Prophet Customer Forecast

Introduction

This notebook will outline the steps to produce the prophet forecast, incorporating historical as well as future data for catalog circulation, marketing investment, and the J.Jill promotional calendar. The first step will be loading the required libraries to handle our data in R.

```
library(data.table)
library(zoo)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(prophet)
```

```
## Loading required package: Rcpp
```

```
## Loading required package: rlang
```

```
##
## Attaching package: 'rlang'
```

```
## The following object is masked from 'package:data.table':
##
##     :=
```

Loading the Data

There are 4 required files used to build this forecast. They are:

1. Historical customer counts, monthly
2. Historical and future investment numbers, monthly
3. Historical and future catalog circulation, monthly
4. Historical and future promotional calendar

First we will investigate the customer data set.

Customer Data

```
# sourced from alteryx workflow: Create Customer Data Set on latest masterXXk.yxdb fi
le

proph = read.csv('processed_customers.csv')
```

Investment Data

```
# sourced from Maria David, manually processed (no alteryx workflow)

inv = read.csv('processed_investments.csv')
```

Catalog Circulation Data

```
# sourced from alteryx workflow: Prep Catalog History 20180613 on Jim's latest circul
ation calendar
circ = read.csv('processed_circ.csv')
```

Promotional Calenendar Data

```
# sourced from alteryx workflow: Prop Promo File 20180717 on Kaitlyn's latest promo c
alendar
promos = read.csv('processed_promos.csv')
```

Now that we have all of the data prepped and ready for modeling, we can merge it into one larger file.

Merge training data

```
regress = merge(proph, inv, by=c('FISCAL_YR', 'FISCAL_MO', 'Segment_Channel'), all.y=
TRUE)
regress$ds <- as.yearmon(paste(regress$FISCAL_YR, regress$FISCAL_MO, sep = "-"))
setnames(regress, 'monthly_customers', 'y')

regress2 = merge(regress, promos, by = c('FISCAL_YR', 'FISCAL_MO', 'FISCAL_QTR', 'Seg
ment_Channel'), all.y=TRUE)
regress3 = merge(regress2, circ, by = c('FISCAL_YR', 'FISCAL_MO', 'FISCAL_QTR', 'Segm
ent'), all.y=TRUE)
regress3 = as.data.table(regress3)

train = regress3[FISCAL_YR >= 2016]
```

Define the forecasting Function

The primary function used to build this forecast is called `make_forecast` defined below. It takes two arguments, a data frame (df) as well as a threshold (threshold). The threshold is expected to be a year for which we want to forecast (i.e. 2019 or 2020). If you want to forecast for 2020, you should set the threshold to 2020, otherwise use 2019.

The forecast works by instantiating a prophet object (m) and setting a variety of parmeters relating to holidays, seasonality, and changepoints. Once the prophet object has been instantiated, a number of regressor variables can be added (any covariate column in the data frame argument can be incorporated as a regressor to the model.)

```
make_forecast <- function(df, threshold) {
  #threshold = 2019
  #df = regress2[Segment_Channel == 'React_Direct1']
  df = as.data.table(df)
  df = df[FISCAL_YR <= threshold]
  m <- prophet(seasonality.mode = 'additive', holidays = holidays, holidays.prior.sca
le = .05, seasonality.prior.scale = 5, changepoint.prior.scale = .034, weekly.seasona
lity = FALSE, daily.seasonality = FALSE, yearly.seasonality = FALSE)
  m <- add_regressor(m, name = 'circ_score')
  #m <- add_regressor(m, name = 'investment')
  #m <- add_regressor(m, name = 'peel_off_depth')
  ##m <- add_regressor(m, name = 'post_card_depth')
  #m <- add_regressor(m, name = 'free_ship_depth')
  #m <- add_regressor(m, name = 'flash_sale_depth')
  #m <- add_regressor(m, name = 'global_depth')
  #m <- add_regressor(m, name = 'fp_entire_depth')
  #m <- add_regressor(m, name = 'sp_entire_depth')
  m <- add_seasonality(m, name='yearly', period=365.25, fourier.order=10, prior.scale
= 0.05)
  #m <- add_seasonality(m, name = 'quarterly', period = 365.25/4, fourier.order = 5,
prior.scale = 15)
  #m <- add_seasonality(m, name = 'monthly', period = 365.25/12, fourier.order = 3, p
rior.scale = 10)
  m <- fit.prophet(m, df[FISCAL_YR <= (threshold - 1)])
  future <- make_future_dataframe(m, periods = 12, freq = 'month')
  future$circ_score = df$circ_score
  #future$investment = df$investment  # FIXME this needs to be correctly specified
  #future$peel_off_depth = df$peel_off_depth
  ##future$post_card_depth = df$post_card_depth
  #future$free_ship_depth = df$free_ship_depth
  #future$flash_sale_depth = df$flash_sale_depth
  #future$global_depth = df$global_depth
  #future$fp_entire_depth = df$fp_entire_depth
  #future$sp_entire_depth = df$sp_entire_depth
  fcst <- predict(m, future)
  prophet_plot_components(m, fcst)
  return(fcst)
}
```

Note that the function takes in a vector called `holidays.` We will need to define this vector to contain information on all of the J.Jill holidays. We do this manually by referencing the J.Jill fiscal calendar.

```
# HOLIDAYS
new_years <- tibble(
  holiday = 'new_years',
  ds = as.Date(c('2017-12-01', '2018-11-01',
                 '2019-12-01')),
  lower_window = 0,
  upper_window = 1
)

womens_day <- tibble(
  holiday = 'womens_day',
  ds = as.Date(c('2016-02-01', '2017-02-01', '2018-02-01',
                 '2019-02-01')),
  lower_window = 0,
  upper_window = 1
)

good_friday <- tibble(
  holiday = 'good_friday',
  ds = as.Date(c('2016-02-01', '2017-03-01', '2018-02-01',
                 '2019-03-01')),
  lower_window = 0,
  upper_window = 1
)

easter <- tibble(
  holiday = 'easter',
  ds = as.Date(c('2016-02-01', '2017-03-01', '2018-02-01',
                 '2019-03-01')),
  lower_window = 0,
  upper_window = 1
)

mothers_day <- tibble(
  holiday = 'mothers_day',
  ds = as.Date(c('2016-04-01', '2017-04-01', '2018-04-01',
                 '2019-04-01')),
  lower_window = 0,
  upper_window = 1
)

memorial <- tibble(
  # on average +17k customers
  holiday = 'memorial',
  ds = as.Date(c('2016-05-01', '2017-05-01', '2018-04-01',
                 '2019-04-01')),
  lower_window = 0,
  upper_window = 1
)

independence <- tibble(
```

```r
  # on average +5k customers
  holiday = 'independence',
  ds = as.Date(c('2016-06-01', '2017-06-01', '2018-05-01',
                 '2019-05-01')),
  lower_window = 0,
  upper_window = 1
)

labor <- tibble(
  # on average +20k customers
  holiday = 'labor',
  ds = as.Date(c('2016-08-01', '2017-08-01', '2018-08-01',
                 '2019-08-01')),
  lower_window = 0,
  upper_window = 1
)

thanksgiv <- tibble(
  holiday = 'thanksgiv',
  ds = as.Date(c('2016-10-01', '2017-10-01', '2018-10-01',
                 '2019-10-01')),
  lower_window = 0,
  upper_window = 1
)

black_fri <- tibble(
  holiday = 'black_fri',
  ds = as.Date(c('2016-10-01', '2017-10-01', '2018-10-01',
                 '2019-10-01')),
  lower_window = 0,
  upper_window = 1
)

cyber_monday <- tibble(
  holiday = 'cyber_monday',
  ds = as.Date(c('2016-11-01', '2017-11-01', '2018-10-01',
                 '2019-11-01')),
  lower_window = 0,
  upper_window = 1
)

christ_eve <- tibble(
  holiday = 'christ_eve',
  ds = as.Date(c('2016-11-01', '2017-11-01', '2018-11-01',
                 '2019-11-01')),
  lower_window = 0,
  upper_window = 1
```

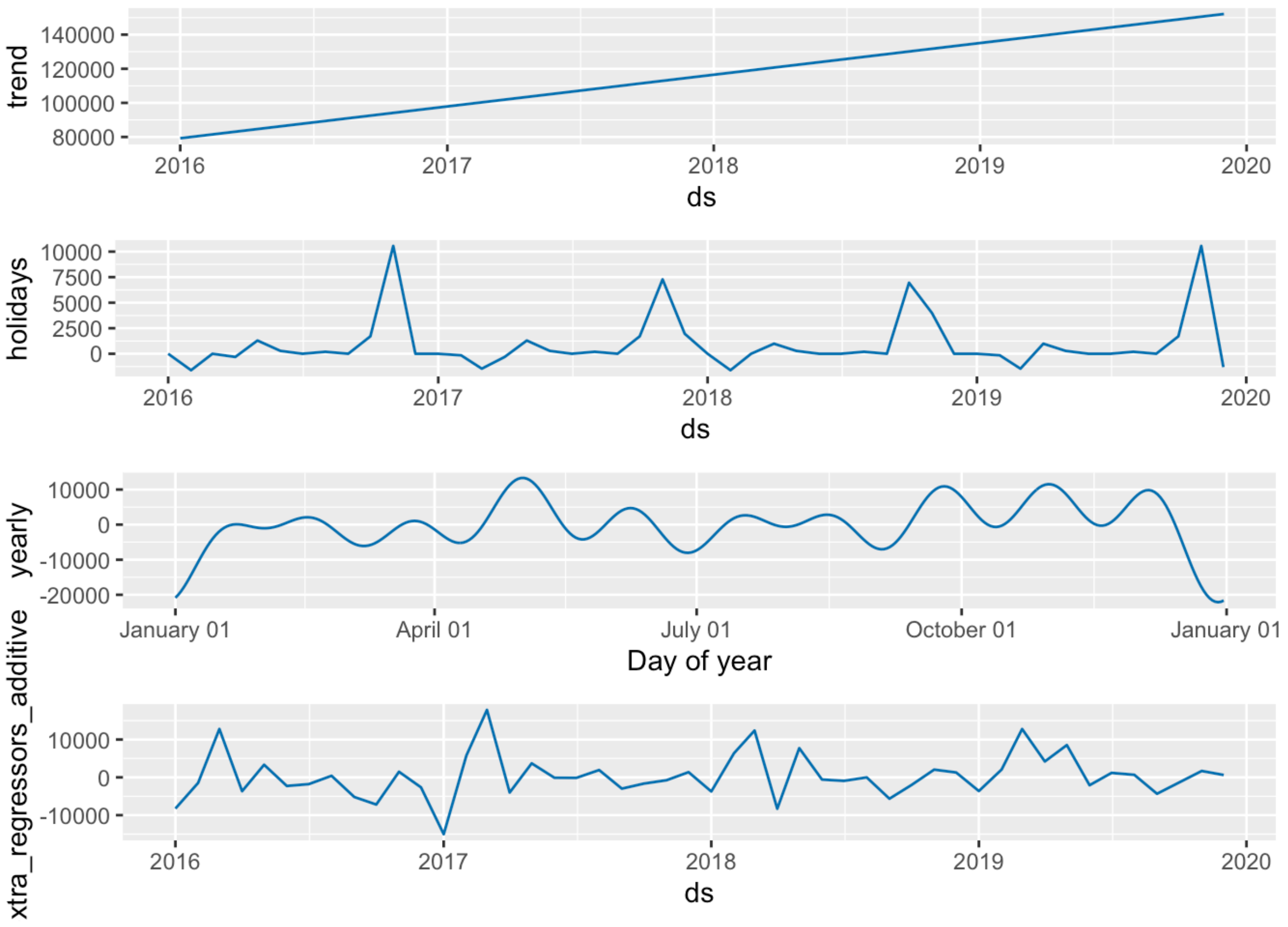
```
)

christ <- tibble(
  holiday = 'christ',
  ds = as.Date(c('2016-11-01', '2017-11-01', '2018-11-01',
                 '2019-11-01')),
  lower_window = 0,
  upper_window = 1
)

nye <- tibble(
  holiday = 'nye',
  ds = as.Date(c('2016-11-01', '2017-12-01', '2018-11-01',
                 '2019-11-01')),
  lower_window = 0,
  upper_window = 1
)







holidays <- bind_rows(nye, christ, cyber_monday, christ_eve,black_fri, thanksgiv, lab
or, independence, memorial, mothers_day, easter, good_friday, womens_day, new_years)
```

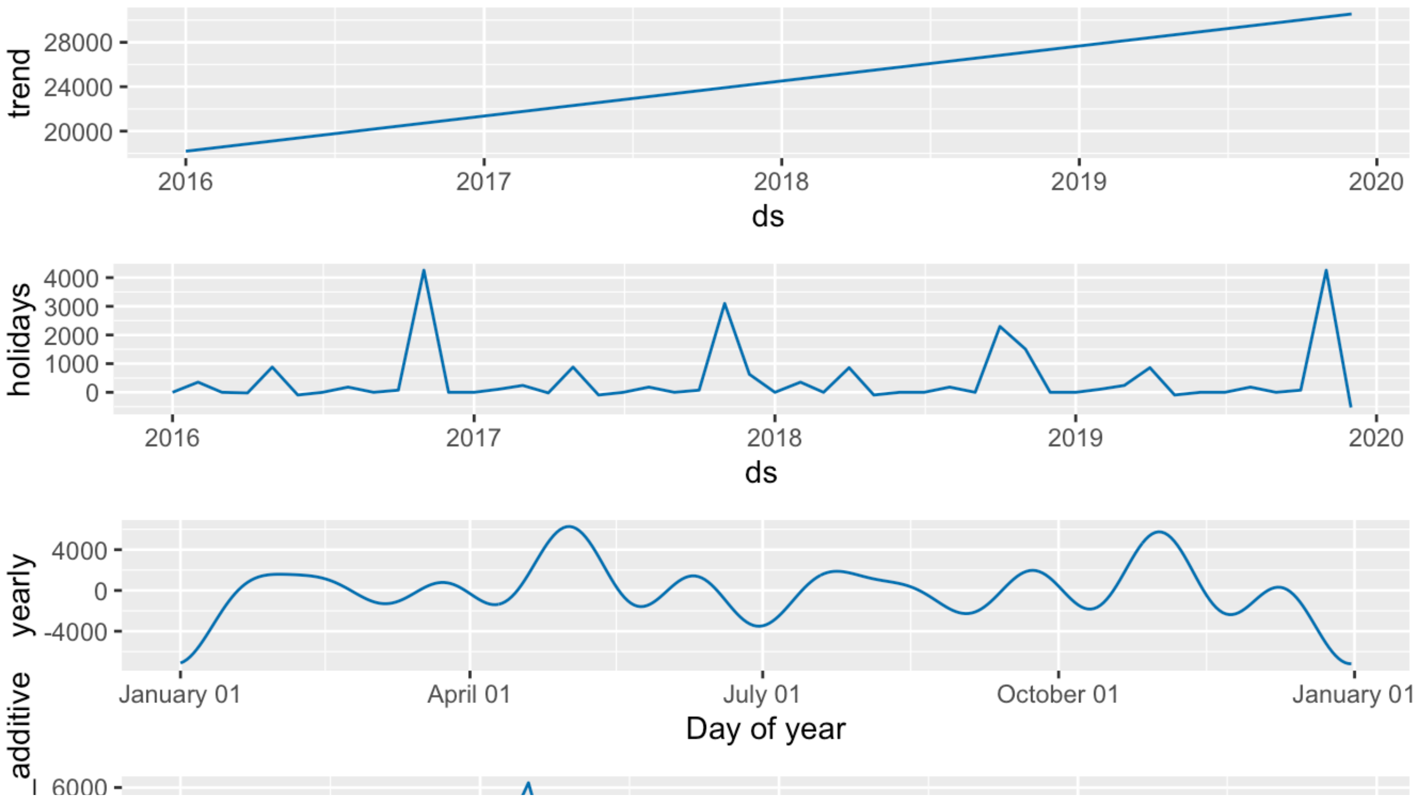
Now that our function is defined and our holidays are coded, the next step is to produce a forecast for each of the segment channels in question. We will use R's built in `dplyr` package to do this programmatically.

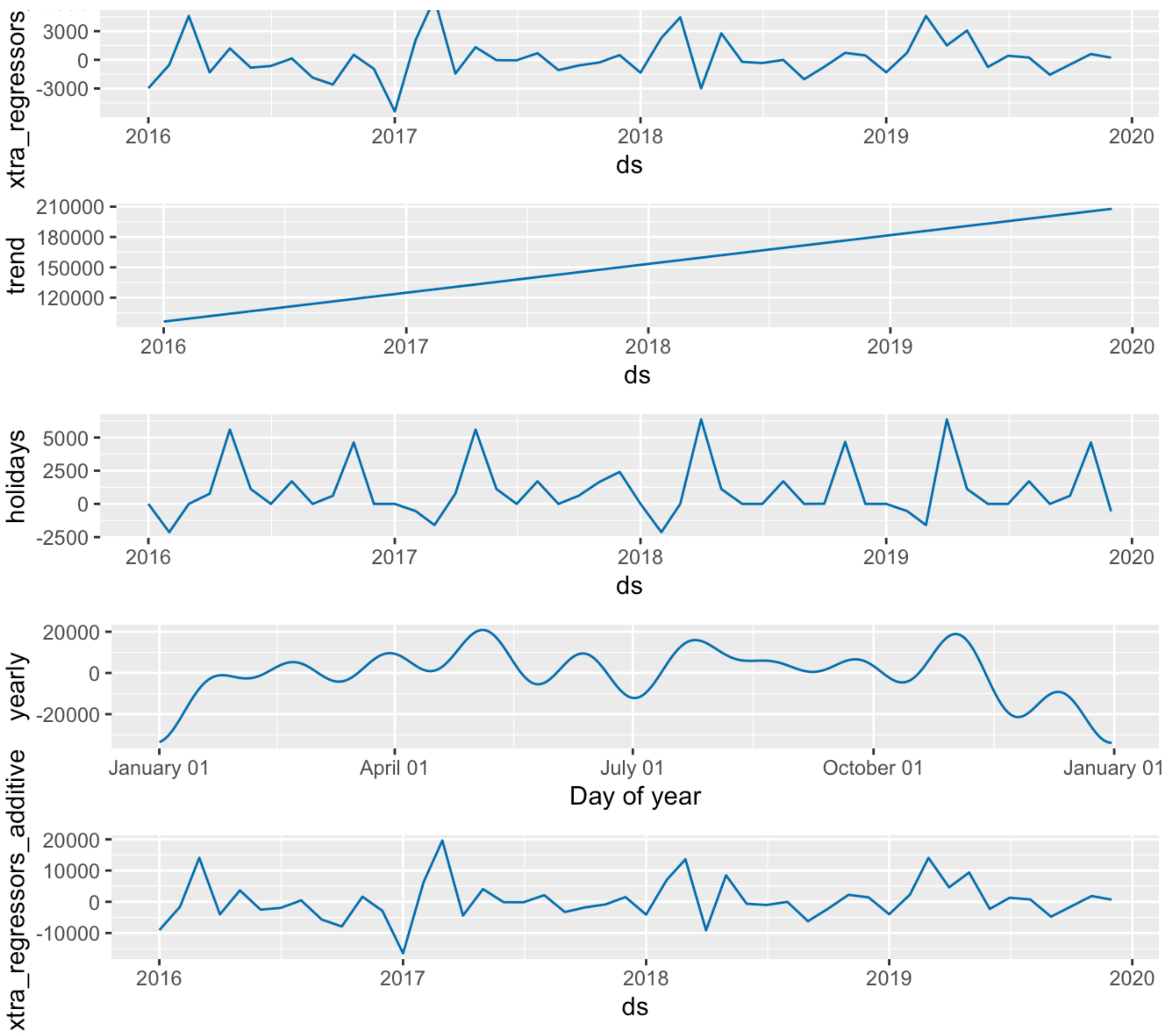
Call our function for each segment channel in our dataframe

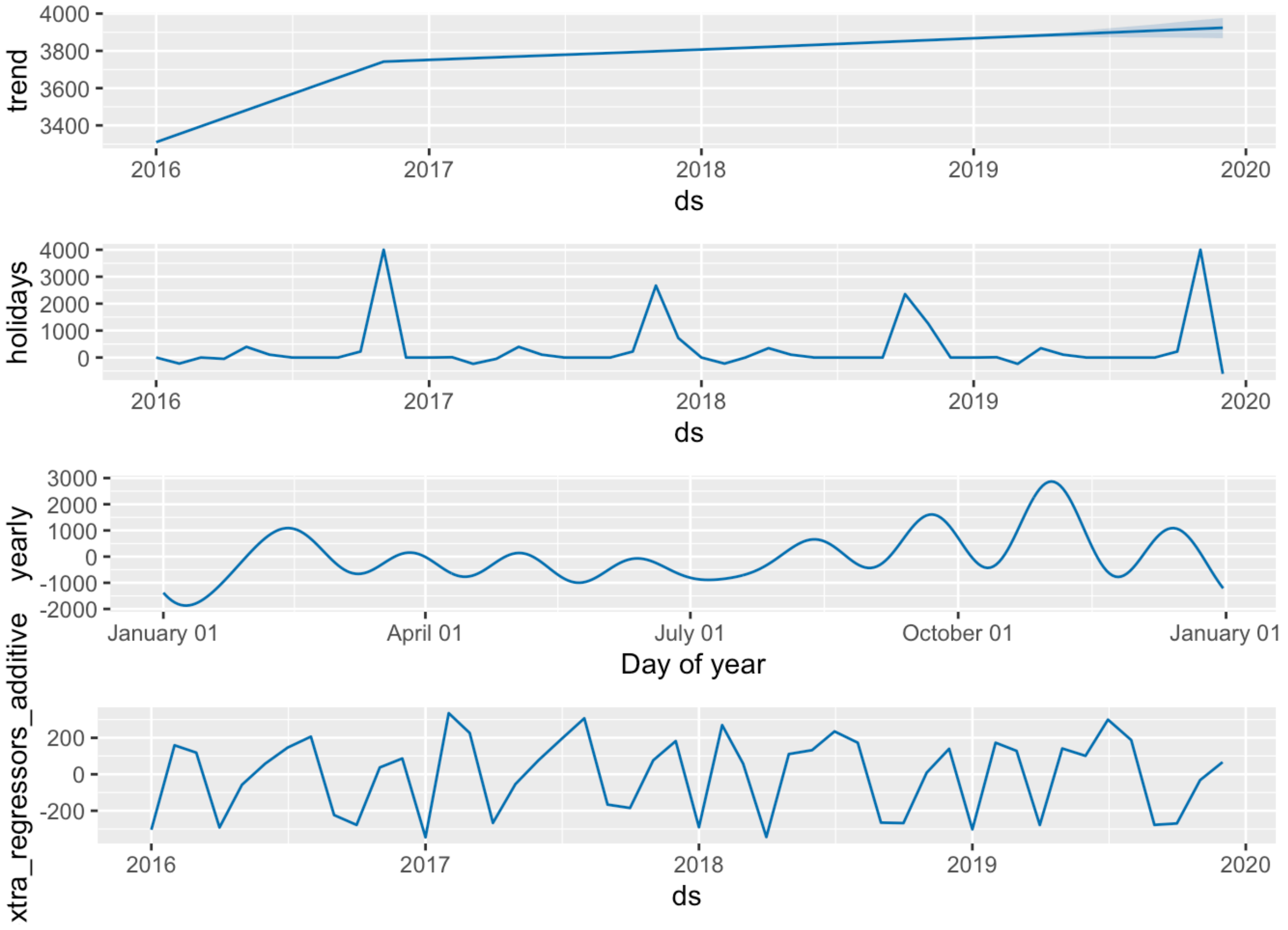
This block of code will run our make forecast function for 2019 9 times (note that we are grouping by Segment Channel). This means the forecast will run once for each segment channel that exists in our data frame.

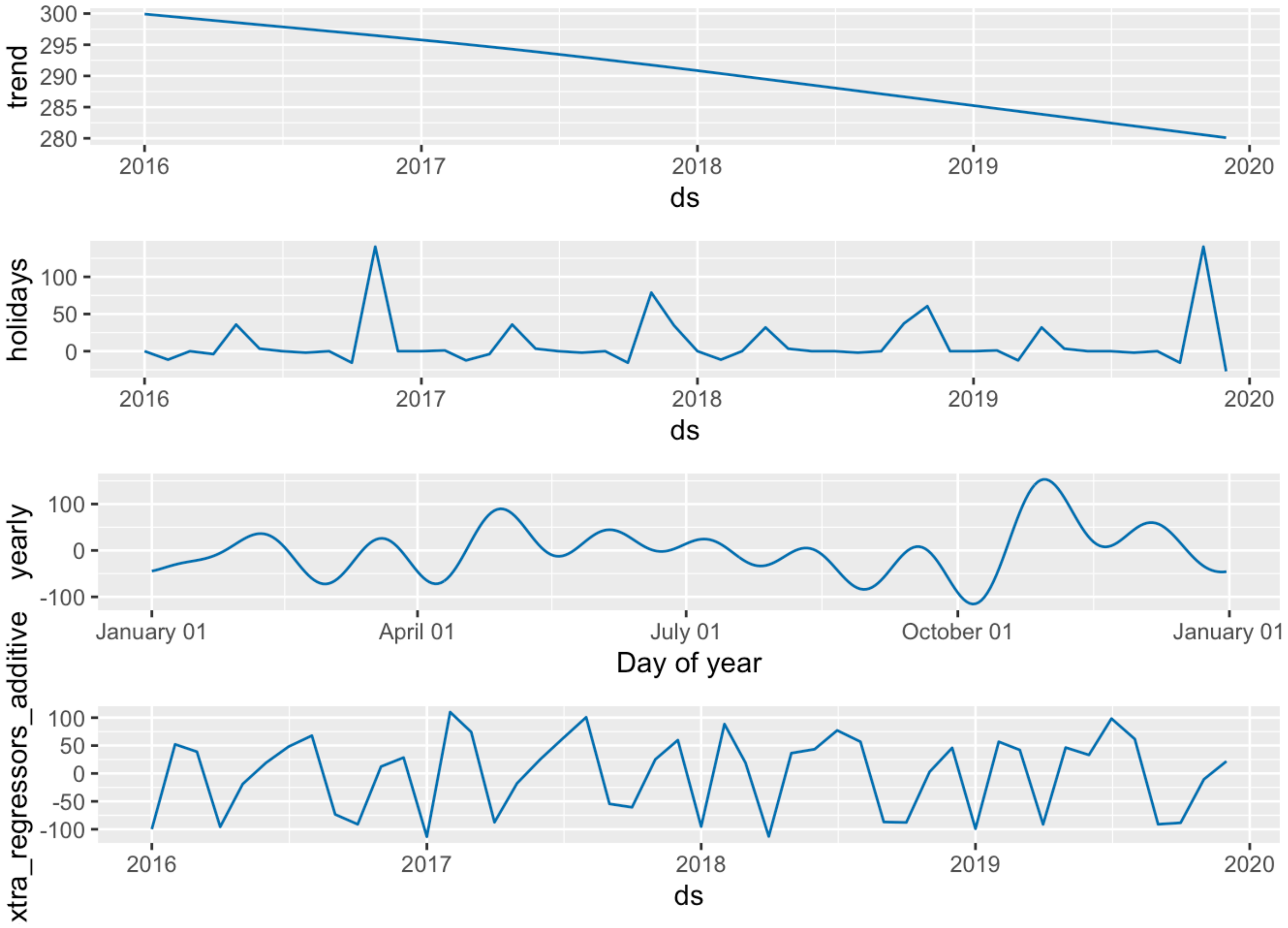
```
fcst = train %>%
  group_by(Segment_Channel) %>%
  do(make_forecast(., 2019)) %>%
  dplyr::select(ds, Segment_Channel, yhat)
```

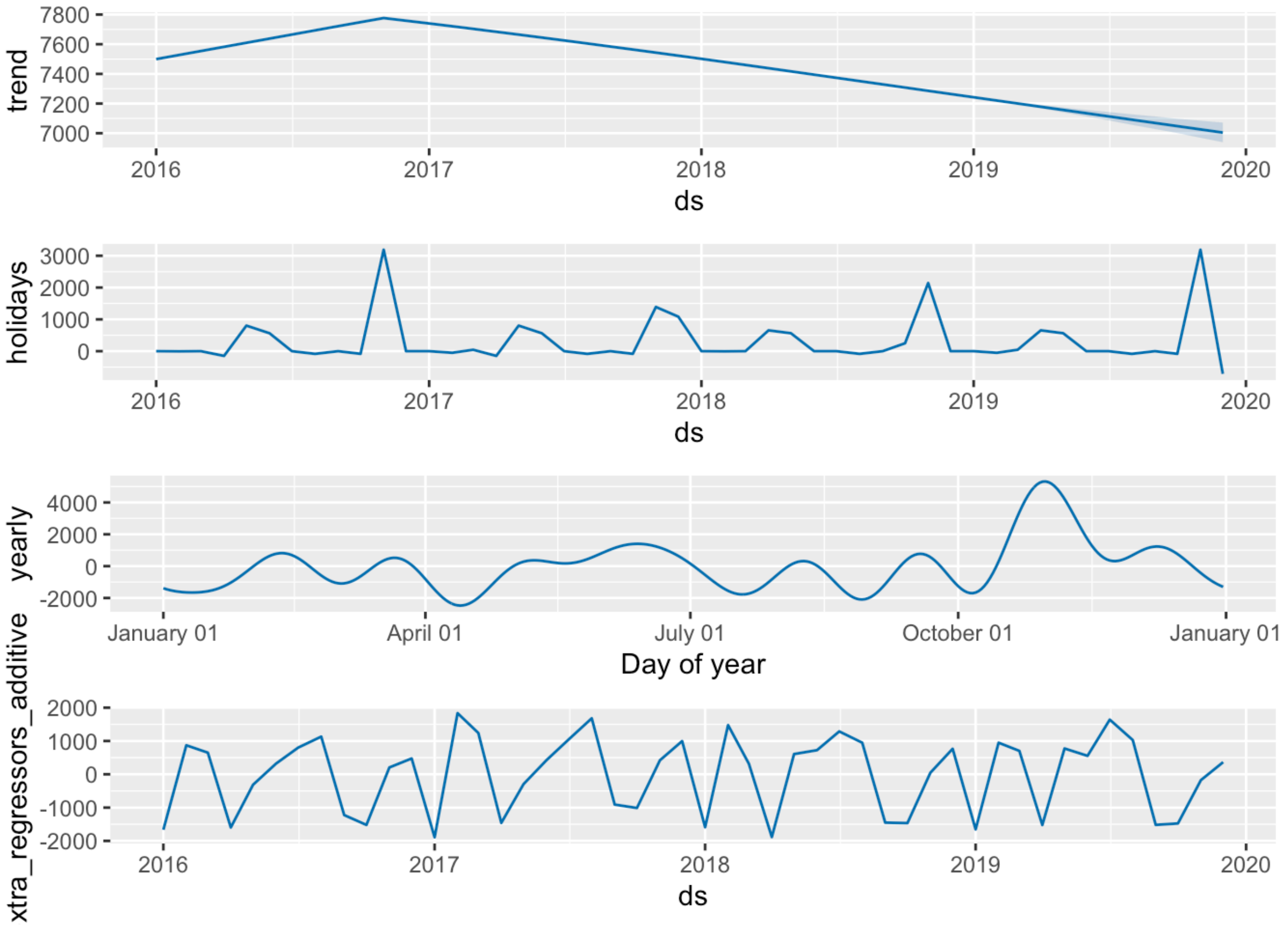

trend
140000
120000
100000
80000
2016
2017
2018
2019
2020
ds
holidays
10000
7500
5000
2500
0
yearly
10000
0
-10000
-20000
January 01
April 01
July 01
October 01
Day of year
xtra_regressors_additive

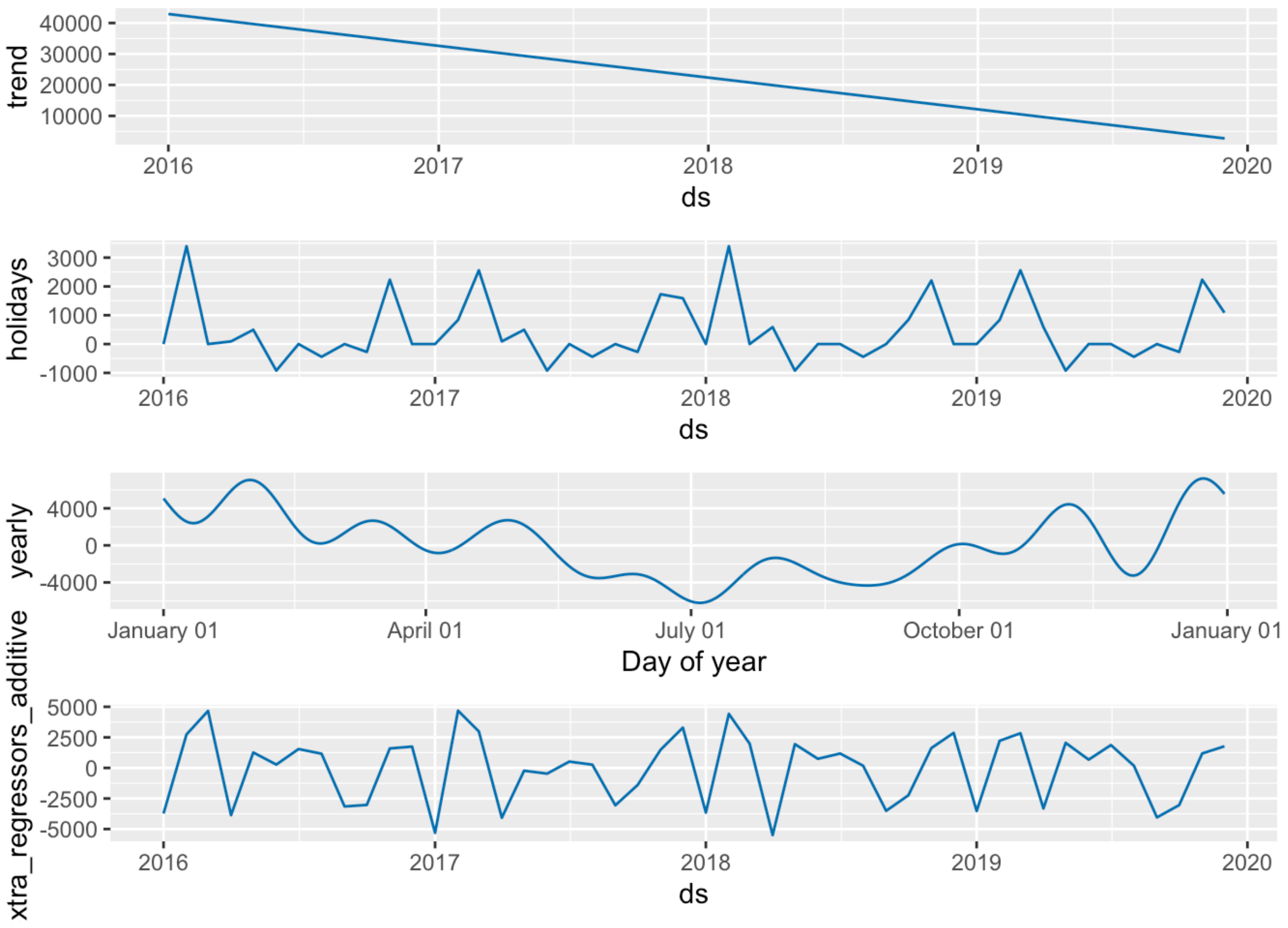

trend
28000
24000
20000
2016
2017
2018
2019
2020
ds
holidays
4000
3000
2000
1000
0
yearly
4000
0
-4000
January 01
April 01
July 01
October 01
Day of year
_additive
6000

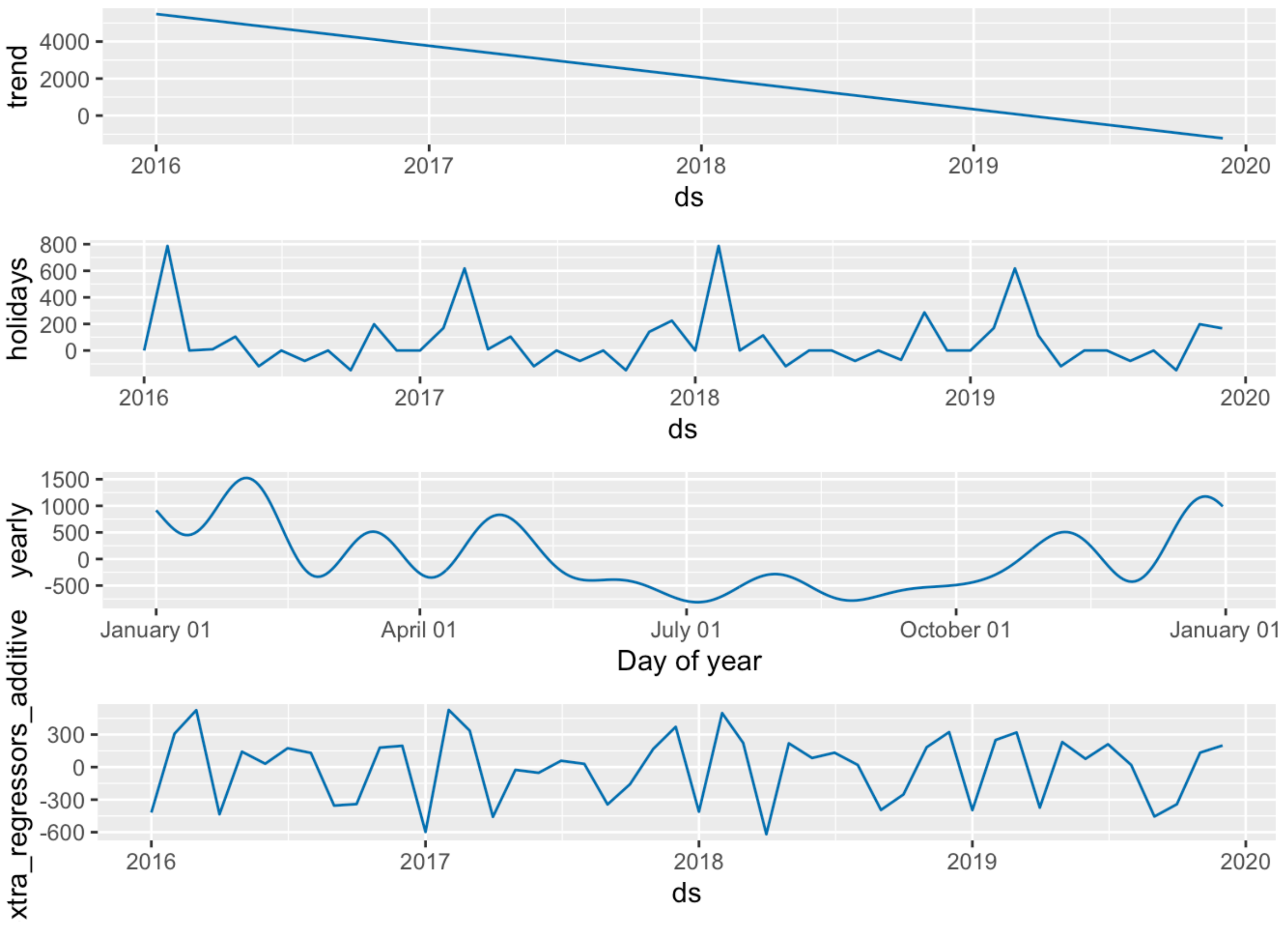

xtra_regressors
3000
0
-3000
2016
2017
2018
2019
2020
ds
trend
210000
180000
150000
120000
2016
2017
2018
2019
2020
ds
holidays
5000
2500
0
-2500
2016
2017
2018
2019
2020
ds
yearly
20000
0
-20000
January 01
April 01
July 01
October 01
January 01
Day of year
xtra_regressors_additive
20000
10000
0
-10000
2016
2017
2018
2019
2020
ds

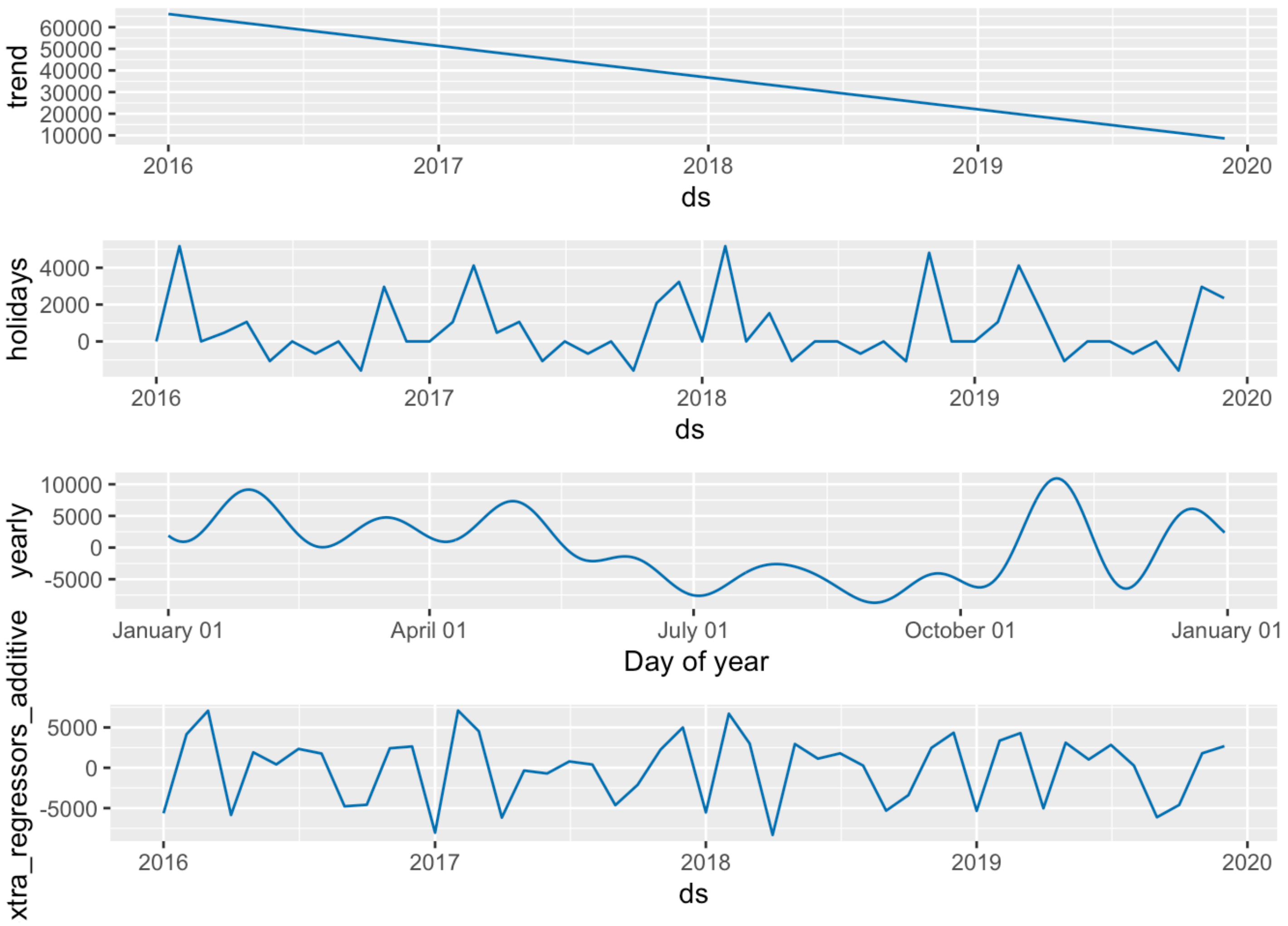

trend
4000
3800
3600
3400
2016
2017
2018
2019
2020
ds
holidays
4000
3000
2000
1000
0
yearly
3000
2000
1000
0
-1000
-2000
January 01
April 01
July 01
October 01
January 01
Day of year
xtra_regressors_additive
200
0
-200


trend
300
295
290
285
280
2016
2017
2018
2019
2020
ds
holidays
100
50
0
yearly
-100
January 01
April 01
July 01
October 01
Day of year
xtra_regressors_additive
-50


trend
7800
7600
7400
7200
7000
2016
2017
2018
2019
2020
ds
holidays
3000
2000
1000
0
2016
2017
2018
2019
2020
ds
yearly
4000
2000
0
-2000
January 01
April 01
July 01
October 01
January 01
Day of year
xtra_regressors_additive
2000
1000
0
-1000
-2000
2016
2017
2018
2019
2020
ds


40000
30000
20000
10000
trend
2016
2017
2018
2019
2020
ds
3000
2000
1000
0
-1000
holidays
2016
2017
2018
2019
2020
ds
4000
0
-4000
yearly
January 01
April 01
July 01
October 01
January 01
Day of year
5000
2500
0
-2500
-5000
xtra_regressors_additive
2016
2017
2018
2019
2020
ds


trend
4000
2000
0
2016
2017
2018
2019
2020
ds
holidays
800
600
400
200
0
ds
yearly
1500
1000
500
0
-500
January 01
April 01
July 01
October 01
January 01
Day of year
xtra_regressors_additive
300
0
-300
-600
ds



Now we have a forecast at the monthly level. The next step will be to calculate how this monthly forecast will translate to quarterly and yearly totals. We will define a function `calc_year_end` to do this for us. This function will use two files, `monthly_pacing` taken from `percentages.txt` and `quarter_totals` taken from `quarter_totals.txt`. Both of these files were sourced from alteryx workflow: `percentages.yxmd`

```
monthly_pacing = fread("percentages.txt", sep= '|')
setnames(monthly_pacing, 'Fiscal_Mo', 'FISCAL_MO')



quarterly_pacing = fread("quarter_totals.txt", sep= '|')
quarterly_pacing[, quarter := ifelse(FISCAL_MO <= 3, 1, ifelse(FISCAL_MO <= 6, 2, ife
lse(FISCAL_MO <= 9, 3, 4)))]
quarterly_pacing[, quarter_sum := sum(cust_month), by=c("quarter", "Customer_Type")]
quarterly_pacing[, table := cust_quarter/quarter_sum]
quarter_table = unique(quarterly_pacing[, c("Customer_Type", "quarter", "table", "qua
rter_sum")])
setnames(quarter_table, "Customer_Type", "Segment_Channel")


calc_year_end <- function(cast_dat, year, hist) {
  et = as.data.table(cast_dat)
  et[, FISCAL_MO := month(ds)]
  et[, quarter := ifelse(FISCAL_MO %in% c(1,2,3), 1, ifelse(FISCAL_MO %in% c(4,5,6),
2, ifelse(FISCAL_MO %in% c(7,8,9), 3,4)))]
  et[, day := format(as.Date(ds,format="%Y-%m-%d"), "%d")]
  et[, FISCAL_YR := year(ds)]
  et = et[ day == '01']
  setnames(et, 'yhat', 'Used_Forecast')



  #wrap up
  df2 = hist[, c('Segment_Channel', 'FISCAL_YR', 'FISCAL_MO', 'y')]
  et2 = merge(et, df2, by=c('Segment_Channel', 'FISCAL_MO', 'FISCAL_YR'), all.x=TRUE)
  et2[, Used_Forecasta := ifelse(is.na(y), Used_Forecast, y)]


  year_agg = merge(et2, monthly_pacing, by = c("Segment_Channel", "FISCAL_MO"))
  year_agg = merge(year_agg, quarter_table, by = c("quarter", "Segment_Channel"))
  year_agg[, Year_Number2 := Used_Forecast * percent]
  year_agg[, Year_Number2a := Used_Forecasta * percent]
  year_agg[, qnum := table*Used_Forecast]
  year_agg[, qnuma := table * Used_Forecasta]


  return(year_agg)
}
```

Examine results of forecasting run

```
final_fcst = calc_year_end(fcst, 2019, regress3)
final_fcst = final_fcst[FISCAL_YR == 2019]
```

```
#estimates = c(1, 0.544586951, 0.366562835, 0.337894522)
#q_s = final_fcst %>% group_by(quarter, Segment_Channel) %>% summarize(sum(qnuma))
#q_s
#sum(q_s*estimates) #1921
```

Once we are happy with the output of our forecast, the final step is to write it to a csv file. The data from this csv file should be pasted into the `Prophet Forecast Data` tab of august_2020_v2 excel workbook. The data and forecast will update automatically after copying and pasting.

Write out forecast to csv file

```
write.csv(final_fcst, '~/Desktop/prophet_aug.csv')
```