Data Exploration for Parts Forecasting

Executive Summary

The company stocks 25,000 lines of parts, approximately 2,000 are active at any one time. The company mostly serves harvest machinery (combines, foragers, rakes etc), so demand is highly seasonal. Given the seasonality the company is offered heavy discounts for ordering parts 6-9months in advance. This requries a confident forecast of parts demand. Currently such an order is made, it is worth approx £0.5m and is simply based on a weighted average of sales from previous years.

A simple 10% improvement in the approach would yield a substantial saving, easily worth the effort of exploring the data and then developing better forecasting models.

This notebook investigates the data and explores ARIMA and tscount time series models. It then goes on to explore clustering of the data, in an effort to deal with a long tail of parts with few sales.

Finally, all these features (sales qty, clusters, ARIMA models, weighted averages, etc) are exported to files from where subseqent notebooks explore statistical modelling methods and deep learning models.

Parts Sales Data

We extract the data from the ERP system using a custom made stored procedure in SQL...

```
require(RODBC)

##Create a connection to the database using ODBC
conn<-odbcConnect(dsn="RODBC",uid="sa",pwd="")

##Extract the required part_number data
parts_all    <- sqlQuery(conn, "EXEC [sp_GetPartsSalesAsSequenceToSequenceVector]",
                         stringsAsFactors=FALSE)
parts_decode <- sqlQuery(conn, "EXEC [sp_GetPartsSalesAsSequenceToSequenceVector_Decoder]",
                         stringsAsFactors=FALSE)

#Close connection
odbcClose(conn)
```

```
library(knitr)
kable(head(parts_all))
```

INVOICE_DATE	INVOICE_NO	DEPOT_CODE	RANDOM_SEQ	CODE	INVOICE_MONTH
2008-08-01	153984	1	0.3103646	2690	2008-08-31
2008-08-01	153984	1	0.0529985	2690	2008-08-31
2008-08-01	153984	1	0.4925043	2690	2008-08-31
2008-08-01	153984	1	0.6184219	2690	2008-08-31
2008-08-01	153984	1	0.2022889	2690	2008-08-31
2008-08-01	153984	1	0.6697388	2690	2008-08-31

```
kable(head(parts_decode))
```

CODE	PART_NUMBER	PART_DESCRIPTION	R_R_PRICE
1	CL-0000000052	GUIDE	80.25
2	CL-0000000101	PLATE	41.93
3	CL-0000000121	KNIFE	7.75
4	CL-0000000170	SPRING	3.96
5	CL-0000000211	BOLT	10.58
6	CL-0000000220	RING	2.78

Explore Data

There are currently 9 years of parts sales data in the table, this is over a million rows of data, enough to consider some machine learning solutions to forecasting.

Reformat the data

The data was originally designed to be used for a word2vec analysis, which will be done later. Meanwhile, we need a clean list of sales, one month of sales per part number, per month

```
library(dplyr)
library(lubridate)

##Let's sum depots into a view of the company as a whole and for each month, not each day
# First apply a 'month end' to each record, as opposed to 'day of month'
parts_all$INVOICE_MONTH      <- parts_all$INVOICE_DATE
day(parts_all$INVOICE_MONTH) <- days_in_month(parts_all$INVOICE_DATE)

# Now add columns we will use in subsequent analysis;
  # PARTS_GRP: the logical group each parts belongs to, based on the part_number.
  # MONTH_NO : the month number (1-12) in which a part is sold, useful for seasonality analysis
# Then summarise the data by month, summing across all depots for each month

parts_all_bymth_sng <- parts_all  %>%
                    group_by(CODE, INVOICE_MONTH) %>%
                      dplyr::summarise(SALES_QTY = n(), SALES_EVENTS = n_distinct(INVOICE_NO)) %>%
                        inner_join(y=parts_decode, by='CODE') %>%
                          select(PART_NUMBER, CODE, INVOICE_MONTH, SALES_QTY, SALES_EVENTS) %>%
                            mutate(PART_GRP = paste0("CL_GRP-",substr(PART_NUMBER, 4, 8)),
                                   MONTH_NO = as.numeric(format(as.Date(INVOICE_MONTH), "%m"))) %>%
                              arrange(PART_NUMBER, INVOICE_MONTH) %>%
                                ungroup()
```

```
kable(head(parts_all_bymth_sng))
```

PART_NUMBER	CODE	INVOICE_MONTH	SALES_QTY	SALES_EVENTS	PART_GRP	MONTH_NO
CL-0000000052	1	2010-06-30	1	1	CL_GRP-00000	
CL-0000000052	1	2014-09-30	1	1	CL_GRP-00000	
CL-0000000052	1	2015-06-30	2	1	CL_GRP-00000	
CL-0000000101	2	2011-07-31	1	1	CL_GRP-00000	
CL-0000000101	2	2011-08-31	1	1	CL_GRP-00000	
CL-0000000101	2	2015-04-30	1	1	CL_GRP-00000	

```
print(paste0("Quantity of part numbers: ",n_distinct(parts_all_bymth_sng$PART_NUMBER)))
```

```
## [1] "Quantity of part numbers: 26835"
```

```
print(paste0("Quantity of part groups: ", n_distinct(parts_all_bymth_sng$PART_GRP)))
```

```
## [1] "Quantity of part groups: 565"
```

```
print(paste0("Date range of data: from=",
             min(parts_all_bymth_sng$INVOICE_MONTH),
             ". to=",
             max(parts_all_bymth_sng$INVOICE_MONTH)))
```

```
## [1] "Date range of data: from=2008-08-31. to=2018-08-31"
```

```
#This allows us to build quantiles to see how skewed the data is
dat_PartCountFrequency <- parts_all_bymth_sng  %>%
                            group_by(PART_NUMBER) %>%
                              dplyr::summarise(Freq=sum(SALES_EVENTS)) %>%
                                arrange(desc(Freq))

quantile(dat_PartCountFrequency$Freq, seq(from=0,to=1,by=0.05))
```

```
##   0%   5%  10%  15%  20%  25%  30%  35%  40%  45%  50%  55%  60%  65%  70%
##    1    1    1    1    1    1    1    1    1    1    2    2    2    3    4
##  75%  80%  85%  90%  95% 100%
##    5    7   10   16   35 2787
```

So, 90% of the parts have been sold less on 16 or fewer occassions (invoices) over the past 10 years. The count of sales events (invoices) in the most frequently sold 5% of parts ranges from 35 to 2770! The vast majority of sales are in the top 5% of parts.

This is an highly skewed data set, as count data often is.

The objective is to arrive at plausible predictions and sequences of parts sales. There is no statistical method, not even deep learning, which will give plausible predictions based on only 1-5 sales over the past 9yrs. These obscure real data and consume processing power, so need to be removed.

Frequently Used Parts Only

We can characterise the parts into groups: Rarities - parts sold on a handful of times ever. Would not stock such a part (aka 'Non stocking'). Low Usage - parts sold a few times, not quite a rarity but still non-stocking Normal Usage - 'normal' usage, as defined below BellWeathers - higher than normal usage, statistically significant figures each year NutsAndBolts - very high levels of usage but this is due to the item being sold, eg nuts which are sold in packets of 100.

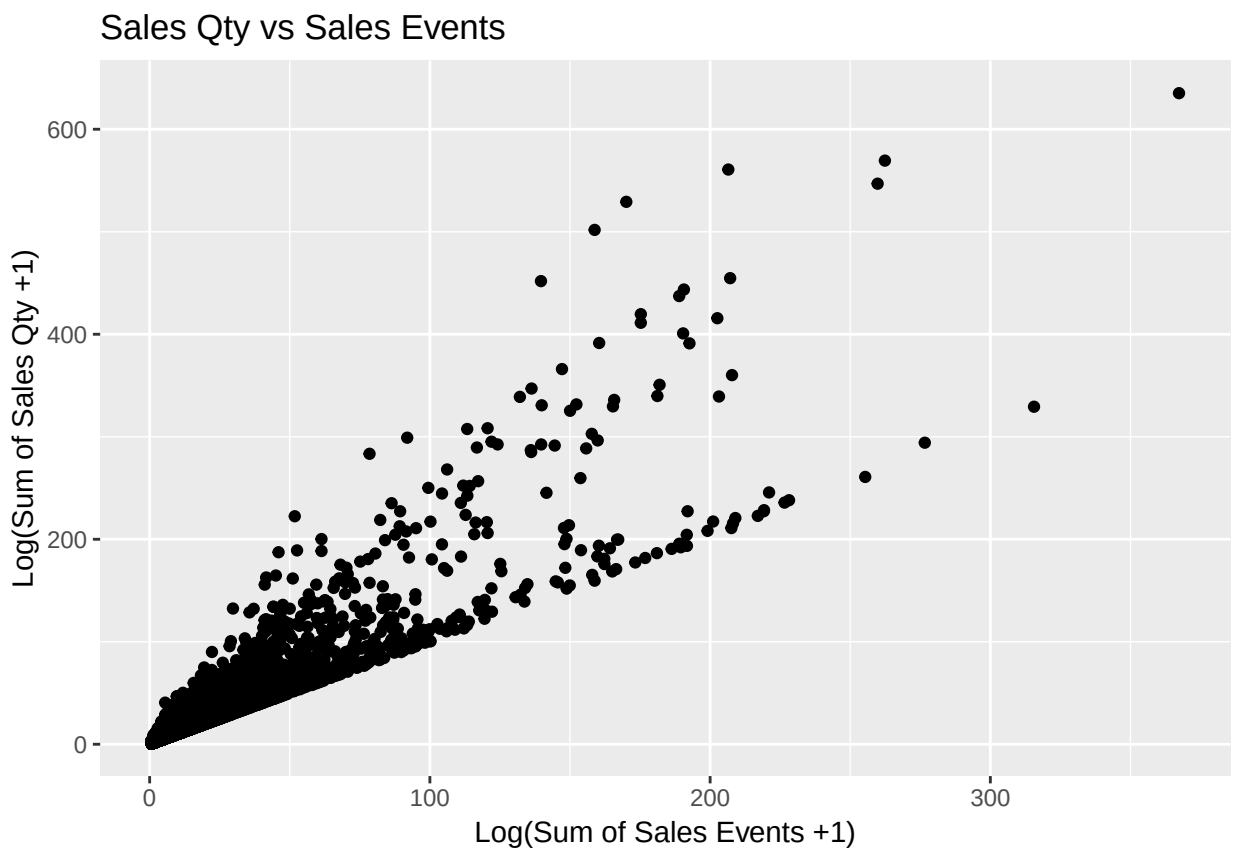
Rarities have almost no statistical contribution to make. We cannot predict their usage, and with so few sales we cannot use them to predict usage of other parts. Despite being 80% of the part_numbers in consideration they are 'outliers' and would skew any PCA analysis.

NutsAndBolts also introduce outliers and may skew analysis.

Let's log the data to bring it into a more manageable scale and take a grpahical look at the distribution:

```
library(dplyr)
library(ggplot2)
library(grid)
library(gridExtra)
```

```
parts_sums <- parts_all_bymth_sng %>%
          group_by(PART_NUMBER) %>%
            mutate(SALES_QTY = log(SALES_QTY+1), SALES_EVENTS = log(SALES_EVENTS+1)) %>%
              summarise(TOTAL_QTY_SOLD = sum(SALES_QTY), TOTAL_SALES_EVENTS = sum(SALES_EVENTS)) %>%
                arrange(PART_NUMBER)

p1 <- ggplot(parts_sums, aes(x=TOTAL_SALES_EVENTS, y=TOTAL_QTY_SOLD)) +
        xlab("Log(Sum of Sales Events +1)") +
        ylab("Log(Sum of Sales Qty +1)") +
        geom_point() +
        ggtitle("Sales Qty vs Sales Events")

p1
```



Even when logged the data is heavily skewed. Variance increases as sales events increases, beware heteroskedacicity.

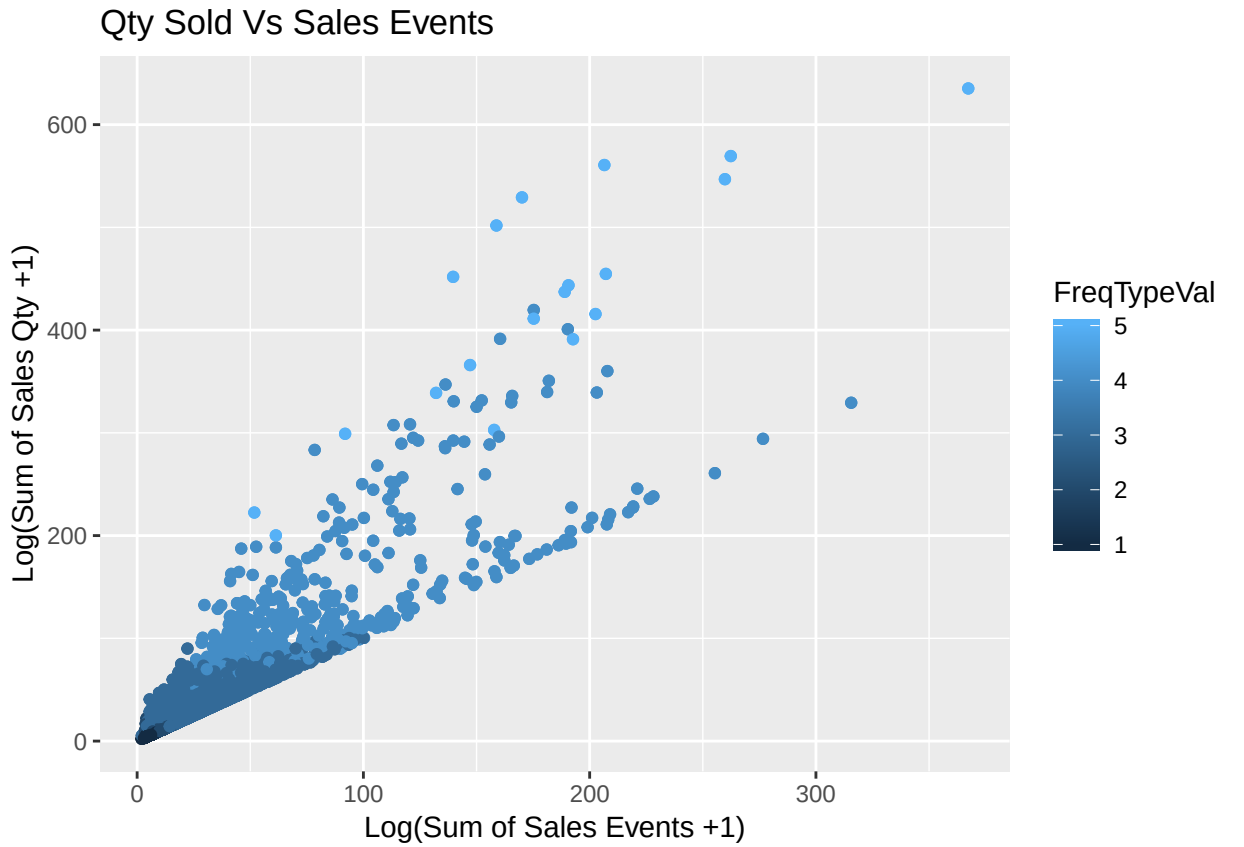
There is little point in us considering forecasting or modelling the 'oneseys and twoseys', those parts which have sold on 3 or less occassions, over the past 10yrs.

Let's clear them out and classify the data int groups which would be meaningful to our parts staff, such as: Rarity, Low usage, Normal usage, Bellweather, etc:

```
#group part numbers with similar prefix (first four digits)
parts_all_bymth_grp <- parts_all_bymth_sng %>%
                        group_by(PART_GRP, INVOICE_MONTH, MONTH_NO) %>%
                          summarise(SALES_QTY = sum(SALES_QTY),
```

```
                                SALES_EVENTS = sum(SALES_EVENTS)) %>%
                        ungroup() %>%
                          select(PART_NUMBER = PART_GRP, INVOICE_MONTH,
                                 SALES_QTY, SALES_EVENTS, MONTH_NO) %>%
                            arrange(PART_NUMBER, INVOICE_MONTH)

#unify the parts and part groups into one table
parts_all_bymth <- rbind(
                      parts_all_bymth_grp %>% mutate(ISGRP = 1) %>%
                        select(ISGRP, PART_NUMBER, INVOICE_MONTH,
                               SALES_QTY, SALES_EVENTS, MONTH_NO)
                      ,
                      parts_all_bymth_sng %>% mutate(ISGRP = 0) %>%
                        select(ISGRP, PART_NUMBER, INVOICE_MONTH,
                               SALES_QTY, SALES_EVENTS, MONTH_NO)
                      ,
                      stringsAsFactors = FALSE
                      )


# Classify into groups

parts_frequent <- parts_all_bymth %>%
                  filter(ISGRP == 0) %>%
                  group_by(PART_NUMBER) %>%
                  summarise(
                            TOTAL_QTY_SOLD         = sum(SALES_QTY),
                            TOTAL_QTY_SOLD_LOG     = sum(log(SALES_QTY+1)),
                            TOTAL_SALES_EVENTS     = sum(SALES_EVENTS),
                            TOTAL_SALES_EVENTS_LOG = sum(log(SALES_EVENTS+1)),
                            FIRST_SALE             = min(INVOICE_MONTH),
                            LAST_SALE              = max(INVOICE_MONTH)
                            ) %>%
                  mutate(FreqType = ifelse((TOTAL_QTY_SOLD <=   12 | TOTAL_SALES_EVENTS <=  3),
                                           'Rarity',
                                    ifelse((TOTAL_QTY_SOLD <=   24 | TOTAL_SALES_EVENTS <=  6),
                                           'Low',
                                    ifelse((TOTAL_QTY_SOLD >=10000),
                                           'NutsAndBolts',
                                    ifelse((TOTAL_QTY_SOLD >=  240 & TOTAL_SALES_EVENTS >= 48),
                                           'BellWeather',
                                           'Normal')))),
                      FreqTypeVal = ifelse((TOTAL_QTY_SOLD <=   12 | TOTAL_SALES_EVENTS <=  3), 1,
                                    ifelse((TOTAL_QTY_SOLD <=   24 | TOTAL_SALES_EVENTS <=  6), 2,
                                    ifelse((TOTAL_QTY_SOLD >=10000) ,                           5,
                                    ifelse((TOTAL_QTY_SOLD >=  240 & TOTAL_SALES_EVENTS >= 48), 4,3))))
                  )

# Let's plot our data, must have at least 6 sales events
plotdat <- parts_frequent %>% filter(TOTAL_SALES_EVENTS >= 6)
p2 <- ggplot(plotdat,
        aes(x=TOTAL_SALES_EVENTS_LOG,
            y=TOTAL_QTY_SOLD_LOG,
```

```
            color=FreqTypeVal))              +
        geom_point()                         +
        ggtitle("Qty Sold Vs Sales Events") +
        xlab("Log(Sum of Sales Events +1)") +
        ylab("Log(Sum of Sales Qty +1)")    +
        geom_point()
p2
```



That looks more or less the same, how many parts were excluding by the filtering process?

```
library(scales)
print(paste0("Parts in original dataset: ", nrow(parts_sums)))
```

```
## [1] "Parts in original dataset: 26835"
```

```
print(paste0("Parts in filtered subset:  ", nrow(plotdat)))
```

```
## [1] "Parts in filtered subset:  6183"
```

```
print(paste0("Proportion filtered out:   ", percent( (nrow(parts_sums)-nrow(plotdat))
                                                    /nrow(parts_sums) )))
```

```
## [1] "Proportion filtered out:   77%"
```

So we've cropped approx 3/4 of the parts! Although the chart appears almost the same. All the cropped data would have been in the far bottom left of the chart, near 0,0.

Below is a funciton for further cropping the data. In the event, we didn't apply any more filters, but the code remains.

```r
# subset the data for analysis

getSelectedTables <- function(parts_all_bymth_equiv,
                              exclude_grps                       = TRUE,
                              SalesMustHaveCommencedBeforeDate = NA,
                              SalesMustHaveCeasedAfterDate     = NA,
                              MinimumQtyOfSalesEvents          = 6){
  require(dplyr)

  #Set default period as wide as possible
  if(is.na(SalesMustHaveCommencedBeforeDate)){
    SalesMustHaveCommencedBeforeDate <- max(parts_all_bymth_equiv$INVOICE_MONTH)
  }
  if(is.na(SalesMustHaveCeasedAfterDate)){
    SalesMustHaveCeasedAfterDate <- min(parts_all_bymth_equiv$INVOICE_MONTH)
  }

  selected_partnumbers <-  parts_all_bymth_equiv %>%
                           group_by(PART_NUMBER, ISGRP) %>%
                             summarise(TOTAL_QTY_SOLD     = sum(SALES_QTY),
                                       TOTAL_SALES_EVENTS = sum(SALES_EVENTS),
                                       FIRST_SALE         = min(INVOICE_MONTH),
                                       LAST_SALE          = max(INVOICE_MONTH)) %>%
                               filter(TOTAL_SALES_EVENTS >= MinimumQtyOfSalesEvents &
                                      FIRST_SALE <= SalesMustHaveCommencedBeforeDate &
                                      LAST_SALE  >= SalesMustHaveCeasedAfterDate ) %>%
                                 ungroup()

  selected_partnumbers_series <- parts_all_bymth_equiv %>%
                                   inner_join(y=selected_partnumbers, by="PART_NUMBER") %>%
                                     select(PART_NUMBER, INVOICE_MONTH, SALES_QTY, SALES_EVENTS) %>%
                                       arrange(PART_NUMBER, INVOICE_MONTH)
  if(exclude_grps){
    selected_partnumbers <- selected_partnumbers %>%
                             filter(ISGRP==0)
    selected_partnumbers_series <- selected_partnumbers_series %>%
                                   inner_join(selected_partnumbers, by="PART_NUMBER")
  }

  #return...
  list(selected_partnumbers, selected_partnumbers_series)
}
```

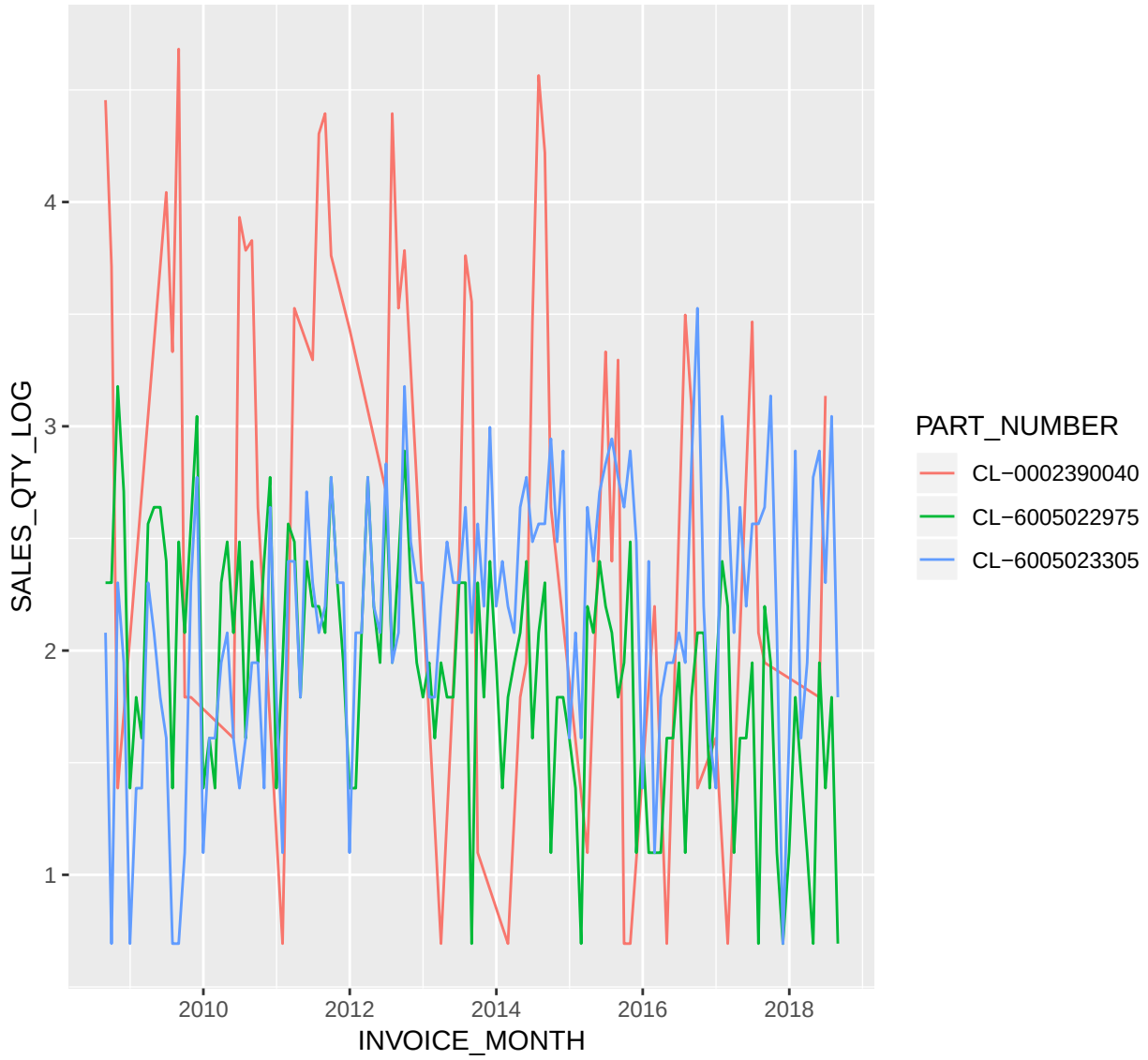
```r
output                       <- getSelectedTables(parts_all_bymth,
                                                  exclude_grps            = TRUE,
                                                  MinimumQtyOfSalesEvents = 6)
selected_partnumbers        <- output[[1]]
selected_partnumbers_series <- output[[2]]
rm(output)

output                       <- getSelectedTables(parts_all_bymth,
                                                  exclude_grps            = TRUE,
                                                  MinimumQtyOfSalesEvents = 0)
selected_partnumbers_all        <- output[[1]]
```

```
selected_partnumbers_series_all <- output[[2]]
rm(output)
```

View Examples of Part Number Series

Let's view five randomly selected series. A log transform [log(1+x)] is used to help plot them on the same scale.

```
library(tidyr)
library(dplyr)
library(ggplot2)

#We can only plot data which has points to plot, so filter by sales events:
plot_data    <- selected_partnumbers %>%filter(TOTAL_SALES_EVENTS > 150)

set.seed(20180701)
random_id    <- sample(1:nrow(plot_data), 3, replace=F)
random_parts <- plot_data$PART_NUMBER[random_id]

plot_data <- selected_partnumbers_series %>%
              filter(PART_NUMBER %in% random_parts) %>%
                mutate(SALES_QTY_LOG = log(1+SALES_QTY))

p <- ggplot(data=plot_data, aes(x=INVOICE_MONTH, y=SALES_QTY_LOG, color=PART_NUMBER)) +
     geom_line()

p
```



So in some series we have clear annual cycles, but across the series there are large differences in scale.

Distribution and Heteroscedasticity

We've looked at the distribution of the data as a whole and it is highly skewed, in no way Gaussian! But what about the distribution of a given part number's sales?

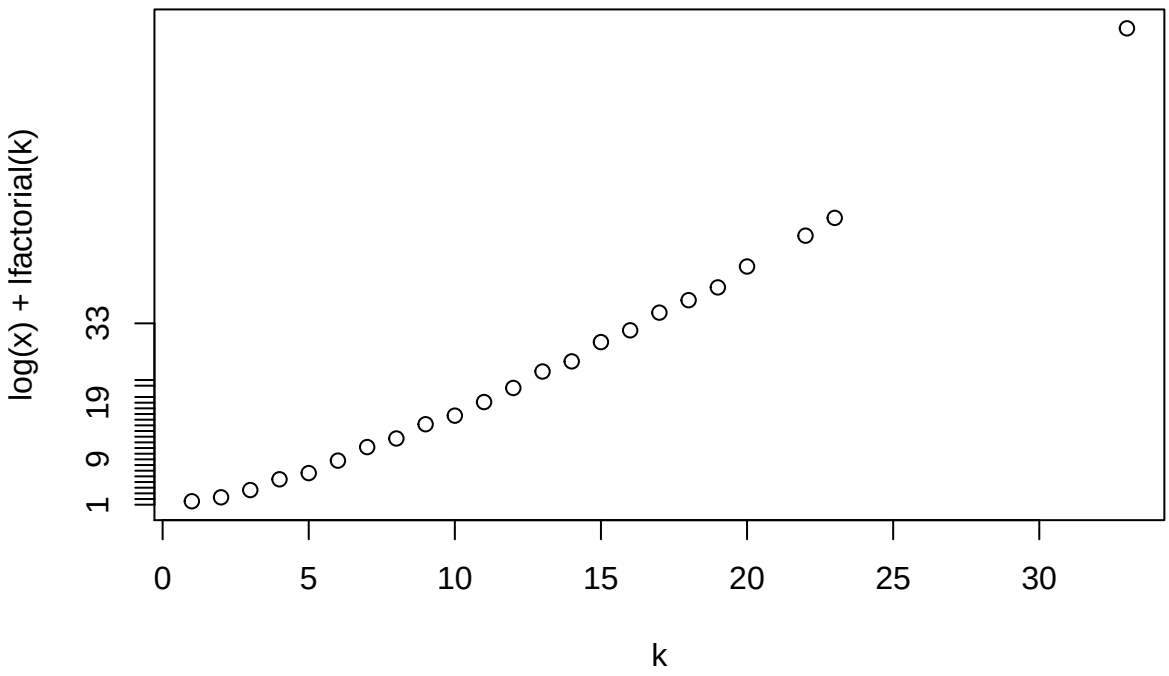
We may end up modelling using regression of some sort. So we would like the data to be normaly distributed, but more importantly, we want it to be homoskedastic (similar variance between predictors and predicted)

Sales are counts and the first thought is that counts are probably in Poisson distributions. But is the underlying process a Poisson process? We could argue this, but just as easy to look at what the data says.

Distribution of CL-6005023305

Let's look at an example series, "CL-6005023305", which of course doesn't represent all other parts, but its a good example to start with. If its Poisson then the plot is of a straight line. If there is a kink, then not a pure Poission.

```
#need to make sure all months are represented, not just those with sales
monthslist <- parts_all_bymth_sng %>% distinct(INVOICE_MONTH) %>% arrange(INVOICE_MONTH)
# remove last date, as nearly always incomplete
monthslist <- monthslist[-nrow(monthslist),]

example_series <- monthslist %>%
                  left_join(selected_partnumbers_series, by="INVOICE_MONTH") %>%
                    filter(PART_NUMBER %in% c("CL-6005023305", "CL-0002389610", "CL-0002389610"))%>%
                      filter(INVOICE_MONTH <= LAST_SALE  & INVOICE_MONTH >= FIRST_SALE )%>%
                        mutate(SALES_QTY = ifelse(is.na(SALES_QTY),0,SALES_QTY))

example_series_a <- example_series %>%
                      filter(PART_NUMBER=="CL-6005023305") %>%
                        select(INVOICE_MONTH, SALES_QTY)

# Data for Poisson Test
y <- example_series_a$SALES_QTY
n <- length(y)
x <- table(y)
k <- as.numeric(names(x))

#Data for Gaussian test

#if its Poisson then the plot is of a straight line.
#If there is a kink, then not a Poission
plot(k,log(x)+lfactorial(k))
```



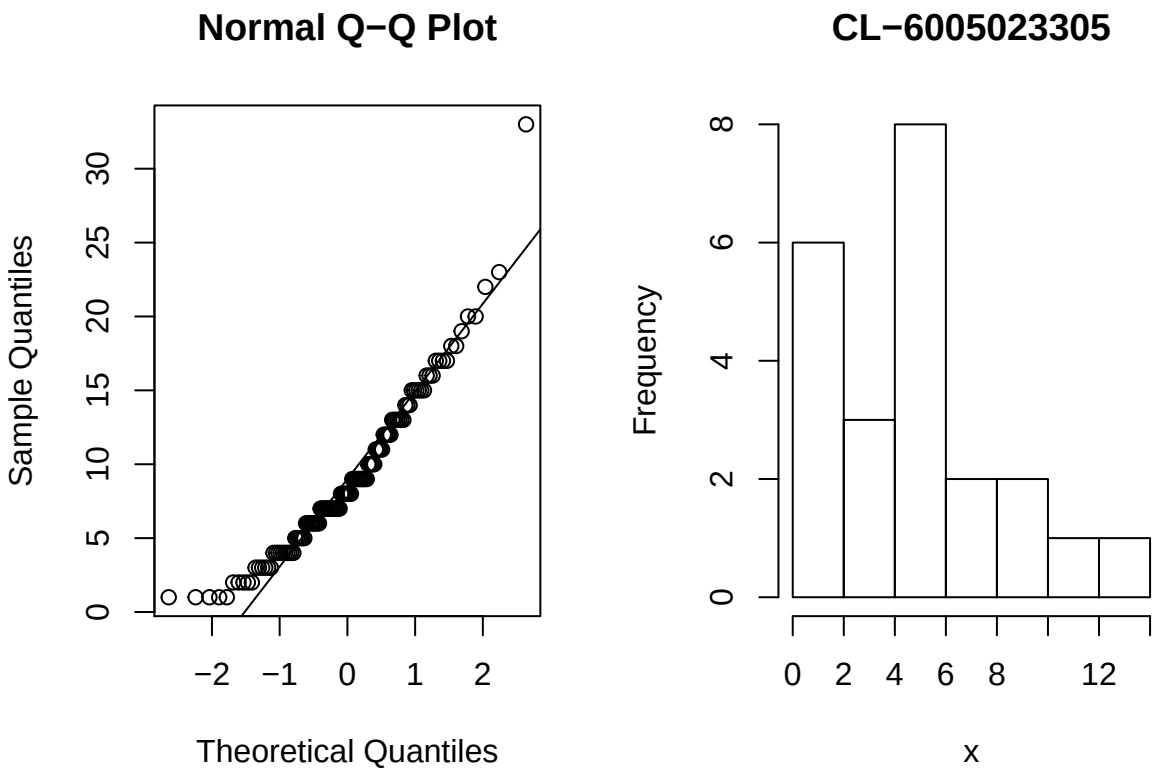
That looks fairly straight, this series could be approximated as a Poisson distribution. We can do a score on that, if over 1 then probably a Poisson.:

```
#score for 'Is Poisson', if over 1 then probably a Poisson.
n*(1-cor(k,log(x)+lfactorial(k))^2)
```

```
## [1] 2.290153
```

So the evidence is building that this is best modelled as a Poisson distribution. How does CL-6005023305 sales data look vs the Gaussian (Normal) distribution? Let's use a QQ plot and a histogram

```
par(mfrow=c(1,2))
qqnorm(y)
qqline(y)
hist(x, main="CL-6005023305")
```

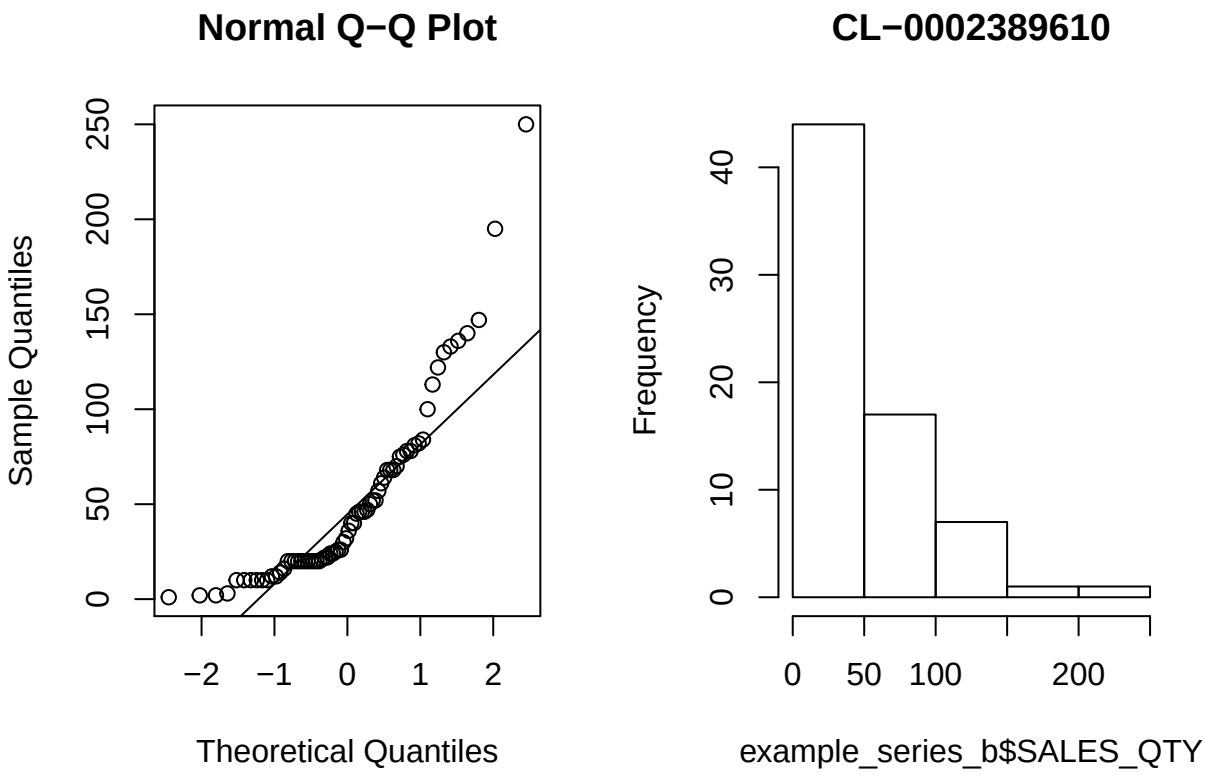


From visual inspection this is somewhat Gaussian, but differs at the extremes.

Distibution of CL-0002389610

Let's try another part...

```
example_series_b <- example_series %>%
                      filter(PART_NUMBER=="CL-0002389610") %>%
                        select(INVOICE_MONTH, SALES_QTY)

# remove last date, as nearly always incomplete
example_series_b <- example_series_b[-(nrow(example_series_b)),]


par(mfrow=c(1,2))

qqnorm(example_series_b$SALES_QTY)
qqline(example_series_b$SALES_QTY)
hist(example_series_b$SALES_QTY, main="CL-0002389610")
```



The histogram is of a count, a Poisson distribution, or possibly an Exponetial series. This is confirmed by the QQ plot, so this part is far closer to a Poisson distribution than a Gaussian distribution.

Distributions for Modelling

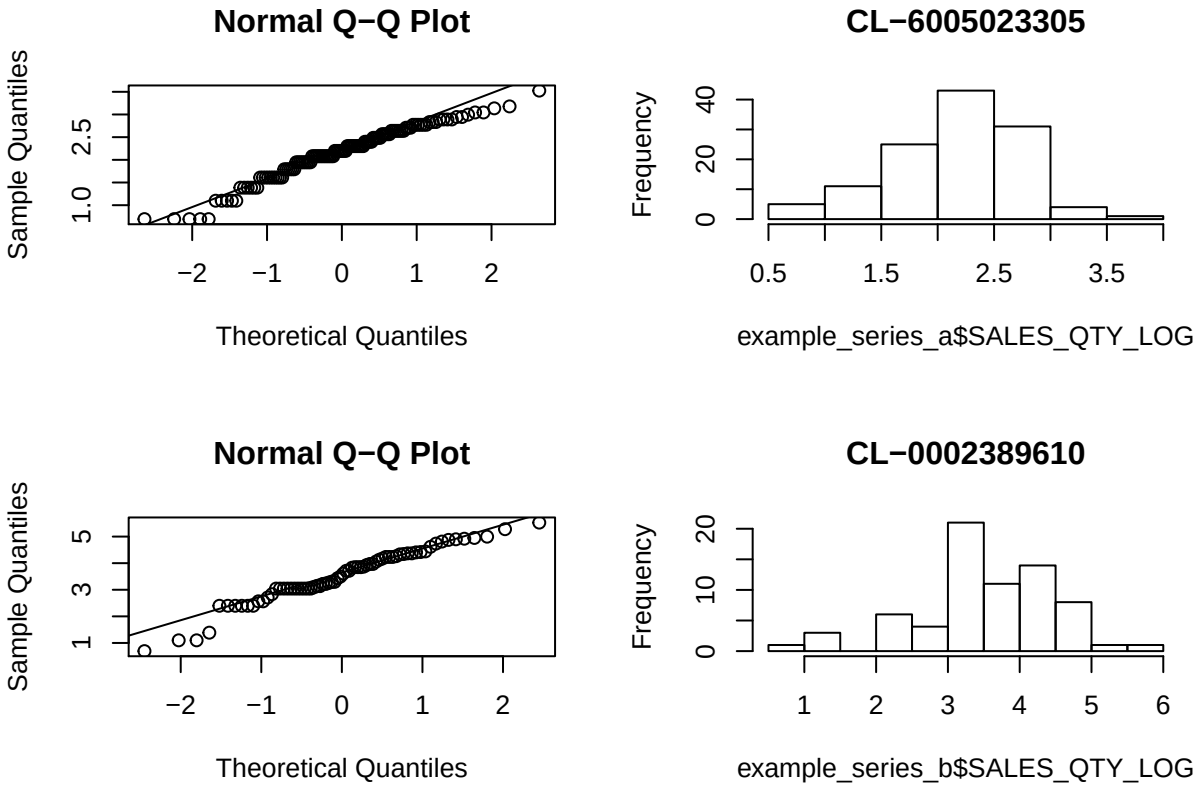
Some models assume that residuals are normally distributed. They work better when the target data is normally distributed. As this is time series data, we can imagine models which take previous counts, either event (invoices) or sales quantities, as inputs to a model forecasting the count next period. Furthermore, transforms liks BoxCox also reduce heteroskedacicity (more later). So it may be productive to consider transforms which move us from data on the Poisson distirbution to the Gaussian.

Log(x+1) to Transform to Gaussian ?

We can transform Poisson data into Gaussian using Box-Cox, but there is a simple, albeit rough, alternative. Poisson data is very closely related to exponential data, so simply applying log(x+1) is a common gambit to shoe horn such data into gaussian distribution for further modelling and analysis. BUT It is shoe horning, its better than no transform, but errors get poorly represented in a log transform. Hence tools which seek to minimise RMSE don't necessarily perform well. They don't know errors are actually the log of the real error!

```
example_series_a$SALES_QTY_LOG <- log(example_series_a$SALES_QTY+1)
example_series_b$SALES_QTY_LOG <- log(example_series_b$SALES_QTY+1)
par(mfrow=c(2,2))

#CL-6005023305
qqnorm(example_series_a$SALES_QTY_LOG)
```

```
qqline(example_series_a$SALES_QTY_LOG)
hist(example_series_a$SALES_QTY_LOG, main="CL-6005023305")

#CL-0002389610
qqnorm(example_series_b$SALES_QTY_LOG)
qqline(example_series_b$SALES_QTY_LOG)
hist(example_series_b$SALES_QTY_LOG, main="CL-0002389610")
```



So log tranform is better than no transform, but we can see the extremes are still not Gaussian. Do we even pass the Shapiro Wilks Normality test?

Note, we are testing the NULL hypothesis that the distribution is normal.If p<0.05 then we must reject the null and assume NOT normal.

```
# need to get over 0.05 for 95% confidence that distribution is Gaussian.
shapiro.test(example_series_a$SALES_QTY_LOG)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  example_series_a$SALES_QTY_LOG
## W = 0.96781, p-value = 0.005657
```

```
shapiro.test(example_series_b$SALES_QTY_LOG)
```

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  example_series_b$SALES_QTY_LOG
## W = 0.96429, p-value = 0.04336
```

CL-0002389610 does not pass for Gaussian. Note, there are real limits to this test's usefulness. To the eye, the histogram of 6005023305 looks convincing more gaussian than the histogram of 0002389610, but the SW scores are the other way.

See this link for discussion of normality tests: https://stackoverflow.com/questions/7781798/seeing-if-data-is-normally-distributed-in-r/7788452#7788452

Box Cox

This is the classic transform to get Gaussian data from non gaussian. Better than a simple log(x+1). First we create a model from which we then 'predict' the boxcox transforms we need.

```
library(caret)

# get model
BoxCox_0002389610 <- BoxCoxTrans(example_series_b$SALES_QTY) #SALES_QTY must be +ve

# predict boxcox transforms
example_series_b$SALES_QTY_BC <- predict(BoxCox_0002389610, example_series_b$SALES_QTY)

# plot hist
#hist(example_series_b$SALES_QTY_BC, main="Box Cox: CL-0002389610")

# Shapiro Wilks
print(shapiro.test(example_series_b$SALES_QTY_BC))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  example_series_b$SALES_QTY_BC
## W = 0.98058, p-value = 0.3491
```

SW is >0.05, so retain null hypothesis that data is gaussian. Hence, Box Cox made our data Gaussian, whereas log(x+1) did not.

Limits of Box Cox

Predictors used with the Box-Cox transformation must be strictly positive. The Yeo-Johnson transformation is similar to the Box-Cox model but can accommodate predictors with zero and/or negative values. Yeo-Johnson is more processor intensive.

Box Cox is good for transforming predictors, but gets interesting in time series where previous target are next period's preditors. Do we boxCox the target as well as the predictors? Not so easy because we havn't got a function to invert the BoxCox of the target, to get it back to a useful figure, although we could derive/plot a mapping. This is where log(x+1) is satisfyingly simple.

Heteroscedasticity

This means the "variance of a target is unequal across the range of values of its predictors". Classic example is income and age. Age is a predictor of income, but variance in income increases as age increases. All start on low wages, some get richeer, some don't.

Heteroskedacity causes problems for regression models because they aim to reduce errors. But, in the above example, the error will be low at low ages, where the variance is low, but high at higher ages where variance is high. What error you get will depend on where your data tends to be clumped.

This text is a placeholder for investigating the problem when looking at a regression model. If there is a problem then the Scale-Location plot resulting from plot(model) will show a red line moving up (or down), as opposed to flat. That plot shows the normalised errors vs the location of the error. If the variance changes with location, ie there is a trend in the line, then we have heteroskedacity.

Box Cox (caret:: BoxCoxTrans()) *of the target variable* can make the residuals homoscedastic. You can apply it as a preProcess: preProcess(x, method = c("BoxCox", "center", "scale"))

TSCount Package for Poisson Data

"Autoregressive Conditional Poisson (ACP) proposed by Heinen (2003) is for cases of count data exhibiting autoregressive behaviour. When a count data set exhibits time dependence the plain Poisson regression is not adequate."

That sounds like our data, it has Poisson distributed counts, but is autoregressive. Autoregressive means the previous count affects the next count. For example we know our data is seasonal, counts in August are affected by counts in July. We also know it has cyclic trends, counts this year are affected by counts last year.

The tscount() package presents a generalised version of functions in the acp() package.

There's a good example of this at: http://www.stat.unc.edu/faculty/pipiras/timeseries/Nonlinear_2_-_Count_time_series_-_Menu.html

```
library(tscount)
```

```
## Warning: package 'tscount' was built under R version 3.5.1
```

```
fcastfrom_tsc <- length(example_series_a$INVOICE_MONTH)-12
fcastto_tsc   <- length(example_series_a$INVOICE_MONTH)

CL6005023305_model <- tsglm(example_series_a$SALES_QTY[-c(fcastfrom_tsc:fcastto_tsc)],
                            model = list(past_obs = c(1,12)),
                            distr = "poisson")

CL6005023305_model_preds <- predict(CL6005023305_model,
                                    n.ahead = 12)

summary(CL6005023305_model)
```

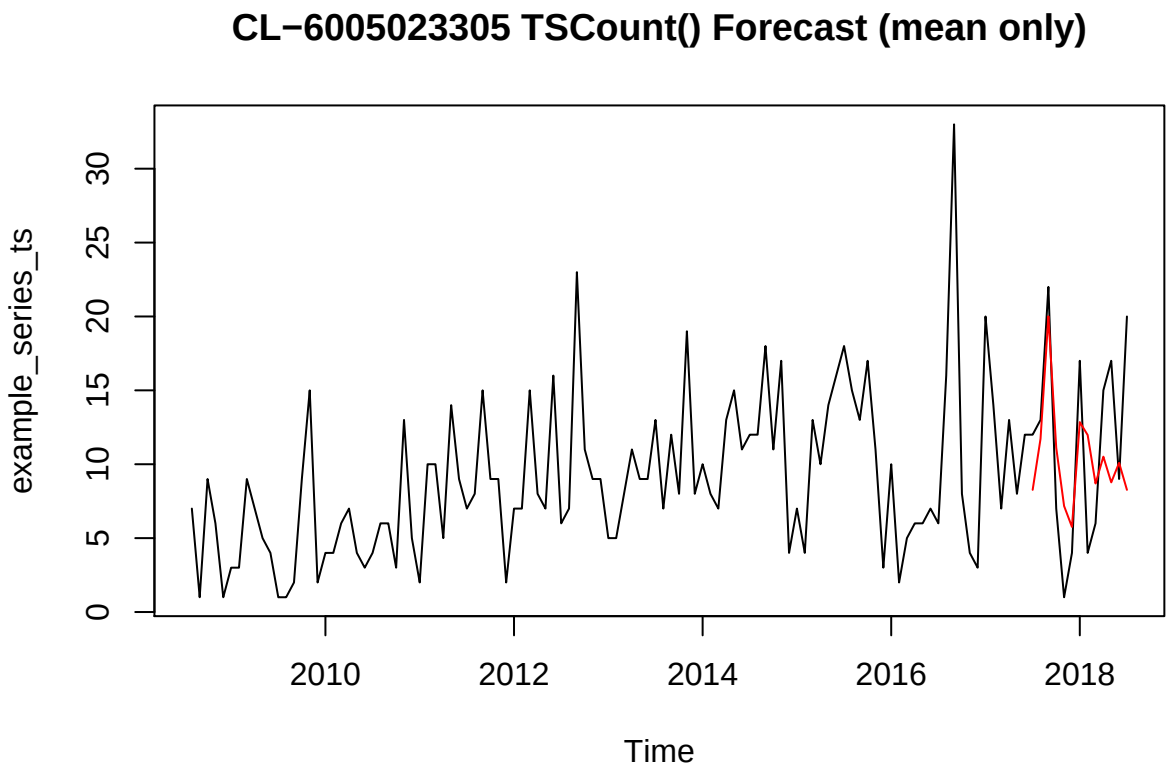
```
##
## Call:
## tsglm(ts = example_series_a$SALES_QTY[-c(fcastfrom_tsc:fcastto_tsc)],
##     model = list(past_obs = c(1, 12)), distr = "poisson")
##
## Coefficients:
## Standard errors and confidence intervals (level =  95 %) obtained
## by normal approximation.
##
## Link function: identity
## Distribution family: poisson
## Number of coefficients: 3
## Log-likelihood: -334.7398
```

```
## AIC: 675.4795
## BIC: 683.498
## QIC: 675.5449
```

```
#NB we can also build regressors into the model, eg:
# timeseries_win <- window(timeseries, end = 1981 + 11/12)
# regressors_win <- window(regressors, end = 1981 + 11/12)
# model <- tsglm(timeseries_win,
#                        model = list(past_obs = c(1, 12)),
#                        link  = "log",
#                        distr = "poisson",
#                        xreg  = regressors_win)
```

Let's see how the model would have done forecasting the final 12months of our example data...

```
library(lubridate)

# get the series as a ts object, hlepful when plotting
start_date <- example_series_a$INVOICE_MONTH[1]
rowcount   <- nrow(example_series_a)
end_date   <- example_series_a$INVOICE_MONTH[rowcount]

example_series_ts <- ts(example_series_a$SALES_QTY,
                        start= c(lubridate:: year(start_date), lubridate::month(start_date)),
                        end  = c(lubridate:: year(end_date),   lubridate::month(end_date)),
                        frequency = 12)

# get forecast as ts object
fcastfrom_tsc <- example_series_a$INVOICE_MONTH[rowcount-12]
fcastto_tsc   <- example_series_a$INVOICE_MONTH[rowcount]

fcast_tsc <-ts(CL6005023305_model_preds$pred,
               start= c(lubridate:: year(fcastfrom_tsc), lubridate::month(fcastfrom_tsc)),
               end  = c(lubridate:: year(fcastto_tsc),   lubridate::month(fcastto_tsc)),
               frequency = 12)

# plot the two series
plot(example_series_ts, main="CL-6005023305 TSCount() Forecast (mean only)")
lines(fcast_tsc, col="red")
```



Not bad, but there is of course, a more commonly used tool for time series forecast, ARIMA.

Using ARIMA

The obvious choice when modelling time series is ARIMA (AutoRegressive, Integrative, Moving Average). There won't be enough data to build an ARIMA model for most part numbers, but the sales quantity will be in the big sellers, where they may be enough data.

ARIMA is applicable to Poisson data, although the AIC optimisation score suffers on non gaussian data.

ARIMA allows us to build a model of the time series and create a forecast, just what we need. We can pitch ARIMA vs a poisson glm model.

Note, there is actually a 'Generalised' ARIMA package for R called gsarima(), which is designed to handle non Gaussian distributions. But insufficient time is available to investigate and compare it properly.

Understanding ARIMA

A general introduction to time series at: https://onlinecourses.science.psu.edu/stat510/node/72/

There is an introduction to ARIMA at: https://www.datascience.com/blog/introduction-to-forecasting-with-arima-in-r-learn-da

Nice explanation, with charts, of stationarity at: https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/

Good examples on differencing and de-trending in R at: http://r-statistics.co/Time-Series-Analysis-With-R.html

and the complete story at: http://people.duke.edu/~rnau/411arim3.htm

ARIMA Example

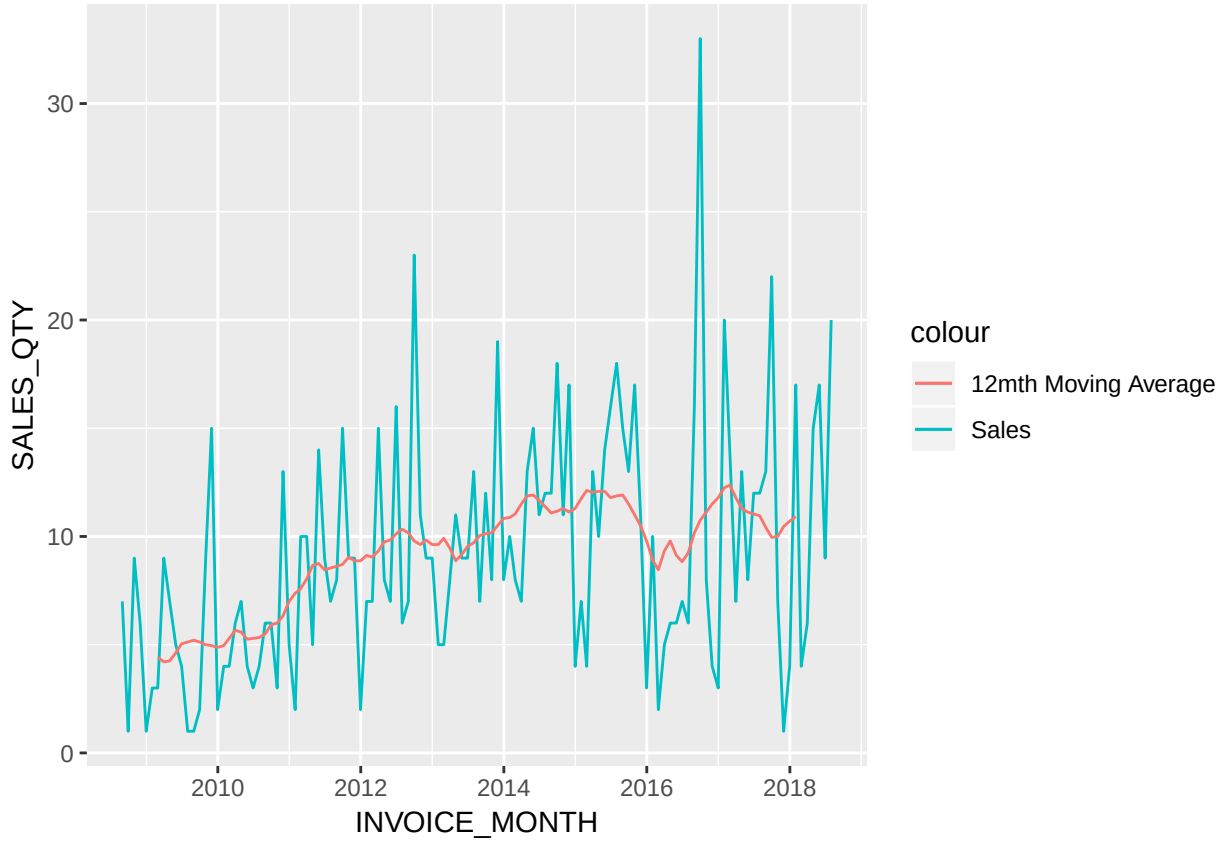
Let's choose an example and explore it as an ARIMA problem. CL-6005023305 is a simple example with many data points.

```
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.5.1
```

```
#get moving average (over rolling 12 month period, i.e. order=12)
example_series_a$ma <- ma(example_series_a$SALES_QTY, order=12)

p <- ggplot(data=example_series_a, aes(x=INVOICE_MONTH)) +
     geom_line(aes(y=SALES_QTY, colour="Sales")) +
     geom_line(aes(y=ma,        colour="12mth Moving Average"))

p
```

```
## Warning: Removed 12 rows containing missing values (geom_path).
```

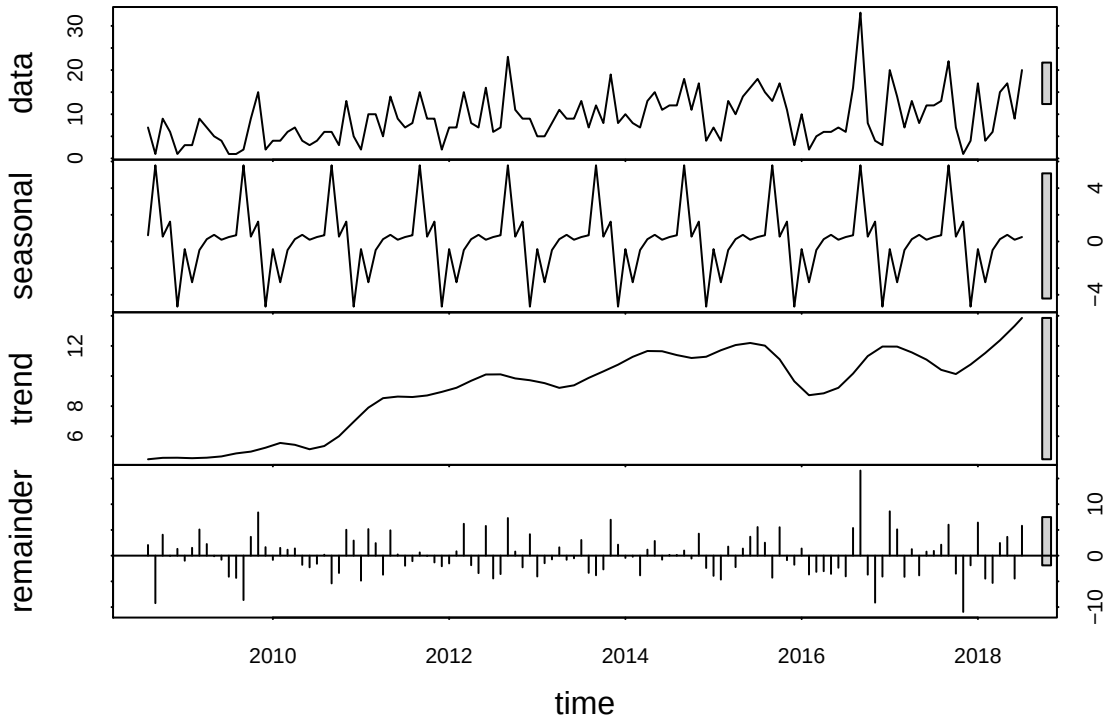


It appears to be seasonal, peaking in mid year. Let's decompose it into its trend, seasonality and residuals

```
library(tseries)
```

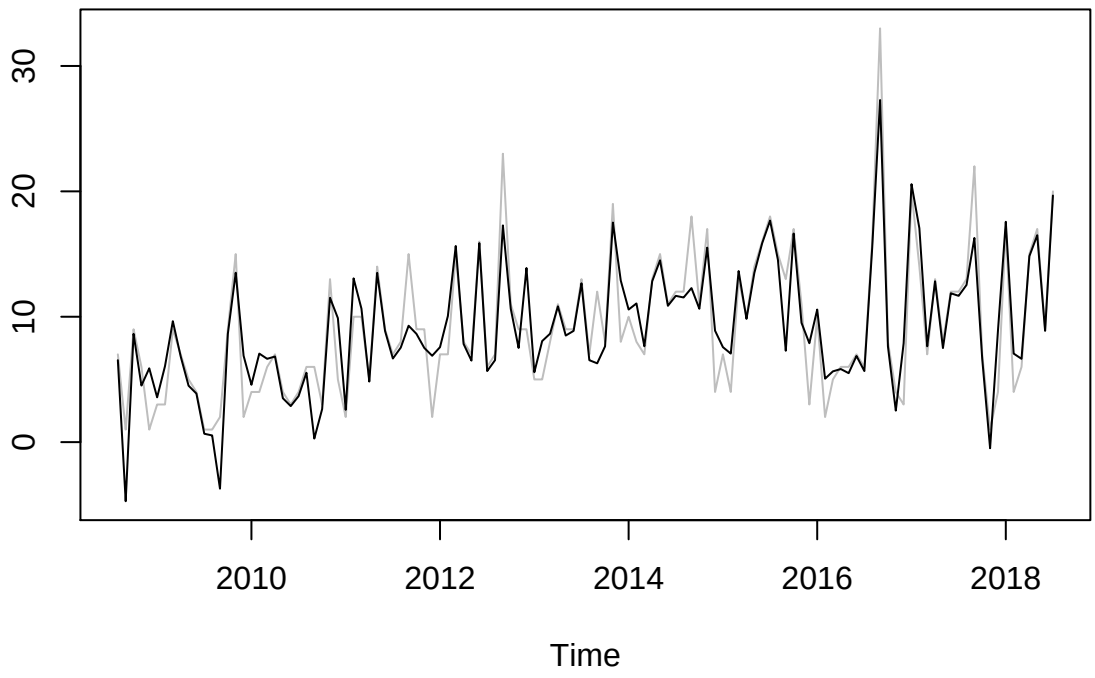
```
## Warning: package 'tseries' was built under R version 3.5.1
```

```r
#convert to ts object

decomposition <- stl(example_series_ts, s.window="periodic")
plot(decomposition)
```



Let's remove the seasonality, we are only interested in annual sales, not seasonal.

```r
example_series_ds <- seasadj(decomposition)
ts.plot(example_series_ts, example_series_ds, gpars = list(col = c("grey", "black")))
```



Stationarity

Nice explanation, with charts, of stationarity at: https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/

We use the augmented Dicky Fuller test (adf) to decide whether the series is stationary. The null hypothesis assumes that the series is NOT stationary. We are looking to reject the null hypothesis, so if $p < 0.05$ we reject and assume the series IS stationary.

```
library(tseries)
adf.test(example_series_ds)
```

```
## Warning in adf.test(example_series_ds): p-value smaller than printed p-
## value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  example_series_ds
## Dickey-Fuller = -4.2685, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
#differencing, if required:
#diff(example_series_ds, differences = 12) # where 12=however many you decide
```

The p_value is <0.05 so we reject non-stationary and conclude it is stationary. Although to the eye it looks like it may have an underlying trend. So let's double check with acf plots

Note, if it had not been stationary then we would have done some 'differencing'. This is the I part of ARIMA. We calculate the differences from point to point, then perform adf on these differences as a series. Full explanation with R examples at: http://r-statistics.co/Time-Series-Analysis-With-R.html

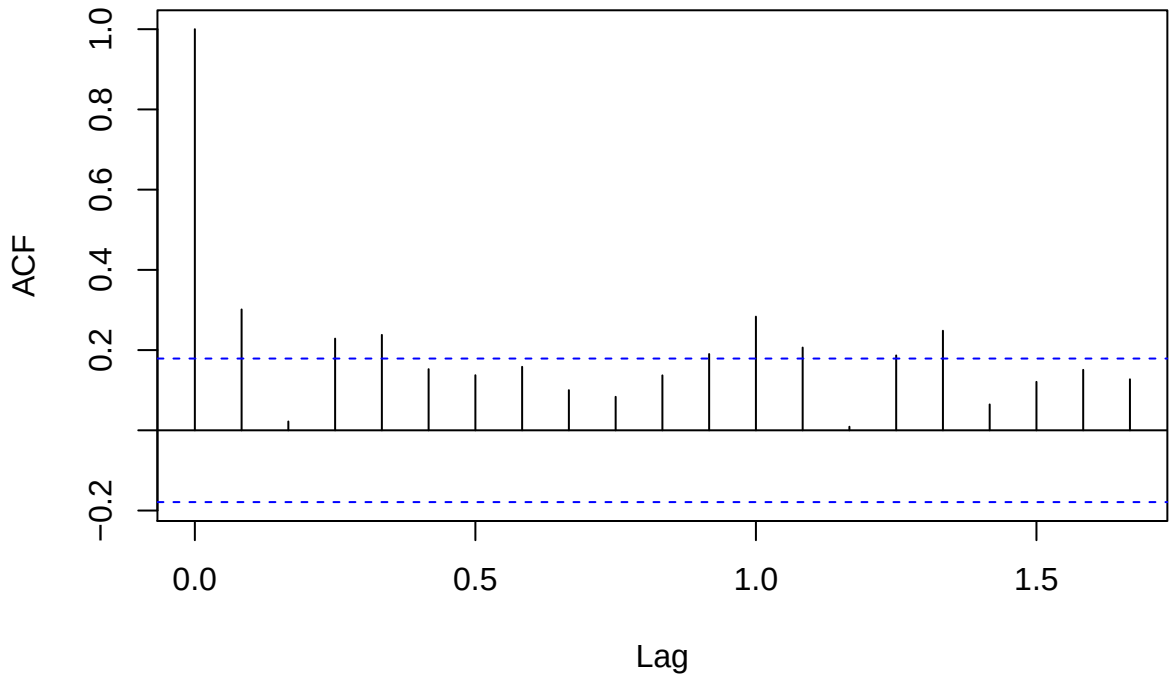
We would have needed two differencing values: seasonal and regular, found using nsdiff() and ndiff()

Autocorrelation plots (also known as ACF or the auto correlation function) are a useful visual tool in determining whether a series is stationary. If the series is correlated with its lags then, generally, there are some trend or seasonal components remaining and therefore its statistical properties are not constant over time.

ACF plots display correlation between a series and itself, but 'x' lags ago. This suggests the order of differencing. ACF plots can also help in determining the order of the MA(q) component of the model.

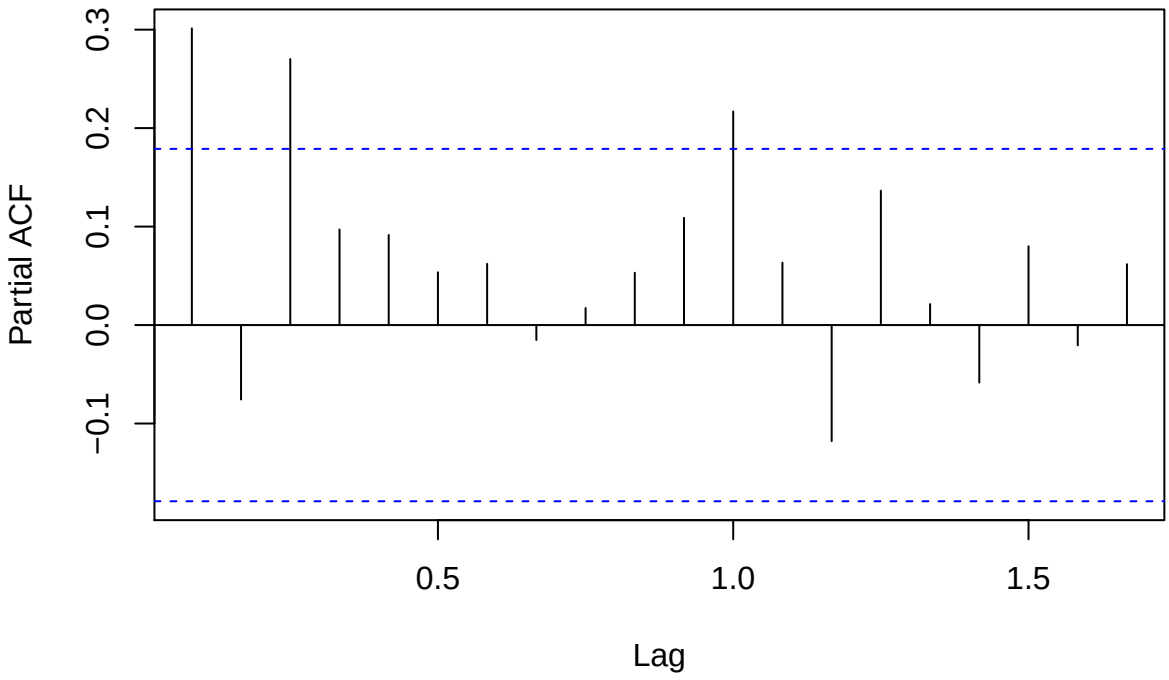
PACF (partial autocorrelation plots) display correlation between a variable and its lags that is NOT explained by previous lags. PACF plots are useful when determining the order of the AR(p) component of the model.

```
acf(example_series_ds, main='')
```



there is a significiant lag at 1, we could difference, but adf.test says don't bother.

```
pacf(example_series_ds, main='')
```



The PACF also points out significant auto correlations at lag 1. This suggests that we might want to test models with AR or MA components of order 1. IF there had been a spike at 7 (>6) it might suggest that there is a 'seasonal' pattern present.

The forecast package allows us to specify the order of the model using the arima() function, or automatically generate a set of optimal (p, d, q) using auto.arima().

We judge goodness of fit using: Akaike information criteria (AIC) Baysian information criteria (BIC) Lower is better.

These criteria are closely related and can be interpreted as an estimate of how much information would be lost if a given model is chosen.

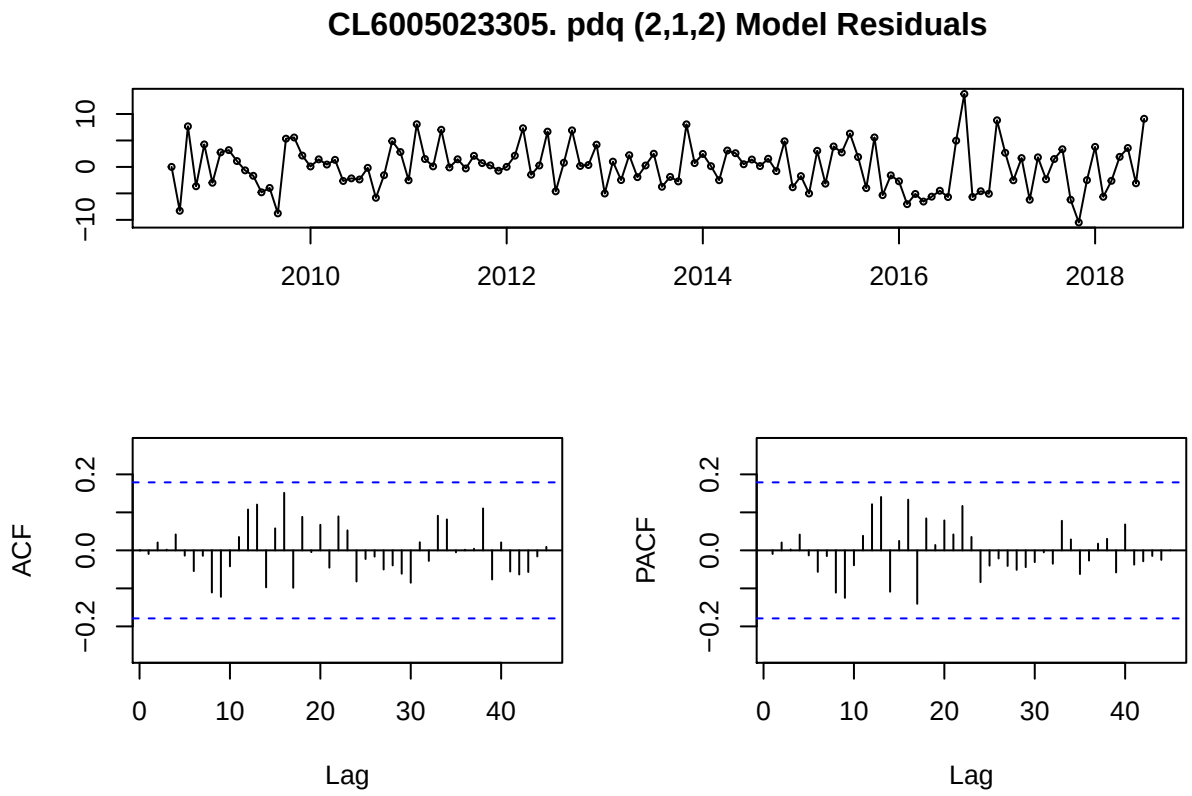
```
library(forecast)
auto.arima(example_series_ds, seasonal=FALSE)
```

```
## Series: example_series_ds
## ARIMA(2,1,2) with drift
##
## Coefficients:
##           ar1      ar2      ma1      ma2   drift
##       -0.3189  -0.1977  -0.4246  -0.5070  0.0639
## s.e.   0.2707   0.1286   0.2732   0.2764  0.0221
##
## sigma^2 estimated as 18.93:  log likelihood=-342.61
## AIC=697.22   AICc=697.97   BIC=713.9
```

Auto.arima used AR(p)=2, I(d)=1, MA(q)=2 Meaning, it autoregressed to the second lag. Applied differenced once, and a Moving Average of order =1

Let's see the model residuals for any patterns:

```
fit <- auto.arima(example_series_ds, seasonal=FALSE)

tsdisplay(residuals(fit), lag.max=45, main='CL6005023305. pdq (2,1,2) Model Residuals')
```

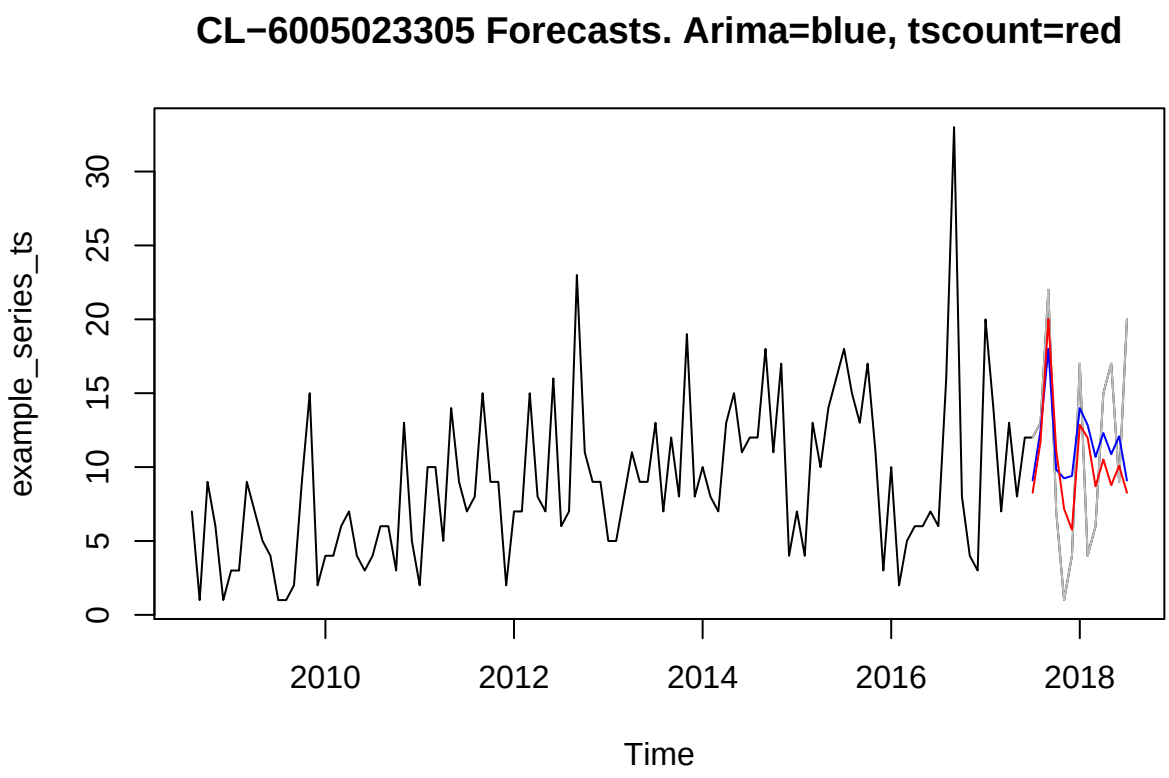


No 'significant' patterns reported by ACF or PACF, so model is as good as can be expected.

Let's ask the model to forecast 12months for which we already have data, ie we 'hold out' that data:

```
# define hold out period

fcastfrom     <- length(example_series_ts)-12
fcastto       <- length(example_series_ts)
trainto       <- length(example_series_ts)-13

fcastfrom_yr  <- year( example_series_a$INVOICE_MONTH[fcastfrom])
fcastfrom_mth <- month(example_series_a$INVOICE_MONTH[fcastfrom])

fcastto_yr    <- year( example_series_a$INVOICE_MONTH[fcastto])
fcastto_mth   <- month(example_series_a$INVOICE_MONTH[fcastto])

trainto_yr    <- year( example_series_a$INVOICE_MONTH[trainto])
trainto_mth   <- month(example_series_a$INVOICE_MONTH[trainto])

# get the forecast and training windows
fcastwin <- window(example_series_ts, start = c(fcastfrom_yr, fcastfrom_mth))
trainwin <- window(example_series_ts, end   = c(trainto_yr,   trainto_mth))
```

```r
# fit to data EXC holdout
fit_trainwin <- auto.arima(trainwin, seasonal = TRUE, approximation = TRUE)

#forecast
fcast <- forecast(fit_trainwin, h = 12)

#plot
fcastfrom <- example_series_a$INVOICE_MONTH[fcastfrom]
fcastto   <- example_series_a$INVOICE_MONTH[fcastto]

fcast_ts <-ts(fcast$mean,
              start= c(fcastfrom_yr, fcastfrom_mth),
              end  = c(fcastto_yr,   fcastto_mth),
              frequency = 12)

# plot mean
plot(example_series_ts, main="CL-6005023305 Forecasts. Arima=blue, tscount=red")
#Add actual
lines(fcastwin, col="grey")
#Add ARIMA forecast
lines(fcast_ts, col="blue")
#Add tscount() forecast
lines(fcast_tsc, col="red")
```



Auto.Arima

We have potentially thousands of time series to model. We cannot hand craft an ARIMA function for each series. Thankfully, auto.arima() is available in the forecast package. Its not perfect, but our only hope for handling so many models.

The above auto.arima output shows ARIMA(0,0,1)(0,0,1)[12]

where the first (0,0,1) is (p,d,q) for the non seasonal component of the model p = AutoRegressive, d = Integrative, q = Moving Average

'p' refers to the use of past values in the regression equation for the series Y. The auto-regressive parameter p specifies the number of lags used in the model.

'd' represents the degree of differencing in the integrated component. Differencing a series involves simply subtracting its current and previous values d times. Often, differencing is used to stabilize the series when the stationarity assumption is not met. A series is said to be stationary when its mean, variance, and autocovariance don't change over time.

'q' represents the error of the model as a combination of previous error terms. The order q determines the number of terms to include in the model

The second (0,0,1) is (P, D, Q) for the seasonality component P = AutoRegressive, D = Integrative, Q = Moving Average (as above)

tscount() vs auto.arima()

We have witnessed two valid time series methods. Which to use on our data? A random sample of approx 1000 time series would be a fair test.

```
#define a function to get the samples
get_PartNumber_sample <- function(Source_df, FreqType_chr, SampleQty_num){
  Source_df %>%
    filter(FreqType==FreqType_chr) %>%
      sample_n(SampleQty_num, replace = FALSE) %>%
        select(PART_NUMBER)
}

# We DONT want a representative sample, else that would be all 'Rarity'.
# we want a definite skew towards Normal and Bellweather, which is where the money is.
# Just a taste of the others
# 2000 total
samplepartnums <- rbind(get_PartNumber_sample(parts_frequent, "BellWeather",   300),
                        get_PartNumber_sample(parts_frequent, "Low",           400),
                        get_PartNumber_sample(parts_frequent, "Normal",       1000),
                        get_PartNumber_sample(parts_frequent, "NutsAndBolts",   15),
                        get_PartNumber_sample(parts_frequent, "Rarity",        285)
                        )

partnumbers_top_pc <- selected_partnumbers %>% #already limited to at least 6 events
                        inner_join(parts_decode %>% select(PART_NUMBER, R_R_PRICE), by="PART_NUMBER") %
                          filter(PART_NUMBER %in% samplepartnums$PART_NUMBER)

series_top_pc      <- selected_partnumbers_series %>% #already limited to at least 6 events
                          filter(PART_NUMBER %in% samplepartnums$PART_NUMBER)
```

We'll need some functions to build and compare Arima and tscount models. They need try/catch logic because there may not be enough data in a given series to build a model.

Function for tscount() Models

```
# function for creating the TS Count object
# error catching is applied because sparse data means TScount model may not always be possible
getTScount_Obj <- function(timeseriesobject) {
  require(tscount)
  tscount_obj <- tryCatch(
    {
      tscount_obj <- tsglm(timeseriesobject,
                           model = list(past_obs = c(1,12)),
                           distr = "poisson")
    },
    error=function(cond) {
      #TimeSeries Modelling Error
      return("Error")
    },
    warning=function(cond) {
      # warningc
      return("Warning")
    },
    finally={  }
  )
  return(tscount_obj)
}
```

Function for ARIMA Models

```
# function for creating the arima model
# error catching is applied because sparse data means ARIMA may not always be possible
getARIMA_Obj <- function(timeseriesobject, seasonal) {
  require(forecast)
  arima_obj <- tryCatch(
    {
      arima_obj <- auto.arima(timeseriesobject,
                              approximation = TRUE, #FALSE was taking too long over 1'000s of tseries
                              seasonal      = seasonal,
                              max.order     = 12) #differencing should not be more than 12
    },
    error=function(cond) {
      #TimeSeries Modelling Error
      return("Error")
    },
    warning=function(cond) {
      # warningc
      return("Warning")
    },
    finally={  }
  )
```

```
  return(arima_obj)
}
```

In order to use the auto.ARIMA and tscount methods, we need a function for converting data to a "R time Series" object. This function must also pad the data with zeros for missing months, because the above data excludes months with zero sales.

```
PART_NUMBER_getts <- NULL

getts <- function(part_number, all_series, lag=0, months_list,
                  all_to_same_period = FALSE, NAtoZero = FALSE,
                  parts_specs = parts_frequent, getEvents = FALSE){

  require(dplyr)
  require(lubridate)

  # for testing its useful to know which part number was last anlaysed...
  PART_NUMBER_getts <<- part_number

  #select the relevant time series
  timeseries_df <- all_series  %>% filter(PART_NUMBER == part_number)
  parts_spec    <- parts_specs %>% filter(PART_NUMBER == part_number)

  #get min and max month

  if(nrow(parts_spec)>0){
    minmonth <- parts_spec$FIRST_SALE[1]
    maxmonth <- parts_spec$LAST_SALE[1]
  }else{
    minmonth <- min(timeseries_df$INVOICE_MONTH)
    maxmonth <- max(timeseries_df$INVOICE_MONTH)
  }

  #pad with months where sales are undefined (i.e. =0), because source sequence
  #excludes months with zero sales.
  timeseries_df <- months_list %>%
                    left_join(timeseries_df, by=("INVOICE_MONTH")) %>%
                      filter(INVOICE_MONTH >= minmonth & INVOICE_MONTH <= maxmonth)


  #if getting events, select events, else select sales qty
  if(getEvents){
    timeseries_df <- timeseries_df %>%
                      select(INVOICE_MONTH, SALES_EVENTS) %>%
                        rename(Quantity = SALES_EVENTS)
  }else{
    timeseries_df <- timeseries_df %>%
                      select(INVOICE_MONTH, SALES_QTY) %>%
                        rename(Quantity = SALES_QTY)
  }

  if(NAtoZero){
    #all values zero if currently NA
    timeseries_df$Quantity[is.na(timeseries_df$Quantity)] <- 0
```

```
  }else{
    #only those dates after minmonth and before maxmonth should be zero if currently NA
    row_nums_natozero <- which(is.na(timeseries_df$Quantity)
                               & timeseries_df$INVOICE_MONTH>=minmonth
                               & timeseries_df$INVOICE_MONTH<=maxmonth)
    timeseries_df$Quantity[row_nums_natozero] <- 0
  }

  #create time series object
  if(all_to_same_period){
    #na.action = na.omit
    timeseries_ts <- ts(timeseries_df$Quantity,
                        start= c(lubridate:: year(months_list[[1]][1]),
                                 lubridate::month(months_list[[1]][1])),
                        end  = c(lubridate:: year(months_list[[1]][nrow(months_list)]),
                                 lubridate::month(months_list[[1]][nrow(months_list)])),
                        frequency = 12)
  }else{
    timeseries_ts <- ts(timeseries_df$Quantity,
                        start= c(lubridate:: year(minmonth),
                                 lubridate::month(minmonth)),
                        end  = c(lubridate:: year(maxmonth),
                                 lubridate::month(maxmonth)),
                        frequency = 12)
  }
  if(lag!=0){
    timeseries_ts <- stats::lag(timeseries_ts,k=lag)
  }
  return(timeseries_ts)
}
```

We also need an absolute method of comparing accuracy of models. This will be used to understand which series are less well modelled than others. Rsquared will suffice.

```
getRsquared <- function(fitted, actual){
 R2 <- 1 - (sum((actual-fitted )^2)/sum((actual-mean(actual))^2))
 return(R2)
}
```

Comparing tscount with auto.arima

```
compare_models <- function(part_number, R_R_PRICE, all_series, months_list, results){

  #useful in test...
  PART_NUMBER_counter <<- part_number

  require(lubridate)
  require(forecast)
  require(tscount)

  all_series          <- all_series %>% filter(PART_NUMBER == part_number)
```

```
final_date_in_series<- min(max(all_series$INVOICE_MONTH),
                           max(months_list$INVOICE_MONTH))
final_date_in_train <- final_date_in_series %m-% months(12)
fcast_from_date     <- final_date_in_train  %m+% months( 1)
fcast_to_date       <- fcast_from_date      %m+% months(11)

# if there is an error we will use the final 12mths in train as the forecast
fallback_from_date  <- final_date_in_train  %m-% months(11)

train_to_year       <- year( final_date_in_train)
train_to_month      <- month(final_date_in_train)

fallback_from_year  <- year( fallback_from_date)
fallback_from_month <- month(fallback_from_date)

fcast_from_year     <- year(fcast_from_date)
fcast_from_month    <- month(fcast_from_date)
fcast_to_year       <- year(fcast_to_date)
fcast_to_month      <- month(fcast_to_date)

WasErrorARIMA       <- FALSE
WasErrorTScount     <- FALSE

#get time series
model_ts        <- getts(part_number        = part_number,
                         all_series         = all_series,
                         months_list        = monthslist,
                         all_to_same_period = FALSE,
                         NAtoZero           = FALSE,
                         getEvents          = FALSE)

window_train    <- window(x         = model_ts,
                          end       = c(train_to_year, train_to_month),
                          frequency = 12)

# get correct forecast
window_fcast    <- window(x         = model_ts,
                          start     = c(fcast_from_year, fcast_from_month),
                          end       = c(fcast_to_year,   fcast_to_month),
                          frequency = 12)

fcast_correct   <- sum(window_fcast)

# get data to use if there is an error
window_fallback <- window(x         = model_ts,
                          start     = c(fallback_from_year, fallback_from_month),
                          end       = c(train_to_year,      train_to_month),
                          frequency = 12)

#get ARIMA model and forecast
model_ARIMA     <- getARIMA_Obj(window_train, seasonal=TRUE)
#if model build errored, just use mean of series
if(length(model_ARIMA)>1){
```

```
    fcast_ARIMA   <- forecast(model_ARIMA, h = 12)
    fcast_ARIMA   <- sum(fcast_ARIMA$mean)
  }else{
    fcast_ARIMA   <- mean(window_fallback)*12
    WasErrorARIMA <- TRUE
  }

  #get tscount() model and forecast
  model_TScount    <- getTScount_Obj(window_train)
  #if model build errored, just use mean of series
  if(length(model_TScount)>1){
    fcast_TScount  <- predict(model_TScount, n.ahead = 12)
    fcast_TScount  <- sum(fcast_TScount$pred)
  }else{
    fcast_TScount  <- mean(window_fallback)*12
    WasErrorTScount<- TRUE
  }

  #save results
  results <- list(part_number,
                  R_R_PRICE,
                  fcast_correct,

                  fcast_ARIMA,
                  fcast_ARIMA - fcast_correct, #error_ARIMA
                  (fcast_ARIMA - fcast_correct) * R_R_PRICE, #errorVal_ARIMA

                  fcast_TScount,
                  fcast_TScount - fcast_correct, #error_TScount
                  (fcast_TScount - fcast_correct) * R_R_PRICE, #errorVal_TScount

                  WasErrorARIMA,
                  WasErrorTScount)

  return(results)
}
```

Now we run the function over our 1,000 parts, comparing tscount models with ARIMA models for 12months forecasts. Takes 10mins.

```
# instantiate results table
comparison_results <- data_frame(PART_NUMBER        = as.character(),
                                 R_R_PRICE          = as.numeric(),
                                 fcast_correct      = as.numeric(),
                                 fcast_ARIMA        = as.numeric(),
                                 error_ARIMA        = as.numeric(),
                                 errorVal_ARIMA     = as.numeric(),
                                 fcast_TScount      = as.numeric(),
                                 error_TScount      = as.numeric(),
                                 errorVal_TScount   = as.numeric(),
                                 WasErrorARIMA      = as.logical(),
                                 WasErrorTScount    = as.logical())

# loop through part numbers, seeking comparison of models
```

```
pb <- txtProgressBar(min = 1, max = nrow(partnumbers_top_pc), style = 3)

for(i in 1:nrow(partnumbers_top_pc)){

  setTxtProgressBar(pb, i)

  # if there are enough months between first sale and last sale, proceed
  FirstSale    <- (partnumbers_top_pc %>% filter(PART_NUMBER ==
                                                 partnumbers_top_pc$PART_NUMBER[i]))$FIRST_SALE
  LastSale     <- (partnumbers_top_pc %>% filter(PART_NUMBER ==
                                                 partnumbers_top_pc$PART_NUMBER[i]))$LAST_SALE
  MthsInSeries <- interval(ymd(FirstSale),ymd(LastSale))
  MthsInSeries <- MthsInSeries %/% months(1)
  if(MthsInSeries > 24){

    # whats the next row number for the results?
    nextrow <- nrow(comparison_results)+1

    # get comparison
    comparison_results[nextrow,] <- compare_models(
                                    part_number   = partnumbers_top_pc$PART_NUMBER[i],
                                    R_R_PRICE     = partnumbers_top_pc$R_R_PRICE[i],
                                    all_series    = series_top_pc,
                                    months_list   = monthslist,
                                    results       = comparison_results
                                    )
  }
}
```

We'll need a function to judge the comparative performance of these models...

```
# summarise comparisons
getPerfCompared <- function(comparison_results,
                            ExcludeARIMAerrors= FALSE,
                            ExcludeTSerrors   = FALSE,
                            groupbyfreqtype   = TRUE){

  # filter by error types
  sumry <- comparison_results %>%
           left_join(parts_frequent %>% select(PART_NUMBER, FreqType), by="PART_NUMBER")

  if(ExcludeARIMAerrors == TRUE){
    sumry <- sumry %>% filter(WasErrorARIMA == FALSE)
  }
  if(ExcludeTSerrors == TRUE){
    sumry <- sumry %>% filter(WasErrorTScount == FALSE)
  }

  #apply groupby, if req'd
  if(groupbyfreqtype){
    sumry <- sumry %>% group_by(FreqType)
  }
  #now summarise
  sumry <- sumry %>%
```

```
            summarise(MeanQtyErrPropn_ARI   = percent(mean(  ifelse(fcast_correct==0, 1,
                                                                      error_ARIMA  / fcast_correct))),
                      MeanQtyErrPropn_TSC   = percent(mean(  ifelse(fcast_correct==0, 1,
                                                                      error_TScount/ fcast_correct))),
                      MedianQtyErrPropn_ARI = percent(median(ifelse(fcast_correct==0, 1,
                                                                      error_ARIMA  / fcast_correct))),
                      MedianQtyErrPropn_TSC = percent(median(ifelse(fcast_correct==0, 1,
                                                                      error_TScount/ fcast_correct))),
                      TotalValErr_ARI       = round(sum(abs(  errorVal_ARIMA)),0),
                      TotalValErr_TSC       = round(sum(abs(  errorVal_TScount)),0),
                      TotalValErrPropn_ARI  = percent(sum(abs(errorVal_ARIMA))/
                                                        sum(  fcast_correct*R_R_PRICE)),
                      TotalValErrPropn_TSC  = percent(sum(abs(errorVal_TScount))/
                                                        sum(fcast_correct*R_R_PRICE))
                      )
  return(sumry)
}
```

```
getPerfCompared(comparison_results,
                ExcludeARIMAerrors = TRUE,
                ExcludeTSerrors    = TRUE,
                groupbyfreqtype    = FALSE)
```

Where the models do not error then ARIMA outperforms TScount, but the similarity is striking. The scale of the business problem is also obvious, over £0.35m of error (either under or over ordering), representing over a third of the total purchasing is in error when using these methods.

Accuracy increases as data increases, hence accuracy is far better for NutsAndBolts than for Normal.

If either model encounters an error then it has to use the prior 12mths sales as their forecast for the next 12mths. This is the case for most 'Low' and 'Rarity' parts, it changes the performance substantially:

```
getPerfCompared(comparison_results,
                ExcludeARIMAerrors = FALSE,
                ExcludeTSerrors    = FALSE,
                groupbyfreqtype    = TRUE)
```

now lets get an overall score, inc all errors

```
getPerfCompared(comparison_results,
                ExcludeARIMAerrors = FALSE,
                ExcludeTSerrors    = FALSE,
                groupbyfreqtype    = FALSE)
```

We'll invest our efforts in modelling with ARIMA rather than tscount. Nevertheless, ARIMA is not very satisfying, over ordering quantities by a quarter on average. Although this is clearly skewed by some poor results.

Perhaps tscount would have been more effective if we had predictors for each part_number time series. Having preditors, or other features, would allow us to explore alternative models to ARIMA and tscount.

Perhaps the timeseries are 'features' of each other. In other words, we can conjecture that part_number time series are related, a movement in one timeseries last month may tell us about another timeseries next month. But out of thousands of time series, which are the best lead indicators? For that we need to build a correlation space with potential leads/lags between part numbers.

So, here are the forthcoming tasks:

- Forecasting using ARIMA – we will ultimately need such a model and forecast for every part at every month end, so as to other train models
- Feature Building for alternative models – Decomposition of the time series into trend and seasonal — Trend is the 12mth moving average, very useful in forecasting 12mth sales, this will be a key feature – Correlation space, the degree to which each part's time series correlates with all other parts — the hope being that we can cluster parts into families which tend to behave in similar ways

Feature and Forecast Building Using Decompose & ARIMA

Decomposing time series can give us new features for our models, eg trend values, seasonal values. ARIMA models can give us forecasts and features. The features are the fitted values and the ARIMA forecast itself.

We also need a function to decompose time series

```
library(forecast)

## function for time series decomposition to trend, seasonal etc.

get_stl <- function(timeseriesobject, s.window="periodic"){
  stl_obj <- tryCatch(
      {
        stl_obj <- stl(timeseriesobject,
                       s.window=s.window)
      },
        error   = function(cond) {
          #Decomposition Modelling Error
          return(NA)
        },
        warning = function(cond) {
          # Decomposition warning
          return(NA)
        },
        finally = {
        }
    )
  return(stl_obj)
}
```

and a format conversion function to help us with month ends. . . .

```
eomonth <- function(DateInDecimalFormat){
  # We will be dealing with dates in decimal format, due to the timeseries object, ts()
  # But we will often need month end dates in date format, as that is how accounting systems work
  # accounting systems are the sourc eof much of the data!
  # So, know that ts objects use the first date of a month to represent that month,
  # eg     2013.500 represents July 2013, but date_decimal(2013.500) = 2013-07-02
  # Worse: 2013.083 represents Feb 2013,  but date_decimal(2013.083) = 2013-01-31
  # This function finds the correct month end for a date in decimal format, as taken from a ts() object
  require(lubridate)

  # get in date format
  InDateFormat <- date_decimal(DateInDecimalFormat)

  # Add two days, so we escape the February problem and are always in the relevant month
```

```
  InDateFormat <- InDateFormat + days(2)

  # round up to NEXT month start
  InDateFormat <- ceiling_date(InDateFormat, "month")

  # subtract a day, so we are back at the correct month end
  InDateFormat <- InDateFormat - days(1)

  # return a date format, not Posix
  InDateFormat <- as.Date(InDateFormat)

  return(InDateFormat)
}
```

Main function to Decompose Timeseries and Build ARIMA models

```
#this is used in testing...
PART_NUMBER_scope        <- NA
date_map_test            <- NULL
dates_in_dateformat_test <- NULL
dates_in_decimal_test    <- NULL

# main function to convert sales to timeseries format
convertToTimeSeries <- function(selected_partnumbers,
                                selected_partnumbers_series,
                                months_list,
                                GetARIMA = TRUE){

  require(lubridate)
  require(dplyr)
  require(pbapply)
  require(parallel)

  selected_series    <- data.frame(PART_NUMBER   = character(),
                                   INVOICE_MONTH = numeric(),
                                   SALES_QTY     = numeric(),
                                   SEASONAL      = numeric(),
                                   TREND         = numeric(),
                                   FITTED        = numeric(),
                                   stringsAsFactors = FALSE)

  selected_series_ts <- list(PART_NUMBER = character(),
                             TIME_SERIES = list(),
                             ARIMA       = list())

  ##################################################################
  # First get all time series

  # Calculate the number of cores
  no_cores <- detectCores() - 1
  # Initiate cluster
```

```
 cl <- makeCluster(no_cores)

 ## Get all time series objects ##
 print("Getting Time series objects")
 timeseries_ts_all <- pbsapply(selected_partnumbers$PART_NUMBER, cl=cl, getts,
                               #'x' param from sapply is part_number.
                               # So first param of getts is 'x' by default
                               # thus is not listed separately here. Other params must be listed separ
                               # NB. getts must 'require' the packages it uses
                               all_series  = selected_partnumbers_series,
                               parts_specs = selected_partnumbers,
                               lag         = 0,
                               months_list = months_list,
                               NAtoZero    = FALSE,
                               all_to_same_period = FALSE)

 stopCluster(cl)
 cl <- makeCluster(no_cores)

 ## Then get all ARIMA objects ##
 if(GetARIMA){
   print("Getting ARIMAs")
   ARIMA_all <- pbsapply(timeseries_ts_all, cl=cl, getARIMA_Obj,
                         # 'x' param from sapply is a timeseries object.
                         # So first param of getARIMA_Obj is the time series by default
                         # thus is not listed separately here.
                         # NB. getARIMA_Obj must 'require' the packages it uses
                         seasonal = TRUE,
                         simplify = FALSE)
 }

 stopCluster(cl)
 cl <- makeCluster(no_cores)

 ## Get decomposition objects ##
 print("Getting decompositions")
 stl_all    <- pbsapply(timeseries_ts_all, cl=cl, get_stl,
                         # 'x' param from sapply is a timeseries object.
                         # So first param of get_stl is the time series by default,
                         # thus is not listed separately here. Other params must be listed separately
                         # NB. get_stl must 'require' the packages it uses
                         s.window = "periodic",
                         simplify = FALSE)

 stopCluster(cl)

 # Prepare for building table of results, not in parallel
 pb <- txtProgressBar(min = 1, max = nrow(selected_partnumbers), style = 3)
 cat(" \n")
 print("Building Table of Results")

################################################################
 a <- 1
```

```
b <- 1
# Build table of results
for(i in 1:nrow(selected_partnumbers)){

  setTxtProgressBar(pb, i)

  #get current part_number, time series, ARIMA and stl
  PART_NUMBER              <- selected_partnumbers[i,1][[1]]
  timeseries_ts            <- timeseries_ts_all[[i]]
  selected_partnumbers_stl <- stl_all[[i]]
  if(GetARIMA){TheARIMA    <- ARIMA_all[[i]]}

  #track progress (useful for testing)
  PART_NUMBER_scope        <<- PART_NUMBER

  #Build basic parts of the table:
  #print(paste0("set basics: ",PART_NUMBER))
  qty_months               <- length(timeseries_ts)
  a                        <- b
  b                        <- a + qty_months -1

  selected_series[a:b, 1]  <- rep(PART_NUMBER, qty_months)      # part number, [probably broadcasts,
  selected_series[a:b, 2]  <- as.numeric(time(timeseries_ts))   # dates (as decimal, eg 30-Jun-2015 =
  selected_series[a:b, 3]  <- as.numeric(timeseries_ts)         # actual sales qty

  #save seasonal and trend components to data frame
  #but only IF decomposition was possible
  #print(paste0("set trends: ",PART_NUMBER,". stl=",length(selected_partnumbers_stl)))
  if(length(selected_partnumbers_stl)>1){
    selected_series[a:b, 4] <- data.frame(selected_partnumbers_stl$time.series[,1])[,1]   # seasonal
    selected_series[a:b, 5] <- data.frame(selected_partnumbers_stl$time.series[,2])[,1]   # trend
  }else{
    selected_series[a:b, 4] <- NA   # seasonal
    selected_series[a:b, 5] <- NA   # trend
  }

  # Get and save ARIMA object for each part_number
  # print(paste0("set PArtNum and Timeseries to tsobj: ",PART_NUMBER))
  selected_series_ts$PART_NUMBER[[i]] <- PART_NUMBER
  selected_series_ts$TIME_SERIES[[i]] <- timeseries_ts

  #assume no ARIMA available until we try
  HasARIMA <- FALSE
  if(GetARIMA){
    if(length(TheARIMA)>1){HasARIMA <- TRUE}
  }
  #print(paste0("set ARIMA: ",PART_NUMBER))
  if(GetARIMA && HasARIMA){
    selected_series_ts$ARIMA[[i]]     <- TheARIMA
    selected_series_ts$R2[[i]]        <- getRsquared(selected_series_ts$ARIMA[[i]]$fitted,
                                                     selected_series_ts$ARIMA[[i]]$x)
    sales_fitted                      <- selected_series_ts$ARIMA[[i]]$fitted
  }else{
```

```
    selected_series_ts$ARIMA[[i]]     <- NA
    selected_series_ts$R2[[i]]        <- as.numeric(NA)
    sales_fitted                      <- rep(as.numeric(NA), qty_months)
  }

  #print(paste0("calc R2_byYr: ", PART_NUMBER))
  # R2 by year
  timeseries_df <- cbind.data.frame(date         = as.numeric(time(timeseries_ts)),
                                    year         = floor(as.numeric(time(timeseries_ts))),
                                    sales_fitted = sales_fitted,
                                    sales_actual = as.numeric(timeseries_ts))

  timeseries_df_year <- timeseries_df %>%
                          group_by(year)%>%
                            summarise(sales_fitted = sum(sales_fitted),
                                      sales_actual = sum(sales_actual))

  #print(paste0("set R2_byYr"))
  if(GetARIMA & HasARIMA){
    selected_series_ts$R2_ByYear[[i]]<- getRsquared(timeseries_df_year$sales_fitted,
                                                    timeseries_df_year$sales_actual)
    selected_series[a:b,6] <- as.numeric(selected_series_ts$ARIMA[[i]]$fitted)
  }else{
    selected_series_ts$R2_ByYear[[i]]<- NA
    selected_series[a:b,6] <- sales_fitted
  }

 #End loop around part numbers
 }

 print("Formatting Table of Results.")

#################################################################
# Set date before first sale or after last sale to NA
# prevents later functions from attempting to correlate to many zeros.
 selected_series <- selected_series %>%
                     inner_join(y = selected_partnumbers, by = "PART_NUMBER") %>%
                       mutate(INVOICE_MONTH_AsDate = eomonth(INVOICE_MONTH)) %>%
                       mutate(SALES_QTY = ifelse((INVOICE_MONTH_AsDate < FIRST_SALE)
                                                 |
                                                 (INVOICE_MONTH_AsDate > LAST_SALE), NA, SALES_QTY),
                              SEASONAL = ifelse((INVOICE_MONTH_AsDate < FIRST_SALE)
                                                 |
                                                 (INVOICE_MONTH_AsDate > LAST_SALE), NA, SEASONAL),
                                 TREND = ifelse((INVOICE_MONTH_AsDate < FIRST_SALE)
                                                 |
                                                 (INVOICE_MONTH_AsDate > LAST_SALE), NA, TREND)) %>%
                         select(PART_NUMBER, INVOICE_MONTH, INVOICE_MONTH_AsDate,
                                SALES_QTY,   SEASONAL,      TREND,        FITTED)

 #selected_series_test2<<-selected_series
 ################################################################
 ##Cbind the SALES_EVENTS into the data
```

```r
selected_series <- selected_series %>%
                    left_join(y  = selected_partnumbers_series,
                              by = c("PART_NUMBER"          = "PART_NUMBER",
                                     "INVOICE_MONTH_AsDate" = "INVOICE_MONTH"))%>%
                      select(PART_NUMBER,
                             INVOICE_MONTH,
                             INVOICE_MONTH_AsDate,
                             SALES_QTY = SALES_QTY.x,
                             SALES_EVENTS,
                             SEASONAL,
                             TREND,
                             FITTED)


#fix NAs in SALES_EVENTS, should only be NA if SALES_QTY also NA
setToZero <- which(is.na(selected_series$SALES_EVENTS) & selected_series$SALES_QTY ==0)
selected_series$SALES_EVENTS[setToZero] <- 0

setToNA   <- which(!is.na(selected_series$SALES_EVENTS) & is.na(selected_series$SALES_QTY))
selected_series$SALES_EVENTS[setToNA]   <- NA

#####################################################################
# Expand to all months for all parts. NA's for blank months
# Uses cross join to get all parts for all months, cross join is executed in dplyr using a hack...
# create fake columns for cross join
months_list_          <- months_list %>% mutate(fake = 1)
selected_partnumbers_ <- selected_partnumbers %>% mutate(fake = 1)

#Now do the joins
selected_series_alldates <-  months_list_ %>%
        rename(INVOICE_MONTH_AsDate = INVOICE_MONTH) %>%

          # this is the cross join, which creates a template of all possible months and parts
          full_join(y = selected_partnumbers_, by = "fake") %>% select(-fake) %>%

            #this is the left join to match the data with the template.
            left_join(y = selected_series, by = c("INVOICE_MONTH_AsDate", "PART_NUMBER")) %>%
              select(PART_NUMBER, INVOICE_MONTH, INVOICE_MONTH_AsDate,
                     SALES_QTY,   SALES_EVENTS,
                     SEASONAL,    TREND,         FITTED,
                     FIRST_SALE,  LAST_SALE,     ISGRP)

# The above process leaves INVOICE_MONTH_AsDate completed, but INVOICE_MONTH as NA if outside first/l
# So we need to ensure there is always a date in decimal format, as we use it later for a spread() in
# the date must be as a month start, not a month end, which matches how ts() objects represent dates

#list of months vs how ts() object represents them:
date_map <- ts(1,
               start= c(lubridate:: year(months_list[[1]][1]),
                        lubridate::month(months_list[[1]][1])),
               end  = c(lubridate:: year(months_list[[1]][nrow(months_list)]),
                        lubridate::month(months_list[[1]][nrow(months_list)])),
               frequency = 12)
```

```
date_map <- cbind.data.frame(INVOICE_MONTH_AsDate = as.Date(months_list$INVOICE_MONTH),
                             INVOICE_MONTH_AsDeci = round(as.numeric(time(date_map)),3))
date_map_test <<- date_map

dates_to_fill <- which(is.na(selected_series_alldates$INVOICE_MONTH))

dates_in_dateformat <- data.frame(INVOICE_MONTH_AsDate = selected_series_alldates$INVOICE_MONTH_AsDat
dates_in_dateformat_test <<- dates_in_dateformat

dates_in_decimal    <- unlist(dates_in_dateformat %>%
                              inner_join(date_map, by="INVOICE_MONTH_AsDate") %>%
                                select(INVOICE_MONTH_AsDeci))
dates_in_decimal_test <<- dates_in_decimal
#make the replacement
selected_series_alldates$INVOICE_MONTH[dates_to_fill] <- dates_in_decimal

#selected_series_alldates_test <<- selected_series_alldates
#####################################################################
# Flag periods where forecasting tools (Arima, ets etc) are relevant
# ie has there been at least some data immediately prior to the date of the row?
selected_series_alldates <- selected_series_alldates %>%
                            #how long ago was first sale?
                            mutate(YearsSinceFirstSale    = as.numeric(
                                                                difftime(INVOICE_MONTH_AsDate,
                                                                         FIRST_SALE,
                                                                         units="days"))/365) %>%
                            #how long ago was last sale?
                            mutate(YearsSinceLastSale     = as.numeric(
                                                                difftime(INVOICE_MONTH_AsDate,
                                                                         LAST_SALE,
                                                                         units="days"))/365) %>%
                            #if first sale not yet happend (-ve interval), or >1yr since last sale
                            #then ignore it, else we calculate the years of data
                            mutate(YearsOfDataForForecast = ifelse(YearsSinceFirstSale < 0
                                                                   |
                                                                   YearsSinceLastSale > 1,
                                                                   NA,
                                                                   YearsSinceFirstSale)) %>%
                              arrange(PART_NUMBER, INVOICE_MONTH_AsDate) %>%
                                select(-YearsSinceFirstSale, -YearsSinceLastSale)

#####################################################################
# Calculate periods relative to final month in data frame. eg most recent 12 months are 1..12
selected_series_alldates <- selected_series_alldates %>%

                            #row_number() OVER(PARTITION BY PART_NUMBER, ORDER BY INVOICE_MONTH_AsD
                            group_by(PART_NUMBER) %>%
                              mutate(Sequence = row_number(desc(INVOICE_MONTH_AsDate))) %>%

                                #get 12mth period, eg mths 1-12=1, mths 13-24=2, mths 25-36=3
                                mutate(AnnualPeriod = 1+((Sequence-1)%/%12)) %>%
                                  ungroup()
#return...
```

```
  return(list(selected_series_alldates, selected_series_ts))
}
```

Let's use the function...

```
convertToTimeSeries <- convertToTimeSeries(partnumbers_top_pc,
                                           series_top_pc,
                                           months_list = monthslist)

selected_series_withNAs    <- convertToTimeSeries[[1]]
selected_series_withNAs_ts <- convertToTimeSeries[[2]]
rm(convertToTimeSeries)

#We may find the de-seasonalised data useful
selected_series_withNAs <- selected_series_withNAs %>% mutate(SALES_QTY_Deseas = SALES_QTY-SEASONAL)
```

Get Forecasts

The function gave us ARIMA models, but not actual forecasts. We need to use the models to generate forecasts for each part at each month end. The forecasts will always be for the subsequent 12mth period.

There are numerous complexities to deal with, not least that a time series may not be valid at a given month end. Each part will have a date on which it was first sold and a date on which is was last sold. No point considering forecasts before first sale, some judgement must be used as we approach the final date as to wehteher that really is the final date or we are simply expecting another sale soon.

There is an alternative to consider. The current method of parts forecasting uses a simple weighted average: Simple Forecast = (3xLast Yrs Sales + 2xPrior Yr's Sales + 1xPrior Prior Yr's Sales) /6 There are then some common sense rules applied on whether to calculate such an average when there have been very few prior sales (<10).

The performance of any model must be compared with this simple weighted average. Therefore it also needs to be calculated for every part number at every month end for a 12month period.

```
#function for simple weighted average forecasts
simple_forecasts_test <- NULL
simple_forecasts_s <- NULL

get_simpleForecast<-function(series_df, ForecastFromMth){
  #calculates simple forecast (3x3yrsago+2x2yrsago+1xlastyr) / 6.

  #first we need to know which month we are forecasting from and whether
  #we have some years of data at that month
  RangeOfData <- series_df %>%
                 select(PART_NUMBER, FIRST_SALE) %>%
                   distinct() %>%
                    mutate(YearsRangeOfData = round(interval(FIRST_SALE,
                                                             ForecastFromMth)
                                                    / years(1)),
                           ForecastFromMth  = ForecastFromMth)

  # get data into format where we can apply this function, requires a 'spread' (aka pivot)
  # The spread is sales in Mths01to12 (yr1), Mths13to24 (yr2) and Mths25to36 (yr3),
  # relative to the present.
```

```
simple_forecasts_s <- series_df %>%
                        group_by(PART_NUMBER, AnnualPeriod) %>%
                          mutate(SalesInPeriod = sum(SALES_QTY, na.rm=TRUE)) %>%
                            distinct(PART_NUMBER, AnnualPeriod, SalesInPeriod) %>%
                              spread(key=AnnualPeriod, value=SalesInPeriod, fill = 0) %>%
                              # name 1,2,3 as SalesQtyMths01to12 etc.
                              # Can only assume yr1 is present.
                                mutate(SalesQtyMths01to12 = as.numeric(`1`)) %>%
                                  ungroup()

#Same for Sales Events too
simple_forecasts_e <- series_df %>%
                      group_by(PART_NUMBER, AnnualPeriod) %>%
                        mutate(EventsInPeriod = sum(SALES_EVENTS, na.rm=TRUE)) %>%
                          distinct(PART_NUMBER, AnnualPeriod, EventsInPeriod) %>%
                            spread(key=AnnualPeriod, value=EventsInPeriod, fill = 0) %>%
                            # name 1,2,3 as EventQtyMths01to12 etc.
                            # Can only assume yr1 is present.
                              mutate(EventQtyMths01to12 = as.numeric(`1`)) %>%
                                ungroup()

# name columns 2 and 3 for simplicity, but only if they exist (there may be less than 2yrs of data)
if("2" %in% colnames(simple_forecasts_s)){
  simple_forecasts_s <- simple_forecasts_s %>% mutate(SalesQtyMths13to24 = as.numeric(`2`))
  simple_forecasts_e <- simple_forecasts_e %>% mutate(EventQtyMths13to24 = as.numeric(`2`))
}else{
  simple_forecasts_s <- simple_forecasts_s %>% mutate(SalesQtyMths13to24 = NA)
  simple_forecasts_e <- simple_forecasts_e %>% mutate(EventQtyMths13to24 = NA)
}

if("3" %in% colnames(simple_forecasts_s)){
  simple_forecasts_s <- simple_forecasts_s %>% mutate(SalesQtyMths25to36 = as.numeric(`3`))
  simple_forecasts_e <- simple_forecasts_e %>% mutate(EventQtyMths25to36 = as.numeric(`3`))
}else{
  simple_forecasts_s <- simple_forecasts_s %>% mutate(SalesQtyMths25to36 = NA)
  simple_forecasts_e <- simple_forecasts_e %>% mutate(EventQtyMths25to36 = NA)
}

# print(paste0("Bringing together simple forecasts"))
# bring together sales qty and sales events
simple_forecasts <- simple_forecasts_s %>%
                      inner_join(y=simple_forecasts_e, by="PART_NUMBER") %>%
                        select(PART_NUMBER,
                               SalesQtyMths25to36, SalesQtyMths13to24, SalesQtyMths01to12,
                               EventQtyMths25to36, EventQtyMths13to24, EventQtyMths01to12)

simple_forecasts_test1 <<- simple_forecasts

# now we can apply those rules, (3x3yrsago+2x2yrsago+1xlastyr) / 6.
simple_forecasts <- simple_forecasts %>%
                      inner_join(y  = RangeOfData,
                                 by = "PART_NUMBER") %>%
                      mutate(SimpleForecast  = case_when(YearsRangeOfData == 1L ~
```

```
                                                    SalesQtyMths01to12,
                                                  YearsRangeOfData == 2L ~
                                                    round((2*SalesQtyMths01to12+
                                                           1*SalesQtyMths13to24)/3),
                                                  YearsRangeOfData >= 3L ~
                                                    round((3*SalesQtyMths01to12+
                                                           2*SalesQtyMths13to24+
                                                           1*SalesQtyMths25to36)/6),
                                                  TRUE ~ NA_real_)) %>%
                          select(PART_NUMBER,
                                 SalesQtyMths25to36, SalesQtyMths13to24, SalesQtyMths01to12,
                                 SimpleForecast,
                                 EventQtyMths25to36, EventQtyMths13to24, EventQtyMths01to12,
                                 ForecastFromMth, YearsRangeOfData)

  # print(paste0("Applying common sense limits"))
  #finally, special rules for what to do when sales in previous yr were zero.
  #we, override any forecast value and set to zero, these rules taken from the old forecast method
  setForecastToZero <- which(
                         (  simple_forecasts$SalesQtyMths01to12 == 0)
                         |
                         (  simple_forecasts$SalesQtyMths01to12 < 10
                          & simple_forecasts$SalesQtyMths13to24 == 0)
                           |
                         (  simple_forecasts$EventQtyMths01to12 <  3
                          & simple_forecasts$EventQtyMths13to24 == 0)
                         |
                         (  simple_forecasts$SalesQtyMths01to12 <= 5
                          & simple_forecasts$SalesQtyMths13to24 <= 2
                          & simple_forecasts$SalesQtyMths25to36 == 0)
                         |
                         (  simple_forecasts$EventQtyMths01to12 <= 1
                          & simple_forecasts$EventQtyMths13to24 <= 1
                          & simple_forecasts$EventQtyMths25to36 == 0)
                         )

  setForecastToZero <- unique(setForecastToZero)

  if(length(setForecastToZero)>0){
    simple_forecasts$SimpleForecast[setForecastToZero] <- 0
  }

  return(simple_forecasts)
}
```

```
# Get 12month Forecasts from a specific month (not from all months)
get_ArimaAndSimpleForecasts <- function(series_df, series_list, months_list){

  require(dplyr)
  require(tidyr)
  require(parallel)

  #get the month from which we will be forecasting
  ForecastFromMth  <- max(months_list$INVOICE_MONTH)
```

```
# Calculate the number of cores
no_cores <- detectCores() - 1
# Initiate cluster
cl <- makeCluster(no_cores)

#########################################################################
#ARIMA forecasts
print(paste0("Getting ARIMA forecasts"))
ARIMAobjects_list <- series_list[3]$ARIMA
ARIMA_forecasts <- pbsapply(ARIMAobjects_list, cl=cl, function(x){

                    require(forecast)
                    require(lubridate)
                    if(length(x)==1){
                      return(NA)
                    }else{
                      #Get max date in time series
                      #uses same code as eomonth() function
                      MaxDateInARIMA <-as.Date(
                        ceiling_date(
                          date_decimal(max(time(x$x))) + days(2), "month") - days(1))

                      # how many months from the end of time series until end of months list?
                      # NB, it is possible that ForecastFromMth < MaxDateInARIMA, so check first
                      if(ForecastFromMth>MaxDateInARIMA){
                        IntervalMths <- interval(ymd(MaxDateInARIMA),ymd(ForecastFromMth))
                        IntervalMths <- IntervalMths %/% months(1)
                      }else{
                        IntervalMths <- 0
                      }

                      # get the arima forecasts
                      Forecasts <- forecast(x, h=12+IntervalMths)

                      # get the sum of the last 12 mean forecasts (ie one year's forecast)
                      len <- length(Forecasts$mean)
                      ForecastMean     <- sum(Forecasts$mean[(len-11):len])

                      #ForecastRange80  <- sum(Forecasts$upper[(len-11):len,1] -
                      #                       Forecasts$lower[(len-11):len,1])
                      return(ForecastMean)
                    }}
                    ,simplify = TRUE)

#Close the cluster, return resources
stopCluster(cl)

#########################################################################
#ARIMA sigma2
print(paste0("Getting ARIMA model sigma2"))
ARIMA_sigma2 <- sapply(ARIMAobjects_list, function(x){
                            if(length(x)==1){
                              return(NA)
```

```
                                }else{
                                  return(x$sigma2)
                                }}
                                ,simplify = TRUE)

  # form ARIMA forecasts and sigma2 into a data frame
  detail_forecasts <- data.frame( PART_NUMBER   = as.character(series_list[1]$PART_NUMBER),
                                  ARIMAforecast = as.numeric(unlist(ARIMA_forecasts)),
                                  ARIMAsigma2   = as.numeric(unlist(ARIMA_sigma2)),
                                  stringsAsFactors = FALSE)

  # if ARIMAsigma2 is NaN, then ARIMA failed, set both to
  setToNA <- which(is.nan(detail_forecasts$ARIMAsigma2))
  detail_forecasts$ARIMAsigma2[setToNA]   <- NA
  detail_forecasts$ARIMAforecast[setToNA] <- NA

  ########################################################################
  #Simple forecasts
  print(paste0("Getting simple forecasts"))

  simple_forecasts <- get_simpleForecast(series_df       = series_df,
                                         ForecastFromMth = ForecastFromMth)

  #unify simple and ARIMA forecasts
  #print(paste0("final step"))
  return_forecasts <- simple_forecasts %>%
                      left_join(y=detail_forecasts, by="PART_NUMBER")
  #print(paste0("return"))
  return(return_forecasts)
}
```

We use this function to build the forecasts much later. Meanwhile, we need to explore Cross Correlations for feature building, with the hope that parts sales sales cluster into families of similar trends.

Feature Building: Cross Correlations

When analysing the relationship between time series we need to consider the 'cross correlation'. This means we allow for the possibility that the time series are correlated but are time shifted (lagged) with respect to each other. For example, plough parts may have a similar demand profile to combine parts, but follow 2 months later. The CCF function in R allows us to examine how the correlation improves as we lag one series with respect to the other, then find the lag which gives the highest correlation.

We set lag.max= 6, because most have summer peaks so will happily correlate at lag=12, but this is not useful information. It is possible that lags between parts can be greater than 11mths. This occurs when one part is being replaced by another. althogh we would only expect that to happen within a group, ie where parts share the first five digits of their part number. Furthermore, this is rare compared with the number of parts which share a summer peak (lag=12)

The optimal lag may be spurious, all we have is the p score to help discriminate. The below code chunk finds the optimal lags using CCF, whilst also recording the correlation if no lag is applied.

Note, if there are 1,000 parts to consider, then there will be 1,000 squared (1,000,000) pair-wise combinations to analyse. To do this for the raw sales data, trends and seasonal data means 3,000,000 combinations. The below implementation is vectorised and conducted in parallel to accelerate it. Takes 10mins.

```r
## Cross Correlations
library(tidyr)
library(pbapply)

getCrossCorrels <- function(series, component, cl, lag.max=6){

  #extract raw sales, trend or seasonal component (each is a column in the series)
  matrix <- series %>%
              select_("PART_NUMBER",
                      "INVOICE_MONTH",
                      component) %>%
                spread_(key   = "PART_NUMBER",
                        value = component,
                        fill  = NA)

  #ignore the invoice month column
  matrix<-matrix[,-1]

  # Calculate all combinations of cross correlations, return the best acf and its associated lag
  # the lag is the amount by which x must be lagged to best correlate with y

  # This has been vectorised into two nested sapply functions because ccf() is not vectorised
  # (unlike rcorr())
  # this section of the code is what uses the 'cl' cluster for parallel processing.
  # Also, pbsapply() is just sapply with a progress bar

  crosscorrel <- pbsapply(1:ncol(matrix), cl = cl, function(x) {
                    sapply(1:ncol(matrix), function(z) {
                          ccf_tmp <- ccf(x         = matrix[ , x],
                                         y         = matrix[ , z],
                                         type      = "correlation",
                                         #not covariance, need standardised index of correl.
                                         plot      = FALSE,
                                         na.action = na.pass,
                                         lag.max   = lag.max)

                          #did it work, or do we have NA?
                          NAtest <- ccf_tmp$acf[which(ccf_tmp$lag==0)]
                          if(is.nan(NAtest) | is.na(NAtest)){
                            # Heck, we got NA, so return NA's
                            returnval <- c(0,0,0,0,0)
                            #note, 0 not NA. Useful in spread() later. er 0 would broadcast.
                          }else{
                            #It worked! Get the return values:
                            returnval <- c(

                             #1: cross correlation at lag = 0
                             ccf_tmp$acf[which(ccf_tmp$lag==0)],

                             #2: p value at lag=0
                             2 * (1 - pnorm(abs(ccf_tmp$acf[which(ccf_tmp$lag==0)]),
                                            mean = 0,
                                            sd   = 1/sqrt(ccf_tmp$n.used))),
```

```
                        #3: max cross correlation
                        max(ccf_tmp$acf),

                        #4: max cross correlation lag
                        ccf_tmp$lag[which.max(abs(ccf_tmp$acf))],

                        #5: max cross correlation's p value
                          2 * (1 - pnorm(abs(max(ccf_tmp$acf)),
                                         mean = 0,
                                         sd   = 1/sqrt(ccf_tmp$n.used)))
                       )
                     }#end if..else
                     return(returnval)
                     })
                })
#crosscorrel_test <<- crosscorrel

#Reformat the data into a tidy format, one table for lags, another for acf values
#whereas the data outputted by the above function gave five rows per part number,
#making it awkward to read.
#for this we define a function within this function...
extractFromMatrix <- function(crosscorrelmatrix, partsmatrix, item_name, item_sequence){

  crosscorrel_item            <- data.frame(crosscorrelmatrix[seq(
                                            from = item_sequence,
                                            to   = nrow(crosscorrelmatrix),
                                            by   = 5),])
  colnames(crosscorrel_item)  <- colnames(partsmatrix)
  row.names(crosscorrel_item) <- colnames(partsmatrix)
  crosscorrel_item            <- tibble::rownames_to_column(crosscorrel_item, "PART_NUMBER.x")
  crosscorrel_item_gathered   <- gather_(data        = crosscorrel_item,
                                         key_col     = "PART_NUMBER.y",
                                         value_col   = item_name  ,
                                         gather_cols = colnames(crosscorrel_item)[-1])
  return(crosscorrel_item_gathered)
}

#extract the cross correls with no lag
ccor_no_lag        <- extractFromMatrix(crosscorrel, matrix, "ACF_NO_LAG", 1) %>%
                      rename(ccor_no_lag = ACF_NO_LAG)

#extract the p value of acfs with no lag
ccor_no_lag_pval   <- extractFromMatrix(crosscorrel, matrix, "ACF_NO_LAG_PVAL", 2) %>%
                      rename(ccor_no_lag_pval = ACF_NO_LAG_PVAL)

#extract the max acf, given a lag
ccor_with_lag      <- extractFromMatrix(crosscorrel, matrix, "ACF_WITH_LAG", 3) %>%
                      rename(ccor_with_lag = ACF_WITH_LAG)

#extract the lag at optimal acf
lag                <- extractFromMatrix(crosscorrel, matrix, "LAG", 4) %>%
                      rename(lag = LAG)
```

```
  #extract the p value of max acf, given a lag
  ccor_with_lag_pval <- extractFromMatrix(crosscorrel, matrix, "ACF_WITH_LAG_PVAL", 5) %>%
                          rename(ccor_with_lag_pval = ACF_WITH_LAG_PVAL)

  #Join data into single table, filter
  crosscorrel_gathered <- ccor_no_lag %>%
                            inner_join(y =ccor_no_lag_pval,
                                       by=c("PART_NUMBER.x", "PART_NUMBER.y")) %>%
                            inner_join(y =ccor_with_lag,
                                       by=c("PART_NUMBER.x", "PART_NUMBER.y")) %>%
                            inner_join(y =lag,
                                       by=c("PART_NUMBER.x", "PART_NUMBER.y")) %>%
                            inner_join(y =ccor_with_lag_pval,
                                       by=c("PART_NUMBER.x", "PART_NUMBER.y")) %>%
                               filter(PART_NUMBER.x!=PART_NUMBER.y
                                       &
                                       PART_NUMBER.y!="INVOICE_MONTH") %>%
                                select(PART_NUMBER.x, PART_NUMBER.y,
                                       ccor_no_lag, ccor_no_lag_pval, ccor_with_lag,
                                       lag, ccor_with_lag_pval) %>%
                                  arrange(PART_NUMBER.x, PART_NUMBER.y)
  return(crosscorrel_gathered)
}
```

Let's use the above function to find the cross correlations between pairs of time series. We'll do this over the full time series available to us and explore the possibilities. We can do the cross correlation for four components (features) of the data: sales quantities, log(sales quantities+1), trends and seasonal movements.

In the final analysis we will need correlations re-calculated at every month end, but the same function can be used.

```
# The following process is processor intensive, best done in parallel
# If not done in parallel then this can take 60mins! In parallel it takes 10mins
library(parallel)
# Calculate the number of cores
no_cores <- detectCores() - 1
# Initiate cluster
cl <- makeCluster(no_cores)

#The log data is simple the sales quantities logged+1, we derive that first:
#lets get cross correls for log of Sales_Qty, we already know its a poisson
selected_series_withNAs_log <- selected_series_withNAs
selected_series_withNAs_log$SALES_QTY <- log(selected_series_withNAs_log$SALES_QTY+1)
selected_series_withNAs_log$TREND     <- log(selected_series_withNAs_log$TREND+1)

#Now calculate the cross correls, default lag.max to 6 else we simply replicate annual seasonality
crosscorrel_rawdata     <- getCrossCorrels(selected_series_withNAs,
                                           component = "SALES_QTY", cl)
crosscorrel_rawdata_log <- getCrossCorrels(selected_series_withNAs_log,
                                           component = "SALES_QTY", cl)
crosscorrel_rawdata_ds  <- getCrossCorrels(selected_series_withNAs,
                                           component = "SALES_QTY_Deseas", cl)

#max lag for Trend (12mth moving averge) must be higher, at 12
crosscorrel_trends      <- getCrossCorrels(selected_series_withNAs,
```

```
                                          component = "TREND", cl, lag.max=12)
crosscorrel_trends_log  <- getCrossCorrels(selected_series_withNAs_log,
                                          component = "TREND", cl, lag.max=12)
crosscorrel_seasonal    <- getCrossCorrels(selected_series_withNAs,
                                          component = "SEASONAL", cl)

#Close the cluster, return resources
stopCluster(cl)
```

These tables all contain duplicate pairs, corr(A vs B) and corr(B vs A), which are the same value. When seeking clusters we don't want corr(A vs B) and corr(B vs A) in the same cluster, its not new information, just duplication.

They should have the same correlation but opposing signs. We want one column to be predictors, the other column targets. So will de-dupe by first selecting the -ve of the two. If the optimal lag was 0 then this is not sufficient and we need other dedupe logic. See below function.

Why the -ve? ccf works like this: ccf(x-variable name, y-variable name), We want to find an 'x' which leads (comes before) 'y', so we can use x as a predictor to y. ccf() returns a -ve lag when x leads y, so these -ve lags are what we filter for.

```
#
getuniquepairs <- function(crosscorrel_gathered){

  #remove +ve lags
  crosscorrel_gathered <- crosscorrel_gathered %>% filter(lag <= 0)

  #get list of uniques, even if 0 lag
  uniquepairs <- data.frame(unique(t(apply(crosscorrel_gathered[,c(1,2)], 1, sort))),
                            stringsAsFactors = F)
  colnames(uniquepairs)<-c("PART_NUMBER.x","PART_NUMBER.y")

  # use join to filter doum to thos eon the list of uniques
  crosscorrel_gathered_unq <- crosscorrel_gathered %>%
                              inner_join(y=uniquepairs, by=c("PART_NUMBER.x","PART_NUMBER.y"))

  return(crosscorrel_gathered_unq)
}
```

```
crosscorrel_rawdata_unq     <- getuniquepairs(crosscorrel_rawdata)
crosscorrel_rawdata_log_unq <- getuniquepairs(crosscorrel_rawdata_log)
crosscorrel_rawdata_ds_unq  <- getuniquepairs(crosscorrel_rawdata_ds)
crosscorrel_trends_unq      <- getuniquepairs(crosscorrel_trends)
crosscorrel_trends_log_unq  <- getuniquepairs(crosscorrel_trends_log)
crosscorrel_seasonal_unq    <- getuniquepairs(crosscorrel_seasonal)
```

```
#let's see an example...
head(crosscorrel_trends_unq)
```

Remember, the lower the p value, the better. Notice higher ACF's are associated with lower P values. This is to be expected.

Let's summarise the performance of the correlations. Do Trends, Log(Trends), Sales Quantities, Log(Sales Quantities) or Seasonal values correlate best?

```
a<-  rbind(
       # Summarise correlation of Sales Quantities
```

```
      crosscorrel_rawdata_unq %>% summarise(
                                    mean_corr_no_lag   = mean(abs(ccor_no_lag),       na.rm = T),
                                    mean_pval_no_lag   = mean(abs(ccor_no_lag_pval),  na.rm = T),
                                    mean_corr_with_lag = mean(abs(ccor_with_lag),     na.rm = T),
                                    mean_pval_with_lag = mean(abs(ccor_with_lag_pval),na.rm = T)
                                    )
      ,# Summarise correlation of Log(Sales Quantities+1)
      crosscorrel_rawdata_log_unq %>% summarise(
                                    mean_corr_no_lag   = mean(abs(ccor_no_lag),       na.rm = T),
                                    mean_pval_no_lag   = mean(abs(ccor_no_lag_pval),  na.rm = T),
                                    mean_corr_with_lag = mean(abs(ccor_with_lag),     na.rm = T),
                                    mean_pval_with_lag = mean(abs(ccor_with_lag_pval),na.rm = T)
                                    )
      ,# Summarise correlation of Sales Quantities Deseasonalised
      crosscorrel_rawdata_ds_unq %>% summarise(
                                    mean_corr_no_lag   = mean(abs(ccor_no_lag),       na.rm = T),
                                    mean_pval_no_lag   = mean(abs(ccor_no_lag_pval),  na.rm = T),
                                    mean_corr_with_lag = mean(abs(ccor_with_lag),     na.rm = T),
                                    mean_pval_with_lag = mean(abs(ccor_with_lag_pval),na.rm = T)
                                    )
      ,# Summarise correlation of Trends
      crosscorrel_trends_unq  %>% summarise(
                                    mean_corr_no_lag   = mean(abs(ccor_no_lag),       na.rm = T),
                                    mean_pval_no_lag   = mean(abs(ccor_no_lag_pval),  na.rm = T),
                                    mean_corr_with_lag = mean(abs(ccor_with_lag),     na.rm = T),
                                    mean_pval_with_lag = mean(abs(ccor_with_lag_pval),na.rm = T)
                                    )
      ,# Summarise correlation of log(Trends+1)
      crosscorrel_trends_log_unq  %>% summarise(
                                    mean_corr_no_lag   = mean(abs(ccor_no_lag),       na.rm = T),
                                    mean_pval_no_lag   = mean(abs(ccor_no_lag_pval),  na.rm = T),
                                    mean_corr_with_lag = mean(abs(ccor_with_lag),     na.rm = T),
                                    mean_pval_with_lag = mean(abs(ccor_with_lag_pval),na.rm = T)
                                    )
      ,# Summarise correlation of Seasonal values
      crosscorrel_seasonal_unq%>% summarise(
                                    mean_corr_no_lag   = mean(abs(ccor_no_lag),       na.rm = T),
                                    mean_pval_no_lag   = mean(abs(ccor_no_lag_pval),  na.rm = T),
                                    mean_corr_with_lag = mean(abs(ccor_with_lag),     na.rm = T),
                                    mean_pval_with_lag = mean(abs(ccor_with_lag_pval),na.rm = T)
                                    )
)
#Let's view the summary:
row.names(a) <- c("sales qty", "log(sales qty+1)",
                  "sales qty deseas","trends", "log(trends+1)", "seasonal")
kable(a)
```

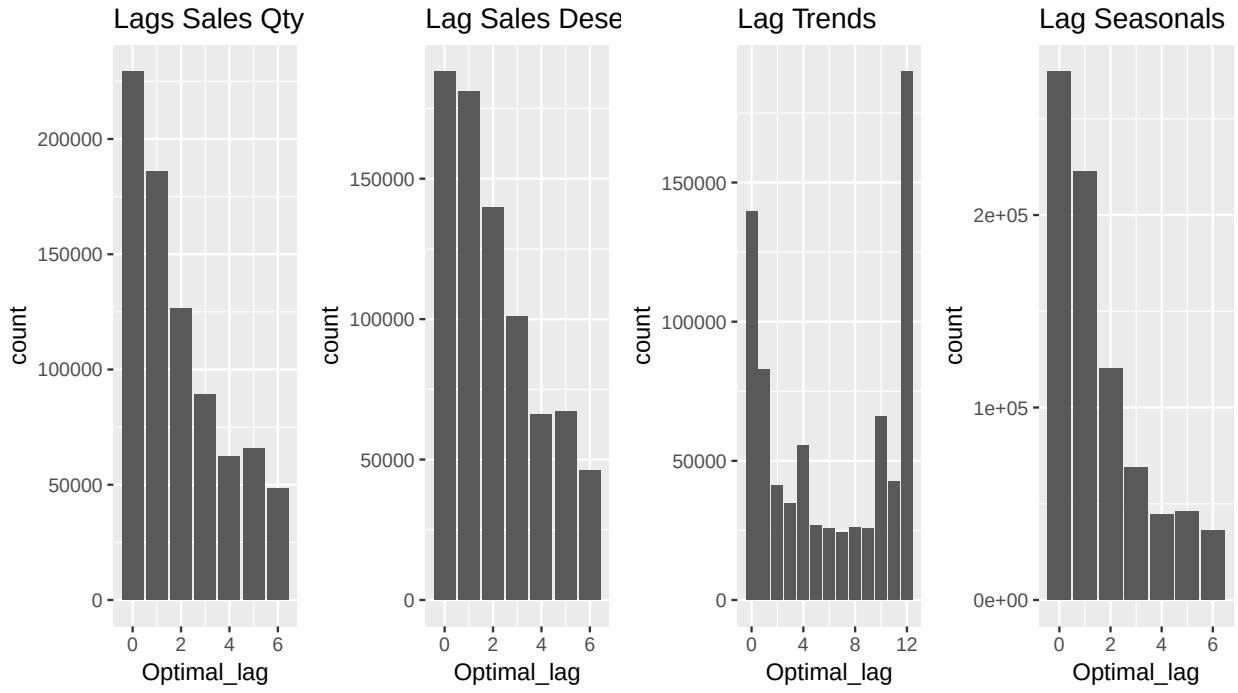
The above stats show that the cross correlation function can substantially improve relationships if a lag is applied. The questions is wehther the lag is reasonable or spurious. There are good reasons to expect lags between parts but we cannot afford the time of the thousands of parts to be certain the lags are not spurious.

Unsurprisingly, we can get the best correlations for the seasonal component when they are 'lagged' into alignment with each other, but the seasonal component is the least useful. We already know most part sales peak annually, that's why we do a seasonal order.

We are most interested in correlation of trends. Thankfully, taking the log of the trend has no effect, we can work with the trend data itself.

Let's see if there are any patterns to the optimal lags chosen in the ccf function:

```
library(ggplot2)
dat <- crosscorrel_rawdata_unq %>% group_by(Optimal_lag=abs(lag)) %>% summarise(count=n())
p <- ggplot(data=dat, aes(x=Optimal_lag, y=count)) +
     geom_bar(stat="identity")+
     ggtitle("Lags Sales Qty")

dat <- crosscorrel_rawdata_ds_unq %>% group_by(Optimal_lag=abs(lag)) %>% summarise(count=n())
q <- ggplot(data=dat, aes(x=Optimal_lag, y=count)) +
     geom_bar(stat="identity")+
     ggtitle("Lag Sales Deseas")

dat <- crosscorrel_trends_unq %>% group_by(Optimal_lag=abs(lag)) %>% summarise(count=n())
r <- ggplot(data=dat, aes(x=Optimal_lag, y=count)) +
     geom_bar(stat="identity") +
     ggtitle("Lag Trends")

dat <- crosscorrel_seasonal_unq %>% group_by(Optimal_lag=abs(lag)) %>% summarise(count=n())
s <- ggplot(data=dat, aes(x=Optimal_lag, y=count)) +
     geom_bar(stat="identity") +
     ggtitle("Lag Seasonals")

require(gridExtra)
grid.arrange(p, q, r, s, ncol=4)
```



The sales qty and seasonals show little more than the profile of the season itself, most peak together in august, but some are grass harvest and peak as early as April, some are maize harvest and peak as late as October (April-October = 6mths).

Cross Correlation Examples

Let's build some more intuition and view some examples of our cross correlations, to ensure highly correlated timeseries do look highly correlated, and vice versa.

```
# find highly correlated pairs, one for each lag, so we can plot them

candidate <- crosscorrel_trends_unq %>%
              #only part numbers with substantial sales
              inner_join(selected_partnumbers %>%
                           select(PART_NUMBER, TOTAL_SALES_EVENTS, TOTAL_QTY_SOLD),
                                  by=c("PART_NUMBER.x" = "PART_NUMBER"))%>%
              inner_join(selected_partnumbers %>%
                           select(PART_NUMBER, TOTAL_SALES_EVENTS, TOTAL_QTY_SOLD),
                                  by=c("PART_NUMBER.y" = "PART_NUMBER"))%>%
              filter(TOTAL_SALES_EVENTS.x >  50 & TOTAL_SALES_EVENTS.y > 50 &
                     TOTAL_QTY_SOLD.x     > 100 & TOTAL_QTY_SOLD.y    > 100) %>%
                filter(ccor_with_lag_pval < 0.05) %>%
                  filter(ccor_with_lag < 0.99 #don't want them exactly same, these are sub parts
                         &
                         ccor_with_lag > 0.95
                         &
                         substr(PART_NUMBER.x,1,8) != substr(PART_NUMBER.y,1,8)) %>%
                          #don't want same group
                    group_by(lag) %>%
                      top_n(5, ccor_with_lag) %>%
                        arrange(desc(ccor_with_lag))
```

```
head(candidate)
```

Let's take an example of two time series and plot their sales and trends against each other

```
CL2376142 <- selected_series_withNAs %>%
                   filter(PART_NUMBER == "CL-0002376142") %>%
                     select(x=INVOICE_MONTH,
                            CL2376142_SalesQty=SALES_QTY,
                            CL2376142_Trend   =TREND)

CLUKHT1023 <- selected_series_withNAs %>%
                   filter(PART_NUMBER =="CL-UKHT1023")%>%
                     select(x=INVOICE_MONTH,
                            CLUKHT1023_SalesQty=SALES_QTY,
                            CLUKHT1023_Trend   =TREND)

Crossed <- CLUKHT1023 %>%
            inner_join(CL2376142, by="x") #x is invoice month

s <- ggplot(data=Crossed,
            aes(x=CL2376142_SalesQty , y=CLUKHT1023_SalesQty )) +
     geom_line(stat="identity") +
     geom_smooth(method = "loess") +
     ggtitle("CL2376142 vs CLUKHT1023 (Sales Qty)")

t <- ggplot(data=Crossed,
            aes(x=CL2376142_Trend , y=CLUKHT1023_Trend )) +
```

```
    geom_line(stat="identity") +
    geom_smooth(method = "loess") +
    ggtitle("CL2376142 vs CLUKHT1023 (Trend)")

grid.arrange(s, t, ncol=2)
```

```
## Warning: Removed 45 rows containing non-finite values (stat_smooth).
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at -0.105
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 1.105
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 1
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used
## at -0.105
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius
## 1.105
```

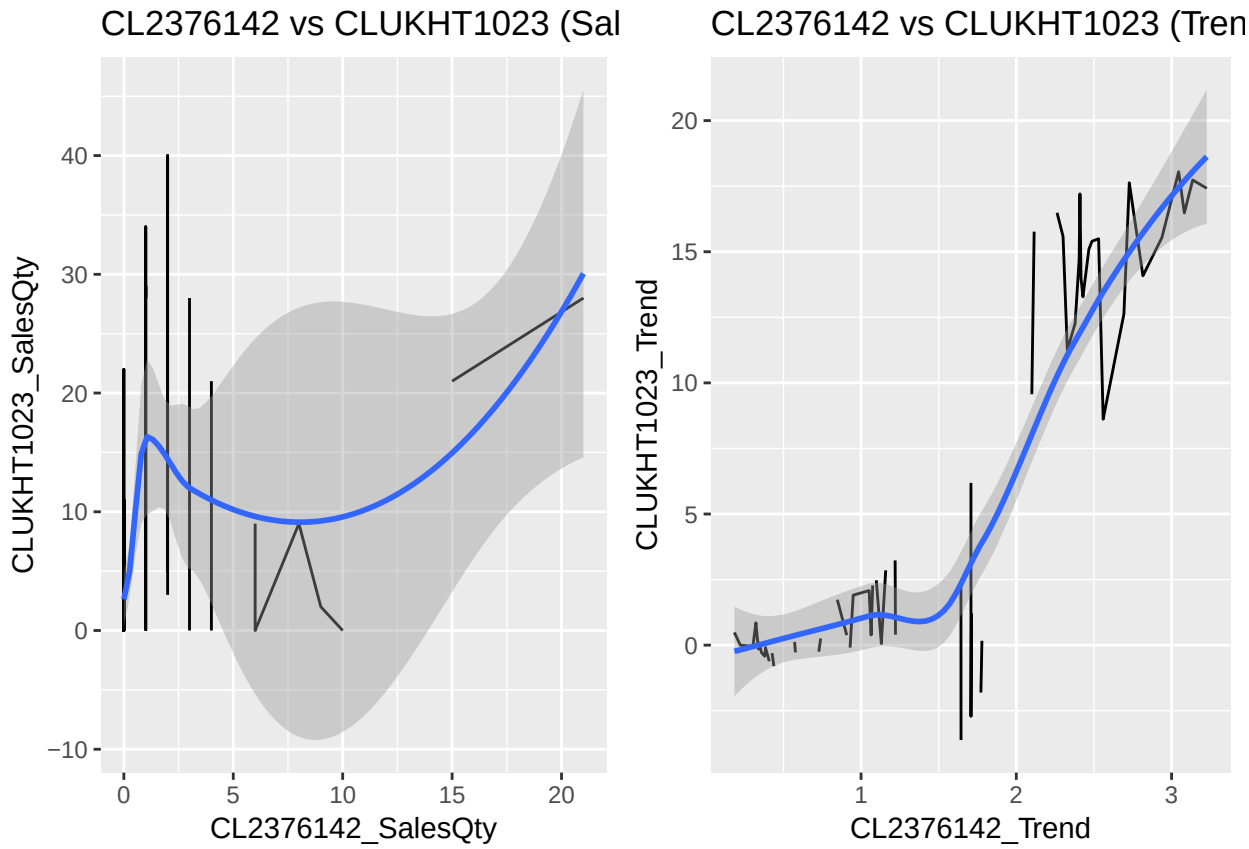
```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal
## condition number 0
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : There are other
## near singularities as well. 1
```

```
## Warning: Removed 28 rows containing missing values (geom_path).
```

```
## Warning: Removed 45 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```



Not entirely what we expected. We expected a fairly straight loess lines which would mean high correlation. There is clear +ve correlation, more sales in one means more sale sin the other, the trend correlated fairly well, but the SalesQty is a messy relationship.

For better or for worse we have our data, so how do these parts relate to each other in the wider solution space, not simply pairwise? Do they cluster?

Transfer Models & Time Series Regression

If x leads y with high correlation and a known lead time, then we can use x to predict y in a simple linear regression model. Let's find an example and give it a try.

```
#find example

ts_regr_candidates <- crosscorrel_trends_unq %>%
                        #only part numbers with substantial sales
                        inner_join(selected_partnumbers %>%
                                     select(PART_NUMBER, TOTAL_SALES_EVENTS, TOTAL_QTY_SOLD),
                                           by=c("PART_NUMBER.x" = "PART_NUMBER"))%>%
                        inner_join(selected_partnumbers %>%
                                     select(PART_NUMBER, TOTAL_SALES_EVENTS, TOTAL_QTY_SOLD),
                                           by=c("PART_NUMBER.y" = "PART_NUMBER"))%>%
                        filter(TOTAL_SALES_EVENTS.x >   50 & TOTAL_SALES_EVENTS.y  > 50  &
                               TOTAL_QTY_SOLD.x     >   50 & TOTAL_QTY_SOLD.y     > 50  &
                               ccor_no_lag          >= 0.70 & ccor_with_lag_pval   < 0.10 &
                               ccor_no_lag != ccor_with_lag & lag <= -10) %>%
```

```
                        filter(substr(PART_NUMBER.x,1,8) != substr(PART_NUMBER.y,1,8)) %>%
                                #don't want same group
                            mutate(improvement = ccor_with_lag - ccor_no_lag) %>%
                              arrange(desc(improvement))
```

```
head(ts_regr_candidates)
```

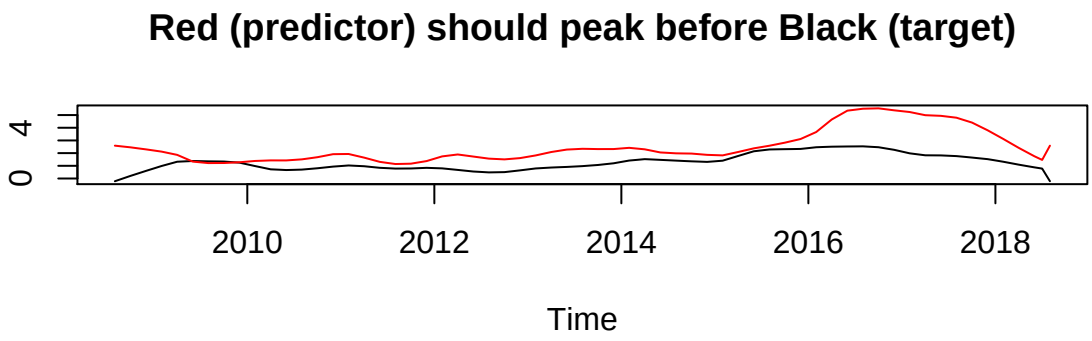
Let's plot two series against each other and see if the lag makes sense. Black is the target, red is the predictor. In order to use red to predict black we expect to see red peaking before black (red leads black)

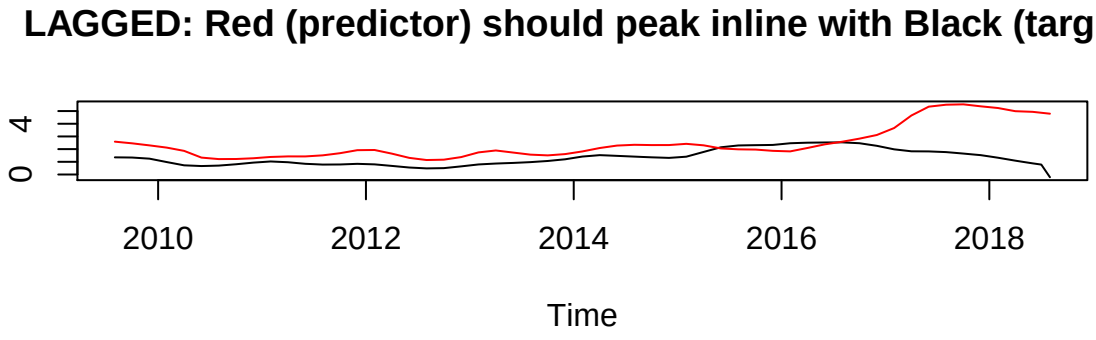
The correlations above were calculated based on trends, so we'll compare time series of trends (12 mth moving averages), not the underlying sales quantities.

Then after we apply the recommended lag, we expect to see the peaks in line.

```
exampley <- selected_series_withNAs %>%
              filter(PART_NUMBER == "CL-0006667431") %>%
                select(PART_NUMBER, INVOICE_MONTH_AsDate, TREND) %>%
                  rename(SALES_QTY = TREND, INVOICE_MONTH = INVOICE_MONTH_AsDate)

examplex <- selected_series_withNAs %>%
              filter(PART_NUMBER == "CL-0002394630") %>%
                select(PART_NUMBER, INVOICE_MONTH_AsDate, TREND) %>%
                  rename(SALES_QTY = TREND, INVOICE_MONTH = INVOICE_MONTH_AsDate)

# get time series objects from the pre built lists
exampley_ts <- getts(unique(exampley$PART_NUMBER), exampley,
                     months_list = monthslist, all_to_same_period = FALSE)

examplex_ts <- getts(unique(examplex$PART_NUMBER), examplex,
                     months_list = monthslist, all_to_same_period = FALSE)*2
```

```
par(mfrow=c(2,1))
ts.plot(ts.intersect(exampley_ts, stats::lag(examplex_ts,0)), col=c("black", "red"),
        main="Red (predictor) should peak before Black (target)")
ts.plot(ts.intersect(exampley_ts, stats::lag(examplex_ts,-12)), col=c("black", "red"),
        main="LAGGED: Red (predictor) should peak inline with Black (target)")
```





Not great, the correlations are finding spurious patterns. We'll proceed with this example anyway...

Let's build a regression model, we'll include as predictors: lags of the target to itself by up to 3periods lags of the predictor by up to 3periods. Then let the model find the best predictors.

Remeber, we are modelling the Trend, the 12mth moving average, not the sales quantities themselves.

```
# apply lags to predictors
# Note, the predictors lead the target. To bring them into line we have to push them forward,
# ie apply -ve lag. Also note dplyr conflicts with stats, so must code stats::lag()
allseries <- ts.intersect(exampley_ts,
                          exampley_ts_12 = stats::lag(exampley_ts,-12),
                          examplex_ts_12 = stats::lag(examplex_ts,-12),
                          examplex_ts_24 = stats::lag(examplex_ts,-24))
# create linear model
model_lm_exampley <- lm(exampley_ts ~ ., data = allseries)

#optimise
library(MASS)
#Mass conflicts with dplyr
select    <- dplyr::select
filter    <- dplyr::filter
summarise <- dplyr::summarise

model_lm_exampley_step <- stepAIC(model_lm_exampley, direction=c("both"), trace=FALSE)
```

```
summary(model_lm_exampley_step)
```

```
##
## Call:
## lm(formula = exampley_ts ~ exampley_ts_12 + examplex_ts_12 +
##     examplex_ts_24, data = allseries)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0245 -0.3029  0.0003  0.2624  0.9257
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.74626    0.12843   5.811 8.65e-08 ***
## exampley_ts_12  1.11130    0.11825   9.398 3.85e-15 ***
## examplex_ts_12 -0.23307    0.06480  -3.596 0.000519 ***
## examplex_ts_24 -0.14236    0.06853  -2.077 0.040529 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4502 on 93 degrees of freedom
## Multiple R-squared:  0.5347, Adjusted R-squared:  0.5197
## F-statistic: 35.63 on 3 and 93 DF,  p-value: 2.024e-15
```

In order to build this example we filtered on good correlations, choosing one of the best available. We were hoping to build a good example. It is concerning that we could only muster R2=0.5347

We could spend more time refining the process, perhaps by looking at different lags. But we need an automated approach which can be repeated thousands of times. We cannot hand craft thousands of such models.

But there is a bigger problem. We need to forecast 12months ahead. All the lags are <6mths, by design. We don't trust lags >6mths because signals get swamped by a common annual seasonality. Even if we attempted such models, its dis-heartening to see our 'good' example turn out so poor.

With lags <6 we would have to predict the predictors before predicting the target timeseries. Not a happy thought.

Nevertheless, it is a common approach. Indeed, the term "transfer function" applies to models in which we predict y from past lags of both y and x (including possibly lag 0), and we also model x and the errors. For prediction of the future past the end of the series, we might first use the model for x to forecast x. Then we would use the forecasted x when we forecast future y.

We've also looked at working with deseasonalised data, but the mean correlations in that component are very poor. Hence we chose Trends.

We could work with annual data, but we only have ten years of data, ie 10 data points per series. Most series will be less than 10yrs long. So few data points per series, its not heartening.

We can proceed with 'clustering'. Before we do, let's consider one more grouping strategy which could be clustered.

Correlation within 5 digit Parts Groups

The obvious cluster to use is each part number's natural group. For example CL-0009568611 belongs to the group of part numbers whose initial digits are: CL-00095. These all have related usage in real life and we may expect a reasonable correlation between the part number which represents the group (ie sum of all part sales in the group) CL-GRP-00095 and the parts which belong to the group

We can then compare this with the correlation we would expect by randomly grouping parts.

```
# To find these cross correlations we simply create a new column for the cross_correls dataframe
# and apply the group names, which are no more than curtailed part numbers.
# Then 'group by' those group names and view the resulting mean cross correlations.
getCorrelsGroupedByPrefix <- function(crosscorrel_data, randomised){


  #ensure NA's handled, simply set to ccor_no_lag
  ccor_NA_index <- which(is.na(crosscorrel_data$ccor_with_lag))
  crosscorrel_data$ccor_with_lag[ccor_NA_index] <- crosscorrel_data$ccor_no_lag[ccor_NA_index]

  # group part pairs by their prefixes, must share a common prefix
  with_prefix <- crosscorrel_data %>%
                 mutate(Prefix.x = ifelse(substr(PART_NUMBER.x,1,6)=="CL_GRP",
                                          substr(PART_NUMBER.x,8,11),
                                          substr(PART_NUMBER.x,4,7)),
                        Prefix.y = ifelse(substr(PART_NUMBER.y,1,6)=="CL_GRP",
                                          substr(PART_NUMBER.y,8,11),
                                          substr(PART_NUMBER.y,4,7)))%>%
                   filter(Prefix.x==Prefix.y)%>%
                     rename(Prefix = Prefix.x) %>%
                       select(-Prefix.y)

  #Randomised data is for comparing mean correlations to establish significance of the result
  #if we want randomised data then we overwrite the real prefixes with randomly assigned prefixes
  if (randomised){

    #We need to know how many randomised groups to create
    count <- nrow(with_prefix %>% select(Prefix) %>% distinct())

    #...and how many parts are in each group, this leads to a weighting for sample(prob...)
    weights = (with_prefix %>%
                 select(PART_NUMBER.x, Prefix) %>%
                  distinct() %>%
                    group_by(Prefix) %>%
                      summarise(count=n()))$count

    weights = weights/sum(weights, na.rm = TRUE)

    #Then assign each part to a random group
    random_grp_assignments <- crosscorrel_data %>%
                                select(PART_NUMBER=PART_NUMBER.x) %>%
                                  distinct() %>%
                                    mutate(Prefix = sample(1:count,
                                                           n(),
                                                           prob = weights,
                                                           replace = TRUE))

    #finally, overwrite the pairwise groupings
    with_prefix <- crosscorrel_data %>%
                     inner_join(y=random_grp_assignments, by=c("PART_NUMBER.x"="PART_NUMBER")) %>%
                       rename(Prefix.x=Prefix) %>%
                         inner_join(y=random_grp_assignments, by=c("PART_NUMBER.y"="PART_NUMBER"))%>%
                           rename(Prefix.y=Prefix) %>%
```

```
                            filter(Prefix.x==Prefix.y)%>%
                                rename(Prefix = Prefix.x) %>%
                                  select(-Prefix.y)
  }
  #The above pairwise correlations are not unique, so we have x vs y AND y vs X.
  #The below code makes the pairs unique, but doesn't care which way around each pair is

  with_prefix_unqpairs <-data.frame(unique(t(apply(with_prefix[,c(1,2)], 1, sort))),
                                    stringsAsFactors = F)
  colnames(with_prefix_unqpairs)<-c("PART_NUMBER.x","PART_NUMBER.y")
  with_prefix_unqcorrels <- with_prefix %>%
                              inner_join(y=with_prefix_unqpairs,
                                         by=c("PART_NUMBER.x","PART_NUMBER.y")) %>%
                                arrange(Prefix, PART_NUMBER.x, PART_NUMBER.y)

  unique_parts_per_prefix <- rbind(with_prefix_unqcorrels %>%
                                     select(Prefix, PART_NUMBER=PART_NUMBER.x) %>%
                                      distinct(),
                                   with_prefix_unqcorrels %>%
                                     select(Prefix, PART_NUMBER=PART_NUMBER.y) %>%
                                      distinct())%>%
                                    distinct() %>%
                                      group_by(Prefix)%>%
                                        summarise(PartsInGrp = n())
  #summarise the results
  summary <- with_prefix_unqcorrels %>%
              inner_join(y=unique_parts_per_prefix,by=c("Prefix")) %>%
                group_by(Prefix) %>%
                 summarise(Mean_Correl_NoLag=mean(abs(ccor_no_lag)),
                           Mean_Correl_WithLag=mean(abs(ccor_with_lag)),
                           CrossCorrelationsInGrp=n(),
                           PartsInGrp=max(PartsInGrp))%>%
                  arrange(Prefix)

  return(list(summary,with_prefix_unqcorrels, unique_parts_per_prefix))
}
```

```
a <- getCorrelsGroupedByPrefix(crosscorrel_trends,  randomised = FALSE)
b <- getCorrelsGroupedByPrefix(crosscorrel_trends,  randomised = TRUE)
c <- getCorrelsGroupedByPrefix(crosscorrel_rawdata, randomised = FALSE)
d <- getCorrelsGroupedByPrefix(crosscorrel_rawdata, randomised = TRUE)

CorrelsByPrefix_trends     <- a[[1]]
CorrelsByPrefix_trends_rnd <- b[[1]]
CorrelsByPrefix_raw        <- c[[1]]
CorrelsByPrefix_raw_rnd    <- d[[1]]
rm(a,b,c,d)

#For future reference we need to know the number of groups
qty_grps_trends <- nrow(CorrelsByPrefix_trends)
```

```
#Print the mean grouped correlation
print(paste0("The data has been arranged into ", nrow(CorrelsByPrefix_trends), " groups." ))
```

```
## [1] "The data has been arranged into 29 groups."
```

```
print(paste0("Mean correlation of sales trends grouped by prefix =",
             mean(CorrelsByPrefix_trends$Mean_Correl_WithLag)))
```

```
## [1] "Mean correlation of sales trends grouped by prefix =0.331314610043276"
```

```
print(paste0("Mean correlation of sales trends grouped randomly =",
             mean(CorrelsByPrefix_trends_rnd$Mean_Correl_WithLag)))
```

```
## [1] "Mean correlation of sales trends grouped randomly =0.318469648893848"
```

```
print(paste0("Mean correlation of raw sales grouped by prefix =",
             mean(CorrelsByPrefix_raw$Mean_Correl_WithLag)))
```

```
## [1] "Mean correlation of raw sales grouped by prefix =0.29498994601423"
```

```
print(paste0("Mean correlation of raw sales grouped randomly =",
             mean(CorrelsByPrefix_raw_rnd$Mean_Correl_WithLag)))
```

```
## [1] "Mean correlation of raw sales grouped randomly =0.268198455521371"
```

Let's see how signficant that result is. We do this by comparing our results with those for randomly assigned prefixes for each part. These random prefixes have the same distirbution as the underlying data (same qty of parts per grp). randomly creating the same number of groups and then averaging the cross correlations within those groups.

We then perform a t-test between these randomly created mean correlations and the properly grouped correlations

```
# combine into single simple table for t test, to tell whether correlations
# in groups are significantly different

perform_ttest <- function(CorrelsByPrefix_data, CorrelsByPrefix_random, name){
  ttest_dat <- rbind( (CorrelsByPrefix_data %>%
                         mutate(Source="ProperlyGroupedData") %>%
                           arrange(desc(PartsInGrp)) %>%
                             select(Prefix, Correl = Mean_Correl_WithLag, Source)
                         )[1:nrow(CorrelsByPrefix_random),]#ensures same qty as in random data
                         ,
                       CorrelsByPrefix_random %>%
                         select(Prefix, Correl = Mean_Correl_WithLag) %>%
                           mutate(Source="RandomlyGroupedData")
                     )
  ttest_dat$Source <- as.factor(ttest_dat$Source)

  #Check correlation values are normally distributed
  #qqnorm(GroupCorrels$Mean_Correl_WithLag)
  #var(GroupCorrels$Mean_Correl_WithLag)
  #qqnorm(crosscorrel_rawdata_unq$ccor_with_lag)
  #var(crosscorrel_rawdata_unq$ccor_with_lag)

  #Hypothesis test, are these significantly different distributions of correlation values?
  ttest_result <- t.test(Correl~ Source, paired = TRUE, var.equal=FALSE, data=ttest_dat)

  #
  print(paste0(name, ": Are grouped correlations signficantly different from random groups?",
               " The T Test's p-value = ",
```

```
                round(ttest_result$p.value,8),
                " . If <0.05 then significantly different (@95% confidence)"))
}

perform_ttest(CorrelsByPrefix_trends, CorrelsByPrefix_trends_rnd, "TRENDS")
```

```
## [1] "TRENDS: Are grouped correlations signficantly different from random groups? The T Test's p-valu
```

```
perform_ttest(CorrelsByPrefix_raw, CorrelsByPrefix_raw_rnd, "RAW SALES")
```

```
## [1] "RAW SALES: Are grouped correlations signficantly different from random groups? The T Test's p-va
```

Clearly grouping by the 5 digit code is not very effective, although possible significant for one feature. Thankfully, there may be better ways of grouping.

Clustering Methods

It is common to apply PCA (principal component analysis) to correlation values in order to then discover groups based on distance in some multi dimensional space. PCA can reduce that space to just 2 or 3 dimensions, hence to plot and view the groups.

The question is, what dimensions to start with? How to characterise the sales of each part number?

OPTION v1:

Parts could be plotted by their cross correlations. So, each part could be characterised by its cross correlation with each of the other parts. Giving us as many dimensions are there are parts. This is a common approach.

There is an awkward aspect to this. The ccf() function may reveal that the optimal lag for part_X vs part_Y is different to the optimal lag for part_X vs part_Z. Should we always use the optimal correlation value? Or set zero lag for all? Also, correlations change over time. There will be trends in correlations over time.

OPTION v2:

Parts could be plotted using their raw sales data by month. So there would be one dimension for each month in the sales history. Alternatively, the decomposed sales data could be used, ie long term trend for each month or seasonal adjustments for each month. We could also plot by sales performance in month of the year, giving us just 12 dimensions. This would highlight the seasonality of each part, but obscure trends over the years.

OPTION v3:

We can use wavelets, a method to find what transforms would be required to project one wave onto another. This allows us to seek common transforms and group waves accordingly. The sales histories can be considered as waves. This option will be explored last.

Let's explore options 1 and 2 to see if they supply us with useful, correlated, clusters of parts.

```
#Prep data for PCA

#OPTION v1, describe partnumbers by OTHER part numbers
pcadata_with_outliers_v1_trend <- crosscorrel_trends_unq %>%
                                  select(PART_NUMBER = PART_NUMBER.x, PART_NUMBER.y, ccor_no_lag) %>%
                                    arrange(PART_NUMBER) %>%
```

```
                                    spread(key=PART_NUMBER, value=ccor_no_lag, fill = 1.0) %>%
                                      rename(PART_NUMBER = PART_NUMBER.y)

pcadata_with_outliers_v1_rawsales <- crosscorrel_rawdata_unq %>%
                                  select(PART_NUMBER = PART_NUMBER.x, PART_NUMBER.y, ccor_no_lag) %>%
                                    arrange(PART_NUMBER) %>%
                                      spread(key=PART_NUMBER, value=ccor_no_lag, fill = 1.0) %>%
                                        rename(PART_NUMBER = PART_NUMBER.y)

pcadata_with_outliers_v1_rawsales_log <- crosscorrel_rawdata_log_unq %>%
                                  select(PART_NUMBER = PART_NUMBER.x, PART_NUMBER.y, ccor_no_lag) %>%
                                    arrange(PART_NUMBER) %>%
                                      spread(key=PART_NUMBER, value=ccor_no_lag, fill = 1.0) %>%
                                        rename(PART_NUMBER = PART_NUMBER.y)

pcadata_with_outliers_v1_rawsales_ds <- crosscorrel_rawdata_ds_unq %>%
                                  select(PART_NUMBER = PART_NUMBER.x, PART_NUMBER.y, ccor_no_lag) %>%
                                    arrange(PART_NUMBER) %>%
                                      spread(key=PART_NUMBER, value=ccor_no_lag, fill = 1.0) %>%
                                        rename(PART_NUMBER = PART_NUMBER.y)

#OPTION v2, describe part numbers by sales performance in each month
pcadata_with_outliers_v2_trend <- selected_series_withNAs %>%
                                    select(PART_NUMBER, TREND, INVOICE_MONTH) %>%
                                      spread(key="INVOICE_MONTH", value="TREND", fill = NA)

pcadata_with_outliers_v2_rawsales <- selected_series_withNAs %>%
                                      select(PART_NUMBER, SALES_QTY, INVOICE_MONTH) %>%
                                        spread(key="INVOICE_MONTH", value="SALES_QTY", fill = NA)

pcadata_with_outliers_v2_rawlog <- selected_series_withNAs_log %>%
                                      select(PART_NUMBER, SALES_QTY, INVOICE_MONTH) %>%
                                        spread(key="INVOICE_MONTH", value="SALES_QTY", fill = NA)

pcadata_with_outliers_v2_trendlog <- selected_series_withNAs_log %>%
                                      select(PART_NUMBER, TREND, INVOICE_MONTH) %>%
                                        spread(key="INVOICE_MONTH", value="TREND", fill = NA)

# remove first month column (second col in table), always NA
pcadata_with_outliers_v2_trend     <- pcadata_with_outliers_v2_trend[,-2]
pcadata_with_outliers_v2_rawsales  <- pcadata_with_outliers_v2_rawsales[,-2]
pcadata_with_outliers_v2_rawlog    <- pcadata_with_outliers_v2_rawlog[,-2]
pcadata_with_outliers_v2_trendlog  <- pcadata_with_outliers_v2_trendlog[,-2]
```

Whichever approach we take we need a function to calculate PCA, preferably without outliers which make PCA less effective.

```
library(tidyr)

# PCA works best when outliers are excluded
# So, scale data and identify outliers
# This means identifying rows (ie partnumbers), whose means are more than 2SD from overall mean.
# For scaled data, overall mean = 0 and overall SD = 1. So seek row means where mean >2 or < -2
```

```r
getPCA <- function(pca_data, SDthreshold = NA, DisplayPlot, setRownames=TRUE){

  #tidy data
  pca_data <- as.data.frame(pca_data) #else problems with tibbles and row names
  pca_data[is.na(pca_data)] <- 0
  if(setRownames){
    row.names(pca_data) <- unlist(pca_data[,1], use.names = FALSE)
    pca_data <- pca_data[,-1]
  }
  if(!is.na(SDthreshold)){
    #identify outliers
    outliers_mean <- mean(as.matrix(pca_data))
    outliers_sd   <- sd(as.matrix(pca_data))
    outliers      <- rowMeans(pca_data)

    #remove outliers
    outliers <- outliers[which(outliers > outliers_mean+SDthreshold*outliers_sd
                               |
                               outliers < (outliers_mean-SDthreshold*outliers_sd))]
    outliers <- attr(outliers, "names")

    #hand coded outliers...
    #outliers <- c(outliers, "CL_GRP-UKHT1","CL_GRP-UKCL0", "CL_GRP-00095",
    #              "CL_GRP-00023","CL_GRP-00062","CL-0006264080", "CL-0006264071" )

    # remove outliers, if there are any
    pca_data <- pca_data[which(!row.names(pca_data) %in% outliers),]
  }else{
    outliers <- NULL
  }

  # PCA
  pca_obj <- prcomp(x=pca_data, center = TRUE, scale. = TRUE)

  if(DisplayPlot){
    #2D plot
    plot(pca_obj$x[,1], pca_obj$x[,2])
    text(pca_obj$x[,1], pca_obj$x[,2], labels=attr(pca_obj$x, "dimnames")[[1]], cex= 0.7)

    #3D plot
    pca_3D <- data.frame(pca_obj$x[,1:3])
    colnames(pca_3D) <- c("x","y","z")
    library(scatterplot3d)
    with(pca_3D, {
      scatterplot3d(x, y, z, angle=30,
                    pch=20, main="PCA dims 1:3, by Part Number.")
                    #xlim=c(-5,10), ylim=c(-100,50), zlim=c(-10,10))
    })
  }

  return(list(pca_obj = pca_obj,
              pca_data= pca_data,
              outliers= outliers))
```

```
}
```

Get PCA of Correlations

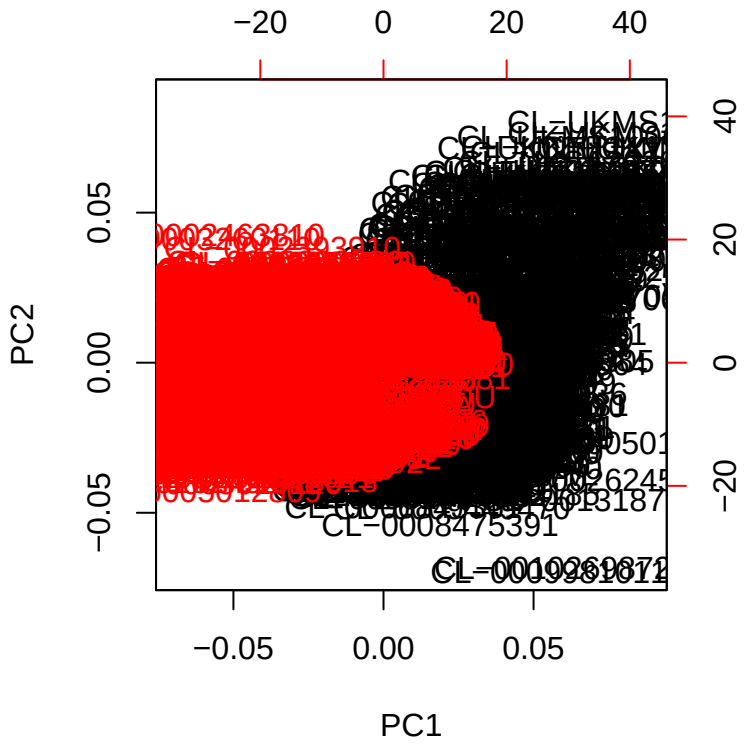
Apply the pca function to the various configurations and plot

```
#OPTION v1a. PCA on correlation matrix of trends
pca_obj_v1_trend_list <- getPCA(pcadata_with_outliers_v1_trend, DisplayPlot=FALSE)
pca_obj_v1_trend      <- pca_obj_v1_trend_list$pca_obj
pca_data_v1_trend     <- pca_obj_v1_trend_list$pca_data
outliers_v1_trend     <- pca_obj_v1_trend_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v1_trend_list$outliers)))
rm(pca_obj_v1_trend_list)

#OPTION v1b. PCA on correlation matrix of raw sales data
pca_obj_v1_raw_list <- getPCA(pcadata_with_outliers_v1_rawsales, DisplayPlot=FALSE)
pca_obj_v1_raw      <- pca_obj_v1_raw_list$pca_obj
pca_data_v1_raw     <- pca_obj_v1_raw_list$pca_data
outliers_v1_raw     <- pca_obj_v1_raw_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v1_raw_list$outliers)))
rm(pca_obj_v1_raw_list)

#OPTION v1b. PCA on correlation matrix of raw sales data
pca_obj_v1_rawlog_list <- getPCA(pcadata_with_outliers_v1_rawsales, DisplayPlot=FALSE)
pca_obj_v1_rawlog      <- pca_obj_v1_rawlog_list$pca_obj
pca_data_v1_rawlog     <- pca_obj_v1_rawlog_list$pca_data
outliers_v1_rawlog     <- pca_obj_v1_rawlog_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v1_raw_list$outliers)))
rm(pca_obj_v1_rawlog_list)

#OPTION v1b. PCA on correlation matrix of raw sales data
pca_obj_v1_rawds_list <- getPCA(pcadata_with_outliers_v1_rawsales_ds, DisplayPlot=FALSE)
pca_obj_v1_rawds      <- pca_obj_v1_rawds_list$pca_obj
pca_data_v1_rawds     <- pca_obj_v1_rawds_list$pca_data
outliers_v1_rawds     <- pca_obj_v1_rawds_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v1_raw_list$outliers)))
rm(pca_obj_v1_rawds_list)

#OPTION v2. PCA on correlation matrix
#NOTE SD threshold set to 0.99 for outliers
pca_obj_v2_trend_list <- getPCA(pcadata_with_outliers_v2_trend, DisplayPlot=FALSE)
pca_obj_v2_trend  <- pca_obj_v2_trend_list$pca_obj
pca_data_v2_trend <- pca_obj_v2_trend_list$pca_data
outliers_v2_trend <- pca_obj_v2_trend_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v2_trend_list$outliers)))
rm(pca_obj_v2_trend_list)

#OPTION v2. PCA on correlation matrix
```

```
#NOTE SD threshold set to 0.5 for outliers
pca_obj_v2_raw_list <- getPCA(pcadata_with_outliers_v2_rawsales, DisplayPlot=FALSE)
pca_obj_v2_raw  <- pca_obj_v2_raw_list$pca_obj
pca_data_v2_raw <- pca_obj_v2_raw_list$pca_data
outliers_v2_raw <- pca_obj_v2_raw_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v2_raw_list$outliers)))
rm(pca_obj_v2_raw_list)

#OPTION v2. PCA on correlation matrix
#NOTE SD threshold set to 0.5 for outliers
pca_obj_v2_rawlog_list <- getPCA(pcadata_with_outliers_v2_rawlog, DisplayPlot=FALSE)
pca_obj_v2_rawlog  <- pca_obj_v2_rawlog_list$pca_obj
pca_data_v2_rawlog <- pca_obj_v2_rawlog_list$pca_data
outliers_v2_rawlog <- pca_obj_v2_rawlog_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v2_raw_list$outliers)))
rm(pca_obj_v2_rawlog_list)

#OPTION v2. PCA on correlation matrix
#NOTE SD threshold set to 0.5 for outliers
pca_obj_v2_trendlog_list <- getPCA(pcadata_with_outliers_v2_trendlog, DisplayPlot=FALSE)
pca_obj_v2_trendlog  <- pca_obj_v2_trendlog_list$pca_obj
pca_data_v2_trendlog <- pca_obj_v2_trendlog_list$pca_data
outliers_v2_trendlog <- pca_obj_v2_trendlog_list$outliers

#print(paste0("Number of outliers excluded from PCA: ",length(pca_obj_v2_raw_list$outliers)))
rm(pca_obj_v2_trendlog_list)
```

We can use biplots to understand our PCA, but they stend to be quit messy because the first two components describe a relatively small proportion of the variance in our data.

```
biplot (pca_obj_v1_trend, scale =1)
```



PCA is helpful because it allows us to reduce the number of dimensions we need to deal with whilst retaining most of the variance. It is a compression algorithm, indeed it can be used for image compression. So, for each option (v1, v2a, v2b) how many dimensions do we need to retain 99% of the variance?

```
# How many PCA dimensions are required to explain 99% of variance?
# cumulative importance can be found on summary(pca_1)$importance[3,]
# Let's use that to find the first dimension which is > 99%

getThresholdDim <- function(pca_obj, approach, DisplaySummary){

  threshold_dim <- min(summary(pca_obj)$importance[3,][which(summary(pca_obj)$importance[3,] > 0.99)])
  threshold_dim <- as.vector(which(summary(pca_obj)$importance[3,]==threshold_dim))

  summary(pca_obj)$importance[3,1:12]
  print(paste0("For PCA on ", approach, ", ",
               threshold_dim,
               " dimensions are required to explain 99% of the variance in the data. ",
               ncol(summary(pca_obj)$importance),
               " dimensions were available"
               ))
  cat(" \n")

  return(threshold_dim)
}
```

```
# V1a PCA on correlations of trends
threshold_dim_v1a <- getThresholdDim(pca_obj_v1_trend, "v1a:correlations of trends",
```

```
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v1a:correlations of trends, 927 dimensions are required to explain 99% of the varian
##
```

```
# V1b PCA on correlations of raw sales
threshold_dim_v1b <- getThresholdDim(pca_obj_v1_raw,   "v1b:correlations of raw sales",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v1b:correlations of raw sales, 886 dimensions are required to explain 99% of the var
##
```

```
# V1c PCA on correlations of raw sales logged
threshold_dim_v1c <- getThresholdDim(pca_obj_v1_rawlog,"v1c:correlations of rawlog sales",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v1c:correlations of rawlog sales, 886 dimensions are required to explain 99% of the
##
```

```
# V1d PCA on correlations of raw sales logged
threshold_dim_v1d <- getThresholdDim(pca_obj_v1_rawds, "v1d:correlations of rawds sales",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v1d:correlations of rawds sales, 948 dimensions are required to explain 99% of the va
##
```

```
# V2a PCA on sales trend data
threshold_dim_v2a <- getThresholdDim(pca_obj_v2_trend, "v2a:sales trends",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v2a:sales trends, 4 dimensions are required to explain 99% of the variance in the da
##
```

```
# V2b PCA on raw sales data
threshold_dim_v2b <- getThresholdDim(pca_obj_v2_raw,   "v2b:raw sales",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v2b:raw sales, 55 dimensions are required to explain 99% of the variance in the data
##
```

```
# V2c PCA on log(raw sales data +1 )
threshold_dim_v2c <- getThresholdDim(pca_obj_v2_rawlog,"v2c:Log(raw sales+1)",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v2c:Log(raw sales+1), 112 dimensions are required to explain 99% of the variance in
##
```

```
# V2c PCA on log(trend data +1 )
threshold_dim_v2d <- getThresholdDim(pca_obj_v2_trendlog, "v2d:Log(trend+1)",
                                    DisplaySummary=TRUE)
```

```
## [1] "For PCA on v2d:Log(trend+1), 16 dimensions are required to explain 99% of the variance in the da
##
```

So sales trends yield especially well to PCA, requiring only 11 of the 114 available dimensions to retain 99% of the variance.

Let's now consider how to 'cluster' the parts within these spaces. There are at least three easily available methods:, Kmeans, DBScan and HClust. Let's see which finds the highest correlation groups.

Note, we now have 3 clustering methods for each of the three data options. So, 9 secnarios to consider before selecting the option which profers the highest correlated groups.

Build Clustering Functions

There are numerous methods for clusterin. we'll do the more common techniques first: - Kmeans - Hierarchical - Clustering - DBScan

Later we'll consider newer techniques, specifically for timeseries clustering:

- Dynamic Time Warping (DTW) from the dtwclust package.
- Wavelets from the DWT (Discrete Wavelet Transform) package (Beware DWT vs DTW)

There is a package called dtwclust which has numerous alterntives, but we ran out of time to explore them.

We will attempt to cluster by each of the main features we already have:

- Raw values for ... – Sales Qty – Trends
- Correlations within... – Sales Qty – Log(Sales Qty+1) – Sales Qty Deseasonalised – Trend – Seasonal

K MEANS

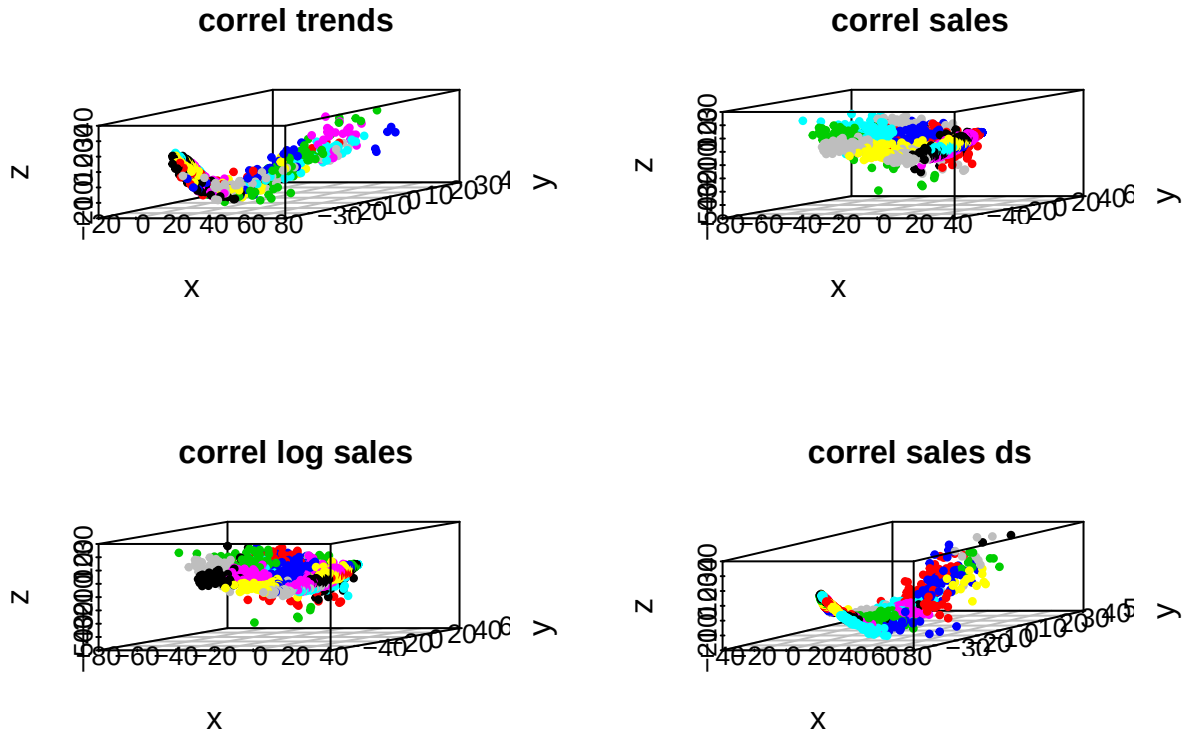
K mean is the simplest clustering function, and sometimes derided because of that. However, the plots show no obvious patterns so a simple distance based function like kmeans may be very effective.

```
#Define function to get clusters using PCA and KMeans
GetClusterAssignment_PcaAndKmeans <- function(pca_obj, threshold_dim, cluster_qty, iter.max=10){

  kmeans_obj <- kmeans(pca_obj$x[,1:threshold_dim],
                       centers  = cluster_qty,
                       iter.max = iter.max)

  cluster_assignment <- cbind.data.frame(PART_NUMBER      = attr(kmeans_obj$cluster, "names"),
                                         CLUSTER          = kmeans_obj$cluster,
                                         stringsAsFactors = F)

  return(list(clust_assign=cluster_assignment, kmeans_obj=kmeans_obj))
  }
```

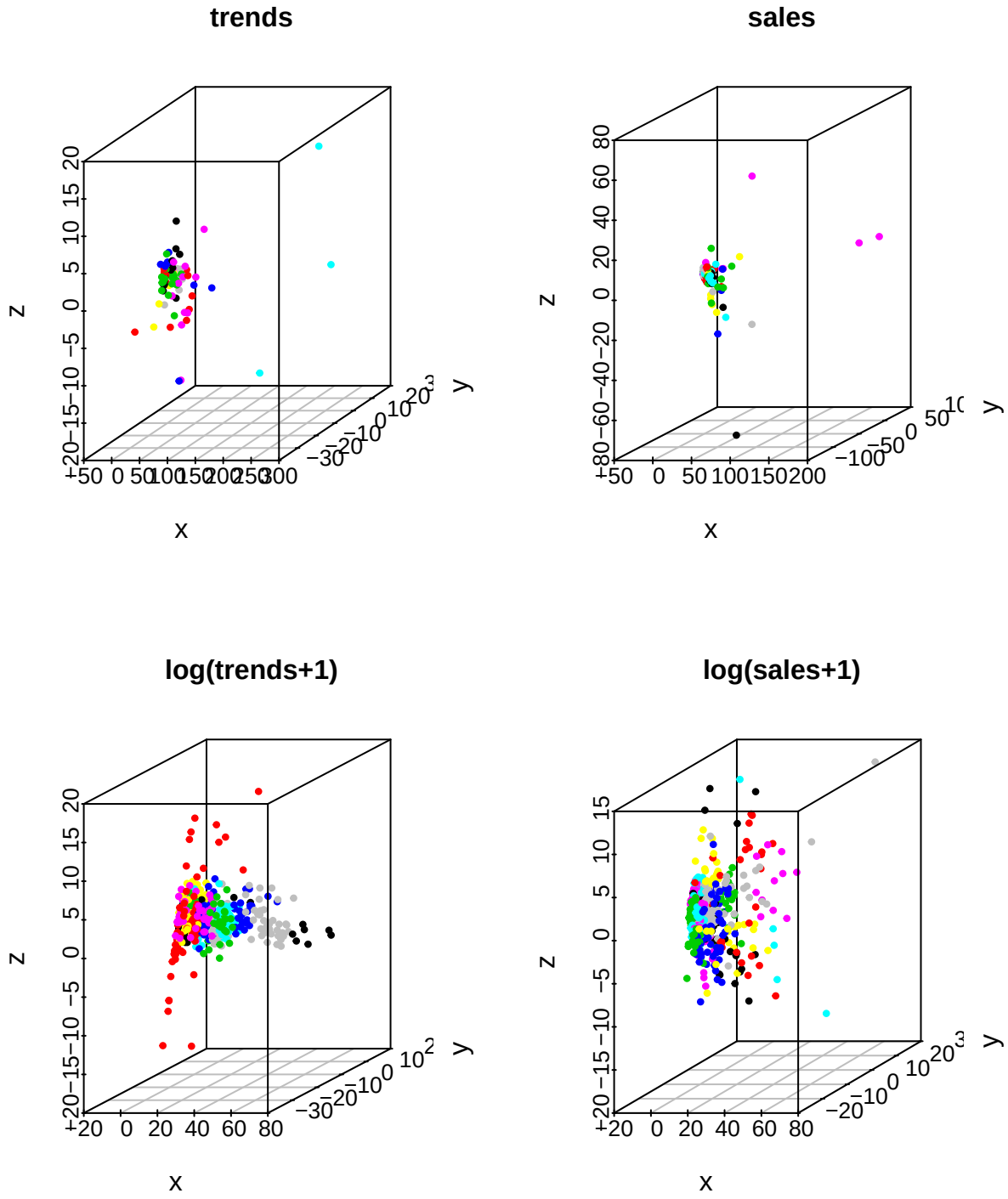
```
#execute function
cluster_assignment_kmeans_v1a  <- GetClusterAssignment_PcaAndKmeans(pca_obj_v1_trend,
                                                                    threshold_dim_v1a,
                                                                    cluster_qty=50)

cluster_assignment_kmeans_v1b  <- GetClusterAssignment_PcaAndKmeans(pca_obj_v1_raw,
                                                                    threshold_dim_v1b,
                                                                    cluster_qty=50)

cluster_assignment_kmeans_v1c  <- GetClusterAssignment_PcaAndKmeans(pca_obj_v1_rawlog,
                                                                    threshold_dim_v1c,
                                                                    cluster_qty=50)

cluster_assignment_kmeans_v1d  <- GetClusterAssignment_PcaAndKmeans(pca_obj_v1_rawds,
                                                                    threshold_dim_v1d,
                                                                    cluster_qty=50)
```

```
cluster_assignment_kmeans_v2a <- GetClusterAssignment_PcaAndKmeans(pca_obj_v2_trend,
                                                                   threshold_dim_v2a,
                                                                   iter.max    = 20,
                                                                   cluster_qty = 50)

cluster_assignment_kmeans_v2b <- GetClusterAssignment_PcaAndKmeans(pca_obj_v2_raw,
                                                                   threshold_dim_v2b,
                                                                   iter.max    = 20,
                                                                   cluster_qty = 50)

cluster_assignment_kmeans_v2c <- GetClusterAssignment_PcaAndKmeans(pca_obj_v2_rawlog,
                                                                   threshold_dim_v2c,
                                                                   iter.max    = 20,
                                                                   cluster_qty = 50)

cluster_assignment_kmeans_v2d <- GetClusterAssignment_PcaAndKmeans(pca_obj_v2_trendlog,
                                                                   threshold_dim_v2d,
                                                                   iter.max    = 20,
                                                                   cluster_qty = 50)
```

```
#Plot results
library(scatterplot3d)
plot3Dcluster <- function(pca_obj, cluster_assignment, approach){

  pca_3D <- data.frame(x = pca_obj$x[,1],
                       y = pca_obj$x[,2],
                       z = pca_obj$x[,3],
                       stringsAsFactors = F)

  with(pca_3D, {scatterplot3d(x, y, z,
                              angle=30,
                              color=cluster_assignment[[1]]$CLUSTER,
                              pch=20,
                              main=paste0("", approach,""))})
}

#execute function
par(mfrow=c(2,2))
plot3Dcluster(pca_obj_v1_trend, cluster_assignment_kmeans_v1a, "correl trends")
plot3Dcluster(pca_obj_v1_raw,   cluster_assignment_kmeans_v1b, "correl sales")
plot3Dcluster(pca_obj_v1_rawlog,cluster_assignment_kmeans_v1c, "correl log sales")
plot3Dcluster(pca_obj_v1_rawds, cluster_assignment_kmeans_v1d, "correl sales ds")
```




```
par(mfrow=c(2,2))
plot3Dcluster(pca_obj_v2_trend,    cluster_assignment_kmeans_v2a, "trends")
plot3Dcluster(pca_obj_v2_raw,      cluster_assignment_kmeans_v2b, "sales")
plot3Dcluster(pca_obj_v2_trendlog, cluster_assignment_kmeans_v2d, "log(trends+1)")
plot3Dcluster(pca_obj_v2_rawlog,   cluster_assignment_kmeans_v2c, "log(sales+1)")
```



HDBScan

DBscan and HDBscan are excellent at identifying complex patterns based on density. But they work best in just 2 or 3 dimensions. Also, we cannot tell the function how many groups to find, which means a pairwise

ttest cant easily be used to compare its performance with the other methods. Instead, we must give the function a minimum number of points per cluster.

```
library(dbscan)
```

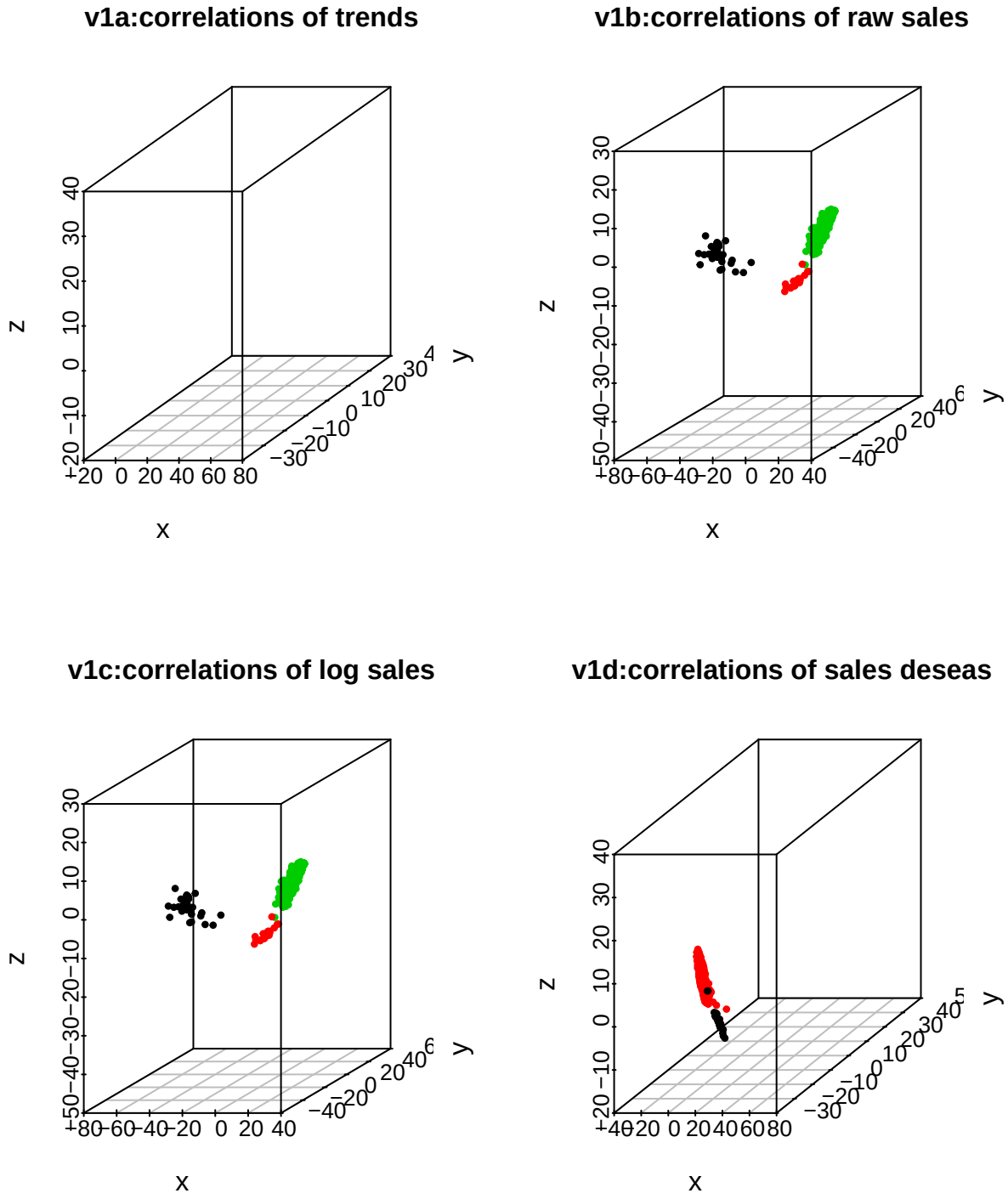
```
## Warning: package 'dbscan' was built under R version 3.5.1
```

```
GetClusterAssignment_PcaAndHDBScan <- function(pca_obj, threshold_dim, min_points_per_cluster=5,
                                               first_guess=30, cluster_qty){

  hdbscan_data<- pca_obj$x[,1:threshold_dim]
  hdbscan_obj <- hdbscan(hdbscan_data, minPts = min_points_per_cluster)

  cluster_assignment <- cbind.data.frame(PART_NUMBER = row.names(pca_obj$x),
                                         CLUSTER = as.vector(hdbscan_obj$cluster),
                                         stringsAsFactors = F)

  return(list(clust_assign=cluster_assignment, hdbscan_obj=hdbscan_obj))
}
```

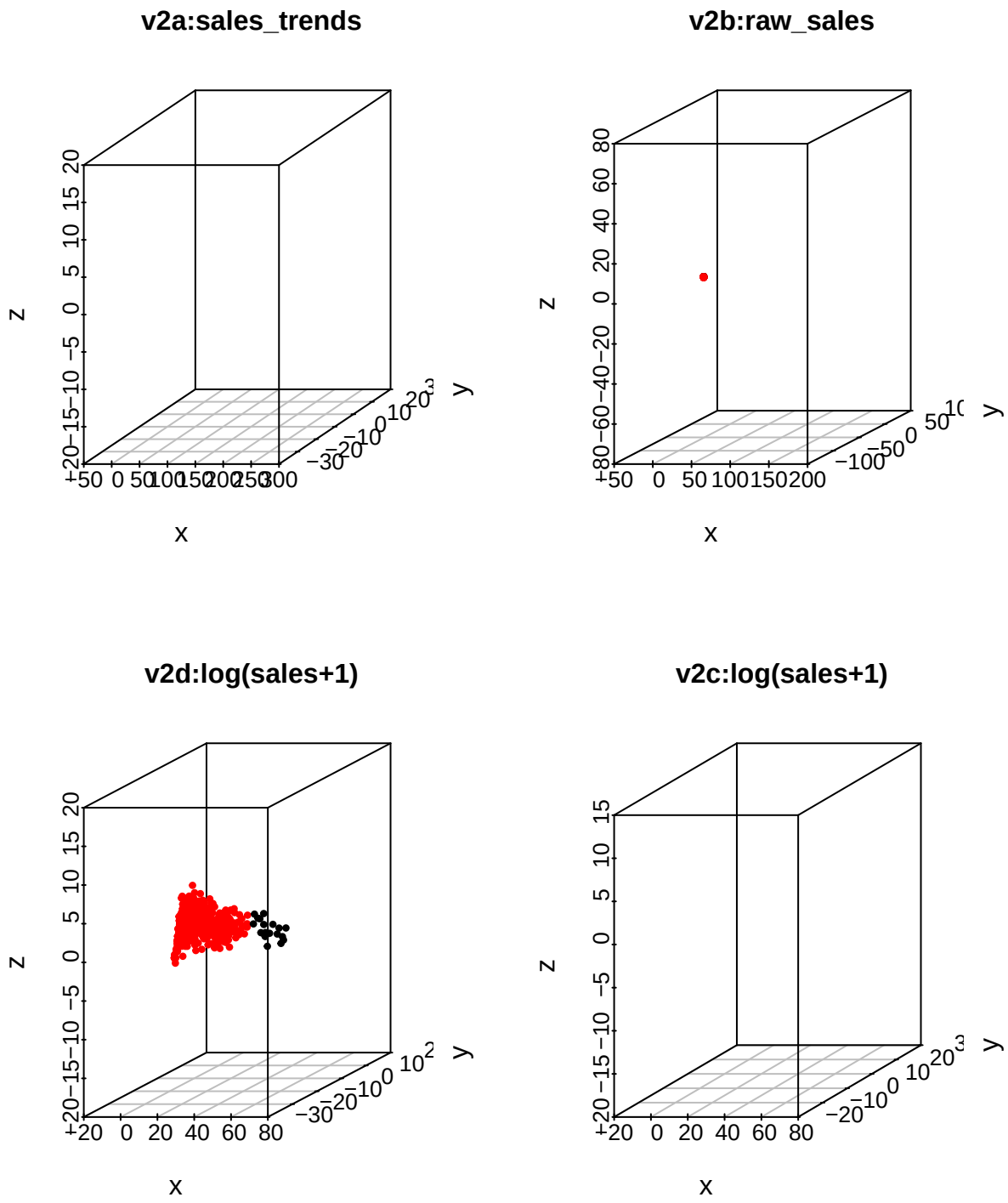
```
#Clusters for PCA of cross correlations of trends
cluster_assignment_hdbscan_v1a  <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v1_trend,
                                                                      threshold_dim=threshold_dim_v1a,
                                                                      min_points_per_cluster=10,
                                                                      first_guess=27,
                                                                      cluster_qty=50)
#Clusters for PCA of cross correlations of raw sales
cluster_assignment_hdbscan_v1b  <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v1_raw,
                                                                      threshold_dim=threshold_dim_v1b,
                                                                      min_points_per_cluster=10,
                                                                      first_guess=27,
                                                                      cluster_qty=50)

#Clusters for PCA of cross correlations of raw sales logged
cluster_assignment_hdbscan_v1c  <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v1_rawlog,
                                                                      threshold_dim=threshold_dim_v1c,
                                                                      min_points_per_cluster=10,
                                                                      first_guess=27,
                                                                      cluster_qty=50)

#Clusters for PCA of cross correlations of raw sales logged
cluster_assignment_hdbscan_v1d  <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v1_rawds,
                                                                      threshold_dim=threshold_dim_v1d,
                                                                      min_points_per_cluster=10,
                                                                      first_guess=27,
                                                                      cluster_qty=50)

#Clusters for PCA of sales trends
cluster_assignment_hdbscan_v2a <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v2_trend,
                                                                     threshold_dim=threshold_dim_v2a,
                                                                     min_points_per_cluster=10,
                                                                     first_guess=12,
                                                                     cluster_qty=50)
#Clusters for PCA of raw sales data
cluster_assignment_hdbscan_v2b <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v2_raw,
```

```
                                                                    threshold_dim=threshold_dim_v2b,
                                                                    min_points_per_cluster=10,
                                                                    first_guess=12,
                                                                    cluster_qty=50)
#Clusters for PCA of log(raw sales data +1)
cluster_assignment_hdbscan_v2c <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v2_rawlog,
                                                                    threshold_dim=threshold_dim_v2c,
                                                                    min_points_per_cluster=10,
                                                                    first_guess=12,
                                                                    cluster_qty=50)
#Clusters for PCA of log(trends+1)
cluster_assignment_hdbscan_v2d <- GetClusterAssignment_PcaAndHDBScan(pca_obj_v2_trendlog,
                                                                    threshold_dim=threshold_dim_v2d,
                                                                    min_points_per_cluster=10,
                                                                    first_guess=12,
                                                                    cluster_qty=50)
```

```
#Plot results
par(mfrow=c(2,2))
plot3Dcluster(pca_obj_v1_trend, cluster_assignment_hdbscan_v1a, "v1a:correlations of trends")
plot3Dcluster(pca_obj_v1_raw,   cluster_assignment_hdbscan_v1b, "v1b:correlations of raw sales")
plot3Dcluster(pca_obj_v1_rawlog,cluster_assignment_hdbscan_v1c, "v1c:correlations of log sales")
plot3Dcluster(pca_obj_v1_rawds, cluster_assignment_hdbscan_v1d, "v1d:correlations of sales deseas")
```




```
par(mfrow=c(2,2))
plot3Dcluster(pca_obj_v2_trend, cluster_assignment_hdbscan_v2a, "v2a:sales_trends")
plot3Dcluster(pca_obj_v2_raw,   cluster_assignment_hdbscan_v2b, "v2b:raw_sales")
plot3Dcluster(pca_obj_v2_trendlog,cluster_assignment_hdbscan_v2d, "v2d:log(sales+1)")
plot3Dcluster(pca_obj_v2_rawlog,cluster_assignment_hdbscan_v2c, "v2c:log(sales+1)")
```



Fundamentally, the data is arranged such that DBscan methods don't find enough clusters. So let's move on to HCLust (hierarchical, ie tree, clustering)

HCLUST

An alternative approach is to use hierarchical clustering. An introduction to this method can be found at the below link, which has a particular reference to hclust for correlations. http://www.sthda.com/english/wiki/print.php?id=237#hierarchical-clustering-and-correlation-based-distance

Hierarchical clustering uses distances, whereas correlations as the inverse of a distance (the higher the correlation, the lower the distance between two parts). So we convert to a distance by simply using 1-correlation.

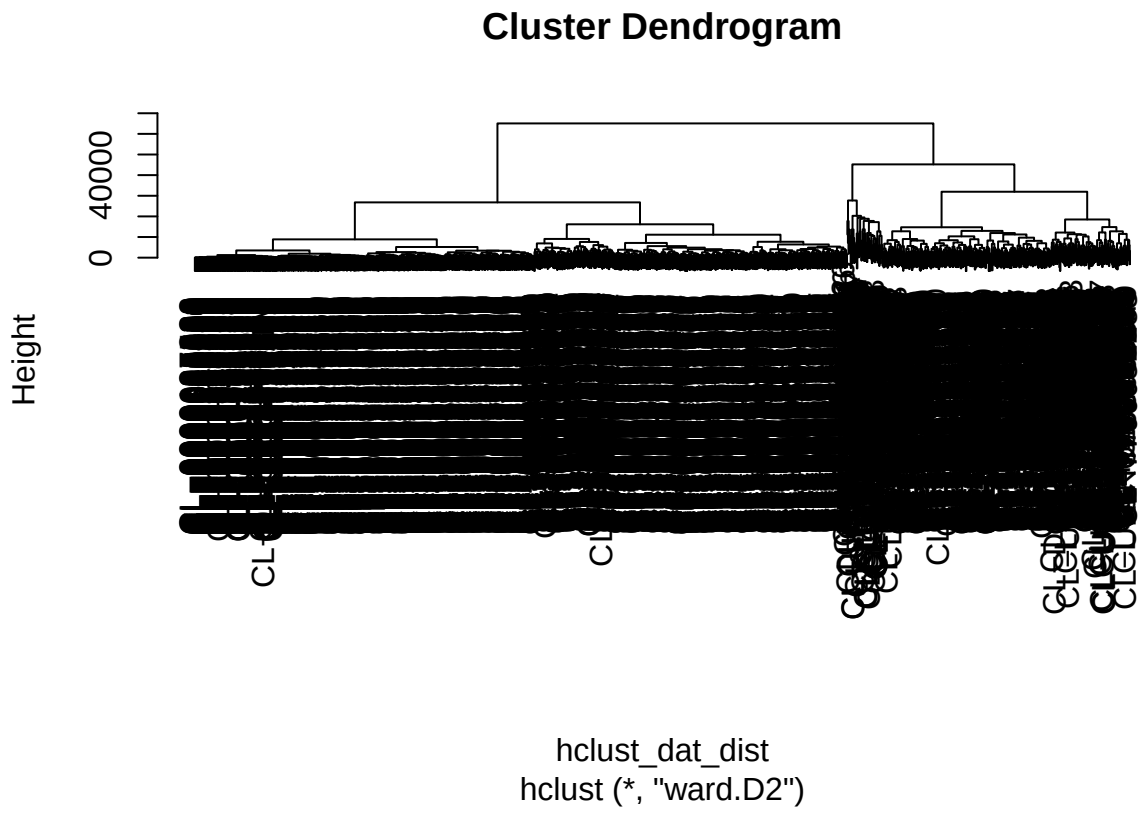
The Ward.D2 method of hierarchical clustering is recommended for correlations.

```
GetClusterAssignment_hclust <- function(data_obj,
                                        cluster_qty,
                                        IsPCA             = TRUE,
                                        InvertForDistance = TRUE,
                                        SquareTheDistance = TRUE,
                                        clust_method      = "ward.D2",
                                        dist_method       = "euclidean"){
  if(IsPCA){
    hclust_dat_scale <- data_obj$x
    names <- attr(data_obj$x,"dimnames")[[1]]
  }else{
    hclust_dat_scale <- data_obj[,-1] #first column is PART_NUMBER
    names <- data_obj[,1]
  }

  if(InvertForDistance){
    hclust_dat_dist  <- dist(1-hclust_dat_scale, method=dist_method)
  }else{
    hclust_dat_dist  <- dist(hclust_dat_scale, method=dist_method)
  }

  if(SquareTheDistance){
    hclust_dat_dist <- hclust_dat_dist*hclust_dat_dist
  }

  hclust_obj       <- hclust(hclust_dat_dist, method=clust_method)
  hclust_obj_cut   <- cutree(hclust_obj, k=cluster_qty)

  cluster_assignment_hclust <- cbind.data.frame(PART_NUMBER = names,
                                                CLUSTER = as.vector(hclust_obj_cut),
                                                stringsAsFactors = F)

  return(list(clust_assign=cluster_assignment_hclust, hclust_obj=hclust_obj))
}
```

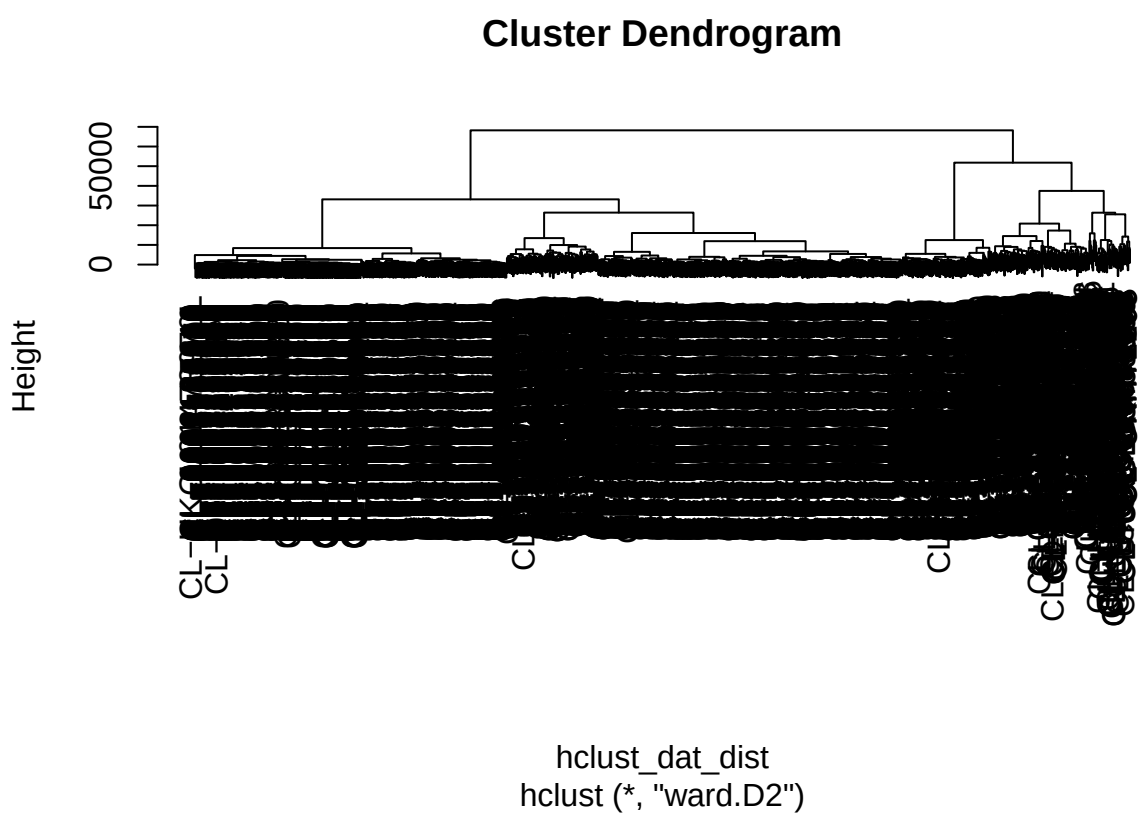
```
cluster_assignment_hclust_v1a <- GetClusterAssignment_hclust(data_obj      = pca_obj_v1_trend,
                                                             cluster_qty   = 50)

cluster_assignment_hclust_v1b <- GetClusterAssignment_hclust(data_obj      = pca_obj_v1_raw,
                                                             cluster_qty   = 50)

cluster_assignment_hclust_v1c <- GetClusterAssignment_hclust(data_obj      = pca_obj_v1_rawlog,
                                                             cluster_qty   = 50)
```

```
cluster_assignment_hclust_v1d <- GetClusterAssignment_hclust(data_obj      = pca_obj_v1_rawds,
                                                             cluster_qty   = 50)


cluster_assignment_hclust_v2a <- GetClusterAssignment_hclust(data_obj      = pca_obj_v2_trend,
                                                             cluster_qty   = 50)

cluster_assignment_hclust_v2b <- GetClusterAssignment_hclust(data_obj      = pca_obj_v2_raw,
                                                             cluster_qty   = 50)

cluster_assignment_hclust_v2c <- GetClusterAssignment_hclust(data_obj      = pca_obj_v2_rawlog,
                                                             cluster_qty   = 50)

cluster_assignment_hclust_v2d <- GetClusterAssignment_hclust(data_obj      = pca_obj_v2_trendlog,
                                                             cluster_qty   = 50)
```

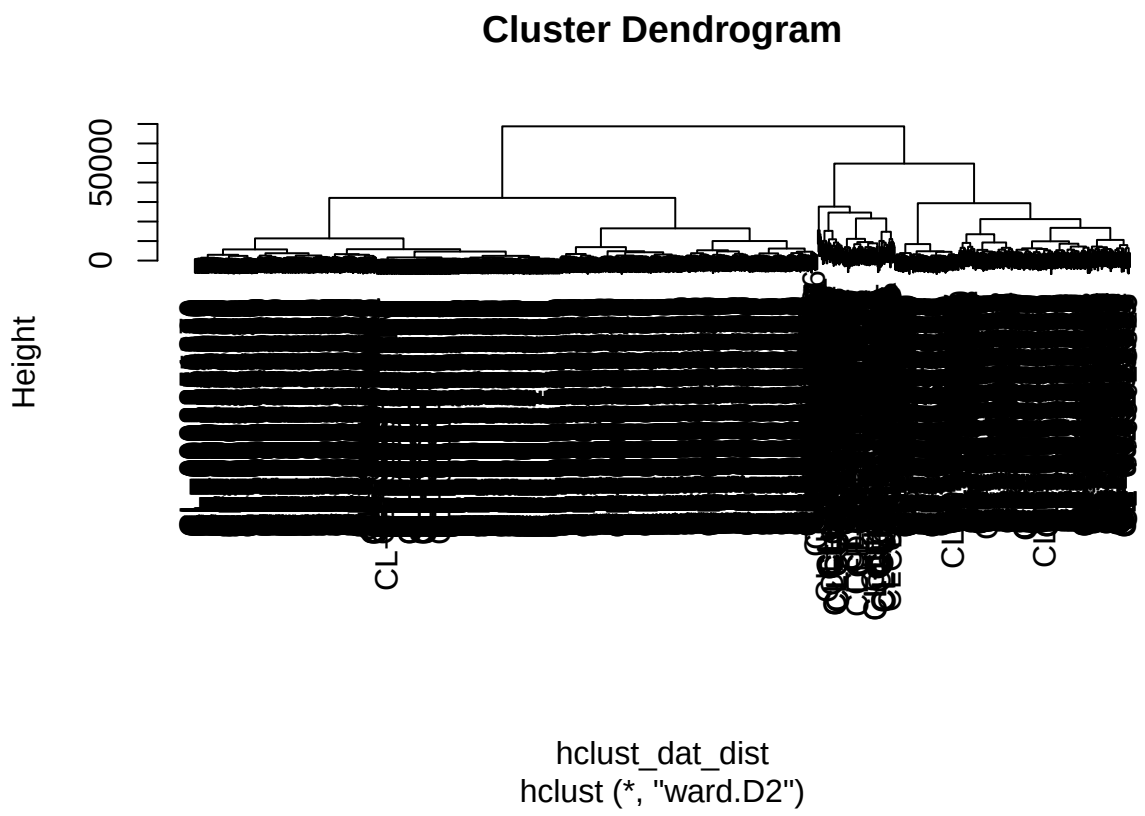
```
plot(cluster_assignment_hclust_v1a[[2]])
```



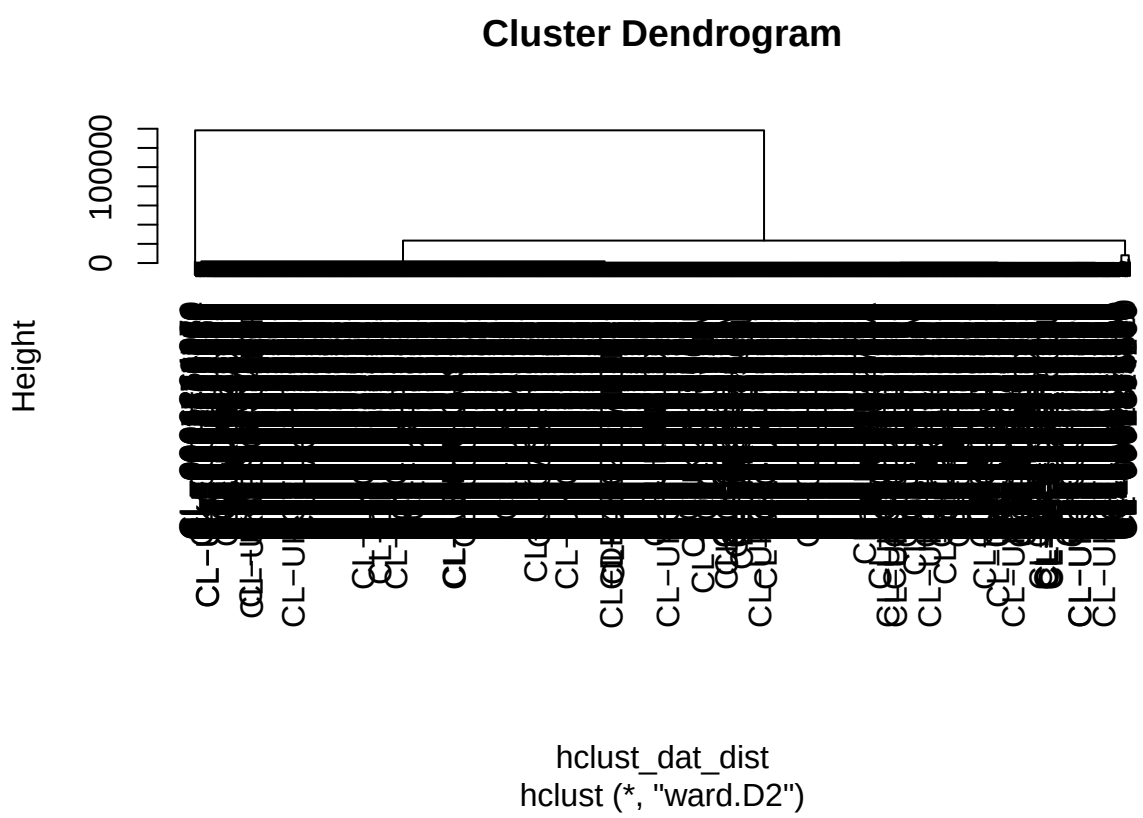

```
plot(cluster_assignment_hclust_v1b[[2]])
plot(cluster_assignment_hclust_v1c[[2]])
```



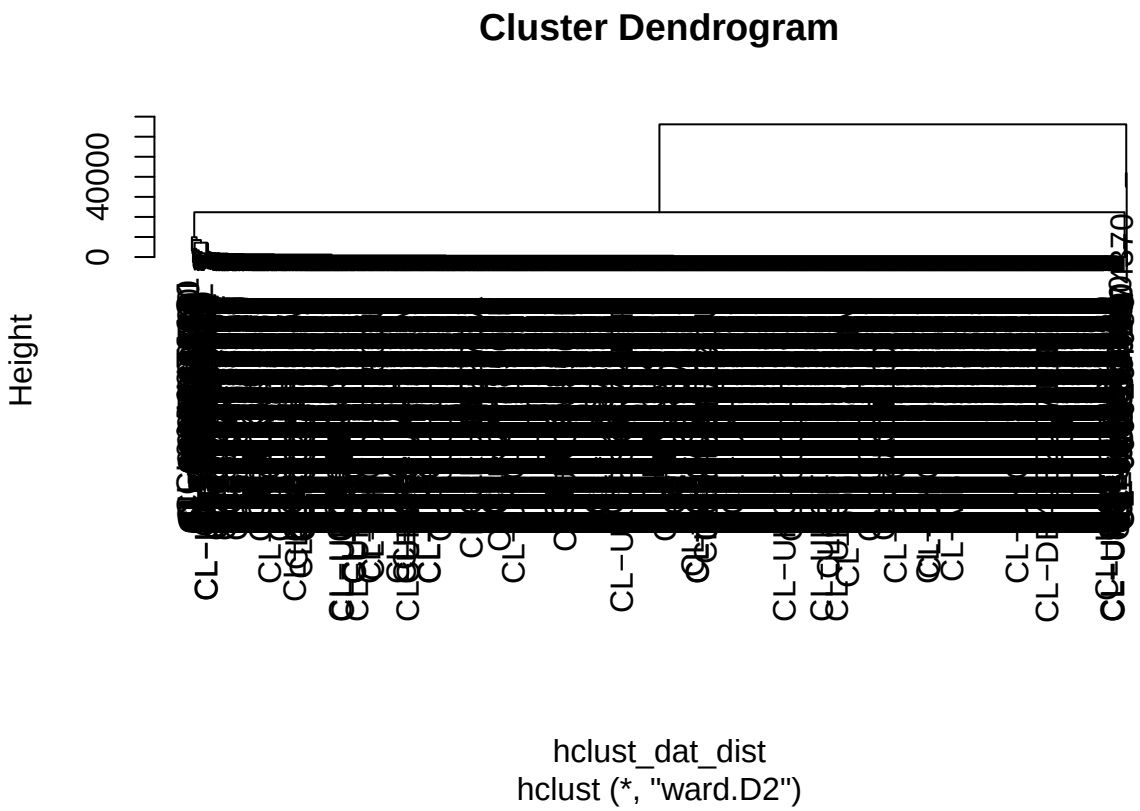

```
plot(cluster_assignment_hclust_v1d[[2]])
```



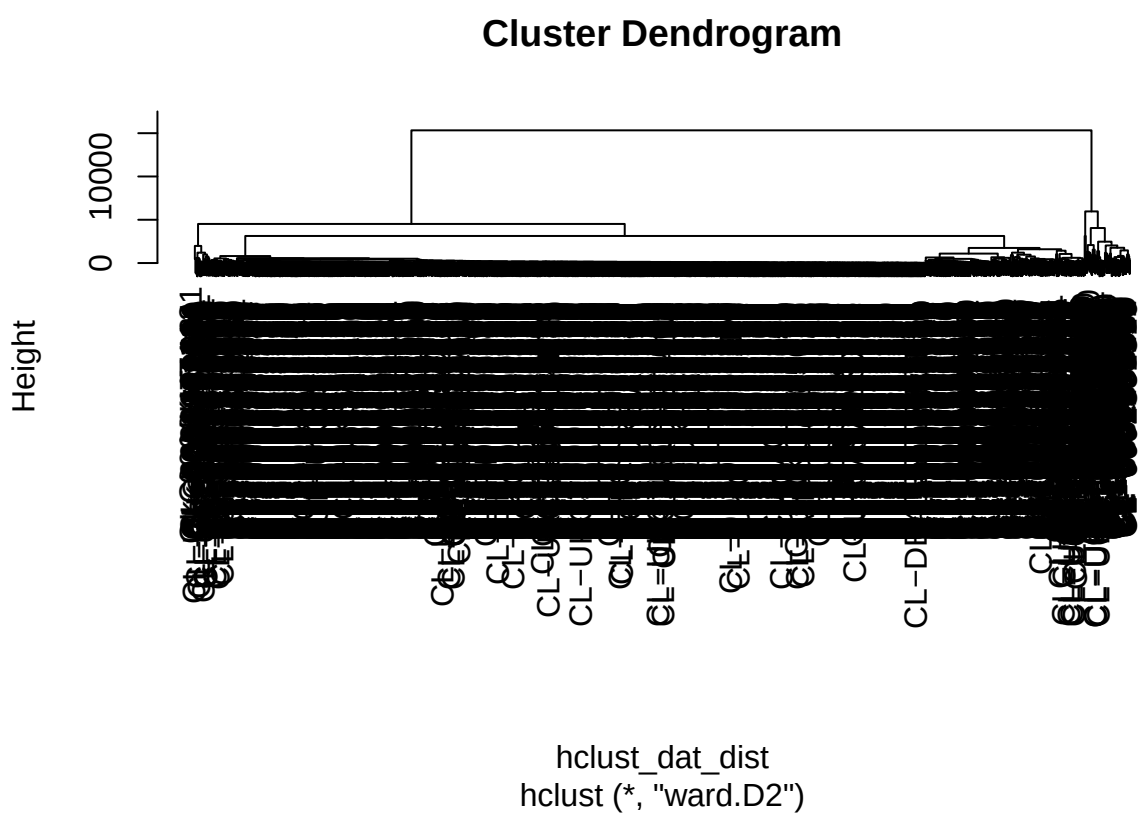

```
plot(cluster_assignment_hclust_v2a[[2]])
```



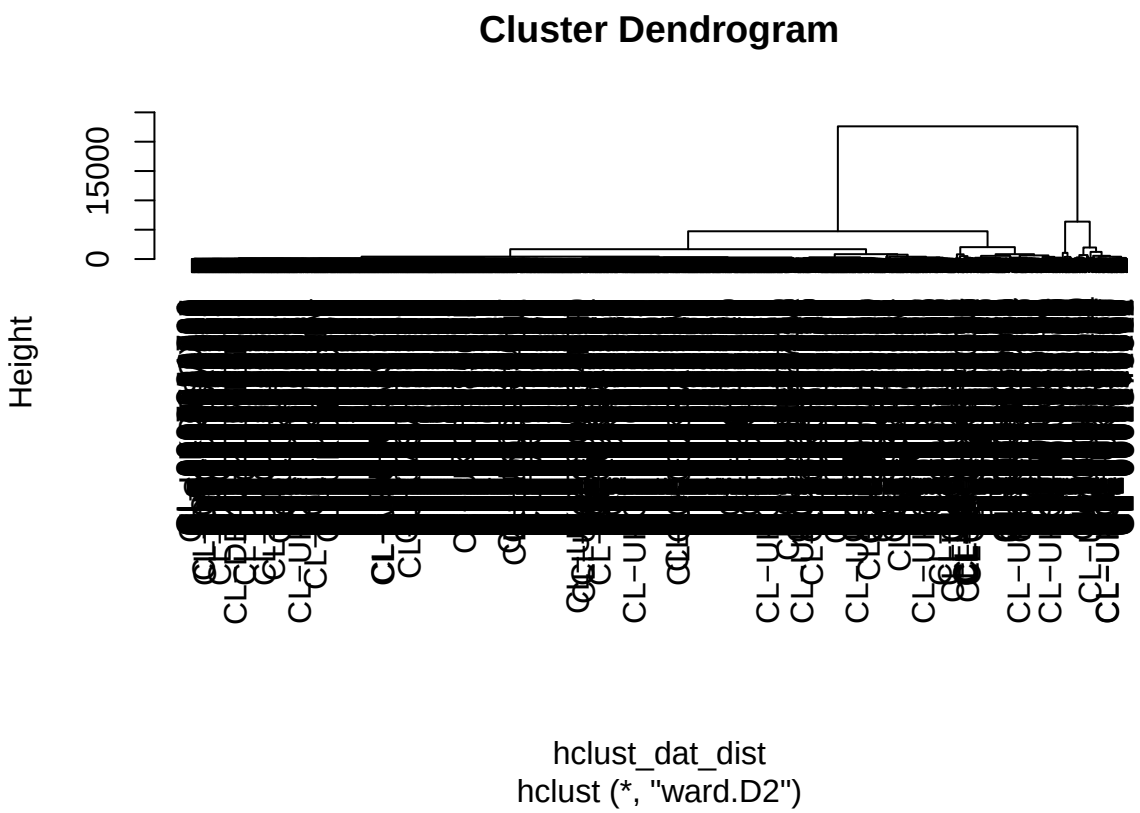

```
plot(cluster_assignment_hclust_v2b[[2]])
```




```
plot(cluster_assignment_hclust_v2c[[2]])
```




```
plot(cluster_assignment_hclust_v2d[[2]])
```



The dendrograms are substantially different for each type of analysis, they can easily be used to identify a comparable number of clusters to our original approach.

We can drop DBscan clustering from our analysis, it just doesn't identify enough clusters, perhaps the data density is too homogenous. So, we need only compare the three data types with the two clustering types, 6 scenarios in total.

So let's build a function to compare the scenarios, to see which delivers the best correlation within each group (i.e. highest mean correlation of groups).

```
getMeanCorrel <- function(crosscorrel_data, cluster_assignment, returnMeanCorels=FALSE){

  mean_correls <- crosscorrel_data %>%
                    inner_join(y=cluster_assignment, by=c("PART_NUMBER.x" = "PART_NUMBER")) %>%
                      rename(CLUSTER.x=CLUSTER) %>%
                        inner_join(y=cluster_assignment, by=c("PART_NUMBER.y" = "PART_NUMBER")) %>%
                          rename(CLUSTER.y=CLUSTER) %>%
                            filter(CLUSTER.x==CLUSTER.y) %>%
                              group_by(CLUSTER.x) %>%
                                rename(CLUSTER=CLUSTER.x) %>%
                                summarise(Mean_Correl_WithLag=mean(abs(ccor_with_lag), na.rm=T),
                                          Mean_Correl_NoLag=mean(abs(ccor_no_lag), na.rm=T))

  parts_per_cluster <- crosscorrel_data %>%
                        distinct(PART_NUMBER.x)%>%
                          inner_join(y=cluster_assignment, by=c("PART_NUMBER.x" = "PART_NUMBER"))%>%
                            group_by(CLUSTER) %>%
                              summarise(parts_per_cluster=n())
```

```
  Mean_Correl_WithLag  <- round(mean(mean_correls$Mean_Correl_WithLag, na.rm=T),3)
  Mean_Correl_NoLag    <- round(mean(mean_correls$Mean_Correl_NoLag, na.rm=T),3)
  MedianPartsPerCluster<- median(parts_per_cluster$parts_per_cluster)

  if(returnMeanCorels){
    return(mean_correls %>% inner_join(parts_per_cluster, by="CLUSTER"))
  }else{
    return(c(Mean_Correl_WithLag, Mean_Correl_NoLag, MedianPartsPerCluster))
  }
}
```

```
#Let's build a table and fill it with all the results so far
ClustPerf <- data.frame(SourceData         = character(),
                        ClusteringMethod   = character(),
                        Mean_CorrelWithLag = numeric(),
                        Mean_CorrelNoLag   = numeric(),
                        Median_PartsPerCluster = numeric(),
                        stringsAsFactors = F)

ClustPerf[ 1,]<-c("Trends", "Prefix",
                  round(mean(CorrelsByPrefix_trends$Mean_Correl_WithLag, na.rm=T),3),
                  round(mean(CorrelsByPrefix_trends$Mean_Correl_NoLag),3),
                  median(CorrelsByPrefix_trends$PartsInGrp))
ClustPerf[ 2,]<-c("Trends", "Random",
                  round(mean(CorrelsByPrefix_trends_rnd$Mean_Correl_WithLag, na.rm=T),3),
                  round(mean(CorrelsByPrefix_trends_rnd$Mean_Correl_NoLag),3),
                  median(CorrelsByPrefix_trends_rnd$PartsInGrp))
ClustPerf[ 3,]<-c("SalesQty", "Prefix",
                  round(mean(CorrelsByPrefix_raw$Mean_Correl_WithLag, na.rm=T),3),
                  round(mean(CorrelsByPrefix_raw$Mean_Correl_NoLag),3),
                  median(CorrelsByPrefix_raw$PartsInGrp))
ClustPerf[ 4,]<-c("SalesQty", "Random",
                  round(mean(CorrelsByPrefix_raw_rnd$Mean_Correl_WithLag, na.rm=T),3),
                  round(mean(CorrelsByPrefix_raw_rnd$Mean_Correl_NoLag),3),
                  median(CorrelsByPrefix_raw_rnd$PartsInGrp))
ClustPerf[ 5,]<-c("CorrelsOfTrends", "KMeans",
                  getMeanCorrel(crosscorrel_trends_unq,      cluster_assignment_kmeans_v1a[[1]]))
ClustPerf[ 6,]<-c("CorrelsOfTrends", "HClust",
                  getMeanCorrel(crosscorrel_trends_unq,      cluster_assignment_hclust_v1a[[1]]))
ClustPerf[ 7,]<-c("CorrelsOfSalesQty", "KMeans",
                  getMeanCorrel(crosscorrel_rawdata_unq,     cluster_assignment_kmeans_v1b[[1]]))
ClustPerf[ 8,]<-c("CorrelsOfSalesQty", "HClust",
                  getMeanCorrel(crosscorrel_rawdata_unq,     cluster_assignment_hclust_v1b[[1]]))
ClustPerf[ 9,]<-c("CorrelsOfLog(SalesQty+1)", "KMeans",
                  getMeanCorrel(crosscorrel_rawdata_log_unq, cluster_assignment_kmeans_v1c[[1]]))
ClustPerf[10,]<-c("CorrelsOfLog(SalesQty+1)", "HClust",
                  getMeanCorrel(crosscorrel_rawdata_log_unq, cluster_assignment_hclust_v1c[[1]]))
ClustPerf[11,]<-c("CorrelsOfSalesQtyDeseason", "KMeans",
                  getMeanCorrel(crosscorrel_rawdata_ds_unq,  cluster_assignment_kmeans_v1d[[1]]))
ClustPerf[12,]<-c("CorrelsOfSalesQtyDeseason", "HClust",
                  getMeanCorrel(crosscorrel_rawdata_ds_unq,  cluster_assignment_hclust_v1d[[1]]))
```

```
ClustPerf[13,]<-c("Trends", "KMeans",
                  getMeanCorrel(crosscorrel_trends_unq,     cluster_assignment_kmeans_v2a[[1]]))
ClustPerf[14,]<-c("Trends", "HClust",
                  getMeanCorrel(crosscorrel_trends_unq,     cluster_assignment_hclust_v2a[[1]]))
ClustPerf[15,]<-c("Sales Qty", "KMeans",
                  getMeanCorrel(crosscorrel_rawdata_unq,    cluster_assignment_kmeans_v2b[[1]]))
ClustPerf[16,]<-c("Sales Qty", "HClust",
                  getMeanCorrel(crosscorrel_rawdata_unq,    cluster_assignment_hclust_v2b[[1]]))
ClustPerf[17,]<-c("Log(Sales+1)", "KMeans",
                  getMeanCorrel(crosscorrel_rawdata_log_unq,cluster_assignment_kmeans_v2c[[1]]))
ClustPerf[18,]<-c("Log(Sales+1)", "HClust",
                  getMeanCorrel(crosscorrel_rawdata_log_unq,cluster_assignment_hclust_v2c[[1]]))
ClustPerf[19,]<-c("Log(Trend+1)", "KMeans",
                  getMeanCorrel(crosscorrel_trends_log_unq, cluster_assignment_kmeans_v2d[[1]]))
ClustPerf[20,]<-c("Log(Trend+1)", "HClust",
                  getMeanCorrel(crosscorrel_trends_log_unq, cluster_assignment_hclust_v2d[[1]]))
```

```
print(paste0("Number of parts in analysis: ",nrow(cluster_assignment_kmeans_v1a$clust_assign),
             cat("\n")))
```

```
##
## [1] "Number of parts in analysis: 1682"
```

```
print(paste0("Number of clusters: ",qty_grps_trends,cat("\n")))
```

```
##
## [1] "Number of clusters: 29"
```

```
print(paste0("Expected median number of parts per cluster: ",round(nrow(cluster_assignment_kmeans_v1a$cl
             cat("\n")))
```

```
##
## [1] "Expected median number of parts per cluster: 58"
```

```
kable((ClustPerf %>% arrange(desc(Mean_CorrelNoLag))))
```

SourceData	ClusteringMethod	Mean_CorrelWithLag	Mean_CorrelNoLag	Median_PartsPerClus
Sales Qty	HClust	0.689	0.669	1
Log(Sales+1)	HClust	0.62	0.606	2
Sales Qty	KMeans	0.601	0.56	2
Wavelets Trends	HClust	0.548	0.556	1
Trends	HClust	0.574	0.553	1
CorrelsOfTrends	HClust	0.613	0.545	5
Log(Trend+1)	HClust	0.565	0.5	4.5
Trends	KMeans	0.53	0.46	13
CorrelsOfTrends	KMeans	0.527	0.458	24.5
Log(Sales+1)	KMeans	0.475	0.443	11.5
Wavelets log(sales qty+1)	Kmeans	0.484	0.439	8
Log(Trend+1)	KMeans	0.497	0.42	27.5
DTW Clustering, Trend	Partitional	0.386	0.33	30
Trends	Prefix	0.331	0.311	23
CorrelsOfSalesQtyDeseason	HClust	0.348	0.283	5.5
CorrelsOfSalesQty	KMeans	0.359	0.274	24.5
Trends	Random	0.318	0.271	20
CorrelsOfSalesQty	HClust	0.366	0.266	7
DTW Clustering, Raw	Partitional	0.355	0.266	15.5

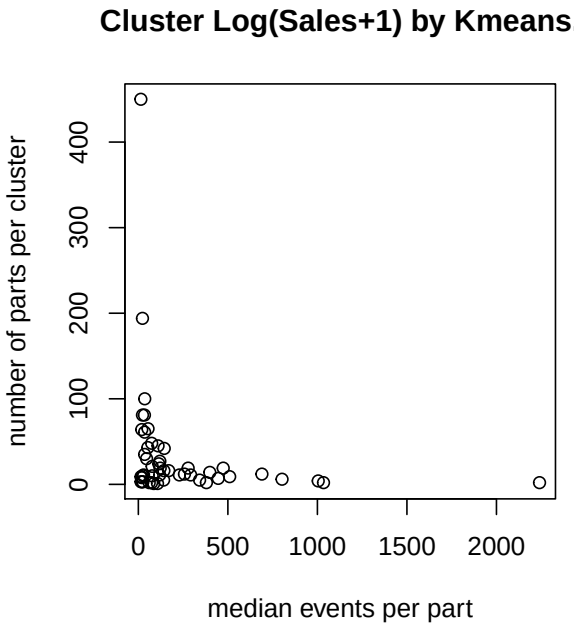
SourceData	ClusteringMethod	Mean_CorrelWithLag	Mean_CorrelNoLag	Median_PartsPerClus
CorrelsOfSalesQtyDeseason	KMeans	0.327	0.262	30
CorrelsOfLog(SalesQty+1)	HClust	0.332	0.258	7
CorrelsOfLog(SalesQty+1)	KMeans	0.316	0.248	25.5
Word2Vec	Kmeans	0.328	0.238	22
SalesQty	Prefix	0.295	0.185	23
DTW Clustering, Raw	Hierarchical	0.336	0.18	5
SalesQty	Random	0.268	0.138	22.5
We need a clustering solutio	n with high mean co	rrelation between its	parts and where th	e median number of p

Our top candidates are: - CorrelsOfTrends, clustering with Kmeans. - Log(Sales+1), clustering with Kmeans.

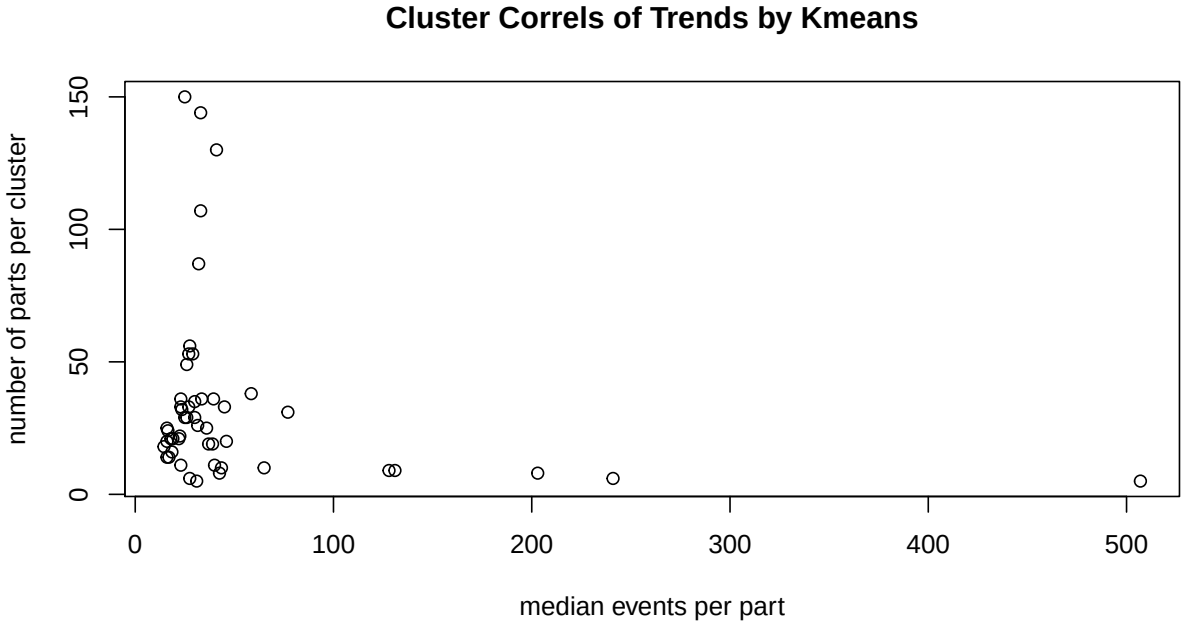
What we'd like is a clustering method that clusters parts together which have few sales events, thus increasing the number of events in the cluster. Clusters that have few members should be clusters where the number of events per part is high. ie we don't mind if the cluster is small, we already have enough data to work with.

So, let's plot the number of parts per cluster vs the median number of events per part number in the cluster. We'd like to see a line faling from left to right, fewer parts per cluster meaning more events per part. We'll do this for both the candidates.

```
plotdat_log_kmean <- parts_frequent %>%
                         inner_join(cluster_assignment_kmeans_v2c[[1]], by="PART_NUMBER") %>%
                           group_by(CLUSTER) %>%
                             summarise(partnums=n(), median_events = median(TOTAL_SALES_EVENTS))

plotdat_log_kmean_b<- plotdat_log_kmean %>% select(x=median_events, y=partnums)

plotdat_corrtrend_kmean <- parts_frequent %>%
                         inner_join(cluster_assignment_kmeans_v1a[[1]], by="PART_NUMBER") %>%
                           group_by(CLUSTER) %>%
                             summarise(partnums=n(), median_events = median(TOTAL_SALES_EVENTS))

plotdat_corrtrend_kmean_b<- plotdat_corrtrend_kmean %>% select(x=median_events, y=partnums)

par(mfrow=c(1,2))
plot(plotdat_log_kmean_b,
     xlab="median events per part", ylab="number of parts per cluster",
     main="Cluster Log(Sales+1) by Kmeans.")
```




```
plot(plotdat_corrtrend_kmean_b,
     xlab="median events per part", ylab="number of parts per cluster",
     main="Cluster Correls of Trends by Kmeans")
```



Noth methods give us the distribution we are looking for. Since trends (i.e. 12mth moving averages) are our core interest and the mean correlation is higher, it appears that clustering the correlation of trends with Kmeans is the most attractive method so far.

Now we'll attempt some more esoteric feature building and clustering algorithms.

Feature Extraction With Discrete Wavelet Transforms (DWT)

Features of Time series are often analysed using fourier transforms (see the 'fft' package), but we have only 117 months in our series. Not much for analysing underlying frequencies.Instead, we will use wavelets, i.e. discrete wavelet transforms. DWT may be considered superior to Fourier Transform in that it captures both frequency and sequence information (sequence = location in time).

First get all time series onto the same date range and presented as ts objects.

```
# this function works, but in hindsight we could have used ts.union() and then replace NA with 0

get_allsameperiod_ts <- function(parts_list, parts_sales_series,
                                 months_list=monthslist, NAtoZero=TRUE){

  # instantiate return
  allsameperiod_ts <- list(PART_NUMBER=character(), TIME_SERIES=list())

  pb <- txtProgressBar(min = 1, max = nrow(parts_list), style = 3)

  #loop thru list of parts, one at a time
  for(i in 1:nrow(parts_list)){

      setTxtProgressBar(pb, i)

      #get current part_number
      part_number_current <- parts_list[i,1][[1]]
      part_number_scope <<- part_number_current

      test<<-parts_sales_series %>% filter(PART_NUMBER == part_number_current)

      #if there is data for the part_number, then begin convert process
      if(nrow(parts_sales_series %>% filter(PART_NUMBER == part_number_current))>0)
      {

        #Convert data into a time series object
        timeseries_ts <- getts(part_number = part_number_current,
                               all_series  = parts_sales_series,
                               lag         = 0,
                               months_list = months_list,
                               NAtoZero    = NAtoZero,
                               all_to_same_period = TRUE)

        allsameperiod_ts$PART_NUMBER[[i]] <- part_number_current
        allsameperiod_ts$TIME_SERIES[[i]] <- timeseries_ts
      }
  }
  return(allsameperiod_ts)
}
```

```
#note, we log the sales_qty then seek wavelets.
#This is because logs have been more successful in previous clustering

allsameperiod_ts <- get_allsameperiod_ts(partnumbers_top_pc,
                                         series_top_pc %>% mutate(SALES_QTY = log(SALES_QTY+1)),
                                         NAtoZero = T)
```

Now we're ready to use extract wavelets

```
#with thanks to R Data Mining http://www.rdatamining.com/examples/time-series-clustering-classification
library(wavelets)

get_wavelets <- function(parts_series_ts){
  wtData <- NULL

  for (i in 1:length(parts_series_ts[[2]])) {
    #transform
    a  <- parts_series_ts[[2]][[i]]

    #get wavelets coefficients
    wt <- dwt(a, filter="haar", boundary="periodic")
    wtData <- rbind(wtData, unlist(c(wt@W,wt@V[[wt@level]])))
  }

  wtData <- as.data.frame(wtData)
  rownames(wtData) <- parts_series_ts[[1]]
  return(wtData)
}
```

```
wtData <- get_wavelets(allsameperiod_ts)

pca_wavelets_list     <- getPCA(wtData, DisplayPlot=FALSE, setRownames = FALSE)
pca_wavelets_pcaobj   <- pca_wavelets_list$pca_obj
pca_wavelets_pcadata  <- pca_wavelets_list$pca_data
pca_wavelets_outliers <- pca_wavelets_list$outliers
rm(pca_wavelets_list)

#now move to PCA
threshold_dim_wavelet <- getThresholdDim(pca_obj        = pca_wavelets_pcaobj,
                                         approach       = "Wavelets",
                                         DisplaySummary = TRUE)
```

```
threshold_dim_wavelet
```

```
## [1] 112
```

Not much benefit in PCA, but the pca object is useful for the kmeans function we already have. Let's see those clusters...

```
wavelet_clusters <- GetClusterAssignment_PcaAndKmeans(pca_obj       = pca_wavelets_pcaobj,
                                                      cluster_qty   = 50,
                                                      threshold_dim = threshold_dim_wavelet,
                                                      iter.max      = 50)

wavelet_clusters_simple <- wavelet_clusters[[1]]

plot3Dcluster(pca_obj            = pca_wavelets_pcaobj,
              cluster_assignment = wavelet_clusters,
              approach           = "Wavelets")

meancorrel_wavelets <- getMeanCorrel(crosscorrel_rawdata_log_unq, wavelet_clusters_simple)
ClustPerf[21,]      <- c("Wavelets log(sales qty+1)","Kmeans",meancorrel_wavelets)
```

How do wavelets based on log(sales qty+1) score relative to the other clustering methods?

```
kable((ClustPerf %>% arrange(desc(Mean_CorrelNoLag))))
```

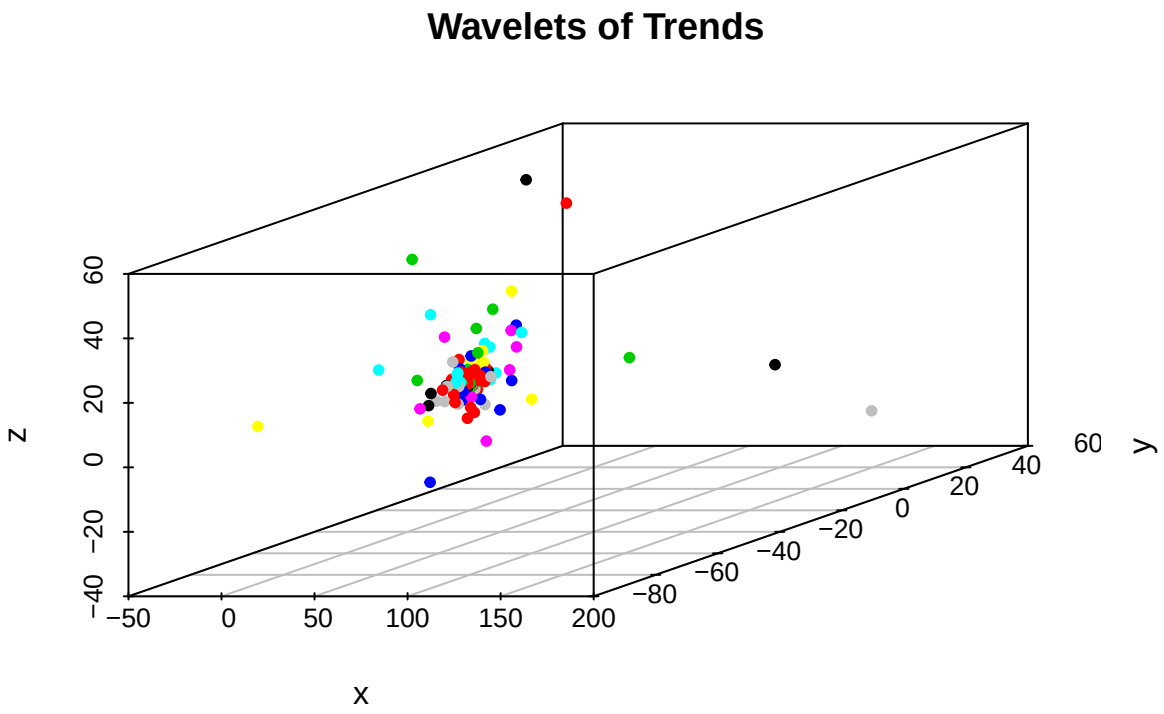
SourceData	ClusteringMethod	Mean_CorrelWithLag	Mean_CorrelNoLag	Median_PartsPerCluster
Sales Qty	HClust	0.689	0.669	1
Log(Sales+1)	HClust	0.62	0.606	2
Sales Qty	KMeans	0.601	0.56	2
Wavelets Trends	HClust	0.548	0.556	1
Trends	HClust	0.574	0.553	1
CorrelsOfTrends	HClust	0.613	0.545	5
Log(Trend+1)	HClust	0.565	0.5	4.5
Trends	KMeans	0.53	0.46	13
CorrelsOfTrends	KMeans	0.527	0.458	24.5
Log(Sales+1)	KMeans	0.475	0.443	11.5
Wavelets log(sales qty+1)	Kmeans	0.484	0.439	8
Log(Trend+1)	KMeans	0.497	0.42	27.5
DTW Clustering, Trend	Partitional	0.386	0.33	30
Trends	Prefix	0.331	0.311	23
CorrelsOfSalesQtyDeseason	HClust	0.348	0.283	5.5
CorrelsOfSalesQty	KMeans	0.359	0.274	24.5
Trends	Random	0.318	0.271	20
CorrelsOfSalesQty	HClust	0.366	0.266	7
DTW Clustering, Raw	Partitional	0.355	0.266	15.5
CorrelsOfSalesQtyDeseason	KMeans	0.327	0.262	30
CorrelsOfLog(SalesQty+1)	HClust	0.332	0.258	7
CorrelsOfLog(SalesQty+1)	KMeans	0.316	0.248	25.5
Word2Vec	Kmeans	0.328	0.238	22
SalesQty	Prefix	0.295	0.185	23
DTW Clustering, Raw	Hierarchical	0.336	0.18	5
SalesQty	Random	0.268	0.138	22.5

Mean correlation within clusters approx 0.4, not so great! Let's revisit the wavelets, but this time on trends...

```
parts_trends_as_sales <-
        selected_series_withNAs %>%
          select(PART_NUMBER, INVOICE_MONTH, TREND) %>%
              mutate(INVOICE_MONTH = eomonth(INVOICE_MONTH), SALES_QTY = TREND)

allsameperiod_ts_trends <- get_allsameperiod_ts(
                              partnumbers_top_pc,
                              parts_trends_as_sales,
                              NAtoZero = TRUE)

wtData_trends           <- get_wavelets(allsameperiod_ts_trends)

pca_wavelets_list_trends     <- getPCA(wtData_trends, DisplayPlot=TRUE, setRownames = FALSE)
pca_wavelets_pcaobj_trends   <- pca_wavelets_list_trends$pca_obj
pca_wavelets_pcadata_trends  <- pca_wavelets_list_trends$pca_data
pca_wavelets_outliers_trends <- pca_wavelets_list_trends$outliers
rm(pca_wavelets_list_trends)

threshold_dim <- getThresholdDim(pca_obj = pca_wavelets_pcaobj_trends,
                                 approach = "Wavelets of Trends",
```

```
                                DisplaySummary = T)

wavelet_clusters_trends <- GetClusterAssignment_hclust(data_obj          = wtData_trends,
                                                       cluster_qty       = 50,
                                                       IsPCA             = FALSE,
                                                       InvertForDistance = FALSE,
                                                       SquareTheDistance = TRUE)

wavelet_clusters_trends_simple <- data.frame(PART_NUMBER = wavelet_clusters_trends[[2]]$labels,
                                             CLUSTER = wavelet_clusters_trends[[1]]$CLUSTER,
                                             stringsAsFactors = F)

meancorrel_wavelets_trends <- getMeanCorrel(crosscorrel_trends_unq, wavelet_clusters_trends_simple)
```

```
plot3Dcluster(pca_obj = pca_wavelets_pcaobj_trends,
              cluster_assignment = wavelet_clusters_trends,
              approach = "Wavelets of Trends")
```



Add the result to the table comparing clustering methods...

```
ClustPerf[22,] <- c("Wavelets Trends","HClust", meancorrel_wavelets_trends)
```

Well compare results in a moment, now to explore the sibling of wavelets, Dynamic Time Warping...

Dynamic Time Warping for Clustering

There is also a package for clustering time series data using Dynamic Time Warping (DTW) DTW is not the same as wavelets, so is valid as an alternative clustering method. https://github.com/asardaes/dtwclust

```
#Using the 'dtwclust' package. See https://github.com/asardaes/dtwclust
library(dtwclust)

#dtwclust reads time series from a list of vectors, each vector being the time series.
allsameperiod_lv <- lapply(allsameperiod_ts[[2]], as.numeric)
attributes(allsameperiod_lv)$names <- allsameperiod_ts[[1]]

# Features from Dynamic Time Warping (DTW), Clustering using partitioning.
dtwclust_raw_partitional   <- tsclust(allsameperiod_lv,
                                      type = "partitional",
                                      k = 50L,
                                      distance = "dtw_basic",
                                      centroid = "pam",
                                      seed = 201807L,
                                      trace = TRUE,
                                      args = tsclust_args(dist = list(window.size = 50L)))

# Features from Dynamics Time Warping (DTW), Clustering using HClust
dtwclust_raw_hierarchical  <- tsclust(allsameperiod_lv,
                                      type = "hierarchical",
                                      k = 50L,
                                      distance = "sbd",
                                      trace = TRUE,
                                      control  = hierarchical_control(method = "average"))
```

Let's repeat the process, but this time using the trend data, not the raw sales.

```
allsameperiod_trends_lv <- lapply(allsameperiod_ts_trends[[2]], as.numeric)
attributes(allsameperiod_lv)$names <- allsameperiod_ts_trends[[1]]

dtwclust_trend_partitional <- tsclust(allsameperiod_trends_lv,
                                      type = "partitional",
                                      k = 50L,
                                      distance = "dtw_basic",
                                      centroid = "pam",
                                      seed = 201807L,
                                      trace = TRUE,
                                      args = tsclust_args(dist = list(window.size = 50L)))
```

Let's summarise the results and present them alongside the previous analysis.

```
dtwclust_raw_partitional_   <- data.frame(
                                PART_NUMBER = allsameperiod_ts[[1]], stringsAsFactors=F,
                                CLUSTER     = attributes(dtwclust_raw_partitional)$cluster
                                )
dtwclust_trend_partitional_ <- data.frame(
                                PART_NUMBER = allsameperiod_ts_trends[[1]], stringsAsFactors=F,
                                CLUSTER     = attributes(dtwclust_trend_partitional)$cluster
                                )
dtwclust_raw_hierarchical_  <- data.frame(
                                PART_NUMBER = allsameperiod_ts[[1]], stringsAsFactors=F,
```

```
                                  CLUSTER     = attributes(dtwclust_raw_hierarchical)$cluster
                                  )


meancorrel_dtwclust_raw_part   <- getMeanCorrel(crosscorrel_rawdata_unq,
                                                dtwclust_raw_partitional_)
meancorrel_dtwclust_trend_part <- getMeanCorrel(crosscorrel_trends_unq ,
                                                dtwclust_trend_partitional_)
meancorrel_dtwclust_raw_hier   <- getMeanCorrel(crosscorrel_rawdata_unq,
                                                dtwclust_raw_hierarchical_)


ClustPerf[23,] <- c("DTW Clustering, Raw",  "Partitional",  meancorrel_dtwclust_raw_part)
ClustPerf[24,] <- c("DTW Clustering, Trend","Partitional",  meancorrel_dtwclust_trend_part)
ClustPerf[25,] <- c("DTW Clustering, Raw",  "Hierarchical", meancorrel_dtwclust_raw_hier)
```

```
kable((ClustPerf %>% arrange(desc(Mean_CorrelWithLag))))
```

SourceData	ClusteringMethod	Mean_CorrelWithLag	Mean_CorrelNoLag	Median_PartsPerCluster
Sales Qty	HClust	0.689	0.669	1
Log(Sales+1)	HClust	0.62	0.606	2
CorrelsOfTrends	HClust	0.613	0.545	5
Sales Qty	KMeans	0.601	0.56	2
Trends	HClust	0.574	0.553	1
Log(Trend+1)	HClust	0.565	0.5	4.5
Wavelets Trends	HClust	0.548	0.556	1
Trends	KMeans	0.53	0.46	13
CorrelsOfTrends	KMeans	0.527	0.458	24.5
Log(Trend+1)	KMeans	0.497	0.42	27.5
Wavelets log(sales qty+1)	Kmeans	0.484	0.439	8
Log(Sales+1)	KMeans	0.475	0.443	11.5
DTW Clustering, Trend	Partitional	0.386	0.33	30
CorrelsOfSalesQty	HClust	0.366	0.266	7
CorrelsOfSalesQty	KMeans	0.359	0.274	24.5
DTW Clustering, Raw	Partitional	0.355	0.266	15.5
CorrelsOfSalesQtyDeseason	HClust	0.348	0.283	5.5
DTW Clustering, Raw	Hierarchical	0.336	0.18	5
CorrelsOfLog(SalesQty+1)	HClust	0.332	0.258	7
Trends	Prefix	0.331	0.311	23
Word2Vec	Kmeans	0.328	0.238	22
CorrelsOfSalesQtyDeseason	KMeans	0.327	0.262	30
Trends	Random	0.318	0.271	20
CorrelsOfLog(SalesQty+1)	KMeans	0.316	0.248	25.5
SalesQty	Prefix	0.295	0.185	23
SalesQty	Random	0.268	0.138	22.5

Neither wavelets nor DTW improve upon our existing clustering choices

Selected data and method to conclude clustering

The clustering data chosen is : Correlations of Trends, via Kmeans High mean correlation, hi median parts per cluster

The clustering method chosen is: KMeans

How many clusters to use?

The use of twenty clusters comes from the prefixes, based on the first 4 digits of a part number. This is rather arbitrary. What is the best number of clusters to use?

```
ClustSizes <- data.frame(QtyOfClusters          = numeric(),
                         Mean_CorrelWithLag     = numeric(),
                         Mean_CorrelNoLag       = numeric(),
                         Median_PartsPerCluster = numeric(),
                         stringsAsFactors = F)

getClusterDetails <- function(pca_obj, threshold_dim, QtyOfClusters, crosscorrel_data){

  cluster_assignment  <- GetClusterAssignment_PcaAndKmeans(pca_obj,
                                                           threshold_dim,
                                                           QtyOfClusters)

  return(c(QtyOfClusters, getMeanCorrel(crosscorrel_data,cluster_assignment[[1]])))
}

for(QtyOfClusters in seq(from=2 , to=100, by=2)){
  ClustSizes[QtyOfClusters/2,] <- getClusterDetails(pca_obj_v1_trend,
                                                    threshold_dim_v1a,
                                                    QtyOfClusters,
                                                    crosscorrel_trends_unq)
}
```

```
plot(x=ClustSizes$QtyOfClusters, y=ClustSizes$Mean_CorrelNoLag)
```

Clusters of 15 members

Aproximately 40 members per cluster leads us to 50 clusters, which is the number we have already used. Its close enough to the knee, at 30 clusters, that is seems reasonable to keep this figure.

Deep Learning Tools: T-SNE

Clustering using these analystical methods has not proven very successful. We can apply a deep learning tool, TNE, to see if we can visualise clusters. Are we missing some pattern that our analysis has failed to find?

TSNE is a process for finding the best way to plot high dimensionsal data on a 2D or 3D chart. 'Best' meaning most equally spread out. If there are clusters then they are often made visible by this process, if not then we see a tidy circle of points as tsne strives to plot points 'equidistant' from each other. A quick guide to the usage of T-SNE can be found at: https://distill.pub/2016/misread-tsne/

T-SNE will be applied to: Raw data, both sales and trends in sales DWT (dynamic wave transform features), both of sales and of trends Correlation space, both correlations in sales and correlations in trends

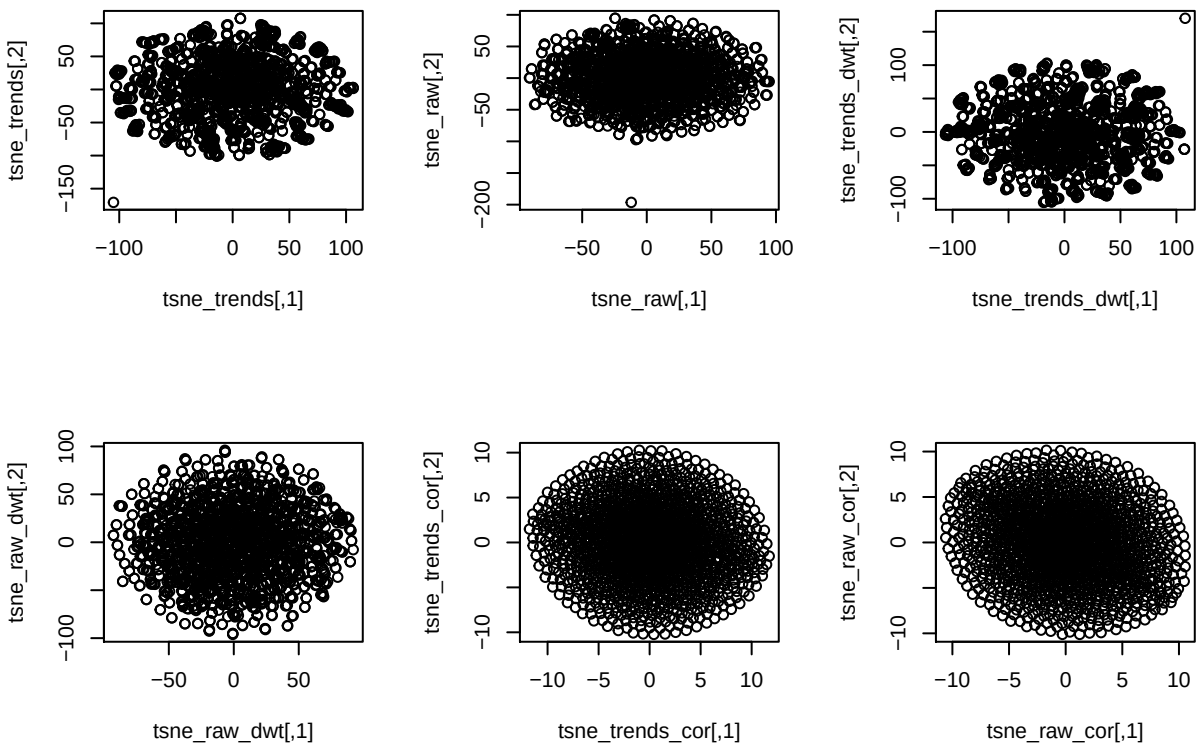
```
library(tsne)
library(grid)
library(gridExtra)
library(base2grob)
```

```
## Warning: package 'base2grob' was built under R version 3.5.1
```

```
get_tsne <- function(X, k, perplexity, initial_dims, epoch_callback=NULL){
  # k = target number of dimensions for plotting (usually 2 or 3)
  # perplexity = target number of neighbours. Lower = more segmented, more spread out.
 return( tsne(X              = X,
              initial_dims   = initial_dims,
              perplexity     = perplexity,
              k              = k,
              epoch_callback = epoch_callback,
              epoch          = 100,
              initial_config = NULL,
              whiten         = TRUE,
              max_iter       = 1000,
              min_cost       = 0)
 )
}
```

```
# perplexity range should be 5 to 50
tsne_trends     <- get_tsne(X = t(as.data.frame(allsameperiod_trends_lv)),
                            k = 2, perplexity = 5, initial_dims = 120)
tsne_raw        <- get_tsne(X = t(as.data.frame(allsameperiod_lv)),
                            k = 2, perplexity = 5, initial_dims = 120)
tsne_trends_dwt <- get_tsne(X = wtData_trends,
                            k = 2, perplexity = 5, initial_dims = 120)
tsne_raw_dwt    <- get_tsne(X = wtData,
                            k = 2, perplexity = 5, initial_dims = 120)
tsne_trends_cor <- get_tsne(X = pcadata_with_outliers_v1_trend[,-1],
                            k = 2, perplexity = 15, initial_dims = 1684)
tsne_raw_cor    <- get_tsne(X = pcadata_with_outliers_v1_rawsales[,-1],
                            k = 2, perplexity = 15, initial_dims = 1684)
```

```
par(mfrow=c(2,3))
plot(tsne_trends)
plot(tsne_raw)
plot(tsne_trends_dwt)
plot(tsne_raw_dwt)
plot(tsne_trends_cor)
plot(tsne_raw_cor)
```



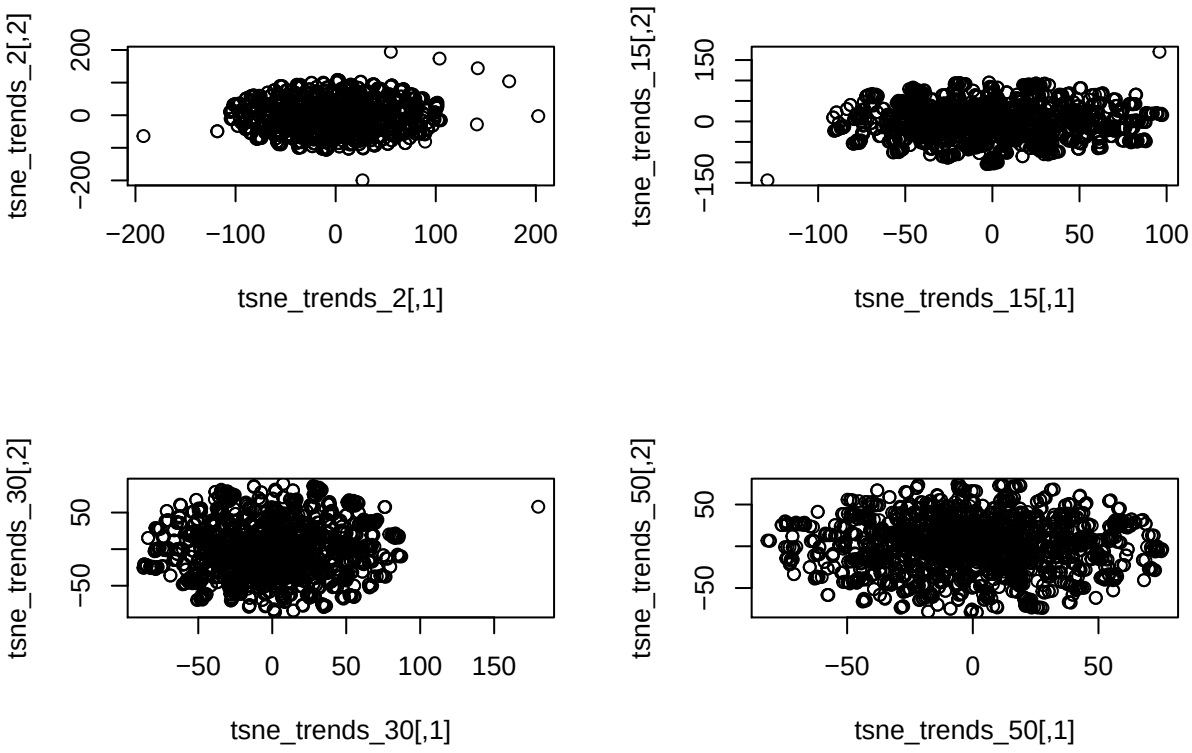
T-SNE Results

T-SNE attempts to put as much distance between points as possible. When it can find no pattern in the differences between points then the points are arranged equidistantly. This means, when there's no informaiton, it presents data in a uniform circle, which is what we see.

Let's vary perplexity for T_SNE on the trends, as this shows some signs of clustering.

```
tsne_trends_2   <- get_tsne(X = t(as.data.frame(allsameperiod_trends_lv)),
                            k = 2, perplexity = 2,  initial_dims = 120)
tsne_trends_15  <- get_tsne(X = t(as.data.frame(allsameperiod_trends_lv)),
                            k = 2, perplexity = 15, initial_dims = 120)
tsne_trends_30  <- get_tsne(X = t(as.data.frame(allsameperiod_trends_lv)),
                            k = 2, perplexity = 30, initial_dims = 120)
tsne_trends_50  <- get_tsne(X = t(as.data.frame(allsameperiod_trends_lv)),
                            k = 2, perplexity = 50, initial_dims = 120)
```

```
par(mfrow=c(2,2))
plot(tsne_trends_2)
plot(tsne_trends_15)
plot(tsne_trends_30)
plot(tsne_trends_50)
```



TSNE is simply confirming our previous difficulties in finding patterns in the data. We've one last experiment to try:

Deep Learning Tools: Embeddings as Features

This section is experimental! Not normally part of feature building.

Word2Vec is a well know deep learning tool whereby we select a target word and ask which words are typtically found near that target word (aka skip gram). This is the inverse of taking a sequence of words and asking what word would normally follow (Continuos Bag of Words, CBOW), which is common for text prediction.

The advantage of skip gram is that it gives us the relative context of a word. Using the distance from one word to a another in the text, the algorithm is able to assign a co-ordinate to each word, placing each word in a 'space' relative to other words. The vectors between co-ordinates contain meaning, for example, we find that the vector between 'man' and 'child' is similar to the vector between 'bull' and 'calf'. This co-ordinate is known as an embedding. When presenting words to models it is very useful to present the word's vector instead.

If we line all the sales of parts in a part, by date and by invoice sequence, then we have a kind of text where the part numbers are words. This sequence contains information about the parts which our model has not used so far. For example, filters are commonly sold with oil. That information, ie embedding, could be a useful feature for the models.

The following code chunks will derive those embeddings.

Convert part sales data in a 'text'

```
# Get the text, concatenate part numbers with their description, may help interpretation at later date.

# First step is to remove duplicates in series.
# For example, in the raw data if we sold two units of CL-23 and one of CL-12 then it would appear as
#   CL-23, CL-23, CL-12.
# We want our data to be
#   CL-23, CL-12.
# Reason is that many parts are sold in 10's or 100's. Word2Vec wouldn't be very useful if presented wi
#   CL-23, CL-23, CL-23, CL-23, CL-23, CL-23, CL-23, CL-23, CL-23, CL-23  etc
# However, we will permit duplicates if the part just happened to be sold on two sequential invoices.
# Also we group by week, to ensure enough parts within any period for Word2Vec to perform useful skip_g
library(dplyr)
library(stringr)
library(lubridate)

the_text_1 <- parts_all %>%
                mutate(year_week = as.numeric(str_c(year(INVOICE_DATE),week(INVOICE_DATE))))%>%
                  arrange(year_week, INVOICE_NO, CODE) %>%
                    mutate(rowtest = str_c(INVOICE_NO, CODE)) %>%
                      filter(rowtest != lag(rowtest, default="1")) %>%
                        arrange(year_week, INVOICE_NO, RANDOM_SEQ)

# Let's exmaine our lexicon, how frequently does each word (aka part) appear?
lexicon_code  <- the_text_1 %>% group_by(CODE) %>% summarise(QtyOfInvoicesWithPart=n())

# The Word2Vecprocess doesn't like tests with thousands of words which appears once or twice.
# Unlike language text, a large proportion (>1/3) of the text is comprised of
# parts with only 1-3 occurences
# This clogs the Word2Vec process with words about which we can learn nothing, so better remove them
the_text_1 <- the_text_1 %>%
                inner_join(lexicon_code, by=c("CODE"="CODE")) %>%
                 filter(QtyOfInvoicesWithPart > 2)

# The tokenizer to be used in the Word2Vec process may interpret underscores, hyphens, slashes etc
# ,which are present in part numbers, as spaces. This will create extraneous 'words'.
# Most importantly, the prefix 'CL' (which is followed by a hyphen) will be the most common word!
# Clearly we don't want this, so we remove these characters from part numbers, being careful
# to keep a mapping table of old part numbers to new
lexicon <- parts_decode %>%
            inner_join(y=lexicon_code, by=c("CODE" = "CODE")) %>%
              mutate(PART_NUMBER_Clean = str_to_lower(
                                           str_replace_all(PART_NUMBER,
                                                           "[[:punct:]\\s]+","x")))
lexicon_count <- nrow(lexicon %>% distinct(PART_NUMBER_Clean))

# Note, the data is already tokenised with partnumbers represented by integers in the 'CODE' field
# The table 'parts_decode' is used to decode the tokens back to part numbers and their description
# However, we want to use the keras tokenizer, so will de-tokenize the data
the_text_2 <- the_text_1 %>%
                inner_join(y=lexicon, by=c("CODE" = "CODE")) %>%
                  select(year_week, INVOICE_NO, PART_NUMBER_Clean)
```

```
# next, we present the data as if each day was a paragraph (a vector)
# and each invoice were a sentence
the_text_3 <- the_text_2 %>%
                group_by(year_week) %>%
                  mutate(PartsInPeriod = str_c(PART_NUMBER_Clean, collapse = " "),
                         PartsInPeriod_Count = n()) %>%
                    arrange(year_week, INVOICE_NO) %>%
                      filter(PartsInPeriod != lag(PartsInPeriod, default="Default")) %>%
                        ungroup() %>%
                          select(PartsInPeriod, PartsInPeriod_Count)

# filter out rows with only one part number, Word2Vec doesn't use those,
# in fact they cause the proces to fail
the_text_4 <- the_text_3 %>%
                filter(PartsInPeriod_Count > 1) %>%
                  select(PartsInPeriod)

# finally, we want the data as a list of vectors of characters, and advantageous if in UTF8
the_text_list <- unlist(the_text_4)
attributes(the_text_list) <- NULL
the_text_list <- iconv(the_text_list, to = "UTF-8")
```

The model needs to be presented with batches of text. This is the task of the 'Generator', as it is commonly named in deep learning models

```
library(reticulate)
library(purrr)
library(keras)

# setup the keras tokenizer, run it to find the number of words it identifies in the corpus
#The run it again, specifying the number of those words
tokenizer <- keras::text_tokenizer()
tokenizer %>% keras::fit_text_tokenizer(the_text_list)

lexicon_count <- length(tokenizer$word_index)

tokenizer <- keras::text_tokenizer(num_words = lexicon_count)
tokenizer %>% keras::fit_text_tokenizer(the_text_list)

# create generator function
skipgrams_generator <- function(text, tokenizer, window_size, negative_samples) {

  gen <- texts_to_sequences_generator(tokenizer, sample(text))

  function() {
    skip <- generator_next(gen) %>%
      skipgrams(
        vocabulary_size  = tokenizer$num_words,
        window_size      = window_size,
        negative_samples = 1
      )
    x <- transpose(skip$couples) %>% map(. %>% unlist %>% as.matrix(ncol = 1))
    y <- skip$labels %>% as.matrix(ncol = 1)
    list(x, y)
```

```
  }
}
```

Define the Keras model for Word2Vec

```
# Defined using the Keras functional API

embedding_size <- 128  # Dimension of the embedding vector.
skip_window    <- 5  # How many words to consider left and right.
num_sampled    <- 1   # Number of negative examples to sample for each word.

# We will first write placeholders for the inputs using the keras layer_input function.

input_target  <- layer_input(shape = 1)
input_context <- layer_input(shape = 1)

# Now let's define the embedding matrix.
# The embedding is a matrix with dimensions (vocabulary, embedding_size)
# this acts as lookup table for the word vectors.

embedding <- layer_embedding(input_dim = tokenizer$num_words + 1,
                             output_dim = embedding_size,
                             input_length = 1,
                             name = "embedding")

target_vector <- input_target %>%
                 embedding() %>%
                   layer_flatten()

context_vector <- input_context %>%
                    embedding() %>%
                      layer_flatten()

# define how the target_vector will be related to the context_vector in order to make
# our network output 1 when the context word really appeared in the context and 0 otherwise
# A common measure is the cosine similarity
# As we don't need the similarity to be normalized inside the network,
# we will only calculate the dot product and then output a dense layer with sigmoid activation

dot_product <- layer_dot(list(target_vector, context_vector), axes = 1)
output <- layer_dense(dot_product, units = 1, activation = "sigmoid")

# Now we will create the model and compile it.

model <- keras_model(list(input_target, input_context), output)
model %>% compile(loss = "binary_crossentropy", optimizer = "adam")

# We can see the full definition of the model by calling summary:
```

```
summary(model)
```

```
## ____________________________________________________________________________
## Layer (type)            Output Shape     Param #  Connected to
```

```
## ==========================================================================
## input_1 (InputLayer)    (None, 1)        0
## __________________________________________________________________________
## input_2 (InputLayer)    (None, 1)        0
## __________________________________________________________________________
## embedding (Embedding)   (None, 1, 128)   1351680  input_1[0][0]
##                                                   input_2[0][0]
## __________________________________________________________________________
## flatten_1 (Flatten)     (None, 128)      0        embedding[0][0]
## __________________________________________________________________________
## flatten_2 (Flatten)     (None, 128)      0        embedding[1][0]
## __________________________________________________________________________
## dot_1 (Dot)             (None, 1)        0        flatten_1[0][0]
##                                                   flatten_2[0][0]
## __________________________________________________________________________
## dense_1 (Dense)         (None, 1)        2        dot_1[0][0]
## ==========================================================================
## Total params: 1,351,682
## Trainable params: 1,351,682
## Non-trainable params: 0
## __________________________________________________________________________
```

Fit the model...

```
# We will fit the model using the fit_generator() function
# We need to specify the number of training steps as well as number of epochs we want to train.
# We are short of data, so go over it 30x
# Takes 20mins on my old Nvidia 1050 GPU

the_text_list_x30 <-rep(the_text_list, times=30)

model %>%
  fit_generator(
    skipgrams_generator(the_text_list_x30, tokenizer, skip_window, negative_samples),
    steps_per_epoch = floor(length(the_text_list_x30)/30),
    epochs = 30
    )
```

```
#save model for future usage
word2vec_serialized <- serialize_model(model, include_optimizer = TRUE)
#model <- unserialize_model(word2vec_serialized)
```

The training proceeded reasonably well, error reducing at each epoch, from 0.6 down to 0.2

Let's see if there is any relationship between the correlation of two part numbers' sales and the consine similarity between those part numbers in Word2Vec space. If there is, then we have some confidence that the Word2Vec process has worked.

First step is to extract the embeddings matrix from the model by using the get_weights() function. We also add row.names to our embedding matrix so we can easily find where each word is.

```
library(dplyr)

embedding_matrix <- get_weights(model)[[1]]

words <- data_frame(word = names(tokenizer$word_index),
                    id = as.integer(unlist(tokenizer$word_index)))
```

```
words <- words %>%
          filter(id <= tokenizer$num_words) %>%
            arrange(id)

row.names(embedding_matrix) <- c("unk", words$word)
#NB the first item is for any word not found, so is also "unk"
```

Understanding the Embeddings

We can now find words that are close to each other in the embedding. We will use the cosine similarity, since this is what we trained the model to minimize.

But first we will find some parts which we epxect to be highly similar, because we already know they are highly correlated.

```
close_pairs <- crosscorrel_rawdata_unq %>%
                  inner_join(y=lexicon, by=c("PART_NUMBER.x"="PART_NUMBER")) %>%
                    rename(PART_NUMBER.x_clean = PART_NUMBER_Clean,
                           QtyOfInvoicesWithPart.x = QtyOfInvoicesWithPart) %>%
                      filter(QtyOfInvoicesWithPart.x>100) %>%
                        inner_join(y=lexicon, by=c("PART_NUMBER.y"="PART_NUMBER")) %>%
                          rename(PART_NUMBER.y_clean = PART_NUMBER_Clean,
                                 QtyOfInvoicesWithPart.y = QtyOfInvoicesWithPart) %>%
                            filter(QtyOfInvoicesWithPart.y>100) %>%
                              filter(PART_NUMBER.x != "unk" & PART_NUMBER.y != "unk" ) %>%
                                    arrange(desc(ccor_no_lag), PART_NUMBER.x_clean)

first_test <- close_pairs[1,]$PART_NUMBER.x_clean
```

```
library(text2vec)
```

```
## Warning: package 'text2vec' was built under R version 3.5.1
```

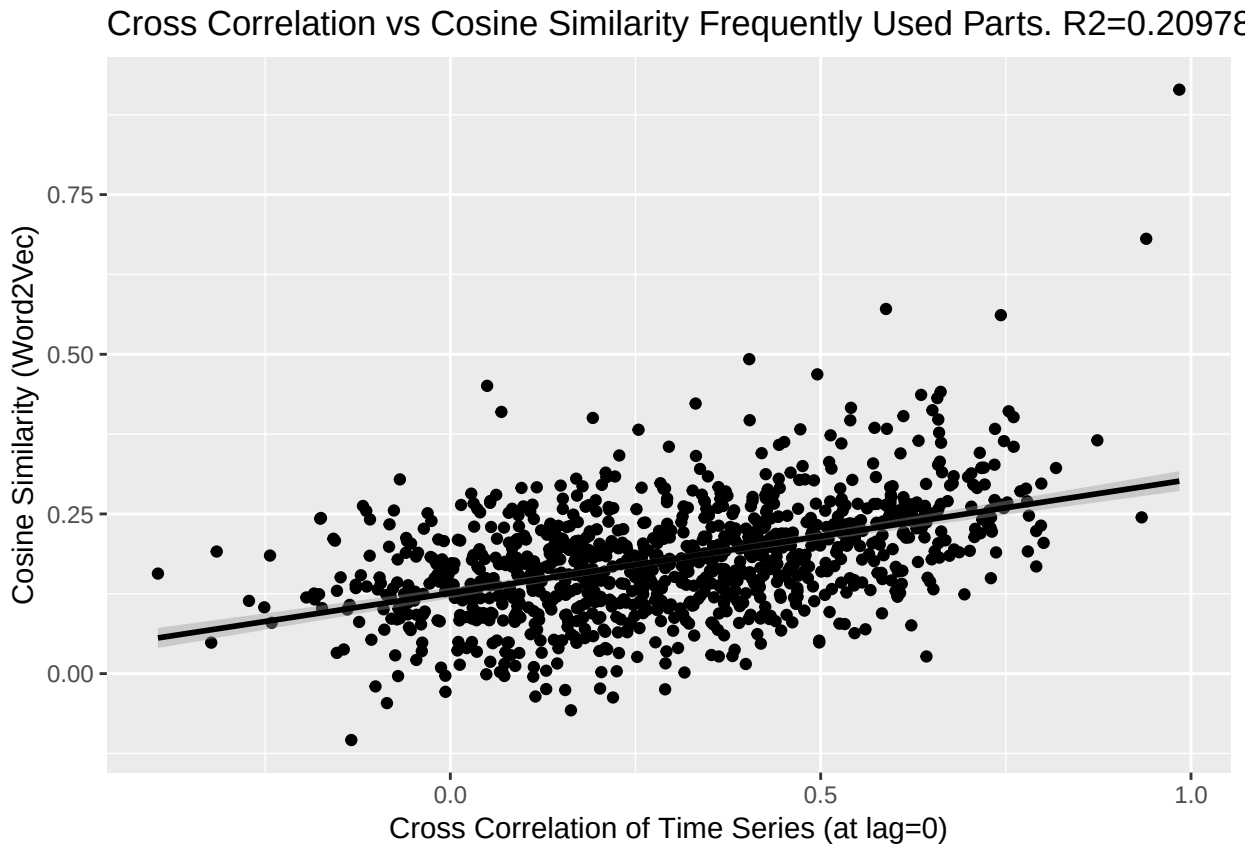
```
find_similar_words <- function(word, embedding_matrix, n = 5) {
  ref <- which(attr(embedding_matrix, "dimnames")[[1]] == word)
  similarities <- embedding_matrix[ref, , drop = FALSE] %>%
    sim2(embedding_matrix, y = ., method = "cosine")

  similarities[,1] %>% sort(decreasing = TRUE) %>% head(n)
}

similar <- find_similar_words(first_test, embedding_matrix)
kable(similar)
```

	x
clx0001474931	1.0000000
clx0005100541	0.6113939
clx0005646982	0.5794133
clxuksa200	0.5502768
clxukcl00773821	0.4867027

Thats working, now let's randomly sample 1,000 pairs to plot only use those row refs which have a substantial number of sales (some information worth looking at)

```r
row_refs <- sample(1:nrow(close_pairs), 1000, replace=F)
close_pairs$CosineSim <- NA

#Calculate Word2Vec cosine similarities for those pairs
close_pairs[row_refs,]$CosineSim <- sapply(row_refs, function(i) {
                            x <- close_pairs[i,]$PART_NUMBER.x_clean
                            y <- close_pairs[i,]$PART_NUMBER.y_clean

                            ref_x <- which(attr(embedding_matrix, "dimnames")[[1]] == x)
                            ref_y <- which(attr(embedding_matrix, "dimnames")[[1]] == y)

                           sim2(x = embedding_matrix[ref_x, , drop = FALSE],
                                y = embedding_matrix[ref_y, , drop = FALSE],
                                method = "cosine")
})

#get a linear model so we can measure R2 between Cross Correlation and Cosine Sim
linmodel <- lm(CosineSim ~ ccor_no_lag, close_pairs[row_refs,])
R2 <- getRsquared(linmodel$fitted.values, close_pairs[row_refs,]$CosineSim)
```

```r
#plot Cosine sim vs Cross correl
m<- ggplot(data=close_pairs[row_refs,],
            aes(x=ccor_no_lag, y=CosineSim)) +
            geom_point() +
            #ylim(-0.1, 0.05) +
            geom_smooth(method = "lm", se=TRUE, color="black", formula = y ~ x) +
            xlab("Cross Correlation of Time Series (at lag=0)") +
            ylab("Cosine Similarity (Word2Vec)") +
            #annotate("text", x = 20, y = -0.4, label = label, parse = F)+
            ggtitle(paste0("Cross Correlation vs Cosine Similarity Frequently Used Parts. R2=",R2 ))
m
```

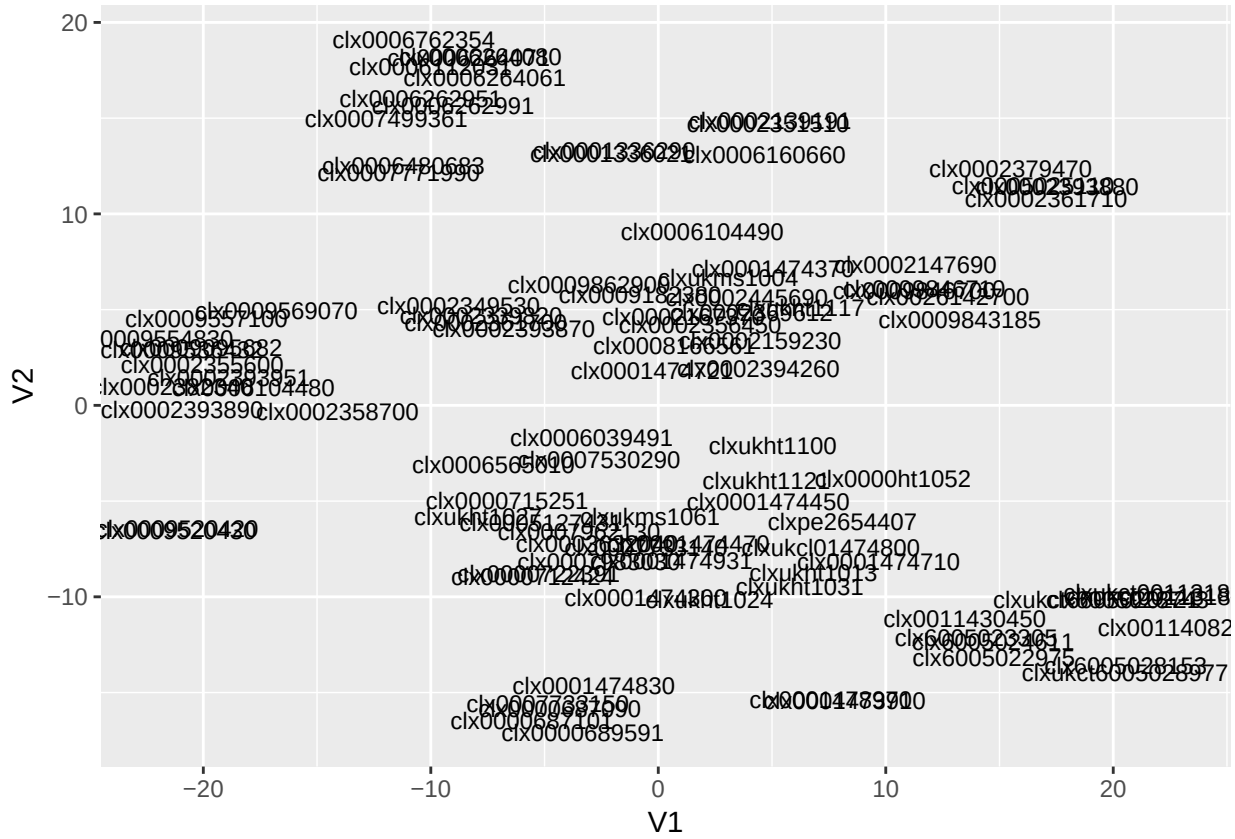


A fifth of the variance is explained by cross correlation of the time series. Enough for us to have faith that the Word2Vec process is generating sensible weights (not random numbers), but little enough for us to see it making a useful addition to cross correlation.

It is common to use tsne to plot such embeddings:

```
library(Rtsne)
library(ggplot2)
#library(plotly)

tsne <- Rtsne(embedding_matrix[2:100,], perplexity = 5, pca = FALSE)

tsne_plot <- tsne$Y %>%
               as.data.frame() %>%
                 mutate(word = row.names(embedding_matrix)[2:100]) %>%
                   ggplot(aes(x = V1, y = V2, label = word)) +
                   geom_text(size = 3)
```

```
tsne_plot
```



These embeddings may be clustering, they might also be forming a circle of no information.

Extract clusters

We want to find clusters which are derived from the cosine similarity. In previous code chunks we have calculated distance using the dist() function and then applied hclust to find clusters. dist() has no option for the cosine method, but we can easily derive our own method for this distance. This has been done in the code chunk below.

Note, similarity is the opposite of distance, so before clustering, this similarity would need to be converted into a distance using: 1- Cosine Similarity/Max(Cosine Similarity) = distance.

However, we have 10,242 words, we will need a table of 10,242 squared (105million) rows to define the distance between each word and all other words. When running the code we get the following error "Error: memory exhausted". Not surprising.

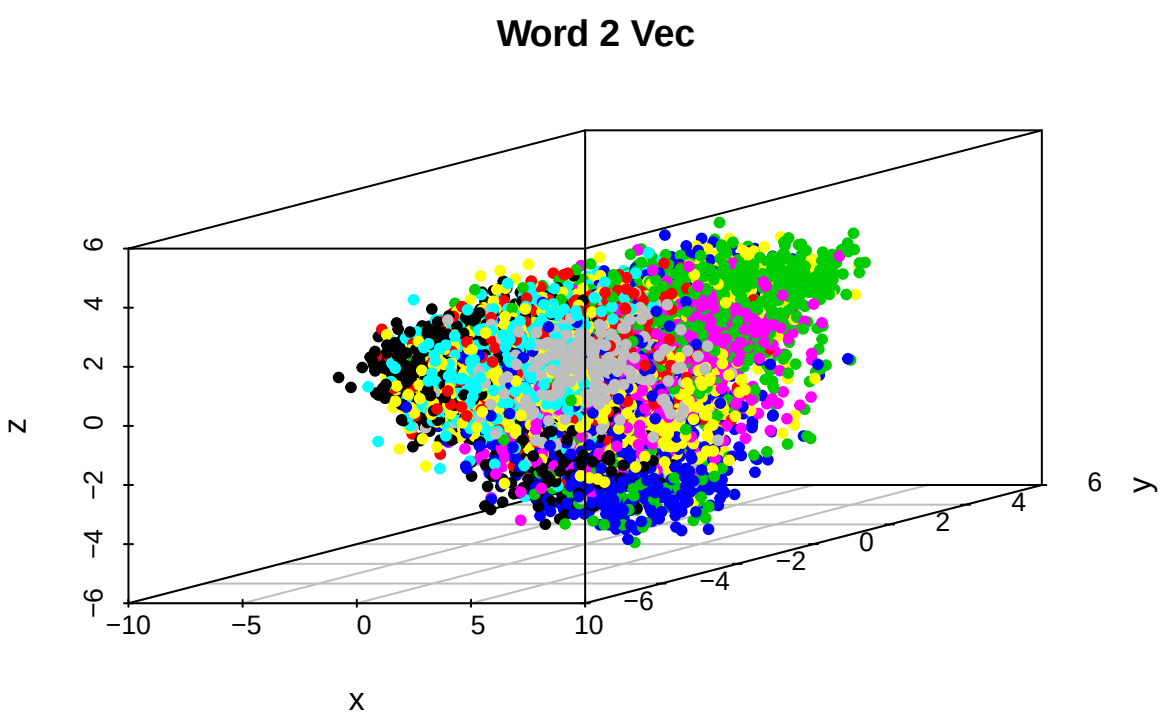
```
library(pbapply)
library(parallel)

get_cosine_similarity <- function(df, cl){
  cosine_sim <- function(dataframe, ix)
  {
    A = df[ix[1],]
    B = df[ix[2],]
    return( sum(A*B)/sqrt(sum(A^2)*sum(B^2)) )
  }
  n   <- nrow(df)
```

```
  cmb <- expand.grid(i=1:n, j=1:n)
  C   <- pbapply(cmb,
                 1,
                 cl = cl,
                 function(cmb){cosine_sim(df, cmb)})
  C   <- matrix(C,n,n)
  return(C)
}

no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
embeddings_cossims <- get_cosine_similarity(embedding_matrix, cl)
stopCluster(cl)
```

Ran out of memory on a 32GB server with 1TB drive. Such is the effect of a matrix 10,242words x 10,242words! Let's fall back on clustering using Kmeans

```
#kmeans with 50 centres, as per above PCA analysis
embeddings_PCA <- getPCA(embedding_matrix, SDthreshold=3, DisplayPlot=F,setRownames=F)
embeddings_PCA_obj      <- embeddings_PCA$pca_obj
embeddings_PCA_data     <- embeddings_PCA$pca_data
embeddings_PCA_outliers <- embeddings_PCA$outliers
rm(embeddings_PCA)

threshold_dim <- getThresholdDim(pca_obj = embeddings_PCA_obj,
                                 approach = "Word2Vec",
                                 DisplaySummary = T)

embeddings_clusters <-  GetClusterAssignment_PcaAndKmeans(pca_obj = embeddings_PCA_obj,
                                                          threshold_dim = threshold_dim,
                                                          cluster_qty = 50,
                                                          iter.max = 20)
```

```
plot3Dcluster(pca_obj = embeddings_PCA_obj,
              cluster_assignment = embeddings_clusters,
              approach = "Word 2 Vec")
```



Out of curiosity, we can use our previous code to find the mean cross correlation of each cluster.

```
embeddings_clusters_sumry <- data.frame(
                               PART_NUMBER_Clean = attr(embedding_matrix,"dimnames")[[1]],
                               CLUSTER = embeddings_clusters$clust_assign$CLUSTER,
                               stringsAsFactors = F) %>%
                                 inner_join(y=lexicon, by="PART_NUMBER_Clean") %>%
                                   select(PART_NUMBER, PART_NUMBER_Clean, CLUSTER)

meancorrel_word2vec <- getMeanCorrel(crosscorrel_rawdata_log_unq, embeddings_clusters_sumry)
ClustPerf[26,]      <- c("Word2Vec","Kmeans", meancorrel_word2vec)
```

```
kable((ClustPerf %>% arrange(desc(Mean_CorrelNoLag))))
```

SourceData	ClusteringMethod	Mean_CorrelWithLag	Mean_CorrelNoLag	Median_PartsPerCluster
Sales Qty	HClust	0.689	0.669	1
Log(Sales+1)	HClust	0.62	0.606	2
Sales Qty	KMeans	0.601	0.56	2
Wavelets Trends	HClust	0.548	0.556	1
Trends	HClust	0.574	0.553	1
CorrelsOfTrends	HClust	0.613	0.545	5
Log(Trend+1)	HClust	0.565	0.5	4.5
Trends	KMeans	0.53	0.46	13
CorrelsOfTrends	KMeans	0.527	0.458	24.5
Log(Sales+1)	KMeans	0.475	0.443	11.5
Wavelets log(sales qty+1)	Kmeans	0.484	0.439	8
Log(Trend+1)	KMeans	0.497	0.42	27.5

SourceData	ClusteringMethod	Mean_CorrelWithLag	Mean_CorrelNoLag	Median_PartsPerCluster
DTW Clustering, Trend	Partitional	0.386	0.33	30
Trends	Prefix	0.331	0.311	23
CorrelsOfSalesQtyDeseason	HClust	0.348	0.283	5.5
CorrelsOfSalesQty	KMeans	0.359	0.274	24.5
Trends	Random	0.318	0.271	20
CorrelsOfSalesQty	HClust	0.366	0.266	7
DTW Clustering, Raw	Partitional	0.355	0.266	15.5
CorrelsOfSalesQtyDeseason	KMeans	0.327	0.262	30
CorrelsOfLog(SalesQty+1)	HClust	0.332	0.258	7
CorrelsOfLog(SalesQty+1)	KMeans	0.316	0.248	25.5
Word2Vec	Kmeans	0.328	0.238	22
SalesQty	Prefix	0.295	0.185	23
DTW Clustering, Raw	Hierarchical	0.336	0.18	5
SalesQty	Random	0.268	0.138	22.5

A rather disappointing performance for Word2Vec.

Word2Vec Hyper-parameter Tuning

The above results followed some hyperparameter tuning. We have these parameters to tune for Word2Vec:

-Dimension of the embedding vector. –embedding_size -How many words to consider left and right. –skip_window
-Number of negative examples to sample for each word. –num_sampled
-How many invoices a part should appear on before being awarded a name of its own in the text, versus simply being labelled “unk” –QtyOfInvoicesWithPart

. . . and these measure of success:

-Error after 30 epochs –Error -RMSE (aka R2) in the relationship between Cosine Similarity and Cross Correlation. –QtyOfInvoicesWithPart

Results from chose configurations:

Embedding Dims Skip Window (+/-) Num_Sampled QtyOfInvoicesWithPart Error R2 64 5 1 >1 0.43 0.00 64 5 1 >2 0.26 0.21 64 5 1 >3 0.28 0.24 64 5 2 >2 0.46 0.02 64 10 1 >2 0.52 0.04

128 5 1 >0 0.20 0.03 128 5 1 >2 0.22 0.20 128 5 1 >3 0.24 0.20 128 10 1 >2 0.48 0.01

Selected configuration:

Embedding Dims Skip Window (+/-) Num_Sampled QtyOfInvoicesWithPart Error R2 128 5 1 >2 0.22 0.20

This is the configuration represented in the code chunks above. Having conducted 30 epochs in hyper-param tuning, the selected model was further trained for another 30 epochs.

Data for the Forecasting Models

Finally, we are ready to summarise the data to be presented to the forecasting model. - Sales Qty & Sales Events - Trends (aka 12mth moving average) - Cluster Assignment from Kmeans on Correlation of Trends

The cluster assignments must change over time, the model must not know the future. Clusters based on data available upto 2018 cannot be available to a model wishing to forecast 2016 using data upto 2015. Hence cluster assignment should be monthly.

```
# Cycle through the months_list, getting ARIMAs and forecasts as would
# have been possible if we were present in that month (ie no future knowledge)

forecasts_fromAllMonths       <- NULL
selected_series_fromAllMonths <- NULL

pb <- txtProgressBar(min = 1, max = nrow(partnumbers_top_pc), style = 3)
cat(" \n")
print("Building source data and forecasts as from all previous months")

for (i in nrow(monthslist):1){ #

  # progress bar and info
  setTxtProgressBar(pb, i-(nrow(monthslist)-1))
  cat(" \n")
  print(paste0("MonthSeq=",i,". Month=", monthslist$INVOICE_MONTH[i]))
  cat(" \n")
  # We are working backwards from the most recent month, so
  monthslist_LimitedMths <- monthslist[1:i,]
  months_from    <- min(monthslist_LimitedMths$INVOICE_MONTH)
  months_to      <- max(monthslist_LimitedMths$INVOICE_MONTH)

  # we need only analyse the PART_NUMBERS which were valid in this monthslist
  partnumbers_LimitedMths <- partnumbers_top_pc %>%
                                     filter(#Last sale must be after earliest date
                                            LAST_SALE  >= months_from
                                            &
                                            #first sale must be before latest date
                                            FIRST_SALE <= months_to)

  series_LimitedMths      <- series_top_pc%>%
                                     filter(#Last sale must be after earliest date
                                            LAST_SALE  >= months_from
                                            &
                                            #first sale must be before latest date
                                            FIRST_SALE <= months_to)

  # Get ARIMAs
  output<- convertToTimeSeries(selected_partnumbers        = partnumbers_LimitedMths,
                               selected_partnumbers_series = series_LimitedMths,
                               months_list                 = monthslist_LimitedMths,
                               GetARIMA = TRUE)

  selected_series_df     <- output[[1]]
  selected_series_lists  <- output[[2]]
  rm(output)

  # flag the series data with the month of data with which it was built
  selected_series_df <- selected_series_df %>% mutate(ForecastFromMth = months_to)

  # Get Forecasts from the ARIMAS
  forecasts_for_month <- get_ArimaAndSimpleForecasts(series_df   = selected_series_df,
                                                     series_list = selected_series_lists,
```

```
                                                    months_list = monthslist_LimitedMths)

  # Append results to larger table of results for all given 'ForecastsFromMth'
  # Remove months which have 'NA' sales.
  # Data set is very sparse, we run out of memery if we leave 'NA' rows in the dataset
  if(i<nrow(monthslist)){

    #Collect Trend value at a given month, at a given 'ForecastFromMth'
    #We use these for clustering
    #remove NA rows and extraneous columns, both to reduce memory consumption
    selected_series_df <- selected_series_df %>%
                            filter(!(is.na(SALES_QTY) & is.na(SALES_EVENTS))) %>%
                              select(ForecastFromMth, PART_NUMBER, INVOICE_MONTH, TREND)

    #Bind Trend values to all previous records
    selected_series_fromAllMonths <- rbind(selected_series_fromAllMonths,
                                           selected_series_df)

    #Collect forecasts and bind to all previous records
    forecasts_fromAllMonths       <- rbind(forecasts_fromAllMonths,
                                           forecasts_for_month)
  }

}
```

Get correlations for each month

```
                    # approx 50 parts per cluster, never less than 4 clusters.
cluster_qty      <- max(round(nrow(selected_series_fromAllMonths %>% distinct(PART_NUMBER))/30,0),4)
clusters_allMths <- NULL
no_cores         <- detectCores() - 1

for (i in (nrow(monthslist)-1):2){ # excludes most recent month, which is incomplete

  currentMth         <- monthslist[i,]$INVOICE_MONTH
  print(paste0("MonthSeq=",i,". Month=",currentMth))

  series_forMth      <- selected_series_fromAllMonths %>%
                          filter(ForecastFromMth == currentMth)

  cl <- makeCluster(no_cores)
  crosscorrel_forMth <- getCrossCorrels(series    = series_forMth,
                                        component = "TREND",
                                        cl        = cl)
  stopCluster(cl)

  #Where a part is correlated with itself, yet result given is NA, then replace with 1

  pca_data_forMth <- crosscorrel_forMth %>%
                       select(PART_NUMBER = PART_NUMBER.x, PART_NUMBER.y, ccor_no_lag) %>%
                         arrange(PART_NUMBER) %>%
                           spread(key=PART_NUMBER, value=ccor_no_lag, fill = 1) %>%
```

```
                            rename(PART_NUMBER = PART_NUMBER.y)


  pca_obj_forMth  <- getPCA(pca_data = pca_data_forMth, DisplayPlot = FALSE)
  pca_obj_forMth  <- pca_obj_forMth$pca_obj
  threshold_dim   <- getThresholdDim(pca_obj_forMth, "v1a:correlations of trends",
                                     DisplaySummary=FALSE)

  clusters_forMth <- GetClusterAssignment_PcaAndKmeans(pca_obj       = pca_obj_forMth,
                                                       threshold_dim = threshold_dim,
                                                       cluster_qty   = cluster_qty)

  # cbind the clusters to the current month of analysis
  clusters_forMth <- cbind(clusters_forMth$clust_assign,
                           currentMth)

  colnames(clusters_forMth) <- c("PART_NUMBER","CLUSTER","ForecastFromMth")

  # rbind results to general table
  clusters_allMths <- rbind(clusters_forMth,
                            clusters_allMths)

}

#The first month has a set of correlations equal to the second month.
#This is a bit of a cheat, but we can't calculate correls on dat set only one month long
currentMth       <- monthslist[2,]$INVOICE_MONTH
clusters_forMth  <- clusters_forMth %>% filter(ForecastFromMth == currentMth)
clusters_forMth$ForecastFromMth <- monthslist[1,]$INVOICE_MONTH
clusters_allMths <- rbind(clusters_forMth, clusters_allMths)
```

Identify clusters through time

A set of clusters assigned for July may be very similar to the set assigned for August, but the cluster labelled '1' in July might be labelled '4' in August.

This is inconvenient for any machine learning, as it becomes much harder to learn anything from clusters when assignments are obscured in this way. The answer is to try and use the same labels over time, wherever possible.

This is done using a 'voting' system. So, if 10 parts belong to cluster 1 in July and 9 of them belong to cluster 4 in August, then its a good bet that cluster 4 can be relabelled cluster 1.This requires us to have the same number of clusters each month and approx the same number of part_numbers. Fortunately, this is already the case for our data. Below is the logic which does the relabelling, working from the earliest month to the latest month.

```
Get_RevisedClusterLabels <- function(PartNumbersCommonToBoth_vc,
                                     ClusterAssignments_Prior_vc,
                                     ClusterAssignments_Next_vc){

  cluster_become <- cbind.data.frame(PART_NUMBER = PartNumbersCommonToBoth_vc,
                                     PriorRef    = as.numeric(ClusterAssignments_Prior_vc),
                                     NextRef     = as.numeric(ClusterAssignments_Next_vc),
                                     stringsAsFactors = FALSE)
```

```
  # Create mapping, using 'voting'. Executed as a count of PriorRef-NextRef pairs
  mapping <- cluster_become %>%
               # Get votes per pair
               count(PriorRef, NextRef) %>%
                arrange(NextRef, desc(n)) %>%
                  # Identify top vote for each pair
                  group_by(NextRef) %>%
                    mutate(Sequence = row_number(desc(n))) %>%
                      # Select only the top vote for each pair
                      filter(Sequence == 1) %>%
                        distinct(PriorRef, NextRef) %>%
                          select(ChangeFrom = NextRef, ChangeTo = PriorRef)

  #Must not map from a value to NA
  #If that is proposed, then ensure the old value is used instead
  mapping_refs_to_swap <- which(is.na(mapping$ChangeTo))
  mapping$ChangeTo[mapping_refs_to_swap] <- mapping$ChangeFrom[mapping_refs_to_swap]

  # Now we create the mapping using the match function(),
  # which returns the index of the matched item.
  # The trick is to have the ChangeFrom refs in the order of ChangeTo
  NextRef_changed <- match(cluster_become$NextRef,
                           (mapping %>% arrange(ChangeTo))$ChangeFrom)

  #Match returns the index of a matched value.
  # If NA is a possibility, then the index of NA is always the highest index in the process
  # This has the unfortunate effect of replacing NA's with a value
  # We must reverse this error, replacing highest values with NA
  if(NA %in% mapping$ChangeFrom){
    refs_to_NA      <- which(NextRef_changed == nrow(mapping))
    NextRef_changed[refs_to_NA] <- NA
  }
  #test
  #test<- cbind.data.frame(PriorRef        = cluster_become$PriorRef,
  #                        NextRef         = cluster_become$NextRef,
  #                        NextRef_changed = NextRef_changed)

  return(NextRef_changed)
}

# spread the clusters such that each column is the clusters for a month,
# early months at left of table, thru to later months at right of table.
clusters_allMths_spd <- clusters_allMths %>%
                        spread(key=ForecastFromMth, value=CLUSTER, fill=NA)

#cycle through the table, left to right, two columns at a time
for(i in 1:(ncol(clusters_allMths_spd)-1)){

  clusters_allMths_spd[,i+1] <-  Get_RevisedClusterLabels(
                                  PartNumbersCommonToBoth_vc  = clusters_allMths_spd$PART_NUMBER,
                                  ClusterAssignments_Prior_vc = clusters_allMths_spd[,i],
                                  ClusterAssignments_Next_vc  = clusters_allMths_spd[,i+1]
                                  )
```

```
}

# gather (aka unspread, or unpivot) the data back into a long  narrow table
# of part_number, month, revised cluster assignment
clusters_fromAllMonths_revised <- clusters_allMths_spd %>%
                                    gather(key   = ForecastFromMth,
                                           value = CLUSTER,
                                           -PART_NUMBER) %>%
                                      mutate(ForecastFromMth = as.Date(ForecastFromMth))%>%
                                        arrange(ForecastFromMth, PART_NUMBER)
```

We now have four tables: 1. series_top_pc - source for SALES_QTY and SALES_EVENTS 2. selected_series_fromAllMonths - was built to find clusters and forecasts. No longer required 3. forecasts_fromAllMonths - source of ARIMAforecast, ARIMAsigma2, SimpleForecast 4. clusters_fromAllMonths_revised - source of CLUSTER

Building Tensors for Predictors and Labels

The data types will form a 3D array (cube) per 'ForecastFromMonth' The model will slide through the Invoice months, reading a table for each month Array dims are: X = PART_NUMBERs [~2500] Y = Invoice months [~120] Z = Data Types [7] Where 'data types' = SALES_QTY, SALES_EVENTS, CLUSTER ARIMAforecast, ARIMASigma2, Simple Forecast, MonthOfYear

```
#Create function to get a slice, one slice per input data type
get_slice_from_series <- function(template_df, series_df, componentToSpread_chr){
  #series_top_pc
  template_df %>%
    left_join(y = series_df, by=c("PART_NUMBER","INVOICE_MONTH")) %>%
      select_("PART_NUMBER", "INVOICE_MONTH", componentToSpread_chr) %>%
        spread_(key   = "INVOICE_MONTH",
                value = componentToSpread_chr,
                fill  = 0) %>%
          arrange(PART_NUMBER)
}

get_slice_from_forecasts <- function(template_df, forecasts_df, componentToSpread_chr){
  #forecasts_fromAllMonths
  template_df %>%
    left_join(y = forecasts_df, by=c("PART_NUMBER"   = "PART_NUMBER",
                                     "INVOICE_MONTH" = "ForecastFromMth")) %>%
       select_("PART_NUMBER", "INVOICE_MONTH", componentToSpread_chr) %>%
        spread_(key   = "INVOICE_MONTH",
                value = componentToSpread_chr,
                fill  = NA) %>%
          arrange(PART_NUMBER)
}
```

```
#All slices must have common part numbers and months, so we create a list to act as a template
# all part numbers
allparts  <- series_top_pc %>% distinct(PART_NUMBER) %>% arrange(PART_NUMBER) %>% mutate(fake=1)

# all relevant months
allmonths <- monthslist %>%
```

```
              filter(INVOICE_MONTH >= min(series_top_pc$INVOICE_MONTH)
                   & INVOICE_MONTH <= max(series_top_pc$INVOICE_MONTH)) %>%
                distinct(INVOICE_MONTH) %>%
                  arrange(INVOICE_MONTH) %>% mutate(fake=1)

#create template list
template_for_slices <- allparts %>%
                        full_join(y = allmonths, by = "fake") %>% select(-fake) %>%
                          distinct(PART_NUMBER, INVOICE_MONTH) %>%
                            arrange(PART_NUMBER, INVOICE_MONTH)

#Slice Source: series_top_pc
slice_Sales      <- get_slice_from_series(template_for_slices, series_top_pc, "SALES_QTY")
slice_Events     <- get_slice_from_series(template_for_slices, series_top_pc, "SALES_EVENTS")

#Slice Source: clusters_fromAllMonths_revised
slice_Cluster    <- template_for_slices %>%
                     left_join(y=clusters_fromAllMonths_revised,
                               by=c("PART_NUMBER"   = "PART_NUMBER",
                                    "INVOICE_MONTH" = "ForecastFromMth")) %>%
                       select(PART_NUMBER, INVOICE_MONTH, CLUSTER) %>%
                         spread(key   = INVOICE_MONTH, value = CLUSTER, fill  = NA)%>%
                           arrange(PART_NUMBER)

#Slice Source: forecasts_fromAllMonths
slice_ARIMAfork  <- get_slice_from_forecasts(template_for_slices,
                                             forecasts_fromAllMonths, "ARIMAforecast")

slice_ARIMAsigm2 <- get_slice_from_forecasts(template_for_slices,
                                             forecasts_fromAllMonths, "ARIMAsigma2")

slice_SimpleFork <- get_slice_from_forecasts(template_for_slices,
                                             forecasts_fromAllMonths, "SimpleForecast")

#############################################################################################
#unify slices into cube
#   X = PART_NUMBERs  [~2500]
#   Y = Invoice months [~120]
#   Z = Data Types        [7]
X      <- nrow(allparts)
Y      <- nrow(allmonths)
Z      <- 7
cube_x <- array(as.numeric(NA), dim=c(X,Y,Z))

#set rownames (aka dimnames) for each dimension

#X names Part numbers
dimnames(cube_x)[[1]] <- allparts$PART_NUMBER

#Y names, Months (decimal format)
dimnames(cube_x)[[2]] <- as.character(allmonths$INVOICE_MONTH)

#Z names, Data types
```

```
dimnames(cube_x)[[3]] <- c("SalesQty","EventsQty","Cluster","ARIMAforecast",
                           "ARIMAsigma2","SimpleForecast", "MonthOfYear")

#now insert slices of data into the empty (NA) dimensions...
cube_x[,,1] <- unname(as.matrix(slice_Sales[,-1])) # exclude col1, PART_NUMBER. Remove names
cube_x[,,2] <- unname(as.matrix(slice_Events[,-1]))
cube_x[,,3] <- unname(as.matrix(slice_Cluster[,-1]))
cube_x[,,4] <- unname(as.matrix(slice_ARIMAfork[,-1]))
cube_x[,,5] <- unname(as.matrix(slice_ARIMAsigm2[,-1]))
cube_x[,,6] <- unname(as.matrix(slice_SimpleFork[,-1]))
cube_x[,,7] <- matrix(month(allmonths$INVOICE_MONTH), nrow=X, ncol=Y, byrow = TRUE)

#save data and structure by serializing to file
setwd("C:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM")
saveRDS(cube_x, file="pca_parts_cube_x.Rda")
```

We now need labels, aka outcomes, aka 'y' correct value for test and train This is already available in the forecasts_fromAllMonths table Its there as SalesQtyMths0to12, which is the correct forecast from the prior periods. Simply create new column, representing the date ending 12mths prior to the ForecastFromMth

```
# Create date column, 12mths prior
forecasts_fromAllMonths$ForecastFromMthLess12 <- forecasts_fromAllMonths$ForecastFromMth %m+% months(-1

# simplify table
slice_correct <- forecasts_fromAllMonths %>% select(PART_NUMBER,
                                                    INVOICE_MONTH = ForecastFromMthLess12,
                                                    CorrectForecast12Mths = SalesQtyMths01to12)

# We need 12 complete months ahead of a given date in order to
# know what the sales were in those 12 months.
# Yet, we won't have 12 months data yet!
# Also, the month we are in, the most recent month, is incomplete by definition
# So we must remove the final 13months, as we cannot provide a 'correct forecast'.

# remove those final 12months.
allmonths_Less13 <- allmonths[1:(nrow(allmonths)-13),]

# spread into X, Y, Z, same as for cube_x
slice_correct <- template_for_slices %>%
                 filter(INVOICE_MONTH %in% allmonths_Less13$INVOICE_MONTH) %>%
                 # removes last 12months
                 left_join(y = slice_correct, by=c("PART_NUMBER" , "INVOICE_MONTH")) %>%
                    select(PART_NUMBER, INVOICE_MONTH, CorrectForecast12Mths) %>%
                     spread(key   = "INVOICE_MONTH",
                            value = CorrectForecast12Mths,
                            fill  = 0) %>%
                       arrange(PART_NUMBER)

#############################################################################################################
#present result as a cube, but 1 deep (tensorflow likes this presentation)
#   X = PART_NUMBERs  [~2500]
#   Y = Invoice months [~120]
#   Z = Correct forecast  [1]
X      <- nrow(allparts)
```

```
Y      <- nrow(allmonths_Less13)
Z      <- 1
cube_y <- array(as.numeric(NA), dim=c(X,Y,Z))

#set rownames (aka dimnames) for each dimension

#X names Part numbers
dimnames(cube_y)[[1]] <- allparts$PART_NUMBER

#Y names, Months (decimal format)
dimnames(cube_y)[[2]] <- as.character(allmonths_Less13$INVOICE_MONTH)

#Z names, Data types
dimnames(cube_y)[[3]] <- "SalesQty"

#now insert slices of data into the empty (NA) dimensions...
cube_y[,,1] <- unname(as.matrix(slice_correct[,-1])) # exclude col1, PART_NUMBER, and final 11mths **

#save data and structure by serializing to file
setwd("C:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM")
saveRDS(cube_y, file="pca_parts_cube_y.Rda")
```

Save other useful tables

```
setwd("C:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM")
library(readr)
write_csv(forecasts_fromAllMonths, path="pca_parts_forecasts_fromAllMonths.csv")
write_csv(parts_decode,            path="pca_parts_parts_decode.csv")
write_csv(parts_frequent,          path="pca_parts_parts_frequent.csv")
write_csv(clusters_fromAllMonths_revised, path="pca_parts_clusters_fromAllMonths_revised.csv")
```

All data, inc long tail

Finally, save all sales and event data (not forecasts, clusters etc) in an array, as alternative model data. This will have all parts, including the 'long tail' of onseys and twoseys. A deep learning model just might be able to sue the data.

```
allparts_inclongtail  <- selected_partnumbers_all %>%
                         filter(FIRST_SALE >= min(monthslist$INVOICE_MONTH)
                                &
                                LAST_SALE <= max(monthslist$INVOICE_MONTH)) %>%
                         distinct(PART_NUMBER) %>%
                           arrange(PART_NUMBER) %>%
                             mutate(fake=1)

# all relevant months
allmonths_inclongtail <- monthslist %>% mutate(fake=1)

#create template list
template_for_slices_inclongtail <- allparts_inclongtail %>%
                                     full_join(y = allmonths_inclongtail, by = "fake") %>%
                                       select(-fake) %>%
                                         distinct(PART_NUMBER, INVOICE_MONTH) %>%
```

```
                                        arrange(PART_NUMBER, INVOICE_MONTH)

#Slice Source: series_top_pc
slice_Sales_inclongtail   <- get_slice_from_series(template_for_slices_inclongtail,
                                                   selected_partnumbers_series_all, "SALES_QTY")
slice_Events_inclongtail  <- get_slice_from_series(template_for_slices_inclongtail,
                                                   selected_partnumbers_series_all, "SALES_EVENTS")

# We'll need prices to judge how well we've performed
slice_Prices_inclongtail <- template_for_slices_inclongtail %>%
                            inner_join(y = parts_decode %>%
                                           select(PART_NUMBER, R_R_PRICE),
                                       by = "PART_NUMBER") %>%
                              spread(key   = "INVOICE_MONTH",
                                     value = R_R_PRICE,
                                     fill  = 0) %>%
                                arrange(PART_NUMBER)
```

We'll save the above data in a moment, but first we need to create simple forecasts for the long tail dataset, which we have not previously done. We need thos esimple forecasts becuase we must compare our deep models with the simple weighted average method. Not much point recommending deep models if no better than weighted averages!

We'll re-use code from above. Should have made it into a function....

```
# calculate simple forecast (3x3yrsago+2x2yrsago+1xlastyr) / 6.
# first get data into format where we can apply this function, requires 'Annual Period'
output <- convertToTimeSeries(selected_partnumbers_all %>%
                                filter(PART_NUMBER %in% allparts_inclongtail$PART_NUMBER),
                              selected_partnumbers_series_all,
                              monthslist,
                              GetARIMA = FALSE)

selected_partnumbers_inclongtail <- output[[1]]
rm(output)

# prep the data frame to which we will bind the forecasts
simple_forecasts_inclongtail <- data.frame(PART_NUMBER = allparts_inclongtail$PART_NUMBER,
                                           stringsAsFactors = FALSE)

# loop around the months getting for simple forecast for each month
pb <- txtProgressBar(min = 1, max = nrow(allmonths_inclongtail), style = 3)
for(i in 1:nrow(allmonths_inclongtail)){
  setTxtProgressBar(pb, i)

  ForecastFromMth <- allmonths_inclongtail$INVOICE_MONTH[i]
  simple_forecasts_inclongtail_currentyr <- get_simpleForecast(selected_partnumbers_inclongtail,
                                                               ForecastFromMth  = ForecastFromMth)

  #replace NA with zero
  for_replacement <- which(is.na(simple_forecasts_inclongtail_currentyr$SimpleForecast))
  simple_forecasts_inclongtail_currentyr$SimpleForecast[for_replacement] <- 0

  simple_forecasts_inclongtail <- cbind(simple_forecasts_inclongtail,
                                        simple_forecasts_inclongtail_currentyr$SimpleForecast)
```

```
}
```

```
# repeat above process for parts
X         <- nrow(allparts_inclongtail)
Y         <- nrow(allmonths_inclongtail)
Z         <- 4
cube_x_lt <- array(as.numeric(NA), dim=c(X,Y,Z))
dimnames(cube_x_lt)[[1]] <- allparts_inclongtail$PART_NUMBER
dimnames(cube_x_lt)[[2]] <- as.character(allmonths_inclongtail$INVOICE_MONTH)
dimnames(cube_x_lt)[[3]] <- c("SalesQty", "EventsQty", "R_R_PRICE", "SimpleForecast")
cube_x_lt[,,1] <- unname(as.matrix(slice_Sales_inclongtail[,-1]))
# excludes col1, PART_NUMBER. Remove names
cube_x_lt[,,2] <- unname(as.matrix(slice_Events_inclongtail[,-1]))
cube_x_lt[,,3] <- unname(as.matrix(slice_Prices_inclongtail[,-1]))
cube_x_lt[,,4] <- unname(as.matrix(simple_forecasts_inclongtail[,-1]))
setwd("C:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM")
saveRDS(cube_x_lt, file="pca_parts_cube_x_inclongtail.Rda")
```

Save the matching correct forecasts for the data inc long tail. We'll just apply a windowed aggregate on the sales_qty dimension of the cube. Window of 12mths, then shunt left by one month (sums are presented at first month of year, whereas we need month before the 12)

```
library(RcppRoll)
sales_qty_inclongtail <- array(data = cube_x_lt[,,1],
                               dim  = c(dim(cube_x_lt)[1],
                                        dim(cube_x_lt)[2]))

#calculate rolling sum, window of 12L, align from the right (work backwards in time)
cube_y_lt <- apply(X      = sales_qty_inclongtail,
                   MARGIN = 1, #apply to rows
                   FUN    = function(row){
                               require(RcppRoll)
                               roll_sum(x     = row,
                                        n     = 12L,
                                        by    = 1L,
                                        fill  = numeric(0),
                                        align = "right",
                                        na.rm = TRUE)})
# transpose
cube_y_lt <- t(cube_y_lt)
# project onto 3D, to match x cube
cube_y_lt <- array(data = cube_y_lt,
                   dim  = c(dim(cube_y_lt)[1],
                            dim(cube_y_lt)[2],
                            1)) # make me 3D!

#Y names, Months (decimal format)
# note remove 11, not 12. we are shunting by 1
allmonths_Less13_inclongtail <- allmonths_inclongtail[1:(nrow(allmonths_inclongtail)-11),]
dimnames(cube_y_lt)[[2]]     <- as.character(allmonths_Less13_inclongtail$INVOICE_MONTH)

#save data and structure by serializing to file
setwd("C:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM")
saveRDS(cube_y_lt, file="pca_parts_cube_y_inclongtail.Rda")
```