



# 8. Introduction to R Packages

Ken Rice  
Tim Thornton

University of Washington

*Seattle, July 2019*

# In this session

---

'Base' R includes pre-installed packages that allow for a fully functioning statistical environment in which a variety of analyses can be conducted.

Thousands of user-contributed extension 'packages' are available that provide enhanced functionality with R. We'll discuss;

- Loading packages, and seeing what's in them
- Finding and installing packages
- Some examples, using available packages on [CRAN](#)

# CRAN Packages

---

One factor in R's success is the way it allows authors to rapidly develop and disseminate packages, containing resources (with documentations) that will be useful to others.

R packages are collections of functions, data, and compiled code, in a well-defined format. These are made available – either direct from R or via a web browser – through the [Comprehensive R Archive Network \(CRAN\)](#)

At the time of writing there are 14,502 packages available on CRAN!

# CRAN Packages

---

Base R's recommended packages (below) are available on CRAN:

**KernSmooth**

Functions for kernel smoothing (and density estimation) corresponding to the book “Kernel Smoothing” by M. P. Wand and M. C. Jones, 1995.

**MASS**

Functions and datasets from the main package of Venables and Ripley, “Modern Applied Statistics with S”. (Contained in the `VR` bundle for R versions prior to 2.10.0.)

**Matrix**

A Matrix package. (Recommended for R 2.9.0 or later.)

**boot**

Functions and datasets for bootstrapping from the book “Bootstrap Methods and Their Applications” by A. C. Davison and D. V. Hinkley, 1997, Cambridge University Press.

**class**

Functions for classification (k-nearest neighbor and LVQ). (Contained in the `VR` bundle for R versions prior to 2.10.0.)

**cluster**

Functions for cluster analysis.

**codetools**

Code analysis tools. (Recommended for R 2.5.0 or later.)

**foreign**

Functions for reading and writing data stored by statistical software like Minitab, S, SAS, SPSS, Stata, Systat, etc.

**lattice**

Lattice graphics, an implementation of Trellis Graphics functions.

**mgcv**

Routines for GAMs and other generalized ridge regression problems with multiple smoothing parameter selection by GCV or UBRE.

**nlme**

Fit and compare Gaussian linear and nonlinear mixed-effects models.

**nnet**

Software for single hidden layer perceptrons (“feed-forward neural networks”), and for multinomial log-linear models. (Contained in the `VR` bundle for R versions prior to 2.10.0.)

**rpart**

Recursive PARTitioning and regression trees.

**spatial**

Functions for kriging and point pattern analysis from “Modern Applied Statistics with S” by W. Venables and B. Ripley. (Contained in the `VR` bundle for R versions prior to 2.10.0.)

**survival**

Functions for survival analysis, including penalized likelihood.

... though *most* downloads provide them automatically.

## Example: the foreign package

---

The foreign package is one of those recommended with base R, and also lives on CRAN. It contains several useful functions that import and export data, to/from a variety of formats.

To use these functions, first load it into your current session;

```
library("foreign")
```

- Its function `read.spss()` reads SPSS data files;

```
dat.spss <- read.spss("http://faculty.washington.edu/tathornt/sisg/hsb2.sav",  
                    to.data.frame=TRUE)
```

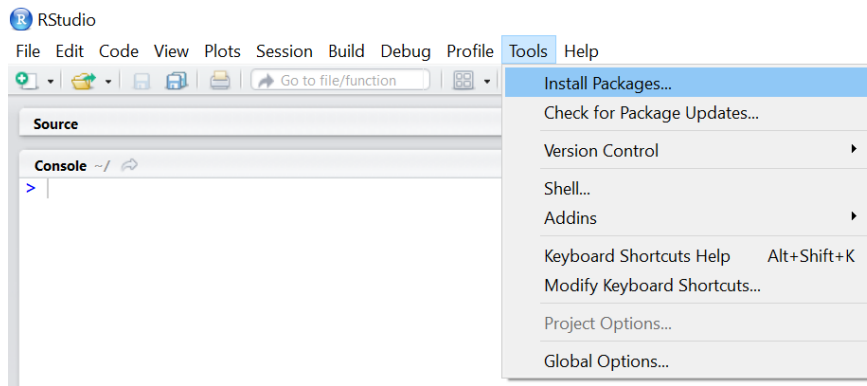
- Its function `read.dta()` reads in Stata files;

```
dat.dta <- read.dta("http://faculty.washington.edu/tathornt/sisg/hsb2.dta")
```

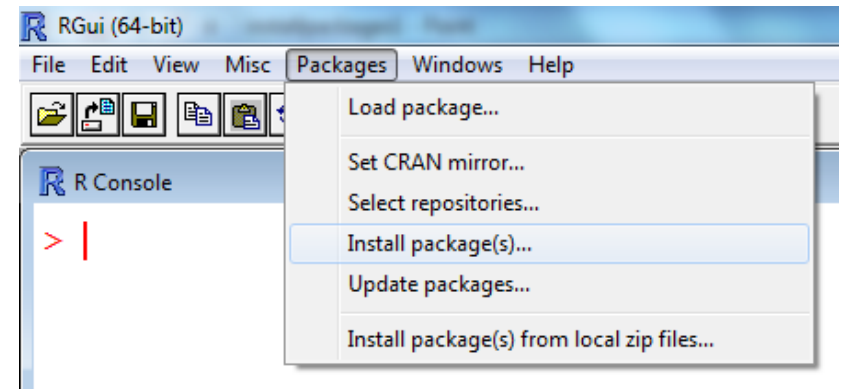
- After loading a package, look up the help files for commands with e.g. `?read.spss` and `?read.dta`
- If you don't *know* its commands, try `library(help="foreign")` or `help(package="foreign")`, or look online e.g. via Google

# Installing packages

Sometimes, we need more than the recommended packages. When you find an R package (e.g. via Google) of use to you, first install it – here, by following the drop-down menus;



RStudio



Base R's GUI

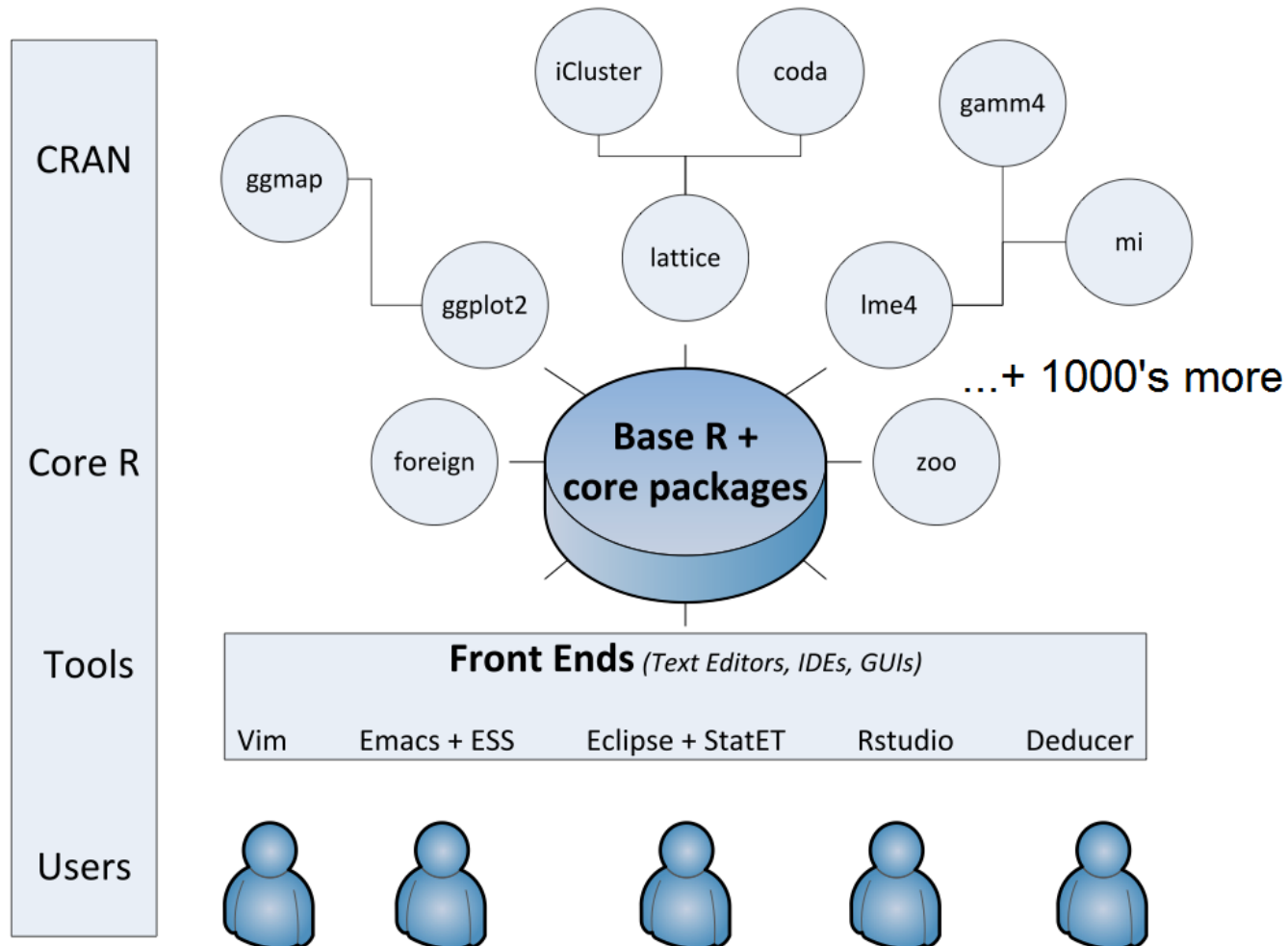
...*after* going online! The coded way is, for the `hexbin` package;

```
install.packages("hexbin")
```

- In base R, expect to specify a CRAN mirror site
- Write-privileges can be an issue; the defaults are sensible

# Installing packages

The wider world of R packages and tools;



# Vignettes

---

In addition to the ‘plain vanilla’ help files, many (but not all) packages include ‘vignettes’. These documents give an overview of the package, usually including some worked examples.

To find vignettes, use `browseVignettes()` – noting the capitalization. For example, once you found out that the `hexbin` package does things you are interested in;

```
install.packages("hexbin") # download it (once, for each version of R)
library("hexbin")         # make it available in the current R session
browseVignettes("hexbin") # see some tutorials
help(package="hexbin")    # get to the help files
```

The `vignette()` function also finds vignettes – but it’s clunkier;

```
vignette(package="hexbin")
vignette(topic="hexagon_binning")
```



# Masking: overlapping object names

---

With so many packages by different authors, it's inevitable that, sometimes, multiple packages use the same name for distinct objects. For example, if you were using both the `plyr` and `reshape` packages in the same session;

```
> library("plyr")
> library("reshape")
Attaching package: 'reshape'
```

The following objects are masked from 'package:plyr':  
`rename`, `round_any`

- *Masking* means two objects have the same name. R uses the one loaded most recently, i.e. the `reshape` version, here
- If you are going to use these functions, pay attention!
- You may not *know* you will use these functions – perhaps `round_any()` is called from within other functions
- If you *really* need to, use the masked version directly with e.g. `plyr::rename()` – or use the `get()` function. But these are best avoided, unless you are an expert – perhaps writing a complicated package of your own

# But what's already been loaded?

---

The `sessionInfo()` command will tell you this – and what version of R you are using;

```
> sessionInfo()
R version 3.4.1 (2017-06-30)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 15063)
```

```
Matrix products: default
```

```
locale:
```

```
[1] LC_COLLATE=English_United States.1252
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets  methods    base
```

```
other attached packages:
```

```
[1] hexbin_1.27.1
```

```
loaded via a namespace (and not attached):
```

```
[1] compiler_3.4.1  tools_3.4.1      grid_3.4.1      lattice_0.20-35
```

NB if you get warnings about R versions, try `update.packages()` to update packages you have, `new.packages()` to see what's available, or (more rarely) `old.packages()` to revert.

## But what's already been loaded?

---

To see the order in which packages in the current R session were attached, use `search()`;

```
> search()
[1] ".GlobalEnv"          "package:hexbin"      "package:lattice"
[4] "package:grid"        "package:foreign"    "tools:RGUI"
[7] "package:stats"      "package:graphics"   "package:grDevices"
[10] "package:utils"      "package:datasets"   "package:methods"
[13] "Autoloads"          "package:base"
```

- `.GlobalEnv` is the familiar command line environment – easy to inspect, with RStudio
- It's also *possible* to `attach()` datasets, which also appear in the `search()` list. This may save you some typing, but beware masking problems if you use object `X` and also `mydata$X` – so this approach is not recommended. (Beware out-of-date teaching resources!)

# Example: the survey package

---

The `survey` package includes a data set named `api` containing California Academic Performance Index for 6194 schools;

```
library("survey")
help(package="survey")      # look for the "api" entry
data(api, package="survey") # make the apipop dataset available
```

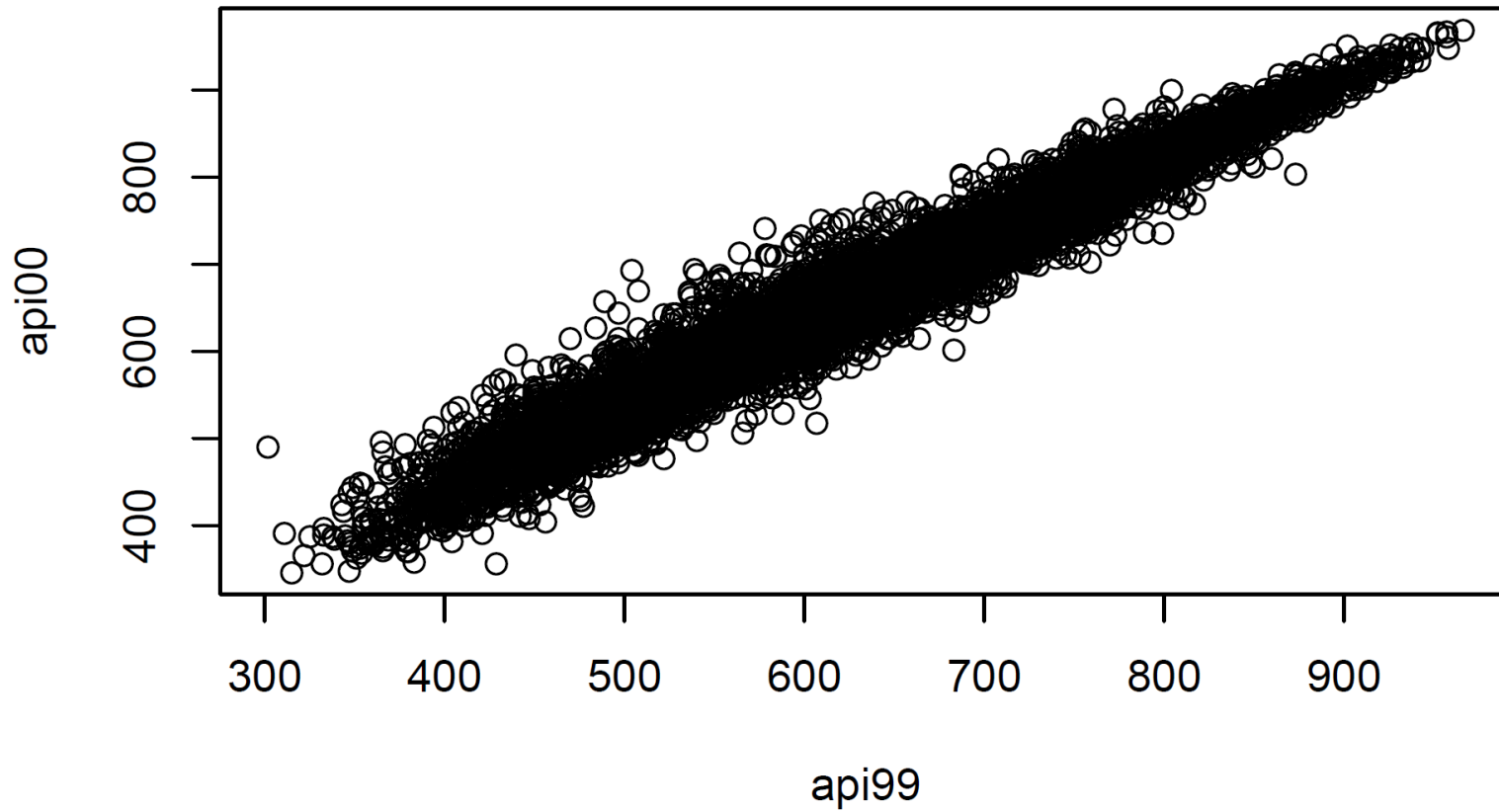
Plotting the data, perhaps colored by school type, we see how crowded scatterplots can be with large data sets;

```
> summary(apipop[,c("api99", "api00", "stype")])
      api99          api00      stype
Min.   :302.0    Min.   :346.0    E:4421
1st Qu.:527.0    1st Qu.:565.0    H: 755
Median :631.0    Median :667.0    M:1018
Mean   :631.9    Mean   :664.7
3rd Qu.:734.0    3rd Qu.:761.0
Max.   :966.0    Max.   :969.0
> plot(api00~api99, data=apipop)
> colors <- c("tomato", "forestgreen", "purple")[apipop$stype]
> plot(api00~api99, data=apipop, col=colors)
```

(Keen people: note we recode E/H/M to 1/2/3 to tomato/forestgreen/purple.)

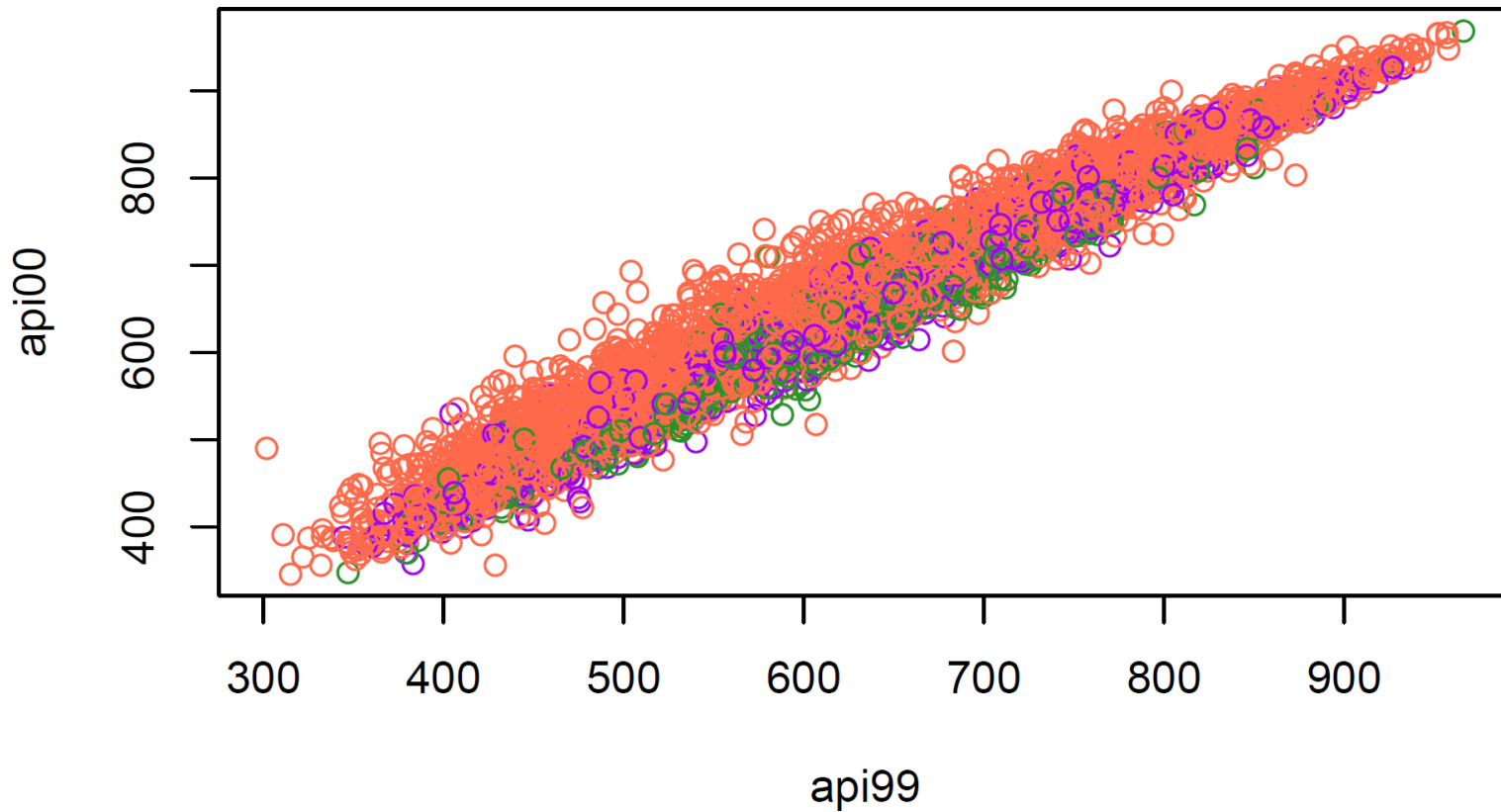
# Example: the survey package

---



# Example: the survey package

---



## Example: the hexbin package

---

When there are many data points and significant overlap, scatterplots become less useful.

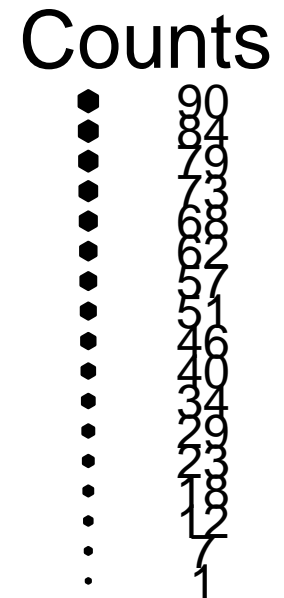
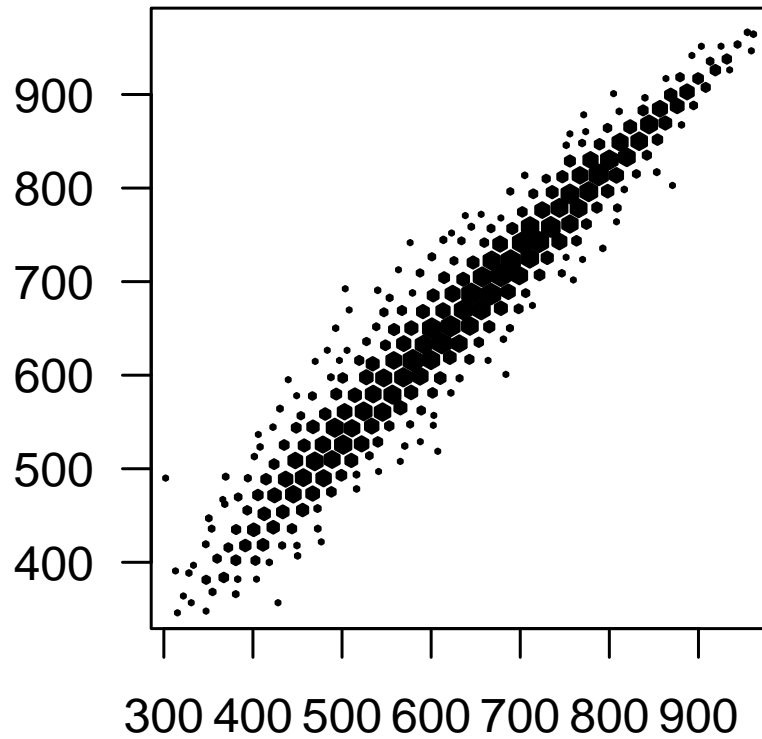
The `hexbin()` function in the `hexbin` package provides a way to aggregate the points in a scatterplot. It computes the number of points in each hexagonal bin.

```
library("hexbin")  
with(apipop, plot(hexbin(api99,api00), style="centroids"))
```

The `style="centroids"` option plots filled hexagons, at the centroid of each bin. The sizes of the plotted hexagons are proportional to the number of points in each bin.

# Example: the hexbin package

---





# Summary

---

- Many functions in R live in optional `packages`, and thousands of packages are available on CRAN for downloading
- The `install.packages()` function is used for installing an extension package
- The `library()` function lists packages, shows help, or loads packages from the package library
- If/when masking occurs, `packagename::function()` can be used to access a function in a package that has been masked due to another loaded package having a function with the same