Introduction to dplyr for Faster Data Manipulation in R

Note: There is a 40-minute video tutorial on YouTube that walks through this document in detail.

Why do I use dplyr?

- Great for data exploration and transformation
- Intuitive to write and easy to read, especially when using the "chaining" syntax (covered below)
- Fast on data frames

dplyr functionality

- Five basic verbs: `filter`, `select`, `arrange`, `mutate`, `summarise` (plus `group_by`)
- Can work with data stored in databases and data tables
- Joins: inner join, left join, semi-join, anti-join (not covered below)
- Window functions for calculating ranking, offsets, and more
- Better than plyr if you're only working with data frames (though it doesn't yet duplicate all of the plyr functionality)
- Examples below are based upon the latest release, version 0.2 (released May 2014)

Loading dplyr and an example dataset

- dplyr will mask a few base functions
- If you also use plyr, load plyr first
- hflights is flights departing from two Houston airports in 2011

```
# load packages
suppressMessages(library(dplyr))
library(hflights)

# explore data
data(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
```

- `tbl_df` creates a "local data frame"
- Local data frame is simply a wrapper for a data frame that prints nicely

```
# convert to local data frame
flights <- tbl_df(hflights)

# printing only shows 10 rows and as many columns as can fit on your screen
flights
```

```
## # A tibble: 227,496 × 21
##     Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
## *  <int> <int>      <int>     <int>   <int>   <int>         <chr>
## 1   2011     1          1         6    1400    1500            AA
## 2   2011     1          2         7    1401    1501            AA
## 3   2011     1          3         1    1352    1502            AA
## 4   2011     1          4         2    1403    1513            AA
## 5   2011     1          5         3    1405    1507            AA
## 6   2011     1          6         4    1359    1503            AA
## 7   2011     1          7         5    1359    1509            AA
## 8   2011     1          8         6    1355    1454            AA
## 9   2011     1          9         7    1443    1554            AA
## 10  2011     1         10         1    1443    1553            AA
## # ... with 227,486 more rows, and 14 more variables: 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>
```

```
# you can specify that you want to see more rows
print(flights, n=20)

# convert to a normal data frame to see all of the columns
data.frame(head(flights))
```

filter: Keep rows matching criteria

- Base R approach to filtering forces you to repeat the data frame's name
- dplyr approach is simpler to write and read
- Command structure (for all dplyr verbs):
  - first argument is a data frame
  - return value is a data frame
  - nothing is modified in place
- Note: dplyr generally does not preserve row names

```
# base R approach to view all flights on January 1
flights[flights$Month==1 & flights$DayofMonth==1, ]
```

```
# dplyr approach
# note: you can use comma or ampersand to represent AND condition
filter(flights, Month==1, DayofMonth==1)
```

```
## # A tibble: 552 × 21
##     Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
##    <int> <int>      <int>     <int>   <int>   <int>         <chr>
## 1   2011     1          1         6    1400    1500            AA
## 2   2011     1          1         6     728     840            AA
## 3   2011     1          1         6    1631    1736            AA
## 4   2011     1          1         6    1756    2112            AA
## 5   2011     1          1         6    1012    1347            AA
## 6   2011     1          1         6    1211    1325            AA
## 7   2011     1          1         6     557     906            AA
## 8   2011     1          1         6    1824    2106            AS
## 9   2011     1          1         6     654    1124            B6
## 10  2011     1          1         6    1639    2110            B6
## # ... with 542 more rows, and 14 more variables: 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>
```

```
# use pipe for OR condition
filter(flights, UniqueCarrier=="AA" | UniqueCarrier=="UA")
```

```
## # A tibble: 5,316 × 21
##     Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
##    <int> <int>      <int>     <int>   <int>   <int>         <chr>
## 1   2011     1          1         6    1400    1500            AA
## 2   2011     1          2         7    1401    1501            AA
## 3   2011     1          3         1    1352    1502            AA
## 4   2011     1          4         2    1403    1513            AA
## 5   2011     1          5         3    1405    1507            AA
## 6   2011     1          6         4    1359    1503            AA
## 7   2011     1          7         5    1359    1509            AA
## 8   2011     1          8         6    1355    1454            AA
## 9   2011     1          9         7    1443    1554            AA
## 10  2011     1         10         1    1443    1553            AA
## # ... with 5,306 more rows, and 14 more variables: 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>
```

```
# you can also use %in% operator
filter(flights, UniqueCarrier %in% c("AA", "UA"))
```

select: Pick columns by name

- Base R approach is awkward to type and to read
- dplyr approach uses similar syntax to `filter`
- Like a SELECT in SQL

```
# base R approach to select DepTime, ArrTime, and FlightNum columns
flights[, c("DepTime", "ArrTime", "FlightNum")]
```

```
# dplyr approach
select(flights, DepTime, ArrTime, FlightNum)
```

```
## # A tibble: 227,496 × 3
##    DepTime ArrTime FlightNum
## *    <int>   <int>     <int>
## 1     1400    1500       428
## 2     1401    1501       428
## 3     1352    1502       428
## 4     1403    1513       428
## 5     1405    1507       428
## 6     1359    1503       428
## 7     1359    1509       428
## 8     1355    1454       428
## 9     1443    1554       428
## 10    1443    1553       428
## # ... with 227,486 more rows
```

```
# use colon to select multiple contiguous columns, and use `contains` to match columns by name
# note: `starts_with`, `ends_with`, and `matches` (for regular expressions) can also be used to match c
select(flights, Year:DayofMonth, contains("Taxi"), contains("Delay"))
```

```
## # A tibble: 227,496 × 7
##     Year Month DayofMonth TaxiIn TaxiOut ArrDelay DepDelay
## *  <int> <int>      <int>  <int>   <int>    <int>    <int>
## 1   2011     1          1      7      13      -10        0
## 2   2011     1          2      6       9       -9        1
## 3   2011     1          3      5      17       -8       -8
## 4   2011     1          4      9      22        3        3
## 5   2011     1          5      9       9       -3        5
## 6   2011     1          6      6      13       -7       -1
## 7   2011     1          7     12      15       -1       -1
## 8   2011     1          8      7      12      -16       -5
## 9   2011     1          9      8      22       44       43
## 10  2011     1         10      6      19       43       43
## # ... with 227,486 more rows
```

"Chaining" or "Pipelining"

- Usual way to perform multiple operations in one line is by nesting
- Can write commands in a natural order by using the %>% infix operator (which can be pronounced as "then")

```
# nesting method to select UniqueCarrier and DepDelay columns and filter for delays over 60 minutes
filter(select(flights, UniqueCarrier, DepDelay), DepDelay > 60)
```

```
# chaining method
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    filter(DepDelay > 60)
```

```
## # A tibble: 10,242 × 2
##    UniqueCarrier DepDelay
##            <chr>    <int>
## 1             AA       90
```

```
## 2             AA       67
## 3             AA       74
## 4             AA      125
## 5             AA       82
## 6             AA       99
## 7             AA       70
## 8             AA       61
## 9             AA       74
## 10            AS       73
## # ... with 10,232 more rows
```

- Chaining increases readability significantly when there are many commands
- Operator is automatically imported from the magrittr package
- Can be used to replace nesting in R commands outside of dplyr

```
# create two vectors and calculate Euclidian distance between them
x1 <- 1:5; x2 <- 2:6
sqrt(sum((x1-x2)^2))
```

```
# chaining method
(x1-x2)^2 %>% sum() %>% sqrt()
```

```
## [1] 2.236068
```

arrange: Reorder rows

```
# base R approach to select UniqueCarrier and DepDelay columns and sort by DepDelay
flights[order(flights$DepDelay), c("UniqueCarrier", "DepDelay")]
```

```
# dplyr approach
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(DepDelay)
```

```
## # A tibble: 227,496 × 2
##    UniqueCarrier DepDelay
##            <chr>    <int>
## 1             OO      -33
## 2             MQ      -23
## 3             XE      -19
## 4             XE      -19
## 5             CO      -18
## 6             EV      -18
## 7             XE      -17
## 8             CO      -17
## 9             XE      -17
## 10            MQ      -17
## # ... with 227,486 more rows
```

```
# use `desc` for descending
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))
```

mutate: Add new variables

- Create new variables that are functions of existing variables

```
# base R approach to create a new variable Speed (in mph)
flights$Speed <- flights$Distance / flights$AirTime*60
flights[, c("Distance", "AirTime", "Speed")]
```

```
# dplyr approach (prints the new variable but does not store it)
flights %>%
    select(Distance, AirTime) %>%
    mutate(Speed = Distance/AirTime*60)
```

```
## # A tibble: 227,496 × 3
##    Distance AirTime    Speed
##       <int>   <int>    <dbl>
## 1       224      40 336.0000
## 2       224      45 298.6667
## 3       224      48 280.0000
## 4       224      39 344.6154
## 5       224      44 305.4545
## 6       224      45 298.6667
## 7       224      43 312.5581
## 8       224      40 336.0000
## 9       224      41 327.8049
## 10      224      45 298.6667
## # ... with 227,486 more rows
```

```
# store the new variable
flights <- flights %>% mutate(Speed = Distance/AirTime*60)
```

summarise: Reduce variables to values

- Primarily useful with data that has been grouped by one or more variables
- `group_by` creates the groups that will be operated on
- `summarise` uses the provided aggregation function to summarise each group

```
# base R approaches to calculate the average arrival delay to each destination
head(with(flights, tapply(ArrDelay, Dest, mean, na.rm=TRUE)))
head(aggregate(ArrDelay ~ Dest, flights, mean))
```

```
# dplyr approach: create a table grouped by Dest, and then summarise each group by taking the mean of A
flights %>%
    group_by(Dest) %>%
    summarise(avg_delay = mean(ArrDelay, na.rm=TRUE))
```

```
## # A tibble: 116 × 2
##     Dest  avg_delay
##    <chr>      <dbl>
## 1    ABQ   7.226259
## 2    AEX   5.839437
## 3    AGS   4.000000
## 4    AMA   6.840095
## 5    ANC  26.080645
## 6    ASE   6.794643
```

```
## 7    ATL   8.233251
## 8    AUS   7.448718
## 9    AVL   9.973988
## 10   BFL -13.198807
## # ... with 106 more rows
```

- `summarise_each` allows you to apply the same summary function to multiple columns at once
- Note: `mutate_each` is also available

```
# for each carrier, calculate the percentage of flights cancelled or diverted
flights %>%
    group_by(UniqueCarrier) %>%
    summarise_each(funs(mean), Cancelled, Diverted)
```

```
## # A tibble: 15 × 3
##    UniqueCarrier   Cancelled    Diverted
##            <chr>       <dbl>       <dbl>
## 1             AA 0.018495684 0.001849568
## 2             AS 0.000000000 0.002739726
## 3             B6 0.025899281 0.005755396
## 4             CO 0.006782614 0.002627370
## 5             DL 0.015903067 0.003029156
## 6             EV 0.034482759 0.003176044
## 7             F9 0.007159905 0.000000000
## 8             FL 0.009817672 0.003272557
## 9             MQ 0.029044750 0.001936317
## 10            OO 0.013946828 0.003486707
## 11            UA 0.016409266 0.002413127
## 12            US 0.011268986 0.001469868
## 13            WN 0.015504047 0.002293629
## 14            XE 0.015495599 0.003449550
## 15            YV 0.012658228 0.000000000
```

```
# for each carrier, calculate the minimum and maximum arrival and departure delays
flights %>%
    group_by(UniqueCarrier) %>%
    summarise_each(funs(min(., na.rm=TRUE), max(., na.rm=TRUE)), matches("Delay"))
```

```
## # A tibble: 15 × 5
##    UniqueCarrier ArrDelay_min DepDelay_min ArrDelay_max DepDelay_max
##            <chr>        <int>        <int>        <int>        <int>
## 1             AA          -39          -15          978          970
## 2             AS          -43          -15          183          172
## 3             B6          -44          -14          335          310
## 4             CO          -55          -18          957          981
## 5             DL          -32          -17          701          730
## 6             EV          -40          -18          469          479
## 7             F9          -24          -15          277          275
## 8             FL          -30          -14          500          507
## 9             MQ          -38          -23          918          931
## 10            OO          -57          -33          380          360
## 11            UA          -47          -11          861          869
## 12            US          -42          -17          433          425
## 13            WN          -44          -10          499          548
## 14            XE          -70          -19          634          628
## 15            YV          -32          -11           72           54
```

- Helper function `n()` counts the number of rows in a group
- Helper function `n_distinct(vector)` counts the number of unique items in that vector

```
# for each day of the year, count the total number of flights and sort in descending order
flights %>%
    group_by(Month, DayofMonth) %>%
    summarise(flight_count = n()) %>%
    arrange(desc(flight_count))
```

```
## Source: local data frame [365 x 3]
## Groups: Month [12]
##
##    Month DayofMonth flight_count
##    <int>      <int>        <int>
## 1      8          4          706
## 2      8         11          706
## 3      8         12          706
## 4      8          5          705
## 5      8          3          704
## 6      8         10          704
## 7      1          3          702
## 8      7          7          702
## 9      7         14          702
## 10     7         28          701
## # ... with 355 more rows
```

```
# rewrite more simply with the `tally` function
flights %>%
    group_by(Month, DayofMonth) %>%
    tally(sort = TRUE)
```

```
## Source: local data frame [365 x 3]
## Groups: Month [12]
##
##    Month DayofMonth     n
##    <int>      <int> <int>
## 1      8          4   706
## 2      8         11   706
## 3      8         12   706
## 4      8          5   705
## 5      8          3   704
## 6      8         10   704
## 7      1          3   702
## 8      7          7   702
## 9      7         14   702
## 10     7         28   701
## # ... with 355 more rows
```

```
# for each destination, count the total number of flights and the number of distinct planes that flew t
flights %>%
    group_by(Dest) %>%
    summarise(flight_count = n(), plane_count = n_distinct(TailNum))
```

```
## # A tibble: 116 × 3
##     Dest flight_count plane_count
##    <chr>        <int>       <int>
```

```
## 1    ABQ         2812         716
## 2    AEX          724         215
## 3    AGS            1           1
## 4    AMA         1297         158
## 5    ANC          125          38
## 6    ASE          125          60
## 7    ATL         7886         983
## 8    AUS         5022        1015
## 9    AVL          350         142
## 10   BFL          504          70
## # ... with 106 more rows
```

- Grouping can sometimes be useful without summarising

```
# for each destination, show the number of cancelled and not cancelled flights
flights %>%
    group_by(Dest) %>%
    select(Cancelled) %>%
    table() %>%
    head()
```

```
## Adding missing grouping variables: `Dest`
```

```
##      Cancelled
## Dest     0    1
##   ABQ 2787   25
##   AEX  712   12
##   AGS    1    0
##   AMA 1265   32
##   ANC  125    0
##   ASE  120    5
```

Window Functions

- Aggregation function (like `mean`) takes n inputs and returns 1 value
- Window function takes n inputs and returns n values
  - Includes ranking and ordering functions (like `min_rank`), offset functions (`lead` and `lag`), and cumulative aggregates (like `cummean`).

```
# for each carrier, calculate which two days of the year they had their longest departure delays
# note: smallest (not largest) value is ranked as 1, so you have to use `desc` to rank by largest value
flights %>%
    group_by(UniqueCarrier) %>%
    select(Month, DayofMonth, DepDelay) %>%
    filter(min_rank(desc(DepDelay)) <= 2) %>%
    arrange(UniqueCarrier, desc(DepDelay))
```

```
## Adding missing grouping variables: `UniqueCarrier`
```

```
# rewrite more simply with the `top_n` function
flights %>%
    group_by(UniqueCarrier) %>%
    select(Month, DayofMonth, DepDelay) %>%
    top_n(2) %>%
    arrange(UniqueCarrier, desc(DepDelay))
```

```
## Adding missing grouping variables: `UniqueCarrier`
```

```
## Selecting by DepDelay
```

```
## Source: local data frame [30 x 4]
## Groups: UniqueCarrier [15]
##
##    UniqueCarrier Month DayofMonth DepDelay
##            <chr> <int>      <int>    <int>
## 1             AA    12         12      970
## 2             AA    11         19      677
## 3             AS     2         28      172
## 4             AS     7          6      138
## 5             B6    10         29      310
## 6             B6     8         19      283
## 7             CO     8          1      981
## 8             CO     1         20      780
## 9             DL    10         25      730
## 10            DL     4          5      497
## # ... with 20 more rows
```

```
# for each month, calculate the number of flights and the change from the previous month
flights %>%
    group_by(Month) %>%
    summarise(flight_count = n()) %>%
    mutate(change = flight_count - lag(flight_count))
```

```
## # A tibble: 12 × 3
##    Month flight_count change
##    <int>        <int>  <int>
## 1      1        18910     NA
## 2      2        17128  -1782
## 3      3        19470   2342
## 4      4        18593   -877
## 5      5        19172    579
## 6      6        19600    428
## 7      7        20548    948
## 8      8        20176   -372
## 9      9        18065  -2111
## 10    10        18696    631
## 11    11        18021   -675
## 12    12        19117   1096
```

```
# rewrite more simply with the `tally` function
flights %>%
    group_by(Month) %>%
    tally() %>%
    mutate(change = n - lag(n))
```

```
## # A tibble: 12 × 3
##    Month     n change
##    <int> <int>  <int>
## 1      1 18910     NA
## 2      2 17128  -1782
## 3      3 19470   2342
## 4      4 18593   -877
## 5      5 19172    579
## 6      6 19600    428
```

```
## 7      7 20548    948
## 8      8 20176   -372
## 9      9 18065  -2111
## 10    10 18696    631
## 11    11 18021   -675
## 12    12 19117   1096
```

Other Useful Convenience Functions

```
# randomly sample a fixed number of rows, without replacement
flights %>% sample_n(5)
```

```
## # A tibble: 5 × 22
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
##   <int> <int>      <int>     <int>   <int>   <int>         <chr>     <int>
## 1  2011     4         12         2     644     918            OO      4640
## 2  2011     5         28         6    2057    2214            CO       299
## 3  2011     8         29         1    1502    1611            XE      3053
## 4  2011     3         23         3    1551    1643            XE      2936
## 5  2011     9          7         3    1556    1658            WN       950
## # ... with 14 more variables: 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 <dbl>
```

```
# randomly sample a fraction of rows, with replacement
flights %>% sample_frac(0.25, replace=TRUE)
```

```
## # A tibble: 56,874 × 22
##     Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
##    <int> <int>      <int>     <int>   <int>   <int>         <chr>
## 1   2011     9          9         5    1006    1143            XE
## 2   2011    10         22         6    1055    1400            XE
## 3   2011     3          8         2    1423    1526            WN
## 4   2011     7          8         5     623     812            WN
## 5   2011     8          2         2    1058    1147            WN
## 6   2011     3         16         3    1526    1741            MQ
## 7   2011     5         31         2    1913    2012            XE
## 8   2011    11         26         6    1659    2049            CO
## 9   2011    10         21         5    1332    1652            CO
## 10  2011     5         14         6     855     949            OO
## # ... with 56,864 more rows, and 15 more variables: 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>, Speed <dbl>
```

```
# base R approach to view the structure of an object
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    227496 obs. of  22 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 ...
##  $ Speed            : num  336 299 280 345 305 ...
```

```
# dplyr approach: better formatting, and adapts to your screen width
glimpse(flights)
```

```
## Observations: 227,496
## Variables: 22
## $ Year              <int> 2011, 2011, 2011, 2011, 2011, 2011, 2011, 20...
## $ Month             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ DayofMonth        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1...
## $ DayOfWeek         <int> 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6,...
## $ DepTime           <int> 1400, 1401, 1352, 1403, 1405, 1359, 1359, 13...
## $ ArrTime           <int> 1500, 1501, 1502, 1513, 1507, 1503, 1509, 14...
## $ UniqueCarrier     <chr> "AA", "AA", "AA", "AA", "AA", "AA", "AA", "A...
## $ FlightNum         <int> 428, 428, 428, 428, 428, 428, 428, 428, 428,...
## $ TailNum           <chr> "N576AA", "N557AA", "N541AA", "N403AA", "N49...
## $ ActualElapsedTime <int> 60, 60, 70, 70, 62, 64, 70, 59, 71, 70, 70, ...
## $ AirTime           <int> 40, 45, 48, 39, 44, 45, 43, 40, 41, 45, 42, ...
## $ ArrDelay          <int> -10, -9, -8, 3, -3, -7, -1, -16, 44, 43, 29,...
## $ DepDelay          <int> 0, 1, -8, 3, 5, -1, -1, -5, 43, 43, 29, 19, ...
## $ Origin            <chr> "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "I...
## $ Dest              <chr> "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "D...
## $ Distance          <int> 224, 224, 224, 224, 224, 224, 224, 224, 224,...
## $ TaxiIn            <int> 7, 6, 5, 9, 9, 6, 12, 7, 8, 6, 8, 4, 6, 5, 6...
## $ TaxiOut           <int> 13, 9, 17, 22, 9, 13, 15, 12, 22, 19, 20, 11...
## $ Cancelled         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ CancellationCode  <chr> "", "", "", "", "", "", "", "", "", "", "", ...
## $ Diverted          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ Speed             <dbl> 336.0000, 298.6667, 280.0000, 344.6154, 305....
```

Connecting to Databases

- dplyr can connect to a database as if the data was loaded into a data frame
- Use the same syntax for local data frames and databases
- Only generates SELECT statements
- Currently supports SQLite, PostgreSQL/Redshift, MySQL/MariaDB, BigQuery, MonetDB
- Example below is based upon an SQLite database containing the hflights data

  - Instructions for creating this database are in the databases vignette

```
# connect to an SQLite database containing the hflights data
my_db <- src_sqlite("my_db.sqlite3")

# connect to the "hflights" table in that database
flights_tbl <- tbl(my_db, "hflights")

# example query with our data frame
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))

# identical query using the database
flights_tbl %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))
```

- You can write the SQL commands yourself
- dplyr can tell you the SQL it plans to run and the query execution plan

```
# send SQL commands to the database
tbl(my_db, sql("SELECT * FROM hflights LIMIT 100"))

# ask dplyr for the SQL commands
flights_tbl %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay)) %>%
    explain()
```

Resources

- Official dplyr reference manual and vignettes on CRAN: vignettes are well-written and cover many aspects of dplyr
- July 2014 webinar about dplyr (and ggvis) by Hadley Wickham and related slides/code: mostly conceptual, with a bit of code
- dplyr tutorial by Hadley Wickham at the useR! 2014 conference: excellent, in-depth tutorial with lots of example code (Dropbox link includes slides, code files, and data files)
- dplyr GitHub repo and list of releases

< END OF DOCUMENT >