Building R Packages

- Primary reference for creating R Packages: *Writing R Extensions* (distributed with R).
- Create a "source package" containing "meta-information" describing the package; R code; data; documentation; and possibly other components.
- The source package is then checked and built into a universal `.tar.gz` format that can be distributed and installed under any operating system on which R runs, or built into a "binary" package for a particular system, such as Windows.

- Under Windows, the ability to build packages requires installing software in addition to R: a working LaTeX, Perl, Unix-like command-line tools, a C++ compiler, and the Microsoft Windows help compiler, all of which are readily available. See the *R Installation and Administration* manual.
- To check a package, assuming that the R `bin` directory is on the Windows path, and that the package subdirectory is in the current directory:

```
R CMD check package-name
```

- To build the package, producing a `tar.gz` file:

```
R CMD build --force package-name
```

- To build a Windows binary package, producing a `.zip` file:

```
R CMD build --binary package-name
```

- To install directly:

```
R CMD INSTALL package-name
```


Package Structure

- An R source package consists of a directory containing:
  - A `DESCRIPTION` file with meta information such as the package name, version, and author, and other packages on which the package depends.
  - An optional `NAMESPACE` file (for a package with a namespace), enumerating, e.g., the objects that are "exported" from the package.
  - An `R` subdirectory containing `.R` files with R code for creating objects such as functions.

- A `man` ("manual") subdirectory that includes `.Rd` documentation files (using LaTeX-like markup). All public objects in the package should be documented.

- A `data` subdirectory containing data objects, such as text files that can be read as R data frames. Thus, a file named `Duncan.txt` would produce a data frame named `Duncan`.

- An `inst` ("install") subdirectory containing arbitrary files and subdirectories to be installed in the package (e.g., a `CHANGES` file documenting the history of changes to the package).

- Possibly other subdirectories (e.g., for compiled C code).

- The function `package.skeleton()` creates the "skeleton" of a source package for objects that are currently in the R workspace.

- Example: The **matrixDemos** package.