Tidyverse

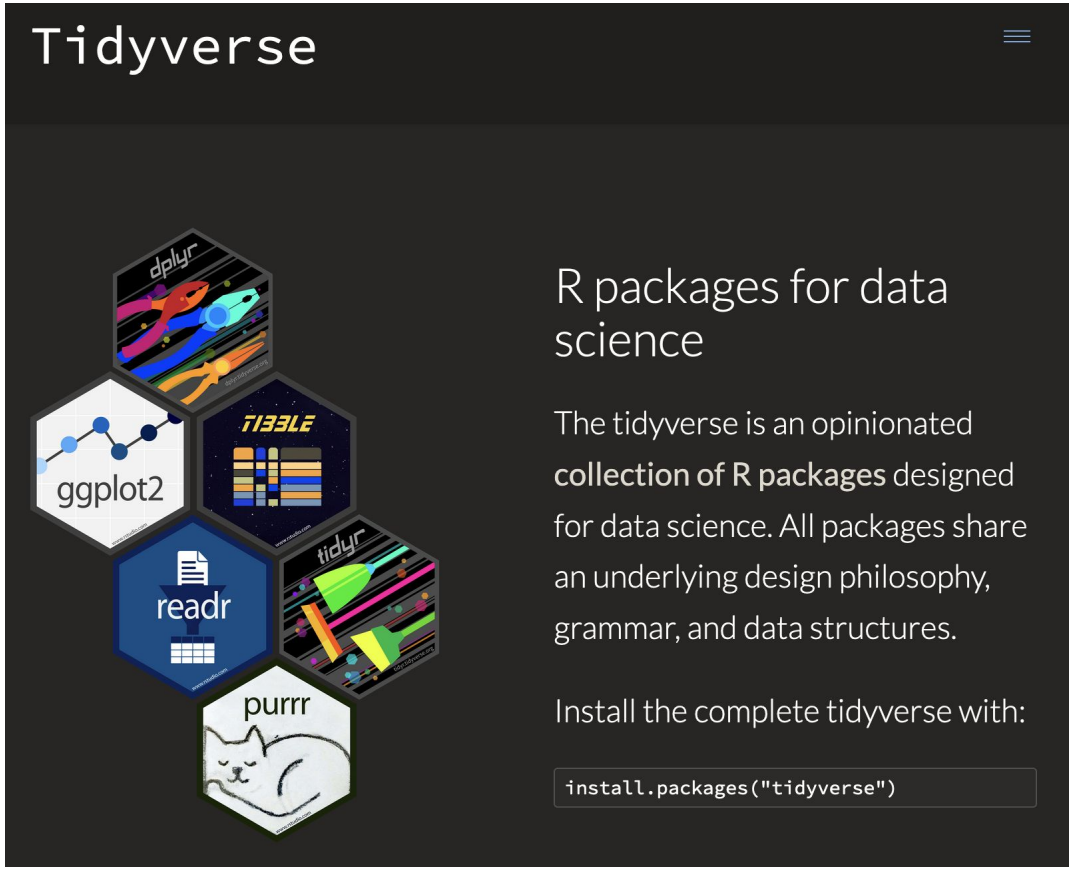
Announcements

- Final Project Pre-Proposals due next week (**Feb 26th**)
  - Make sure to schedule an appointment with me for consultations!
- Class Slack is up, so feel free to join!
  - Invite link is in Class Information section on Moodle
- No DataCamp next week to account for short period between classes.

1. Background

What is the Tidyverse?

- A collection of packages aiming to create “Tidy” data in R.
- dplyr: Data Manipulation
- tidyr: More Data Manipulation!
- ggplot2: Custom plotting



What is “Tidy” Data

- Each Variable has its own Column
- Each Observation has its own Row
- Each Type of Observational Unit Forms a Table
  - If you have different types of observational units, they should be stored in separate tables.
- Variables are Homogeneous
  - Each column should contain values of the same type.
  - For example, a column should not mix numbers and text.
- Consistent Variable Names
- One Variable Per Column
- Avoids Data Duplication

Spot the difference

Untidy

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

Tidy

city	large	small
New York	23	14
London	22	16
Beijing	121	56

Why do we want “Tidy” Data

- **Simplicity and Accessibility**: Tidy data formats make it easier to manipulate, visualize, and model your data.
- **Facilitates Data Transformation and Aggregation**: Tidy datasets are easier to transform, subset, and summarize using tools like dplyr and tidyr in R.
- **Compatibility with Data Analysis Tools and Practices**: Tidy data is often required or preferred by various data analysis tools, libraries, and practices because it provides a clear and consistent structure for data processing.

Why is this useful in spatial analysis?

- According to the 2020 Anaconda State of Data Science report, data scientists spend around **26%** of their time wrangling data.
- Knowing how to manipulate large data sets in R is something otherwise impractical in applications like Excel (unless you're a finance Excel guru).
- Data is useless until it's manipulated into a useful form.

PREFACE:
YOU DO NOT HAVE TO MEMORIZE THESE COMMANDS BY HEART

This is to expose you to data manipulation concepts to keep in mind as you work.

Install & Use Tidyverse Packages

```
> install.packages(tidyverse)

> require(tidyverse)
```

2. dplyr

DPLYR Data Manipulation - Cheat Sheet

- **select()** to select or drop variables
- **rename()** to rename columns
- **filter()** to filter the data
  - filter(data, column == “___”)
- **mutate()** to add or alter columns
- **group_by()** & **summarize()** allows you to specify variables in which to group your dataset, and then summarize aggregations of each group
  - After summarizing you’ll only have one observation for each group
- **arrange()** to order your data in the way you want
- **inner_join()** / **left_join() / right_join() / full_join()**

select()

- select() is used to subset columns in a data frame

name	age	height
Alice	24	5.5
Bob	22	6.0

```
# Selecting only the 'name' and 'age' columns
selected_df <- select(df, name, age)
```

name	age
Alice	24
Bob	22

rename()

- rename() is great for renaming columns to be more descriptive!
- > rename(data,
- > NewCol = “Old Col”,
- > NewCol2 = “Old Col 2”,
- > …
- >)

```
# Let's first rename some of the columns for clarity
state.x77_renamed <- state.x77_df %>%
  rename(
    Population = `Population`,
    PerCapita_Income = `Income`,
    Illiteracy_Percent = `Illiteracy`,
    LifeExp = `Life Exp`,
    MurderRate_Per100k = `Murder`,
    HSGrad_Percent = `HS Grad`,
    Frost = `Frost`,
    Area_SqMi = `Area`
  )
```

	Population	PerCapita_Income	Illiteracy_Percent	LifeExp	MurderRate_Per100k	HSGrad_Percent	Frost	Area_SqMi
Alabama	3615	3624	2.1	69.05	15.1	41.3	20	50708
Alaska	365	6315	1.5	69.31	11.3	66.7	152	566432
Arizona	2212	4530	1.8	70.55	7.8	58.1	15	113417
Arkansas	2110	3378	1.9	70.66	10.1	39.9	65	51945
California	21198	5114	1.1	71.71	10.3	62.6	20	156361
Colorado	2541	4884	0.7	72.06	6.8	63.9	166	103766

filter() - Logic Background

- 1 == 1
  - > TRUE
- 2 > 5
  - > FALSE
- ```
  > vec = c(1, 5, 8, 9, 48, 16)
  > vec < 10
  [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE
  ```
- ```
  > vec = c(1, 5, 8, 9, 48, 16)
  > vec_logic = vec < 10
  > vec_subset = vec[vec_logic]
  > vec_subset
  [1] 1 5 8 9
  ```

filter()

- filter(dataframe, logical) allows you to filter data based on some logic

```r
# Filter rows where speed is greater than 15
fast_cars <- filter(cars, speed > 15)
head(fast_cars)
```

```
  speed dist
1    16   32
2    16   40
3    17   32
4    17   40
5    17   50
6    18   42
```

Equivalent to:

```r
fast_cars = cars[cars$speed > 15,]
```

Pipes %>% - Basics

- Allow you to string together multiple functions
- Creates more condensed, more readable, and more efficient code
- Pipes function by utilizing the element before the pipe as the input for functions after the pipe.

variable = function(data, param) # basic
variable = data %>% function(param)

- Note: You don't have to specify data in function as it is piped into the function.

variable = data %>%
function1(param) %>%
function2(param) %>% …

%in%

```
> unique(gapminder$country)
  [1] Afghanistan              Albania
  [3] Algeria                  Angola
  [5] Argentina                Australia
  [7] Austria                  Bahrain
  [9] Bangladesh               Belgium
 [11] Benin                    Bolivia
 [13] Bosnia and Herzegovina   Botswana
 [15] Brazil                   Bulgaria
 [17] Burkina Faso             Burundi
 [19] Cambodia                 Cameroon
 [21] Canada                   Central African Republic
 [23] Chad                     Chile
 [25] China                    Colombia
 [27] Comoros                  Congo, Dem. Rep.
 [29] Congo, Rep.              Costa Rica
 [31] Cote d'Ivoire            Croatia
 [33] Cuba                     Czech Republic
```

```
gapminder_filt <- gapminder %>%
  filter(country %in% c("Albania", "Belgium"))
```

country	continent	year	lifeExp	pop	gdpPercap
Albania	Europe	1952	55.230	1282697	1601.056
Albania	Europe	1957	59.280	1476505	1942.284
Albania	Europe	1962	64.820	1728137	2312.889
Albania	Europe	1967	66.220	1984060	2760.197
Albania	Europe	1972	67.690	2263554	3313.422
Albania	Europe	1977	68.930	2509048	3533.004
Albania	Europe	1982	70.420	2780097	3630.881
Albania	Europe	1987	72.000	3075321	3738.933
Albania	Europe	1992	71.581	3326498	2497.438
Albania	Europe	1997	72.950	3428038	3193.055
Albania	Europe	2002	75.651	3508512	4604.212
Albania	Europe	2007	76.423	3600523	5937.030
Belgium	Europe	1952	68.000	8730405	8343.105
Belgium	Europe	1957	69.240	8989111	9714.961
Belgium	Europe	1962	70.250	9218400	10991.207
Belgium	Europe	1967	70.940	9556500	13149.041
Belgium	Europe	1972	71.440	9709100	16672.144
Belgium	Europe	1977	72.800	9821800	19117.974
Belgium	Europe	1982	73.930	9856303	20979.846
Belgium	Europe	1987	75.350	9870200	22525.563
Belgium	Europe	1992	76.460	10045622	25575.571
Belgium	Europe	1997	77.530	10199787	27561.197
Belgium	Europe	2002	78.320	10311970	30485.884
Belgium	Europe	2007	79.441	10392226	33692.605

mutate()

- mutate(dataframe, new_column = logic)

```
# Add a new column that calculates the speed in km/h (from mph)
cars_km <- mutate(cars, speed_km = speed * 1.60934)
head(cars_km)
```

	speed	dist	speed_km
1	4	2	6.43736
2	4	10	6.43736
3	7	4	11.26538
4	7	22	11.26538
5	8	16	12.87472
6	9	10	14.48406

Equivalent to:

```
cars_km = cars
cars_km$speed_km = cars$speed * 1.60934
```

group_by() & summarize()

- group_by() is used to split a data frame into groups based on one or more variables.
  - It changes the grouping of the data frame without altering the data itself.
- summarize() allows you to run statistics on your data

```r
# Group by speed and calculate the average distance for each speed
avg_distance_by_speed <- cars %>%
  group_by(speed) %>%
  summarize(avg_distance = mean(dist))
head(avg_distance_by_speed)
```

speed <dbl>	avg_distance <dbl>
4	6
7	13
8	16
9	1Ø
1Ø	26
11	22.5

Differences Between mutate() & summarize()

- **Data Reduction**:
  - summarize() reduces data to fewer rows
  - mutate() keeps the same number of rows
- **Purpose**:
  - summarize() is for aggregation
  - mutate() is for modification or creation of columns without aggregation
- **Output**:
  - summarize() is typically used for statistical summaries
  - mutate() is used for adding features or modifying a dataset while retaining its original structure

arrange()

- arrange() allows you to change the order of your data rows based on column.
- By default arrange() sorts by ascending
- You must use **desc()** to get descending order

```
# We'll use arrange() to sort the data based on Life Expectancy
state.x77__ascending_LE <- arrange(state.x77_renamed, LifeExp)
state.x77__descending_LE <- arrange(state.x77_renamed, desc(LifeExp))
```

Let’s look at our newly arranged data!

```
> head(state.x77__ascending_LE)
               Population PerCapita_Income Illiteracy_Percent LifeExp MurderRate_Per100k HSGrad_Percent Frost Area_SqMi
South Carolina       2816             3635                2.3   67.96               11.6           37.8    65      30225
Mississippi          2341             3098                2.4   68.09               12.5           41.0    50      47296
Georgia              4931             4091                2.0   68.54               13.9           40.6    60      58073
Louisiana            3806             3545                2.8   68.76               13.2           42.2    12      44930
Nevada                590             5149                0.5   69.03               11.5           65.2   188     109889
Alabama              3615             3624                2.1   69.05               15.1           41.3    20      50708
```

```
> head(state.x77__descending_LE)
             Population PerCapita_Income Illiteracy_Percent LifeExp MurderRate_Per100k HSGrad_Percent Frost Area_SqMi
Hawaii              868             4963                1.9   73.60               6.2           61.9     0      6425
Minnesota          3921             4675                0.6   72.96               2.3           57.6   160     79289
Utah               1203             4022                0.6   72.90               4.5           67.3   137     82096
North Dakota        637             5087                0.8   72.78               1.4           50.3   186     69273
Nebraska           1544             4508                0.6   72.60               2.9           59.3   139     76483
Kansas             2280             4669                0.6   72.58               4.5           59.9   114     81787
```

Joins

- **inner_join()** : Only the records that have matching values in both A and B.
- **left_join()** : All records from A, along with any matching records in B.
- **right_join()** : All records from B, along with any matching records in A.
- **full_join()** : All records from both A and B, with NULLs where there are no matches.

```
head(cars)
```

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10


```
# Inner join with avg_distance_by_speed data frame
cars_with_avg_distance <- inner_join(cars,
                                     avg_distance_by_speed,
                                     by = "speed")

head(cars_with_avg_distance)
```

speed	dist	avg_distance
4	2	6
4	10	6
7	4	13
7	22	13
8	16	16
9	10	10

Pipes %>% - Advanced

- Test Score Example:
  - result <- df %>%
  - **select**(id, score) %>% # 1. Select columns
  - **filter**(score > 60) %>% # 2. Filter rows with score > 60
  - **arrange**(desc(score)) %>% # 3. Arrange in descending order
  - **mutate**(pass = score >= 70) %>% # 4. Add a new column 'pass'
  - **summarize**(
  - average_score = mean(score), # 5. Calculate average score
  - pass_count = sum(pass)) # 6. Count how many passed

3. tidyr

TidyR Data Manipulation - Cheat Sheet

- separate() splits a column by a character string separator
- unite() ‘unites’ multiple columns into a single column
- pivot_longer() collapses multiple columns into two columns
- pivot_wider() generates multiple columns from two columns

separate()

Separate splits a column by a character string separator.

```
separate(storms, date, c("year", "month", "day"), sep = "-")
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

Credit to Jeroen Boeye from DataCamp

unite()

Unite unites columns into a single column.

```
unite(storms2, "date", year, month, day, sep = "-")
```

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

Credit to Jeroen Boeye from DataCamp

Wide vs. Long Data

- **Wide Data**
  - Characterized by having many columns representing different variables, often observed at different times or conditions.
  - It's called "wide" because it can have a *large number of columns*, especially if there are many time points or conditions.
- **Long Data**
  - One column per variable, another for it’s value
  - It's called "long" because it can have a *large number of rows*, often significantly, especially with many time points or conditions.

pivot_longer()

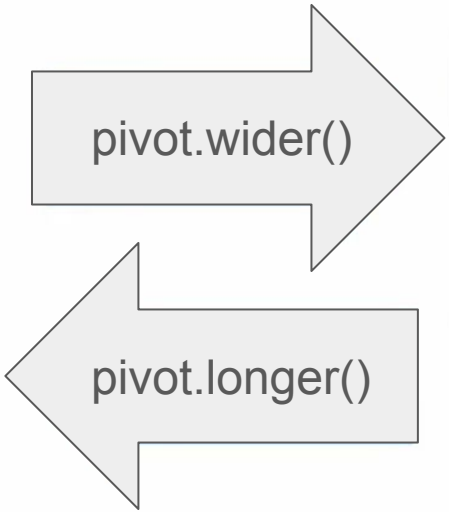
- pivot_longer() converts data from wide to long format
- **pivot_longer(data, cols, names_from, values_from)**
  - cols: Columns in wide format data that need to be condensed
  - names_to: The name of the new column in the long format data that will contain the consolidated column names from the wide format.
  - values_to: The name of the new column in the long format data that will contain the values from the consolidated columns in the wide format.

pivot_wider()

- pivot_wider() converts data from long to wide format
- **pivot_wider(data, names_from, values_from)**
  - names_from: Column in long format data that contains categoricals
  - values_from: Column in long format data that contains values

```
df_wide <- df_long %>%
  pivot_wider(
        names_from = size,
        values_from = amount)
```

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



city	large	small
New York	23	14
London	22	16
Beijing	121	56

```
df_long <- df_wide %>%
  pivot_longer(
        cols = c(large,small),
        names_to = "size",
        values_to = "amount")
```

Credit to Jeroen Boeye from DataCamp

Removing Variables from Transformations

- The - sign before a variable indicates that this variable should be left out of the gathering process, meaning it should not be gathered into key-value pairs but rather should remain as a separate column.
  - Example: the city column is ok as is, so we want to ignore it.

city	large	small
New York	23	14
London	22	16
Beijing	121	56

```
df_long <- df_wide %>%
  pivot_longer(
    cols = -city,
    names_to = "size",
    values_to = "amount")
```

- Alternatively you can list out columns with: cols = c(large, small)

gather() & spread() —> pivot.longer() & pivot.wider()

- gather() & spread() have been depreciated and replaced by the pivot functions!
- If you see gather() or spread(), just know that they do the same thing as the pivot functions!

4. Dealing with incomplete data

Useful functions for working with incomplete data

- na.rm = TRUE
- na.omit()
- replace_na()
- fill()
- complete()

Why is incomplete data worrying?

- Many analytics will not run properly with missing data
- Excessively incomplete data can be rendered useless if not handled properly
  - Example: Removing all incomplete rows may leave you with too few rows for statistical significance.
- NA data is rarely ever truly NA in the real world, which can impose bias

Function Options - na.rm=TRUE

- na.rm=TRUE is a very common parameter for getting functions to ignore NA
- Examples
  - mean(data, na.rm=TRUE)
  - min(data, na.rm=TRUE)
  - max(data, na.rm=TRUE)
  - sum(data, na.rm=TRUE)
  - …

na.omit()

- Omits an NA rows in your dataset
- If a row has ANY NA columns it will be entirely removed
- Example:
  - na.omit(df)

replace_na()

- Useful for specific replacements of NA
- Example:
  - Converting from NA -> NULL
    - data <- data %>%
    - mutate(Col1 = **replace_na(Col1, NULL)**)
  - Raster conversion from NA -> -9999
    - raster <- raster %>%
    - **replace_na(list(Val = -9999)**)

fill()

- Fills missing values in selected columns using the next or previous entry.
  - This is useful where values are not repeated, and are only recorded when they change.
- fill(data, columns, .direction = c("down", "up", "downup", "updown"))
  - **data**: The dataframe you're working with.
  - **columns**: The columns in which you want to fill missing values.
  - **.direction**: Specifies the direction to fill missing values:
    - "down": Forward fill (fills NA with the last observed non-NA value).
    - "up": Backward fill (fills NA with the next observed non-NA value).
    - "downup": First applies forward fill, then backward fill on the result.
    - "updown": First applies backward fill, then forward fill on the result.

fill() - Example

```
airquality_filled <- airquality %>%
  fill(Ozone, .direction = "down")
```

airquality dataset

Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6
23	299	8.6	65	5	7
19	99	13.8	59	5	8
8	19	20.1	61	5	9
NA	194	8.6	69	5	10
7	NA	6.9	74	5	11
16	256	9.7	69	5	12

fill()

Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
18	NA	14.3	56	5	5
28	NA	14.9	66	5	6
23	299	8.6	65	5	7
19	99	13.8	59	5	8
8	19	20.1	61	5	9
8	194	8.6	69	5	10
7	NA	6.9	74	5	11
16	256	9.7	69	5	12

Note: the . in .direction is helps to differentiate it from any column that might be named direction in your data.

complete()

- Ensures that your dataset contains a row for every combination of specified key variables.
- Adds missing combinations as new rows with NA values in non-key columns.
- Useful for preparing data for analysis that requires complete datasets.
- complete(data, ..., fill = list())
  - **data**: The input dataset. This is the dataset you want to ensure has complete cases.
  - **...**: Columns or expressions defining the groups.
  - **fill** (optional): A named list that provides values to use instead of NA for missing combinations in non-key columns. This allows you to specify default values for newly created rows.

complete() - Example

```
completed_sales <- sales %>%
  complete(Store, Product, Day, fill = list(Sales = 0))
```

Store	Product	Sales	Day
Store A	Product 1	10	Monday
Store A	Product 2	15	Tuesday
Store B	Product 1	12	Monday
Store B	Product 3	8	Wednesday
Store C	Product 2	15	Tuesday

Store	Product	Day	Sales
Store A	Product 1	Monday	10
Store A	Product 1	Tuesday	0
Store A	Product 1	Wednesday	0
Store A	Product 2	Monday	0
Store A	Product 2	Tuesday	15
Store A	Product 2	Wednesday	0
Store A	Product 3	Monday	0
Store A	Product 3	Tuesday	0
Store A	Product 3	Wednesday	0
Store B	Product 1	Monday	12
Store B	Product 1	Tuesday	0
Store B	Product 1	Wednesday	0
Store B	Product 2	Monday	0
Store B	Product 2	Tuesday	0
Store B	Product 2	Wednesday	0
Store B	Product 3	Monday	0
Store B	Product 3	Tuesday	0
Store B	Product 3	Wednesday	8
Store C	Product 1	Monday	0
Store C	Product 1	Tuesday	0
Store C	Product 1	Wednesday	0
Store C	Product 2	Monday	0
Store C	Product 2	Tuesday	15
Store C	Product 2	Wednesday	0
Store C	Product 3	Monday	0
Store C	Product 3	Tuesday	0
Store C	Product 3	Wednesday	0

Thanks!