R For Data Science *Cheat Sheet*

xts

Learn R for data science **Interactively** at www.DataCamp.com

xts

eXtensible Time Series (xts) is a powerful package that provides an extensible time series class, enabling uniform handling of many R time series classes by extending zoo.

Load the package as follows:

```
> library(xts)
```

xts Objects

xts objects have three main components:

- **coredata**: always a matrix for xts objects, while it could also be a vector for zoo objects
- **index**: vector of any `Date`, `POSIXct`, `chron`, `yearmon`, `yearqtr`, or `DateTime` classes
- **xtsAttributes**: arbitrary attributes

Creating xts Objects

```
> xts1 <- xts(x=1:10, order.by=Sys.Date()-1:10)
> data <- rnorm(5)
> dates <- seq(as.Date("2017-05-01"),length=5,by="days")
> xts2 <- xts(x=data, order.by=dates)
> xts3 <- xts(x=rnorm(10),
              order.by=as.POSIXct(Sys.Date()+1:10),
              born=as.POSIXct("1899-05-08"))
> xts4 <- xts(x=1:10, order.by=Sys.Date()+1:10)
```

Convert To And From xts

```
> data(AirPassengers)
> xts5 <- as.xts(AirPassengers)
```

Import From Files

```
> dat <- read.csv(tmp_file)
> xts(dat, order.by=as.Date(rownames(dat),"%m/%d/%Y"))
> dat_zoo <- read.zoo(tmp_file,
                      index.column=0,
                      sep=",",
                      format="%m/%d/%Y")
> dat_zoo <- read.zoo(tmp,sep=",",FUN=as.yearmon)
> dat_xts <- as.xts(dat_zoo)
```

Export xts Objects

```
> data_xts <- as.xts(matrix)
> tmp <- tempfile()
> write.zoo(data_xts,sep=",",file=tmp)
```

Replace & Update

`> xts2[dates] <- 0`	Replace values in `xts2` on `dates` with `0`
`> xts5["1961"] <- NA`	Replace dates from `1961` with `NA`
`> xts2["2016-05-02"] <- NA`	Replace the value at 1 specific index with `NA`

Applying Functions

```
> ep1 <- endpoints(xts4,on="weeks",k=2)
[1]  0  5 10
> ep2 <- endpoints(xts5,on="years")
[1]   0  12  24  36  48  60  72  84  96 108 120 132 144
```

Take index values by time

`> period.apply(xts5,INDEX=ep2,FUN=mean)`	Calculate the yearly mean
`> xts5_yearly <- split(xts5,f="years")`	Split `xts5` by year
`> lapply(xts5_yearly,FUN=mean)`	Create a list of yearly means

`> do.call(rbind,`
`lapply(split(xts5,"years"),`
`function(w) last(w,n="1 month")))`
	Find the last observation in each year in `xts5`

`> do.call(rbind,`
`lapply(split(xts5,"years"),`
`cumsum))`
	Calculate cumulative annual passengers

`> rollapply(xts5, 3, sd)`	Apply sd to rolling margins of `xts5`

Selecting, Subsetting & Indexing

Select

`> mar55 <- xts5["1955-03"]`	Get value for March 1955

Subset

`> xts5_1954 <- xts5["1954"]`	Get all data from 1954
`> xts5_janmarch <- xts5["1954/1954-03"]`	Extract data from Jan to March '54
`> xts5_janmarch <- xts5["/1954-03"]`	Get all data until March '54
`> xts4[ep1]`	Subset `xts4` using `ep2`

`first()` and `last()`

`> first(xts4,'1 week')`	Extract first 1 week
`> first(last(xts4,'1 week'),'3 days')`	Get first 3 days of the last week of data

Indexing

`> xts2[index(xts3)]`	Extract rows with the index of `xts3`
`> days <- c("2017-05-03","2017-05-23")`	
`> xts3[days]`	Extract rows using the vector `days`
`> xts2[as.POSIXct(days,tz="UTC")]`	Extract rows using `days` as POSIXct
`> index <- which(.indexwday(xts1)==0|.indexwday(xts1)==6)`	Index of weekend days
`> xts1[index]`	Extract weekend days of `xts1`

Missing Values

`> na.omit(xts5)`	Omit NA values in `xts5`
`> xts_last <- na.locf(xts2)`	Fill missing values in `xts2` using last observation

`> xts_last <- na.locf(xts2,`
`fromLast=TRUE)`
	Fill missing values in `xts2` using next observation

`> na.approx(xts2)`	Interpolate NAs using linear approximation

Arithmetic Operations

`coredata()` or `as.numeric()`

`> xts3 + as.numeric(xts2)`	Addition
`> xts3 * as.numeric(xts4)`	Multiplication
`> coredata(xts4) - xts3`	Subtraction
`> coredata(xts4) / xts3`	Division

Shifting Index Values

`> xts5 - lag(xts5)`	Period-over-period differences
`> diff(xts5,lag=12,differences=1)`	Lagged differences

Reindexing

```
> xts1 + merge(xts2,index(xts1),fill=0)
                   e1
2017-05-04 5.231538
2017-05-05 5.829257
2017-05-06 4.000000
2017-05-07 3.000000
2017-05-08 2.000000
2017-05-09 1.000000
```

Addition

```
> xts1 - merge(xts2,index(xts1),fill=na.locf)
                    e1
2017-05-04 5.231538
2017-05-05 5.829257
2017-05-06 4.829257
2017-05-07 3.829257
2017-05-08 2.829257
2017-05-09 1.829257
```

Subtraction

Merging

```
> merge(xts2,xts1,join='inner')
                 xts2 xts1
2017-05-05 -0.8382068   10
```

Inner join of `xts2` and `xts1`

```
> merge(xts2,xts1,join='left',fill=0)
                  xts2 xts1
2017-05-01  1.7482704    0
2017-05-02 -0.2314678    0
2017-05-03  0.1685517    0
2017-05-04  1.1685649    0
2017-05-05 -0.8382068   10
```

Left join of `xts2` and `xts1`, fill empty spots with `0`

```
> rbind(xts1, xts4)
```

Combine `xts1` and `xts4` by rows

Inspect Your Data

`> core_data <- coredata(xts2)`	Extract core data of objects
`> index(xts1)`	Extract index of objects

Class Attributes

`> indexClass(xts2)`	Get index class
`> indexClass(convertIndex(xts,'POSIXct'))`	Replacing index class
`> indexTZ(xts5)`	Get index class
`> indexFormat(xts5) <- "%Y-%m-%d"`	Change format of time display

Time Zones

`> tzone(xts1) <- "Asia/Hong_Kong"`	Change the time zone
`> tzone(xts1)`	Extract the current time zone

Periods, Periodicity & Timestamps

`> periodicity(xts5)`	Estimate frequency of observations
`> to.yearly(xts5)`	Convert `xts5` to yearly OHLC
`> to.monthly(xts3)`	Convert `xts3` to monthly OHLC
`> to.quarterly(xts5)`	Convert `xts5` to quarterly OHLC
`> to.period(xts5,period="quarters")`	Convert to quarterly OHLC
`> to.period(xts5,period="years")`	Convert to yearly OHLC
`> nmonths(xts5)`	Count the months in `xts5`
`> nquarters(xts5)`	Count the quarters in `xts5`
`> nyears(xts5)`	Count the years in `xts5`
`> make.index.unique(xts3,eps=1e-4)`	Make index unique
`> make.index.unique(xts3,drop=TRUE)`	Remove duplicate times
`> align.time(xts3,n=3600)`	Round index time to the next `n` seconds

Other Useful Functions

`> .index(xts4)`	Extract raw numeric index of `xts1`
`> .indexwday(xts3)`	Value of week(day), starting on Sunday, in index of `xts3`
`> .indexhour(xts3)`	Value of hour in index of `xts3`
`> start(xts3)`	Extract first observation of `xts3`
`> end(xts4)`	Extract last observation of `xts4`
`> str(xts3)`	Display structure of `xts3`
`> time(xts1)`	Extract raw numeric index of `xts1`
`> head(xts2)`	First part of `xts2`
`> tail(xts2)`	Last part of `xts2`

DataCamp
Learn R for Data Science **Interactively**

R For Data Science *Cheat Sheet*

data.table

Learn R for data science **Interactively** at www.DataCamp.com

data.table

data.table is an R package that provides a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.

Load the package:

```
> library(data.table)
```

Creating A `data.table`

```
> set.seed(45L)
> DT <- data.table(V1=c(1L,2L),
                   V2=LETTERS[1:3],
                   V3=round(rnorm(4),4),
                   V4=1:12)
```

Create a `data.table` and call it `DT`

Subsetting Rows Using `i`

```
> DT[3:5,]
> DT[3:5]
> DT[V2=="A"]
> DT[V2 %in% c("A","C")]
```

Select 3rd to 5th row
Select 3rd to 5th row
Select all rows that have value `A` in column `V2`
Select all rows that have value `A` or `C` in column `V2`

Manipulating on Columns in `j`

```
> DT[,V2]
 [1] "A" "B" "C" "A" "B" "C" ...
```

Return `V2` as a vector

```
> DT[,.(V2,V3)]
```

Return `V2` and `V3` as a `data.table`

```
> DT[,sum(V1)]
 [1] 18
```

Return the sum of all elements of `V1` in a vector

```
> DT[,.(sum(V1),sd(V3))]
    V1        V2
 1: 18 0.4546055
```

Return the sum of all elements of `V1` and the std. dev. of `V3` in a `data.table`

```
> DT[,.(Aggregate=sum(V1),
        Sd.V3=sd(V3))]
    Aggregate     Sd.V3
 1:        18 0.4546055
```

The same as the above, with new names

```
> DT[,.(V1,Sd.V3=sd(V3))]
```

Select column `V2` and compute std. dev. of `V3`, which returns a single value and gets recycled

```
> DT[,.(print(V2),
        plot(V3),
        NULL)]
```

Print column `V2` and plot `V3`

Doing `j` by Group

```
> DT[,.(V4.Sum=sum(V4)),by=V1]
    V1 V4.Sum
 1:  1     36
 2:  2     42
```

Calculate sum of `V4` for every group in `V1`

```
> DT[,.(V4.Sum=sum(V4)),
        by=.(V1,V2)]
```

Calculate sum of `V4` for every group in `V1` and `V2`

```
> DT[,.(V4.Sum=sum(V4)),
        by=sign(V1-1)]
   sign V4.Sum
1:    0     36
2:    1     42
```

Calculate sum of `V4` for every group in `sign(V1-1)`

```
> DT[,.(V4.Sum=sum(V4)),
        by=.(V1.01=sign(V1-1))]
```

The same as the above, with new name for the variable you're grouping by

```
> DT[1:5,.(V4.Sum=sum(V4)),
            by=V1]
```

Calculate sum of `V4` for every group in `V1` after subsetting on the first 5 rows

```
> DT[,.N,by=V1]
```

Count number of rows for every group in `V1`

General form: `DT[i, j, by]`

"Take `DT`, subset rows using `i`, then calculate `j` grouped by `by`"

Adding/Updating Columns By Reference in `j` Using :=

```
> DT[,V1:=round(exp(V1),2)]
> DT
     V1 V2      V3 V4
 1: 2.72  A -0.1107  1
 2: 7.39  B -0.1427  2
 3: 2.72  C -1.8893  3
 4: 7.39  A -0.3571  4
 ...
```

`V1` is updated by what is after `:=`
Return the result by calling `DT`

```
> DT[,c("V1","V2"):=list(round(exp(V1),2),
                         LETTERS[4:6])]
> DT[,':='(V1=round(exp(V1),2),
           V2=LETTERS[4:6])][]
         V1 V2      V3 V4
 1:   15.18  D -0.1107  1
 2: 1619.71  E -0.1427  2
 3:   15.18  F -1.8893  3
 4: 1619.71  D -0.3571  4
```

Columns `V1` and `V2` are updated by what is after `:=`
Alternative to the above one. With `[]`, you print the result to the screen

```
> DT[,V1:=NULL]
> DT[,c("V1","V2"):=NULL]
> Cols.chosen=c("A","B")
> DT[,Cols.Chosen:=NULL]

> DT[,(Cols.Chosen):=NULL]
```

Remove `V1`
Remove columns `V1` and `V2`

Delete the column with column name `Cols.chosen`
Delete the columns specified in the variable `Cols.chosen`

Indexing And Keys

```
> setkey(DT,V2)
> DT["A"]
    V1 V2      V3 V4
 1:  1  A -0.2392  1
 2:  2  A -1.6148  4
 3:  1  A  1.0498  7
 4:  2  A  0.3262 10
```

A key is set on `V2`; output is returned invisibly
Return all rows where the key column (set to `V2`) has the value `A`

```
> DT[c("A","C")]
> DT["A",mult="first"]

> DT["A",mult="last"]
```

Return all rows where the key column (`V2`) has value `A` or `C`
Return first row of all rows that match value `A` in key column `V2`
Return last row of all rows that match value `A` in key column `V2`

```
> DT[c("A","D")]
    V1 V2      V3 V4
 1:  1  A -0.2392  1
 2:  2  A -1.6148  4
 3:  1  A  1.0498  7
 4:  2  A  0.3262 10
 5: NA  D      NA NA
```

Return all rows where key column `V2` has value `A` or `D`

```
> DT[c("A","D"),nomatch=0]
    V1 V2      V3 V4
 1:  1  A -0.2392  1
 2:  2  A -1.6148  4
 3:  1  A  1.0498  7
 4:  2  A  0.3262 10
```

Return all rows where key column `V2` has value `A` or `D`

```
> DT[c("A","C"),sum(V4)]
```

Return total sum of `V4`, for rows of key column `V2` that have values `A` or `C`

```
> DT[c("A","C"),
     sum(V4),
     by=.EACHI]
   V2 V1
 1: A 22
 2: C 30
```

Return sum of column `V4` for rows of `V2` that have value `A`, and anohter sum for rows of `V2` that have value `C`

```
> setkey(DT,V1,V2)
> DT[.(2,"C")]
   V1 V2      V3 V4
 1: 2  C  0.3262  6
 2: 2  C -1.6148 12
```

Sort by `V1` and then by `V2` within each group of `V1` (invisible)
Select rows that have value `2` for the first key (`V1`) and the value `C` for the second key (`V2`)

```
> DT[.(2,c("A","C"))]
   V1 V2      V3 V4
 1: 2  A -1.6148  4
 2: 2  A  0.3262 10
 3: 2  C  0.3262  6
 4: 2  C -1.6148 12
```

Select rows that have value `2` for the first key (`V1`) and within those rows the value `A` or `C` for the second key (`V2`)

Advanced Data Table Operations

```
> DT[.N-1]
> DT[,.N]
> DT[,.(V2,V3)]
> DT[,list(V2,V3)]
> DT[,mean(V3),by=.(V1,V2)]
   V1 V2      V1
 1: 1  A  0.4053
 2: 1  B  0.4053
 3: 1  C  0.4053
 4: 2  A -0.6443
 5: 2  B -0.6443
 6: 2  C -0.6443
```

Return the penultimate row of the `DT`
Return the number of rows
Return `V2` and `V3` as a `data.table`
Return `V2` and `V3` as a `data.table`
Return the result of `j`, grouped by all possible combinations of groups specified in `by`

`.SD` & `.SDcols`

```
> DT[,print(.SD),by=V2]
> DT[,.SD[c(1,.N)],by=V2]
> DT[,lapply(.SD,sum),by=V2]
```

Look at what `.SD` contains
Select the first and last row grouped by `V2`
Calculate sum of columns in `.SD` grouped by `V2`

```
> DT[,lapply(.SD,sum),by=V2,
      .SDcols=c("V3","V4")]
   V2     V3 V4
 1: A -0.478 22
 2: B -0.478 26
 3: C -0.478 30
```

Calculate sum of `V3` and `V4` in `.SD` grouped by `V2`

```
> DT[,lapply(.SD,sum),by=V2,
      .SDcols=paste0("V",3:4)]
```

Calculate sum of `V3` and `V4` in `.SD` grouped by `V2`

Chaining

```
> DT <- DT[,.(V4.Sum=sum(V4)),
           by=V1]
    V1 V4.Sum
 1:  1     36
 2:  2     42
```

Calculate sum of `V4`, grouped by `V1`

```
> DT[V4.Sum>40]
> DT[,.(V4.Sum=sum(V4)),
       by=V1][V4.Sum>40]
    V1 V4.Sum
 1:  2     42
```

Select that group of which the sum is >40
Select that group of which the sum is >40 (chaining)

```
> DT[,.(V4.Sum=sum(V4)),
       by=V1][order(-V1)]
    V1 V4.Sum
 1:  2     42
 2:  1     36
```

Calculate sum of `V4`, grouped by `V1`, ordered on `V1`

`set()`-Family

`set()`

Syntax: `for (i in from:to) set(DT, row, column, new value)`

```
> rows <- list(3:4,5:6)
> cols <- 1:2
> for(i in seq_along(rows))
    {set(DT,
         i=rows[[i]],
         j=cols[i],
         value=NA)}
```

Sequence along the values of `rows`, and for the values of `cols`, set the values of those elements equal to `NA` (invisible)

`setnames()`

Syntax: `setnames(DT,"old","new")[]`

```
> setnames(DT,"V2","Rating")
> setnames(DT,
           c("V2","V3"),
           c("V2.rating","V3.DC"))
```

Set name of `V2` to `Rating` (invisible)
Change 2 column names (invisible)

`setnames()`

Syntax: `setcolorder(DT,"neworder")`

```
> setcolorder(DT,
              c("V2","V1","V4","V3"))
```

Change column ordering to contents of the specified vector (invisible)

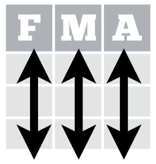
DataCamp
Learn Python for Data Science **Interactively**

Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio®

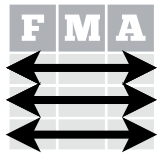
Tidy Data - A foundation for wrangling in R

In a tidy data set:



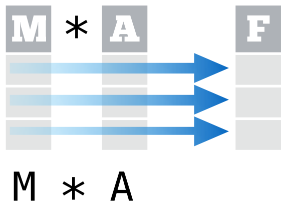
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

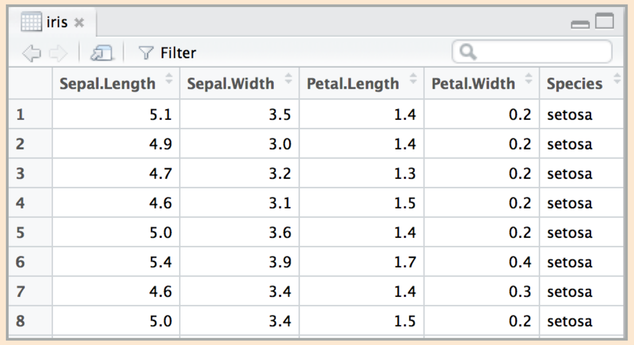
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



	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
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

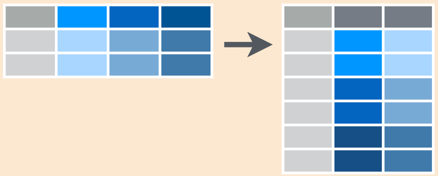
`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

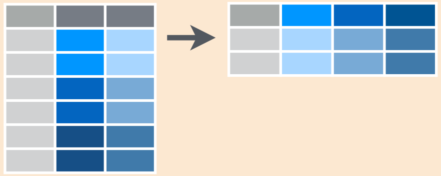
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Reshaping Data - Change the layout of a data set



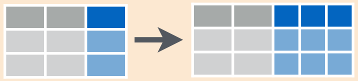
tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



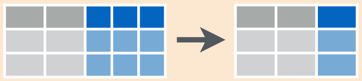
tidyr::**spread(pollution, size, amount)**

Spread rows into columns.



tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.



tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

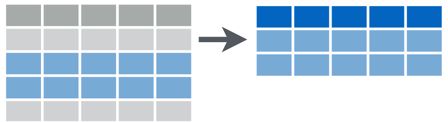
dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)



dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

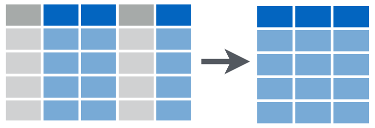
dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

`<`	Less than	`!=`	Not equal to
`>`	Greater than	`%in%`	Group membership
`==`	Equal to	`is.na`	Is NA
`<=`	Less than or equal to	`!is.na`	Is not NA
`>=`	Greater than or equal to	`&,|,!,xor,any,all`	Boolean operators

Subset Variables (Columns)



dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)

Select columns whose name contains a character string.

select(iris, **ends_with("Length")**)

Select columns whose name ends with a character string.

select(iris, **everything()**)

Select every column.

select(iris, **matches(".t.")**)

Select columns whose name matches a regular expression.

select(iris, **num_range("x", 1:5)**)

Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of(c("Species", "Genus"))**)

Select columns whose names are in a group of names.

select(iris, **starts_with("Sepal")**)

Select columns whose name starts with a character string.

select(iris, Sepal.Length**:**Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, **-**Species)

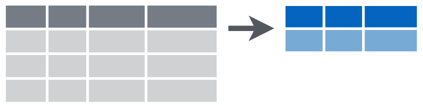
Select all columns except Species.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with **browseVignettes(package = c("dplyr", "tidyr"))** • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Summarise Data



dplyr::**summarise(iris, avg = mean(Sepal.Length))**

Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**

Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**

Count number of rows with each unique value of variable (with or without weights).

summary function

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first**
First value of a vector.

dplyr::**last**
Last value of a vector.

dplyr::**nth**
Nth value of a vector.

dplyr::**n**
# of values in a vector.

dplyr::**n_distinct**
# of distinct values in a vector.

IQR
IQR of a vector.

min
Minimum value in a vector.

max
Maximum value in a vector.

mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

sd
Standard deviation of a vector.

Group Data

dplyr::**group_by(iris, Species)**

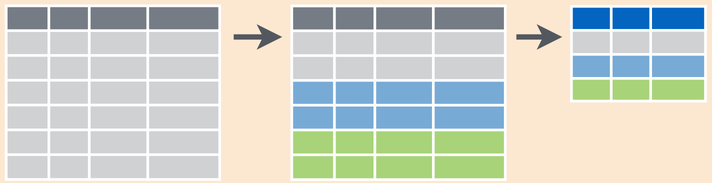
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**

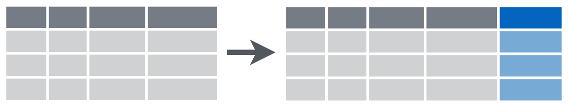
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)

Compute separate summary row for each group.



Make New Variables



dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**

Compute and append one or more new columns.

dplyr::**mutate_each(iris, funs(min_rank))**

Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**

Compute one or more new columns. Drop original columns.

window function

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead**
Copy with values shifted by 1.

dplyr::**lag**
Copy with values lagged by 1.

dplyr::**dense_rank**
Ranks with no gaps.

dplyr::**min_rank**
Ranks. Ties get min rank.

dplyr::**percent_rank**
Ranks rescaled to [0, 1].

dplyr::**row_number**
Ranks. Ties got to first value.

dplyr::**ntile**
Bin vector into n buckets.

dplyr::**between**
Are values between a and b?

dplyr::**cume_dist**
Cumulative distribution.

dplyr::**cumall**
Cumulative `all`

dplyr::**cumany**
Cumulative `any`

dplyr::**cummean**
Cumulative `mean`

cumsum
Cumulative `sum`

cummax
Cumulative `max`

cummin
Cumulative `min`

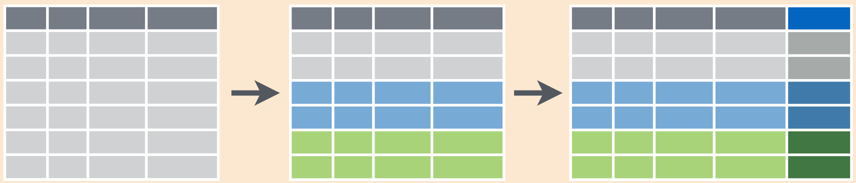
cumprod
Cumulative `prod`

pmax
Element-wise `max`

pmin
Element-wise `min`

iris %>% group_by(Species) %>% mutate(…)

Compute new variables by group.



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**

All rows in a that have a match in b.

x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**

All rows in a that do not have a match in b.

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Set Operations

x1	x2
B	2
C	3

dplyr::**intersect(y, z)**

Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::**union(y, z)**

Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::**setdiff(y, z)**

Rows that appear in y but not z.

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::**bind_rows(y, z)**

Append z to y as new rows.

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::**bind_cols(y, z)**

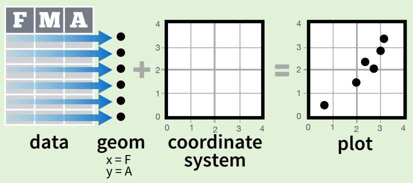
Append z to y as new columns.
Caution: matches rows by position.

Data Visualization
with ggplot2
Cheat Sheet

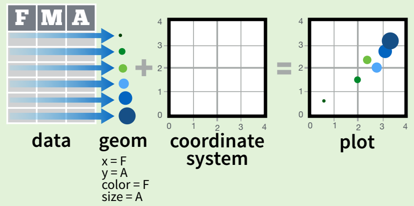
RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

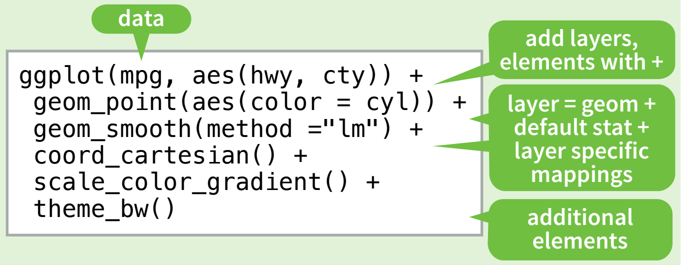
aesthetic mappings   data   geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, **aes**(x = cty, y = hwy))

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().



```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

data
add layers, elements with +
layer = geom + default stat + layer specific mappings
additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + **geom_density**(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + **geom_dotplot()**
x, y, alpha, color, fill

a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + **geom_histogram**(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + **geom_polygon**(aes(group = group))
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + **geom_path**(lineend="butt", linejoin="round', linemitre=1)
x, y, alpha, color, linetype, size

d + **geom_ribbon**(aes(ymin=unemploy - 900, ymax=unemploy + 900))
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + **geom_segment**(aes(
xend = long + delta_long,
yend = lat + delta_lat))
x, xend, y, yend, alpha, color, linetype, size

e + **geom_rect**(aes(xmin = long, ymin = lat,
xmax= long + delta_long,
ymax = lat + delta_lat))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + **geom_blank()**

f + **geom_jitter()**
x, y, alpha, color, fill, shape, size

f + **geom_point()**
x, y, alpha, color, fill, shape, size

f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight

f + **geom_rug**(sides = "bl")
alpha, color, linetype, size

f + **geom_smooth**(model = lm)
x, y, alpha, color, fill, linetype, size, weight

f + **geom_text**(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + **geom_dotplot**(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill

g + **geom_violin**(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + **geom_jitter()**
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + **geom_bin2d**(binwidth = c(5, 0.5))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + **geom_area()**
x, y, alpha, color, fill, linetype, size

j + **geom_line()**
x, y, alpha, color, linetype, size

j + **geom_step**(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + **geom_crossbar**(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + **geom_errorbar()**
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + **geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size

k + **geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + **geom_map**(aes(map_id = state), map = map) +
expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

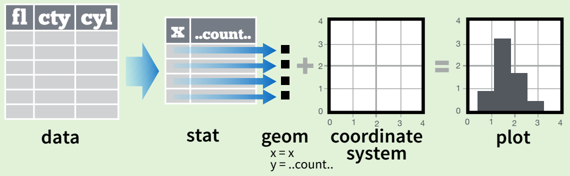
m + **geom_contour**(aes(z = z))
x, y, z, alpha, colour, linetype, size, weight

m + **geom_raster**(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

m + **geom_tile**(aes(fill = z))
x, y, alpha, color, fill, linetype, size

Stats - An alternative way to build a layer

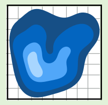
Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. stat_bin(geom="bar") does the same as geom_bar(stat="bin")

stat function · layer specific mappings · variable created by transformation



i + **stat_density2d**(aes(fill = ..level..), geom = "polygon", n = 100)

geom for layer · parameters for stat

1D distributions

a + **stat_bin**(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + **stat_bindot**(binwidth = 1, binaxis = "x")
x, y, | ..count.., ..ncount..
a + **stat_density**(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + **stat_bin2d**(bins = 30, drop = TRUE)
x, y, fill | ..count.., ..density..
f + **stat_binhex**(bins = 30)
x, y, fill | ..count.., ..density..
f + **stat_density2d**(contour = TRUE, n = 100)
x, y, color, size | ..level..

3 Variables

m + **stat_contour**(aes(z = z))
x, y, z, order | ..level..
m+ **stat_spoke**(aes(radius= z, angle = z))
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + **stat_summary_hex**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..
m + **stat_summary2d**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

Comparisons

g + **stat_boxplot**(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + **stat_ydensity**(adjust = 1, kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

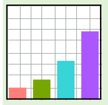
f + **stat_ecdf**(n = 40)
x, y | ..x.., ..y..
f + **stat_quantile**(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
x, y | ..quantile.., ..x.., ..y..
f + **stat_smooth**(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + **stat_function**(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
x | ..y..
f + **stat_identity()**
ggplot() + **stat_qq**(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..x.., ..y..
f + **stat_sum()**
x, y, size | ..size..
f + **stat_summary**(fun.data = "mean_cl_boot")
f + **stat_unique()**

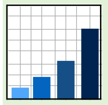
Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



n <- b + **geom_bar**(aes(fill = fl))
n

scale_ · aesthetic to adjust · prepackaged scale to use · scale specific arguments



n + **scale_fill_manual(**
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), **breaks** =c("d", "e", "p", "r"),
name = "fuel", **labels** = c("D", "E", "P", "R"))

range of values to include in mapping · title to use in legend/axis · labels to use in legend/axis · breaks to use in legend/axis

General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

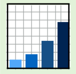
Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

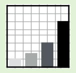
Color and fill scales

Discrete

n <- b + **geom_bar(**
aes(fill = fl))

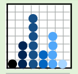


n + **scale_fill_brewer(**
palette = "Blues")
For palette choices:
library(RcolorBrewer)
display.brewer.all()

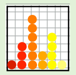


n + **scale_fill_grey(**
start = 0.2, end = 0.8,
na.value = "red")

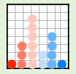
Continuous



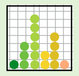
o <- a + **geom_dotplot(**
aes(fill = ..x..))



o + **scale_fill_gradient(**
low = "red",
high = "yellow")

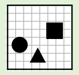


o + **scale_fill_gradient2(**
low = "red", hight = "blue",
mid = "white", midpoint = 25)



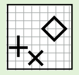
o + **scale_fill_gradientn(**
colo**u**rs = terrain.colors(6))
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape scales



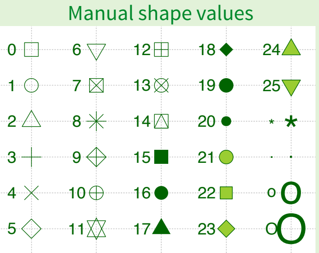
p <- f + **geom_point(**
aes(shape = fl))

p + **scale_shape(**
solid = FALSE)

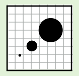


p + **scale_shape_manual(**
values = c(3:7))
Shape values shown in chart on right

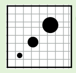
Manual shape values



Size scales



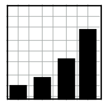
q <- f + **geom_point(**
aes(size = cyl))



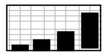
q + **scale_size_area**(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

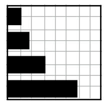
r <- b + geom_bar()



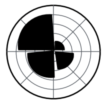
r + **coord_cartesian**(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system



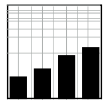
r + **coord_fixed**(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units



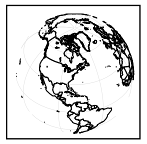
r + **coord_flip()**
xlim, ylim
Flipped Cartesian coordinates



r + **coord_polar**(theta = "x", direction=1)
theta, start, direction
Polar coordinates



r + **coord_trans**(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.



z + **coord_map**(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + **facet_grid(. ~ fl)**
facet into columns based on fl

t + **facet_grid(year ~ .)**
facet into rows based on year

t + **facet_grid(year ~ fl)**
facet into both rows and columns

t + **facet_wrap(~ fl)**
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(y ~ x, **scales = "free"**)
x and y axis limits adjust to individual facets
- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, **labeller = label_both)**

fl: c · fl: d · fl: e · fl: p · fl: r

t + facet_grid(. ~ fl, **labeller = label_bquote(**alpha ^ .(x)**))**

α^c · α^d · α^e · α^p · α^r

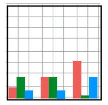
t + facet_grid(. ~ fl, **labeller = label_parsed)**

c · d · e · p · r

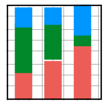
Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

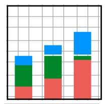
s <- ggplot(mpg, aes(fl, fill = drv))



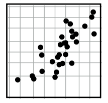
s + geom_bar(**position = "dodge"**)
Arrange elements side by side



s + geom_bar(**position = "fill"**)
Stack elements on top of one another, normalize height



s + geom_bar(**position = "stack"**)
Stack elements on top of one another



f + geom_point(**position = "jitter"**)
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + geom_bar(**position = position_dodge(width = 1)**)

Labels

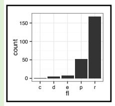
t + **ggtitle**("New Plot Title")
Add a main title above the plot
t + **xlab**("New X label")
Change the label on the X axis
t + **ylab**("New Y label")
Change the label on the Y axis
t + **labs**(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

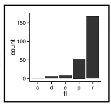
Legends

t + **theme**(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
t + **guides**(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + **scale_fill_discrete**(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.

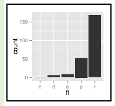
Themes



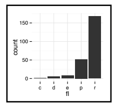
r + **theme_bw()**
White background with grid lines



r + **theme_classic()**
White background no gridlines



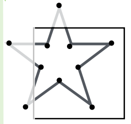
r + **theme_grey()**
Grey background (default theme)



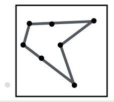
r + **theme_minimal()**
Minimal theme

ggthemes - Package with additional ggplot2 themes

Zooming



Without clipping (preferred)
t + **coord_cartesian(**
xlim = c(0, 100), ylim = c(10, 20))



With clipping (removes unseen data points)
t + **xlim**(0, 100) + **ylim**(10, 20)
t + **scale_x_continuous**(limits = c(0, 100)) +
scale_y_continuous(limits = c(0, 100))

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 end 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-imaginging of the data frame.
- **stringr** provides a cohesive set of functions designed to make working with strings as easy as posssible
- **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

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

Loading in the data

`> library(datasets)`	Load the datasets package
`> library(gapminder)`	Load the gapminder package
`> attach(iris)`	Attach iris data to the R search path

dplyr

Filter

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

```
> iris %>%
     filter(Species=="virginica")
> iris %>%
     filter(Species=="virginica",
            Sepal.Length > 6)
```

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.

```
> iris %>%
     arrange(Sepal.Length)
> iris %>%
     arrange(desc(Sepal.Length))
```

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 `%>%`:

```
> iris %>%
    filter(Species=="virginica") %>%
    arrange(desc(Sepal.Length))
```

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.

```
> iris %>%
    mutate(Sepal.Length=Sepal.Length*10)
> iris %>%
    mutate(SLMm=Sepal.Length*10)
```

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.

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

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:

```
> 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))
```

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

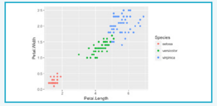
Scatter plots allow you to compare two variables within your data. To do this with ggplot2, you use `geom_point()`

```
> iris_small <- iris %>%
   filter(Sepal.Length > 5)
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width)) +
    geom_point()
```

Compare petal width and length

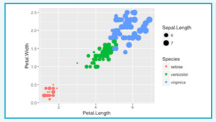
Additional Aesthetics

- Color



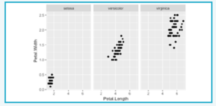
```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width,
                         color=Species)) +
    geom_point()
```

- Size



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

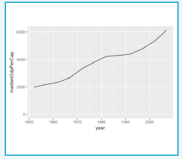
Faceting



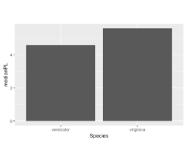
```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width)) +
    geom_point()+
    facet_wrap(~Species)
```

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)
```



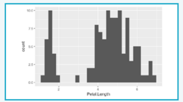
Bar Plots



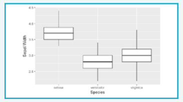
```
> by_species <- iris %>%
    filter(Sepal.Length>6) %>%
    group_by(Species) %>%
    summarize(medianPL=median(Petal.Length))
> ggplot(by_species, aes(x=Species,
                         y=medianPL)) +
    geom_col()
```

Histograms

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



Box Plots



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

DataCamp
Learn R for Data Science Interactively