```
knitr::opts_chunk$set(echo = TRUE)
```

Substantive Tests: Exploratory substantive tests

The first steps in substantive testing in an audit is to compare client's transaction files to the Trial Balance. Traditionally this was accomplished by 'footing' (i.e., calculating a total, perhaps using a vintage 10-key hand-crank adding machine) and 'agree' (i.e., checking whether the total agrees with the Trial Balance account comprised by the transactions being totaled). The auditor has more sophisticated tools available today. This section explores particular tools that might be used to complete the exploratory analysis of clients transactions files.

As an example, consider the sales transactions generated by the code in chapter 12. Sales will be the largest entry on the trial balance, thus a thorough exploration of the transactions generating this balance is demanded by the audit.

```
library(broom)
library(readr)
library(tidyverse)
library(compare)


## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}


real_world_cash_sales <-
  read.csv("real_world_cash_sales.csv", na.strings="0",
stringsAsFactors=FALSE)

real_world_credit_sales  <-
  read.csv("real_world_credit_sales.csv", na.strings="0",
stringsAsFactors=FALSE)

real_world_sales <-
  rbind(real_world_cash_sales,real_world_credit_sales)

sales_journal <-
  read.csv("sales_journal.csv", na.strings="0", stringsAsFactors=FALSE)
```

```
credit_sales_journal <-
  sales_journal %>%
  filter(cash_not_ar == 0)
```

How many records are there? What fields exist? Of which type? Is there missing data? Is the data in a reasonable range? What sort of distribution does it have? Running a summary statistical function over the whole dataset is a great first step to understanding what you have, and whether it is valid. A simple `summary` tool exists in base R (in the following code chunk, which is not evaluated, but left for the reader to explore).

```
summary(credit_sales_journal)
```

This highlights which fields have missing data. But it doesn't show the overall count of records, nor is there a standard deviation, nor does it reveal groupings that might be of use in stratification or error audits. Nor is it easy to convert this to an auditable report for the working papers.

Fortunately, several R functions provide more useful summarization tools for the auditor. Three of these will be discussed here, with examples of each for the sales transactions, and differing applications for reasons that I will articulate.

- describe, from the *Hmisc* package
- describe from *psych*
- skim from *skimr*

Here are comparisons of output from these different summarization tools that may or may not meet the specific needs of an audit or test. The output of `Hmisc`'s `describe` is significantly more informative for dates, ommissions and duplicated transactions than the other two packages, but the output tends to be rather crude, and is left for the reader to explore. Here is a code chunk to, e.g., `describe` the credit sales journal.

```
library(Hmisc)
Hmisc::describe(credit_sales_journal)
```

The *psych* describe function reports all the numeric and categorical fields in its output, but the categorical fields are recoded as numbers, and summarized in a fashion normally associated with numeric fields. For numeric fields, the function reports a plethora of statistics: count, mean, standard deviation, median, trimmed mean (removing outliers), mean absolute deviation, min, max, range, skewness, kurtosis and standard error. If any of these are meaningful for the auditor, this package will generate them for each numeric field in the dataset. Note that accounting transactions will often be right-skewed and kurtotic, and

this function allows the auditor to investigate how far the transaction distribution differs from a Normal distribution.

```
library(kableExtra)
library(psych)
psych::describe(credit_sales_journal) %>%
 kable("latex", booktabs = T) %>%
  kable_styling(
    bootstrap_options = "striped",
    full_width = F,
    latex_options="scale_down")
```

Another useful feature is the availability of statistics for specific summary-by-group variation. In the following example, the auditors wants to see sales statistics by customer.

```
library(psych)
head(psych::describeBy(credit_sales_journal,
credit_sales_journal$customer_no) )
```

The 'mat=TRUE' parameter can be added to produce output in matrix format, which can be used for creating tables and output for the workpapers. The `tidy` function of `broom` allows saving the summary information in a table for later use in the audit.

The skim from `skimr` function provides an informative and compact summary, which can be saved as tables or reports in the working papers. I make the assumption that working papers are likely retained in computer format, even if there are separate physical documents retained. The output works with `kable` and `tidyverse` packages like `dplyr`. The histograms of field values can be useful in determining sample size and distributions that should be considered for tests. The `skimr` package works well with tidyverse packages.

```
library(tidyverse)
library(skimr)

skim_with(numeric = list(hist = NULL))
sales_rpt <-
  group_by(credit_sales_journal, customer_no) %>%
  skim()

head(sales_rpt)
```

Creating Trial Balance Figures in One Step

Given the full set of transaction files that will be used in the audit, it should be possible to generate an independent auditor's set of trial balance figures to be compared to the client's. The following code shows how this might look for the transaction files generated in chapter 12.

```
library(tidyverse)
library(broom)
library(kableExtra)


## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}


sales_journal <-
  read_csv("sales_journal.csv",
    col_types = cols(X1 = col_skip()))

purchase_journal <-
  read_csv("purchase_journal.csv",
    col_types = cols(X1 = col_skip()))

perpetual_inventory_ledger <-
  read_csv("perpetual_inventory_ledger",
    col_types = cols(X1 = col_skip()))

fyear_end_ar_ledger <-
  read_csv("fyear_end_ar_ledger.csv",
    col_types = cols(X1 = col_skip()))

fyear_begin_inventory_ledger <-
  read_csv("fyear_begin_inventory_ledger.csv",
    col_types = cols(X1 = col_skip()))

expenditures <-
  read_csv("expenditures.csv",
    col_types = cols(X1 = col_skip()))

disbursement_journal <-
```

```
  read_csv("disbursement_journal.csv",
    col_types = cols(X1 = col_skip()))

deposit_daily <-
  read_csv("deposit_daily.csv",
    col_types = cols(X1 = col_skip()))

daily_ar_balance <-
  read_csv("daily_ar_balance.csv",
    col_types = cols(X1 = col_skip()))

collections_journal <-
  read_csv("collections_journal.csv",
    col_types = cols(X1 = col_skip()))

Sales <-
  sum(sales_journal$sales_count *
        sales_journal$sales_unit_price)

Sales_Returns <-
  sum(sales_journal$sales_return *
     sales_journal$sales_unit_price)

Purchases <-
  sum(purchase_journal$po_count *
        purchase_journal$unit_cost)

Vendor_Disbursements <-
  sum(disbursement_journal$unit_cost *
disbursement_journal$no_units_ordered)

YE_Inventory <-
  perpetual_inventory_ledger %>%
  group_by(sku) %>%
  inner_join(sales_journal, by="sku") %>%
  slice(n()) %>%
  mutate(inv_extended = unit_cost * stock_on_hand) %>%
  ungroup() %>%
  select(inv_extended) %>%
  sum()

Accounts_Receivable <- sum(fyear_end_ar_ledger$amount)

Accounts_R2 <-
  max(daily_ar_balance$ar_balance)


Begin_Inventory <-
  fyear_begin_inventory_ledger %>%
  group_by(sku) %>%
  mutate(inv_extended = unit_cost * stock_on_hand) %>%
  ungroup() %>%
  select(inv_extended) %>%
  sum()
```

```
Cost_of_Goods_Sold <-
    sum(sales_journal$sales_count *
        sales_journal$unit_cost) +
  YE_Inventory -
  Begin_Inventory


Misc_Expenses <-
  sum(expenditures$amount)

Collections_on_AR <-
  collections_journal %>%
  group_by(invoice_no) %>%
  inner_join(sales_journal, by="sku")%>%
  filter(cash_not_ar.x==0) %>%
  slice(n())

Uncollected_AR <-
  sum(Collections_on_AR$sales_extended.x) -
  sum(Collections_on_AR$collection_amount.x)

Collections_on_AR <-
  sum(Collections_on_AR$collection_amount.x)



Cash_Deposits <-
  sum(deposit_daily$deposit_amount)

Cash_in_Bank <-
  Cash_Deposits -
  Purchases -
  Misc_Expenses


Change_in_Equity <-
  Sales_Returns +
  Collections_on_AR+
  Purchases+
  Cost_of_Goods_Sold+
  Cash_in_Bank+
  Accounts_Receivable+
  YE_Inventory+
  Misc_Expenses-
  Sales-
  Uncollected_AR


## construct the trial balance and format this with knitr::kable and
kableExtra

data.frame(
  "Account"=c("Sales",
```

```
                "Sales Returns",
                "Collections_on_AR",
                "Uncollected_AR",
                "Purchases",
                "Cost_of_Goods_Sold",
                "Cash_in_Bank (change from start of yr)",
                "Accounts_Receivable",
                "YE_Inventory",
                "Misc_Expenses",
                "Change_in_Equity",
                "Total"
                ),


    "Dr"= c(0,
            Sales_Returns,
            Collections_on_AR,
            0,
            Purchases,
            Cost_of_Goods_Sold,
            Cash_in_Bank,
            Accounts_Receivable,
            YE_Inventory,
            Misc_Expenses,
            0,
            Sales_Returns+
              Collections_on_AR+
              Purchases+
              Cost_of_Goods_Sold+
              Cash_in_Bank+
              Accounts_Receivable+
              YE_Inventory+
              Misc_Expenses
            ),
    "Cr"= c(    Sales,
                0,
                0,
                Uncollected_AR,
                0,
                0,
                0,
                0,
                0,
                0,
                Sales_Returns +
                  Collections_on_AR+
                  Purchases+
                  Cost_of_Goods_Sold+
                  Cash_in_Bank+
                  Accounts_Receivable+
                  YE_Inventory+
                  Misc_Expenses-
                  Sales-
                  Uncollected_AR,
```

```
               Sales+
                 Uncollected_AR+
                 Change_in_Equity
               )
) %>%
  mutate_if(is.numeric, format, digits=4,nsmall = 2, big.mark=",") %>%
  kable(., align='r', caption = "Trial Balance", "latex", booktabs = T) %>%
  kable_styling(full_width = F)
```

Accounts Receivable Auditing

Footing and Agreeing to the Trial Balance

'Foot' means total the balance of the outstanding accounts receivable at year end. 'Agree' means that this balance should be traced to the trial balance. If these totals do not match, search for a journal entry that is incorrect. Investigate reconciling items. If the client has journal entries in the accounts receivable account in the general ledger, the auditors will likely want to review the justification for the larger amounts. This means that these journal entries should be fully documented.

Load the relevant clients files into the R workspace.

```
rm(list=ls())
library(readr)


## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}

# the real_world_ar_ledger file
# also contains the results of confirmation;
# the auditor takes a sample of client_ar_ledger records,
# and auditing obtains information from the real world
# which is simulated here by
# left_join(client_ar_ledger[{sample},], real_world_ar_ledger)

real_world_fyear_end_ar_ledger <-
  read.csv("real_world_fyear_end_ar_ledger.csv", na.strings="0",
           stringsAsFactors=FALSE)

fyear_end_ar_ledger <-
```

```
  read.csv("fyear_end_ar_ledger.csv", na.strings="0",
           stringsAsFactors=FALSE)

customer_credit_limits <-
  read.csv("customer_credit_limits.csv", na.strings="0",
           stringsAsFactors=FALSE)

sales_journal <-
  read.csv("sales_journal.csv", na.strings="0",
           stringsAsFactors=FALSE)

real_world_credit_sales <-
  read.csv("real_world_credit_sales.csv", na.strings="0",
           stringsAsFactors=FALSE)

real_world_cash_sales <-
  read.csv("real_world_cash_sales.csv", na.strings="0",
           stringsAsFactors=FALSE)
```

Foot the client_ar_ledger file and agree the total to the Accounts Receivable entry on the Trial Balance. Compute the Accounts Receivable balances by customer in preparation for confirmations.

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

# set the fiscal year end date to match the dataset
fyear_end <- paste0(format(Sys.Date(), '%Y'), '-12-31')


## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}

## create the unpaid_ar
sales_journal <-
  read.csv("sales_journal.csv", na.strings="0", stringsAsFactors=FALSE)

sales_journal[is.na(sales_journal)] <-  0

credit_sales_journal <-
  sales_journal %>%
  filter(cash_not_ar == 0)
```

```
unpaid_ar <-
  credit_sales_journal[
    credit_sales_journal$collection_date >= fyear_end &
      credit_sales_journal$shipper_date <= fyear_end,]

foot <-
  unpaid_ar %>%
  select(sales_extended) %>%
  sum()


cat("\n\n A/R balance = ",foot)

# create a list of unpaid AR by customer

unpaid_ar_by_cust <-
  unpaid_ar %>%
  select(customer_no, invoice_date, invoice_no, sales_extended) %>%
  group_by(customer_no) %>%
  select(sales_extended) %>%
  summarise(customer_ye_balance=sum(sales_extended))

unpaid_ar_by_cust %>%
  kable(caption="Year End A/R by Customer",
        "latex", booktabs = T) %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed"),
full_width = F)
```

Tests of Supporting Evidence

Sample invoices from the accounts receivable aging report and compare them to supporting documentation to see if they were billed in the correct amounts, to the correct customers, and on the correct dates. Trace invoices to shipping log. The auditors will match invoice dates to the shipment dates for those items in the shipping log, to see if sales are being recorded in the correct accounting period. This can include an examination of invoices issued after the period being audited, to see if they should have been included in a prior period. We compute these in the following code chunk

```
library(readr)
library(pwr)
library(tidyverse)


size <- as.numeric(nrow(credit_sales_journal))
## data set size
Delta <- .05*size
## detect 5% occurrence error
sigma <- .3*size
```

```
## variability (guess ~1/3 rd)
effect <- Delta/sigma



sample <- pwr.t.test(
  d=effect,
  sig.level = 0.05,
  power = .8,
  type="one.sample",
  alternative="greater")                      ## look for overstatement of
earnings


cat("\n \n Attribute sample size for occurrence of error = ",
ceiling(sample$n))



size <-
  as.numeric(
    sum(
      credit_sales_journal$sales_unit_price*
        credit_sales_journal$sales_count))
## data set size
mu <-
  mean(
    credit_sales_journal$sales_unit_price*
      credit_sales_journal$sales_count)
## average value of transaction
Delta <- .05*mu
## detect 5% amount intolerable error
sigma <-
  sd(credit_sales_journal$sales_unit_price*
       credit_sales_journal$sales_count)
## variability
effect <- Delta/sigma


sample <- pwr.t.test(
  d=effect,
  sig.level = 0.05,
  power = .8,
  type="one.sample",
  alternative="greater")   ## look for sales value too large

cat("\n Attribute sample size for amount of error = ", ceiling(sample$n))
```

Acceptance Sampling

Acceptance sampling for substantive tests results in a decision of whether or not to 'accept' the account as either (1) being fairly presented, or (2) containing material (intolerable) error. Errors estimates from acceptance samples are in *amounts* which may be viewed on a continuous scale of value for statistical tests, or as discrete *dollar-unit* amounts for dollar (monetary) unit sampling. If it is found that a particular account balance is materially in error, the most common step is to present this to the client, and request that accounts be adjusted to fairly present that client firm's financial position.

Note that parameter inputs that would yield a sample size of less than 1.3 from the function below stop calculations and result in an `error in uniroot function` termination of the `pwr.t.test` function. In this situation, the software is essentially telling you not to take any samples, and skip this test. This is probably not advisable from a risk perspective. The alternate conclusion would be that this implies one or more of the particular settings chosen for (1) data set size, (2) detection occurrence rate, (3) variability or (4) effect size do not make sense. They should be revised, generally in the following order: (1) variability, (2) detection occurrence rate, (3) effect size, with in interim testing will be the *intolerable* or *out-of-control* error rate.

```
library(readr)
library(pwr)

## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}

Client_ar_ledger <- read_csv("fyear_end_ar_ledger.csv", col_types = cols(X1
= col_skip()))


## data set size
size <-  as.numeric(nrow(Client_ar_ledger))

## average value of transaction
mu  <-   mean(Client_ar_ledger$amount)

Delta <- .1*mu                                  # detect 10% (material)
error in sample

sigma <- sd(Client_ar_ledger$amount)    # variability

effect <- Delta/sigma



sample <- pwr.t.test(
  d=effect,
```

```
  sig.level = 0.05,
  power = .8,
  type="one.sample",
  alternative="greater")   ## look for AR value too large

cat("\n Acceptance sample size = ", ceiling(sample$n))
```

The sample size then can be used in the following code chunk example to test for errors in the account. Assume that the sample size is 15, and our external documents are

```
library(broom)
library(readr)
library(tidyverse)
library(compare)

## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}


Client_ar_ledger <- read_csv("fyear_end_ar_ledger.csv", col_types = cols(X1
= col_skip()))

real_world_ar <-
  read_csv("real_world_fyear_end_ar_ledger.csv",
  col_types = cols(X1 = col_skip()))

acceptance_sample_ar <-
Client_ar_ledger[runif(15,1,nrow(Client_ar_ledger)),]

audited_sample <-
  inner_join(
    acceptance_sample_ar,
    real_world_ar,
    by="invoice_no") %>%
  select(
    invoice_no,
    customer_no.x,
    amount.x,
    confirm_exception,
    confirm_response)
```

```
exceptions <- audited_sample[audited_sample$confirm_exception == 1,]

occ_error_rate_ar <- nrow(exceptions) / nrow(acceptance_sample_ar)
amt_error_rate_ar <- sum(exceptions$amount.x) /
sum(acceptance_sample_ar$amount)

cat("\n\n Occurrence error in Accounts Receivable on the Trial Balance ",
occ_error_rate_ar)
cat("\n\n Amount error in Accounts Receivable on the Trial Balance ",
amt_error_rate_ar)
```

Accounts Receivable Confirmation

```
library(pwr)
library(tidyverse)
library(kableExtra)

intolerable_error <- 5000


## Effect size (Cohen's d):
## difference between the means from H0 & Ha divided by the pooled standard
deviation
cohen_d <- intolerable_error / sd(unpaid_ar_by_cust$customer_ye_balance)

sample_size <- pwr.t.test(
  d=cohen_d,
  sig.level = 0.05,
  power = .8,
  type="one.sample",
  alternative="greater") ## look for overstatement of AR = assets
overstated

ar_sample <- sample_n(unpaid_ar_by_cust, ceiling(sample_size$n), replace=T)


kable(ar_sample,
      caption = "Customer Balances for Sampling and Confirmation",
      "latex",
      booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")
```

Confirmation and Estimation of Error in Account

Once the auditor has completed the confirmation and follow-up for confirmation of accounts receivable, the sample results can be used to compute an upper bound on the accounts receivable balance. Were the

client to post a balance in excess of this upper bound, the accounts receivable balance could be considered intolerably (materially) in error.

```
library(tidyverse)
library(lubridate)

default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}

real_world_credit_sales <-
  read.csv("real_world_credit_sales.csv", na.strings="0",
stringsAsFactors=FALSE)
real_world_fyear_end_ar_ledger <-
  read_csv("real_world_fyear_end_ar_ledger.csv",
    col_types = cols(X1 = col_skip()))

## create real_world_unpaid_ar & real_world_ar_by_customer
## note that these 'real_world' values are not available to the auditor
## the auditor extracts information about them
## from a sample of obtained after the A/R audit confirmation process

real_world_unpaid_ar <-
  real_world_credit_sales[
    real_world_credit_sales$collection_date >= fyear_end &
      real_world_credit_sales$shipper_date <= fyear_end,]


real_world_ar_by_customer <-
  real_world_unpaid_ar %>%
  group_by(customer_no)


real_world_ar_by_customer$sales_extended <-
  real_world_ar_by_customer$sales_count *
  real_world_ar_by_customer$sales_unit_price

real_world_ar_by_customer %>%
    group_by(customer_no) %>%
    summarize(customer_ye_balance = sum(sales_extended))


## use the clients transaction files to compute the sample as a % of
population

post_confirm_ar_balances <-
```

```
  left_join(ar_sample,real_world_fyear_end_ar_ledger, by="customer_no")

post_confirm_ar_balances[is.na(post_confirm_ar_balances)] <- 0
post_confirm_ar_balances <-
  post_confirm_ar_balances %>%
  mutate(balance_per_customer = customer_ye_balance * confirm_pct) %>%
  select(customer_no, customer_ye_balance, balance_per_customer) %>%
  mutate(overstatement=customer_ye_balance - balance_per_customer)

mean_trans_error <- mean(post_confirm_ar_balances$overstatement)
sd_trans_error <- sd(post_confirm_ar_balances$overstatement)
population_error_est_95_ub <-
  nrow(fyear_end_ar_ledger) *
  1.96 * sd_trans_error +
  mean_trans_error

cat("\n\n Population error estimate 95% upper conf bound = ",
population_error_est_95_ub)
```

```
Population error estimate 95% upper conf bound =  820351.3
```

Post Confirmation Tests

After confirmations are received, and secondary follow-up procedures are pursued to confirm the balances, the following code chunk presents the results.

```
library(kableExtra)


ar_working_papers <-
  full_join(stratum_est,real_world_fyear_end_ar_ledger, by="customer_no")

kable(ar_working_papers, caption = "Audit Workpapers for A/R
Confirmation","latex", booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")
```

Post-confirmation the figure of interest is the error in the population (and whether or not it is material). I investigate the performance of stratified and PPS sampling to estimate populations error, starting with a file of errors in the confirmed A/R balances. Clearly, this will not be available to the auditor, rather only a sample of it will be. I provide it to benchmark the performance of a 3-stratum and PPS approach to sampling. I calculate overstatements of the A/R account because the Conservatism principle dictates that I want to find errors that overstate income (thus A/R balance overstated).

The R *survey* package supports PPS (i.e.,, arbitrary unequal probability) sampling with replacement, or using the with-replacement single-stage approximation to a multistage design. No special notation is required: just specify the correct sampling weights. To specify a PPS design, the sampling probabilities must be given in the prob argument of `svydesign`, or in the `fpc` argument, with prob and weight unspecified. In addition, it is necessary to specify which PPS computation should be used, with the `pps` argument. The optional variance argument species the Horvitz-Thompson (variance="HT") or Yates-Grundy (variance="YG") estimator, with the default being "HT".

```
library(TeachingSampling)
library(tidyverse)
library(broom)
library(knitr)
library(kableExtra)
library(survey)

unpaid_ar <-
  sales_journal[1:15] %>%
  filter(cash_not_ar == 0 &
         collection_date >= fyear_end &
         shipper_date <= fyear_end) %>%
  select(-X)

confirmed_unpaid_ar <-
  real_world_credit_sales %>%
  filter(collection_date >= fyear_end &
         shipper_date <= fyear_end) %>%
  select(-X) %>%
  select(customer_no,
         invoice_no,
         invoice_date,
         sales_count,
         sales_return,
         sales_unit_price) %>%
  mutate(real_sales = sales_unit_price * (sales_count-sales_return)) %>%
  left_join(unpaid_ar, by="invoice_no") %>%
  mutate(overstatement = sales_extended - real_sales) %>%
  select(customer_no =customer_no.x,
         invoice_no = invoice_no,
         invoice_date = invoice_date.x,
         overstatement)
confirmed_unpaid_ar[is.na(confirmed_unpaid_ar)] <- 0



## Separate the dataset into three strata and set the sample proportions
confirmed_unpaid_ar$stratum <-
  cut(confirmed_unpaid_ar$overstatement, breaks=3, labels=c("small",
"medium", "large"))

sample_prop <- data.frame(prop = c(.05,.1,.2), stratum = c("small",
"medium", "large"))
```

```
# Define the size of each stratum
N <- NULL
for(i in 1:3) N[i] <- summary(confirmed_unpaid_ar$stratum)[[i]]
Nh <- as.numeric(N[1:3])                          ## vector of stratum
sizes
nh = as.numeric(ceiling(N * sample_prop$prop))    ## vector of samples size
by stratum

# Draw a stratified sample and estimate
samp <- S.STSI(confirmed_unpaid_ar$overstatement, Nh, nh)
ar_sample <- confirmed_unpaid_ar[samp,]

est <- data.frame(confirmed_unpaid_ar$overstatement)
estimate <- E.STSI(confirmed_unpaid_ar$stratum, Nh, nh, est )

## normalize the estimate to reflect the number of transactions in the
client's file
normalized_estimate <-  nrow(confirmed_unpaid_ar) *
estimate[,,2]/estimate[1,,1]
normalized_estimate <- round(normalized_estimate, digits=0)

tidy(normalized_estimate) %>%
kable(caption = "3-Stratum Horvitz–Thompson estimate of Total Error in A/R
Balance") %>%
  kable_styling(bootstrap_options = "striped")
```

```
# Draw a PPS sample of size 'size' and estimate draw a random sample
according to a
# PPS with replacement design
size <- 5
sample_prop <- S.PPS(size,confirmed_unpaid_ar$overstatement)
# returns a matrix of m rows and two columns.
# Each element of the first column indicates the unit that was selected.
# Each element of the second column indicates the selection probability of
this unit

sample_ar <- confirmed_unpaid_ar[sample_prop[,1],]

est <- data.frame(confirmed_unpaid_ar$overstatement)
estimate <- E.PPS(est,sample_prop[,2])


## normalize the estimate to reflect the number of transactions in the
client's file
normalized_estimate <-  nrow(unpaid_ar) * estimate[,2]/estimate[1,1]
normalized_estimate <- round(normalized_estimate, digits=0)
```

```
tidy(normalized_estimate) %>%
kable(caption = "PPS Hansen-Hurwitz Estimate of Overstatement of A/R
account balance","latex", booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")

actual_overstatement <- sum(confirmed_unpaid_ar$overstatement)
cat("\n\n Actual Overstatement of A/R balance in Population =
",actual_overstatement)
```

Estimation and Accrual of the Allowance for Doubtful Accounts

The Allowance for Doubtful (uncollectable) Accounts is a contra-account, i.e., it offsets the accounts receivable account balance to calculate the net revenues that have been earned. Some of the accounts receivable at year-end will not be paid, but we don't know which of these will not be paid (otherwise, the client probably would not have sold to this customer). These are expenses which must be reported as a contra-account (Cr balance) to the accounts receivable account (Dr balance). The estimation policy that the client applies will be based on historical payments and unpaid debt write-offs.

The following code chunks present examples of predictive algorithms to estimate the future collection of accounts receivable. Though there are various approaches to estimating when customers will pay their accounts, if at all. I adopt a 'time to payment' estimate, where uncollectable accounts will theoretically have an infinite time to payment. At year end, the longer an outstanding receivable has remained unpaid, the more likely we are to decide that the time to payment will be infinite. Just as with traditional methods for estimating the Allowance for Uncollectable A/R, I apply a particular decision model for estimating uncollectables based on the predicted aging of A/R

```
library(tidyverse)
library(ggplot2)
library(lubridate)

sales_journal[is.na(sales_journal)] <- 0

credit_sales_journal <-
  sales_journal[,1:15] %>%
  filter(cash_not_ar == 0) %>%    ## credit only
  mutate(collection_delay =
           .01 + time_length(        ## add .01 to get rid of 0's so logs
can be computed
             ymd(collection_date) -
             ymd(invoice_date),
             unit = "day"),
         customer_no =
           as.character(customer_no)
         ) %>%
  as.data.frame()


## Simple OLS regression
```

```
l_mod <-
  lm(collection_delay ~
        customer_no +
        unit_cost +
        sales_unit_price +
        sales_count +
        sales_return,
        data=credit_sales_journal)

summary(l_mod)


sq <- seq(1,length(l_mod$residuals))
res <- data.frame(sq,l_mod$residuals)

ggplot(res, aes(sample=l_mod$residuals)) +
  stat_qq() +
  stat_qq_line()+
  labs(title = "Linear Model of Time to Payment, Residuals QQ Plot")
```

GLM-Exponential Regression

Time to payment can be better modeled using an exponential distribution, which is commonly used to model waiting times for Poisson processes. The exponential distribution is a specific Gamma distribution where the dispersion parameter (which is what distinguishes an exponential distribution from the more general Gamma distribution) does not affect the parameter estimates in a generalized linear model, only the standard errors of the parameters/confidence intervals/p-values etc. The Gamma family is parametrized in glm() by two parameters: mean and dispersion; the "dispersion" regulates the shape. Therefore fit a GLM with the Gamma family, and then produce a "summary" with dispersion parameter set equal to 1, since this value corresponds to the exponential distribution in the Gamma family.

```
exp_mod <-
  glm(collection_delay ~
        customer_no +
        unit_cost +
        sales_unit_price +
        sales_count +
        sales_return,
        data=credit_sales_journal,
      family = Gamma(link = "inverse")
  )

sq <- seq(1,length(exp_mod$residuals))
res <- data.frame(sq,exp_mod$residuals)

ggplot(res, aes(sample=exp_mod$residuals)) +
```

```
  stat_qq() +
  stat_qq_line() +
  labs(title = "GLM Model of Time to Payment, Residuals QQ Plot")
```

Reviewing the Q-Q plot of residuals for the GLM model with Gaussian fit and an inverse link shows that it tends to overcorrect for the misfit in the linear model. Nonetheless, fit improves on a simple linear model. How well does the GLM model fit the collection timeseries? The following code chunk compares our predicted collections to actual collections. There is substantially more variance in the actual transaction flows than the predicted flows (which is to be expected), but the predictions are centered at the same place as the actual collections, and over time, this should be sufficient to accurately estimate the allowance for doubtful accounts.

```
## how well does the glm model predict?


predictors <-
  credit_sales_journal %>%
  select(customer_no,
         unit_cost,
         sales_unit_price,
         sales_count,
         sales_return
          )

pred_collection_delay <-
  1 / predict(exp_mod, dispersion=1, predictors)

plt <-
  credit_sales_journal %>%
  select(invoice_date, collection_delay) %>%
  cbind(pred_collection_delay)

ggplot() +
  geom_line(data = plt, aes(x = invoice_date, y = collection_delay), color
= "blue") +
  geom_line(data = plt, aes(x = invoice_date, y = pred_collection_delay),
color = "red") +
  xlab('invoice_date') +
  ylab('delay (blue) vs. predicted (red)')
```

Time series forecasting

There are many methods for forecasting time series which can be broadly categorized as either (1) econometric methods, such as ARIMA; (2) signal processing methods, such as Fourier Analysis; or (3) machine learning methods which involve recurrent neural networks (such as LSTM networks). In the current section, I demonstrate some standard ARIMA models which improve on existing methods for accruing the

Allowance for Uncollectable A/R. Machine learning models show promise in more completely extracting information from client records. Forecasts from machine learning, or signal processing models can be used in the same fashion that is demonstrated here in the application of econometric models.

If we assume that there are time dependent effects that influence the rate of payment, we could actually model payments history as an autoregressive integrated moving average (ARIMA) using R's time series tools, as shown in the code chunk below.

```
library(ggplot2)
library(lubridate)
library(forecast)
library(tidyquant)
library(timetk)
library(sweep)
library(tidyverse)

sales_journal[is.na(sales_journal)] <- 0

## create a time-series object

credit_sales <-
  sales_journal[,1:15] %>%
  filter(cash_not_ar == 0) %>%    ## credit only
  mutate(collection_delay =
             time_length(
             ymd(collection_date) -
             ymd(invoice_date),
             unit = "day"),
         customer_no =
           as.character(customer_no)
         ) %>%
    as.data.frame() %>%
    group_by(collection_delay) %>%
    summarise_if(is.numeric, sum, na.rm = TRUE) %>%
    select(collection_amount) %>%
    ts(frequency=365, start = c(2019,1))


moving_sales <- ma(credit_sales , order=30) ## 1 month = 30 day moving
average
#plot(moving_sales)

plot(moving_sales,
  xlab="Time to Pay",
  ylab="Amount Collected")

## Fit and forecast with auto.arima()
## The forecast package offers auto.arima() function to fit ARIMA models.
It can also be manually fit using Arima(). A caveat with ARIMA models in R
is that it does not have the functionality to fit long seasonality of more
than 350 periods eg: 365 days for daily data or 24 hours for 15 sec data.
```

```
autoArimaFit <- auto.arima(credit_sales)
#plot(forecast(autoArimaFit, h=20))

plot(forecast(autoArimaFit, h=20),
  xlab="Time to Pay",
  ylab="Amount Collected")


sales_journal[is.na(sales_journal)] <- 0

## create a time-series object

credit_sales <-
  sales_journal[,1:15] %>%
  filter(cash_not_ar == 0) %>%    ## credit only
  mutate(collection_delay =
             time_length(
             ymd(collection_date) -
             ymd(invoice_date),
             unit = "day"),
         customer_no =
           as.character(customer_no)
         ) %>%
    as.data.frame() %>%
    group_by(customer_no, collection_delay) %>%
    summarise(collections = sum(collection_amount)) %>%
  as.tibble()

credit_sales %>%
    ggplot(aes(x = collection_delay, y = collections, group = customer_no))
+
    geom_area(aes(fill = customer_no), position = "stack") +
    labs(title = "Customer Payment Performance", x = "Days to Pay", y =
"Sales") +
    scale_y_continuous() +
    theme_tq()
```

Forecasting Accounts Receivable Collections

The forecasting workflow involves a few basic steps:

- Step 1: Coerce to a ts object class.
- Step 2: Apply a model (or set of models)
- Step 3: Forecast the models (similar to predict)
- Step 4: Tidy the forecast

First, I need to get the data organized into groups by month of the year.

```
daily_collections_by_customer <-
    sales_journal[,1:15] %>%
    filter(cash_not_ar == 0) %>%    ## credit only
    mutate(invoice_day = decimal_date(as_date(invoice_date))) %>%
    group_by(customer_no, invoice_day) %>%
    summarise(daily_collections = sum(collection_amount)) %>%
    mutate(invoice_dt =
            round_date(date_decimal(invoice_day),
                       unit = "day" ) ) %>%
  select(customer_no, invoice_date=invoice_dt, daily_collections)




# Next, we use the nest() function from the tidyr package to consolidate
each time series by
# group. The newly created list-column, “data.tbl”, contains the
“order.month” and “total.qty”
# columns by group from the previous step. The nest() function just bundles
the data together
# which is very useful for iterative functional programming.


daily_collections_nest <-
  daily_collections_by_customer %>%
    group_by(customer_no) %>%
    nest()


### Step 1: Coerce to a ts object class

# In this step we map the tk_ts() function into a new column “data.ts”. The
procedure is
# performed using the combination of dplyr::mutate() and purrr::map(),
which works really well
# for the data science workflow where analyses are built progressively. As
a result, this
# combination will be used in many of the subsequent steps in this vignette
as we build the
# analysis.

#### mutate and map

# The mutate() function adds a column, and the map() function maps the
contents of a list-column # (.x) to a function (.f). In our case, .x =
data.tbl and .f = tk_ts. T


daily_collections_ts <-
  daily_collections_nest %>%
    mutate(data.ts = map(.x       = data,
```

```
                          .f       = tk_ts,
                          select   = -invoice_date,
                          start    = 2019,
                          freq     = 365))

## Step 2: Modeling a time series

# Next, we map the Exponential Smoothing ETS (Error, Trend, Seasonal) model
function, ets, from
#  the forecast package. Use the combination of mutate to add a column and
map to interactively
# apply a function rowwise to a list-column. In this instance, the function
to map the ets
# function and the list-column is “data.ts”. We rename the resultant column
“fit.ets” indicating # an ETS model was fit to the time series data.


daily_collections_fit <-
  daily_collections_ts %>%
    mutate(fit.ets = map(data.ts, ets))




### sw_tidy -- do some model inspection with the sweep tidiers.
# To get the model parameters for each nested list, we can combine sw_tidy
within the mutate and map combo.
# The only real difference is now we unnest the generated column (named
“tidy”).
# because it’s easier to compare the model parameters side by side, we add
one additional call to spread() from the tidyr package.


daily_collections_fit %>%
    mutate(tidy = map(fit.ets, sw_tidy)) %>%
    unnest(tidy) %>%
    spread(key = customer_no, value = estimate)

### sw_glance -- view the model accuracies also by mapping sw_glance within
the mutate and map combo.


daily_collections_fit %>%
    mutate(glance = map(fit.ets, sw_glance)) %>%
    unnest(glance)



### sw_augment


augment_fit_ets <-
  daily_collections_fit %>%
    mutate(augment = map(fit.ets, sw_augment, timetk_idx = TRUE,
```

```
rename_index = "date")) %>%
    unnest(augment)




# plot the residuals for the nine categories like so. Unfortunately we do
see some very

augment_fit_ets %>%
    ggplot(aes(x = date, y = .resid, group = customer_no)) +
    geom_hline(yintercept = 0, color = "grey40") +
    geom_line(color = palette_light()[[2]]) +
    geom_smooth(method = "loess") +
    labs(title = "A/R Collection Delay by Customer",
         subtitle = "ETS Model Residuals", x = "") +
    theme_tq() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    facet_wrap(~ customer_no, scale = "free_y", ncol = 3)


### sw_tidy_decomp -- create decompositions using the same procedure with
sw_tidy_decomp() and the mutate and map combo.

daily_collections_fit %>%
    mutate(decomp = map(fit.ets, sw_tidy_decomp, timetk_idx = TRUE,
rename_index = "date")) %>%
    unnest(decomp)


## Step 3: Forecasting the model


daily_collections_fcast <-
  daily_collections_fit %>%
    mutate(fcast.ets = map(fit.ets, forecast, h = 12))

## Step 4: Tidy the forecast --  apply sw_sweep to get the forecast in a
nice “tidy” data frame. #  use the argument fitted = FALSE to remove the
fitted values from the forecast


daily_collections_fcast_tidy <-
  daily_collections_fcast %>%
    mutate(sweep = map(fcast.ets, sw_sweep, fitted = FALSE, timetk_idx =
TRUE)) %>%
    unnest(sweep)


## Visualization


daily_collections_fcast_tidy %>%
    ggplot(aes(x = index, y = daily_collections, color = key, group =
```

```
customer_no)) +
    geom_ribbon(aes(ymin = lo.95, ymax = hi.95),
                fill = "#D5DBFF", color = NA, size = 0) +
    geom_ribbon(aes(ymin = lo.80, ymax = hi.80, fill = key),
                fill = "#596DD5", color = NA, size = 0, alpha = 0.8) +
    geom_line() +
    labs(title = "A/R Collection Delay by Customer",
         subtitle = "ETS Model Forecasts",
         x = "", y = "Units") +
    scale_color_tq() +
    scale_fill_tq() +
    facet_wrap(~ customer_no, scales = "free_y", ncol = 3) +
    theme_tq() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Calculating Allowance for Uncollectable Accounts

Let the client's allowance for uncollectables accrual policy be:

- 1% of A/R under 45 days old are uncollectable
- 2% of A/R between 45 days and 120 days old are uncollectable
- 5% of A/R over 120 days old are uncollectable

The statistical estimation from the time-series analysis previously completed provides confidence bounds, for the forecast of optimistic, pessimistic and expected uncollectables. This is 'overkill' for audits, and I will restrict the application of the client's accrual policy for just the expected rate of collection of receivables, based on prior experience with each customer. Then we can apply policy, using forecast daily collections and outstanding A/R using the following code chunk.

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

fyear_end <- paste0(format(Sys.Date(), '%Y'), '-12-31')   # set the fiscal
year end date to match the dataset


## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}
```

```
sales_journal <-
  read.csv("sales_journal.csv", na.strings="0", stringsAsFactors=FALSE)


credit_sales_journal <-
  sales_journal %>%
  filter(cash_not_ar == 0)



## calculate each customer's unpaid_ar

ar_by_customer <-
  credit_sales_journal[
    credit_sales_journal$collection_date >= fyear_end &
      credit_sales_journal$shipper_date <= fyear_end,] %>%
  mutate(age = ymd(fyear_end) - ymd(invoice_date)) %>%
  group_by(customer_no) %>%
  summarize(sum(sales_extended),
            average_age = mean(age)) %>%
  rename(customer_ye_balance='sum(sales_extended)')


## calculate each customer's forecasted collections
#(from the daily_collections_fcast_tidy data.frame)
#including 80 and 95% confidence limits

forecast_by_customer <-
daily_collections_fcast_tidy %>%
  select(customer_no,key, daily_collections,lo.80,hi.80,lo.95,hi.95) %>%
  filter(key == "forecast") %>%
  group_by(customer_no) %>%
  summarize(f_collections = mean(daily_collections),
            f_lo.80  = mean(lo.80),
            f_lo.95  = mean(lo.95),
            f_hi.80  = mean(hi.80),
            f_hi.95  = mean(hi.95)
            ) %>%
  inner_join(ar_by_customer, by="customer_no") %>%
  mutate(days_to_pay = customer_ye_balance / f_collections,
         age_when_fully_paid = days_to_pay + average_age)

## apply the client's allowance for uncollectables estimation
#policy assuming the steady daily rate of payment from the ETS

allowance_for_uncollectable_ar <-
  forecast_by_customer %>%
  mutate(age_when_fully_paid =
           as.numeric(age_when_fully_paid),
         uc_lt_45 =
           ifelse(age_when_fully_paid <
                    45, customer_ye_balance *  .01, 0),
         uc_45_120 =
```

```
            ifelse(age_when_fully_paid >=
                     45 & age_when_fully_paid <= 120,
                   (1 - 45 / age_when_fully_paid) *
                     customer_ye_balance *  .02 +
                     (45 / age_when_fully_paid) *
                     customer_ye_balance *  .01,  0),
          uc_gt_120 =
            ifelse(age_when_fully_paid >
                     120,
                   (1 - 120 / age_when_fully_paid) *
                     customer_ye_balance *  .05   +
                     (75 / age_when_fully_paid) * customer_ye_balance *
                     .02 + 45 *
                     f_collections *  .01,  0),
          allowance = uc_lt_45 + uc_45_120 + uc_gt_120,
                   pct_of_ye_balance = allowance / customer_ye_balance
            ) %>%
  select(customer_no, allowance, pct_of_ye_balance)

total_allowance <- sum(allowance_for_uncollectable_ar$allowance)
cat("Allowance for Uncollectable Accounts = ", total_allowance)

kable(allowance_for_uncollectable_ar, caption = "Accrual Calculations for
Allowance for Uncollectables (by customer)","latex", booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")
```

Stratified Samples and Monetary Unit Sampling

In some cases the sample designer has access to an "auxiliary variable" or "size measure", believed to be correlated to the variable of interest, for each element in the population. These data can be used to improve accuracy in sample design. One option is to use the auxiliary variable as a basis for stratification.

Stratification has a long history in auditing as a method of audit cost control, focusing the auditing effort on higher value items. The standard approach to stratification involves simple partitionings of the transaction documents into low and high value documents. During the 1960's and 1970's sampling methods were developed around the idea of 'monetary-unit sampling' which suggested that, for substantive auditing, individual dollars in an account balance should be sampled, rather the transaction documents.

Currently, monetary-unit sampling uses methods developed for survey work called probability proportional to size ('PPS') sampling, in which the selection probability for each element is set to be proportional to its size measure, up to a maximum of 1. Probability proportional to size (PPS) sampling is a method of sampling from a finite population in which a size measure is available for each population unit before sampling and where the probability of selecting a unit is proportional to its size. Its use arises in two particular contexts: (i) multistage sampling and (ii) single-stage sampling of establishments. Unbiased estimation is obtained using the Hansen–Hurvitz estimator in the case of PPS with replacement sampling and the Horvitz–Thompson estimator in the case of probability proportional to size without replacement (PPSWOR) sampling.

I provide examples below, using R's *TeachingSampling* package for sampling and estimation with stratification and PPS. Note that the *TeachingSampling* package estimates both population size and amount from stratified and PPS samples. Since we already know the sample size from the client's file, we can reduce uncertainty by converting to means and standard errors and then extrapolating to the population.

PPS and monetary-unit sampling provide substantial increases in accuracy over stratified sampling with effective control of audit costs. They are also easier to set up (basically, no more difficult than using unstratified random sampling). The following code chunks create a 3-level stratification, and for comparison, a PPS sampling based on dollar value of the invoice amount.

```
library(TeachingSampling)
library(tidyverse)
library(broom)
library(knitr)
library(kableExtra)

unpaid_ar <-
  sales_journal[1:15] %>%
  filter(cash_not_ar == 0 &
         collection_date >= fyear_end &
         shipper_date <= fyear_end) %>%
  select(-X)

## Separate the dataset into three strata and set the sample proportions
unpaid_ar$stratum <-
  cut(unpaid_ar$sales_extended, breaks=3, labels=c("small", "medium",
"large"))

sample_prop <- data.frame(prop = c(.05,.1,.2), stratum = c("small",
"medium", "large"))

# Define the size of each stratum
N <- NULL
for(i in 1:3) N[i] <-summary(unpaid_ar$stratum )[[i]]
Nh <- as.numeric(N[1:3])                          ## vector of stratum
sizes
nh = as.numeric(ceiling(N * sample_prop$prop))    ## vector of samples size
by stratum

# Draw a stratified sample and estimate
ar_sample <- unpaid_ar[S.STSI(unpaid_ar$stratum, Nh, nh),]
est <- data.frame(unpaid_ar$sales_extended)
estimate <- E.STSI(unpaid_ar$stratum, Nh, nh, est )

## normalize the estimate to reflect the number of transactions in the
client's file
normalized_estimate <-  nrow(unpaid_ar) * estimate[,,2]/estimate[1,,1]
normalized_estimate <- round(normalized_estimate, digits=0)

tidy(normalized_estimate) %>%
kable(caption = "3-Stratum Horvitz–Thompson estimate of Total Unpaid
```

```
A/R","latex", booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")
```

PPS

R contains routines to implement probability proportional to scale sampling, which is used in population studies and other areas. The following code shows how this can be applied in an audit setting.

```
library(TeachingSampling)
library(tidyverse)
library(broom)
library(knitr)
library(kableExtra)

unpaid_ar <-
  sales_journal[1:15] %>%
  filter(cash_not_ar == 0 &
         collection_date >= fyear_end &
         shipper_date <= fyear_end) %>%
  select(-X)

# Draw a PPS sample of size 'size' and estimate draw a random sample
according to a
# PPS with replacement design
size <- 5
sample_prop <- S.PPS(size,unpaid_ar$sales_extended)
# returns a matrix of m rows and two columns.
# Each element of the first column indicates the unit that was selected.
# Each element of the second column indicates the selection probability of
this unit

sample_ar <- unpaid_ar[sample_prop[,1],]

est <- data.frame(unpaid_ar$sales_extended)
estimate <- E.PPS(est,sample_prop[,2])


## normalize the estimate to reflect the number of transactions in the
client's file
normalized_estimate <-  nrow(unpaid_ar) * estimate[,2]/estimate[1,1]
normalized_estimate <- round(normalized_estimate, digits=0)

tidy(normalized_estimate) %>%
kable(caption = "PPS Hansen-Hurwitz estimate of Population Total ","latex",
booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")
```

Computer Analytic Workpaper Support

The first step in setting up workpapers is the access to the client files. Here I bring in files germane to the inventory audit.

```
rm(list=ls())
library(readr)
library(tidyverse)
library(broom)
library(stringr)
library(data.table)

## Set the directory for the new files (modify the path as needed)
default_dir <- "/home/westland/audit_analytics_book/audit_simulated_files/"

if (file.exists(default_dir)){
    setwd(default_dir)
} else {
    dir.create(default_dir)
    setwd(default_dir)
}


sales_journal <-
  read_csv("sales_journal.csv",
    col_types = cols(X1 = col_skip()))

purchase_journal <-
  read_csv("purchase_journal.csv",
    col_types = cols(X1 = col_skip()))

perpetual_inventory_ledger <-
  read_csv("perpetual_inventory_ledger",
    col_types = cols(X1 = col_skip()))

fyear_end_ar_ledger <-
  read_csv("fyear_end_ar_ledger.csv",
    col_types = cols(X1 = col_skip()))

fyear_begin_inventory_ledger <-
  read_csv("fyear_begin_inventory_ledger.csv",
    col_types = cols(X1 = col_skip()))

expenditures <-
  read_csv("expenditures.csv",
    col_types = cols(X1 = col_skip()))

disbursement_journal <-
  read_csv("disbursement_journal.csv",
    col_types = cols(X1 = col_skip()))
```

```
deposit_daily <-
  read_csv("deposit_daily.csv",
    col_types = cols(X1 = col_skip()))

daily_ar_balance <-
  read_csv("daily_ar_balance.csv",
    col_types = cols(X1 = col_skip()))

collections_journal <-
  read_csv("collections_journal.csv",
    col_types = cols(X1 = col_skip()))

ap_ledger <-
  read_csv("ap_ledger.csv",
    col_types = cols(X1 = col_skip()))


real_world_ye_inventory <-
  perpetual_inventory_ledger %>%
  group_by(sku) %>%
  inner_join(sales_journal, by="sku") %>%
  slice(n()) %>%
  mutate(inv_extended = unit_cost * stock_on_hand)
```

Inventory

Footing, Reconcile the inventory count to the general ledger.

Trace the valuation compiled from the physical inventory count to the company's general ledger, to verify that the counted balance was carried forward into the company's accounting records.

```
library(tidyverse)

## get the counts by SKU

foot_count <- perpetual_inventory_ledger %>%
  group_by(sku) %>%
  slice(n()) %>%
  select(stock_on_hand) %>%
  as.data.frame()

## get the unit cost by SKU

foot_sku <- purchase_journal %>%
  group_by(sku) %>%
  slice(n()) %>%
```

```
  select(unit_cost) %>%
  as.data.frame()

## extend

foot_value <- perpetual_inventory_ledger %>%
  group_by(sku) %>%
  inner_join(sales_journal, by="sku") %>%
  slice(n()) %>%
  mutate(inv_value = stock_on_hand * unit_cost) %>%
  ungroup() %>%
  select(inv_value)  %>%
  sum()


cat("Value of inventory (per client books) at year end = $",
format(foot_value, big.mark=","))

## compare this to the (unobservable) true value of year end inventory

true_foot_value <-
  real_world_ye_inventory %>%
  group_by(sku) %>%
  inner_join(sales_journal, by="sku") %>%
  slice(n()) %>%
  mutate(inv_value = stock_on_hand * unit_cost.x)%>%
  ungroup() %>%
  select(inv_value)  %>%
  sum()


cat("\n Compare to the (unobservable)
    true value of inventory at year end = $",
    format(true_foot_value, big.mark=",") )
```

Cutoff analysis.

The auditors will examine your procedures for halting any further receiving into the warehouse or shipments from it at the time of the physical inventory count (assuming they have a single year-end count) so that in-transit inventory is excluded. They typically test the last few receiving and shipping transactions prior to the physical count, as well as transactions immediately following it, to see if you are properly accounting for them.

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

```
fyear_begin <- paste0(format(Sys.Date(), '%Y'), '-01-01')  # date can be
hard-coded if needed
fyear_end <- paste0(format(Sys.Date(), '%Y'), '-12-31')

sales_journal$invoice_date <- as_date(sales_journal$invoice_date)
sales_journal$shipper_date <- as_date(sales_journal$shipper_date)

sales_cutoff_errors <- sales_journal %>%
  filter(invoice_date <= fyear_end & shipper_date > fyear_end) %>%
  select(customer_no,invoice_no, invoice_date, shipper_no, shipper_date,
sales_extended)


# print the inventory receipts cutoff worksheet

perpetual_inventory_ledger %>%
  group_by(sku) %>%
  slice(n()) %>%
  left_join(ap_ledger, by="sku") %>%
  filter(receiver_date > fyear_end) %>%
    kable(
      caption = "Inventory Receipt Cutoff Worksheet
      (verify that none of these receipts were included in Y/E inventory)",
      "latex", booktabs = T) %>%
    kable_styling(bootstrap_options = "striped",
    latex_options="scale_down")
```

Duplicates and Omissions

Companies index each high-volume class of journal entry (e.g., sales, receipts, etc) with an index number. Before the widespread use of computer entry for transactions, these were actually preprinted at the top of a multipart document, and auditors were expected to assure that preprinted transaction entry documents were being used, and the sequence of index numbers was fully accounted for. Omissions or duplicates of an index number might indicate an error in processing which would over or understate the account, and which would cause other internal control problems. The following code finds duplicates and omissions (gaps in the indexing number on the journal entry transaction document)

```
library(tidyverse)
library(stringr)
library(dplyr)


# duplicated records
dup_purchase <- purchase_journal[duplicated(purchase_journal$po_no),]
n <- nrow(dup_purchase)
```

```
cat("\n # of duplicate purchases = ", n)

dup_sales <- sales_journal[duplicated(sales_journal$invoice_no),]
n <- nrow(dup_sales)
cat("\n # of duplicate sales = ", n)



receiver_journal <-
  perpetual_inventory_ledger %>%
  group_by(sku) %>%
  slice(n()) %>%
  left_join(ap_ledger, by="sku")

dup_receiver <- receiver_journal[duplicated(receiver_journal$receiver_no),]
n <- nrow(dup_receiver)
cat("\n # of duplicate receivers = ", n)

dup_shipment <- sales_journal[duplicated(sales_journal$shipper_no),]
n <- nrow(dup_shipment)
cat("\n # of duplicate shipments = ", n)



## omissions

po <- as.numeric(substring(purchase_journal$po_no,2))
po_min <- as.numeric(min(po))
po_max <- as.numeric(max(po))

omit <- as.data.frame(setdiff(po_min:po_max,po))
n <- nrow(omit)
cat("\n # of omitted purchase records = ", n)

invoice <- as.numeric(substring(sales_journal$invoice_no,2))
invoice_min <- as.numeric(min(invoice))
invoice_max <- as.numeric(max(invoice))

omit <- as.data.frame(setdiff(invoice_min:invoice_max,invoice))
n <- nrow(omit)
cat("\n # of omitted sales records = ", n)


receiver <- as.numeric(substring(receiver_journal$receiver_no,4))
receiver_min <- as.numeric(min(receiver))
receiver_max <- as.numeric(max(receiver))

omit <- as.data.frame(setdiff(receiver_min:receiver_max,receiver))
n <- nrow(omit)
cat("\n # of omitted receiver records = ", n)


shipments <- as.numeric(substring(sales_journal$shipper_no,2))
shipments_min <- as.numeric(min(shipments))
```

```
shipments_max <- as.numeric(max(shipments))

omit <- as.data.frame(setdiff(shipments_min:shipments_max,shipments))
n <- nrow(omit)
cat("\n # of omitted sales records = ", n)
```

Physical Count Exceptions to Perpetual Inventory Ledger

The perpetual inventory ledger maintains a running balance of inventory at cost, and is updated from sales and purchase transaction documents. The actual balance of inventory will vary from the perpetual inventory ledger in both item count and value for several reasons, e.g.:

1. Obsolescence or other cause of decline in market value
2. Damage to items in transit or storage
3. Misclassification or misplacement, often a problem with high volume retailers like Walmart
4. Markdown of value because a returned item was refurbished, or delivered with an open box
5. Quality control issues that reduce finished goods yield, or require markdown of value.

The auditors want to be comfortable with the procedures you use to count the inventory. This means that they will discuss the counting procedure with you, observe counts as they are being done, test count some of the inventory themselves and trace their counts to the amounts recorded by the company's counters, and verify that all inventory count tags were accounted for. If you have multiple inventory storage locations, they may test the inventory in those locations where there are significant amounts of inventory. They may also ask for confirmations of inventory from the custodian of any public warehouse where the company is storing inventory. The physical count of inventory is not just a count, but offers employees and auditors the opportunity to inspect the condition and marketability of inventory. Items where value has been significantly marked down, will be subject to Lower of Cost or Market (LOCOM) revaluation discussed in the next subsection.

```
library(kableExtra)
library(tidyverse)

real_world_ye_inventory <-

read_csv("/home/westland/audit_analytics_book/audit_simulated_files/real_wo
rld_ye_inventory.csv",
     col_types = cols(X1 = col_skip()))
real_world_ye_inventory[is.na(real_world_ye_inventory)] <- 0

inventory_count_differences <-
  perpetual_inventory_ledger %>%
  group_by(sku) %>%
  left_join(real_world_ye_inventory, by="sku") %>%
  slice(n()) %>%    ## the final slice, by SKU, will be what is in-stock at
year end
  filter(exception != "No exception, count is accurate") %>%
  mutate(err_perpetual = stock_on_hand - ye_stock_on_hand) %>%
```

```
  select(sku, err_perpetual, unit_cost, count_exception, exception,
actual_unit_market) %>%
  as.data.frame()


kable(inventory_count_differences,
      caption = "Perpetual vs. Actual Overstatement
      (negative implies understatement of perpetual inventory)",
      "latex", booktabs = T) %>%
    kable_styling(bootstrap_options = "striped")
```

Test for lower of cost or market (LOCOM)

Inventory is typically valued at acquisition cost. The physical count of inventory is not just a count, but offers employees and auditors the opportunity to inspect the condition and marketability of inventory. Items where value has been significantly marked down, will be subject to Lower of Cost or Market (LOCOM) revaluation discussed previously. Auditors are required to apply the lower of cost or market rule to revalue inventory to its net realizable value, and will do so by comparing a selection of market prices to their recorded costs.

```
library(tidyverse)

real_world_ye_inventory[is.na(real_world_ye_inventory)] <- 0

inventory_count_differences <- perpetual_inventory_ledger %>%
  group_by(sku) %>%
  slice(n()) %>%    ## the final slice, by SKU, will be what is in-stock at
year end
  left_join(real_world_ye_inventory, by="sku") %>%
  filter(exception != "No exception, count is accurate") %>%
  mutate(err_perpetual = stock_on_hand - ye_stock_on_hand) %>%
  select(sku,
         stock_on_hand,
         ye_stock_on_hand,
         unit_cost,
         actual_unit_market,
         err_perpetual,
         count_exception,
         exception) %>%
  mutate(ye_cost = stock_on_hand * unit_cost,
         ye_market = ye_stock_on_hand * actual_unit_market,
         inv_markdown = ye_cost - ye_market) %>%
  select(sku, ye_cost, ye_market, inv_markdown) %>%
  as.data.frame()
```

```
total_inv_markdown <- sum(inventory_count_differences$inv_markdown)
cat("\n Total Amount to Markdown
    Client's Trial Balance Inventory Amount
    (by LOCOM rule) = ",
    total_inv_markdown)

library(knitr)
library(kableExtra)
kable(inventory_count_differences, caption = "Market and Cost of Inventory
by SKU","latex", booktabs = T) %>%
  kable_styling(bootstrap_options = "striped") %>%
   footnote (general = "",
             symbol = c(round(total_inv_markdown,2)),
             general_title = "Total Markdown for LOCOM rule",
             title_format = c("italic", "underline")
   )
```

Audit a subset of high-value items.

If there are items in the inventory that are of unusually high value, the auditors will likely spend extra time counting them in inventory, ensuring that they are valued correctly, and tracing them into the valuation report that carries forward into the inventory balance in the general ledger.

```
library(tidyverse)

high_value <- perpetual_inventory_ledger %>%
  left_join(purchase_journal, by="sku") %>%
  select(sku,stock_on_hand,unit_cost) %>%
  mutate(value = stock_on_hand * unit_cost) %>%
  arrange(desc(value)) %>%
  slice(1:5) %>%   ## choose highest 5 in value
  as.data.frame()

library(knitr)
library(kableExtra)
kable(high_value, caption = "Top 5 inventory SKU by value","latex",
booktabs = T) %>%
  kable_styling(bootstrap_options = "striped")
```

Inventory allowances.

The auditors will determine whether the amounts you have recorded as allowances for obsolete inventory or scrap are adequate, based on your procedures for doing so, historical patterns, "where used" reports, and reports of inventory usage (as well as by physical observation during the physical count). If you do not have such allowances, they may require you to create them.

Obsolete inventory is not just that inventory that is identified as obsolete in the physical count, but also inventory on hand that is not expected to sell in some established period into the future. Management is often keen to control the holding cost (cost of warehouse space and management) of inventory, as well as the technological obsolescence (newer products arrive that are lower cost or better performing substitutes for the inventory). Allowances for obsolete inventory are often computed by predicting sales, and determining if there is excess inventory. Below is a code chunk to perform this calculation.

```
library(tidyverse)

## assume that the client maintains an inventory balance sufficient for 1
year of sales,
## and any excess balance is considered obsolete inventory that should be
marked down
## to disposal value (which is determined to be 50% of original unit cost)

sales_of_sku_per_year <- sales_journal %>%    ## sales_journal is only for
this fiscal year, so this rate is just the sum of sales in the
sales_journal
  group_by(sku) %>%
  summarise(sales_rate = sum(sales_count, na.rm = T))

unit_costs <- purchase_journal %>%
    group_by(sku) %>%
    slice(n()) %>%
    select(sku, unit_cost) %>%
    as.data.frame()

allowance_for_obsolete_inv <- perpetual_inventory_ledger %>%
  group_by(sku) %>%
  slice(n()) %>%    ## the final slice, by SKU, will be what is in-stock at
year end
  left_join(sales_of_sku_per_year, by="sku") %>%
  mutate(years_on_hand = stock_on_hand / sales_rate) %>%
  left_join(sales_of_sku_per_year, by="sku") %>%
  left_join(unit_costs, by="sku") %>%
  select(sku, years_on_hand, stock_on_hand, sales_rate = sales_rate.x,
unit_cost) %>%
  filter(years_on_hand > 1) %>%  ## more than 1 yearstock on hand
  mutate(allowance = (years_on_hand - 1) *  stock_on_hand * unit_cost) %>%
  select(sku, years_on_hand, stock_on_hand, sales_rate, unit_cost,
allowance) %>%
  as.data.frame()

if(nrow(allowance_for_obsolete_inv) == 0){cat("\n\n No Obsolete Inventory
```

```
  for any SKU  \n\n")}

  library(knitr)
  library(kableExtra)
  kable(allowance_for_obsolete_inv, caption = "Allowance for Obsolete
  Inventory by SKU",
        "latex", booktabs = T) %>%
    kable_styling(bootstrap_options = "striped")
```