Introduction to dplyr and magrittr

Denver R Users Group

www.meetup.com/DenverRUG

Peter DeWitt

peter.dewitt@ucdenver.edu

1 July 2014

Goals:

- Showcase `dplyr`, compare the ease of use compared to base R.
- Introduce the data manipulation grammar and philosophy behind `dplyr`
- Illustrate the usefulness of the forward-piping operator which is part of `dplyr` and extended further in `magrittr`.

dplyr: a grammar of data manipulation

- Authored by Hadley Wickham and Romain Francois
- Current CRAN version 0.2

dplyr: a grammar of data manipulation

- Authored by Hadley Wickham and Romain Francois
- Current CRAN version 0.2
- Paraphrasing from a post on the RStudio blog `http://blog.rstudio.org/2014/01/17/introducing-dplyr`
  - `dplyr` is the next iteration of `plyr`
  - focuses only on `data.frames`
  - faster, thanks in part to Francois work in `Rcpp`, some use of multiple processors.
  - improved API.
  - interface with remote database (PostgreSQL, MySQL, SQLite, and Google bigquery) tables using the same verbs for interacting with `data.frames`. (Extendible to other backends)
  - Common operations:
    - `group_by`, `summarize`, `mutate`, `filter`, `select`, and `arrange`.

Data Import

dplyr does not have special tools for reading in data, but, if you need to rbind sets together...

```
# FAAs wildlife strikes on aircraft since 1990.  The data
# can be downloaded, in a Microsoft Access DB,  from
# http://www.faa.gov/airports/airport_safety/wildlife/database/
# Tables in the DB were exported to csv files.
# A data dictionary, in an Excel file, was also
# included in the download from faa.gov

# column classes are set (in R code not shown) to ensure
# that each column of the imported data is of the same class
wls.90.99 <-
  read.csv("../data/STRIKE_REPORTS (1990-1999).csv",
           colClasses = clclss)
wls.00.09 <-
  read.csv("../data/STRIKE_REPORTS (2000-2009).csv",
           colClasses = clclss)
wls.10.14 <-
  read.csv("../data/STRIKE_REPORTS (2010-Current).csv",
           colClasses = clclss)
```

Data Import

```
# Base does not require the columns to be of the same class,
# only the same name
# dplyr requires that the columnns are of the same class.
dim(wls.90.99)

## [1] 30150    94

nrow(wls.90.99) + nrow(wls.00.09) + nrow(wls.10.14)

## [1] 142911

bnchmrk <-
  benchmark(base = rbind(wls.90.99, wls.00.09, wls.10.14),
            dplyr = rbind_list(wls.90.99, wls.00.09, wls.10.14),
            replications = 100)
bnchmrk[, c("test", "replications", "elapsed", "relative")]

##    test replications elapsed relative
## 1  base          100   92.98    3.968
## 2 dplyr          100   23.43    1.000
```

Data Import

```
wls_df <- rbind(wls.90.99, wls.00.09, wls.10.14)
class(wls_df)

## [1] "data.frame"

wls <- rbind_list(wls.90.99, wls.00.09, wls.10.14)
class(wls)

## [1] "data.frame"

# A data frame tbl wraps a local data frame. The main
# advantage to using a tbl_df over a regular data frame is
# the printing: tbl objects only print a few rows and all
# the columns that fit on one screen, providing describing
# the rest of it as text. [source: R help doc]
wls_tbl_df <- tbl_df(wls)
class(wls_tbl_df)

## [1] "tbl_df"     "tbl"         "data.frame"
```

Data Printing

```
# print(wls_df)  # takes a long time, not helpful
# head(wls_df)   # two many columns to be useful
print(wls_tbl_df, n = 2)

## Source: local data frame [142,911 x 94]
##
##    INDEX_NR OPID          OPERATOR     ATYPE AMA AMO EMA EMO
## 1    100000  AAL AMERICAN AIRLINES     B-727 148  10  34  10
## 2    100001  UAL   UNITED AIRLINES B-737-300 148  24  10  01
## ..      ...  ...               ...       ... ... ... ... ...
## Variables not shown: AC_CLASS (chr), AC_MASS (int), NUM_ENGS
##   (chr), TYPE_ENG (chr), ENG_1_POS (chr), ENG_2_POS (int),
##   ENG_3_POS (chr), ENG_4_POS (int), REG (chr), FLT (chr),
##   REMAINS_COLLECTED (lgl), REMAINS_SENT (lgl), INCIDENT_DATE
##   (chr), INCIDENT_MONTH (int), INCIDENT_YEAR (int),
##   TIME_OF_DAY (chr), TIME (int), AIRPORT_ID (chr), AIRPORT
##   (chr), STATE (chr), FAAREGION (chr), ENROUTE (chr), RUNWAY
##   (chr), LOCATION (chr), HEIGHT (int), SPEED (int), DISTANCE
##   (dbl), PHASE_OF_FLT (chr), DAMAGE (chr), STR_RAD (lgl),
##   DAM_RAD (lgl), STR_WINDSHLD (lgl), DAM_WINDSHLD (lgl),
##   STR_NOSE (lgl), DAM_NOSE (lgl), STR_ENG1 (lgl), DAM_ENG1
##   (lgl), STR_ENG2 (lgl), DAM_ENG2 (lgl), STR_ENG3 (lgl),
```

The verbs

- "Variable and function names should be lowercase. Use an underscore (_) to separate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful (this is not easy!)." - Hadley Wickham, `http://adv-r.had.co.nz/Style.html`
- Verbs in `dplyr`
  - `select`,
  - `arrange`,
  - `filter`,
  - `mutate`,
  - `summarize`.

select

```
# Select columns of a data.frame, tbl_df.
wls_yr <- select(wls_tbl_df, INCIDENT_YEAR, AIRPORT,
                 ENG_1_POS, ENG_2_POS, DAM_ENG1, DAM_ENG2,
                 HEIGHT, DISTANCE, SPEED)
print(wls_yr, n = 5)
```

```
## Source: local data frame [142,911 x 9]
##
##    INCIDENT_YEAR                     AIRPORT ENG_1_POS
## 1           1992 DALLAS/FORT WORTH INTL ARPT         5
## 2           1996             SACRAMENTO INTL         1
## 3           1996         DENVER INTL AIRPORT         1
## 4           1996             EPPLEY AIRFIELD         1
## 5           1996 WASHINGTON DULLES INTL ARPT         1
## ..           ...                         ...       ...
## Variables not shown: ENG_2_POS (int), DAM_ENG1 (lgl),
##   DAM_ENG2 (lgl), HEIGHT (int), DISTANCE (dbl), SPEED (int)
```

select

```
# relative speed
bnch <-
  benchmark(base  = wls_tbl_df[, c("INCIDENT_YEAR", "AIRPORT",
                                    "ENG_1_POS", "ENG_2_POS",
                                    "DAM_ENG1", "DAM_ENG2",
                                    "HEIGHT", "DISTANCE", "SPEED")],
            dplyr = select(wls_tbl_df,
                           INCIDENT_YEAR, AIRPORT,
                           ENG_1_POS, ENG_2_POS,
                           DAM_ENG1, DAM_ENG2,
                           HEIGHT, DISTANCE, SPEED),
            replications = 100)
select(bnch, test, replications, elapsed, relative)

##    test replications elapsed relative
## 1  base          100   0.007    1.000
## 2 dplyr          100   0.032    4.571
```

Selection of columns might be slower, but, there are some tools to help speed up the coding, and maintenance.

select

```
# num_range("x", 1:5, width = 2): selects all variables
# (numerically) from x01 to x05.
select(wls_tbl_df, num_range("DAM_ENG", 1:4))

## Source: local data frame [142,911 x 4]
##
##    DAM_ENG1 DAM_ENG2 DAM_ENG3 DAM_ENG4
## 1     FALSE    FALSE    FALSE    FALSE
## 2     FALSE    FALSE    FALSE    FALSE
## 3     FALSE    FALSE    FALSE    FALSE
## 4     FALSE    FALSE    FALSE    FALSE
## 5     FALSE    FALSE    FALSE    FALSE
## 6     FALSE    FALSE    FALSE    FALSE
## 7     FALSE    FALSE    FALSE    FALSE
## 8     FALSE    FALSE    FALSE    FALSE
## 9     FALSE    FALSE    FALSE    FALSE
## 10    FALSE    FALSE    FALSE    FALSE
## ..      ...      ...      ...      ...
```

select

```
# starts_with(x, ignore.case = FALSE): names starts with x
select(wls_tbl_df, starts_with("DAM"))

## Source: local data frame [142,911 x 15]
##
##    DAMAGE DAM_RAD DAM_WINDSHLD DAM_NOSE DAM_ENG1 DAM_ENG2
## 1       N   FALSE        FALSE    FALSE    FALSE    FALSE
## 2           FALSE        FALSE    FALSE    FALSE    FALSE
## 3           FALSE        FALSE    FALSE    FALSE    FALSE
## 4       N   FALSE        FALSE    FALSE    FALSE    FALSE
## 5       N   FALSE        FALSE    FALSE    FALSE    FALSE
## 6       M   FALSE        FALSE    FALSE    FALSE    FALSE
## 7       N   FALSE        FALSE    FALSE    FALSE    FALSE
## 8      M?   FALSE        FALSE    FALSE    FALSE    FALSE
## 9       N   FALSE        FALSE    FALSE    FALSE    FALSE
## 10          FALSE        FALSE    FALSE    FALSE    FALSE
## ..    ...     ...          ...      ...      ...      ...
## Variables not shown: DAM_ENG3 (lgl), DAM_ENG4 (lgl), DAM_PROP
##   (lgl), DAM_WING_ROT (lgl), DAM_FUSE (lgl), DAM_LG (lgl),
##   DAM_TAIL (lgl), DAM_LGHTS (lgl), DAM_OTHER (lgl)
```

select

```
# ends_with(x, ignore.case = FALSE): names ends in x
select(wls_tbl_df, ends_with("4"))

## Source: local data frame [142,911 x 2]
##
##    STR_ENG4 DAM_ENG4
## 1     FALSE    FALSE
## 2     FALSE    FALSE
## 3     FALSE    FALSE
## 4     FALSE    FALSE
## 5     FALSE    FALSE
## 6     FALSE    FALSE
## 7     FALSE    FALSE
## 8     FALSE    FALSE
## 9     FALSE    FALSE
## 10    FALSE    FALSE
## ..      ...      ...
```

select

```
# matches(x, ignore.case = FALSE): selects all variables
# whose name matches the regular expression x
select(wls_tbl_df, matches("ENG|DAM"))

## Source: local data frame [142,911 x 26]
##
##    NUM_ENGS TYPE_ENG ENG_1_POS ENG_2_POS ENG_3_POS ENG_4_POS
## 1         3        D         5         6         5        NA
## 2         2        D         1         1                  NA
## 3         2        D         1         1                  NA
## 4         2        D         1         1                  NA
## 5         2        D         1         1                  NA
## 6         2        D         1         1                  NA
## 7         3        D         5         6         5        NA
## 8         2        C         4         4                  NA
## 9         2        D         1         1                  NA
## 10        2        D         5         5                  NA
## ..      ...      ...       ...       ...       ...       ...
## Variables not shown: DAMAGE (chr), DAM_RAD (lgl),
##   DAM_WINDSHLD (lgl), DAM_NOSE (lgl), STR_ENG1 (lgl),
##   DAM_ENG1 (lgl), STR_ENG2 (lgl), DAM_ENG2 (lgl), STR_ENG3
##   (lgl), DAM_ENG3 (lgl), STR_ENG4 (lgl), DAM_ENG4 (lgl),
```

select

```
# contains(x, ignore.case = FALSE): selects all
# variables whose name  contains x
select(wls_tbl_df, contains("ENG"))

## Source: local data frame [142,911 x 14]
##
##    NUM_ENGS TYPE_ENG ENG_1_POS ENG_2_POS ENG_3_POS ENG_4_POS
## 1         3        D         5         6         5        NA
## 2         2        D         1         1                  NA
## 3         2        D         1         1                  NA
## 4         2        D         1         1                  NA
## 5         2        D         1         1                  NA
## 6         2        D         1         1                  NA
## 7         3        D         5         6         5        NA
## 8         2        C         4         4                  NA
## 9         2        D         1         1                  NA
## 10        2        D         5         5                  NA
## ..      ...      ...       ...       ...       ...       ...
## Variables not shown: STR_ENG1 (lgl), DAM_ENG1 (lgl), STR_ENG2
##   (lgl), DAM_ENG2 (lgl), STR_ENG3 (lgl), DAM_ENG3 (lgl),
##   STR_ENG4 (lgl), DAM_ENG4 (lgl)
```

select

What about dropping variables?

```
print(wls_yr, n = 2)

## Source: local data frame [142,911 x 9]
##
##    INCIDENT_YEAR                          AIRPORT ENG_1_POS
## 1           1992 DALLAS/FORT WORTH INTL ARPT         5
## 2           1996             SACRAMENTO INTL         1
## ..           ...                         ...       ...
## Variables not shown: ENG_2_POS (int), DAM_ENG1 (lgl),
##   DAM_ENG2 (lgl), HEIGHT (int), DISTANCE (dbl), SPEED (int)

print(select(wls_yr, -AIRPORT, -starts_with("ENG")), n = 3)

## Source: local data frame [142,911 x 6]
##
##    INCIDENT_YEAR DAM_ENG1 DAM_ENG2 HEIGHT DISTANCE SPEED
## 1           1992    FALSE    FALSE    300       NA   142
## 2           1996    FALSE    FALSE      0        0    NA
## 3           1996    FALSE    FALSE      0        0    NA
## ..           ...      ...      ...    ...      ...   ...
```

arrange

arrange: reorder the rows. Multiple inputs are ordered from left-to-right.

```
dat <- data.frame(var1 = c(3, 8, 2, 1),
                  var2 = c("E", "A", "A", "B"))
dat
```

```
##   var1 var2
## 1    3    E
## 2    8    A
## 3    2    A
## 4    1    B
```

```
# this would be very helpful for collecting data by a
# subject id, visit number, ...
```

arrange

```
arrange(dat, var2)

##   var1 var2
## 1    8    A
## 2    2    A
## 3    1    B
## 4    3    E

arrange(dat, var2, var1)

##   var1 var2
## 1    2    A
## 2    8    A
## 3    1    B
## 4    3    E

# this would be very helpful for collecting data by a
# subject id, visit number, ...
```

filter

filter: return only a subset of the rows. If multiple conditions are supplied they are combined with &.

```
dim(wls_yr)

## [1] 142911      9

filter(wls_yr, INCIDENT_YEAR > 2000, INCIDENT_YEAR <= 2005)

## Source: local data frame [31,947 x 9]
##
##    INCIDENT_YEAR                         AIRPORT ENG_1_POS
## 1           2001           JOHN F KENNEDY INTL
## 2           2001       SAN FRANCISCO INTL ARPT         1
## 3           2001                  ORLANDO INTL         1
## 4           2001                  MOLOKAI ARPT         4
## 5           2001          LAMBERT-ST LOUIS INTL         5
## 6           2001             KANSAS CITY INTL         1
## 7           2001                       UNKNOWN         1
## 8           2001         AKRON-CANTON REGIONAL         7
## 9           2001 DESTIN-FORT WALTON BEACH ARPT         5
## 10          2001           JOHN F KENNEDY INTL         1
##
```

filter

```
bnch <-
  benchmark(base = subset(wls_yr, INCIDENT_YEAR > 2000 & INCIDENT_YEAR
            dplyr = filter(wls_yr, INCIDENT_YEAR > 2000, INCIDENT_YEAR
            replications = 100)
select(bnch, test, replications, elapsed, relative)

##    test replications elapsed relative
## 1  base          100   8.038    5.402
## 2 dplyr          100   1.488    1.000
```

mutate

mutate: add new columns. Multiple inputs create multiple columns.

```
eng.lbls <- c("mounted below the wing", "mounted above the wing",
              "part of the wing root", "nacelle-mounted on the wing",
              "mounted on the aft fuselage")
str(mutate(wls_yr,
       SPEED_MPH = SPEED * 1.15078, # SPEED was in knots
       ENG_1_POS = factor(ENG_1_POS, 19:23, eng.lbls),
       ENG_2_POS = factor(ENG_2_POS, 19:23, eng.lbls)))
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame': 142911 obs. of  10 variabl
##  $ INCIDENT_YEAR: int  1992 1996 1996 1996 1996 1996 1991 1993 1995
##  $ AIRPORT      : chr  "DALLAS/FORT WORTH INTL ARPT" "SACRAMENTO INT
##  $ ENG_1_POS    : Factor w/ 5 levels "mounted below the wing",..: NA
##  $ ENG_2_POS    : Factor w/ 5 levels "mounted below the wing",..: NA
##  $ DAM_ENG1     : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ DAM_ENG2     : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ HEIGHT       : int  300 0 0 0 1000 5000 0 1500 0 100 ...
##  $ DISTANCE     : num  NA 0 0 0 NA NA 0 NA 0 NA ...
##  $ SPEED        : int  142 NA NA NA NA NA 100 220 NA 135 ...
##  $ SPEED_MPH    : num  163 NA NA NA NA ...
```

mutate

```
bnch <-
benchmark(base = within(wls_yr, {
                        SPEED_MPH = SPEED * 1.15078
                        ENG_1_POS = factor(ENG_1_POS, 19:23, eng.lbls)
                        ENG_2_POS = factor(ENG_2_POS, 19:23, eng.lbls)}
          dplyr = mutate(wls_yr,
                         SPEED_MPH = SPEED * 1.15078,
                         ENG_1_POS = factor(ENG_1_POS, 19:23, eng.lbls)
                         ENG_2_POS = factor(ENG_2_POS, 19:23, eng.lbls)
          replications = 100)
select(bnch, test, replications, elapsed, relative)

##    test replications elapsed relative
## 1  base          100   5.824    1.067
## 2 dplyr          100   5.458    1.000
```

summarize

summarise: reduce each group to a single row. Multiple inputs create multiple output summaries. (Two spellings: `summarize` and `summarise`.)

```
summarise(wls_yr,
          "Mean speed" = mean(SPEED, na.rm = TRUE),
          "SD speed"   = sd(SPEED, na.rm = TRUE),
          n            = sum(!is.na(SPEED)))

## Source: local data frame [1 x 3]
##
##   Mean speed SD speed     n
## 1      141.3    46.09 58938
```

group_by

```
summarise(group_by(wls_yr, ENG_1_POS),
          "Mean speed" = mean(SPEED, na.rm = TRUE),
          "SD speed"   = sd(SPEED, na.rm = TRUE),
          n            = sum(!is.na(SPEED)))

## Source: local data frame [11 x 4]
##
##    ENG_1_POS Mean speed SD speed     n
## 1                113.20    40.40  1303
## 2          1     154.81    43.14 27634
## 3          2      62.00    43.39     4
## 4          3     108.32    33.48    31
## 5          4     123.90    41.74  7953
## 6          5     143.80    42.31 17701
## 7          6      99.06    36.67   482
## 8          7      83.91    29.34  3829
## 9          A      90.00       NA     1
## 10         C        NaN       NA     0
## 11         T        NaN       NA     0
```

group_by

```
bnch <-
  benchmark(base = aggregate(SPEED ~ ENG_1_POS, wls_yr,
                             function(x) c(mean = mean(x, na.rm = TRUE),
                                           sd   = sd(x, na.rm = T),
                                           n    = sum(!is.na(x)))),
            dplyr = summarise(group_by(wls_yr, ENG_1_POS),
                              "Mean speed" = mean(SPEED, na.rm = TRUE),
                              "SD speed"   = sd(SPEED, na.rm = TRUE),
                              n            = sum(!is.na(SPEED))),
            replications = 100)
select(bnch, test, replications, elapsed, relative)

##    test replications elapsed relative
## 1  base          100  79.457    49.72
## 2 dplyr          100   1.598     1.00
```

Say we need to filter, group_by, and summarise data

```
# What is the mean distance from the airport, in kilometers,
# where the strike took place, by damage to engine, on twin
# engine aircraft, between 2002 and 2010, inclusive?
summarize(group_by(mutate(filter(wls, INCIDENT_YEAR >= 2002, INCIDENT_Y
    2010, NUM_ENGS == 2), DISTANCE_KM = DISTANCE * 1.60934), DAM_ENG1,
    DAM_ENG2), `mean distance in KM` = mean(DISTANCE_KM, na.rm = TRUE))

## Source: local data frame [4 x 3]
## Groups: DAM_ENG1
##
##   DAM_ENG1 DAM_ENG2 mean distance in KM
## 1    FALSE    FALSE              1.3715
## 2    FALSE     TRUE              1.3228
## 3     TRUE    FALSE              0.8347
## 4     TRUE     TRUE              0.6584

# Without a comment to explain, how long would it take to
# explain the above code?  You need to read from the inside
# out.  THERE IS A BETTER WAY
```

Chain together multiple operations.

```
wls %>%
filter(INCIDENT_YEAR >= 2002,
       INCIDENT_YEAR <= 2010,
       NUM_ENGS      == 2) %>%
mutate(DISTANCE_KM = DISTANCE * 1.60934) %>%
group_by(DAM_ENG1, DAM_ENG2) %>%
summarise("mean distance in KM" = mean(DISTANCE_KM, na.rm = TRUE))

## Source: local data frame [4 x 3]
## Groups: DAM_ENG1
##
##   DAM_ENG1 DAM_ENG2 mean distance in KM
## 1    FALSE    FALSE              1.3715
## 2    FALSE     TRUE              1.3228
## 3     TRUE    FALSE              0.8347
## 4     TRUE     TRUE              0.6584
```

More detailed examples of the forward-piping operator follow.

joining data sets

- dplyr version 0.2 has the following joins:
  - inner_join,
  - left_join,
  - semi_join, and
  - anti_join.
- Stated milestone for version 0.3 includes
  - outer_join,
  - right_join, and
  - cross_join.

Improved memory usage

```
this_df2 <- this_df <- data.frame(var1 = 1:5, var2 = rnorm(5))
changes(this_df, this_df2)

## <identical>

this_df$var1 <- rexp(5, rate = 2)
changes(this_df, this_df2)

## Changed variables:
##           old       new
## var1      0x157ae870 0x62dca90
##
## Changed attributes:
##           old       new
## row.names 0x1640b390 0x157bd6b0
```

Improved memory usage

```
this_df2 <- this_df <- data.frame(var1 = 1:5, var2 = rnorm(5)) %>% tbl_
changes(this_df, this_df2)

## <identical>

this_df <- this_df %>% mutate(var1 = rexp(5))
changes(this_df, this_df2)

## Changed variables:
##           old        new
## var1      0x1936de38 0xd580f48
##
## Changed attributes:
##           old        new
## names     0x87e4618  0xc1e0958
## class     0x18ed2ad0 0xd580980
## row.names 0x4958c40  0x4958fa0
```

dplyr memory usage

From the `vignette("memory", "dplyr")`

- `tbl_df()` and `group_by()` don't copy columns
- `select()` never copies columns, even when you rename them
- `mutate()` never copies columns, except when you modify an existing column
- `arrange()` must copy because you're changing the order of every column. This is an expensive operation for big data, but you can generally avoid it using the order argument to window functions
- `summarise()` creates new data, but it's usually at least an order of magnitude smaller than the original data.

Window Functions

```
# see vignette("window-functions", package = "dplyr")
```

Other Data Sources

- `dplyr` works for
  - `data.frames`,
  - `data.tables`, databases, and multidimensional arrays.
  - Same verbs used for all data sources.
  - See `vignette("databases", package = "dplyr")` for more details.

data.table vs dplyr

From the
dplyr introduction vignette:

- For multiple operations, data.table can be faster because you usually use it with multiple verbs at the same time. For example, with data table you can do a mutate and a select in a single step, and it's smart enough to know that there's no point in computing the new variable for the rows you're about to throw away.
- The advantages of using `dplyr` with data tables are:
  - For common data manipulation tasks, it insulates you from reference semantics of data.tables, and protects you from accidentally modifying your data.
  - Instead of one complex method built on the subscripting operator ([), it provides many simple methods.

magrittr: a forward-pipe operator for R

ceci n'est pas un pipe (this is not a pipe)

- dplyr functionality is made more powerful via the %>%, or equivalently, \%.%$, operator.
- Additional functionally provided by the magrittr package authored by Stefan Bache and Hadley Wickham.
- These operators are similar to
  - F#'s $| >$, or
  - Linux's $|$.
- Use of these operators will drastically change your R syntax.
- Helpful to writing complex, nested, operations.
- "Read from left to right instead of inside out."

magrittr: a foward-pipe operator for R

Examples

```
data(diamonds, package = "ggplot2")

# find the mean price of the diamonds
# Standard R syntax
mean(diamonds$price)

## [1] 3933

# with the pipe
diamonds %>%
extract("price") %>%
unlist() %>%
mean()

## [1] 3933
```

What's the point?

Reproducibility

The data, code, sides, etc. all at
github.com/dewittpe/dplyr-demo

```
print(sessionInfo(), locale = FALSE)

## R version 3.1.0 (2014-04-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## other attached packages:
## [1] rbenchmark_1.0.0 dplyr_0.2        magrittr_1.0.1
## [4] knitr_1.6        vimcom_0.9-93    setwidth_1.0-3
## [7] colorout_1.0-3
##
## loaded via a namespace (and not attached):
##  [1] assertthat_0.1  codetools_0.2-8 digest_0.6.4
##  [4] evaluate_0.5.5  formatR_0.10    highr_0.3
##  [7] parallel_3.1.0  qwraps_0.2.2    Rcpp_0.11.1
## [10] stringr_0.6.2   tcltk_3.1.0     tools_3.1.0
```

DRUG

- Future MeeetUp Topics:
  - (Possible) iPython / R speaker for later in July
  - We need others speakers!
- MeetUp locations/times