Technical Specification

[Retail Analytics – Sales Forecasting]

Author: [Ipsita Panda]

Date: [Mar 16, 2016]

Version: [Draft]

# Document Control

**Document location**

| Location |
| --- |
| <http://172.25.164.110:9000/OEG-MC/Sales-Forecast> |

**Author**

| Position | Name | Contact no |
| --- | --- | --- |
| ASE | Ipsita Panda | 9556688700 |

**Revision history**

| Version | Issue date | Author/editor | Description/Summary of changes |
| --- | --- | --- | --- |
| DRAFT | Feb 25, 2016 | Ipsita Panda | Draft version of Sales Forecasting |
|  |  |  |  |

**Reviewed by**

| Version | Issue date | Name | Position | Review date |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Approvals**

Approval refers to the approver’s acceptance of the content and overall intention of this document, including acceptance of any commitments described in order to successfully deliver the initiative. The approver, where relevant, also confirms that this document complies with relevant strategies, policies and regulatory requirements.

| Version | Issue date | Name | Position | Approval date |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

**Related documents**

| Document | Location |
| --- | --- |
| Sales Forecasting pdf  Plotly articles | <https://media.readthedocs.org/pdf/a-little-book-of-r-for-time-series/latest/a-little-book-of-r-for-time-series.pdf>  <https://plot.ly/r/subplots> |

# Table of Contents

1 INTRODUCTION 4

1.1 Objectives 4

1.2 Scope 4

2 Functionality 4

2.1 Screenshots with explanation of each controls 4

3 Achitecture 4

3.1 Architecture Overview 4

3.2 Component Structure 4

4 Design 4

4.1 Data Dictionary 4

5 CODE 4

5.1 R & JS File Names and Structure 4

# INTRODUCTION

## Objectives

**Sales Forecasting:** Forecasting refers to the process of using statistical procedures to predict future values of a time series based on historical trends. This is a method of analyzing the future trends in a retail outlet depending upon the historic data of the outlet. Its purpose is to as accurately, as possible to predict what quantity ofgoods will be sold in near future.

## Scope

It works for transactional level point of sales data of a retail store.

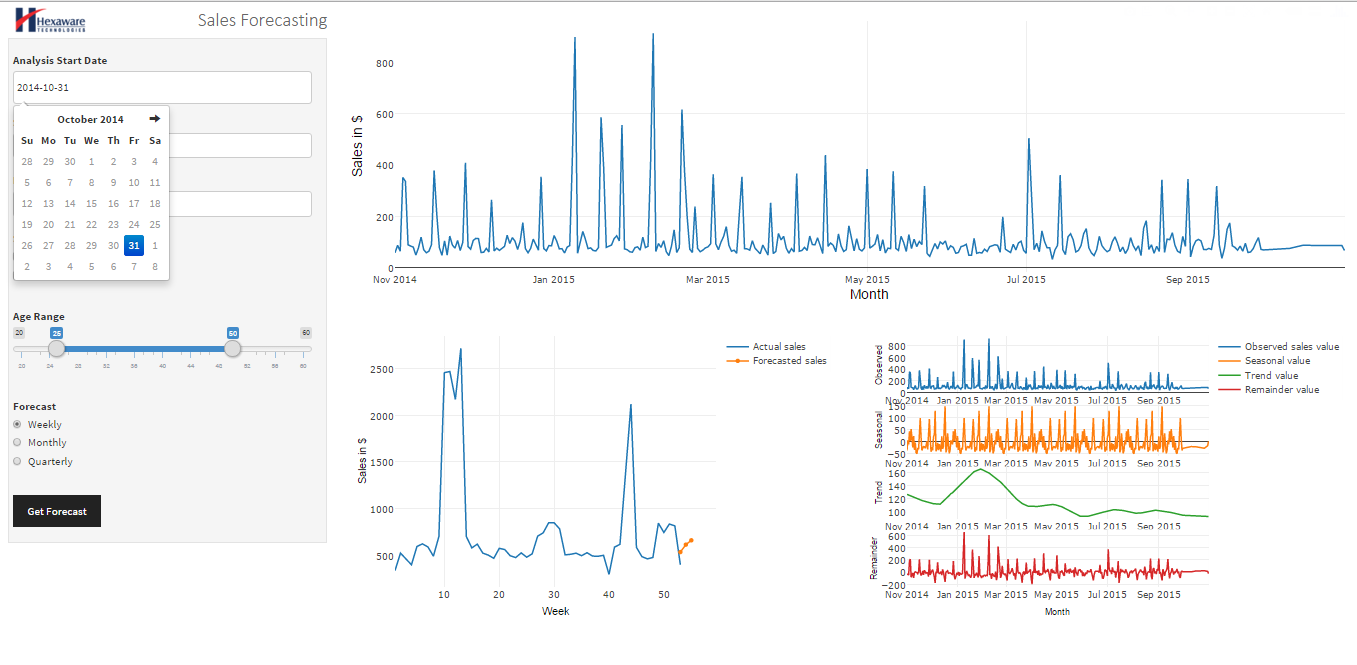
# Functionality

## Screenshots with explanation of each controls

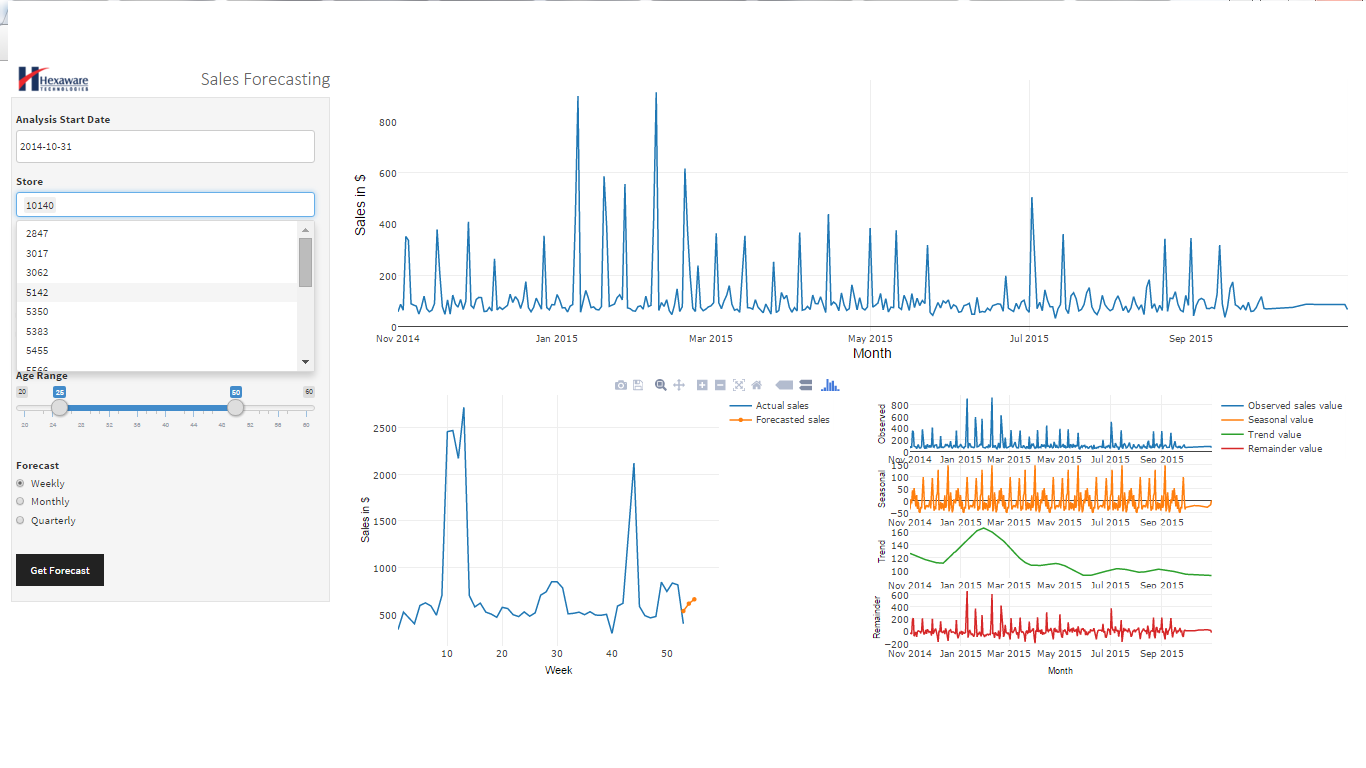
Main Screen(default)



Sales Forecasting Date filter



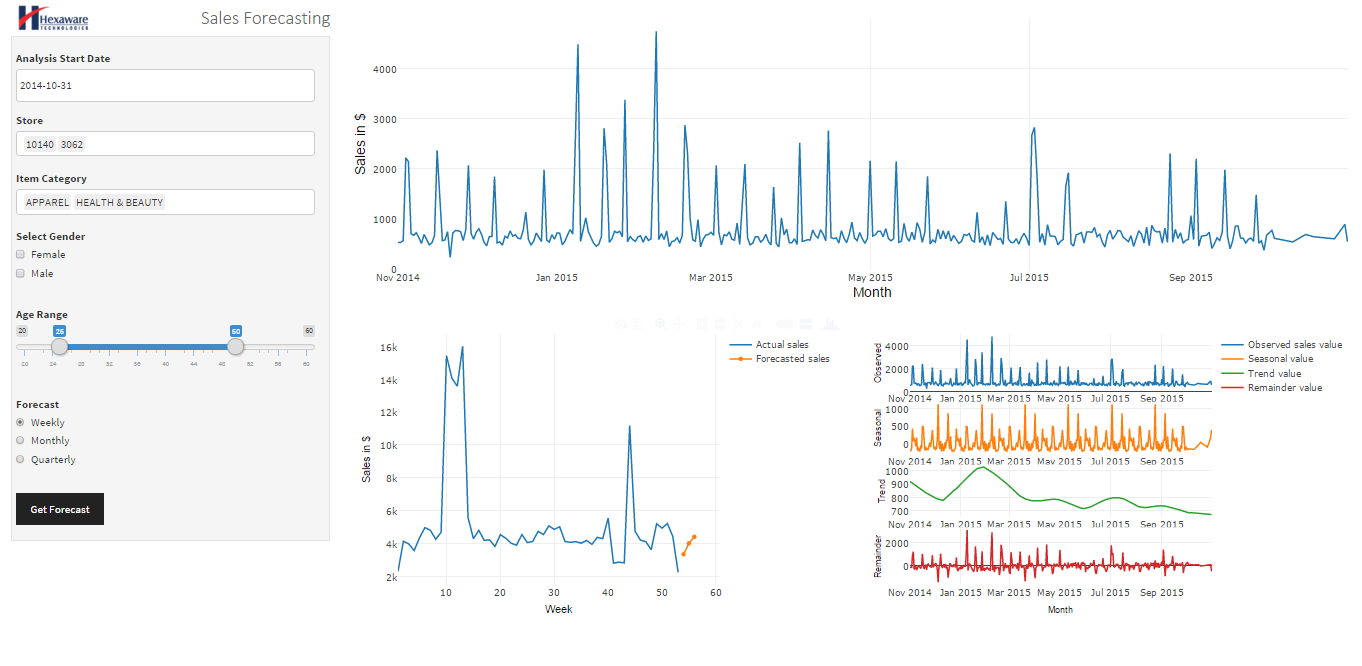
Sales Forecasting Store filter



Sales Forecasting Category Filter



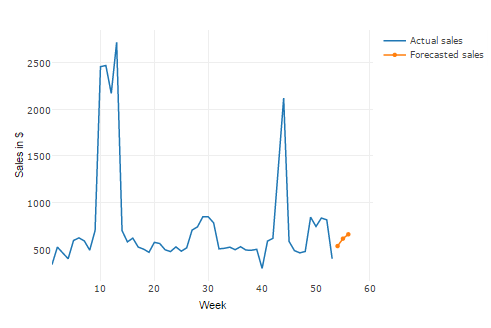
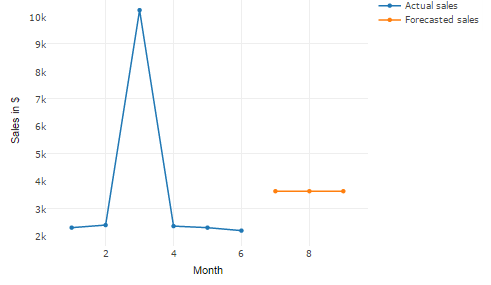
RFM calculation



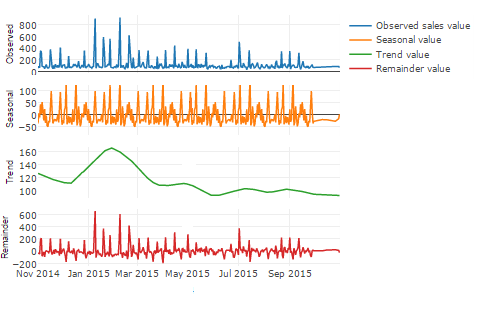
Sales Forecasting **‘ Forecast Chart’** Zoomed-in

Weekly Forecast

Monthly Forecast

Sales Forecasting **‘ Decompose Chart’** Zoomed-in



Architecture

## Architecture Overview

Front end – R Shiny

Middle tier – R

Backend – SQL Server 2012

**Front-end / Client Side:**  
The top most visible layer is what is called the Frontend. This is written in R Shiny for styling, interactivity and function. The data from this layer has to make it to the user's browser which then turns all the code into something beautiful and interactive.  
  
**Middle tier:**  
Here we have the "Middle tier" which is where we usually use a programming language R. The middle tier usually decides how to render the front-end depending on it's business logic.

**Back-end:**  
In order for most web application to function, we need to be able to store data somewhere. That is served by a database of some sort. Here we are using SQL Server 2012.

## Component Structure

Every Shiny app is composed of a two parts: a web page that shows the app to the user, and a computer that powers the app. The computer that runs the app can either be your own laptop (such as when you’re running an app from RStudio) or a server somewhere else. You, as the Shiny app developer, need to write these two parts (you’re not going to write a computer, but rather the code that powers the app). In Shiny terminology, they are called UI (user interface) and server.

# Design

## Data Dictionary

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description of the field** |
|  |  |  |
| store | varchar | Store number |
| transaction\_date | date | Date of the transaction as recorded in the POS system |
| transaction\_id | varchar | Unique Transaction identifier |
| transaction\_hr | varchar | Time of transaction |
| item | varchar | Product identification number |
| price | real | Price of the product |
| quantity | real | Quantity of product bought |
| region | varchar | Demographic information |
| category | nvarchar | Category to which the product belongs |
| sku | nvarchar | Stock Keeping Unit refers to a specific stock item |
| sku\_id | float | Unique Stock Keeping Unit number |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# CODE

Function getForecastData

NOTE-This function is repeated thrice in the code, for fetching data week-wise, month-wise and quarter-wise using the DATEPART function. Input remains the same for all three functions, and output depends upon which DATEPART function is used inside the query.

INPUT PARAMETERS- StartDate, stores, category

OUTPUT PARAMETER- Dataframe with transaction\_date and sum(quantity) as Total\_Sales, starting from the startDate and with the provided filters for store and category. For week-wise datepart, the output will be a dataframe displaying the week number to which the date belongs, alongwith the Total\_Sales . For month-wise, the output will come in a month-wise pattern and same follows for quarter.

DESCRIPTION- Creating a customized function which will do the following:-

FILTERS

1) Filter the data within the provided start date(Analysis start date)

2) Filter the data according to the Store number passed as argument

3) Filter the data according to the Item Category passed as argument

Function SalesForecast

NOTE-This function is repeated thrice in the code, for providing the forecasted chart according to week-wise, month-wise and quarter-wise data. Difference between the three functions is the frequency parameter passed. For week-wise forecast, frequency is 52; for month-wise, frequency is 12; for quarter-wise, frequency is 4.

INPUT PARAMETERS- First input is dataframe consisting of transaction\_date and Total\_Sales. The input to this function will be the output which we get from the above explained getForecastData function. Second input is Frequency .

OUTPUT PARAMETER- Forecasted plotly chart showing the relative forecasted values alongwith the actual values.

DESCRIPTION- Uses ARIMA(Auto-Regressive Integrated Moving Average) model to forecast the store and department level sales.

ARIMA model is classified as an ARIMA(p,d,q) model, where:

* **p** is the number of autoregressive terms,
* **d** is the number of nonseasonal differences needed for stationarity, and
* **q** is the number of lagged forecast errors in the prediction equation.

Function decompose

INPUT PARAMETERS- First input is dataframe consisting of transaction\_date and Total\_Sales. The input to this function will be the output which we get from the above explained getForecastData function. Second input is Frequency .

OUTPUT PARAMETER- Decomposition chart comprising four graphs -original data, seasonal component, trend component and the remainder and this shows the periodic seasonal pattern extracted out from the original data. Plotly is used to display the dynamic decompose chart.

DESCRIPTION- Uses STL (Seasonal and Trend decomposition using Loess ) function which helps to divide up a time series into three components namely: the trend, seasonality and remainder. The seasonal component is found by loess smoothing the seasonal sub-series (the series of all January values, ...). The seasonal values are removed, and the remainder smoothed to find the trend. The overall level is removed from the seasonal component and added to the trend component. This process is iterated a few times. The remainder component is the residuals from the seasonal plus trend fit.

#Importing libraries  
library(RODBC)  
library(Rcpp)

## Warning: package 'Rcpp' was built under R version 3.1.3

library(plyr)  
library(rstudio)  
  
  
date\_calender<-function(){  
   
 myconn <-odbcConnect("retail", uid="sa", pwd="Password123")  
   
   
   
 query="SELECT min([Transaction\_date]) as min\_date,max([Transaction\_date]) as max\_date  
 from [dbo].[Main\_data\_retail\_analytics\_VERSION\_2]"  
   
 result\_date=sqlQuery(myconn, query)  
 result\_min\_date=as.Date(result\_date$min\_date)  
 result\_max\_date=as.Date(result\_date$max\_date)  
 final\_result\_date=c(result\_min\_date,result\_max\_date)  
   
   
   
}  
  
  
store\_ddl<-function(){  
   
 myconn <-odbcConnect("retail", uid="sa", pwd="Password123")  
   
   
   
 query="SELECT distinct([Store\_No]) as store\_name  
 from [dbo].[Main\_data\_retail\_analytics\_VERSION\_2]"  
   
 result\_store=sqlQuery(myconn, query)  
   
 result\_store<-as.character(result\_store$store\_name)  
   
   
}  
  
itemCategory\_ddl<-function(){  
   
 myconn <-odbcConnect("retail", uid="sa", pwd="Password123")  
   
   
 query="SELECT distinct([Item\_Category\_Code\_Desc]) as item\_name  
 from [dbo].[Main\_data\_retail\_analytics\_VERSION\_2]"  
   
 result\_item=sqlQuery(myconn, query)  
   
 result\_item<-as.character(result\_item$item\_name)  
   
   
}  
  
  
  
getRFM<-function(StartDate,EndDate,stores,items,WeightRecency,WeightFrequency,WeightMonetary)  
{   
   
   
 #Pasting start and end date between '' to print them in SQL query  
 vcharStartDate1<-paste("'",StartDate,"'",sep="")  
 vcharEndDate1<-paste("'",EndDate,"'",sep="")  
   
   
 #Pasting Items between '' to print them in SQL query  
   
 if(items=="All Items")  
 {  
 vcharItemString=paste("'Activewear'",",","'Baby & Toddler Clothing'",",","'One-Pieces'",",","'Outerwear'",",","'Outfit Sets'",",","'Pants'",",","'Shirts & Tops'",",","'Shorts'",",","'Skirts'",",","'Sleepwear & Loungewear'",",","'Suits'",",","'Swimwear'",",","'Traditional & Ceremonial Clothing'",",","'Underwear & Socks'",",","'Uniforms'",",","'Wedding & Bridal Party Dresses'")  
   
   
 } else {  
 if(length(items)<=1)  
 {  
 vcharItemString=paste("'",items[1],"'",sep="")  
 } else {  
 vcharItemString=paste("'",items[1],"'",sep="")  
 for(i in 2:length(items)){  
 vcharItemString <-paste(vcharItemString,paste("'",items[i],"'",sep=""),sep=",")  
   
 }  
 }  
 }  
   
 #Pasting Stores between '' to print them in SQL query  
 if(stores=="All Stores")  
 {  
 vcharStoreString=paste("'S0001'",",","'S0002'",",","'S0003'",",","'S0004'",",","'S0005'",",","'S0006'",",","'S0007'",",","'S0008'",",","'S0009'",",","'S0010'",",","'S0011'",",","'S0012'",",","'S0013'",",","'S0014'",",","'S0015'",",","'S0017'",",","'S0018'",",","'S0020'",",","'S0021'",",","'S0022'",",","'S0023'",",","'S0024'",",","'S0025'",",","'S0026'",",","'S0027'",",","'S0028'",",","'S0029'",",","'S0032'",",","'S0033'",",","'S0034'",",","'S0035'",",","'S0036'",",","'S0037'",",","'S0038'",",","'S0040'",",","'S0041'",",","'S0042'",",","'S0043'",",","'S0044'",",","'S0045'",",","'S0046'",",","'S0047'",",","'S0048'",",","'S0049'",",","'S0050'",",","'S0051'",",","'S0052'",",","'S0053'",",","'S0057'",",","'S0058'",",","'S0059'")  
   
   
 } else {  
 if(length(stores)<=1)  
 {  
 vcharStoreString=paste("'",stores[1],"'",sep="")  
   
 } else {  
 vcharStoreString=paste("'",stores[1],"'",sep="")  
 for(i in 2:length(stores)){  
 vcharStoreString <-paste(vcharStoreString,paste("'",stores[i],"'",sep=""),sep=",")  
   
 }  
 }  
   
 }  
   
   
   
 #Writing a dynamic SQL query to fetch the data with required filters  
 #SQLconnection name-retail,user-sa,password-Password123  
 SQLquery <- sprintf('SELECT count(Loyalty\_Card\_No)as frequency\_data,  
 sum(Price)as monetary\_data,max(Transaction\_date)as recency\_date,Loyalty\_Card\_No ,Store\_No ,Item\_Category\_Code\_Desc   
 from [dbo].Main\_data\_retail\_analytics\_VERSION\_2 where Transaction\_date >=%s And Transaction\_date <%s and Item\_Category\_Code\_Desc IN (%s)And Store\_No IN (%s) group by Loyalty\_Card\_No ,Store\_No ,Item\_Category\_Code\_Desc',vcharStartDate1,vcharEndDate1,vcharItemString,vcharStoreString)  
   
 SQLconn <-odbcConnect("retail", uid="sa", pwd="Password123")  
 vcharSQLResult=sqlQuery(SQLconn, SQLquery)  
 close(SQLconn)  
   
 dfSQLResult<-as.data.frame(vcharSQLResult,stringsAsFactors=FALSE)  
  
  
 #Calculating Recency,Frequency and Monetary  
   
 vdifftimeRecent<- difftime(EndDate ,dfSQLResult$recency\_date , units = c("days"))  
  
 #Binding Recency,Frequency and Monetary with their respective loyalty Cards  
 vmatRecency= cbind("Recency"=vdifftimeRecent,"Loyalty\_Card\_No"=as.character(dfSQLResult$Loyalty\_Card\_No))  
 vmatFrequency <- cbind("Frequency"=dfSQLResult$frequency\_data,"Loyalty\_Card\_No"=as.character(dfSQLResult$Loyalty\_Card\_No))  
 vmatMonetary<-cbind("Monetary"=dfSQLResult$monetary\_data,"Loyalty\_Card\_No"=as.character(dfSQLResult$Loyalty\_Card\_No))  
   
   
 #Forcing R not to use exponential scientific notation  
 #options("scipen"=100, "digits"=4)  
   
 #Converting matrices into dataframes so that we can access the columns of those dataframes individually  
 dfRecency=as.data.frame(vmatRecency,stringsAsFactors=FALSE)  
 dfFrequency=as.data.frame(vmatFrequency,stringsAsFactors=FALSE)  
 dfMonetary=as.data.frame(vmatMonetary,stringsAsFactors=FALSE)  
   
 #Joining recency,frequency and monetary for corresponding LoyaltyCardno  
 vmatMerged\_data=cbind("Recency"=dfRecency$Recency,"Frequency"=dfFrequency$Frequency,"Monetary"=dfMonetary$Monetary,"Loyalty\_Card\_No"=as.character(dfRecency$Loyalty\_Card\_No))  
   
 #converting the above matrix into dataframe  
 dfMerged\_data= as.data.frame(vmatMerged\_data)  
   
   
 #Calculating inverse of Recency and Standardizing it by dividing it by 365  
 #Changing class of Recency,Frequency and Monetary into numeric so that we can apply mathematical functions to them  
 ## IMPORTANT POINT TO NOTE  
 #if we directly convert as.numeric from a factor value it converts it wrongly so we are using as.numeric(paste(Merged\_Data$Monetary)))  
   
 vnumRecency\_std=paste(as.numeric((365/as.numeric(paste(dfMerged\_data$Recency)))))  
 vnumFrequency\_std=as.numeric(paste((dfMerged\_data$Frequency)))  
 Monetary\_new1=as.numeric(paste(dfMerged\_data$Monetary))  
 vnumMonetary\_std=((Monetary\_new1)/(vnumFrequency\_std))  
   
   
 #Converting matrices into dataframes so that we can access the columns of those dataframes individually  
 dfRecency\_std=as.data.frame(vnumRecency\_std)  
 dfFrequency\_std=as.data.frame(vnumFrequency\_std)  
 dfMonetary\_std=as.data.frame(vnumMonetary\_std)  
   
 #Changing class of Recency,Frequency and Monetary columns into numeric so that we can apply mathematical functions to them  
 dfRecency\_std$vnumRecency\_std=as.numeric(paste(dfRecency\_std$vnumRecency\_std))  
 dfFrequency\_std$vnumFrequency\_std=as.numeric(paste(dfFrequency\_std$vnumFrequency\_std))  
 dfMonetary\_std$vnumMonetary\_std=as.numeric(paste(dfMonetary\_std$vnumMonetary\_std))  
   
 #Calculating an RFM Score as  
 #RFM Score = R\*WR+F\*WF+M\*WM  
 #where R=Recency,F=Frequency,M=Monetary   
 #and WR= Weight for Recency,WF= Weight for Frequency,WM=Weight for Monetary  
 #Then combining the Recency,Frequency,Monetary and RFM values for each loyalty Card  
   
 vmatRFM\_Score=cbind("Loyalty\_Card\_No"=as.character(dfRecency$Loyalty\_Card\_No),"Recency"=dfRecency\_std$vnumRecency\_std,"Frequency"=dfFrequency\_std$vnumFrequency\_std,"Monetary"=dfMonetary\_std$vnumMonetary\_std,"RFM\_Score"=paste(as.numeric(((dfRecency\_std$vnumRecency\_std\*WeightRecency)+(dfFrequency\_std$vnumFrequency\_std\*WeightFrequency)+(dfMonetary\_std$vnumMonetary\_std\*WeightMonetary)))))  
 dfFinal\_RFM\_Score=as.data.frame(vmatRFM\_Score)  
   
 #Calculating the maximum RFM Score for the filtered data  
 vnummax\_RFM=max(as.numeric(paste(dfFinal\_RFM\_Score$RFM\_Score)))  
   
 #Calculating the number of rows of the filtered data  
 vintrowcount\_RFM=nrow(dfFinal\_RFM\_Score)  
   
 #Calculating relative RFM by dividing each RFM\_score by maximum RFM Score during the time period selected  
 #If the number of rows is 0 then Relative RFM should come as 0  
 if (vintrowcount\_RFM==0) {  
 Relative\_RFM=0  
 } else {  
 Relative\_RFM=(as.numeric(paste(dfFinal\_RFM\_Score$RFM\_Score))/vnummax\_RFM)  
 }  
   
   
 dfRelative\_RFM= as.data.frame(Relative\_RFM,stringsAsFactors=FALSE)  
   
   
 #Adding a column Bins to the dataframe  
  
 dfFinalRFMScore=cbind(dfFinal\_RFM\_Score,dfRelative\_RFM)  
 dfFinalRFMScoretable=cbind(Bins="0",dfFinalRFMScore)  
 dfFinalRFMScoretable = as.data.frame(dfFinalRFMScoretable)  
 dfFinalRFMScoretable$Bins=as.character(dfFinalRFMScoretable$Bins)  
   
   
 #By creating bins we divide the records according to the relative RFM of each loyalty Card no  
 #0<Relative RFM<0.20 - Bin "0.00-0.20"  
 #0.20<Relative RFM<0.40 - Bin "0.20-0.40"  
 #0.40<Relative RFM<0.60 - Bin "0.40-0.60"  
 #0.60<Relative RFM<0.80 - Bin "0.60-0.80"  
 #0.80<Relative RFM<1.00 - Bin "0.80-1.00"   
   
 i=1  
 for (i in 1:nrow(dfFinalRFMScore)){  
 if(dfFinalRFMScoretable$Relative\_RFM[i]>0 & dfFinalRFMScoretable$Relative\_RFM[i]<=0.2){  
 dfFinalRFMScoretable$Bins[i]<-"0.00-0.20"  
 }  
 else if(dfFinalRFMScoretable$Relative\_RFM[i]>0.2 & dfFinalRFMScoretable$Relative\_RFM[i]<=0.4){  
 dfFinalRFMScoretable$Bins[i]<-"0.21-0.40"  
 }  
 else if(dfFinalRFMScoretable$Relative\_RFM[i]>0.4 & dfFinalRFMScoretable$Relative\_RFM[i]<=0.6){  
 dfFinalRFMScoretable$Bins[i]<-"0.41-0.60"  
 }  
 else if(dfFinalRFMScoretable$Relative\_RFM[i]>0.6 & dfFinalRFMScoretable$Relative\_RFM[i]<=0.8){  
 dfFinalRFMScoretable$Bins[i]<-"0.61-0.80"  
 }  
 else {  
 dfFinalRFMScoretable$Bins[i]<-"0.81-1.00"  
 }  
   
   
 }   
   
 #Creating a final dataframe with loyalty card No,recency,frequency,monetary values,RFM Score,Relative RFM and the Bin value  
 dfFinalRFMScoretable=as.data.frame(dfFinalRFMScoretable,stringsAsFactors=FALSE)  
   
 #Rounding values of columns to 2 digits for clarity  
 dfFinalRFMScoretable$Recency=round(as.numeric(paste(as.numeric(365/as.numeric((paste(dfFinalRFMScore$Recency)))))))  
 dfFinalRFMScoretable$Frequency=round(as.numeric(paste(dfFinalRFMScoretable$Frequency)),2)  
 dfFinalRFMScoretable$Monetary=round(as.numeric(paste(dfFinalRFMScoretable$Monetary)),2)  
 dfFinalRFMScoretable$RFM\_Score=round(as.numeric(paste(dfFinalRFMScoretable$RFM\_Score)),2)  
 dfFinalRFMScoretable$Relative\_RFM=round(as.numeric(paste(dfFinalRFMScoretable$Relative\_RFM)),2)  
   
 #Creating a final output table to be displayed in visualization dashboard  
 dfFinalRFMScoretable=as.data.frame(dfFinalRFMScoretable,stringsAsFactors=FALSE)  
   
   
   
}#end of function getRFM  
  
  
  
  
getTable <- function(datasubset,binSelection){  
   
   
   
 if(binSelection=="All")  
 datasubset  
   
 else if(binSelection=="0.00-0.20")   
 subset(datasubset,datasubset$Bins == "0.00-0.20")  
   
 else if(binSelection=="0.21-0.40")  
 subset(datasubset,datasubset$Bins =="0.21-0.40")  
   
 else if(binSelection=="0.41-0.60")  
 subset(datasubset,datasubset$Bins =="0.41-0.60")  
   
 else if(binSelection=="0.61-0.80")  
 subset(datasubset,datasubset$Bins =="0.61-0.80")  
   
 else (binSelection=="0.81-1.00")  
 subset(datasubset,datasubset$Bins == "0.81-1.00")  
   
}