Executive Summary

Big Data & Information System- Group 4
Prof. Curry

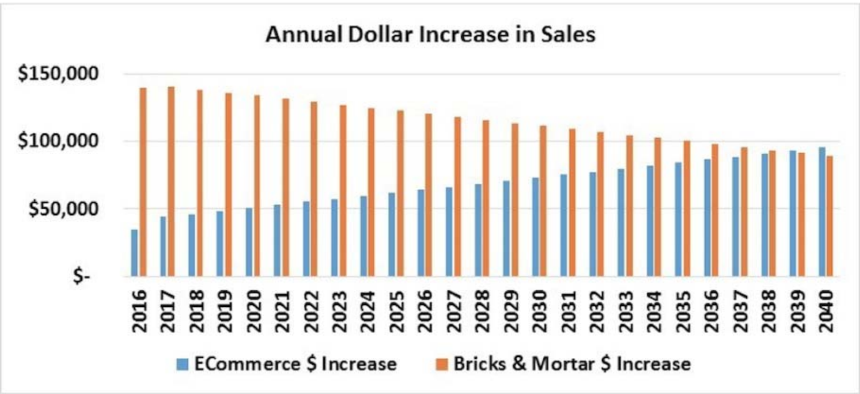
Yongnian Cao
Amita Akole
Vishal Goyal
Shivam Khare
Sunil Kumar
Sushil Satya

E-Business Introduction

E-commerce is a very dynamically evolving industry, and this is primarily because of its underlying ever-changing technology.Companies like Amazon, E-bay are capable of building predictive algorithms being executed in real time on big data environment.

There has been an increasing emphasis on big data analysis in e-commerce in recent years.The business community within the past few years has been characterized by talking about the effectiveness of e-commerce and how this type of innovation should be actualized. Amazon.com is a leader in collecting, storing, processing, and analyzing personal information from you and every other customer as a means of determining how customers are spending their money.

This information is used to recommend additional products that other customers purchased when buying those same items, which product is the most popular, what season/ occasion people prefer for online shopping so that we can advertise more during that phase. These analysis help us to figure out every customer's behaviour. These analyses help us to take decisions to increase company's profit. For example, when you add a DVD to your online shopping cart, similar movies purchased by other customers are also recommended for you to purchase. In this way, Amazon uses the power of suggestion to encourage you to buy on impulse as a means of further satisfying your shopping experience and spending more money.



(Dollars in millions, so the top line is $150 billion)

PROBLEM STATEMENT

Simulation:

The following diagrams are coding sections for product count, city count, and visiting time count of the customers with their respective outputs.

1. Product count result hypothesis: We assume customers prefer to buy some daily necessaries, books, and clothes.

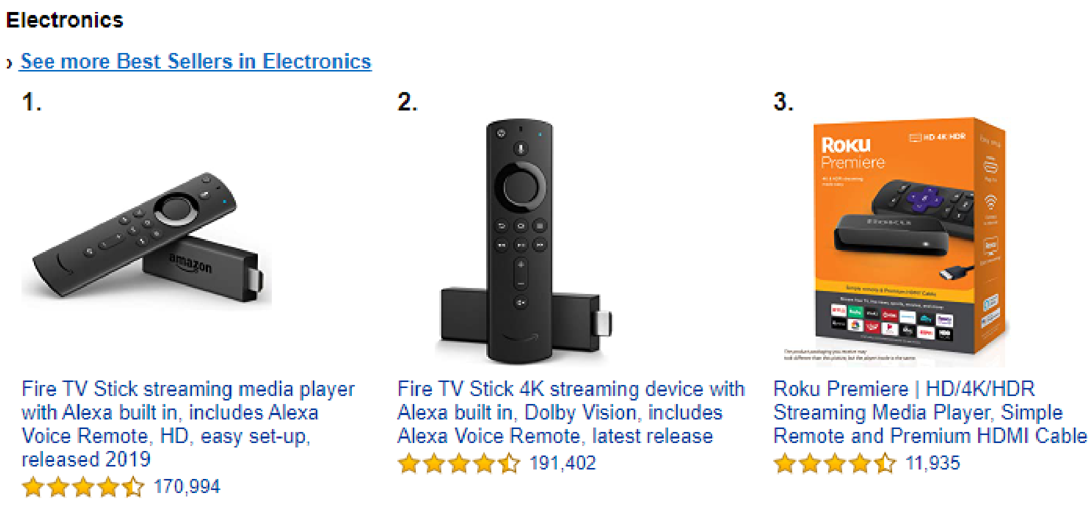
2. City count result hypothesis: We assume some higher population cities present a higher sales count like in New York City, California.

3.The data set doesn't show the user's gender. It causes a little inconvenience in some analysis that we will be performing.

Data Preparation

Overview the data set:

There are 8 columns in our E-Business data set namely Event_time, Event_type, Product_Id, Category_Id, Brand, Price, User_Id, Session_Id, and City_Id.



Strength:

1. This data set is easier to understand and format friendly rather than JSON format.
2. This data set is simple to maintain. Sometimes, we delete some data occasionally. We can restore by using SQL sentence in our information database.

Opportunities

In our project, the information will be used to recommend additional products that other customers purchased when buying those same items. Additionally, this will show results of which product is the most popular, what season/ occasion people prefer for online shopping so that we can advertise more during that phase.

Sandbox & Testing

What is an Analytics Sandbox?

An Analytics Sandbox is a separate environment that is part of the overall data lake architecture, meaning that it is a centralized environment meant to be used by multiple users and is maintained with the support of IT.   Here are some key characteristics of a modern Analytics Sandbox.

Advantages of an Analytics Sandbox

There are many advantages to having an Analytics Sandbox as part of your data architecture. Perhaps most significant is that it decreases the amount of time that it takes a business to gain knowledge and insight from their data. It does this by providing an on-demand/always ready environment that allows analysts to quickly dive into and process large amounts of data and prototype their solutions without kicking off a big BI project. In other words, it enables agile BI by empowering your advanced users.

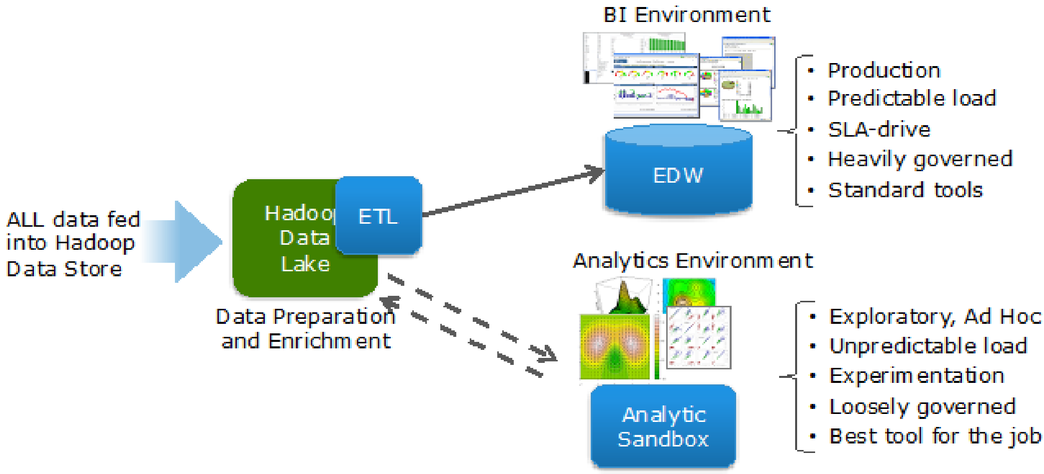


Figure 3.1 Sandbox Diagram

Production data-set

A production data set used for production tasks such as creating and updating features. In our project, production data set is Amazon's product sale by October including 1 million customer's records in the first week. This data set includes customer information like user-id, city-id, and session-id, also product meta data like descriptions, category information, price, and brand.

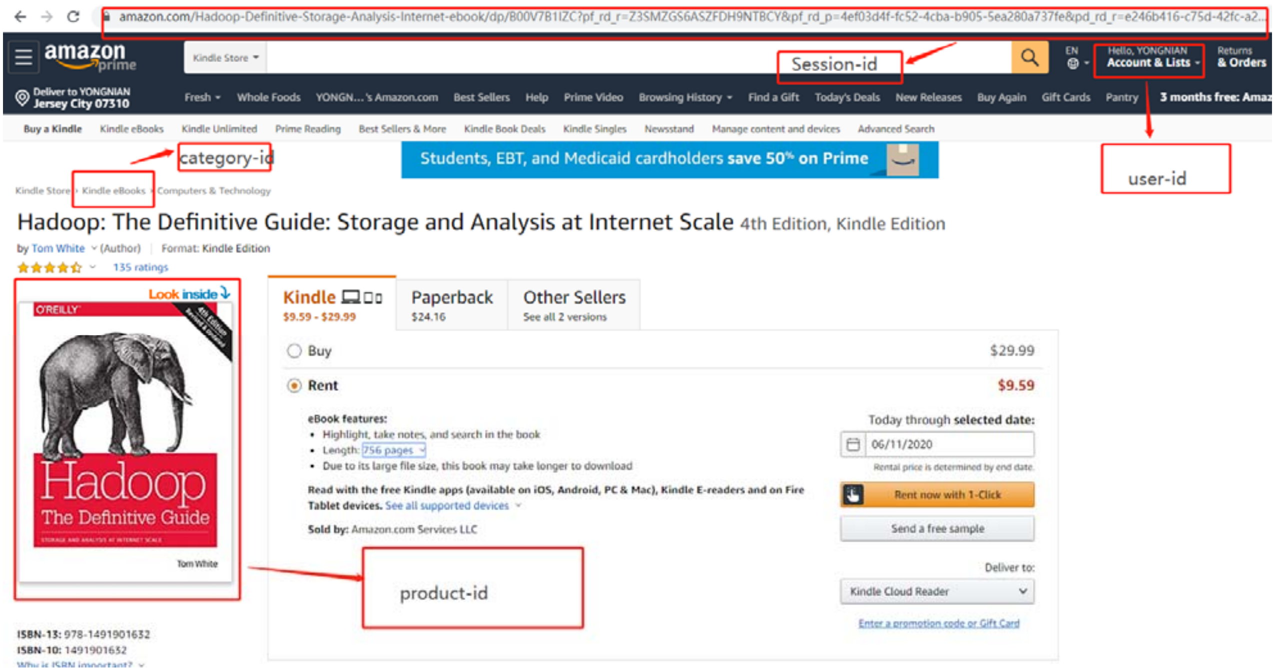


Figure 4.1 Original Data

Data Process

Data Processing Procedure

Actually, This data set came from PC and mobile app. Our company's web server capture every customer's visiting, order and payment behaviour and store into Nginx. We get the original data set from web department colleagues and extract, transform and load it. The part is ETL.

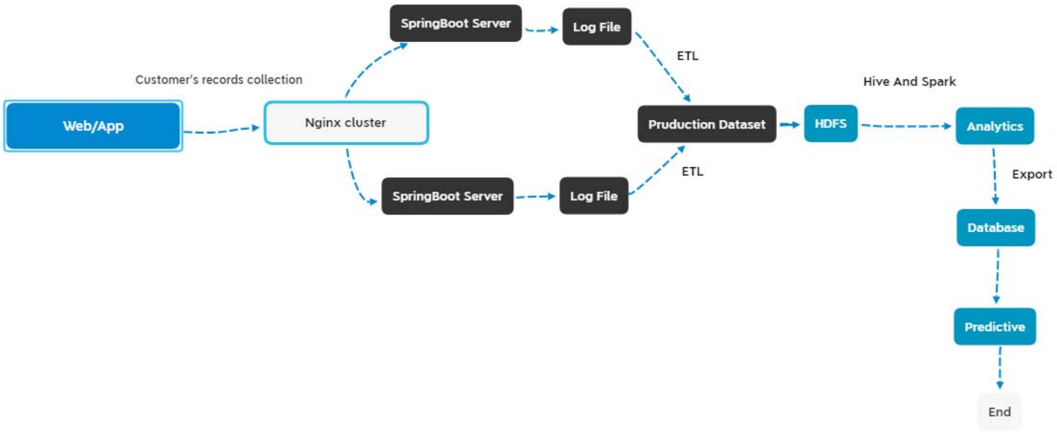


Figure 5.1 Data Transformation

Load data

We need to load our data-set before we take ETL part. We get the direction where our data-set store in and load into Spark. The code in the Photo 1.1.

Why do we need to ETL our data-set?

Sometimes, a Web server cannot collect some of the customer's information and the value is missing. These missing values might affect our results if we did not filter them. In this case, we filter the user-id, product-category-id these columns value is null.

In our case, we load our data and filter some missing value. In our case, we want to calculate the product, city, and time count. So we need to clean the wrong value in these columns. Also, we count how many pieces are left in our data set. If we filter too much, we need to fill by average value or the most frequent instead of deleting them. The code in the photo 1.2.

1	event_time	event_type	product_id	category_ic	brand	price	user_id	user_session	city_id
2	2019-10-01 00:00:00 UTC	cart	5773203	149	runail	2.62	463240011	26dd6e6e-4dac-4778-8d2c-9	6
3	2019-10-01 00:00:03 UTC	cart	5773353	149	runail	2.62	463240011	26dd6e6e-4dac-4778-8d2c-9	2
4	2019-10-01 00:00:07 UTC	cart	5881589	178	lovely	13.48	429681830	49e8d843-adf3-428b-a2c3-fe	1
5	2019-10-01 00:00:07 UTC	cart	5723490	149	runail	2.62	463240011	26dd6e6e-4dac-4778-8d2c-9	1
6	2019-10-01 00:00:15 UTC	cart	5881449	149	lovely	0.56	429681830	49e8d843-adf3-428b-a2c3-fe	5
7	2019-10-01 00:00:16 UTC	cart	5857269	149	runail	2.62	430174032	73dea1e7-664e-43f4-8b30-d	1
8	2019-10-01 00:00:19 UTC	cart	5739055	149	kapous	4.75	377667011	81326ac6-daa4-4f0a-b488-fd	5
9	2019-10-01 00:00:24 UTC	cart	5825598	191		0.56	467916806	2f5b5546-b8cb-9ee7-7ecd-8	1
10	2019-10-01 00:00:25 UTC	cart	5698989	149		1.27	385985999	d30965e8-1101-44ab-b45d-c	9
11	2019-10-01 00:00:26 UTC	view	5875317	149		1.59	474232307	445f2b74-5e4c-427e-b7fa-6e	4
12	2019-10-01 00:00:28 UTC	view	5692917	149	lianail	5.54	555446068	4257671a-efc8-4e58-96c2-3a	10
13	2019-10-01 00:00:28 UTC	remove_from_cart	5834172	149	runail	0.95	429681830	49e8d843-adf3-428b-a2c3-fe	3
14	2019-10-01 00:00:30 UTC	remove_from_cart	5809103	149	irisk	0.6	429681830	49e8d843-adf3-428b-a2c3-fe	3
15	2019-10-01 00:00:30 UTC	remove_from_cart	5809103	149	irisk	0.6	429681830	49e8d843-adf3-428b-a2c3-fe	1
16	2019-10-01 00:00:32 UTC	remove_from_cart	5779403	149		12.22	429681830	49e8d843-adf3-428b-a2c3-fe	1
17	2019-10-01 00:00:33 UTC	remove_from_cart	5779403	149		12.22	429681830	49e8d843-adf3-428b-a2c3-fe	8
18	2019-10-01 00:00:34 UTC	cart	5670337	149		2.38	546705258	3b5c65c0-bb1c-453b-b340-4	6
19	2019-10-01 00:00:42 UTC	cart	5836522	149	nagaraku	0.4	429681830	49e8d843-adf3-428b-a2c3-fe	7
20	2019-10-01 00:00:43 UTC	cart	5836522	149	nagaraku	0.4	429681830	49e8d843-adf3-428b-a2c3-fe	2
21	2019-10-01 00:00:48 UTC	view	5819638	149		21.75	546705258	3b5c65c0-bb1c-453b-b340-4	8
22	2019-10-01 00:00:48 UTC	cart	5859414	149	masura	2.37	555442940	618f3d7d-2939-47ea-8f1d-07	1
23	2019-10-01 00:00:53 UTC	view	5856191	149	runail	24.44	507355498	944c7e9b-40bd-4112-a05b-8	8
24	2019-10-01 00:00:55 UTC	cart	5859413	191	masura	2.37	555442940	618f3d7d-2939-47ea-8f1d-07	2
25	2019-10-01 00:00:56 UTC	remove_from_cart	5881589	149	lovely	13.48	429681830	49e8d843-adf3-428b-a2c3-fe	2
26	2019-10-01 00:01:01 UTC	cart	5723518	191	runail	2.62	430174032	c2bbd970-a5ad-42dd-a59b-f	2
27	2019-10-01 00:01:02 UTC	remove_from_cart	5848908	149	bpw.style	1.9	429681830	49e8d843-adf3-428b-a2c3-fe	1
28	2019-10-01 00:01:03 UTC	cart	5677366	191	estel	7.3	524009100	8bbff347-0be1-470e-8860-d	2
29	2019-10-01 00:01:03 UTC	cart	5859411	149	masura	2.37	555442940	618f3d7d-2939-47ea-8f1d-07	2
30	2019-10-01 00:01:03 UTC	remove_from_cart	5729011	191	ingarden	0.79	429681830	49e8d843-adf3-428b-a2c3-fe	2
31	2019-10-01 00:01:05 UTC	remove_from_cart	5858981	149	de.lux	0.79	429681830	49e8d843-adf3-428b-a2c3-fe	1
32	2019-10-01 00:01:05 UTC	remove_from_cart	5858981	149	de.lux	0.79	429681830	49e8d843-adf3-428b-a2c3-fe	7
33	2019-10-01 00:01:06 UTC	remove_from_cart	5857269	149	runail	2.62	430174032	c2bbd970-a5ad-42dd-a59b-f	7
34	2019-10-01 00:01:06 UTC	cart	5859410	149	masura	2.37	555442940	618f3d7d-2939-47ea-8f1d-07	2
35	2019-10-01 00:01:06 UTC	remove_from_cart	5728995	149	ingarden	0.79	429681830	49e8d843-adf3-428b-a2c3-fe	1
36	2019-10-01 00:01:07 UTC	remove_from_cart	5312	149	runail	1.27	467916806	2f5b5546-b8cb-9ee7-7ecd-8	10
37	2019-10-01 00:01:07 UTC	remove_from_cart	5728995	149	ingarden	0.79	429681830	49e8d843-adf3-428b-a2c3-fe	8
38	2019-10-01 00:01:07 UTC	remove_from_cart	5312	160	runail	1.27	467916806	2f5b5546-b8cb-9ee7-7ecd-8	8
39	2019-10-01 00:01:10 UTC	remove_from_cart	5823915	149	milv	1.59	429681830	49e8d843-adf3-428b-a2c3-fe	1

Figure 5.1 Analytical Data set

No	Column	Description
1	Event_time	Which time customer is visiting website
2	Event_type	behavior of the customer doing
3	Product_Id	What is the id of Product
4	Category_Id	Selected product belongs to which category
5	Brand	The brand name of product
6	Price	The single price of product
7	User_Id	The id number of this user
8	Session_Id	The Session id of one website access behavior
9	City_id	The location of this visiting action

Figure 5.2 The description of analytical data set

Data manipulation

Data Mining

We want to know every day's customer's count and calculate every day's new customer, the active customer. This business request helps us to find some problems. For example: Our company average customer's number is 20000 a day but there was 10000 yesterday. So we can test some problems based on this result. Firstly, we need to put customer's data into different files by their visiting time and we cannot change the data. The output is a text format file. The code in the 2.2-1.

2019-10-01.txt	2020/4/23 11:21
2019-10-02.txt	2020/4/23 11:21
2019-10-03.txt	2020/4/23 11:21
2019-10-04.txt	2020/4/23 11:21
2019-10-05.txt	2020/4/23 11:21
2019-10-06.txt	2020/4/23 11:21
2019-10-07.txt	2020/4/23 11:21

```
hive (is669project)> select * from new_customer_count limit 10;
OK
new_customer_count.user_id      new_customer_count.dt
109256370       2019-10-01
114598339       2019-10-01
115036271       2019-10-01
115752073       2019-10-01
118155758       2019-10-01
118375341       2019-10-01
121830501       2019-10-01
121981995       2019-10-01
122626605       2019-10-01
128562720       2019-10-01
```

In this case, we need to upload our result into Hadoop and taking some HIVE analysis. We use scripts to load data into the data warehouse. The code in the photo 2.2-2.

We want to calculate how many new customers every day. We use SQL sentence to join the 'active customer table' and 'new customer table'. For example, if some customers appear in today's file but not in our database. It shows these customers are new. We calculate every day's new customer count and active customer count respectively.

```
hive (is669project)> select * from ads_new_customer_count;
OK
ads_new_customer_count.dt       ads_new_customer_count.count
2019-10-01      19230
2019-10-02      10926
2019-10-03      14068
```

We prepare to export the result from our cluster into our database by Sqoop. The reason we take this step because the manager is easier to observe the result every day and this way helps us to take other business requests. Like, calculate the ratio between new customers and active customers every day. We set the directory of our result and import the destination of our database. The export script in the 2.2-3.

dt	acive_user_count
2019-10-01	19230
2019-10-02	11682
2019-10-03	16323

Methodology

1. product count analysis

This shows the count of products sold from each department like Electronics or Books or Video Games etc. Firstly, we need to pick the 'product' column and set by (product,1) the "key-value-Tuple". And the next step would be to aggregate them together. And the second photo shows that creating the data schema and store into the local database. The result is in the third photo and we set the table information to fit our data needs. The code in the photo 3.1.

product_name	product_count
Video Games	4648
Home and Kitchen	6076
Tools and Home Improvement	1170
Digital Music	7362
Apps for Android	1150
Office Products	1105
CDs and Vinyl	914
Automotive	31990
Patio, Lawn and Garden	5359
Pet Supplies	4162
Electronics	12101
Amazon Instant Video	2541
Grocery and Gourmet Food	1015
Books	876971
Health and Personal Care	4665
Clothing, Shoes and Jewelry	28461
Toys and Games	3508
Musical Instruments	2424
Baby	6645
Sports and Outdoors	4643
Cell Phones and Accessories	3551
Movies and TV	1423
Beauty	34541
Kindle Store	2065

2. City count analysis

This shows the count of products sold in each city. We want to find some popular cities. This result helps the advertisement department colleagues to calculate every city's budget and convert their ROI. The code means picking up city columns and convert database schema. Converting the data into (city,1) pair-value and aggregate them together in the next step. Finally, we store into the local database.The detailed code in the photo 3.2.

city_name	city_count
Massachusetts	65214
California	197134
Florida	65552
Illinois	65607
Pennsylvania	65780
Texas	65495
Georgia	64940
New York	327524
Ohio	65668
NewJersey	65576

3. Visiting time count:

This shows the count of different visiting times. We want to find every customer visiting behaviour, which period time customers would like to browse our website. It helps us to display the number of product advertisements and adjust web server pressure. The code is the same as the previous two, picking up the time column and aggregate them. Eventually, we store the result into the database.

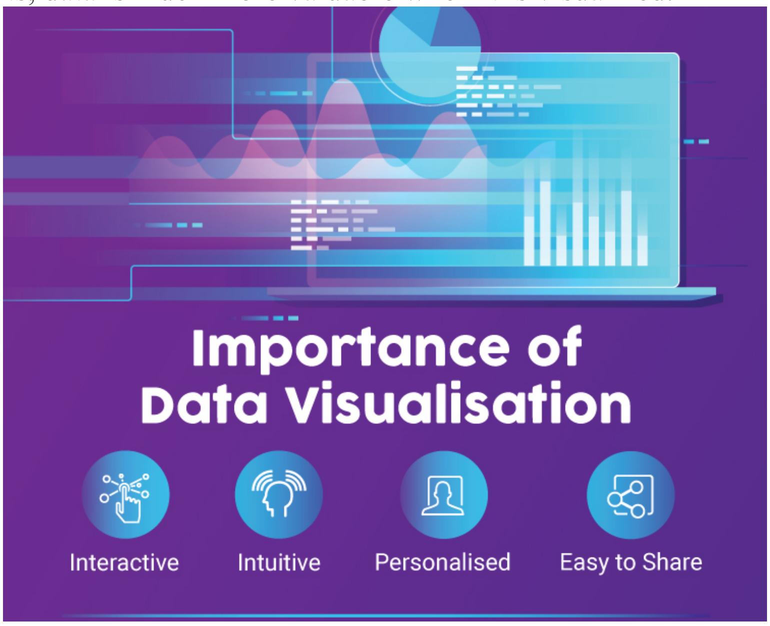
time	count
09	54892
16	50116
06	44255
14	52512
19	67140
12	60688
20	57474
15	47721
00	16455
21	35413
05	35554
02	18080
04	24124
18	67076
01	15577
17	61521
22	23909
23	14834
03	20467
13	55066
11	60797
07	50835
08	53175
10	60809

Data Visualization

Outcomes Visualization and Recommendation
Why visualization is important?

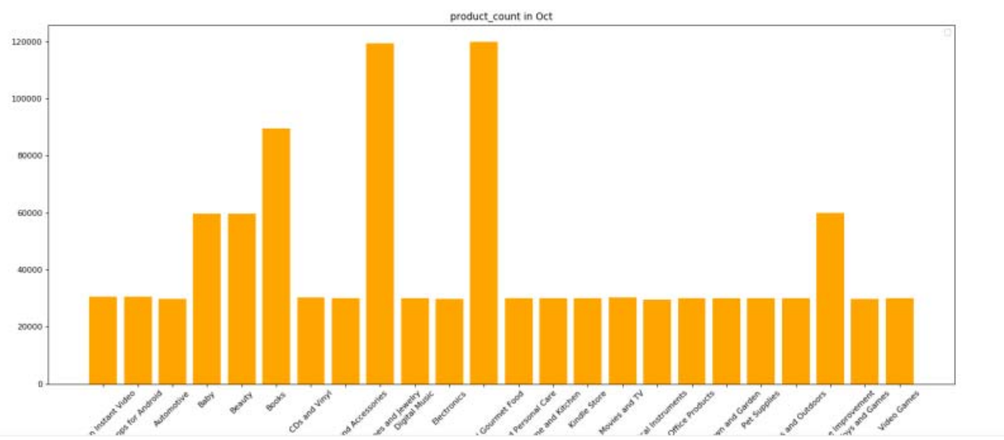
Data visualization is the representation of data or information in a graph, chart, or other visual format. It communicates relationships of the data with images. This is important because it allows trends and patterns to be more easily seen.

We need data visualization because a visual summary of information makes it easier to identify patterns and trends than looking through thousands of rows on a spreadsheet. It's the way the human brain works. Since the purpose of data analysis is to gain insights, data is much more valuable when it is visualized.



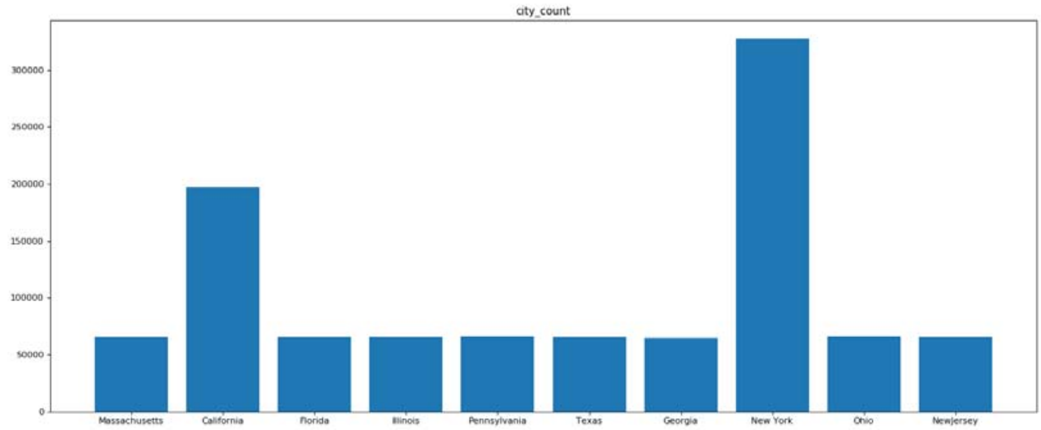
1. **Product number count**

In our visualization case, we use mataplotlib this API to implement our needs. Firstly, we need to connect our database by Python. And next step we convert our data format into Data-Frame and set x, y axis. Finally, we depict the result into bar chart. The code into photo 4.1.

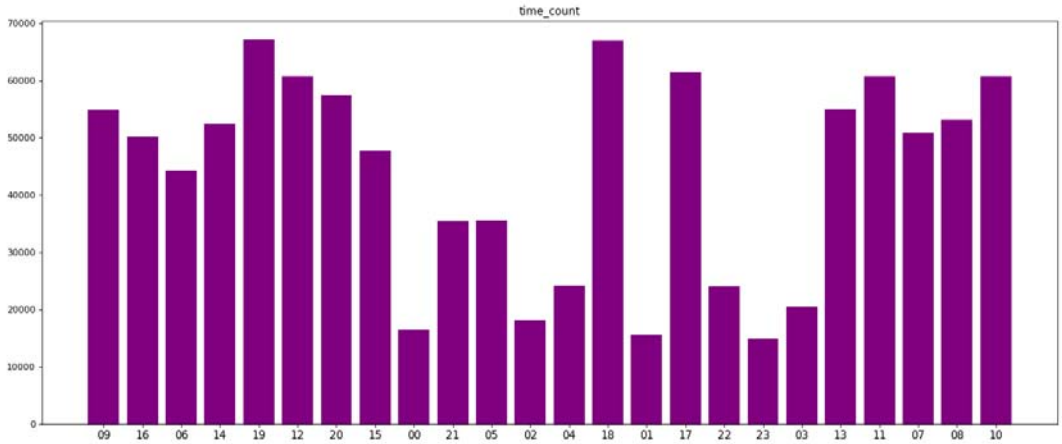


2. City number count

The second and third are also same steps in the first one. We set x, y columns and change the name of x and y. Finally, we compile the code and get the result. The code in the photo 4.2 and 4.3.



3. Visiting period count



Hypothesis prove:

We need to get another data-set and take the same steps in our last data to prove our hypothesis is right. We choose the customer records in Nov and prepare to compare the result between them.

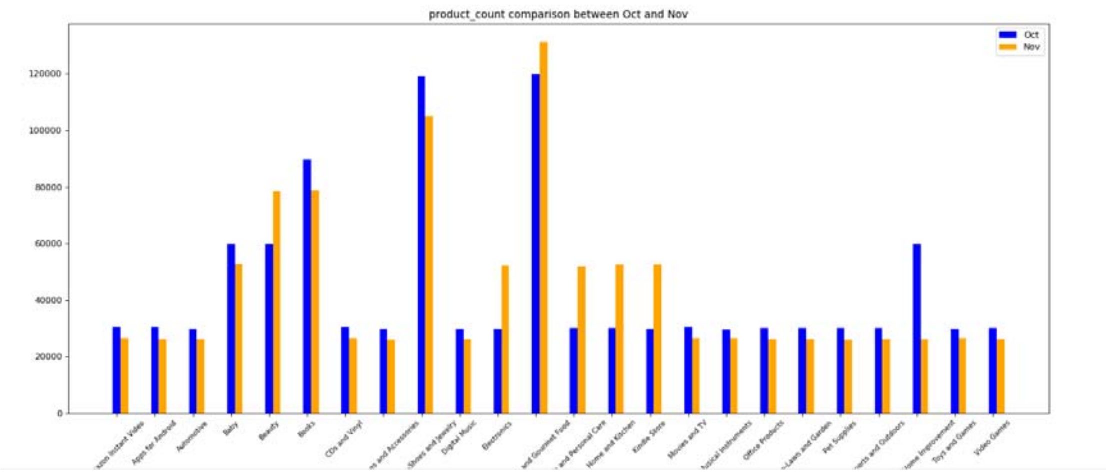
1. Product count hypothesis conclusion

October data-set

time	count
09	54892
16	50116
06	44255
14	52512
12	60688
19	67140
20	57474
02	18080
21	35413
15	47721
05	35554
17	61521
18	67076
00	16455
01	15577
13	55066
03	20467
07	50835
23	14834
04	24124
10	60809
22	23909
11	60797
08	53175

November data-set

time	count
06	44322
19	64627
14	53698
20	58822
21	37814
15	52070
02	10981
17	57570
18	63198
00	9501
13	58074
03	14792
04	21360
22	22175
07	55474
10	64127
11	61525
08	62244
09	62874
16	55423
12	61220
05	33008
01	10424
23	13034



Hypothesis 1 Conclusion:

Based on the last picture, we see the books, Clothes and food are the main source during October and November. Beauty and clothes products are also popular in the market. We should provide this result to the product departments. It helps them to explore more size, color, and features in these kinds of things.

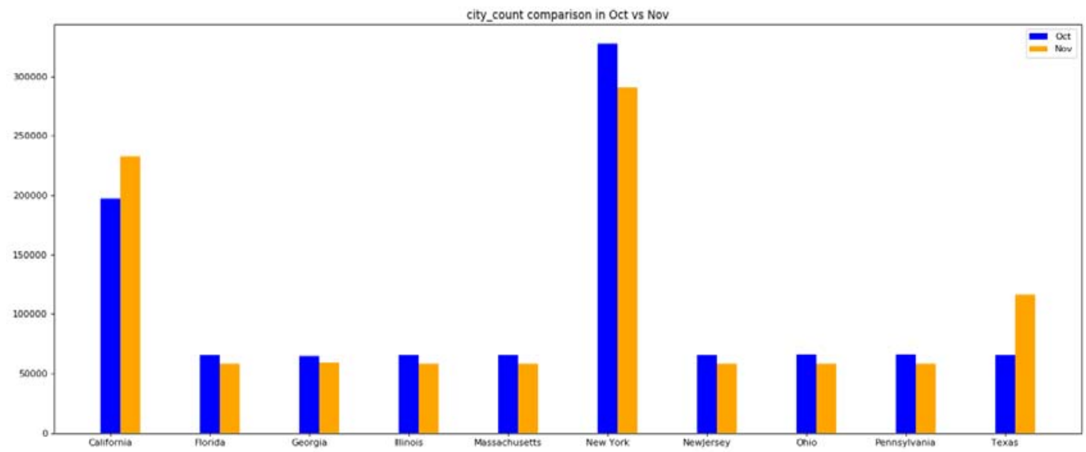
2. **City count hypothesis conclusion**

October data-set

city_name	city_count
Illinois	65607
New York	327524
Pennsylvania	65780
Massachusetts	65214
Texas	65495
NewJersey	65576
Ohio	65668
California	197134
Florida	65552
Georgia	64940

November data-set

city_name	city_count
Illinois	58202
Massachusetts	58465
Pennsylvania	58197
California	232639
Texas	116383
Florida	58194
Georgia	58821
New York	290953
NewJersey	58355
Ohio	58148



Hypothesis 2 Conclusion:

Base on the comparison between Oct and Nov. Our assumption about some big cities might have higher sales than others are correct. New York City and California present a higher customer's number. We should put more advertisements in these areas to find more potential customers.

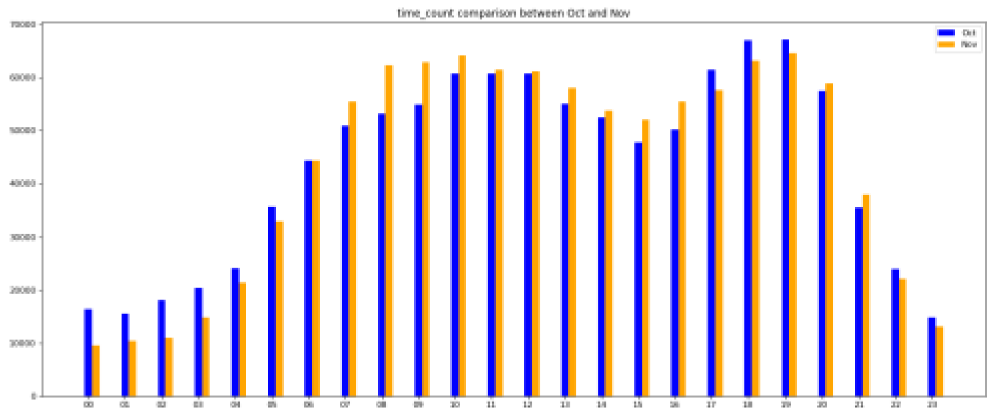
3. Time count hypothesis

Analytical Data Set

time	count
09	54892
16	50116
06	44255
14	52512
12	60688
19	67140
20	57474
02	18080
21	35413
15	47721
05	35554
17	61521
18	67076
00	16455
01	15577
13	55066
03	20467
07	50835
23	14834
04	24124
10	60809
22	23909
11	60797
08	53175

Production Data Set

time	count
06	44322
19	64627
14	53698
20	58822
21	37814
15	52070
02	10981
17	57570
18	63198
00	9501
13	58074
03	14792
04	21360
22	22175
07	55474
10	64127
11	61525
08	62244
09	62874
16	55423
12	61220
05	33008
01	10424
23	13034



Hypothesis 3 conclusions:

We guess customers would like to visit our website during the evening period than at night. It seems our hypothesis is right. We need to allocate our advertisement in the evening time to improve the ROI.

Predictive Analysis

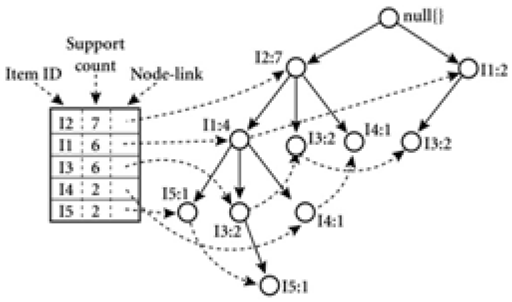
Predictive Analysis

In this Project. We can test the correlation between different items that customer buy. We need to group the product that different people purchase. After then, we want to find a high relationship in every product. So we set the confidence to filter the low relation item. Finally, the result is shown in Table 5. We can see some top relationship items. When the customer buys beef and cake, they also would buy some milk. Through the result, we can see the confidence between them is 0.94. But the beef, cake and strawberry, this confidence is 0.96. So we think when people have been bought beef and cake, they prefer to buy strawberry than milk. Thus, we can discuss with the product manager or market staff to make some strategies to increase the company profit. The code in the photo 5.1.

FP-Growth Algorithm:

1) In our project, we will use the FP-Growth algorithm to find the connection between products and customers as it has proved to be the most advanced and efficient implementation of frequent pattern mining.

2) The mining data is decomposed into sub-datasets according to the frequent patterns identified which leads to the more focused search of smaller databases.



Product relations in October

antecedent	consequent	confidence
Baby,Tools and Home Improvement	Grocery and Gourmet Food	0.8026745204170979
Baby,Beauty	Grocery and Gourmet Food	0.803071292170592
Tools and Home Improvement,Beauty	Grocery and Gourmet Food	0.8043753963221306
Baby,Tools and Home Improvement	Clothing-Shoes and Jewelry	0.8016397357319112

Product relations in November

antecedent	consequent	confidence
Health and Personal Care,Home and Kitchen	Grocery and Gourmet Food	0.9147086668387985
Home and Kitchen,Baby	Grocery and Gourmet Food	0.9139957716701903
Electronics,Home and Kitchen	Grocery and Gourmet Food	0.9137562974980787
Electronics,Kindle Store	Grocery and Gourmet Food	0.9136045120492223
Electronics,Baby	Grocery and Gourmet Food	0.9129659505816422
Kindle Store,Baby	Grocery and Gourmet Food	0.9120748299319728
Health and Personal Care,Baby	Grocery and Gourmet Food	0.9143756915482169
Health and Personal Care,Kindle Store	Grocery and Gourmet Food	0.911865864144454
Health and Personal Care,Electronics	Grocery and Gourmet Food	0.911787531258084
Kindle Store,Home and Kitchen	Grocery and Gourmet Food	0.9118048447628796

Conclusion

It's clear that baby and beauty stuff presents a stronger relation with grocery things in October records. Thus, we can display some beauty and baby products when customers search for baby products. And health products connect with the grocery foods in November records. As the same, we need to show more health related products when people visit grocery departments.

Future Application: Understand the customer behaviour

Recommendation systems can be a very powerful tool in a company's arsenal. Some of the applications include being able to anticipate seasonal purchases based on recommendations, determine important purchases, and give better recommendations to customers which can increase retention and brand loyalty.

Customer's behaviour is essential for company to find the success for its current product situation as well as new product launches. Understanding every customer's attitude and favor become a key role in the market.

Limitations

Missing Values : There are some missing values in the data set and we cannot take a few data to make some prediction.

Spreaded Data set: The city name and category name in the other document. We need to take lots of "join sentence" in SQL. It takes more time and has adverse effects on code efficiency.

Gender Data Missing: The data set doesn't show the user's gender. It causes a little inconvenience in some analysis that we will be performing.

Code Screen shot:

```
val spark = SparkSession.builder().appName( name = "practice").master( master = "local[4]").getOrCreate()
val sc = spark.sparkContext
val productCSV = spark.read.option("header",true).option("schema",true).csv( path = "D:\\logs\\input\\2019-Nov.csv")
val sqlContext = spark.sqlContext
```

1.1 Load data-set

```
val spark = SparkSession.builder().appName( name = "practice").master( master = "local[4]").getOrCreate()
val sc = spark.sparkContext
val productCSV = spark.read.option("header",true).option("schema",true).csv( path = "D:\\logs\\input\\2019-Oct.csv")
val sqlContext = spark.sqlContext

val productIdFilterRDD= productCSV.rdd.filter(row => {
  row.getString(3) != null && row.getString(7) != null && row.getString(6) != null
})
val productIdActionRDD : RDD[(String, Row)] = productIdFilterRDD.map(row => {
  (row.getString(3),row)
})
println("productIdActionRDD:"+productIdActionRDD.count())


val productInfoCSV = spark.read.option("head",true).csv( path = "D:\\logs\\input\\product_info.csv")
```

1.2 ETL data-set

```
val productCSV = spark.read.option("header",true).option("schema",true).csv( path = "D:\\logs\\input\\2019-Oct.csv")
val sqlContext = spark.sqlContext

import spark.implicits._
val data = productCSV.map(row => {
  val event_time = row.getString(0)
  val event_type = row.getString(1)
  val product_id = row.getString(2)
  val category_id = row.getString(3)
  val brand = row.getString(4)
  val price = row.getString(5)
  val user_id = row.getString(6)
  val user_session = row.getString(7)
  val city_id = row.getString(8)
  val value = "," + event_type + "," + product_id + "," + category_id + "," + brand + "," + price + "," +
    user_id + "," + user_session + "," + city_id
  (event_time, value)
})

data.rdd.saveAsHadoopFile( path = "D:\\logs\\input\\out",classOf[String],classOf[String],classOf[RDDMultipleClassPath])

spark.stop()
```

2.2-1 split data code

```
[root@slave1 ~]# cd apps/hive-1.2.1/bin
[root@slave1 bin]# ./hive

Logging initialized using configuration in jar:file:/root/apps/hive-1.2.1/lib/hive-common-1.2.1.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/root/apps/hadoop-2.7.3/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/root/apps/hbase-1.2.1/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hive> use is669project;
OK
Time taken: 1.429 seconds
hive> load data inpath '/is669project/2019-10-01.txt' into table customer_info partition(dt='2019-10-01');
Loading data to table is669project.customer_info partition (dt=2019-10-01)
Partition is669project.customer_info{dt=2019-10-01} stats: [numFiles=1, numRows=0, totalSize=14927365, rawDataSize=0]
OK
Time taken: 1.739 seconds
hive> select * from customer_info limit 2;
OK
2019-10-01 00:00:00 UTC         cart    5773203 149     runail  2.62    463240011       26dd6e6e-4dac-4778-8d2c-92e149dab885    6       2019-10-01
2019-10-01 00:00:03 UTC         cart    5773353 149     runail  2.62    463240011       26dd6e6e-4dac-4778-8d2c-92e149dab885    2       2019-10-01
Time taken: 0.437 seconds, Fetched: 2 row(s)
hive> load data inpath '/is669project/2019-10-02.txt' into table customer_info partition(dt='2019-10-02');
Loading data to table is669project.customer_info partition (dt=2019-10-02)
Partition is669project.customer_info{dt=2019-10-02} stats: [numFiles=1, numRows=0, totalSize=7402861, rawDataSize=0]
OK
Time taken: 0.661 seconds
hive> load data inpath '/is669project/2019-10-03.txt' into table customer_info partition(dt='2019-10-03');
Loading data to table is669project.customer_info partition (dt=2019-10-03)
Partition is669project.customer_info{dt=2019-10-03} stats: [numFiles=1, numRows=0, totalSize=13095874, rawDataSize=0]
OK
Time taken: 0.602 seconds
hive>
```

2.2-2 load data into cluster

```
[root@slave2 bin]# clear
[root@slave2 bin]# sqoop list-databases --connect jdbc:mysql://localhost:3306 --username root --password 123456
Warning: /root/apps/sqoop/bin/../../hcatalog does not exist! HCatalog jobs will fail.
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: /root/apps/sqoop/bin/../../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
Warning: /root/apps/sqoop/bin/../../zookeeper does not exist! Accumulo imports will fail.
Please set $ZOOKEEPER_HOME to the root of your Zookeeper installation.
20/04/16 16:27:49 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
20/04/16 16:27:49 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
20/04/16 16:27:49 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
information_schema
ebusiness
gmall
hive
mysql
performance_schema
spark_project_ebusiness
```

2.2-3 export data into database

```
val productInfoCSV = spark.read.option("head",true).csv( path = "D:\\logs\\input\\product_info.csv")
// TODO. First Part
/**
  * product Count
  */
val productInfoRDD = productInfoCSV.rdd.map(row => {
  (row.getString(0), row.getString(1))
})

val productInfoMapBroadcast = sc.broadcast(productInfoRDD.collectAsMap())
// val productInfoBroadcast = sc.broadcast(productInfoRDD)

val productAndOne = productInfoRDD.join(productIdActionRDD).map(tp => {
  (tp._2._1,1)
})

val productNumCountRDD: RDD[Row] = productAndOne.reduceByKey(_+_).map(tp => {
  val productName: String = tp._1
  val productNum: Int = tp._2

  Row(productName,productNum)
})
```

3.1 product count code

```
/**
  * city_count
  */
val cityInfoCSV = spark.read.option("header",true).csv( path = "D:\\logs\\input\\city_info.csv")
cityInfoCSV.show( numRows = 5, truncate = false)
val cityInfoMap: collection.Map[String, String] = cityInfoCSV.rdd.map(row => {
  val cityId = row.getString(0)
  val cityName = row.getString(1)
  println(cityName)
  (cityId, cityName)
}).collectAsMap()
val cityInfoMapBroadcast = sc.broadcast(cityInfoMap)

val cityIdAndNumActionRDD: RDD[Row] = productIdActionRDD.map(tp => {
  val cityInfoMap = cityInfoMapBroadcast.value
  val cityName = cityInfoMap(tp._2.getString(8))
  (cityName, 1)
}).reduceByKey(_ + _).map(tp => {
  Row(tp._1,tp._2)
})
```

3.2 City count code

```
//time_count
val timeActionRDD = productIdActionRDD.map(tp => {
  val time = tp._2.getString(0).split( regex = " ")(1).substring(0, 2)
  (time, 1)
}).reduceByKey(_+_).map(tp => {
  Row(tp._1,tp._2)
}).sortBy(_.getString(0))

val timeSchema = StructType(
  Array(
    StructField("time", StringType, true),
    StructField("count", IntegerType, true)
  )
)

sqlContext.createDataFrame(timeActionRDD,timeSchema)
  .write.mode( saveMode = "overwrite").jdbc(url, table = "time_count_table_copy",prop)
```

3.3 Time count code

```
sql2 = "SELECT * FROM product_count_table ORDER BY product_name"

cursor.execute(sql2)


product_count_data_oct = cursor.fetchall()


df_oct = pd.DataFrame(list(product_count_data_oct),columns=["city","product_count"])

x = df_oct.iloc[:,0]
y = df_oct.iloc[:,1]

plt.bar(x,y,color='orange')
plt.xticks(rotation=45)
plt.legend()
plt.title("product_count in Oct")
```

4.1 Product count visualization code

```
sql3 = "SELECT * FROM time_count_table ORDER BY time"
_sql3 = "SELECT * FROM time_count_table_copy ORDER BY time"
cursor.execute(sql3)
cursor2.execute(_sql3)

time_count_data_oct = cursor.fetchall()
time_count_data_nov = cursor2.fetchall()

df_oct = pd.DataFrame(list(time_count_data_oct),columns=["time","time_count"])
df_nov = pd.DataFrame(list(time_count_data_nov),columns=["time","time_count"])

time_count_data = pd.concat([df_oct,df_nov.iloc[:,1]],axis=1)

x3 = list(time_count_data.iloc[:,0])
y3 = list(time_count_data.iloc[:,1])
_y3 = list(time_count_data.iloc[:,2])

bar_width = 0.2
index = np.arange(len(x3))

plt.bar(index,y3,color='blue',width=0.2,label='Oct')
plt.bar(index + bar_width,_y3,color='orange',width=0.2,label='Nov')

plt.xticks(index,x3)
plt.legend()
plt.title("time_count comparison between Oct and Nov")
```

4.2 City count visualization code

```python
sql3 = "SELECT * FROM time_count_table ORDER BY time"

cursor.execute(sql3)


time_count_data_oct = cursor.fetchall()


df_oct = pd.DataFrame(list(time_count_data_oct),columns=["time","time_count"])

x3 = df_oct.iloc[:,0]
y3 = df_oct.iloc[:,1]


bar_width = 0.2
index = np.arange(len(x3))

plt.bar(index,y3,color='blue',width=0.2,label='Oct')


plt.xticks(index,x3)
plt.legend()
plt.title("time_count comparison between Oct ")
```

4.3 Time count visualization code

```scala
val productRDD: RDD[Array[String]] = userId2categoryNameRDD.filter(tp => {
  tp._2.split( regex = ",").length > 1
}).map(tp => {
  tp._2.split( regex = ",")
})


val fpgrowth = new FPGrowth().setMinSupport(0.2).setNumPartitions(4).run(productRDD)
val rulesRDD = fpgrowth.generateAssociationRules( confidence = 0.9)

val relationRDD: RDD[Row] = rulesRDD.filter(rule => {
  rule.antecedent.length <= 2 && rule.consequent.length == 1
}).map(rule => {
  val antecedent = rule.antecedent.mkString(",")
  val consequence = rule.consequent.mkString(",")
  val confidence = rule.confidence

  Row(antecedent, consequence, confidence)
})

println("relationRDD:"+relationRDD.count())

val relationSchema = StructType(Array(
  StructField("antecedent",StringType,true),
  StructField("consequent",StringType,true),
  StructField("confidence",DoubleType,true)
))
```

5.1 Product relations code