R-bloggers
R news and tutorials contributed by (750) R bloggers

- Home
- About
- RSS
- add your blog!
- Learn R
- R jobs���
- Contact us

Welcome!

Follow @rbloggers 53.4K

Here you will find daily **news and tutorials about R**, contributed by over 750 bloggers.
There are many ways to **follow us -**
By e-mail:

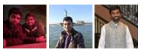
Your e-mail here
Subscribe
45931 readers
BY FEEDBURNER

On Facebook:

R blog…
70K likes

Liked

You and 3 other friends like this



If you are an R blogger yourself you are invited to add your own R content feed to this site (**Non-English** R bloggers should add themselves- here)

Jobs for R-users

- Financial Analyst/Modeler @ Mesa, Arizona, U.S.
- Research volunteer in Cardiac Surgery @ Philadelphia, Pennsylvania, U.S.
- Fit GLM distribution models to data using R
- Financial services startup looking for freelance Shiny/R/UI developer
- Data Scientists – PhD Paradise –

Germany

Popular Searches

- eclipse

Recent Posts

- Tutorial: Publish an R function as a SQL Server stored procedure with the sqlrutils package
- Soccer data sparring: Scraping, merging and analyzing exercises
- Seeking guidance in choosing and evaluating R packages
- Pets or livestock? Naming your RMarkdown chunks
- How to make best use of the byte compiler in R
- Bayesian Simple Linear Regression with Gibbs Sampling in R
- US Immigration Enforcement – Part 1
- Determine the CV of a Calculated Lab Reportable – Bioavailable Testosterone
- Exploring Assumptions of K-means Clustering using R
- Implementing Parallel Processing in R
- Radial kernel Support Vector Classifier
- Elegant correlation table using xtable R package
- R and Interactive Graphics
- Geospatial Queries using Pymongo in R
- More on "The Part-Time R-User"

Other sites

- SAS blogs
- Jobs for R-users

Useful dplyr Functions

(w/examples)

July 10, 2017
By S. Richter-Walsh

(This article was first published on **Environmental Science and Data Analytics**, and kindly contributed to R-bloggers)

The R package ***dplyr*** is an extremely useful resource for data cleaning, manipulation, visualisation and analysis. It contains a large number of very useful functions and is, without doubt, one of my top 3 R packages today (***ggplot2*** and ***reshape2*** being the others). When I was learning how to use dplyr for the first time, I used DataCamp which offers some fantastic interactive courses on R. I also made use of a decent data wrangling cheat sheet which can be found here.

There are many useful functions contained within the dplyr package. This post does not attempt to cover them all but does look at the major functions that are commonly used in data manipulation tasks. These are:

```
select()
filter()
mutate()
group_by()
summarise()
arrange()
join()
```

The data used in this post are taken from the UCI Machine Learning Repository and contain census information from 1994 for the USA. The dataset can be used for classification of income class in a machine learning setting and can be obtained here.

```
require(dplyr)

# Data file
file <- "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"

# Some sensible variable names
df_names <- c("age", "wrkclass", "fnlweight", "education_lvl", "edu_score",
 "marital_status", "occupation", "relationship", "ethnic", "gender",
 "cap_gain", "cap_loss", "hrs_wk", "nationality", "income")

# Import the data
df <- read.csv(file, header = F,
 sep = ",",
 na.strings = c(" ?", " ", ""),
 row.names = NULL,
 col.names = df_names)
```

Many data manipulation tasks in dplyr can be performed with the assistance of the forward-pipe operator (%>%). The pipe was first introduced in the ***magrittr*** package and has since been included in the dplyr package. It is an incredibly useful tool for fluid data manipulation and results in highly readable code.

The census dataset requires a bit of preprocessing to get it ready for classification algorithms. This post does not cover preprocessing nor does it include predictive modelling.

The first function I would like to introduce removes duplicate entries which, in fact, *is* a preprocessing step one may carry out in a data analysis. It is so useful that it must be included.

```
# Remove duplicate rows and check number of rows
df %>% distinct() %>% nrow()

# Drop duplicate rows and assign to new dataframe object
df_clean <- df %>% distinct()

# Drop duplicates based on one or more variables
df %>% distinct(gender, .keep_all = T)
df %>% distinct(gender, education_lvl, .keep_all =  T)
```

Taking random samples of data is easy with dplyr.

```
# Sample random rows with or without replacement
sample_n(df, size = nrow(df) * 0.7, replace = F)
sample_n(df, size = 20, replace = T)

# Sample a proportion of rows with or without replacement
```

```
sample_frac(df, size = 0.7, replace = F)
sample_frac(df, size = 0.8, replace = T
```

Renaming variables is also easy with dplyr.

```
# Rename one or more variables in a dataframe
df <- df %>%
 rename("INCOME" = "income")

df <- df %>%
 rename("INCOME" = "income", "AGE" = "age")
```

The main “verbs” of dplyr are now introduced. Let’s begin with the **select()** verb which filters a dataframe by column.

```
# Select specific columns (note that INCOME is the new name from earlier)
df %>%
 select(education_lvl, INCOME)

# With dplyr 0.7.0 the pull() function extracts a variable as a vector
df %>%
 pull(age)

# Drop a column using the - operator (variable can be referenced by name or column position)
df %>%
 select(-edu_score)

df %>%
 select(-1, -4)

df %>%
 select(-c(2:6))
```

Some useful helper functions are available in dplyr and can be used in conjunction with the select() verb. Here are some quick examples.

```
# Select columns with their names starting with "e"
df %>%
 select(starts_with("e"))

# The negative sign works for dropping here too
df %>%
 select(-starts_with("e"))

# Select columns with some pattern in the column name
df %>%
 select(contains("edu"))

# Reorder data to place a particular column at the start followed by all others using everything()
df %>%
 select(INCOME, everything())

# Select columns ending with a pattern
df %>%
 select(ends_with("e"))

df %>%
 select(ends_with("_loss"))
```

The next major verb we look at is **filter()** which, surprisingly enough, filters a dataframe by row based on one or more conditions.

```
# Filter rows to retain observations where age is greater than 30
df %>%
 filter(age > 30)

# Filter by multiple conditions using the %in% operator (make sure strings match)
df %>%
 filter(relationship %in% c(" Unmarried", " Wife"))

# You can also use the OR operator (|)
df %>%
 filter(relationship == " Husband" | relationship == " Wife")

# Filter using the AND operator
df %>%
 filter(age > 30 & INCOME == " >50K")

# Combine them too
df %>%
 filter(education_lvl %in% c(" Doctorate", " Masters") & age > 30)

# The NOT condition (filter out doctorate holders)
df %>%
 filter(education_lvl != " Doctorate")

# The grepl() function can be conveniently used with filter()
df %>%
 filter(grepl(" Wi", relationship))
```

Next, we look at the **summarise()** verb which allows one to dynamically summarise groups of data and even pipe groups to ggplot data visualisations.

```
# The summarise() verb in dplyr is useful for summarising grouped data
df %>%
 filter(INCOME == " >50K") %>%
 summarise(mean_age = mean(age),
           median_age = median(age),
           sd_age = sd(age))

# Summarise multiple variables using summarise_at()
df %>%
 filter(INCOME == " >50K") %>%
 summarise_at(vars(age, hrs_wk),
 funs(n(),
      mean,
      median))

# We can also summarise with custom functions
# The . in parentheses represents all called variables
df %>%
 summarise_at(vars(age, hrs_wk),
 funs(n(),
      missing = sum(is.na(.)),
      mean = mean(., na.rm = T)))

# Create a new summary statistic with an anonymous function
df %>%
 summarise_at(vars(age),
 function(x) { sum((x - mean(x)) / sd(x)) })

# Summarise conditionally using summarise_if()
df %>%
 filter(INCOME == " >50K") %>%
 summarise_if(is.numeric,
              funs(n(),
                   mean,
                   median))

# Subset numeric variables and use summarise_all() to get summary statistics
ints <- df[sapply(df, is.numeric)]
summarise_all(ints,
              funs(mean,
                   median,
                   sd,
                   var))
```

Next up is the **arrange()** verb which is useful for sorting data in ascending or descending order (ascending is default).

```
# Sort by ascending age and print top 10
df %>%
 arrange(age) %>%
 head(10)

# Sort by descending age and print top 10
df %>%
 arrange(desc(age)) %>%
 head(10)
```

The **group_by()** verb is useful for grouping together observations which share common characteristics.

```
# The group_by verb is extremely useful for data analysis
df %>%
 group_by(gender) %>%
 summarise(Mean = mean(age))

df %>%
 group_by(relationship) %>%
 summarise(total = n())

df %>%
 group_by(relationship) %>%
 summarise(total = n(),
           mean_age = mean(age))
```

The **mutate()** verb is used to create new variables from existing local variables or global objects. New variables, such as sequences, can be also specified within mutate().

```
# Create new variables from existing or global variables
df %>%
 mutate(norm_age = (age - mean(age)) / sd(age))


# Multiply each numeric element by 1000 (the name "new" is added to the original variable name)
df %>%
 mutate_if(is.numeric,
           funs(new = (. * 1000))) %>%
           head()
```

The **join()** verb is used to merge rows from disjoint tables which share a primary key ID or some other common variable. There are many join variants but I will consider just left, right, inner and full joins.

```
# Create ID variable which will be used as the primary key
df <- df %>%
 mutate(ID = seq(1:nrow(df))) %>%
 select(ID, everything())
```

```
# Create two tables (purposely overlap to facilitate joins)
table_1 <- df[1:50 , ] %>%
 select(ID, age, education_lvl)

table_2 <- df[26:75 , ] %>%
 select(ID, gender, INCOME)

# Left join joins rows from table 2 to table 1 (the direction is implicit in the argument order)
left_join(table_1, table_2, by = "ID")

# Right join joins rows from table 1 to table 2
right_join(table_1, table_2, by = "ID")

# Inner join joins and retains only complete cases
inner_join(table_1, table_2, by = "ID")

# Full join joins and retains all values
full_join(table_1, table_2, by = "ID"
```

That wraps up a brief demonstration of some of dplyr’s excellent functions. For additional information on the functions and their arguments, check out the help documentation using the template: ?

References

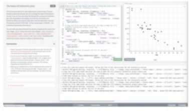
Hadley Wickham, Romain Francois, Lionel Henry and Kirill Müller (2017). dplyr: A
Grammar of Data Manipulation. R package version 0.7.0.
https://CRAN.R-project.org/package=dplyr

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New
York, 2009.

Related

Data Manipulation in R with dplyr – Part 1
dplyr is one of the packages in R that makes R so loved by data scientists. It has three main goals:
December 17, 2015
In "R bloggers"

Enernoc smart meter data - forecast electricity consumption with similar day approach in R
Deployment of smart grids gives space to an occurrence of new methods of machine learning and data
November 11, 2016
In "R bloggers"



Tutorials for learning R
December 10, 2015
In "R guest posts"

135 SHARES Share Tweet

Comments are closed.

Search R-bloggers

Custom Search

Recent popular posts

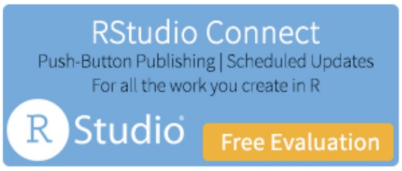
- Elegant correlation table using xtable R

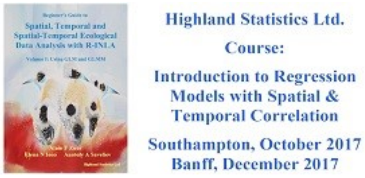
- package
- Exploring Assumptions of K-means Clustering using R
- R and Interactive Graphics
- Bayesian Simple Linear Regression with Gibbs Sampling in R

Most visited articles of the week

1. How to write the first for loop in R
2. Installing R packages
3. Using apply, sapply, lapply in R
4. Tutorials for learning R
5. How to perform a Logistic Regression in R
6. In-depth introduction to machine learning in 15 hours of expert videos
7. Elegant correlation table using xtable R package
8. Exploring Assumptions of K-means Clustering using R
9. How to Make a Histogram with Basic R

Sponsors


RStudio Connect
Push-Button Publishing | Scheduled Updates
For all the work you create in R
R Studio
Free Evaluation


Highland Statistics Ltd.
Course:
Introduction to Regression Models with Spatial & Temporal Correlation
Southampton, October 2017
Banff, December 2017

Register for free updates in all areas of statistics
STATISTICS VIEWS
Bringing Statistics Together
WILEY

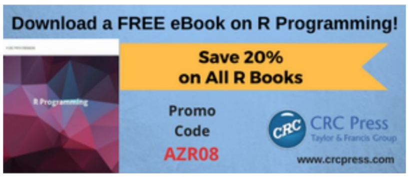
AI Learning
Accelerator
gain AI Insights
enroll for free
#ODSC

Accelerate AI / NOV 12 - 14
SAN FRANCISCO
Learn how Data Science and AI
impact your industry.
ai.odsc.com

R Courses for Professionals
Download R templates for machine learning
The fastest way to learn
data science!
VIEW SNEAK PEAK
DATA SOCIETY

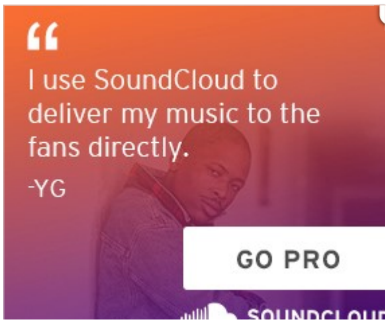
data science
environment
Integrate
Data Science into
Business Processes

FANTASY FOOTBALL ANALYTICS
USING STATISTICS & R
TO IMPROVE YOUR
FANTASY FOOTBALL TEAM


Download a FREE eBook on R Programming!
Save 20%
on All R Books
Promo
Code
AZR08
CRC Press
Taylor & Francis Group
www.crcpress.com

NYC DATA SCIENCE
ACADEMY
Become a Data Scientist
Develop expertise in R, Python, Hadoop & Spark
in just 12 Weeks
Apply for Data Science Bootcamp

AUTOMATIC MACHINE LEARNING
ML JAR
Train & tune predictive models with one click



Jobs for R users

- Financial Analyst/Modeler @ Mesa, Arizona, U.S.
- Research volunteer in Cardiac Surgery @ Philadelphia, Pennsylvania, U.S.
- Fit GLM distribution models to data using R
- Financial services startup looking for freelance Shiny/R/UI developer
- Data Scientists – PhD Paradise – Germany
- Technical Project Manager
- Software Developer

Full list of contributing R-bloggers
R-bloggers was founded by Tal Galili, with gratitude to the R community.
Is powered by WordPress using a bavotasan.com design.