

Introduction to the tidyverse

Chirag Lakhani/Chirag Patel

02/01/2018

R and the tidyverse

1. What is R?

- ▶ R is an open source programming language and software environment for statistical computing
- ▶ It consists of base R syntax and an ecosystem of R packages for advanced computation
- ▶ Most R packages are created and maintained by volunteers

2. What is the Tidyverse?

- ▶ The tidyverse is an ecosystem of packages within R meant to simplify data analysis and visualization
- ▶ Originally conceived of and developed by Hadley Wickham but has grown into a thriving community of developers

The relationship between different components of the tidyverse

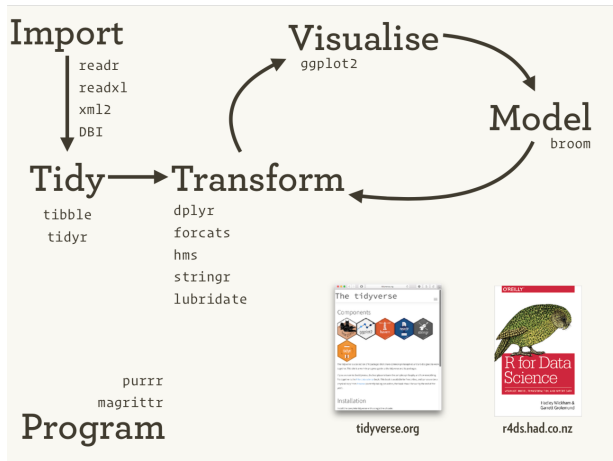


Figure 1: Packages within the Tidyverse

Plan for today's lecture

In today's lecture we assume that data has been loaded on tools for analyzing data, namely:

1. dplyr: used to manipulate data
2. magrittr: piping operation that makes data transformation/manipulation more readable
3. tidyr: used to make tidy data (method of storing data for future analysis)
4. ggplot2: used to develop visualizations
5. broom: used to convert statistical models into tidy data

Dataset that will be used in this tutorial

We will use the **iris** dataset as a running example throughout this lecture

```
kable(iris %>% sample_n(6), row.names = FALSE)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.8	1.5	0.3	setosa
5.6	2.7	4.2	1.3	versicolor
5.0	3.5	1.6	0.6	setosa
4.6	3.4	1.4	0.3	setosa
6.1	2.6	5.6	1.4	virginica
5.9	3.0	4.2	1.5	versicolor

The verbs of dplyr

dplyr is used to manipulate and manage data, it abstracts most common data manipulation tasks into verbs (6 most common below):

1. filter: filters out rows according to some conditions
2. arrange: reorders rows according to some conditions
3. select: selects a subset of columns
4. mutate: adds a new column as a function of existing columns
5. summarize: collapses a data frame to a single row
6. group_by: breaks a data frame into groups of rows

See Data Transformation Cheatsheet for more data manipulation verbs

The concept of piping of data

- ▶ We can compose dplyr verbs together using a piping construct (inspired by functional programming) to create very complicated data manipulation tasks.
- ▶ **Pipes** use the pipe operator where output from one function/operation will be input into the next

Magrittr (%>%)

Piping is denoted by the symbol %>%. One can chain various verbs together using %>%

x %>% f(y) means that x is 'piped' into the function f(x,y)

```
head(iris, n=2)
```

```
##      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
```

```
iris %>% head( n=2)
```

```
##      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
```


Example: Using filter

```
##### Picking rows with filter() #####
```

```
### Base R equivalent
```

```
iris[iris$Species == "virginica",]
```

```
### using dplyr::filter()
```

```
filter(iris, Species == "virginica")
```

```
# Equivalent code with %>% pipe
```

```
iris %>% filter(Species == "virginica")
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
6.0	2.2	5.0	1.5	virginica
5.8	2.8	5.1	2.4	virginica
7.7	3.0	6.1	2.3	virginica
4.9	2.5	4.5	1.7	virginica

Example: Using filter with two conditions

```
iris %>%  
  filter(Species == "virginica", Sepal.Length > 7.5)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
7.6	3.0	6.6	2.1	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica
7.7	2.8	6.7	2.0	virginica
7.9	3.8	6.4	2.0	virginica
7.7	3.0	6.1	2.3	virginica

Example: Using select

```
iris %>% select(Species, Petal.Length)
```

Species	Petal.Length
setosa	1.4
setosa	1.4
setosa	1.3
setosa	1.5
setosa	1.4
setosa	1.7
setosa	1.4
setosa	1.5
setosa	1.4
setosa	1.5
setosa	1.5
setosa	1.6
setosa	1.4
setosa	1.1

Example: Using unselect

```
iris %>% select(-Species)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
4.8	3.0	1.4	0.1
4.3	3.0	1.1	0.1

Example: Combining filter and select

```
iris %>%  
  filter(Species == "virginica", Sepal.Width > 3.5) %>%  
  select(Petal.Width)
```

Petal.Width

2.5

2.2

2.0

Example: Using mutate

```
iris %>%  
  mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%  
  select(-Sepal.Length, -Sepal.Width)
```

Petal.Length	Petal.Width	Species	Sepal.Area
1.4	0.2	setosa	17.85
1.4	0.2	setosa	14.70
1.3	0.2	setosa	15.04
1.5	0.2	setosa	14.26
1.4	0.2	setosa	18.00
1.7	0.4	setosa	21.06
1.4	0.3	setosa	15.64
1.5	0.2	setosa	17.00
1.4	0.2	setosa	12.76
1.5	0.1	setosa	15.19
1.5	0.2	setosa	19.98
1.6	0.2	setosa	16.32

Arrange

```
iris %>%  
  mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%  
  filter(Sepal.Area < 15) %>%  
  arrange(Sepal.Area) %>%  
  select(-Sepal.Length, -Sepal.Width)
```

Petal.Length	Petal.Width	Species	Sepal.Area
3.5	1.0	versicolor	10.00
1.3	0.3	setosa	10.35
3.3	1.0	versicolor	11.50
3.3	1.0	versicolor	11.76
4.5	1.7	virginica	12.25
4.0	1.3	versicolor	12.65
3.0	1.1	versicolor	12.75
1.4	0.2	setosa	12.76
1.1	0.1	setosa	12.90
3.8	1.1	versicolor	13.20

Example: Using summarise

```
iris %>%  
  mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%  
  filter(Sepal.Area < 15) %>%  
  summarise(count=n(), mean=mean(Sepal.Area))
```

count	mean
29	13.36724

Example: Using summarise and group_by

```
iris %>%  
  mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%  
  filter(Sepal.Area < 15) %>% group_by(Species) %>%  
  summarise(count=n(), mean=mean(Sepal.Area))
```

Species	count	mean
setosa	11	13.69545
versicolor	15	13.15333
virginica	3	13.23333

Grouped Dataframes

- ▶ Using the `group_by` operator will create a grouped data frame which acts as multiple data frames partitioned by groups used in the `group_by` condition.
- ▶ All operations will not be done on the entire data frame but on each grouped subset
- ▶ You can use the `ungroup()` operations to remove the grouping variable

The concept of tidy data

When working with data it is important to normalize data. Concept introduced in the database community by E.F. Codd. In the tidyverse a tidy dataset is defined as follows:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

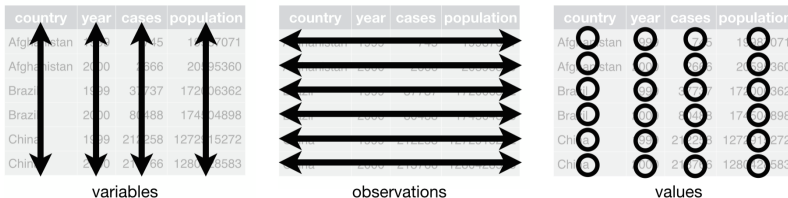


Figure 2: Structure of Tidy Data

Is this data tidy or not tidy?

Is Iris tidy or not tidy?

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.2	3.5	1.5	0.2	setosa
5.6	2.5	3.9	1.1	versicolor
5.1	3.8	1.9	0.4	setosa
4.8	3.0	1.4	0.1	setosa
6.0	3.0	4.8	1.8	virginica
6.5	3.0	5.2	2.0	virginica

Yes, because each row are the measurements for 1 flower making it one observation.

Is this data tidy or not tidy?

What about this data set?

Each row is a country and the columns '1999' and '2000' represents the counts of a disease during those years.

```
table4a
```

```
## # A tibble: 3 × 3
##   country `1999` `2000`
## *   <chr>   <int>   <int>
## 1 Afghanistan    745    2666
## 2      Brazil  37737   80488
## 3        China 212258  213766
```

Is this data tidy or not tidy?

- ▶ This is not tidy because the columns '1999' and '2000' represents measurements for different years.
- ▶ Why do we care?
 - ▶ Imagine we are the WHO and we are constantly updating this database it would be more difficult to add new columns to the new database rather than simply appending new data.

tidyr can be used to massage into a tidy structure

There are two fundamental verbs of data tidying:

1. `gather`: takes multiple columns, and gathers them into key-value pairs: it makes “wide” data longer.
 - ▶ `gather(data, key, value, ..., na.rm = FALSE, convert = FALSE)`
 - ▶ ... represents columns that are meant to be turned into key-value pairs.
2. `spread`: takes two columns (key & value) and spreads in to multiple columns, it makes “long” data wider.
 - ▶ `spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE)`

Example for using gather

We will take the iris data and turn all measurements into key value pairs.

```
iris_long <- iris %>%  
  gather(measurement, value_iris, Sepal.Length:Petal.Width)
```

Species	measurement	value_iris
virginica	Sepal.Width	2.7
virginica	Petal.Width	2.1
versicolor	Sepal.Length	5.8
versicolor	Petal.Width	1.3
setosa	Petal.Width	0.2

Example for using spread

```
mtcarsSpread <- mtcarsNew %>% spread(attribute, value)
```

cars	attribute	value
Toyota Corolla	hp	65.00
Maserati Bora	wt	3.57
Hornet Sportabout	wt	3.44
Mazda RX4	wt	2.62
Volvo 142E	wt	2.78

cars	cyl	disp	drat	hp	mpg	qsec	wt
Mazda RX4 Wag	6	160.0	3.90	110	21.0	17.02	2.875
Merc 450SLC	8	275.8	3.07	180	15.2	18.00	3.780
AMC Javelin	8	304.0	3.15	150	15.2	17.30	3.435
Chrysler Imperial	8	440.0	3.23	230	14.7	17.42	5.345
Fiat X1-9	4	79.0	4.08	66	27.3	18.90	1.935

Philosophy behind ggplot2

- ▶ ggplot2 is a graphics package in R that works directly on data frames
- ▶ Developed by Hadley Wickham as an R package that uses a grammar of graphics (introduced by Leland Wilkinson).
- ▶ ggplot2 grammar
 - ▶ data: the data frame being plotted
 - ▶ geometrics: the geometric shape that will represent the data
 - ▶ aesthetics: aesthetics of the geometric object
 - ▶ Color, size, shape, etc.

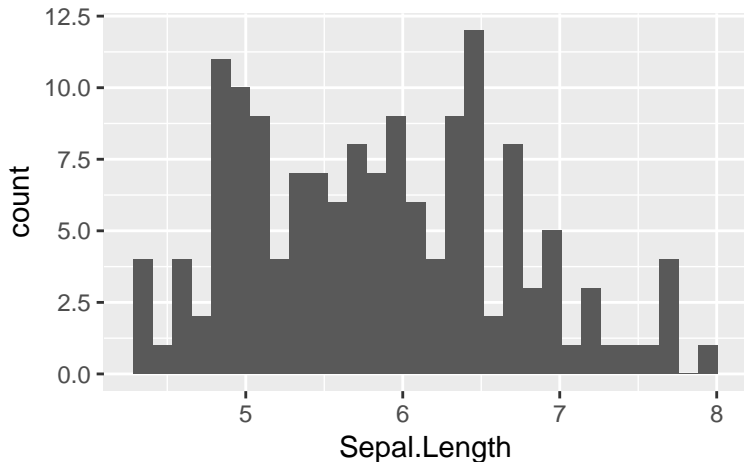
ggplot2 cheat sheet

Different types of plots

- ▶ Histogram - `geom_histogram()`
- ▶ Density Plot - `geom_density()`
- ▶ Bar chart - `geom_bar()`
- ▶ Violin plot - `geom_violin()`
- ▶ Box plot - `geom_boxplot()`
- ▶ Scatter plot - `geom_scatter()`

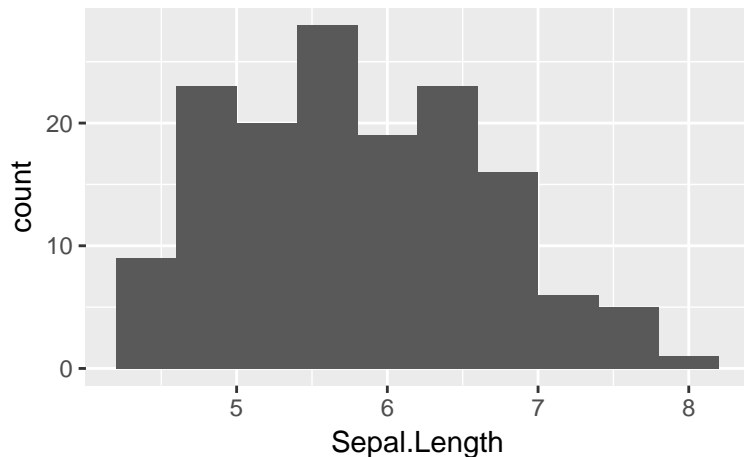
Example: Basic Histogram

```
ggplot(iris,aes(x=Sepal.Length)) +  
  geom_histogram()
```



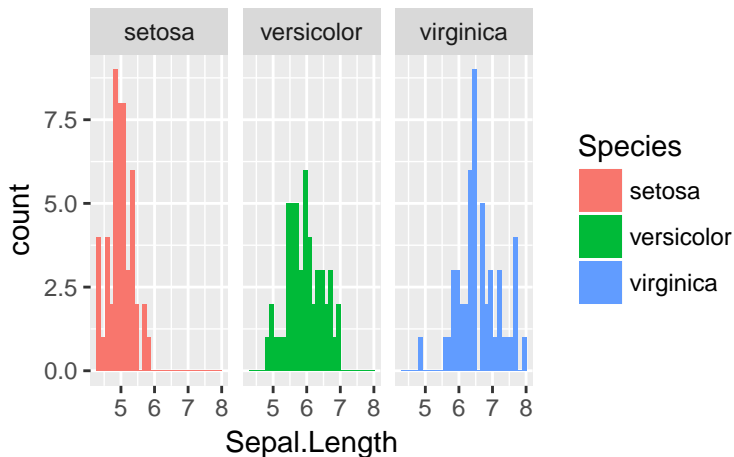
Example: Basic Histogram with 10 bins instead of default of 30

```
ggplot(iris,aes(x=Sepal.Length)) +  
  geom_histogram(bins = 10)
```



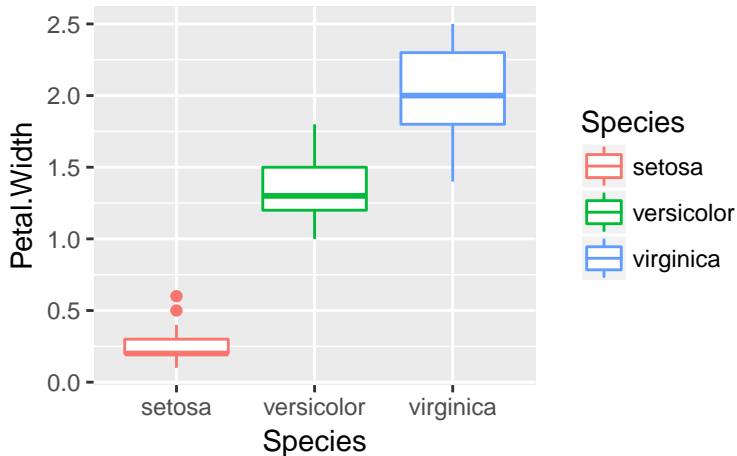
Example: Histogram with facet_wrap with Species and fill = Species

```
ggplot(iris,aes(x=Sepal.Length,  
               fill=Species)) +  
  geom_histogram() + facet_wrap(~Species)
```



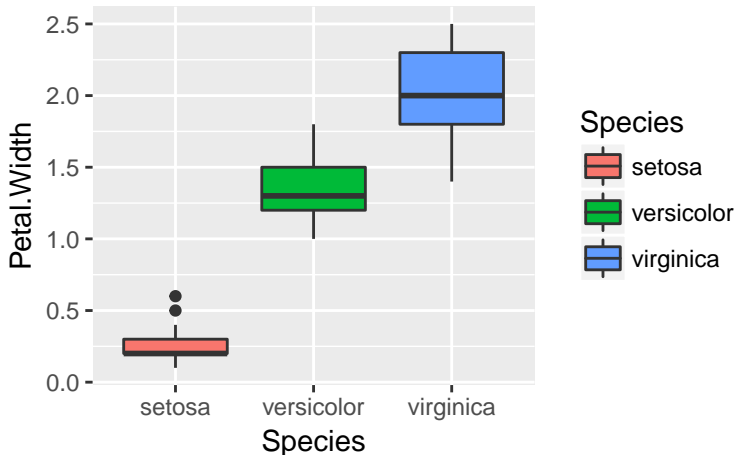
Example: Boxplot with color = Species

```
ggplot(iris,aes(x=Species, y=Petal.Width,  
               color=Species)) +  
  geom_boxplot()
```



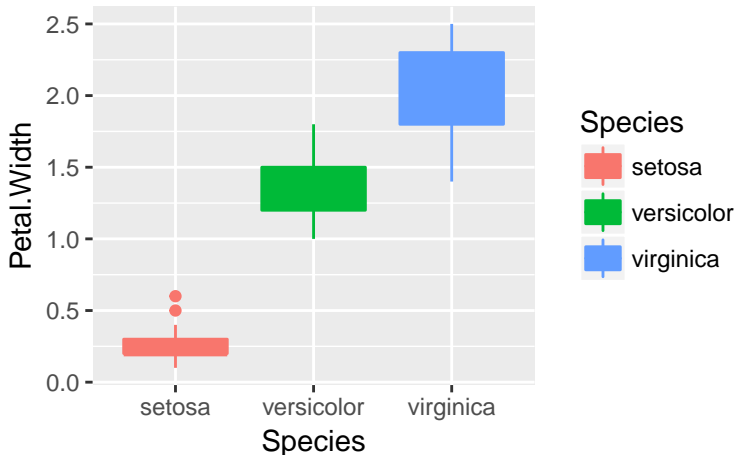
Example: Boxplot with fill = Species

```
ggplot(iris,aes(x=Species, y=Petal.Width,  
               fill=Species)) +  
  geom_boxplot()
```



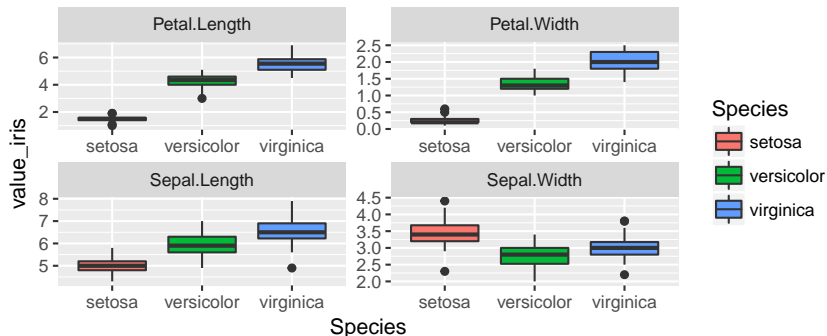
Example: Boxplot with fill = Species and color = Species

```
ggplot(iris,aes(x=Species, y=Petal.Width,  
color=Species ,fill=Species)) +  
  geom_boxplot()
```



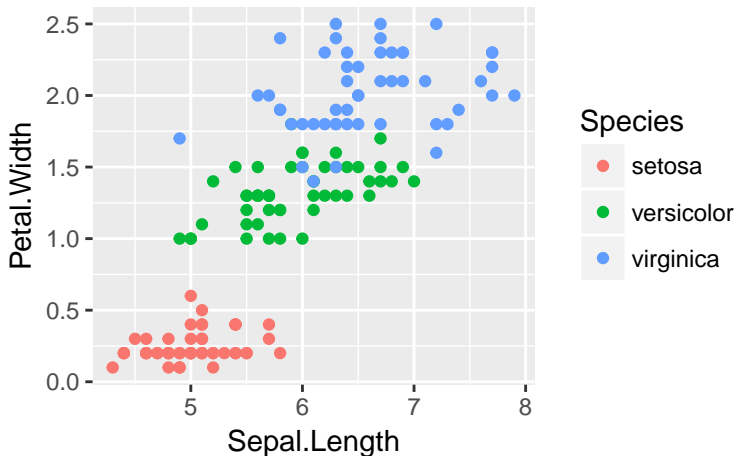
We can use `iris_long` to plot all 4 flower measurements together

```
ggplot(iris_long, aes(x=Species, y=value_iris,
  fill=Species)) + geom_boxplot() +
  facet_wrap(~measurement, scales = 'free')
```



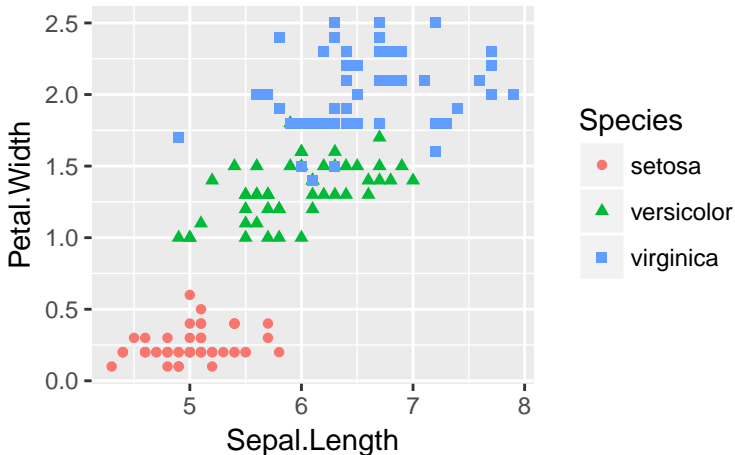
Example: Scatterplot with color = Species

```
ggplot(iris,aes(x=Sepal.Length, y=Petal.Width,  
               color=Species)) +  
  geom_point()
```



Example: Scatterplot with color = Species and shape = Species

```
ggplot(iris,aes(x=Sepal.Length, y=Petal.Width,  
               color=Species, shape=Species)) +  
  geom_point()
```



Linear Modeling in Base R

One of the more interesting aspects of the tidyverse is that these methods can also be applied to linear modeling. In a typical linear regression you get a fit object which conflate many things.

Most models produce output at three levels:

1. Model level: R^2 , residual standard error, MSE
2. Term level: coefficient estimates, p-values, per-cluster information
3. Observation level: predictions, residuals, cluster assignments

Linear Modeling in Base R

```
fit <- lm(Sepal.Length ~ Sepal.Width +  
          Petal.Length + Petal.Width, data = iris)  
#summary(fit)
```

```
Call:  
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width,  
    data = iris)  
  
Residuals:  
    Min       1Q   Median       3Q      Max   
-0.82816 -0.21989  0.01875  0.19709  0.84570  
  
Coefficients:  
            Estimate Std. Error t value Pr(>|t|)      
(Intercept)  1.85600    0.25078   7.401 9.85e-12 ***  
Sepal.Width   0.65084    0.06665   9.765 < 2e-16 ***  
Petal.Length  0.70913    0.05672  12.502 < 2e-16 ***  
Petal.Width  -0.55648    0.12755  -4.363 2.41e-05 ***  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 0.3145 on 146 degrees of freedom  
Multiple R-squared:  0.8586,    Adjusted R-squared:  0.8557  
F-statistic: 295.5 on 3 and 146 DF,  p-value: < 2.2e-16
```

Figure 4: model summary

Philosophy behind broom

broom provides functions to tidy model output in each of these three ways.

1. `glance()`: construct a concise one-row summary of the model. This typically contains values such as R^2 , adjusted R^2 , and residual standard error that are computed once for the entire model.
2. `tidy()`: constructs a data frame that summarizes the model's statistical findings. This includes coefficients and p-values for each term in a regression, per-cluster information in clustering applications, or per-test information for multtest functions.
3. `augment()`: add columns to the original data that was modeled. This includes predictions, residuals, and cluster assignments.

Glance function in broom

```
kable(glance(fit))
```

rsquared	adj.rsquared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual
0.8586117	0.8557065	0.3145491	295.5391	0	4	-37.32136	84.64272	99.6959	14.4454	146

Figure 5: model summary

Tidy function in broom

```
kable(tidy(fit))
```

term	estimate	std.error	statistic	p.value
(Intercept)	1.8559975	0.2507771	7.400984	0.00e+00
Sepal.Width	0.6508372	0.0666474	9.765380	0.00e+00
Petal.Length	0.7091320	0.0567193	12.502483	0.00e+00
Petal.Width	-0.5564827	0.1275479	-4.362929	2.41e-05

Running many regressions in parallel

We can use the broom + dplyr to run many regressions in parallel by using the group_by function. We will need to use the do function from dplyr which allows you to do arbitrary computations within the dplyr framework.

```
regressions <- iris %>%  
  group_by(Species) %>%  
  do(fit=lm(Sepal.Length ~ Sepal.Width +  
            Petal.Length + Petal.Width, data = .))
```

Running many regressions in parallel

The regressions object now contains three fit objects, one for each Species

```
regressions
```

```
## Source: local data frame [3 x 2]
## Groups: <by row>
##
## # A tibble: 3 x 2
##   Species      fit
## *   <fctr>    <list>
## 1   setosa <S3: lm>
## 2 versicolor <S3: lm>
## 3  virginica <S3: lm>
```

Extract model fits from each regression

```
regressions %>% glance(fit)
```

Species <ctr>	r.squared <dbl>	adj.r.squared <dbl>	sigma <dbl>	statistic <dbl>	p.value <dbl>	df <int>	logLik <dbl>	AIC <dbl>	BIC <dbl>
setosa	0.5751375	0.5474291	0.2371318	20.75678	1.192439e-08	4	3.094582	3.810836	13.37095
versicolor	0.6050314	0.5792725	0.3348067	23.48831	2.279929e-09	4	-14.152292	38.304584	47.86470
virginica	0.7652193	0.7499075	0.3179986	49.97584	1.622474e-14	4	-11.576972	33.153944	42.71406

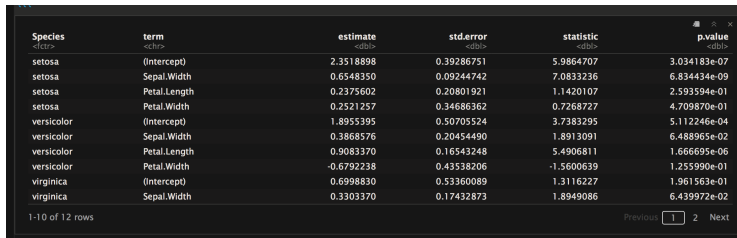
3 rows | 1-10 of 12 columns

Figure 6: model summary

Extract model coefficients from each regression

You can extract each model coefficient for each regression from one data frame. This is a grouped data frame which you need to ungroup in order to do further analysis such as plotting.

```
regressions %>% tidy(fit) %>% ungroup()
```



The screenshot shows a R console window with a dark background. It displays a data frame with 6 columns: Species, term, estimate, std.error, statistic, and p.value. The data is grouped by Species (setosa, versicolor, virginica) and includes coefficients for the intercept and each predictor variable (Sepal.Width, Petal.Length, Petal.Width). The p-values are shown in scientific notation. At the bottom, it indicates '1-10 of 12 rows' and has navigation buttons for 'Previous', '1', '2', and 'Next'.

Species <fctr>	term <chr>	estimate <dbl>	std.error <dbl>	statistic <dbl>	p.value <dbl>
setosa	(Intercept)	2.3518898	0.39286751	5.9864707	3.034183e-07
setosa	Sepal.Width	0.6548350	0.09244742	7.0833236	6.834434e-09
setosa	Petal.Length	0.2375602	0.20801921	1.1420107	2.593594e-01
setosa	Petal.Width	0.2521257	0.34686362	0.7268727	4.709870e-01
versicolor	(Intercept)	1.8955395	0.50705524	3.7383295	5.112246e-04
versicolor	Sepal.Width	0.3868576	0.20454490	1.8913091	6.488965e-02
versicolor	Petal.Length	0.9083370	0.16543248	5.4906811	1.666695e-06
versicolor	Petal.Width	-0.6792238	0.43538206	-1.5600639	1.255990e-01
virginica	(Intercept)	0.6998830	0.53360089	1.3116227	1.961563e-01
virginica	Sepal.Width	0.3303370	0.17432873	1.8949086	6.439972e-02

Figure 7: model summary

Acknowledgements

Borrowed heavily from presentations below:

<http://john-ensley.com/slides/tidyverse/>

http://sjspielman.org/bio5312_fall2017/slides/day2_intro_to_tidyverse.pdf

Other Resources

- ▶ Data Science in the Tidyverse
 - ▶ <https://www.rstudio.com/resources/videos/data-science-in-the-tidyverse/>
- ▶ Tidyverse website
 - ▶ <https://www.tidyverse.org/>
- ▶ R tutorial for beginners
 - ▶ <https://eringrand.github.io/RTutorials/>
- ▶ R for Data Science Textbook
 - ▶ <http://r4ds.had.co.nz/>