# Chapter 1: Introduction

### Convert data.frame to table

A tbl is just a special kind of data.frame. They make your data easier to look at, but also easier to work with. On top of this, a tbl is straightforwardly derived from a data.frame structure using [tbl\_df()](http://www.rdocumentation.org/packages/dplyr/functions/tbl_df).

The tbl format changes how R displays your data, but it does not change the data's underlying data structure. A tbl inherits the original class of its input, in this case, a data.frame. This means that you can still manipulate the tbl as if it were a data.frame; you can do anything with the hflights tbl that you could do with the hflights data.frame.

#### Instructions

* Convert hflights (which is a data.frame) into a tbl named hflights and display it in your console window. Notice the easy-to-read layout.
* To see how tbls behave like data.frames, save the UniqueCarrier column of hflights as an object named carriers, using standard R syntax only.

# Convert the hflights data.frame into a hflights tbl

hflights <-tbl\_df(hflights)

# Display the hflights tbl

hflights

# Create the object carriers, containing only the UniqueCarrier variable of hflights

carriers <- hflights$UniqueCarrier

### Changing labels of hflights, part 1 of 2

You can "clean" hflights the same way you would clean a data.frame. A bit of cleaning would be a good idea since the UniqueCarrier variable of hflights uses a confusing code system.

You can create a lookup table with a named vector. When you subset the lookup table with a character string (like the character strings in UniqueCarrier), R will return the values of the lookup table that correspond to the names in the character string. To see how this works, run following code in the console:

two <- c("AA", "AS")

lut <- c("AA" = "American",

"AS" = "Alaska",

"B6" = "JetBlue")

two <- lut[two]

two

#### Instructions

* Use lut to translate the UniqueCarrier column of hflights and assign its result again to the UniqueCarrier column.
* It's rather hard to assess whether your solution is actually the right one, since the UniqueCarrier variable does not appear when you display hflights. Use the [glimpse()](http://www.rdocumentation.org/packages/dplyr/functions/glimpse) function to inspect the raw values of the variables.

# Both the dplyr and hflights packages are loaded into workspace

lut <- c("AA" = "American", "AS" = "Alaska", "B6" = "JetBlue", "CO" = "Continental", "DL" = "Delta", "OO" = "SkyWest", "UA" = "United", "US" = "US\_Airways", "WN" = "Southwest", "EV" = "Atlantic\_Southeast", "F9" = "Frontier", "FL" = "AirTran", "MQ" = "American\_Eagle", "XE" = "ExpressJet", "YV" = "Mesa")

# Use lut to translate the UniqueCarrier column of hflights

hflights$UniqueCarrier<-lut[hflights$UniqueCarrier]

# Inspect the resulting raw values of your variables

glimpse(hflights)

### Changing labels of hflights, part 2 of 2

To fix the concepts covered in the previous exercise, you are challenged with a similar exercise and minimal additional help. This time, you are to change the labels in the CancellationCode column. This column lists reason why a flight was cancelled and uses a non-informative alphabetical code. To see this, execute unique(hflights$CancellationCode) in the console.

The actual meaning of the codes is the following:

* "A" stands for "carrier",
* "B" stands for "weather",
* "C" stands for "FFA" and
* "D" denotes "security".
* "E" denotes that the flight was "not cancelled".

#### Instructions

* Using the information above, create a lookup table lut that is able to convert the alphabetical codes into the more meaningful strings.
* Use lut to change the labels of the CancellationCode column of hflights.
* As before, check your results by glimpsing at them.

# The hflights tbl you built in the previous exercise is available in the workspace.

# Build the lookup table

lut <- c("A"="carrier","B"="weather","C"="FFA","D"="security","E"="not cancelled")

# Use the lookup table to create a vector of code labels. Assign the vector to the CancellationCode column of hflights

hflights$CancellationCode<-lut[hflights$CancellationCode]

# Inspect the resulting raw values of your variables

glimpse(hflights)

### The five verbs and their meaning

As Garrett pointed out, the dplyr package contains five key data manipulation functions, also called verbs:

* [select()](http://www.rdocumentation.org/packages/dplyr/functions/select), which returns a subset of the columns,
* [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter), that is able to return a subset of the rows,
* [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange), that reorders the rows according to single or multiple variables,
* [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate), used to add columns from existing data,
* [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise), which reduces each group to a single row by calculating aggregate measures.

If you want to find out more about these functions, consult the documentation by clicking on the functions above. What order of operations should we use to to find the average value of the ArrDelay (arrival delay) variable for all American Airline flights in the hflights tbl?

Feel free to play around in the console; hflights is preloaded.

#### Instructions

Answ: filter(), then summarise().

# Chapter 2: Select & Mutate

### (2.1) Choosing is not loosing! The select verb

To answer the simple question whether flight delays tend to shrink or grow during a flight, we can safely discard a lot of the variables of each flight. To select only the ones that matter, we can use [select()](http://www.rdocumentation.org/packages/dplyr/functions/select). Its syntax is plain and simple:

select(data, Var1, Var2, ...) ,

the first argument being the tbl you want to select variables from and the VarXarguments the variables you want to retain. You can also use the : and -operators inside of select, similar to indexing a data.frame with hard brackets. [select()](http://www.rdocumentation.org/packages/dplyr/functions/select) lets you apply them to names as well as integer indexes. The - operator allows you to select everything except a column or a range of columns.

#### Instructions

* Use [select()](http://www.rdocumentation.org/packages/dplyr/functions/select) to return a copy of hflights that contains only the columns ActualElapsedTime, AirTime, ArrDelay and DepDelay.
* Print hflights and assert that indeed, this dataset has not changed.
* Return a copy of hflights containing the columns Origin up to Cancelled. Do not use integer indexes.
* Find the most concise way to select: columns Year up to and including DayOfWeek, columns ArrDelay up to and including Diverted. You may want to examine the order of hflight's column names before you begin.

# hflights is pre-loaded as a tbl, together with the necessary libraries.

# Return a copy of hflights that contains the four columns related to delay.

select(hflights,ActualElapsedTime,AirTime,ArrDelay,DepDelay)

# print hflights, nothing has changed!

hflights

# Return a copy of hflights containing the columns Origin up to Cancelled

select(hflights,Origin:Cancelled)

# Answer to last question: be concise!

#select(hflights,Year:DayOfWeek,ArrDelay:Diverted)

select(hflights,-(DepTime:AirTime))

### Helper functions for variable selection

dplyr comes with a set of helper functions that can help you select variables. These functions find groups of variables to select, based on their names.

dplyr provides 6 helper functions, each of which only works when used inside [select()](http://www.rdocumentation.org/packages/dplyr/functions/select).

* starts\_with("X"): every name that starts with "X",
* ends\_with("X"): every name that ends with "X",
* contains("X"): every name that contains "X",
* matches("X"): every name that matches "X", which can be a regular expression,
* num\_range("x", 1:5): the variables named x01, x02, x03, x04 and x05,
* one\_of(x): every name that appears in x, which should be a character vector.

Watch out: Surround character strings with quotes when you pass them to a helper function, but do not surround variable names with quotes if you are not passing them to a helper function.

#### Instructions

* Use select and a helper function to return a tbl copy of hflights that contains just ArrDelay and DepDelay.
* Use a combination of helper functions and variable names to return the UniqueCarrier, FlightNum, TailNum, Cancelled, and CancellationCodecolumns of hflights.
* Find the most concise way to return the following columns with select and its helper functions: DepTime, ArrTime, ActualElapsedTime, AirTime, ArrDelay, DepDelay. Use only helper functions!

# As usual, hflights is pre-loaded as a tbl, together with the necessary libraries.

# Return a tbl containing just ArrDelay and DepDelay

select(hflights, ends\_with("Delay"))

# Return a tbl as described in the second exercise, using both helper functions and variable names

select(hflights, UniqueCarrier, ends\_with("Num"), starts\_with("Cancel"))

# Return a tbl as described in the third exercise, using only helper functions.

select(hflights, contains("Tim"), contains("Del"))

### Comparison to basic R

To see the added value of the dplyr package, it is useful to compare its syntax with basic R. Up to now, you have only considered functionality that is also available without the use of dplyr. However, the elegance and ease-of-use of dplyr should be clear from this short set of exercises. To provide continuity, you will keep on working with the hflights dataset.

#### Instructions

* For exercises 1 to 3, duplicate each of the basic R commands shown in the editor on the right by using dplyr-specific syntax. Don't use integer indexes!

# both hflights and dplyr are available

# Exercise 1

ex1r <- hflights[c("TaxiIn","TaxiOut","Distance")]

ex1d <- select(hflights,contains("Taxi"),Distance)

# Exercise 2

ex2r <- hflights[c("Year","Month","DayOfWeek","DepTime","ArrTime")]

ex2d <- select(hflights,Year:ArrTime,-DayofMonth)

# Exercise 3

ex3r <- hflights[c("TailNum","TaxiIn","TaxiOut")]

ex3d <- select(hflights,starts\_with("T"))

## (2.2) The second of five verbs: mutate

### Mutating is creating

[mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) is the second of five data manipulation functions you will get familiar with in this course. In contrast to [select()](http://www.rdocumentation.org/packages/dplyr/functions/select), which retains a subset of all variables, [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) creates new columns which are added to a copy of the dataset.

Let's briefly recap the syntax:

mutate(data, Mutant1 = expr(Var0,Var1,...))

Here, data is the tbl you want to use to create new columns. The second argument is an expression that assigns the result of any R function using already existing variables Var0, Var1, ... to a new variable Mutant1.

#### Instructions

* Add a new variable named ActualGroundTime that measures the difference between ActualElapsedTime and AirTime to a copy of hflights. Save your results as g1.
* Which variables in g1 do you think count as a plane's "ground time"? Use [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) to add these variables together and save them as GroundTime. Save your results as g2. Were you right? If you are GroundTime should equal ActualGroundTime whenever an NA is not involved. Check this by printouts in the console.
* Add a new variable to g2 named AverageSpeed that denotes the average speed that each plane flew in miles per hour. Save the resulting dataset as g3. Use Distance / AirTime \* 60.

# hflights and dplyr are loaded and ready to serve you.

# Add the new variable ActualGroundTime to a copy of hflights and save the result as g1.

g1 <- mutate(hflights,ActualGroundTime=ActualElapsedTime-AirTime)

# Add the new variable GroundTime to a copy of g1 and save the result as g2.

g2 <- mutate(g1,GroundTime=TaxiIn+TaxiOut)

# Add the new variable AverageSpeed to a copy of g2 and save the result as g1.

g3 <- mutate(g2,AverageSpeed=Distance/AirTime\*60)

### Add multiple variables using mutate

So far you've added variables to hflights one at a time, but you can also use [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) to add multiple variables at once. To create more than one variable, place a comma between each variable that you define inside [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate).

#### Instructions

* In the video, we created a new variable named loss with the first line of code on the right. Modify this code to create a second variable, loss\_percent, the percentage of the DepDelay represented by ArrDelay - DepDelay. Assign the result to m1.
* You will notice that the m1 is redundantly defined: loss\_percent uses ArrDelay - DepDelay, which is already defined as the loss variable. [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate)allows you to use a new variable while creating a second variable (or third variable, or etc.) in the same call. Rewrite the command above to use loss to define loss\_percent and assign the result to m2.
* Using [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate), create three new variables: (1) TotalTaxi, which is the sum of all taxiing times; (2) ActualGroundTime, which is the difference of ActualElapsedTime and AirTime; (3) Diff, the difference between the two newly created variables. Assign the result to m3. Observe that the Diffcolumn will be zero at all times if you solved the exercise correctly!

# hflights and dplyr are ready, are you?

# Add a second variable loss\_percent to the dataset and save the result to m1.

m1 <- mutate(hflights, loss = ArrDelay - DepDelay,loss\_percent=(ArrDelay-DepDelay)/DepDelay\*100)

# Remove the redundancy from your previous exercise and reuse loss to define the loss\_percent variable.

# Assign the result to m2

m2 <- mutate(hflights,loss=ArrDelay-DepDelay,loss\_percent=loss/DepDelay\*100)

# Add the three variables as described in the third exercise and save the result to m3

m3 <- mutate(hflights,TotalTaxi=TaxiIn+TaxiOut,ActualGroundTime=ActualElapsedTime-AirTime,Diff=TotalTaxi-ActualGroundTime)

### Recap on mutate and select

As of now, you mastered two of the five data manipulation functions that are at the core of dplyr: [select()](http://www.rdocumentation.org/packages/dplyr/functions/select) and [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate).

Which statement concerning the following four lines of code is correct?

(1) hflights <- select(hflights, -(Year:Month), -(DepTime:Diverted))

(2) select(hflights, starts\_with("D"))

(3) select(hflights, -(Year:Month), -(DepTime:Diverted))

(4) hflights <- select(hflights, starts\_with("Day"))

Feel free to experiment in the console, hflights is loaded as a tbl.

Answ: (1) and (4) lead to the same hflights variable and output; (2) and (3) do not.

# Chapter 3: Filter & Arrange

## (3.1) The third of five verbs: filter

### Logical operators

R comes with a set of logical operators that you can use to extract rows with [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter). These operators are

* x < y, TRUE if x is less than y
* x <= y, TRUE if x is less than or equal to y
* x == y, TRUE if x equals y
* x != y, TRUE if x does not equal y
* x >= y, TRUE if x is greater than or equal to y
* x > y, TRUE if x is greater than y
* x %in% c(a, b, c), TRUE if x is in the vector c(a, b, c)

#### Instructions

* Extract from hflights all flights that traveled 3000 or more miles. Save the result to f1. Check the hint if you have trouble with the syntax.
* Extract from hflights all flights flown by one of JetBlue, Southwest, or Delta airlines and assign the result to f2.
* Extract from hflights all flights where taxiing took longer than the actual flight. Try to avoid the use of [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) and do the math directly in the logical expression of [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter). Save the result to f3.

# hflights is at your disposal as a tbl, with clean carrier names

# All flights that traveled 3000 miles or more.

f1 <- filter(hflights,Distance>=3000)

# All flights flown by one of JetBlue, Southwest, or Delta airlines

f2 <- filter(hflights,UniqueCarrier%in%c("JetBlue","Southwest","Delta"))

# All flights where taxiing took longer than flying

f3 <- filter(hflights,TaxiIn+TaxiOut>AirTime)

### Combining tests using boolean operators

R also comes with a set of boolean operators that you can use to combine multiple logical tests into a single test. These include &, |, and !, respectively the and, orand not operators.

You can thus use R's & operator to combine logical tests in [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter), but that is not necessary. If you supply [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter) with multiple tests separated by commas, it will return just the rows that satisfy each test (as if the tests were joined by an &operator).

Finally, [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter) makes it very easy to screen out rows that contain NA's, R's symbol for missing information. You can identify an NA with the [is.na()](http://www.rdocumentation.org/packages/base/functions/NA) function.

#### Instructions

* Use R's logical and boolean operators to select just the rows where a flight left before 5:00 am or arrived after 10:00 pm. Save the result to f1.
* Save all of the flights that departed late but arrived ahead of schedule to f2.
* Assign all cancelled weekend flights to f3.
* Find all of the flights that were cancelled after being delayed. These are flights that were cancelled, while having a DepDelay greater than zero. Assign the result to f4.

# hflights is at your service as a tbl!

# all flights that departed before 5am or arrived after 10pm.

f1 <- filter(hflights,DepTime<500 | ArrTime>2200)

# all flights that departed late but arrived ahead of schedule

f2 <- filter(hflights,DepDelay>0 & ArrDelay<0)

# all cancelled weekend flights

f3 <- filter(hflights,Cancelled==1 & DayOfWeek%in%c(6,7))

# all flights that were cancelled after being delayed

f4 <- filter(hflights,Cancelled==1, !is.na(DepDelay))

### *Blend together what you've learned!*

So far, you have learned three data manipulation functions in the dplyr package. Time for a summarizing exercise to check your understanding. You will generate a new database from the hflights database that contains some useful information on flights that had JFK airport as their destination. You will need [select()](http://www.rdocumentation.org/packages/dplyr/functions/select), [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate), as well as [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter).

#### Instructions

* First, use [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter) to select the flights that had JFK as their destination and save this result to c1.
* Second, add a new column named Date to a copy of c1. To make Date, [paste()](http://www.rdocumentation.org/packages/base/functions/paste) together the Year, Month and DayofMonth variables, separated by a "-". Save the result to c2.
* Finally, retain only some columns to provide an overview: Date, DepTime, ArrTime and TailNum. Do not assign the resulting database to a variable; just print it to the console.

# hflights is already available in the workspace

# Select the flights that had JFK as their destination

c1 <- filter(hflights,Dest=="JFK")

# Combine the Year, Month and DayOfMonth variables to create a Date column

c2 <- mutate(c1,Date=paste(Year,Month,DayofMonth,sep="-"))

# Retain only a subset of columns to provide an overview

select(c2,Date,DepTime,ArrTime,TailNum)

### Recap on select, mutate and filter

If you mastered the first three functions, i.e. [select()](http://www.rdocumentation.org/packages/dplyr/functions/select), [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) and [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter), you can already reveal interesting information from the dataset. Through a combination of these expressions or by the use of a one-liner, try to answer the following question: How many weekend flights flew a distance of more than 1000 miles but had a total taxiing time below 15 minutes?

The hflights dataset is pre-loaded as a tbl so you can start experimenting immediately.

Answ: 155 flights

## (3.2) Almost there: the arrange verb

### Arranging your data

The syntax of [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange) is the following:

arrange(data, Var0, Var1, ... )

Here, data is again the tbl you're working with and Var0, Var1, ... are the variables according to which you arrange. When Var0 does not provide closure on the order, Var1 and possibly additional variables will serve as tie breakers to decide the arrangement.

[arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange) can be used to rearrange rows according to any type of data. If you pass [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange) a character variable, for example, R will rearrange the rows in alphabetical order according to values of the variable. If you pass a factor variable, R will rearrange the rows according to the order of the levels in your factor (running[levels()](http://www.rdocumentation.org/packages/base/functions/levels) on the variable reveals this order).

#### Instructions

* Arrange dtc, defined on the right, by departure delays so that the shortest departure delay is at the top of the data set. Assign the result to a1.
* Arrange dtc so that flights that were cancelled for the same reason appear next to each other and assign the resulting tbl to a2.
* Arrange hflights so that flights by the same carrier appear next to each other and within each carrier, flights that have smaller departure delays appear before flights that have higher departure delays. Do this in a one-liner and store the result in a3.

# dplyr and the hflights tbl are available

dtc <- filter(hflights, Cancelled == 1, !is.na(DepDelay))

# Arrange dtc by departure delays

a1 <- arrange(dtc,DepDelay)

# Arrange dtc so that cancellation reasons are grouped

a2 <- arrange(dtc,CancellationCode)

# Arrange according to carrier and departure delays

a3 <- arrange(hflights,UniqueCarrier , DepDelay)

### Reverse the order of arranging

By default, [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange) arranges the rows from smallest to largest. Rows with the smallest value of the variable will appear at the top of the data set. You can reverse this behavior with the [desc()](http://www.rdocumentation.org/packages/dplyr/functions/desc) function. [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange) will reorder the rows from largest to smallest values of a variable if you wrap the variable name in [desc()](http://www.rdocumentation.org/packages/dplyr/functions/desc) before passing it to [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange).

In addition to this small extension, you will make some trickier exercises.

#### Instructions

* Arrange hflights so that flights by the same carrier appear next to each other and within each carrier, flights that have larger departure delays appear before flights that have smaller departure delays. Save the result to a1.
* Arrange the flights inhflights by their total delay (the sum of DepDelay and ArrDelay). Can you do this without defining any new variables? Save the result to a2.
* First use filter to keep flights that have Dallas Forth Worth (DFW) as destination and departed before 8:00 am. Next, arrange the result according to AirTime such that the longest flights come first. Do this in a one-liner, without using a help variable! Store the result of this tricky exercise in a3.

# dplyr and the hflights tbl are available

# Arrange according to carrier and decreasing departure delays

a1 <- arrange(hflights,UniqueCarrier,desc(DepDelay))

# Arrange flights by total delay (normal order).

a2 <- arrange(hflights,DepDelay+ArrDelay)

# Filter out flights leaving to DFW before 8am and arrange according to decreasing AirTime

a3 <- arrange(filter(hflights,Dest=="DFW",DepTime<800),desc(AirTime))

### Recap on select, mutate, filter and arrange

Four down, one more to go! As you might have noticed, our data analysis possibilities expand with every manipulation function we learn. Can you find the appropriate strategy for the following problem?

Again using hflights (available as a tbl in the console), what steps do you take to print a list containing only TailNum of flights that departed too late, sorted by total taxiing time?

Answ: First filter(), then mutate(), arrange() and finally select().

# Chapter 4: Summarize & the pipe operator

## (4.1) Last but not least: summarise

### The syntax of summarise

[summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise), the last of the 5 verbs, follows the same syntax as [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate), but the resulting dataset consists of a single row instead of an entire new column in the case of [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate). Below, a typical [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) function is repeated to show the syntax, without going into detail on all arguments:

summarise(data, sumvar = sum(A),

avgvar = avg(B))

In contrast to the four other data manipulation functions, [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) does not return a copy of the dataset it is summarizing; instead, it builds a new dataset that contains only the summarzing statistics.

#### Instructions

* Use [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) to further explore hflights. Determine the shortest distance flown (save this as a variable named min\_dist) and the longest distance flown (save this as a variable named max\_dist). Save your solution to variable s1.
* Calculate the longest Distance for diverted flights, and save this as a variable named max\_div. You will need one of the four other verbs to do this! Do this in a one-liner and save the result to s2.

# hflights and dplyr are loaded in the workspace

# Determine the shortest and longest distance flown and save statistics to min\_dist and max\_dist resp.

s1 <- summarise(hflights, min\_dist = min(Distance), max\_dist = max(Distance))

# Determine the longest distance for diverted flights, save statistic to max\_div. Use a one-liner!

s2 <- summarise(filter(hflights, Diverted==1), max\_div = max(Distance))

### Aggregate functions

You can use any function you like in [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise), so long as the function can take a vector of data and return a single number. R contains many aggregating functions, as dplyr calls them. Here are some of the most useful:

* [min(x)](http://www.rdocumentation.org/packages/base/functions/Extremes) - minimum value of vector x.
* [max(x)](http://www.rdocumentation.org/packages/base/functions/Extremes) - maximum value of vector x.
* [mean(x)](http://www.rdocumentation.org/packages/base/functions/mean) - mean value of vector x.
* [median(x)](http://www.rdocumentation.org/packages/stats/functions/median) - median value of vector x.
* [quantile(x, p)](http://www.rdocumentation.org/packages/stats/functions/quantile) - pth quantile of vector x.
* [sd(x)](http://www.rdocumentation.org/packages/stats/functions/sd) - standard deviation of vector x.
* [var(x)](http://www.rdocumentation.org/packages/stats/functions/cor) - variance of vector x.
* [IQR(x)](http://www.rdocumentation.org/packages/stats/functions/IQR) - Inter Quartile Range (IQR) of vector x.
* diff(range(x)) - total range of vector x.

#### Instructions

* Remove rows that have NA's in the arrival delay column and save the resulting dataset to temp1. Then create a table with the following variables (and variable names): the minimum arrival delay (earliest), the average arrival delay (average), the longest arrival delay (latest), and the standard deviation for arrival delays (sd). Save this data.frame in the variable s1.
* Filter hflights such that only rows that have no NA TaxiIn and no NA TaxiOut are kept; save this temporary result to temp2. Then create a data.frame using summarise(), with one variable, max\_taxi\_diff that shows the biggest absolute difference in time between TaxiIn and TaxiOut for a single flight.

# hflights is available

# Calculate summarizing statistics for flights that have an ArrDelay that is not NA

temp1 <- filter(hflights, !is.na(ArrDelay))

s1 <- summarise(temp1, earliest = min(ArrDelay),

average = mean(ArrDelay),

latest = max(ArrDelay),

sd = sd(ArrDelay))

# Calculate the maximum taxiing difference for flights that have taxi data available

temp2 <- filter(hflights, !is.na(TaxiIn), !is.na(TaxiOut))

s2 <- summarise(temp2, max\_taxi\_diff = max(abs(TaxiIn – TaxiOut)))

### dplyr aggregate functions

dplyr provides several helpful aggregate functions of its own, in addition to the ones that are already defined in R. These include:

* [first(x)](http://www.rdocumentation.org/packages/dplyr/functions/nth) - The first element of vector x.
* [last(x)](http://www.rdocumentation.org/packages/dplyr/functions/nth) - The last element of vector x.
* [nth(x, n)](http://www.rdocumentation.org/packages/dplyr/functions/nth) - The nth element of vector x.
* [n()](http://www.rdocumentation.org/packages/dplyr/functions/n) - The number of rows in the data.frame or group of observations that [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) describes.
* [n\_distinct(x)](http://www.rdocumentation.org/packages/dplyr/functions/n_distinct) - The number of unique values in vector x.

Next to these dplyr-specific functions, you can also turn a logical test into an aggregating function with [sum()](http://www.rdocumentation.org/packages/base/functions/sum) or [mean()](http://www.rdocumentation.org/packages/base/functions/mean). A logical test returns a vector of TRUE's and FALSE's. When you apply [sum()](http://www.rdocumentation.org/packages/base/functions/sum) or [mean()](http://www.rdocumentation.org/packages/base/functions/mean) to such a vector, R coerces each TRUE to a 1 and each FALSE to a 0. This allows you to find the total number or proportion of observations that passed the test, respectively.

#### Instructions

* Create a table with the following variables (and variable names): the total number of observations in hflights (n\_obs), the total number of carriers that appear in hflights (n\_carrier), the total number of destinations that appear in hflights (n\_dest), and the destination of the flight that appears in the 100th row of hflights (dest100). Save the result to s1.
* Keep all of the flights in hflights flown by American Airlines ("American"), and assign the result to aa. Then calculate the total number of flights flown by American Airlines (n\_flights), the total number of cancelled flights (n\_canc), the percentage of cancelled flights (p\_canc), and the average arrival delay of the flights whose delay does not equal NA (avg\_delay). Store the final result in s2.

# hflights is available with full names for the carriers

# Calculate the summarizing statistics of hflights

s1 <- summarise(hflights, n\_obs = n(),

n\_carrier = n\_distinct(UniqueCarrier),

n\_dest = n\_distinct(Dest),

dest100 = nth(Dest, 100))

# Calculate the summarizing statistics for flights flown by American Airlines (carrier code "American")

aa <- filter(hflights, UniqueCarrier == "American")

s2 <- summarise(aa, n\_flights = n(),

n\_canc = sum(Cancelled == 1),

p\_canc = mean(Cancelled == 1) \* 100,

avg\_delay = mean(ArrDelay, na.rm = TRUE))

## (4.2) Chaining your functions: the pipe operator

### Overview of syntax

Garrett clearly explained the use and functionality of the %>%, but let's make sure you got the point. The following two statements are completely analogous:

mean(c(1, 2, 3, NA), na.rm = TRUE)

c(1, 2, 3, NA) %>% mean(na.rm = TRUE)

The %>% operator allows you to extract the first argument of a function from the arguments list and put it in front of it, thus solving the Dagwood sandwich problem.

#### Instructions

Use dplyr functions and the pipe operator to transform the following English sentences into R code:

* Take the hflights data set and then …
* Add a variable named diff that is the result of subtracting TaxiIn from TaxiOut, and then …
* Pick all of the rows whose diff value does not equal NA, and then …
* Summarise the data set with a value named avg that is the mean diff value.

Store the result in the variable p.

# hflights and dplyr are both loaded and ready to serve you

# Write the 'piped' version of the English sentences.

p <- hflights%>%mutate(diff=TaxiOut-TaxiIn)%>%filter(!is.na(diff))%>%summarise(avg=mean(diff))

### Drive or fly? Part 1 of 2

You can answer sophisticated questions by combining the verbs of dplyr. Over the next few exercises you will examine whether it sometimes makes sense to drive instead of fly. You will begin by making a data set that contains relevant variables. Then, you will find flights whose equivalent average velocity is lower than the velocity when traveling by car.

#### Instructions

* Define a data set named d that contains just the Dest, UniqueCarrier, Distance, and ActualElapsedTime columns of hflights as well as two additional variables: RealTime and mph.RealTime should equal the actual elapsed time plus 100 minutes. This will be an estimate of how much time a person spends getting from point A to point B while flying, including getting to the airport, security checks, etc. mph will be the miles per hour that a person on the flight traveled based on the RealTime of the flight.
* On many highways you can drive at 70 mph. Continue with d to calculate the following variables: n\_less, the number of flights whose mph value does not equal NA that traveled at less than 70 mph in real time; n\_dest, the number of destinations that were traveled to at less than 70 mph; min\_dist, the minimum distance of these flights; max\_dist, the maximum distance of these flights.

# hflights is pre-loaded

# Part 1, concerning the selection and creation of columns

d <- hflights %>%

select(Dest, UniqueCarrier, Distance, ActualElapsedTime) %>%

mutate(RealTime = ActualElapsedTime + 100, mph = Distance / RealTime \* 60)

# Part 2, concerning flights that had an actual average speed of less then 70 mph.

d %>%

filter(!is.na(mph), mph <= 70) %>%

summarise( n\_less = n(),

n\_dest = n\_distinct(Dest),

min\_dist = min(Distance),

max\_dist = max(Distance))

### Drive or fly? Part 2 of 2

The previous exercise suggested that some flights might be less efficient than driving in terms of speed. But is speed all that matters? Flying imposes burdens on a traveler that driving does not. For example, airplane tickets are very expensive. Air travelers also need to limit what they bring on their trip and arrange for a pick up or a drop off. Given these burdens we might demand that a flight provide a large speed advantage over driving.

#### Instructions

Let's define preferable flights as flights that are 150% faster than driving, i.e. that travel 105 mph or greater in real time. Also, assume that cancelled or diverted flights are less preferable than driving.

Write an adapted version of the solution to the previous exercise in an all-in-one fashion (i.e. in a single expression without intermediary variables, using %>%) to find:

* n\_non - the number of non-preferable flights in hflights,
* p\_non - the percentage of non-preferable flights in hflights,
* n\_dest - the number of destinations that non-preferable flights traveled to,
* min\_dist - the minimum distance that non-preferable flights traveled,
* max\_dist - the maximum distance that non-preferable flights traveled.
* To maintain readability in this advanced exercise, start your operations with a [select()](http://www.rdocumentation.org/packages/dplyr/functions/select) function to retain only the five columns that will be needed for the subsequent calculation steps.

# hflights and dplyr are loaded and ready to roll

# Solve the exercise using a combination of dplyr verbs and %>%

hflights %>%

select(Dest, Cancelled, Distance, ActualElapsedTime, Diverted) %>%

mutate(RealTime = ActualElapsedTime + 100, mph = Distance / RealTime \* 60) %>%

filter(mph <= 105 | Cancelled == 1 | Diverted == 1) %>%

summarise( n\_non = n(),

p\_non = n\_non / nrow(hflights) \* 100,

n\_dest = n\_distinct(Dest),

min\_dist = min (Distance),

max\_dist = max(Distance))

### Advanced piping exercises

Because piping is a very powerful and commonly used feature, you are challenged with one more exercise. Become a piping master!

#### Instructions

How many flights were overnight flights (flights whose arrival time is earlier than their departed time). Do not count NA values! Use summarise() with the n() function. The resulting data frame should contain a column named n.

# hflights and dplyr are loaded

# Count the number of overnight flights

hflights %>%

filter(!is.na(DepTime), !is.na(ArrTime), DepTime > ArrTime) %>%

summarise(n = n())

# Chapter 5: Group by & working with Databases

## (5.1) Get group-wise insights: group\_by

### Unite and conquer using group\_by

As Garrett explained, [group\_by()](http://www.rdocumentation.org/packages/dplyr/functions/group_by) lets you define groups within your data set. Its influence becomes clear when calling [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) on a grouped dataset: summarizing statistics are calculated for the different groups separately.

The syntax for this function is again straightforward:

group\_by(data, Var0, Var1, ...)

Here, data is the tbl dataset you work with, and Var0, Var1, ... are the variables you want to group by. If you pass on several variables as arguments, the number of separate sets of grouped observations will increase, but their size will decrease.

#### Instructions

* Use [group\_by()](http://www.rdocumentation.org/packages/dplyr/functions/group_by) and [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) to compare the individual carriers. For each carrier, count the total number of flights flown by the carrier (n\_flights), the total number of cancelled flights (n\_canc), the percentage of cancelled flights (p\_canc), and the average arrival delay of the flights whose delay does not equal NA (avg\_delay). Once you've calculated these results, [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange) the carriers from low to high by their average arrival delay. Use percentage of flights cancelled to break any ties. Which airline scores best based on these statistics?
* Come up with a way to answer this question: At which day of the week is the average total taxiing time highest? Use [group\_by()](http://www.rdocumentation.org/packages/dplyr/functions/group_by), [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) and [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange); you should avoid the use of [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate). Define the grouped average total taxiing time to be avg\_taxi.

**# hflights is in the workspace as a tbl, with translated carrier names**

**# Make the calculations to end up with ordered statistics per carrier**

**hflights %>% group\_by(UniqueCarrier) %>%**

**summarise(n\_flights=n(), n\_canc=sum(Cancelled), p\_canc=mean(Cancelled)\*100, avg\_delay=mean(ArrDelay,na.rm=TRUE)) %>%**

**arrange(avg\_delay, p\_canc)**

**# Answer the question: At which day of the week is total taxiing time highest?**

**hflights %>% group\_by(DayOfWeek) %>%**

**summarise(avg\_taxi=mean(TaxiIn+TaxiOut,na.rm=TRUE)) %>%**

**arrange(desc(avg\_taxi))**

### Combine group\_by with mutate

You can also combine [group\_by()](http://www.rdocumentation.org/packages/dplyr/functions/group_by) with [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate). When you mutate grouped data, [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) will calculate the new variables independently for each group. This is particularly useful when [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate) uses the [rank()](http://www.rdocumentation.org/packages/base/functions/rank) function, which calculates within group rankings. [rank()](http://www.rdocumentation.org/packages/base/functions/rank) takes a group of values and calculates the rank of each value within the group, e.g.

rank(c(21, 22, 24, 23))

has output

[1] 1 2 4 3

As with [arrange()](http://www.rdocumentation.org/packages/dplyr/functions/arrange), [rank()](http://www.rdocumentation.org/packages/base/functions/rank) ranks values from the largest to the smallest and this behavior can be reversed with the [desc()](http://www.rdocumentation.org/packages/dplyr/functions/desc) function.

#### Instructions

* Discard flights whose arrival delay equals NA, then rank the carriers by the proportion, i.e. the fraction, of their flights that arrived delayed (call this variable p\_delay) and arrange the results based on the ranking. Create the rank variable explicitly using [mutate()](http://www.rdocumentation.org/packages/dplyr/functions/mutate).
* In a similar fashion, rank the carriers by the average delay of flights that are delayed (ArrDelay > 0). Call this summarizing variable avg. Then arrange the data based on the results. Again, build a rank variable explicitly.

# dplyr is loaded, hflights is loaded with fancy carrier names

# Solution to first instruction

hflights %>%

group\_by(UniqueCarrier) %>%

filter(!is.na(ArrDelay)) %>%

summarise(p\_delay = mean(ArrDelay > 0)) %>%

mutate(rank = rank(p\_delay)) %>%

arrange(rank)

# Solution to second instruction

hflights %>%

group\_by(UniqueCarrier) %>%

filter(!is.na(ArrDelay), ArrDelay > 0) %>%

summarise(avg = mean(ArrDelay)) %>%

mutate(rank = rank(avg)) %>%

arrange(rank)

### Advanced group\_by exercises

By now you've learned the fundamentals of dplyr: the five data manipulation verbs and the additional [group\_by()](http://www.rdocumentation.org/packages/dplyr/functions/group_by) function to discover interesting groupwise statistics. The next challenges are an all-encompassing review of the concepts you have learned about. Answer the following questions as precisely as possible, i.e. the resulting dataset should only contain the information that is needed to answer the question.

Remember that the *hflights* database contains only flights that flew from Houston; there is thus no need to filter on the *Origin* column.

#### Instructions

* Which plane (by tail number) flew out of Houston the most times? How many times? Name the column with this frequency n. Assign the result to adv1. To answer this question precisely, you will have to [filter()](http://www.rdocumentation.org/packages/dplyr/functions/filter) as a final step to end up with only a single observation in adv1.
* How many airplanes only flew to one destination from Houston? Save the resulting dataset in adv2, that contains only a single column, named nplanesand a single row.
* Find the most visited destination for each carrier and save your solution to adv3. Your solution should contain four columns:
  + UniqueCarrier and Dest,
  + n, how often a carrier visited a particular destination,
  + rank, how each destination ranks per carrier. rank should be 1 for every row, as you want to find the most visited destination for each carrier.
* For each destination, find the carrier that travels to that destination the most. Store the result in adv4. Again, your solution should contain 4 columns: Dest, UniqueCarrier, n and rank.

# dplyr and hflights (translated carrier names) are available.

# Which plane (by tail number) flew out of Houston the most times? How many times?

adv1 <- hflights %>%

#filter(Origin=="HOU")%>%

group\_by(TailNum)%>%

summarize(n=n())%>%

select(TailNum,n)%>%

filter(n==max(n))

# How many airplanes only flew to one destination from Houston?

adv2 <- hflights %>%

#filter(Origin=="HOU")%>%

group\_by(TailNum)%>%

summarize(ndest=n\_distinct(Dest))%>%

filter(ndest==1)

# Find the most visited destination for each carrier

adv3 <- hflights %>%

group\_by(UniqueCarrier,Dest)%>%

summarise(n=n())%>%

arrange(desc(n))%>%

filter(n==max(n))

# Find the carrier that travels to each destination the most.

adv4<-hflights%>%

group\_by(Dest,UniqueCarrier)%>%

summarise(n=n())%>%

arrange(desc(n))%>%

filter(n==max(n))

## (5.2) dplyr and databases

### dplyr deals with different types

hflights2 is a copy of hflights that is saved as a data table. hflights2 was made available in the background using the following code:

library(data.table)

hflights2 <- as.data.table(hflights)

hflights2 contains all of the same information as hflights, but the information is stored in a different data structure. You can see this structure by typing hflights2at the command line.

Even though hflights2 is a different data structure, you can use the same dplyr functions to manipulate hflights2 as you used to manipulate hflights.

#### Instructions

* Use [summarise()](http://www.rdocumentation.org/packages/dplyr/functions/summarise) to calculate n\_carrier, the total number of unique carriers inhflights2. Save the result to the variable s2. Whether or not you use the pipe is up to you!

# hflights2 is pre-loaded as a data.table

# Use summarise to calculate n\_carrier

s2<-summarise(hflights2,n\_carrier=n\_distinct(UniqueCarrier))

### dplyr and mySQL databases

nycflights is a mySQL database on the DataCamp server. It contains information about flights that departed from New York City in 2013. This data is similar to the data in hflights, but it does not contain information about cancellations or diversions (you can access the same data in the nycflights13 R package).

nycflights, an R object that stores a connection to the nycflights tbl that lives outside of R on the datacamp server, will be created for you on the right. You can use such connection objects to pull data from databases into R. This lets you work with datasets that are too large to fit in R.

You can learn a connection language to make sophisticated queries from such a database, or you can simply use dplyr. When you run a dplyr command on a database connection, dplyr will convert the command to the database's native language and do the query for you. As such, just the data that you need from the database will be retrieved. This will usually be a fraction of the total data, which will fit in R withouth memory issues.

For example, we can easily retrieve a summary of how many carriers and how many flights flew in and out of New York City in 2013 with the code (note that in nycflights, the UniqueCarrier variable is named carrier):

summarise(nycflights,

n\_carriers = n\_distinct(carrier),

n\_flights = n())

#### Instructions

* Try to understand the already available code on the right. This code will create a reference to a tbl that resides in DataCamp's servers.
* Glimpse at nycflights. Although nycflights is a reference to a tbl in a remote database, there is no difference in syntax nor output!
* Group the nycflights data by carrier, then create a grouped summary of the data that shows the number of flights (n\_flights) flown by each carrier and the average arrival delay (avg\_delay) of flights flown by each carrier. Finally, arrange the carriers by average delay from low to high. Assign the result to dbsumm and use the pipe operator to maintain oversight in your calculations.

# set up a src that connects to the mysql database (src\_mysql is provided by dplyr)

my\_db <- src\_mysql(dbname = "dplyr",

host = "dplyr.csrrinzqubik.us-east-1.rds.amazonaws.com",

port = 3306, user = "dplyr", password = "dplyr")

# and reference a table within that src

nycflights <- tbl(my\_db, "dplyr")

# nycflights is now available as an R object that references to the remote nycflights table.

# glimpse at nycflights

glimpse(nycflights)

# Calculate the grouped summaries detailed in the instructions.

dbsumm <- nycflights%>%

group\_by(carrier)%>%

summarise(n\_flights=n(),avg\_delay=mean(arr\_delay))%>%

arrange(avg\_delay)