lab\_deliverable\_2

Yihang Duanmu, Minjun Kim, Annelise Schreiber

2025-10-01

library(tidyverse); library(broom); library(kableExtra); library(here); library(janitor); library(lubridate); library(readr); library(dplyr); library(emmeans); library(ggplot2)

First, install the necessary packages.

# UCI Facebook file is semicolon-separated  
fb\_raw <- readr::read\_delim("dataset\_Facebook.csv", delim = ";", show\_col\_types = FALSE)  
  
# Standardize names and coerce a few key fields  
fb <- fb\_raw |>  
 janitor::clean\_names() |>  
 mutate(  
 post\_month = suppressWarnings(as.integer(post\_month)),  
 post\_weekday = suppressWarnings(as.integer(post\_weekday)),  
 post\_hour = suppressWarnings(as.integer(post\_hour)),  
 paid = if\_else(is.na(paid), 0L, as.integer(paid)),  
 lifetime\_post\_consumers = as.numeric(lifetime\_post\_consumers)  
 )

Load the csv file and standardize the column names.

fb2 <- fb |>  
 mutate(  
 post\_weekday\_num = post\_weekday,  
 post\_hour\_num = post\_hour,  
 post\_month\_num = post\_month  
 )  
  
# Weekday factor (1–7 expected)  
wk\_labels <- c("Mon","Tue","Wed","Thu","Fri","Sat","Sun")  
if (all(!is.na(fb2$post\_weekday\_num)) && all(fb2$post\_weekday\_num %in% 1:7)) {  
 wday\_fac <- factor(fb2$post\_weekday\_num, levels = 1:7, labels = wk\_labels, ordered = TRUE)  
} else {  
 # fallback if strings slipped through  
 wday\_fac <- factor(as.character(fb$post\_weekday), ordered = TRUE)  
}  
  
# Hour factor: detect actual range (handles 0–23 or 1–23)  
hour\_levels <- sort(unique(fb2$post\_hour\_num[!is.na(fb2$post\_hour\_num)]))  
hour\_fac <- factor(fb2$post\_hour\_num, levels = hour\_levels, ordered = TRUE)  
  
# Month factor (1–12 -> Jan..Dec)  
if (all(!is.na(fb2$post\_month\_num)) && all(fb2$post\_month\_num %in% 1:12)) {  
 month\_fac <- factor(fb2$post\_month\_num, levels = 1:12, labels = month.abb, ordered = TRUE)  
} else {  
 month\_fac <- factor(fb$post\_month, ordered = TRUE)  
}  
  
# Paid flag as tidy factor  
paid\_fac <- factor(if\_else(is.na(fb$paid) | fb$paid == 0, "Unpaid", "Paid"),  
 levels = c("Unpaid","Paid"))

This block standardizes your time and paid fields for modeling. First, it builds fb2 by safely coercing post\_weekday, post\_hour, and post\_month to integers (silently turning non-numeric text into NA). Then it creates ordered factors: wday\_fac maps weekday codes (or text like “Monday”) to “Mon”–“Sun”; hour\_fac turns hours into an ordered 0–23 factor; and month\_fac maps 1–12 to “Jan”–“Dec” (or falls back to whatever strings exist). Finally, paid\_fac recodes the numeric paid flag (0/1, with NA treated as 0) into a tidy two-level factor “Unpaid”/“Paid”. The result is a clean set of categorical predictors with stable, meaningful levels for plots and regression.

dat <- tibble(  
 consumers = fb$lifetime\_post\_consumers,  
 paid = paid\_fac,  
 wday = wday\_fac,  
 hour= hour\_fac,  
 month = month\_fac,  
 idx = seq\_len(nrow(fb))  
)  
  
summary(dat$consumers)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 9.0 332.5 551.5 798.8 955.5 11328.0

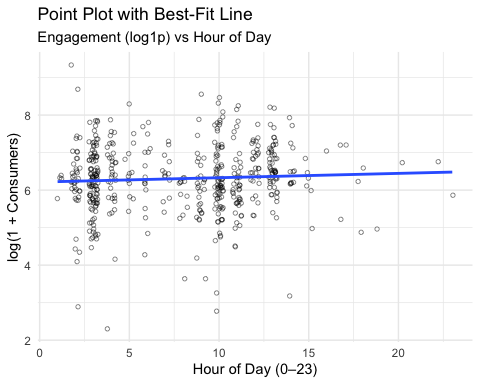
This code builds the analysis tibble dat—it picks the response (consumers) and the cleaned timing predictors (wday, hour, month), adds the promotion flag (paid), and creates a simple time-order index (idx). The summary(dat$consumers) line quickly checks the outcome’s distribution; the output (min 9, Q1 ≈ 333, median ≈ 552, mean ≈ 799, Q3 ≈ 956, max ≈ 11,328) shows a right-skewed variable with large upper outliers, which motivates options like log1p(consumers) or robust SEs in later models.

Question 1: Does the date and time in which a post is made impact the engagement that said post receives?

We used log1p(consumers) because engagement counts are extremely right-skewed with a few huge outliers, which compresses most points near zero on the raw scale and can overly influence the fit. The log transform (with the “+1” to safely handle zeros) stabilizes variance and makes residuals closer to OLS assumptions, so estimates and p-values are more reliable. It also improves readability of the scatter, revealing patterns that raw counts hide. As a bonus, effects on the log scale are easy to interpret approximately as percent changes on the original scale.

plot\_df <- dat |> mutate(hour\_num = as.numeric(as.character(hour)))  
  
ggplot(plot\_df, aes(x = hour\_num, y = log1p(consumers))) +  
 geom\_point(shape = 1, alpha = 0.45, size = 1.2, position = position\_jitter(width = 0.25, height = 0)) +  
 geom\_smooth(method = "lm", se = FALSE) +  
 labs(title = "Point Plot with Best-Fit Line",  
 subtitle = "Engagement (log1p) vs Hour of Day",  
 x = "Hour of Day (0–23)", y = "log(1 + Consumers)") +  
 theme\_minimal()

## `geom\_smooth()` using formula = 'y ~ x'



wk\_map <- setNames(1:7, c("Mon","Tue","Wed","Thu","Fri","Sat","Sun"))  
plot\_df2 <- dat |> mutate(wday\_num = unname(wk\_map[as.character(wday)]))  
  
ggplot(plot\_df2, aes(x = wday\_num, y = log1p(consumers))) +  
 geom\_point(shape = 1, alpha = 0.45, size = 1.2, position = position\_jitter(width = 0.15, height = 0)) +  
 geom\_smooth(method = "lm", se = FALSE) +  
 scale\_x\_continuous(breaks = 1:7, labels = names(wk\_map)) +  
 labs(title = "Point Plot with Best-Fit Line",  
 subtitle = "Engagement (log1p) vs Weekday",  
 x = "Weekday", y = "log(1 + Consumers)") +  
 theme\_minimal()

## `geom\_smooth()` using formula = 'y ~ x'

