Setting the stage for data science: integration of data management skills in introductory and second courses in statistics

Nicholas J. Horton, Benjamin S. Baumer, and Hadley Wickham

March 25, 2015

Statistics students need to develop the capacity to make sense of the staggering amount of information collected in our increasingly data-centered world. In her 2013 book, Rachel Schutt succinctly summarized the challenges she faced as she moved into the workforce: "It was clear to me pretty quickly that the stuff I was working on at Google was different than anything I had learned at school." The same challenges are seen by students who enter the workforce as analysts. The widely cited McKinsey report called for the training of hundreds of thousands of workers with the skills to make sense of the rich and sophisticated data now available to make decisions (along with millions of new managers with the ability to comprehend these results). The disconnect between the complex analyses now demanded in industry and the instruction available in academia is a major challenge for the profession. Statistical educators play a key role in helping to prepare the next generation of statisticians and data scientists.

The updated ASA guidelines for undergraduate programs in statistics (American Statistical Association, 2014) describe a number of specific capacities needed to make sense of the data around us. This includes data management and related skills (see SIDEBAR: What's in a name), visualization, knowledge of statistics, experience with forecasting and prediction, and sophisticated communication skills. Nolan and Temple Lang (2010) argued that "the ability to express statistical computations is an essential skill," and that major changes to foster this capacity are needed in the statistics curriculum at the graduate and undergraduate levels. Providing students with a basis in data manipulation and management skills can help ensure that they develop the ability to frame and answer statistical questions with richer supporting data. Inculcating our students with these general capacities, distinct from ungrounded training in specific technologies, may help them thrive in an era where data analysis techniques can extract meaning from larger and more complex data.

Instilling a framework for "Data Tidying" (see also http://www.jstatsoft.org/v59/i10/paper) and creating opportunities for students to practice with real, messy datasets are a necessary but not sufficient way to engage students. To help achieve these aims, important precursors to data science need to be incorporated into a range of introductory and second courses in statistics. (If not then, when?)

Successful implementation is likely to be facilitated by the routine use of reproducible analysis tools (described in detail in "Five Concrete Reasons Your Students Should Be Learning to Analyze Data in the Reproducible Paradigm", http://chance.amstat.org/2014/09/reproducible-paradigm) into first and second stage courses, and we assume that such approaches are used (to model this we have made copies of the R Markdown and formatted files for these analyses available at http://www.amherst.edu/~nhorton/precursors).

A framework for data-related skills The statistical data analysis cycle involves the formulation of questions, collection of data, analysis, and interpretation of results (see Figure 1). Data preparation and manipulation is not just a first step, but a key component of this cycle (which will often be nonlinear). When working with data, analysts must first determine what is needed, describe this solution in terms that a computer can understand, and execute the code. Here we illustrate how the `dplyr` package in R (http://cran.r-project.org/web/packages/dplyr) can be used to build a powerful and broadly accessible foundation for data manipulation. This approach is attractive because it provides simple functions that correspond to the most common data manipulation operations (or *verbs*) and uses efficient storage approaches so that the analyst can focus on the analysis. (Other systems could certainly be used in this manner, see for example http://iase-web.org/icots/9/proceedings/pdfs/ICOTS9_C134_CARVER.pdf.)

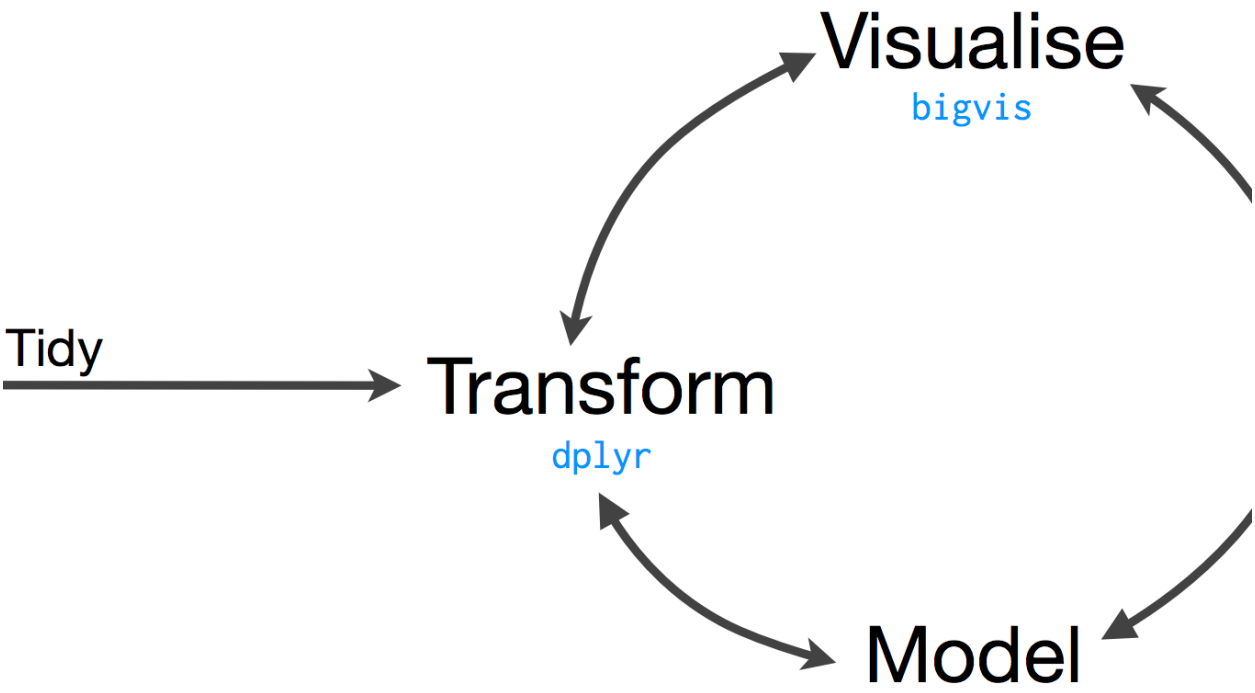

Figure 1: Statistical data analysis cycle (source: bit.ly/bigrdata4)

verb	meaning
select()	select variables (or columns)
filter()	subset observations (or rows)
mutate()	add new variables (or columns)
arrange()	re-order the observations
summarise()	reduce to a single row
group_by()	aggregate
left_join()	merge two data objects
distinct()	remove duplicate entries
collect()	force computation and bring data back into R

Table 1: Key verbs to support data management and manipulation (see http://bit.ly/bigrdata4 for more details)

To illustrate these verbs in action, we consider analysis of airline delays in the United States. This dataset, constructed from information made available by the Bureau of Transportation Statistics, was utilized in the ASA Data Expo 2009 (Wickham, JCGS, 2011). This rich data repository contains more than 150 million observations corresponding to each commercial airline flight in the United States between 1987 and 2012.

Because of the magnitude of data (larger than can be easily accommodated in R or similar general purpose statistical packages), access via a database is recommended (see SIDEBAR: Databases). We demonstrate how to undertake analysis using the tools in the `dplyr` package. (A smaller dataset is available for New York City flights in 2013 within the `nycflights13` package. The interface in R looks almost identical in terms of the `dplyr` functionality, with the same functions being used.)

Students can use this dataset to address questions that they find real and relevant. (It is not hard to find motivation for investigating patterns of flight delays. Ask students: have you ever been stuck in an airport because your flight was delayed or cancelled and wondered if you could have predicted the delay if you'd had more data?)

```
now <- Sys.time(); now
```

```
## [1] "2015-03-25 14:36:34 EDT"
```

```
require(dplyr); require(mosaic); require(lubridate); require(igraph)
my_db <- src_sqlite(path="ontime.sqlite3")
```

This example uses data from a database with multiple tables.

```
flights <- tbl(my_db, "flights")   # link to some useful tables
airports <- tbl(my_db, "airports")
airlines <- tbl(my_db, "airlines")
airplanes <- tbl(my_db, "airplanes")
```

We start with an analysis focused on three smaller airports in the Northeast. This illustrates the use of `filter()`, which allows the specification of a subset of rows of interest in the `airports` table (or dataset). We first start by exploring the `airports` table.

```
filter(airports, IATA %in% c('ALB', 'BDL', 'BTV'))   # what are these airports?
```

```
## Source: sqlite 3.8.6 [ontime.sqlite3]
## From: airports [3 x 7]
## Filter: IATA %in% c("ALB", "BDL", "BTV")
##
##   IATA                  Airport          City State Country Latitude
## 1  ALB               Albany Cty        Albany    NY     USA 42.74812
## 2  BDL    Bradley International Windsor Locks    CT     USA 41.93887
## 3  BTV Burlington International    Burlington    VT     USA 44.47300
## Variables not shown: Longitude (dbl)
```

Next we aggregate the counts of flights at all three of these airports at the monthly level (in the `flights` flight-level table), using the `group_by()` and `summarise()` functions. The `collect()` function forces the evaluation. These functions are collected using the `%>%` operator allows the results from one object or function to be *piped* to the next in an efficient manner.

```
airportcounts <- flights %>%
   filter(Dest %in% c('ALB', 'BDL', 'BTV')) %>%
   group_by(Year, Month, Dest) %>%
   summarise(count = n()) %>%
   collect()
```

Next we add a new column by constructing a date variable (using `mutate()` and helper functions from the `lubridate` package), then generate a time series plot.

```
airportcounts <- airportcounts %>%
  mutate(Date = ymd(paste(Year, "-", Month, "-01", sep="")))
head(airportcounts)
```

```
## Source: local data frame [6 x 5]
## Groups: Year, Month
##
##   Year Month Dest count       Date
## 1 1987    10  ALB   957 1987-10-01
## 2 1987    10  BDL  2580 1987-10-01
## 3 1987    10  BTV   549 1987-10-01
## 4 1987    11  ALB   950 1987-11-01
## 5 1987    11  BDL  2442 1987-11-01
## 6 1987    11  BTV   496 1987-11-01
```

```
xyplot(count ~ Date, groups=Dest, type=c("p","l"), lwd=2, auto.key=list(columns=3),
       ylab="number of flights per month", xlab="Year", data=airportcounts)
```

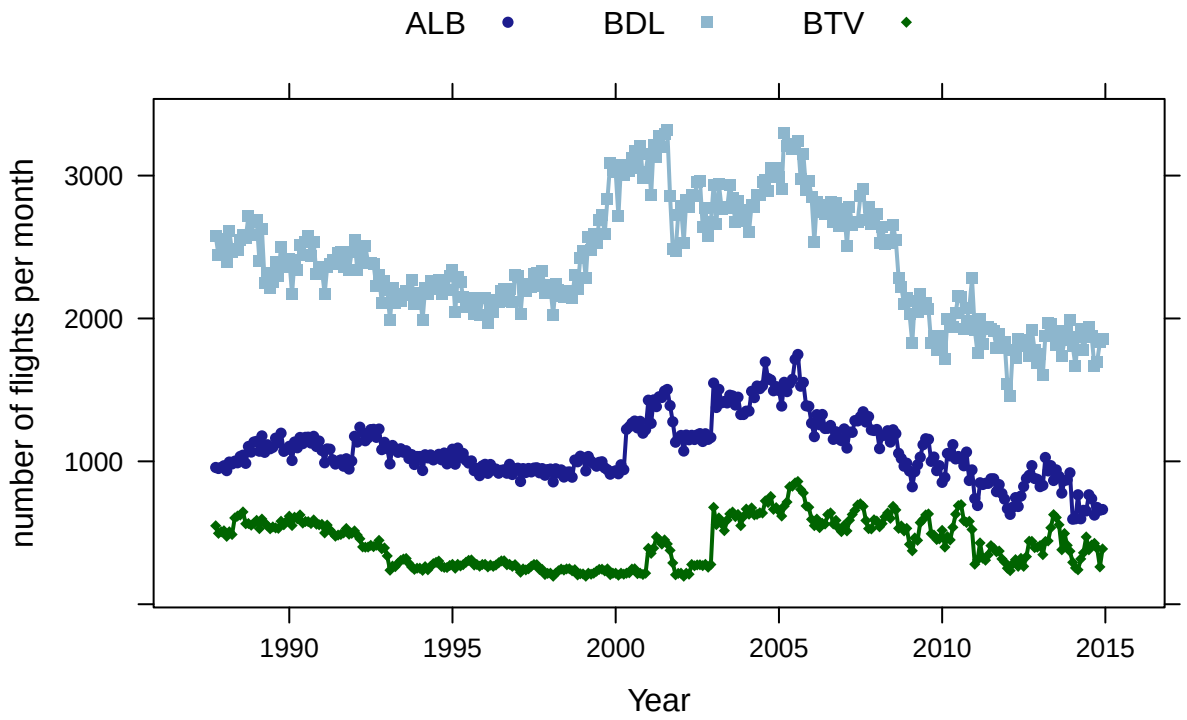

Figure 2: Comparison of the number of flights arriving at three airports over time

We observe in Figure 2 that there are some interesting patterns over time for these airports. Bradley (serving Hartford, CT and Springfield, MA) has the largest monthly volumes, with Burlington the least number of flights. At all three airports, there has been a decline in the number of flights from 2005 to 2012.

Another important verb is `arrange()`, which in conjunction with `slice()` lets us display the months with the largest number of flights. Here we need to use `ungroup()`, since otherwise the data would remain aggregated by year, month, and destination.

```
airportcounts %>%
  ungroup() %>%
  arrange(desc(count)) %>%
```

```
  select(count, Year, Month, Dest) %>%
  head() # list only the first six observations
```

```
## Source: local data frame [6 x 4]
##
##   count Year Month Dest
## 1  3318 2001     8  BDL
## 2  3299 2005     3  BDL
## 3  3289 2001     7  BDL
## 4  3279 2001     5  BDL
## 5  3242 2005     8  BDL
## 6  3219 2005     5  BDL
```

Another analysis might compare flight delays between two airlines serving a city pair. For example, which airline was most reliable flying from Chicago to Minneapolis/St. Paul in January, 2012? We can calculate an average delay for each day for United, Delta, and American. (Note that we do not address flight cancellations: this exercise is left to the reader.)

Let's begin by taking a closer look at what variables are available, using a direct SQL call (see SIDEBAR: Databases).

```
tbl(my_db, sql("SELECT * FROM flights LIMIT 10"))  # or tbl(my_db, "flights") %>% head()
```

```
## Source: sqlite 3.8.6 [ontime.sqlite3]
## From: <derived table> [?? x 29]
##
##    Year Month DayOfMonth DayOfWeek DepTime CRSDepTime ArrTime CRSArrTime
## 1  1987    10         14         3     741        730     912        849
## 2  1987    10         15         4     729        730     903        849
## 3  1987    10         17         6     741        730     918        849
## 4  1987    10         18         7     729        730     847        849
## 5  1987    10         19         1     749        730     922        849
## 6  1987    10         21         3     728        730     848        849
## 7  1987    10         22         4     728        730     852        849
## 8  1987    10         23         5     731        730     902        849
## 9  1987    10         24         6     744        730     908        849
## 10 1987    10         25         7     729        730     851        849
## ..  ...   ...        ...       ...     ...        ...     ...        ...
## Variables not shown: UniqueCarrier (chr), FlightNum (int), TailNum (int),
##   ActualElapsedTime (int), CRSElapsedTime (int), AirTime (int), ArrDelay
##   (int), DepDelay (int), Origin (chr), Dest (chr), Distance (int), TaxiIn
##   (int), TaxiOut (int), Cancelled (int), CancellationCode (int), Diverted
##   (int), CarrierDelay (int), WeatherDelay (int), NASDelay (int),
##   SecurityDelay (int), LateAircraftDelay (int)
```

We create the analytic dataset through use of `select()` (to pick the variables to be included), `filter()` (to select a tiny subset of the observations) and repeat the previous aggregation.

```
delays <- flights %>%
  select(Origin, Dest, Year, Month, DayOfMonth, UniqueCarrier, ArrDelay) %>%
  filter(Origin == 'ORD' & Dest == 'MSP' & Year == 2012 & Month == 1 &
         (UniqueCarrier %in% c("UA", "MQ", "DL"))) %>%
```

```
  group_by(Year, Month, DayofMonth, UniqueCarrier) %>%

  summarise(meandelay = mean(ArrDelay), count = n()) %>%
  collect()
```

Merging is another key capacity for students to master. Here, the full carrier names are merged (or joined, in SQL parlance) to facilitate the comparison, using the `left_join()` function to provide a less terse full name for the airlines.

```
carriernames <- airlines %>%
  filter(Code %in% c("UA", "MQ", "DL")) %>%
  collect()
merged <- left_join(delays, carriernames, by=c("UniqueCarrier" = "Code"))
head(merged)
```

```
## Source: local data frame [6 x 7]
## Groups: Year, Month, DayofMonth
##
##   Year Month DayofMonth UniqueCarrier meandelay count
## 1 2012     1          1            MQ  8.750000     4
## 2 2012     1          1            UA -5.000000     2
## 3 2012     1          2            DL  2.333333     3
## 4 2012     1          2            MQ 24.166667     6
## 5 2012     1          2            UA -3.000000     3
## 6 2012     1          3            DL  8.000000     2
## Variables not shown: Description (chr)
```

```
densityplot(~ meandelay, group=UniqueCarrier, auto.key=TRUE, xlab="Average daily delay (in minutes)",
  data=merged)
```

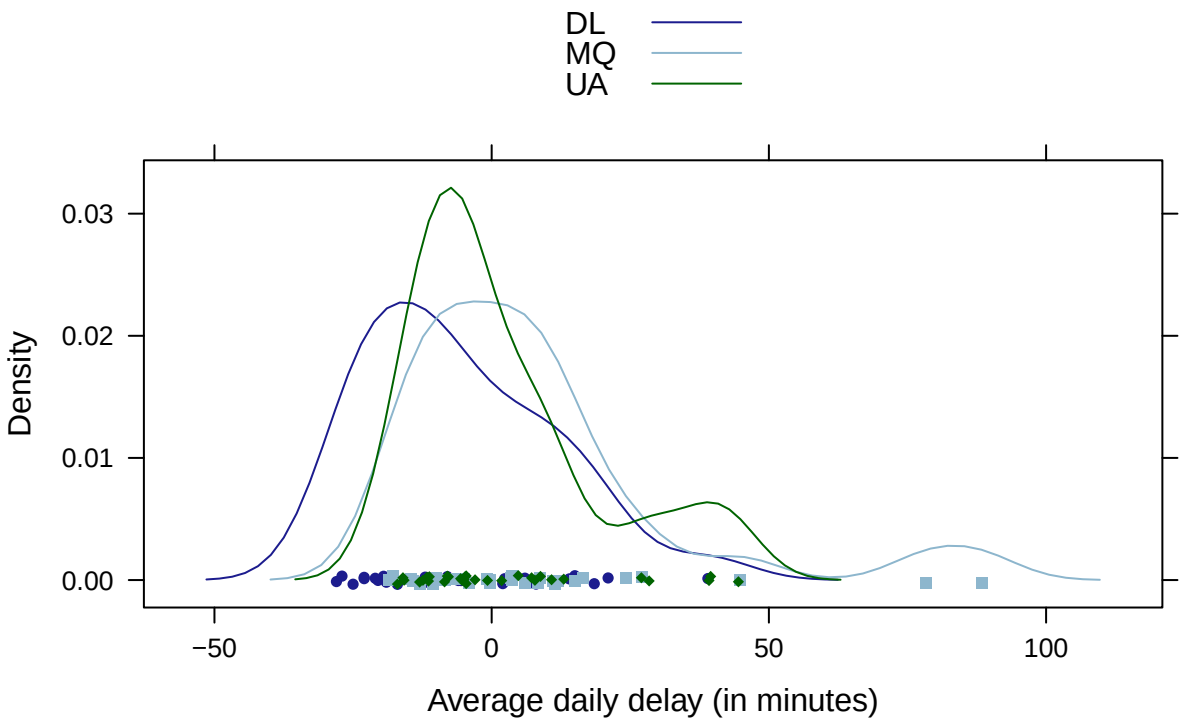

Figure 3: Comparison of mean flight delays from O'Hare to Minneapolis/St. Paul in January, 2012 for three airlines

We see in Figure 3 that the airlines are fairly reliable, though there were some days with average delays of 50 minutes or more (three of which were accounted for by American Eagle).

```
filter(delays, meandelay > 50)
```

```
## Source: local data frame [2 x 6]
## Groups: Year, Month, DayofMonth
##
##   Year Month DayofMonth UniqueCarrier meandelay count
## 1 2012     1         13            MQ  78.33333     6
## 2 2012     1         21            MQ  88.40000     5
```

Finally, we can drill down to explore all flights on Mondays in the year 2001.

```
Mondayflights <- flights %>%
  filter(DayofWeek==1 & Year > 2000 & Year < 2002) %>%
  group_by(Year, Month, DayOfMonth) %>%
  summarise(count = n()) %>%
  collect()
Mondayflights <- mutate(Mondayflights, Date=ymd(paste(Year, "-", Month, "-", DayOfMonth, sep="")))
xyplot(count ~ Date, type="l", ylab="Count of flights on Mondays", data=Mondayflights)
ladd(panel.abline(v=ymd("2001-09-11"), lty=2))
```

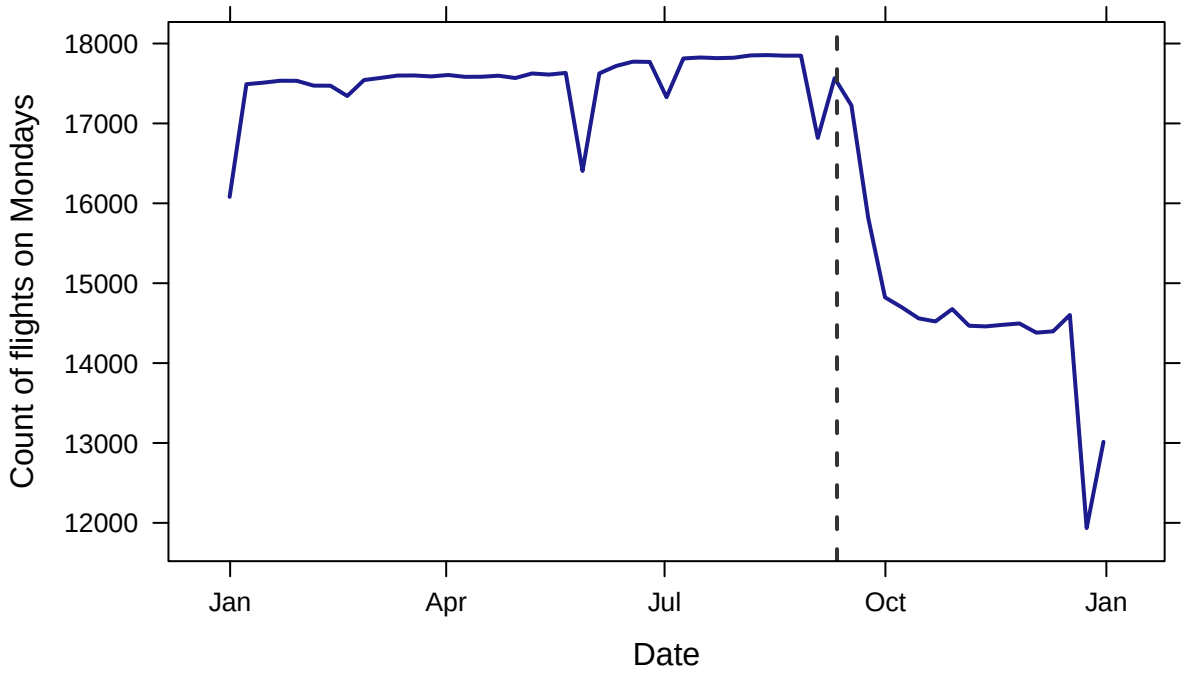

Figure 4: display of counts of commercial flights on Mondays in the United States in 2001. The clear impact of 9/11 can be seen.

Integrating bigger questions and datasets into the curriculum This opportunity to make a complex and interesting dataset accessible to students in introductory statistics is quite compelling. In the introductory

(or first) statistics course, exploration of airline delays was introduced without any technology through use of the "Judging Airlines" model eliciting activity (MEA) developed by the CATALST Group (http://serc.carleton.edu/sp/library/mea/examples/example5.html). This MEA guides students to develop ideas regarding center and variability and the basics of informal inferences using small samples of data for pairs of airlines flying out of Chicago.

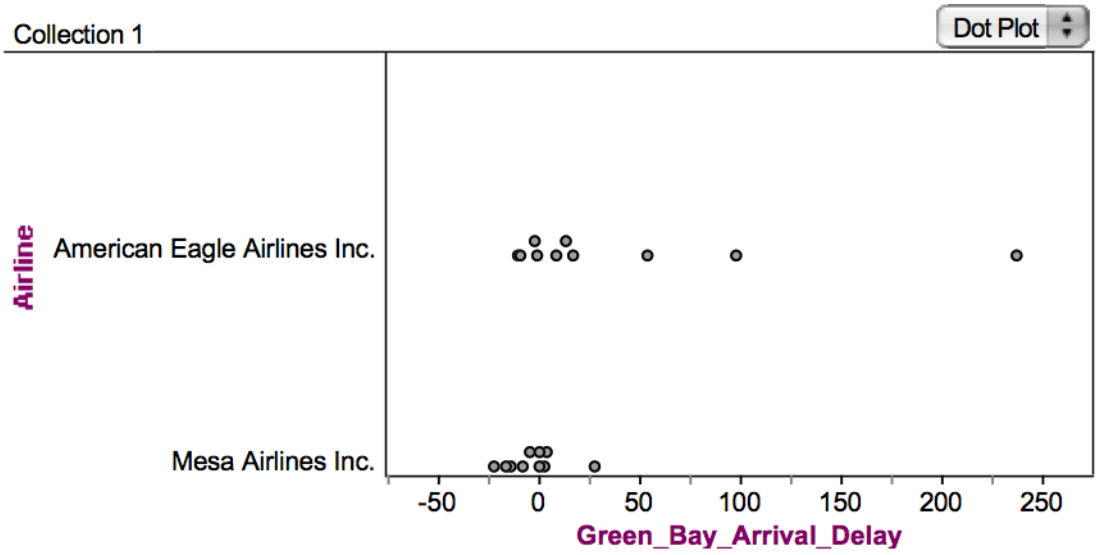

Figure 5: display of sample airline delays data for a city pair used in the "Judging Airlines" MEA

Figure 5 displays sample airline delays for ten flights each for American Eagle Airlines and Mesa Airlines flying from Chicago to Green Bay, Wisconsin. As part of this activity, students need to describe five possible sample statistics which could be used to compare the flight delays by airline. These might include the average, the maximum, the median, the 90th percentile, or the fraction that are late. Finally, they need to create a rule that incorporates at least two of those summary statistics that can be used to make a decision about whether one airline is more reliable. A possible rule might be to declare an airline is better than another if that airline has half an hour less average delay, and that same airline has 10% less delayed flights than the other (if the two measures of reliability differ in direction for the two airlines, no call is made).

To finish the assignment, students are provided with data for another four city pairs, asked to carry out their rule on these new "test" datasets, then summarize their results in a letter to the editor of *Chicago Magazine*.

Later in the course, the larger dataset can be reintroduced in several ways. It can be brought into class to illustrate univariate summaries or bivariate relationships (including more sophisticated visualization and graphical displays). Students can pose questions through projects or other extended assignments. A lab activity could have students explore their favorite airport or city pair (when comparing two airlines they will often find that only one airline services that connection, particularly for smaller airports.) Students could be asked to return to the informal "rule" they developed in an extension to assess its performance. Their rule can be programmed in R, and then carried out on a series of random samples from the flights from that city on that airline within that year. This allows them to see how often their rule picked an airline as being more reliable (using various subsets of the observed data as the "truth"). Finally, students can summarize the population of all flights, as a way to better understand sampling variability. This process reflects the process followed by analysts working with big data: sampling is used to generate hypotheses that are then tested against the complete dataset.

In a second course, more time is available to develop diverse statistical and computational skills. This includes more sophisticated data management and manipulation with explicit learning outcomes that are a central part of the course syllabus.

Other data wrangling and manipulation capacities can be introduced and developed using this example, including more elaborate data joins/merges (since there are tables providing additional (meta)data about

planes). As an example, consider the paths of plane N355NB, which flew out of Bradley airport in January, 2008.

```
filter(airplanes, TailNum=="N355NB")
```

```
## Source: sqlite 3.8.6 [ontime.sqlite3]
## From: airplanes [1 x 9]
## Filter: TailNum == "N355NB"
##
##   TailNum        Type Manufacturer IssueDate    Model Status
## 1  N355NB Corporation       AIRBUS  11/14/02 A319-114  Valid
## Variables not shown: AircraftType (chr), EngineType (chr), Year (chr)
```

```
singleplane <- filter(flights, TailNum=="N355NB") %>%
  select(Year, Month, DayOfMonth, Dest, Origin, Distance) %>%
  collect()
singleplane %>%
  group_by(Year) %>%
  summarise(count = n(), totaldist = sum(Distance))
```

```
## Source: local data frame [13 x 3]
##
##    Year count totaldist
## 1  2002   152    136506
## 2  2003  1367   1224746
## 3  2004  1299   1144288
## 4  2005  1366   1149142
## 5  2006  1484   1149036
## 6  2007  1282   1010146
## 7  2008  1318   1095109
## 8  2009  1235   1094532
## 9  2010  1368   1143189
## 10 2011  1406    919893
## 11 2012  1213    807246
## 12 2013  1339    921673
## 13 2014  1114    766247
```

```
sum(~ Distance, data=singleplane)
```

```
## [1] 12561753
```

```
singleplane %>%
  group_by(Dest) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  filter(count > 300)
```

```
## Source: local data frame [9 x 2]
##
##   Dest count
## 1  MSP  2977
```

```
## 2  DTW  2371
## 3  LGA   865
## 4  MEM   854
## 5  ATL   841
## 6  SLC   564
## 7  DCA   429
## 8  BOS   380
## 9  SNA   319
```

We see that this Airbus A319 has been very active, with around 1300 flights per year since it came online in late 2002. It tends to spend much of its time in Minneapolis/St. Paul (MSP), and is has amassed more than 11 million miles in the air!

Mapping is also possible, since the latitude and longitude of the airports are provided. Figure 6 displays a map of flights from Bradley airport in 2013.

```
require(RSQLite)
con <- dbConnect(SQLite(), "ontime.sqlite3")

orig <- dbGetQuery(con, "SELECT * FROM airports WHERE IATA = 'BDL'")
dests <- dbGetQuery(con, "SELECT a.IATA, Latitude, Longitude, sum(1) as numFlights
FROM flights o LEFT JOIN airports a ON o.Dest = a.IATA
WHERE (Year = 2013 AND Origin = 'BDL' AND Cancelled != 0)
GROUP BY a.IATA")
```

```
require(igraph)
E <- data.frame(src = "BDL", dest = dests$IATA, weight = dests$numFlights)
V <- rbind(orig[,c("IATA", "Latitude", "Longitude")], dests[,c("IATA", "Latitude", "Longitude")])
airport <- graph.data.frame(E, directed = TRUE, V)
E(airport)$width <- with(E, 50 * weight / sum(weight))
require(maps)
map('state', xlim=c(-105, -65), ylim=c(25, 46))
plot(airport, layout=as.matrix(V[c("Longitude", "Latitude")])
     , add=TRUE, rescale=FALSE
     , edge.width = with(E, log(200 * weight / sum(weight)))
     , edge.curved=TRUE
     , edge.arrow.size = 0.3
     , vertex.size = 30, vertex.label.cex = 0.8
     , vertex.label.dist = 5, vertex.label.degree = ifelse(V$Latitude > 30, -pi/2, pi/2))
```

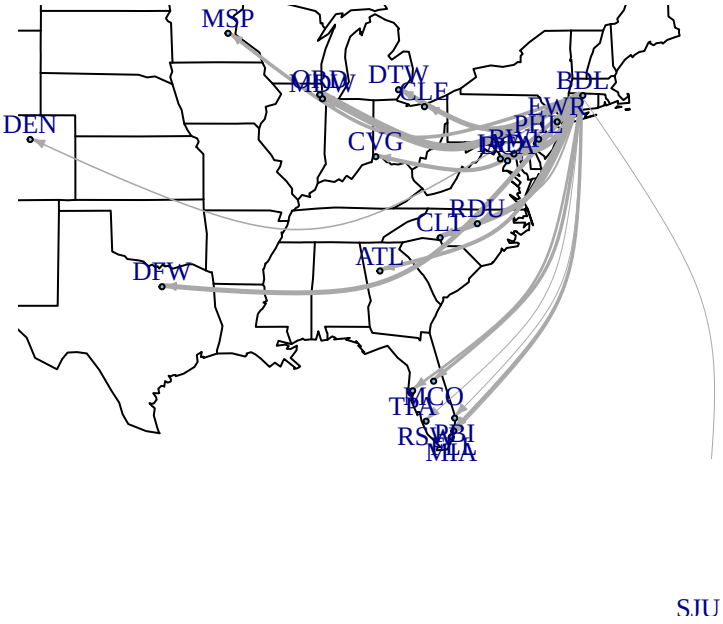

Figure 6: Map of flights from Bradley airport in 2013

Linkage to other data scraped from the Internet (e.g. detailed weather information for a particular airport or details about individual planes) may allow other questions to be answered (this has already been included in the `nycflights13` package (see SIDEBAR: Databases). Other approaches to analysis of big data in R (e.g. dplyr package) can also be introduced. Use of a database to access this rich dataset helps to excite students about the power of statistics, introduce tools that can help energize the next generation of data scientists, and build useful data-related skills.

Conclusion

Many have argued that statistics students need additional facility to express statistical computations. By introducing students to commonplace tools for data management, visualization, and reproducible analysis in data science and applying these to real-world scenarios, we prepare them to think statistically in the era of big data.

In addition, there have been other calls for an increased use of computing in the statistics curriculum at the undergraduate level (American Statistical Association, 2014). In an era of increasingly big data, we agree that this is an imperative to develop in students, beginning with the introductory course. Some in the data science world argue that statistics is only relevant for "small" or "medium data" and "traditional tools." We believe that the integration of these precursors to data science into our curricula—early and often—will help statisticians be part of the dialogue regarding *Big Data and Big Questions*.

Others have argued that statistics courses are mired in teaching techniques developed by pre-computer-era statisticians to circumvent their lack of computational power. We concur that there are barriers and costs to the introduction of reproducible analysis tools and more sophisticated data management and manipulation skills to our courses. Further guidance and research results are needed to guide our work in this area, along with illustrated examples, case studies, and faculty development. But these impediments must not slow down our adoption. As Schutt cautions in her book, statistics could be viewed as obsolete if this challenge is not embraced.

Finzer (2013) noted that such changes are also needed before university level, and that the U.S. K-12 education system "does not provide meaningful learning experiences designed to develop understanding of data science concepts or a fluency with data science skills". He concludes that statistics educators—who generally understand data, have substantial expertise in computation, and have developed a variety of data habits of mind—are well-positioned to advocate for major changes in the training of future data scientists. We

believe that the time to move forward in this manner is now, and believe that these these basic data-related skills provide a foundation for such efforts.

ACKNOWLEDGMENTS A previous version of this paper was presented in July, 2014 at the International Conference on Teaching Statistics (ICOTS9) in Flagstaff, AZ.

REFERENCES American Statistical Association Undergraduate Guidelines Workgroup (2014). 2014 Curriculum guidelines for undergraduate programs in statistical science. Alexandria, VA: American Statistical Association, http://www.amstat.org/education/curriculumguidelines.cfm.

Horton, N.J., Baumer, Ben S., and Wickham, H. (2014). Teaching precursors to data science in introductory and second courses in statistics, http://arxiv.org/abs/1401.3269.

Baumer, B., Cetinkaya-Rundel, M., Bray, A., Loi, L. & Horton, N.J. (2014). R Markdown: Integrating a reproducible analysis tool into introductory statistics, *Technology Innovations in Statistics Education*, http://escholarship.org/uc/item/90b2f5xh.

Finzer, W. (2013). The data science education dilemma. *Technology Innovations in Statistics Education*, http://escholarship.org/uc/item/7gv0q9dc.

Nolan, D. and Temple Lang, D. (2010). Computing in the statistics curricula, *The American Statistician*, 64, 97–107.

Wickham, H. (2011). ASA 2009 Data Expo, *Journal of Computational and Graphical Statistics*, 20(2):281-283.

SIDEBAR: What's in a word?

In their 2010 American Statistician paper, Deborah Nolan and Duncan Temple Lang describe the need for students to be able to "compute with data" to be able to answer statistical questions. Diane Lambert of Google calls this the capacity to "think with data". Statistics graduates need to be manage data, analyze it accurately, and communicate findings effectively. The Wikipedia data science entry states that "data scientists use the ability to find and interpret rich data sources, manage large amounts of data despite hardware, software, and bandwidth constraints, merge data sources, ensure consistency of datasets, create visualizations to aid in understanding data, build mathematical models using the data, present and communicate the data insights/findings to specialists and scientists in their team and if required to a non-expert audience." But what is the best word or phrase to describe these computational and data-related skills?

"Data wrangling" has been suggested as one possibility (and returned about 131,000 results on Google), though this connotes the idea of a long and complicated dispute, often involving livestock, which may not end well.

"Data grappling" is another option (about 7,500 results on Google), though this perhaps less attractive as it suggests pirates (and grappling hooks) or wrestling as combat sport or self defense.

"Data munging" (about 35,000 results on Google) is a common term in computer science used to describe changes to data (both constructive and destructive) or mapping from one format to another. A disadvantage of this term is that it has a somewhat pejorative sentiment.

"Data tidying" (about 900 results on Google) brings to mind the ideas of "bringing order to" or "arranging neatly".

"Data curation" (about 322,000 results on Google) is a term that focuses on a long-term time scale for use (and preservation). While important, this may be perceived a dusty and stale task.

"Data cleaning" (or "data cleansing", about 490,000 results on Google) is the process to identify and correct (or remove) invalid records from a dataset. Other related terms include "data standardization" and "data harmonization".

A search for "Data manipulation" yielded about 740,000 results on Google. Interestingly, this term on Wikipedia redirects to the "Misuse of statistics" page, implying the analyst might have malignant intentions and could torture the data to tell a particular story. The Wikipedia "Data manipulation language" page has no such negative connotations (and describes the Structured Query Language [SQL] as one such language). This dual meaning stems from the definition (from Merriam-Webster) of manipulate:

- To manage or utilize skillfully
- To control or play upon by artful, unfair, or insidious means especially to one's own advantage

"Data management" was the most common term, with more than 33,000,000 results on Google. The DAMA Data Management Body of Knowledge (DAMA-DMBOK, http://www.dama.org/files/public/DI_DAMA_DMBOK_Guide_Presentation_2007.pdf) provides a definition: "Data management is the development, execution and supervision of plans, policies, programs and practices that control, protect, deliver and enhance the value of data and information assets." While somewhat clinical (and decidedly non-sexy), this may be the most appropriate phrase to describe the type of data-related skills students need to make sense of the information around them.

SIDEBAR: Making bigger datasets accessible through databases

Nolan and Temple Lang (2010) stress the importance of knowledge of information technologies, along with the ability to work with large datasets. Relational databases, first popularized in the 1970's, provide fast and efficient access to terabyte-sized files. These systems using a structured query language (SQL) to specify data operations. Surveys of graduates from statistics programs have noted that familiarity with databases and SQL would have been helpful as they moved to the workforce.

Database systems have been highly optimized and tuned since they were first invented. Connections between general purpose statistics packages such as R and database systems can be facilitated through use of SQL. Table 2 describes key operators for data manipulation in SQL.

```
verb          meaning
-------------------------------------------
SELECT      create a new result set from a table
FROM        specify table
WHERE       subset observations
GROUP       aggregate
ORDER       re-order the observations
DISTINCT    remove duplicate values
JOIN        merge two data objects
```

Table 2: Key operators to support data management and manipulation in SQL (structured query language)

Use of a SQL interface to large datasets is attractive as it allow the exploration of datasets that would be impractical to analyze using general purpose statistical packages. In this application, much of the heavy lifting and data manipulation is done within the database system, with the results made available within the general purpose statistics package.

The ASA Data Expo 2009 website (http://stat-computing.org/dataexpo/2009) provides full details regarding how to download the Expo data (1.6 gigabytes compressed, 12 gigabytes uncompressed through 2008), set up a database using SQLite (http://www.sqlite.org), add indexing, and then access it from within R or RStudio. This is very straightforward to undertake, though there are some limitations to the capabilities of SQLite.

MySQL (http://www.mysql.com, described as the world's most popular open source database) and PostgreSQL are more fully-featured systems (albeit with somewhat more complex installation and configuration).

The use of SQL within R (or other systems) is straightforward once the database has been created (either locally or remotely). An add-on package (such as `RMySQL` or `RSQLite`) must be installed and loaded, then a connection made to a local or remote database. In combination with tools such as R Markdown (which make it easy to provide a template and support code) students can start to tackle more interesting and meatier questions using larger databases set up by their instructors. Instructors wanting to integrate databases into their repertoire may prefer to start with SQLite, then graduate to more sophisticated systems (which can be accessed remotely) using MySQL.

The `dplyr` package encapsulates and replaces the SQL interface for either system. It also features *lazy* evaluation, where operations are not undertaken until absolutely necessary.

Another option is for the user to specify SQL SELECT commands. For example, the following code would replicate the creation of the dataset of counts for the three airports used to create Figure 2 using SQL (as opposed to using the interface within `dplyr`).

```
airportflights <-
  dbGetQuery(con, "SELECT Dest, Year, Month, DayOfMonth, DayOfWeek, sum(1) as numFlights
    FROM ontime WHERE (Dest = 'ALB' OR Dest = 'BDL' OR Dest = 'BTV')
    GROUP BY Year, Month, DayOfMonth, Dest")
```

In this example, a set of variables are selected (along with a derived variable which sums the number of flights) from the three airports of interest. The results are aggregated by day and destination (at which point there are more manageable). The `dbGetQuery` function in the `RMySQL` package returns a dataframe containing the results from the SQL SELECT call. The SQL syntax is similar, but not identical, to the `dplyr` syntax.

Is setting up a database too much effort? We think not. But those willing to explore can undertake similar analyses using the `nycflights13` package on CRAN. This includes five dataframes that can be accessed within R.

```
require(nycflights13)
airlines
```

```
## Source: local data frame [16 x 2]
##
##    carrier                        name
## 1       9E           Endeavor Air Inc.
## 2       AA      American Airlines Inc.
## 3       AS        Alaska Airlines Inc.
## 4       B6             JetBlue Airways
## 5       DL        Delta Air Lines Inc.
## 6       EV    ExpressJet Airlines Inc.
## 7       F9      Frontier Airlines Inc.
## 8       FL AirTran Airways Corporation
## 9       HA      Hawaiian Airlines Inc.
## 10      MQ                   Envoy Air
## 11      OO       SkyWest Airlines Inc.
## 12      UA       United Air Lines Inc.
## 13      US             US Airways Inc.
## 14      VX              Virgin America
## 15      WN      Southwest Airlines Co.
## 16      YV          Mesa Airlines Inc.
```

```
airports
```

```
## Source: local data frame [1,397 x 7]
##
##    faa                            name      lat        lon  alt tz dst
## 1  04G              Lansdowne Airport 41.13047  -80.61958 1044 -5   A
## 2  06A  Moton Field Municipal Airport 32.46057  -85.68003  264 -5   A
## 3  06C            Schaumburg Regional 41.98934  -88.10124  801 -6   A
## 4  06N                Randall Airport 41.43191  -74.39156  523 -5   A
## 5  09J          Jekyll Island Airport 31.07447  -81.42778   11 -4   A
## 6  0A9 Elizabethton Municipal Airport 36.37122  -82.17342 1593 -4   A
## 7  0G6        Williams County Airport 41.46731  -84.50678  730 -5   A
## 8  0G7  Finger Lakes Regional Airport 42.88356  -76.78123  492 -5   A
## 9  0P2   Shoestring Aviation Airfield 39.79482  -76.64719 1000 -5   U
## 10 0S9          Jefferson County Intl 48.05381 -122.81064  108 -8   A
## .. ...                            ...      ...        ...  ... .. ...
```

```
planes
```

```
## Source: local data frame [3,322 x 9]
##
##    tailnum year                type     manufacturer     model engines
## 1   N10156 2004 Fixed wing multi engine          EMBRAER EMB-145XR       2
## 2   N102UW 1998 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 3   N103US 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 4   N104UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 5   N10575 2002 Fixed wing multi engine          EMBRAER EMB-145LR       2
## 6   N105UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 7   N107US 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 8   N108UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 9   N109UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 10  N110UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## ..     ...  ...                   ...              ...       ...     ...
## Variables not shown: seats (int), speed (int), engine (chr)
```

```
flights
```

```
## Source: local data frame [336,776 x 16]
##
##    year month day dep_time dep_delay arr_time arr_delay carrier tailnum
## 1  2013     1   1      517         2      830        11      UA  N14228
## 2  2013     1   1      533         4      850        20      UA  N24211
## 3  2013     1   1      542         2      923        33      AA  N619AA
## 4  2013     1   1      544        -1     1004       -18      B6  N804JB
## 5  2013     1   1      554        -6      812       -25      DL  N668DN
## 6  2013     1   1      554        -4      740        12      UA  N39463
## 7  2013     1   1      555        -5      913        19      B6  N516JB
## 8  2013     1   1      557        -3      709       -14      EV  N829AS
## 9  2013     1   1      557        -3      838        -8      B6  N593JB
## 10 2013     1   1      558        -2      753         8      AA  N3ALAA
## ..  ...   ... ...      ...       ...      ...       ...     ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time
##   (dbl), distance (dbl), hour (dbl), minute (dbl)
```

```
weather
```

```
## Source: local data frame [8,719 x 14]
## Groups: month, day, hour
##
##    origin year month day hour  temp  dewp humid wind_dir wind_speed
## 1     EWR 2013     1   1    0 37.04 21.92 53.97      230   10.35702
## 2     EWR 2013     1   1    1 37.04 21.92 53.97      230   13.80936
## 3     EWR 2013     1   1    2 37.94 21.92 52.09      230   12.65858
## 4     EWR 2013     1   1    3 37.94 23.00 54.51      230   13.80936
## 5     EWR 2013     1   1    4 37.94 24.08 57.04      240   14.96014
## 6     EWR 2013     1   1    6 39.02 26.06 59.37      270   10.35702
## 7     EWR 2013     1   1    7 39.02 26.96 61.63      250    8.05546
## 8     EWR 2013     1   1    8 39.02 28.04 64.43      240   11.50780
## 9     EWR 2013     1   1    9 39.92 28.04 62.21      250   12.65858
## 10    EWR 2013     1   1   10 39.02 28.04 64.43      260   12.65858
## ..    ...  ...   ... ...  ...   ...   ...   ...      ...        ...
## Variables not shown: wind_gust (dbl), precip (dbl), pressure (dbl), visib
##   (dbl)
```

```
end <- Sys.time(); end
```

```
## [1] "2015-03-25 14:37:02 EDT"
```

```
end - now  # elapsed
```

```
## Time difference of 27.54015 secs
```