Audit Analytics with R

Jonathan Lin

2020-07-07

Contents

Welcome

This is the website for Audit Analytics in R. This audience of this book is for:

- Audit leaders who are looking to design their environment to encourage code sharing and data products,
- Audit data analytics practitioners, who are looking to leverage R in their data analytics tasks.

You will learn what tools and technologies are well suited for a modern audit analytics toolkit, as well as learn skills with R to perform data analytics tasks. Consider this book to be your roadmap of practical items to implement and follow.

If you are brand new to R, it is encouraged you to read https://rstudio-education.github.io/hopr/ and https://r4ds.had.co.nz. While some foundations will be covered in this book, this book is focused on an applied view of R to the financial auditor practice.

About the author

Chapter 1

Introduction

Within the accounting and audit profession, analytics has been around for several decades, under the concept of Computer Aided Auditing Techniques (CAATs). The software of choice was led by ACL and its ACL Analytics software. ACL Analytics was a significant audit enabler at the time, as it allowed direct access to analyze mainframe data and flat files that were otherwise inaccessible by mainstream software on the market. It enabled audit teams to obtain transparency in analysis, a rigorous audit trail, and even automation of scripts.

As computers, data analytic technology and accessibility of coding in the Accounting practice has become mainstream, there are far more tools that enable auditors to become far more powerful and self sufficient than ever before. Tools that are typically reserved for software engineers and statisticians have empowered financial auditors to expand their breadth and scope.

A traditional internal audit team would consider themselves to be consumers of information, limited by flat files sent by emails from their stakeholders. While most internal auditors have considered data analytics in one way or another, the realm of possibilities and challenges have outpaced audit shop capabilities. The expectation now is for auditors to be fully integrated into the business, and contribute directly to the management of the financial and IT risks the company faces on a regular basis.

The most effective way to meet this new standard is to implement a current data analytics architecture and empower your team to leverage modern data analytics techniques.

Chapter 2

Approach

For many audit teams, having data analytics employed in your audit team is generally a given. The advantages are well articulated - 100% population coverage, timely response, and targeted high risk identification and results are true and real.

Unfortunately, any short-burst of investment in data analytics training tends to not last more than a few months. This is due to lack of applied skills, sustained management interest, and consistency of values. While regulations for adherence to SEC and Sox requirements exist and are necessary, the means to approaching and achieving compliance has been largely static in nature since the introduction of Sox.

The root cause is that audit teams culturally view analytics as a skill, and not as a cultural shift. On a micro level, while not every individual on your audit team needs to be a ‘coder’, everyone needs to share the same values and philosophy for any audit analytics program to have success. Your core analytics team will set the foundation for the rest of the team to play on, but if they don’t show up, the analytics program is limited at best.

Having a combination of the right values and technologies will enable your team to propel forward, and should amplify your teams efforts. By thoughtfully choosing your tools and encouraging sustainable processes, your team will create a positive cycle of development, learning and deployment.

2.1 Code-based development

Consider the creation and auditing of a spreadsheet, where every row and column has the potential to be manipulated. Following the motto, ‘trust but verify’, an auditor would need to examine each cell value, the formulas, the relationships, and keep a sharp eye out for manual adjustments. The auditor also needs to validate the source of data in the spreadsheet. With no built-in tracability to how the spreadsheet is used after it has been created, it contributes to the madness that is the spreadsheet ecosystem.

While it is easier to superficially consume information from a spreadsheet, to understand the inner workings is a tedious task into itself. And unfortunately, both controls and audits are conducted without a second thought within spreadsheets, as it is easier to tackle the task without fundamentally changing the approach itself.

Contrast this to a code-based environment. To achieve anything in code, you need to be explicit and specific on the mechanisms taken to reach the end state. Each line will tell the program what inputs it needs, how it is processed, and the collection of lines tells you what is achieved. The beauty of this is that it also tells the reader exactly what was executed to achieve the end result. A code based environment is inherently self documenting.

Once a baseline code has been established, finding ongoing changes becomes trivial. Similar to black-line functionality in Microsoft Word or Track Changes in Google Docs, its far easier to detect how code (and the process it supports) has changed.

Perhaps with some merit, code can be difficult to read at times, as it is still a completely different language. **Notebooks** address this problem. Notebooks are interactive renderings of code, containing not only the code, but can also include sections of commentary as to why something was done, and can include data visualizations and interactivity. It is the ultimate form of auditability.

2.2 Automate relentlessly

By writing your routines in code, then the next logical step is to automate. Automation not only frees up your time from performing a task, but it also frees up your mental critical thinking and creative processing power. Every professional has a cognitive load - don't waste it on routine tasks.

A common use case for automation in audit is the creation of a data mart that is relevant to auditors. A data mart is a collection of pre-processed data that is suited for the group using it - for example, a data mart may hold a subset of accounts payable information. Imagine that instead of needing to email HR for the latest employee list, the audit team merely needs to check within its internal database, makes research a snap Instead of asking the vendor management team on how much money was spent per vendor, having this information in an audit mart already available means you can spend more time thinking about how to assess these vendors for risk.

Gone are the days where you only audit a topic once a year. A byproduct of every audit is a monitoring mechanism - how do you know something has gone off the rails? Traditional remediation paths include a follow-up in the future - after that, there are few mechanisms to faithfully maintain confidence that the process is still operating reasonably.

If you've coded your audit, then you have already done the hard work of structuring how to extract your data data, finding exceptions and distributing actionable results for your stakeholders. Automation takes that one step further, and allows you to ensure the steps are done repeatedly. The value you derived initially from your audit can now be done continuously.

Once you've audited the identification of issues and the metrics around them, then you can focus on the automation the handling and resolution of them. With a true audit results database, you can the results of automated monitoring into it, enabling on-demand availability, visibility, transparency, and even reporting, all which can now be part of that automation process.

2.3 Share everything

> Tribal knowledge is information or knowledge that is known within a tribe but often unknown outside of it. A tribe, in this sense, may be a group or subgroup of people that share such a common knowledge. From a corporate perspective, "Tribal Knowledge or know-how is the collective wisdom of the organization. It is the sum of all the knowledge and capabilities of all the people". (tri, 2020)

When you code the procedures, not only are you creating letters to yourself in the future, but you're also writing letters to everyone else on your team, even those who haven't joined. These fully contained notebooks serve as a guide to those on your team who are learning what you have developed.

The next thing to do is to put them in a central location where everyone on your team can access them. While you can opt for files and folders on a network drive, more practical technologies exist. **Code repositories**, such as git and subversion, serve a purpose in your data analytics environment to track changes to code and notebooks. You can encourage your team to upload notebooks to these repositories to access the latest and greatest code that your team has developed for problems already solved. New team members can go into a repository to learn how code has solved prior problems, and become inspired to solve problems of their own.

As your team gains more experience and consistency, it may be more practical to write repeatable processes and functions instead of copy and pasting code between notebooks. Code **packages** enable you to share your code and functions to solve specific problems, templates to encourage consistency amongst team members, and are easily distributed to your team for quick deployment.

2.4 Always deliver useful product

The outputs of your work are useless if they are not adopted by the business. This means you need to prioritize your customer and iterate often, to win buy-in, gain momentum and stay on course.

Audit teams are always surprised when I say that they have customers. Audit department interactions are across the entire company and up-and-down the entire chain - from manufacturing to engineering, IT to accounting, analysts to the c-suite. Audit departments handle all these groups, with kid-gloves and diplomacy. Yet we tend to ignore the #1 customer - ourselves. Audit teams are usually at the mercy of what the business deems most convenient for them - whether in terms of reports, testing controls or even where files are kept.

Historically, this was tolerated. Audit teams were composed of accountants that specialized in Excel, IT auditors in testing general controls, and a focus adhering to evolving accounting standards. With the advance in maturity of code-based tools and even commercial audit software, there has been never a better time to code intuition.

The idea of these data products is that they promote the cognitive ease - this means, repeated experience, clear display, primed idea, and good mood.(Kahneman, 2011) By helping your team feel at ease, but they should deliver both the most trivial of tasks and the most time-consuming tests of applied knowledge. By removing the need to execute these tasks, then the customer can focus on what they're good at - in the case of auditors, its applying professional judgement. These products can be taken up in the form of data generation, data and trend visualization, or standardize reporting.

2.5 Don't be a hero

Not everything needs to be fixed by you, especially in code. In a well-functioning organization where hundreds of people are supporting different software, application or databases, sometimes the best way to solve the problems found isn't directly in the work you do, but how you get others to fix systemic issues in the source itself. Key indicators of this are when you have to start hard-coding compensating controls in your code because the upstream data isn't standardized.

For example, in a typical company, employees are issued unique ID numbers. This is preferably at the source of truth - the Human Resources department. This identifier is generally considered reliable and stable, especially if is the source of truth. Consider now that you're now trying to join the data to a credit card system, where employees are issued credit cards based on their first and last name. If you take an assumption to join this dataset using an individuals first and last name, that would be a reasonable first attempt. However, last names can change over time, or individuals may prefer to go with their middle names, or even simple spelling mistakes can occur. Instead of compensating by adding different ways of 'joining' information, its more effective to work with the application's data owner to see if they would be willing to adopt the unique employee ID instead as a field in their system.

2.6 "The opportunity cost"

> The idea of opportunity costs presumes the fungibility of human experience: all our activities are equivalent or interchangeable once they are reduced to the abstract currency of clock time, and its wage correlate. (Crawford, 2009)

An internal auditors' largest limitation is time. The biggest barrier to the adoption and execution of data analytic talents is the fact there is 'more' perceived value in doing something else. A common excuse is that a task will only be performed once (or even worse, at an irregular basis) - auditors may instead opt to perform a manual task in an unsustainable way. Instead of learning a new skill, our opportunity cost is projected against how much time can be used have to perform this task at hand.

If your existence was highly limited by the lifespan of a fly, it would be hard to argue against such a position. However, your career is long (hopefully), fruitful and full of exciting and interesting work.

Truly engaging work should challenging and rewarding, and applying coding to audit lends itself to a skill worth mastering. Using these skills, no matter how immature, will continue to pay off dividends in future implementations, no matter how insignificant or incremental. Implementing repeatable processes means not only does the end consumer receive their product quicker, but the data auditor is enabled to continuously refine or tackle another code-related problem. And that is worth the opportunity cost, today.

Chapter 3

Architecture

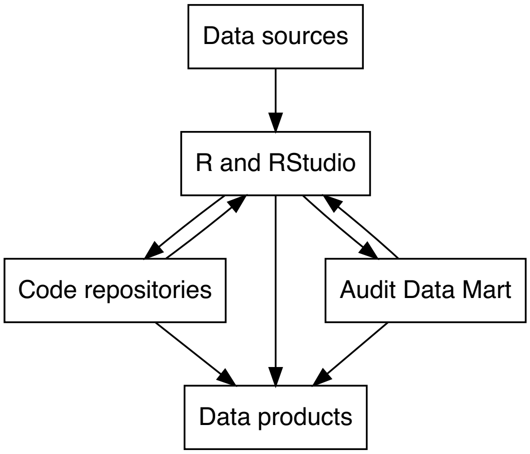

Figure 3.1: Internal Audit Data Analytics Architecture

The architecture and tools you select should support and amplify your team, and not be a burden to maintain. Maintenance of databases and applications can also be delegated to application support teams, so you can focus on implementing analytics and delivering data products.

Other things to consider: You should have, at a bare minimum, direct access to read-only internal databases or data warehouses. A generic service user account has its merits here when it comes to automation - tying an eventual automation to a user with password expiry every 90 days will make updating passwords feel like quarterly financial reporting.

Any software you consider should be able to talk to your company's internal and external applications. Be wary of software that locks you in or makes access to data cumbersome, as it limits your ability to integrate with your company's tools as you gain sophistication.

3.1 R and RStudio

R is a programming language with an emphasis on statistics. It is considered free software, free in the perspective that you can run, distribute, and change it as you like.

What makes R so great is the large suite of packages that are available to use to help analyze information. What doesn't exist in base R will likely have been developed by someone else: there are hundreds of packages that support data connectivity (DBI, xlsx), day-to-day manipulation and analysis (dplyr, ggplot2), and even auditing (jfa, MUS). CRAN is the central repository for all these packages that meet minimum quality standards.

As R is a language, you may want to consider an application to code in, similar to how you may write memos in Microsoft Word. RStudio Desktop is an integrated development environment (IDE) that has an open source version and is free to use. There are also free versions of RStudio Server, as well as commercially supported versions of its Desktop, Server, and two unique products that we will go into more later - Package Manager and Connect.

With both R and RStudio installed, you can perform the minimum requirements of your audit data analysis career: you can download data, wrangle it, visualize it, and save completed analyses. The potential is limitless though, as it enables all the other technologies to operate with it - machine learning, automation, dashboarding, and code sharing.

3.2 Code repositories

Here is a typical logistical challenge faced in even the smallest of audit teams. Person A will write the first version of a script to download data (V1). Person B in the team may want to use it to download data from elsewhere, so they will get an emailed script file from person B, modify it, and start to use it. Person A may make improvements to their own file (now V2), but Person B will not benefit from the improvements. Knowledge are immediately siloed, and changes become more difficult to share between auditors.

One of the most effective ways to solve this problem is to leverage a code repository (also known as version control). While version control has several software engineering advantages, the most notable advantages for audit teams are:

- Centralizing domain knowledge and sharing code,
- Tracking and tracing changes to code throughout time (including who, what was changed, and when), and
- Ability to test with an isolated code set before rolling out to production.

Code versioning technologies resonate closely with IT General Controls and even the COBIT framework.

3.2.1 Git

git is a type of version control. Another free, open source software, and the basic usage of the tool is accessible.

The basics of git are:

- Pull code from the remote server to the local computer.
- Write and save code on your local computer.
- Commit code on your local computer.
- Push code from your local computer to the remote server.

While this is a superficial example of how to use git, it is enough to get the most basic of audit teams started. The git hole can go very deep, so if things don't go according to plan, just remember that you can always make a new folder and start fresh.

Several different server technologies support git - github, Azure Repos within Azure DevOps, gitlab are all willing to host your code. If you're using these to hold proprietary or sensitive code, it would be wise to get your company's support and pay for the ability to have a private repository that only your team can see.

3.2.2 Packages

As you write more code and templates, you will eventually want to share these new techniques with others on your team. Packages put your best practices together, including templates, functions to solve common problems, and templates for common workflows. In short: packages contain tasks help you do your job, and save you time.

Packages go beyond the tangible and provide several qualitative benefits as well. They standardize your team's workflow, create consistency and readability standards, and get your team to speak a single language. It creates a cohesive foundation where anyone on the team can contribute, and a library for those who wish to learn more.

However, you do not need to force your team to copy-and-paste a folder or even a Word document from a template folder. The most elegant way to distribute your code is via packages. These packages can be hosted on a private or public code repository, enabling your team to download best practices with a simple line of code. You can also compile your packages and leave them on a network drive.

The best part? You can give your package a creative name that represents the culture of your internal audit team or organization. Just don't call it auditR.

3.3 Data products

All auditors face this problem at one point or another. An audit finding doesn't matter until someone takes accountability for it. And if an audit finding is too vague, or does not have the buy-in from the right stakeholders, it is as good as unsolicited advice.

The data analysis you deliver are vulnerable to the same expectations. Not only do they need to be objective and accurate, but they also need to be accessible, relevant, and delivered at the right time. If there are too many barriers for entry or usage, someone will revert back to their old processes and methods. Here is a sobering thought: when was the last time you made a beautiful visualization for someone, only to be asked for the detailed Excel worksheet afterwards? Or instead of an automated report that users can self-serve, someone asks for an email instead?

Data products are deliverable that enhance your customer's ability to gather, process and analyze information. They facilitate decision making and are integrated into their processes. They should be designed thoughtfully and simply, masking the complexity underneath. In short, they should make your co-auditors feel like rockstars. The reports you write and code you develop should strive for the goal of being accessible by your team, on-time and on-demand.

3.3.1 Galvanize Highbond

Galvanize, the parent company of ACL Analytics and one of the most popular companies in the audit software industry, invested significant efforts into their cloud-based working paper solution, Highbond. One of their modules, Highbond Results, is an unrivaled for audit exception and remediation workflow.

The idea with an audit exception workflow is that audit testing will identify an actionable transaction or outcome. This may be an exception within a process, a control requiring execution, or even a request for additional information or clarification. Once a process has been designed, Highbond Results will allow you to focus on the users who should action the workflow and the rules for setting up triggers.

While Galvanize is a separate, third-party product that is not directly in the R universe, it does integrate with R through its API. The API enables you to upload findings and results directly into a Results set, allowing you to create the workflows on the website. The API also enables access to its Projects data that your audit team may already use to document audits, offering advantages to audit teams that design their workpaper environments effectively.

Highbond Results also provides capabilities to do no-code based visualizations hosted on the web. Once set up, they offer a stable method of delivering storyboards and visualizations.

Galvanize cloud-based tools are fully hosted, meaning audit teams pay for high availability and security maintained by a professional team. Galvanize supports the security, design and coding for hosting a tool online, allowing you to focus on designing workflows for your internal customers.

3.3.2 RStudio Connect

RStudio PBC, which became a Certified B Corporation in 2020, offers the RStudio Connect solution that allows R deliverable to be hosted online. These include:

- RMarkdown notebooks, which are fully self-contained analytics. These notebooks perform full analyses from start to finish, including downloading data, wrangling, analysis and data visualization.
- These notebooks can also act at Extract, Transform, and Load (ETL) processes. These notebooks have the advantage of being automatically scheduled, with rules that can notify stakeholders if need be. The loading component can be any destination - most popular is the audit data mart, or into other web applications via API connectivity (example: Galvanize Highbond Results).
- The hosting of Shiny apps, which are interactive web applications, offer a way to analyze and present information in an intelligent, slick manner. The analysis performed in R can be factored into a Shiny app, which can be hooked directly into your data.

For audit teams with expertise in programming, RStudio Connect offers some of the best capabilities for publishing visualizations and analysis to your teams and internal stakeholders. With its git capability, it can also receive updates from a code repository, integrating tightly with a team's best practices of a code repository.

RStudio server software will require talent and cost to stand-up and maintain, and should be considered in an environment where automation and internal hosting of data products will bring advantages to an internal audit team.

Lightweight alternatives include the hosting a free Shiny Server or low-cost publishing to Shinyapps.io.

3.3.3 RStudio Package Manager

As your team develops more code and functions, there becomes a greater need to distribute these best practices easily. RStudio Package Manager offers the capability to distribute code packages to your coworkers and even the broader organization. By integrating with your code repository, it can bundle new functionality added by your team and distribute it. It offers versioning of packages, for those audit environments where reproducability is paramount.

An alternative is also miniCRAN, a reduced feature set yet free and open source version, or even simply hosting packages on a network drive.

3.4 Data sources

The end goal of automation ETLs, and creating and hosting data products for audit customers to consume. To achieve these goals, an audit team needs direct, programmatic access to the data being audited. By programmatic, we mean that code can be used to access data, versus going through a front-end user interface.

3.4.1 Databases

At its core, databases hold transactional information that runs the business. Databases generally can fit two separate use cases: Online Transaction Processing (OLTP), for high speed transaction writing, and Online Analytic Processing (OLAP), for analyzing to support business decisions. Typically OLTP databases act as a source of truth, and send updates to the OLAP database.

The language of choice to access internal databases is Structured Query Language (SQL). As a defacto standard for accessing databases since 1970's, all relational databases still leverage SQL. While each brand may have subtle nuances in the way SQL works, this essential language will allow your audit team to access a majority of sources within the company. Non-relational databases and NoSQL are becoming more mainstream, as well as graph databases, so your audit team will need to tool up as necessary.

Audit teams historically had a bad reputation for 'bringing down the database'. While this used to be quite common, nowadays computing power is so accessible and cheap that the fear is generally unwarranted. Great caution should still be taken, where you should avoid querying any production database that supports customers or staff directly, unless absolutely necessary. Other strategies are useful depending on the circumstances:

- Filter the data with WHERE clauses,

- Perform joins in the database, instead of downloading multiple tables individually and joining them on the desktop,
- Download data in chunks or segments, split by day, week or month, or
- Schedule queries during off-business hours.

3.4.2 External sources

As more applications move to the cloud, using SQL to access data becomes more difficult. While some of this data can be brought into an internal data warehouse or used as an integration, more often than not transactional data is left online within the tool.

Certain online cloud software providers will make an Application Programming Interface (API) available for customers These APIs open a window to the cloud application, where subsets of data can be downloaded from the system. Each vendor may provide API documentation, and then can be accessed via packages like httr and digested with jsonlite.

While APIs enable audit teams to retrieve data, it adds a layer of difficulty in obtaining data:

- Each API behaves differently - authentication, calls, and the data return all may vary between systems.
- APIs endpoints, the part that allows a tool to query it, tend to be highly specific-use cases, and may provide a limited scope of data at one time.
- Data may not be in the format that you want, or even exist - you are at the mercy what the API supplies.
- If an API does allow for a larger data download, it may be limited by pagnation, where multiple results are spread out over multiple pages.
- APIs may be 'rate-limited', which means it may restrict the number of queries, whether in parallel or in sequence.

Tips for audit teams needing to rely on cloud providers:

- API access should be part of the requirements before signing an agreement with a provider.
- Ask for, and generate, an API key for usage, and treat it like a password.
- If APIs are out of the skill of your team, consider asking your IT department to schedule a data pull into your audit data mart.

3.5 Audit data mart

While internal databases are accessible, generally they hold the entirety of the company's data, which is far too much information. Only a fraction of the data is considered important and significant to the audit team.

An audit data mart is a key piece of infrastructure that will sit between your data products and data source. This data mart should contain highly specific, refined and cleaned data, which improves the speed and responsiveness of data products and data-on-demand. The audit data mart can also be secured to your audit team members, so confidentiality is not compromised.

To take full advantage of an audit data mart, an automated ETL process should connect directly to internal databases, and perform transformations to get to a clean end product. ETLs can be created and scheduled within RStudio Connect.

Chapter 4

Setup

There are countless number of guides to setting up your local R and RStudio environment. If you're learning on your own, its easy to get some of the below applications and packages installed.

4.1 R and RStudio

One of the most respected introductions to R is R for Data Science by Hadley and Garrett, and the Prerequisities section is set up for installing R.

4.2 Common packages

We will use several common packages; if you haven't installed the yet, feel free to install the packages:

```
install.packages(c('tidyverse'))
```

4.3 The AUDITPACKAGENAME

```
# TODO publish audit package
```

4.4 Github (or another git-related brand)

There are many different companies that provide git - Azure, Atlassian, Gitlab, etc. If you're learning, github is a fantastic free source to set up shop at. Within your own company, see what the IT and developers are using, as its far more convenient to jump onto that.

Like the R tutorial, Happy Git and GitHub for the useR by Jenny Bryan et. al. is a practical guide to installing R, setting up keys and the core fundamentals.

Chapter 5

Import data

In this chapter, we will download some datasets and import them. You will need the following packages to follow along.

```
library(dplyr) # For manipulating data

library(readr) # For reading flat files

library(DBI) # For database connections

# For API connections
library(httr)
library(jsonlite)
```

5.1 Delimited files

The most common method of obtaining data is via flat files, usually in the form of comma separated files (CSV). While delimited data sources are the most convenient for data sources where direct data connections are otherwise unobtainable, they are not set up for long term sustainability and automation.

The base package, installed with all instances of R, and `read.table()` is a convenient built-in standard function for importing CSV files. Another package, readr, includes a similar function called `read_delim()`, which is faster and allows for easy altering of column specifications, which directs the data types each column is imported as (for example, overriding an employee's identification number as a character versus an numeric).

Before importing the file, lets download the file from the repository that contains the Vendor Master. This dataset contains the vendor system ID number, the date it was added, and other traits.

```
dir.create("data", showWarnings = FALSE) # Creates a directory in your project
download.file(url = "https://github.com/jonlinca/auditanalytics/raw/master/data/vendor_
              destfile = "data/vendor_master.csv", mode = "wb") # Downloads this csv f
```

When importing delimited files, there will always be a few aspects to consider. The *delimiter* is the character that separates each field - while commas (,) are popular, pipes (|) and tabs are also used as they tend to be less common. Using an uncommon character was a typical workaround when exporting data from legacy systems, as commas within text fields were incorrectly parsed as extra columns. If possible, *qualifiers* should used to enclose text with a field, typically quotes or double quotes. This will indicate to the system that everything within those quotes belongs to a specific field.

In this case, if you view the file (on the Files panel on the right, go to the data folder, click on vendor_master.csv, and select View File), you will see the data is separated by commas.

```
raw_vendors <- read_delim('data/vendor_master.csv', delim = ",")
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   name = col_character(),
##   date_added = col_date(format = ""),
##   spend_2015 = col_double(),
##   spend_2016 = col_double(),
##   spend_2017 = col_double(),
##   spend_2018 = col_double()
## )
```

The message indicates that default column types were assigned to each field imported. If it all the fields imported as expected, this message can be ignored.

However, ID numbers, while presented as a number, don't have a real value in any calculations. As a result, you can specify a column specification via the col_types argument, copy and pasting the framework in the message and changing the fields as need be:

```
cols <- cols(
  id = col_character(), # Changed from col_double()
  name = col_character(),
  date_added = col_date(format = ""),
  spend_2015 = col_double(),
  spend_2016 = col_double(),
  spend_2017 = col_double(),
  spend_2018 = col_double()
)

raw_vendors <- read_delim('data/vendor_master.csv', delim = ",", col_types = cols)

glimpse(raw_vendors)
```

```
## Rows: 6
## Columns: 7
## $ id         <chr> "835", "76", "582", "2691", "2276", "2669"
## $ name       <chr> "Williams, Franklin", "Le, Jacob", "Martinez, Amy", "Gar...
## $ date_added <date> 2019-09-05, 2019-05-14, 2019-07-15, 2019-10-08, 2019-09...
## $ spend_2015 <dbl> NA, NA, NA, 82075.98, 18603.04, 74661.33
## $ spend_2016 <dbl> NA, NA, NA, 100953.15, 26820.08, 132848.95
## $ spend_2017 <dbl> NA, NA, NA, 178028.85, 43208.78, 182072.08
## $ spend_2018 <dbl> NA, NA, NA, 95205.89, 25214.04, 148938.38
```

While well exported delimited files can be useful, they often contain hidden surprises. Consider this rather innocuous csv file from active directory (the controller for Windows authentication), with a username and manager fields:

```
# Active directory file, with just a username and manager field.

download.file(url = "https://github.com/jonlinca/auditanalytics/raw/master/data/active_directory.
              destfile = "data/active_directory.csv", mode = "wb") # Downloads this csv file int

raw_ad <- read_delim('data/active_directory.csv', delim = ";")
```

```
## Parsed with column specification:
## cols(
##   uid = col_character(),
##   manager = col_character()
## )
```

```
## Warning: 2 parsing failures.
## row col  expected    actual                        file
##   2  -- 2 columns 1 columns 'data/active_directory.csv'
##   4  -- 2 columns 1 columns 'data/active_directory.csv'
```

These warnings indicate that there were columns expected (as dictated by the first line of column headers), but missing in one or more lines. You can inspect the csv file in the RStudio interface by clicking on the file on the navigation pane to the right, and select ‘View File’. You will notice that the location for both accounts is on a new line, but it belongs to the prior record. The raw characters can be confirmed within R, by reading the file directly as is (i.e. raw):

```
ad_char <- readChar('data/active_directory.csv', file.info('data/active_directory.csv')

print(ad_char)
```

```
## [1] "uid;manager\r\njonlin; jason durrand\nlocation: calgary\r\nronaldlee; reid tur
```

These special characters are hidden within the seemingly innocuous delimited file, and are typical of systems where information is extracted from, especially Windows, `\r` represents a carriage return, and `\n` represents a line feed. Together, `\r\n` represents a new line and new record, while `\n` can appear in a file when a new line is made by pressing Shift+Enter.

In this common yet inconvenient case, these can be substituted out with regular expressions. Regular expressions are a standard, cryptic yet powerful way to match text in strings. We will cover specific use cases of these in Searching Text.

In this case, the below regular expression only replaces `\n` when there is no `\r` preceding it. The `gsub()` function will try to match the regular expression criteria in the in the first field, with its replacement value in the second field.

```
gsub("(?<!\\r)\\n"," ", ad_char, perl = TRUE)
```

```
## [1] "uid;manager\r\njonlin; jason durrand location: calgary\r\nronaldlee; reid turra
```

The gsub output shows that the manager’s name and location no longer has a `\n` in between them. As a result, it can now be imported cleanly.

```
ad_raw <- read_delim(gsub("(?<!\\r)\\n"," ", ad_char, perl = TRUE), delim = ";")

print(ad_raw)
```

```
## # A tibble: 2 x 2
##   uid       manager
##   <chr>     <chr>
## 1 jonlin    " jason durrand location: calgary"
## 2 ronaldlee " reid turra location: melbourne"
```

5.2 Databases

It is likely that the company you are auditing will have their data stored in a database. While having skills in SQL is recommended, having R skills means you are able to perform basic queries on databases. There are many different database brands and vendors in the world, and thus there are many different subtleties on how SQL works for each vendor, but they mostly adhere to the same principles,.

The Open Databases Connectivity (ODBC) standard allows different vendors to write drivers, or the technical back-end methods, to connect to their database. Generally, you will need a driver that matches the vendor and version of the database you're using. Installing a driver is straight forward

The Database Interface (DBI) is the interaction between R and the driver Practically, it enables R to send queries to the database via the driver that is defined in the ODBC.

The most common way to connect to a database on your network is to install the vendor drivers, and then create a Data Source Name (DSN). To properly create this DSN, you'll need the name of your database, as well as read-only credentials. Alternatively, you may specify the server name, database schema and credentials explicitly, which offers some advantages from a portability perspective as your other team mates will not need to create DSNs, and only need to install the drivers themselves.

For this example, we will use an SQLite database, which are self sustained databases files and perfect for lightweight applications (including training!) Again, lets start by downloading the file, or in this case, a database:

```
dir.create("data", showWarnings = FALSE)

download.file(url = "https://github.com/jonlinca/auditanalytics/raw/master/data/rauditanalytics.s
              destfile = "data/rauditanalytics.sqlite", mode = "wb")
```

One thing that is different about connecting to databases is that you need to set up a database connection within R. This will usually consist of a driver (in this case, `RSQLite::SQLite()`), a DSN or a file location (`data/rauditanalytics.sqlite`). In the help file, it asks for other needed fields as well; user, password, host etc. At your company, having that information along with ports, schema names, and whether or not its a trusted connection (authenticating automatically with your own active directory credentials). If you haven't yet already at your company, request a read-only account that can get this information.

We're going to establish a connection with the database, creating the `con` connection object:

```
con <- dbConnect(RSQLite::SQLite(), "data/rauditanalytics.sqlite")
```

You can confirm it works by listing the tables in the database. One important thing to remember is that you'll be passing this connection object each time as you perform your commands. For example, if you want to see what tables exist in the database and schema, you will still need to tell the command which database you want to connect to.

```
dbListTables(con)
```

```
## [1] "gl"      "vendors"
```

Typically to get data out of a database, you'll need to communicate to it in its own language to obtain data - SQL (Structured Query Language). Here is an example of how to select all the records in a table:

```
dbGetQuery(con, 'select * from gl')
```

```
##   je_num   amount gl_date gl_date_char vendor_id                   account
## 1      1  8346.79   18024   2019-05-08      2691        exp_materials_6000
## 2      1 -8346.79   18024   2019-05-08        NA liab_accountspayable_2000
##                        description
## 1 Quality control testing supplies
## 2 Quality control testing supplies
##  [ reached 'max' / getOption("max.print") -- omitted 1998 rows ]
```

While knowing SQL is advantageous (and eventually critical), sometimes switching between languages is a hassle, especially when performing basic tasks within a database. Using the dplyr package, you can generate several of the same queries using the dplyr syntax. All you need to do is create a reference to the table in the connection object - in this case, the `con` connection object contains the `gl` table:

```
db_gl <- tbl(con, 'gl')

db_gl
```

```
## # Source:   table<gl> [?? x 7]
## # Database: sqlite 3.30.1
## #   [/Users/jon/Documents/R/auditanalytics/data/rauditanalytics.sqlite]
##     je_num  amount gl_date gl_date_char vendor_id account       description
##      <int>   <dbl>   <dbl> <chr>            <int> <chr>         <chr>
##  1      1   8347.   18024 2019-05-08        2691 exp_material~ Quality control ~
##  2      1  -8347.   18024 2019-05-08          NA liab_account~ Quality control ~
##  3      2  12272.   18096 2019-07-19        2669 exp_material~ Packaging and bo~
##  4      2 -12272.   18096 2019-07-19          NA liab_account~ Packaging and bo~
##  5      3   2883.   18110 2019-08-02        2276 exp_material~ Paper
##  6      3  -2883.   18110 2019-08-02          NA liab_account~ Paper
##  7      4  15971.   17907 2019-01-11        2276 exp_material~ Paper
##  8      4 -15971.   17907 2019-01-11          NA liab_account~ Paper
##  9      5   7945.   18039 2019-05-23        2691 exp_material~ Quality control ~
## 10      5  -7945.   18039 2019-05-23          NA liab_account~ Quality control ~
## # ... with more rows
```

Not only is it pointing to the same table, but its also performing the same query:

```
db_gl %>%
  show_query()
```

```
## <SQL>
## SELECT *
## FROM `gl`
```

There are a few differences though between using these approaches.

- `dbGetQuery()` will return a data.frame that is immediately usable, while creating the table connection via `tbl()` results in a preview of the query but hasn't been formally downloaded in the database. In order to use the data within R with `tbl()`, a further `collect()` is needed. Instead of performing a data download every time, it is advantageous to preview the results first before collecting it.
- `dbGetQuery` requires the entire SQL query to be pre-written, and can not be further leveraged within the database. However, the `tbl` object can still be further manipulated. This is especially useful when creating new fields or performing joins in same database.

Lets say that you wanted to filter amounts greater than $75,000 in the database. In the SQL method, you would need to type a whole new SQL query:

```
dbGetQuery(con, 'select * from gl where amount > 75000')
```

```
##   je_num   amount gl_date gl_date_char vendor_id            account
## 1     97 76842.18   18237   2019-12-07      2276 exp_materials_6000
## 2    527 97230.25   18002   2019-04-16      2691 exp_materials_6000
##                        description
## 1                            Paper
## 2 Quality control testing supplies
```

Where in the dplyr method, you would only need to build on the same connection object already established. Identical queries, and results

```
db_gl_threshold <- db_gl %>%
  dplyr::filter(amount > 75000)

db_gl_threshold
```

```
## # Source:   lazy query [?? x 7]
## # Database: sqlite 3.30.1
## #   [/Users/jon/Documents/R/auditanalytics/data/rauditanalytics.sqlite]
##   je_num amount gl_date gl_date_char vendor_id account      description
##    <int>  <dbl>   <dbl> <chr>            <int> <chr>        <chr>
## 1     97 76842.   18237 2019-12-07        2276 exp_materia~ Paper
## 2    527 97230.   18002 2019-04-16        2691 exp_materia~ Quality control tes~
```

```
show_query(db_gl_threshold)
```

```
## <SQL>
## SELECT *
## FROM `gl`
## WHERE (`amount` > 75000.0)
```

And when you're happy with this, simply `collect()` to save the query results as a data frame. To be nice to your database administrators, you should also disconnect from the database to free up a connection.

```
gl_threshold <- db_gl_threshold %>%
  collect()

dbDisconnect(con)
```

We will go through an advanced application of setting up database queries in Audit Package Creation, which builds upon leveraging dplyr to create the pieces that enable powerful queries.

5.3 APIs

As more applications are hosted on the cloud, it is an important skill to obtain information from them without resorting to manually triggered reports. Data can be accessed from these systems via a Application Programming Interface (API), and typically it is exposed via the method Representational State

Transfer (REST). An API will allow one application to talk to another. Some examples of APIs are web sites with search functions, or loading a list of comments other users have published, or determining who is friends with whom. REST advises how the endpoint is structured, and also suggests how the data is returned.

While APIs may be more complicated, the end objective is to obtain data is quite straight forward. A user needs to know what they want, then:

- match what is desired to the 'endpoint', a pre-defined url or path where the data resides,
- send a valid request to `GET` the data, and
- receive the response.

Here is an oversimplistic example. Lets say you want to see what repositories I have available to the public. You can list the repositories I have through the github api via the httr package and the `GET` command. For anything sensitive, it will likely be protected and require authentication as well.

```r
library(httr)
library(jsonlite)

response <- GET('https://api.github.com/users/jonlinca/repos')

response
```

```
## Response [https://api.github.com/users/jonlinca/repos]
##   Date: 2020-07-08 01:43
##   Status: 200
##   Content-Type: application/json; charset=utf-8
##   Size: 28.7 kB
## [
##   {
##     "id": 129155075,
##     "node_id": "MDEwOlJlcG9zaXRvcnkxMjkxNTUwNzU=",
##     "name": "ACLandR_DeployingModels",
##     "full_name": "jonlinca/ACLandR_DeployingModels",
##     "private": false,
##     "owner": {
##       "login": "jonlinca",
##       "id": 29468012,
## ...
```

The structure of the data in an API looks different than a data frame. Typically there is an overall response (indicating success or the nature of the failure), the type of content (in this case, 'application/json'), and the data itself. The entire download is usually stored as a list with multiple references:

```
names(response)
```

```
##  [1] "url"         "status_code" "headers"     "all_headers" "cookies"
##  [6] "content"     "date"        "times"       "request"     "handle"
```

And the data you want will be in `list$content`:

```
head(response$content)
```

```
## [1] 5b 0a 20 20 7b 0a
```

This raw data is generally uninterpretable, and it is because the file is in a different format - in this case, the Content-Type indicates it is a json file. Its quite easy to convert the data in this format to something more understandable with the httr `content()` and jsonlite `fromJSON()` and `toJSON()` functions:

```
content <- content(response)

cleancontent <- jsonlite::fromJSON(jsonlite::toJSON(content), flatten = TRUE)

# Print it out
cleancontent[,c('id','name')]
```

```
##          id                   name
## 1 129155075 ACLandR_DeployingModels
## 2 164948874          ACLandR_KMeans
## 3 269707027          auditanalytics
## 4 183726578                calgaryr
## 5 277917653              galvanizer
```

Some data sources may be difficult to obtain data from, or perhaps you're not quite ready at the technical skill level to develop your own data connectivity for APIs. One alternative for such information is the use of a third party tool - for example, CData supports an interface that allows you to interact with programs like Office 365 (including emails and file audit logs) directly.

Chapter 6

Explore

6.1 Dimensions

6.2 Convert between types

6.3 Break down each column

6.4 Visualize

Chapter 7

Manipulate

7.1 Basic math

7.2 Dates

7.3 Sort

7.4 Summary statistics - Outliers

7.5 Clean text

7.6 Join

Chapter 8

Test

8.1 Mutate - If statements

8.2 Summarize

8.3 Outliers

8.4 Duplicates

8.5 Age

8.6 Benford Analysis

8.7 Relative size factor testing

8.8 Search text

Chapter 9

Report

9.1 RMarkdown

9.2 Graphics

9.3 Export

Chapter 10

Applied Audit Analytics

10.1 Audit R Package creation

10.2 Continious Monitoring

10.3 Controls Automation

10.4 Audit Data Mart

10.5 All-in-one toolkits

Chapter 11

Other practices to follow

11.1 Documentation

As you start, you should promote basic habits instilled into you. Documenting your basic thought process in .R files is generally expected, and the ‘why’ a certain process outlined with comments (lines starting with a #). The why is important as it explains to code reviewers (external auditors and your peers) the rationale for your approach, or unusual quirks about the data you are transforming.

RMarkdown files become valuable as communication mediums for reports, allowing you to embed a mix of code, graphics, and interactive tables. While most of the exploratory work can be done within a basic .R file, having the ability to readily ‘knit’ a document for sharing increases the people you can share your work with.

Whether you are using R or RMarkdown files, its convenient to have these files as your primary sources of editing as you can use the keyboard shortcuts command-return or control-enter to send a command from the script file to the console.

11.2 Passwords

TODO Storing secrets - .Renviron, config, keyring

Bibliography

(2020). Tribal knowledge.

Crawford, M. B. (2009). *Shop Class As Soulcraft: An Inquiry into the Value of Work*.

Kahneman, D. (2011). *Thinking, Fast and Slow*.