Introduction to `tidyverse` in R

DSC 152 – Week 2 Discussion

Apr. 8th, 2026

What is `tidyverse`?

- Collection of R packages sharing common design philosophy and syntax
- Packages installed together all at once with `install.packages("tidyverse")` and are loaded with `library(tidyverse)`
- Some common ones include:
  - `readr`: reads data into R from files
  - `dplyr`: helps filter, transform, summarize, operate on dataframes
  - `ggplot2`: creates better plots and visualizations
  - `stringr`: deals with strings, character data

R Data Types

A simple 'formal introduction' to data types in R:

Python	R	Examples
`float`, `int`	`double`	`3.14`, `-5`
`str`	`character`	`"hello"`
`bool`	`logical`	`TRUE`, `FALSE`

- Most numbers (numeric values) are stored as `doubles` in R
- Check object's types with `typeof(x)` or `class(x)`
- Can convert to numerics via `as.numeric(x)`
- Types in R matter: `"1"` is not the same as `1`

Dataframes (traditionally)

- Dataframes in R are functionally the same as pandas DataFrames
- They are essentially collections of vectors (columns) that must be the same length

```r
# Define vectors
names <- c('Alice', 'Bob', 'Charles')
age <- c(32, 28, 29)
is_active <- c(TRUE, TRUE, FALSE)

# Use vectors to create dataframe
employees <- data.frame(names, age, is_active)

# Define data frames using vecotrs themselves
employees2 <- data.frame(
    names = c('Alice', 'Bob', 'Charles'),
    age = c(32, 28, 29),
    is_active = c(TRUE, TRUE, FALSE)
)
head(employees)  # head(eployees2) should be the same
```

Dataframes (with `tidyverse`)

- `Tibble` is a modern, lazier version of a dataframe that is used in `tidyverse` library
- Prints more cleanly than normal dataframes and is more informative with respect to data types
- Logic to create this 'dataframe' remains the same

```r
# Create dataframe using original data
employees <- data.frame(names, age, is_active)

# Create dataframe using tibble
employees_tibble <- tibble(names, age, is_active)
```

Five Functions of `dplyr`

Some key functions you will use quite often:

- `filter()`: keep rows that only match a condition
- `summarize()`: compute a particular summary statistic per group in the data
- `group_by()`: split data into groups (used commonly alongside `summarize`)
- `mutate()`: create or apply function/transformation to column
- `select()`: keep only certain columns

We will see how to use these in some examples soon.

Pipe Operator %>%

R code without the pipe operator:

```
summarize(
    filter(df, year = 2000),
    avg_scores = mean(scores)
)
```

R code with the pipe operator:

```
df %>%
    filter(year == 2000) %>%
    summarize(avg_scores = mean(scores))
```

- Pipe passes whatever is on its left as the first argument to the function on its right
- Lets you read operations top-to-bottom instead of inside-out (avoids complicated nested expressions)

Grammar of ggplot2 graphics

Three key components:

(i) **Data**: dataframe (data) you are plotting from

(ii) **Aesthetic mapping**: which variables get mapped to which visual properties

```
aes(x = ..., y = ..., fill = ..., color = ...)
```

(iii) **Geometry**: kind of plot to visually draw/produce (e.g. `geom_histogram()`, `geom_boxplot()`, `geom_point()`)

```
1 ggplot(my_data, aes(x = score)) +
2     geom_histogram()
```

Layers are added with + (not the pipe operator %>%) and every ggplot starts with `ggplot()` so that the other geometric layers are stacked on top.

Importing data

```r
# Import library
library(tidyverse)

# Tab-delimited .txt file
df <- read_delim(''myfile.txt'', delim = ''\t'')

# Comma-separated .csv file
df <- read_csv(''my_file.csv'')

# Helps to glimpse afterwards to get idea of data
glimpse(df)
```

- In RStudio: Environment panel → "Import Dataset" → "From Text (readr)..."
- Set delimiter, check the data preview, then copy the code from the "Code Preview" box into your markdown file as a chunk of R code
- `glimpse()` function shows column names, types, first couple values

Live Demo in R

- Importing data
- String splitting
- Mutations / functions applied to dataframes using pipe operator
- Problems

Couple Problems

(1) Consider three runners, Alice, Bob, and Charles, who each ran a 10k last weekend. Their respective times were "32:14", "29:58", and "30:49".

   (a) Create a `tibble` dataframe `runners` using `tidyverse` logic storing each runner's name and their 10k time.

   (b) Parse their time, which was originally a string, into the total number of seconds for each runner. Call this new column `total_seconds`. (Hint: the function `as.numeric(ms(x))` converts `x` into the number of seconds, as a numeric data type.)

   (c) What is `typeof()` on the runners' 10k times (i.e., run it on the column in the dataframe)? What about `typeof(total_seconds)`? Print both.

Recall that `df$name` gets you the column called `name` in the dataframe.

Couple Problems (cntd.)

(2) Suppose you're given quiz scores for various students in a particular semester.

```
quizzes <- tibble(student = rep(c("A", "B", "C", "D", "E", "F"), each=2),
                  section = rep(c("Section 1", "Section 2"), each=6),
                  quiz = rep(1:2, times=6),
                  score = c(88, 72, 95, 80, 65, 70, 68, 74, 90, 85, 82, 96))
```

(a) Using one chain, compute each student's mean quiz score and their total points across all quizzes.

(b) Extend the chain you previously wrote to keep only students whose mean score is above 75. Then arrange the students from highest to lowest mean (descending order).

(c) Using the original data, make a histogram of `score`. Color the bars by `section` using `fill = section` inside your `aes()` call. Add a title and axes.

(d) Instead of just using a histogram, make a boxplot of score by section (use `geom_boxplot`) with `section` on the x-axis and `score` on the y-axis.

Solution to #1

```r
# (a) Create the tibble dataframe
runners <- tibble(
  names = c("Alice", "Bob", "Charles"),
  runtimes = c("32:14", "29:58", "30:49")
)
glimpse(runners)

# (b) Pare time into total seconds
runners <- runners %>%
  mutate(
    total_seconds = as.numeric(ms(runtimes))
  )
glimpse(runners)

# (c) Check and print types
type_time <- typeof(runners$runtimes)
type_seconds <- typeof(runners$total_seconds)
print(paste("Type of 10k times column:", type_time))
print(paste("Type of total_seconds column:", type_seconds))
```

Solution to #2

```r
# (a) Create one chain to group by student
student_summary <- quizzes %>%
  group_by(student) %>%
  summarize(mean_score = mean(score), total_pts = sum(score))
student_summary

# (b) Extend chain to now filter based on students who do better than 75
quiz_summary <- student_summary %>%
  filter(mean_score > 75) %>%
  arrange(desc(mean_score))
quiz_summary

# (c) Create histogram
ggplot(data=quizzes, aes(x=score, fill=section)) +
  geom_histogram(binwidth = 5, color='gray') +
  labs(title = 'Distribution of Quiz Scores by Section', x = 'Quiz Score',
       y = 'Frequency')

# (d) Create boxplot
ggplot(quizzes, aes(x = section, y = score, fill = section)) +
  geom_boxplot() +
  labs(title = "Quiz Scores by Section", x = "Section", y = "Score") +
  theme(legend.position = "none") # Hide legend as x-axis labels suffice
```