Day 2: Data Visualization in `R`

FSU Summer Methods School

Therese Anders

5/7/2019

Contents

1 Why data wrangling in a visualization workshop? 1

2 Introduction to `dplyr` 2

2.1 Using `select()` and introducing the Piping Operator `%>%` . . . 3

2.2 Introducting `filter()` . . . 3

2.3 Introducting `mutate()` . . . 5

2.4 Introducting `summarise()` and `arrange()` . . . 6

2.5 Joins . . . 12

3 Heatmaps 13

4 Alluvial diagrams 16

5 Primer on `tidyr` 22

5.1 Wide versus long data . . . 22

5.2 `gather()` . . . 23

1 Why data wrangling in a visualization workshop?

This workshop focuses on data visualization. However, in practice, data visualization is only the last part in a long stream of data gathering, cleaning, wrangling, and analysis.

`ggplot2` is the most powerful when we have "tidy" data. There are three rules for tidy data, based on Hadley Wickham's R for Data Science.

1. "Each variable must have its own column."

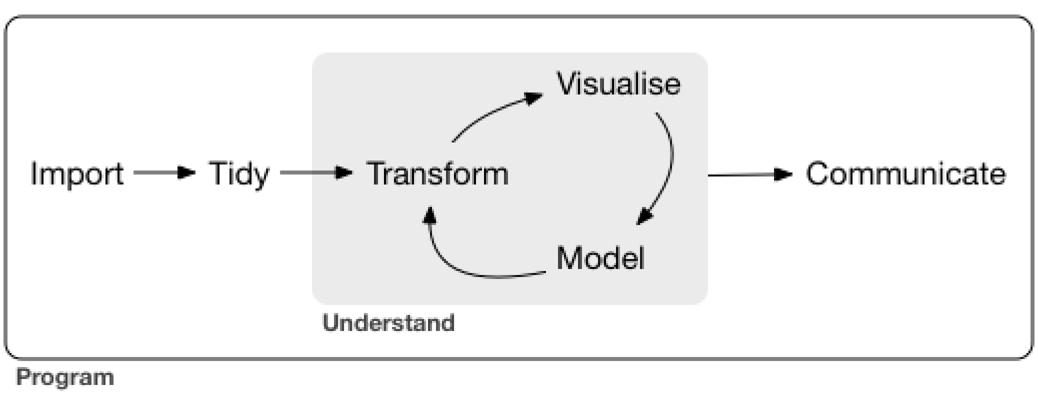


Figure 1: https://d33wubrfki0l68.cloudfront.net/571b056757d68e6df81a3e3853f54d3c76ad6efc/32d37/diagrams/data-science.png

2. “Each observation must have its own row.”
3. “Each value must have its own cell.”

If the data are in a tidy format, we can pass separate variables to separate aesthetics and create layered displays of multiple variables. Thus an important component of creating interesting data visualizations is to get the data to be in the right format. We will also learn a number of new data visualization tools as part of the data wrangling section, including

- Bar charts
- Error bars on plots

RStudio offers a great Data wrangling cheat sheet you should take a look at.

2 Introduction to `dplyr`

`dplyr` does not accept tables or vectors, just data frames (similar to `ggplot2`)! `dplyr` uses a strategy called “Split - Apply - Combine”. Some of the key functions include:

- `select()`: Subset columns.
- `filter()`: Subset rows.
- `arrange()`: Reorders rows.
- `mutate()`: Add columns to existing data.
- `summarise()`: Summarizing data set.
- joins: Combine two data frames together

First, lets dowload the package and call it using the `library()` function.

```r
# install.packages("dplyr")
library(dplyr)
```

Today, we will be working with a data set from the `hflights` package. The data set contains all flights from the Houston IAH and HOU airports in 2011. Install the package `hflights`, load it into the library, extract the data frame into a new object called `raw` and inspect the data frame.

NOTE: The `::` operator specifies that we want to use the *object* `hflights` from the *package* `hflights`. In the case below, this explicit programming is not necessary. However, it is useful when functions or objects are contained in multiple packages to avoid confusion. A classic example is the `select()` function that is contained in a number of packages besides `dplyr`.

```r
# install.packages("hflights")
library(hflights)
raw <- hflights::hflights
str(raw)
```

```
## '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 ...
```

2.1 Using `select()` and introducing the Piping Operator `%>%`

Using the so-called **piping operator** will make the `R` code faster and more legible, because we are not saving every output in a separate data frame, but passing it on to a new function. First, let's use only a subsample of variables in the data frame, specifically the year of the flight, the airline, as well as the origin airport, the destination, and the distance between the airports.

Notice a couple of things in the code below:

- We can assign the output to a new data set.
- We use the piping operator to connect commands and create a single flow of operations.
- We can use the select function to rename variables.
- Instead of typing each variable, we can select sequences of variables.
- Note that the `everything()` command inside `select()` will select all variables.

```r
data <-  raw %>%
  dplyr::select(Month,
              DayOfWeek,
              DepTime,
              ArrTime,
              ArrDelay,
              TailNum,
              Airline = UniqueCarrier, #Renaming the variable
              Time = ActualElapsedTime, #Renaming the variable
              Origin:Cancelled) #Selecting a number of columns.
names(data)
```

```
##  [1] "Month"     "DayOfWeek" "DepTime"   "ArrTime"   "ArrDelay"
##  [6] "TailNum"   "Airline"   "Time"      "Origin"    "Dest"
## [11] "Distance"  "TaxiIn"    "TaxiOut"   "Cancelled"
```

Suppose, we didn't really want to select the `Cancelled` variable. We can use `select()` to drop variables.

```r
data <- data %>%
  dplyr::select(-Cancelled)
```

2.2 Introducting `filter()`

There are a number of key operations when manipulating observations (rows).

- `x < y`
- `x <= y`
- `x == y`
- `x != y`
- `x >= y`

- `x > y`
- `x %in% c(a,b,c)` is `TRUE` if `x` is in the vector `c(a, b, c)`.

Suppose, we wanted to filter all the flights that have their destination in the greater Los Angeles area, specifically Los Angeles (LAX), Ontario (ONT), and John Wayne (SNA) airports. Note that based on the **hflights** dataset, there are no flights from the Houston area to Bob Hope (BUR) or Long Beach (LGB) airports.

```
airports <- c("LAX", "ONT", "SNA")

la_flights <- data %>%
  filter(Dest %in% airports)
```

Caution: The following command does not return the flights to LAX or ONT!

```
head(la_flights)
```

```
##   Month DayOfWeek DepTime ArrTime ArrDelay TailNum Airline Time Origin
## 1     1         1    1916    2103        2  N76522      CO  227    IAH
## 2     1         1     747     936        5  N67134      CO  229    IAH
## 3     1         1    1433    1629       14  N73283      CO  236    IAH
## 4     1         1    1750    1921        6  N34282      CO  211    IAH
## 5     1         1     917    1120       15  N76515      CO  243    IAH
## 6     1         1    1550    1736        8  N76502      CO  226    IAH
##   Dest Distance TaxiIn TaxiOut
## 1  LAX     1379      8      20
## 2  LAX     1379     11      17
## 3  LAX     1379     10      27
## 4  ONT     1334      5      17
## 5  SNA     1347      6      35
## 6  LAX     1379     13      15
```

```
la_flights_alt <- data %>%
  filter(Dest == c("LAX", "ONT"))
head(la_flights_alt)
```

```
##   Month DayOfWeek DepTime ArrTime ArrDelay TailNum Airline Time Origin
## 1     1         1    1916    2103        2  N76522      CO  227    IAH
## 2     1         1    1433    1629       14  N73283      CO  236    IAH
## 3     1         1    2107    2247        7  N73270      CO  220    IAH
## 4     1         1     920    1116        5  N77867      CO  236    IAH
## 5     1         1    1325    1538       32  N26210      CO  253    IAH
## 6     1         1    1749    1938        6  N73860      CO  229    IAH
##   Dest Distance TaxiIn TaxiOut
## 1  LAX     1379      8      20
## 2  LAX     1379     10      27
## 3  LAX     1379      7      12
## 4  LAX     1379      8      33
## 5  LAX     1379     11      30
## 6  LAX     1379     15      14
```

Why? We are basically returning all values for which the following is `TRUE` (using the correct output of the `la_flights` data frame:

`Dest[1] == LAX`

`Dest[2] == ONT`

`Dest[3] == LAX`

```
Dest[4] == ONT ...
```

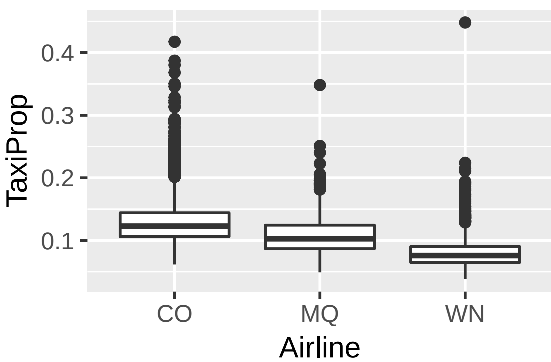
2.3 Introducting `mutate()`

Currently, we have two taxi time variables in our data set: `TaxiIn` and `TaxiOut`. I care about total taxi time, and want to add the two together. Also, people hate sitting in planes while it is not in the air. To see how much time is spent taxiing versus flying, we create a variable which measures the proportion of taxi time of total time of flight.

```
la_flights <- data %>%
  filter(Dest %in% airports) %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time)
```

We can the graph the average proportion of taxi time per airline.

```
library(ggplot2)
ggplot(la_flights,
       aes(x = Airline,
           y = TaxiProp)) +
  geom_boxplot()
```



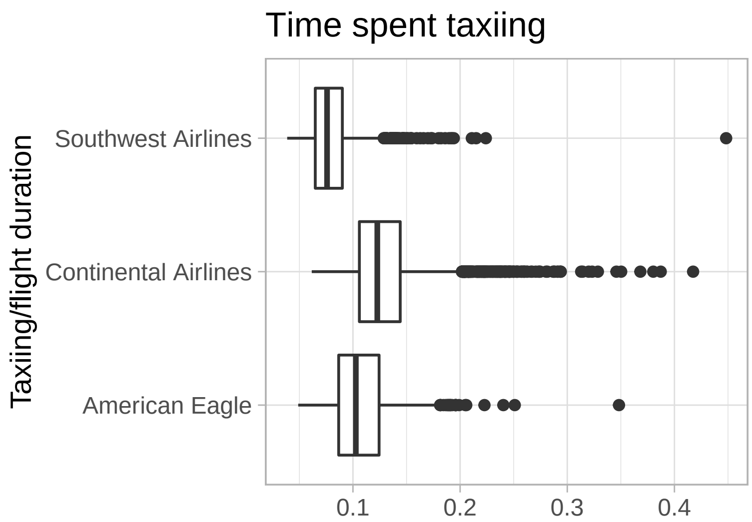
There is only three airlines flying to LA out of Houston. Lets create a new variable with the airline name using the `case_when()` function to make the graph more informative.

```
table(la_flights$Airline)
```

```
##
##   CO   MQ   WN
## 6471  810 1396
```

```
la_flights <- data %>%
  filter(Dest %in% airports) %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time,
         AirlineName = case_when(
           Airline == "CO" ~ "Continental Airlines",
           Airline == "MQ" ~ "American Eagle",
           Airline == "WN" ~ "Southwest Airlines"
         ))
ggplot(la_flights,
       aes(x = AirlineName,
           y = TaxiProp)) +
  geom_boxplot() +
```

```
  coord_flip() +
  labs(title = "Time spent taxiing",
       x = "Taxiing/flight duration",
       y = "") +
  theme_light()
```



2.4 Introducting `summarise()` and `arrange()`

One of the most powerful `dplyr` features is the `summarise()` function, especially in combination with `group_by()`.

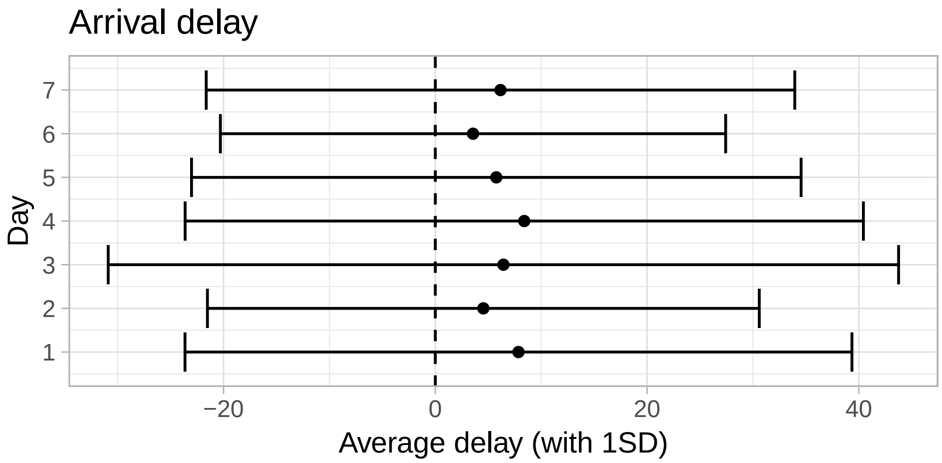
First, in a simple example, lets compute the average delay from Houston to Los Angeles by each day of the week. Note that the arrival delay variable is given in minutes. Also, I want to know what standard deviation of the delay is for each day of the weak. Note, that because there are missing values, we need to tell `R` what to do with them.

```
la_flights_delay <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(av_delay = mean(ArrDelay, na.rm = T),
            sd_delay = sd(ArrDelay, na.rm = T))
```

We can use error bars to show the standard deviation of the delay time for each day of the weak. I add a line to denote no delay using the `geom_hline()` aesthetic.

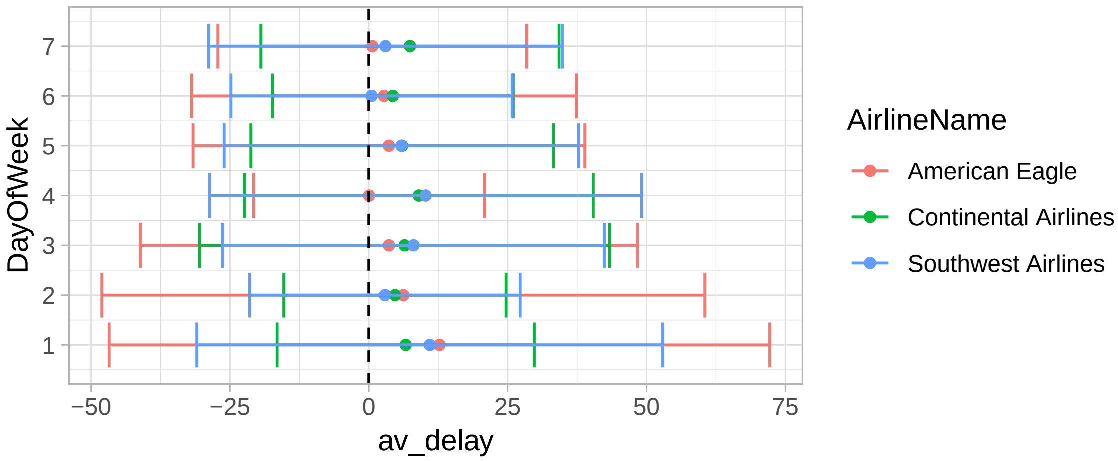
```
ggplot(la_flights_delay,
       aes(x = DayOfWeek,
           y = av_delay,
           ymin = av_delay - sd_delay,
           ymax = av_delay + sd_delay)) +
  geom_point() +
  geom_errorbar() +
  geom_hline(yintercept = 0,
             linetype = "dashed") +

  # Making the graph prettier
  scale_x_continuous(breaks = seq(1,7)) +
  theme_light() +
```

```r
  labs(y = "Average delay (with 1SD)",
       x = "Day",
       title = "Arrival delay") +
  coord_flip()
```



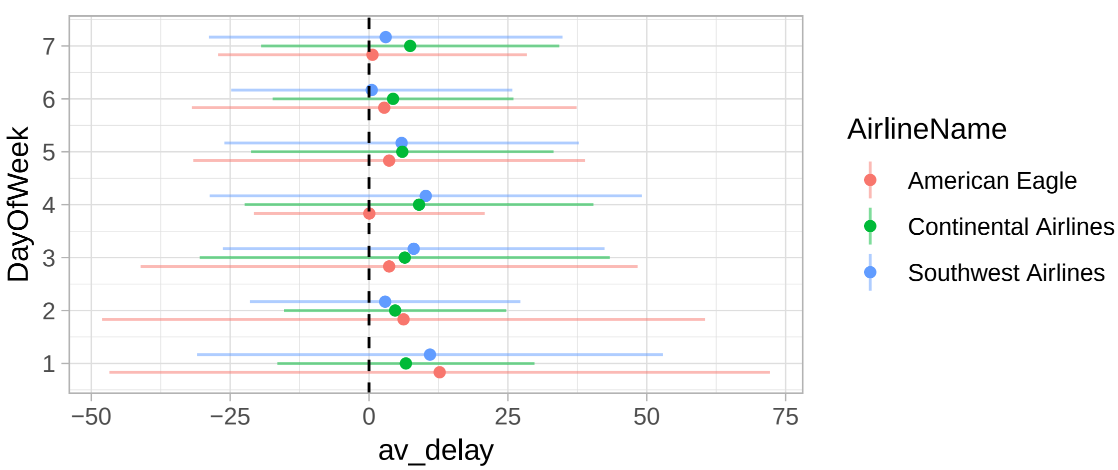
Suppose, I wanted to know whether some airlines have on average shorter arrival delays than others. We can add the airline to the `group_by()` function to compute the mean and standard deviation of arrival delay per day and airline.

```r
la_flights_delay_airline <- la_flights %>%
  group_by(DayOfWeek, AirlineName) %>%
  summarise(av_delay = mean(ArrDelay, na.rm = T),
            sd_delay = sd(ArrDelay, na.rm = T))

# Plotting it
ggplot(la_flights_delay_airline,
       aes(x = DayOfWeek,
           y = av_delay,
           ymin = av_delay - sd_delay,
           ymax = av_delay + sd_delay,
           color = AirlineName)) +
  geom_point() +
  geom_errorbar() +
  geom_hline(yintercept = 0,
             linetype = "dashed") +

  # Making graph prettier
  theme_light() +
  coord_flip() +
  scale_x_continuous(breaks = seq(1,7))
```



To de-clutter the graph, below, I use the `geom_linerange()` aesthetic rather than `geom_errorbar()`. I can use the `position = dodge` command within the `geom_point()` and `geom_linerange()` aesthetic to display the values for each airline next to each other, instead on top of each other. Note that I could have used `position = dodge` with `geom_errorbar()` as well; the functionality is essentially the same.

```
ggplot(la_flights_delay_airline,
       aes(x = DayOfWeek,
           y = av_delay,
           ymin = av_delay - sd_delay,
           ymax = av_delay + sd_delay,
           color = AirlineName)) +
  geom_point(position = position_dodge(width = 0.5)) +
  geom_linerange(position = position_dodge(width = 0.5),
                 alpha = 0.5) +
  geom_hline(yintercept = 0,
             linetype = "dashed") +

  # Making graph prettier
  theme_light() +
  coord_flip() +
  scale_x_continuous(breaks = seq(1,7))
```



2.4.1 Dataviz: Barplots

Suppose we wanted to know which airline operates the most flights out of either Houston airport. Here, we will be using the operator `n()` to tell `dplyr` to count all the observations for the groups specified in `group_by()`. After computing the result, I would like to arrange the output from highest number of flights to lowest number.

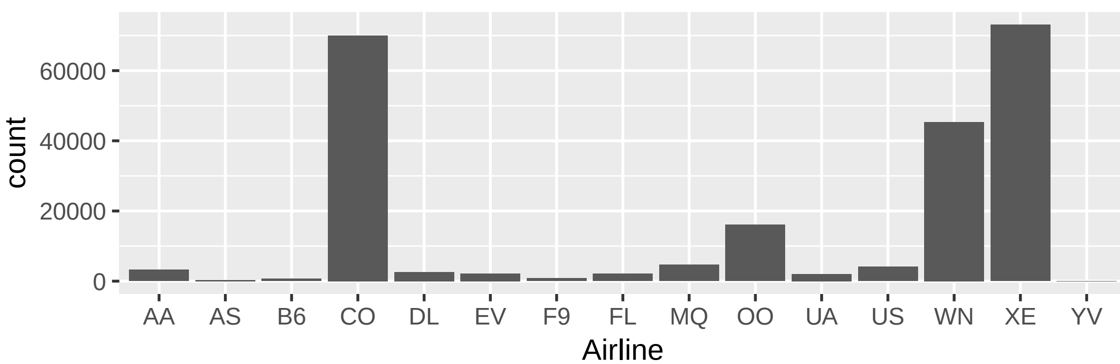
```
carriers <- data %>%
  group_by(Airline) %>%
  summarise(NoFlights = n()) %>%
  arrange(desc(NoFlights))
```

We can display the result graphically using the `geom_bar()` aesthetic. Note the following details on the usage of `geom_bar()` from the `ggplot2` package documentation below.

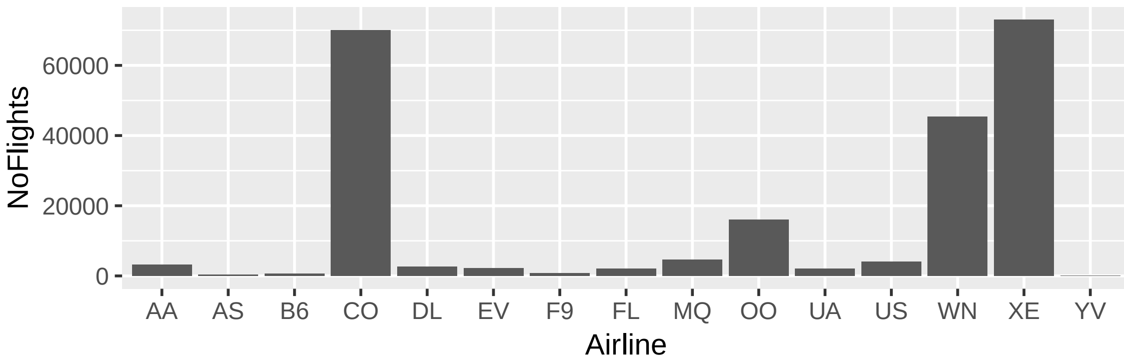
"The heights of the bars commonly represent one of two things: either a count of cases in each group, or the values in a column of the data frame. By default, geom_bar uses stat="bin". This makes the height of each bar equal to the number of cases in each group, and it is incompatible with mapping values to the y aesthetic. If you want the heights of the bars to represent values in the data, use stat="identity" and map a value to the y aesthetic." (https://www.rdocumentation.org/packages/ggplot2/versions/1.0.1/topics/geom_bar)

Thus, the creating the count variables using `group_by()` and `summarise()` is not absolutely necessary. However, for more complicated groupings of data, I highly recommend creating a separate data frame and "hard code" groupings of interest before graphing.

```
# Using default geom_bar(stat = "bin#) on the original data
ggplot(data,
       aes(x = Airline)) +
  geom_bar()
```

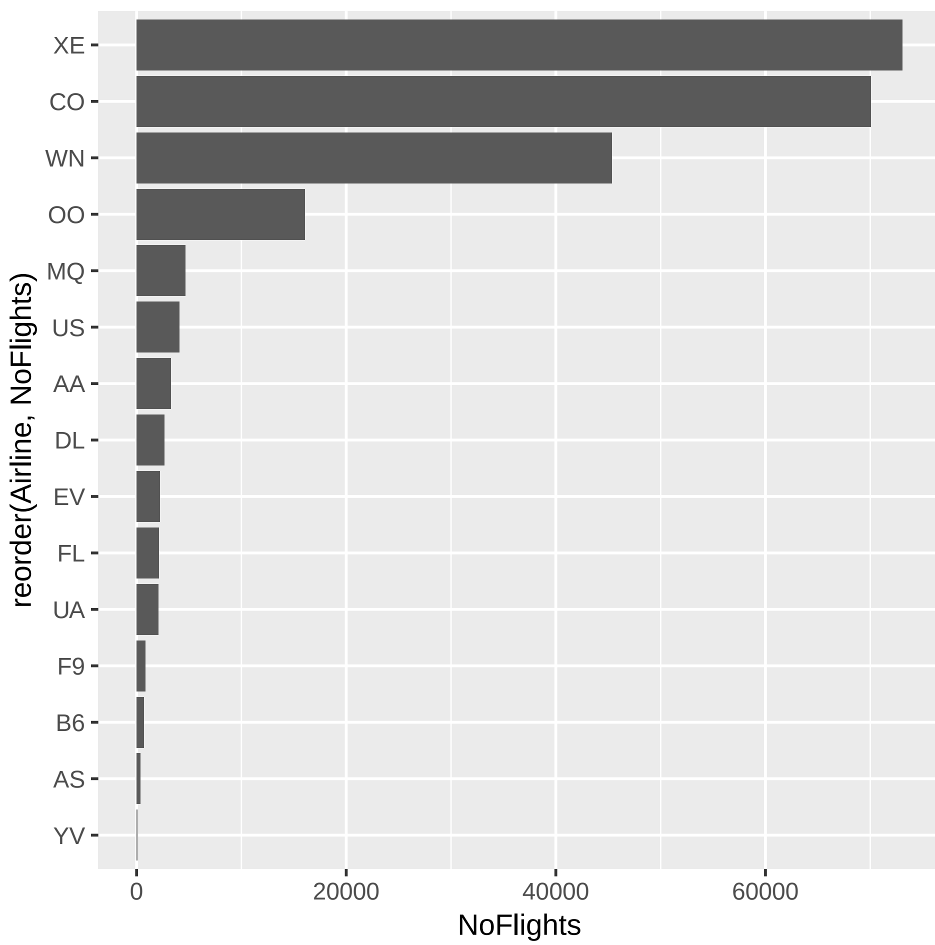



```
# Using geom_bar(stat = "identity") on grouped data
ggplot(carriers,
       aes(x = Airline,
           y = NoFlights)) +
  geom_bar(stat = "identity")
```



Lets make the flight more legible. We want the airline codes on the y-axis and the bars sorted from most to least flights.

```
# Using default geom_bar(stat = "bin#) on the original data
ggplot(carriers,
       aes(x = reorder(Airline, NoFlights),
           y = NoFlights)) +
  geom_bar(stat = "identity") +
  coord_flip()
```

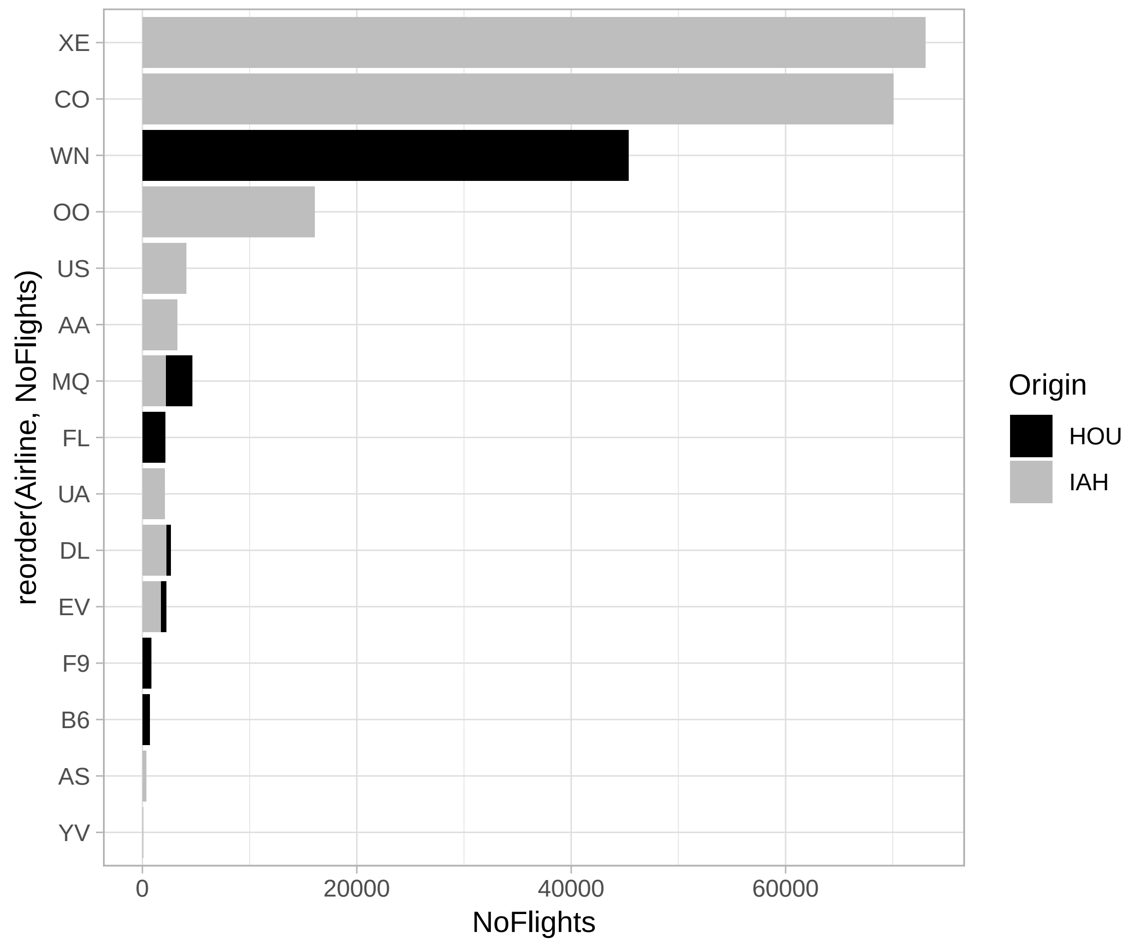


We can also create a stacked barplot, distinguishing between the two Houston airports.

```r
table(data$Airline)
```

```
##
##    AA    AS    B6    CO    DL    EV    F9    FL    MQ    OO    UA    US
##  3244   365   695 70032  2641  2204   838  2139  4648 16061  2072  4082
##    WN    XE    YV
## 45343 73053    79
```

```r
carriers2 <- data %>%
  group_by(Airline, Origin) %>%
  summarise(NoFlights = n()) %>%
  arrange(desc(NoFlights))

# Using default geom_bar(stat = "bin#) on the original data
ggplot(carriers2,
       aes(x = reorder(Airline, NoFlights),
           y = NoFlights,
           fill = Origin)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  scale_fill_manual(values = c("black", "grey")) +
  theme_light()
```



2.5 Joins

`dplyr` has powerful tools to merge data frames together. Because we want to focus on data visualization here, I will not go over all possible joints in depth. Please see the Data Wrangling Cheat Sheet and the dplyr documentation for more details.

Suppose, we have two data frames: `x` and `y`. The basic syntax for data merging with `dplyr` is the following:

`output <- join(A, B, by = "variable")`

We will focus on the following three join functions:

- `left_join()`: Join only those rows from `y` that appear in `x`, retaining all data in `x`. Here, `x` is the "master."
- `right_join()`: Join only those rows from `x` that appear in `y`, retaining all data in `y`. Here, `y` is the "master."
- `full_join()`: Join data from `x` and `y` upon retaining all rows and values. This is the maximum join possible. Neither `x` nor `y` is the "master."

For demonstration purposes, lets create a new data frame that contains the name of the city for each of the Greater Los Angeles Area airports.

```
loc_airport <- data.frame(code = c("LAX", "ONT", "SNA", "BUR"),
                          location = c("Los Angeles", "Ontario", "Santa Ana", "Burbank"))
loc_airport
```

```
##   code    location
## 1  LAX Los Angeles
## 2  ONT     Ontario
## 3  SNA   Santa Ana
## 4  BUR     Burbank
```

First, we treat the `la_flights` data frame as the master and join it with the data frame containing the airport locations using `left_join()`. If the variable names in both data frames were the same, `dplyr` would automatically join the correct columns. Here, we manually match the column names.

```
la_flights_new <- left_join(la_flights, loc_airport,
                            by = c("Dest" = "code"))
```

```
## Warning: Column `Dest`/`code` joining character vector and factor, coercing
## into character vector
```

```
table(la_flights_new$Dest)
```

```
##
##  LAX  ONT  SNA
## 6064  952 1661
```

Second, lets create a similar result using `right_join()`. Again, `la_flights` is the master data frame.

```
la_flights_new2 <- right_join(loc_airport, la_flights,
                              by = c("code" = "Dest"))
```

```
## Warning: Column `code`/`Dest` joining factor and character vector, coercing
## into character vector
```

```
table(la_flights_new2$code)
```

```
##
##  LAX  ONT  SNA
## 6064  952 1661
```

Finally, for demonstration, we create a third data frame using `full_join()`. Because all observations are retained, this join creates one observation with empty values for the Burbank value in `loc_airport`. For most applications, this would be an undesirable outcome. However, below, we use the fact that all possible values are retained to set up the data for visualization.

```
la_flights_new3 <- full_join(la_flights, loc_airport,
                             by = c("Dest" = "code"))
```

```
## Warning: Column `Dest`/`code` joining character vector and factor, coercing
## into character vector
```

```
table(la_flights_new3$Dest)
```

```
##
##  BUR  LAX  ONT  SNA
##    1 6064  952 1661
```

```
la_flights_new3[la_flights_new3$Dest == "BUR",]
```

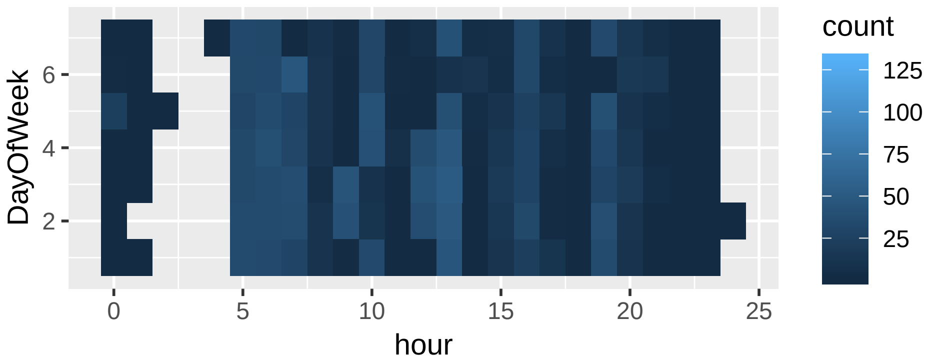
```
##      Month DayOfWeek DepTime ArrTime ArrDelay TailNum Airline Time Origin
## 8678    NA        NA      NA      NA       NA    <NA>    <NA>   NA   <NA>
##      Dest Distance TaxiIn TaxiOut TaxiTotal TaxiProp AirlineName location
## 8678  BUR       NA     NA      NA        NA       NA        <NA>  Burbank
```

3 Heatmaps

For this example, we will go back to our original `data` tibble that contains the complete set of flight data for the Houston airports in 2011. Suppose we wanted to know, what are the busiest times at each of the two Houston airports, George Bush Intercontinental/Houston Airport (IAH) and William P. Hobby Airport (HOU). We create a new summary data frame that counts the number of departures per hour and day for each of the airports. We display these data using heatmaps.

To do so, we need to create a new variable that codes the hour of departure, using information from the `DepTime` variable. There are more advanced workflows available using the `stringr` and/or `lubridate` packages (both are part of the `tidyverse`). However, because we want to focus on data visualization, I simply divide the departure time by 100 and then use the `floor()` function to extract the hour of departure.

```
departures <- data %>%

  mutate(hour = floor(DepTime/100))  %>%

  group_by(Origin,
           Dest,
           DayOfWeek,
           hour) %>%
  summarise(count = n())

ggplot(departures,
       aes(x = hour,
           y = DayOfWeek,
           fill = count)) +
  geom_tile()
```

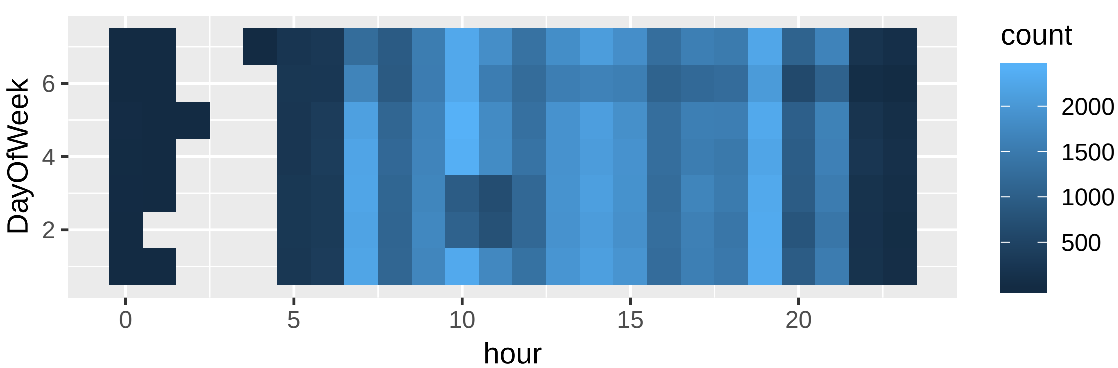


There are a number of ways to improve the plot. We will go through them step by step. Their is a weird observation in hour 24 that should not be there. The hour has to be either 23 or 0. Lets re-code this observation to an hour value of 0 using the `replace()` function from `dplyr`.

```
test <- data %>%
  filter(!is.na(DepTime)) %>%
  mutate(hour = floor(DepTime/100)) %>%
  filter(hour >= 24)
test
```

```
##   Month DayOfWeek DepTime ArrTime ArrDelay TailNum Airline Time Origin
## 1     5         2    2400     144      310  N14940      XE  104    IAH
##   Dest Distance TaxiIn TaxiOut hour
## 1  DFW      224      8      17   24
```

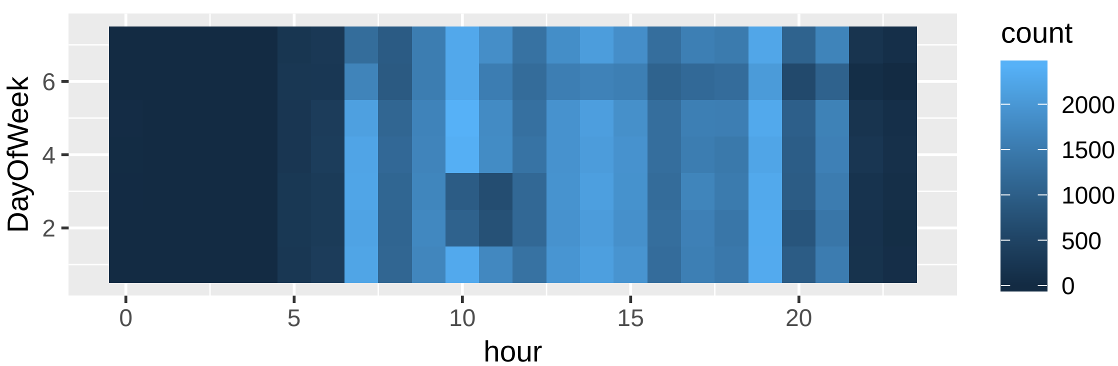
```
# Recoding
departures <- data %>%
  filter(!is.na(DepTime)) %>%
  mutate(hour = floor(DepTime/100))  %>%
  mutate(hour = replace(hour, hour >= 24, 0)) %>%
  group_by(Origin,
           DayOfWeek,
           hour) %>%
  summarise(count = n())

ggplot(departures,
       aes(x = hour,
           y = DayOfWeek,
           fill = count)) +
  geom_tile()
```



There are a number of possible observations that do not have value in the data frame, in particular in the early morning ours. For this application, we can assume that that these observations are not actually missing, but that there are no flights during these time slots.

Therefore, we create a data frame with all possible combinations of the variable values for day of the week and hour using `expand.grid()`, and use the `full_join()` function to create a new data frame. Similar to the application above, this procedure will result in missing values. We again use the `replace` function to re-code these missing values to zero.

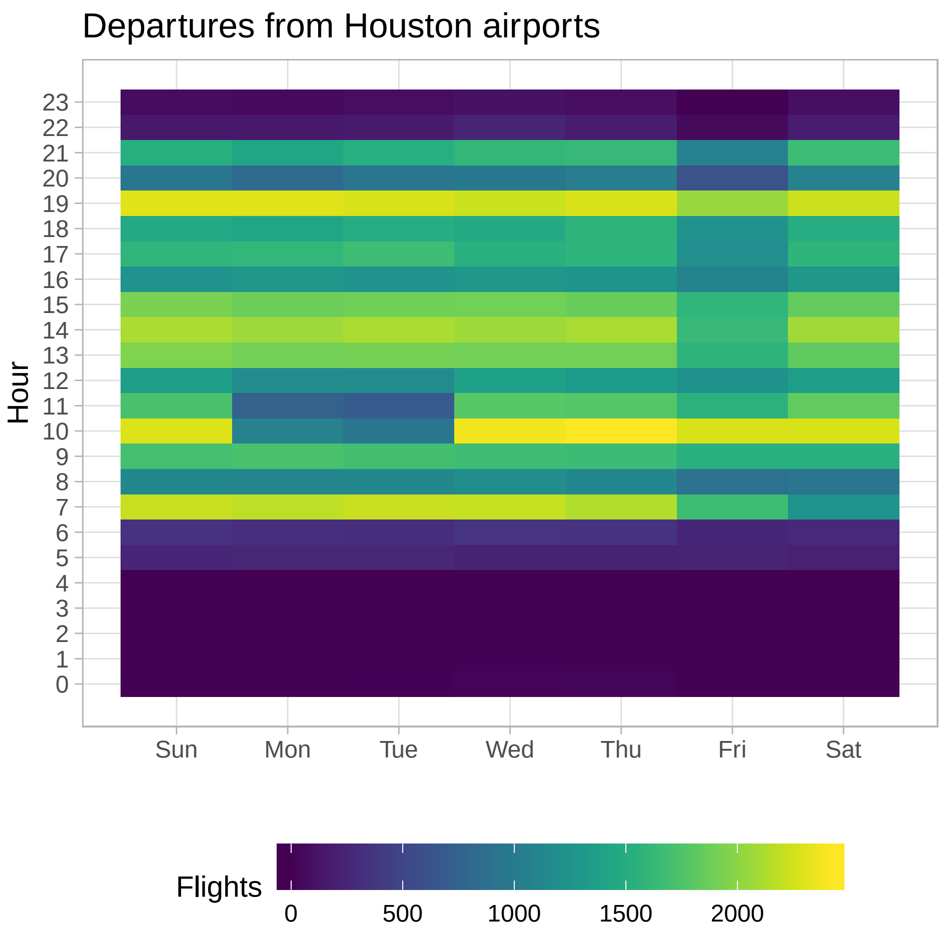
```
# Empty data frame
combo <- expand.grid(DayOfWeek = seq(1, 7),
                     hour = seq(0,23),
                     Origin = c("IAH", "HOU"))

# Merging
departures <- data %>%
  filter(!is.na(DepTime)) %>%
  mutate(hour = floor(DepTime/100))  %>%
  mutate(hour = replace(hour, hour >= 24, 0)) %>%
  group_by(Origin,
           DayOfWeek,
           hour) %>%
  summarise(count = n()) %>%

  # joining empty data frame
  full_join(combo) %>%

  # replacing missing values with zero
  mutate(count = replace(count, is.na(count), 0))

# visualizing it
ggplot(departures,
       aes(x = hour,
           y = DayOfWeek,
           fill = count)) +
  geom_tile()
```



Now, lets change the appearance of the graph. Below, we use color scales from the `viridis` package.

```
# install.packages("viridis")
library(viridis)


# visualizing it
ggplot(departures,
```

```
    aes(x = hour,
        y = DayOfWeek,
        fill = count)) +
  geom_tile() +
  scale_fill_viridis(name = "Flights") +
  scale_x_continuous(breaks = seq(0,23)) +
  scale_y_continuous(breaks = seq(1,7),
                     labels = c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")) +
  theme_light() +
  theme(panel.grid.minor = element_blank(),
        legend.position = "bottom",
        legend.key.width = unit(1.5, "cm")) +
  coord_flip() +
  labs(x = "Hour",
       y = "",
       title = "Departures from Houston airports")
```

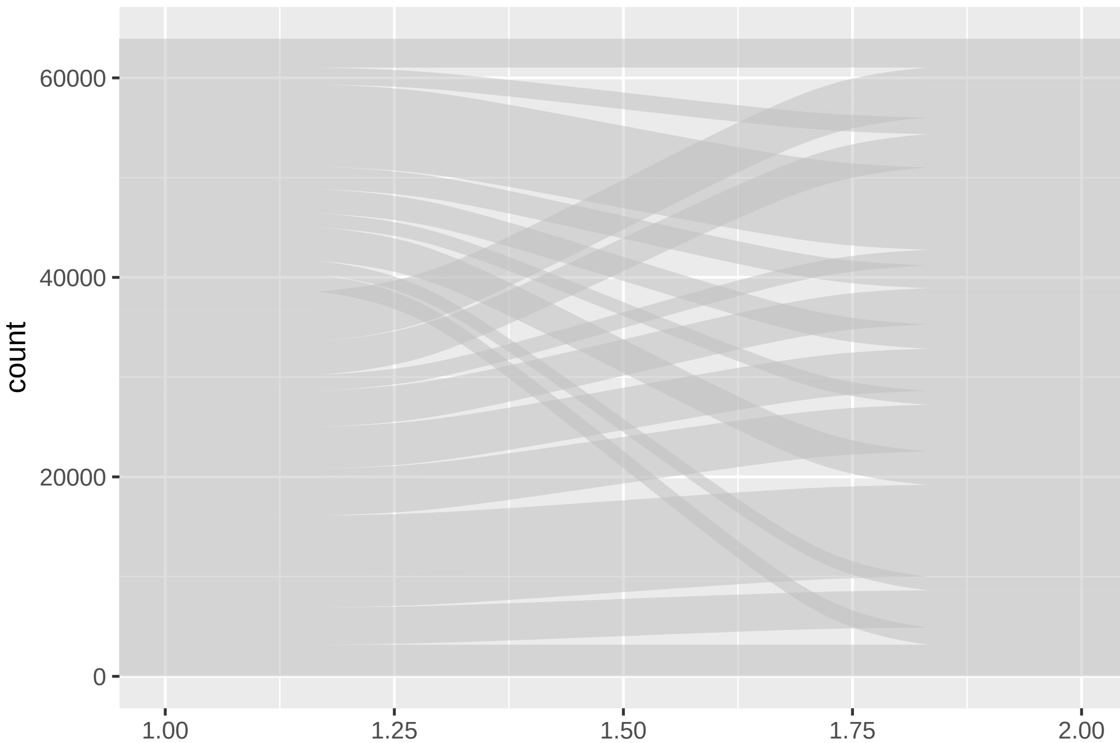


4 Alluvial diagrams

What are the flows between the two Houston airports and the ten most common destinations? We can visualize the combination of origin airport (IAH versus HOU) and the destination airport using alluvial diagrams. Below, we use the `ggalluvial` package, which contains the `geom_alluvium()` aesthetic.

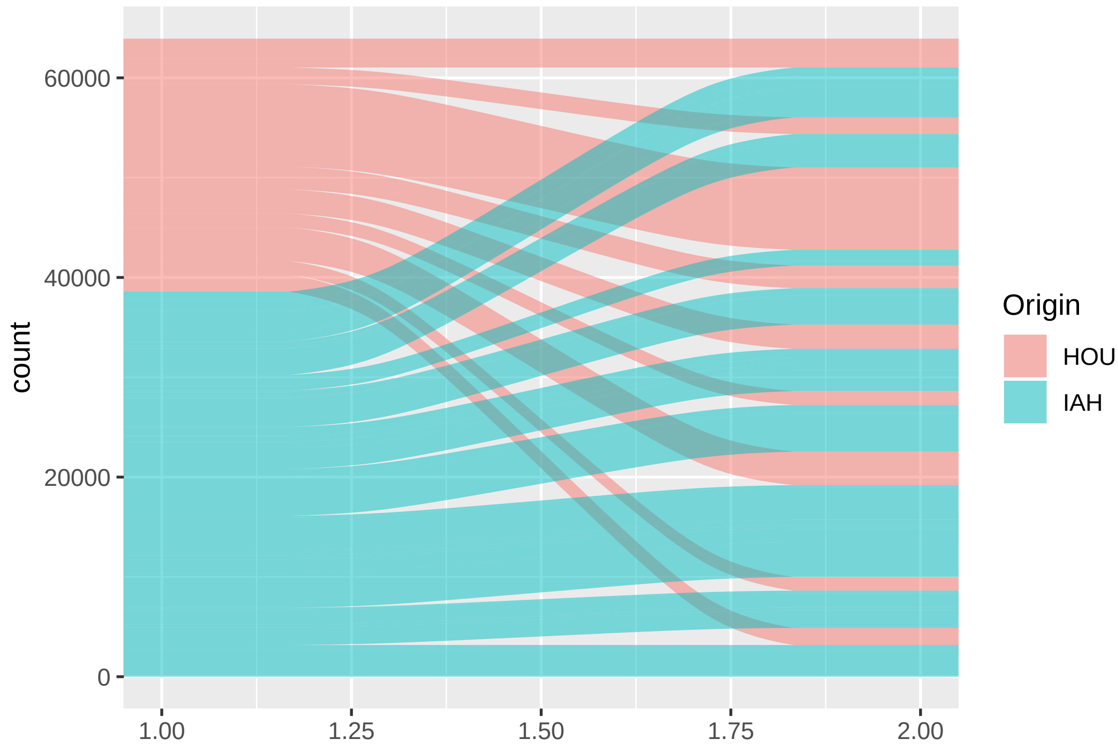
First, we create a frequency table for all observed combinations of origin and destination airport for the ten most common destinations using `group_by()` and `slice()`.

```
dest_top10 <- data %>%
  group_by(Dest) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  slice(1:10)

flows <- data %>%
  filter(Dest %in% dest_top10$Dest) %>%
  group_by(Origin,
           Dest,
           Airline) %>%
  summarise(count = n())

# install.packages("ggalluvial")
library(ggalluvial)
ggplot(flows,
       aes(y = count,
           axis1 = Origin,
           axis2 = Dest)) +
  geom_alluvium()
```



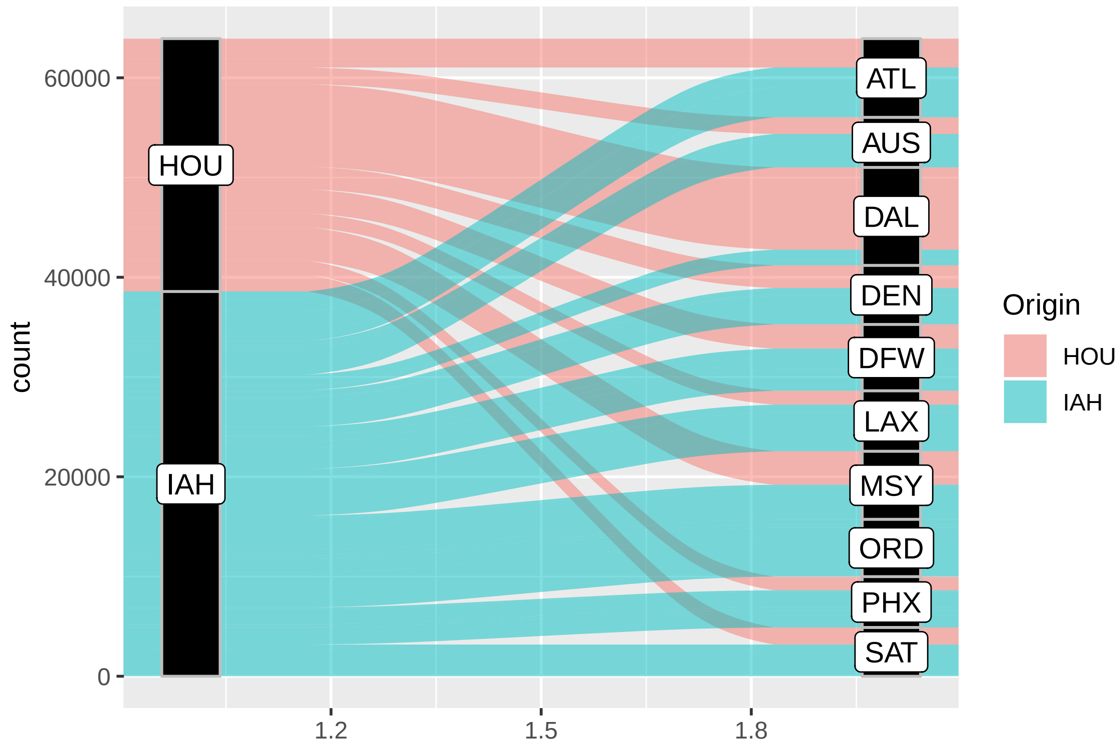
We can use fill to make the graph more interesting.

```
ggplot(flows,
       aes(y = count,
           axis1 = Origin,
           axis2 = Dest)) +
  geom_alluvium(aes(fill = Origin))
```



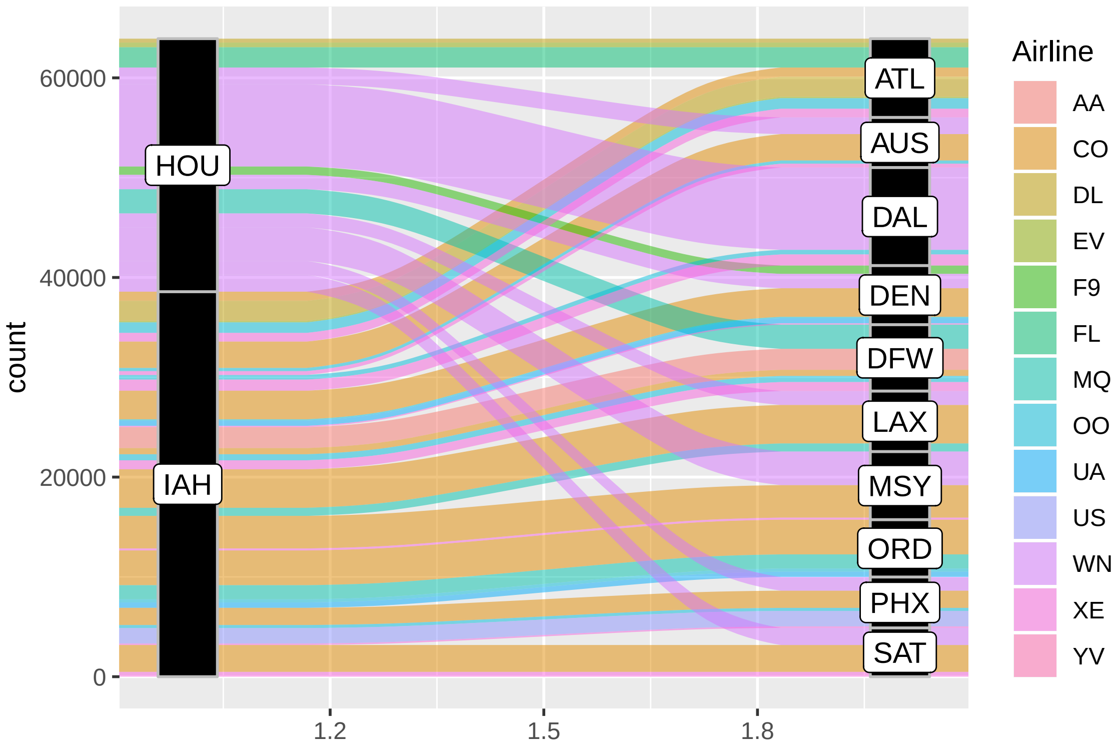
We can add labels to illustrate the destination airport. We also add the `geom_stratum()` aesthetic to clarify the grouping.

```
ggplot(flows,
       aes(y = count,
           axis1 = Origin,
           axis2 = Dest)) +
  geom_alluvium(aes(fill = Origin)) +
  geom_stratum(width = 1/12, fill = "black", color = "grey") +
  geom_label(stat = "stratum", label.strata = TRUE)
```



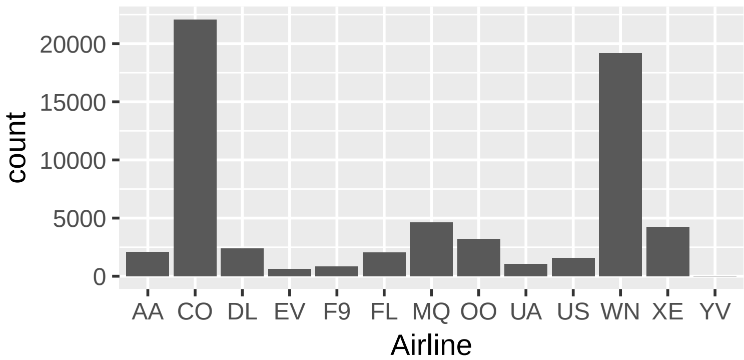
The plot above looks nice, but the distinction by fill is not necessarily needed. We could instead display an additional variable, for example the airline.

```
ggplot(flows,
       aes(y = count,
           axis1 = Origin,
           axis2 = Dest)) +
  geom_alluvium(aes(fill = Airline)) +
  geom_stratum(width = 1/12, fill = "black", color = "grey") +
  geom_label(stat = "stratum", label.strata = TRUE)
```



The plot above is hardly legible because there are too many airlines displayed. Lets only label the most common ones. First, we create a quick barplot to check who are the most common carriers on the top ten routes. Then, create a new variable coding only the most common, i.e. Continental (CO), Southwest (WN), and Other using `case_when()`.

```r
ggplot(flows,
       aes(x = Airline,
           y = count)) +
  geom_bar(stat = "identity")
```

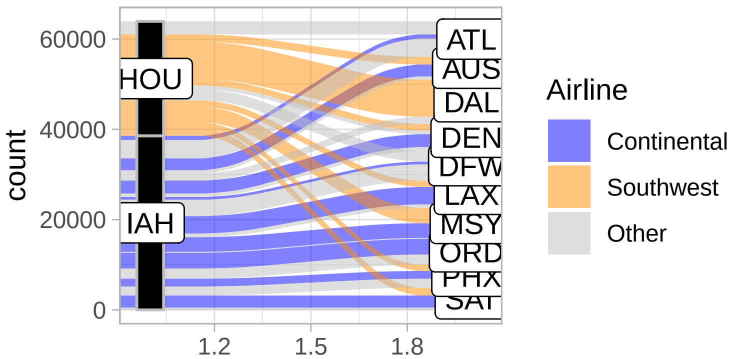



```r
# Creating new indicator
flows <- flows %>%
  mutate(Airline_reduced = case_when(
    Airline == "CO" ~ "Continental",
    Airline == "WN" ~ "Southwest",
    T ~ "Other"
  ) %>% factor(levels = c('Continental', 'Southwest', 'Other')))
table(flows$Airline_reduced)
```

```
##
```

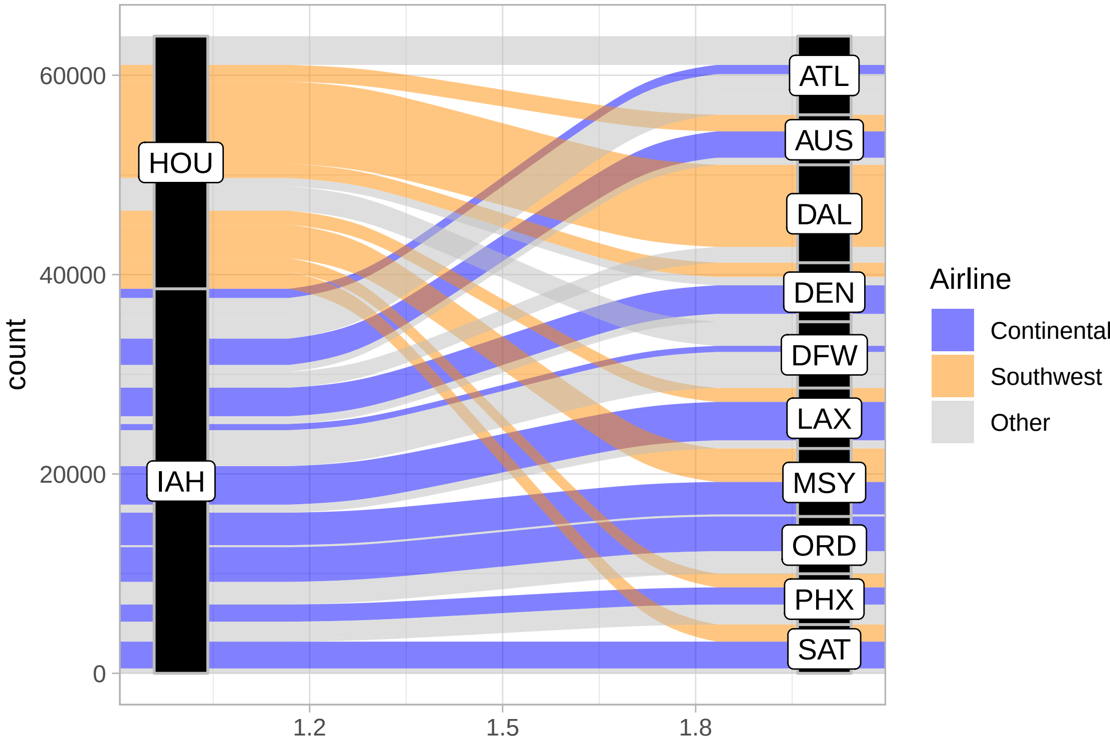
```
## Continental   Southwest       Other
##           9           7          30
```

```r
# Re-plotting alluvial diagram
ggplot(flows,
       aes(y = count,
           axis1 = Origin,
           axis2 = Dest)) +
  geom_alluvium(aes(fill = Airline_reduced)) +
  geom_stratum(width = 1/12, fill = "black", color = "grey") +
  geom_label(stat = "stratum", label.strata = TRUE) +
  scale_fill_manual(name = "Airline",
                    values = c("Continental" = "blue",
                               "Southwest" = "darkorange",
                               "Other" = "grey")) +
  theme_light()
```



Now, we can re-plot the alluvial diagram.

```r
ggplot(flows,
       aes(y = count,
           axis1 = Origin,
           axis2 = Dest)) +
  geom_alluvium(aes(fill = Airline_reduced)) +
  geom_stratum(width = 1/12, fill = "black", color = "grey") +
  geom_label(stat = "stratum", label.strata = TRUE) +
  scale_fill_manual(name = "Airline",
                    values = c("Continental" = "blue",
                               "Southwest" = "darkorange",
                               "Other" = "grey")) +
  theme_light()
```



5 Primer on `tidyr`

Another important task in data management is data re-shaping. Often, data does not come in the format that we need for data merging, data visualization, statistical analysis, or vectorized programming.

The `tidyr` package offers two main functions for data re-shaping:

- `gather()`: Shaping data from wide to long.
- `spread()`: Shaping data from long to wide.

5.1 Wide versus long data

For **wide** data formats, each unit's responses are in a single row. For example:

Country	Area	Pop1990	Pop1991
A	300	56	58
B	150	40	45

For **long** data formats, each row denotes the observation of a unit at a given point in time. For example:

Country	Year	Area	Pop
A	1990	300	56
A	1991	300	58
B	1990	150	40
B	1991	150	45

5.2 gather()

We use the `gather()` function to reshape data from wide to long. In general, the syntax of the data is as follows:

```
new_df <- gather(old_df, key, value, columns to gather)
```