# Report automation and file management using VBA, R, Python

## Introduction

It is usually a pseudo-secret that a lot of work that happens today, mostly in a corporate environment is repetitive, when it comes to tracking daily progress or making daily reports and getting it to someone every day. About two or three decades ago, when much of the work was expected to be done manually, taking care of the repetition meant doing it over and over again until you become an expert in the same that you can’t go wrong.

However, today, with the availability of a plethora of tools, techniques and technological infrastructure, there are easy ways to overcome any repetitive work and remove the redundancy of doing the same operations over and over again. This article introduces you to a way to automate an end-to-end data reporting solution that has to happen every day. It uses nothing but the general tools you would be familiar with if you are working with data – Excel (VBA), R and Python, and would be helpful for anyone involved in generating daily reports and want to remove the redundancy, particularly if your firm is not too sound and steady when it comes to advanced technical infrastructure.

Note:-

1. Parts of the code are inspired by StackOverflow solutions (aren’t we all!)
2. This definitely isn’t the best way to get this done – just one of the ways. Anyone who is interested to improve the flow is welcome to create a branch at [this git repository](https://github.com/abhijithasok/Report-management-automation-using-VBA-R-Python) .

## Scenario

The following is the scenario being tackled:-

There is a daily email, of a dataset that contains the data of all organizational purchases (real estate construction) over the past 1 year, until the previous day. We need to find out those items that were purchased on the previous day whose unit price in that particular purchase is:

1. More than or

2. Less than, the average unit price for the same item over the past 1 year.

These reports would later be used in Time Series implementations and predictions for optimizing purchases over time.

The workbook that contains all such purchases is separated into different tabs (based on whether the unit price has increased or decreased from the past one year average) as well as individual charts of unit prices plotted over time for all items, over all purchases made over the previous year and the raw data. This is are to be sent to a set of people via email every day.

The data-based challenges in creating the workbook can be roughly outlined as follows:

1. The same item could have been purchased in different units (eg: kg, quintal) over the past year. The unit prices would thus be based on the corresponding unit in the raw data, thereby making a direct match error-prone.

2. Some items could be purchased as different forms of quantity (eg: cement, purchased by weight and purchased by volume). Even though they may have different units in the raw data, such purchases of the same item in different forms of quantity are to be treated as two separate items itself.

The workflow for this automation is as follows (on a daily loop):

1. Retrieve the xls file from the corresponding e-mail

2. Convert the xls file to csv (optional step, but it eases the compatibility with R)

3. Perform operations and generate the output workbook and figures in the required format

4. Compress the figures and the original data (so that they fit well into an e-mail

5. Send the e-mail

## Tools

The tool usage is as follows:

1. VB code on Microsoft Outlook that identifies an e-mail with a specific header (that also contains the date of that particular day) and downloads the attachment along with it into a specified folder.

2. VB code on Microsoft Excel that picks up this downloaded xls file and converts it into a csv file of the same name and saves it in the same folder

3. R script that accepts the following inputs:

a. This converted csv file of raw data

b. A list of unit pairs, along with a dividing factor to convert the first unit of each pair to the second unit of each pair

c. A list of unit pairs that have to be treated without any conversion, due to differences in forms of quantity

The R script generates a workbook of 2 tabs - one containing the list of items purchased on the previous day whose price is higher than the average price for that item over the past one year and the second, the list of items in the same context whose price is lower than the past one year average. The R script also generates time-based line plots for each of the items.

4. Python script that takes in the original raw data, as well as the folders that contain the figures generated through the R script and compresses both.

5. R script that creates an e-mail Outlook object and sends an e-mail to specified recipients with the compressed original data, compressed figures(plots) as well as the output workbook from the R script as attachments.

These 5 scripts/codes, can be scheduled one after the other on Windows Task Scheduler for a particular time every day, to keep the flow fully automated.

## File legend

A typical raw data XLS which arrives in the e-mail Inbox everyday has the following variables in it, in order:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Company Name | The company under the conglomerate which made the purchase order |
| Project Name | The project for which the purchase order was made |
| Project code | Project code(ID for project) |
| PO NO | Order number |
| PO date | Order date |
| Vendor Code | UID of vendor who is supplying the order |
| Vender Name | Name of vendor who is supplying the order |
| Item Group Name | Type of item |
| Item code | UID of item group |
| **Item desc** | **Major description of item** |
| **Extra Desc** | **Minor description of item(usually the brand)** |
| UOM | Unit of purchase |
| Major Cate. | Major category code for item group |
| Minor Cate. | Minor category code for item group |
| PO qty | Order quantity |
| Unit Price | Unit price of purchase |
| Basic amt | Total basic amount |
| Tax amt | Tax over basic amount |
| Total amt | Total amount(basic+tax) |
| Payment terms | Payment Plan |
| Prepayment if any | Binary, to flag advance payments |

The file containing the units to be converted, contains the following variables:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| UOM Original | Unit 1 |
| UOM Converted | Unit 2 |
| Dividing factor | A number which when the value in Unit 1 is divided by, gives it in Unit 2 |

The file containing the units to not be converted, in spite of being for the same item, contains the following variables:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Unconversion Past | Unit 1 |
| Unconversion Present | Unit 2 |

## Script detailing

Let’s take a look at the scripts to get the automation done, one-by-one:

1. VB Script on Outlook to auto-download an attachment as it comes.

Note: This VB script would work only if Outlook is open, i.e., it won’t auto-download the attachment unless MS Outlook is open. In a corporate environment where official e-mails are very important, we would ideally expect Outlook to be open during office hours throughout, every day. Hence, this script would suffice.

(In case the attachment has to be downloaded irrespective of whether Outlook is open or not, a Python script has to be used. That is a better option in case you have to work with a Gmail account, rather than an Outlook. However, since we are talking about a corporate environment here, we will stick with Outlook).

First, we define the subroutine and define the variables to use, and their datatypes:

============================== <Code Begin > =============================

Public Sub Extract\_Outlook\_Email\_Attachments()

Dim OutlookOpened As Boolean

Dim outApp As Outlook.Application

Dim outNs As Outlook.NameSpace

Dim outFolder As Outlook.MAPIFolder

Dim outAttachment As Outlook.Attachment

Dim outItem As Object

Dim outMailItem As Outlook.MailItem

Dim inputDate As String, subjectFilter As String

Dim saveInFolder As String

============================== <Code End > ===============================

Next, we define the folder into which we need to save the attachment.

============================== <Code Begin > =============================

saveInFolder = "C:\Users\abhijithasok\Documents\Purchase Daily Dashboards\Raw Data\"

If Right(saveInFolder, 1) <> "\" Then saveInFolder = saveInFolder & "\"

============================== <Code End > ===============================

You can alter ‘saveInFolder’(the location into which the files are to be saved) according to the path in your personal system. Even if you forget to add a ‘\’ at the end of the path, the second step would take care of it, so that the script won’t throw an error in case it cannot access the folder.

The next part is optional. It just creates a prompt with a textbox, in case you want to run the script manually to download attachments of a specific date. If you are sure that the e-mail with the attachment(s) come in at around the same time every day and that the script needs to run only at the prescribed time, you may very well skip the following step:

============================== <Code Begin > =============================

inputDate = InputBox("Enter date to filter the email subject", "Extract Outlook email attachments")

If inputDate = "" Then Exit Sub

============================== <Code End > ===============================

Next, we have to define the subject line of the e-mail that comes in every day, that which to download the attachment from, into the folder path we defined earlier.

============================== <Code Begin > =============================

subjectFilter = "Daily Purchase Item Price detail for Last 1 Year"

============================== <Code End > ===============================

The subject line in this scenario is the same every day. In case your subject line is variable, i.e., with that particular Day’s date included, you can use the ‘Format()’ and ‘Date()’ functions of VBA together in the subject string to automate that as well. For an example of how that works, refer to the explanation of the next VB script in this article(the script for conversion from xls to csv .

Now, we check for the existence of an Outlook object(which can be used like a central node to run Outlook operations through code) that the script can access and run its operations on and create one if it doesn’t exist.

============================== <Code Begin > =============================

OutlookOpened = False

On Error Resume Next

Set outApp = GetObject(, "Outlook.Application")

If Err.Number <> 0 Then

Set outApp = New Outlook.Application

OutlookOpened = True

End If

On Error GoTo 0

============================== <Code End > ===============================

In case the Outlook Application isn’t created or Outlook isn’t open at the scheduled time, the script has to exit.

============================== <Code Begin > =============================

If outApp Is Nothing Then

MsgBox "Cannot start Outlook.", vbExclamation

Exit Sub

End If

============================== <Code End > ===============================

Now, we set the folder from which the particular e-mail has to be accessed. Here, ‘Inbox’ has been given by default, but if you have previously set Outlook rules that automatically download it to some other folder, you can define that accordingly.

============================== <Code Begin > =============================

Set outNs = outApp.GetNamespace("MAPI")

Set outFolder = outNs.Folders("Personal Folders").Folders("Inbox")

============================== <Code End > ===============================

The following code snippet now loops through the folder structure in ‘outFolder’, checks each e-mail’s subject against our preferred subject defined earlier in ‘subjectFilter’, and if matched, downloads the attachments in the same into the path defined in ‘saveInFolder’.

============================== <Code Begin > =============================

If Not outFolder Is Nothing Then

For Each outItem In outFolder.Items

If outItem.Class = Outlook.OlObjectClass.olMail Then

Set outMailItem = outItem

If outMailItem.Subject = subjectFilter Then

Debug.Print outMailItem.Subject

For Each outAttachment In outMailItem.Attachments

outAttachment.SaveAsFile saveInFolder & outAttachment.FileName

Next

End If

End If

Next

End If

============================== <Code End > ===============================

The attachment(s) is now saved into the particular folder. We justinitialize the object parameter to default for the next run now.

============================== <Code Begin > =============================

If OutlookOpened Then outApp.Quit

Set outApp = Nothing

End Sub

============================== <Code End > ===============================

1. Excel VB script to convert xls to csv

This is an optional step. I chose to convert it to csv since I’m more comfortable with it personally, but you can very well skip this step if need be and import the file into R using ‘read.xls’.

The following VBA subroutine will open an xls file of the defined name at the defined path and save it as a csv file at the same location. It’s best to save this script in an independent excel macro file (xlsm).

Note that the file name of the attachment in this scenario is of a flexible form that incorporates that particular day’s date in the name. For example, an attachment received in an e-mail on Jan 5, 2017 would be named “OMX\_PO\_ITEM\_RATE\_ALT\_05-JAN-2017.xls”. The code automates the inclusion of the date as well. Those who were looking to understand how to do this, for the e-mail subject line in the previous VB script should pay particular attention to the following script:

============================== <Code Begin > =============================

Private Sub Workbook\_Open()

Workbooks.Open Filename:="C:\Users\abhijithasok\Documents\Purchase Daily Dashboards\Raw Data\OMX\_PO\_ITEM\_RATE\_ALT\_" & UCase(Format(Date, "dd-mmm-yyyy")) & ".xls"

ActiveWorkbook.SaveAs Filename:="C:\Users\abhijithasok\Documents\Purchase Daily Dashboards\Raw Data\OMX\_PO\_ITEM\_RATE\_ALT\_" & UCase(Format(Date, "dd-mmm-yyyy")) & ".csv", \_

FileFormat:=xlCSV, CreateBackup:=False

ActiveWindow.Close False

ThisWorkbook.Close False

End Sub

============================== <Code End > ===============================

1. R script to perform the major operations, generate the workbook of segregated data and their corresponding line plots.

First, we need to import all required packages into the system. I used R 3.3.2 on RStudio 0.99.903. We don’t ideally need all these packages to get the script running. Many of them are alternatives to reading and writing xls and xlsx files (I used the xlsx package). Many R users have reported errors while reading xls and xlsx into R (also a reason why I preferred to convert it into a csv using a VB script earlier). In case xlsx throws an error, you can try out the functions in one of the other listed packages, like ‘gdata’, ‘XLConnect’ or ‘readxl’.

‘ggplot’ is a package that hardly needs any explanation – it is used to create a plot. ‘ggrepel’ is used to ensure that labels on the ggplot plots don’t overlap and that the plot is readable.

There is no harm in importing all of them into the system, but in case you don’t want to do that, it’s best to follow the code and import packages as and when required.

============================== <Code Begin > =============================

library(colorspace)

library(ggplot2)

library(ggrepel)

library(devtools)

library(readxl)

library(XLConnect)

library(rJava)

library(xlsx)

library(gdata)

library(gtools)

library(dplyr)

library(openxlsx)

library(rtools)

============================== <Code End > ===============================

We have already converted the xls into a csv file. The first step would be to read that data into R. Since the file name is flexible as per the particular day’s date, we use ‘format()’ and ‘Sys.Date()’ to define that in the file name.

============================== <Code Begin > =============================

rawname <- paste0("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/Raw Data/OMX\_PO\_ITEM\_RATE\_ALT\_",toupper(format(Sys.Date(),"%d-%b-%Y")),".csv")

rawdata <- read.csv(file = rawname,header=T,stringsAsFactors = F)

============================== <Code End > ===============================

A file which is received on a particular day contains the data of purchases made until the previous day, starting exactly 1 year ago. Assuming the present day is T, the comparison is to be made between all purchases of T-1 and all purchases of 0 to T-2. The dataset has to be separated accordingly.

============================== <Code Begin > =============================

pastdata <- rawdata[rawdata$PO.date != format(Sys.Date()-1,"%d-%b-%y"),]

presentdata <- rawdata[rawdata$PO.date == format(Sys.Date()-1,"%d-%b-%y"),]

============================== <Code End > ===============================

Make sure you don’t get confused with the data frame names above. ‘pastdata’ refers to all purchases made till 2 days before the present day and ‘presentdata’ refers to the all purchases made on the previous day. They have been named so for ease of comparison.

Next, we need to read two conversion tables to address the specific data challenges that we spoke about in the beginning. One of the tables contain the different unit combination possibilities for purchased items and a corresponding ‘dividing factor’ that would let you convert the first unit into the second, in a row(eg: kg, quintal). The second conversion table lists out the unit combinations that, even if appearing in different purchases for the same item, should be treated differently(eg: weight units, volume units).

Here, ‘uomcon’ represents the former and ‘uomuncon’ represents the latter.

============================== <Code Begin > =============================

uomcon <- read.csv("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/UOM Conversion list.csv",header=T)

uomuncon <- read.csv("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/UOM Unconversion.csv",header=T)

============================== <Code End > ===============================

Before the operations begin, it is advisable to tag the entries from ‘presentdata’ and ‘pastdata’ separately with ID numbers for easy identification and manipulation. This can be done using the date of purchase as well, comparing it to check whether it is the previous day’s date or not but this just eases the process and removes the comparison step every time.

============================== <Code Begin > =============================

pastdata$pastID <- ""

pastdata$pastID <- 1 : nrow(pastdata) #Unique ID for past instances

presentdata$presentID <- ""

presentdata$presentID <- 1 : nrow(presentdata) #Unique ID for present instances

============================== <Code End > ===============================

The next step is to make sure the differences in the units of purchases for the same item are cleared before comparisons are made. The primary key to identify unique items is a set of 2 description columns, namely ‘Item.desc’ and ‘Extra.Desc’, representing the major and minor descriptions of the item. The ‘Item.desc’ column typically contains information about what the item is and the ‘Extra.Desc’ column typically contains information about what particular brand of item in the present purchase. In order to compare between the unit of purchase of a purchase made in the past and its unit in the purchase made on the previous day(if any), we compare between the ‘Item.desc’ and ‘Extra.Desc’ columns of ‘pastdata’ and ‘presentdata’ and find all existing matches for every purchase in ‘pastdata’ and add the corresponding unit from the ‘presentdata’ purchase into a new column in ‘pastdata’. This is a parallel to Excel’s ‘INDEX MATCH with multiple criteria’ and can be achieved in R using the ‘merge’ function, followed by removing any duplicate entries for the purchase of the same item on the same day(since we are interested in the unit price and not in overall amount spent).

============================== <Code Begin > =============================

m <- merge(pastdata,presentdata[,c("Item.desc","Extra.Desc","UOM")],by = c("Item.desc","Extra.Desc"),all.x=T)

m <- m[!duplicated(m),]

============================== <Code End > ===============================

Here, ‘UOM’ refers to ‘Unit of Measurement’. We rename the original UOM column of ‘pastdata’, back to ‘UOM’ and the new UOM column created using UOMs of ‘presentdata’, as ‘Present.UOM’. We also remove the ‘pastID’ column from the newly made data frame ‘m’ since the merge operation is complete and we no longer need it.

============================== <Code Begin > =============================

colnames(m)[which(names(m) == "UOM.x")] <- "UOM"

colnames(m)[which(names(m) == "UOM.y")] <- "Present.UOM"

m$pastID<-NULL

============================== <Code End > ===============================

Now that we have UOMs of every item in ‘pastdata’ and their corresponding UOMs in ‘presentdata’ if any, we can compare between them and create a flag variable to flag those past purchases where the units are different.

============================== <Code Begin > =============================

m$Different.UOM<-""

m$Different.UOM <- ifelse(m$UOM == m$Present.UOM,0,1)

============================== <Code End > ===============================

Now that the ‘Different.UOM’ shows us the purchases are where the units are different between past and present, we need to flag those rows where this difference is valid, i.e. they have to be treated as different items due to the difference in the type of unit(eg: by weight and by volume – cubic meter and kg). To do this, we create a new flag in the ‘pastdata’ data frame.

The information of which all <’UOM’, ‘Present.UOM’> combinations are to be treated as different items are present in the ‘uomuncon’ data frame which we read in before. To ease the process, we add a flag variable to the ‘uomuncon’ data frame with a value of 1 in all rows, so that merely by matching the item descriptions, we can fill in the flag variable in ‘pastdata’ with 1 in all rows that match a row in uomuncon. This matching is again done using ‘merge’, followed by duplicate clearance and column renaming.

============================== <Code Begin > =============================

m$Unconversion.Flag <- ""

uomuncon$Unconversion.Flag <- 1

m <- merge (m, uomuncon, by.x=c("UOM","Present.UOM"), by.y=c("Unconversion.Past", "Unconversion.Present"), all.x=TRUE)

m$Unconversion.Flag.x<-NULL

colnames(m)[which(names(m) == "Unconversion.Flag.y")] <- "Unconversion.Flag"

============================== <Code End > ===============================

Now that we have identified which are the purchases that have to be left without unit conversion in spite of being the same item, all other purchases need a conversion of the unit price into the unit of the particular purchase that happened on the previous day. We could have chosen the unit of item in ‘pastdata’ as well, but since our exercise is to compare the ‘present’ with everything else in the past, it’s better to assume the ‘present’ as a standard.

This is done through another ‘merge’ operation, followed by an if condition, that feeds in the converted value of the unit price to the new column ‘Unit.Price.in.Present.Unit’ using the corresponding dividing factors present in the ‘uomcon’ data frame.

============================== <Code Begin > =============================

m$Unit.Price.in.Present.Unit<-""

m <- merge(m,uomcon,by.x=c("UOM","Present.UOM"),by.y=c("UOM.Original","UOM.Convert"),all.x=TRUE)

m$Unit.Price.in.Present.Unit <- ifelse(m$Different.UOM==1 & m$Unconversion.Flag == 0, m$Unit.Price/m$Dividing.factor, m$Unit.Price)

============================== <Code End > ===============================

Now that we have converted the unit prices in the unit of the ‘presentdata’ purchase wherever applicable, we need a final list of the units themselves that we are using. When there was no unit conversion that happened, we just carry the unit of the ‘pastdata’ entry forward and if a conversion had happened, we carry the unit of the ‘presentdata’ entry forward. This ensures that any purchase that had its unit converted would be reported in the ‘presentdata’ unit and that which did not have its unit converted would be reported in the ‘pastdata’ unit(which would in effect, take care of those rows where the units were indeed different, but were not meant to be converted).

============================== <Code Begin > =============================

m$Converted.UOM <- ""

m$Converted.UOM <- ifelse(m$UOM == m$Present.UOM, m$UOM, ifelse (m$Unconversion.Flag == 0, m$Present.UOM, m$UOM))

============================== <Code End > ===============================

This concludes our operations directly on ‘pastdata’. It would now have the following columns in it(the ones in blue are the newly created columns):

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Company Name | The company under the conglomerate which made the purchase order |
| Project Name | The project for which the purchase order was made |
| Project code | Project code(UID for project) |
| PO NO | Order number |
| PO date | Order date |
| Vendor Code | UID of vendor who is supplying the order |
| Vender Name | Name of vendor who is supplying the order |
| Item Group Name | Type of item |
| Item code | UID of item group |
| **Item desc** | **Major description of item** |
| **Extra Desc** | **Minor description of item(usually the brand)** |
| UOM | Unit of purchase |
| Major Cate. | Major category code for item group |
| Minor Cate. | Minor category code for item group |
| PO qty | Order quantity |
| Unit Price | Unit price of purchase |
| Basic amt | Total basic amount |
| Tax amt | Tax over basic amount |
| Total amt | Total amount(basic+tax) |
| Payment terms | Payment Plan |
| Prepayment if any | Binary, to flag advance payments |
| **Present UOM** | **Corresponding Present unit for the particular past purchase** |
| **Different UOMs** | **Flag signifying which all past purchases have differing units from present** |
| **Unconversion Flag** | **Flag signifying which all purchases are to be left without converting the unit, among those that have different past and present units** |
| **Unit Price in Present Unit** | **Unit Price of the item for that particular past purchase, converted into the unit of the item in the present purchase, if applicable** |
| **Converted UOM** | **The final unit, converted or otherwise, as per each purchase** |

Now, we move to the operations on ‘presentdata’, using this final version of ‘pastdata’, to create the final reports to be sent over.

First, we clear off ‘presentID’ which we used initially to match the units between past and present, and then store the original version of ‘presentdata’ into another variable, so that it can be used for operations and the original version can be preserved for any later reference.

============================== <Code Begin > =============================

presentdata$presentID <-NULL

n <- presentdata

============================== <Code End > ===============================

Now that we have final converted units along with the converted unit prices in the ‘pastdata’ data frame, we compute the minimum and maximum historical prices over the past year, starting two days before the current date, for every purchase made in the ‘presentdata’, i.e. the purchases of the previous day, by matching ‘Item.desc’, ‘Extra.Desc’ and the newly made ‘Converted.UOM’. These are then copied into the ‘presentdata’ table using ‘merge’ and the columns are renamed.

============================== <Code Begin > =============================

aggmin <- aggregate (Unit.Price.in.Present.Unit ~ Item.desc + Extra.Desc + Converted.UOM, m, function(x) min(x) )

aggmax <- aggregate (Unit.Price.in.Present.Unit ~ Item.desc + Extra.Desc + Converted.UOM, m, function(x)max(x))

minmerge <- merge (n, aggmin, by.x = c("Item.desc","Extra.Desc","UOM"), by.y = c("Item.desc","Extra.Desc","Converted.UOM"), all.x=T)

maxmerge <- merge (minmerge, aggmax, by.x = c("Item.desc","Extra.Desc","UOM"), by.y = c("Item.desc","Extra.Desc","Converted.UOM"), all.x=T)

n <- maxmerge

colnames(n)[which(names(n) == "Unit.Price.in.Present.Unit.x")] <- "Past.Min.Price"

colnames(n)[which(names(n) == "Unit.Price.in.Present.Unit.y")] <- "Past.Max.Price"

============================== <Code End > ===============================

Now, we calculate the average price from the past. Note that in this case, the average price is simply the average of the maximum and minimum prices for that particular present purchase and not the average among all the purchases made in the past year. This is because, what is of interest here is to gauge the range of prices that were received in the past year for a comparison with the present price and importance isn’t given to the quantity of the item purchased at a particular unit price. In case you would like to have the average prices over the entire year, it is just a matter of another ‘aggregate’ function.

============================== <Code Begin > =============================

n$Past.Avg.Price <- ""

n$Past.Avg.Price <- (n$Past.Min.Price + n$Past.Max.Price)/2

============================== <Code End > ===============================

Based on this average unit price and the present unit price of the item, we mark the particular purchase as ‘INCREASE’ for an increase in unit price from past average, ‘DECREASE’ for a decrease in unit price from past average, ‘NO CHANGE’, for the unit price remaining the same and ‘NEW ITEM’, if the purchased item was never bought in the past. This is done through a new variable defined as ‘Inc.Dec.Prev.Avg’. Accordingly, we also calculate the amount of change in the unit price and the percentage change in the same with respect to the past unit price, as their own variables. In addition, we also rearrange the order of the variables in the data frame so that all the original variables present in the original data come first, followed by the variables we just created.

============================== <Code Begin > =============================

n$Inc.Dec.Prev.Avg <- ""

n$Inc.Dec.Prev.Avg <- ifelse(n$Past.Avg.Price != 0, ifelse (n$Unit.Price > n$Past.Avg.Price, "INCREASE", ifelse(n$Unit.Price<n$Past.Avg.Price,"DECREASE", ifelse (n$Unit.Price == n$Past.Avg.Price, "NO CHANGE","NEW ITEM"))),"NEW ITEM")

n$Change.Amount <- ""

n$Change.Amount <- abs(n$Unit.Price - n$Past.Avg.Price) #Change amount

n$Change.Percentage <- ""

n$Change.Percentage <- ifelse (!is.na ((n$Change.Amount/n$Past.Avg.Price)), paste0(round(((n$Change.Amount/n$Past.Avg.Price)\*100),digits=2),"%"),"NA")

n <- n[,c(4:12,1:3,13:27)]

============================== <Code End > ===============================

Now that our data operations are completed, we split this final overall ‘workbook’ of sorts into respective increase and decrease-specific workbooks which contain solely those purchases whose unit prices rose and fell from the past average, and create a final workbook with these two data frames as its two tabs.

============================== <Code Begin > =============================

inclist <- na.omit(n[n$Inc.Dec.Prev.Avg == "INCREASE",])

declist <- na.omit(n[n$Inc.Dec.Prev.Avg == "DECREASE",])

write.xlsx2(inclist,file = paste0("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/Inc\_Dec/Increase\_Decrease workbook - ",format(Sys.Date(),"%b %d, %Y"),".xlsx"),sheetName = "Increase in Price from Past Avg", append = FALSE, row.names = FALSE)

write.xlsx2(declist,file = paste0("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/Inc\_Dec/Increase\_Decrease workbook - ",format(Sys.Date(),"%b %d, %Y"),".xlsx"),sheetName = "Decrease in Price from Past Avg", append=TRUE, row.names = FALSE)

============================== <Code End > ===============================

We have successfully prepared one of our final attachments completely. The next step is to generate line charts for all these purchased items using past and present prices, with date on the X-axis and unit price on the Y-axis. We will also segregate a particular chart into multiple plots in the same chart, based on the company under which the purchase is being made, for easy viewing.

To this end, first we will create folders to store the generated figures. We have a folder called ‘Figures’ that store all the figures generated on a daily basis. Inside the ‘Figures’ folder, we create a folder with the present date in the name to hold the day’s figures. Inside that folder, we create two separate folders for Increase figures and Decrease figures, with the respective day’s date in the folder name as well.

We also initialize the increase and decrease data frames into new variables so that they can be used to generate the figures. This is very much an optional step but just helps mentally differentiate well between sections of the code.

============================== <Code Begin > =============================

mainDir <- "C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/Figures"

subDir <- paste0(format(Sys.Date(),"%b %d")," generated figures")

dir.create(file.path(mainDir, subDir))

setwd(file.path(mainDir, subDir))

subDir1 <- paste0("Increase - ",format(Sys.Date(),"%b %d"))

subDir2 <- paste0("Decrease - ",format(Sys.Date(),"%b %d"))

dir.create(file.path(mainDir, subDir, subDir1))

dir.create(file.path(mainDir, subDir, subDir2))

datainc <- inclist

datadec <- declist

============================== <Code End > ===============================

We can now generate the figures corresponding to the increase and decrease purchases. We run loops through all rows of the increase and decrease data frames separately. Inside each loop, we choose out the item and extra descriptions of each row in the data frame, extract the past data for the same from the raw data file, plot entire line plots for each, by company and save them into the appropriate folder created. Loop constructs are generally not preferred in languages like R, since they are slow, but a loop construct would be easier to understand as a learner, particularly since the data size of the increase and decrease list isn’t that high. Once you have got the hang of it, you can very well define a function and run the same through the apply() function.

For the figure names, we just attach the iteration number (i), followed by the first words of the particular item description and extra description clubbed together. This can be altered at ease, as long as the names would be unique.

The ggplot function generates plots as usual (geom\_text\_repel makes sure that the labels, i.e., the unit prices for each point don’t overlap on the plot). The ‘ggsave’ function saves the generated figures one-by-one into the respective folders.

============================== <Code Begin > =============================

for (i in 1:nrow(datainc))

{

itemdesc<-datainc[i,10]

extradesc<-datainc[i,11]

tsdata <- rawdata[which(rawdata$Item.desc==itemdesc & rawdata$Extra.Desc==extradesc), ]

tsdata$PO.date<-as.Date(tsdata$PO.date, "%d-%b-%y")

tsdata$Item.desc<-gsub("/","-",tsdata$Item.desc)

tsdata$Extra.Desc<-gsub("/","-",tsdata$Extra.Desc)

name<-paste0("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/Figures/",substring(format(Sys.Date(),"%b %d, %Y"),1,6)," generated figures/Increase - ",substring(format(Sys.Date(),"%b %d, %Y"),1,6),"/",i," - ",gsub( " .\*$", "", itemdesc),gsub( " .\*$", "", extradesc),".jpg") #Plot save destination (flexible naming based on current date, item description, extra description)

tryCatch({

p <- ggplot(tsdata, aes(y=tsdata$Unit.Price, x=tsdata$PO.date, color=gsub( " .\*$", "",tsdata$Company.Name)), type="n", xlab="Date", ylab="Unit Price") +

geom\_point() + geom\_line() + geom\_text\_repel(aes(label=tsdata$Unit.Price), size=3) + ggtitle(paste(itemdesc,extradesc, sep=" ")) +

labs(x="Date",y="Unit Price") + scale\_colour\_discrete(name = "Company Name")

ggsave(filename = name, plot=p, width = 25, height = 10, units = "cm") #saving the generated plots

}, error=function(e){cat("ERROR :",conditionMessage(e), "\n")})

}

for (i in 1:nrow(datadec))

{

itemdesc<-datadec[i,10]

extradesc<-datadec[i,11]

tsdata <- rawdata[which(rawdata$Item.desc==itemdesc & rawdata$Extra.Desc==extradesc), ]

tsdata$PO.date<-as.Date(as.character(tsdata$PO.date), "%d-%b-%y")

tsdata$Item.desc<-gsub("/","-",tsdata$Item.desc)

tsdata$Extra.Desc<-gsub("/","-",tsdata$Extra.Desc)

name<-paste0("C:/Users/abhijithasok/Documents/Purchase Daily Dashboards/Figures/",substring(format(Sys.Date(),"%b %d, %Y"),1,6)," generated figures/Decrease - ",substring(format(Sys.Date(),"%b %d, %Y"),1,6),"/",i," - ",gsub( " .\*$", "", itemdesc),gsub( " .\*$", "", extradesc),".jpg") #Plot save destination (flexible naming based on current date, item description, extra description)

tryCatch({

p <- ggplot(tsdata, aes(y=tsdata$Unit.Price, x=tsdata$PO.date, color=gsub( " .\*$", "",tsdata$Company.Name)), type="n", xlab="Date", ylab="Unit Price") +

geom\_point() + geom\_line() + geom\_text\_repel(aes(label=tsdata$Unit.Price), size=3) + ggtitle(paste(itemdesc,extradesc, sep=" ")) +

labs(x="Date",y="Unit Price") + scale\_colour\_discrete(name = "Company Name")

ggsave(filename = name, plot=p, width = 25, height = 10, units = "cm") #saving the generated plots

}, error=function(e){cat("ERROR :",conditionMessage(e), "\n")})

}

============================== <Code End > ===============================

This concludes the operations with this R script.

1. Python script to compress the generated figures and the raw data

Since the raw data was sent as an XLS file with formatting inside, that was lost once we converted it to CSV, it is preferable to send the original file with the formatting, in the e-mail, for reference. But, since we will have more files to attach in the e-mail, it is better to compress the heavier files(in this case, the raw data and the generated figures) to make them fit better into the e-mail.

The following script was run on Python 2.7 on an iPython Notebook.

First, we import the required packages. The ‘os’ package enables us to access directories (folders) in our system, the ‘zipfile’ package enables us to compress our files and folders and the ‘datetime’ package enables us to obtain the current date and time(which we would need, since our files and folders contain the date in their names). We also use ‘datetime’ to get the current date.

============================== <Code Begin > =============================

import os

import zipfile

import datetime

now = datetime.datetime.now()

============================== <Code End > ===============================

We now access the folder containing the raw data and create a Zipfile object on the particular date’s raw data file, by accessing it using the ‘now’ variable, included in the file name(which effectively means we have compressed the file). Following that, we write that file into the same location and close the object.

============================== <Code Begin > =============================

os.chdir("C:\\Users\\abhijithasok\\Documents\\Purchase Daily Dashboards\\Raw Data")

zf = zipfile.ZipFile("OMX\_PO\_ITEM\_RATE\_ALT\_" + (now.strftime("%d-%b-%Y")).upper() + ".zip", "w", zipfile.ZIP\_DEFLATED) #Compressing data file

zf.write("OMX\_PO\_ITEM\_RATE\_ALT\_" + (now.strftime("%d-%b-%Y")).upper() + ".xls") #Saving data file

zf.close()

============================== <Code End > ===============================

We now follow the same procedure for the figures, but proceed with the file saving operation within a nested for-loop, since we are compressing the entire folder containing the figures for that particular date and need to preserve the internal folder structure.

============================== <Code Begin > =============================

os.chdir("C:\\Users\\abhijithasok\\Documents\\Purchase Daily Dashboards\\Figures")

zf = zipfile.ZipFile(now.strftime("%b %d") + " generated figures.zip", "w", zipfile.ZIP\_DEFLATED) #Compressing data folder

for dirname, subdirs, files in os.walk(now.strftime("%b %d") + " generated figures"):

zf.write(dirname)

for filename in files:

zf.write(os.path.join(dirname, filename)) #Saving data folder

zf.close()

============================== <Code End > ===============================

This completes the function of this Python script. We have effectively created our second and third email attachments completely, thereby readying all our email attachments. What is remaining now is to send the attachments over as an e-mail to selected people in the organization.

1. R script to send e-mail with attachments

This functionality is executed by creating an Outlook object using the ‘RDCOMClient’ package. This assumes that you are already logged into an official email account on MS Outlook on your local system.

============================== <Code Begin > =============================

library(RDCOMClient)

OutApp <- COMCreate("Outlook.Application") #Creating an email object

outMail = OutApp$CreateItem(0) #Configuration of email parameters

============================== <Code End > ===============================

Every part of the e-mail, such as ‘To’, ‘Cc’, ‘Bcc’, ‘Body’, ‘Subject’, Attachment’ etc, are parameters of this e-mail object and have to be provided separately.

The format for all except attachments is the same, i.e. ‘outMail[[<email part>]] = <content string>’. Multiple e-mail recipients are to be separated by a ‘;’.

============================== <Code Begin > =============================

outMail[["To"]] = "abc@xyz.com"

outMail[["CC"]] = paste("abc1@xyz.com","abc2@xyz.com","abc3@xyz.com","abc4@xyz.com",sep=";")

outMail[["subject"]] = paste0("Daily Purchase statistics - ",format(Sys.Date(),"%b %d, %Y"))

outMail[["body"]] = paste0("Hi,

PFA the data, report and figures for ",format(Sys.Date(),"%B %d")," (",format(Sys.Date()-1,"%B %d")," purchases vs past).",

"Thanks!")

============================== <Code End > ===============================

In the case of attachments, an additional ‘Add’ function enables adding the particular attachment to the e-mail. We add our three attachments – the compressed raw data file, the increase/decrease items workbook and the compressed folder of generated figures, to our e-mail.

============================== <Code Begin > =============================

outMail[["Attachments"]]$Add(paste0("C:\\Users\\abhijithasok\\Documents\\Purchase Daily Dashboards\\Raw Data\\OMX\_PO\_ITEM\_RATE\_ALT\_",toupper(format(Sys.Date(),"%d-%b-%Y")),".zip")) #Attaching compressed raw data

outMail[["Attachments"]]$Add(paste0("C:\\Users\\abhijithasok\\Documents\\Purchase Daily Dashboards\\Inc\_Dec\\Increase\_Decrease workbook - ",format(Sys.Date(),"%b %d, %Y"),".xlsx")) #Attaching created workbook

outMail[["Attachments"]]$Add(paste0("C:\\Users\\abhijithasok\\Documents\\Purchase Daily Dashboards\\Figures\\",substring(format(Sys.Date(),"%b %d, %Y"),1,6)," generated figures",".zip")) #Attaching compressed folders of generated figures

============================== <Code End > ===============================

Finally, we send the mail.

============================== <Code Begin > =============================

outMail$Send()

============================== <Code End > ===============================

## Conclusion & further remarks

This concludes the entire process. Scheduling scripts 2-5 one after the other on any scheduler, typically the ‘Task Scheduler’ on Windows, ensures smooth execution of the same, without you having to even worry about anything. The first VB script will automatically do its job once it is keyed into the VB editor inside Outlook.

RStudio now has an option for a task scheduler within itself on which you can alternatively schedule your R scripts. Refer [this link](https://www.r-bloggers.com/new-rstudio-add-in-to-schedule-r-scripts/) to set it up.

Alternatively, you could combine scripts 3-5 together as a Python script and call the R scripts from within Python. This would enable you to schedule only 2 scripts in total. If you would like to set it up, you could refer [this link](https://www.r-bloggers.com/integrating-python-and-r-part-ii-executing-r-from-python-and-vice-versa/).

You could also write both R scripts, and even all the scripts entirely in Python and have it executed as a single code too. R was both a personal choice due to comfort and a contextual choice due to the diversity in tools used for the single exercise, for better learning. A higher priority has not been given to the best possible code, but to the easily understandable code, since this is a learning exercise meant to enable the readers to improve the optimize the code further on their own, after understanding it.

I hope this has been of help. Feel free to reach out, in case you have any questions.