Introduction to R pipes and dplyr

Paul M. Magwene

The magrittr pipe operator

An R package called `magrittr` popularized a new operator called a pipe (Note: pipes are an older computing concept dating back to the origins of the Unix operating)

- The pipe operator is %>%

Using pipes:

- `x %>% f()` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x,y)`.

Examples:

- Single argument function:

```
pi %>% cos()
[1] -1
```

- For single argument functions, after the pipe you can drop the parentheses:

```
pi %>% cos  # same as above
[1] -1
```

- Multi-argument functions

```
100 %>% log(base=10) # 100 is treated as the first argument
[1] 2
```

Building pipelines with the pipe operator

The pipe operator allows us to build analysis "pipelines".

A pipeline series of function calls that filter and/or transform our data

```
letters %>%           # start with letters vector
  str_to_upper %>%    # convert to upper case
  tail(10) %>%        # get last 10 elements
  str_flatten("-")    # join into single string, separated by '-'
[1] "Q-R-S-T-U-V-W-X-Y-Z"
```

The pipe operator helps to make our intent clearer, as compared to nested function calls:

```
str_flatten(tail(str_to_upper(letters), 10), "-")
[1] "Q-R-S-T-U-V-W-X-Y-Z"
```

The "built-in" pipe operator (R 4.1+)

The magrittr pipe operator became popular enough that the R developers added a native pipe operator to the R language starting in release 4.1.

- The native pipe operator is `|>`

The native pipe operator works similarly to the magrittr pipe:

- `x |> f()` is equivalent to `f(x)`
- `x |> f(y)` is equivalent to `f(x,y)`.

`magrittr` pipe allows you to drop parentheses for single argument functions, but this is not allowed with the native pipe:

```
pi %>% cos   # equivalent to pi %>% cos()
pi |> cos  # this is an error with the native pipe
pi |> cos()  # must add parentheses with native pipe
```

Placeholder, magrittr pipes

`magrittr` pipe allows you to use a "placeholder" symbol (`.`) for cases where the object your piping is not the first argument in the function being called:

```
# the period (.) is the placeholder
c("A", "B", "C") %>%
  data.frame(x = c(1,2,3), y = .)
  x y
1 1 A
2 2 B
3 3 C
```

`magrittr` placeholder can appear multiple times:

```
c("A", "B", "C") %>%
  tibble(x = c(1,2,3), y = ., z = str_to_lower(.))
# A tibble: 3 x 3
      x y     z
  <dbl> <chr> <chr>
1     1 A     a
2     2 B     b
3     3 C     c
```

Placeholder, native pipes

A placeholder symbol (_) for use with the native pipe was introduced in R 4.2, but it has several limitations:

- Can only appear once
- Can only be used with named arguments

This works:

```
c("A", "B", "C") |>
  tibble(x = c(1,2,3), y = _)
# A tibble: 3 x 2
      x y
  <dbl> <chr>
1     1 A
2     2 B
3     3 C
```

But this doesn't:

```
c("A", "B", "C") |>
  tibble(x = c(1,2,3), y = _, z = str_to_lower(_))
```

What is dplyr?

dplyr is a package that provides a "grammar for data manipulation"

Core "verbs" in the dplyr package:

Column manipulation:

- `select()`
- `mutate()`

Row manipulation:

- `filter()`
- `arrange()`

Collapsing and Grouping:

- `summarize()`
- `group_by()`

All these functions return new data frames instead of modifying existing data frames

select() subsets columns

```
# select two columns
select(iris, Sepal.Length, Petal.Length) |> head(3)
  Sepal.Length Petal.Length
1          5.1          1.4
2          4.9          1.4
3          4.7          1.3
```

```
# select everything BUT the species column
select(iris, -Species) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
```

`select()` has some helper functions for powerful filtering

See https://tidyselect.r-lib.org/reference/language.html

- `var1:var10`: variables lying between var1 on the left and var10 on the right. Also works with numeric indices.
- `everything()`: all variables.
- `all_of(vars)/any_of(vars)`: matches names stored in the character vector vars.
- `last_col()`: furthest column on the right.
- `where(predicate)`: all variables where predicate function returns TRUE. e.g `where(is.numeric)`
- `starts_with()`: Starts with an exact prefix.
- `ends_with()`: Ends with an exact suffix.
- `contains()`: Contains a literal string.
- `matches()`: Matches a regular expression.
- `num_range()`: Matches a numerical range like x01, x02, x03.

select() helper examples

```
select(iris, starts_with("Petal")) |> head(3)
  Petal.Length Petal.Width
1          1.4         0.2
2          1.4         0.2
3          1.3         0.2
```

```
select(iris, ends_with("Length")) |> head(3)
  Sepal.Length Petal.Length
1          5.1          1.4
2          4.9          1.4
3          4.7          1.3
```

```
select(iris, where(is.numeric)) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
```

mutate() adds or transforms columns

```
mutate(iris, Species = str_to_upper(Species)) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  SETOSA
2          4.9         3.0          1.4         0.2  SETOSA
3          4.7         3.2          1.3         0.2  SETOSA
```

```
# mutate can refer to multiple existing columns
iris |>
  select(starts_with("Sepal")) |>
  mutate(Sepal_Sum = Sepal.Width + Sepal.Length) |>
  head(3)
  Sepal.Length Sepal.Width Sepal_Sum
1          5.1         3.5       8.6
2          4.9         3.0       7.9
3          4.7         3.2       7.9
```

```
# can refer to columns created "on the fly"
iris |>
  select(starts_with("Sepal")) |>
  mutate(Sepal_Sum = Sepal.Width + Sepal.Length,
         Sepal_Width_Ratio = Sepal.Width / Sepal_Sum) |>
  head(3)
  Sepal.Length Sepal.Width Sepal_Sum Sepal_Width_Ratio
1          5.1         3.5       8.6         0.4069767
2          4.9         3.0       7.9         0.3797468
3          4.7         3.2       7.9         0.4050633
```

`filter()` selects rows that match criteria

```
# get only the I. setosa specimens
filter(iris, Species == "setosa") |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
```

```
# filter on multiple criteria, treated as "AND"
filter(iris, Species == "setosa", Sepal.Length < 5) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          4.9         3.0          1.4         0.2  setosa
2          4.7         3.2          1.3         0.2  setosa
3          4.6         3.1          1.5         0.2  setosa
```

arrange() sorts rows according to values of one or more columns

```
# sort by Sepal.Length
arrange(iris, Sepal.Length) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          4.3         3.0          1.1         0.1  setosa
2          4.4         2.9          1.4         0.2  setosa
3          4.4         3.0          1.3         0.2  setosa
```

```
# sort on multiple columns: by Sepal.Length then by Petal.Length
arrange(iris, Sepal.Length, Petal.Length) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          4.3         3.0          1.1         0.1  setosa
2          4.4         3.0          1.3         0.2  setosa
3          4.4         3.2          1.3         0.2  setosa
```

```
# sort in descending order using helper desc()
arrange(iris, desc(Sepal.Length)) |> head(3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          7.9         3.8          6.4         2.0 virginica
2          7.7         3.8          6.7         2.2 virginica
3          7.7         2.6          6.9         2.3 virginica
```

summarize() transforms and collapses

summarize() applies functions to one or more variables (columns) in the data frame, reducing a vector of values to a single value and returning the results in a data frame

```
summarize(iris,
          avg.Sepal.Length = mean(Sepal.Length),
          avg.Petal.Length = mean(Petal.Length))
  avg.Sepal.Length avg.Petal.Length
1         5.843333            3.758
```

group_by() is used for conditioning (faceting) and transforming

```
# apply grouping
grouped.df <- group_by(iris, Species)

# summarize grouped data frame
summarize(grouped.df,
          avg.Sepal.Length = mean(Sepal.Length),
          avg.Petal.Length = mean(Petal.Length))
# A tibble: 3 x 3
  Species    avg.Sepal.Length avg.Petal.Length
  <fct>                 <dbl>             <dbl>
1 setosa                 5.01              1.46
2 versicolor             5.94              4.26
3 virginica              6.59              5.55
```

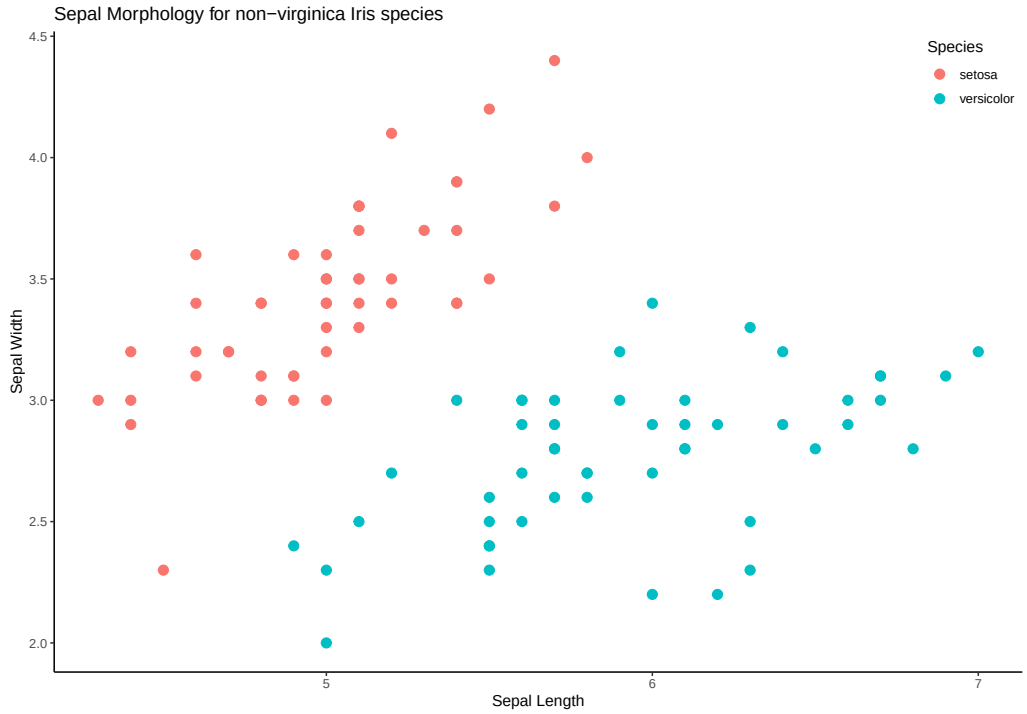
The dplyr functions are designed to work well with piping!

```
iris |>
  filter(Species != "virginica") |>
  group_by(Species) |>
  summarize(avg.Sepal.Length = mean(Sepal.Length),
            avg.Sepal.Width = mean(Sepal.Width))
# A tibble: 2 x 3
  Species    avg.Sepal.Length avg.Sepal.Width
  <fct>                 <dbl>           <dbl>
1 setosa                 5.01            3.43
2 versicolor             5.94            2.77
```

dplyr pipelines feed naturally into ggplot

But note that ggplot layers are "added" *not* piped:

```
iris |>
  filter(Species != "virginica") |>
  ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
    geom_point(size=3) +
    labs(x = "Sepal Length", y = "Sepal Width",
         title = "Sepal Morphology for non-virginica Iris species") +
    theme_classic() +
    theme(legend.justification = c(1, 1), legend.position = c(1, 1))
```



Other `dplyr` verbs of interest

Column manipulation:

- `rename()` - change column names
- `relocate()` - change column positions
- `pull()` - similar to `$`, returns a vector rather than a data frame

Row manipulation:

- `slice()` - index rows by integer locations.
  - `slice` variants: `slice_head()`, `slice_tail()`, `slice_min()`, `slice_max()`, `slice_sample()`
- `distinct()` - collapse data frames into rows that are unique; usually used with subset of columns

Group manipulation

- `count()` - counts observations in a specified groups