Logistic Regression Final

Eli Andrae, Sergio Bacon, Ted McKee

1. Executive Summary

Our project focused on an in-depth analysis of the Google Merch Store Data, covering over 900,000 entries from August 1, 2016, to August 1, 2017. Key variables such as traffic source, dates, browser, total pageviews, total time on site, bounces, transactions, revenue, user country, and purchase count were analyzed to understand consumer behavior and enhance the effectiveness of online shopping. We used a holiday data set to examine the impact of global holidays on sales. Different R libraries were used including dplyr, ggplot2, countrycodes, scales, and lubridate, to clean and visualize the data. Our analyses included generating bar plots, line plots, and correlation plots to answer questions about the highest revenue streams, holiday impacts on sales, and bounce rates.

Based on our findings, we concluded that strategic last-minute promotions and a focused marketing spending on high-value traffic sources are pivotal for driving sales. Additionally, leveraging predictive models to optimize targeting can significantly improve marketing efficiency. These recommendations are aimed at making Googles online shopping more prevalent and contributing to the growth of the global economy. By understanding the nuances of consumer behavior through rigorous data analysis, we can enhance the overall shopping experience and drive sustained growth for the Google Merch Store.

To find more information on the project, including data collection and data pre-preparation you can visit our GitHub

2. Setup

2.1 Load Required Libraries

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## 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(broom)
library(scales)
```

```
##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##     discard
##
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(lubridate)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(ggridges)
```

```
## Warning: package 'ggridges' was built under R version 4.4.3
```

2.2 Load Data

```
google_store <- read_csv("../data/google_store.csv")
```

```
## Rows: 903553 Columns: 17
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): fullVisitorId, traffic_source, traffic_medium, campaign_name, devic...
## dbl (9): visitId, table_date, date, total_pageviews, total_time_on_site, bou...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
daily_stats <- read_csv("../data/daily_stats.csv")
```

```
## Rows: 36283 Columns: 5
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## chr  (1): user_country
## dbl  (2): total_sales, days_till_holiday
## lgl  (1): is_holiday
## date (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

First we will clear the NA values to make the data usable in a Regression

3. Data Cleaning & Feature Engineering

- Cleaning our NA Values and Creating The Purchase Variable

```
google_store <- google_store %>%
  mutate(
    purchase = ifelse(is.na(revenue) | revenue == 0, 0, 1),
    bounces = replace_na(bounces, 0),
    revenue = replace_na(revenue, 0),
    total_time_on_site = replace_na(total_time_on_site, 0),
    transactions = replace_na(transactions, 0),
    session_quality = replace_na(session_quality, 50),
    user_country = replace_na(user_country, "Not Observed")
  )
which(colSums(is.na(google_store)) > 0)
```

```
## named integer(0)
```

3.1 Feature Engineering Using Date Functions

Now we will add some extra features to the data. Due to compute times some of these have been excluded from the file and presaved as a CSV file. Not every country and holiday is included, however, I tried my best to include as many as possible using date.nager API. View `DataPreparation.Rmd` for more info on how the holidays were accessed and `days_till_holiday` was calculated.

```
# Modification Using Lubridate
google_store <- google_store %>%
  mutate(
    date = ymd(google_store$date),
    weekday = wday(date, label = TRUE, abbr =  TRUE),
    is_weekend = wday(date) %in% c(1, 7),
    month = month(date, label = TRUE, abbr = TRUE),
    month = factor(month,
                   levels = c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"),
                   ordered = FALSE))
```

3.2 Merging Holiday Data and Visior Data

```
# Joining with a dataset of holidays and days until the next holiday
google_store <- left_join(
  google_store,
  daily_stats[,c("date", "user_country", "is_holiday", "days_till_holiday")],
  by=c("date", "user_country"))

# Finding the number of previous visits and purchases each visitor has made.
google_store <- google_store %>%
  group_by(fullVisitorId) %>%
  summarize(
    previous_visit_count = n(), previous_purchase_count = sum(purchase)) %>%
  right_join(google_store, 'fullVisitorId')
```

4. Dataset Summary Statistics

So far the dataset contains 275 different traffic sources and 54 different browsers biggest browsers including Chrome, Firefox, Safari, Safari(in app), Edge, YaBrowser, IOS, Android

4.1 Total Browsers

```
num_browsers <- google_store %>%
  summarize(different_browsers = n_distinct(browser))
print(num_browsers)
```

```
## # A tibble: 1 x 1
##   different_browsers
##                <int>
## 1                 54
```

4.2 Highest Revenue per Browser

```
browser_revenue <- google_store %>%
  group_by(browser) %>%
  summarize(total_revenue = sum(revenue, na.rm = TRUE)) %>%
  arrange(desc(total_revenue))
print(browser_revenue %>% head(10))
```

```
## # A tibble: 10 x 2
##    browser           total_revenue
##    <chr>                     <dbl>
##  1 Chrome                1562955.
##  2 Firefox                142659.
##  3 Safari                  57410.
##  4 Internet Explorer        8902.
```

```
##  5 Edge                     7551.
##  6 Opera                     278.
##  7 Safari (in-app)           207.
##  8 Android Webview           151.
##  9 Amazon Silk                37.0
## 10 (not set)                   0
```

```
top_browser <- browser_revenue %>%
  top_n(1, total_revenue)
print(top_browser)
```

```
## # A tibble: 1 x 2
##   browser total_revenue
##   <chr>           <dbl>
## 1 Chrome       1562955.
```

4.3 Highest Revenues per Browser & Traffic Source

Direct from Chrome and Google search from Chrome were the two highest in both transactions and revenue. As you go lower, there is much less for each transactions and revenue.

```
result <- google_store %>%
  group_by(traffic_source, browser) %>%
  summarize(
    total_transactions = sum(transactions, na.rm = TRUE),
    total_revenue = sum(revenue, na.rm = TRUE)
  ) %>%
  arrange(desc(total_revenue), desc(total_transactions))
```

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

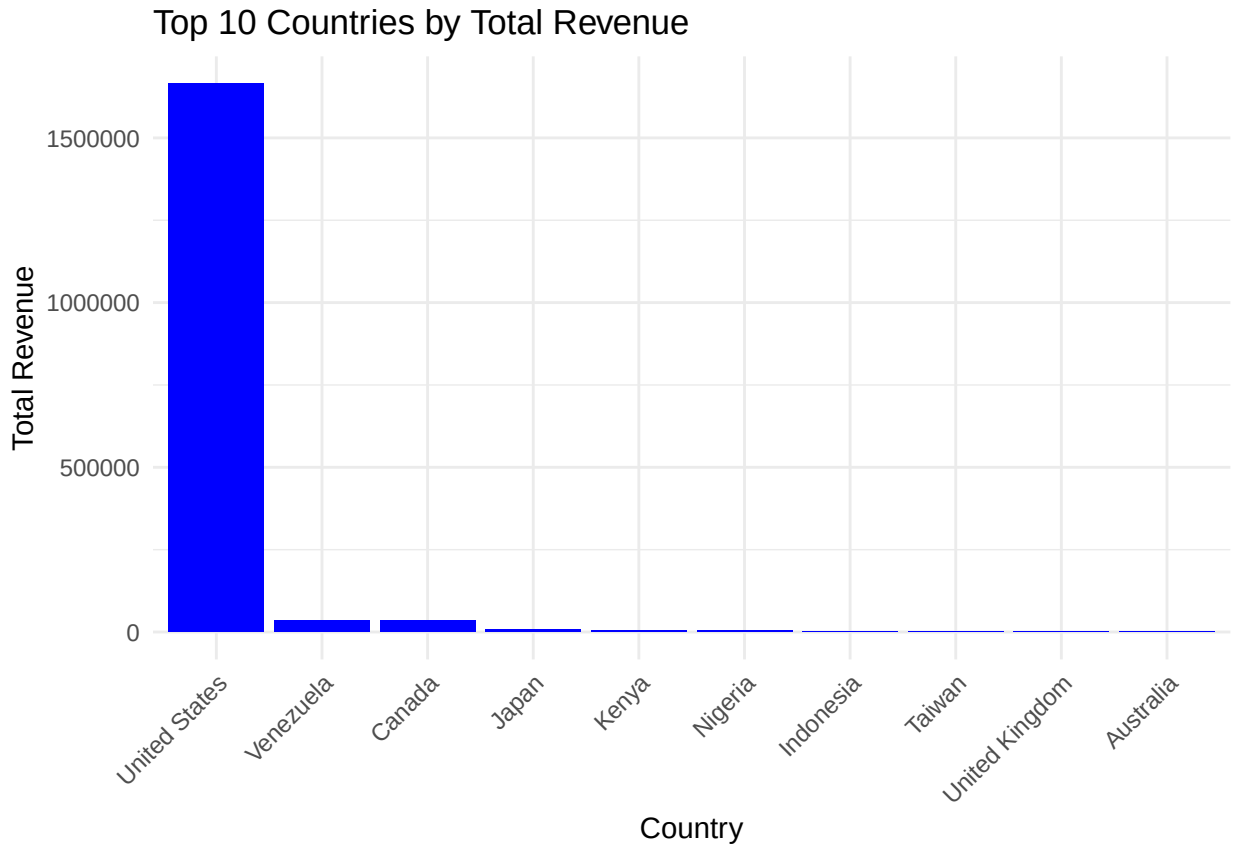
```
print(result %>% head(10))
```

```
## # A tibble: 10 x 4
## # Groups:   traffic_source [5]
##    traffic_source  browser total_transactions total_revenue
##    <chr>           <chr>                <dbl>         <dbl>
##  1 (direct)        Chrome                8523      1262756.
##  2 google          Chrome                1987       236031.
##  3 dfa             Firefox                 22       110588.
##  4 (direct)        Safari                 417        34262.
##  5 (direct)        Firefox                101        25317.
##  6 mail.google.com Chrome                  70        24831.
##  7 google          Safari                 334        21040.
##  8 dfa             Chrome                 106        17910.
##  9 (direct)        Edge                    35         6305.
## 10 dealspotr.com   Chrome                  38         5745.
```

4.4 Highest Revenues per Country

Sales in this dataset are overwhelmingly from the United States

```
country_revenue <- google_store %>%
  group_by(user_country) %>%
  summarize(total_revenue = sum(revenue, na.rm = TRUE)) %>%
  arrange(desc(total_revenue))

top_10_countries <- country_revenue %>%
  top_n(10, total_revenue)

ggplot(top_10_countries, aes(x = reorder(user_country, -total_revenue), y = total_revenue)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(title = "Top 10 Countries by Total Revenue",
       x = "Country",
       y = "Total Revenue") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
print(top_10_countries)
```

```
## # A tibble: 10 x 2
##    user_country   total_revenue
##    <chr>                  <dbl>
```

```
##  1 United States       1664261.
##  2 Venezuela             36082.
##  3 Canada                34922.
##  4 Japan                  7629.
##  5 Kenya                  5286.
##  6 Nigeria                3314.
##  7 Indonesia              2678.
##  8 Taiwan                 2016.
##  9 United Kingdom         1962.
## 10 Australia              1811.
```

4.5 Largest Transactions

```
library(dplyr)

top_n_purchases <- google_store %>%
  arrange(desc(revenue)) %>%
  slice_head(n = 10)

print(top_n_purchases[, c("fullVisitorId", "previous_visit_count", "previous_purchase_count", "revenue"]
```

```
## # A tibble: 10 x 4
##    fullVisitorId       previous_visit_count previous_purchase_count revenue
##    <chr>                             <int>                   <dbl>   <dbl>
##  1 1957458976293878100                 278                      16  47082.
##  2 1957458976293878100                 278                      16  32154.
##  3 9417857471295131045                  16                       5  25249.
##  4 1957458976293878100                 278                      16  17860.
##  5 5632276788326171571                   6                       1  16033.
##  6 9417857471295131045                  16                       5  14060.
##  7 3924372865099736100                   1                       1   9228.
##  8 1957458976293878100                 278                      16   8681.
##  9 4471415710206918415                  34                       4   8258.
## 10 7496147812697146114                   3                       1   7004.
```

It appears the same visitor made a large amount of the largest transactions. In the top 10, they appeared 4 times.

5 Rigline Monthly Sales

For this section, I wrote out the code, fed it to ChatGPT, and said go crazy with the design of the graph.

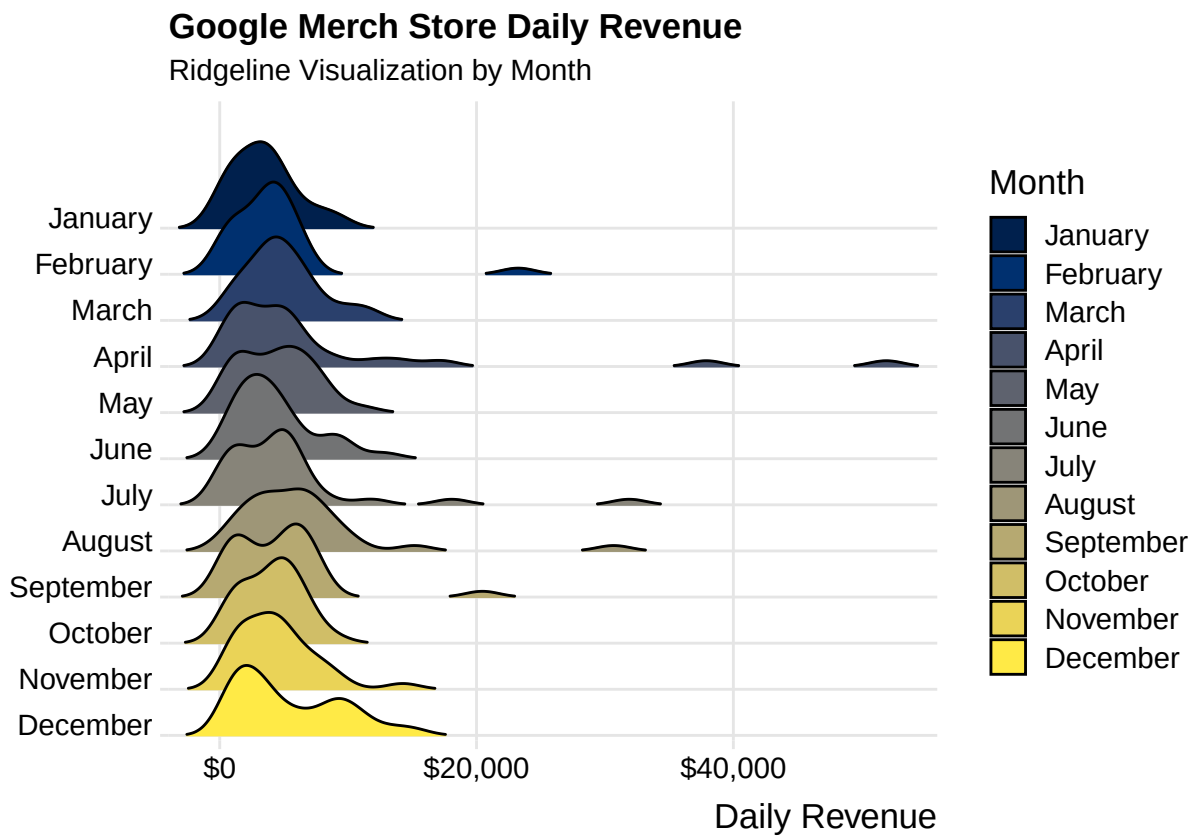
```
revenue_group <- google_store %>%
  group_by(date) %>%
  summarize(total_revenue = sum(revenue)) %>%
  mutate(month = month(date, label = TRUE, abbr = FALSE))
revenue_group
```

```
## # A tibble: 366 x 3
```

```
##    date       total_revenue month
##    <date>             <dbl> <ord>
##  1 2016-08-01         6288. August
##  2 2016-08-02         1467. August
##  3 2016-08-03            0  August
##  4 2016-08-04         1275. August
##  5 2016-08-05         5789. August
##  6 2016-08-06         1938. August
##  7 2016-08-07         2487. August
##  8 2016-08-08         5428. August
##  9 2016-08-09         6011. August
## 10 2016-08-10         4550. August
## # i 356 more rows
```

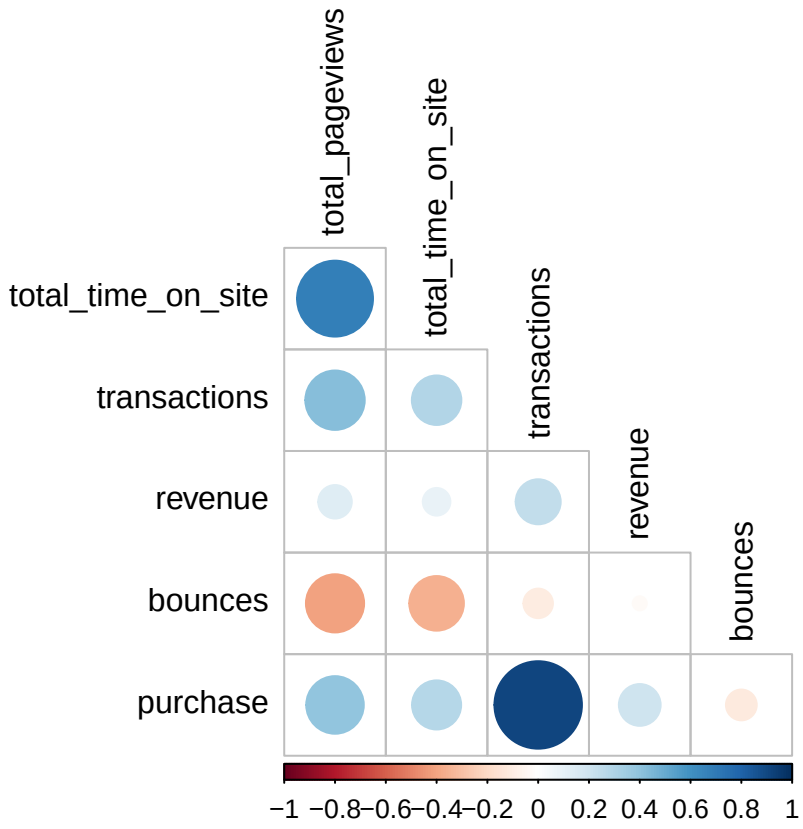
```r
# Create ridgeline plot with a custom theme
ggplot(revenue_group, aes(x = total_revenue, y = month, fill = month)) +
  geom_density_ridges_gradient(
    scale = 2,
    rel_min_height = 0.01,
    gradient_lwd = 1.0
  ) +
  # Format x-axis as currency and remove extra space on the left
  scale_x_continuous(
    labels = dollar_format(prefix = "$", suffix = "", largest_with_cents = 1e3),
    expand = c(0, 0)
  ) +
  # Give y some extra space at the bottom, flip the factor order
  scale_y_discrete(
    expand = expansion(mult = c(0.01, 0.25)),  # Close expansion() properly
    limits = rev(levels(revenue_group$month))  # Then set limits
  ) +
  # Update labels and add a caption if desired
  labs(
    title = "Google Merch Store Daily Revenue",
    subtitle = "Ridgeline Visualization by Month",
    x = "Daily Revenue"
  ) +
  # Keep ridges theme but tweak fonts/grid
  theme_ridges(font_size = 13, grid = TRUE) +
  # Remove y axis title
  theme(axis.title.y = element_blank()) +
  # Apply the cividis color scale
  scale_fill_viridis_d(
    name = "Month",
    option = "cividis"
  )
```

```
## Picking joint bandwidth of 1320
```



Observations: - Each month has a median somewhere around the $0 - $10,000 range. - Some months, like April, June, July, August, November, and December have tails indicating multiple high revenue sales days. - February, April, July, August, and September have very high revenue outlier days with April having the highest. - Identifies "consistent" months, e.g. January, February, September, and October. vs. "volitile" months. e.g. April, July, August, September, and December. Appears to be a "Christmas Daily Revenue Boost" in December. # 6. Looking at Linear Correlation

```
corr <- cor(
  google_store[
    c("total_pageviews", "total_time_on_site", "transactions",
      "revenue", "bounces", "purchase")])
corrplot(corr, type='lower', diag=FALSE, tl.col = 'black')
```



This correlation plot shows a lot of what we would expect to occur between these variables. For example, spending more time on the site strongly correlates to an increased number of pages viewed on the site. What is interesting about this correlation plot is the weak positive correlations between revenue, total pageviews, time spent on site, and transactions. Most people would assume that spending more time on the site and purchasing more items means that you are spending more money but that is not the case as shown by the weak correlations. This means that there is a significant group of shoppers who come to the site knowing exactly what they want and purchasing it quickly. It also means that people often buy a few very expensive items as opposed to a large number of cheaper items.

7. Finding Optimal Pre-Holiday Marketing Period

We believe there may be an optimal period to market before holidays. To find out, we're going to calculate the total number of sales grouped by each number of days_till_holiday. This should give us an idea of the most effective time to market pre-holiday

```
holiday_group <- daily_stats %>%
  group_by(days_till_holiday) %>%
  summarize(total_sales_before_holiday = sum(total_sales)) %>%
  arrange(desc(total_sales_before_holiday)) %>%
  na.omit()

arrange(holiday_group, desc(total_sales_before_holiday)) %>% head(5)
```

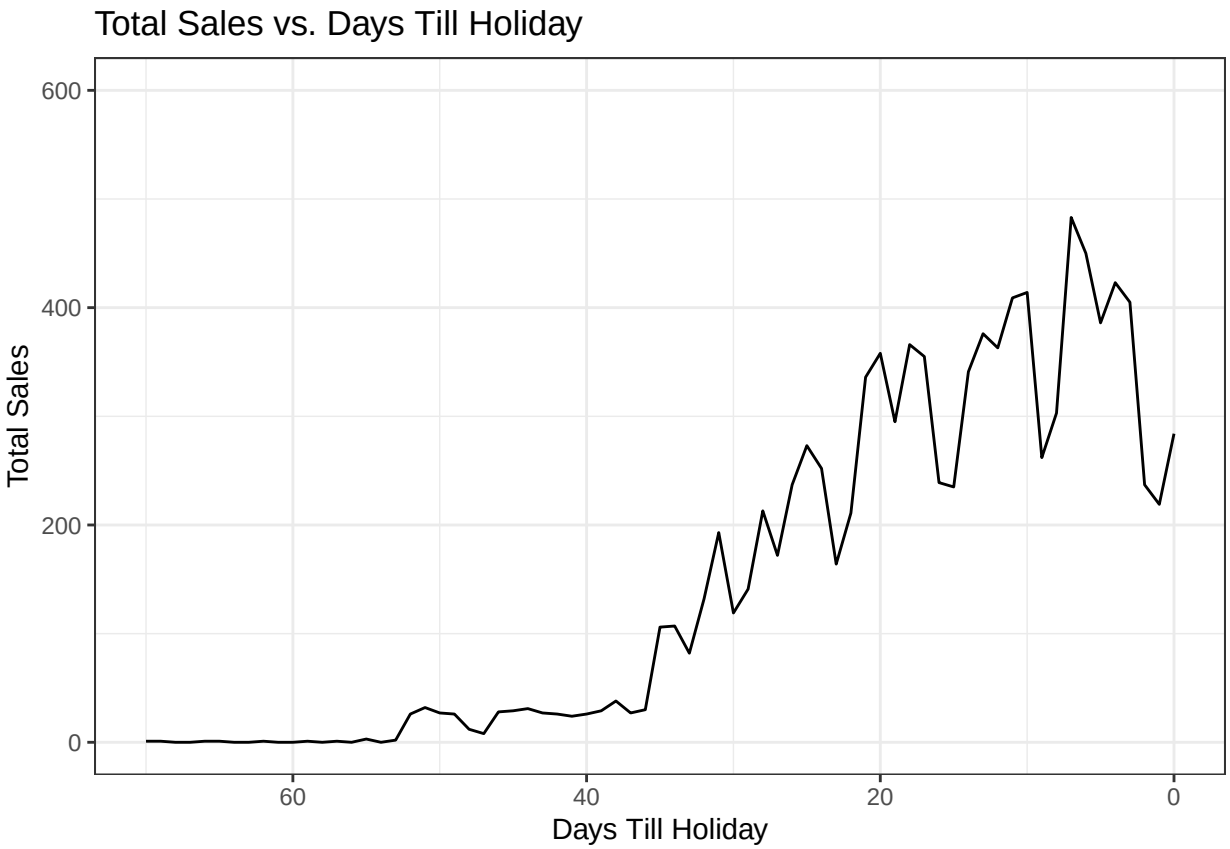
```
## # A tibble: 5 x 2
```

```
##   days_till_holiday total_sales_before_holiday
##               <dbl>                      <dbl>
## 1                 7                        483
## 2                 6                        450
## 3                 4                        423
## 4                10                        414
## 5                11                        409
```

The most common days to place an order before a holiday are 7, 6, and 4 days. It is useful to market much closer to a holiday then I may have expected.

```
holiday_plot <- ggplot(
  holiday_group, aes(x=days_till_holiday, y=total_sales_before_holiday)
) +
  geom_line() +
  scale_x_reverse(limits = c(70, 0)) +
  scale_y_continuous(limits = c(0, 600)) +
    labs(
        title='Total Sales vs. Days Till Holiday',
        x='Days Till Holiday',
        y='Total Sales'
    ) +
    theme_bw()
holiday_plot
```

```
## Warning: Removed 155 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



- Just as I suspected, as it gets closer to a holiday, sales go up around the world wi
- In order to handle this in the logistic regression model, we attempted an inverse transformation on days but that yielded minimal results. Instead we found the more effective method for the model was to find total country sales for each number of days before the holiday. The number of sales for that day can then be used as a weight in the logistic regression model.

Here I will add the days number of sales to each days till holiday

```
google_store <- google_store %>%
  left_join(holiday_group, "days_till_holiday")
```

8. Logestic Regression

- In this section we will build a logistic regression model build for our data set to determine likelihood of a bounce pre visiting, and likelihood of a purchase of a user on the site. As a marketer, this allows us to determine if it is worthwhile advertising to a user, (if they will invest time on the site) based on the way they found the site, and purchase likelihood allows us to start preparing for an order before it's made, and decide if it's worthwhile to send a follow up message if the order is not placed but the user is likely to high (TP), or if the user is not worthwhile advertising to in general high (TN).

8.1 Defining Model

```
logistic_regression <- function(
    df,
    response_var,
    factor_limit = 5,
    numeric_features = NA,
    categorical_features,
    theta = 0.5){

  # Hardcoding this out to make my life simpeler
  df <- df %>%
    filter(traffic_source != "analytics.google.com")

  # Make sure there's no columns with straggling NA values.
  df = na.omit(df)
  # Custom filter list
  df <- df %>% filter()
  # First I need to take an aggressive approach to cleaning the data to avoid
  # mismatches. For this model I will only consider the top n most common
  # features in order to adjust speed and accuracy.
  message("Trimming Features")
  # Generally I would define this out of the function.
  # It's a bit more readable here though.
  top_n_values <- function(column, n) {
    names(head(sort(table(column), decreasing = TRUE), n))}

  # Looping through each row and removing categorical
  # features not in the top n most common. This simplifies the dataset to allow
  # reasonable compute times.
  for (col in categorical_features) {
    top_values <- top_n_values(df[[col]], factor_limit)
    df <- df[df[[col]] %in% top_values, ]}
    #df <- factor(df[[col]], levels = top_values)
  sprintf("The dataset has been reduced to %s rows", nrow(df)) %>% message()

  # Need to make sure numeric features isn't NA.
  # Then standardize them.
  if (!anyNA(numeric_features)){
    for (col in numeric_features){
      df[[col]] <- scale(df[[col]])
    }}

  message("Splitting Dataset")
  # Split the dataset into test-train sets
  train_indices <- createDataPartition(
    df[[response_var]], p = 0.8, list = FALSE
  )

  use_columns <- c(numeric_features, categorical_features, response_var) %>%
    na.omit()

  log_reg_train <- df[train_indices, use_columns]
```

```
  log_reg_test <- df[-train_indices, use_columns]


  message("Building Model")

  # Put together model formula
  model_formula <- as.formula(paste(response_var, "~ ."))
  # Build the model
  model <- glm(
    formula = model_formula,
    family = binomial(link = "logit"),
    data = log_reg_train)

  # Predict values for test & train set
  message("Predicting Values")
  # Helper function for simplicity
  predict_values <- function(df, theta=0.5, model){
    df$prediction <- predict(
      model, newdata = df, type = "response")
    df$prediction <- ifelse(df$prediction > theta, 1, 0)
    return(df)}

  log_reg_train <- predict_values(log_reg_train, theta, model)
  log_reg_test <- predict_values(log_reg_test, theta, model)

  message("Determining results")
  results <- function(df, data_type){
    accuracy <- sum(df[[response_var]] == df$prediction, na.rm=TRUE) / nrow(df)
    confusion_matrix_count <- table(df[[response_var]], df$prediction)
    confusion_matrix <- prop.table(confusion_matrix_count, margin = 1)
    sprintf("Data Type: %s", data_type) %>% print()
    sprintf("The accuracy was %s", percent(accuracy)) %>% print()
    tp <- confusion_matrix[2, 2]
    tn <- confusion_matrix[1, 1]
    sprintf("The TP rate is %s", percent(tp)) %>% print()
    sprintf("The TN rate is %s", percent(tn)) %>% print()
    return(list(accuracy, tp, tn))
  }
  train_results <- results(log_reg_train, "Train")
  test_results <- results(log_reg_test, "Test")
  return(list(model, train_results, test_results))
}
```

8.2 Purchase Prediction

```
categorical_features <- c(
  "traffic_source", "traffic_medium",
  "operating_system", "browser")

numeric_features <- c(
  "total_time_on_site", "total_pageviews", "previous_visit_count",
```

```
  "previous_purchase_count", "total_sales_before_holiday")


purchase_reg <- logistic_regression(
  df = google_store,
  response_var = "purchase",
  factor_limit = 5,
  numeric_features = numeric_features,
  categorical_features = categorical_features,
  theta = 0.05 # Eyeballed optimal theta
)
```

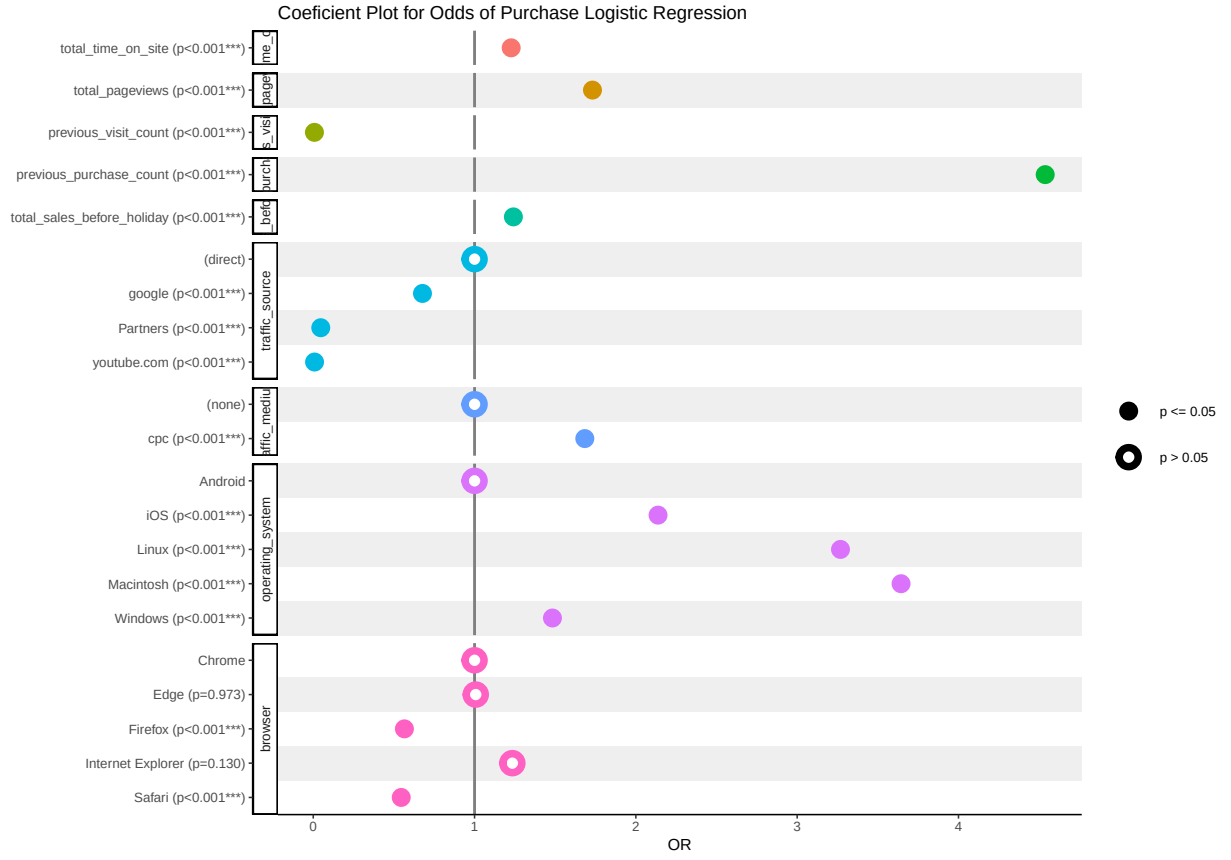
```
## Trimming Features

## The dataset has been reduced to 622956 rows

## Splitting Dataset

## Building Model

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Predicting Values

## Determining results

## [1] "Data Type: Train"
## [1] "The accuracy was 98%"
## [1] "The TP rate is 91%"
## [1] "The TN rate is 98%"
## [1] "Data Type: Test"
## [1] "The accuracy was 98%"
## [1] "The TP rate is 90%"
## [1] "The TN rate is 98%"
```

Purchase Model Results: - Ended up with extremely high accuracy and TN rate. Theta can be adjusted in small increments(+-0.01) to increase TP or NT rate. - I attempted a ridge regression model. I was able to avoid 0 or 1 probability error, however, I lost the modularity of the model. Did not have time to improve upon.

```
purchase_model <- purchase_reg[[1]]

purchase_plot <- ggstats::ggcoef_model(
  purchase_model, exponentiate = TRUE, conf.int = FALSE
) +
  theme_classic(base_size = 6) +
  scale_x_continuous(
    labels = scales::number_format(accuracy = 1)) +
  labs(title = "Coeficient Plot for Odds of Purchase Logistic Regression")
```

```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```

```
purchase_plot
```




```
# ggsave("../images/purchase_model_plot.png",
#        plot = last_plot(),
#        width = 8,
#        height = 6)
```

Observations:

- Time_on_site: Positive purchase odds
- Pageviews: Very positive Previous
- visit count: Lowers
- Previous purchase count: Greatly increases odds (NOTE: Didn't account for current visit purchase.)
- Sales before holiday: Increases offs of purchase Traffic source: Direct and google search work best
- Traffic Medium: Cost-Per-Click (Googles targeted ad system) does better then natural traffic. Proves current marketing is effective.
- OS: Supprisingly Mac and Linux have the highest purchase rates. Browser: Firefox and safari decrease purchase odds.

8.3 Bounce Prediction

High TN Results: - Reduced marketing spend if targeting ideal potential buyers. - Don't spend money on visitors unlikely to make a purchase. - can exclude non-buyers from campaigns or personalized offers - Fraud

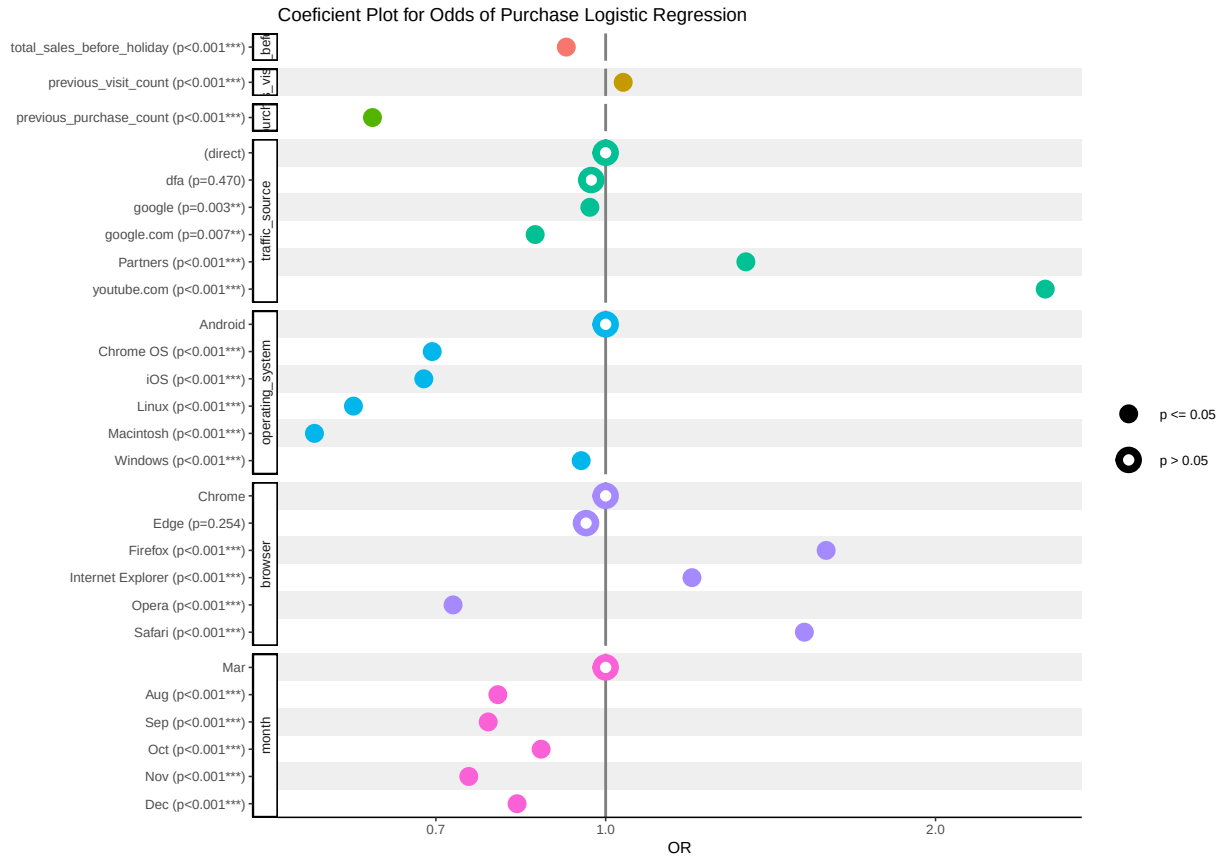
or Bot Detection. - If bot detection is an issue this can eliminate false positives and reduce revenue loss for sales that won't go through.

```
categorical_features <- c(
  "traffic_source", "operating_system", "browser", "month")

numeric_features <- c("total_sales_before_holiday",
                      "previous_visit_count", "previous_purchase_count")


bounce_reg <- logistic_regression(
  df = google_store,
  response_var = "bounces",
  factor_limit = 6,
  categorical_features = categorical_features,
  numeric_features = numeric_features,
  theta = 0.5
)
```

```
## Trimming Features

## The dataset has been reduced to 401749 rows

## Splitting Dataset

## Building Model

## Predicting Values

## Determining results

## [1] "Data Type: Train"
## [1] "The accuracy was 62%"
## [1] "The TP rate is 51%"
## [1] "The TN rate is 73%"
## [1] "Data Type: Test"
## [1] "The accuracy was 62%"
## [1] "The TP rate is 51%"
## [1] "The TN rate is 73%"
```

- This model does much worse, however, the statistically significant coefficients can be useful for further marketing.

```
bounce_model <- bounce_reg[[1]]
ggstats::ggcoef_model(bounce_model,
                      exponentiate = TRUE,
                     conf.int=FALSE) +
  theme_classic(base_size = 6) +
  labs(
    title = "Coeficient Plot for Odds of Purchase Logistic Regression"
  )
```




```
# ggsave("../images/bounce_model_plot.png",
#        plot = last_plot(),
#        width = 8,
#        height = 6)
```

- Bounces are when you visit the site and click off at the home page
- Lower bounce rates are better as customers see more of the product
- Right around 50% bounce rate in dataset
- Holiday sales variable decreases bounce odds
- Lots of visits increases bounce likelihood. Might be worthwhile to find ways to offer sales to high visit rate customers to incentivize clicks further into website
- Traffic_source: natural traffic from search have lowest bounce rates. Partners (paid advertisers) and Youtube are much less likely to interact with site. (Youtube affaliate links/advertisement may not be as worthwhile.)
- More niche browsers are more likely to bounce. Edge may have slightly lower bounce rate (further tesing needed)
- Top 5 sales months are right around phone launches. (Doesn't sell phones on this site) I suggest phone hype drives merch sales. All lower bounce rates