R Notebook

Code ▾

Introduction

I updated this notebook in order to make it easier for beginners to understand how it works. It is not a secret that R's data.table (https://cran.r-project.org/web/packages/data.table/) has quite exotic syntax. At first sight, it is fairly obscure, but it is fully comparable (https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html) with SQL. Moreover, **data.table** is extremely fast (https://github.com/Rdatatable/data.table/wiki/Benchmarks-%3A-Grouping) and memory efficient, especially with *by reference* operations. Having learnt **data.table** you can't help using it in your projects (https://i.pinimg.com/originals/ce/82/ba/ce82ba5fd7dbb77738f872f4e7167b38.png) ;-)

Preparations

Let's load the packages and provide some basic parameters for our future model:

Hide

```
library(data.table)
library(lightgbm)
library(ggplot2)

set.seed(0)

cores = 8
epochs = 2500
eta = 0.075

h <- 28 # forecast horizon
max_lags <- 420 # number of observations to shift by
tr_last <- 1913 # last training day
fday <- as.IDate("2016-04-25") # first day to forecast
nrows <- Inf
```

Also we need auxilary functions:

- **free()** just calls a garbage collector

Hide

```
free <- function() invisible(gc())
```

- **create_dt()** creates a training or testing data table from a wide-format file with leading zeros removed. Pay attention to the cool feature of the **melt()** function and **data.table**: we can choose columns by regex patterns. Notice that I refer to the columns of the second table using *i.* prefix when merging the data tables (e.g. **i.event_name_1** is a column of the **cal** table).

Hide

```
create_dt <- function(is_train = TRUE, nrows = Inf) {

  if (is_train) { # create train set
    dt <- fread("./data/sales_train_validation.csv", nrows = nrows)
    cols <- dt[, names(.SD), .SDcols = patterns("^d_")]
    dt[, (cols) := transpose(lapply(transpose(.SD),
                                    function(x) {
                                      i <- min(which(x > 0))
                                      x[1:i-1] <- NA
                                      x})), .SDcols = cols]
    free()
  } else { # create test set
    dt <- fread("./data/sales_train_validation.csv", nrows = nrows,
                drop = paste0("d_", 1:(tr_last-max_lags))) # keep only max_lags days from the tr
ain set
    dt[, paste0("d_", (tr_last+1):(tr_last+2*h)) := 0] # add empty columns for forecasting
  }

  dt <- na.omit(melt(dt,
                     measure.vars = patterns("^d_"),
                     variable.name = "d",
                     value.name = "sales"))

  cal <- fread("./data/calendar.csv")
  dt <- dt[cal, `:=`(date = as.IDate(i.date, format="%Y-%m-%d"), # merge tables by reference
                     wm_yr_wk = i.wm_yr_wk,
                     event_name_1 = i.event_name_1,
                     snap_CA = i.snap_CA,
                     snap_TX = i.snap_TX,
                     snap_WI = i.snap_WI), on = "d"]

  prices <- fread("./data/sell_prices.csv")
  dt[prices, sell_price := i.sell_price, on = c("store_id", "item_id", "wm_yr_wk")] # merge agai
n
}
```

- **create_fea()** adds lags, rolling features and time variables to the data table. **frollmean()** is a fast rolling function to calculate means on sliding window. Notice how we use *:=* operation to add new columns.

Hide

```
create_fea <- function(dt) {
  dt[, `:=`(d = NULL, # remove useless columns
            wm_yr_wk = NULL)]

  cols <- c("item_id", "store_id", "state_id", "dept_id", "cat_id", "event_name_1")
  dt[, (cols) := lapply(.SD, function(x) as.integer(factor(x))), .SDcols = cols] # convert chara
cter columns to integer
  free()

  lag <- c(7, 28)
  lag_cols <- paste0("lag_", lag) # lag columns names
  dt[, (lag_cols) := shift(.SD, lag), by = id, .SDcols = "sales"] # add lag vectors

  win <- c(7, 28) # rolling window size
  roll_cols <- paste0("rmean_", t(outer(lag, win, paste, sep="_"))) # rolling features columns n
ames
  dt[, (roll_cols) := frollmean(.SD, win, na.rm = TRUE), by = id, .SDcols = lag_cols] # rolling
 features on lag_cols

  dt[, `:=`(wday = wday(date), # time features
            mday = mday(date),
            week = week(date),
            month = month(date),
            year = year(date))]
}
```

The next step is to prepare data for training. The data set itself is quite large, so we constantly need to collect garbage. Here I use the last 28 days for validation:

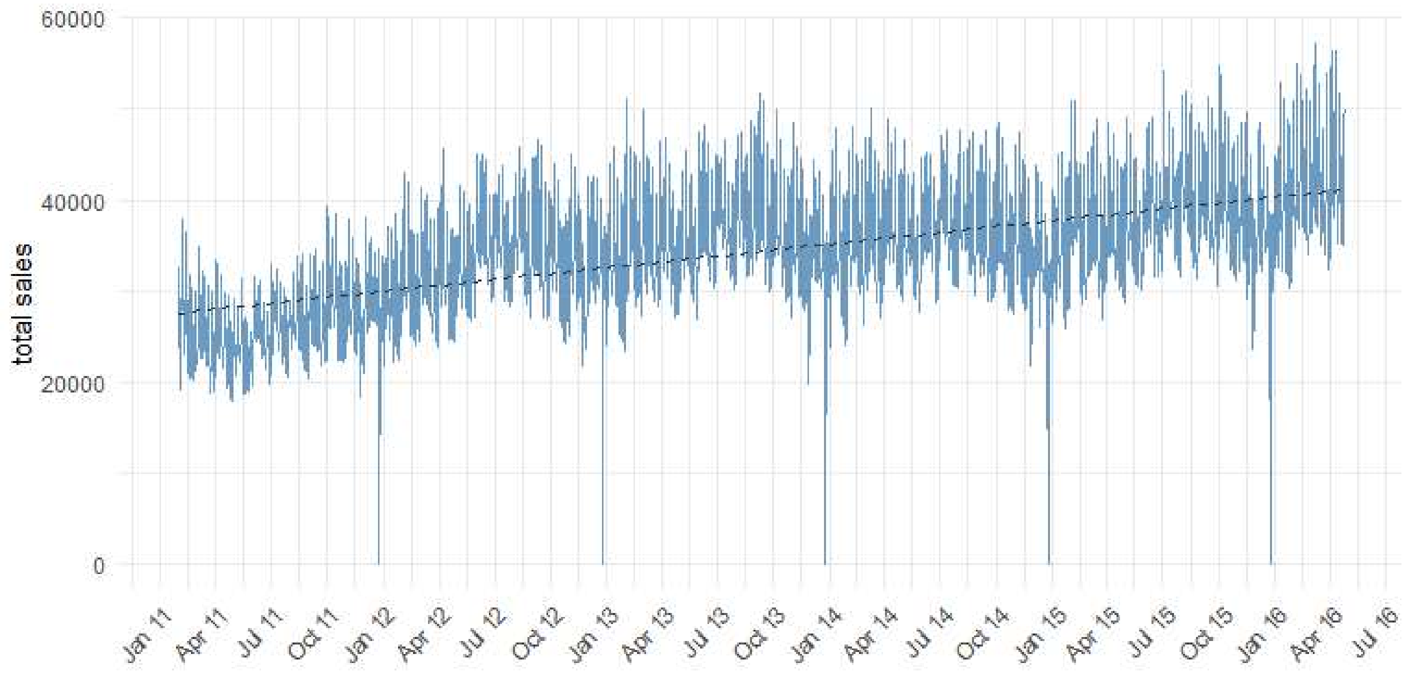
Hide

```
tr <- create_dt()
free()
```

Just to get the general idea let's plot grouped sales across all items:

Hide

```
tr[, .(sales = unlist(lapply(.SD, sum))), by = "date", .SDcols = "sales"
   ][, ggplot(.SD, aes(x = date, y = sales)) +
       geom_line(size = 0.3, color = "steelblue", alpha = 0.8) +
       geom_smooth(method='lm', formula= y~x, se = FALSE, linetype = 2, size = 0.5, color = "gra
y20") +
       labs(x = "", y = "total sales") +
       theme_minimal() +
       theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position="none") +
       scale_x_date(labels=scales::date_format ("%b %y"), breaks=scales::date_breaks("3 months"
))]
```



Hide

```
free()
```

We can see a trend and peaks at the end of each year. If only our model could model seasonality and trend… Ok, let's proceed to the next step and prepare a dataset for lightgbm:

Hide

```
create_fea(tr)
free()

tr <- na.omit(tr) # remove rows with NA to save memory
free()

idx <- tr[date <= max(date)-h, which = TRUE] # indices for training
y <- tr$sales
tr[, c("id", "sales", "date") := NULL]
free()
```

Hide

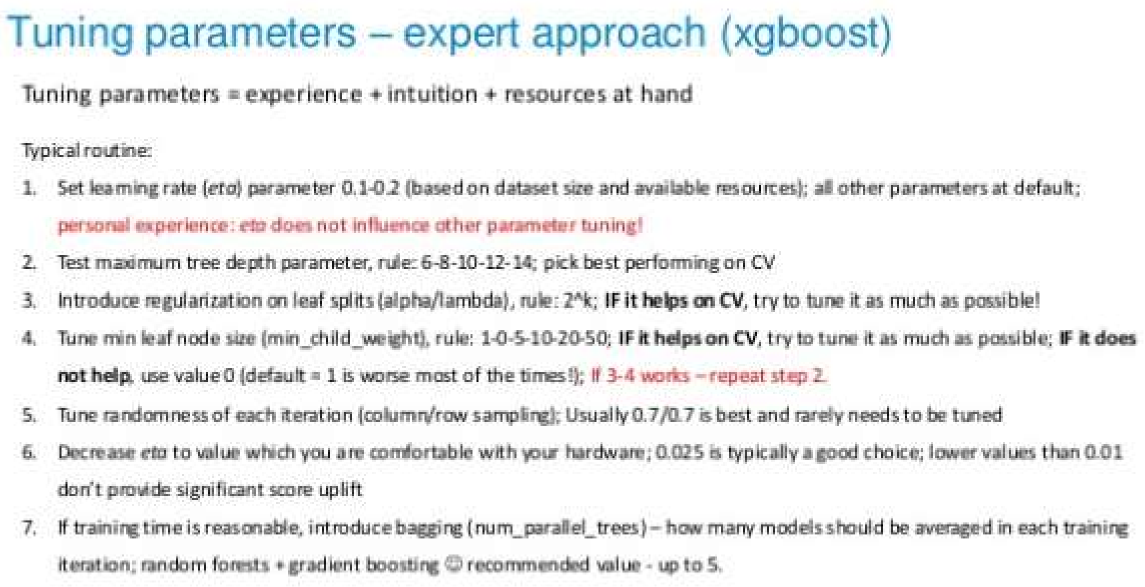
```
tr <- data.matrix(tr)
free()
```

Hide

```
cats <- c("item_id", "store_id", "state_id", "dept_id", "cat_id",
          "wday", "mday", "week", "month", "year",
          "snap_CA", "snap_TX", "snap_WI") # list of categorical features
xtr <- lgb.Dataset(tr[idx, ], label = y[idx], categorical_feature = cats) # construct lgb datase
t
xval <- lgb.Dataset(tr[-idx, ], label = y[-idx], categorical_feature = cats)

rm(tr, y, cats, idx)
free()
```

Training model

It's time to train our not so simple model with **poisson** (https://en.wikipedia.org/wiki/Poisson_distribution) loss, which is suitable for counts. Lately I tune tree models manually following this approach:



Tuning parameters – expert approach (xgboost)

Tuning parameters = experience + intuition + resources at hand

Typical routine:

1. Set learning rate (*eta*) parameter 0.1-0.2 (based on dataset size and available resources); all other parameters at default; personal experience: *eta* does not influence other parameter tuning!
2. Test maximum tree depth parameter, rule: 6-8-10-12-14; pick best performing on CV
3. Introduce regularization on leaf splits (alpha/lambda), rule: 2^k; **IF it helps on CV**, try to tune it as much as possible!
4. Tune min leaf node size (min_child_weight), rule: 1-0-5-10-20-50; **IF it helps on CV**, try to tune it as much as possible; **IF it does not help**, use value 0 (default = 1 is worse most of the times!); If 3-4 works – repeat step 2.
5. Tune randomness of each iteration (column/row sampling); Usually 0.7/0.7 is best and rarely needs to be tuned
6. Decrease *eta* to value which you are comfortable with your hardware; 0.025 is typically a good choice; lower values than 0.01 don't provide significant score uplift
7. If training time is reasonable, introduce bagging (num_parallel_trees) – how many models should be averaged in each training iteration; random forests + gradient boosting ☺ recommended value - up to 5.

Hide

```
p <- list(objective = "poisson",
          metric ="rmse",
          force_row_wise = TRUE,
          learning_rate = eta,
          num_leaves = 128,
          min_data = 100,
          sub_feature = 0.8,
          sub_row = 0.75,
          bagging_freq = 1,
          lambda_l2 = 0.1,
          nthread = cores)

m_lgb <- lgb.train(params = p,
                   data = xtr,
                   nrounds = epochs, # 4000,
                   valids = list(val = xval),
                   early_stopping_rounds = 400,
                   eval_freq = 50)
```

```
[LightGBM] [Warning] Unknown parameter: force_row_wise
[LightGBM] [Warning] Met categorical feature which contains sparse values. Consider renumbering
to consecutive integers started from zero
[LightGBM] [Warning] Unknown parameter: force_row_wise
[LightGBM] [Info] Total Bins 4575
[LightGBM] [Info] Number of data: 44235060, number of used features: 21
[LightGBM] [Warning] Unknown parameter: force_row_wise
[LightGBM] [Info] Start training from score 0.355880
[1]:    val's rmse:3.43623
[51]:   val's rmse:2.07949
[101]:  val's rmse:2.04125
[151]:  val's rmse:2.0389
[201]:  val's rmse:2.03642
[251]:  val's rmse:2.0341
[301]:  val's rmse:2.03024
[351]:  val's rmse:2.02917
[401]:  val's rmse:2.02715
[451]:  val's rmse:2.02654
[501]:  val's rmse:2.02495
[551]:  val's rmse:2.0224
[601]:  val's rmse:2.02168
[651]:  val's rmse:2.01971
[701]:  val's rmse:2.01819
[751]:  val's rmse:2.01754
[801]:  val's rmse:2.0166
[851]:  val's rmse:2.0171
[901]:  val's rmse:2.01622
[951]:  val's rmse:2.01597
[1001]: val's rmse:2.01557
[1051]: val's rmse:2.01436
[1101]: val's rmse:2.014
[1151]: val's rmse:2.01372
[1201]: val's rmse:2.01283
[1251]: val's rmse:2.0124
[1301]: val's rmse:2.01167
[1351]: val's rmse:2.01117
[1401]: val's rmse:2.01063
[1451]: val's rmse:2.01015
[1501]: val's rmse:2.00996
[1551]: val's rmse:2.00971
[1601]: val's rmse:2.00927
[1651]: val's rmse:2.00899
[1701]: val's rmse:2.00881
[1751]: val's rmse:2.00854
[1801]: val's rmse:2.00857
[1851]: val's rmse:2.00827
[1901]: val's rmse:2.00802
[1951]: val's rmse:2.0079
[2001]: val's rmse:2.00799
[2051]: val's rmse:2.00815
[2101]: val's rmse:2.00836
[2151]: val's rmse:2.00828
[2201]: val's rmse:2.00816
```

```
[2251]: val's rmse:2.00807
[2301]: val's rmse:2.00796
[2351]: val's rmse:2.00836
```

Hide

```
cat("Best score:", m_lgb$best_score, "at", m_lgb$best_iter, "iteration")
```

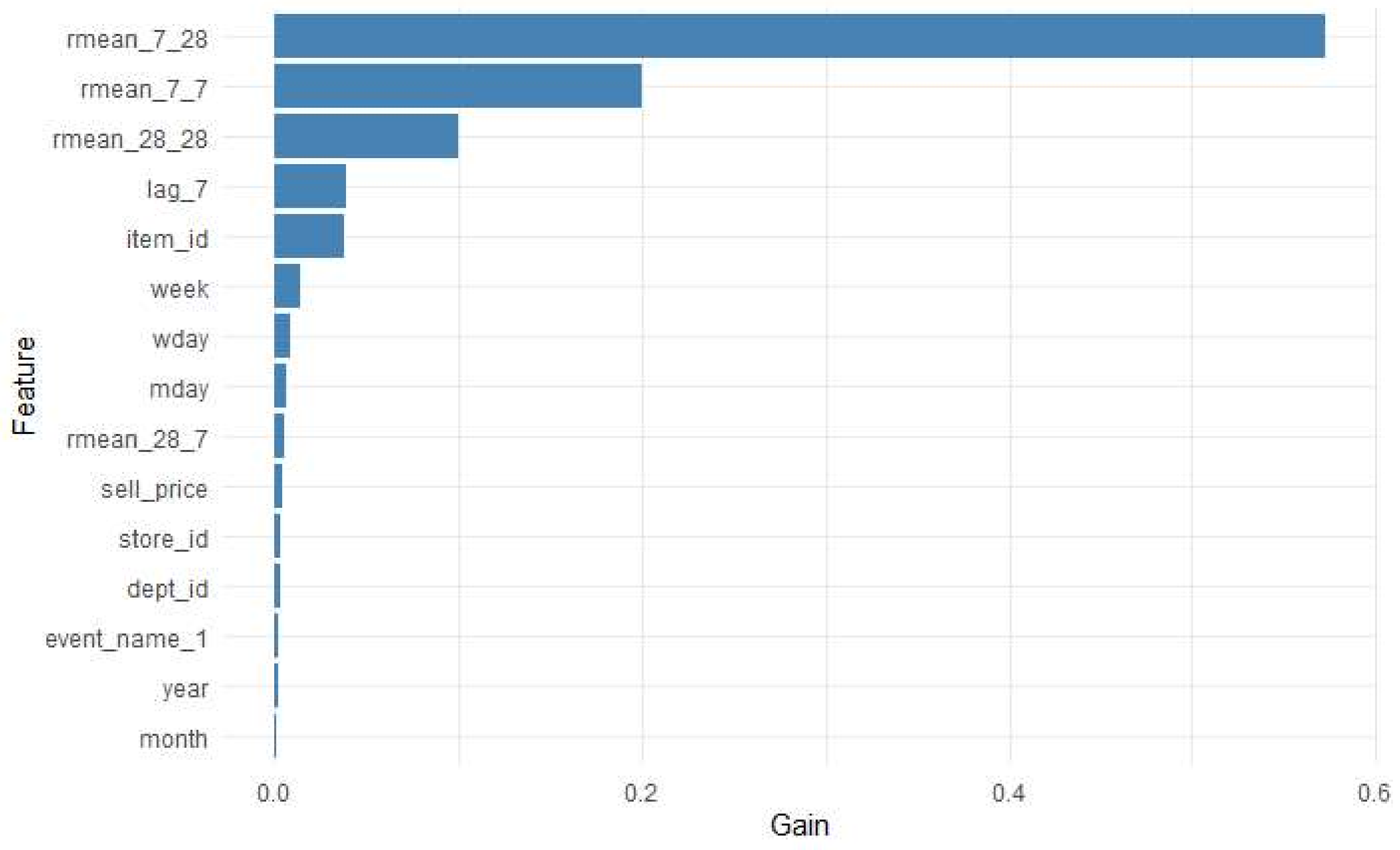
```
Best score: 2.007784 at 1958 iteration
```

Hide

```
imp <- lgb.importance(m_lgb)

rm(xtr, xval, p)
free()
```

Hide

```
imp[order(-Gain)
    ][1:15, ggplot(.SD, aes(reorder(Feature, Gain), Gain)) +
        geom_col(fill = "steelblue") +
        xlab("Feature") +
        coord_flip() +
        theme_minimal()]
```



We can see that rolling features are very important. I have to admit that chaining with **data.table** is very convenient, especially when we want to use **j** for side effects.

Forecasting

And now the hard part. As we are using 7-day lag features we have to forecast day by day in order to use the latest predictions for the current day. This slows down the forecasting process tremendously. Also, tree models are unable to extrapolate that’s why here we use some kind of “magic” multiplier which slightly inflates predictions.

Hide

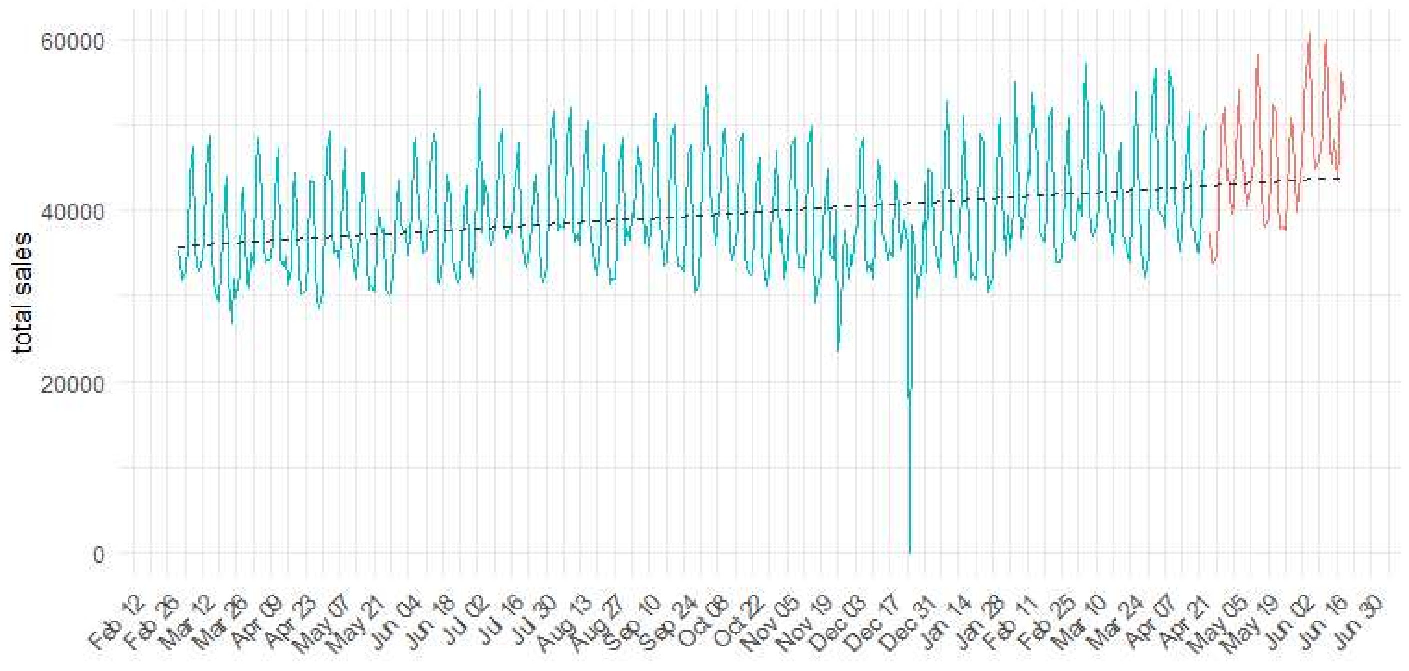
```
te <- create_dt(FALSE, nrows)

for (day in as.list(seq(fday, length.out = 2*h, by = "day"))){
  cat(as.character(day), " ")
  tst <- te[date >= day - max_lags & date <= day]
  create_fea(tst)
  tst <- data.matrix(tst[date == day][, c("id", "sales", "date") := NULL])
  te[date == day, sales := 1.03*predict(m_lgb, tst, n_jobs = cores)]
}
```

```
2016-04-25  2016-04-26  2016-04-27  2016-04-28  2016-04-29  2016-04-30  2016-05-01  2016-05-02
2016-05-03  2016-05-04  2016-05-05  2016-05-06  2016-05-07  2016-05-08  2016-05-09  2016-05-10
2016-05-11  2016-05-12  2016-05-13  2016-05-14  2016-05-15  2016-05-16  2016-05-17  2016-05-18
2016-05-19  2016-05-20  2016-05-21  2016-05-22  2016-05-23  2016-05-24  2016-05-25  2016-05-26
2016-05-27  2016-05-28  2016-05-29  2016-05-30  2016-05-31  2016-06-01  2016-06-02  2016-06-03
2016-06-04  2016-06-05  2016-06-06  2016-06-07  2016-06-08  2016-06-09  2016-06-10  2016-06-11
2016-06-12  2016-06-13  2016-06-14  2016-06-15  2016-06-16  2016-06-17  2016-06-18  2016-06-19
```

Let’s plot our predictions along with given values:

Hide

```
te[, .(sales = unlist(lapply(.SD, sum))), by = "date", .SDcols = "sales"
   ][, ggplot(.SD, aes(x = date, y = sales, colour = (date < fday))) +
       geom_line() +
       geom_smooth(method='lm', formula= y~x, se = FALSE, linetype = 2, size = 0.3, color = "gra
y20") +
       labs(x = "", y = "total sales") +
       theme_minimal() +
       theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position="none") +
       scale_x_date(labels=scales::date_format ("%b %d"), breaks=scales::date_breaks("14 day"))]
```



Well, not great not terrible.

Finally, the last step is to save predictions having casted them into the wide format:

Hide

```
te[date >= fday
   ][date >= fday+h, id := sub("validation", "evaluation", id)
     ][, d := paste0("F", 1:28), by = id
       ][, dcast(.SD, id ~ d, value.var = "sales")
         ][, fwrite(.SD, "sub_dt_lgbV2.csv")]
```

```
NULL
```