Vanessa Chan, Stats M148 Homework 6

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(prophet)
```

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

```
## Attaching package: 'rlang'
##
## The following object is masked from 'package:data.table':
##
##     :=
##
## The following objects are masked from 'package:purrr':
##
##     %@%, flatten, flatten_chr, flatten_dbl, flatten_int, flatten_lgl,
##     flatten_raw, invoke, splice
```

```
library(rstan)
```

```
## Loading required package: StanHeaders
##
## rstan version 2.32.6 (Stan version 2.32.2)
##
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
##
##
## Attaching package: 'rstan'
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
data <- fread("dat_train1.csv")
head(data)
```

```
##    customer_id account_id ed_id      event_name     event_timestamp
## 1:    15849251  383997507     4 browse_products 2021-11-04 14:11:15
## 2:    15849251  383997507     4 browse_products 2021-11-04 14:11:29
## 3:    15849251  383997507     4 browse_products 2021-11-04 14:12:10
## 4:    15849251  383997507     4 browse_products 2021-11-04 14:12:21
## 5:    15849251  383997507     4 browse_products 2021-11-04 14:12:24
## 6:    15849251  383997507     2  campaign_click 2021-11-29 06:00:00
##    journey_steps_until_end
## 1:                       1
## 2:                       2
## 3:                       3
## 4:                       4
## 5:                       5
## 6:                       6
```

Cleaning from previous homeworks

```
data <- data.frame(data)
colnames(data) <- c("customer_id", "account_id", "ed_id", "event_name", "event_timestamp", "journey_ste
```

```
# library(tidyverse)
dist_data <- distinct(data[+-6])

rand <- sample(unique(dist_data$customer_id)+ 10000)
random_sample <- dist_data %>%
  filter(customer_id %in% rand)
```

Task 1

Time series first action

```
not_user <- c(15, 16, 17, 12, 13, 14, 37, 28)
end_data <- max(random_sample$event_timestamp)
rand1 <- random_sample %>%
  group_by(customer_id, account_id) %>%
  filter(max(event_timestamp) <= end_data -days(60)) %>%
  filter(!(ed_id %in% not_user))

daily <- rand1 %>%
  group_by(customer_id, account_id) %>%
  mutate(start = min(event_timestamp)) %>%
  distinct(customer_id, account_id, start) %>%
  group_by(start) %>%
  summarise(count = n_distinct(customer_id, account_id))

weekly <- rand1 %>%
  group_by(customer_id, account_id) %>%
  mutate(start = min(event_timestamp)) %>%
  mutate(year = year(start), week = week(start)) %>%
  distinct(customer_id, account_id, year, week) %>%
  group_by(year, week) %>%
  summarise(count = n_distinct(customer_id, account_id))
```

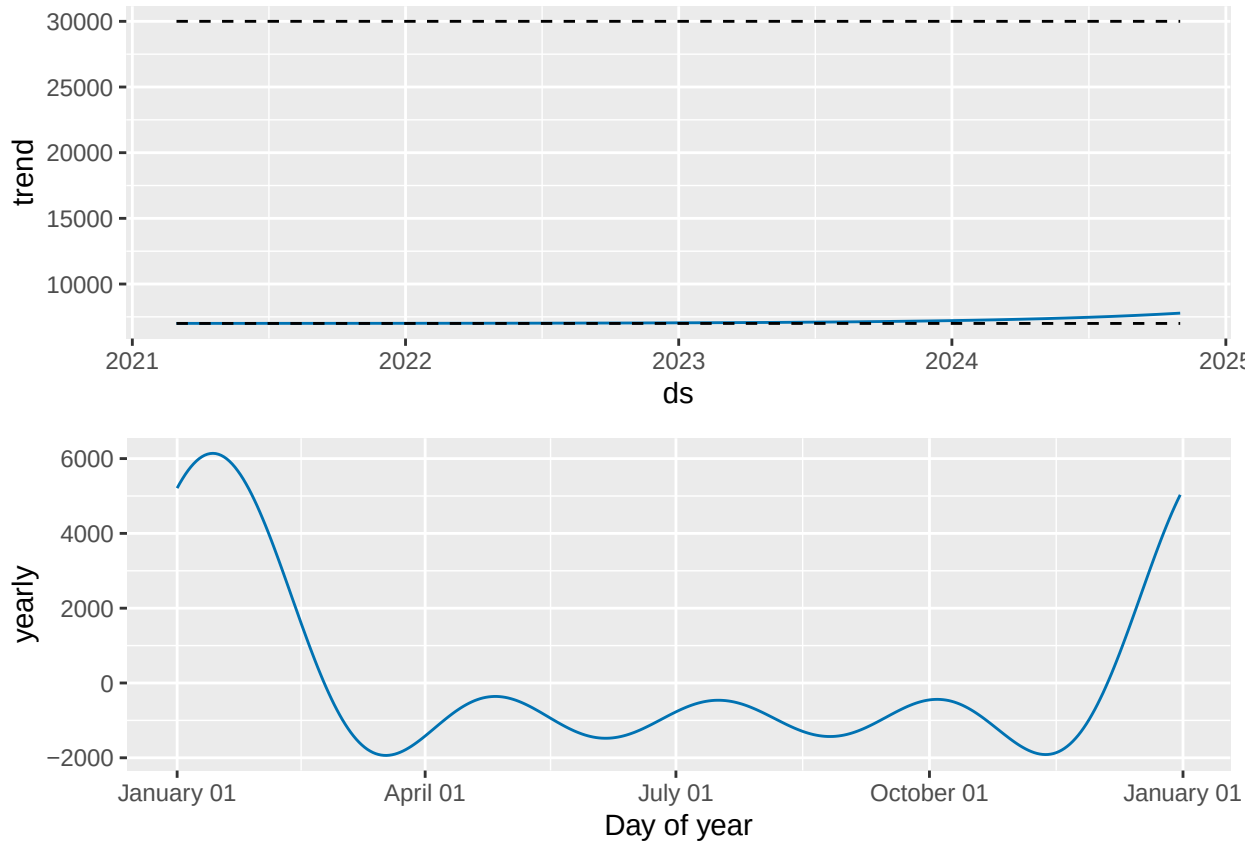
```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
monthly1 <- rand1 %>%
  group_by(customer_id, account_id) %>%
  mutate(start = min(event_timestamp)) %>%
  mutate(year = year(start), month = month(start)) %>%
  distinct(customer_id, account_id, year, month) %>%
  group_by(year, month) %>%
  summarise(count = n_distinct(customer_id, account_id))
```

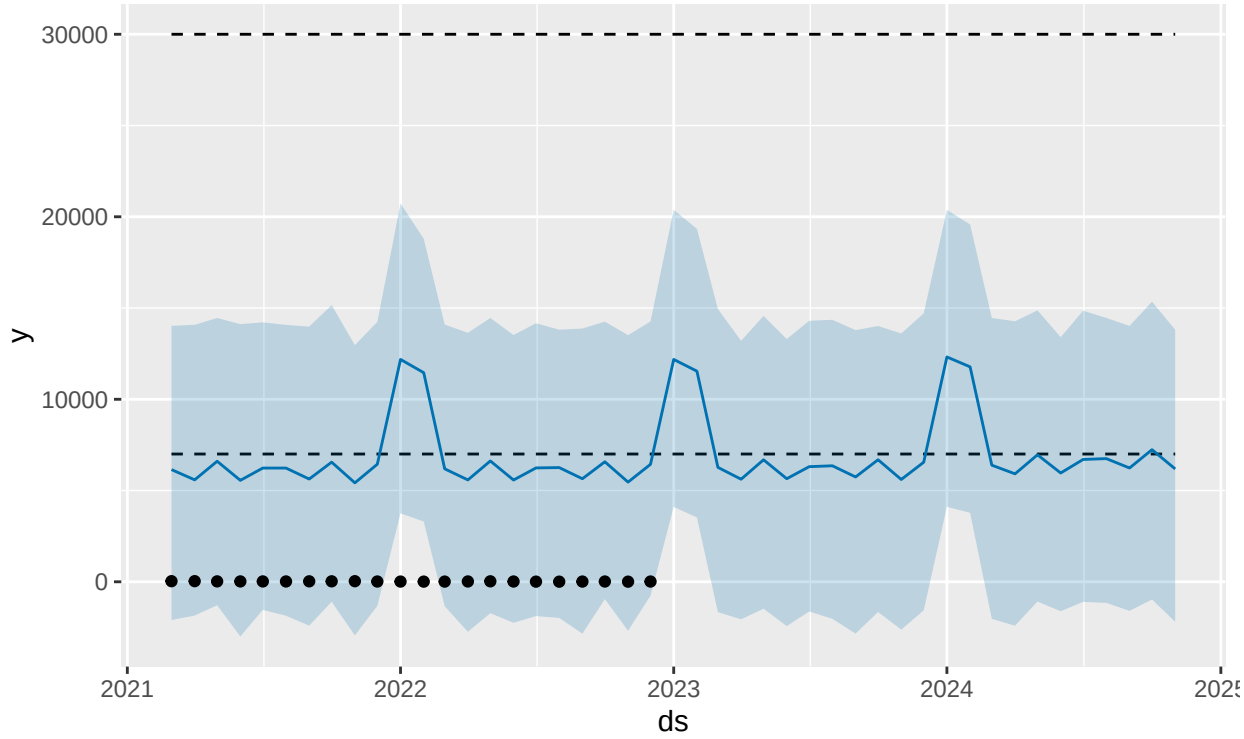
```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
# impute missing december 2022 with december 2021 count
new_row <- data.frame(year = 2022, month = 12, count = 14)
monthly1 <- bind_rows(monthly1, new_row)
```

```
month_dates <- character(0)
for (i in 1:nrow(monthly1)) {
  month_dates <- c(month_dates, paste(monthly1$year[i], monthly1$month[i], "01", sep = "-"))
}
month_dates <- as.Date(month_dates)
ts <- data.frame(ds = month_dates, cap = 30000, floor = 7000, y = monthly1$count)
ts[1, 4] <- ts[4, 4]
ts[2, 4] <- ts[4, 4]
ts[3, 4] <- ts[4, 4]
ts <- ts[c(-1,-2, -3),]
ts_model <- prophet(ts, growth = "logistic", yearly.seasonality = 4, seasonality.mode = "additive",
                    n.changepoints = 0, weekly.seasonality = FALSE,
                    daily.seasonality = FALSE)

future_dates <- make_future_dataframe(ts_model, periods = 23, freq = "month")
future_dates <- data.frame(future_dates, floor = 7000, cap = 30000)
forecast <- predict(ts_model, future_dates)
prophet_plot_components(ts_model, forecast)
```



```
plot(ts_model, forecast)
```



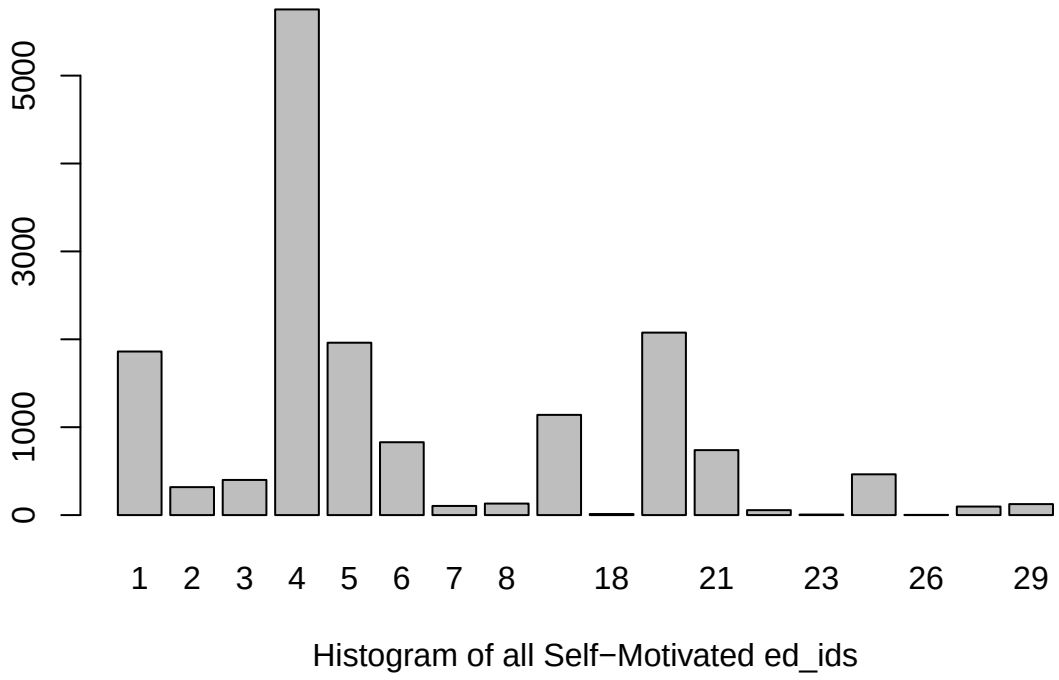
The time series predicts that aside from the the seasonal pattern, there will only be a slight increase in users from year to year.

Task 2

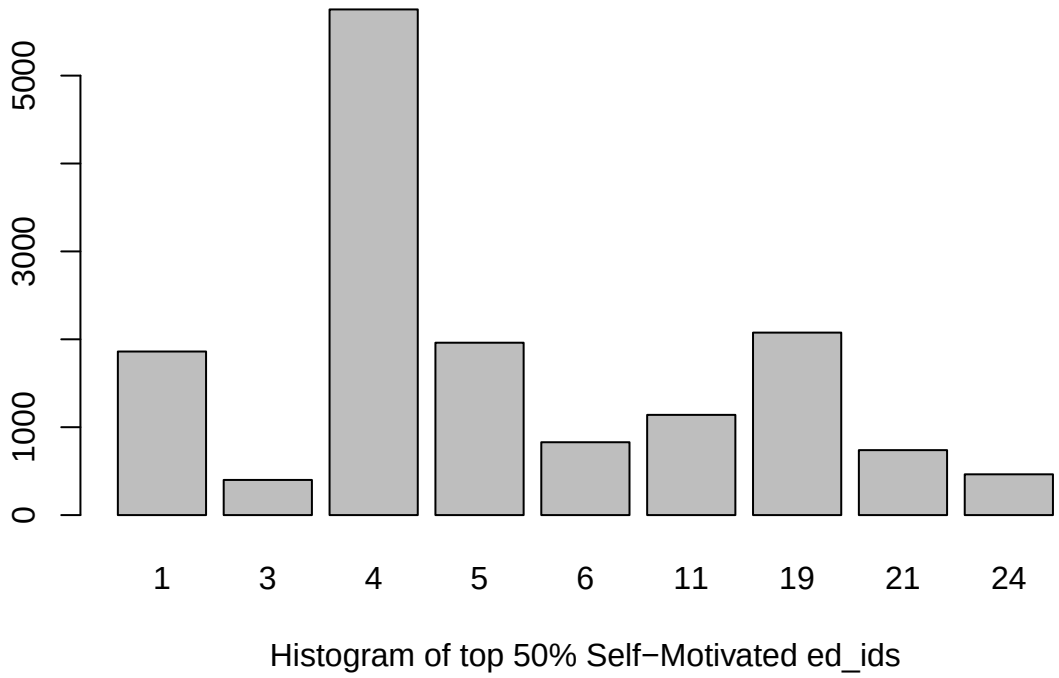
One thing we could have done to improve our model is to have examined the ed_ids more carefully. During class, Prof. Maierhoefer (you) explained that as a class, we should have looked into which ed_ids were user-motivated and which ed-ids were site-automated. Below is a spreadsheet of the user-motivated ed_ids that I have manually looked into. While we took this into account for our time-series, we did not take it into account with our Randomforest models.

Aside from that, Jason and I were also discussing that not all ed_ids have a significant number of occurences. It would probably make our data more streamlined to filter out some of these ed_ids. As a result, I have tabulated the top 50% most common user-motivated ed_ids. In the future, we should use just these ed_ids.

```
events <- random_sample %>%
  select(ed_id) %>%
  filter(!(ed_id %in% not_user))

barplot(table(events), xlab = "Histogram of all Self-Motivated ed_ids")
```




```
# top half most common user-motivated ed_ids
barplot(table(events)[table(events) > 320], xlab = "Histogram of top 50% Self-Motivated ed_ids")
```



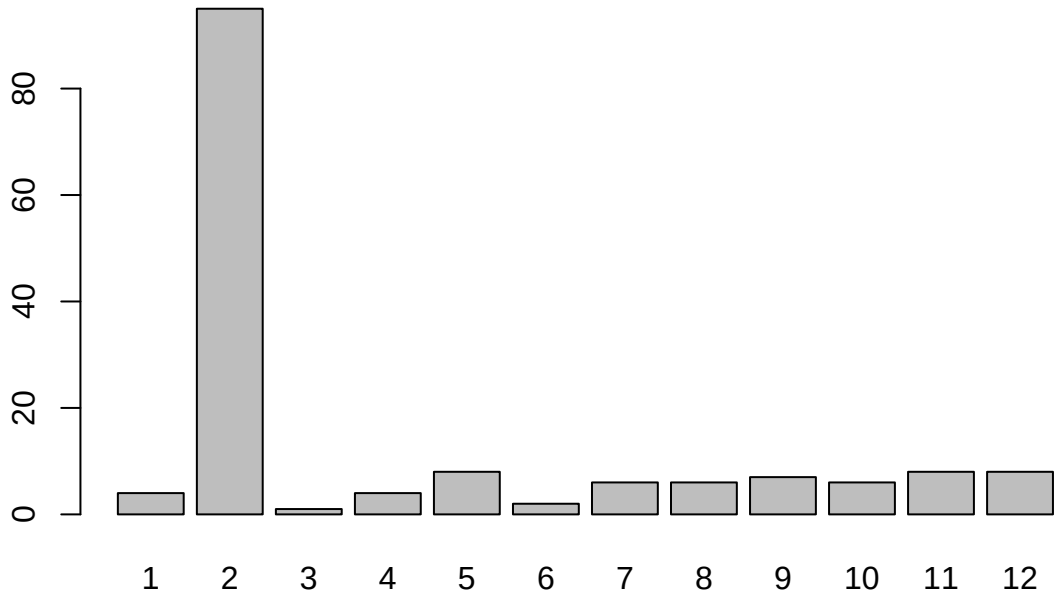
Task 4 Analysis of returns: With Jason looking into promotions, I wanted to look at when people were returning to Fingerhut after 60 days of inactivity. I first used the lag function in dplyr to figure out the times between each step (which can also be used for the LSTM model training). Then, I filtered all "gaps" that are 60 days or longer, meaning that a customer "returned" to Fingerhut.

```
returns <- random_sample %>%
  group_by(customer_id,account_id) %>%
  select(customer_id, account_id, event_timestamp) %>%
  mutate(gaps = event_timestamp - lag(event_timestamp)) %>%
  filter(gaps > 60*60*25*60)


monthly2_ts <- returns %>%
  group_by(customer_id, account_id) %>%
  mutate(year = year(event_timestamp), month = month(event_timestamp)) %>%
  filter(year == 2022) %>%
  group_by(year, month) %>%
  summarise(count = n_distinct(customer_id, account_id))
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
monthly2 <- returns %>%
  group_by(customer_id, account_id) %>%
  mutate(year = year(event_timestamp), month = month(event_timestamp)) %>%
  filter(year == 2022) %>%
```

```
  group_by(month) %>%
  summarise(count = n_distinct(customer_id, account_id))


barplot(monthly2[[2]], names.arg = monthly2[[1]])
```



My barplot indicates that a very significant amount of people return in February, which as of Jason's work on promotions is the month where most of the promotions are issued. We could move towards showing a direct correlation between returning users and promotions, rather than this current interesting but implicit correlation.

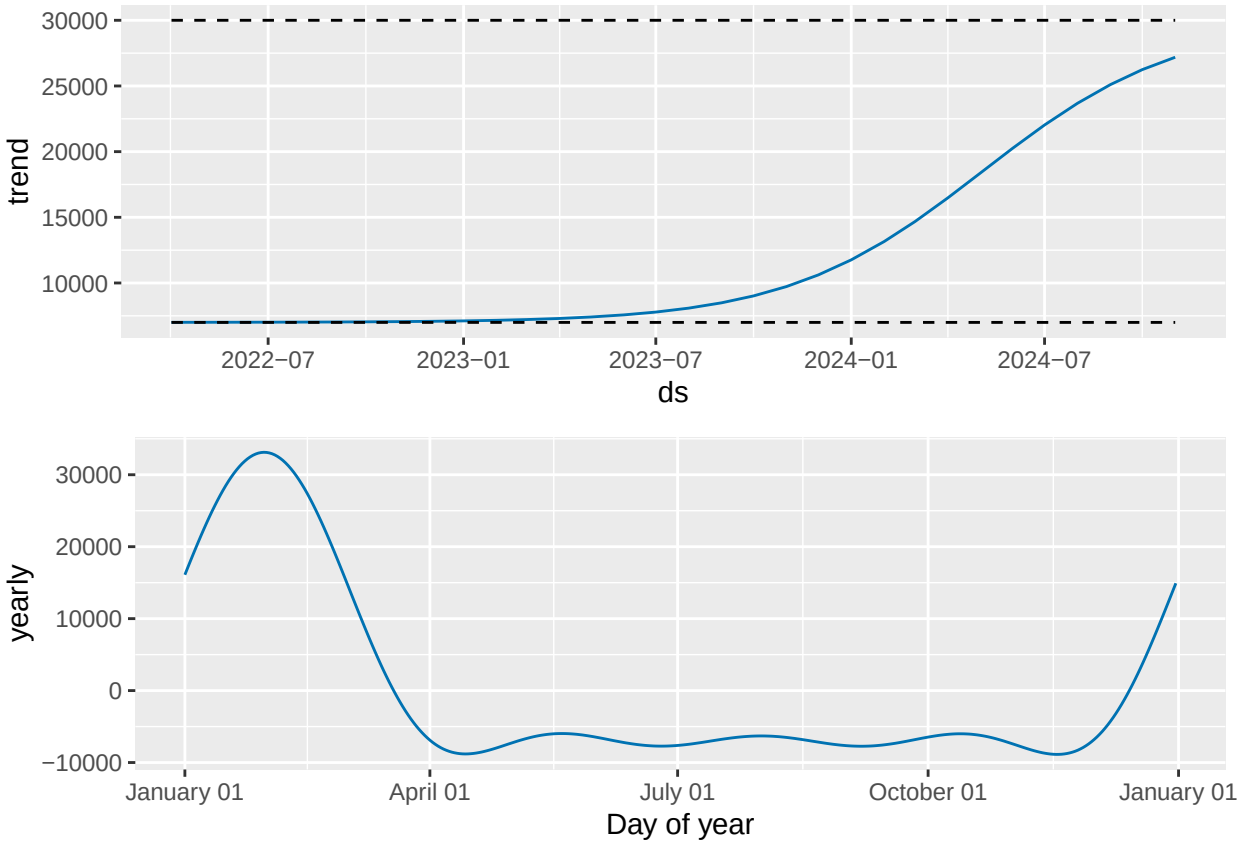
Emerging Recommendation

If January-February is already their busiest month (holiday seasons), perhaps it would make sense to shift promotions to a slower period to try to pull back more users. To further flesh this out, maybe we can compare user returns between those who did and did not receive promotions.

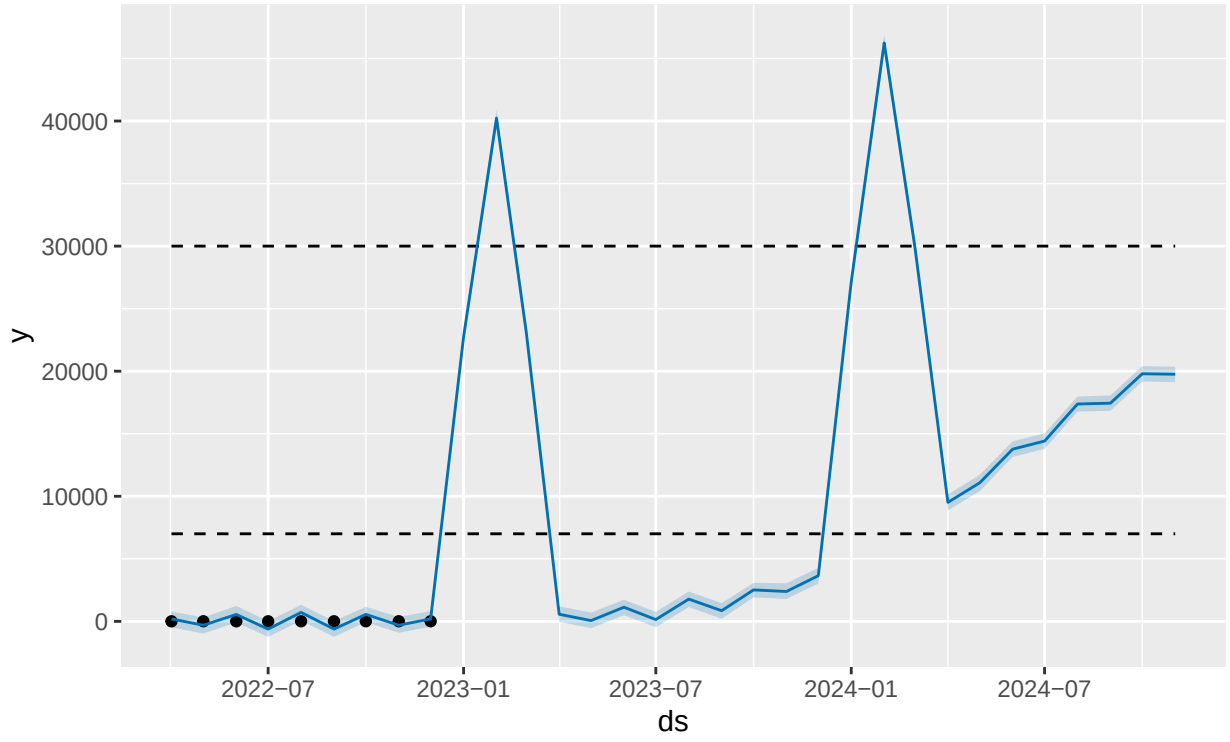
An Ultimately Slightly Irrelevant Time Series

As Prof Almohalwas once said, "do not kill a mosquito with a nuclear bomb". I couldn't train this time series on any year other than 2022 because no other year (2020, 2021, 2023) had a full year's data for returns. Because of this, I believe the predictions are too optimistic and need to be refined.

```r
month_dates <- character(0)
for (i in 1:nrow(monthly2_ts)) {
  month_dates <- c(month_dates, paste(monthly2_ts$year[i], monthly2_ts$month[i], "01", sep = "-"))
}
month_dates <- as.Date(month_dates)
ts <- data.frame(ds = month_dates, cap = 30000, floor = 7000, y = monthly2_ts$count)
ts[1, 4] <- ts[4, 4]
ts[2, 4] <- ts[4, 4]
ts[3, 4] <- ts[4, 4]
ts <- ts[c(-1,-2, -3),]
ts_model <- prophet(ts, growth = "logistic", yearly.seasonality = 4, seasonality.mode = "additive",
                    n.changepoints = 0, weekly.seasonality = FALSE,
                    daily.seasonality = FALSE)

future_dates <- make_future_dataframe(ts_model, periods = 23, freq = "month")
future_dates <- data.frame(future_dates, floor = 7000, cap = 30000)
forecast <- predict(ts_model, future_dates)
prophet_plot_components(ts_model, forecast)
```




```r
plot(ts_model, forecast)
```


40000
30000
20000
10000
0
y
2022−07
2023−01
2023−07
2024−01
2024−07
ds