4-3-2019

# A Grammar for Reproducible and Painless Extract-Transform-Load Operations on Medium Data

Benjamin S. Baumer
*Smith College*, bbaumer@smith.edu

# A Grammar for Reproducible and Painless Extract-Transform-Load Operations on Medium Data

Benjamin S. Baumer*
Program in Statistical and Data Sciences, Smith College

May 24, 2018

**Abstract**

Many interesting data sets available on the Internet are of a *medium* size—too big to fit into a personal computer's memory, but not so large that they won't fit comfortably on its hard disk. In the coming years, data sets of this magnitude will inform vital research in a wide array of application domains. However, due to a variety of constraints they are cumbersome to ingest, wrangle, analyze, and share in a reproducible fashion. These obstructions hamper thorough peer-review and thus disrupt the forward progress of science. We propose a predictable and pipeable framework for R (the state-of-the-art statistical computing environment) that leverages SQL (the venerable database architecture and query language) to make reproducible research on medium data a painless reality.

*Keywords:* statistical computing, reproducibility, databases, data wrangling

*The author gratefully acknowledges the editor, associate editor, and two anonymous reviewers for helpful comments, as well as Carson Sievert, Nicholas Horton, Weijia Zhang, Wencong Li, Rose Gueth, Trang Le, and Eva Gjekmarkaj for their contributions to this work. Email: bbaumer@smith.edu

# 1 Introduction

## 1.1 Motivation

Scientific research is increasingly driven by "large" data sets. However, the definition of "large" is relative. We define *medium* data to be those who are too big to store in memory on a personal computer, but not so big that they won't fit on a hard drive. Typically, this means data on the order of several gigabytes (see Table 1). Publicly accessible medium data sets (PAMDAS) are now available in a variety of application domains. A few examples are the Citi Bike bike-sharing program in New York City, campaign contributions from the Federal Election Commission, and on-time airline records from the Bureau of Transportation Statistics.

While PAMDAS provide *access*, they do not remove all barriers to reproducible research on these data. Because of their size, reading these data into memory will either take an unbearably long time, exhaust the computer's memory until it grinds to a halt, or simply not work at all. A sensible solution is to download the data to a local storage device and then import it into a relational database management system (RDBMS). RDBMS's have been around since the 1970s, and provide a scalable solution for data of this magnitude. High-quality, open source implementations (e.g., MySQL, PostgreSQL, SQLite) are prevalent. However, creating a new database from scratch is time-consuming and requires knowledge of database administration. While these skills are not difficult to acquire, they are not always emphasized in the traditional undergraduate curriculum in either statistics (American Statistical Association Undergraduate Guidelines Workgroup

| "Size" | actual size | hardware | software |
|---|---|---|---|
| small | < several GB | RAM | R |
| medium | several GB – a few TB | hard disk | SQL |
| big | many TB or more | computing cluster | Spark? |

Table 1: Relative sizes of data from the point-of-view of personal computer users. We focus on medium data, which are too large to fit comfortably into the memory of a typical personal computer, but not so large that they won't fit comfortably on the hard drive of such a computer. In 2018, desktop computers typically ship with hard drives of at most four terabytes. Most laptops use solid-state hard drives which hold less than one terabyte.

2014) or computer science (The Joint Task Force on Computing Curricula 2013).

The process of downloading the raw data from its authoritative source and importing it into a RDBMS is often called *Extract-Transform-Load* (ETL). Professionals who work with data spend a disproportionate amount of their time on such tasks. Their solutions are sometimes idiosyncratic, platform- or architecture-specific, poorly documented, unshared, and involve custom scripts written in various (and often multiple) languages. Thus, the ETL process constitutes a barrier to reproducible research, because there is not a good way of verifying that two people downloading data from the same source will end up with the *exact* same set of data in their local data store. This matters because any subsequent data analysis could be sensitive to small perturbations in the underlying data set. Like Claerbout (1994) and Donoho (2010), we recognize the necessity of data-based research being backed by open data, with well-documented data analysis code that is shared publicly and executable on open-source platforms.

Sharing the local data store is often also problematic, due to licensing restrictions and the sheer size of the data. While PAMDAS may be free to download, there may be legal barriers to publicly sharing a local data store that is essentially a reproduction of those original data (see, for example Greenhouse (2008)). File size limitations imposed by software repositories (e.g., GitHub, CRAN) may make distribution via those channels unfeasible. Moreover, sharing medium data sets through the cloud may be expensive or unrealistic for many individuals and small companies.

## 1.2 Our contribution

We propose a software framework for simultaneously solving two related but distinct problems when analyzing PAMDAS: 1) how to build a relational database with little or no knowledge of SQL, and; 2) how to ensure reproducibility in published research succinctly. Our solution consists of a package for R (R Core Team 2018) that provides a *core* framework for ETL operations along with a series of *peripheral* packages that extend the ETL framework for a specific PAMDAS. The core `etl` package is available on CRAN (Baumer 2016). Seven different peripheral packages are in various states of development, but in principle there is no limit to how many could be created. These packages are all fully

open source (hosted on GitHub with Creative Commons licenses), cross-platform (to the extent allowed by R, MySQL, PostgreSQL, SQLite, etc.), and fit into the state-of-the-art `tidyverse` (Wickham 2017*b*) paradigm popular among R users. Specifically, the `etl` package extends the functionality of existing database tools in R (see Section 5.1) by maintaining local storage locations and employing a consistent grammar for ETL operations (see Section 3).

The `etl` suite of packages will make it easier to bring PAMDAS to data analysts of all stripes while lowering barriers to entry and enhancing transparency, usability, and reproducibility. For the most part, knowledge of SQL will not be required to access these data through R. (See Kline et al. (2005) for a primer on SQL.)

In Section 2, we provide motivating examples that illustrate how the use of the `etl` framework can facilitate the construction of medium databases for PAMDAS, and how that ability can improve reproducibility in published research. We explicate the grammar employed by `etl`—and how it speeds adoption—in Section 3. In Section 4, we describe how a typical R user can use the `etl` package and its dependent packages to build medium databases with relative ease. In Section 5, we briefly outline how an R developer can rapidly create their own `etl`-dependent packages. We conclude with a brief discussion in Section 6. In our supplementary materials, Section A situates our work in the existing ecosystem of R tools, Section B provides a short example of how to use `etl`, Section C discusses performance benchmarks, and Section D illustrates how cloud computing services can be used in conjunction with the `etl` package.

# 2 Motivating examples

## 2.1 ETL workflow for on-time airline data

The `etl` package provides the foundation for `etl`-dependent packages that focus on specific data sets. In this example, we illustrate how one of these packages—`airlines`—can be used to build a medium database of flight information. These data are available from the Bureau of Transportation Statistics via monthly ZIP files.

First, we load the `airlines` package. We then use the `src_mysql_cnf()` function pro-

vided by etl to create a database connection to a preconfigured remote MySQL [1] database server [2]. (The database to which we connect is also called "airlines".)

```
library(airlines)
db <- src_mysql_cnf("airlines", groups = "scidb")
```

Next, we instantiate an object called ontime. The source of ontime's data is the R package called "airlines". In this case, we specify the db argument to be the connection to our MySQL database, and the dir argument for local storage. Any files we download or transform will be stored in dir. Among other things, ontime is a src_dbi object—an interface to a database—meaning that it can take advantage of the many functions—mainly provided by the dbplyr (Wickham 2017*a*) package—that work on such objects. We postpone a more detailed discussion of this until Section 3.2.

```
ontime <- etl("airlines", db = db, dir = "~/dumps/airlines")
```

We then perform our ETL operations. We first initialize the database with etl_init(), which in this case loads table schemas from an SQL script provided by the airlines package. Next, the etl_extract() function downloads data from 1987–2016. This results in one ZIP file for each month being stored in a subdirectory of dir. Next, we use the etl_transform() function to unzip these files and grab the relevant CSVs [3]. While the etl_transform() function takes the same arguments as etl_extract(), those arguments needn't take the same values. For purposes of illustration we choose to transform only the data from the decade of the 1990s. Finally, the etl_load() function reads the CSV data from 1996 and 1997 into the database, but only from the first half of the year, plus September.

[1] MySQL is a popular open-source relational database management system. The src_mysql_cnf() function reads server information and credentials from a configuration file stored in the user's home directory.

[2] The command library(airlines) makes user-facing functions from both the airlines and etl packages available.

[3] CSV stands for Comma-Separated Values, and is a common data format.

```
ontime %>%
  etl_init() %>%
  etl_extract(years = 1987:2016) %>%
  etl_transform(years = 1990:1999) %>%
  etl_load(years = 1996:1997, months = c(1:6, 9))
```

We note that this process—which may take several hours—results in a relational database with multiple tables occupying several dozen gigabytes on disk, and comprising several million rows of data (the full data set across all years contains more than 160 million rows).

```
ontime

## dir:  728 files occupying 26.255 GB
## src:  mysql 5.5.58-0ubuntu0.14.04.1-log [bbaumer@scidb.smith.edu:/airlines]
## tbls: airports, carriers, flights, planes, summary, weather
```

Moreover, `ontime` is constructed such that we can use existing functionality from the `dplyr` package (Wickham & Francois 2016) to access the data from a specific airport, say, Bradley International (BDL), which serves Hartford, CT and Springfield, MA. Please see Section 3.2 for more technical details.

```
ontime %>%
  tbl("flights") %>%
  filter(year == 1996, dest == "BDL") %>%
  head(3)

## # Source:   lazy query [?? x 21]
## # Database: mysql 5.5.58-0ubuntu0.14.04.1-log
## #   [bbaumer@scidb.smith.edu:/airlines]
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <int>    <int>
## 1  1996    10     1      637            640        -3      940
```

```
## 2  1996    10     1      652            653        -1     1106
## 3  1996    10     1      707            710        -3      836
## # ... with 14 more variables: sched_arr_time <int>, arr_delay <int>,
## #   carrier <chr>, tailnum <chr>, flight <int>, origin <chr>, dest <chr>,
## #   air_time <int>, distance <int>, cancelled <int>, diverted <int>,
## #   hour <int>, minute <int>, time_hour <chr>
```

Thus, with just a few simple lines of R code—and no knowledge of SQL—the `airlines` package allows us to create a medium-sized relational database suitable for analysis with popular `dplyr` tools.

## 2.2 Reproducible research with Citi Bike data

The following example using the `citibike` package illustrates how reproducibility of published research in the natural and social sciences could be improved through use of the `etl` framework.

The lack of reproducibility in published scientific research in the natural and social sciences is problematic. Here, we revisit a series of operations research efforts analyzing load balancing for stations in the Citi Bike municipal bike sharing system in New York City (O'Mahony & Shmoys 2015, Singhvi et al. 2015, O'Mahony 2015) and demonstrate how the `etl` framework improves data analytic workflows. The data from this system has fueled several research efforts since its launch in July 2013.

The system's engineers face a problem balancing the load of bikes among stations. Since one cannot ensure that bikes rented from one station will be returned to that station, how can one ensure that there will always be enough bikes at a particular station to meet demand?

Singhvi et al. (2015) provided the following description of their data set:

> We obtained bike usage statistics for April, May, June and July 2014 from Citi Bike's website (`https://www.citibikenyc.com/system-data`). This dataset contains start station id, end station id, station latitude, station longitude and trip time for each bike trip. 332 bike stations have one or more originating bike

> trips. 253 of these are in Manhattan while 79 are in Brooklyn (left panel of Figure 1). We processed this raw data to get the number of bike trips between each station pair during morning rush hours.

This is a fairly specific description of how the data were acquired, since it cites a URL, a specific date range, and the exact number of stations present. However, is it sufficient information for someone else to verify that they are working with the same data set?

Using the `citibike` package, we attempt to reproduce this data set by creating a connection to a (in this case local) database, initializing it, and then populating that database with a single call to `etl_update()`:

```
library(citibike)
bikes <- etl("citibike", dir = "~/dumps/citibike/",
             db = src_mysql_cnf("citibike"))
```

```
bikes %>%
  etl_update(years = 2014, months = 4:7)
```

Leveraging `dplyr` again, the following pipeline confirms the number of unique stations.

```
trips <- bikes %>%
  tbl("trips")
trips %>%
  group_by(Start_Station_ID) %>%
  summarize(num_trips = n()) %>%
  filter(num_trips >= 1) %>%
  collect() %>%
  nrow()

## [1] 332
```

How confident are you that we now have a copy of the same data as these researchers? We have the same number of stations, but do we have the same number of rows? Do the

rows contain the same information? These questions are impossible to verify given the description above.

Behind the scenes, the authors certainly wrote code to download and process these data from the Citi Bike website. Indeed, they admit as much in the last sentence of the quotation above. Moreover, the figures in the paper were clearly produced in R. Thus, this research provides a perfect instance where the use of the `citibike` package could have standardized the exact data set upon which their research is based. The inclusion of a few short lines of code would ensure that all parties are analyzing the same data set.

In another effort, Faghih-Imani & Eluru (2016) model bike demand using spatio-temporal data from the Citi Bike system. Their description of the data is less specific than that of Singhvi et al. (2015), however they include an appendix containing some summary statistics. There is no clear way to verify the integrity of the data set. They write:

> We focused on the month of September, 2013; i.e. the peak month of the usage in 2013. Therefore, the final sample consists of 237,600 records (330 stations $\times$ 24 hours $\times$ 30 days).

Here again, a single call to `etl_update()` could have ensured that all users have the same data set:

```
etl_update(bikes, year = 2013, months = 9)
```

The number of records reported is somewhat misleading, since many stations had no trips during some hours of some day. In fact, the following pipeline returns only 167,258 records.

```
trips %>%
  filter(YEAR(Start_Time) == 2013) %>%
  group_by(Start_Station_ID, DAY(Start_Time), HOUR(Start_Time)) %>%
  summarize(N = n(),
            num_stations = COUNT(DISTINCT(Start_Station_ID)),
            num_days = COUNT(DISTINCT(DAYOFYEAR(Start_Time)))) %>%
  collect() %>%
```

```
  nrow()

## [1] 167258
```

In both cases, our attempt to verify the data used by these researchers was greatly aided by the `citibike` package. Moreover, because the `citibike` package employs a consistent grammar and fits into the popular `tidyverse`, it is far easier to use than say, a `bash` script posted on one of these researchers' website.

# 3 A grammar for ETL

While the individual steps necessary to process a PAMDAS into a RDBMS vary greatly, the three major steps of downloading the data, wrangling it, and importing it into a database are universal. The `etl` framework is designed to take advantage of this common structure. This achieves two major goals: to abstract the idiosyncratic complications of each PAMDAS away from the user, and; to restrict the developer's obligation to only those idiosyncracies.

The use of the term "grammar" in a data science context is not novel. Wilkinson et al. (2006) described a "grammar of graphics" that was implemented in R as `ggplot2` (Wickham 2009). Similarly, `dplyr` (Wickham & Francois 2016) provides a "grammar of data manipulation." A grammar consists of verbs and nouns that can be combined in logical ways to intentional effect. The benefit of having one is that once a user understands the grammar, they should be able to read and write longer sequences of code fluently. The use of the pipe operator provided by the `magrittr` package (Bache & Wickham 2014) is crucial to allowing pipelines (i.e., "sentences") to be composed from short sequences of commands (i.e., "phrases").

The design of `etl` is very much in this spirit—it is an extension of the grammar of data manipulation provided by `dplyr`. We present `etl` as a grammar for ETL operations that is rich enough to describe a great many ETL processes, but simple enough to contain only a handful of verbs. In Section 3.4, we illustrate how different ETL "sentences" can cover several common use cases. These cases are informed by our experience working with data of this magnitude in a variety of professional contexts over the past 15 years.

## 3.1 Tidyverse design

The `etl` package fits into a growing collection of R packages known as the `tidyverse` (Wickham 2017*b*). These packages are designed for interoperability and emphasize functions that are *pure*, *predictable*, and *pipeable*, as described by Hadley Wickham.

**Pure** The output of a function is entirely dependent on the input to the function. Pure functions make no changes to other objects in the environment.

**Predictable** Functions names, arguments, and behaviors are consistent, such that if you can learn how to use one function, you have a head start on understanding how to use others.

**Pipeable** Functions return objects of the same type as their first argument, so that pipeable operations can be chained together to produce *pipelines*.

Functions in the `etl` package are predictable and pipeable, but not pure. This is by design—while the predictability and pipeability make `etl` easy to use and compatible with the `tidyverse`, these functions also necessarily download files, store them locally, and interact with databases outside of R. These changes to the computing environment are unavoidable given the nature of the task.

## 3.2 ETL nouns

At the center of any `etl` pipeline is an object that is created by the `etl()` function, whose first argument is a character string naming the package that provides access to the data. The package `foo` creates objects of class `etl_foo`. If it is not installed, `etl()` will throw an error.

```
etl("nyctaxi")

## No database was specified so I created one for you at:
## /tmp/Rtmpdcqje1/file4fd51f7d8e53.sqlite3
```

```
## dir:  0 files occupying 0 GB
## src:  sqlite 3.22.0 [/tmp/Rtmpdcqje1/file4fd51f7d8e53.sqlite3]
## tbls:

etl("foo")

## Error in etl.default("foo"):  Please make sure that the 'foo' package is
installed
```

Recall that all `etl` objects are `src_dbi` objects. Thus, `print()`, `summary()`, and `is()` methods for `etl` objects extend those provided by other packages. Here, we illustrate a few of these features.

```
class(ontime)

## [1] "etl_airlines" "etl"          "src_dbi"      "src_sql"
## [5] "src"

# summary(ontime)
# output suppressed for space
src_tbls(ontime)

## [1] "airports" "carriers" "flights"  "planes"   "summary"  "weather"
```

Moreover, like all `src_dbi` objects, every `etl` object is stored as a list and maintains a `DBIConnection` to a database in `con`.

```
str(ontime)

## List of 2
##  $ con  :Formal class 'MySQLConnection' [package "RMySQL"] with 1 slot
##   .. ..@ Id: int [1:2] 0 0
##  $ disco:<environment: 0x53b1638>
##  - attr(*, "class")= chr [1:5] "etl_airlines" "etl" "src_dbi" "src_sql" ...
```

```
##  - attr(*, "pkg")= chr "airlines"
##  - attr(*, "dir")= chr "/home/bbaumer/dumps/airlines"
##  - attr(*, "raw_dir")= chr "/home/bbaumer/dumps/airlines/raw"
##  - attr(*, "load_dir")= chr "/home/bbaumer/dumps/airlines/load"
```

Accessing this `con` allows one to make use of the extensive functionality provided by the `DBI` (R Special Interest Group on Databases (R-SIG-DB) et al. 2016) package.

```
DBI::dbGetInfo(ontime$con)
DBI::dbListTables(ontime$con)
```

`etl` objects can interface with any RDBMS that can become a `src_dbi`. In particular, in addition to SQLite, MySQL/MariaDB, PostgreSQL, Google BigQuery, and MonetDB—all of which are supported through standalone R packages—`DBI` functionality can be used with a variety of other RDBMSs through the `odbc` package (Hester & Wickham 2017), which supports Amazon Redshift, Apache Hive, Apache Impala, Microsoft SQL Server, Oracle, Salesforce, and Teradata. [4] For those that lack a supportive IT infrastructure, low cost, low maintenance access to many of these technologies is available through cloud computing vendors, such as Amazon RDS (see Appendix D for a brief tutorial).

The main difference between an `etl` object and a `src_dbi` object is that an `etl` object has attributes that point toward `dir`—a directory where files can be safely read and written. If no `dir` argument is specified, a temporary directory is created and used. Within `dir`, two subdirectories are automatically created: `raw` and `load`. Raw files downloaded via `etl_extract()` are placed in `raw`. `etl_transform()` reads those files and writes the resulting transformed files to `load`. Finally, the `etl_load()` function reads files from `load` and imports them into the database.

## 3.3 ETL verbs

The workhorses of `etl` are the three main verbs. Each takes an `etl` object as its first argument and returns an `etl` object invisibly, enabling these functions to be piped.

[4] See `https://db.rstudio.com/databases/` for the most current list.

- `etl_extract()`: download data from the Internet and place the raw files in the `raw` directory. The `default` method grabs data provided by the named package.

- `etl_transform()`: read files in the `raw` directory, perform any necessary data wrangling operations, and write CSV files to the `load` directory. The `default` method copies all CSVs in the `raw` directory to the `load` directory.

- `etl_load()`: import CSV files from the `load` directory into the database. The `default` method imports all CSVs in the `load` directory into eponymous tables.

Writing these three functions becomes each `etl`-dependent package maintainer's responsibility. We discuss this in greater detail in Section 5.

While these three main verbs may be the most universal, two other commonly-used verbs are `etl_init()` and `etl_cleanup()`.

- `etl_init()`: initializes the database by either running a SQL initialization script or by simply deleting all of the tables in the existing database. That script can be bundled by the package maintainer or passed as a file path or character vector. It can also be written in generic SQL or in a flavor of SQL specific to a particular database engine. This enables R users to make use of features that exist in one database implementation but not another (e.g., partitions in MySQL which are not available in SQLite). This step is optional, since `DBI::dbWriteTable()` will perform column type interpolation during the `etl_load()` phase if the corresponding tables don't already exist.

- `etl_cleanup()`: delete files from either the `raw` or `load` directories using regular expression pattern matching.

For convenience, two additional verbs are provided:

- `etl_update()`: chains the extract, transform, and load phases together, passing the same arguments to each.

- `etl_create()`: runs the full chain including initialization, update, and cleanup.

```
getS3method("etl_update", "default")

## function(obj, ...) {
##   obj <- obj %>%
##     etl_extract(...) %>%
##     etl_transform(...) %>%
##     etl_load(...)
##   invisible(obj)
## }
## <environment: namespace:etl>

getS3method("etl_create", "default")

## function(obj, ...) {
##   obj <- obj %>%
##     etl_init(...) %>%
##     etl_update(...) %>%
##     etl_cleanup(...)
##   invisible(obj)
## }
## <environment: namespace:etl>
```

## 3.4 Common use cases

In Section 2.2, we showed how a single call to `etl_update()` could be used to populate a static database with data specific to a time interval. While this one-shot usage may be the most common, the `etl` grammar is flexible enough to accommodate other use cases.

**Regular updates for updated data** You receive a daily dump of customer data from a vendor in files that are overwritten. Run a script each day that contains `etl_create()` to rebuild your database.

**Regular updates for new data** Your web logs are archived into a new file each month. Run `etl_init()` once, then run `etl_update()` monthly when new data becomes available.

**Asynchronous updates** A temporary network disruption may result in corrupt downloads or a broken pipeline. The `etl` design makes it relatively easy to, say, use `etl_cleanup()` to delete a single corrupted month of airline data, then re-run `etl_update()` on just that month. Since the data are stored in a database, the order of the rows is generally irrelevant. Some care is necessary to avoid duplicate rows, however.

**Reconfigure and reload** An update to `airlines` adds a partitioning scheme to the `flights` table. You update your database by running `update.packages()`, followed by `etl_init()` and `etl_load()`. You do not need to download or transform the data again.

**Porting a database** You create a local copy of a database, verify its contents, and then port it to a remote server by defining a new database connection, and then calling `etl_load()` on the new `etl` object.

# 4 The `etl` package for R users

The `etl` framework is designed to make PAMDAS accessible to R users who may not have experience with SQL. The following `etl`-dependent packages—which are in various stages of development—can be used in a manner similar to the `airlines` and `citibike` packages illustrated in Section 2, since they all employ the grammar described in Section 3. For further examples, please see Appendix B and the "Using `etl`" vignette [5]. These packages—combined with the ability to convert data in any R package to a relational database as described in Section 4.2—lower barriers of entry to medium data for even novice R users.

[5] `https://cran.r-project.org/web/packages/etl/vignettes/using_etl.html`

## 4.1 PAMDAS accessible via `etl`

The following `etl`-dependent packages are available on GitHub (and CRAN where indicated):

**macleish** (CRAN) weather and spatial data from the Smith College MacLeish Field Station in Whately, MA (Baumer et al. 2017)

**airlines** on-time flight data from the Bureau of Transportation Services for all domestic flights since October 1987 (Baumer 2017*a*)

**imdb** a mirror of the Internet Movie Database (Baumer 2017*b*)

**nyc311** calls to New York City's non-emergency municipal services hotline (Baumer & Li 2017)

**fec** campaign finance contributions and spending from the Federal Election Commission (Baumer & Gjekmarkaj 2017)

**citibike** trip data for New York City's municipal bike sharing service (Zhang 2017)

**nyctaxi** (CRAN) trip data from the New York City Taxi and Limousine Commission (Li 2018)

In Section 5, we explain how these packages can be developed rapidly using the `etl` framework. In some cases, these packages can be reduced to a few lines of R code.

## 4.2 ETL for small data bundled in R packages

The `etl` package can also perform default ETL operations on data stored in any R package. Here, we build a database of five tables included in the `nasaweather` package (Wickham 2014).

```
nasa <- etl("nasaweather") %>%
  etl_update()
```

```
## No database was specified so I created one for you at:
## /tmp/Rtmpdcqje1/file4fd559e57b2.sqlite3
## Loading 5 file(s) into the database...

nasa

## dir:  10 files occupying 0.008 GB
## src:  sqlite 3.22.0 [/tmp/Rtmpdcqje1/file4fd559e57b2.sqlite3]
## tbls: atmos, borders, elev, glaciers, storms
```

This functionality is a convenience, since data bundled in R packages are usually small, but it nevertheless allows R users to create relational databases with minimal effort. We note here that since no pre-existing database connection was specified, a SQLite database was created in a temporary directory.

# 5 The `etl` package for R developers

## 5.1 Database functionality in R

Recent advances in R computing have made accessing databases through R a relatively painless process.

In R, a `data.frame` is a two-dimensional array of data that consists of rows and columns. It is logically analogous to a *table* in SQL parlance, but with two crucial differences in implementation: first, a `data.frame` is stored in memory, whereas a table is usually written to disk; second, a `data.frame` need not and cannot be indexed, whereas tables are often indexed. The `tibble` package in R extends the `data.frame` to the more flexible `tbl` data structure (Müller & Wickham 2017). The `dbplyr` package further extends the functionality of `tbl`'s to be backed by a local or remote database (Wickham 2017*a*). A common interface to such databases is provided by the `DBI` package (R Special Interest Group on Databases (R-SIG-DB) et al. 2016). Each RDBMS has its own R package that implements the `DBI` programming interface. For example, the `RMySQL` package implements the `DBI` specification for MySQL (Ooms et al. 2017), while the `RSQLite` package implements the `DBI` specification

for SQLite (Müller et al. 2017). Through this chain of interfaces, a `tbl_mysql` appears to an R user to be a familiar `data.frame`, but in fact, it is akin to a `VIEW` of the underlying MySQL table, and thus occupies virtually no space in R's memory, and can make use of SQL indexes.

This infrastructure provides a backdrop for the popular data wrangling package `dplyr` (Wickham & Francois 2016), which re-imagines SQL `SELECT` syntax as a pipeable sequence of data verbs. This approach is attractive because R users can perform SQL-style operations from within R without having to learn SQL. Furthermore, if the `dbplyr` functionality is employed, R users can offload the execution of these operations to more powerful RDBMS's.

## 5.2 Extending `etl`

The `etl` package provides tools to speed the development of `etl`-dependent packages. The `create_etl_package()` function creates a new R package by calling `devtools::create()` (Wickham & Chang 2017), while also adding `etl` to the `Depends` section of the `DESCRIPTION` file, and providing the code template shown below, with `foo` replaced by `newpkg`.

```
proj_dir <- file.path(tempdir(), "newpkg")
create_etl_package(proj_dir)

## Creating package 'newpkg' in '/tmp/Rtmpdcqje1'
## No DESCRIPTION found.  Creating with values:
## * Creating 'newpkg.Rproj' from template.
## * Adding '.Rproj.user', '.Rhistory', '.RData' to ./.gitignore
## * Creating R/etl.R template source file...
## * Adding etl to Depends
## Next:
## Are you sure you want Depends?  Imports is almost always the better choice.
```

A developer can then immediately compile a functioning R package, which in this case downloads Houston public school district data. The `default` method for `etl_extract()` pulls data provided by the package, which in this case is pointless because there is no

such data. Conversely, the `etl_foo` method provided in the template illustrates how—in a simple case—a vector of URLs and a call to `smart_download()` is sufficient to complete the extract phase.

```
## #' My ETL functions
## #' @import etl
## #' @inheritParams etl::etl_extract
## #' @export
## #' @examples
## #' \dontrun{
## #' if (require(dplyr)) {
## #'   obj <- etl("foo") %>%
## #'     etl_create()
## #' }
## #' }
##
## etl_extract.etl_foo <- function(obj, ...) {
##   # Specify the URLs that you want to download
##   src <- c("http://www.stat.tamu.edu/~sheather/book/docs/datasets/HoustonChronicle.cs
##
##   # Use the smart_download() function for convenience
##   etl::smart_download(obj, src, ...)
##
##   # Always return obj invisibly to ensure pipeability!
##   invisible(obj)
## }
```

Since the raw data is already in a CSV format, the `default` methods for `etl_transform()` and `etl_load()` are sufficient to complete the ETL cycle for this simple example, so there is no need to write `etl_foo` methods for these functions. After changing to the root directory of the new package, one can install, load, and use `newpkg` just like any other.

```
devtools::install()
library(newpkg)
districts <- etl("newpkg") %>%
  etl_create()
districts %>%
  tbl("HoustonChronicle")
```

This functionality leverages `devtools` to allow intermediate R users with no package development experience (e.g., advanced undergraduate statistics and data science majors) to begin creating `etl`-dependent R packages, such as those listed in Section 4.1. For more information, please see the "Extending `etl`" vignette [6].

## 5.3 Additional functionality for developers

The `etl` package contains several additional functions that are useful for developers. Some of these may eventually be passed upstream to `DBI`. Briefly,

- `dbRunScript()`: execute a sequence of arbitrary SQL commands. This takes a full SQL script and passes the individual SQL statements to `DBI::dbExecute()`.
- `dbWipe()`: delete all of the tables in a database
- `match_files_by_year_months()`, `extract_date_from_filename()`, and `valid_year_month()` assist with working with dates—specifically in conjunction with files that may encode dates in their names (e.g., `201307-citibike-tripdata.zip`)
- `smart_download()` and `smart_upload()`: only download and upload files that don't already exist
- `src_mysql_cnf()` use the `~/.my.cnf` configuration file to connect to MySQL

[6] https://cran.r-project.org/web/packages/etl/vignettes/extending_etl.html

# 6 Conclusion

## 6.1 Future Work

The `etl` package does not solve all problems for those working with medium data. There is considerable room for improving the performance of the `etl` package itself. First, some of the ETL operations should be parallelizable. In particular, `etl_transform()` is a good candidate, since it is always working locally. While in some cases the bottleneck for `etl_extract()` will be the speed of the user's Internet connection, in others parallel threads—such as those provided by the `parallel` package—could significantly improve performance. For `etl_load()`, the database engine may not support simultaneous imports to the same table. Second, reading and writing data files to the disk is time-consuming. It is possible that new file formats such as `feather` could reduce latency (Wickham 2016*a*). Third, the data ends up being stored on disk three times: once in its raw format (hopefully compressed), once as a CSV (uncompressed), and once in the database's native file format (optimized). Importing the compressed files directly into the database may be possible in some cases, but care must be taken to ensure the predictability of these functions. Using symbolic links rather than copying files might also be appropriate in some cases. One can of course use `etl_cleanup()` to delete either or both of the first two instances, but perhaps a more streamlined process is possible, at least in some cases.

The `etl` package fuels rapid development of dependent packages, even among novice R developers. We know this because many of the `etl`-dependent packages referenced above were partially developed by undergraduate students. A broad adoption of these `etl`-dependent packages and a larger installed user base would increase interest in the project and lead to a more robust infrastructure. We plan to continue this work in the future.

## 6.2 Discussion

As data grow larger and larger, more and more people will need to develop the skills necessary to work with them. Yet there is limited room in the undergraduate curriculum for such training. Moreover, exposing students to truly big data requires expensive technical

infrastructure, training, and support that will remain burdensome to many faculty members for the foreseeable future. A more realistic approach towards helping students develop their capacity to work with larger data sets is to focus on medium data (rather than big data). These data are still challenging and will still help students develop their understanding of scalability issues, while at the same time having a much lower barrier to entry for both students and faculty.

At the same time, producing reproducible research on medium data is more difficult than it is on small data—and many researchers already have a hard time with that. As medium and big data become more prevalent in published research, we must not soften our insistence on reproducibility.

Among educators, interest in exposing statistics students to larger and more complex data is growing. Recent guidelines about undergraduate majors in statistics (American Statistical Association Undergraduate Guidelines Workgroup 2014) and data science (De Veaux et al. 2017) endorsed by the American Statistical Association emphasize the necessity of exposing students to such data. Horton et al. (2015) advocate for discussing medium data as a "precursor" to big data. However, all of the aforementioned challenges to working with medium data present barriers to statistics educators who are quite comfortable with R, but may not have sufficient experience with SQL.

We propose this `etl` package as a mechanism for facilitating reproducible research on medium data for R users. This has the dual benefit of lowering barriers to entry (minimal SQL required) for larger and more complex data sets, while simultaneously aiding the reproducibility of any subsequent research. Not everyone needs to be a data engineer, but many need to wrangle medium data—`etl` provides a powerful but simplified interface for the latter.

# References

Almquist, Z. W. (2010), 'US Census spatial and demographic data in R: The UScensus2000 suite of packages', *Journal of Statistical Software* **37**(6), 1–31.
**URL:** *http://www.jstatsoft.org/v37/i06/*

American Statistical Association Undergraduate Guidelines Workgroup (2014), *2014 Curriculum Guidelines for Undergraduate Programs in Statistical Science*, American Statistical Association.
**URL:** *http://www.amstat.org/education/curriculumguidelines.cfm*

Anderson, G. B. & Eddelbuettel, D. (2017), 'Hosting Data Packages via drat: A Case Study with Hurricane Exposure Data', *The R Journal* **9**(1), 486–497.
**URL:** *https://journal.r-project.org/archive/2017/RJ-2017-026/index.html*

Bache, S. M. & Wickham, H. (2014), *magrittr: A Forward-Pipe Operator for R*. R package version 1.5.
**URL:** *https://CRAN.R-project.org/package=magrittr*

Ball, R. & Medeiros, N. (2012), 'Teaching integrity in empirical research: A protocol for documenting data management and analysis', *The Journal of Economic Education* **43**(2), 182–189.
**URL:** *http://dx.doi.org/10.1080/00220485.2012.659647*

Baumer, B., Çetinkaya 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* **8**(1).
**URL:** *http://escholarship.org/uc/item/90b2f5xh*

Baumer, B. S. (2016), *etl: Extract-Transform-Load Framework for Medium Data*. R package version 0.3.3.
**URL:** *http://github.com/beanumber/etl*

Baumer, B. S. (2017*a*), *airlines: Historical On-time Flight Data*. R package version 0.2.2.9011.
**URL:** *http://github.com/beanumber/airlines*

Baumer, B. S. (2017*b*), *imdb: Populate a database with data from the IMDB*. R package version 0.0.2.9004.
**URL:** *https://github.com/beanumber/imdb*

Baumer, B. S. & Gjekmarkaj, E. (2017), *fec: Campaign finance for Federal Elections*. R package version 0.0.0.9010.
**URL:** *http://github.com/beanumber/fec*

Baumer, B. S., Goueth, R., Li, W., Zhang, W. & Horton, N. J. (2017), *macleish: Retrieve Data from MacLeish Field Station*. R package version 0.3.1.
**URL:** *http://github.com/beanumber/macleish*

Baumer, B. S. & Li, W. (2017), *nyc311: Access the NYC 311 Data*. R package version 0.0.1.9002.
**URL:** *http://github.com/beanumber/nyc311*

Boettiger, C. (2015), 'An introduction to docker for reproducible research', *ACM SIGOPS Operating Systems Review* **49**(1), 71–79.

Boettiger, C., Chamberlain, S., Hart, E. & Ram, K. (2015), 'Building software, building community: lessons from the rOpenSci project', *Journal of Open Research Software* **3**(1).
**URL:** *https://openresearchsoftware.metajnl.com/articles/10.5334/jors.bu/*

Boettiger, C. & Eddelbuettel, D. (2017), 'An introduction to rocker: Docker containers for r', *arXiv preprint arXiv:1710.03675* .
**URL:** *https://arxiv.org/pdf/1710.03675*

Çetinkaya-Rundel, M. & Rundel, C. (2017), 'Infrastructure and tools for teaching computing throughout the statistical curriculum', *The American Statistician* (just accepted).
**URL:** *https://peerj.com/preprints/3181.pdf*

Claerbout, J. (1994), Hypertext documents about reproducible research, Technical report, Stanford University.
**URL:** *http://sepwww.stanford.edu/sep/jon/nrc.html*

De Veaux, R. D., Agarwal, M., Averett, M., Baumer, B. S., Bray, A., Bressoud, T. C., Bryant, L., Cheng, L. Z., Francis, A., Gould, R., Kim, A. Y., Kretchmar, M., Lu, Q., Moskol, A., Nolan, D., Pelayo, R., Raleigh, S., Sethi, R. J., Sondjaja, M., Tiruviluamala, N., Uhlig, P. X., Washington, T. M., Wesley, C. L., White, D. & Ye, P. (2017),

'Curriculum guidelines for undergraduate programs in data science', *Annual Review of Statistics and Its Application* **4**(1), 1–16.
**URL:** *http://www.annualreviews.org/doi/abs/10.1146/annurev-statistics-060116-053930*

Donoho, D. L. (2010), 'An invitation to reproducible computational research', *Biostatistics* **11**(3), 385–388.
**URL:** *https://academic.oup.com/biostatistics/article/11/3/385/257703*

Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M. & Stodden, V. (2009), 'Reproducible research in computational harmonic analysis', *Computing in Science & Engineering* **11**(1).

Eckel, S. P. & Peng, R. D. (2009), 'Interacting with local and remote data repositories using the stashr package', *Computational Statistics* **24**(2), 247–254.
**URL:** *https://link.springer.com/content/pdf/10.1007/s00180-008-0124-x.pdf*

Eddelbuettel, D. & François, R. (2011), 'Rcpp: Seamless R and C++ integration', *Journal of Statistical Software* **40**(8), 1–18.
**URL:** *http://www.jstatsoft.org/v40/i08/*

Eglen, S. J., Marwick, B., Halchenko, Y. O., Hanke, M., Sufi, S., Gleeson, P., Silver, R. A., Davison, A. P., Lanyon, L., Abrams, M. et al. (2017), 'Toward standard practices for sharing computer code and programs in neuroscience', *Nature Neuroscience* **20**(6), 770–773.
**URL:** *https://www.biorxiv.org/content/early/2017/02/28/045104*

Faghih-Imani, A. & Eluru, N. (2016), 'Incorporating the impact of spatio-temporal interactions on bicycle sharing system demand: A case study of New York CitiBike system', *Journal of Transport Geography* **54**, 218–227.
**URL:** *http://www.sciencedirect.com/science/article/pii/S0966692316303143*

Fuentes, M. (2016), 'Reproducible research in JASA', *AMSTAT News* .
**URL:** *http://magazine.amstat.org/blog/2016/07/01/jasa-reproducible16/*

Greenhouse, L. (2008), 'No ruling means no change for fantasy baseball leagues'.
**URL:** *http://www.nytimes.com/2008/06/03/sports/baseball/03fantasy.html*

Hardin, J., Hoerl, R., Horton, N. J., Nolan, D., Baumer, B., Hall-Holt, O., Murrell, P., Peng, R., Roback, P., Temple Lang, D. et al. (2015), 'Data science in statistics curricula: Preparing students to 'think with data'', *The American Statistician* **69**(4), 343–353.
**URL:** *http://www.tandfonline.com/doi/abs/10.1080/00031305.2015.1077729*

Hester, J. & Wickham, H. (2017), *odbc: Connect to ODBC Compatible Databases (using the DBI Interface)*. R package version 1.1.3.
**URL:** *https://CRAN.R-project.org/package=odbc*

Horton, N. J., Baumer, B. S. & Wickham, H. (2015), 'Setting the stage for data science: integration of data management skills in introductory and second courses in statistics', *Chance* **28**(2).
**URL:** *http://chance.amstat.org/2015/04/setting-the-stage/*

Ioannidis, J. P. (2005), 'Why most published research findings are false', *PLoS medicine* **2**(8), e124.
**URL:** *http://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.0020124*

Kline, K. E., Kline, D., Hunt, B. & Heymann-Reder, D. (2005), *SQL in a Nutshell*, O'Reilly: Sebastopol, CA.

Knuth, D. E. (1984), 'Literate programming', *The Computer Journal* **27**(2), 97–111.

Li, W. (2018), Tools for understanding taxicab and e-hail service use in New York City, Undergraduate honors thesis, Smith College.
**URL:** *http://scholarworks.smith.edu/theses/*

Marwick, B. (2017), 'Computational reproducibility in archaeological research: Basic principles and a case study of their implementation', *Journal of Archaeological Method and Theory* **24**(2), 424–450.
**URL:** *https://osf.io/preprints/socarxiv/q4v73/download?format=pdf*

Müller, K. & Wickham, H. (2017), *tibble: Simple Data Frames*. R package version 1.3.3.
**URL:** *https://CRAN.R-project.org/package=tibble*

Müller, K., Wickham, H., James, D. A. & Falcon, S. (2017), *RSQLite: 'SQLite' Interface for R*. R package version 2.0.
**URL:** *https://CRAN.R-project.org/package=RSQLite*

O'Mahony, E. D. (2015), Smarter tools for (Citi) bike sharing, PhD thesis, Cornell University.
**URL:** *https://ecommons.cornell.edu/bitstream/handle/1813/40922/edo22.pdf?sequence=1*

O'Mahony, E. & Shmoys, D. B. (2015), Data analysis and optimization for (Citi)Bike sharing, *in* B. Bonet & S. Koenig, eds, 'Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA.', AAAI Press, pp. 687–694.
**URL:** *http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9698*

Ooms, J., James, D., DebRoy, S., Wickham, H. & Horner, J. (2017), *RMySQL: Database Interface and 'MySQL' Driver for R*. R package version 0.10.12.
**URL:** *https://CRAN.R-project.org/package=RMySQL*

Peng, R. D. & Dominici, F. (2008), *Statistical methods for environmental epidemiology with R*, Springer: New York.
**URL:** *https://link.springer.com/content/pdf/10.1007/978-0-387-78167-9.pdf*

R Core Team (2018), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
**URL:** *https://www.R-project.org/*

R Special Interest Group on Databases (R-SIG-DB), Wickham, H. & Müller, K. (2016), *DBI: R Database Interface*. R package version 0.5.
**URL:** *https://CRAN.R-project.org/package=DBI*

Sievert, C. (2014), 'Taming PITCHf/x data with pitchRx and XML2R', *The R Journal*

**6**(1).

**URL:** *http://journal.r-project.org/archive/2014-1/sievert.pdf*

Singhvi, D., Singhvi, S., Frazier, P. I., Henderson, S. G., O'Mahony, E., Shmoys, D. B. & Woodard, D. B. (2015), Predicting bike usage for New York City's bike sharing system, *in* B. Dilkina, S. Ermon, R. A. Hutchinson & D. Sheldon, eds, 'Computational Sustainability, Papers from the 2015 AAAI Workshop, Austin, Texas, USA, January 26, 2015.', Vol. WS-15-06 of *AAAI Workshops*, AAAI Press.

**URL:** *http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10115*

The Joint Task Force on Computing Curricula (2013), Curriculum guidelines for undergraduate degree programs in computer science, Technical report, Association for Computing Machinery (ACM) and IEEE Computer Society.

**URL:** *http://www.acm.org/education/CS2013-final-report.pdf*

Walker, K. & Rudis, B. (2017), *Tigris: Load Census TIGER/Line Shapefiles into R.* R package version 0.3.3.

**URL:** *https://CRAN.Rproject.org/package=tigris*

Wickham, H. (2009), *ggplot2: Elegant Graphics for Data Analysis*, Springer Verlag: New York, NY.

**URL:** *http://ggplot2.org*

Wickham, H. (2013), *hflights: Flights that departed Houston in 2011.* R package version 0.1.

**URL:** *https://CRAN.R-project.org/package=hflights*

Wickham, H. (2014), *nasaweather: Collection of datasets from the ASA 2006 data expo.* R package version 0.1.

**URL:** *https://CRAN.R-project.org/package=nasaweather*

Wickham, H. (2015), *R packages*, O'Reilly Media, Inc.: Sebastopol, CA.

**URL:** *http://r-pkgs.had.co.nz/*

Wickham, H. (2016*a*), *feather: R Bindings to the Feather 'API'*. R package version 0.3.1.
**URL:** *https://CRAN.R-project.org/package=feather*

Wickham, H. (2016*b*), *nycflights13: Flights that Departed NYC in 2013*. R package version 0.2.0.
**URL:** *https://CRAN.R-project.org/package=nycflights13*

Wickham, H. (2017*a*), *dbplyr: A 'dplyr' Back End for Databases*. R package version 1.1.0.
**URL:** *https://CRAN.R-project.org/package=dbplyr*

Wickham, H. (2017*b*), *tidyverse: Easily Install and Load 'Tidyverse' Packages*. R package version 1.1.1.
**URL:** *https://CRAN.R-project.org/package=tidyverse*

Wickham, H. & Chang, W. (2017), *devtools: Tools to Make Developing R Packages Easier*. R package version 1.13.4.
**URL:** *https://CRAN.R-project.org/package=devtools*

Wickham, H. & Francois, R. (2016), *dplyr: a grammar of data manipulation*. R package version 0.5.0.9000.
**URL:** *https://github.com/hadley/dplyr*

Wilkinson, L., Wills, D., Rope, D., Norton, A. & Dubbs, R. (2006), *The grammar of graphics*, Springer.

Williams, V. V. (2012), Multiplying matrices faster than Coppersmith-Winograd, *in* H. J. Karloff & T. Pitassi, eds, 'Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012', ACM, pp. 887–898.
**URL:** *http://doi.acm.org/10.1145/2213977.2214056*

Zhang, W. (2017), Improving access to open-source data about the NYC bike sharing system (Citi Bike), Undergraduate honors thesis, Smith College.
**URL:** *http://scholarworks.smith.edu/theses/1871/*

# Supplementary Materials for:

# A Grammar for Reproducible and Painless Extract-Transform-Load Operations on Medium Data

# A Extended discussion of related work

In this section we summarize the major considerations that make the `etl` package a progressive step towards reproducible research on medium data for R users.

## A.1 Reproducible research

To understand the current challenges we face in conducting reproducible research on PAMDAS, one must start with the notion of literate programming (Knuth 1984). In literate programming, source code is woven into an annotated narrative, so that one could read the source code and understand not just the code itself, but also how each piece of code fits into the larger design.

This idea leads to the notion of *reproducibility* in computational science. Donoho (2010) paraphrases Claerbout (1994):

> An article about a computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result.

Ioannidis (2005) argues that most published research is false, and while his arguments are *statistical* rather than *computational*, they only help to underscore the importance of computational reproducibility.

In academia, a diverse set of fields including computer science (Donoho et al. 2009), economics (Ball & Medeiros 2012), archeology (Marwick 2017) and neuroscience (Eglen et al. 2017) are actively debating how they will recognize reproducible research. Organizations like Project TIER (`http://www.projecttier.org/`) and the Open Science Framework (`https://osf.io/`) provide protocols for conducting reproducible research, while statistics and data science educators are instilling reproducible practices in their students (Baumer et al. 2014). Top-tier journals like the *Journal of the American Statistical Association* have appointed reproducibility editors (Fuentes 2016).

Thus, while the need for research in all fields to be reproducible is clear, the specifications for what qualifies as reproducible are less clear, and the path towards achieving reproducibility is murkier still.

## A.2 Medium data

In the past few years, *big data* has become an omnipresent buzzword that taps into our collective fascination with things that are massive. However, while a few enormous companies (e.g., Google, Facebook, Amazon, Walmart, etc.) generate and analyze truly big data (on the order of *exabytes* (EB), which are equal to 1000 *petabytes* (PB), which are equal to 1000 *terabytes* (TB), which are equal to 1000 *gigabytes* (GB)), most people who analyze data will never interact meaningfully with data of that size.

Most people will only encounter data that is *small* (a few gigabytes at most). These data fit effortlessly into a computer's memory, and thus the user experiences no challenges related to the data's size. Because a computer can access data in memory at lightning-fast speeds, efficient data analysis algorithms like searching ($O(n)$), sorting ($O(n \log n)$), and multiplying matrices (e.g., fitting a regression model) ($O(n^{2.376})$ (Williams 2012)) will run nearly instantly—even on a laptop. [7] Thus, for people working with small data, fundamental computer science concepts like the distinction between hardware and software, algorithmic efficiency, and bus speeds are immaterial.

For the vast majority of us who are unlikely to ever interact meaningfully with truly big data, *medium data* is both a viable solution and an accessible introduction to the challenges of big data (Horton et al. 2015). In Table 1, we constrast the relative sizes of data from the point of view of a personal computer user. Medium data is on the order of several gigabytes to a few terabytes. These data are large enough that they will not comfortably fit in memory on a personal computer without consequences, making a memory-only application like (vanilla) R a dubious candidate for data analysis. However, medium data are not so large they won't fit on a single hard disk, making them accessible to a single user without access to a computing cluster. An SQL-based RDBMS remains an appropriate storage and retrieval solution for medium data.

[7] Computer scientists use Big-O notation to describe the running time of algorithms by comparing the order of magnitude of the number of steps the algorithm takes to execute on an input of size $n$. An algorithm that runs in $O(n)$ time is *linear*, in the sense that the amount of time it will take to run is linearly proportional to the size of the input.

## A.3 Existing challenges

The fundamental challenge of big data is scalability, but medium data comes with its own challenges. In the end, investment in properly setting up an RDBMS pays off in more efficient analysis.

First, everything with medium data takes a little longer, since the aforementioned algorithms are no longer instantaneous. A single line of code might take one minute to execute instead of a millisecond, but these brief delays compound. Thus, those who employ efficient code and workflows are rewarded for their efforts with shorter execution times.

Second, a data analyst has to know something about SQL administration in order to set up a database. Many introductory data science courses that teach SQL focus on writing `SELECT` queries to retrieve data from an existing database—not on writing table schemas and defining keys and indexes (Hardin et al. 2015).

Third, getting PAMDAS set up involves often laborious ETL operations. Downloading medium data is not instantaneous and is dependent on the speed of one's Internet connection. Wrangling data is notoriously time-consuming work: reasonable estimates suggest this may occupy as much as 50–80% of a data scientist's time.

For these reasons, a responsible data scientist will record their ETL operations in a script. But these scripts are often problematic, ad hoc solutions. Some common problems include:

**Portability** Shell scripts may not port across operating systems. While Apple's OS X operating system is POSIX-compliant, not all flavors of GNU/Linux are. Microsoft Windows requires additional software to implement a compatibility layer, and thus any such scripts are not likely to run on Windows without careful modification.

**Usability** Under time pressure, data scientists are likely to write scripts that work for them, and not necessarily for other people. Their scripts may be idiosyncratic and difficult for another person to use or modify.

**Version Control** Even if a data scientist uses a formal version control system like `git` and GitHub, a script that ran when it was written may not run at all points in the future.

**Languages** ETL scripts may be written in `bash`, Python, R, SQL, Perl, PHP, Ruby, Scala, Julia, or any combination of these languages and others. There may be good reasons for mixing different languages but ease of portability decreases with each additional language.

One recommended solution for bundling ETL scripts for R users is to create an R package (Wickham 2015). Packages provide users with software that extends the core functionality of R, and often data that illustrates the use of that functionality. R packages hosted on CRAN—the authoritative central repository—are checked for quality and documentation, helping to ensure their *usability*. Since R is cross-platform, these packages are *portable*. CRAN itself maintains distinct *versioning*, and while R packages are mostly written in R, there are a number of ways in which code from other *languages* can be embedded into an R package (e.g., `Rcpp` provides functionality to bundle `C++` code (Eddelbuettel & François 2011)).

However, by design the types of data that can be contained in an R package hosted on CRAN are limited. First, packages are designed to be small, so that the amount of data stored in a package is supposed to be less than 5 *megabytes*. Furthermore, these data are static, in that CRAN allows only monthly releases. Alternative package repositories—such as GitHub—are also limited in their ability to store and deliver data that could be changing in real-time to R users. In Table 2 we contrast two different CRAN packages for on-time airline flight data (Wickham 2016*b*, 2013), with an `etl`-dependent package that allows the user to build their own database of flight data (Baumer 2017*a*). We note the change in scope that the `airlines` package allows: whereas the two existing data sets are restricted to small, static data from flights departing two Houston-area airports in 2011, or three New York City-area airports in 2013, respectively, the `airlines` package covers all domestic flights since 1987 departing from more than 350 airports nationwide, with more data available monthly.

Many R packages facilitate the retrieval of data from specific sources. In particular, the rOpenSci group maintains dozens of such packages (Boettiger et al. 2015). Other popular small CRAN packages that serve as APIs to large data sets include `tigris` (Walker & Rudis 2017) and `UScensus2010` (Almquist 2010). While these packages are undoubtedly useful,

| package | timespan | airports | size |
|---|---|---|---|
| hflights | 2011 | IAH, HOU | 2.1 MB |
| nycflights13 | 2013 | LGA, JFK, EWR | 4.4 MB |
| airlines | 1987–present | $\approx 350$ | $> 6$ GB |

Table 2: Alternative packaging of on-time flight data from the Bureau of Transportation Statistics in R. We note that the full scope of flight data is only accessible through the airlines package.

they are written by many different authors, and the syntax employed across packages varies greatly. In short, there is no consistent "grammar" (see Section 3). These packages are peripherals without a core.

Some dependency approaches do exist. Peng & Dominici (2008) illustrate how a small package for CRAN that interacts with large data repositories not hosted on CRAN could facilitate research in environmental epidemiology. These repositories are maintained by the package author through the use of a second package (Eckel & Peng 2009). More recently, the drat package provides a core that facilitates the creation of peripheral packages (Anderson & Eddelbuettel 2017). In this scheme the peripheral packages contain large amounts of data. The major drawback to both of these approaches is the requirement that the researcher maintain the large data repositories.

Boettiger (2015) advocates for the container-based solution Docker as an alternative packaging structure for reproducible research, and more recently Rocker (Boettiger & Eddelbuettel 2017), which provides Docker containers for R and RStudio. Çetinkaya-Rundel & Rundel (2017) promote this approach as university instructors. We see etl as fitting nicely into this paradigm, serving to further reduce barriers to reproducibility.

Perhaps the closest competitor to our approach is pitchRx (Sievert 2014), which performs ETL operations for a specific data set—in this case, detailed pitch information from Major League Baseball. Our approach places similar core functionality in the etl package and separates the data-source-specific functionality into small, easy-to-write packages that can be hosted on CRAN. The developer need not maintain any large data repositories—they need only to maintain the small bits of code that interact with the data provider. If, for any reason, the source data changes, etl users still retain copies of the raw data as they downloaded it.

We imagine that many of these aforementioned packages could be re-factored to have `etl` as a depedendency.

# B A toy example

Here, we illustrate the functionality of the `etl` package on the built-in `mtcars` data set.

The first step is to instantiate an `etl` object using the `etl()` function. We use the `etl_create()` function to perform the entire ETL cycle on an object named `my_cars`. During this process, a local SQLite database is created in a temporary directory, that database is initialized, the `mtcars` data is "downloaded" (i.e., in this case, from memory), transformed, and finally uploaded to that same SQLite database.

```
my_cars <- etl("mtcars") %>%
  etl_create()

## No database was specified so I created one for you at:
## /tmp/Rtmpdcqje1/file4fd547ea2707.sqlite3
## Initializing DB using SQL script init.sqlite
## Extracting raw data...
## Transforming raw data...
## Loading 6 file(s) into the database...
```

The object `my_cars` is both an `etl_mtcars` object and a `src_dbi` object—and can thus do anything that any other `src_dbi` object can do. It also maintains a connection to the SQLite database, has two folders (e.g., `raw` and `load`) where it can store files, and knows about a table called `mtcars` that exists in the SQLite database.

```
class(my_cars)

## [1] "etl_mtcars" "etl"        "src_dbi"    "src_sql"    "src"

summary(my_cars)
```

```
## files:
##   n     size                 path
## 1 6 0.004 GB  /tmp/Rtmpdcqje1/raw
## 2 6 0.004 GB /tmp/Rtmpdcqje1/load
##       Length Class            Mode
## con   1      SQLiteConnection S4
## disco 2      -none-           environment

my_cars

## dir:  12 files occupying 0.008 GB
## src:  sqlite 3.22.0 [/tmp/Rtmpdcqje1/file4fd547ea2707.sqlite3]
## tbls: atmos, borders, elev, glaciers, mtcars, storms
```

Since `my_cars` is a `DBI` data source, the data stored in the SQLite database can be accessed in the usual manner. Here, we compute the average fuel economy for these cars. Note that these computations are performed by SQLite.

```
my_cars %>%
  tbl("mtcars") %>%
  group_by(cyl) %>%
  summarize(N = n(), mean_mpg = mean(mpg))

## Warning:  Missing values are always removed in SQL.
## Use `AVG(x, na.rm = TRUE)` to silence this warning

## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.22.0 [/tmp/Rtmpdcqje1/file4fd547ea2707.sqlite3]
##     cyl     N mean_mpg
##   <int> <int>    <dbl>
## 1     4    11     26.7
## 2     6     7     19.7
## 3     8    14     15.1
```

The my_cars object itself occupies very little of R's memory.

```
my_cars %>%
  object.size() %>%
  print(units = "Kb")

## 3.2 Kb
```

# C Benchmarking

Recall that in Section 2.2 we created a tbl_dbi called trips that is connected to a database table of Citi Bike trip rentals. In this example we illustrate how the ability of dplyr to offload certain computations to SQL can result in marked performance improvements, even on the same computer.

```
class(trips)

## [1] "tbl_dbi"  "tbl_sql"  "tbl_lazy" "tbl"
```

Previously, we used the following pipeline to compute the number of unique combinations of stations, days, and hours in the month of September 2013. In the code below, we make use of the lazy evaluation design of dplyr to push the computation to MySQL. Note that the functions in uppercase are MySQL functions—not R functions. The collect() verb is applied only after the database is queried so that R can count the number of resulting rows. Because MySQL is good at doing this type of operation, and only 167,258 rows of data are sent from MySQL to R, this computation takes only a few seconds.

```
system.time(
trips_sept <- trips %>%
  filter(YEAR(start_time) == 2013) %>%
  group_by(start_station_id, DAY(start_time), HOUR(start_time)) %>%
  summarize(N = n(),
```

```
            num_stations = COUNT(DISTINCT(start_station_id)),
            num_days = COUNT(DISTINCT(DAYOFYEAR(start_time)))) %>%
  collect()
)

##    user  system elapsed
##   0.309   0.000   1.750

nrow(trips_sept)

## [1] 167258
```

Conversely, we can use the `lubridate` package for assistance with dates, and the `collect()` function to bring the data into R for summarization. Note here that only the `filter()` operation is actually performed by MySQL, while the rest of the operations are performed in R.

```
library(lubridate)
system.time(
trips_sept <- trips %>%
  filter(YEAR(start_time) == 2013) %>%
  collect() %>%
  group_by(start_station_id, day(start_time), hour(start_time)) %>%
  summarize(N = n(),
            num_stations = n_distinct(start_station_id),
            num_days = n_distinct(yday(start_time)))
)

##    user  system elapsed
##  28.461   0.855  29.322

nrow(trips_sept)

## [1] 167258
```

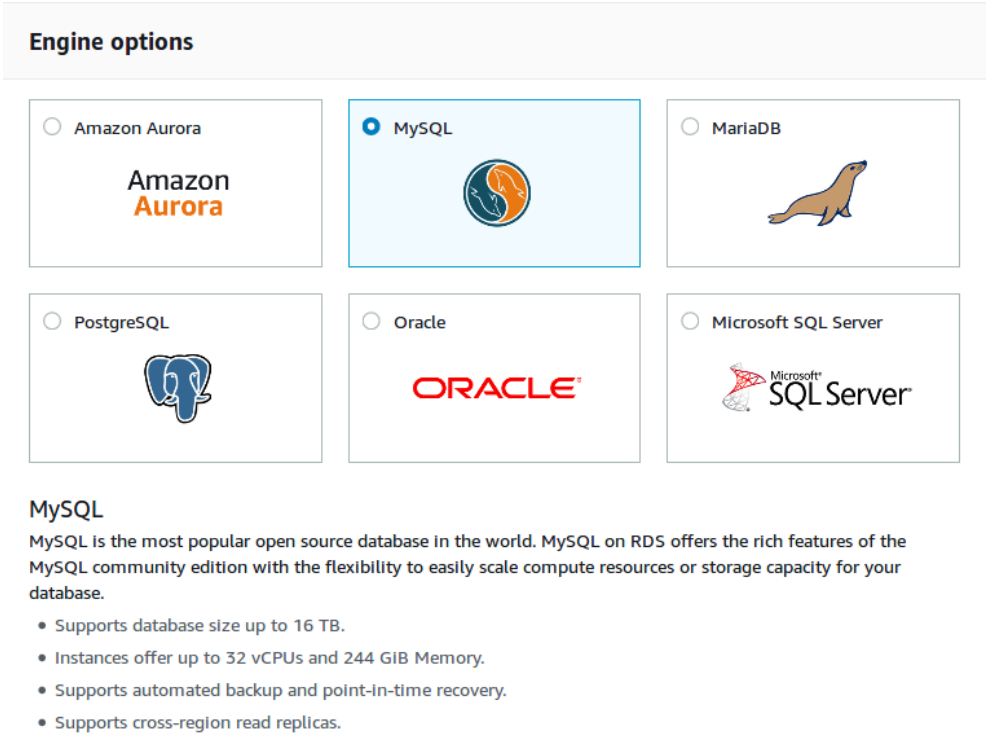


Figure 1: Amazon RDS

This latter method is much slower since it has to transfer more than 1 million rows of data from MySQL to R, instead of only 167,258. The delay with the second method is noticeable enough to start a conversation with students about scalability.

# D Using Amazon RDS

In this section we provide a brief tutorial explaining how to set up a medium database of taxi trip information on Amazon RDS (a cloud-based service) and populate it.

First, you must set up an Amazon Web Services account at `https://aws.amazon.com/rds/`. Our goal is to launch a new relational database service instance. In this example we will create a MySQL database that uses the Free Usage Tier (to avoid fees). In Figure 1, we show how to select the MySQL engine from among the available options.

Since we are simply testing this service, we select the "Dev/Test" usage case, which is the only one that is available under the Free Usage Tier (see Figure 2).

Next, in Figure 3 we allocate only minimal resources to this database instance. The

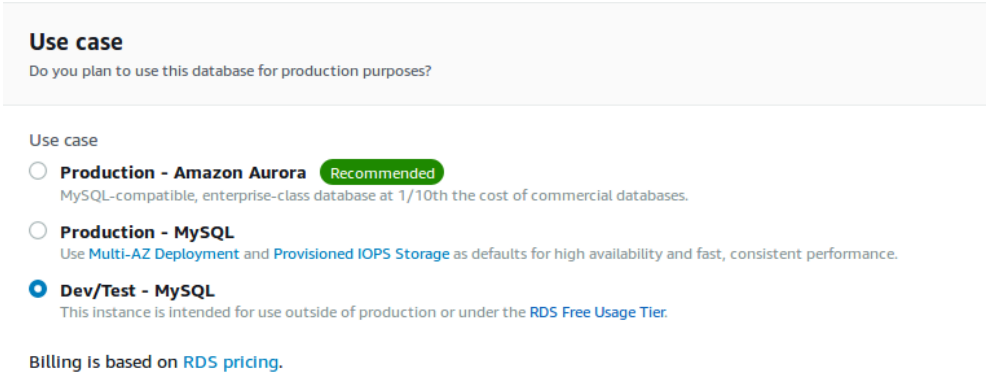


Figure 2: Amazon RDS

`db.t2.micro` instance has only 1 CPU and 1 gigabyte of memory. This is the only allowable configuration in the Free Usage Tier.

In Figure 4, we elect to make our database publicly accessible. This is an important deviation from the default, which is to restrict access to a Virtual Private Cloud. Without selecting "Yes" here, we would not be able to connect to our database from our R client. Please consult the documentation on Amazon in order to fully understand your security settings. Note also that by default, public access is only granted from *your* IP address.

In the next step, we set up a username, password, and schema. These are specific to the MySQL instance on our cloud-based database server. After accepting all of the default options on the remaining screens, our instance will launch. This process creates a virtual MySQL server that is running on Amazon's servers. The hostname for that server is shown in your Instance dashboard under "Endpoint".

```
host <- "etl-test.cdc7tgkkqd0n.us-east-1.rds.amazonaws.com"
```

If we didn't set up a schema on the MySQL server called `nyctaxi` already, we can create one using the Terminal tab available in RStudio. Be sure to use the credentials for the MySQL instance that you specified.

```
mysql -h etl-test.cdc7tgkkqd0n.us-east-1.rds.amazonaws.com -u bbaumer -p -e
"CREATE DATABASE IF NOT EXISTS nyctaxi;"
```

Finally, we load the `nyctaxi` package and connect to our database instance.

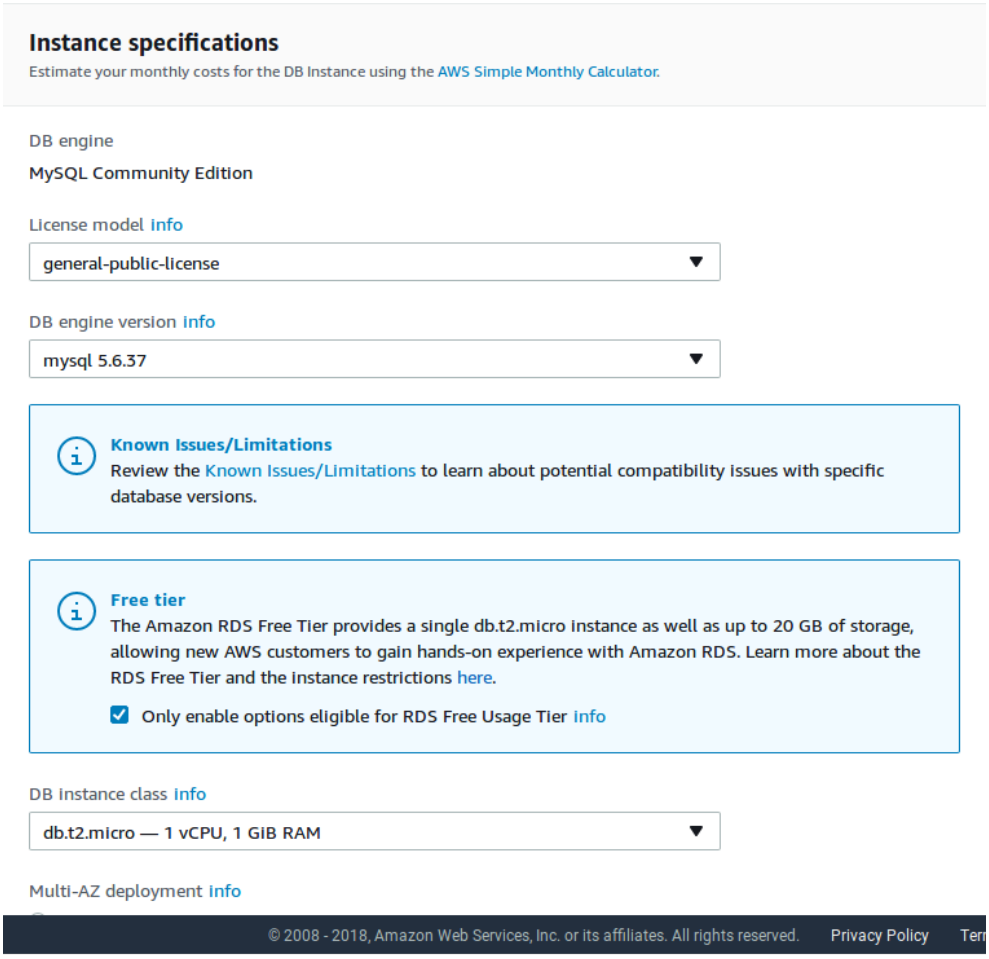


Figure 3: Amazon RDS

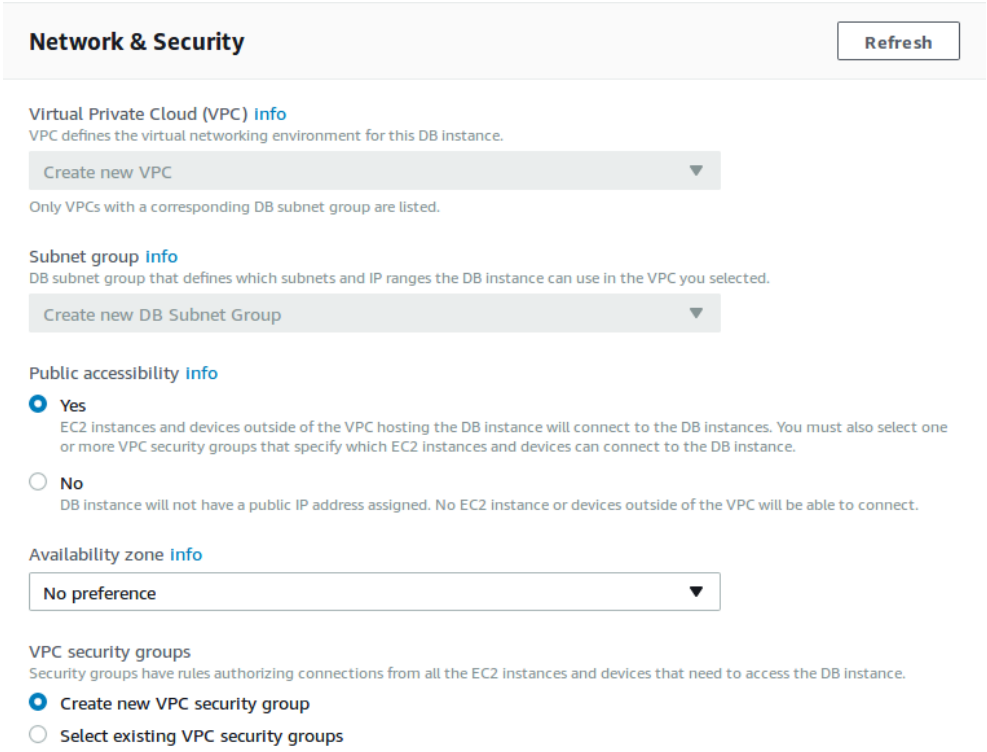


Figure 4: Amazon RDS

```
library(nyctaxi)
db_rds <- src_mysql(dbname = "nyctaxi",
                    host = "etl-test.cdc7tgkkqd0n.us-east-1.rds.amazonaws.com",
                    user = "bbaumer",
                    password = "xxxxxxxx")
```

The `etl` grammar now allows us to easily populate the database.

```
rides <- etl("nyctaxi", db = db_rds, dir = "~/dumps/nyctaxi")
rides %>%
  etl_update(years = 2014, months = 3)
```