NULL
dfUpper$transfer <- NULL
```

0.26 coalesce Example

I want to add a new variable to consolidate all non-missing values from other variables.

- *gcsRecent* is the new variable I want
- *totalgcs*, *gcstot1*, *gcstot2*, *emstotalgcs* are the fragmented variables with some missing in each

```
dfUpper <- dfUpper %>%
    mutate(gcsRecent = coalesce(gcsRecent, totalgcs, gcstot1, gcstot2, emstotalgcs))
```

coalesce() has a order – non-missing values in the first variable will be used first, followed by the next, etc.

0.27 Renaming

Simply changing a column name from an old name to a new name.

Uses `rename()` within `dplyr`.

```
df <- df %>%
  rename(newname = oldname)
```

0.28 Deletion

Removes a column from a data frame. This can be done in a couple ways.

With `dplyr`:

```
dfnew <- df %>%
  select(-var1, -var2)
```

This will remove `var1` and `var2` from `df` when it writes `dfnew`.

With base R:

```
df$oldvar <- NULL
```

This sets `oldvar` to "nothing"