Introduction to R
Part 2 – the Tidyverse
Extending R through packages: There’s a package for everything
Contributed Packages

Available Packages

Currently, the CRAN package repository features 15364 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual `R Installation and Administration` (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 41 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, OS X, Solaris and Windows.
You can install packages using `install.packages()` in RStudio

```
> install.packages("ggplot2")
```

The downloaded binary packages are in

```
/var/folders/q8/wptgtbdn1pz0cfgrz39gq00m0000gn/T//RtmpvQgw1u/downloaded_packages
```
“Tidy datasets are all alike but every messy dataset is messy in its own way” — Hadley Wickham
Tidy data

Three rules:
1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table
Example: Contingency table

<table>
<thead>
<tr>
<th></th>
<th>survived</th>
<th>died</th>
</tr>
</thead>
<tbody>
<tr>
<td>drug</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>placebo</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

*not tidy*
Example: Contingency table

<table>
<thead>
<tr>
<th></th>
<th>survived</th>
<th>died</th>
</tr>
</thead>
<tbody>
<tr>
<td>drug</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>placebo</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

not tidy

<table>
<thead>
<tr>
<th>treatment</th>
<th>outcome</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>drug</td>
<td>survived</td>
<td>15</td>
</tr>
<tr>
<td>drug</td>
<td>died</td>
<td>3</td>
</tr>
<tr>
<td>placebo</td>
<td>survived</td>
<td>4</td>
</tr>
<tr>
<td>placebo</td>
<td>died</td>
<td>12</td>
</tr>
</tbody>
</table>

tidy
### Example: Contingency table

<table>
<thead>
<tr>
<th></th>
<th>survived</th>
<th>died</th>
</tr>
</thead>
<tbody>
<tr>
<td>drug</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>placebo</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

*not tidy*

<table>
<thead>
<tr>
<th>patient</th>
<th>treatment</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>drug</td>
<td>survived</td>
</tr>
<tr>
<td>2</td>
<td>drug</td>
<td>died</td>
</tr>
<tr>
<td>3</td>
<td>drug</td>
<td>survived</td>
</tr>
<tr>
<td>4</td>
<td>placebo</td>
<td>died</td>
</tr>
</tbody>
</table>

*tidy*
The **tidyr** library provides functions for transforming tables.

### Example

<table>
<thead>
<tr>
<th>treatment</th>
<th>survived</th>
<th>died</th>
</tr>
</thead>
<tbody>
<tr>
<td>drug</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>placebo</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

### Transformation

- **pivot_wider()**

<table>
<thead>
<tr>
<th>patient</th>
<th>treatment</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>drug</td>
<td>survived</td>
</tr>
<tr>
<td>2</td>
<td>drug</td>
<td>died</td>
</tr>
<tr>
<td>3</td>
<td>drug</td>
<td>survived</td>
</tr>
<tr>
<td>4</td>
<td>placebo</td>
<td>died</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
dplyr: a package for data manipulation
Working with tidy data in R: tidyverse

Fundamental actions on data tables:
• make new columns — `mutate()`
• combine tables, adding columns — `left_join()`
• combine tables, adding rows — `bind_rows()`
• choose rows — `filter()`
• choose columns — `select()`
• arrange rows — `arrange()`
• calculate summary statistics — `summarize()`
• work on groups of data — `group_by()`
Working with tidy data in R: tidyverse

Fundamental actions on data tables:
- make new columns — `mutate()`
- combine tables, adding columns — `left_join()`
- combine tables, adding rows — `bind_rows()`
- choose rows — `filter()`
- choose columns — `select()`
- arrange rows — `arrange()`
- calculate summary statistics — `summarize()`
- work on groups of data — `group_by()`
**select ()**: pick columns
`select()`: pick columns
Choose the two columns Species and Sepal.Width

> select(iris, Species, Sepal.Width)
Choose the two columns Species and Sepal.Width

```r
> select(iris, Species, Sepal.Width)

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>3.5</td>
</tr>
<tr>
<td>setosa</td>
<td>3.0</td>
</tr>
<tr>
<td>setosa</td>
<td>3.2</td>
</tr>
<tr>
<td>setosa</td>
<td>3.1</td>
</tr>
<tr>
<td>setosa</td>
<td>3.6</td>
</tr>
<tr>
<td>setosa</td>
<td>3.9</td>
</tr>
<tr>
<td>setosa</td>
<td>3.4</td>
</tr>
<tr>
<td>setosa</td>
<td>3.4</td>
</tr>
<tr>
<td>setosa</td>
<td>2.9</td>
</tr>
<tr>
<td>setosa</td>
<td>3.1</td>
</tr>
<tr>
<td>setosa</td>
<td>3.7</td>
</tr>
<tr>
<td>setosa</td>
<td>3.4</td>
</tr>
<tr>
<td>setosa</td>
<td>3.0</td>
</tr>
<tr>
<td>setosa</td>
<td>3.0</td>
</tr>
</tbody>
</table>
```
filter(): pick rows
filter(): pick rows
Choose rows with Sepal.Width > 4

> filter(iris, Sepal.Width > 4)
Choose rows with Sepal.Width > 4

```r
> filter(iris, Sepal.Width > 4)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.7         4.4          1.5         0.4  setosa
2          5.2         4.1          1.5         0.1  setosa
3          5.5         4.2          1.4         0.2  setosa
```
select(): pick columns
summarize(): collapse multiple rows
summarize(): collapse multiple rows
> summarize(iris, mean_sepal_length = mean(Sepal.Length),
  sd_sepal_length = sd(Sepal.Length))
Calculate mean and standard deviation of Sepal.Length

```r
> summarize(iris, mean_sepal_length = mean(Sepal.Length),
           sd_sepal_length = sd(Sepal.Length))

   mean_sepal_length sd_sepal_length
1        5.843333       0.8280661
```
`group_by()` : set up groupings
group\textunderscore by() : set up groupings
Calculate mean and standard deviation of Sepal.Length, grouped by Species

```r
> summarize(group_by(iris, Species),
            mean_sepal_length = mean(Sepal.Length),
            sd_sepal_length  = sd(Sepal.Length))
```
Calculate mean and standard deviation of Sepal.Length, grouped by Species

```r
> iris %>%
  group_by(Species) %>%
  summarize(mean_sepal_length = mean(Sepal.Length),
            sd_sepal_length   = sd(Sepal.Length))
```

Source: local data frame [3 x 3]

<table>
<thead>
<tr>
<th>Species</th>
<th>mean_sepal_length</th>
<th>sd_sepal_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>5.006</td>
<td>0.3524897</td>
</tr>
<tr>
<td>versicolor</td>
<td>5.936</td>
<td>0.5161711</td>
</tr>
<tr>
<td>virginica</td>
<td>6.588</td>
<td>0.6358796</td>
</tr>
</tbody>
</table>
Piping multiple functions together

Call the target data set in first position

The data is implicitly passed and processed through each subsequent function

```r
> iris %>%
>  group_by(., Species) %>%
>  summarize(., mean_length = mean(Sepal.Length),
>             sd_length = sd(Sepal.Length))
```
Piping multiple functions together

This implicit operate/pass forward behavior is facilitated by the pipe

```r
iris %>%
  group_by(Species) %>%
  summarize(mean_length = mean(Sepal.Length),
            sd_length = sd(Sepal.Length))
```
ggplot2: a package for data manipulation
ggplot2: A grammar of graphics

Traditional plotting: You **are** a painter
  – Manually place individual graphical elements

ggplot2: You **employ** a painter
  – Describe conceptually how data should be visualized
Most confusing key concept: aesthetic mapping

Maps data values to visual elements of the plot
A few examples of aesthetics

- position
- shape
- size
- color
Let’s go over a simple example: mean height and weight of boys/girls ages 10-20

<table>
<thead>
<tr>
<th>age (yrs)</th>
<th>height (cm)</th>
<th>weight (kg)</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>138</td>
<td>32</td>
<td>M</td>
</tr>
<tr>
<td>15</td>
<td>170</td>
<td>56</td>
<td>M</td>
</tr>
<tr>
<td>20</td>
<td>177</td>
<td>71</td>
<td>M</td>
</tr>
<tr>
<td>10</td>
<td>138</td>
<td>33</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>162</td>
<td>52</td>
<td>F</td>
</tr>
<tr>
<td>20</td>
<td>163</td>
<td>53</td>
<td>F</td>
</tr>
</tbody>
</table>

Data from: http://www.cdc.gov/growthcharts/
Map age to x, height to y, visualize using points

ggplot(data, aes(x=age, y=height)) + geom_point()
Let's color the points by sex

```r
ggplot(data, aes(x=age, y=height, color=sex)) + geom_point()
```

★ NOTE: “color” aesthetic is for coloring points & lines; “fill” aesthetic is for coloring bars & distributions
And change point size by weight

ggplot(data, aes(x=age, y=height, color=sex, size=weight)) + geom_point()
And connect the points with lines

ggplot(data, aes(x=age, y=height, color=sex, size=weight)) + geom_point() + geom_line()
The weight-to-size mapping should only be applied to points

ggplot(data, aes(x=age, y=height, color=sex)) + geom_point(aes(size=weight)) + geom_line()
We can also make side-by-side plots (called facets)

ggplot(data, aes(x=age, y=height, color=sex)) + geom_point(aes(size=weight)) + geom_line() + facet_wrap(~sex)
All the geoms with all their options are described on the ggplot2 web page

https://ggplot2.tidyverse.org/reference/
Example 1: Change scaling of x axis

```r
ggplot(movies, aes(x=length, y=rating)) +
  geom_point() +
  scale_x_log10()
```
Example 1: Change scaling of x axis

```r
ggplot(movies, aes(x=length, y=rating)) + geom_point() + xlim(1, 50)
```
Example 2: Change color scaling

```r
ggplot(iris, aes(x=Species, fill=Species)) + geom_bar() + scale_fill_grey()
```
Example 2: Change color scaling

```r
ggplot(iris, aes(x=Species, fill=Species)) + geom_bar() + scale_fill_brewer()
```
Some color scaling options in ggplot2

- `scale_color_gradient()`, `scale_fill_gradient()`
- `scale_color_discrete()`, `scale_fill_discrete()`
- `scale_color_brewer()`, `scale_fill_brewer()`
- `scale_color_distiller()`, `scale_fill_distiller()`
- `scale_color_colorblind()`, `scale_fill_colorblind()`
- `scale_color_manual()`, `scale_fill_manual()`

```r
palette_pretty <- c("#0072B2", "#E69F00", "#009E24", "#FF7F00", "#F0E442", "#D55E00", "#CC79A7", "#999999")
palette_bgy <- c("#FFFCC", "#A1DAB4", "#F0E442", "#999999")
palette_wine <- c("#bcb37b", "#9e993d", "#8f8f8f", "#790000", "#5b0b0b")
palette_cb <- c("#E69F00", "#56B4E9", "#009E24", "#F0E442", "#0072B2", "#D55E00", "#CC79A7", "#999999")
```