

R For Data Science Cheat Sheet

Tidyverse for Beginners

Learn More R for Data Science Interactively at www.datacamp.com



Tidyverse

The **tidyverse** is a powerful collection of R packages that are actually data tools for transforming and visualizing data. All packages of the tidyverse share an underlying philosophy and common APIs.

The core packages are:



- **ggplot2**, which implements the grammar of graphics. You can use it to visualize your data.



- **dplyr** is a grammar of data manipulation. You can use it to solve the most common data manipulation challenges.



- **tidyr** helps you to create tidy data or data where each variable is in a column, each observation is a row and each value is a cell.



- **readr** is a fast and friendly way to read rectangular data.



- **purrr** enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.



- **tibble** is a modern re-imaging of the data frame.



- **stringr** provides a cohesive set of functions designed to make working with strings as easy as possible



- **forcats** provide a suite of useful tools that solve common problems with factors.

You can install the complete tidyverse with:

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

Then, load the core tidyverse and make it available in your current R session by running:

```
> library(tidyverse)
```

Note: there are many other tidyverse packages with more specialised usage. They are not loaded automatically with `library(tidyverse)`, so you'll need to load each one with its own call to `library()`.

Useful Functions

<pre>> tidyverse_conflicts() > tidyverse_deps() > tidyverse_logo()</pre>	Conflicts between tidyverse and other packages List all tidyverse dependencies Get tidyverse logo, using ASCII or unicode characters
<pre>> tidyverse_packages() > tidyverse_update()</pre>	List all tidyverse packages Update tidyverse packages

Loading in the data

<pre>> library(datasets) > library(gapminder) > attach(iris)</pre>	Load the datasets package Load the gapminder package Attach iris data to the R search path
---	--

dplyr

Filter

`filter()` allows you to select a subset of rows in a data frame.

<pre>> iris %>% filter(Species=="virginica") > iris %>% filter(Species=="virginica", Sepal.Length > 6)</pre>	Select iris data of species "virginica" Select iris data of species "virginica" and sepal length greater than 6.
---	---

Arrange

`arrange()` sorts the observations in a dataset in ascending or descending order based on one of its variables.

<pre>> iris %>% arrange(Sepal.Length) > iris %>% arrange(desc(Sepal.Length))</pre>	Sort in ascending order of sepal length Sort in descending order of sepal length
--	---

Combine multiple `dplyr` verbs in a row with the pipe operator `%>%`:

<pre>> iris %>% filter(Species=="virginica") %>% arrange(desc(Sepal.Length))</pre>	Filter for species "virginica" then arrange in descending order of sepal length
---	---

Mutate

`mutate()` allows you to update or create new columns of a data frame.

<pre>> iris %>% mutate(Sepal.Length=Sepal.Length*10) > iris %>% mutate(SLMm=Sepal.Length*10)</pre>	Change Sepal.Length to be in millimeters Create a new column called SLMm
--	---

Combine the verbs `filter()`, `arrange()`, and `mutate()`:

```
> iris %>%
  filter(Species=="Virginica") %>%
  mutate(SLMm=Sepal.Length*10) %>%
  arrange(desc(SLMm))
```

Summarize

`summarize()` allows you to turn many observations into a single data point.

<pre>> iris %>% summarize(medianSL=median(Sepal.Length)) > iris %>% filter(Species=="virginica") %>% summarize(medianSL=median(Sepal.Length))</pre>	Summarize to find the median sepal length Filter for virginica then summarize the median sepal length
--	--

You can also summarize multiple variables at once:

```
> iris %>%
  filter(Species=="virginica") %>%
  summarize(medianSL=median(Sepal.Length),
            maxSL=max(Sepal.Length))
```

`group_by()` allows you to summarize within groups instead of summarizing the entire dataset:

<pre>> iris %>% group_by(Species) %>% summarize(medianSL=median(Sepal.Length), maxSL=max(Sepal.Length)) > iris %>% filter(Sepal.Length>6) %>% group_by(Species) %>% summarize(medianPL=median(Petal.Length), maxPL=max(Petal.Length))</pre>	Find median and max sepal length of each species Find median and max petal length of each species with sepal length > 6
---	--

ggplot2

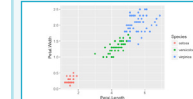
Scatter plot

Scatter plots allow you to compare two variables within your data. To do this with `ggplot2`, you use `geom_point()`

<pre>> iris_small <- iris %>% filter(Sepal.Length > 5) > ggplot(iris_small, aes(x=Petal.Length, y=Petal.Width)) + geom_point()</pre>	Compare petal width and length
---	--------------------------------

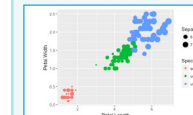
Additional Aesthetics

• Color



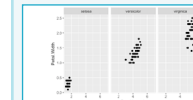
```
> ggplot(iris_small, aes(x=Petal.Length,
                        y=Petal.Width,
                        color=Species)) +
  geom_point()
```

• Size



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

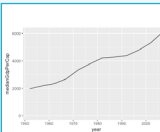
Faceting



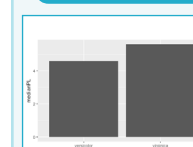
```
> ggplot(iris_small, aes(x=Petal.Length,
                        y=Petal.Width)) +
  geom_point() +
  facet_wrap(~Species)
```

Line Plots

```
> by_year <- gapminder %>%
  group_by(year) %>%
  summarize(medianGdpPerCap=median(gdpPerCap))
> ggplot(by_year, aes(x=year,
                    y=medianGdpPerCap)) +
  geom_line() +
  expand_limits(y=0)
```



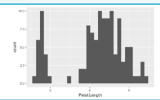
Bar Plots



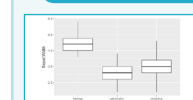
```
> by_species <- iris %>%
  filter(Sepal.Length>6) %>%
  group_by(Species) %>%
  summarize(medianPL=median(Petal.Length))
> ggplot(by_species, aes(x=Species,
                        y=medianPL)) +
  geom_col()
```

Histograms

```
> ggplot(iris_small, aes(x=Petal.Length)) +
  geom_histogram()
```



Box Plots



```
> ggplot(iris_small, aes(x=Species,
                        y=Sepal.Length)) +
  geom_boxplot()
```

