E-COMMERCE SALES ANALYSIS

Using PySpark & AWS

Geoff Nel
Rijul Banerjee
Min Young (Alice) Yang

Overview

As the world becomes more technologically advanced, companies are taking the route of e-commerce retail businesses rather than leasing out real estate to build their merchandising empires. E-commerce has developed vastly over the years, from CompuServe introducing the world to its first e-commerce business in 1969 by utilizing dial-up connectivity to Paypal pioneering the first e-commerce payment transaction system in 1998, following the boom of online shopping platforms and changing the buying habits of consumers all over the world. With shopping readily available to consumers at the tip of their fingers and the increase of online competitions, businesses now face a new challenge of understanding the buying behaviors of their customers, seasonal peaks, popularity amongst categories, and setting the right pricing trend for their products.

Our objective is to analyze the habitual buying behavior of consumers during the months of October, November, December, and January on an undisclosed online shopping platform. The analysis will be done on purchase time, category popularity, brand trends, and pricing on event types. Listed below is our project module.

Project Module

Phase	**Description**	**Explanation**	**Timeline**
Phase 1	Dataset Assignment	Assigning each member with a dataset to analyze.	Week 1
Phase 2	Data Exploration	Members will explore their assigned dataset to generate ideas on what to analyze.	Week 2
Phase 3	Data Pre-processing	Cleaning the dataset to fit the models,	Week 3
Phase 4	Exploratory Data Analysis	Generating visualization to understand the story of the business.	Week 4-5
Phase 5	Data Combination	Appending all datasets together.	Week 6
Phase 6	Combined Data Analysis (Ensemble)	Analyzing the appended dataset with the models from subset data to view the trends of each month.	Week 7
Phase 7	Finalize Analysis	Finalize ensemble analysis.	Week 8
Phase 8	Present Findings	Final Presentation	Week 9

Collaboration Details

There were four separate datasets showing the purchasing behavior for the months of October, November, December, and January. Each member of the team was assigned one month's data to explore and analyze independently. Geoff Nel took on the January dataset to analyze the spending habits of customers on a time trend by parsing the date variables into day of the week and time of the day. Rijul Banerjee was assigned the November dataset to explore the

changes of customer behavior as the holiday season approaches, while evaluating the price distinction on views and purchases. Alice (Min Young) Yang was appointed the October dataset to analyze the routine of customer viewing, carting, and purchasing behavior within a non-holiday month while investigating the popularity of items within each category. With each assignment, the members explored different methods to analyze their dataset by applying various codes with different packages to later implement to an ensemble analysis of all the datasets combined.

Data Description

The e-commerce data was found on Kaggle.com, originally consisting of two months of information on a multi-category online store with 14 gigabytes of data. Additional data on proceeding months were provided separately due to the limitation on file storage on the platform. Each observation constituted an event associated with an event time, product ID, category ID, category code, brand, price, user ID, and user session code. Each event shows the action taken by the user, whether the product was viewed, placed into the cart, or purchased. The event time stamped a date and hour for each event, while the product ID identifies the item that was viewed, carted, or purchased. The category ID distinguishes each category in numeric codes and the category code expressed the category in terms. The brand lists out the make of the products and the prices show the value of each item. The user ID identifies each individual user and the user session shows the period of each event during the user's session on the e-commerce site.

```
+--------------------+----------+----------+-------------------+-------------+--------+-----+---------+--------------------+
|          event_time|event_type|product_id|        category_id|category_code|   brand|price|  user_id|        user_session|
+--------------------+----------+----------+-------------------+-------------+--------+-----+---------+--------------------+
|2020-01-01 00:00:...|      view|   5809910|1602943681873052386|         null| grattol| 5.24|595414620|4adb70bb-edbd-498...|
|2020-01-01 00:00:...|      view|   5812943|1487580012121948301|         null|kinetics| 3.97|595414640|c8c5205d-be43-4f1...|
|2020-01-01 00:00:...|      view|   5798924|1783999068867920626|         null|  zinger| 3.97|595412617|46a5010f-bd69-4fb...|
|2020-01-01 00:00:...|      view|   5793052|1487580005754995573|         null|    null| 4.92|420652863|546f6af3-a517-475...|
|2020-01-01 00:00:...|      view|   5899926|2115334439910245200|         null|    null| 3.92|484071203|cff70ddf-529e-4b0...|
+--------------------+----------+----------+-------------------+-------------+--------+-----+---------+--------------------+
only showing top 5 rows
```

Data Statistics & Format

The four independent datasets: October, November, December, and January data, contained 9 variables event_time, event_type, product_id, category_id, brand, price, user_id, and user_sessions. There are 4 different data types: string, integer, long, and double. The Event_time, event_type, category_code, brand, user_sessions were displayed as a string data type, while the product_id and user_id were presented in integer data type, the category_code was demonstrated as long data, and price was shown as a double data type.

The October dataset had 42,448,764 observations with 5.3 gigabytes of file size, the November dataset had 67,501,979 observations with 8.4 gigabytes of file size, the December dataset is in an archive compressed file that had 67,542,878 observations with 2.7 gigabytes (9.1 GB uncompressed), and the January dataset had 55,967,041 observations with 2.2 gigabytes (7.6GB uncompressed). After counting the total records within each month, we see that there is a range between 42.4 to 67.5 million event impressions per month. October having the lowest total events at 42.4 million, and January containing 56 million. Both November and December log a total of 67 million total events for each month.

```
Total columns are: 9,
Total records are: 42448764
The column names are:
                0
0     event_time
1     event_type
2     product_id
3    category_id
4  category_code
5          brand
6          price
7        user_id
8   user_session
```

```
Total columns are: 9,
Total records are: 67501979
The column names are:
                0
0     event_time
1     event_type
2     product_id
3    category_id
4  category_code
5          brand
6          price
7        user_id
8   user_session
```

```
Total columns are: 9,
Total records are: 67542878
The column names are:
                0
0     event_time
1     event_type
2     product_id
3    category_id
4  category_code
5          brand
6          price
7        user_id
8   user_session
```

```
Total columns are: 9,
Total records are: 55967041
The column names are:
                0
0     event_time
1     event_type
2     product_id
3    category_id
4  category_code
5          brand
6          price
7        user_id
8   user_session
```

```
root
 |-- event_time: string (nullable = true)
 |-- event_type: string (nullable = true)
 |-- product_id: integer (nullable = true)
 |-- category_id: long (nullable = true)
 |-- category_code: string (nullable = true)
 |-- brand: string (nullable = true)
 |-- price: double (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- user_session: string (nullable = true)
```

After exploring each dataset and the variable structures we examined all the months for the null values, and we found that there were signs of data consistency issues. The months of October and November both have roughly 30% of the data missing for the category code column. In comparison, December, and January both have roughly 10% of the values missing for the same column. This percentage accounts for a total of between 13 million to 21 million events for the months of October and November, respectively.

December and January each have a relatively high total record count yet a lower total missing record count within the category code column. These months are missing a total of 7 million and 5 million total values, respectively. Additionally, we do notice that there are missing records for both brand and user session, although missing values within user session is not considerable or significant. There is an average of about 13% missing data points in the brand column across all months; the lower range for January is 11.7% and there is a high for October of14.4%. Upon this discovery, we found it more useful to keep the null values in respect of our price and event type analysis.

By Month Null Value Count & Percent

OCTOBER

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	0	0	0	13515609	6113008	0	0	2

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0.0	0.0	0.0	0.0	31.84	14.4	0.0	0.0	0.0

Null:

Cat Code - 31.8%

Brand - 14.5%

NOVEMBER

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	0	0	0	21898171	9218235	0	0	10

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0.0	0.0	0.0	0.0	32.44	13.66	0.0	0.0	0.0

Null:

Cat Code – 32.4%

Brand – 13.7%

DECEMBER

Null:

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	0	0	0	7088848	8115813	0	0	21

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0.0	0.0	0.0	0.0	10.5	12.02	0.0	0.0	0.0

Cat Code - 10.5%

Brand – 12.0%

Null:

Cat Code – 9.0%

Brand – 11.67%

JANUARY

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	0	0	0	5044890	6532738	0	0	19

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0.0	0.0	0.0	0.0	9.01	11.67	0.0	0.0	0.0

Data pre-processing

The datasets were presented in tabular form, but to perform a deep dive analysis on event_types the categorical variables need to be transposed into columns. There were three major event types (view, cart, and purchase) to enhance the comprehension of the actions taken by users during each month. After transposing the event types into columns, we then used the groupby function to reduce the categories to understand the popularity of products in each category. The count function was also used to tally the total of each category that was clicked on by users to see which category gained highest interest among customers. The average price was calculated to recognize fluctuations throughout the months to see the incline and decline of prices in a given month. The event_time was transformed into a date format and parsed to distinctly see the trends of user activity in hours per day and days per weeks cycles. Each step was repeated for all the datasets to see the changes of consumer behavior as the month proceeded.

Data Analysis

During our research on the e-commerce dataset, we analyzed the consumer behavior during the four months by evaluating five core variables: time, event type, category, brand, and price. We examined the time variable to understand the behavioral shifts among consumers as time passed and noticed that in the month of January, there were major spikes in the total purchase count, reaching a maximum of approximately 20,000 purchases as a cumulative for both Saturday and Sunday. All other values for the month do seem normal otherwise. We further see that Mondays and Sundays carry the most amount of purchases for December, while Tuesdays were the best day in total purchase count for October, and Wednesdays in January.

The event type analysis showed a short consumer journey of viewing an item to checking out to purchasing the item. We found that all four months had higher views than cart or purchase, especially in the months of November and December. The results of the analysis concluded to be that there are generally more views and purchases in November and December due to the holiday seasons and sales; November is the month of Thanksgiving, followed by the biggest sale event of the years - Black Friday, consumers are more likely to purchase items during this time of the year due to the incentives of sale discounts applied. December is the month of Christmas, one of the biggest holidays of giving and receiving gifts, therefore the purchase counts in this month are ranked the highest among all others. The analysis also confirmed that during a non-holiday season there are less view and purchases.

The category analysis displayed a high popularity in electronic goods, especially during the month of November, which affirms our analysis on consumer incentive purchases. We see that computers and electronics reaches a peak during November due to the exceptional Black Friday event sales that go on during the month, which incentivizes consumers to make purchases on items at a discounted rate and pay less for more value. We also saw a spike in sports good in November and December, which concluded to be precisely around the time NFL and NBA seasons begin to hype up. Construction good seemed to generate a high peak season around December and January, which showed that many consumers tend to start fixing and/or augmenting their properties during the New Years' time.

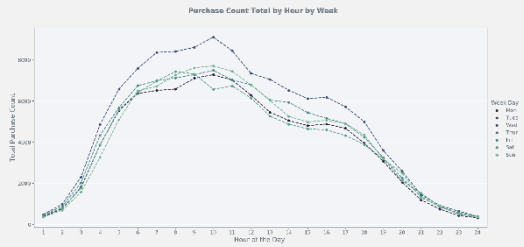
Data Visualization and Explanation

I. Analysis on Time Variable

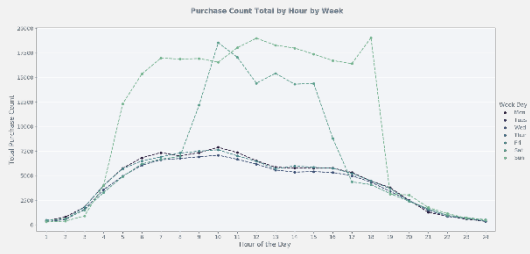
Below we can see that when we review the purchase count of each month by the hour of the day, there are general trends. For instance, purchases are generally increasing between 3am and 6am, while generally trending down after 6pm. We also notice that there are two peaks within the hours of the day across each of the months. The first peak that we see is around 9am, while the second is around 4pm.

By Month Review of Purchase Count by Hour of the Day

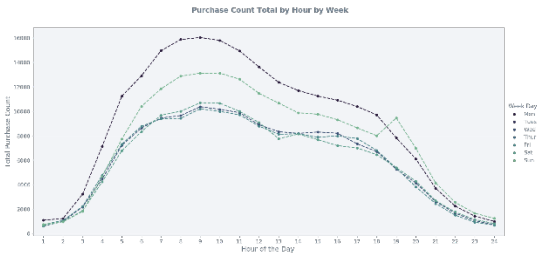
OCTOBER



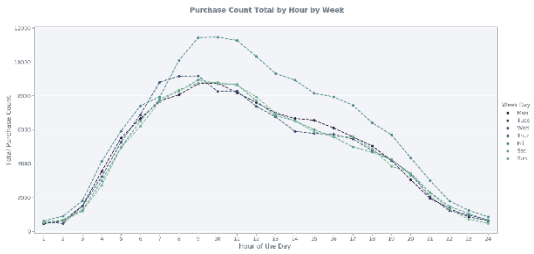
NOVEMBER



DECEMBER



JANUARY

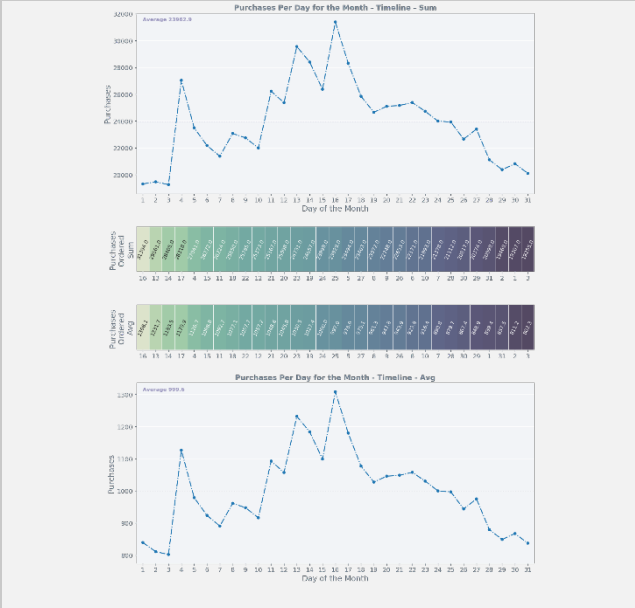


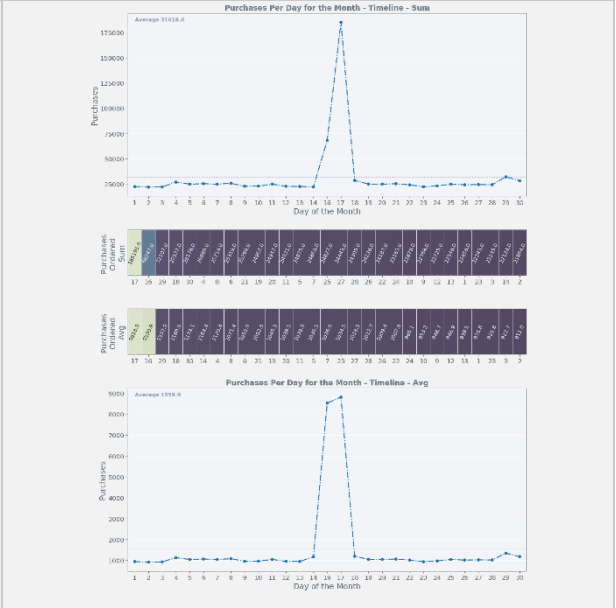
Description:

The above graphs illustrate the by month purchase count per hour of the day by weekday. The x-axis contains the hour of the day, while the y-axis shows the count. The week is color coded Mon-Fri darkest-lightest.

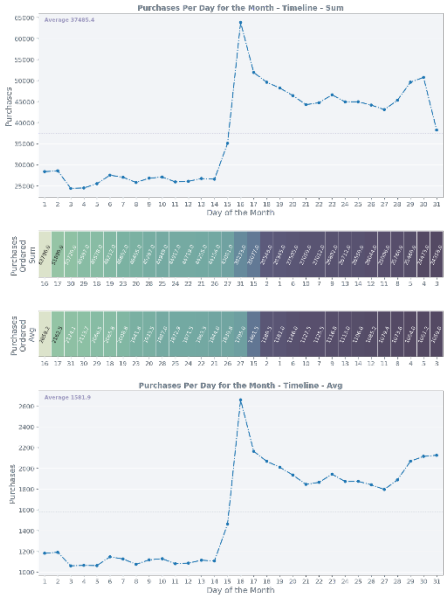
By Month Review of Purchase Count & Average by Month Day

OCTOBER	**NOVEMBER**

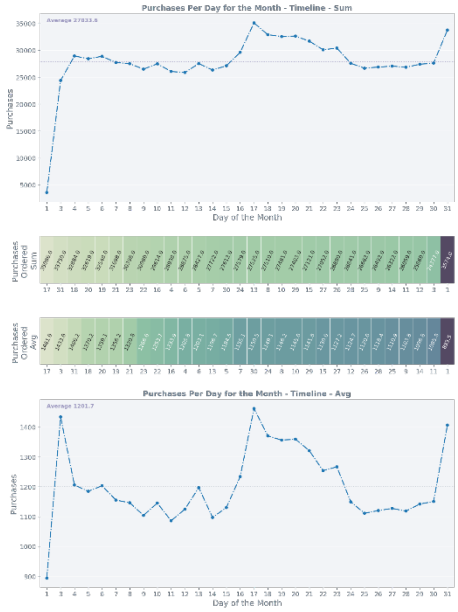




DECEMBER



JANUARY



Description:

The above graphs illustrate the by month purchase count per day of the month. This is consolidated by both total count and average. We have also included a row-wise heatmap of the purchase count and average by month in-between each by-month-day graph in descending order. Finally, the average line per graph is also included.

In looking at the graphs above, we notice that each month has a general bi-monthly spike, where there is one spike mid-month and another smaller spike at the end of the month. This does not include October, which although it does have one mid-month spike, it also has several other smaller spikes throughout the first half of the month. In looking at November, we see that there is a mid-month spike in both overall purchases by count and on average by a factor of roughly

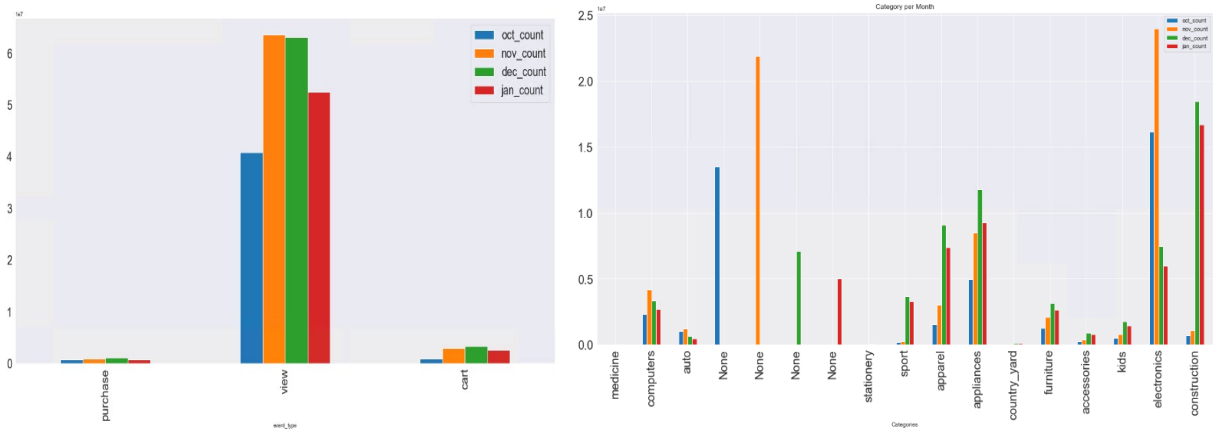
eight times. This very likely fits with our previously analyzed purchase count by day, where we saw a major spike in purchase counts on a consecutive Saturday and Sunday in November.

In January, we also notice that purchases drop to the lowest of all the months. By the end of December, we see that average purchases are hovering around 2000, while on the first day of January we see that average purchases start from an average low of 900, which is the lowest of the four months. For January, this is quickly reconciled in the following recorded day, where we see a spike up to 1400, with a continuing average of 1200 for the month. One note we must make in respect to our analysis within the month of January, is that the 2nd of January is missing from the dataset. All other months seem to contain all days for the month, while January does not.

II. Event Type by Month

The event type by month shows that November has the highest views on products, followed by December, January, then October. However, December shows to have the highest purchases and carts. We concluded that many users like to explore items during the holiday season but start purchasing items right before Christmas.

event_type	oct_count	nov_count	dec_count	jan_count
purchase	742849	916939	1162048	835007
view	40779399	63556110	62986067	52490785
cart	926516	3028930	3394763	2641249



III. Category

When we move onto reviewing revenue by category code, we do see a few issues of concern. For the months of October and November we see that the top one and two revenue generating categories are electronics, which comes in between $175 million and $200 million; the next category, being appliances, dropping back to roughly $15 million $2 million. This is a large margin and we can speculate that this eCommerce business would focus their revenue in the area of electronics, however, when we begin reviewing the top revenue generators for the following months, December and January, we see a different trend. The top revenue generating category is now construction, with a similar second place maintained by appliances. Appliances contribute much more than the preceding two months, with an average of about $38 million as compared to roughly $9 million for the months of October and November.

On these concerns, we can formulate two hypotheses: one, there was an adjustment to how categories are prescribed in the months of December and January as opposed to the previous months; two, the categories are defined in a more liquid way than is discernible through the data. It may be possible that certain electronics fall into multiple categories and when a product is purchased, it is the leading category from where the user searched that is then logged by the database. For instance, if a product is categorized in both electronics and construction and if a user finds and purchases the product through searching in the construction category, then that purchase is logged as a construction purchase event even if the product is largely an electronic device.

IV. Revenue & Brand per month

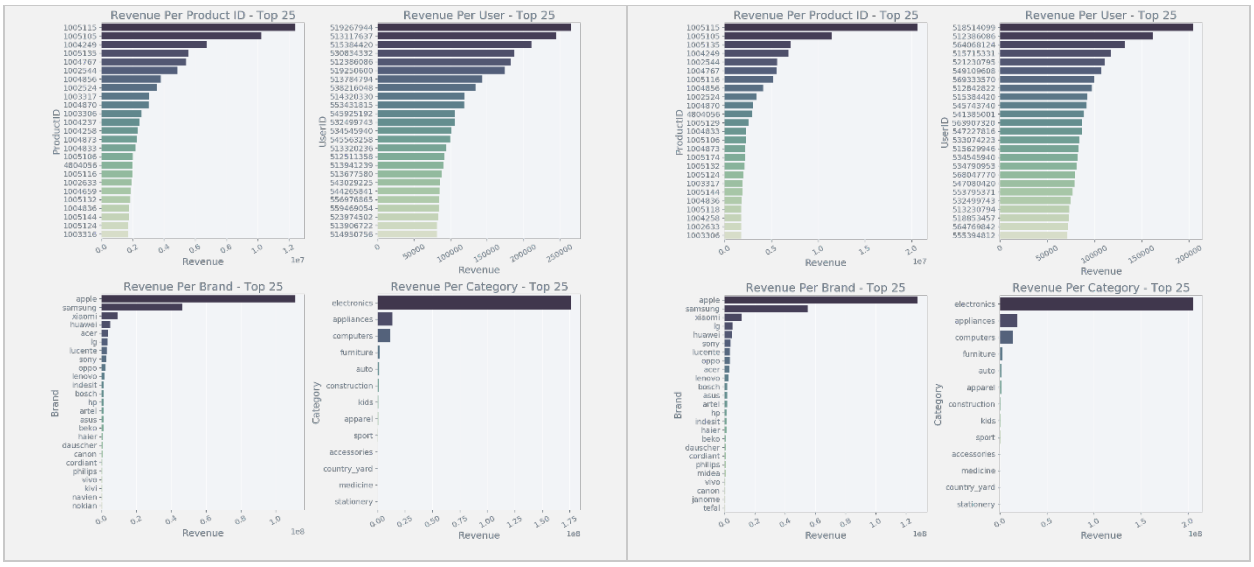
In looking at the top revenue generators for each month, we see that there is at least one, sometimes two, products that lead in revenue generation by a large extent. For all four months we see that Apple is the leading revenue generator. This comes in at an average of roughly $126 million in revenue. The next closest is Samsung at an average across all months of $61 million.

The top revenue generating category for December and January is then construction as opposed to electronics for October and November. Where electronics comprised an average of $187 million in October and November, construction makes up an average total of roughly $198 million. These numbers do not make sense when the top 5 leading revenue generating brands all fall into the electronics category. It is feasible for the months of October and November, but there is concern for December and January. In looking further into the graphs below, we can further see the by-month revenue generation by Product ID and User ID. We note that although our analysis would benefit by having a more descriptive product identification, the ID is all that is available to us.

Here, we see that the products with ID's equal to 1005115 and 1005105 lead each of the four months in revenue generation. They comprise an average total revenue of $152 million and $122 million, respectively. We then also see that there are three product ids that are consistently in the top; these are 1004249, 1005135, and 1004767.

Finally, by looking at the user ID category that is available to us, we see that unlike the other categories where there is consistently a top product or brand, the top users differ per month. From the months of October to January, these are as follows: October, with 519267944, 513117637, 515384420; November, with 518514099, 512386086, 564068124; December, with 553431815, 569333570, 513901034; January, with 562104312, 515428951, 563599039. Within this top three review, there are no User ID's that carry over from month to month; however, it may be worth reviewing these users on a yearly basis. Due to the fact that the revenue per user is in the millions, it is more likely that these are business accounts and that there may be a per year trend when reviewing the purchase patterns of certain businesses.

A By Month Review of Top Revenue Generating Categories	
OCTOBER	***NOVEMBER***



DECEMBER ***JANUARY***

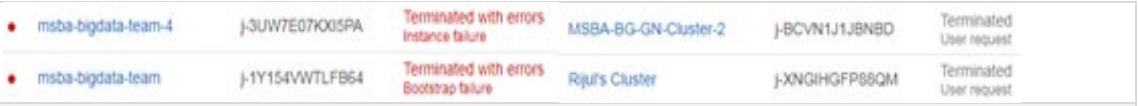
Description:

The above graphs are intended to breakdown the overall revenue of each of the categorical values available to us. These include Product ID, User ID, Brand, and Category by Month. These are visible in order from top left, moving right, to bottom left, whereby the bottom most right contains the category revenue for January.

Challenges & Limitations

When we initially started this project, we decided to leverage the cluster computing that spark is known for. This led us into setting up cluster computing on AWS. AWS utilizes EMR's, which is their form of a managed cluster platform. Our team tried several iterations of EMRs on the platform, where it was possible to initiate the EMR instance, however, it was not possible to remote into the instance. This issue was not resolvable, which lead us to review additional avenues for running spark. These EMR instances are noted below.

EMR Cluster Computing - Issue



Description:
Above, we see four examples of EMR instances. The two on the left failed to launch, while the instances on the right had success in launching, although it was not possible to remote into these instances.

Following the EMR setup issues, we continued to use AWS, but resorted to using a single EC2 instance. We ensured that there was an allocation of 32 GB of RAM to the system, an essential component to spark; after successfully logging into the system, we did encounter additional challenges within the EC instance and in using the spark framework. These are noted as follows:

1. Importing large data files from the S3 bucket.
2. Adjusting the PySpark specific programming syntax.
3. Converting necessary values, like the timestamp and category values.
4. Learning how to leverage big data processing techniques.
5. Refining processing efficiency and runtime.

Description:
The runtime on large files became a challenge for us in that even minor changes to a query could amount in a wall time of between 20 to 30 mins. Above we see this for four queries in the month of December.

Code Deployment

Spark – Initial Setup

In [1]:
```
from IPython.core.display import display, HTML

def jupyterwidthfunction():
#     width = input('What width do you want? ')
#     width=int(width)
    display(HTML("<style>.container { width:95% !important; }</style>"))

jupyterwidthfunction()
```
executed in 11ms, finished 06:32:18 2020-05-30

In [2]:
```
# source: https://www.kaggle.com/fatmakursun/pyspark-ml-tutorial-for-beginners

import os
import pandas as pd
import numpy as np
import boto3

from IPython.display import display as disp

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext
from pyspark.context import SparkContext
from pyspark import RDD, since, keyword_only

from pyspark.sql.types import *
import pyspark.sql.functions as F
from pyspark.sql.functions import udf, col

import seaborn as sns
import matplotlib.pyplot as plt
import time

# from pyspark import SparkConf
# from pyspark.sql import SparkSession, SQLContext
```
executed in 1.53s, finished 06:32:20 2020-05-30

In [3]:
```
from pyspark.sql import SQLContext
sc = SparkContext()
sqlContext = SQLContext(sc)
```
executed in 3.55s, finished 06:32:23 2020-05-30

In [4]:
```
spark = SparkSession.builder.master("yarn").appName("MyApp").getOrCreate()
spark.sparkContext._conf.getAll()
```
executed in 589ms, finished 06:32:24 2020-05-30

Description:
Imported packages to use throughout the analysis. We started with the basic packages for pyspark and a few visualization packages.

Spark - Descriptive Statistics

In [12]:
```
# Count columns and see the names
cols = []
for i in df_oct.columns:
    #print(i)
    cols.append(i)
cols = pd.DataFrame(cols)

print("Total columns are: {}, \nTotal records are: {} \nThe column names are: \n{}".format(len(cols),df_oct.count(), cols))

# Count columns and see the names
cols = []
for i in df_nov.columns:
    #print(i)
    cols.append(i)
cols = pd.DataFrame(cols)

print("Total columns are: {}, \nTotal records are: {} \nThe column names are: \n{}".format(len(cols),df_nov.count(), cols))

# Count columns and see the names
cols = []
for i in df_dec.columns:
    #print(i)
    cols.append(i)
cols = pd.DataFrame(cols)

print("Total columns are: {}, \nTotal records are: {} \nThe column names are: \n{}".format(len(cols),df_dec.count(), cols))

# Count columns and see the names
cols = []
for i in df_jan.columns:
    #print(i)
    cols.append(i)
cols = pd.DataFrame(cols)

print("Total columns are: {}, \nTotal records are: {} \nThe column names are: \n{}".format(len(cols),df_jan.count(), cols))
```
executed in 6m 38s, finished 07:01:38 2020-05-30

Description:
We started our initial analysis process by first evaluating and exploring the data structures. This later on helped determine which variables needed changes in format.

Parsing Column Text with Delimiter

In [19]:
```
from pyspark.sql.functions import split

split_col=split(df_oct['category_code'], "\\.")
df_edt_oct = df_oct.withColumn('category', split_col.getItem(0)).drop("category_code")
df_edt_oct.show(5)

split_col=split(df_nov['category_code'], "\\.")
df_edt_nov = df_nov.withColumn('category', split_col.getItem(0)).drop("category_code")
df_edt_nov.show(5)

split_col=split(df_dec['category_code'], "\\.")
df_edt_dec = df_dec.withColumn('category', split_col.getItem(0)).drop("category_code")
df_edt_dec.show(5)

split_col=split(df_jan['category_code'], "\\.")
df_edt_jan = df_jan.withColumn('category', split_col.getItem(0)).drop("category_code")
df_edt_jan.show(5)
```
executed in 831ms, finished 08:05:33 2020-05-30

Description:

Category_code had different categories with subcategories within the same column. The text was separated by a '.' delimiter. We used the parsed data to group the categories to further analyze the effects on consumer behavior for each category.

Spark – Setting SQL Tables

SET SQL TABLE VIEWS

```
# Create a temporary table view
df_oct.createOrReplaceTempView("oct_table")

# Creating Purchase table
df_oct.filter('PurchaseYN == 1').createOrReplaceTempView('oct_purchased')
```

executed in 97ms, finished 10:18:00 2020-05-25

```
# Create a temporary table view
df_nov.createOrReplaceTempView("nov_table")

# Creating Purchase table
df_nov.filter('PurchaseYN == 1').createOrReplaceTempView('nov_purchased')
```

executed in 81ms, finished 10:18:01 2020-05-25

```
# Create a temporary table view
df_dec.createOrReplaceTempView("dec_table")

# Creating Purchase table
df_dec.filter('PurchaseYN == 1').createOrReplaceTempView('dec_purchased')
```

executed in 131ms, finished 19:42:45 2020-05-24

```
# Create a temporary table view
df_jan.createOrReplaceTempView("jan_table")

# Creating Purchase table
df_jan.filter('PurchaseYN == 1').createOrReplaceTempView('jan_purchased')
```

executed in 85ms, finished 19:42:46 2020-05-24

Description:

Spark allows for the use of both python's object-oriented programming style as well the use of SQL queries. In order to use SQL, we needed to set the table views. We set this for all months for two main contingencies, all data in all months, and for where the event type is equal to a purchase.

Breaking Down Timestamp Column

```
# Transfrom Datse and cols function
# Convert the remaining datetype fields "df_jan" file
# SOURCE: https://www.w3schools.com/sql/func_mysql_date_format.asp
import pyspark.sql.functions as sparkf
from pyspark.sql.functions import col
from pyspark.sql.functions import from_unixtime, date_format, to_date
from pyspark.sql.functions import to_timestamp, unix_timestamp, hour, minute
from pyspark.sql.functions import split

def transformDatesColValues(dataframe):
    # Convert 'event_time' to a data/time timestamp
    dataframe = dataframe.withColumn(
        "event_time",
        to_timestamp(dataframe['event_time'], 'yyyy-MM-dd HH:mm:ss'))

    dataframe = dataframe.withColumn('event_totd', date_format('event_time', 'HH:mm:ss')) \
                    .withColumn('event_date', date_format('event_time', 'MM-dd-yyyy'))

    # Creating new columsn for event date - event hour - event minute
    dataframe = dataframe.withColumn("event_date", to_date(dataframe['event_time'], 'yyyy-MM-dd'))\
                    .withColumn("event_hour", hour(dataframe['event_time'])+1)\
                    .withColumn("event_minute", minute(dataframe['event_time']))

    # Create a integer type time column in minutes which becomes type(int)
    dataframe = dataframe.withColumn( 'Time1440Mins',
        (dataframe['event_hour'] * 60 + dataframe['event_minute']))

    # collect day of the week - week day as mon-fri - day of the month - week of the year
    dataframe = dataframe.withColumn('dotw_num',  date_format('event_time', 'u'))\
                             .withColumn('dotw_day', date_format('event_time', 'E'))\
                             .withColumn('dotm_num', date_format('event_time', 'd'))\
                             .withColumn('woty_num', date_format('event_time', 'w'))
    dataframe = dataframe.withColumn('dotw_num',  dataframe['dotw_num'].cast('integer'))\
                    .withColumn('dotm_num',  dataframe['dotm_num'].cast('integer'))\
                    .withColumn('woty_num',  dataframe['woty_num'].cast('integer'))

    # extracting the primary category - setting the data types
    #split_col=
    dataframe = dataframe.withColumn('primary_cat', dataframe['category_code']) \
                             .withColumn('primary_cat', split(dataframe['category_code'], "\\.").getItem(0))
    dataframe = dataframe.withColumn('product_id',  dataframe['product_id'].cast('string'))\
                             .withColumn('category_id',  dataframe['category_id'].cast('string'))

    # adding a purchase column for predictions
    dataframe = dataframe.withColumn('PurchaseYN', (dataframe['event_type']=='purchase').cast('integer'))
    print('Completed')
    return(dataframe)
```

Description:

A mainstay of our data is the timestamp data type field. There is a timestamp for each event that occurs in the data. We incorporated this into our analysis by further breaking it down.

The main time oriented fields used included 'day of the week' as a number(1-7), the 'day of the week as a day' (Mon-Sun), the 'day of the month' (1-30/31), and the 'week of the year' (1-52).

Link to Presentation

https://docs.google.com/presentation/d/1OLSDaMFsIXUtIbYxLenwFW616bq_6mds/edit#slide=id.g861a51a686_1_0