Business and Data Analytics

A typical first step in analytical review of a company might be to review the industry statistics to find out what part, if any, of our client's financial accounts are outliers in the industry. Unusual account values or relationships can indicate audit risks that would require a broader scope of auditing. Exploratory statistics are easy to implement, yet are ideal for quickly highlighting unusual account values. We will first load a dataset of industry statistics that have been downloaded from the Wharton Research Data Service repository, and conduct simple summaries of the data. The R package *plotluck* automatically chooses graphs that are appropriate for the data, and is an ideal tool for quick visualization of data. We ask: What variables are contains and how they are distributed? In general, R lets the dot symbols stands for “all variables in the data set” and ".~1" regresses the variable against the intercept, thus giving the distribution (essentially an smoothed histogram)

```
library(tidyverse)
library(plotluck)
library(broom)

industry_stats <-
read.csv("/home/westland/audit_analytics_book/aaa_chapters/tables/ch_2_data
set.csv")

summary(industry_stats)

plotluck(industry_stats, .~1)
```

There are four continuous (employees, income, sales and capitalization) and one categorical (fiscal year) variable; we will get rid of the ticker. Where there is a significant range of values in a variable, plotluck automatically presents the x-axis on a log scale. Let's explore income and the influence of other variables. Output is set to 'verbose' in order to estimate conditional entropies for ordering the plot (lower values indicate stronger informational value). There is a clear correlation of income with capitalization, which we can assess by applying frequency weights.

```
industry_stats <- industry_stats[,-1]
plotluck(industry_stats, income~.)
```

How has the industry's profit changed over the recent past?

```
plotluck(industry_stats, income~fiscal_year)
```

How do sales and headcount influence income? This chart is 3-dimensional, with color providing a third dimension for assessing correlations. During analytical review, the auditor can use such analyses to find whether the client's accounts vary from industry averages.

```
plotluck(industry_stats, income~sales+employees, weights=capitalization)
```

How has this distribution changed over time?

```
plotluck(industry_stats, income~employees|fiscal_year)
plotluck(industry_stats, income~sales|fiscal_year)
plotluck(industry_stats, income~capitalization|fiscal_year)
```

Accounting Data Types: Numerical vs. Categorical

There are two basic types of structured data: numeric and categorical. Numeric data comes in two forms: continuous, such as wind speed or time duration, and discrete, such as the count of the occurrence of an event. Categorical data takes only a fixed set of values, such as a type of TV screen (plasma, LCD, LED, etc.) or a state name (Alabama, Alaska, etc.). Binary data is an important special case of categorical data that takes on only one of two values, such as 0/1, yes/no, or true/false. Another useful type of categorical data is ordinal data in which the categories are ordered; an example of this is a numerical rating (1, 2, 3, 4, or 5).

Why do we bother with a taxonomy of data types? It turns out that for the purposes of data analysis and predictive modeling, the data type is important to help determine the type of visual display, data analysis, or statistical model. In fact, data science software, such as R and Python, uses these data types to improve computational performance. More important, the data type for a variable determines how software will handle computations for that variable.

```
library(tidyverse)
library(plotluck)
library(broom)

sox_stats <-
  read.csv(

"/home/westland/audit_analytics_book/aaa_chapters/tables/ch_2_data_types.cs
v")
```

```
summary(sox_stats)
```

Continuous (interval, float, numeric) data

Many of our key financial metrics are continuous measures. Conceptually, we can subdivide continuous data without end. We can look at the density functions of continuous data using the plotluck package.

Discrete (integer, count) data

In practice, our minimum level of resolution in accounts is one dollar (or single monetary unit). This resolution limit is in fact the basis of audit procedures such as dollar-unit sampling. We are not limited to dollar levels of resolution; annual reports may use thousands or millions of dollars as their level of resolution. In this example we use ggplot to produce a histogram of discrete values. The level of resolution determines the number of bins, in this case 20. Human cognition tends to be overwhelmed by large numbers of bins, which is why we often see financial statistics summarized at levels of resolution larger than one dollar.

```
library(ggplot2)

sox_stats <-
  read.csv(

"/home/westland/audit_analytics_book/aaa_chapters/tables/ch_2_data_types.cs
v")
sox_stats[,c("audit_fee","non_audit_fee","tax_fees")] %>%
ggplot(aes(x=audit_fee)) + geom_histogram(bins=20)
```

Categorical (enums, enumerated, factors, nominal, polychotomous) data

Categorical data can take on only a specific set of values representing a set of possible categories. The firm's Chart of Accounts imposes a categorical data scheme on the economic events which are the basis of journal entries in the accounting system. In our Sarbanes-Oxley data, we might be interested in the variation over time of audit fees, categorized by the firms they were charged to.

```
sox_stats <-
  sox_stats[,c("ticker","audit_fee","non_audit_fee","tax_fees")]
sox_stats[which(as.character(sox_stats$ticker)<'AI'), ] %>%
  ggplot(aes(x=ticker, y= audit_fee)) +
  geom_violin()  +
    scale_y_continuous(trans = "log10")
```

```
library(tidyverse)
library(plotluck)
library(broom)

bank_fin <-

read.csv("/home/westland/audit_analytics_book/aaa_chapters/tables/ch_2_yaho
o_fin.csv")
bank_fin$change <-
  as.numeric(bank_fin$change)
bank_fin$capitalization <-
  as.numeric(bank_fin$capitalization )
bank_fin <-
  gather(bank_fin,
         key="metric",
         value = value,
         price,
         change,
         percent_change,
         volume,
         vol_ave,
         capitalization,
         pe_ratio)

bank_fin %>% ggplot(aes(x=metric, y=value)) +
  geom_boxplot() +
  scale_y_continuous(trans = "log10")
```

Binary (dichotomous, logical, indicator, Boolean) data

Binary data represents a special case of categorical data with just two categories. These are data's way of providing answers to *yes / no* or *true / false*. An audit opinion will provide a yes / no answer concerning whether the financial statements are fairly presented. Usually we are not just interested in the answer, but why particular circumstances, treatments or parameter settings resulted in this answer. In the following figure, we are interested in whether credit card fraud is influenced by the fees paid to the auditor. We analyze a binary variable by looking at the variation in other variables under a $0$ or $1$ value of the binary variable.

```
library(tidyverse)
library(plotluck)
library(broom)

sox_stats <-
  read.csv(

"/home/westland/audit_analytics_book/aaa_chapters/tables/ch_2_data_types.cs
v")

ggplot(sox_stats, aes(x=non_audit_fee, y=audit_fee, col=card)) +
  geom_violin() +
  labs(col = "Fraud = 1 (green)")
```

We can similarly inspect the influence of audit fees on all of the binary variables from SOX reporting in the SOX dataset.

```
library(tidyverse)
library(plotluck)
library(broom)

sox_stats <-
  read.csv(

"/home/westland/audit_analytics_book/aaa_chapters/tables/ch_2_data_types.cs
v")

sox_stats$card <-
  as.integer(sox_stats$card)

sox_stats1 <-
  gather(sox_stats,
         key="metric",
         value = value,
         effective_303,
         mat_weak_303,
         sig_def_303,
         effective_404,
         auditor_agrees_303)

ggplot(sox_stats1, aes(x=non_audit_fee, y=audit_fee, col=metric)) +
  geom_violin() +
  scale_x_continuous(trans = "log2") +
  scale_y_continuous(trans = "log2")
```

Ordinal (ordered factor) data

Ordinal data is categorical data with an explicit ordering. Ordinal data provides an important control over documents of original entry in accounting systems. When a journal entry of any sort is generated, it must be uniquely identifiable, and generally the sequence of identifying numbers are assigned in chronological sequence. In the days of paper transaction processing, accountants expected firms to initiate transactions on prenumbered forms, often with numerous copies made simultaneously through the use of (messy) carbon paper. Modern systems assign these numbers internally, but auditors still consider sequential number of input documents to be one of the more important internal controls in a system.

```
library(tidyverse)
library(lubridate)
library(kableExtra)


## create a function to generate a random date in the current year

rdate <- function(x,
                  min = paste0(format(Sys.Date(), '%Y'), '-01-01'),
                  max = paste0(format(Sys.Date(), '%Y'), '-12-31'),
                  sort = TRUE) {
  dates <- sample(seq(as.Date(min), as.Date(max), by = "day"), x, replace =
TRUE)
  if (sort == TRUE) {
    sort(dates)
  } else {
    dates
  }
}

## generate 1000 invoices with dates

invoice_no <- date <- 1:1000  ## placeholder
journal_ent_no <- cbind.data.frame(invoice_no,date)
date <- rdate(1000)
journal_ent_no$date <- date[order(date)]
journal_ent_no$invoice_no <- seq(1,1000) + rbinom(1000,1,.1) # add some
errors

duplicates <- duplicated(journal_ent_no$invoice_no)
raw <- seq(1,1000)
journal_dups <- cbind.data.frame(raw,duplicates)

ggplot(journal_dups, aes(x=invoice_no, y=raw, col=duplicates)) +
  geom_point()
```

Tables are always an option for omissions and duplications

```
library(tidyverse)
library(kableExtra)
library(reshape2)


journal_ent_no[journal_dups$duplicates==TRUE,] %>%
  kable(longtable=T, caption = "Duplicated Invoices") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"),
                full_width = F, font_size=10,) %>%
  column_spec(1, width = "3em") %>%
  column_spec(2, width = "4em")
```

But graphs provide immediate access to the degree of problem, by looking at the number of exceptions in the graph, and is most visible if exceptions are rendered in a contrasting color such as the green used in the accompanying charts. Often this is all that the auditor needs to render an opinion on internal controls, as it is the client's responsibility to correct these problems.

```
journal_dups$omit <- 0
journal_dups$invoice <- journal_ent_no$invoice_no
journal_dups[journal_dups$raw[!journal_dups$raw %in%
journal_ent_no$invoice],"omit"] <- 1
journal_dups$omit <- as.logical(journal_dups$omit)

ggplot(journal_dups, aes(x=invoice_no, y=raw, col=omit)) +
  geom_point()+
  labs(col = "Invoice Omitted?")
```

Data Storage and Retrieval

The amount of recorded data produced by human activity has probably been growing exponentially for more than a millennium. Most data today is digitized, both for archival storage as well as display. This is good for the environment (newsprint alone in the 1970s accounted for 17% of US waste) but it also means that this data is potentially available for computer processing. The current amount of digitized data is around 20 trillion GB. Much of this increase has been fueled by new structures for storing and retrieving data $-$ video, text, and vast streams of surveillance data $-$ that have arisen since the commercialization of the internet in the 1990s.

```
library(plotluck)


big_data <-
read.csv("/home/westland/audit_analytics_book/aaa_chapters/tables/ch1_amoun
t_data.csv") %>% gather("data_type", "value", -year)

## gather is a function from tidyverse that replaces columns with
indicators, and is used here to gather multiple lines on a single graph

# big_data <-  gather(big_data,  "transient", "financial", "text",
"multimedia", "mobile")

big_data$amount <- sqrt(big_data$value)

big_data %>%
 ggplot(aes(x = year, y = amount, col=data_type)) +
  geom_point(aes(color = data_type, size = amount), alpha = 0.5) +
  scale_size(range = c(0.5, 12)) +
  scale_y_continuous(trans = "sqrt") +
  xlim(1990,2020) +
  xlab("Year") +
    ylab("Number of Bits of Storage")
```

The R language uses seven main storage formats for data: vectors, matrices, arrays, data frames, tibbles (tidyverse), lists and factors.

Vectors

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("one","two","three") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector

## Refer to elements of a vector using subscripts.

a[c(2,4)] # 2nd and 4th elements of vector
```

Matrices

All columns in a matrix must have the same mode(numeric, character, etc.) and the same length. The general format is

```
vector <- seq(1, 25)
r <- 5
c <- 5
```

```
char_vector_rownames <- as.character(seq(1, 5))
char_vector_colnames <- as.character(seq(1, 5))

mat <- matrix(vector, nrow=r, ncol=c, byrow=FALSE,
  dimnames=list(char_vector_rownames, char_vector_colnames))

mat

mat <- matrix(vector, nrow=r, ncol=c, byrow=TRUE,
  dimnames=list(char_vector_rownames, char_vector_colnames))

mat

# byrow=TRUE indicates that the matrix should be filled by rows.
# byrow=FALSE indicates that the matrix should be filled by columns (the
default).
# dimnames provides optional labels for the columns and rows.

# generate 5 x 4 numeric matrix
y <- matrix(1:20, nrow=5,ncol=4)

# another example
cells <- c(1,26,24,68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
  dimnames=list(rnames, cnames))

# Identify rows, columns or elements using subscripts.

mat[, 4] # 4th column of matrix
mat[3,] # 3rd row of matrix
mat[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

Arrays

Arrays are similar to matrices but can have more than two dimensions.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames =
list(row.names,column.names,
   matrix.names))
print(result)
```

Data Frames, Data Tables and Tibbles

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.) similar to SAS and SPSS datasets. Data frames represent the most widely used format for data storage and retrieval in the R language

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed") # variable names

# There are a variety of ways to identify the elements of a data frame .

mydata[1:2] # columns 1 and 2 of data frame
mydata[c("ID","Color")] # columns ID and Age from data frame
mydata$X1 # variable x1 in the data frame
```

Tibbles are the tidyverse's improvements on data frames, designed to address problems in data analysis at an earlier point. Otherwise they are used exactly as data frames.

In R, the basic rectangular data structure is a data.frame. A data.frame also has an implicit integer index based on the row order. While a custom key can be created through the row.names attribute, the native R data.frame does not support user-specified or multilevel indexes. To overcome this deficiency, two new packages are gaining widespread use: data.table and dplyr. Both support multilevel indexes and offer significant speedups in working with a data.frame.

Lists

An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name.

```
a <- b <- "seven"
z <- y <- 0

# example of a list with 4 components -
# a string, a numeric vector, a matrix, and a scaler
w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)

# example of a list containing two lists
list1 <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)
list2 <- list(name="Joe", mynumbers=b, mymatrix=z, age=10,11)
```

```
v <- c(list1,list2)

## Identify elements of a list using the [[]] convention.

w[[1]] # 1st component of the list
v[["mynumbers"]] # component named mynumbers in list
```

Factors

Tell R that a variable is nominal by making it a factor. The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.

```
# variable gender with 20 "male" entries and
# 30 "female" entries
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
# R now treats gender as a nominal variable

summary(gender)

# An ordered factor is used to represent an ordinal variable.

# variable rating coded as "large", "medium", "small'

rating <- c( "medium","large", "small")
rating <- ordered(rating)
rating
# recodes rating to 1,2,3 and associates
# 1=large, 2=medium, 3=small internally
# R now treats rating as ordinal
```

R treats factors as nominal (i.e., label or naming) variables and ordered factors as ordinal variables in statistical procedures and graphical analyses. You can use options in the factor( ) and ordered( ) functions to control the mapping of integers to strings (overriding the alphabetical ordering). You can also use factors to create value labels.

Useful Functions for Dataset Inspection and Manipulation

```
library(kableExtra)

arry <-
read.csv("~/audit_analytics_book/aaa_chapters_FINAL/tables/morph_array.csv"
)

arry %>%
  kable(longtable=T) %>%
  kable_styling(
    bootstrap_options = c("striped", "hover", "condensed"),
    full_width = F,
    font_size=10)

length(arry) # number of elements or components
str(arry)    # structure of an object
class(arry)  # class or type of an object
names(arry)  # names

list1 <- list("a","b","c")

c(list1, list1)       # combine objects into a vector
cbind(list1, list1) # combine objects as columns
rbind(list1, list1) # combine objects as rows

list1     # prints the object
newobject <- edit(list1) # edit copy and save as newobject
fix(list1)               # edit in place

ls()       # list current objects
rm(list1) # delete an object
```

R packages required for this chapter

This chapter's code requires the following packages to be installed: `tidyverse`, `kableExtra`, `plotluck`, `broom`, `ggplot2`, `lubridate`and `reshape2`.

Note: There are two steps to using a package. First it must be *installed*, i.e., copied to a location on your computer where R can access it. Then it must be *loaded* into the working memory of *R*. To install, for example the `tidyverse` package, type *install.packages("tidyverse")* and then press the *Enter/Return* key. To load the previously installed package type *library(tidyverse)*. The `tidyverse` package is now available for use by your program code.