End-to-End Data Solution for Evaluating a Multiple Linear Regression and Negative Binomial Regression Model to Predict E-Commerce First Customer Acquisitions

Joanne Senoren

Master of Science, Data Analytics

Student ID: 011826418

November 21, 2025

Table of Contents

Section A: Research Question

A1. Research Question, Context, and Justi,sfication

Today's marketing teams face the challenge of turning high-volume, high-variety data into meaningful insights that inform data-driven spending decisions and strategic planning. The research question for this paper is: "How can digital media spend, performance metrics, and seasonality be processed through an ELT (extract, load, transform) solution into a model-ready dataset to evaluate the performance of a Multiple Linear Regression and Negative Binomial Regression model in predicting first purchase acquisitions to support optimizing budget allocation?"

The study is set in the apparel e-commerce sector, where organizations run campaigns across Google and Meta advertising placements. By leveraging an ELT process, the study can analyze statistical patterns in media spend, performance metrics, and seasonal trends to inform marketing strategies aimed at acquiring new customers.

The research question is justified because marketing teams often rely on naïve, calculated metrics, such as customer acquisition cost, which fail to account for statistical relationships between media spend, metrics, and seasonality. By benchmarking these relationships, the study provides insights that enable more informed budget allocation and improved marketing effectiveness.

A2. Hypothesis Discussion

The null hypothesis for the study states that there is no significant relationship between channel media spend, calculated metrics, seasonality, and first purchases, and neither models can effectively predict first customer acquisitions. For media planning and marketing, failure to

reject the null hypothesis means that the models cannot provide a dependable predictive analytical approach to media budget optimization.

The alternative hypothesis proposes that at least one model includes statistically significant predictors ($p < 0.05$) and demonstrates sufficient explanatory power, defined as an Adjusted $R^2 \geq 0.60$ for multiple linear regression or a Pseudo-$R^2 \geq 0.10$ for Negative Binomial Regression, with 70% of predictions falling within ±20% of the actual values.

The R^2 metrics were updated from the proposal to represent a reasonably well-fitted model. The original 0.06 benchmarks were too close to zero, indicating that only 6% of the variance is explained by the model. To better reflect a reasonably well-fitted model, the Adjusted R^2 threshold was raised to 0.60, indicating that the model explains 60% of the variance in first-time customer acquisitions. The Pseudo-R^2 threshold of 0.30 was chosen to represent a meaningful improvement over a null model.

Adjusted R^2 penalizes the inclusion of irrelevant predictors, providing a more accurate measure of explained variance. For example, an adjusted R^2 of 0.6 indicates that 60% of the variance in first-time customer acquisitions can be explained by the predictors. Pseudo-R^2, by contrast, is used in models where variance is not directly defined, comparing the fitted model to a null model (O'Brien, 2011).

The second portion of the alternate hypothesis means that the model is considered useful if 70% of predicted values fall within ±20% of actual results. This alternate hypothesis is supported by Fareniuk and Chornius (2023), who studied marketing mix models (MMM) – statistical models that measure ROI against media spend and relevant variables – and have found

that these models can optimize media budgets based on their selected ROI (in-store traffic) across online and offline media placements.

Section B: Data Collection

B1. Relevant Data and Collection Process

This study utilized data from Figshare.com and the Nager API. Figshare provided anonymized e-commerce campaign data from Google and Meta, spanning 2020–2024, which included media spend, impressions, clicks, and daily first-time purchases (Figshare, 2024). The data were filtered to include only campaigns within the United States and the apparel vertical, ensuring relevance and consistency. All organizations meeting the minimum row requirement were retained to provide a sufficiently large sample for statistical analysis.

An external API called Nager produces a list of all historically observed and public holidays across the USA (*Worldwide Public Holidays - Nager.Date*, n.d.). The API JSON payload provides the essential information to derive holiday and weekend flags using the date data from Figshare.

Both datasets were loaded into Jupyter Notebook using Python, filtered for relevant variables, and exported as CSV files for Google, Meta, and internal customer data. The combined datasets were then reviewed for completeness and alignment with real-world campaign report deliveries from media vendors.

The Nager API was validated by testing an endpoint URL (e.g., https://date.nager.at/api/v3/PublicHolidays/2021/US), which returns a JSON response of all U.S. public holidays for the specified year. Figure 1 shows the relevant files with targeted columns to be processed through the ELT solution. The table in Figure 2 shows a Nager API output example.

Figure 1

Dataset Files and Target Columns

Files	Relevant Columns
google_media.csv	google_shopping_spend google_paid_search_spend google_pmax_spend google_display_spend google_video_spend google_shopping_impressions google_paid_search_impressions google_pmax_impressions google_display_impressions google_video_impressions google_shopping_clicks google_paid_search_clicks google_pmax_clicks google_display_clicks google_video_clicks
meta_media.csv	meta_facebook_spend meta_instagram_spend meta_other_spend meta_facebook_impressions meta_instagram_impressions meta_other_impressions meta_facebook_clicks meta_instagram_clicks meta_other_clicks
daily_transactions.csv	first_purchases date_day organisation_id

Figure 2

Nager API Output Object Example

```
[
  {
    "date": "2020-01-01",
    "localName": "New Year's Day",
    "name": "New Year's Day",
    "countryCode": "US",
    "fixed": false,
    "global": true,
    "counties": null,
    "launchYear": null,
    "types": [
      "Public"
    ]
  },
```


B2. Data Gathering Advantages and Disadvantages

The main advantage of using the Figshare dataset is that it provides real-world media campaign details across specific digital advertising placements. Media plans and media buys are created after vendors and advertisers negotiate the advertising placements (Kantar Media, 2013). The Nager API provides a lookup dictionary of all historical public holidays, allowing for the quick calculation of most holiday flags using the date day values in the Figshare data.

A primary disadvantage of the Figshare data was that the data provider did not specify whether non-digital placements ran alongside the same campaigns, which could have influenced the outcomes of the target variable. Additionally, the creative messaging for each placement was not included to maintain brand anonymity. However, creative messaging is crucial for understanding what motivates customers to make online purchases. As a result of these missing details, insights from the data will be used as industry benchmarks only (figshare, 2024).

B3. Data Gathering Challenges

A primary challenge with the Figshare data was that multiple organizations needed to be included to meet the minimum threshold of 7,000 rows. Consequently, organization IDs had to

be incorporated as explanatory features in the regression models to account for inter-organizational variability and to identify which organizations most strongly influence trends and patterns.

Another challenge came from the Nager API, which provides only float and observed holidays, omitting fixed holidays that may fall on weekends. For instance, Christmas Day (December 25) is a fixed holiday that can occur on a Sunday, while the observed holiday may be the following Monday. Both types of holidays impact business operations (Ruehl, 2024) and were therefore included in the analysis. Fixed holiday dates were manually added during the silver-stage data cleaning process to ensure completeness.

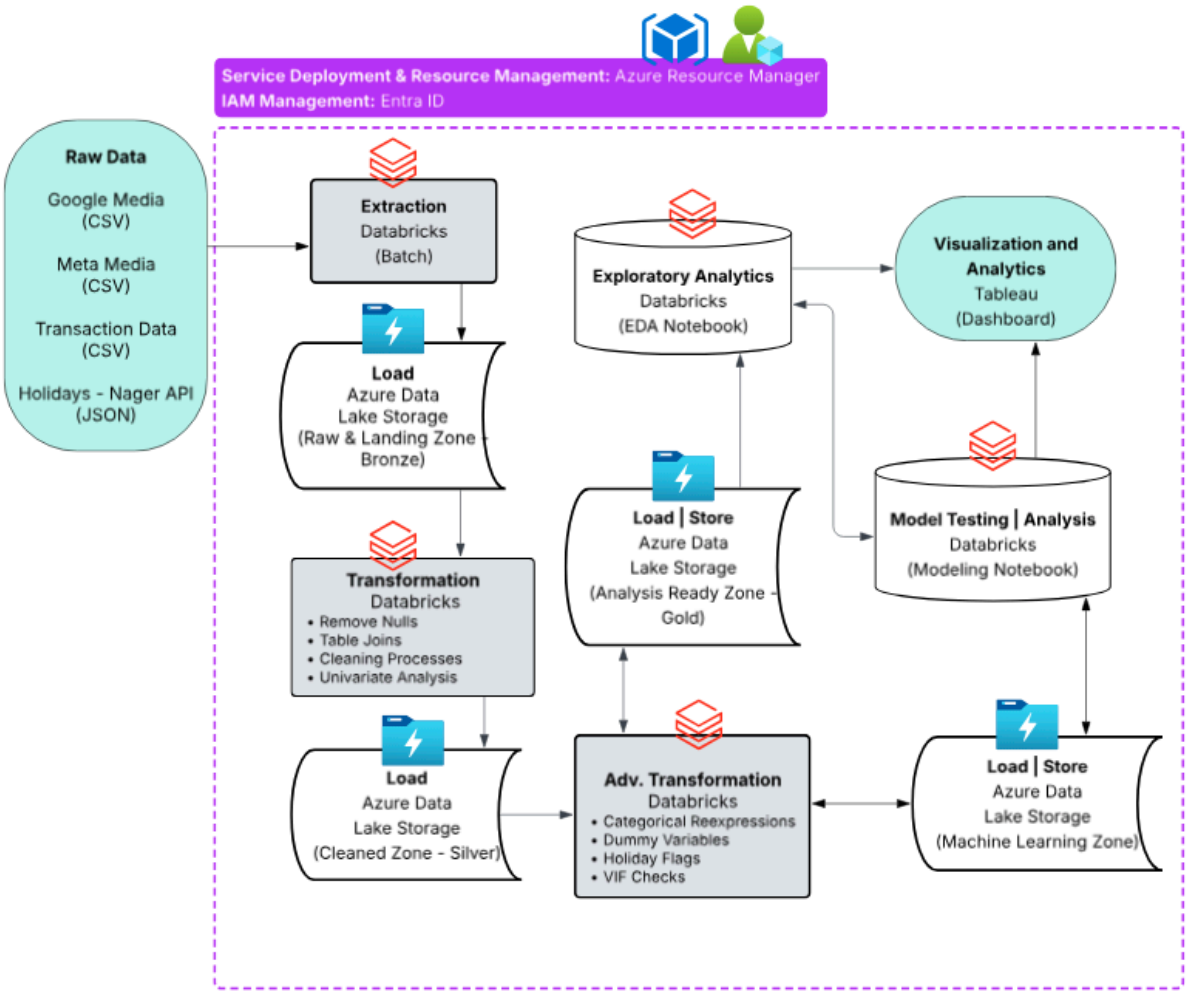
Section C: Data Extraction & Preparation

C1. Data Extraction and Preparation Process

The data extraction and preparation process consists of a big data ELT solution in the Azure environment using Databricks and Azure Data Lake Storage. The ELT process is a better solution than ETL because it easily scales to big data (Santos & Costa, 2020). Modern digital marketing data is inherently high volume, high velocity, and high variety (e.g., advertising clicks, impressions, purchase data), making ELT a suitable choice (*Big Data Marketing: What It Is and Why It Matters*, n.d.). Figure 3 shows a conceptual data process flow diagram and ELT architecture in the Azure Databricks environment.

Figure 3

Process Flow Diagram and ELT Architecture

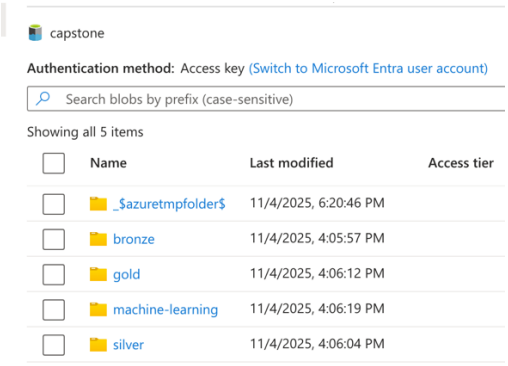


The Medallion architecture is a data design pattern that structures data through three sequential stages in an ELT process: bronze (raw data), silver (cleaned data), and gold (analysis-ready data) (Mssaperla, n.d.). Within the Azure Data Lakehouse, storage folders are organized to correspond to each stage of the data pipeline. Raw CSV files are initially loaded into the bronze folders (Figure 3A), preserving the unprocessed source data. At the same time, a fetch request is made to the Nager API using the minimum and maximum years from the internal dataset. The resulting JSON payload is also stored in the bronze layer (Figures 3B–3C). This approach ensures that both raw campaign data and external holiday information are captured and preserved for cleaning and transformation.

Figure 3

Bronze Files and API Outputs

3A. CSV Files and API Output Folders in Bronze Stage

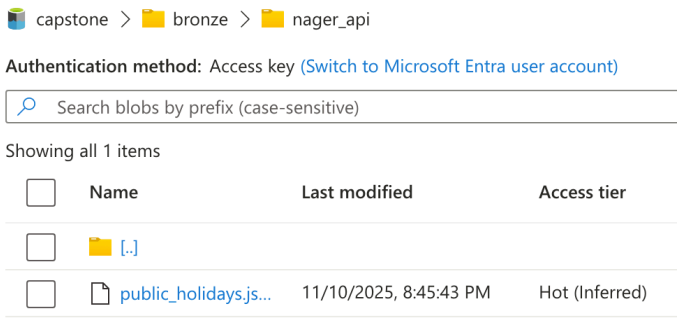


3B. Fetch Request Code

```
# Fetch Holiday Data from Nager API using internal file min and max dates and store raw JSON load into bronze folder
# Get date range from internal data
min_date = internal_df.agg({"date_day": "min"}).collect()[0][0]
max_date = internal_df.agg({"date_day": "max"}).collect()[0][0]

start_year = min_date.year
end_year = max_date.year

# Set container for holidays so we don't get duplicates
fetched_holidays = set()

excluded_holidays = ["Good Friday", "Lincoln's Birthday", "Truman Day"]

for year in range(start_year, end_year + 1):
    try:
        url = f'https://date.nager.at/api/v3/PublicHolidays/{year}/US'
        response = requests.get(url)
        response.raise_for_status()
        holidays = response.json()

        for h in holidays:
            if h["name"] not in excluded_holidays:
                fetched_holidays.add(datetime.datetime.strptime(h['date'], "%Y-%m-%d").date().isoformat())

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data for {year}: {e}")

    # Print completed fetch request and year of holiday
    print(f'Completed holiday fetch request for {year}')

# Save holiday array as JSON to bronze folder
holiday_path = "/dbfs/mnt/capstone/bronze/nager_api/public_holidays.json"

# convert fetch_holidays to array
fetched_holidays = list(fetched_holidays)

with open(holiday_path, 'w') as f:
    json.dump(fetched_holidays, f)

# Sort list
fetched_holidays = sorted(fetched_holidays)

print(fetched_holidays)
Completed holiday fetch request for 2020
Completed holiday fetch request for 2021
Completed holiday fetch request for 2022
Completed holiday fetch request for 2023
Completed holiday fetch request for 2024
['2020-01-01', '2020-01-20', '2020-02-17', '2020-05-25', '2020-07-03', '2020-09-07', '2020-10-12', '2020-11-11', '2020-11-26', '20
5-31', '2021-06-18', '2021-07-05', '2021-09-06', '2021-10-11', '2021-11-11', '2021-11-25', '2021-12-24', '2021-12-31', '2022-01-17
'2022-09-05', '2022-10-10', '2022-11-11', '2022-11-24', '2022-12-26', '2023-01-02', '2023-01-16', '2023-02-20', '2023-05-29', '202
-10', '2023-11-23', '2023-12-25', '2024-01-01', '2024-01-15', '2024-02-19', '2024-05-27', '2024-06-19', '2024-07-04', '2024-09-02'
```

3C. Saved JSON Payload in Bronze Folder

```
# Save holiday array as JSON to bronze folder
holiday_path = "/dbfs/mnt/capstone/bronze/nager_api/public_holidays.json"

# convert fetch_holidays to array
fetched_holidays = list(fetched_holidays)

with open(holiday_path, 'w') as f:
    json.dump(fetched_holidays, f)
```



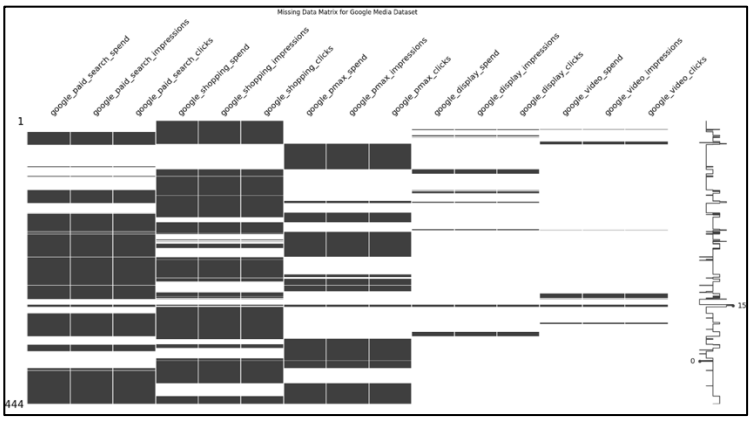
The media datasets had a significant number of null rows across spend, clicks, and impressions per channel. Figure 4A and 4B confirm that these missing values are not missing at random (MNAR), instead of missing completely at random (MCAR) or missing at random (MAR). This means that the missing data is systemic, appearing only on the same days where spend, impressions, and clicks are null. Moving forward, these specific zero days are referred to as no-spend or inactive days. They were imputed from null to zero values to maintain data continuity in model development, curtail model bias, avoid inflating model coefficients, and overfitting (Little & Rubin, 2019).

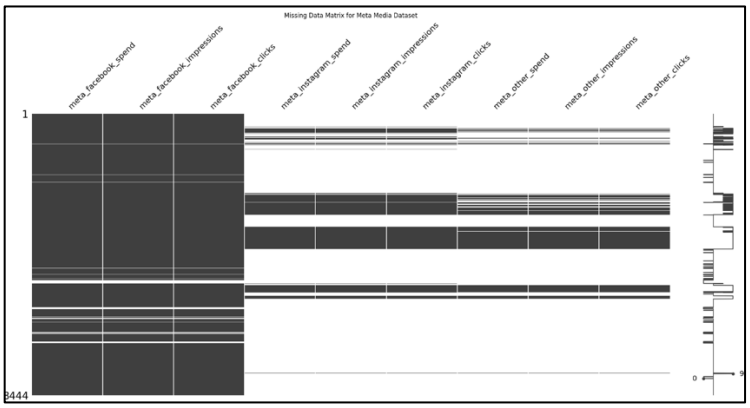
Figure 4

Checking Missingness in Google Media and Meta Media

4A. missingno Matrix Code and Graphs

```
# Sort by date_day
google_pd_df = google_pd_df.sort_values(by=['organization_id', 'date_day'])

google_order = ['google_paid_search_spend', 'google_paid_search_impressions', 'google_paid_search_clicks',
    'google_shopping_spend', 'google_shopping_impressions', 'google_shopping_clicks', 'google_pmax_spend', 'google_pmax_impressions', 'google_pmax_clicks',
    'google_display_spend', 'google_display_impressions', 'google_display_clicks', 'google_video_spend', 'google_video_impressions', 'google_video_clicks']

meta_order = ['meta_facebook_spend', 'meta_facebook_impressions', 'meta_facebook_clicks', 'meta_instagram_spend', 'meta_instagram_impressions',
'meta_instagram_clicks', 'meta_other_spend', 'meta_other_impressions', 'meta_other_clicks']

msno.matrix(google_pd_df[google_order])
plt.title("Missing Data Matrix for Google Media Dataset")

msno.matrix(meta_pd_df[meta_order])
plt.title("Missing Data Matrix for Meta Media Dataset")
```





4B. Calculating Null Row Count (all row counts should be the same per channel)

```python
combined_df = google_media_df.join(
    meta_media_df,
    on=["organization_id", "date_day"],
    how="left"
)

# Define groups of columns to check
groups = [
    ['meta_facebook_spend', 'meta_facebook_impressions', 'meta_facebook_clicks'],
    ['meta_instagram_spend', 'meta_instagram_impressions', 'meta_instagram_clicks'],
    ['meta_other_spend', 'meta_other_impressions', 'meta_other_clicks'],
    ['google_paid_search_spend', 'google_paid_search_impressions', 'google_paid_search_clicks'],
    ['google_shopping_spend', 'google_shopping_impressions', 'google_shopping_clicks'],
    ['google_pmax_spend', 'google_pmax_impressions', 'google_pmax_clicks'],
    ['google_display_spend', 'google_display_impressions', 'google_display_clicks'],
    ['google_video_spend', 'google_video_impressions', 'google_video_clicks']
]

# Function to count zeros per column
def count_nulls(df, cols):
    counts = {}
    for col in cols:
        counts[col] = df.filter(F.col(col).isNull() | F.isnan(F.col(col))).count()
    return counts

# Check each group
for group in groups:
    null_counts = count_nulls(combined_df, group)
    print(f"Group: {group[0].split('_')[0]} / {group[0].split('_')[1]}")
    print(null_counts)

    if len(set(null_counts.values())) == 1:
        print(f'All nulls match: {list(null_counts.values())[0]} rows')
        print(f'Inactive media days present in data.\n')
    else:
        print(f'Mismatch in null counts!')
        print(f'Check null rows\n')
```

```
Group: meta / facebook
{'meta_facebook_spend': 367, 'meta_facebook_impressions': 367, 'meta_facebook_clicks': 367}
All nulls match: 367 rows
Inactive media days present in data.

Group: meta / instagram
{'meta_instagram_spend': 6566, 'meta_instagram_impressions': 6566, 'meta_instagram_clicks': 6566}
All nulls match: 6566 rows
Inactive media days present in data.

Group: meta / other
{'meta_other_spend': 6885, 'meta_other_impressions': 6885, 'meta_other_clicks': 6885}
All nulls match: 6885 rows
Inactive media days present in data.

Group: google / paid
{'google_paid_search_spend': 3106, 'google_paid_search_impressions': 3106, 'google_paid_search_clicks': 3106}
All nulls match: 3106 rows
Inactive media days present in data.
```

```
Group: google / shopping
{'google_shopping_spend': 2883, 'google_shopping_impressions': 2883, 'google_shopping_clicks': 2883}
All nulls match: 2883 rows
Inactive media days present in data.

Group: google / pmax
{'google_pmax_spend': 4579, 'google_pmax_impressions': 4579, 'google_pmax_clicks': 4579}
All nulls match: 4579 rows
Inactive media days present in data.

Group: google / display
{'google_display_spend': 7898, 'google_display_impressions': 7898, 'google_display_clicks': 7898}
All nulls match: 7898 rows
Inactive media days present in data.

Group: google / video
{'google_video_spend': 8085, 'google_video_impressions': 8085, 'google_video_clicks': 8085}
All nulls match: 8085 rows
Inactive media days present in data.
```

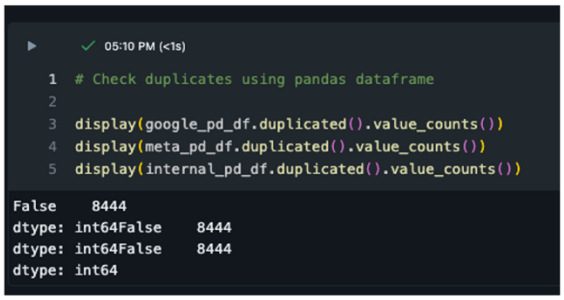
Data cleaning occurs in the “Bronze to Silver” Databricks notebook, which addresses nulls and duplicates, standardizes naming conventions, renames columns, and filters the API data

for the appropriate holidays (Figures 5A–5D). The cleaned holiday array was transformed to line-delimited JSON with ISO date formats and then saved in the silver folder as shown in Figure 6A.

Figure 5

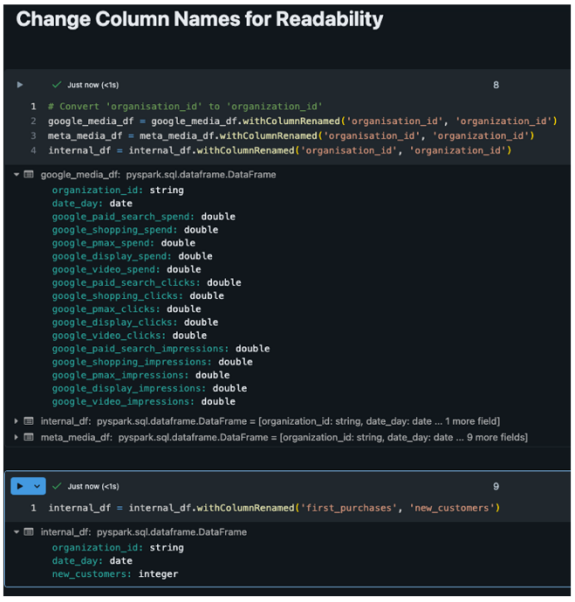
Data Cleaning Code

5A. Checking for Duplicates

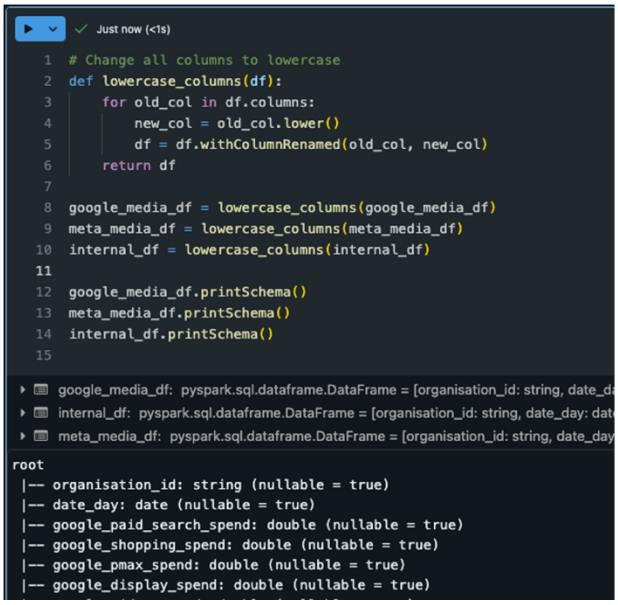


5B. Filling Nulls with Zeros

5C. Renaming Columns



5D. Reformatting Columns



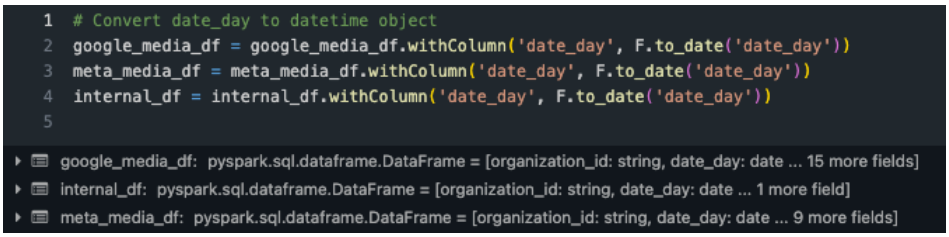
During the silver to gold stage, transformations are completed to make it analysis-ready for the gold stage. These transformations consist of converting the date_day column to a datetime type (Figure 6A), merging the three datasets into one dataframe based on their unique composite

id of organization_id and date_day, adding fixed holidays left out by the API, creating weekend, creating holiday flags, creating a month column, and a click-through rate column (Figures 6B – 6G) was created by calculating clicks divided by impressions to help with exploratory analysis. The write code and storage details are illustrated in Figures 6H–6I.

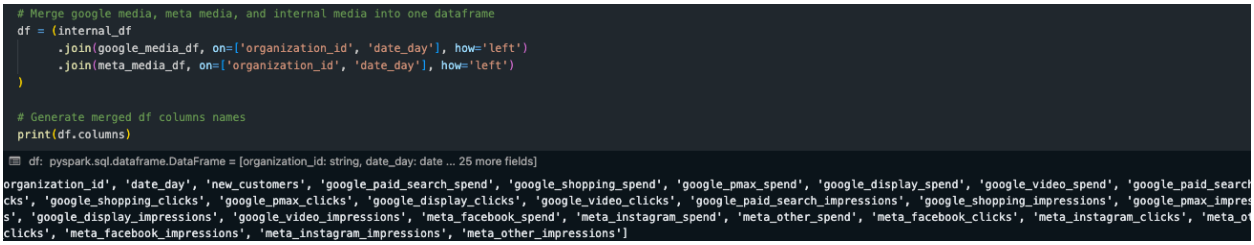
Figure 6

Silver Stage Transformations

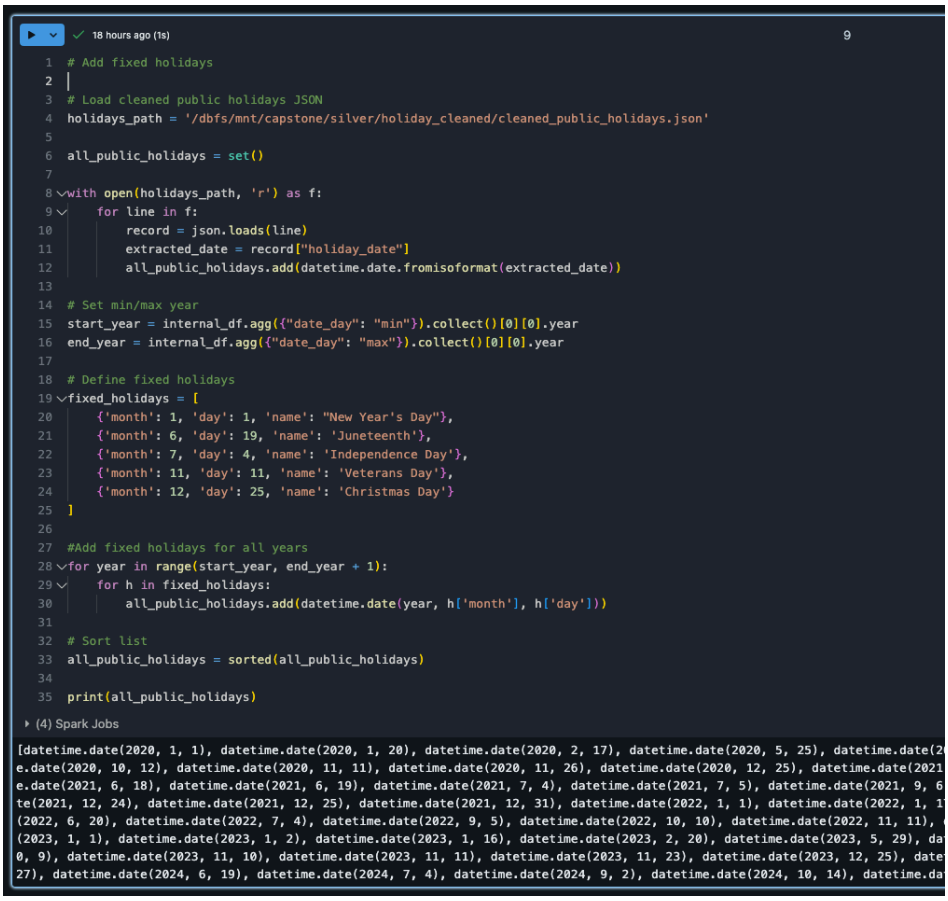
6A. ‘date_day’ Type Transformation



6B. Merging Datasets



6C. Generating Fixed Holidays



```
18 hours ago (1s)                                                    9

# Add fixed holidays

# Load cleaned public holidays JSON
holidays_path = '/dbfs/mnt/capstone/silver/holiday_cleaned/cleaned_public_holidays.json'

all_public_holidays = set()

with open(holidays_path, 'r') as f:
    for line in f:
        record = json.loads(line)
        extracted_date = record["holiday_date"]
        all_public_holidays.add(datetime.date.fromisoformat(extracted_date))

# Set min/max year
start_year = internal_df.agg({"date_day": "min"}).collect()[0][0].year
end_year = internal_df.agg({"date_day": "max"}).collect()[0][0].year

# Define fixed holidays
fixed_holidays = [
    {'month': 1, 'day': 1, 'name': "New Year's Day"},
    {'month': 6, 'day': 19, 'name': 'Juneteenth'},
    {'month': 7, 'day': 4, 'name': 'Independence Day'},
    {'month': 11, 'day': 11, 'name': 'Veterans Day'},
    {'month': 12, 'day': 25, 'name': 'Christmas Day'}
]

#Add fixed holidays for all years
for year in range(start_year, end_year + 1):
    for h in fixed_holidays:
        all_public_holidays.add(datetime.date(year, h['month'], h['day']))

# Sort list
all_public_holidays = sorted(all_public_holidays)

print(all_public_holidays)

▸ (4) Spark Jobs

[datetime.date(2020, 1, 1), datetime.date(2020, 1, 20), datetime.date(2020, 2, 17), datetime.date(2020, 5, 25), datetime.date(2
e.date(2020, 10, 12), datetime.date(2020, 11, 11), datetime.date(2020, 11, 26), datetime.date(2020, 12, 25), datetime.date(2021
e.date(2021, 6, 18), datetime.date(2021, 6, 19), datetime.date(2021, 7, 4), datetime.date(2021, 7, 5), datetime.date(2021, 9, 6
te(2021, 12, 24), datetime.date(2021, 12, 25), datetime.date(2021, 12, 31), datetime.date(2022, 1, 1), datetime.date(2022, 1, 1
(2022, 6, 20), datetime.date(2022, 7, 4), datetime.date(2022, 9, 5), datetime.date(2022, 10, 10), datetime.date(2022, 11, 11),
(2023, 1, 1), datetime.date(2023, 1, 2), datetime.date(2023, 1, 16), datetime.date(2023, 2, 20), datetime.date(2023, 5, 29), da
0, 9), datetime.date(2023, 11, 10), datetime.date(2023, 11, 11), datetime.date(2023, 11, 23), datetime.date(2023, 12, 25), date
27), datetime.date(2024, 6, 19), datetime.date(2024, 7, 4), datetime.date(2024, 9, 2), datetime.date(2024, 10, 14), datetime.da
```

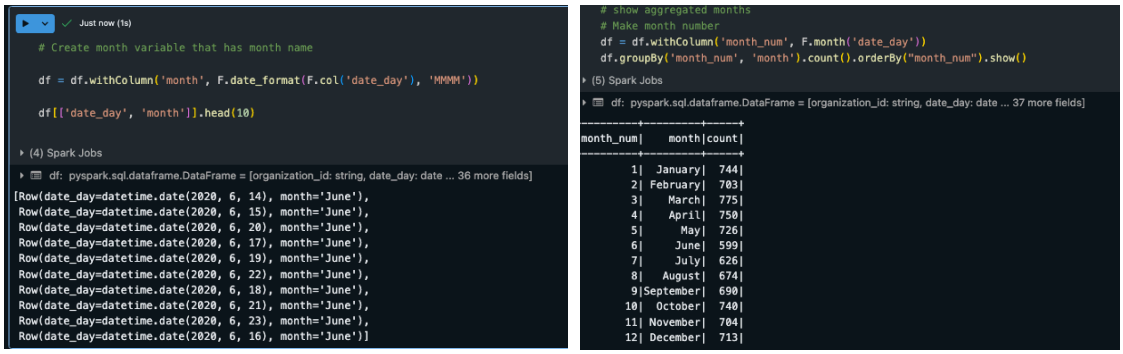
6D. Holiday Flag Creation

```
18 hours ago (1s)                                                    10

# Create Holiday Flag Variable
df = df.withColumn('date_only', F.to_date(F.col('date_day')))

# 3. Create the 'is_public_holiday' flag
df = df.withColumn('is_public_holiday', F.when(F.col('date_only').isin(all_public_holidays), 1).otherwise(0))

# Drop the temporary column
df = df.drop('date_only')

print('Public Holiday Flag Created Successfully.')

▸ df: pyspark.sql.dataframe.DataFrame = [organization_id: string, date_day: date ... 35 more fields]
Public Holiday Flag Created Successfully.
```

6E. Weekend Flag Creation

```
18 hours ago (1s)

# Create Weekend Flag Variable

# 0=Monday, 6=Sunday. Weekends are Saturday (5) and Sunday (6).
df = df.withColumn(
    'day_of_week',
    ((F.dayofweek(F.col("date_day")) + 5) % 7)
)

# Create the flag: 1 if day of week is 5 or 6, 0 otherwise
df = df.withColumn('is_weekend', F.when(F.col('day_of_week') >= 5, 1).otherwise(0))

# Drop the intermediate column
df = df.drop('day_of_week')

print("Weekend Flag Created Successfully.")
print(df[['date_day', 'is_public_holiday', 'is_weekend']].head()) # Output new variables

(4) Spark Jobs

df:  pyspark.sql.dataframe.DataFrame = [organization_id: string, date_day: date ... 35 more fields]

Weekend Flag Created Successfully.
Row(date_day=datetime.date(2020, 6, 14), is_public_holiday=0, is_weekend=1)
```

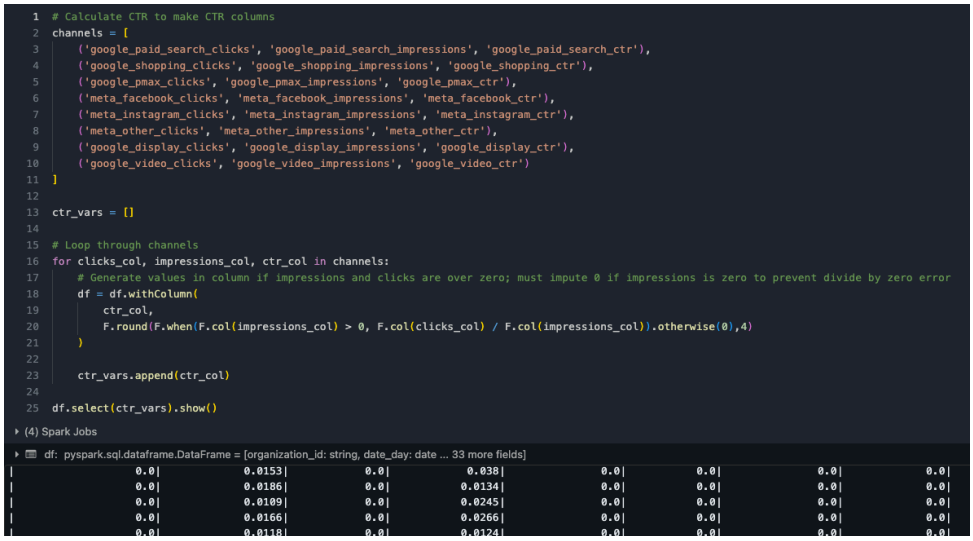
6F. Months Column Creation



```
Just now (1s)

# Create month variable that has month name

df = df.withColumn('month', F.date_format(F.col('date_day'), 'MMMM'))

df[['date_day', 'month']].head(10)

(4) Spark Jobs

df:  pyspark.sql.dataframe.DataFrame = [organization_id: string, date_day: date ... 36 more fields]

[Row(date_day=datetime.date(2020, 6, 14), month='June'),
 Row(date_day=datetime.date(2020, 6, 15), month='June'),
 Row(date_day=datetime.date(2020, 6, 20), month='June'),
 Row(date_day=datetime.date(2020, 6, 17), month='June'),
 Row(date_day=datetime.date(2020, 6, 19), month='June'),
 Row(date_day=datetime.date(2020, 6, 22), month='June'),
 Row(date_day=datetime.date(2020, 6, 18), month='June'),
 Row(date_day=datetime.date(2020, 6, 21), month='June'),
 Row(date_day=datetime.date(2020, 6, 23), month='June'),
 Row(date_day=datetime.date(2020, 6, 16), month='June')]
```

```
# show aggregated months
# Make month number
df = df.withColumn('month_num', F.month('date_day'))
df.groupBy('month_num', 'month').count().orderBy("month_num").show()

(5) Spark Jobs

df:  pyspark.sql.dataframe.DataFrame = [organization_id: string, date_day: date ... 37 more fields]

+---------+---------+-----+
|month_num|    month|count|
+---------+---------+-----+
|        1|  January|  744|
|        2| February|  703|
|        3|    March|  775|
|        4|    April|  750|
|        5|      May|  726|
|        6|     June|  599|
|        7|     July|  626|
|        8|   August|  674|
|        9|September|  690|
|       10|  October|  740|
|       11| November|  704|
|       12| December|  713|
```

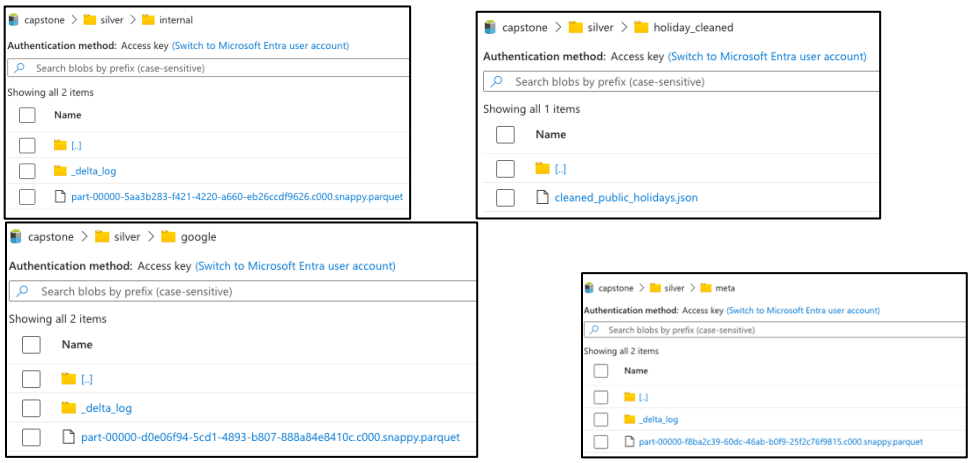
6G. Click-through Rate Variable Creation



```
# Calculate CTR to make CTR columns
channels = [
    ('google_paid_search_clicks', 'google_paid_search_impressions', 'google_paid_search_ctr'),
    ('google_shopping_clicks', 'google_shopping_impressions', 'google_shopping_ctr'),
    ('google_pmax_clicks', 'google_pmax_impressions', 'google_pmax_ctr'),
    ('meta_facebook_clicks', 'meta_facebook_impressions', 'meta_facebook_ctr'),
    ('meta_instagram_clicks', 'meta_instagram_impressions', 'meta_instagram_ctr'),
    ('meta_other_clicks', 'meta_other_impressions', 'meta_other_ctr'),
    ('google_display_clicks', 'google_display_impressions', 'google_display_ctr'),
    ('google_video_clicks', 'google_video_impressions', 'google_video_ctr')
]

ctr_vars = []

# Loop through channels
for clicks_col, impressions_col, ctr_col in channels:
    # Generate values in column if impressions and clicks are over zero; must impute 0 if impressions is zero to prevent divide by zero error
    df = df.withColumn(
        ctr_col,
        F.round(F.when(F.col(impressions_col) > 0, F.col(clicks_col) / F.col(impressions_col)).otherwise(0),4)
    )

    ctr_vars.append(ctr_col)

df.select(ctr_vars).show()

(4) Spark Jobs

df:  pyspark.sql.dataframe.DataFrame = [organization_id: string, date_day: date ... 33 more fields]

|              0.0|           0.0153|              0.0|           0.038|              0.0|              0.0|              0.0|              0.0|
|              0.0|           0.0186|              0.0|          0.0134|              0.0|              0.0|              0.0|              0.0|
|              0.0|           0.0109|              0.0|          0.0245|              0.0|              0.0|              0.0|              0.0|
|              0.0|           0.0166|              0.0|          0.0266|              0.0|              0.0|              0.0|              0.0|
|              0.0|           0.0118|              0.0|          0.0124|              0.0|              0.0|              0.0|              0.0|
```

6H. Silver Stage Writing to Storage Code

6I. Silver Stage Folders



The analysis-ready gold data is stored in the gold folder, and a queryable table is made available for business teams to use (Figure 7A). The gold data is used to produce bivariate visualizations and to uncover patterns or trends during the exploratory analysis and model development phases.

Additional feature engineering and wrangling specific to the regression models for the study are implemented on the gold-level dataset, as shown in Figures 7B through 7E. Transformations for the models consist of the following actions: integrating adstock into the spend variables to account for advertising exposure decay, and generating one-hot encoded features for months and organization IDs.

Adstock is an advertising decay calculation that accounts for delayed consumer response to media exposure (Rubino, 2025). For example, a customer who sees a promotional ad on June 6, 2024, might not make a first purchase until June 8, 2024. Adstock incorporates this delayed effect by allocating a portion of the original day's spend into subsequent days, using the recurrence formula:

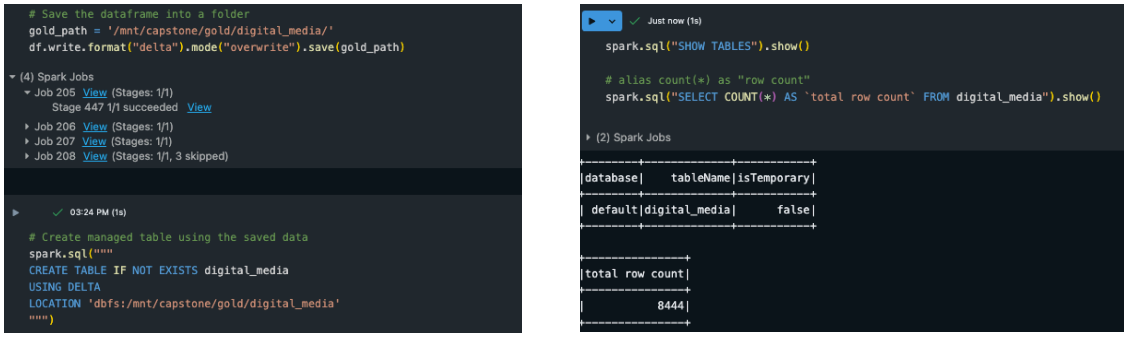
AdStock today = Today's Spend + (Decay Rate × Yesterday's remaining Spend impact)

The decay rate benchmarks per placement, as recommended by Rubino (2025), are depicted in the decay rate dictionary in Figure 7B.

Figure 7

Gold-Level Transformations and Model Preparation Code

7A. Saving Silver to Gold and Gold Table Creation



7B. Integrating Adstock and Log-Transformation to Channel Spend

(Model Preparation)

```python
# Make adstock then log transform spend cols
spend_cols =['google_paid_search_spend', 'google_shopping_spend', 'google_pmax_spend',
             'google_display_spend', 'google_video_spend', 'meta_facebook_spend',
             'meta_instagram_spend', 'meta_other_spend']

df = df.sort_values(['organization_id', 'date_day'])

# Decay rates by channel
decay_rates = {
    'google_paid_search_spend': 0.45,
    'google_shopping_spend': 0.45,
    'google_pmax_spend': 0.45,
    'google_display_spend': 0.65,
    'google_video_spend': 0.75,
    'meta_facebook_spend': 0.70,
    'meta_instagram_spend': 0.70,
    'meta_other_spend': 0.70
}

# Make new list of created adstock columns
adstock_spend_cols = []

# Function to calculate ad stock for a single column
def adstock(series, decay):
    adstock_series = series.copy()
    for i in range(1, len(series)):
        adstock_series.iloc[i] = series.iloc[i] + decay * adstock_series.iloc[i-1]
    return adstock_series

# Apply ad stock to all spend columns
for col in spend_cols:

    decay = decay_rates.get(col)
    df[col] = df[col].clip(lower=1e-6) # Set floor to minimize extreme minimum values
    df[col + '_adstock'] = adstock(df[col], decay=decay)
    adstock_spend_cols.append(col + '_adstock')
    print(f'Adstock integrated to {col}.')

# Loop over continuous columns and log transform using np.log1p() to handle zero values safely
# (avoid negative values and keeps zero as 0)
for col in spend_cols:
   df[f'{col}_adstock_log'] = np.log1p(df[col])
   print(f'Log transformed {col}.')

df[adstock_spend_cols].head()
```

```
▸ df: pandas.core.frame.DataFrame = [organization_id: object, date_day: object ... 54 more fields]
Adstock integrated to google_paid_search_spend.
Adstock integrated to google_shopping_spend.
Adstock integrated to google_pmax_spend.
Adstock integrated to google_display_spend.
Adstock integrated to google_video_spend.
Adstock integrated to meta_facebook_spend.
Adstock integrated to meta_instagram_spend.
Adstock integrated to meta_other_spend.
Log transformed google_paid_search_spend.
Log transformed google_shopping_spend.
Log transformed google_pmax_spend.
Log transformed google_display_spend.
Log transformed google_video_spend.
Log transformed meta_facebook_spend.
Log transformed meta_instagram_spend.
Log transformed meta_other_spend.
```

7C. Months Dummy Variables Creation (Model Preparation)

```python
# Make months dummy variables
# Order months so January is dropped
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']

# Convert the 'month' column to an ordered
df['month'] = pd.Categorical(df['month'], categories=month_order, ordered=True)

month_dummies = pd.get_dummies(df['month'], drop_first=True, dtype=int)

month_dummies.head()
```

month_dummies: pandas.core.frame.DataFrame = [February: int64, March: int64 ... 9 more fields]

	February	March	April	May	June	July	August	September	October	November	December
464	0	0	0	0	1	0	0	0	0	0	0
1069	0	0	0	0	1	0	0	0	0	0	0
7324	0	0	0	0	1	0	0	0	0	0	0
1446	0	0	0	0	1	0	0	0	0	0	0
2637	0	0	0	0	1	0	0	0	0	0	0

7D. Organization ID Dummy Variables (Model Prep)

```python
# Org dummies labeled org alias + last four characters of org ID

df['org_alias_label'] = df['organization_id'].apply(
    lambda x: f"Org_{org_dict[x]}_{x[-4:]}"
)

print(df['org_alias_label'].head(5))

# Now create dummy variables
org_dummies = pd.get_dummies(df['org_alias_label'], drop_first=True, dtype=int)

org_dummies.head()
```

org_dummies: pandas.core.frame.DataFrame = [Org_B_9c1f: int64, Org_C_b1a6: int64 ... 5 more fields]

```
464     Org_A_8d2f
1069    Org_A_8d2f
7324    Org_A_8d2f
1446    Org_A_8d2f
2637    Org_A_8d2f
Name: org_alias_label, dtype: object
```

	Org_B_9c1f	Org_C_b1a6	Org_D_3de0	Org_E_1ea2	Org_F_ee4f	Org_G_3136	Org_H_4fce
464	0	0	0	0	0	0	0
1069	0	0	0	0	0	0	0
7324	0	0	0	0	0	0	0
1446	0	0	0	0	0	0	0
2637	0	0	0	0	0	0	0

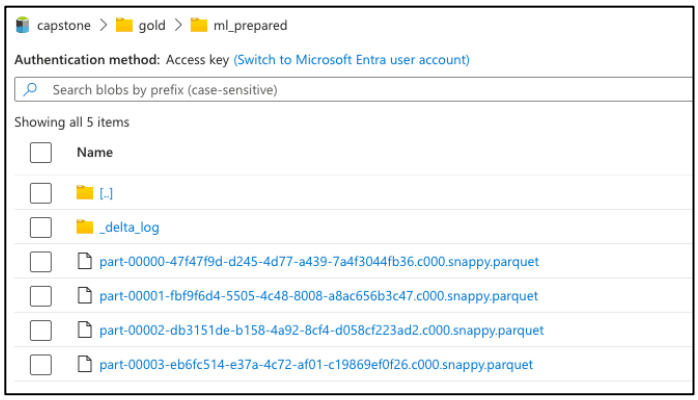
7E. Saving Gold to Machine Learning Folder

```python
# Convert pd df back to spark
spark_df = spark.createDataFrame(df)

# Save df in machine-learning folder
path_name = f'/mnt/capstone/gold/ml_prepared'

spark_df.write.format('delta').mode('overwrite').save(path_name)
print(f'Saved cleaned data to {path_name}')
```

(1) Spark Jobs

spark_df: pyspark.sql.dataframe.DataFrame = [organization_id: string, date_day: date ... 74 more fields]

```
Saved cleaned data to /mnt/capstone/gold/ml_prepared
```



C2. Data Extraction Techniques, Preparation Tools, and Justification

The raw, cleaned, and transformed datasets are stored in the Azure Data Lakehouse (ADLS). One advantage of using ADLS is that it can securely store unstructured, semi-structured, and structured data in one place, making it a single source of integration for analytics environments, such as Databricks (Normesta, n.d.). One disadvantage is that working with ADLS requires understanding its specialized data schemas and data types, in addition to learning the correct methods for efficiently converting the stored data in a Spark or SQL environment.

The Python code for the data extraction, cleaning, and transformations stages is organized in designated Databricks notebooks following the Medallion architecture. The notebooks' separation of concerns ensures maintainability and modularity between the layers (*Databricks Notebooks | Databricks on AWS*, 2025). For example, if only the bronze stage needs updating, such as extracting a new batch of files, only the bronze notebook needs to be run, thereby saving processing power. One disadvantage in managing multiple notebooks is that with each new task, another level of complexity is added to the workflow of the ELT or analytical process.

A Spark cluster in Databricks is initialized to process the ELT tasks in each notebook. Apache Spark's distributed processing and parallelization across worker nodes is advantageous because task activities can remain highly efficient for high-volume data cleaning and

transformations using PySpark (*Apache Spark Overview | Databricks on AWS*, 2025). One disadvantage of using PySpark is that it is not compatible with most Python visualization libraries, such as matplotlib or seaborn, that require Pandas dataframes. A workaround to this issue involves creating a copy of the Spark dataframe and converting it to Pandas for visualizations, while using the original Spark dataframe for cleaning and transformations to maintain distributed processing.

The ELT process integrates Python libraries, PySpark, Apache Spark, and Databricks features to efficiently execute extraction, cleaning, transformation, and storage tasks. The requests and JSON libraries facilitate API data retrieval and JSON payload handling, while pandas and numpy support data manipulation and visualization within Pandas dataframes. The datetime library converts date strings into date objects for variables such as holiday flags. PySpark and Spark SQL enable distributed processing across a Spark cluster, ensuring scalability for large datasets. The missingno library is used to visualize and assess patterns of missing data. Finally, the cleaned and transformed data are stored as Delta tables in ADLS, which provide ACID compliance and support reliable downstream analytics (Normesta, n.d.).

A key limitation of the ELT and data preparation process is its reliance on the completeness and accuracy of the source data. Missing or inconsistent data may introduce biases that cannot be fully corrected through cleaning and transformation.

Section D: Data Analysis

D1. Data Analysis Process and Calculations

The analysis of the cleaned and transformed data consisted of a three-stage approach: exploratory analysis, data model development and training, and model evaluation. Exploratory

data analysis supports the model development and evaluation stages because it provides information on key characteristics of the data, such as distributions, correlations, and trends that affect model assumptions and guide feature engineering decisions (Willems, 2017).

1. Exploratory Analysis (EDA)

The exploratory analysis began with observations of the raw descriptive statistics for the predictor and explanatory variables. The first observation (Figure 8A) consists of evaluating the distribution for the 'new_customers' variable since it will be the target variable for the models. Its main takeaways are the following:

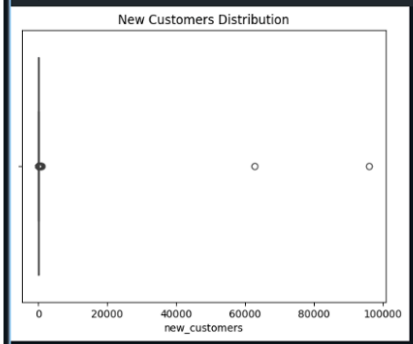
- There is a significant standard deviation, indicating substantial variability between organizations or media spend per channel. The standard deviation is much higher than the mean, pointing to overdispersion, which aligns with the negative binomial model assumption (Rahul, 2025).
- The minimum value of new customers is zero, but this makes sense given that some days are inactive, no-spend days. Seventy-five percent or less of rows had 76 or fewer customers, meaning that most of the data have low counts.
- The maximum value is extreme at ~96,000 new customers, and this is most likely due to an input issue. Outliers were identified and filtered using an interquartile range approach (Figures 8B–8C), resulting in a mean of 51.57 and a median of 33, confirming the skewed distribution (Figure 8D).
- Its right-skewed distribution violates the multiple linear regression requirement of a normal distribution as shown in Figure 8E. It would need to be log-transformed for the model (Bobbitt, 2022).

Figure 8

New Customers Observations and Changes

8A. Raw New Customers Histogram

```
sns.boxplot(x=df['new_customers'])
plt.title('New Customers Distribution')
plt.show()

print(df['new_customers'].describe())
```




```
count     8444.000000
mean        73.106111
std       1249.310612
min          0.000000
25%         11.000000
50%         33.000000
75%         76.000000
max      96010.000000
Name: new_customers, dtype: float64
```

8B. New Customers IQR Code

```
# Gets Q1 and Q3 values
q1, q3 = np.percentile(df['new_customers'], [25, 75])

# Calculate interquartile range
iqr = q3 - q1

# Calculate upper limit
upper = q3 + (3 * iqr)
print('Upper Limit:', upper)

# Set variable for count
upper_count = 0

# Count outliers in Population
for x in df['new_customers']:
    if x > upper:
        upper_count += 1

print('Upper Outliers:', upper_count)

range_val = df['new_customers'].max() - upper
print('Upper Outlier Range:', range_val)
```

```
Upper Limit: 271.0
Upper Outliers: 71
Upper Outlier Range: 95739.0
```

8C. Filter Code

```
df = df[df['new_customers'] <= upper]
print(df['new_customers'].max())
```

```
▸ df: pandas.core.frame.DataFrame = [organization_id: object, date_day: object
Upper Limit: 271.0
271
```

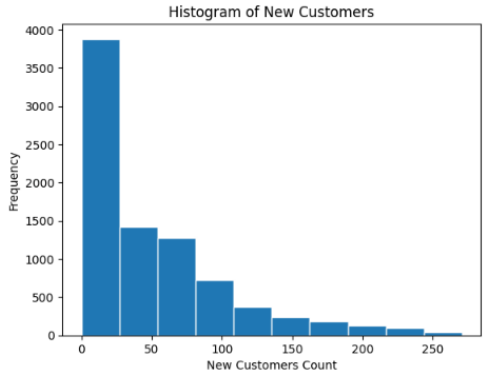
8D. Descriptive Statistics After Treatment

```
df['new_customers'].describe()
```

```
count    8373.000000
mean       51.593336
std        52.643205
min         0.000000
25%        11.000000
50%        33.000000
75%        74.000000
max       271.000000
Name: new_customers, dtype: float64
```

8E. Histogram After Treatment

```
plt.hist(df['new_customers'], edgecolor='white')
plt.title('Histogram of New Customers')
plt.xlabel('New Customers Count')
plt.ylabel('Frequency')
plt.show()
```



Subsequently, log(x+1) transformations using np.log1p were applied across the continuous variables to address the high number of structural zeros (i.e., no-spend days). Log transformations address the extreme skewness, stabilizing the data's variance, enabling the performance of meaningful bivariate analyses, and supporting subsequent regression model developments (GeeksforGeeks, 2025).

The log-transformed data histogram (Figure 9A) reveals a notable bimodal distribution with subgroups in the data, showing a large group of days generating a lower number of new daily customers and another large group of days achieving a much higher count of new customers.

The histogram in Figure 9B, showing only the active media subset, confirms that these two groups are not due to inactive days, and their bimodal distribution persists even after the inactive days are excluded. This graph indicates that the two customer acquisition levels (low versus high) are driven by differences within active campaigns, not just by the presence or absence of media spend. The first peak of days, as shown in Figure 9C, consists of low customer counts, around eight new customers per day, and the second peak has a much higher count, about seventy-eight new customers. This pattern suggests that the combined campaigns result in

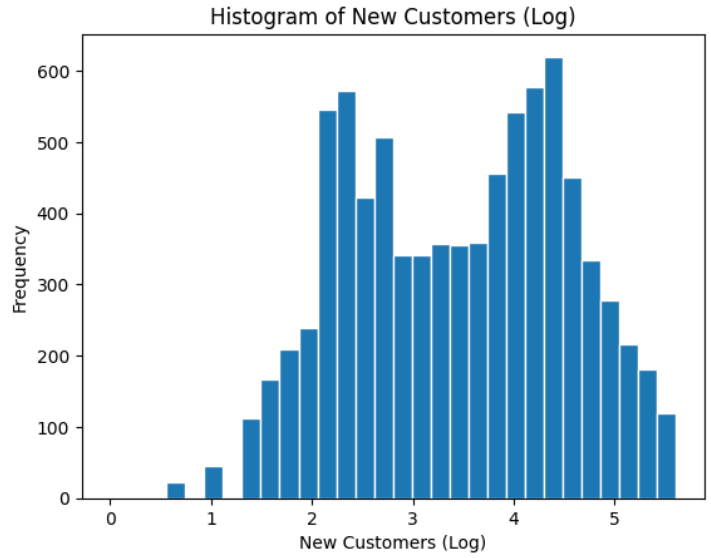
prolonged periods of low new customer days, followed by sudden spikes in customer acquisition days.

Figure 9

Logarithmic Transformations

9A. New Customers After Log-Transform Filtered for Inactive Days

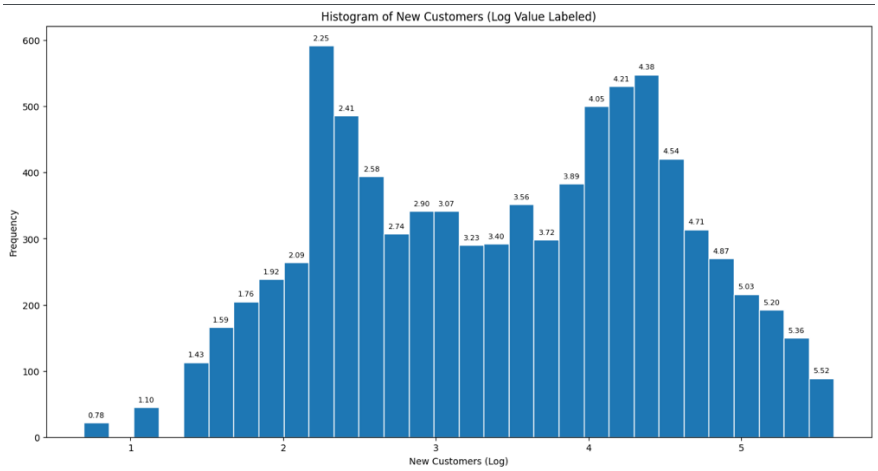
```
# Add a small constant to avoid log(0)
df['new_customers_log'] = np.log1p(df['new_customers'])

# QQ plot for log-transformed data
plt.hist(df['new_customers_log'], bins=30, edgecolor='white')
plt.title('Histogram of New Customers (Log)')
plt.xlabel('New Customers (Log)')
plt.ylabel('Frequency')
plt.show()
```



9B. New Customers (Log) Filter for Active Days Only

```
# make copy of a df but only with spend_cols > 0
media_on = df[df[spend_cols].sum(axis=1) > 0]

fig, ax = plt.subplots(figsize=(16, 8))

# create histogram
counts, bins, patches = plt.hist(media_on['new_customers_log'], bins=30, edgecolor='white')

# get bin midpoints
bin_mids = 0.5 * (bins[1:] + bins[:-1])

# Add log value labels so we can get the values of the peaks
for x_mid, y in zip(bin_mids, counts):
    if y > 0:  # only label non-empty bins
        plt.text(x_mid, y + max(counts)*0.01, f"{x_mid:.2f}",
                 ha='center', va='bottom', fontsize=8, rotation=0)

plt.title('Histogram of New Customers (Log Value Labeled)')
plt.xlabel('New Customers (Log)')
plt.ylabel('Frequency')
plt.show()
```

media_on: pandas.core.frame.DataFrame = [organization_id: object, date_day: datetime64[ns] ... 38 more fields]



9C. Peak Values Exponentiation

```
# Revert logged value to raw value
peak_one = np.expm1(2.25)
peak_two = np.expm1(4.37)

print(f'Peak one new customers is {peak_one:.0f}')
print(f'Peak two new customers is {peak_two:.0f}')

Peak one new customers is 8
Peak two new customers is 78
```

The raw descriptive statistics of the continuous predictor variables (spend, impressions, clicks, and click-through rate) are also right-skewed, as indicated by the mean being higher than the median across Figures 10A through 10D (Kosourova, 2025). The inactive days are also prevalent in several channels. Specifically, Google PMax, Google Display, Google Video, and

Meta Other show zero values in 50% or more of the data across spend, impressions, clicks, and click-through rate (CTR), indicating they are deployed occasionally. Google PMax notably has a low number of ‘media on’ days. However, on the days when the channel is active, there are high investments (max spend: approximately $4,000), resulting in a spike in impressions and clicks. Meta Facebook is the most consistently and highly invested channel with a significant average spend (~$728), driving efficient campaigns with a strong CTR of ~ 2% (Molnar, 2025). Google Paid Search appears to be much more efficient, with a CTR of 6.39% (Moons, 2025). However, this metric may be misleading, as Paid Search operates at a lower level of the purchase funnel, capturing high-intent users who are already looking to buy items. In contrast, Meta Facebook, as an ‘always-on’ channel, generates a much higher mean clicks at ~877 versus Google Paid Search’s ~97 mean clicks. Assuming Meta Facebook’s larger click volume translates to a higher total number of conversions, its value as a scalable channel likely outweighs Google Paid Search's higher click rate.

Figure 10

New Customers and Continuous Variables Descriptive Statistics

10A. Spend Descriptive Statistics

```
# Make list of spend, clicks, impressions, and ctr
spend_cols = ['google_paid_search_spend', 'google_shopping_spend', 'google_pmax_spend', 'google_display_spend', 'google_video_spend', 'meta_facebook_spend',
'meta_instagram_spend', 'meta_other_spend']

df[spend_cols].describe()
```

	google_paid_search_spend	google_shopping_spend	google_pmax_spend	google_display_spend	google_video_spend	meta_facebook_spend	meta_instagram_spend	meta_other_spend
count	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000
mean	111.999188	109.938747	222.555839	0.846990	2.226776	728.061344	167.030984	0.659080
std	179.664011	176.662166	412.302809	5.265753	31.868686	1096.250169	539.956466	6.811191
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	157.460000	0.000000	0.000000
50%	33.740000	33.620000	0.000000	0.000000	0.000000	365.450000	0.000000	0.000000
75%	156.920000	136.780000	334.280000	0.000000	0.000000	803.950000	0.000000	0.000000
max	2051.804113	1478.520000	4146.857037	100.770000	2324.390000	18905.400789	7116.750000	218.680000

10B. Impressions Descriptive Statistics

```
impressions_cols = ['google_paid_search_impressions', 'google_shopping_impressions', 'google_pmax_impressions', 'google_display_impressions',
'google_video_impressions', 'meta_facebook_impressions', 'meta_instagram_impressions', 'meta_other_impressions']

df[impressions_cols].describe()
```

	google_paid_search_impressions	google_shopping_impressions	google_pmax_impressions	google_display_impressions	google_video_impressions	meta_facebook_impressions	meta_instagram_impressions	meta_other_impressions
count	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000
mean	1265.395318	16248.698794	18698.223815	398.614714	173.979577	43030.600024	9578.983877	51.831243
std	2726.349152	27417.604295	31680.238000	2269.024538	2165.302079	56396.593462	29025.052818	514.836048
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	13082.000000	0.000000	0.000000
50%	404.000000	3821.000000	0.000000	0.000000	0.000000	27142.000000	0.000000	0.000000
75%	1690.000000	19680.000000	26006.000000	0.000000	0.000000	51268.000000	0.000000	0.000000
max	80229.000000	718345.000000	660906.000000	48791.000000	84104.000000	985501.000000	332996.000000	23856.000000

10C. Clicks Descriptive Statistics

```
clicks_cols = ['google_paid_search_clicks', 'google_shopping_clicks', 'google_pmax_clicks', 'google_display_clicks', 'google_video_clicks', 'meta_facebook_clicks',
'meta_instagram_clicks', 'meta_other_clicks']

df[clicks_cols].describe()
```

	google_paid_search_clicks	google_shopping_clicks	google_pmax_clicks	google_display_clicks	google_video_clicks	meta_facebook_clicks	meta_instagram_clicks	meta_other_clicks
count	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000
mean	97.447749	144.991401	197.734981	2.688523	0.783112	877.201481	152.600024	0.610773
std	162.076400	347.410611	341.383096	23.373169	10.514478	1158.217087	471.024895	9.967329
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	249.000000	0.000000	0.000000
50%	23.000000	40.000000	0.000000	0.000000	0.000000	524.000000	0.000000	0.000000
75%	107.000000	157.000000	256.000000	0.000000	0.000000	1079.000000	0.000000	0.000000
max	1098.000000	17212.000000	3099.000000	743.000000	789.000000	16646.000000	6643.000000	509.000000

10D. Click-Through Rate (CTR) Descriptive Statistics


```
clicks_cols = ['google_paid_search_clicks', 'google_shopping_clicks', 'google_pmax_clicks', 'google_display_clicks', 'google_video_clicks', 'meta_facebook_clicks',
'meta_instagram_clicks', 'meta_other_clicks']

df[clicks_cols].describe()
```

	google_paid_search_clicks	google_shopping_clicks	google_pmax_clicks	google_display_clicks	google_video_clicks	meta_facebook_clicks	meta_instagram_clicks	meta_other_clicks
count	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000	8373.000000
mean	97.447749	144.991401	197.734981	2.688523	0.783112	877.201481	152.600024	0.610773
std	162.076400	347.410611	341.383096	23.373169	10.514478	1158.217087	471.024895	9.967329
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	249.000000	0.000000	0.000000
50%	23.000000	40.000000	0.000000	0.000000	0.000000	524.000000	0.000000	0.000000
75%	107.000000	157.000000	256.000000	0.000000	0.000000	1079.000000	0.000000	0.000000
max	1098.000000	17212.000000	3099.000000	743.000000	789.000000	16646.000000	6643.000000	509.000000

Due to the right-skew distribution across all the continuous variables, a log(x+1) transformation in Figure 11A was applied to reduce the effect of extreme values and compress the range (Fatima, n.d.). The transformed variables were then plotted to better visualize the distributions of active media days (Figure 11B-11D). CTR was excluded from plotting, as the descriptive statistics sufficiently characterize its distribution, and a plot would not provide additional value. Consequently, for visualizations, continuous variables were analyzed using only active media days—days with non-zero spend, impressions, and clicks, across all channels. This

approach enables clearer interpretation of variable distributions and relationships, eliminating the structural zeros introduced by inactive periods. However, for the regression modeling phase, all days - including inactive ones - were retained to preserve the entire time-based sequence of the data and avoid introducing bias by removing structurally valid zero-activity days

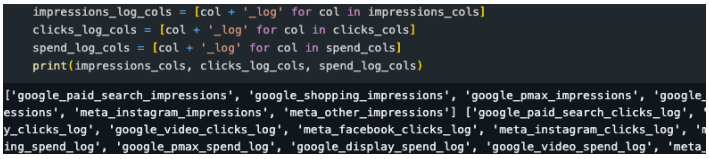
Figure 11

Log Transformation for Continuous Variables

11A. Code Examples for Log Transformation and Plotting

```
cont_cols = spend_cols + impressions_cols + clicks_cols
# Loop over continuous columns and log transform using np.log1p() to handle zero values safely
# (avoid negative values and keeps zero as 0)
for col in cont_cols:
    df[f'{col}_log'] = np.log1p(df[col])
    print(df[f'{col}_log'].describe())
```

```
count    8373.000000
mean        2.816471
std         2.453231
min         0.000000
25%         0.000000
50%         3.547892
75%         5.062089
max         7.626962
Name: google_paid_search_spend_log, dtype: float64
```

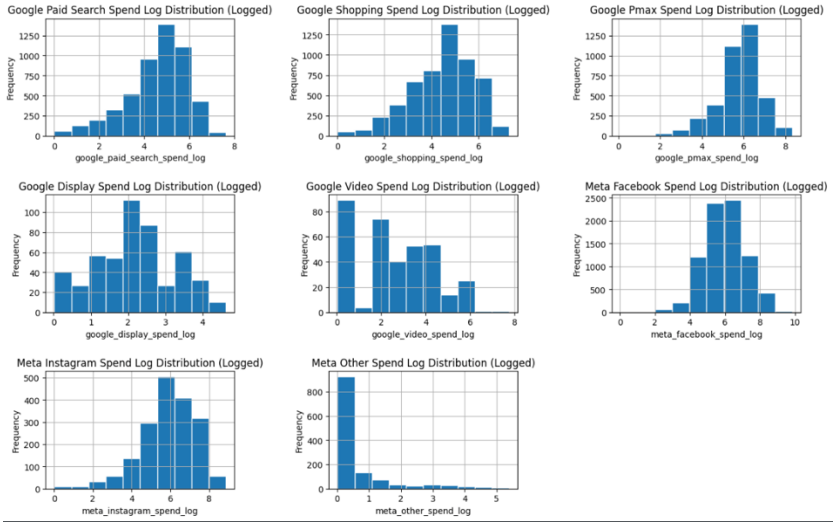


```
impressions_log_cols = [col + '_log' for col in impressions_cols]
clicks_log_cols = [col + '_log' for col in clicks_cols]
spend_log_cols = [col + '_log' for col in spend_cols]
print(impressions_cols, clicks_log_cols, spend_log_cols)
```

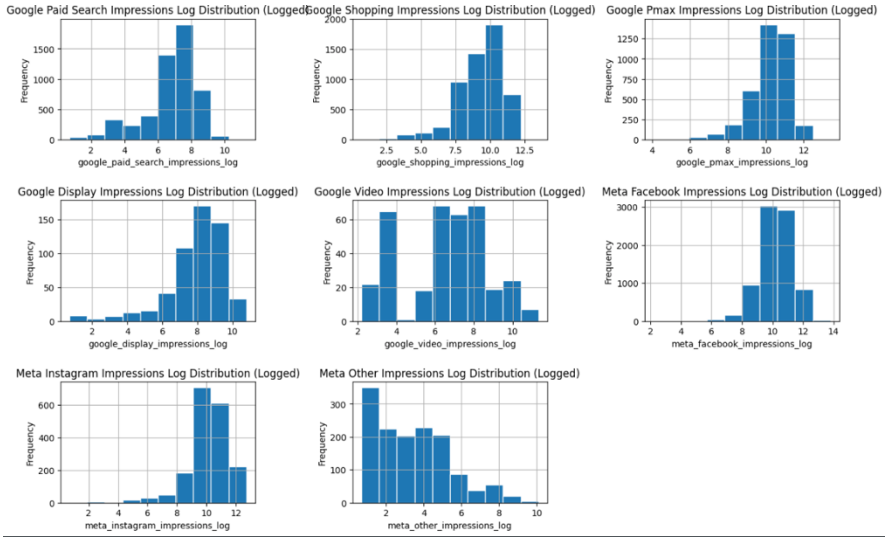
```
['google_paid_search_impressions', 'google_shopping_impressions', 'google_pmax_impressions', 'google_
essions', 'meta_instagram_impressions', 'meta_other_impressions'] ['google_paid_search_clicks_log', '
y_clicks_log', 'google_video_clicks_log', 'meta_facebook_clicks_log', 'meta_instagram_clicks_log', 'm
ing_spend_log', 'google_pmax_spend_log', 'google_display_spend_log', 'google_video_spend_log', 'meta_
```

```
fig, (axes) = plt.subplots(3, 3, figsize=(16, 10))
fig.subplots_adjust(hspace= .5, wspace=.5)

for var, ax in zip(spend_log_cols, axes.flat):
    var_title = var.replace('_', ' ').title()

    # Filter non-spend days
    data = df[df[var] > 0][var]

    ax.set_title(f'{var_title} Distribution (Logged)')
    ax.set_ylabel('Frequency')
    ax.set_xlabel(var)

    data.hist(ax=ax, edgecolor='white')

axes[2,2].remove()
plt.show()
```

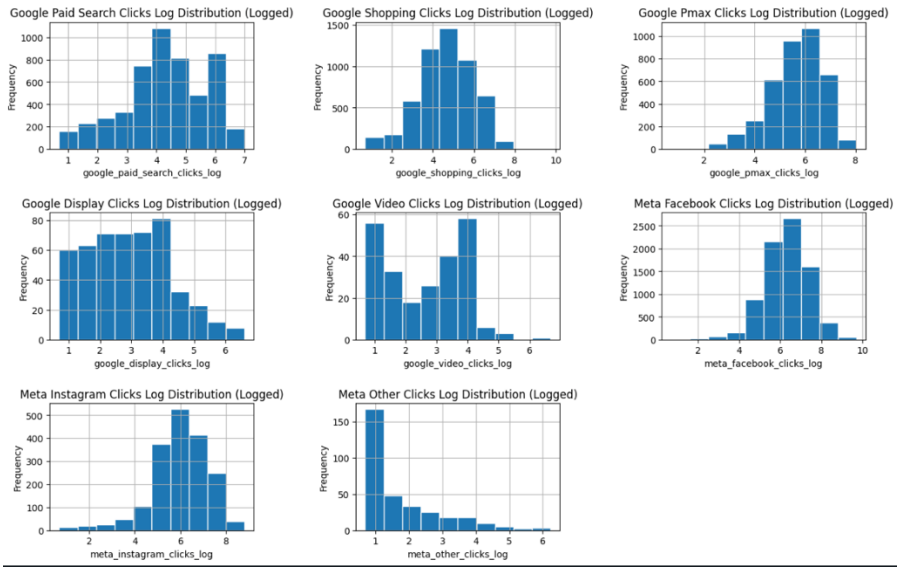
11B. Visualization: Log-Transformed Spend by Channels



11C. Visualization: Log-Transformed Impressions by Channels



11D. Visualization: Log-Transformed Clicks by Channels



The log-transformed histograms from Figure 11B through 11D reveal clear distinctions in consistency and scale across the different channels. Google Shopping, Meta Facebook, and Google PMax show the highest level of stability, with normal distributions across spend, clicks, and impressions. These channels show predictable performance and suggest they will be reliable coefficients for the regression models. Google Display and Google Video, in contrast, exhibit irregular and sporadic distributions, suggesting inconsistent implementation of both channels, possibly due to sudden, short campaigns.

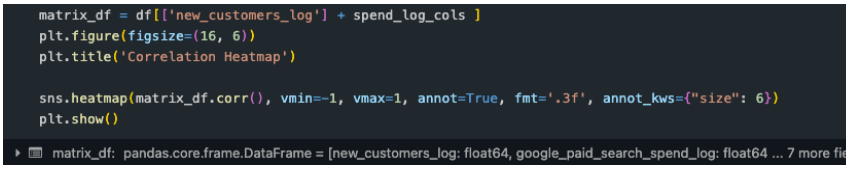
The correlation heat maps in Figures 12A to 12C illustrate the statistical relationships between the log of new customers and the log of spend, impressions, and clicks per channel, as indicated by the correlation coefficients (*r)*. The spend by channel relationships in Figure 12A show that Google PMax Spend ($r = 0.299$) and Google Paid Search Spend ($r = 0.272$) have the strongest correlation in driving new customers. In contrast, Google Display Spend has a weak and negative correlation value of -0.080. Interestingly, Meta Facebook's spend has a weak correlation with new customers, despite driving a high number of clicks and impressions. The click and impression heat maps demonstrate the collinear effects of the spend variables, as each

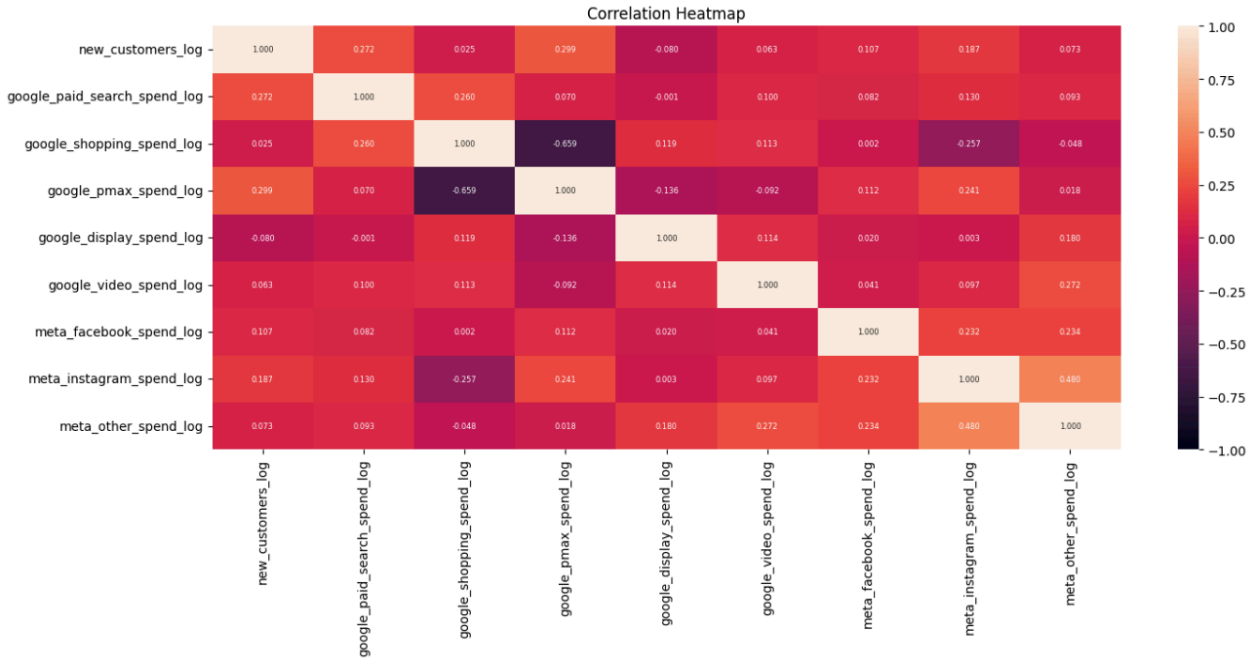
channel closely follows the same correlations with new customers. There is a strong negative collinearity between Google PMax and Google Shopping across spend, impressions, and clicks, with *r* values ranging from -0.654 to - 0.732, indicating that one or the other channel may be diverting traffic from the other, as they originate from the same ad platform. The spend channels are selected as variables for the regression model, given the focus on media budget allocation in this study. All identified multicollinearities will be addressed using the VIF-based reduction method prior to modeling.

Figure 12

Bivariate Visualizations for Media Continuous Variables

12A. Spend Correlation Heatmap

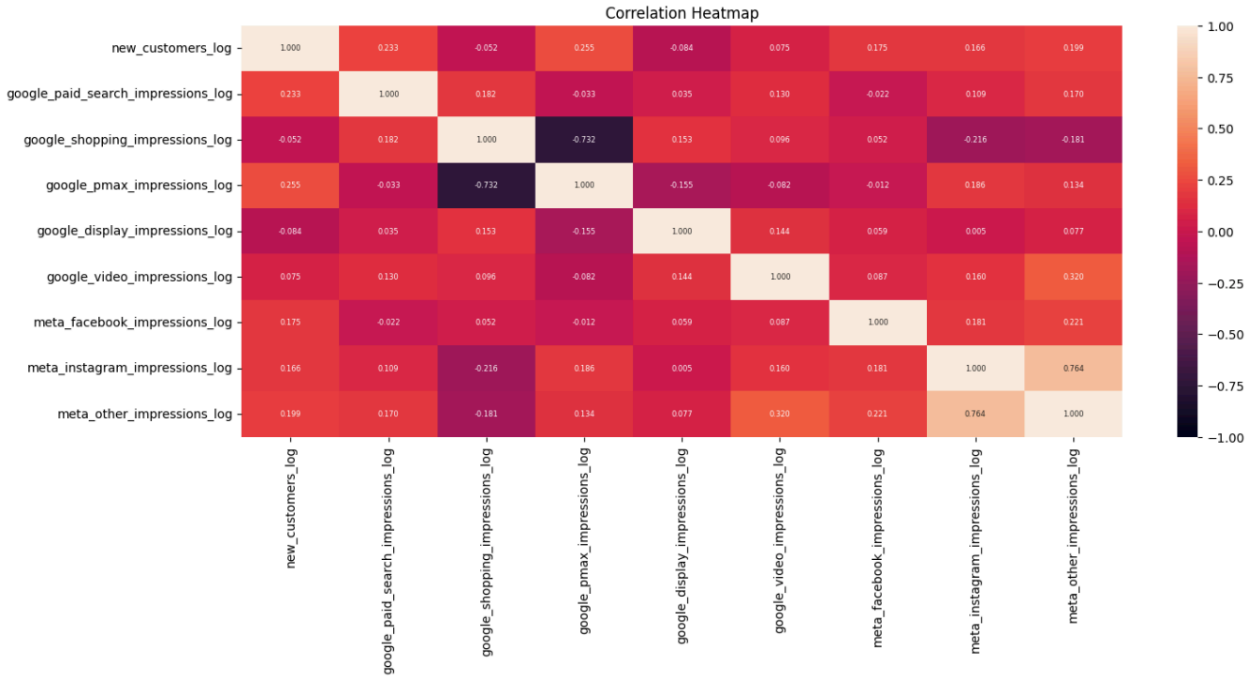




12B. Impressions Correlation Heat Map

```
# Generate heat map for continous variables for logged spend
matrix_df = df[['new_customers_log'] + impressions_log_cols]
plt.figure(figsize=(16, 6))
plt.title('Correlation Heatmap')

sns.heatmap(matrix_df.corr(), vmin=-1, vmax=1, annot=True, fmt='.3f', annot_kws={"size": 6})
plt.show()
```

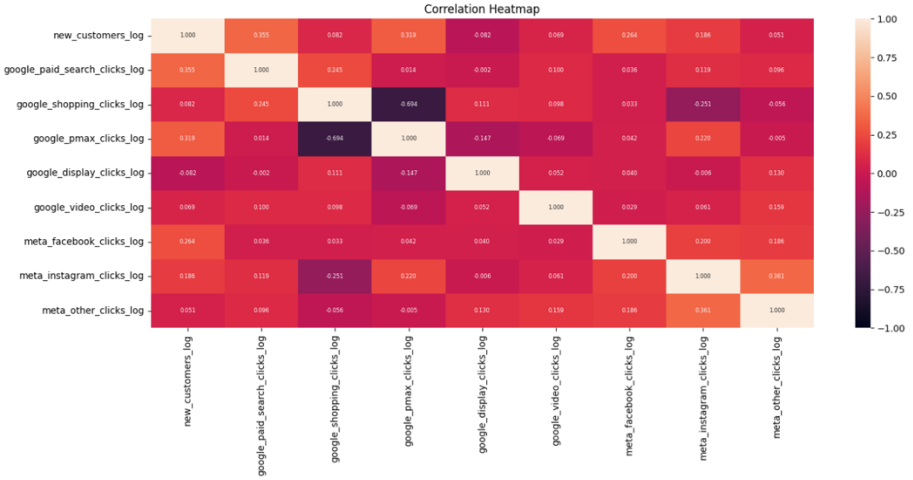
matrix_df: pandas.core.frame.DataFrame = [new_customers_log: float64, google_paid_search_impressions_log: float64 ... 7 more fields]



12C. Clicks Correlation Heat Map

```
# Generate heat map for continous variables for logged spend
matrix_df = df[['new_customers_log'] + clicks_log_cols]
plt.figure(figsize=(16, 6))
plt.title('Correlation Heatmap')

sns.heatmap(matrix_df.corr(), vmin=-1, vmax=1, annot=True, fmt='.3f', annot_kws={"size": 6})
plt.show()
```

matrix_df: pandas.core.frame.DataFrame = [new_customers_log: float64, google_paid_search_clicks_log: float64 ... 7 more fields]



Univariate analysis and visualizations in Figure 13 were also completed for the categorical variables is_public_holiday, is_weekend, organization_id, and no-spend day indicators across channels. Binary flag variables ([channel]_no_spend_day), shown in Figure 13A, were created for the EDA phase to explicitly capture the effect of no-spend periods on new customer acquisitions to understand patterns of inactive day effects. To make organization_id visualizations more readable, an alias variable was created that maps back to each organization in the id column (Figure 13B).

The organization chart in Figure 13C displays an imbalanced distribution, with three organizations accounting for most of the data, ranging from 15.8% to 18.8%. There is a notable trail with organizations E and G accounting for merely 5.5% and 6% of the data, respectively. Its uneven distribution must be reflected in the model as individual organizational features to represent the influence of each organization on the coefficients accurately. There is an equal number of months in the dataset, which suggests that months will be a reliable feature for the model, potentially denoting seasonality. About 3.5% of the total data consists of public holidays, and 28.5% of the days are weekends. While public holidays account for only a small portion of the data, including them in the model is significant for distinguishing high seasonality and conversation time periods, such as Thanksgiving and Christmas Sales. The is_weekend variable could be a valuable indicator for shifts in new customer acquisitions throughout the week.

The no-spend day per channel variables show that a large percentage of the data have inactive media days. In contrast, Google Paid Search, Meta Facebook, and Google Shopping have the most active media days. Meta Facebook and Google Paid Search have synonymously high performance across impressions and clicks. However, Google Shopping falls behind the third-highest performing channel, Google PMax, despite having a higher number of active days

by ~19%. Including no-spend day indicators is essential to exploratory analysis because zero-spend days represent the absence of media activity, not low performance. These rows are reflected as spend = 0 in the spend variables for the regression models.

Figure 13

Categorical Variable Univariate Visualizations

13A. *_no_spend_day Flags Creation

13B. Organization Alias Creation

```
    # get unique organizations ids
    orgs = df['organization_id'].unique()
    print(orgs)
```

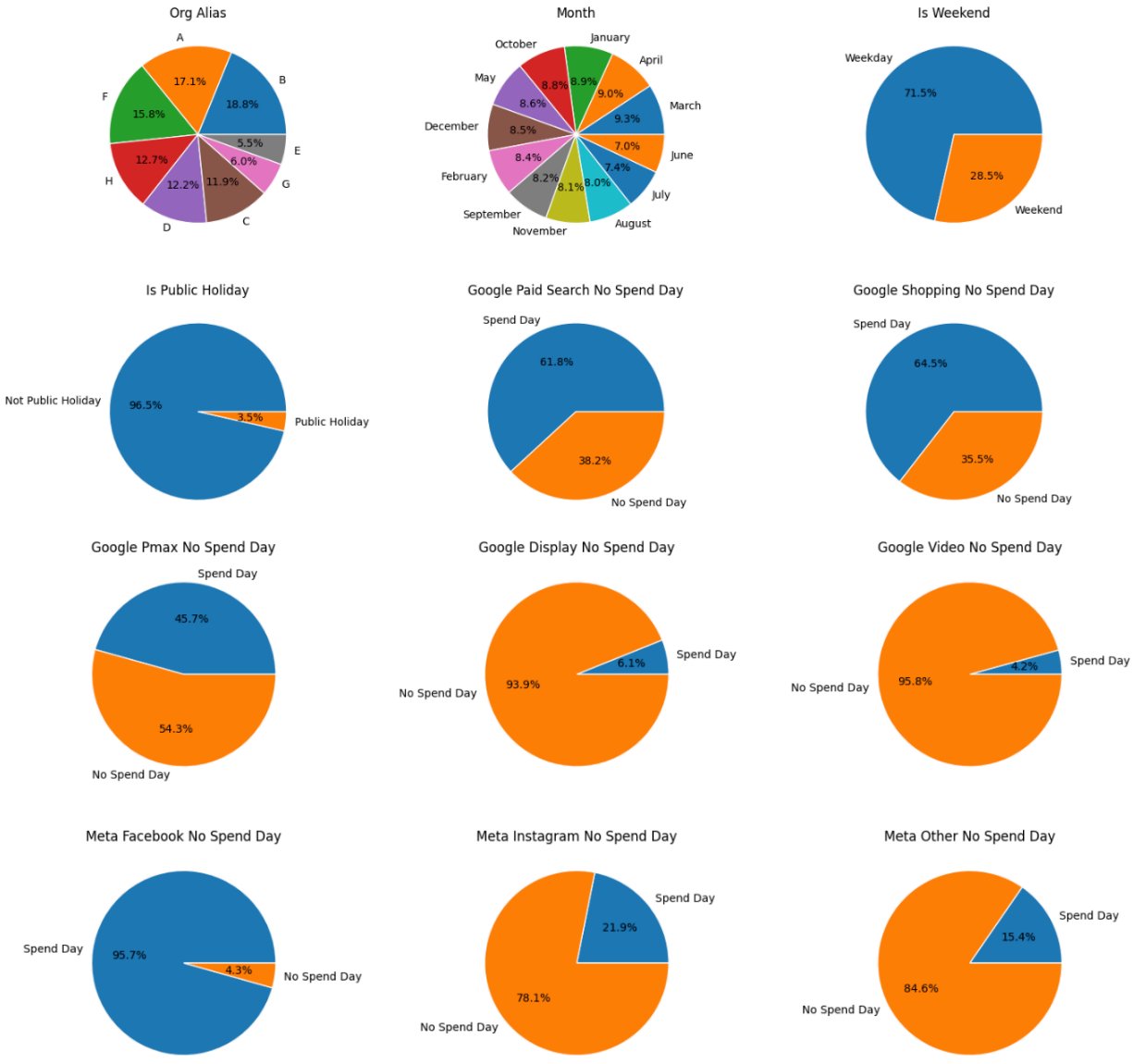
```
['09113a73f8618b48b3cf53b24cd78d2f' '2b15eedfc4faa6293dc4b3bf8c1c9c1f'
 '4a762f02ca755b22d37393e8dbeab1a6' '560f5cf4dce8824a907c84162e553de0'
 '6be433bd50b2da88aa18f31056a71ea2' '70f3b9fb9a8f232814ce705d7641ee4f'
 '8c7d01375c3eca8e3ecfe8f1872b3136' 'ff5c2c32d7f1c21594c518fc6a6d4fce']
```

```
orgs = ['09113a73f8618b48b3cf53b24cd78d2f', '2b15eedfc4faa6293dc4b3bf8c1c9c1f',
 '4a762f02ca755b22d37393e8dbeab1a6', '560f5cf4dce8824a907c84162e553de0',
 '6be433bd50b2da88aa18f31056a71ea2', '70f3b9fb9a8f232814ce705d7641ee4f',
 '8c7d01375c3eca8e3ecfe8f1872b3136', 'ff5c2c32d7f1c21594c518fc6a6d4fce']

# Make variable where Organization Id has alias variable like Org Alias: 'A'
org_alias = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Create dictionary of organization ids and aliases
org_dict = dict(zip(orgs, org_alias))

# Create variable with organization aliases that map back to organization
# '09113a73f8618b48b3cf53b24cd78d2f' = 'Org. A'
org_alias = df['organization_id'].map(org_dict)
df['org_alias'] = org_alias

# make dataframe with count for organization_id and org_alias using pandas
org_map = df.groupby(['org_alias', 'organization_id']).count().reset_index()
org_count = df.groupby(['org_alias']).count().reset_index()

print('Organization Map:')
print(org_map[['org_alias', 'organization_id']])
print('Organization Counts:')
print(org_count[['org_alias', 'organization_id']])
```

```
org_count: pandas.core.frame.DataFrame = [org_alias: object, orga
org_map: pandas.core.frame.DataFrame = [org_alias: object, organ
Organization Map:
  org_alias                   organization_id
0         A  09113a73f8618b48b3cf53b24cd78d2f
1         B  2b15eedfc4faa6293dc4b3bf8c1c9c1f
2         C  4a762f02ca755b22d37393e8dbeab1a6
3         D  560f5cf4dce8824a907c84162e553de0
4         E  6be433bd50b2da88aa18f31056a71ea2
5         F  70f3b9fb9a8f232814ce705d7641ee4f
6         G  8c7d01375c3eca8e3ecfe8f1872b3136
7         H  ff5c2c32d7f1c21594c518fc6a6d4fce
Organization Counts:
  org_alias  organization_id
0         A             1428
1         B             1576
2         C              997
3         D             1021
4         E              459
5         F             1322
6         G              505
7         H             1064
```

13C. Categorical Pie Chart Code and Visualization

```python
# Generate categorical univariate
cat_vars = ['org_alias', 'month', 'is_weekend', 'is_public_holiday'] + no_spend_cols

# Univariate for Categorical; By Percentage Frequency

# Establish figure and axes
fig, axes = plt.subplots(4, 3, figsize=(18, 18))
fig.subplots_adjust(hspace=.25, wspace=.25)

# Loop over list of variables to generate pie charts
for var, ax in zip(cat_vars, axes.flat):

    binary_vars = ['is_weekend', 'is_public_holiday'] + no_spend_cols

    # Make labels match order of binary values; else match values to labels
    if var in binary_vars:
        labels, count = get_binary_labels(var)
    else:
        count = df[var].value_counts().values.tolist()
        labels = df[var].value_counts().index.tolist()

    var_title = var.replace('_', ' ').title()
    ax.set_title(var_title)

    ax.pie(count,
           labels=labels,
           autopct='%1.1f%%',
           wedgeprops = {"edgecolor" : "white",
                         'linewidth': 1})
plt.show()
```



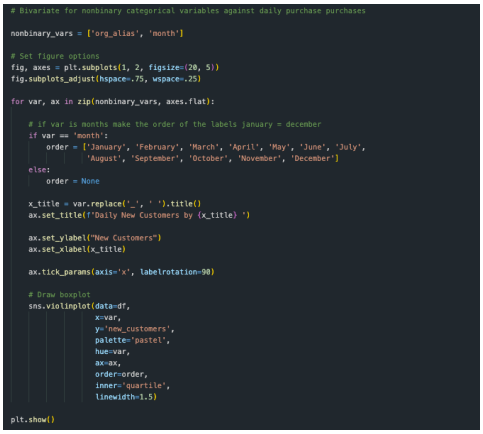
The bivariate visualizations and descriptive statistics (Figures 14A – 14C) display the distribution of each categorical variable against new customers. The organization violin chart in Figure 14B shows a high degree of median variance between different organizations and new customers, with the lowest at ~10 and the highest at ~100 new customers. Some organizations (C, D, A) exhibit a wide interquartile range, indicating high variability and volatility. Organization C is notable because it has the highest median for new customers, the highest volatility, and a bi-modal distribution. Given that it accounts for only ~12% of the data, Organization C exhibits sporadic campaign activities yet generate a significant number of acquisitions. Although organizations B and H have the lowest number of first purchase customers, they have the narrowest IQR, suggesting more stable and consistent activity.

Figure 14

Bivariate Analysis of Categorical Variables

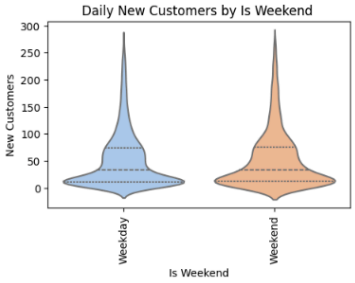
14A. Categorical Bivariate Graph Code

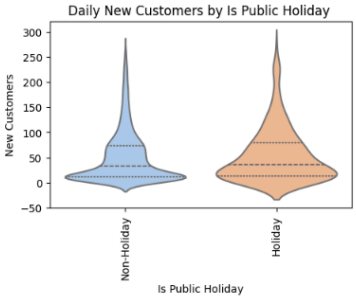
```
# Bivariate for binary categorical variables against daily purchase purchases
binary_vars = ['is_weekend', 'is_public_holiday'] + no_spend_cols

# Generate display df function
def make_display_df(df, var):

    temp_df = df.copy()

    if var == 'is_weekend':
        temp_df['is_weekend'] = df['is_weekend'].replace({0: 'Weekday', 1: 'Weekend'})
        order=['Weekday', 'Weekend']
    elif var == 'is_public_holiday':
        temp_df['is_public_holiday'] = df['is_public_holiday'].replace({0: 'Non-Holiday', 1: 'Holiday'})
        order=['Non-Holiday', 'Holiday']
    else:
        temp_df[var] = df[var].replace({0: 'No Spend Day', 1: 'Spend Day'})
        order=['No Spend Day', 'Spend Day']

    return temp_df, order

# Set figure options
fig, axes = plt.subplots(4, 3, figsize=(16, 20))
fig.subplots_adjust(hspace=.75, wspace=.25)

for var, ax in zip(binary_vars, axes.flat):
    # Make labels match order of binary values; else match values to labels
    plot_df, order = make_display_df(df, var)

    x_title = var.replace('_', ' ').title()
    ax.set_title(f'Daily New Customers by {x_title} ')

    ax.set_ylabel("New Customers")
    ax.set_xlabel(x_title)

    ax.tick_params(axis='x', labelrotation=90)

    # Draw boxplot
    sns.violinplot(data=plot_df,
                   x=var,
                   y='new_customers',
                   palette='pastel',
                   hue=var,
                   ax=ax,
                   order=order,
                   inner='quartile',
                   linewidth=1.5)

axes[3,1].remove()
axes[3,2].remove()
plt.show()
```

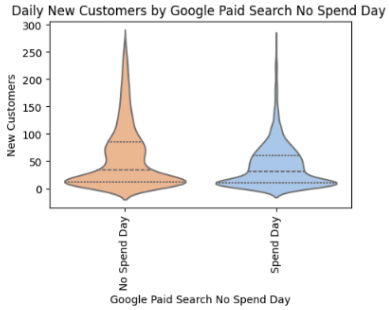


```
# Bivariate for nonbinary categorical variables against daily purchase purchases

nonbinary_vars = ['org_alias', 'month']

# Set figure options
fig, axes = plt.subplots(1, 2, figsize=(20, 5))
fig.subplots_adjust(hspace=.75, wspace=.25)

for var, ax in zip(nonbinary_vars, axes.flat):

    # if var is months make the order of the labels january = december
    if var == 'month':
        order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
                 'August', 'September', 'October', 'November', 'December']
    else:
        order = None

    x_title = var.replace('_', ' ').title()
    ax.set_title(f'Daily New Customers by {x_title} ')

    ax.set_ylabel("New Customers")
    ax.set_xlabel(x_title)

    ax.tick_params(axis='x', labelrotation=90)

    # Draw boxplot
    sns.violinplot(data=df,
                   x=var,
                   y='new_customers',
                   palette='pastel',
                   hue=var,
                   ax=ax,
                   order=order,
                   inner='quartile',
                   linewidth=1.5)

plt.show()
```

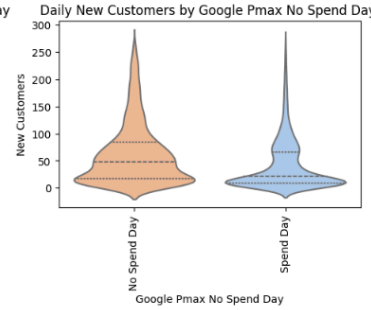
14B. Categorical Bivariate Graphs

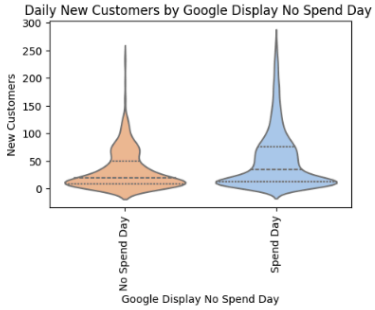

Daily New Customers by Is Weekend
New Customers
Weekday
Weekend
Is Weekend

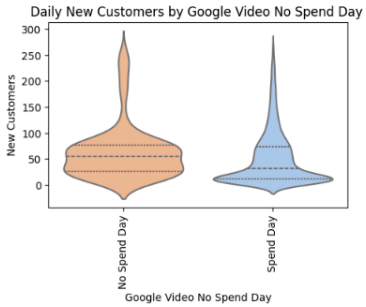

Daily New Customers by Is Public Holiday
New Customers
Non-Holiday
Holiday
Is Public Holiday

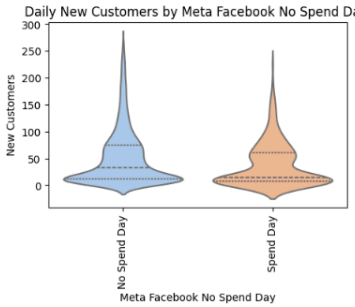

Daily New Customers by Google Paid Search No Spend Day
New Customers
No Spend Day
Spend Day
Google Paid Search No Spend Day

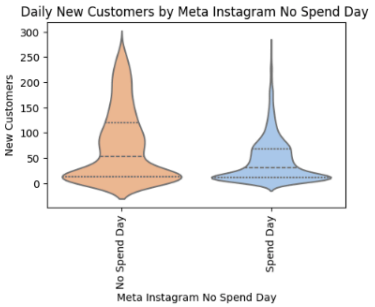

Daily New Customers by Google Shopping No Spend Day
New Customers
No Spend Day
Spend Day
Google Shopping No Spend Day

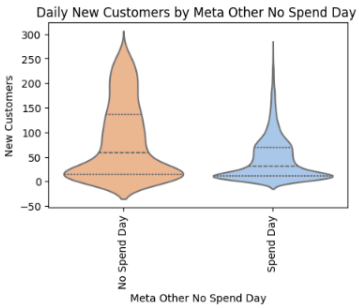

Daily New Customers by Google Pmax No Spend Day
New Customers
No Spend Day
Spend Day
Google Pmax No Spend Day

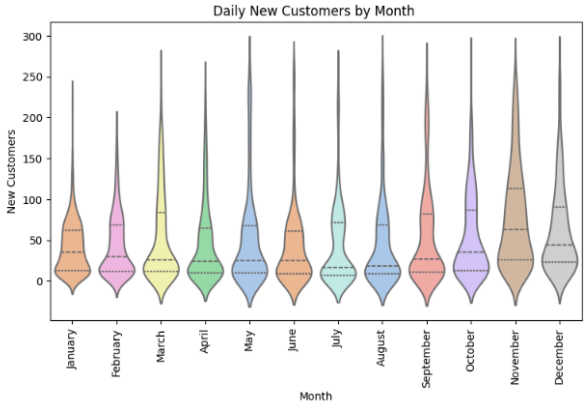

Daily New Customers by Google Display No Spend Day
New Customers
No Spend Day
Spend Day
Google Display No Spend Day


Daily New Customers by Google Video No Spend Day
New Customers
No Spend Day
Spend Day
Google Video No Spend Day


Daily New Customers by Meta Facebook No Spend Day
New Customers
No Spend Day
Spend Day
Meta Facebook No Spend Day


Daily New Customers by Meta Instagram No Spend Day
New Customers
No Spend Day
Spend Day
Meta Instagram No Spend Day


Daily New Customers by Meta Other No Spend Day
New Customers
No Spend Day
Spend Day
Meta Other No Spend Day



14C. Categorical Bivariate Descriptive Statistics Code and Outputs

```
# Descriptive Stats for each against new customers

# Loop through and get desc_stats
for var in cat_vars:
    desc_stats = (
        df.groupby(var)['new_customers']
      .describe()
      .reset_index()
    )

    # show desc_stats
    display(desc_stats)
```

	org_alias	count	mean	std	min	25%	50%	75%	max
1	A	1428	79.98109243697479	37.66663504458819	21	56	72	93	270
2	B	1576	10.589467005076141	8.079344038762578	1	6	9	12	113
3	C	998	111.09819639278557	67.83402347625	4	49	105	161	271
4	D	1021	94.06562193927522	47.13710294293667	9	62	80	118	270
5	E	459	36.57298474945534	52.04680260684451	0	10	17	36	264
6	F	1322	27.898638426626324	26.550487770847504	1	11	20	35.75	264
7	G	505	48.853465346534655	27.188026821593244	13	31	42	58	235
8	H	1064	14.8796992481203	14.24759561183901	1	8	12	17	239

	month	count	mean	std	min	25%	50%	75%	max
1	April	750	46.03066666666667	49.13563004319654	2	10	24	65	242
2	August	674	46.724035608308604	55.669085306617006	1	9	18	69	270
3	December	709	62.96332863187588	54.31945483052413	2	23	44	90	270
4	February	703	43.778093883357045	38.230508368980885	1	12	30	68.5	187
5	January	744	40.975806451612904	31.426419959784372	1	13	35	62	228
6	July	618	43.31391585760518	49.821059061106055	1	7	16	71.75	271
7	June	586	44.04266211604096	51.62600440358329	0	9	25	61	264
8	March	775	52.865806451612904	53.53696643467101	1	12	26	84	254
9	May	716	51.18156424581006	60.320709274178164	1	10	24.5	68	267
10	November	676	77.5828402366864	61.04978370388772	4	25.75	63	113	264
11	October	739	54.82679296346414	51.86316984966742	1	13	35	87	270
12	September	683	54.049780380673496	57.35842327528001	1	11	27	82	260

	is_public_holiday	count	mean	std	min	25%	50%	75%	max
1	0	8078	51.48229759841545	52.604246559024695	0	11	33	74	271
2	1	295	54.63389830508475	53.70068410036101	1	13	36	79	270

	is_weekend	count	mean	std	min	25%	50%	75%	max
1	0	5986	51.286835950551286	52.21815702978981	0	11	33	74	270
2	1	2387	52.36196062002514	53.69770118293763	1	12	33	75	271

	google_paid_search_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	5178	57.73271533410583	58.21822241610256	1	12	34	85	271
2	1	3195	41.643505477308295	40.12810947359018	0	10	31	61	270

	meta_other_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	1286	82.65940902021772	76.40217580554678	1	14	58.5	137	271
2	1	7087	45.95611683363906	44.819280469742786	0	11	31	69	270

	meta_instagram_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	1830	74.49672131147541	71.58013277094189	1	12	53	119.75	271
2	1	6543	45.18752865657955	43.88930397157026	0	11	31	67	270

	meta_facebook_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	8009	52.1975277812461	53.05378624232781	1	12	33	75	271
2	1	364	38.29945054945055	40.45125488456249	0	8	15	60.25	226

	google_video_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	355	63.04225352112676	53.30236652713622	6	25	54	76	264
2	1	8018	51.086430531304565	52.55956438995091	0	11	32	74	271

	google_display_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	508	33.38385826771653	34.78868719400089	1	9	19	49	239
2	1	7865	52.76948506039415	53.38069749695346	0	12	34	76	271

	google_pmax_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	3824	62.34178870292887	56.648477778057625	1	17	48	84	271
2	1	4549	42.55792481864146	47.17159768024344	0	9	21	66	270

	google_shopping_no_spend_day	count	mean	std	min	25%	50%	75%	max
1	0	5400	49.23777777777778	52.90765174819872	1	10	27	73	271
2	1	2973	55.87184661957619	51.89546568375379	0	14	43	78	267

The descriptive statistics (Figure 14C) indicate that is_weekend and is_holiday suggest a minor increase in new customers, but they acquire at least one new customer, whereas on weekdays and non-holidays, no new customers are acquired. Considerable variation exists across organizations and months, with Organization C as the top performer, generating an average of 111 new customers, while Organization B is the lowest, averaging about 11 new customers. Seasonal effects are evident, with November and December showing higher averages of ~78 and 63 new customers, respectively, suggesting that seasonality may influence model outcomes.

Across all media channels, the no-spend indicators reveal that days with active spend consistently generate at least one new customer, while inactive days consistently have zero (Figure 14C). Despite having sparse media days, Meta Other and Meta Instagram achieve the highest average new customers on active days (~83 and ~74, respectively). Google Video, which is also sparse and sporadic, drives the highest minimum number of new customers on active days (6) and ranks third in average new customers (~63). These patterns highlight both the potential predictive value of these sparse channels and the challenge they present for regression modeling due to the limited number of active observations. They will be included in the model, but their

coefficients may be unstable, have significant standard errors, or be deemed non-significant during variable selection.

2. Multi-Linear Regression Analysis

The multiple linear regression analysis was performed using predictor features comprised of log-transformed, adstock-integrated spend variables (e.g., Google, Meta platforms), seasonal indicators (months, weekend, public holidays), and organization ID dummy variables. The data were initially split into training and testing sets, as shown in Figure 15A, to evaluate the model's predictive ability against the established hypothesis thresholds.

To confirm the independence assumption among predictors, variance inflation factor (VIF) scores were calculated for both models. As shown in Figures 15A – 15B, all VIF scores were below 5, indicating low to moderate collinearity according to Bobbitt's (2022) statistical standards. Consequently, all selected features were kept for the first model run.

Figure 15

Train/Test Split and Variance Inflation Factor Check

15A. Multiple Linear Regression Train/Test Split and VIF Check

```python
X_vif_const = sm.add_constant(X_train)

# First VIF Check
vif_df = pd.DataFrame()
vif_df['feature'] = X_vif_const.columns
vif_df['VIF'] = [variance_inflation_factor(X_vif_const.values, i)
                 for i in range(X_vif_const.shape[1])]

print(vif_df.sort_values(by='VIF', ascending=False))
```

Output Terminal Debug console

	feature	VIF
0	const	40.985143
1	google_paid_search_spend_adstock_log	3.087276
3	google_pmax_spend_adstock_log	2.975164
21	Org_C_b1a6	2.701939
2	google_shopping_spend_adstock_log	2.677363
26	Org_H_4fce	2.632165
22	Org_D_3de0	2.618688
23	Org_E_1ea2	2.430526
7	meta_instagram_spend_adstock_log	2.419600
8	meta_other_spend_adstock_log	2.182393
20	Org_B_9c1f	2.138972
24	Org_F_ee4f	1.907291
6	meta_facebook_spend_adstock_log	1.876427
10	March	1.867802
11	April	1.850264
12	May	1.843068
17	October	1.833043
19	December	1.797846
15	August	1.792131
18	November	1.779707
9	February	1.772208
16	September	1.771241
14	July	1.734225
13	June	1.693633
25	Org_G_3136	1.686769
5	google_video_spend_adstock_log	1.214513
4	google_display_spend_adstock_log	1.173096
27	is_public_holiday	1.021897
28	is_weekend	1.003619

15B. Negative Binomial Regression Train/Test Split and VIF Check

```python
# Select for pre-model VIF Check
pred_vars = ['google_paid_search_spend_adstock','google_shopping_spend_adstock', 'google_pmax_spend_adstock',
       'google_display_spend_adstock', 'google_video_spend_adstock','meta_facebook_spend_adstock',
       'meta_instagram_spend_adstock','meta_other_spend_adstock','February','March', 'April', 'May', 'June',
       'July', 'August', 'September','October', 'November', 'December', 'Org_B_9c1f','Org_C_b1a6', 'Org_D_3de0', 'Org_E_1ea2',
       'Org_F_ee4f', 'Org_G_3136','Org_H_4fce', 'is_public_holiday', 'is_weekend']
```

```
✓ 11:32 AM (<1s)                                                    5
```

```python
# Set up train/test split
X = df[pred_vars]

y = df['new_customers']

# Give 20% to test; 80% to train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)
```

```python
X_vif_const = sm.add_constant(X_train)

# First VIF Check
vif_df = pd.DataFrame()
vif_df['feature'] = X_vif_const.columns
vif_df['VIF'] = [variance_inflation_factor(X_vif_const.values, i)
                 for i in range(X_vif_const.shape[1])]

print(vif_df.sort_values(by='VIF', ascending=False))
```

Output Terminal Debug console

	feature	VIF
0	const	17.714231
7	meta_instagram_spend_adstock	3.182257
6	meta_facebook_spend_adstock	2.753164
3	google_pmax_spend_adstock	2.405552
21	Org_C_b1a6	2.242730
1	google_paid_search_spend_adstock	2.152499
25	Org_G_3136	2.085065
23	Org_E_1ea2	2.076612
22	Org_D_3de0	2.061102
8	meta_other_spend_adstock	1.887238
10	March	1.886709
20	Org_B_9c1f	1.874722
26	Org_H_4fce	1.867999
11	April	1.861738
19	December	1.844800
17	October	1.827404
12	May	1.826809
9	February	1.782731
16	September	1.779402
18	November	1.775170
15	August	1.751657
24	Org_F_ee4f	1.713503
14	July	1.685753
13	June	1.658427
2	google_shopping_spend_adstock	1.606677
4	google_display_spend_adstock	1.068833
5	google_video_spend_adstock	1.053775
27	is_public_holiday	1.024517
28	is_weekend	1.004507

The first Ordinary Least Squares (OLS) Multilinear Regression in Figure 16A achieved a strong adjusted R^2 of 0.779, meaning that the model can explain approximately 78% of the variance in the data. The F-statistic value of 845, with a p-value of 0.00, demonstrates that the combination of features yields a statistically significant model.

A model reduction method using backward stepwise elimination, as shown in Figure 16B, was employed to remove individual features that were not statistically significant, with p-values greater than 0.05 (Larose & Larose, 2019). The final model has the same adjusted R^2 of 0.779 (Figure 16C), but with a much higher F-statistic (1314), as expected, due to retaining only statistically relevant features.

Figure 16

Multiple Linear Regression Code and Outputs

16A. First Linear Regression Model

```
#First Model

# Add constant for intercept
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Fit OLS regression
model = sm.OLS(y_train, X_train_const).fit()

# Summary of coefficients
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      new_customers_log   R-squared:                       0.780
Model:                            OLS   Adj. R-squared:                  0.779
Method:                 Least Squares   F-statistic:                     845.2
Date:                Fri, 21 Nov 2025   Prob (F-statistic):               0.00
Time:                        21:38:53   Log-Likelihood:                -4945.2
No. Observations:                6698   AIC:                             9948.
Df Residuals:                    6669   BIC:                         1.015e+04
Df Model:                          28
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                     3.0321      0.040     76.027      0.000       2.954       3.110
google_paid_search_spend_adstock_log     0.0387      0.004      8.703      0.000       0.030       0.047
google_shopping_spend_adstock_log     0.0797      0.004     18.616      0.000       0.071       0.088
google_pmax_spend_adstock_log     0.0634      0.004     17.342      0.000       0.056       0.071
google_display_spend_adstock_log     0.0057      0.012      0.490      0.624      -0.017       0.028
google_video_spend_adstock_log     0.0016      0.011      0.144      0.885      -0.020       0.023
meta_facebook_spend_adstock_log     0.1354      0.005     26.669      0.000       0.125       0.145
meta_instagram_spend_adstock_log     0.0640      0.004     16.635      0.000       0.056       0.072
meta_other_spend_adstock_log     0.0835      0.020      4.119      0.000       0.044       0.123
February                  0.0493      0.030      1.648      0.099      -0.009       0.108
March                     0.1195      0.029      4.097      0.000       0.062       0.177
April                     0.0286      0.029      0.974      0.330      -0.029       0.086
May                       0.1241      0.030      4.175      0.000       0.066       0.182
June                      0.0248      0.032      0.782      0.434      -0.037       0.087
July                     -0.0090      0.031     -0.289      0.773      -0.070       0.052
August                   -0.0035      0.031     -0.114      0.909      -0.064       0.057
September                 0.1242      0.030      4.132      0.000       0.065       0.183
October                   0.1384      0.030      4.671      0.000       0.080       0.197
November                  0.5077      0.030     16.654      0.000       0.448       0.568
December                  0.3086      0.030     10.387      0.000       0.250       0.367
Org_B_9c1f               -2.1245      0.023    -91.938      0.000      -2.170      -2.079
Org_C_b1a6               -0.5508      0.032    -17.308      0.000      -0.613      -0.488
Org_D_3de0               -0.0429      0.031     -1.403      0.161      -0.103       0.017
Org_E_1ea2               -1.8041      0.043    -42.058      0.000      -1.888      -1.720
Org_F_ee4f               -1.1988      0.023    -51.058      0.000      -1.245      -1.153
Org_G_3136               -0.8685      0.034    -25.529      0.000      -0.935      -0.802
Org_H_4fce               -2.1474      0.030    -71.317      0.000      -2.206      -2.088
is_public_holiday         0.0278      0.034      0.817      0.414      -0.039       0.095
is_weekend                0.0099      0.014      0.716      0.474      -0.017       0.037
==============================================================================
Omnibus:                      176.202   Durbin-Watson:                   2.028
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              431.966
Skew:                          -0.019   Prob(JB):                     1.58e-94
Kurtosis:                       4.244   Cond. No.                         98.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

16B. Multiple Linear Regression Model Reduction Code and Output

```python
# Model Reduction
alpha = 0.05 # p-value threshold

# Start with all features except const
pvalues = pd.DataFrame(model.pvalues).reset_index()
pvalues.columns = ['feature', 'pvalue']
pvalues = pvalues[pvalues['feature'] != 'const']

# Initialize updated list
updated_list = pvalues['feature'].tolist()

while True:
    # Fit model with current features
    X = X_train_const[updated_list]
    X = sm.add_constant(X)
    model = sm.OLS(y_train, X).fit()

    # Get p-values excluding const
    pvals = model.pvalues.drop('const')

    # Find max p-value
    max_pval = pvals.max()
    max_feature = pvals[pvals == max_pval].index[0]

    if max_pval > alpha:
        print(f"Dropping {max_feature} with p-value {max_pval:.4f}")
        updated_list.remove(max_feature)
    else:
        print("All remaining features are significant.")
        break

print("Final feature list:", updated_list)
```

```
Final feature list: ['google_paid_search_spend_adstock_log', 'google_shopping_spend_adstock_log', 'google_pmax_spend_adstock_log', 'meta_facebook_spend_adstock_log', 'meta_instagram_spend_adstock_log', 'meta_other_spend_adstock_log', 'March', 'May', 'September', 'October', 'November', 'December', 'Org_B_9c1f', 'Org_C_b1a6', 'Org_E_1ea2', 'Org_F_ee4f', 'Org_G_3136', 'Org_H_4fce']
```

16C. Final Multiple Linear Regression Model Code and Summary

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      new_customers_log   R-squared:                       0.780
Model:                            OLS   Adj. R-squared:                  0.779
Method:                 Least Squares   F-statistic:                     1314.
Date:                Fri, 21 Nov 2025   Prob (F-statistic):               0.00
Time:                        21:41:12   Log-Likelihood:                -4949.7
No. Observations:                6698   AIC:                             9937.
Df Residuals:                    6679   BIC:                         1.007e+04
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                          3.0347      0.032     93.456      0.000       2.971       3.098
google_paid_search_spend_adstock_log     0.0354      0.004      9.854      0.000       0.028       0.042
google_shopping_spend_adstock_log        0.0799      0.004     18.812      0.000       0.072       0.088
google_pmax_spend_adstock_log            0.0634      0.004     17.494      0.000       0.056       0.071
meta_facebook_spend_adstock_log          0.1371      0.005     27.430      0.000       0.127       0.147
meta_instagram_spend_adstock_log         0.0647      0.004     17.492      0.000       0.057       0.072
meta_other_spend_adstock_log             0.0871      0.020      4.435      0.000       0.049       0.126
March                                    0.1031      0.022      4.649      0.000       0.060       0.147
May                                      0.1087      0.023      4.761      0.000       0.064       0.154
September                                0.1088      0.023      4.647      0.000       0.063       0.155
October                                  0.1239      0.023      5.388      0.000       0.079       0.169
November                                 0.4924      0.024     20.276      0.000       0.445       0.540
December                                 0.2939      0.023     12.666      0.000       0.248       0.339
Org_B_9c1f                              -2.1138      0.022    -96.411      0.000      -2.157      -2.071
Org_C_b1a6                              -0.5310      0.028    -19.117      0.000      -0.586      -0.477
Org_E_1ea2                              -1.7976      0.042    -42.462      0.000      -1.881      -1.715
Org_F_ee4f                              -1.1800      0.020    -59.272      0.000      -1.219      -1.141
Org_G_3136                              -0.8636      0.034    -25.659      0.000      -0.930      -0.798
Org_H_4fce                              -2.1271      0.024    -88.471      0.000      -2.174      -2.080
==============================================================================
Omnibus:                      176.915   Durbin-Watson:                   2.027
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              432.837
Skew:                          -0.031   Prob(JB):                     1.02e-94
Kurtosis:                       4.244   Cond. No.                         61.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

16D. Multiple Linear Regression Actual vs. Predicted Values Check

```
# Test Model
X_test_reduced = X_test[final_features].copy()
X_test_reduced = sm.add_constant(X_test_reduced)

# Create predictions; generates logged predictions
y_test_pred_log = final_model.predict(X_test_reduced)

# Convert to raw value estimates of new customers
y_test_pred_raw = np.round(np.expm1(y_test_pred_log)).astype(int)
y_test_raw = np.expm1(y_test)

# Set to series
y_test_pred_raw = pd.Series(y_test_pred_raw.round(0).astype(int)
, index=X_test_reduced.index, name="Predicted")
y_test_raw = pd.Series(y_test_raw.round(0).astype(int), index=X_test_reduced.index, name="Actual")

# Create comparison dataframe between predictions vs. actual
comparison_df = pd.concat([y_test_raw, y_test_pred_raw], axis=1)
print(comparison_df.sample(10, random_state=25))

# Confirm against predictive hypothesis
within_20pct = (y_test_pred_raw >= 0.8 * y_test_raw) & (y_test_pred_raw <= 1.2 * y_test_raw)
accuracy_20pct = within_20pct.mean()
print(f"\nProportion of predictions within ±20% of actual counts: {accuracy_20pct:.2%}")
```

```
▸ comparison_df:  pandas.core.frame.DataFrame = [Actual: int64, Predicted: int64]
▸ X_test_reduced:  pandas.core.frame.DataFrame = [const: float64, google_paid_search_spend_adstock_log: float64 ... 17 more fields]
      Actual  Predicted
5747      15         15
1734     100        127
8103       7          8
6089      21         28
6838     108        130
4889      29         34
4425       2          7
1655     243        132
8235       6          7
7187     119        109

Proportion of predictions within ±20% of actual counts: 36.24%
```

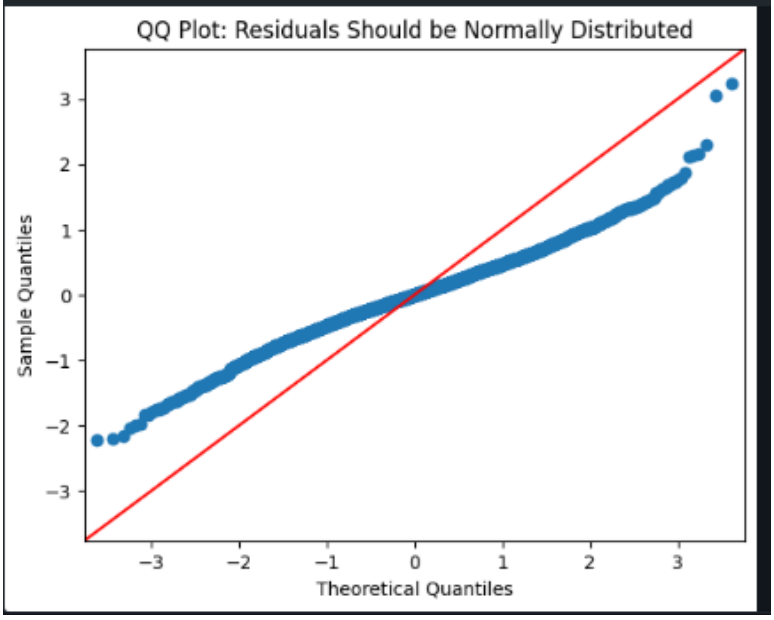
16E. Multiple Linear Regression Calculation for RMSE and MAE

```
# Actual and Predicted
y_true = comparison_df['Actual']
y_pred = comparison_df['Predicted']

# RMSE
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)

# MAE
mae = mean_absolute_error(y_true, y_pred)
mae = np.round(mae)

print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
```

```
RMSE: 30.73
MAE: 18.00
```

16F. Multiple Linear Regression QQ Plot (Should Follow Line)

```
# Plot Residuals
residuals = final_model.resid

# Check Normality with QQ Plot
sm.qqplot(residuals, line='45')
plt.title('QQ Plot: Residuals Should be Normally Distributed')
plt.show()
```



The trained model was applied to the test set, and the resulting predicted customer values were converted back to their raw format for comparison against the actual new customer values (Figure 16D). The model was able to predict within ±20% of the actual value for only 36.24% of the test data points as opposed to the hypothesis's threshold of 70%. Figure 16E shows that the RMSE of 30 new customers is significantly higher than the MAE (18), indicating the error is influenced by a few large prediction outliers. Given that the average new customer count is only 54, this high RMSE suggests the model struggles with predicting the sporadic high-customer days.

Figure 16F shows the residual QQ plot. The slight S-shaped deviation at the tails indicates deviation from normal distribution, suggesting heavier tails in the residual distribution. This pattern implies mild skewness and implies a violation of the normality assumption for the multiple linear regression model.

Based on the issues noted above in addition to overdispersion in new customer counts, a Negative Binomial model was subsequently evaluated to determine if it could provide a more accurate prediction, since they are better suited for count data with overdispersion.

3. Negative Binomial Regression Analysis

The negative binomial model was fitted with the same predictors, but using the raw new customer count, as it can accommodate overdispersion and a skewed distribution (Rahul, 2025). The initial model in Figure 17A yielded a pseudo-R^2 value of 0.5548, which exceeds the hypothesis threshold of 0.30. This means that the model's explanatory power is 55.48% better than a null model without predictors. Its reduced model (Figure 17B), with only statistically relevant features, brought the pseudo-R^2 slightly to 55.45%, as shown in the final model in Figure 17C. The predictive test yielded a 34.59% prediction coverage within the ±20% margin

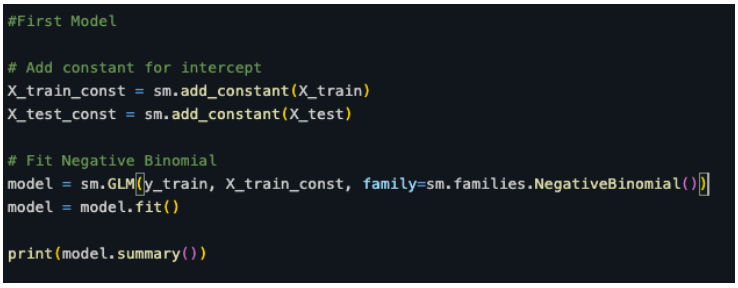
(Figure 17D). The high RMSE of ~31 with a lower MAE of 19 mirrors similar results as the linear regression model, indicating that this negative binomial model was also severely affected by the extremely high customer days (Figure 17E).

Deviance residuals are the difference between observed and predicted values, scaled according to the model's assumed variance. When plotted against predicted values, they should appear randomly scattered, indicating constant variance (homoscedasticity) and a good model fit. However, Figure 17F shows a distinct cone-shaped pattern, widest at low predicted values, suggesting that the model fits poorly, particularly for observations with lower counts.

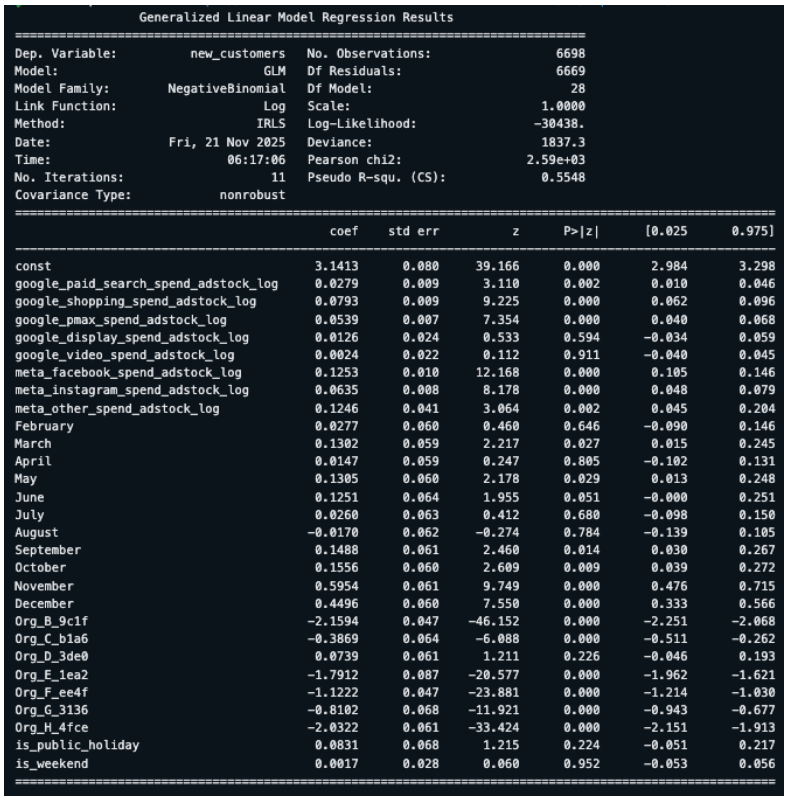
Figure 17

Negative Binomial Regression Code and Outputs

17A. First Negative Binomial Regression Model



```
#First Model

# Add constant for intercept
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Fit Negative Binomial
model = sm.GLM(y_train, X_train_const, family=sm.families.NegativeBinomial())
model = model.fit()

print(model.summary())
```



Generalized Linear Model Regression Results

Dep. Variable:	new_customers	No. Observations:	6698
Model:	GLM	Df Residuals:	6669
Model Family:	NegativeBinomial	Df Model:	28
Link Function:	Log	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-30438.
Date:	Fri, 21 Nov 2025	Deviance:	1837.3
Time:	06:17:06	Pearson chi2:	2.59e+03
No. Iterations:	11	Pseudo R-squ. (CS):	0.5548
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	3.1413	0.080	39.166	0.000	2.984	3.298
google_paid_search_spend_adstock_log	0.0279	0.009	3.110	0.002	0.010	0.046
google_shopping_spend_adstock_log	0.0793	0.009	9.225	0.000	0.062	0.096
google_pmax_spend_adstock_log	0.0539	0.007	7.354	0.000	0.040	0.068
google_display_spend_adstock_log	0.0126	0.024	0.533	0.594	-0.034	0.059
google_video_spend_adstock_log	0.0024	0.022	0.112	0.911	-0.040	0.045
meta_facebook_spend_adstock_log	0.1253	0.010	12.168	0.000	0.105	0.146
meta_instagram_spend_adstock_log	0.0635	0.008	8.178	0.000	0.048	0.079
meta_other_spend_adstock_log	0.1246	0.041	3.064	0.002	0.045	0.204
February	0.0277	0.060	0.460	0.646	-0.090	0.146
March	0.1302	0.059	2.217	0.027	0.015	0.245
April	0.0147	0.059	0.247	0.805	-0.102	0.131
May	0.1305	0.060	2.178	0.029	0.013	0.248
June	0.1251	0.064	1.955	0.051	-0.000	0.251
July	0.0260	0.063	0.412	0.680	-0.098	0.150
August	-0.0170	0.062	-0.274	0.784	-0.139	0.105
September	0.1488	0.061	2.460	0.014	0.030	0.267
October	0.1556	0.060	2.609	0.009	0.039	0.272
November	0.5954	0.061	9.749	0.000	0.476	0.715
December	0.4496	0.060	7.550	0.000	0.333	0.566
Org_B_9c1f	-2.1594	0.047	-46.152	0.000	-2.251	-2.068
Org_C_b1a6	-0.3869	0.064	-6.088	0.000	-0.511	-0.262
Org_D_3de0	0.0739	0.061	1.211	0.226	-0.046	0.193
Org_E_1ea2	-1.7912	0.087	-20.577	0.000	-1.962	-1.621
Org_F_ee4f	-1.1222	0.047	-23.881	0.000	-1.214	-1.030
Org_G_3136	-0.8102	0.068	-11.921	0.000	-0.943	-0.677
Org_H_4fce	-2.0322	0.061	-33.424	0.000	-2.151	-1.913
is_public_holiday	0.0831	0.068	1.215	0.224	-0.051	0.217
is_weekend	0.0017	0.028	0.060	0.952	-0.053	0.056

17B. Negative Binomial Regression Model Reduction Code and Output

```python
# Model Reduction
alpha = 0.05 # p-value threshold

# Start with all features except const
pvalues = pd.DataFrame(model.pvalues).reset_index()
pvalues.columns = ['feature', 'pvalue']
pvalues = pvalues[pvalues['feature'] != 'const']

# Initialize updated list
updated_list = pvalues['feature'].tolist()

while True:
    # Fit model with current features
    X = X_train_const[updated_list]
    X = sm.add_constant(X)
    model = sm.GLM(y_train, X, family=sm.families.NegativeBinomial()).fit()

    # Get p-values excluding const
    pvals = model.pvalues.drop('const')

    # Find max p-value
    max_pval = pvals.max()
    max_feature = pvals[pvals == max_pval].index[0]

    if max_pval > alpha:
        print(f"Dropping {max_feature} with p-value {max_pval:.4f}")
        updated_list.remove(max_feature)
    else:
        print("All remaining features are significant.")
        break

print("Final feature list:", updated_list)
```

```
Final feature list: ['google_paid_search_spend_adstock_log', 'google_shopping_spend_adstock_log', 'google_pmax_spend_adstock_log', 'meta_facebook_spend_adstock_log', 'meta_instagram_spend_adstock_log', 'meta_other_spend_adstock_log', 'March', 'May', 'June', 'September', 'October', 'November', 'December', 'Org_B_9c1f', 'Org_C_b1a6', 'Org_E_1ea2', 'Org_F_ee4f', 'Org_G_3136', 'Org_H_4fce']
```

17C. Final Negative Binomial Regression Model Code and Summary

```
# Generate final model
final_features = updated_list

X_train_reduced = X_train[final_features].copy()
X_train_reduced = sm.add_constant(X_train_reduced)

final_model = sm.GLM(y_train, X_train_reduced, family=sm.families.NegativeBinomial()).fit()
print(final_model.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:          new_customers   No. Observations:                 6698
Model:                            GLM   Df Residuals:                     6678
Model Family:        NegativeBinomial   Df Model:                           19
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -30440.
Date:                Fri, 21 Nov 2025   Deviance:                       1841.4
Time:                        06:26:35   Pearson chi2:                 2.61e+03
No. Iterations:                    11   Pseudo R-squ. (CS):             0.5545
Covariance Type:            nonrobust
=============================================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------
const                         3.1783      0.066     48.020      0.000       3.049       3.308
google_paid_search_spend_adstock_log     0.0342      0.007      4.747      0.000       0.020       0.048
google_shopping_spend_adstock_log     0.0795      0.009      9.311      0.000       0.063       0.096
google_pmax_spend_adstock_log     0.0541      0.007      7.448      0.000       0.040       0.068
meta_facebook_spend_adstock_log     0.1232      0.010     12.160      0.000       0.103       0.143
meta_instagram_spend_adstock_log     0.0611      0.007      8.182      0.000       0.046       0.076
meta_other_spend_adstock_log     0.1284      0.039      3.255      0.001       0.051       0.206
March                         0.1210      0.045      2.671      0.008       0.032       0.210
May                           0.1198      0.047      2.565      0.010       0.028       0.211
June                          0.1149      0.052      2.215      0.027       0.013       0.217
September                     0.1397      0.048      2.927      0.003       0.046       0.233
October                       0.1478      0.047      3.154      0.002       0.056       0.240
November                      0.5891      0.049     12.026      0.000       0.493       0.685
December                      0.4385      0.047      9.346      0.000       0.347       0.530
Org_B_9c1f                   -2.1675      0.045    -48.681      0.000      -2.255      -2.080
Org_C_b1a6                   -0.4204      0.055     -7.578      0.000      -0.529      -0.312
Org_E_1ea2                   -1.8095      0.086    -21.051      0.000      -1.978      -1.641
Org_F_ee4f                   -1.1511      0.040    -28.853      0.000      -1.229      -1.073
Org_G_3136                   -0.8186      0.067    -12.167      0.000      -0.950      -0.687
Org_H_4fce                   -2.0753      0.049    -42.655      0.000      -2.171      -1.980
=============================================================================================
```

17D. Negative Binomial Regression Actual vs. Predicted Values Check

```
X_test_reduced = X_test[final_features].copy()
X_test_reduced = sm.add_constant(X_test_reduced)

y_test_pred = final_model.predict(X_test_reduced)
y_test_pred = pd.Series(y_test_pred.round(0).astype(int),
                        index=X_test_reduced.index,
                        name="Predicted")

y_test_actual = pd.Series(y_test.round(0).astype(int),
                          index=X_test_reduced.index,
                          name="Actual")


comparison_df = pd.concat([y_test_actual, y_test_pred], axis=1)
print("Sample of predictions vs actual:")
print(comparison_df.sample(10, random_state=25))

# Handle zeros safely: only consider rows where actual > 0
within_20pct = (y_test_pred[mask] >= 0.8 * y_test_actual[mask]) & \
               (y_test_pred[mask] <= 1.2 * y_test_actual[mask])
accuracy_20pct = within_20pct.mean()
print(f"\nProportion of predictions within ±20% of actual counts: {accuracy_20pct:.2%}")
```

```
Sample of predictions vs actual:
      Actual  Predicted
5747      15         19
1734     100        154
8103       7          9
6089      21         33
6838     108        136
4889      29         41
4425       2          9
1655     243        145
8235       6          8
7187     119        127

Proportion of predictions within ±20% of actual counts: 34.59%
```

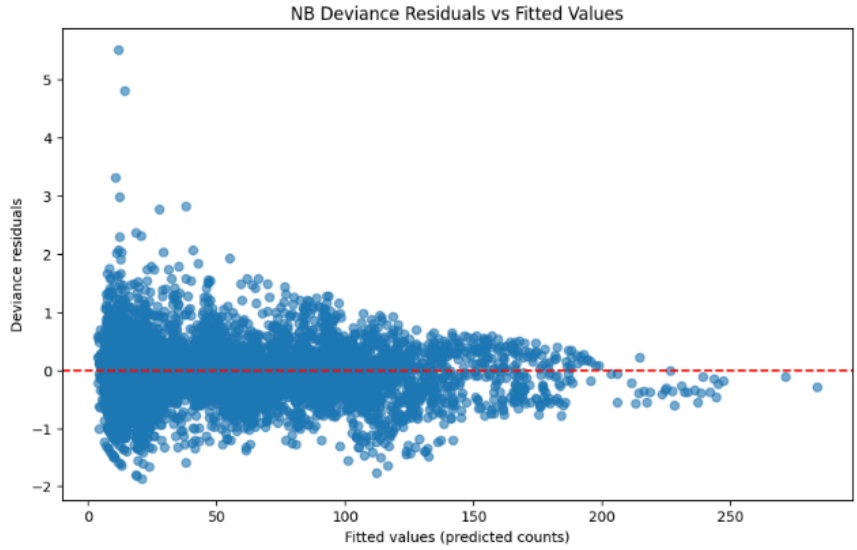
17E. Negative Binomial Regression Calculation for RMSE and MAE

```
    # Actual and Predicted
    y_true = comparison_df['Actual']
    y_pred = comparison_df['Predicted']

    # RMSE
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)

    # MAE
    mae = mean_absolute_error(y_true, y_pred)
    mae = np.round(mae)


    print(f"RMSE: {rmse:.2f}")
    print(f"MAE: {mae:.2f}")
```

```
RMSE: 30.61
MAE: 19.00
```

17F. Negative Binomial Deviance Residual Plot (Should Be Random)

```
#Get predicted mean
fitted_mean= final_model.fittedvalues
deviance_resid = final_model.resid_deviance

plt.figure(figsize=(10, 6))
plt.scatter(fitted_mean, deviance_resid, alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted values (predicted counts)')
plt.ylabel('Deviance residuals')
plt.title('NB Deviance Residuals vs Fitted Values')
plt.show()
```



4. Model Comparison

The OLS model demonstrated a stronger overall explanatory power with R^2 of .779%, which can effectively explain 78% of variance in the data. When compared to a null model, the Negative Binomial's pseudo-R^2 of 0.55 is about 55% better. OLS also had a higher predictive accuracy at 36.42% predictive coverage within ±20% of actual values versus the Negative Binomial's predictive accuracy of 34.59%. In terms of RMSE (30.6 vs. 30.7) and MAE (19.0 vs. 18), OLS is only slightly better than Negative Binomial. Diagnostic checks also indicate that OLS provides a more stable fit with a moderate deviation in its QQ-plot than the pronounced cone-shape of the Negative Binomial's deviance residuals plot. Overall, the OLS model offers both superior explanatory strength and more reliable performance.

D2. Data Analysis Techniques and Justifications

The exploratory data analysis (EDA) focused on evaluating target variables against model assumptions and identifying patterns in media planning. The techniques applied, which include descriptive statistics, outlier removal, log transformations, and plotting bivariate/univariate relationships, served as essential pre-model processing steps to clarify the data's shape, spread, and distribution (Larose & Larose, 2019). The main limitation of EDA is its complete reliance on the available dataset. It inherently excludes the influence of unobserved confounding variables, such as weather, offline marketing activities, or other external factors that run concurrently with the campaigns. Consequently, the derived insights are limited to the present data and may be influenced by these unaccounted-for, unknown factors.

The analytical process for both the Linear and Negative Binomial models followed a structured approach: Models were first built using the full feature set. VIF analysis was used to

check and moderate multicollinearity. Initial performance was assessed using both target hypothesis metrics and statistical significance metrics. The models were then refined by reducing the features to only those that were statistically significant, to boost predictive performance. Finally, residual plots and auxiliary metrics were used to thoroughly test model assumptions and quantify predictive power. Both models belong to the family of Generalized Linear Models (GLMs) and failed to show a substantial difference in predictive capability. This outcome suggests that the uneven characteristics of the data, specifically the high spikes of new customers on certain days, overrode the models' differing assumptions, and limited the models' abilities to completely reject the null hypothesis.

Section E: Outcomes

E1. Outcomes, Implications, and Limitations Discussion

The analysis was designed to test a composite hypothesis using two models: Multiple Linear Regression and Negative Binomial Regression. The alternate hypothesis had two primary components: statistical significance and predictive accuracy. The hypothesis and alternate hypothesis are as follows:

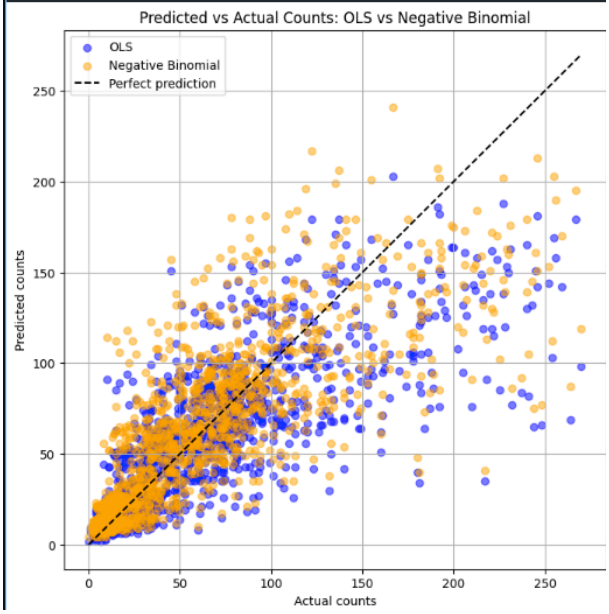
> H_0: "There is no significant relationship between channel media spend, calculated metrics, seasonality, and first purchases, and neither models can effectively predict first customer acquisitions."
>
> H_1: "At least one model includes statistically significant predictors ($p < 0.05$) and demonstrates sufficient explanatory power, defined as an Adjusted $R^2 \geq 0.60$ for Multiple Linear Regression or a Pseudo-$R^2 \geq 0.10$ for Negative Binomial Regression, with 70% of predictions falling within ±20% of actual values."

While the models identified statistically significant predictors and demonstrated strong explanatory power, they did not achieve the accuracy threshold of 70% prediction coverage within ±20% of actual values. The null hypothesis is partially rejected because both models show significant predictors and sufficient explanatory power, but the alternative hypothesis is not fully satisfied since neither model meets the ±20% prediction accuracy requirement (only 34–35% vs the 70% threshold). The plot in Figure 18 depicts how the models are unable to accurately predict the sudden high customer count days with extremely erroneous high customer count predictions.

Figure 18

Predictions vs. Actual Scatterplot for Both Models

```
# Pred. vs. Actual Comparison Plot
plt.figure(figsize=(8,8))

# Scatter for OLS
plt.scatter(df_ols['actual'], df_ols['ols_pred'], c
            olor='blue', alpha=0.5, label='OLS')

# Scatter for NB
plt.scatter(df_nb['actual'], df_nb['nb_pred'],
            color='orange', alpha=0.5, label='Negative Binomial')

# Identity line (perfect prediction)
max_val = max(df_ols['actual'].max(), df_nb['actual'].max())
plt.plot([0, max_val], [0, max_val], 'k--', label='Perfect prediction')

plt.xlabel('Actual counts')
plt.ylabel('Predicted counts')
plt.title('Predicted vs Actual Counts: OLS vs Negative Binomial')
plt.legend()
plt.grid(True)
plt.show()
```



The main implication is that the models offer high descriptive value, but they lack the necessary reliability to support an optimizing predictive analytical approach to media budget allocation. The models' development, evaluations, and predictive abilities were severely limited due to the inherent challenges in the dataset, which included skewed distributions, extreme descriptive statistics, and spikes in new customer counts.

The dataset includes many observations with zero new customers, which can be challenging for predictive models. Both the OLS and the Negative Binomial model were able to handle these zeros adequately, producing predictions that are close to actual zero counts. This suggests that the model captures the baseline behavior of inactive days.

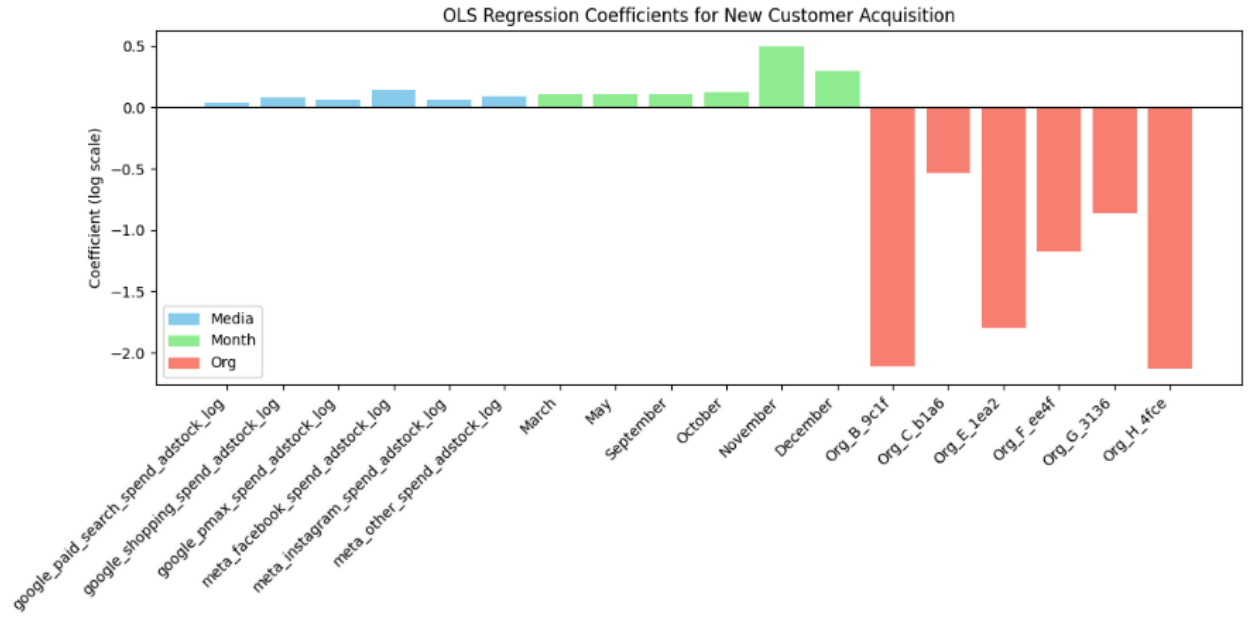
E2. Actionable Insights

The statistical results of the models can still provide useful insights to media planners and marketers regarding planning priorities around seasonality and media channels. The descriptive statistics, especially the bivariate information generated during the exploratory analysis stage, provide relevant insights supported by the statistical significance of the models.

The OLS Linear Regression Model provided a stronger statistically significant explanatory power and model fit compared to the Negative Binomial model. For business insights and presentation, this model can provide statistical backing to media budget recommendations. Based on the coefficients of the log-transformed variables, the chart in Figure 19 identifies significant drivers of first customer acquisitions, providing actionable insights on media channel allocation and seasonal effects. This image would be used in a visualization environment such as Tableau for business reporting.

Figure 19

Predictor Variable Effects on First Customer Acquisitions



Based on the chart, some key actionable items include:

- **Invest in high-impact media channels:** Meta Facebook and Google Shopping are the two strongest channels that show the strongest positive influence. Prioritizing budget allocation to these channels could maximize first acquisition growth.
- **Leverage seasonality:** October, November, and December show the highest peaks for new customers. Planning promotions and sales based on first purchase incentives could drive additional acquisition potential.
- **Plan around organizational strengths:** Some organizations perform closer to the baseline (smaller negative coefficients). Focus high-effort campaigns or resource-intensive strategies in these stronger-performing organizations to maximize ROI while addressing weaker ones with lower-cost solutions.

Data enrichment strategies should be implemented to develop and refine more robust models, potentially enhancing current models for improved predictions. Additional unknown data mentioned earlier in the paper, such as creative messaging, offline media channel spend,

competitive data, and even weather, could enrich the current data. Other organizations could also be included in the study if they provide additional active media days.

E3. Future Study Proposals

The bi-modal distribution that was observed from exploratory analysis suggests the presence of two subsets within the data. Future studies could apply mixture models that address these two groups to explicitly model the two peaks. By fitting separate variables for each group, the mixture models could support more accurate predictions that can capture both low and high acquisition segments. Based on the elevated prediction errors on higher customer counts, using mixture models could improve model predictive performance.

A long-term study should be dedicated to integrating the external and confounding variables identified as limitations. Accounting for these factors is necessary to move the analysis beyond descriptive correlation toward causal inference, providing the confidence required to accurately quantify the return on investment for marketing budget allocation. This approach would allow decision-makers to implement strategies with a clearer understanding of the drivers behind first-purchase behavior and more reliably optimize media spend across channels and organizations**.**

Section F: References

1. *Apache Spark overview | Databricks on AWS*. (2025, October 10). https://docs.databricks.com/aws/en/spark/
2. *Big Data Marketing: What it is and why it matters*. (n.d.). SAS. Retrieved October 5, 2025, from https://www.sas.com/en_us/insights/big-data/big-data-marketing.html

3. Bobbitt, Z. (2022, October 12). *How to calculate VIF in Python*. Statology. https://www.statology.org/how-to-calculate-vif-in-python/
4. *Databricks notebooks | Databricks on AWS*. (2025, September 3). Retrieved October 10, 2025, from https://docs.databricks.com/aws/en/notebooks
5. Fatima, N. (n.d.). *Understanding NP.log and Np.log1P in NUMPy*. Medium. Retrieved October 10, 2025, from https://medium.com/@noorfatimaafzalbutt/understanding-np-log-and-np-log1p-in-numpy-99cefa89cd30
6. figshare. (2024, June 3). *Multi-Region Marketing Mix Modelling (MMM) dataset for several eCommerce brands*. Figshare. Retrieved October 5, 2025, from https://figshare.com/articles/dataset/Multi-Region_Marketing_Mix_Modeling_MMM_Dataset_for_Several_eCommerce_Brands/25314841
7. Kantar Media. (2013). 10 Best Practices for Digital Media Planning. In *SRDS* (pp. 3–10) [Book]. Retrieved October 5, 2025, from https://kantarmedia.srds.com/common/downloads/10-Best-Practices-for-Digital-Media-Planning-full.pdf
8. Kosourova, E. (2025, August 18). *Measures of Central Tendency: A Comprehensive Overview*. Datacamp. Retrieved November 15, 2025, from https://www.datacamp.com/tutorial/central-tendency
9. Larose, C. D., & Larose, D. T. (2019). *Data science using Python and R*. https://doi.org/10.1002/9781119526865
10. Little, R., & Rubin, D. (2019). Statistical Analysis with Missing Data, Third Edition. In *Wiley series in probability and statistics*. https://doi.org/10.1002/9781119482260

11. Molnar, A. (2025, September 22). *2025 Facebook Ads Benchmarks*. TheeDigital. Retrieved November 8, 2025, from https://www.theedigital.com/blog/facebook-ads-benchmarks
12. Moons, D. (2025, June 13). *27 Google Ads Benchmarks (2025)*. Store Growers. Retrieved November 8, 2025, from https://www.storegrowers.com/google-ads-benchmarks/
13. Mssaperla. (n.d.). *What is the medallion lakehouse architecture? - Azure Databricks*. Microsoft Learn. Retrieved October 3, 2025, from https://learn.microsoft.com/en-us/azure/databricks/lakehouse/medallion
14. Normesta. (n.d.). *Use Azure Storage Explorer with Azure Data Lake Storage - Azure Storage*. Microsoft Learn. Retrieved October 7, 2025, from https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-explorer
15. O'Brien, C. M. (2011). Negative Binomial Regression, Second edition by Joseph M. Hilbe. *International Statistical Review*, *79*(3), 483–484. https://doi.org/10.1111/j.1751-5823.2011.00159_4.x
16. Rahul, K. (2025, January 7). *Negative Binomial Regression*. Wall Street Mojo. Retrieved October 7, 2025, from https://www.wallstreetmojo.com/negative-binomial-regression/
17. Rubino, N. (2025, August 25). Advertising AdStock: A practical guide with Excel template & Python code. *Nico Rubino*. Retrieved November 10, 2025, from https://nicorubino.com/2025/07/20/advertising-adstock/
18. Ruehl, E. (2024, November 14). Floating Holiday Meaning + Example Policy. *SHRM*. https://www.shrm.org/topics-tools/news/benefits-compensation/floating-holiday-meaning-example-policy/

19. Willems, K. (2017, March 15). *Python Exploratory Data Analysis Tutorial*. Datacamp. Retrieved November 4, 2025, from https://www.datacamp.com/tutorial/exploratory-data-analysis-python

20. *Worldwide public Holidays - Nager.Date*. (n.d.). Retrieved October 4, 2025, from https://date.nager.at/