DMC@ISU: Iowa State University Data Mining Cup Team 2015

Initial Exploration

Spring 2015, Iowa State

Due Date:	April 7 2015

I am using the following packages:

```
library(ggplot2)
library(lubridate)
library(xtable)
library(rCharts)
library(plyr)
library(reshape2)
```

and my working directory is set to `dmc2015/ian`.

0.1 Starting with the basics

0.1.1 Reading the data correctly

The data is appears to be "pipe delimited", the same format used in the 2013 DMC. To be on the safe side we should possible scan the file first:

```
# scan the first five lines
raw_file <- scan(file = "../data/raw_data/DMC_2015_orders_class.txt",
    what = "character", nlines = 5, sep = "\n", na.strings = "")
raw_file[2]

## [1] "6054|2015-03-10 08:09:55|8877a81d04e853bc31c50da2bd2afaff|2015-03-10 08:03:09|a35974920ba3c9b3a

raw_file[4]

## [1] "6056|2015-03-10 10:42:51|313124dadc91f974ef726b76f1c4d014|2015-03-10 09:51:31|bd25bbf47a12fdb71
```

Notice that some of the variables have commas separating them. Additionally, there are `||||` at the ends of the lines (the classification data is missing what we need to predict).
Read the data into R:

```
# training set ('historical data')
trn <- read.delim("../data/raw_data/DMC_2015_orders_train.txt",
    stringsAsFactors = FALSE, sep = "|", quote = "")

# test set ('future data')
tst <- read.delim("../data/raw_data/DMC_2015_orders_class.txt",
    stringsAsFactors = FALSE, sep = "|", quote = "")
```

I will break up the data set into the following sets of columns for easy display:
Session information The variables `orderID`, `orderTime`, `userID`, and `couponsReceived` are described as "order number", "time of order", "user identifier", and "time of coupon generation" respectively. The response (?) `basketValue` could also go here. These variable all describe who was shopping and when.

```
print(xtable(head(trn[, c(1:4, ncol(trn))])), table.placement = "H")
```

	orderID	orderTime	userID	couponsReceived	basketValue
1	1	2015-01-06 09:38:35	f77fdea046ce0a40eb30ac35dde882cd	2015-01-06 09:34:53	187.60
2	2	2015-01-06 10:03:19	49fb109b47d0cee1753b63c24574d9d5	2015-01-06 10:00:44	185.93
3	3	2015-01-06 10:08:13	b767e0e5cda30fc2e7c39f8049eebb5a	2015-01-06 09:29:16	208.07
4	4	2015-01-06 13:23:23	cae358d7b2d59aafe8e41e70b363a198	2015-01-06 13:13:12	185.80
5	5	2015-01-06 13:37:22	9ddbc3f477d31b609f962555d72503ad	2015-01-06 12:48:41	272.12
6	6	2015-01-06 15:26:36	1c8dc3fa9ef6d8734fa26e5af8be0b0b	2015-01-06 15:14:20	126.01

Coupon 1, 2, and 3 Each coupon has its own set of descriptors attached to it. For instance, coupon i has `couponIDi` (an ID), `pricei` (the "current" price of the product to which coupon i pertains), `basePricei` (the "original" price of the product to which coupon i pertains), `rewardi` (the "score value" for the value-added of the product for the retailer) `premiumProducti` (indicates if the product is premium), `brandi` (the brand of the product), `productGroupi` (the "product line" of the product), and `categoryIDsi` (identifies the categories of the first coupon product). Finally, there is also `couponiUsed`.
Coupon 1:

```
print(xtable(head(trn[, which(grepl("1", names(trn)))[1:5]])),
    table.placement = "H")
```

	couponID1	price1	basePrice1	reward1	premiumProduct1
1	89d62b666d585f03f262b1f6a6abdd2c	3.24	5.40	1.57	0
2	093c94d44333cd07a4318a0231b967c8	2.32	1.59	1.57	0
3	a10bb278883972b3e977d30f4c5d526c	7.92	2.64	1.26	1
4	0a0a18c1985e5a2580968bbc90492574	2.50	2.08	1.57	0
5	f6e2fa236e98883e972bf0fc58f292e6	12.27	2.45	1.26	0
6	a6f321b6e5cdc83deb463f46d1921df2	7.50	1.25	0.63	0

```
print(xtable(head(trn[, which(grepl("1", names(trn)))[6:7]])),
    table.placement = "H")
```

	brand1	productGroup1
1	788c2fd7ba973489e6529275c01211db	bfdca7185ad7366f965590bab1417db5
2	f6601412b868a05c118961e9d7869bc8	e8f2ae4556952195821c46ea2da1d21c
3	4e03e74099990e8529eb7e7328220af1	c06dc2b4a5f7ecdfa27a83acf7a29423
4	a38b030516e52b6f126fb96c159aba25	ff897252b17f195b5216eef189a775ab
5	a38b030516e52b6f126fb96c159aba25	c06dc2b4a5f7ecdfa27a83acf7a29423
6	f6601412b868a05c118961e9d7869bc8	113593efe095675415192b4aa1ab8653

```
print(xtable(head(trn[, which(grepl("1", names(trn)))[8:9]])),
    table.placement = "H")
```

	categoryIDs1	coupon1Used
1	b7e3b0110f96213f5087034b92be487a,4b3e2dbd4d59c43bf6982405a5f7fce7	0
2	796c5815752a170021838a78bf7fab34	1
3	b7e3b0110f96213f5087034b92be487a,667daa357f8999be0805a6d62d7d3970	0
4	b7e3b0110f96213f5087034b92be487a	1
5	b7e3b0110f96213f5087034b92be487a	0
6	b78123c345b9f8a300a7fac7a735a8bd,6f735673e0cbf770a14c38aa256011a7	1

Coupon 2:

```r
print(xtable(head(trn[, which(grepl("2", names(trn)))[1:5]])),
    table.placement = "H")
```

	couponID2	price2	basePrice2	reward2	premiumProduct2
1	f2d9a4e781b3f198f244d6c130152bb8	5.19	0.57	1.57	0
2	4947578494e36ec93b27470bf0d9b868	3.70	1.85	0.94	0
3	af9837e0d8f484f3520285f3d5f3deb7	4.17	1.39	1.26	1
4	edba887265f47c8dde81e0a24d472519	3.66	0.73	1.57	1
5	179e167fd0dc215ddf6346999a1d8ba6	5.74	0.88	1.57	1
6	7d6c167459169a4ad31357870b1f70fe	4.35	1.44	1.26	0

```r
print(xtable(head(trn[, which(grepl("2", names(trn)))[6:7]])),
    table.placement = "H")
```

	brand2	productGroup2
1	f6601412b868a05c118961e9d7869bc8	041a02d031057ba1c2e210fe6b0db0f4
2		fa5cc31a659b346bcb8c069991fcd326
3	4e03e74099990e8529eb7e7328220af1	7ebc31798e2d38e862cc336f448781a9
4	4e03e74099990e8529eb7e7328220af1	008b99d9c2fd9d58dd1483e364f8e8d3
5	4e03e74099990e8529eb7e7328220af1	e54573aaf3c08c40545b97cc5ddf1732
6		fee2c21073bf07c9cd87566fe98fcfb2

```r
print(xtable(head(trn[, which(grepl("2", names(trn)))[8:9]])),
    table.placement = "H")
```

	categoryIDs2	coupon
1	f12c4112c9ea685dc2d2705b1448bfbc,a0ae3bad823bea2ffa642bf47daa4a77	
2	f929d7624d13f00e1ca52c9edb5c03ba	
3	f12c4112c9ea685dc2d2705b1448bfbc,667daa357f8999be0805a6d62d7d3970	
4	f12c4112c9ea685dc2d2705b1448bfbc,667daa357f8999be0805a6d62d7d3970,a0ae3bad823bea2ffa642bf47daa4a77	
5	f12c4112c9ea685dc2d2705b1448bfbc,667daa357f8999be0805a6d62d7d3970	
6	7a04f5ee0a1363e6af5ca20434ed72a0,4b3e2dbd4d59c43bf6982405a5f7fce7	

Coupon 3:

```r
print(xtable(head(trn[, which(grepl("3", names(trn)))[1:5]])),
    table.placement = "H")
```

	couponID3	price3	basePrice3	reward3	premiumProduct3
1	bd6402915feb3dd717c579f3f133ba22	12.92	12.92	2.20	0
2	7d105576cee40e8f9ffd798903c9da81	3.89	0.06	2.20	0
3	4270374b0780f856a6afdf98bf759f1a	2.73	0.88	1.26	0
4	5de73019819ae0aafb3d8224b0686558	5.74	4.25	1.57	0
5	4430e468db7670b2ffbefd6cdb9d89e3	4.91	2.45	1.57	1
6	476adba5b863c0bde731b9fb9b18e64b	5.51	1.43	3.14	0

```
print(xtable(head(trn[, which(grepl("3", names(trn)))[6:7]])),
    table.placement = "H")
```

	brand3	productGroup3
1		94a7d11a8972ed94819c748a1d1b42f8
2	f6601412b868a05c118961e9d7869bc8	f5d6f7cd5f84fd41fbc4535f0fb77b89
3	304979e999d95fc59d85cba5f1633595	e006513b5c7be290e81c8bd614362d07
4	304979e999d95fc59d85cba5f1633595	b1f57b4c3876f032a2b0ea1f714b0940
5	4e03e74099990e8529eb7e7328220af1	d1b083d764bac332ee2715c369cf7ec1
6		5d7f92459adc7208db382fdf5d8f5896

```
print(xtable(head(trn[, which(grepl("3", names(trn)))[8:9]])),
    table.placement = "H")
```

	categoryIDs3	coupon3
1	796c5815752a170021838a78bf7fab34,a0ae3bad823bea2ffa642bf47daa4a77,4b3e2dbd4d59c43bf6982405a5f7fce7	
2	f929d7624d13f00e1ca52c9edb5c03ba	
3	796c5815752a170021838a78bf7fab34,4b3e2dbd4d59c43bf6982405a5f7fce7	
4	796c5815752a170021838a78bf7fab34,4b3e2dbd4d59c43bf6982405a5f7fce7	
5	796c5815752a170021838a78bf7fab34	
6	796c5815752a170021838a78bf7fab34	

0.2 Exploring the Data

0.2.1 Session information

There are two time variables which could be better formatted using `lubridate`. Additionally, it allows us to extract the time, date, and the day of the week separately. I add an indicator for weekend instead of weekday and Friday/Saturday vs the rest of the week. I wrote this function so that it could be done quickly for both variables in the training and in the test set:

```
TimeFeatures <- function(dsn, varn) {
    # don't overwrite you data
    stopifnot(!(paste0(varn, "DoW") %in% names(dsn)))

    # store time variables in this set
    dsn.time <- dsn[, c("orderID", varn)]

    # add information about order dates
    dsn.time[, varn] <- ymd_hms(dsn.time[, varn], tz = "CET")

    # split the date-time variable into date and time
    timedate <- ldply(strsplit(dsn[, varn], " "))

    # get time of day, date, and day of week alone
    dsn.time$Date <- ymd(timedate[, 1])
    dsn.time$Time <- hms(timedate[, 2])

    # get the day of the week
    dsn.time$DoW <- wday(dsn.time[, varn], label = TRUE,
        abbr = FALSE)
```

```
    # weekend or preweekend indicators
    dsn.time$Weekend <- factor(1 * (dsn.time$DoW %in%
        c("Saturday", "Sunday")))
    dsn.time$FriSat <- factor(1 * (dsn.time$DoW %in%
        c("Friday", "Saturday")))

    # merge with original data set
    names(dsn.time)[-c(1:2)] <- paste0(varn, names(dsn.time)[-c(1:2)])

    dsn <- merge(dsn.time, dsn[, names(dsn) != varn],
        by = "orderID")
    return(dsn)
}

# Whatever you do to the training set
trn <- TimeFeatures(trn, "orderTime")
trn <- TimeFeatures(trn, "couponsReceived")

# try if you can to do the same to the test set
tst <- TimeFeatures(tst, "orderTime")
tst <- TimeFeatures(tst, "couponsReceived")
```

Now that we have created the time variable as a backbone, we can make some interesting plots:
Also notice that orders are numbered as they made.

```
qplot(orderTime, orderID, data = trn)
```

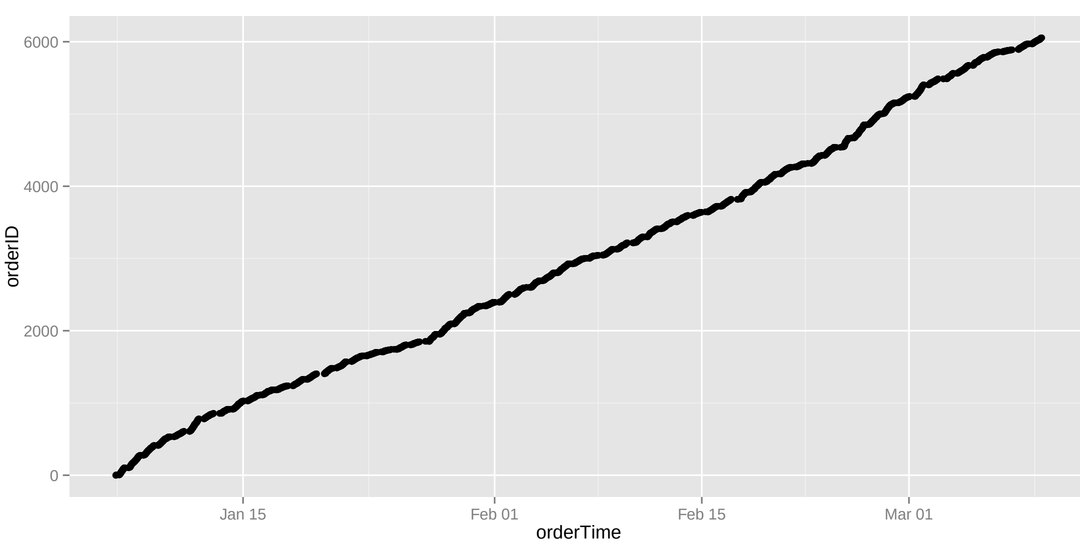


Figure 1: As time passes, new orders are processed. Notice that the number of new orders is generally smooth, indicating that customers are buying items consistently.

We can verify that orders are numbered in increasing order through time:

```
sum(sapply(1:nrow(trn), function(i) sum(trn$orderTime[i] <
    trn$orderTime & trn$orderID[i] > trn$orderID)))

## [1] 0
```

This creates a bit of a puzzle in our next plot:

```
qplot(couponsReceived, orderTime, data = trn)
```

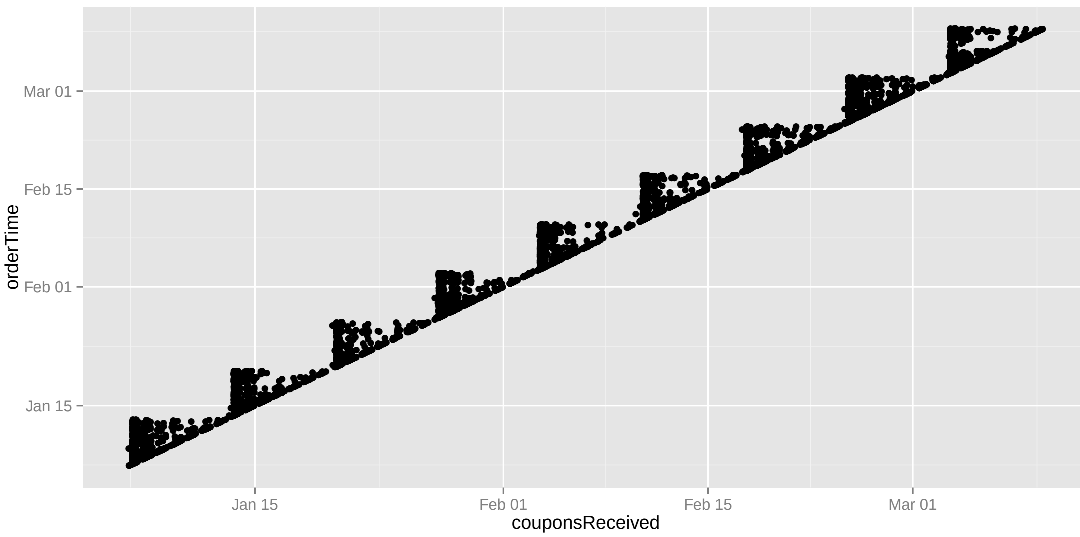


Figure 2: Orders alway follow receiving of coupons, but the odd shapes indicate that there is structure to the data beyond the fact that shoppers get coupons before purchasing any products.

It seems that customers can not use coupons indefinetly. In fact, the difference between two points never exceeds one week.

```
# longest time to act on coupon
max(difftime(trn$orderTime, trn$couponsReceived, units = "days"))

## Time difference of 6.401204 days

# shortest time to act on coupon
min(difftime(trn$orderTime, trn$couponsReceived, units = "secs"))

## Time difference of 6 secs
```

The earliest a fastest a customer acts on a coupon is 6 seconds! Lets consider the possibility that customers are receiving coupons in gigantic batches. Then it makes sense that the coupons will have one-week limits, and then a new batch of coupons are sent out. For example, if we take the first time and add 7 days, any customer receiving coupons has only until the 7th day after any customer in that batch receives a coupon to place an order.
We can use this idea to demarcate time into batches. This function assumes

- Coupons are sent in batches
- Coupons are sent at the same time
- A person has 7 days after the coupons are sent to use them, regardless of when the coupon is "received"

In addition to batchID, we can add the "pseudo-expiration" date of the coupon and "date sent" variables to our data:

```
GetBatchInfo <- function(initial_batch.ymd_hms, nbatch = 10,
    weeks2expire = 1, train = trn, test = tst) {
    # start sending coupons: initial_batch.ymd_hms =
    # '2015-01-06 1:00:00' initial_batch.ymd_hms =
    # '2015-01-03 1:00:00' How many batches are there?
    # nbatch = 10 how long are coupons valid?
    # weeks2expire = 1 what is the training set?  train
    # = trn what is the test set?  test = tst

    require(lubridate)
    stopifnot(is.POSIXct(ymd_hms(initial_batch.ymd_hms)) &
        is.POSIXt(ymd_hms(initial_batch.ymd_hms)))

    # expiration date one week after coupons are sent
    couponLengthValid <- weeks(weeks2expire)

    # batches go out when the last batch expires
    weeksbtwnbatches <- weeks2expire

    # coupons start on
    batch.start <- ymd_hms(initial_batch.ymd_hms, tz = "CET")

    # make data frame
    couponBatches <- data.frame(sendDate = batch.start +
        (0:(nbatch - 1)) * couponLengthValid, expireDate = batch.start +
        1:nbatch * couponLengthValid, batch = factor(1:nbatch))

    # create time interval
    couponBatches$validInterval <- with(couponBatches,
        interval(sendDate, expireDate))

    # give the training set batchID
    train$batchID <- 0
    train$couponsExpire <- batch.start + years(1)
    train$couponsSent <- batch.start
    for (i in 1:nbatch) {
        orderinbatch <- train$orderTime %within% couponBatches$validInterval[i] &
            train$couponsReceived %within% couponBatches$validInterval[i]
        couponinbatch <- train$couponsReceived %within%
            couponBatches$validInterval[i]
        if (sum(orderinbatch) > 0)
            train$batchID[orderinbatch] <- couponBatches$batch[i]
        if (sum(couponinbatch) > 0) {
            train$couponsSent[couponinbatch] <- couponBatches$couponsSent[i]
            train$couponsExpire[couponinbatch] <- couponBatches$couponsExpire[i]
        }
    }
    train$batchID <- as.factor(train$batchID)
    train$dataset <- "train"

    test$batchID <- 0
```

```
    test$couponsExpire <- batch.start + years(1)
    test$couponsSent <- batch.start
    for (i in 1:nbatch) {
        orderinbatch <- test$orderTime %within% couponBatches$validInterval[i] &
            test$couponsReceived %within% couponBatches$validInterval[i]
        couponinbatch <- test$couponsReceived %within%
            couponBatches$validInterval[i]
        if (sum(orderinbatch) > 0)
            test$batchID[orderinbatch] <- couponBatches$batch[i]
        if (sum(couponinbatch) > 0) {
            test$couponsSent[couponinbatch] <- couponBatches$couponsSent[i]
            test$couponsExpire[couponinbatch] <- couponBatches$couponsExpire[i]
        }
    }
    test$batchID <- as.factor(test$batchID)
    test$dataset <- "test"

    # create the plots
    batch.invalid <- data.frame(batch.violation = c(train$batchID ==
        0, test$batchID == 0))

    # plots help us make sure that the batches make
    # sense
    p1 <- ggplot() + geom_point(data = cbind(rbind(train,
        test), batch.invalid), aes(x = couponsReceived,
        y = orderTime, shape = dataset, size = batch.violation))

    p2 <- ggplot() + geom_rect(data = couponBatches,
        aes(xmin = sendDate, xmax = sendDate + weeks(1),
            ymin = sendDate, ymax = expireDate, fill = batch),
        alpha = I(0.4)) + geom_point(data = cbind(rbind(train,
        test), batch.invalid), aes(x = couponsReceived,
        y = orderTime, shape = dataset), size = I(0.9))

    p3 <- ggplot() + geom_point(data = cbind(rbind(train,
        test), batch.invalid), aes(x = couponsReceived,
        y = orderTime, color = batchID), size = I(0.9))

    train <- train[, -which(names(train) == "dataset")]
    test <- test[, -which(names(test) == "dataset")]

    results <- list(train = train, test = test, plots = list(p1,
        p2, p3))

    return(results)
}
```

If we start putting the batches together using Tuesday January 1st 2015 at 9:00 am we get some ugly results:

```
# this batch leads to bad results
batchres <- GetBatchInfo("2015-01-06 00:00:00")
```

Customers are using coupons much more than a week after they were sent out if the first batch was sent at 9:00 am: we have orders where the customer is receiving the coupon in one batch, but using them in a different batch (batch violations):

```
batchres$plots[[1]]
batchres$plots[[2]]
batchres$plots[[3]]
```

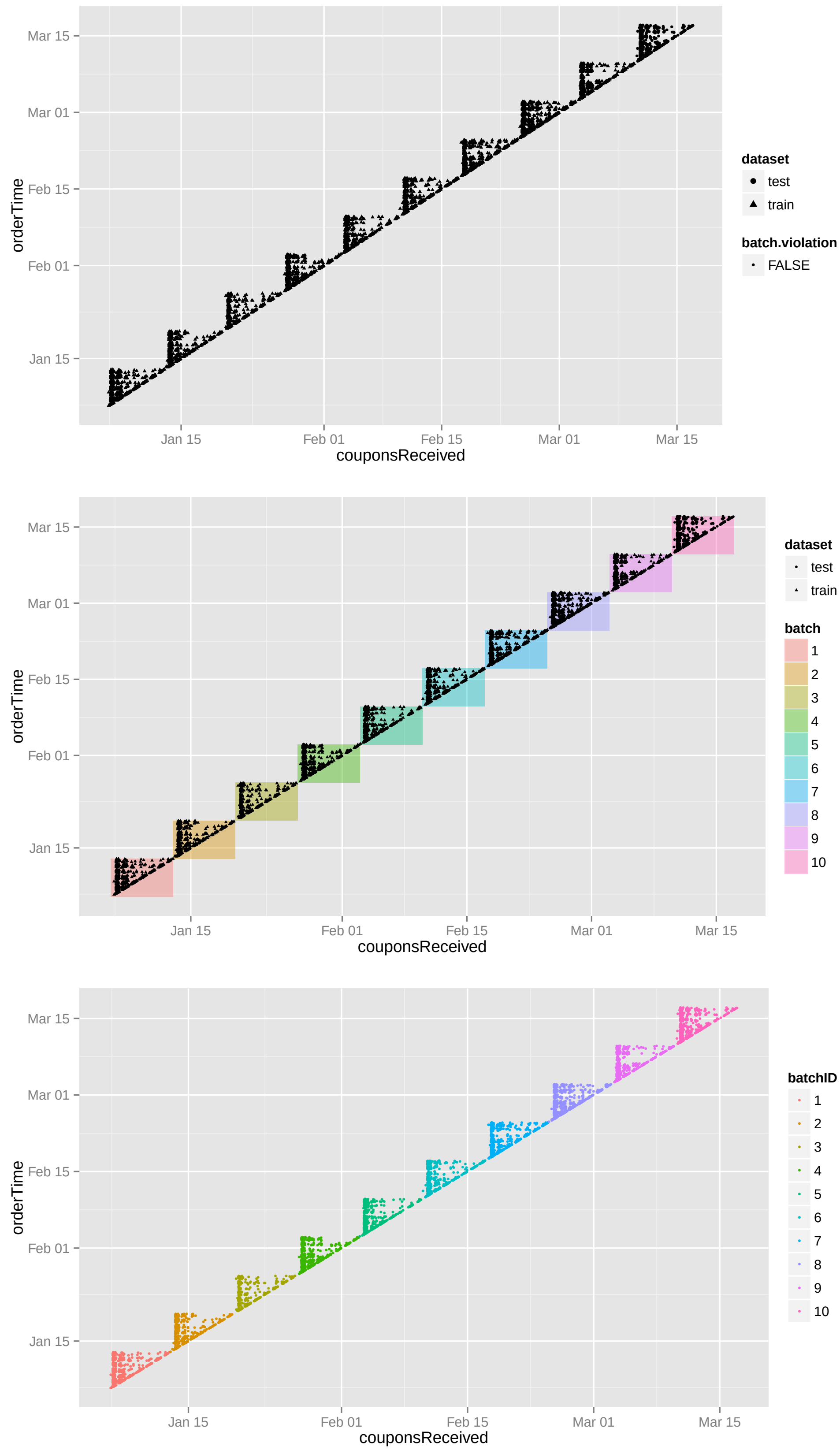
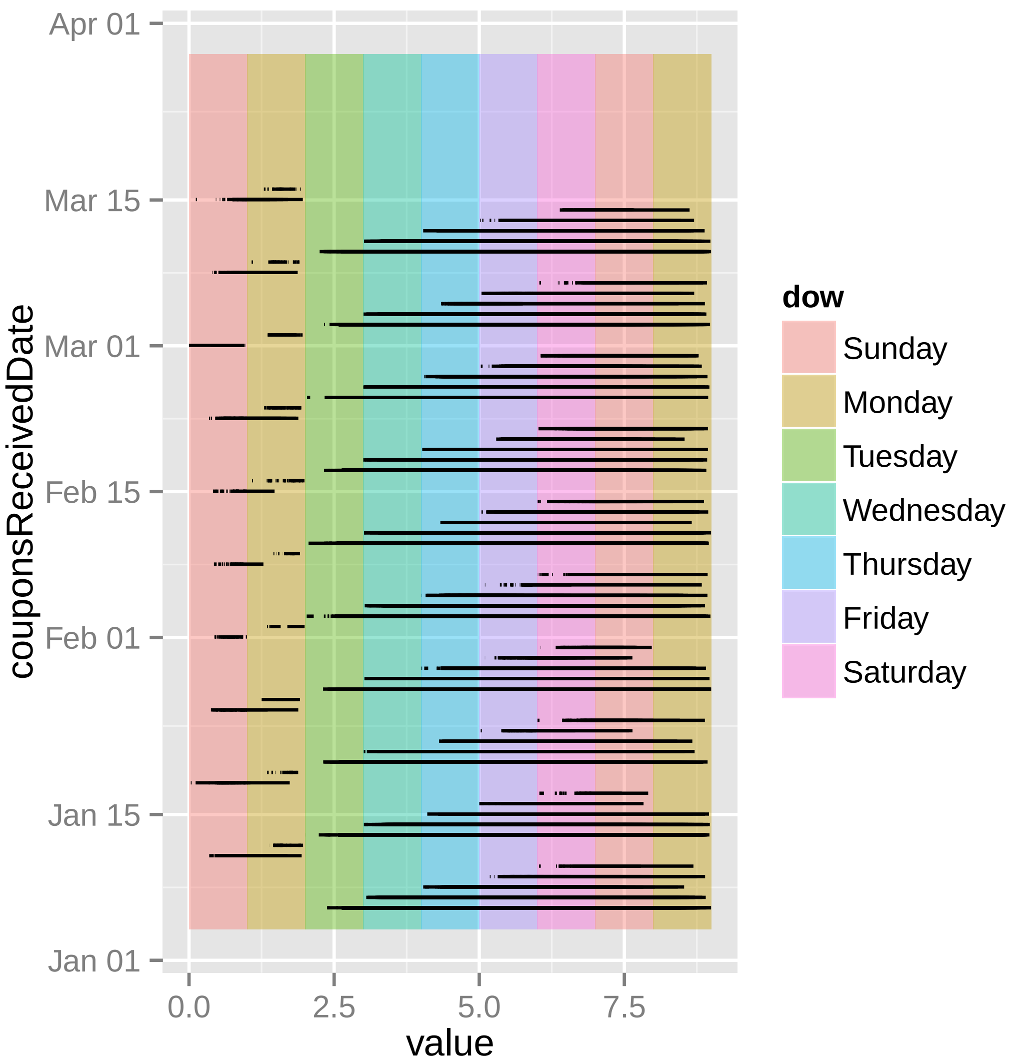


Figure 3: Orders alway follow receiving of coupons, but the odd shapes indicate that there is structure to the data beyond the fact that shoppers get coupons before purchasing any products.

we may be able to find gaps on the Tuesdays that help us figure out when the coupons are and we can plot these as well:

```
# combine the results
d <- rbind(trn, tst)

# the only issue is if they get coupons at the very
# last moment and then order something this means
# they order coupons on a Tuesday and get coupons
# on a Tuesday batch.timing =
# d[which(d$orderTimeDoW == 'Tuesday' &
# d$couponsReceivedDoW %in%
# c('Monday','Tuesday')),]
min(d$couponsReceivedDate)

## [1] "2015-01-06 UTC"

wday(min(d$couponsReceivedDate), label = TRUE)

## [1] Tues
## 7 Levels: Sun < Mon < Tues < Wed < ... < Sat

# start all the weeks between on Sunday
start.weeks <- ymd_hms("2015-01-04 00:00:00", tz = "CET")
d$StartOfWeek <- start.weeks
for (i in 1:12) {
    chng <- d$couponsReceived > start.weeks + weeks(i)
    if (sum(chng) > 0) {
        d$StartOfWeek[which(chng)] <- start.weeks +
            weeks(i)
    }
}

# lets look for gaps
b.m <- melt(d, c("orderID", "couponsReceivedDate",
    "StartOfWeek"), c("orderTime", "couponsReceived"))
b.m$value <- as.numeric(difftime(b.m$value, b.m$StartOfWeek,
    units = "days"))

DeliveryDates <- data.frame(daystr = rep(0:floor(max(b.m$value))),
    dayend = rep(1:ceiling(max(b.m$value))), lowery = start.weeks,
    uppery = start.weeks + 12 * weeks(1), dow = wday(start.weeks +
        days(1) * (0:floor(max(b.m$value))), label = TRUE,
        abbr = FALSE))

ggplot() + geom_rect(data = DeliveryDates, aes(xmin = daystr,
    xmax = dayend, ymin = lowery, ymax = uppery, fill = dow),
    alpha = I(0.4)) + geom_line(data = b.m, aes(x = value,
    y = couponsReceivedDate, group = orderID))
```



So using Tuesdays at midnight to demarcate batches seems like a safe solution. We can save the `order -> batch` assgnments as:

```
# this batch leads to bad results
batchres <- GetBatchInfo("2015-01-06 00:00:00")
write.csv(batchres$train[, c("orderID", "batchID",
    "couponsSent", "couponsExpire")], file = "trainingset.batchID.csv",
    row.names = FALSE, quote = FALSE, na = "")
write.csv(batchres$test[, c("orderID", "batchID", "couponsSent",
    "couponsExpire")], file = "testingset.batchID.csv",
    row.names = FALSE, quote = FALSE, na = "")
```