Pipelines and workflows

Abhijit Dasgupta

BIOF 339

BIOF 339: Practical R

Pipes in the tidyverse

Pipes

We've seen two types of pipes in R.

The pipe operator %>% from the magrittr package

```
library(tidyverse) # includes magrittr
library(palmerpenguins)

penguins %>%
  group_by(species) %>%
  mutate(across(bill_length_mm:body_mass_g,
                function(x) replace_na(x, mean(x, na
  ungroup() %>%
  summarise(across(bill_length_mm:body_mass_g, media
```

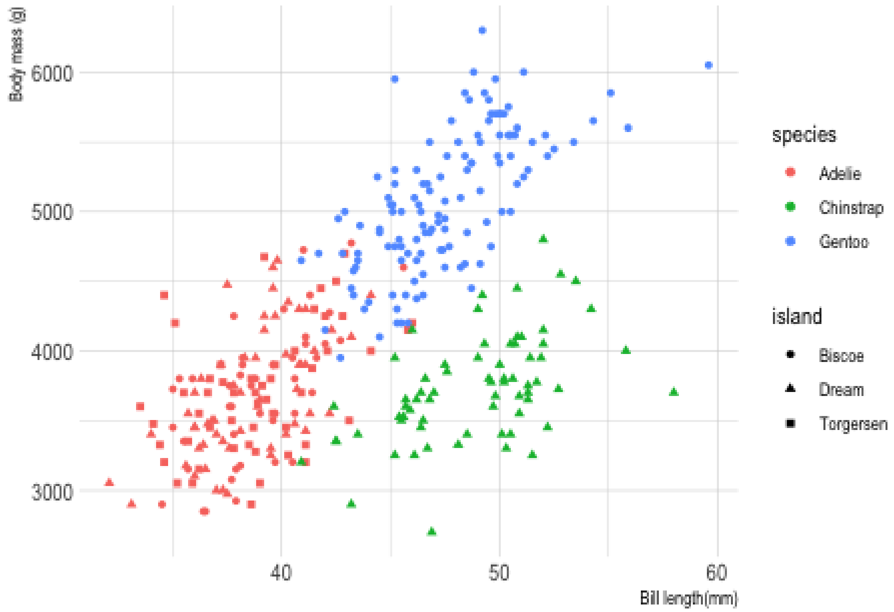
The + symbol used as a pipe-like operator in ggplot2

```
ggplot(penguins,
       aes(x = bill_length_mm,
           y = body_mass_g))+
  geom_point(aes(color = species, shape = island))
```

Pipes

You can combine the two pipes into a workflow to create a visualization

```
penguins %>%
  group_by(species) %>%
  mutate(across(bill_length_mm:body_mass_g,
                function(x) replace_na(x, mean(x, na
  ungroup() %>%
  ggplot(aes(x = bill_length_mm,
             y = body_mass_g)) +
    geom_point(aes(shape = island,
                   color = species))+
    labs(x = 'Bill length(mm)',
         y = 'Body mass (g)') +
    hrbrthemes::theme_ipsum()
```



The ggplot pipe has to be at the end of the workflow. Also note, we're not adding the data argument to `ggplot` since it is tidyverse-compatible and slots the end of the previous pipe into the `data` argument

Rowwise operations

The dplyr package allows you to do rowwise operations much more easily than before within a pipe using the `rowwise` function. For example

```
mpg %>%
  select(manufacturer, year, cty, hwy) %>%
  rowwise() %>%
  mutate(avg_mpg = mean(c(hwy, cty)))
```

```
# A tibble: 234 × 5
# Rowwise:
   manufacturer  year   cty   hwy avg_mpg
   <chr>        <int> <int> <int>   <dbl>
 1 audi          1999    18    29    23.5
 2 audi          1999    21    29    25
 3 audi          2008    20    31    25.5
 4 audi          2008    21    30    25.5
 5 audi          1999    16    26    21
 6 audi          1999    18    26    22
 7 audi          2008    18    27    22.5
 8 audi          1999    18    26    22
 9 audi          1999    16    25    20.5
10 audi          2008    20    28    24
# … with 224 more rows
```

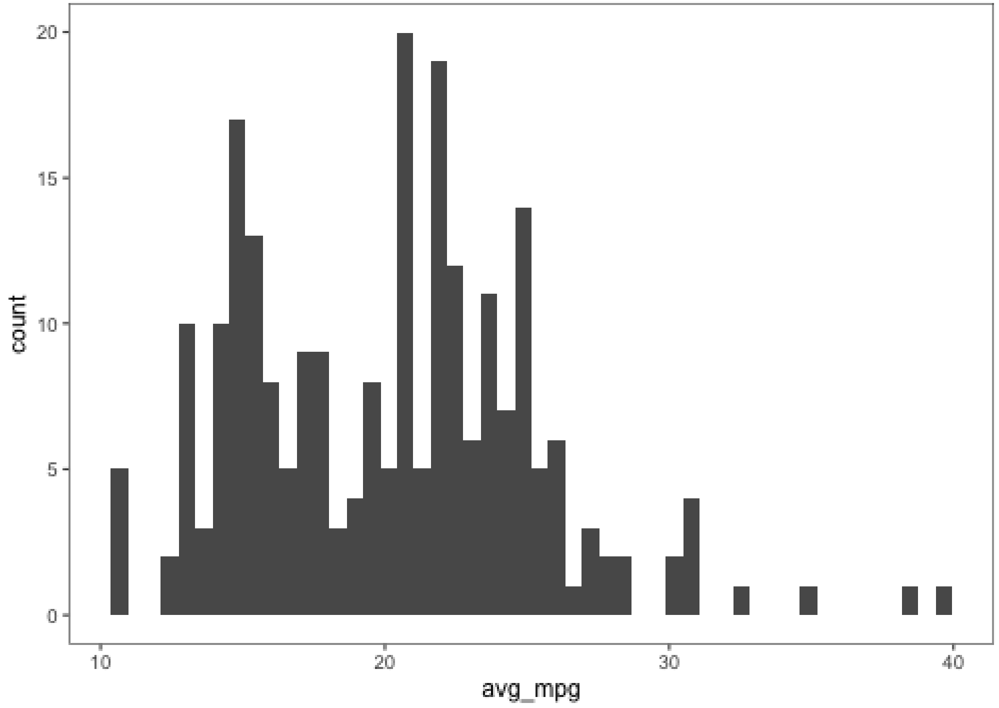
The `rowwise` function creates groups, one per row, and allows operations to occur along rows and across columns.

> What would the result be if you omitted the `rowwise` function in the pipe?

Rowwise operations

If you want to continue the pipe to incorporate the more traditonal column-wise operations, you need to use `ungroup` before proceeding

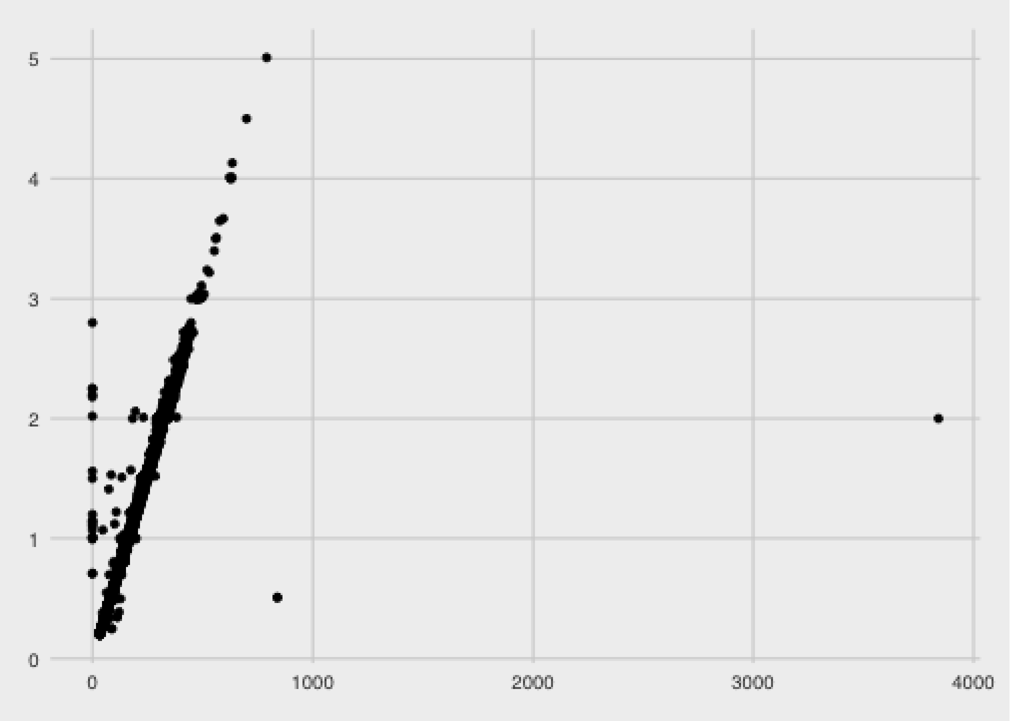
```
mpg %>%
  select(manufacturer, year, cty, hwy) %>%
  rowwise() %>%
  mutate(avg_mpg = mean(c(hwy, cty))) %>%
  ungroup() %>%
  ggplot(aes(x = avg_mpg)) +
    geom_histogram(bins = 50)+
    ggthemes::theme_few()
```



Rowwise operations

There are some nice shortcuts, in line with the `select` function, even with rowwise operations

```
diamonds %>%
  select(carat, x:z) %>%
  rowwise() %>%
  mutate(vol = prod(c_across(x:z))) %>%
  ungroup() %>%
  ggplot(aes(x = vol,
             y = carat))+
    geom_point() +
    ggthemes::theme_fivethirtyeight()
```



Much more details about the possibilities of the `rowwise` function are available here

BIOF 339: Practical R

Prepping data for modeling

Recipes

The idea of the recipes package is to define a recipe or blueprint that can be used to sequentially define the encodings and preprocessing of the data (i.e. “feature engineering”)

This is done in the context of supervised modeling, e.g. regression, decision trees

The idea is to define the dependent and independent variables, and then creating a pipeline to modify the independent variables through various statistical procedures.

Recipes

We'll start with the credit data in the modeldata package

```
library(recipes)
library(modeldata)
data("credit_data")

glimpse(credit_data)
```

```
Rows: 4,454
Columns: 14
$ Status    <fct> good, good, bad, good, good, good, good, good, good, bad, go…
$ Seniority <int> 9, 17, 10, 0, 0, 1, 29, 9, 0, 0, 6, 7, 8, 19, 0, 0, 15, 33, …
$ Home      <fct> rent, rent, owner, rent, rent, owner, owner, parents, owner,…
$ Time      <int> 60, 60, 36, 60, 36, 60, 60, 12, 60, 48, 48, 36, 60, 36, 18, …
$ Age       <int> 30, 58, 46, 24, 26, 36, 44, 27, 32, 41, 34, 29, 30, 37, 21, …
$ Marital   <fct> married, widow, married, single, single, married, married, s…
$ Records   <fct> no, no, yes, no, no, no, no, no, no, no, no, no, no, no, yes…
$ Job       <fct> freelance, fixed, freelance, fixed, fixed, fixed, fixed, fix…
$ Expenses  <int> 73, 48, 90, 63, 46, 75, 75, 35, 90, 90, 60, 60, 75, 75, 35, …
$ Income    <int> 129, 131, 200, 182, 107, 214, 125, 80, 107, 80, 125, 121, 19…
$ Assets    <int> 0, 0, 3000, 2500, 0, 3500, 10000, 0, 15000, 0, 4000, 3000, 5…
$ Debt      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2500, 260, 0, 0, 0, 2000…
$ Amount    <int> 800, 1000, 2000, 900, 310, 650, 1600, 200, 1200, 1200, 1150,…
$ Price     <int> 846, 1658, 2985, 1325, 910, 1645, 1800, 1093, 1957, 1468, 15…
```

Recipes

Create an initial recipe based on the model that will be fit

```
rec <- recipe(Status ~ Seniority + Time + Age + Records, data = credit_data)
```

```
rec
```

```
Data Recipe

Inputs:

      role #variables
   outcome          1
 predictor          4
```

```
summary(rec, original=TRUE)
```

```
# A tibble: 5 × 4
  variable   type    role      source  
  <chr>      <chr>   <chr>     <chr>   
1 Seniority  numeric predictor original
2 Time       numeric predictor original
3 Age        numeric predictor original
4 Records    nominal predictor original
5 Status     nominal outcome   original
```

Recipes

Add a step to convert nominal variables into dummies

```
(dummied <- rec %>% step_dummy(Records))
```

```
Data Recipe

Inputs:

      role #variables
   outcome          1
 predictor          4

Operations:

Dummy variables from Records
```

Then apply it to your data

```
dummied <- prep(dummied, training = credit_data)
with_dummy <- bake(dummied, new_data = credit_data)
head(with_dummy)
```

```
# A tibble: 6 × 5
  Seniority  Time   Age Status Records_yes
      <int> <int> <int> <fct>        <dbl>
1         9    60    30 good             0
2        17    60    58 good             0
3        10    36    46 bad              1
4         0    60    24 good             0
5         0    36    26 good             0
6         1    60    36 good             0
```

Recipes

The recipes package provides a rich variety of data steps that can be used to prepare a data set.

```
iris_recipe <- iris %>%
  recipe(Species ~ .) %>%
  step_corr(all_predictors()) %>%
  step_center(all_predictors(), -all_outcomes()) %>%
  step_scale(all_predictors() , -all_outcomes()) %>%
  prep()
iris_recipe
```

```
Data Recipe

Inputs:

      role #variables
   outcome          1
 predictor          4

Training data contained 150 data points and no missing data.

Operations:

Correlation filter removed Petal.Length [trained]
Centering for Sepal.Length, Sepal.Width, Petal.Width [trained]
Scaling for Sepal.Length, Sepal.Width, Petal.Width [trained]
```

Recipes

This recipe can then be applied to the same or a different dataset

```
iris1 <- bake(iris_recipe, iris)
glimpse(iris1)
```

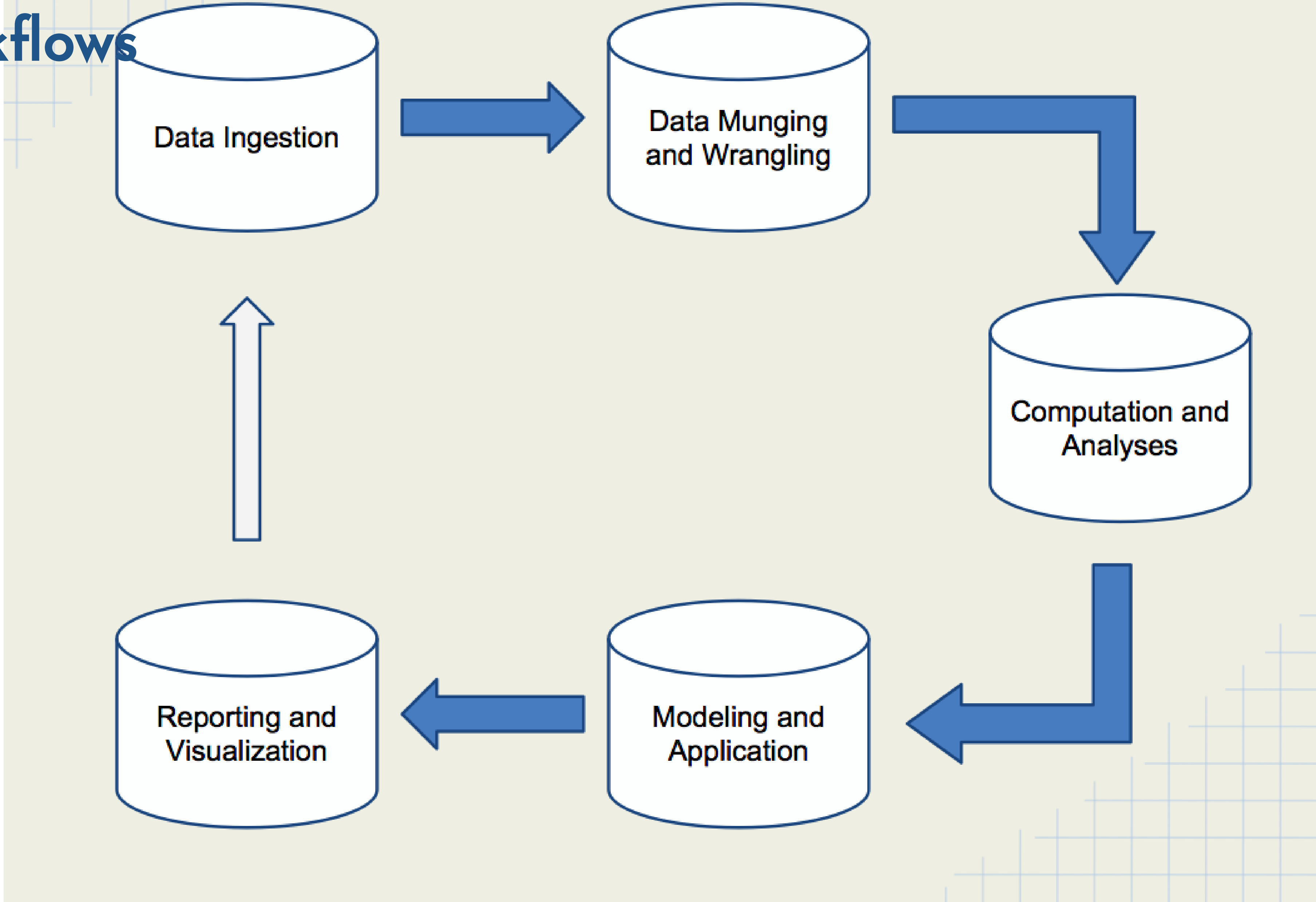
```
Rows: 150
Columns: 4
$ Sepal.Length <dbl> -0.89767388, -1.13920048, -1.38072709, -1.50149039, -1.01…
$ Sepal.Width  <dbl> 1.01560199, -0.13153881, 0.32731751, 0.09788935, 1.245030…
$ Petal.Width  <dbl> -1.3110521, -1.3110521, -1.3110521, -1.3110521, -1.311052…
$ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s…
```

You can go into more details at tidymodels.org, with a nice introduction here

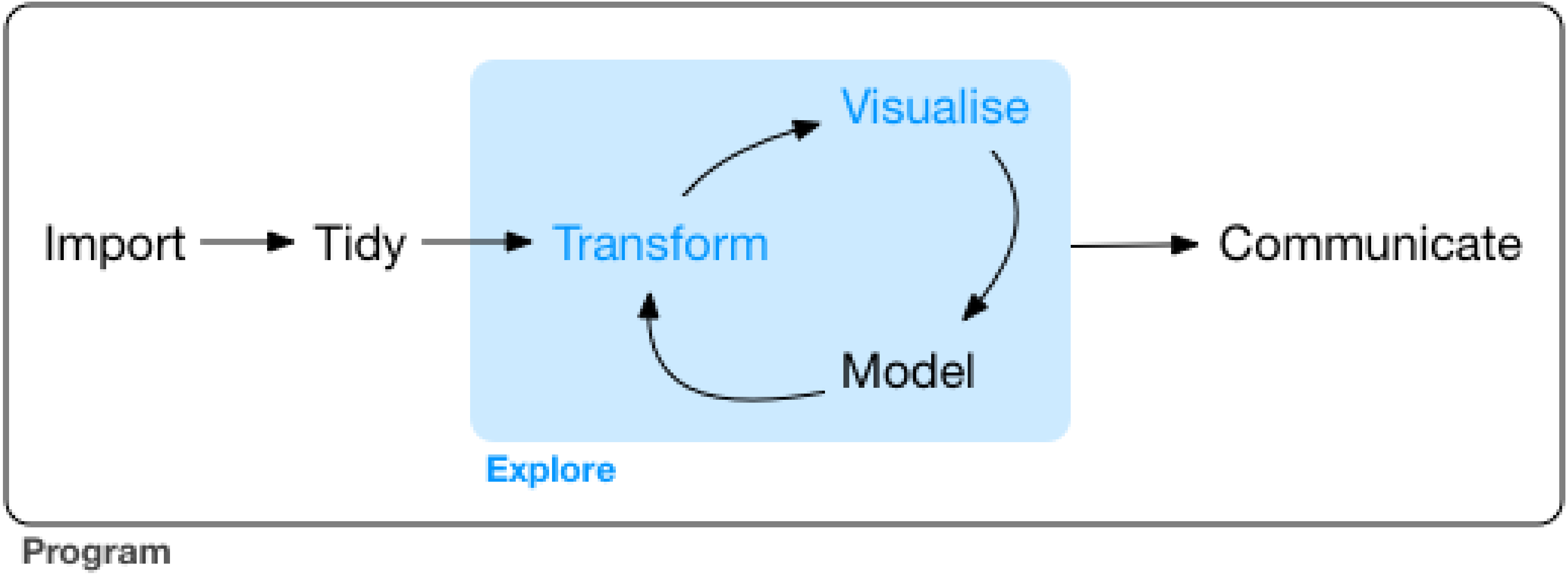
BIOF 339: Practical R

Workflows

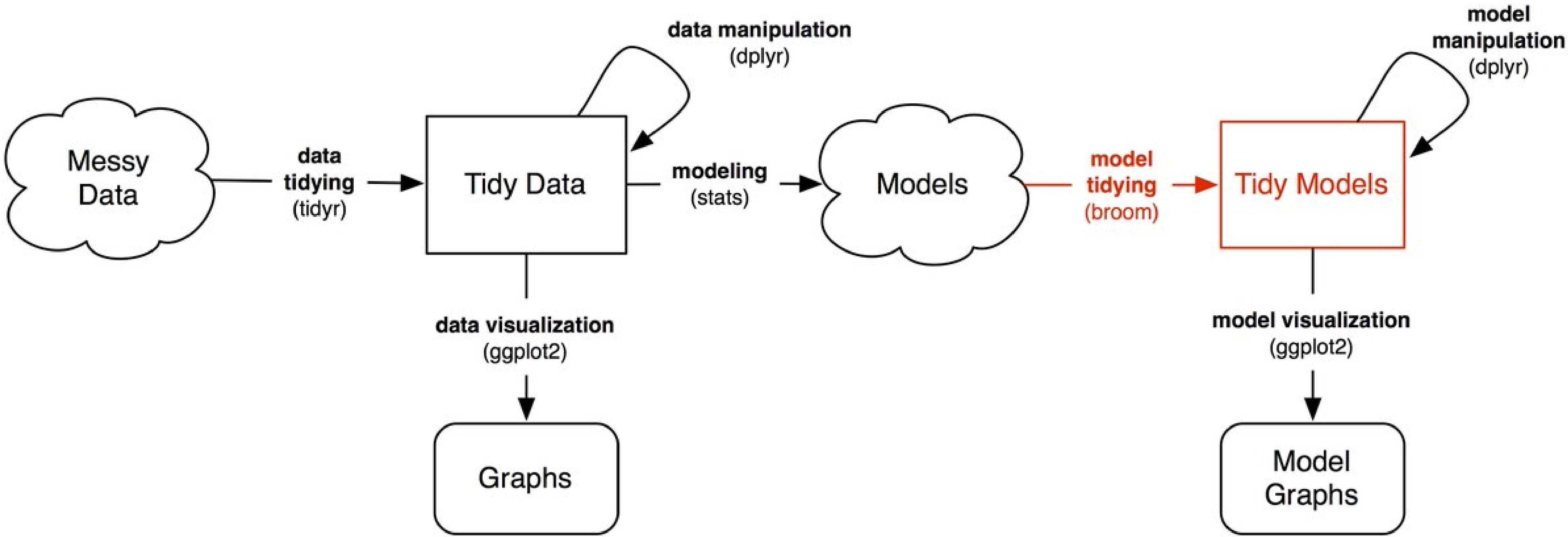
Workflows


Data Ingestion
Data Munging and Wrangling
Computation and Analyses
Modeling and Application
Reporting and Visualization

Workflows


Import
Tidy
Transform
Visualise
Model
Communicate
Explore
Program

Workflows


Messy Data
data tidying (tidyr)
Tidy Data
data manipulation (dplyr)
modeling (stats)
Models
model tidying (broom)
Tidy Models
model manipulation (dplyr)
data visualization (ggplot2)
Graphs
model visualization (ggplot2)
Model Graphs

Workflows

- Create one script file for each node in your workflow
- Save intermediate data or objects using `saveRDS` so that
    - they can be imported quickly by the next step
    - Each link in the chain can be checked and verified
- You can summarize your entire workflow within one script:

```
source('01-ingest.R')
source('02-munge.R')
source('03-exploreviz.R')
source('04-eda.R')
source('05-models.R')
source('06-results.R')
```

Workflows

A personal story

I wrote a paper using R Markdown with a reasonable pipeline for data analyses, modeling and visualization

Output to Word for submission to a journal

Three months later, reviews came in asking for using updated data

Changed the data at the beginning of my workflow, ran the workflow, and had revised manuscript in 10 minutes.

Quickest turnaround ever!!

Workflows

Some ideas (*Efficient Programming* by Gillespie and Lovelace)

1. Start without writing code but with a clear mind and perhaps a pen and paper. This will ensure you keep your objectives at the forefront of your mind, without getting lost in the technology.
2. Make a plan. The size and nature will depend on the project but timelines, resources and ‘chunking’ the work will make you more effective when you start.
3. Select the packages you will use for implementing the plan early. Minutes spent researching and selecting from the available options could save hours in the future.
4. Document your work at every stage; work can only be effective if it’s communicated clearly and code can only be efficiently understood if it’s commented.
5. Make your entire workflow as reproducible as possible. knitr can help with this in the phase of documentation.