Using dplyr to group, manipulate and summarize data

Working with large and complex sets of data is a day-to-day reality in applied statistics. The package `dplyr` provides a well structured set of functions for manipulating such data collections and performing typical operations with standard syntax that makes them easier to remember. It is also very fast, even with large collections. To increase it's applicability, the functions work with connections to databases as well as data.frames. *dplyr* builds on *plyr* and incorporates features of *Data.Table*, which is known for being fast snf efficient in handling large datasets.

```
setwd("~/Documents/Computing with Data/24_dplyr/")
library(dplyr)
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

As a data source to illustrate properties with we'll use the flights data that we're already familiar with.

```
library(hflights)
head(hflights)
##      Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
## 5424 2011     1          1         6    1400    1500            AA
## 5425 2011     1          2         7    1401    1501            AA
## 5426 2011     1          3         1    1352    1502            AA
## 5427 2011     1          4         2    1403    1513            AA
## 5428 2011     1          5         3    1405    1507            AA
## 5429 2011     1          6         4    1359    1503            AA
##      FlightNum TailNum ActualElapsedTime AirTime ArrDelay DepDelay Origin
## 5424       428  N576AA               60      40      -10        0    IAH
## 5425       428  N557AA               60      45       -9        1    IAH
## 5426       428  N541AA               70      48       -8       -8    IAH
## 5427       428  N403AA               70      39        3        3    IAH
## 5428       428  N492AA               62      44       -3        5    IAH
## 5429       428  N262AA               64      45       -7       -1    IAH
##      Dest Distance TaxiIn TaxiOut Cancelled CancellationCode Diverted
## 5424  DFW      224      7      13         0                          0
## 5425  DFW      224      6       9         0                          0
## 5426  DFW      224      5      17         0                          0
## 5427  DFW      224      9      22         0                          0
## 5428  DFW      224      9       9         0                          0
## 5429  DFW      224      6      13         0                          0
str(hflights)
## 'data.frame':    227496 obs. of  21 variables:
##  $ Year             : int  2011 2011 2011 2011 2011 2011 2011 2011 2011 2011 ...
##  $ Month            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ DayofMonth       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ DayOfWeek        : int  6 7 1 2 3 4 5 6 7 1 ...
##  $ DepTime          : int  1400 1401 1352 1403 1405 1359 1359 1355 1443 1443 ...
##  $ ArrTime          : int  1500 1501 1502 1513 1507 1503 1509 1454 1554 1553 ...
##  $ UniqueCarrier    : chr  "AA" "AA" "AA" "AA" ...
##  $ FlightNum        : int  428 428 428 428 428 428 428 428 428 428 ...
##  $ TailNum          : chr  "N576AA" "N557AA" "N541AA" "N403AA" ...
##  $ ActualElapsedTime: int  60 60 70 70 62 64 70 59 71 70 ...
##  $ AirTime          : int  40 45 48 39 44 45 43 40 41 45 ...
##  $ ArrDelay         : int  -10 -9 -8 3 -3 -7 -1 -16 44 43 ...
##  $ DepDelay         : int  0 1 -8 3 5 -1 -1 -5 43 43 ...
##  $ Origin           : chr  "IAH" "IAH" "IAH" "IAH" ...
##  $ Dest             : chr  "DFW" "DFW" "DFW" "DFW" ...
##  $ Distance         : int  224 224 224 224 224 224 224 224 224 224 ...
##  $ TaxiIn           : int  7 6 5 9 9 6 12 7 8 6 ...
##  $ TaxiOut          : int  13 9 17 22 9 13 15 12 22 19 ...
##  $ Cancelled        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ CancellationCode : chr  "" "" "" "" ...
##  $ Diverted         : int  0 0 0 0 0 0 0 0 0 0 ...
```

There are over a quarter of a million records and 21 variables, which is good sized. *dplyr* can work fine with data.frames like this, but converting it to a `tbl_df` object gives a nice summary view of the data:

```
hflights_df <- tbl_df(hflights)
class(hflights_df)
## [1] "tbl df"     "tbl"        "data.frame"
hflights_df
## Source: local data frame [227,496 x 21]
##
##      Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
## 5424 2011     1          1         6    1400    1500            AA
## 5425 2011     1          2         7    1401    1501            AA
## 5426 2011     1          3         1    1352    1502            AA
## 5427 2011     1          4         2    1403    1513            AA
## 5428 2011     1          5         3    1405    1507            AA
## 5429 2011     1          6         4    1359    1503            AA
## 5430 2011     1          7         5    1359    1509            AA
## 5431 2011     1          8         6    1355    1454            AA
## 5432 2011     1          9         7    1443    1554            AA
## 5433 2011     1         10         1    1443    1553            AA
## ..    ...   ...        ...       ...     ...     ...           ...
## Variables not shown: FlightNum (int), TailNum (chr), ActualElapsedTime
##   (int), AirTime (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest
##   (chr), Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
```

It prints sample data appropriate foir the window size.

Basic manipulations of data

Much work with data involvces subsetting, defining new columns, sorting or otherwise manipulating the data. *dplyr* has five functions (verbs) for such actions, that all start with a data.frame or `tbl_df` and produce another one.

`filter`

```
f_df <- filter(hflights_df, Month == 1, UniqueCarrier == "AA")
f df
## Source: local data frame [273 x 21]
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1400    1500            AA       428
## 2  2011     1          2         7    1401    1501            AA       428
## 3  2011     1          3         1    1352    1502            AA       428
## 4  2011     1          4         2    1403    1513            AA       428
## 5  2011     1          5         3    1405    1507            AA       428
## 6  2011     1          6         4    1359    1503            AA       428
## 7  2011     1          7         5    1359    1509            AA       428
## 8  2011     1          8         6    1355    1454            AA       428
## 9  2011     1          9         7    1443    1554            AA       428
## 10 2011     1         10         1    1443    1553            AA       428
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
```

Here we got the January flights for AA. This is like `subset` but the syntax is a little different. We don't need &; it is added to comma separated conditions. For an "or" you add | explicitly.

```
filter(hflights_df, UniqueCarrier == "AA" | UniqueCarrier == "UA")
## Source: local data frame [5,316 x 21]
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1400    1500            AA       428
## 2  2011     1          2         7    1401    1501            AA       428
## 3  2011     1          3         1    1352    1502            AA       428
## 4  2011     1          4         2    1403    1513            AA       428
## 5  2011     1          5         3    1405    1507            AA       428
## 6  2011     1          6         4    1359    1503            AA       428
## 7  2011     1          7         5    1359    1509            AA       428
## 8  2011     1          8         6    1355    1454            AA       428
```

```
## 9  2011     1          9         7    1443    1554            AA       428
## 10 2011     1         10         1    1443    1553            AA       428
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
```

`arrange`

This function reorders the data based on specified columns.

```
arrange(hflights_df, Month, DayofMonth, desc(AirTime))
## Source: local data frame [227,496 x 21]
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1447    1925            CO        77
## 2  2011     1          1         6    1145    1612            CO        73
## 3  2011     1          1         6     942    1356            CO         1
## 4  2011     1          1         6    1757    2043            CO       570
## 5  2011     1          1         6    1824    2106            AS       731
## 6  2011     1          1         6    1908    2150            CO       567
## 7  2011     1          1         6    1506    1741            CO       767
## 8  2011     1          1         6    1226    1459            CO       267
## 9  2011     1          1         6    2028    2316            CO       670
## 10 2011     1          1         6     935    1214            CO       170
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
```

This could be done with `order` but the syntax is much harder.

`select`

This works like the select option to subset.

```
select(hflights_df, Year:DayOfWeek, TailNum, ActualElapsedTime)
## Source: local data frame [227,496 x 6]
##
##      Year Month DayofMonth DayOfWeek TailNum ActualElapsedTime
## 5424 2011     1          1         6  N576AA                60
## 5425 2011     1          2         7  N557AA                60
## 5426 2011     1          3         1  N541AA                70
## 5427 2011     1          4         2  N403AA                70
## 5428 2011     1          5         3  N492AA                62
## 5429 2011     1          6         4  N262AA                64
## 5430 2011     1          7         5  N493AA                70
## 5431 2011     1          8         6  N477AA                59
## 5432 2011     1          9         7  N476AA                71
## 5433 2011     1         10         1  N504AA                70
## ..    ...   ...        ...       ...     ...               ...
```

`mutate`

This adds new columns, often computed on old ones. But you can refer to new coilumns you just created.

```
mutate(hflights_df, gain = ArrDelay - DepDelay, gain_per_hour = gain/(AirTime/60))
## Source: local data frame [227,496 x 23]
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1400    1500            AA       428
## 2  2011     1          2         7    1401    1501            AA       428
## 3  2011     1          3         1    1352    1502            AA       428
## 4  2011     1          4         2    1403    1513            AA       428
## 5  2011     1          5         3    1405    1507            AA       428
## 6  2011     1          6         4    1359    1503            AA       428
## 7  2011     1          7         5    1359    1509            AA       428
## 8  2011     1          8         6    1355    1454            AA       428
## 9  2011     1          9         7    1443    1554            AA       428
## 10 2011     1         10         1    1443    1553            AA       428
```

```
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int), gain (int), gain_per_hour (dbl)
```

summarize

This produces a summary statistic, which when computed on the un-grouped data isn't very interesting.

```
summarize(hflights_df, delay = mean(ArrDelay, na.rm = T))
## Source: local data frame [1 x 1]
##
##   delay
## 1 7.094
```

Grouping

A major strength of *dplyr* is the ability to group the data by a variable or variables and then operate on the data "by group". With *plyr* you can do much the same using the `ddply` function or it's relatives, `dlply` and `daply`. However, there are advantages to having grouped data as an object in its own right.

Problem: Compute mean arrival delay by plane, along with other useful data.

```
library(plyr)
delay1 <- ddply(hflights, .(TailNum), function(df) {
    data.frame(count = nrow(df), dist = mean(df$Distance, na.rm = T), delay = mean(df$ArrDelay,
        na.rm = T))
})
library(dplyr)
```

The *dplyr* way to do this is as follows.

First create a version of the data grouped by plane.

```
planes <- group_by(hflights_df, TailNum)
planes
## Source: local data frame [227,496 x 21]
## Groups: TailNum
##
##      Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
## 5424 2011     1          1         6    1400    1500            AA
## 5425 2011     1          2         7    1401    1501            AA
## 5426 2011     1          3         1    1352    1502            AA
## 5427 2011     1          4         2    1403    1513            AA
## 5428 2011     1          5         3    1405    1507            AA
## 5429 2011     1          6         4    1359    1503            AA
## 5430 2011     1          7         5    1359    1509            AA
## 5431 2011     1          8         6    1355    1454            AA
## 5432 2011     1          9         7    1443    1554            AA
## 5433 2011     1         10         1    1443    1553            AA
## ..    ...   ...        ...       ...     ...     ...           ...
## Variables not shown: FlightNum (int), TailNum (chr), ActualElapsedTime
##   (int), AirTime (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest
##   (chr), Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
```

Shows all the data but indicates a group.

The information we want are summary statistics by plane. Just use the `summarize` function.

```
delay2 <- summarize(planes, count = n(), dist = mean(Distance, na.rm = T), delay = mean(ArrDelay,
    na.rm = T))
delay2
## Source: local data frame [3,320 x 4]
##
##    TailNum count   dist  delay
## 1           795  938.7    NaN
## 2   N0EGMQ    40 1095.2  1.919
## 3   N10156   317  801.7  8.199
## 4   N10575    94  631.5 18.149
## 5   N11106   308  775.0 10.102
## 6   N11107   345  768.1  8.053
## 7   N11109   331  772.5 10.280
## 8   N11113   282  772.8  4.057
```

```
## 9   N11119   130  790.2  7.397
## 10  N11121   333  774.8  6.741
## ..     ...   ...    ...    ...
```

Giving us nice summary statistics per plane. The syntax is easier to understand and it's faster.

aggregate functions

The function `n()` is one of several aggregate functions that are useful to employ with `summarise` on grouped data. Besides the typical ones like `mean`, `max`, etc., there are also `n_distinct`, `first`, `last`, `nth()`.

```
destinations <- group_by(hflights_df, Dest)
summarise(destinations, planes = n distinct(TailNum), flights = n())
## Source: local data frame [116 x 3]
##
##    Dest planes flights
## 1   ABQ    716    2812
## 2   AEX    215     724
## 3   AGS      1       1
## 4   AMA    158    1297
## 5   ANC     38     125
## 6   ASE     60     125
## 7   ATL    983    7886
## 8   AUS   1015    5022
## 9   AVL    142     350
## 10  BFL     70     504
## ..  ...    ...     ...
```

Grouping by multiple variables

When we do this we have the ability to easily compute summary stats by different combinations of the grouping variables.

Suppose we group the data into daily flights.

```
daily <- group by(hflights df, Year, Month, DayofMonth)
# To get the number of flights per day
per_day <- summarize(daily, number_flights = n())
per day
## Source: local data frame [365 x 4]
## Groups: Year, Month
##
##    Year Month DayofMonth number flights
## 1  2011     1          1            552
## 2  2011     1          2            678
## 3  2011     1          3            702
## 4  2011     1          4            583
## 5  2011     1          5            590
## 6  2011     1          6            660
## 7  2011     1          7            661
## 8  2011     1          8            500
## 9  2011     1          9            602
## 10 2011     1         10            659
## ..  ...   ...        ...            ...
```

We have access to each of the grouping variables. Notice that in the summary data.frame, we have Year and Month as grouping variables. We can get the number of flights per month by summarizing as follows.

```
per_month <- summarize(per_day, number_flights = sum(number_flights))
per_month
## Source: local data frame [12 x 3]
## Groups: Year
##
##    Year Month number flights
## 1  2011     1          18910
## 2  2011     2          17128
## 3  2011     3          19470
## 4  2011     4          18593
## 5  2011     5          19172
## 6  2011     6          19600
## 7  2011     7          20548
## 8  2011     8          20176
## 9  2011     9          18065
## 10 2011    10          18696
```

```
## 11 2011    11          18021
## 12 2011    12          19117
```

Now the only grouping variable is year. We backed out of the grouping variables by granularity. This is OK for counts and sums but for variances, e.g., this wouldn't work. You need to compute on the raw variables.

Chaining

There is a nice way to pass the result of one function to another. This is possible because so many `dplyr` functions take a data table as input and output another data table.
For example:

```
a1 <- group_by(hflights, Year, Month, DayofMonth)
a2 <- select(a1, Year:DayofMonth, ArrDelay, DepDelay)
a3 <- summarise(a2, arr = mean(ArrDelay, na.rm = TRUE), dep = mean(DepDelay,
    na.rm = TRUE))
a4 <- filter(a3, arr > 30 | dep > 30)
hflights %.% group_by(Year, Month, DayofMonth) %.% select(Year:DayofMonth, ArrDelay,
    DepDelay) %.% summarise(arr = mean(ArrDelay, na.rm = TRUE), dep = mean(DepDelay,
    na.rm = TRUE)) %.% filter(arr > 30 | dep > 30)
## Source: local data frame [14 x 5]
## Groups: Year, Month
##
##    Year Month DayofMonth   arr   dep
## 1  2011     2          4 44.08 47.17
## 2  2011     3          3 35.13 38.20
## 3  2011     3         14 46.64 36.14
## 4  2011     4          4 38.72 27.95
## 5  2011     4         25 37.80 22.26
## 6  2011     5         12 69.52 64.52
## 7  2011     5         20 37.03 26.55
## 8  2011     6         22 65.52 62.31
## 9  2011     7         29 29.56 31.87
## 10 2011     9         29 39.20 32.50
## 11 2011    10          9 61.90 59.53
## 12 2011    11         15 43.68 39.23
## 13 2011    12         29 26.30 30.79
## 14 2011    12         31 46.48 54.17
```

Working with databases

`dplyr` has been written to work with data.frames and connections to remote databases in a variety of formats. This permits handling very large amounts of data with a standard syntax.
Here we'll do an example of working with an SQLite database. `dplyr` contains all we need to set up a sample database on disk and connect to it.

```
my_db <- src_sqlite("my_db.sqlite3", create = T)
## Loading required package: RSQLite
## Loading required package: DBI
## Loading required package: RSQLite.extfuns
```

This is a database connection, although there is nothing in it yet. Now we'll copy a bunch of flight data into it.

```
hflights_sqlite <- copy_to(my_db, hflights, temporary = FALSE, indexes = list(c("Year",
    "Month", "DayofMonth"), "UniqueCarrier", "TailNum"))
class(hflights_sqlite)
## [1] "tbl_sqlite" "tbl_sql"    "tbl"
hflights sqlite
## Source: sqlite 3.7.17 [my_db.sqlite3]
## From: hflights [227,496 x 21]
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1400    1500            AA       428
## 2  2011     1          2         7    1401    1501            AA       428
## 3  2011     1          3         1    1352    1502            AA       428
## 4  2011     1          4         2    1403    1513            AA       428
## 5  2011     1          5         3    1405    1507            AA       428
## 6  2011     1          6         4    1359    1503            AA       428
## 7  2011     1          7         5    1359    1509            AA       428
## 8  2011     1          8         6    1355    1454            AA       428
## 9  2011     1          9         7    1443    1554            AA       428
```

```
## 10 2011     1         10         1    1443    1553            AA       428
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
```

This copies the hflights df and creates indices on the day, carrier and tailnumber to aid searching on these variables. hflights_sqlite is a table object that behaves like a data.frame table but is connected to the SQLite database created on the disk.

The basic verbs for manipulating and transforming data tables operate the same way.

Examples:

```
filter(hflights_sqlite, depDelay > 240)
## Source: sqlite 3.7.17 [my_db.sqlite3]
## From: hflights [389 x 21]
## Filter: depDelay > 240
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1         28         5    1516    1916            CO         1
## 2  2011     1         27         4    2137    2254            CO       150
## 3  2011     1         20         4     635     807            CO        59
## 4  2011     1         17         1    1838    2109            CO       746
## 5  2011     1         11         2    1442    1727            CO      1646
## 6  2011     1         15         6    1737    2035            DL      1590
## 7  2011     1          5         3    2025    2304            EV      5003
## 8  2011     1         24         1    1742    2050            EV      5003
## 9  2011     1         24         1    1930    2316            EV      5214
## 10 2011     1         27         4    2128      18            XE      2008
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
arrange(hflights_sqlite, Year, Month, DayofMonth)
## Source: sqlite 3.7.17 [my_db.sqlite3]
## From: hflights [227,496 x 21]
## Arrange: Year, Month, DayofMonth
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1400    1500            AA       428
## 2  2011     1          1         6     728     840            AA       460
## 3  2011     1          1         6    1631    1736            AA      1121
## 4  2011     1          1         6    1756    2112            AA      1294
## 5  2011     1          1         6    1012    1347            AA      1700
## 6  2011     1          1         6    1211    1325            AA      1820
## 7  2011     1          1         6     557     906            AA      1994
## 8  2011     1          1         6    1824    2106            AS       731
## 9  2011     1          1         6     654    1124            B6       620
## 10 2011     1          1         6    1639    2110            B6       622
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int)
mutate(hflights_sqlite, speed = AirTime/Distance)
## Source: sqlite 3.7.17 [my_db.sqlite3]
## From: hflights [227,496 x 22]
##
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1  2011     1          1         6    1400    1500            AA       428
## 2  2011     1          2         7    1401    1501            AA       428
## 3  2011     1          3         1    1352    1502            AA       428
## 4  2011     1          4         2    1403    1513            AA       428
## 5  2011     1          5         3    1405    1507            AA       428
## 6  2011     1          6         4    1359    1503            AA       428
## 7  2011     1          7         5    1359    1509            AA       428
## 8  2011     1          8         6    1355    1454            AA       428
```

```
## 9  2011     1          9         7    1443    1554            AA       428
## 10 2011     1         10         1    1443    1553            AA       428
## ..  ...   ...        ...       ...     ...     ...           ...       ...
## Variables not shown: TailNum (chr), ActualElapsedTime (int), AirTime
##   (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest (chr),
##   Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
##   CancellationCode (chr), Diverted (int), speed (int)
```

R only reaches into the database when absolutely necessary.

- It never pulls data back to R unless you explicitly ask for it.
- It delays doing any work until the last possible minute, collecting together everything you want to do then sending that to the database in one step.

```
c1 <- filter(hflights_sqlite, DepDelay > 0)
c2 <- select(c1, Year, Month, DayofMonth, UniqueCarrier, DepDelay, AirTime,
    Distance)
c3 <- mutate(c2, Speed = Distance/AirTime * 60)
c4 <- arrange(c3, Year, Month, DayofMonth, UniqueCarrier)
```

All of this is happening in **R** to tables inside the **R** session but no calls have been made to the SQLlite database until we require c4 to be printed.

```
c4
## Source: sqlite 3.7.17 [my db.sqlite3]
## From: hflights [109,996 x 8]
## Filter: DepDelay > 0
## Arrange: Year, Month, DayofMonth, UniqueCarrier
##
##    Year Month DayofMonth UniqueCarrier DepDelay AirTime Distance Speed
## 1  2011     1          1            AA        8      41      224   300
## 2  2011     1          1            AA        1      37      224   360
## 3  2011     1          1            AA        1     113      964   480
## 4  2011     1          1            AA        6      39      224   300
## 5  2011     1          1            B6       54     188     1428   420
## 6  2011     1          1            CO       17     466     3904   480
## 7  2011     1          1            CO       15      43      305   420
## 8  2011     1          1            CO        8      36      191   300
## 9  2011     1          1            CO       18      41      305   420
## 10 2011     1          1            CO       16      30      140   240
## ..  ...   ...        ...           ...      ...     ...      ...   ...
```

This only pulled out 10 rows. Notice that it retains reference to the chain of operations that created it; it looks like more than a table.

```
class(c4)
## [1] "tbl_sqlite" "tbl_sql"    "tbl"
names(c4)
## [1] "src"       "from"      "select"    "summarise" "mutate"    "where"
## [7] "group_by"  "order_by"  "query"
c4$src
## src:  sqlite 3.7.17 [my_db.sqlite3]
## tbls: hflights, sqlite_stat1
```

The component that may be most informative is query.

```
c4$query
## <Query> SELECT "Year", "Month", "DayofMonth", "UniqueCarrier", "DepDelay", "AirTime", "Distance", "Distance" /
"AirTime" * 60.0 AS "Speed"
## FROM "hflights"
## WHERE "DepDelay" > 0.0
## ORDER BY "Year", "Month", "DayofMonth", "UniqueCarrier"
## <SQLiteConnection: DBI CON (52673, 0)>
```

This is the SQL code that is actually executed on the database.

To tell **R** to complete this call to the database and download all rows we use the command `collect`.

```
subtbl <- collect(c4)
class(subtbl)
## [1] "tbl_df"     "tbl"        "data.frame"
subtbl
## Source: local data frame [109,996 x 8]
##
##    Year Month DayofMonth UniqueCarrier DepDelay AirTime Distance Speed
## 1  2011     1          1            AA        8      41      224   300
## 2  2011     1          1            AA        1      37      224   360
## 3  2011     1          1            AA        1     113      964   480
## 4  2011     1          1            AA        6      39      224   300
```

```
## 5  2011     1          1           B6       54     188     1428   420
## 6  2011     1          1           CO       17     466     3904   480
## 7  2011     1          1           CO       15      43      305   420
## 8  2011     1          1           CO        8      36      191   300
## 9  2011     1          1           CO       18      41      305   420
## 10 2011     1          1           CO       16      30      140   240
## ..  ...   ...        ...          ...      ...     ...      ...   ...
```

This has lost the SQLite feature; it is just a data.frame table.