Data wrangling or Manipulation in R

Huma Asif

Research computing center

University of Chicago

Outline

- Data wrangling
- Introduction about the tidyverse package
- Importing data in R
- Data manipulation with dplyr
- Pipe operators to link several functions
- Joining data with dplyr
- Hunting matching strings
- Data visualization to explore and explain the results

Data wrangling

- Data wrangling refers to the process of selecting, filtering and transforming data into a form valuable for analysis
- To find variables and observations of interest
- To join multiple datasets together
- To quickly reshape your data in a desired format
- To use group-wise summaries to explore hidden information

Data wrangling: tidyverse package

Tidyverse is the collection of packages within R for importing, manipulating and visualizing data.

Loading Package

- Install and load the tidyverse package

  install.packages("tidyverse")

  library("tidyverse")
- This will load the following packages
- **readr** : import files
- **dplyr**: filtering, selecting, sorting , summarizing and reshaping data
- **tidyr**: tidy data
- **ggplot2**: data visual communication
- **tibble**: to explore tibbles, a special kind of data frame
- **stringr**: strings processing
- **purrr**: simplify the iteration
- **forcats**: to work with factors

Connecting to Research computing cluster (RCC)

- Login on midway

ssh -Y CNETID@midway2.rcc.uchicago.edu

Secure shell (ssh) : a secure way to access computer over a network

-Y : enable X11 forwarding and the remote machine is treated as a trusted client

- RCC user guide:

https://rcc.uchicago.edu/docs/connecting/index.html

Module system

- RCC uses a software module system to manage the software packages that are loaded into your environment.
- We will first load the required modules

  module load python/anaconda-2020.02

  module avail R

  module load R/3.6.1
- To see the list of software modules currently loaded in your environment use the *list* command

  module list
- RCC user guide:

https://rcc.uchicago.edu/docs/software/index.html

RCC compute nodes

- Once you have logged in on Midway, you will be connected to the login nodes. You can use login nodes for any work that is short-running and not computationally intensive.
- For intensive computation and long-running process, we can use compute nodes

Cluster	Partition	Compute cores (CPUs)	Memory	Other configuration details
midway	broadwl	28 x Intel E5-2680v4 2.4GHz	64 GB	EDR and FDR Infiniband interconnect
	broadwl-lc	28 x Intel E5-2680v4 @ 2.4 GHz	64 GB	10G Ethernet interconnect
	bigmem2	28 x Intel E5-2680v4 @ 2.4 GHz	512 GB	FDR Infiniband interconnect
	gpu2	28 x Intel E5-2680v4 @ 2.4 GHz	64 GB	4 x Nvidia K80 GPU

- You can see a summary of the partitions on Midway using **sinfo** command
- RCC user guide:

https://rcc.uchicago.edu/docs/running-jobs/index.html#login-nodes-vs-compute-nodes

Let’s start !

Jupyter software

- Interactive computing environment to experiment and share code
  - jupyter notebook
  - jupyter lab
- When you launch the notebook the first component which is shown is
  - Jupyter notebook dashboard
    - Open notebook documents, add or delete files
    - Manage running kernels
      - Kernel is a “computational engine” that executes the code
- Installing R kernel for Jupyter notebook
  - You can install IRkernel package by running the following command in an R console
    - Install.packages(“IRkernel”)
    - Making the kernel available to Jupyter
      - IRkernel::installspec()

Running Jupyter notebook on Midway compute node

- Jupyter notebook script : launch-jlab.sh script
- The launch-jlab.sh script allows one to run a jupyter lab notebook from a compute node with the resources specified in the header of the script.
- Modify the modules and resources

  less launch-jlab

  vim launch-jlab.
- To launch the script,

  sh launch-jlab.sh
- The compute nodes are only accessible from the campus network
- So we will first connect to the campus VPN before trying to launch a Jupyter notebook on a compute node to connect from our local web browser
- Useful links:

https://vpn.uchicago.edu/+CSCOE+/portal.html

https://rcc.uchicago.edu/docs/software/environments/R/index.html#mdoc-r

Connecting to the Juypter Lab Session from your Web Browser

- Two sets of instructions are displayed to screen when the lab session begins.
- The first option is for those connected to UChicago network via VPN so we will choose this option.
- Check the status of your job using

  squeue
- RCC user guide:

https://rcc.uchicago.edu/docs/software/environments/python/index.html#running-jupyter-notebooks

https://git.rcc.uchicago.edu/jhskone/jupyter-lab/tree/master

```
  WAITING FOR RESOURCES TO BECOME AVAILABLE (CTRL-C TO EXIT) .....................
------------------------------------------------------------------------
  STARTING JUPYTER NOTEBOOK SERVER ON NODE midway2-0183
  THIS SESSION WILL TIMEOUT IN 23 HOUR(S) AND 59 MINS
  SESSION LOG WILL BE STORED IN nb_session_6331029.log
------------------------------------------------------------------------

  TO ACCESS THIS NOTEBOOK SERVER THERE ARE TWO OPTIONS THAT DEPEND
  ON WHETHER YOU ARE CONNECTED TO THE CAMPUS NETWORK OR NOT

  IF CONNECTED TO THE CAMPUS NETWORK YOU SIMPLY NEED TO COPY AND
  AND PASTE THE FOLLOWING URL INTO YOUR LOCAL WEB BRWOSER:

  http://10.50.221.183:8418/?token=e019b6b70acfad932bd53fb2df2e1e98e82e064263339f8e

  IF NOT ON THE CAMPUS NETWORK, DO THE FOLLOWING TWO STEPS

  1.) REVERSE TUNNEL FROM YOUR LOCAL MACHINE TO MIDWAY BY COPYING
      AND PASTING THE FOLLOWING SSH COMMAND TO YOUR LOCAL TERMINL
      AND EXECUITING IT

   ssh -N -f -L 8418:10.50.221.183:8418 humaasif@midway2.rcc.uchicago.edu

  2.) THEN LAUNCH THE JUPYTER LAB FROM YOUR LOCAL WEB BROWSER BY
      COPYING AND PASTING THE FOLLOWING FULL URL WITH TOKEN INTO
      YOUR LOCAL WEB BROWSER:

  http://localhost:8418/?token=e019b6b70acfad932bd53fb2df2e1e98e82e064263339f8e

------------------------------------------------------------------------
  TO KILL THIS NOTEBOOK SERVER ISSUE THE FOLLOWING COMMAND:

     scancel 6331029

------------------------------------------------------------------------
```

Download the workshop material

- Download the workshop material to your home directory on the RCC cluster
- URL: https://github.com/rcc-uchicago/workshopDataWranglingOct.git
- Run this command to download the material

git clone https://github.com/rcc-uchicago/workshopDataWranglingOct.git

For GeneRIFs database:

https://drive.google.com/file/d/1To0ydveyQ1sapb_Ws1BZRyNA4BtrDEig/view?usp=sharing

Or

https://uchicago.box.com/s/3018iv1tyz1bcx5zmvpfr3frc6jbxepi

- Transfer the file from your local computer to the server

scp filename CNETID@midway2.rcc.uchicago.edu:/PATH/TO/FOLDER

Data types and structures in R

- Some of the very basic data types in R are
- Decimal values like 10.5 are called numerics (is.numeric() function)
- Whole numbers like 4 are called integers. To treat 4 as integer, you should suffix it will “L” i.e. 4L (is.integer() function)
- R has two primary ways of handling character data (text or string values): character and factor (is.character())
- Boolean values (TRUE or FALSE) are called logical (is.logical())
- You can check the data type with the class() function
- Main data structures in R are
- Vectors, matrix, list and data frame (str() function)
- Specialized data frame is called tibble
- Difference between tibble and data frame is in
  - It never convert strings to factors
  - printing

Example Datasets

- List of differentially expressed genes (DEGs)
  - Gene_list1-organ.csv
- Database:
- **GeneRIF**
- GeneRIFs are available from the National Center for Biotechnology Information (NCBI) Gene database.
- GeneRIF : Gene Reference Into Function (NCBI)
  - a concise phrase describing a function of gene.

(Jimeno-Yepes et al, 2013)

Importing data in R: readr & data.table

- There are several ways to import data in R.
- Data from flat files i.e. simple text files that display data as tables
- utils package (default R package, multiple times slower)
- readr package
- Data from flat files i.e. simple text files that display data as tables
- From a comma separated values (csv) files using read_csv()
  - Gene_list1.csv
- From a tabs separated files using read_tsv()
  - generifs_basic
- data.table package
  - install.packages("data.table")
  - library(data.table)
  - fread() infer column types and separators and is extremely fast
- Print out the current working directory using getwd() function
- Change the current working directory to the workshop directory using setwd() function
- List all the files that exists in your working directory using list.files() / dir() function
- Check the dimensions of your data using dim() function

Structure of GeneRIF

- **Tax-ID :** Taxonomy ID of species
- **Gene-ID:** Entrez Gene ID
- **PubMed-ID:** a published paper describing that function, implemented by supplying the PubMed ID of a citation in PubMed.
- **Last-updated-time stamp:** Time when the file is updated
- **GeneRIF :** Lcn2-mediated regulation of labile iron protects the host against sepsis

Data manipulation with dplyr

- dplyr involves using the “grammar of data” to perform data manipulation.
- The four basic dplyr verbs to explore and transform a dataset
- select() : subset variables (columns)
  - Select helpers : contains(), starts_with(), ends_with(), last_col()
  - To remove a variable add a minus infront of column name
    - How to select GeneID?
- filter() : subset observations based on a condition (rows)
  - Filter Genes With geneRIF for Human and Mouse
- arrange() : sort the data based on one or more variables
- mutate() : add new variables or change the existing variables
  - Add taxonomy details

Pipe operators to link several functions

- A pipe operator %>% a new paradigm to link multiple functions
- It takes the output from the function that comes before it and feeds it into the first argument of the function that comes after the pipe.

Object %>% function(…, arg2, arg3,…)

e.g.

```
File %>%
select(TaxID, GeneID) %>%
group_by(TaxID ) %>%
mutate(n = n())
```

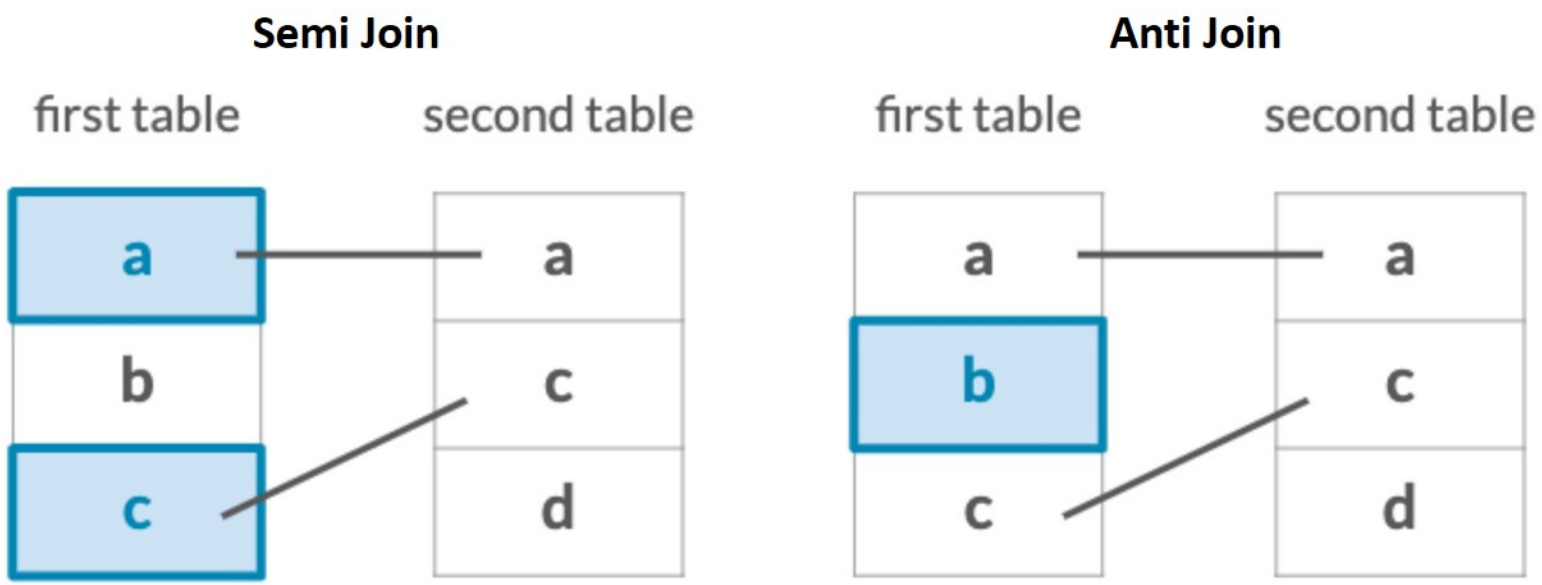
- We can read it as take file, THEN, select TaxID and GeneID THEN group by TaxID and THEN add a count column

Reshaping: Joining data with dplyr

- An important process of data manipulation is joining multiple tables together.
- There are two types of verbs designed for joining tables in dplyr
- Filtering joins: filter data based on data from another table
  - filter observations from one table based on whether or not they match an observation in another table
- Mutating joins: combine variables from multiple table
  - add new variables to one table from matching observations in another
- Keys : variable (or set of variables) used to connect the tables. The key column is encoded by “by” argument.

Types of filtering joins

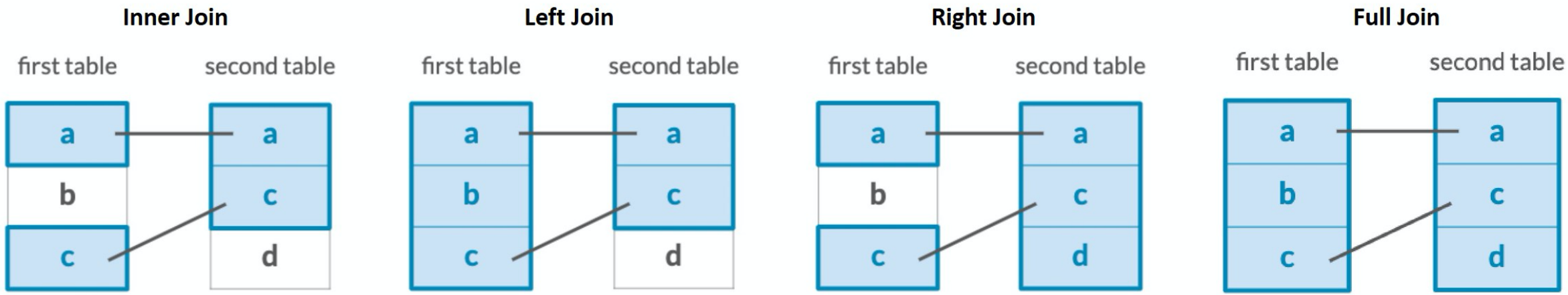
- semi_join: Filter the first table for observations that match the second table.
- anti_join : Filter the first table for observations that don’t match the second table



Types of mutating joins

- Outer join: observations that appear in at least one of the tables
  - full_join: keeps all observations from both tables
  - left_join : keeps all the observation of first or left-hand table, whether or not it occurs in the second or right table
  - right_join : keep all the observations in the second or right table , whether or not they appear in first or left-hand table
- inner_join : Observations that appear in both tables
  - It works the same way with either table in either position. Just the order of columns will appear different depending on which table comes first.

Mutating Joins



Extracting genes of interest from GeneRIF database using dplyr joins

Let’s practice !

Hunting matching strings

- Strings : elements of character vectors are known as strings. You use quotes to tell R to interpret something as string.
- We will use stringr package to manipulate the textual data.
  - Find a string that contains a pattern
  - Replacing parts of strings that match a pattern
- All stringr functions start with str_
- It uses regular expressions (regexps) : a language for describing patterns in text
- Main functions we will use today are
- str_c() : c is the short of concatenate, stringr version of paste
- str_detect() : Does the string contain the pattern?
  - It returns a logical vector with TRUE for elements that contain the pattern and FALSE otherwise
- str_replace(): replaces a pattern in the input strings with a specified replacement string. It replaces just the first occurrence of the pattern
- str_replace_all(): replaces every occurrence of the pattern

Aggregating and Transforming data

- Aggregating Data: to take many observations and summarize them into one
  - group_by() : split the data based on a variable and then apply function to each partition
  - count(): to find out the number of observation. (sort and wt arguments)
    - To change the name of the column use rename()
  - summarize() : collapsing a large data into a single observation
- Transforming data:
  - select, rename, mutate
  - Transmute : combination of select and mutate

Let’s practice !

Exporting Data

- Now that you have learnt how to extract useful information from your data, you may want to save it. You can do it using
  - fwrite() function from data.table package
  - or write_csv() function from readr package
  - write_csv() function generates CSV files
- Wrap the functions inside system.time() to see how long these functions take
- We can also write results in one excel sheet using write.xlsx() from openxlsx package
- Transfer file from server to local directory
  - scp CNETID@midway2.rcc.uchicago.edu:/FolderPath/workshopOutputFile.xlsx DestinationFolder
  - scp : secure copy protocol

Sanity check

- Let's check few results
- Go to NCBI Gene database

  https://www.ncbi.nlm.nih.gov/gene/
- Genes without GeneRIF
  - GeneID : 99543, 110785
- Genes with GeneRIF
  - GeneID: 117586, 12268
  - Genes with GeneRIF (without Disease 1)
    - GeneID: 11425, 57319
  - Genes with GeneRIF (with Disease 1)
    - GeneID: 11450, 216799

Data visualization

- Graphics are built on underlying grammar and we will explore this using ggplot2 package
- Two key concepts
  - Layer grammatical elements
  - Aesthetic mapping
- We will learn about three essential grammatical elements
  - Data : the data we want to visualize
  - Aesthetics: the scales onto which the data will be mapped
  - Geometries: the visual elements used for the data
- Optional layers include
  - Themes: non-data components of plot e.g. background, title, labels, etc.
  - Coordinates: The space on which the data will be plotted
  - Facets : Plotting small multiples
- Layers are added using a + (plus) sign

Explore and Explain

- Most frequent genes with disease annotation
- Number of genes by organ
- Number of genes by timepoint and organ
- Functional annotation of genes
  - Genes involved in different biological pathways
- We can try bar plot
  - geom_bar() : Counts the number of genes at each position
  - geom_col() is used when we want the heights of the bars to represent exact values in the data
- Or a scatter plot
  - geom_point()

Let’s practice !