Advanced Data Manipulation and Modelling with `tidyverse`

Tidy Evaluation, Nested Data, and Model Building

Professor Alex

1 Introduction to Tidy Evaluation in `dplyr`

1.1 Overview

- Modern data manipulation in R with `tidyverse`
- Focus on tidy evaluation principles
- Practical applications in data science workflows

1.2 What We'll Cover Today

Part 1: Tidy Evaluation in `dplyr`

- Data masking
- Tidy selection
- Programming with `dplyr`

Part 2: Nested Data & Modelling

- Nested data frames
- Model building
- Result aggregation

2 Part 1: Tidy Evaluation in `dplyr`

2.1 What is Tidy Evaluation?

- **Tidy Evaluation** is a form of non-standard evaluation used in the `tidyverse`.
- It allows simpler expressions for data manipulation, making code more concise and readable.
- Two main concepts: **Data Masking** and **Tidy Selection**.

2.2 Why Use Tidy Evaluation?

- Reduces the need for `df$column` syntax; instead, use `column` directly.
- Enables smooth data exploration and efficient manipulation, especially in interactive analysis.
- Supports consistent and flexible workflows across multiple `dplyr` functions.

2.3 Practical Example: Tidy Evaluation

```
# Filtering without Tidy Evaluation
starwars[starwars$homeworld == "Naboo" & starwars$species == "Human", ]

# Filtering with Tidy Evaluation
starwars %>% dplyr::filter(homeworld == "Naboo", species == "Human")
```

1. Use the `$` operator to reference columns.
    - Not recommended for `dplyr` workflows.
    - Good when developing scripts or functions to avoid external dependencies.
2. Use the `dplyr::filter()` function with direct column references.
    - Cleaner and more readable.
    - Preferred method for `dplyr` workflows.

2.4 Overview of Data Masking

- **Data Masking** allows using data frame columns as if they were variables in the environment.
- Avoids repetitive references to the data frame.
- Key for efficient filtering, summarizing, and mutating.

2.5 Data Masking in Action

- Distinguishes between **env-variables** (programming variables) and **data-variables** (statistical variables).
- Env-variables are created with `<-`, while data-variables are within the data frame.

2.6 Practical Example: Data Masking

2.6.1 Env-Variables

```
# Using df as an env-variable
df <- tibble::tibble(x = runif(3), y = runif(3))
df %>% dplyr::mutate(z = x + y)
```

```
# A tibble: 3 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1 0.634 0.266 0.900
2 0.331 0.933 1.26
3 0.366 0.283 0.649
```

2.6.2 Data-Variables

```
# Filtering with Data Masking
# Here, "homeworld" and "species" are data-variables
starwars %>%
  dplyr::filter(homeworld == "Naboo" & species == "Human") %>%
  dplyr::select(name, homeworld, species)
```

```
# A tibble: 5 x 3
  name          homeworld species
  <chr>         <chr>     <chr>
1 Palpatine     Naboo     Human
2 Padmé Amidala Naboo     Human
3 Ric Olié      Naboo     Human
4 Quarsh Panaka Naboo     Human
5 Dormé         Naboo     Human
```

2.7 Tidy Selection

What is Tidy Selection?

- **Tidy Selection** is a set of tools provided by `tidyselect` package to easily select columns by name, type, or position.
- Allows for selecting columns dynamically within functions like
  - `dplyr::select()`
  - `dplyr::mutate()`
  - `dplyr::across()`

2.8 Tidy Selection Syntax

- Examples:
  - `tidyselect::starts_with()`
  - `tidyselect::ends_with()`
  - `tidyselect::where()`
- Can specify columns using multiple criteria within a single function.

2.9 Practical Example: Tidy Selection

2.9.1 Prefix

```
# Selecting columns that start with "c"
mtcars %>% select(starts_with("c"))
```

```
                  cyl carb
Mazda RX4           6    4
Mazda RX4 Wag       6    4
Datsun 710          4    1
Hornet 4 Drive      6    1
Hornet Sportabout   8    2
Valiant             6    1
Duster 360          8    4
Merc 240D           4    2
Merc 230            4    2
Merc 280            6    4
Merc 280C           6    4
Merc 450SE          8    3
Merc 450SL          8    3
Merc 450SLC         8    3
Cadillac Fleetwood  8    4
Lincoln Continental 8    4
Chrysler Imperial   8    4
Fiat 128            4    1
Honda Civic         4    2
Toyota Corolla      4    1
Toyota Corona       4    1
Dodge Challenger    8    2
```

```
AMC Javelin           8    2
Camaro Z28            8    4
Pontiac Firebird      8    2
Fiat X1-9             4    1
Porsche 914-2         4    2
Lotus Europa          4    2
Ford Pantera L        8    4
Ferrari Dino          6    6
Maserati Bora         8    8
Volvo 142E            4    2
```

2.9.2 Contains

```
# Selecting columns that contain a letter "t"
mtcars %>% select(contains("t"))
```

```
                    drat    wt
Mazda RX4           3.90 2.620
Mazda RX4 Wag       3.90 2.875
Datsun 710          3.85 2.320
Hornet 4 Drive      3.08 3.215
Hornet Sportabout   3.15 3.440
Valiant             2.76 3.460
Duster 360          3.21 3.570
Merc 240D           3.69 3.190
Merc 230            3.92 3.150
Merc 280            3.92 3.440
Merc 280C           3.92 3.440
Merc 450SE          3.07 4.070
Merc 450SL          3.07 3.730
Merc 450SLC         3.07 3.780
Cadillac Fleetwood  2.93 5.250
Lincoln Continental 3.00 5.424
Chrysler Imperial   3.23 5.345
Fiat 128            4.08 2.200
Honda Civic         4.93 1.615
Toyota Corolla      4.22 1.835
Toyota Corona       3.70 2.465
Dodge Challenger    2.76 3.520
AMC Javelin         3.15 3.435
Camaro Z28          3.73 3.840
Pontiac Firebird    3.08 3.845
Fiat X1-9           4.08 1.935
Porsche 914-2       4.43 2.140
```

```
Lotus Europa        3.77 1.513
Ford Pantera L      4.22 3.170
Ferrari Dino        3.62 2.770
Maserati Bora       3.54 3.570
Volvo 142E          4.11 2.780
```

2.9.3 Match

```
# Selecting columns that start with letters a-g
mtcars %>% select(matches("^[a-g]"))
```

```
                  cyl  disp drat am gear carb
Mazda RX4           6 160.0 3.90  1    4    4
Mazda RX4 Wag       6 160.0 3.90  1    4    4
Datsun 710          4 108.0 3.85  1    4    1
Hornet 4 Drive      6 258.0 3.08  0    3    1
Hornet Sportabout   8 360.0 3.15  0    3    2
Valiant             6 225.0 2.76  0    3    1
Duster 360          8 360.0 3.21  0    3    4
Merc 240D           4 146.7 3.69  0    4    2
Merc 230            4 140.8 3.92  0    4    2
Merc 280            6 167.6 3.92  0    4    4
Merc 280C           6 167.6 3.92  0    4    4
Merc 450SE          8 275.8 3.07  0    3    3
Merc 450SL          8 275.8 3.07  0    3    3
Merc 450SLC         8 275.8 3.07  0    3    3
Cadillac Fleetwood  8 472.0 2.93  0    3    4
Lincoln Continental 8 460.0 3.00  0    3    4
Chrysler Imperial   8 440.0 3.23  0    3    4
Fiat 128            4  78.7 4.08  1    4    1
Honda Civic         4  75.7 4.93  1    4    2
Toyota Corolla      4  71.1 4.22  1    4    1
Toyota Corona       4 120.1 3.70  0    3    1
Dodge Challenger    8 318.0 2.76  0    3    2
AMC Javelin         8 304.0 3.15  0    3    2
Camaro Z28          8 350.0 3.73  0    3    4
Pontiac Firebird    8 400.0 3.08  0    3    2
Fiat X1-9           4  79.0 4.08  1    4    1
Porsche 914-2       4 120.3 4.43  1    5    2
Lotus Europa        4  95.1 3.77  1    5    2
Ford Pantera L      8 351.0 4.22  1    5    4
Ferrari Dino        6 145.0 3.62  1    5    6
Maserati Bora       8 301.0 3.54  1    5    8
Volvo 142E          4 121.0 4.11  1    4    2
```

2.9.4 Type

```
# Selecting all numeric columns
mtcars %>% select(where(is.numeric))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

2.10 Tidy Selection: Common Selectors

1. Select variables that match a pattern
    - `starts_with()`, `ends_with()`, `contains()`, `matches()`, `num_range()`

2. Select variables from character vector
    - `all_of()`, `any_of()`
3. Select all variables or the last variable
    - `everything()`, `last_col()`
4. Select variables with a function
    - `where()`

2.11 Programming with `dplyr`

What is Indirection in `dplyr`?

- **Indirection** refers to using variables dynamically within `dplyr` functions.
- Useful in functions and loops, especially when column names are stored as strings or variables.

2.12 Techniques for Indirection

- Use `{{ }}` to embrace expressions within function arguments.
- `.data[[ ]]` helps when column names are in character vectors.

2.13 Practical Example: Indirection with Embracing

```
# Embracing with {{ }}
var_summary <- function(data, var) {
  data %>% dplyr::summarise(
    n = dplyr::n(),
    min = min({{ var }}),
    max = max({{ var }})
  )
}
mtcars %>%
  dplyr::group_by(cyl) %>%
  var_summary(mpg)
```

```
# A tibble: 3 x 4
    cyl     n   min   max
  <dbl> <int> <dbl> <dbl>
1     4    11  21.4  33.9
2     6     7  17.8  21.4
3     8    14  10.4  19.2
```

2.14 Across Multiple Columns

- `dplyr::across()` function allows applying operations across **multiple columns**.
- Useful for summarizing, mutating, and filtering multiple columns.

Syntax:

```
across(.cols, .fns, .names = NULL, .unpack = FALSE)
```

- `.cols`: `<tidy-select>` Columns to apply the function.
- `.fns`: Function to apply to the columns.
  - A function, e.g. `mean`
  - A `purrr`-style lambda function, e.g. `~mean(.x, na.rm = TRUE)`
  - A named list of functions, e.g. `list(mean = mean, n_miss = ~ sum(is.na(.x)))`

2.15 Practical Example: Using `across()`

2.15.1 Single Function

```
# Applying mean across multiple columns
vars <- c("mpg", "vs")
mtcars %>%
  dplyr::summarise(dplyr::across(
    tidyselect::all_of(vars),
    mean
  ))
```

```
       mpg     vs
1 20.09062 0.4375
```

2.15.2 Multiple Functions

```r
# Applying multiple functions across numeric columns
mtcars %>%
  dplyr::summarise(dplyr::across(
    tidyselect::where(is.numeric),
    list(avg = mean, std = sd)
  ))
```

```
   mpg_avg  mpg_std cyl_avg  cyl_std disp_avg disp_std   hp_avg   hp_std
1 20.09062 6.026948  6.1875 1.785922 230.7219 123.9387 146.6875 68.56287
  drat_avg  drat_std  wt_avg    wt_std qsec_avg qsec_std vs_avg    vs_std
1 3.596563 0.5346787 3.21725 0.9784574 17.84875 1.786943 0.4375 0.5040161
   am_avg    am_std gear_avg  gear_std carb_avg carb_std
1 0.40625 0.4989909   3.6875 0.7378041   2.8125   1.6152
```

2.15.3 String to Factor

```r
# Converting character columns to factors
starwars %>%
  mutate(across(where(is.character), as.factor)) %>%
  select(name, where(is.factor))
```

```
# A tibble: 87 x 8
   name           hair_color skin_color eye_color sex   gender homeworld species
   <fct>          <fct>      <fct>      <fct>     <fct> <fct>  <fct>     <fct>
 1 Luke Skywalker blond      fair       blue      male  mascu~ Tatooine  Human
 2 C-3PO          <NA>       gold       yellow    none  mascu~ Tatooine  Droid
 3 R2-D2          <NA>       white, bl~ red       none  mascu~ Naboo     Droid
 4 Darth Vader    none       white      yellow    male  mascu~ Tatooine  Human
 5 Leia Organa    brown      light      brown     fema~ femin~ Alderaan  Human
 6 Owen Lars      brown, gr~ light      blue      male  mascu~ Tatooine  Human
 7 Beru Whitesun~ brown      light      blue      fema~ femin~ Tatooine  Human
 8 R5-D4          <NA>       white, red red       none  mascu~ Tatooine  Droid
 9 Biggs Darklig~ black      light      brown     male  mascu~ Tatooine  Human
10 Obi-Wan Kenobi auburn, w~ fair       blue-gray male  mascu~ Stewjon   Human
# i 77 more rows
```

2.16 Flexible Summary Functions

- Functions can accept variable column expressions to compute summaries dynamically.
- `dplyr::across()` is key for creating flexible summaries over multiple columns.

Custom Summarization Syntax

- Use `dplyr::across()` inside `dplyr::summarise()` to calculate statistics on chosen columns.
- Control output column names using `.names` within `dplyr::across()`.

2.17 Practical Example: Custom Summarization

2.17.1 Embracing {{ }}

```
# Function to calculate means of numeric columns
summarise_mean <- function(data, vars) {
  data %>%
    summarise(
      n = n(),
      across({{ vars }}, ~ mean(.x, na.rm = TRUE), .names = "{.col}_avg")
    )
}
mtcars %>% group_by(cyl) %>% summarise_mean(where(is.numeric))
```

```
# A tibble: 3 x 13
    cyl     n mpg_avg disp_avg hp_avg drat_avg wt_avg qsec_avg vs_avg am_avg
  <dbl> <int>   <dbl>    <dbl>  <dbl>    <dbl>  <dbl>    <dbl>  <dbl>  <dbl>
1     4    11    26.7     105.   82.6     4.07   2.29     19.1  0.909  0.727
2     6     7    19.7     183.  122.      3.59   3.12     18.0  0.571  0.429
3     8    14    15.1     353.  209.      3.23   4.00     16.8  0      0.143
# i 3 more variables: gear_avg <dbl>, carb_avg <dbl>, n_avg <dbl>
```

2.17.2 Using ...

```
# Function to calculate means of numeric columns
summarise_mean <- function(.data, ...) {
  .data %>%
    summarise(
      n = n(),
      across(c(...), ~ mean(.x, na.rm = TRUE), .names = "{.col}_avg")
    )
}
mtcars %>% group_by(cyl) %>% summarise_mean(mpg, disp, hp, wt)
```

```
# A tibble: 3 x 6
    cyl     n mpg_avg disp_avg hp_avg wt_avg
  <dbl> <int>   <dbl>    <dbl>  <dbl>  <dbl>
1     4    11    26.7     105.   82.6   2.29
2     6     7    19.7     183.  122.    3.12
3     8    14    15.1     353.  209.    4.00
```

2.18 Name Injection in Variable Names

Dynamic Name Creation with Name Injection

- `:=` allows creating dynamic names for variables in a pipeline.
- Useful when column names are generated programmatically or from environment variables.

2.19 Practical Example: Name Injection

```
# Dynamic name assignment
name <- "dynamic_var"
tibble("{name}" := 2)
```

```
# A tibble: 1 x 1
  dynamic_var
        <dbl>
1           2
```

```
# Using dynamic name within a function
my_df <- function(x) {
  tibble("{{ x }}_2" := x * 2)
}
my_var <- 10
my_df(my_var)
```

```
# A tibble: 1 x 1
  my_var_2
     <dbl>
1       20
```

3 Part 2: Nested Data & Modelling

3.1 What is a Nested Data Frame?

- A **Nested Data Frame** contains one or more columns as lists of data frames.
- Useful for organizing grouped data or subsetting for complex transformations and modeling.

3.2 Creating Nested Data Frames

- Use `tidyr::nest()` to group columns into list-columns.
- Each list entry represents a data subset, allowing efficient grouping and nested operations.

```
# Creating a nested data frame by grouping
df <- tribble(
  ~g, ~x, ~y,
   1,  1,  2,
   2,  4,  6,
   2,  5,  7,
   3, 10,  NA
)
df %>% tidyr::nest(data = c(x, y))
```

```
# A tibble: 3 x 2
      g data
  <dbl> <list>
1     1 <tibble [1 x 2]>
2     2 <tibble [2 x 2]>
3     3 <tibble [1 x 2]>
```

3.3 Practical Example: (Un)Nesting

3.3.1 Original Data

```
# Display some columns
starwars %>% select(species, name, height, mass, gender, homeworld)
```

```
# A tibble: 87 x 6
   species name               height  mass gender    homeworld
   <chr>   <chr>               <int> <dbl> <chr>     <chr>
 1 Human   Luke Skywalker        172    77 masculine Tatooine
```

```
 2 Droid   C-3PO                167    75 masculine Tatooine
 3 Droid   R2-D2                 96    32 masculine Naboo
 4 Human   Darth Vader          202   136 masculine Tatooine
 5 Human   Leia Organa          150    49 feminine  Alderaan
 6 Human   Owen Lars            178   120 masculine Tatooine
 7 Human   Beru Whitesun Lars   165    75 feminine  Tatooine
 8 Droid   R5-D4                 97    32 masculine Tatooine
 9 Human   Biggs Darklighter     183    84 masculine Tatooine
10 Human   Obi-Wan Kenobi        182    77 masculine Stewjon
# i 77 more rows
```

3.3.2 Nested Data

```r
# Nesting data by species
starwars %>%
  dplyr::group_by(species) %>%
  tidyr::nest()
```

```
# A tibble: 38 x 2
# Groups:   species [38]
   species        data
   <chr>          <list>
 1 Human          <tibble [35 x 13]>
 2 Droid          <tibble [6 x 13]>
 3 Wookiee        <tibble [2 x 13]>
 4 Rodian         <tibble [1 x 13]>
 5 Hutt           <tibble [1 x 13]>
 6 <NA>           <tibble [4 x 13]>
 7 Yoda's species <tibble [1 x 13]>
 8 Trandoshan     <tibble [1 x 13]>
 9 Mon Calamari   <tibble [1 x 13]>
10 Ewok           <tibble [1 x 13]>
# i 28 more rows
```

3.4 Using Nested Data for Modeling

- Nested data is ideal for **fitting models across groups**.
- Each subset in a list-column can hold an individual model or analysis result.

3.5 Practical Example: Nesting `mtcars` for Modeling

- Group `mtcars` by `cyl` (number of cylinders) and nest data within each group.

```
# Nesting mtcars data by cylinder
mtcars_nested <- mtcars %>%
  group_by(cyl) %>%
  nest()
mtcars_nested
```

```
# A tibble: 3 x 2
# Groups:   cyl [3]
    cyl data
  <dbl> <list>
1     6 <tibble [7 x 10]>
2     4 <tibble [11 x 10]>
3     8 <tibble [14 x 10]>
```

3.6 Building Models on Nested Data

- Use `purrr::map()` to fit models within each nested group.
- Store each model in a new list-column.

```
# Fit models to each subset
mtcars_nested <- mtcars_nested %>%
  mutate(model = map(data, \(df) lm(mpg ~ wt, data = df)))
mtcars_nested
```

```
# A tibble: 3 x 3
# Groups:   cyl [3]
    cyl data               model
  <dbl> <list>             <list>
1     6 <tibble [7 x 10]>  <lm>
2     4 <tibble [11 x 10]> <lm>
3     8 <tibble [14 x 10]> <lm>
```

3.7 Extracting Predictions

- Generate predictions for each model using `map()` to apply `predict()`.

```
# Generate predictions
mtcars_nested <- mtcars_nested %>%
  mutate(predictions = map(model, predict))
mtcars_nested
```

```
# A tibble: 3 x 4
# Groups:   cyl [3]
    cyl data               model  predictions
  <dbl> <list>             <list> <list>
1     6 <tibble [7 x 10]>  <lm>   <dbl [7]>
2     4 <tibble [11 x 10]> <lm>   <dbl [11]>
3     8 <tibble [14 x 10]> <lm>   <dbl [14]>
```

3.8 Applying `broom` for Model Summaries

Available Functions:

- `broom::tidy()`
- `broom::glance()`
- `broom::augment()`

Common Use Cases:

- Model coefficients (coefficients, p-values)
- Model statistics (R-squared, AIC)
- Predictions and residuals

- `broom` provides functions like `tidy()`, `augment()`, and `glance()` to convert model outputs to data frames.
- This allows for consistent analysis and visualization of multiple models' outputs.
- For linear models, `lm()`, see:
  - `broom::tidy.lm()`
  - `broom::glance.lm()`
  - `broom::augment.lm()`

3.9 Practical Example: Tidying with `broom::tidy()`

```r
# Tidying model results
library(broom)
mtcars_nested <- mtcars_nested %>%
  mutate(tidy_results = purrr::map(model, broom::tidy))

# Unnesting to view all results in a flat format
mtcars_nested %>% unnest(tidy_results)
```

```
# A tibble: 6 x 9
# Groups:   cyl [3]
    cyl data     model  predictions term    estimate std.error statistic p.value
  <dbl> <list>   <list> <list>      <chr>      <dbl>     <dbl>     <dbl>   <dbl>
1     6 <tibble> <lm>   <dbl [7]>   (Inter~    28.4      4.18       6.79 1.05e-3
2     6 <tibble> <lm>   <dbl [7]>   wt         -2.78     1.33      -2.08 9.18e-2
3     4 <tibble> <lm>   <dbl [11]>  (Inter~    39.6      4.35       9.10 7.77e-6
4     4 <tibble> <lm>   <dbl [11]>  wt         -5.65     1.85      -3.05 1.37e-2
5     8 <tibble> <lm>   <dbl [14]>  (Inter~    23.9      3.01       7.94 4.05e-6
6     8 <tibble> <lm>   <dbl [14]>  wt         -2.19     0.739     -2.97 1.18e-2
```

3.10 Advanced Tidying: `augment()` and `glance()`

- **`augment()`**: Adds model predictions and residuals to the original data.
- **`glance()`**: Summarizes model fit statistics.

```r
# Adding model summaries
mtcars_nested <- mtcars_nested %>%
  mutate(
    augmented = map(model, augment),
    glance_results = map(model, glance)
  )
```

3.11 Model Fit Summaries

3.11.1 Data

```r
mtcars_nested %>%
  select(cyl, data, model, tidy_results, augmented, glance_results)
```

```
# A tibble: 3 x 6
# Groups:   cyl [3]
    cyl data               model  tidy_results     augmented glance_results
  <dbl> <list>             <list> <list>           <list>    <list>
1     6 <tibble [7 x 10]>  <lm>   <tibble [2 x 5]> <tibble>  <tibble [1 x 12]>
2     4 <tibble [11 x 10]> <lm>   <tibble [2 x 5]> <tibble>  <tibble [1 x 12]>
3     8 <tibble [14 x 10]> <lm>   <tibble [2 x 5]> <tibble>  <tibble [1 x 12]>
```

3.11.2 Tidy

```
mtcars_nested %>%
  select(cyl, data, model, tidy_results) %>%
  unnest(tidy_results)
```

```
# A tibble: 6 x 8
# Groups:   cyl [3]
    cyl data               model  term      estimate std.error statistic p.value
  <dbl> <list>             <list> <chr>        <dbl>     <dbl>     <dbl>   <dbl>
1     6 <tibble [7 x 10]>  <lm>   (Interce~    28.4      4.18       6.79 1.05e-3
2     6 <tibble [7 x 10]>  <lm>   wt           -2.78     1.33      -2.08 9.18e-2
3     4 <tibble [11 x 10]> <lm>   (Interce~    39.6      4.35       9.10 7.77e-6
4     4 <tibble [11 x 10]> <lm>   wt           -5.65     1.85      -3.05 1.37e-2
5     8 <tibble [14 x 10]> <lm>   (Interce~    23.9      3.01       7.94 4.05e-6
6     8 <tibble [14 x 10]> <lm>   wt           -2.19     0.739     -2.97 1.18e-2
```

3.11.3 Augmented

```
mtcars_nested %>%
  select(cyl, data, model, augmented) %>%
  unnest(augmented)
```

```
# A tibble: 32 x 11
# Groups:   cyl [3]
     cyl data     model    mpg    wt .fitted .resid   .hat .sigma .cooksd
   <dbl> <list>   <list> <dbl> <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
 1     6 <tibble> <lm>    21    2.62    21.1 -0.125 0.467   1.30  0.00947
 2     6 <tibble> <lm>    21    2.88    20.4  0.584 0.220   1.26  0.0454
 3     6 <tibble> <lm>    21.4  3.22    19.5  1.93  0.155   0.772 0.299
 4     6 <tibble> <lm>    18.1  3.46    18.8 -0.690 0.297   1.24  0.105
 5     6 <tibble> <lm>    19.2  3.44    18.8  0.355 0.280   1.29  0.0250
 6     6 <tibble> <lm>    17.8  3.44    18.8 -1.05  0.280   1.15  0.217
 7     6 <tibble> <lm>    19.7  2.77    20.7 -1.01  0.301   1.15  0.231
```

```
 8     4 <tibble> <lm>    22.8  2.32    26.5 -3.67  0.0913  3.26  0.0670
 9     4 <tibble> <lm>    24.4  3.19    21.6  2.84  0.343   3.31  0.289
10     4 <tibble> <lm>    22.8  3.15    21.8  1.02  0.321   3.51  0.0325
# i 22 more rows
# i 1 more variable: .std.resid <dbl>
```

3.11.4 Glance

```
mtcars_nested %>%
  select(cyl, data, model, glance_results) %>%
  unnest(glance_results)
```

```
# A tibble: 3 x 15
# Groups:   cyl [3]
    cyl data     model  r.squared adj.r.squared sigma statistic p.value    df
  <dbl> <list>   <list>     <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>
1     6 <tibble> <lm>       0.465         0.357  1.17      4.34  0.0918     1
2     4 <tibble> <lm>       0.509         0.454  3.33      9.32  0.0137     1
3     8 <tibble> <lm>       0.423         0.375  2.02      8.80  0.0118     1
# i 6 more variables: logLik <dbl>, AIC <dbl>, BIC <dbl>, deviance <dbl>,
#   df.residual <int>, nobs <int>
```

3.12 Practical Application: Model Comparison

- Easily compare models across groups with tidied outputs.
- Sort or filter by fit statistics, visualize parameter estimates, or compare residuals.

```
# Example: Sorting by R-squared from glance results
mtcars_nested %>%
  select(cyl, data, model, glance_results) %>%
  unnest(glance_results) %>%
  arrange(desc(r.squared))
```

```
# A tibble: 3 x 15
# Groups:   cyl [3]
    cyl data     model  r.squared adj.r.squared sigma statistic p.value    df
  <dbl> <list>   <list>     <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>
1     4 <tibble> <lm>       0.509         0.454  3.33      9.32  0.0137     1
2     6 <tibble> <lm>       0.465         0.357  1.17      4.34  0.0918     1
3     8 <tibble> <lm>       0.423         0.375  2.02      8.80  0.0118     1
# i 6 more variables: logLik <dbl>, AIC <dbl>, BIC <dbl>, deviance <dbl>,
#   df.residual <int>, nobs <int>
```

4 Model Selection: Comparing Different Models

4.1 Linear Regression Models

- Create multiple linear models on `mtcars` data for each cylinder group.
- Fit models and store results in a nested data frame.
- Nest the `mtcars` data by cylinder group:

```
mtcars_nested <- mtcars %>%
  group_by(cyl) %>%
  nest()
mtcars_nested
```

```
# A tibble: 3 x 2
# Groups:   cyl [3]
    cyl data
  <dbl> <list>
1     6 <tibble [7 x 10]>
2     4 <tibble [11 x 10]>
3     8 <tibble [14 x 10]>
```

4.2 Fit Linear Models

- Consider building linear models for each cylinder group:
  - `mpg ~ wt`
  - `mpg ~ wt + hp`
  - `mpg ~ wt + hp + qsec`

```
# Fit linear models
mtcars_lm <- mtcars_nested %>%
  mutate(
    model_wt = map(data, \(df) lm(mpg ~ wt, data = df)),
    model_wt_hp = map(data, \(df) lm(mpg ~ wt + hp, data = df)),
    model_wt_hp_qsec = map(data, \(df) lm(mpg ~ wt + hp + qsec, data = df))
  )
mtcars_lm
```

```
# A tibble: 3 x 5
# Groups:   cyl [3]
    cyl data               model_wt model_wt_hp model_wt_hp_qsec
  <dbl> <list>             <list>   <list>      <list>
1     6 <tibble [7 x 10]>  <lm>     <lm>        <lm>
2     4 <tibble [11 x 10]> <lm>     <lm>        <lm>
3     8 <tibble [14 x 10]> <lm>     <lm>        <lm>
```

4.3 All Model Statistics

- Use `pivot_longer()` to reshape the data for comparison.
- Using `broom::glance()` to extract model statistics.

```
# Extract model statistics
mtcars_glance <- mtcars_lm %>%
  pivot_longer(
    cols = starts_with("model"),
    names_to = "model",
    values_to = "model_fit"
  ) %>%
  mutate(
    model_stats = map(model_fit, glance)
  )
mtcars_glance
```

```
# A tibble: 9 x 5
# Groups:   cyl [3]
    cyl data              model           model_fit model_stats
  <dbl> <list>            <chr>           <list>    <list>
1     6 <tibble [7 x 10]> model_wt        <lm>      <tibble [1 x 12]>
2     6 <tibble [7 x 10]> model_wt_hp     <lm>      <tibble [1 x 12]>
3     6 <tibble [7 x 10]> model_wt_hp_qsec <lm>     <tibble [1 x 12]>
4     4 <tibble [11 x 10]> model_wt       <lm>      <tibble [1 x 12]>
5     4 <tibble [11 x 10]> model_wt_hp    <lm>      <tibble [1 x 12]>
6     4 <tibble [11 x 10]> model_wt_hp_qsec <lm>    <tibble [1 x 12]>
7     8 <tibble [14 x 10]> model_wt       <lm>      <tibble [1 x 12]>
8     8 <tibble [14 x 10]> model_wt_hp    <lm>      <tibble [1 x 12]>
9     8 <tibble [14 x 10]> model_wt_hp_qsec <lm>    <tibble [1 x 12]>
```

4.4 Unnest and Compare Models

- Use `unnest()` to extract model statistics for comparison.

```
# Pivot longer for comparison
mtcars_lm_stats <- mtcars_glance %>%
  unnest(model_stats)
mtcars_lm_stats
```

```
# A tibble: 9 x 16
# Groups:   cyl [3]
    cyl data     model model_fit r.squared adj.r.squared sigma statistic p.value
  <dbl> <list>   <chr> <list>        <dbl>         <dbl> <dbl>     <dbl>   <dbl>
```

```
1     6 <tibble> mode~ <lm>          0.465         0.357  1.17      4.34  0.0918
2     6 <tibble> mode~ <lm>          0.589         0.383  1.14      2.87  0.169
3     6 <tibble> mode~ <lm>          0.602         0.204  1.30      1.51  0.371
4     4 <tibble> mode~ <lm>          0.509         0.454  3.33      9.32  0.0137
5     4 <tibble> mode~ <lm>          0.681         0.601  2.85      8.53  0.0104
6     4 <tibble> mode~ <lm>          0.701         0.572  2.95      5.46  0.0299
7     8 <tibble> mode~ <lm>          0.423         0.375  2.02      8.80  0.0118
8     8 <tibble> mode~ <lm>          0.497         0.406  1.97      5.44  0.0228
9     8 <tibble> mode~ <lm>          0.507         0.360  2.05      3.43  0.0601
# i 7 more variables: df <dbl>, logLik <dbl>, AIC <dbl>, BIC <dbl>,
#   deviance <dbl>, df.residual <int>, nobs <int>
```

4.5 Model Selection

- Select the best model based on, for example, lowest BIC values.
- Use `dplyr::slice_min()` to select the row with the minimum BIC value.

```
# Select best model by BIC
mtcars_lm_stats %>%
  slice_min(BIC) %>%
  relocate(BIC, .after = model_fit)
```

```
# A tibble: 3 x 16
# Groups:   cyl [3]
    cyl data     model   model_fit   BIC r.squared adj.r.squared sigma statistic
  <dbl> <list>   <chr>   <list>    <dbl>     <dbl>         <dbl> <dbl>     <dbl>
1     4 <tibble> model_~ <lm>       60.3     0.681         0.601  2.85      8.53
2     6 <tibble> model_~ <lm>       25.5     0.465         0.357  1.17      4.34
3     8 <tibble> model_~ <lm>       65.2     0.423         0.375  2.02      8.80
# i 7 more variables: p.value <dbl>, df <dbl>, logLik <dbl>, AIC <dbl>,
#   deviance <dbl>, df.residual <int>, nobs <int>
```

4.6 Summary and Best Practices

- Use tidy evaluation for cleaner code
- Leverage `nest`'ed data frames for grouped analyses
- Apply `broom` for consistent model outputs
- Consider computational efficiency and readability

4.7 Resources

Documentation

- tidyselect: Selection Helpers
- Nested data
- dplyr programming
- broom documentation
- broom and dplyr

Further Reading

- R for Data Science
- Advanced R by Hadley Wickham
- tidyverse blog posts