Audit Report for Macy’s

By Mughundhan Chandrasekar

4/27/2017

Creating an Environment

We create a suitable environment for performing the required operations on the given data-sets by loading the required libraries before-hand. The data-sets that are required are also loaded during preliminary stage.

```
rm(list=ls())
library(sqldf)
library(plyr)
library(readxl)
library(stringr)
library(lubridate)
library(benford.analysis)
library(pwr)
library(pps)
```

We create few functions for enhancing re-usability and efficiency of the project.

```
importAccounts = function() {
  library(readxl, readr)
  path = "/Users/Mughundhan/UIC/UIC Academics/SPRING 2017/AUDIT/Final
Presentation/Dataset" ## folder for files downloaded from UIC Blackboard
  files = c("arConfirmations.csv", "custCredit.csv", "empReimbursements.csv",
"inventoryCounts.csv", "inventoryPerpetual.csv", "arCollections.csv",
"purchases.csv", "sales.csv")
  dataFrameList = list()
  for(i in 1:length(files)){
    dataFrameName = strsplit(files[i], ".", fixed = TRUE)[[1]][1]
    fileType = strsplit(files[i], ".", fixed = TRUE)[[1]][2]
    if(fileType == "xlsx") {
      dataFrame = read_excel(paste(path, files[i], sep = "/"))
    } else {
      dataFrame = read.csv(paste(path, files[i], sep = "/"))
    }
    namedFrame = assign(dataFrameName, dataFrame)
    dataFrameList[[dataFrameName]] = namedFrame
  }
  return(dataFrameList)
}


convertAccounts = function(accounts) {
  library(stringr)
  library(lubridate)
  for(i in 1:length(accounts)) {
    for (n in 1:length(accounts[[i]])) {
      dataFrame = accounts[[i]]
      if(str_detect(names(dataFrame[n]), "date") |
str_detect(names(dataFrame[n]), "dateColl")){
        if(is.factor(dataFrame[[n]])){
          accounts[[i]][[n]] = ymd(as.character(dataFrame[[n]]))
        }
      }
      else if(str_detect(names(dataFrame[n]), "sku") |
str_detect(names(dataFrame[n]), "invoice")
              | str_detect(names(dataFrame[n]), ".no") |
str_detect(names(dataFrame[n]), ".No")  | str_detect(names(dataFrame[n]),
"customer")){
        accounts[[i]][[n]] = as.character(dataFrame[[n]])
      }
      else if (str_detect(names(dataFrame[n]), "cashtrue")) {
        accounts[[i]][[n]] = as.logical(dataFrame[[n]])
      }
      else if(str_detect(names(dataFrame[n]), "Amount")){
        accounts[[i]][[n]] = as.numeric(dataFrame[[n]])
      }
    }
  }
  return(accounts)
}


createCostofGoodsSold = function(accounts){
  costOfGoodsSold = merge(accounts$sales, accounts$inventoryPerpetual,
by="sku", all.x=T)
  costOfGoodsSold$COGS = costOfGoodsSold$unitcost * costOfGoodsSold$qty
  accounts[["costOfGoodsSold"]] = costOfGoodsSold
  return(accounts)
}


createUnpaidAccountsReceivable = function(accounts) {
  splitSalesbyTransaction = split(accounts$sales, accounts$sales$cashtrue)
  credit = splitSalesbyTransaction[["FALSE"]]
  allCreditAccounts = merge(credit, accounts$arCollections, by="invoice",
all.x = T)
  allCreditAccounts$notCollected = is.na(allCreditAccounts$amt.received)
  allCreditAccountsbyCollection = split(allCreditAccounts,
allCreditAccounts$notCollect)
  unpaidAccountsReceivable = allCreditAccountsbyCollection[["TRUE"]]
  accounts[["unpaidAccountsReceivable"]] = unpaidAccountsReceivable
  return(accounts)
}



createAllowanceForDoubtfulAccounts = function(accounts) {
  x = accounts$unpaidAccountsReceivable
  endDateVector = rep(ymd("2016/12/31"),
length(accounts$unpaidAccountsReceivable$invoice))
  x$endDate = endDateVector
  x$daysSincePurchase = x$endDate - x$date
  x$interval = findInterval(x$daysSincePurchase, c(90, 180))
  accounts[["doubtfulAccounts"]] = x
  return(accounts)
}



createOutofStock = function(accounts){
  salesBySKU = aggregate(qty~sku, accounts$sales,sum)
  purchasesBySKU = aggregate(quantity~sku,accounts$purchases,sum)
  purcahsesSalesBySKU = merge(salesBySKU, purchasesBySKU, by="sku")
  purchasesSalesInventoryBySKU = merge(purcahsesSalesBySKU,
accounts$inventory, by="sku")
  purchasesSalesInventoryBySKU$turnover =
(purchasesSalesInventoryBySKU$qtypurchasesSalesInventoryBySKU$quantity)/purch
asesSalesInventoryBySKU$endstock
  turnover =
data.frame(purchasesSalesInventoryBySKU$sku,purchasesSalesInventoryBySKU$turn
over)
  colnames(turnover)=c("sku","times")
  accounts[["turnover"]] = turnover
  return(accounts)
}



createAccountsByYear = function(accounts, year) {
  for(i in 1:length(accounts)) {
    for (n in 1:length(accounts[[i]])) {
      dataFrame = accounts[[i]]
      dateColumnExists = FALSE
      if(str_detect(names(dataFrame[n]), "date") |
str_detect(names(dataFrame[n]), "dateColl")){
        dateColumn = n
        dateColumnExists = TRUE
        break()
      }
    }
    if(dateColumnExists == TRUE) {
      accounts[[i]]$year = year(accounts[[i]][[dateColumn]])
      dataFramebyYear = split(accounts[[i]], accounts[[i]][["year"]])
      accounts[[i]] = dataFramebyYear[[year]]
    }
  }
  return(accounts)
}
```

Now, we make use of the above functions to **Filter Audit Year-2016's Transactions** and few rows of the Audit Year-2016's Transactions are displayed below

```
accounts = importAccounts()
accounts = convertAccounts(accounts)
accounts2016 = createAccountsByYear(accounts, year = "2016")
accounts2016 = createCostofGoodsSold(accounts2016)
accounts2016 = createUnpaidAccountsReceivable(accounts2016)
accounts2016 = createAllowanceForDoubtfulAccounts(accounts2016)
#head(accounts2016)
```

Questions And Solutions

Now let us have a look at the solutions for the questions posted on Blackboard.

1.PLANNING AND RISK ASSESSMENT

Part 1: High Risk Accounts

- For ease of understanding, we have displayed it in a tabular format.

```
##       RISKS                              IMPACT LIKELIHOOD RISK_FACTOR
##  [1,] "Cash"                             "5"    "7"        "35"
##  [2,] "Accounts Receivable"              "8"    "9"        "72"
##  [3,] "Inventory"                        "6"    "8"        "48"
##  [4,] "Fixed Assets"                     "8"    "4"        "32"
##  [5,] "Accounts Payable"                 "3"    "7"        "21"
##  [6,] "Cost of Goods Sold"               "8"    "5"        "40"
##  [7,] "Depreciation Expense"             "5"    "6"        "30"
##  [8,] "Sales Revenue (net)"              "7"    "8"        "56"
##  [9,] "Employee Expenses"                "9"    "7"        "63"
## [10,] "Allowances for Doubtful Accounts" "6"    "7"        "42"
```

- We considered the complete data set for the internal controls and substantive tests. This is because considering the complete dataset would enable us to audit the accounts precisely and efficiently. R Studio is a powerful tool that helped us take the whole dataset into account.

We also ran a t-test using pwr library and we got the random sample size values for account receivable audit and inventory audit. These are as follows -

```
##      Audits                        Sizes
## [1,] "Accounts Receivable Audit" "1483718"
## [2,] "Inventory Audit"           "185774"
```

2. TESTS OF INTERNAL CONTROLS

Part(1): Customers who exceeded their Credit Limit

```
findCreditNegatives = function(accounts) {
  library(plyr, dplyr)
  #Prepare Sales table
  sales = split(accounts$sales, accounts$sales$cashtrue)[["FALSE"]]
  sales = subset(sales, select = c(date, cust.no, total))
  names(sales)[names(sales) == "total"] = "trans"
  sales$trans = sales$trans*-1
  #Prepare Collections table
  collections = merge(accounts$sales, accounts$arCollections, by = "invoice",
all.x = T)
  collections = na.omit(collections)
  collections = subset(collections, select = c(dateColl, cust.no.x,
amt.received))
  names(collections)[names(collections) == "dateColl"] = "date"
  names(collections)[names(collections) == "amt.received"] = "trans"
  names(collections)[names(collections) == "cust.no.x"] = "cust.no"
  #TransactionsTable
  transTable = rbind(sales, collections)
  transTable = arrange(transTable, date)
  #Create TransByCustomer
  transByCustomer = split(transTable, transTable$cust.no)

  #Loop through customers
  badCreditAccount = data.frame()
  for(i in 1:length(transByCustomer)) {
    customer = transByCustomer[[i]]
    customerNumber = transByCustomer[[i]][1,]$cust.no
    customer$subTotal =
accounts$custCredit[as.numeric(customerNumber),]$limit
    #loop through customer
    for(n in 1:length(customer$subTotal)) {
      if(n != 1) {
        customer[n,]$subTotal = customer[n - 1,]$subTotal +
customer[n,]$trans
        if(sign(customer[n,]$subTotal) == -1) {
          badCreditAccount = rbind(badCreditAccount, customer[n,])
          break
        }
      }
    }
  }
  accounts[["overlimitCreditApprovals"]] = badCreditAccount
  return(accounts)
```

```r
}
accounts2016 = findCreditNegatives(accounts2016)
#head(accounts2016$overlimitCreditApprovals)

## [1] 485
```

- Inference: On performing the above functionality, we arrive at the conclusion that, Number of customers exceeding credit limit sums upto 485.

Part (2.a): DUPLICATE TRANSACTIONS

```r
findDuplicates = function(dataframe, column) {
  dataframe$test = as.numeric(dataframe[[column]])
  dataframe$dup = duplicated(dataframe$test)
  x = split(dataframe, dataframe$dup)
  y = x[["TRUE"]]
  print(y)
  print ("Duplicates (head)")
  head(y)
}
findDuplicates(dataframe = accounts2016$sales, column = "invoice")

## NULL
## [1] "Duplicates (head)"

## NULL
```

Part (2.b): OMITTED TRANSACTIONS

```r
findMissingEntries =function(max,set) {
  good = 1:max
  test = as.numeric(set)
  missing = setdiff(good, set)
  print(missing)
  print ("Missing (head)")
  head(missing)
}
#head(findMissingEntries(max = length(accounts2016$sales$invoice), set =
accounts2016$sales$invoice))
```

Part (2.c): TRANSACTION CUT OFF TEST

```r
findSalesNotIn2016 = function(accounts) {
  x = accounts$sales
  x$year = year(accounts$sales$date)
  y = split(x, x$year)
  z = rbind(y[["2015"]], y[["2017"]])
  print("Transactions not in 2016")
  print(z)
  print ("Transactions not in 2016 (head)")
  head(z)
}
#head(findSalesNotIn2016(accounts))
```

Question 3: RECOMPUTE THE TRIAL BALANCE

PART (0)

```
accountTotals = function(accounts) {

  #SALES REVENUE:
  print("Sales Revenue")
  totalSalesRevenue = sum(accounts$sales$total)
  print(totalSalesRevenue)

  #SALES RETURNS:
  print("Sales Returns")
  x = aggregate((returns)*unitprice ~ sku, accounts$inventoryPerpetual, sum)
  print(sum(x$`(returns) * unitprice`))

  #COGS:
  print("COGS")
  totalCOGS = sum(accounts$costOfGoodsSold$COGS)
  print(totalCOGS)

  #ACCOUNTS RECEIVABLE:
  print("Accounts Receivable")
  totalAR = sum(accounts$unpaidAccountsReceivable$total)
  print(sum(accounts$unpaidAccountsReceivable$total))

  #COLLECTIONS:
  print("Collections")
  totalCollections = sum(accounts$arCollections$amt.received)
  print(totalCollections)

  #INVENTORY:
  print("Inventory Perpetual on 1/1/2016")
  print(sum(accounts$inventoryPerpetual$beginstock))
  print("Inventory Perpetual on 12/31/2016")
  print(sum(accounts$inventoryPerpetual$endstock))
  print("Inventory Perpetual Cost on 1/1/2016")
  beginInventoryValue =
sum(accounts$inventoryPerpetual$unitcost*accounts$inventoryPerpetual$beginsto
ck)
  print(beginInventoryValue)
  print("Inventory Perpetual Cost on 12/31/2016")
  endInventoryValue =
sum(accounts$inventoryPerpetual$unitcost*accounts$inventoryPerpetual$endstock
)
  print(endInventoryValue)

  #PURCHASES:
  print("Purchases Cost")
  totalPurchasesCost =
sum(accounts$purchases$unitcost*accounts$purchases$quantity)
  print(totalPurchasesCost)

  #EMPLOYEE REIMBURSEMENTS:
  print("Employee Reimbursements total")
  totalEmployeeReimbursements = sum(accounts$empReimbursements$Amount)
  print(totalEmployeeReimbursements)

}

accountTotals(accounts2016)
```

```
## [1] "Sales Revenue"
## [1] 960030574
## [1] "Sales Returns"
## [1] 2014072
## [1] "COGS"
## [1] 350802594
## [1] "Accounts Receivable"
## [1] 333286020
## [1] "Collections"
## [1] 650887909
## [1] "Inventory Perpetual on 1/1/2016"
## [1] 25086639
## [1] "Inventory Perpetual on 12/31/2016"
## [1] 25059323
## [1] "Inventory Perpetual Cost on 1/1/2016"
## [1] 151790200
## [1] "Inventory Perpetual Cost on 12/31/2016"
## [1] 152765109
## [1] "Purchases Cost"
## [1] 418576367
## [1] "Employee Reimbursements total"
## [1] 72750312
```

PART (1.a): Foot(total)

- For SALES Foot(total):

```
## [1] "Foot(total) of Sales"

## [1] 960030574
```

PART (1.b): Statistical summary of the transactions in the datasets

```
summarizeAccount = function(accounts) {
  for(i in 1:length(accounts)){
    print(names(accounts[i]))
    print(summary(accounts[[i]]))
  }
}
summarizeAccount(accounts2016)
```

```
## [1] "arConfirmations"
##        X            invoice            cust.no          amt.received
##  Min.   :     4   Length:411248      Length:411248      Min.   : -129.0
##  1st Qu.:229246   Class :character   Class :character   1st Qu.:  246.2
##  Median :458039   Mode  :character   Mode  :character   Median :  629.3
##  Mean   :458228                                         Mean   :  991.4
##  3rd Qu.:687396                                         3rd Qu.: 1343.3
##  Max.   :916833                                         Max.   :15174.1
## [1] "custCredit"
##  customer.no            limit
##  Length:1000        Min.   :131000
##  Class :character   1st Qu.:268750
##  Mode  :character   Median :278000
##                     Mean   :276868
##                     3rd Qu.:286000
##                     Max.   :314000
## [1] "empReimbursements"
##   Receipt.No        Employee.No            Amount
##  Length:12428       Length:12428       Min.   :    1
##  Class :character   Class :character   1st Qu.: 2921
##  Mode  :character   Mode  :character   Median : 5860
##                                        Mean   : 5854
##                                        3rd Qu.: 8781
##                                        Max.   :11706
## [1] "inventoryCounts"
##      sku              defective         endstock        returns
##  Length:2000        Min.   :  55.0   Min.   : 5005   Min.   :  7.0
##  Class :character   1st Qu.: 156.8   1st Qu.: 8750   1st Qu.: 25.0
##  Mode  :character   Median : 226.0   Median :12632   Median : 42.0
##                     Mean   : 315.5   Mean   :12560   Mean   : 62.2
##                     3rd Qu.: 387.0   3rd Qu.:16335   3rd Qu.: 75.0
##                     Max.   :1825.0   Max.   :20112   Max.   :485.0
## [1] "inventoryPerpetual"
##        X              sku               unitcost        unitprice
##  Min.   :   1.0   Length:2000        Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 500.8   Class :character   1st Qu.: 3.940   1st Qu.: 9.838
##  Median :1000.5   Mode  :character   Median : 5.965   Median :15.095
##  Mean   :1000.5                      Mean   : 6.061   Mean   :16.572
##  3rd Qu.:1500.2                      3rd Qu.: 8.070   3rd Qu.:22.260
##  Max.   :2000.0                      Max.   :15.710   Max.   :54.160
##    beginstock       endstock       defective         returns
##  Min.   : 5007   Min.   : 5002   Min.   :  53.0   Min.   :  7.0
##  1st Qu.: 8857   1st Qu.: 8719   1st Qu.: 154.8   1st Qu.: 25.0
##  Median :12576   Median :12602   Median : 225.0   Median : 41.5
##  Mean   :12543   Mean   :12530   Mean   : 313.4   Mean   : 61.8
##  3rd Qu.:16218   3rd Qu.:16305   3rd Qu.: 384.0   3rd Qu.: 74.0
##  Max.   :19996   Max.   :20000   Max.   :1813.0   Max.   :485.0
## [1] "arCollections"
##        X             invoice             cust.no
##  Min.   :      1   Length:660320      Length:660320
##  1st Qu.: 274908   Class :character   Class :character  
##  Median : 548774   Mode  :character   Mode  :character  
##  Mean   : 549552                                        
##  3rd Qu.: 824492                                        
##  Max.   :1099998                                        
##     dateColl           amt.received         year     
##  Min.   :2016-01-01   Min.   :    0.0   Min.   :2016  
##  1st Qu.:2016-05-23   1st Qu.:  244.2   1st Qu.:2016  
##  Median :2016-08-14   Median :  626.6   Median :2016  
##  Mean   :2016-08-04   Mean   :  985.7   Mean   :2016  
##  3rd Qu.:2016-10-25   3rd Qu.: 1338.1   3rd Qu.:2016  
##  Max.   :2016-12-31   Max.   :15002.3   Max.   :2016  
## [1] "purchases"
##        X             sku               unitcost         quantity   
##  Min.   :    1   Length:24000       Min.   : 0.000   Min.   : 976  
##  1st Qu.: 6001   Class :character   1st Qu.: 3.940   1st Qu.:2518  
##  Median :12000   Mode  :character   Median : 5.965   Median :2884  
##  Mean   :12000                      Mean   : 6.061   Mean   :2887  
##  3rd Qu.:18000                      3rd Qu.: 8.070   3rd Qu.:3268  
##  Max.   :24000                      Max.   :15.710   Max.   :4215  
##       date               PO.no                year     
##  Min.   :2016-01-05   Length:24000       Min.   :2016  
##  1st Qu.:2016-03-25   Class :character   1st Qu.:2016  
##  Median :2016-06-17   Mode  :character   Median :2016  
##  Mean   :2016-06-17                      Mean   :2016  
##  3rd Qu.:2016-09-08                      3rd Qu.:2016  
##  Max.   :2016-12-02                      Max.   :2016  
## [1] "sales"
##        X             invoice             sku                 qty        
##  Min.   :      1   Length:1083467     Length:1083467     Min.   :  0.00  
##  1st Qu.: 325200   Class :character   Class :character   1st Qu.: 15.00  
##  Median : 650363   Mode  :character   Mode  :character   Median : 40.00  
##  Mean   : 650261                                         Mean   : 53.44  
##  3rd Qu.: 975510                                         3rd Qu.: 77.00  
##  Max.   :1300000                                         Max.   :433.00  
##   cashtrue            date              unitprice         total        
##  Mode :logical   Min.   :2016-01-01   Min.   : 0.00   Min.   :    0.0  
##  FALSE:916833    1st Qu.:2016-04-01   1st Qu.: 9.84   1st Qu.:  180.6  
##  TRUE :166634    Median :2016-07-01   Median :15.14   Median :  526.0  
##  NA's :0         Mean   :2016-07-01   Mean   :16.58   Mean   :  886.1  
##                  3rd Qu.:2016-10-01   3rd Qu.:22.26   3rd Qu.: 1202.1  
##                  Max.   :2016-12-31   Max.   :54.16   Max.   :15174.1  
##    cust.no              year     
##  Length:1083467     Min.   :2016  
##  Class :character   1st Qu.:2016  
##  Mode  :character   Median :2016  
##                     Mean   :2016  
##                     3rd Qu.:2016  
##                     Max.   :2016  
## [1] "costOfGoodsSold"
##      sku                 X.x            invoice               qty
##  Length:1083467     Min.   :      1   Length:1083467     Min.   :  0.00
##  Class :character   1st Qu.: 325200   Class :character   1st Qu.: 15.00
##  Mode  :character   Median : 650363   Mode  :character   Median : 40.00
##                     Mean   : 650261                      Mean   : 53.44
##                     3rd Qu.: 975510                      3rd Qu.: 77.00
##                     Max.   :1300000                      Max.   :433.00
##   cashtrue            date              unitprice.x        total
##  Mode :logical   Min.   :2016-01-01   Min.   : 0.00   Min.   :    0.0
##  FALSE:916833    1st Qu.:2016-04-01   1st Qu.: 9.84   1st Qu.:  180.6
##  TRUE :166634    Median :2016-07-01   Median :15.14   Median :  526.0
##  NA's :0         Mean   :2016-07-01   Mean   :16.58   Mean   :  886.1
##                  3rd Qu.:2016-10-01   3rd Qu.:22.26   3rd Qu.: 1202.1
##                  Max.   :2016-12-31   Max.   :54.16   Max.   :15174.1
##    cust.no               year           X.y         unitcost
##  Length:1083467     Min.   :2016   Min.   :   1   Min.   : 0.000
##  Class :character   1st Qu.:2016   1st Qu.: 501   1st Qu.: 3.940
##  Mode  :character   Median :2016   Median :1001   Median : 5.960
##                     Mean   :2016   Mean   :1001   Mean   : 6.061
##                     3rd Qu.:2016   3rd Qu.:1500   3rd Qu.: 8.070
##                     Max.   :2016   Max.   :2000   Max.   :15.710
##   unitprice.y      beginstock       endstock       defective
##  Min.   : 0.00   Min.   : 5007   Min.   : 5002   Min.   :  53.0
##  1st Qu.: 9.84   1st Qu.: 8858   1st Qu.: 8722   1st Qu.: 155.0
##  Median :15.14   Median :12575   Median :12603   Median : 225.0
##  Mean   :16.58   Mean   :12544   Mean   :12529   Mean   : 313.7
##  3rd Qu.:22.26   3rd Qu.:16217   3rd Qu.:16304   3rd Qu.: 385.0
##  Max.   :54.16   Max.   :19996   Max.   :20000   Max.   :1813.0
##     returns            COGS
##  Min.   :  7.00   Min.   :   0.00
##  1st Qu.: 25.00   1st Qu.:  69.85
##  Median : 42.00   Median : 201.96
##  Mean   : 61.87   Mean   : 323.78
##  3rd Qu.: 74.00   3rd Qu.: 449.48
##  Max.   :485.00   Max.   :5022.50
## [1] "unpaidAccountsReceivable"
##    invoice               X.x              sku                 qty
##  Length:337361      Min.   :     17   Length:337361      Min.   :  0.00
##  Class :character   1st Qu.: 325013   Class :character   1st Qu.: 21.00
##  Mode  :character   Median : 652251   Mode  :character   Median : 47.00
##                     Mean   : 650967                      Mean   : 59.59
##                     3rd Qu.: 976766                      3rd Qu.: 85.00
##                     Max.   :1300000                      Max.   :433.00
##
##   cashtrue            date               unitprice         total
##  Mode :logical   Min.   :2016-01-01   Min.   : 0.00   Min.   :    0.0
##  FALSE:337361    1st Qu.:2016-08-04   1st Qu.: 9.83   1st Qu.:  244.4
##  NA's :0         Median :2016-10-10   Median :15.10   Median :  627.7
##                  Mean   :2016-09-20   Mean   :16.57   Mean   :  987.9
##                  3rd Qu.:2016-11-25   3rd Qu.:22.26   3rd Qu.: 1339.7
##                  Max.   :2016-12-31   Max.   :54.16   Max.   :15174.1
##
##   cust.no.x             year.x          X.y          cust.no.y
##  Length:337361      Min.   :2016   Min.   : NA      Length:337361
##  Class :character   1st Qu.:2016   1st Qu.: NA      Class :character
##  Mode  :character   Median :2016   Median : NA      Mode  :character
##                     Mean   :2016   Mean   :NaN
##                     3rd Qu.:2016   3rd Qu.: NA
##                     Max.   :2016   Max.   : NA
##                                    NA's   :337361
##     dateColl       amt.received        year.y       notCollected
##  Min.   :NA       Min.   : NA      Min.   : NA      Mode:logical
##  1st Qu.:NA       1st Qu.: NA      1st Qu.: NA      TRUE:337361
##  Median :NA       Median : NA      Median : NA      NA's:0
##  Mean   :NA       Mean   :NaN      Mean   :NaN
##  3rd Qu.:NA       3rd Qu.: NA      3rd Qu.: NA
##  Max.   :NA       Max.   : NA      Max.   : NA
##  NA's   :337361   NA's   :337361   NA's   :337361
## [1] "doubtfulAccounts"
##    invoice               X.x              sku                 qty
##  Length:337361      Min.   :     17   Length:337361      Min.   :  0.00
##  Class :character   1st Qu.: 325013   Class :character   1st Qu.: 21.00
##  Mode  :character   Median : 652251   Mode  :character   Median : 47.00
##                     Mean   : 650967                      Mean   : 59.59
##                     3rd Qu.: 976766                      3rd Qu.: 85.00
##                     Max.   :1300000                      Max.   :433.00
##
##   cashtrue            date              unitprice         total
##  Mode :logical   Min.   :2016-01-01   Min.   : 0.00   Min.   :    0.0
##  FALSE:337361    1st Qu.:2016-08-04   1st Qu.: 9.83   1st Qu.:  244.4
##  NA's :0         Median :2016-10-10   Median :15.10   Median :  627.7
##                  Mean   :2016-09-20   Mean   :16.57   Mean   :  987.9
##                  3rd Qu.:2016-11-25   3rd Qu.:22.26   3rd Qu.: 1339.7
##                  Max.   :2016-12-31   Max.   :54.16   Max.   :15174.1
##
##   cust.no.x             year.x          X.y          cust.no.y
##  Length:337361      Min.   :2016   Min.   : NA      Length:337361
##  Class :character   1st Qu.:2016   1st Qu.: NA      Class :character
##  Mode  :character   Median :2016   Median : NA      Mode  :character
##                     Mean   :2016   Mean   :NaN
##                     3rd Qu.:2016   3rd Qu.: NA
##                     Max.   :2016   Max.   : NA
##                                    NA's   :337361
##     dateColl       amt.received        year.y       notCollected
##  Min.   :NA       Min.   : NA      Min.   : NA      Mode:logical
##  1st Qu.:NA       1st Qu.: NA      1st Qu.: NA      TRUE:337361
##  Median :NA       Median : NA      Median : NA      NA's:0
##  Mean   :NA       Mean   :NaN      Mean   :NaN
##  3rd Qu.:NA       3rd Qu.: NA      3rd Qu.: NA
##  Max.   :NA       Max.   : NA      Max.   : NA
##  NA's   :337361   NA's   :337361   NA's   :337361
##     endDate           daysSincePurchase    interval
##  Min.   :2016-12-31   Length:337361     Min.   :0.0000
##  1st Qu.:2016-12-31   Class :difftime   1st Qu.:0.0000
##  Median :2016-12-31   Mode  :numeric    Median :0.0000
##  Mean   :2016-12-31                     Mean   :0.6416
##  3rd Qu.:2016-12-31                     3rd Qu.:1.0000
##  Max.   :2016-12-31                     Max.   :2.0000
##
## [1] "overlimitCreditApprovals"
##       date              cust.no               trans
##  Min.   :2016-04-13   Length:1000        Min.   :-11475.32
##  1st Qu.:2016-06-07   Class :character   1st Qu.: -3626.09
##  Median :2016-06-28   Mode  :character   Median : -2202.72
##  Mean   :2016-07-02                      Mean   : -2629.86
##  3rd Qu.:2016-07-23                      3rd Qu.: -1213.84
##  Max.   :2016-11-24                      Max.   :   -55.32
##     subTotal
##  Min.   :-8736.986
##  1st Qu.:-1487.470
##  Median : -726.845
##  Mean   :-1123.257
##  3rd Qu.: -302.742
##  Max.   :   -0.782
```

PART (1.c): What does the above results indicate?

- The solution for this shall be inferred from the ***Summary.txt*** file, which was generated as output file.

PART (2): Range of dates of sales, purchases and collections

```
createDailySales = function(accounts) {
  totalSales = accounts$sales
  totalSales$amt = totalSales$qty * totalSales$unitprice
  dailySales = aggregate(amt~date,totalSales,sum)
  accounts[["dailySales"]] = dailySales
  return(accounts)
}


createDailyPurchases = function(accounts) {
  totalPurchases = accounts$purchases
  totalPurchases$amt = totalPurchases$quantity * totalPurchases$unitcost
  dailyPurchases = aggregate(amt~date,totalPurchases,sum)
  accounts[["dailyPurchases"]] = dailyPurchases
  return(accounts)
}


createDailyCollections= function(accounts) {
  totalCollections = accounts$arCollections
  dailyCollections = aggregate(amt.received~dateColl,totalCollections,sum)
  accounts[["dailyCollected"]] = dailyCollections
  return(accounts)
}
```

PART (2.a): Compute the min max quartiles etc:

PART (2.b): Compute daily averages

The above questions shall be solved in a simple way by calling the built-in R functions along with the reusable functions which we created. Since both the questions involves a similar approach, we are going to make use of an unified approach to solve the same (as shown below):

```
accounts2016 = createDailySales(accounts2016)
summary(accounts2016$dailySales)

##       date                  amt
##  Min.   :2016-01-01   Min.   :1758475
##  1st Qu.:2016-04-01   1st Qu.:2022356
##  Median :2016-07-01   Median :2840832
##  Mean   :2016-07-01   Mean   :2623034
##  3rd Qu.:2016-09-30   3rd Qu.:2912291
##  Max.   :2016-12-31   Max.   :3098749

accounts2016 = createDailyPurchases(accounts2016)
summary(accounts2016$dailyPurchases)

##       date                  amt
##  Min.   :2016-01-05   Min.   :34881364
##  1st Qu.:2016-03-25   1st Qu.:34881364
##  Median :2016-06-17   Median :34881364
##  Mean   :2016-06-17   Mean   :34881364
##  3rd Qu.:2016-09-08   3rd Qu.:34881364
##  Max.   :2016-12-02   Max.   :34881364

accounts2016 = createDailyCollections(accounts2016)
summary(accounts2016$dailyCollected)

##     dateColl            amt.received
##  Min.   :2016-01-01   Min.   : 355863
##  1st Qu.:2016-04-01   1st Qu.:1360810
##  Median :2016-07-01   Median :1937318
##  Mean   :2016-07-01   Mean   :1778382
##  3rd Qu.:2016-09-30   3rd Qu.:2292952
##  Max.   :2016-12-31   Max.   :2555145
```

PART (2.c): Do the ranges of dates of sales, purchases and collections lie within the fiscal year (2016) being audited?

From the above, we shall infer that the Range falls within the fiscal year only if filtered data is passed else it doesnot happen.

PART (2.d): If not, what corrections do you need to make to properly conduct the audit calculations you have made previously?

If the range doesnot fall in the audit year, then apply year filter using **lubridate** feature

PART (2.e): Would any of your computed account balances in the Trial Balance change because of your findings?

Computed accounts would not change unless the non filtered data set is used.

Question 3: Employee Expenditure Audit

Implementing Benford's Law

```r
#Benford test
accounts2016$empReimbursements$Employee.No =
as.integer(accounts2016$empReimbursements$Employee.No)
accounts2016$empReimbursements$Receipt.No =
as.integer(accounts2016$empReimbursements$Receipt.No)

auditEmployeeReim = function(accounts) {
amtPerEmployee = aggregate(accounts$empReimbursements$Amount, by =
list(accounts$empReimbursements$Employee.No), sum)
names(amtPerEmployee)[names(amtPerEmployee) == "Group.1"] = "employeeNumber"
names(amtPerEmployee)[names(amtPerEmployee) == "x"] = "Amount"
employeeAmt50000 = amtPerEmployee[which(amtPerEmployee$Amount>=50000),]
accounts[["employeeAmt50000"]] = employeeAmt50000
return(accounts)
}

accounts2016 = auditEmployeeReim(accounts2016)

print(head(accounts2016$employeeAmt50000))
```

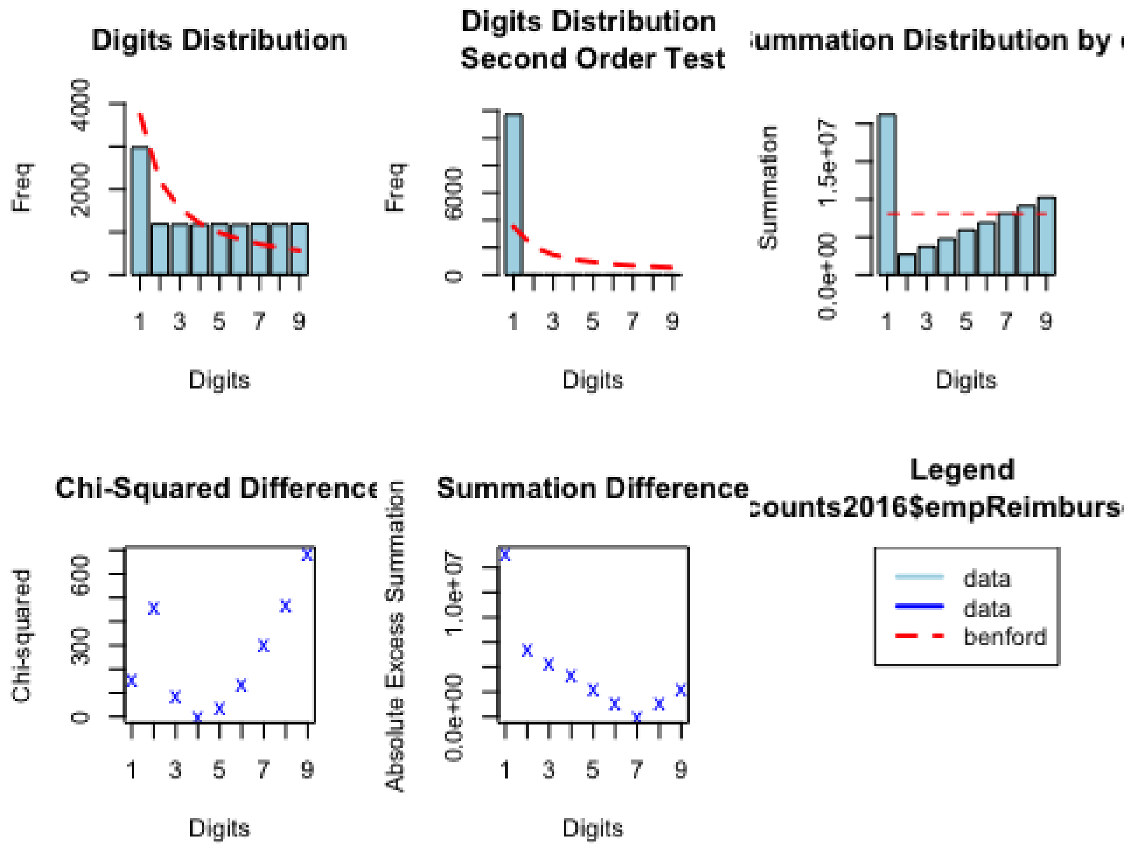
```
##   employeeNumber Amount
## 1              0 719370
## 2              1 713562
## 3              2 630122
## 4              3 735776
## 5              4 740818
## 6              5 745801
```

- Inference: We can see that all the employees have exceeded the spending limit of 50000

```
#Amount
benford_Emp_amount <-
benford(accounts2016$empReimbursements$Amount,number.of.digits = 1, sign =
"both", round = 3 )
benford_Emp_amount

##
## Benford object:
##
## Data: accounts2016$empReimbursements$Amount
## Number of observations used = 12428
## Number of obs. for second order = 11705
## First digits analysed = 1
##
## Mantissa:
##
##    Statistic Value
##         Mean  0.58
##          Var  0.11
##  Ex.Kurtosis -1.10
##     Skewness -0.54
##
##
## The 5 largest deviations:
##
##   digits absolute.diff
## 1      2       1003.46
## 2      1        764.20
## 3      9        624.33
## 4      8        546.28
## 5      7        468.28
##
## Stats:
##
##  Pearson's Chi-squared test
##
## data:  accounts2016$empReimbursements$Amount
## X-squared = 2345.9, df = 8, p-value < 2.2e-16
##
##
##  Mantissa Arc Test
##
## data:  accounts2016$empReimbursements$Amount
## L2 = 0.11869, df = 2, p-value < 2.2e-16
##
## Mean Absolute Deviation: 0.03892285
## Distortion Factor: 24.47812
##
## Remember: Real data will never conform perfectly to Benford's Law. You
should not focus on p-values!
```

```
plot(benford_Emp_amount)
```



```
suspects_amount <- getSuspects(benford_Emp_amount,
accounts2016$empReimbursement, how.many=2)
suspects_amount

##       Receipt.No Employee.No Amount
##    1:       3550          57   1283
##    2:       3551          27  11485
##    3:       3552          23  10400
##    4:       3555          28  10518
##    5:       3558          64   1259
##   ---
## 4158:      15951          39   1915
## 4159:      15953          89   1309
## 4160:      15958           8   1017
## 4161:      15960           1   2848
## 4162:      15962          56   1015
```

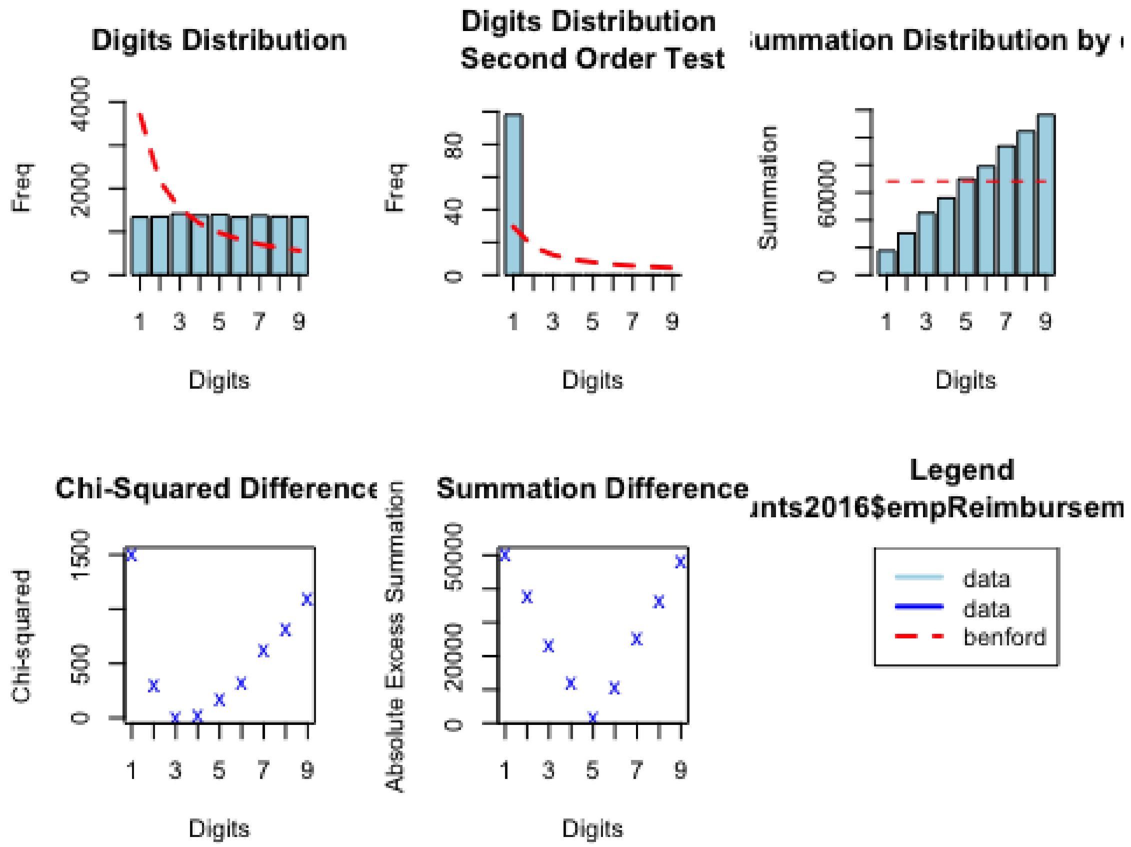
Part (3): Predicted vs actual first digits in Receipt and Employee Number columns

Plots are included in-order to enhance the understandability of the client.

```
#Employee Number
benford_Emp_EmpNo <-
benford(accounts2016$empReimbursements$Employee.No,number.of.digits = 1, sign
= "both", round = 3 )
benford_Emp_EmpNo

##
## Benford object:
##
## Data: accounts2016$empReimbursements$Employee.No
## Number of observations used = 12302
## Number of obs. for second order = 98
## First digits analysed = 1
##
## Mantissa:
##
##    Statistic  Value
##         Mean  0.667
##          Var  0.068
##  Ex.Kurtosis -0.242
##     Skewness -0.809
##
##
## The 5 largest deviations:
##
##   digits absolute.diff
## 1      1       2362.27
## 2      2        819.27
## 3      9        784.09
## 4      8        720.72
## 5      7        665.58
##
## Stats:
##
##  Pearson's Chi-squared test
##
## data:  accounts2016$empReimbursements$Employee.No
## X-squared = 4902.7, df = 8, p-value < 2.2e-16
##
##
##  Mantissa Arc Test
##
## data:  accounts2016$empReimbursements$Employee.No
## L2 = 0.12099, df = 2, p-value < 2.2e-16
##
## Mean Absolute Deviation: 0.05962069
## Distortion Factor: 39.07338
##
## Remember: Real data will never conform perfectly to Benford's Law. You
should not focus on p-values!
```

```
plot(benford_Emp_EmpNo)
```



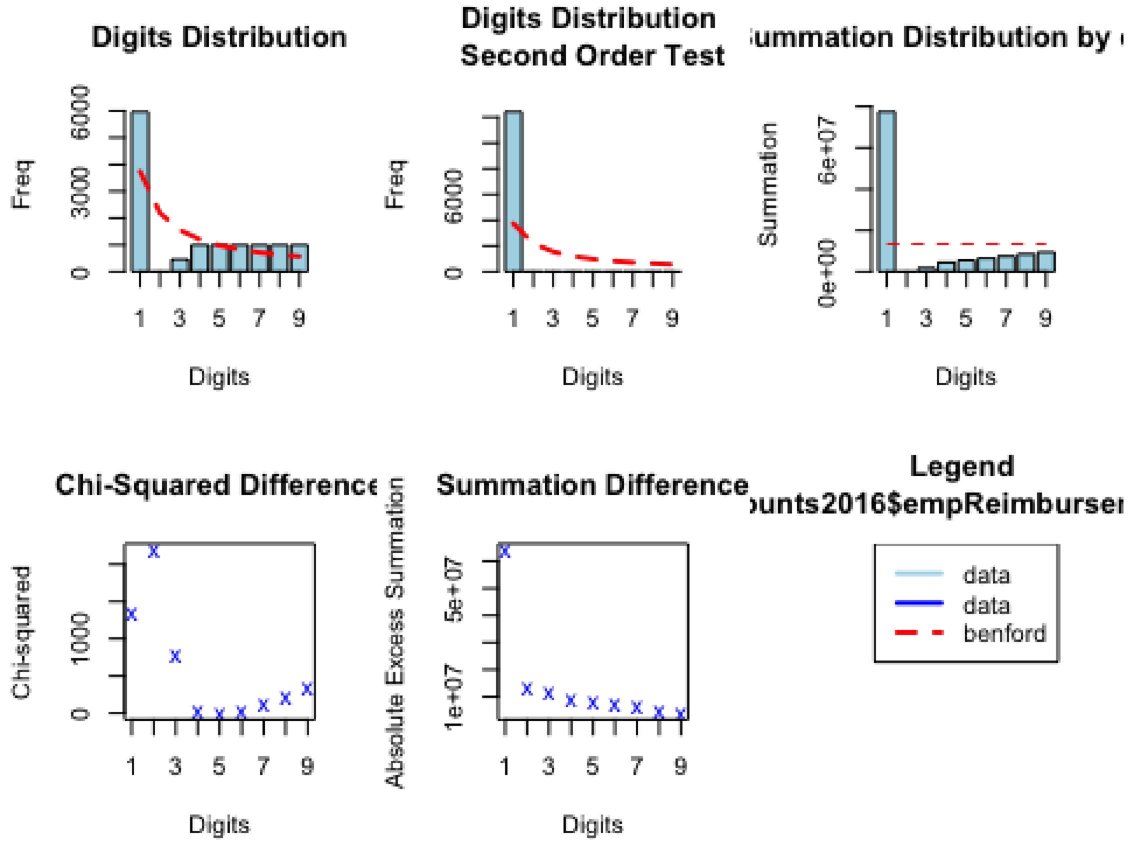
```
suspects_employee <- getSuspects(benford_Emp_EmpNo,
accounts2016$empReimbursement, how.many=2)
suspects_employee

##       Receipt.No Employee.No Amount
##    1:       3542          26   4131
##    2:       3551          27  11485
##    3:       3552          23  10400
##    4:       3554          13   5172
##    5:       3555          28  10518
##   ---
## 2684:      15930          18  11517
## 2685:      15935           2   4484
## 2686:      15955          20   5731
## 2687:      15960           1   2848
## 2688:      15966          20   6993

#Receipts
benford_Emp_Receipts <-
benford(accounts2016$empReimbursements$Receipt.No,number.of.digits = 1, sign
= "both", round = 3 )
benford_Emp_Receipts
```

```
##
## Benford object:
##
## Data: accounts2016$empReimbursements$Receipt.No
## Number of observations used = 12428
## Number of obs. for second order = 12427
## First digits analysed = 1
##
## Mantissa:
##
##    Statistic  Value
##         Mean  0.475
##          Var  0.133
##  Ex.Kurtosis -1.756
##     Skewness  0.058
##
##
## The 5 largest deviations:
##
##   digits absolute.diff
## 1      1       2227.80
## 2      2       2188.46
## 3      3       1093.74
## 4      9        431.33
## 5      8        364.28
##
## Stats:
##
##  Pearson's Chi-squared test
##
## data:  accounts2016$empReimbursements$Receipt.No
## X-squared = 4998.5, df = 8, p-value < 2.2e-16
##
##
##  Mantissa Arc Test
##
## data:  accounts2016$empReimbursements$Receipt.No
## L2 = 0.25415, df = 2, p-value < 2.2e-16
##
## Mean Absolute Deviation: 0.06234307
## Distortion Factor: 5.974498
##
## Remember: Real data will never conform perfectly to Benford's Law. You
should not focus on p-values!
```

```r
plot(benford_Emp_Receipts)
```



Part (4): Report any Suspicious findings:

Suspicious findings are reported below:

```
suspects <- getSuspects(benford_Emp_amount, accounts2016$empReimbursement,
how.many=2)
suspects

##       Receipt.No Employee.No Amount
##    1:       3550          57   1283
##    2:       3551          27  11485
##    3:       3552          23  10400
##    4:       3555          28  10518
##    5:       3558          64   1259
##   ---
## 4158:      15951          39   1915
## 4159:      15953          89   1309
## 4160:      15958           8   1017
## 4161:      15960           1   2848
## 4162:      15962          56   1015
```

Question 4: Accounts Receivable Audit

Part (1): UNPAID ACCOUNTS RECEIVABLE

```
print("Unpaid Accounts Receivable")

## [1] "Unpaid Accounts Receivable"

totalAR = sum(accounts2016$unpaidAccountsReceivable$total)
print(sum(accounts2016$unpaidAccountsReceivable$total))

## [1] 333286020
```

Part (2): ALLOWANCE FOR DOUBTFUL ACCOUNTS

```
print("Uncollected Accounts Receivable")

## [1] "Uncollected Accounts Receivable"

accounts2016 = createUnpaidAccountsReceivable(accounts2016)
print(sum(accounts2016$unpaidAccountsReceivable$total))

## [1] 333286020

print("Allowance for Doubtful Accounts")

## [1] "Allowance for Doubtful Accounts"

accounts2016 = createAllowanceForDoubtfulAccounts(accounts2016)
doubtfulTotals = aggregate(total~interval, accounts2016$doubtfulAccounts,
sum)
print(0.3*doubtfulTotals$total[2] + 0.5*doubtfulTotals$total[3])

## [1] 58398058
```

Part (4): SALES CUT OFF TEST

```
findSalesNotIn2016 = function(accounts) {
  x = accounts$sales
  x$year = year(accounts$sales$date)
  y = split(x, x$year)
  z = rbind(y[["2015"]], y[["2017"]])
  print("Transactions not in 2016")
  print(z)
  print ("Transactions not in 2016 (head)")
  head(z)
}
#head(findSalesNotIn2016(accounts))
```

Part (6 a)

```
d=1000000/333286020
library(pwr)
pwr.t.test (n = NULL, d = 0.003, sig.level = 0.05, power = 0.8, type =
"one.sample")
```

```
##
##      One-sample t test power calculation
##
##               n = 872097.5
##               d = 0.003
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

```
mergeSalesAndARConfirmations = function(accounts) {
  allARAccounts = merge(accounts$arCollections, accounts$arConfirmations,
by="invoice", all.x = T)
  allARAccounts = subset(allARAccounts, select = c(invoice, amt.received.x,
amt.received.y))
  allARAccounts = na.omit(allARAccounts)
  accounts[["allARConfirmationsAndCollections"]] = allARAccounts
  return(accounts)
}
accounts2016 = mergeSalesAndARConfirmations(accounts2016)
```

Part (6 b):

The Percentage Error is given below:

```
sampleConfirmation =
accounts2016$allARConfirmationsAndCollections[ppss(accounts2016$allARConfirma
tionsAndCollection$amt.received.y, 1483718),]
distinctSampleConfirmation = unique(sampleConfirmation)
difference = sum(distinctSampleConfirmation$amt.received.y -
distinctSampleConfirmation$amt.received.x)
totalConfirmedAmounts = sum(distinctSampleConfirmation$amt.received.y)
percentageError = (difference/totalConfirmedAmounts)*100
percentageError
```

```
## [1] 0.02969088
```

Part (7):

- The error percentage i.e. percentage change in audited value against the recorded values is observed to be around 0.029% percent.
- Since the error is too negligible we can consider it to be more or less accurate

Question 5: Inventory Audit

Part 1:

The total cost of Goods sold is accounted for the year 2016 and is given below:

```
accounts2016 = createCostofGoodsSold(accounts2016)
sum(accounts2016$costOfGoodsSold$COGS)
```

```
## [1] 350802594
```

Part 1 a:

- The accounting principle which is important in accurately making this calculation is the **Matching Principle**. In accrual accounting, the matching principle states that expenses should be recorded during the period in which they are incurred, regardless of when the transfer of cash occurs.

Part 2 a:

The detailed summary of the MarkUp percentages (Max, Min, Quartiles) are computed and shown below:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   0.503   1.078   1.745   1.739   2.374   3.000   11004
```

Part 3 a: Stocked out

```
findOutOfStockDemand = function(accounts) {
  library(plyr)
  #prepare tables
  sales = subset(accounts$sales, select = c(sku, date, qty))
  sales$qty = sales$qty*-1

  purchases = accounts$purchases
  purchases$qty = purchases$quantity
  purchases = subset(purchases, select = c(sku, date, qty))

  inventoryTrans = rbind(sales, purchases)
  inventoryTrans = arrange(inventoryTrans, date)

  #Create dataframe by sku
  inventoryTransBySku = split(inventoryTrans, inventoryTrans$sku)

  stockOutSkus = list()
  for(i in 1:length(inventoryTransBySku)) {
    sku = inventoryTransBySku[[i]]
    skuNumber = as.numeric(sku[1,]$sku)
    sku$onHand = accounts$inventoryPerpetual[skuNumber,]$beginstock

    for(n in 1:length(sku$qty)) {
      if(n == 1) {
        sku[n,]$onHand = sku[n,]$onHand + sku[n,]$qty
      }
      else {
        sku[n,]$onHand = sku[n-1,]$onHand + sku[n,]$qty
      }
    }
    if(sum(sku$onHand < 0) > 0) {
      stockOutSkus[[length(stockOutSkus) + 1]] = skuNumber
    }
    inventoryTransBySku[[i]] = sku
  }
  stockOutTrans = data.frame()
  for(i in 1:length(stockOutSkus)){
    skuNumber = stockOutSkus[[i]]
    sku = inventoryTransBySku[[as.character(skuNumber)]]
    times = which(diff(sign(sku$onHand)) > 0)
    for(n in 1:length(times)) {
      stockOutTrans = rbind(stockOutTrans, sku[times[n],])
    }
  }
  accounts[["stockOutTrans"]] = stockOutTrans
  return(accounts)
}
accounts2016 = findOutOfStockDemand(accounts2016)
accounts2016$stockOutTrans = na.omit(accounts2016$stockOutTrans)
head(accounts2016$stockOutTrans$sku)
```

```
## [1] "1084" "1084" "1095" "1124" "1230" "1230"
```

Part 4(a)

```
d=1000000/152765109
d
```

```
## [1] 0.006545997
```

```
library(pwr)
pwr.t.test (n = NULL, d = 0.0065, sig.level = 0.05, power = 0.8, type =
"one.sample")
```

```
##
##      One-sample t test power calculation
##
##               n = 185773.8
##               d = 0.0065
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

```
mergeInventoryPerpetualAndCounts = function(accounts) {
  allInventory = merge(accounts$inventoryPerpetual, accounts$inventoryCounts,
by="sku", all.x = T)
  allInventory = subset(allInventory, select = c(sku, beginstock,endstock.x,
endstock.y,unitcost,defective.y,returns.y))
  allInventory = na.omit(allInventory)
  accounts[["allInventoryMatched"]] = allInventory
  return(accounts)
}
accounts2016 = mergeInventoryPerpetualAndCounts(accounts2016)
```

Part (4 b):

The **Percentage Error** is computed and displayed below:

```
sampleConfirmation =
accounts2016$allInventoryMatched[ppss(accounts2016$allInventoryMatched$endsto
ck.y, 185774),]
distinctSampleConfirmation = unique(sampleConfirmation)
sum(distinctSampleConfirmation$endstock.x)

## [1] 25059323

difference = sum(distinctSampleConfirmation$endstock.y -
distinctSampleConfirmation$endstock.x)
totalConfirmedAmounts = sum(distinctSampleConfirmation$endstock.y)
percentageError = (difference/totalConfirmedAmounts)*100
percentageError

## [1] 0.241898
```

Part (4 c):

The inventory is overstatied by 0.24 % and this would impact the balance sheet. But, this would impact only to a minimal extent.

Part 5: Foot total(inventory accounts balance -> endstock x unitprice)

```
totalInventoryBalanceAfterAdjusting =
sum(accounts2016$allInventoryMatched$endstock.y*accounts2016$allInventoryMatc
hed$unitcost)
totalInventoryBalanceAfterAdjusting

## [1] 153129104
```

- From the above, we shall infer that, Difference: $364,104 after computing the inventory counts, this indicates there is a deviation from the stated trial balance value

Part 6 : Ageing of Inventory

The aged inventory total is computed and given as follows:

```
createInventoryAgeingData = function(accounts){
  inventoryAgeing = merge(accounts$sales, accounts$allInventoryMatched,
by="sku", all.x=T)
  inventoryAgeing = subset(inventoryAgeing, select = c(sku, date,
qty,unitcost,beginstock,endstock.y,total))
  inventoryAgeing$COGS = inventoryAgeing$unitcost * inventoryAgeing$qty
  inventoryAgeing$AvgInvCost = ((inventoryAgeing$endstock.y +
inventoryAgeing$beginstock)* inventoryAgeing$unitcost / 2)
  inventoryAgeing$turnover = inventoryAgeing$COGS/inventoryAgeing$AvgInvCost
  accounts[["inventoryAgeing"]] = inventoryAgeing
  return(accounts)
}
```

```
accounts2016 = createInventoryAgeingData(accounts2016)
names(accounts2016$inventoryAgeing)[names(accounts2016$inventoryAgeing) ==
"endstock.y"] = "endstock"

createInventoryAgeingFinal = function(accounts){
  accountsInventoryAgeingSorted=accounts$inventoryAgeing
  accountsInventoryAgeingSortedFiltered = sqldf("Select sku, sum(qty) as
qty,unitcost,endstock,AvgInvCost from accountsInventoryAgeingSorted group by
sku")
  accountsInventoryAgeingSortedFiltered$COGS =
accountsInventoryAgeingSortedFiltered$qty*accountsInventoryAgeingSortedFilter
ed$unitcost
  accountsInventoryAgeingSortedFiltered$turnOverRatio =
accountsInventoryAgeingSortedFiltered$COGS/accountsInventoryAgeingSortedFilte
red$AvgInvCost
  #accountsInventoryAgeingSortedFiltered =
accountsInventoryAgeingSortedFiltered[!(accountsInventoryAgeingSortedFiltered
$turnOverRatio==0),]
  accountsInventoryAgeingSortedFiltered$age = 365 /
accountsInventoryAgeingSortedFiltered$turnOverRatio
  accounts[["inventoryAgeingFinal"]] = accountsInventoryAgeingSortedFiltered
  return(accounts)
}
accounts2016 = createInventoryAgeingFinal(accounts2016)
accounts2016_backup = accounts2016

#head(accounts2016$inventoryAgeingFinal)
effectiveCostUnderSixty=0
effectiveCostOverSixtyLessOneEighty=0
effectiveCostOver180Less365=0
effectiveCostOver365=0
i=as.integer()
accounts2016$inventoryAgeingFinal$age =
as.numeric(accounts2016$inventoryAgeingFinal$age)
#na.omit(accounts2016$inventoryAgeingFinal)

inventoryAgeingCheckData = accounts2016$inventoryAgeingFinal
#inventoryAgeingCheckData[complete.cases(inventoryAgeingCheckData),]


for (i in 1:2000){
  #print(i)
  #print(accounts2016$inventoryAgeingFinal$age[i])
  if(is.na(accounts2016$inventoryAgeingFinal$age[i])){
    next
  }
  if(accounts2016$inventoryAgeingFinal$age[i] < 60){
    effectiveCostUnderSixty = effectiveCostUnderSixty +
(accounts2016$inventoryAgeingFinal$unitcost[i]*accounts2016$inventoryAgeingFi
nal$endstock[i])
  }else
    if(accounts2016$inventoryAgeingFinal$age[i]>=60 &&
accounts2016$inventoryAgeingFinal$age[i]<180){
      effectiveCostOverSixtyLessOneEighty =
effectiveCostOverSixtyLessOneEighty +
(0.50)*(accounts2016$inventoryAgeingFinal$unitcost[i]*accounts2016$inventoryA
geingFinal$endstock[i])
    }else
      if(accounts2016$inventoryAgeingFinal$age[i]>=180 &&
accounts2016$inventoryAgeingFinal$age[i]<365){
        effectiveCostOver180Less365 = effectiveCostOver180Less365 +
(accounts2016$inventoryAgeingFinal$unitcost[i]*accounts2016$inventoryAgeingFi
nal$endstock[i])
      }else{
        effectiveCostOver365 = effectiveCostOver365 +
(accounts2016$inventoryAgeingFinal$unitcost[i]*accounts2016$inventoryAgeingFi
nal$endstock[i])
      }
}

agedInventoryTotal = effectiveCostUnderSixty +
effectiveCostOverSixtyLessOneEighty + effectiveCostOver180Less365 +
effectiveCostOver365
agedInventoryTotal

## [1] 106273976
```

The computed value for **effectiveCostUnderSixty** is given below:

```
effectiveCostUnderSixty

## [1] 0
```

The computed value for **effectiveCostOverSixtyLessOneEighty** is given below:

```
effectiveCostOverSixtyLessOneEighty

## [1] 46855128
```

The computed value for **effectiveCostOver180Less365** is given below:

```
effectiveCostOver180Less365

## [1] 59418847
```

The computed value for **effectiveCostOver365** is given below:

```
effectiveCostOver365

## [1] 0
```

Part 6 a

The Percentage of total less than 60

```
percentageOfTotalLess60 = (effectiveCostUnderSixty/agedInventoryTotal)*100
percentageOfTotalLess60

## [1] 0
```

Part 6 b

The Percentage of total computed for the range between 60 and 180

```
percentageOfTotalOver60Less180 =
(effectiveCostOverSixtyLessOneEighty/agedInventoryTotal)*100
percentageOfTotalOver60Less180

## [1] 44.089
```

Part 6 c

The Percentage of total computed for the range between 180 and 365

```
percentageOfTotalOver180Less365 =
(effectiveCostOver180Less365/agedInventoryTotal)*100
percentageOfTotalOver180Less365

## [1] 55.911
```

Part 6 d

The Percentage of total computed for the range above 365

```
percentageOfTotalOver365 = (effectiveCostOver365/agedInventoryTotal)*100
percentageOfTotalOver365

## [1] 0
```

Part 7:

```
counter=0
for(i in 1:2000){
  if(is.na(accounts2016$inventoryAgeingFinal$COGS[i])){
    print("NA")
    print(i)
    print("NA")
    next
  }
if((accounts2016$inventoryAgeingFinal$COGS[i]/accounts2016$inventoryAgeingFin
al$endstock[i]) < 10){
    print(accounts2016$inventoryAgeingFinal$sku[i])
    counter=counter+1
}
}
counter

## [1] 629
```

- Inference : There are a total of 628 unique SKUs that had a turnover of less than 10 times.

Part 8: Market Test Inventory

```
marketTestInventory = function(accounts)
{
  inventoryPerpMarketTest = subset(accounts$inventoryPerpetual, select =
c(sku, unitprice, unitcost))
  InventoryMarketTest =
merge(accounts$inventoryCounts,inventoryPerpMarketTest,by="sku")
  InventoryMarketTest$diff = (InventoryMarketTest$unitprice-
InventoryMarketTest$unitcost) * InventoryMarketTest$endstock
  accounts[["InventoryMarketTest"]] = InventoryMarketTest
  return(accounts)
}

accounts2016 = marketTestInventory(accounts2016)

#print(head(accounts2016$InventoryMarketTest[InventoryMarketTest$diff < 0,]))
#NULL
```

Part 9 and 10 (Preface)

```
salesInventoryMerge =
merge(accounts2016$sales,accounts2016$allInventoryMatched,by="sku")
aggregateQuantity=aggregate(salesInventoryMerge$qty,by=list(salesInventoryMer
ge$sku),sum)
names(aggregateQuantity)[names(aggregateQuantity) == "Group.1"] = "sku"
names(aggregateQuantity)[names(aggregateQuantity) == "x"] = "qty"
head(salesInventoryMerge)

##   sku      X invoice qty cashtrue       date unitprice total cust.no year
## 1   1 505903  505903   4     TRUE 2016-10-12       5.7  22.8     373 2016
## 2   1 278696  278696 122    FALSE 2016-08-02       5.7 695.4     606 2016
## 3   1 962588  962588  12    FALSE 2016-07-22       5.7  68.4     106 2016
## 4   1 454907  454907   2    FALSE 2016-05-24       5.7  11.4     882 2016
## 5   1 688592  688592  39    FALSE 2016-12-25       5.7 222.3     427 2016
## 6   1 917373  917373 104    FALSE 2016-06-18       5.7 592.8     527 2016
##   beginstock endstock.x endstock.y unitcost defective.y returns.y
## 1       6714      12175      12344     3.73         100        12
## 2       6714      12175      12344     3.73         100        12
## 3       6714      12175      12344     3.73         100        12
## 4       6714      12175      12344     3.73         100        12
## 5       6714      12175      12344     3.73         100        12
## 6       6714      12175      12344     3.73         100        12

head(aggregateQuantity)
```

```
##    sku   qty
## 1    1 14338
## 2   10 30161
## 3  100 25475
## 4 1000 29117
## 5 1001 28488
## 6 1002 27687
```

```
salesInventoryMerge =
merge(salesInventoryMerge[,c('sku','unitprice','unitcost','beginstock','endst
ock.y')],aggregateQuantity,by="sku")
salesInventoryMerge=unique((salesInventoryMerge))
head(salesInventoryMerge)
```

```
##       sku unitprice unitcost beginstock endstock.y   qty
## 1       1      5.70     3.73       6714      12344 14338
## 283    10      3.32     1.88      13325      11346 30161
## 838   100     19.00     8.07       5341       9374 25475
## 1358 1000     17.23     8.29      17136      16128 29117
## 1912 1001     21.77     5.62      16363       8068 28488
## 2422 1002     23.19     8.78      19098      10995 27687
```

Part 9:

- Nrv < cost where NRV = unitprice - costprice - otherexpenses(which is zero in this case)

```
counter=0
for(i in 1:2000){
  if(is.na(salesInventoryMerge$unitprice[i]) |
is.na(salesInventoryMerge$unitcost[i])){
    print("NA")
    print(i)
    print("NA")
    next
  }
  if((salesInventoryMerge$unitprice[i]-(salesInventoryMerge$unitcost[i])) <
salesInventoryMerge$unitcost[i]){
    print(salesInventoryMerge$sku[i])
    counter=counter+1
  }
}
counter
```

```
## [1] 433
```

- Inference: We arrive at the conclusion that, **433 inventory items** have Net Realizable value less than cost.

Part 10:

- NRV < 110% of cost where NRV = unitprice - costprice - otherexpenses(which is salescommission=10% of unitcost)

```
counter=0
for(i in 1:2000){
  if(is.na(salesInventoryMerge$unitprice[i]) |
is.na(salesInventoryMerge$unitcost[i])){
    print("NA")
    print(i)
    print("NA")
    next
  }
  if((salesInventoryMerge$unitprice[i] - salesInventoryMerge$unitcost[i] -
(0.1 * salesInventoryMerge$unitcost[i])) < (1.1 *
salesInventoryMerge$unitcost[i])){
    print(salesInventoryMerge$sku[i])
    counter=counter+1
  }
}
counter

## [1] 587
```

- Inference: We arrive at the conclusion that, **587 inventory items** have Net Realizable value less than 110% of the cost and Sales Commission that are 10% of cost.

Notes for Questions 5 - Part 9 and 10

- If this calculation does result in a loss, you should charge the loss to the cost of goods sold expense with a debit, and credit the #inventory account to reduce the value of the inventory account. If the loss is material, you may want to segregate it in a separate #loss account, which more easily draws the attention of a reader of a company's financial statements.

- Net realizable value is actually only one of the factors you consider in determining the lower of cost or market, so see the Lower of #Cost or Market article for a complete explanation.

- Net realizable value can also refer to the aggregate total of the ending balances in the trade accounts receivable account and the #offsetting allowance for doubtful accounts. This net amount represents the amount of cash that management expects to realize once it #collects all outstanding accounts receivable.

Part 11 and 12

```
purchasePerSKU = arrange(accounts2016$purchases,accounts2016$purchases$sku)
purchasePerSKU = subset(purchasePerSKU, select = c(sku, quantity))
purchasePerSKU =
aggregate(purchasePerSKU$quantity,by=list(purchasePerSKU$sku),sum)
names(purchasePerSKU)[names(purchasePerSKU) == "Group.1"] = "sku"
names(purchasePerSKU)[names(purchasePerSKU) == "x"] = "quantity"
#purchasePerSKU
mergedPurchaseAndInventory =
merge(purchasePerSKU,accounts2016$allInventoryMatched,by="sku")
```

```
mergedPurchaseAndInventory = subset(mergedPurchaseAndInventory, select =
c(sku,quantity,beginstock,returns.y,defective.y))
mergedPurchaseAndInventory$defectiveRate =
(mergedPurchaseAndInventory$defective.y /
(mergedPurchaseAndInventory$quantity))*100
sum(mergedPurchaseAndInventory$defectiveRate > 1)
```

```
## [1] 566
```

```
mergedPurchaseAndInventory$returnRate = (mergedPurchaseAndInventory$returns.y
/ (mergedPurchaseAndInventory$quantity))*100
sum(mergedPurchaseAndInventory$returnRate > 1)
```

```
## [1] 15
```

```
salesPerSKU =
aggregate(accounts2016$sales$qty,by=list(accounts2016$sales$sku),sum)
names(salesPerSKU)[names(salesPerSKU) == "Group.1"] = "sku"
names(salesPerSKU)[names(salesPerSKU) == "x"] = "quantity"
mergedSalesAndInventory =
merge(salesPerSKU,accounts2016$allInventoryMatched,by="sku")
mergedSalesAndInventory = subset(mergedSalesAndInventory, select =
c(sku,quantity,beginstock,returns.y,defective.y))
mergedSalesAndInventory$defectiveRate = (mergedSalesAndInventory$defective.y
/ (mergedSalesAndInventory$quantity))*100
sum(mergedSalesAndInventory$defectiveRate > 1)
```

```
## [1] 724
```

```
mergedSalesAndInventory$returnRate = (mergedSalesAndInventory$returns.y /
(mergedSalesAndInventory$quantity))*100
sum(mergedSalesAndInventory$returnRate > 2)
```

```
## [1] 0
```