**Exploratory Data Analysis in Python, R and SQL**

**An Introduction to Data Analytics and Data Science**

By Dorcas C. Mbaeri

Spring 2018

In Partial Fulfillment of

Math 4395 - Senior Project

Department of Mathematics and Statistics

University of Houston-Downtown

Faculty Advisor:

Dr. J. Patrick King \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Committee Member:

Dr. Timothy Redl \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Committee Member:

Prof. Mitsue Nakamura \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Department Chairman:

Dr. Ryan Pepper \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table of Contents**

[Abstract 2](#_Toc512941553)

[Introduction: What Exploratory Data Analysis Means in Data Analytics and Science 3](#_Toc512941554)

[Using R for Data Exploration 5](#_Toc512941555)

[ggplot2 5](#_Toc512941556)

[R Markdown 6](#_Toc512941557)

[Essential Python Packages for Data Exploration and Analysis 7](#_Toc512941558)

[NumPy 7](#_Toc512941559)

[Pandas 7](#_Toc512941560)

[Matplotlib 8](#_Toc512941561)

[IPython and Jupyter Notebook 8](#_Toc512941562)

[Using SQL to Extract Data Insights 9](#_Toc512941563)

[Disadvantages of Relational Databases 12](#_Toc512941564)

[Advantages of Relational Databases 14](#_Toc512941565)

[Conclusion 15](#_Toc512941566)

[Works Cited 16](#_Toc512941567)

[Appendix A: E-Commerce Store (Zappos.com) Data Set Description 19](#_Toc512941568)

[Appendix B: Data Analysis Using Jupyter Notebook 20](#_Toc512941569)

[Appendix C: Employing R Markdown for Data Analysis 21](#_Toc512941570)

[Appendix D: Transact-SQL Queries and Explanation on e-Commerce Store Database 22](#_Toc512941571)

# **Abstract**

With a surge of big data analytics in the past few decades, acquiring skills to draw “deep” understanding of big data in various domains is a crucial aspect of a being a data scientist or big data analyst. Every data analysis process begins with exploration: visualizing the dimensions of the data, its structure, missing values, the data types of each field or column and more. As this is the first step, it is a vital and foundational exploration that affects the results and insights extracted from it. In this research, we discuss a few mechanisms to accomplish this data exploration process as well as evaluating their advantages and disadvantages. Depending on the objective of the analysis, it is better to use Python or R for visualization to note any patterns or trends. On the other hand, using a relational database - if there are multiple data sets to get numerical or descriptive data - is very flexible compared to using Python or R but could lag in performance. A demonstration of exploratory data analysis is included in the Appendices for Python (Jupyter Notebook), R (R Markdown) and SQL (SQL Server) to explore a data set from an e-commerce store, Zappos.

Dorcas Mbaeri

Dr. Patrick King (Faculty Adviser)

Math 4395 - Senior Project

April 30, 2018

# **Introduction: What Exploratory Data Analysis Means in Data Analytics and Science**

Exploratory data analysis, or EDA, was said to have been coined by John W. Tukey in 1977 when he wrote the book “Exploratory Data Analysis” and developed the field. EDA is an essential step after acquiring data and before generating any model. This focus of this step or approach is to get an idea of how a specific data analysis problem should be carried out. According to the National Institute of Standards and Technology, EDA approach includes various techniques - mostly graphical - to: maximize insight into a data set, uncover underlying structure, extract important variables, detect outliers or anomalies, test underlying assumptions, develop parsimonious models, and determine optimal factor setting.

Although EDA employs statistical graphics, it is a philosophical approach to how a data set should be dissected, what to look for, how we look for it, and how we interpret our findings. This approach is a key component to any data science endeavor before diving into machine learning or statistical modeling. Depending on the type of data we have, EDA techniques may differ. There are generally two types of data: structured and unstructured. Structured data has a high degree or organization, such as numeric or categorical data. Some examples are phone numbers or gender. Unstructured data, however, simply does not fall under the structured form of data that we are used to. Some examples of unstructured data are photos, images, audio, language text, etc. Deep learning - a subset or approach to machine learning - is an emerging field that uses specialized sets of algorithms, such as decision trees, Bayesian networks, clustering and reinforcement learning to work well with unstructured data. With categorical structured data, a frequency table/ plot, pie charts, or bar charts are a few common statistics to describe each categorical variable (Andrade). Numeric variables can be visualized using boxplots, bar charts or histograms. Furthermore, employing summary or descriptive statistics could be very useful for univariate, bivariate and multivariate visualizations. Summary statistics include mean, median, mode, maximum, minimum, range, quartiles/ percentiles, variance, standard deviation, coefficient of determination, skewness and kurtosis.

In addition, there are often two types of assumptions that affect the validity of data analysis: technical and business (Mawer). During EDA, various technical assumptions are assessed to help determine appropriate analytical models and algorithms. Some of these assumptions could include no collinearity between variables, variance in the data that is independent of the data’s value and missing or corrupted data. On the contrary, business assumptions are a bit more elusive and intuitive than technical assumptions. Such assumption can be deeply entangled with the problem and how it is framed. Validating both types of assumptions are vital, otherwise it will result in poor predictions and incorrect conclusions. After assessing and validating them, data analysts and scientists alike can get a bigger view of the data set and how each field interact with each other.

# **Using R for Data Exploration**

R is an open-source statistical programming - and interpreted - language that originates from another language called S, which was developed by John Chambers and colleagues at Bell Laboratories in the mid-1970s. S, and ultimately R, was designed for statisticians, data analyst, and data scientist to perform data analysis. “Neither R nor S not statistically-oriented programming languages in general are encountered frequently outside the cloisters of data science; within those precincts, R has become the hot analytical programming too of choice for data scientists in every industry from insurance to banking to marketing to pharmaceutical development,” (Understanding How R Is Used in Data Science). From doing standard statistical analysis to advanced algorithms like structural equation model, random forests and penalized regression, R is the first and most versatile tool well-suited for such analysis. The statistical programming language offers a variety of features to data scientists, including linear and non-linear modeling, time-series analysis, clustering, extensibility and interfaces to other programming languages and sizeable shared code package repository. Furthermore, RStudio is the most popular and preferred Integrated Development Environment (IDE) of R. It is also accessible from several scripting languages including Python.

## **ggplot2**

ggplot2 is a powerful data visualization package in R. It is based on the Grammar of Graphics (by Leland Wilkinson) and was created by Hadley Wickham. ggplot2 allows users to create graphics that represent both univariate and multivariate numerical and categorical data in a simple manner (Kabacoff). Grouping visualized using colors, symbol, size and transparency. The components of a ggplot2 graph include: data, aesthetic mapping, geometric object, statistical transformations, scales, coordinate system, position adjustments and faceting. Compared to basic graphs, ggplot2 is more verbose for simple/canned graphics, less verbose for complex/custom graphics, does not have methods and uses a different system for adding plot elements (R Graphics).

## **R Markdown**

R Markdown is the R version of markdown that allows for easy-to-read, easy-to-write plain text format. Some benefits of using R Markdown are: output to HTML (can be used for online publication), PDF or Word document, interactive documents, version control, integrated with R Studio, and free web publishing through R Pubs.

# **Essential Python Packages for Data Exploration and Analysis**

Python is a scripting-programming language that is very useful for data analysis and interactive, exploratory computing and data visualization. Python is also an open-source programing that has gained popularity among its many users because of its readability that allows beginners to learn. Within the data science community, Python is quite powerful because of the large and active ecosystem of third-party packages including: Numpy for manipulation of homogenous array-based data, Pandas for manipulation of heterogeneous and labeled data, Matplotlib for publication-quality visualizations, and IPython and Jupyter Notebook for interactive execution and sharing of code (Vanderplas).

## **NumPy**

Numerical Python, or NumPy for short, is the foundational package for scientific computing in Python. It is built on numerical arrays, which allows for more efficiency of data storage and manipulation of dynamically growing data. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python (Vanderplas). Among its many uses, NumPy provides functions for performing element-wise computations with arrays or mathematical operations between arrays, tools for reading and writing array-based data sets to disk, a fast and efficient multi-dimensional array object ***ndarray***, linear algebra operations, Fourier transformation and random number generation, and tools for integrating C, C++, and Fortran code to Python (McKinney).

## **Pandas**

Pandas was built on top of NumPy and provides a fast, easy and expressive way to work with rich data. Its primary object is the **DataFrame**, which is a two-dimensional tabular, column-oriented data structure with row and column labels. This object also works well with missing values, grouping data, pivots, and more. It offers sophisticated indexing for reshaping, slicing and dicing the data set, and selecting subsets of the data.

## **Matplotlib**

Matplotlib is the most popular data visualization package in Python. It was created by John D. Hunter in 2002 as a patch to IPython for enabling interactive MATLAB-style plotting via gnu plot from the IPython command line. Its integration with IPython offers a comfortable interactive environment for plotting and exploring data (McKinney).

## **IPython and Jupyter Notebook**

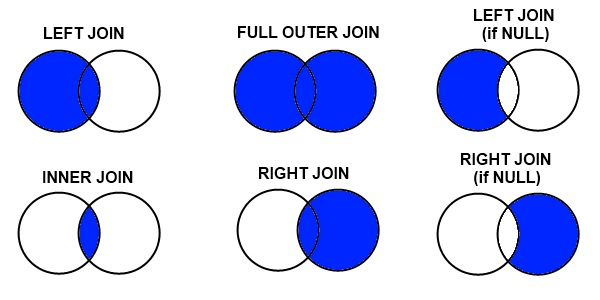
IPython is an interactive interface to Python. It is an enhanced Python shell designed to accelerate the writing, testing and debugging of Python code. Jupyter Notebook is an extension of IPython shell. It provides a browser-based notebook that is beneficial for development, collaboration, sharing, and publication of data science results. It is also especially useful when incorporating data analysis with visualization from matplotlib plots.

# **Using SQL to Extract Data Insights**

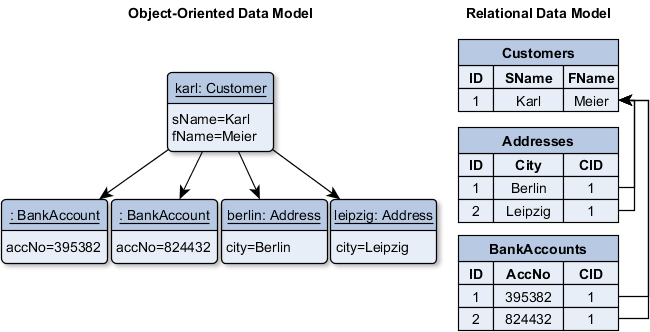
In this era of big data, databases are very powerful storage units to store and analyze data. There are three typical types of databases: hierarchical, network and relational. Hierarchical or network databases are usually designed for a specific application. In contrast, relational databases can have very complex (or simple) relationships between tables. Furthermore, this type of database does not rely on any application: they can be modified without affecting the application(s) they are connected to. For this reason, relational databases have become the most common storage mechanism (Hung).

Structured Query Language, or SQL for short, is a powerful programming language that is commonly used to communicate with relational database (and non-relational databases). It allows analysts and developers alike to add, delete, extract, modify, or operate on any information within the relational database. In addition, SQL can be used to perform complex analytics and even change the structure of the database itself. There are several types or “flavors” of SQL, including PostgreSQL, Procedural Language/ Structured Query Language (PL/SQL), Transact- SQL (T-SQL) and the ANSI standard SQL. Microsoft SQL uses T-SQL and the database application of this project will be done using SQL Server to store the e-commerce store customer transaction and T-SQL to extract useful information.

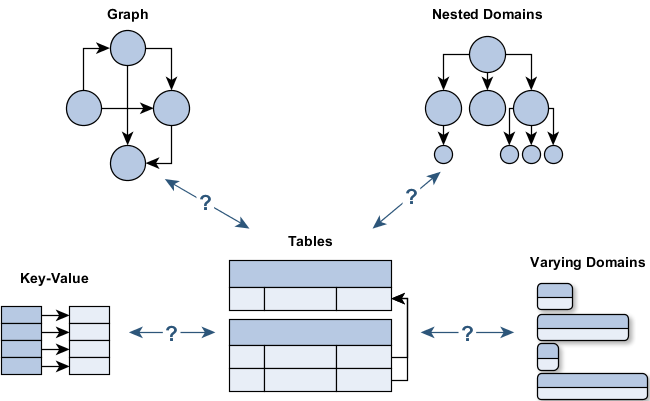
SQL commands or queries can provide insights to our data and the relations between data tables. While SQL is a programming language, it is not the conventional programming languages, such as Java, C/C++, Python or R. SQL statements specify WHAT data operations should be performed but not HOW to perform them. Whereas more conventional programming languages like Python scripts uses its interpreter to execute each line of code. SQL commands can be more concise and save more time than these languages. They are also efficient in that they do not require a lot of code to do advanced querying. Some powerful frameworks we implement and use to study our data include:

* **Projection**: extract columns of interest within the table using the SELECT clause
* **Filters**: specifies rows of interest using the WHERE clause
* **Joins**: combines two tables based on a shared key attributes in these tables and involves number of ways to join them
  + *Inner*: return all rows where here is at least one match in both tables
  + *Full Outer*: return all rows from both tables with matching rows from both sides, where available. If there is no match, the missing side will contain null.
  + *Outer Left*: returns all rows from left table and matched rows from right table; if there are none, the missing side will be null.
  + *Outer Right*: returns all rows from right table and match rows from left table; if left table has no match, it will display null.
  + *Cross*: returns a Cartesian product of the tables, i.e. all rows from left table for each row in the right table.
* **Aggregate**: includes several aggregate functions and operations that summarizes the dataset in detail
  + SUM
  + COUNT
  + MAX
  + MIN
  + AVG (average)
  + GROUP BY (groups rows into specific categories when applying aggregate functions)
  + HAVING (filters results based on calculations in the SELECT clause and/or aggregations derived from the GROUP BY processing)

## **Disadvantages of Relational Databases**

* One drawback to using relational databases lies in its incapability to structure a database model as an object. In an object-oriented perspective, an entity or object can have multiple sub-objects (underlying objects) and have a connection via references (Hauer). However, with relational databases, this model will have to be spliced and flattened into multiple “normalized” tables. This can lead to complex queries of joining several tables and thus performance issues if we try to extract an object.

In the above model, Customer Karl has references to two BankAccounts objects and two Address objects. In the schema (right), there are three tables for each class (Customers, Addresses and BankAccounts) and each table is filled with the corresponding data. Moreover, the entries of the addresses and banks accounts have a foreign key pointing to the entry in the Customer table. This schema relation is quite an opposite from the original object model. The data distribution over several tables gets even more complicated when intermediate tables are necessary.

* This leads to another disadvantage: some domains simply cannot be mapped to a relational model. Depending on the domain model, if the model is a highly-cross linked graph or deeply nested domain models or hierarchies (*image to the right*), it will be extremely difficult - nearly impossible - to implement a relational database to fully traverse the graph, which will require large number of necessary joins.
* Because the database schema is normalized, multiple joins are necessary to obtain the needed data. But, joins are expensive. They can be messy, confusing and can disrupt the structure of the database if implemented in stored procedures, views, etc. Denormalizing the schema will decrease the number of needed joins, which will thus improve the performance. However, this can also lead to redundancy and danger of inconsistency, which must be addressed at the application layer.

## **Advantages of Relational Databases**

* Relational databases ensure that commits are atomic, consistent, isolated and durable. If an error occurs, the whole transaction can be rolled back, restoring a consistent state. This is important in certain domains like banking and e-commerce.
* They also ensure safety due to defined schema and strict constraint checks (e.g. columns types, uniqueness constraints, referential integrity). The data has a fixed structure. This means that you cannot accidentally misspell a column name, submit a string instead of an integer or enter non-existing foreign key. The database will refuse such queries. However, data validation and consistent checking often must be done in the application or user interface layer. Complex business validation cannot be done in the database, which is why it questionable if a redundant check done by the database is necessary due to maintenance effort and performance problems.
* There are plenty of toolkits available: triggers, stored procedures, advanced indexes, views and more.

# **Conclusion**

For every data scientist or data analyst, data exploration is necessary to avoid making inaccurate predictions and developing models or algorithms that are not appropriate. Assumption without validation is dangerous to the field of predictive analytics. Depending on the type of data, its structure and the goal of the problem, the tool/ software used may differ. Python and R are quite similar in that they offer a diverse and interactive interface that is beneficial to data scientists for sharing of code or insights. However, Python is much easier to learn and has a huge user platform which enables the development of its rich packages, such as pandas and NumPy. R is older than Python and has been established for decades as the go-to statistical programming language for various business/ industry application. Though this may strengthen its reputation, it might also be a hindrance as R is currently not able to handle large data sets. Relational databases, such as SQL Server provides a secure and reliable access to big data. Yet, it may not be idea for object-oriented data model and its performance could be a serious problem to businesses if there are complex schemas present.

# **Works Cited**

*NIST/SEMATECH e-Handbook of Statistical Methods*, National Institute of Standards and Technology, 1 June 2003, www.itl.nist.gov/div898/handbook/.

Andrade, Andrew. “Exploratory Data Analysis.” *Exploratory Data Analysis*, Data Science Guide, datascienceguide.github.io/exploratory-data-analysis.

Halilovic, Inge. “Markdown for Jupyter Notebooks Cheatsheet – IBM Watson Data – Medium.” *Medium*, IBM Watson Data, 7 Feb. 2017, medium.com/ibm-data-science-experience/markdown-for-jupyter-notebooks-cheatsheet-386c05aeebed.

Hauer, Philipp. “Why Relational Databases Are Not the Cure-All. Strength and Weaknesses.” *Philipp Hauer's Blog*, 14 May 2015, blog.philipphauer.de/relational-databases-strength-weaknesses-mongodb/.

AiO. “Housing Data Exploratory Analysis.” *Detailed Exploratory Data Analysis Using R*, Kaggle, 26 Sept. 2016, www.kaggle.com/notaapple/detailed-exploratory-data-analysis-using-r.

Hung, Jenny. “4 Reasons Aspiring Data Scientists Must Learn SQL - Vertabelo Academy.” *Vertabelo Academy Blog*, Vertabelo Academy, 28 Aug. 2017, academy.vertabelo.com/blog/four-reasons-aspiring-data-scientists-must-learn-sql/.

Jeevan, Manu. “Exploratory Data Analysis in Python Using Pandas, Matplotlib and Numpy.” *Big Data Made Simple - One Source. Many Perspectives.*, Crayon Data, 4 Jan. 2016, 10:56, bigdata-madesimple.com/exploratory-data-analysis-in-python-using-pandas-matplotlib-and-numpy/.

Kabacoff, Robert. “Graphics with ggplot2.” *Quick-R: ggplot2 Graphs*, DataCamp, www.statmethods.net/advgraphs/ggplot2.html.

Mawer, Chloe. “The Value of Exploratory Data Analysis.” *Silicon Valley Data Science*, Silicon Valley Data Science LLC, 19 Sept. 2017, www.svds.com/value-exploratory-data-analysis/.

McKinney, Wes. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. OReilly, 2013, books.google.com/books?hl=en&lr=&id=v3n4\_AK8vu0C&oi=fnd&pg=PR3&dq=how is python used in exploratory data analysis&ots=rgFIbotzsD&sig=ZkZa0onNw74hYx4dF1uD11XBirA#v=onepage&q&f=false.

*R Graphics*. tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html.

Siddiqi, Adnan. “Introduction to Exploratory Data Analysis in Python.” *Medium*, Python Pandemonium, 3 Mar. 2018, medium.com/python-pandemonium/introduction-to-exploratory-data-analysis-in-python-8b6bcb55c190.

“SQL - The Natural Language for Analysis.” *Oracle White Paper*, Oracle Database, June 2015, www.oracle.com/technetwork/database/bi-datawarehousing/wp-sqlnaturallanguageanalysis-2431343.pdf.

“Understanding How R Is Used in Data Science.” *Data Science Masters Degree Programs*, www.datasciencegraduateprograms.com/data-science-with-r/.

Vanderplas, Jacob T. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly, 2017.

Wickham, Hadley, and Garrett Grolemund. *R For Data Science: Import, Tidy, Transform, Visualize and Model Data*. OReilly, 2017.

# Appendix A: E-Commerce Store (Zappos.com) Data Set Description

**Summary**

The data used in this research project was acquired from Zappos.com, an e-commerce clothing and shoe store. This a structured data set in an Excel spreadsheet and contains 21,061 customer transactions from January 2013 - December 2013. Some of the fields captured in this data set includes the date of each transaction, the sites visited under the Zappos general website umbrella, the number of visits that occurred in one session (one sitting), the technology device used, such as iPhone, Android, Windows, BlackBerry, etc., the amount the customer spent on each session and each visit, and much more (both categorical and numerical variables). There are no personal customer information present in this data. Any information that can be used to trace back to a customer is disregarded. However, there is an identifier field named *new\_customer* that differentiates a completely new customer (recently registered or opened an account) from a returning customer (already has an account) and from a guest/visitor (has no account and did not open one at the occurrence of the transaction). Below is a detailed description of all fifteen columns in the dataset from Zappos.com.

***day***

The *day* column represents the date and time of the transaction. The dates are in the “MM/DD/YYYY 12:00:00 AM” format and all transactions were for the year 2013. This field has no blanks:

* 2366 transactions were on January 2013;
* 2137 were on February 2013;
* 2327 transactions occurred on June 2013;
* 2035 were on July 2013;
* 2462 were on August 2013;
* 2347 transactions were captured on September 2013;
* 2464 were on October 2013;
* 2389 were on November 2013; and
* 2534 transactions happened on December 2013

As we can see, only nine months’ worth of customer transaction were collected in this dataset. There are no data for the months of March, April and May. From the above, December 2013 has the largest number of customer transactions followed by October.

***site***

The *site* column is the company’s site visited by users and - in the scope of this dataset - they include Acme, Botly, Pinnacle, Sortly, Tabular, and Widgetry. There are no blanks in this field:

* 7392 users visited the Acme site;
* 804 visited Botly;
* 5725 visited Pinnacle;
* 5532 users visited the Sortly site;
* 804 visited Tabular; and
* 804 visited the Widgetry site

Majority of the users visited the Acme, Pinnacle and Sortly site - in that order.

***new\_customer***

This column identifies if a user is a new customer, returning customer or neither. New customers are assigned the data value 1, returning customers are assigned the value 0, while neither are null (blanks):

* 7066 users are returning customers (0);
* 5736 users are new customers (1); and
* 8259 users are neither (null)

Majority of the users were either new or returning, which is a good thing when we delve deeper into finding correlation between several fields.

***platform***

This column contains the type of device (Android, iPhone, iOS, Windows, etc.) the users use when navigating through the Zappos website to search or buy a product:

* 3172 users were on Android devices;
* 1589 users used a Blackberry device;
* 1349 users were on ChromeOS;
* 3435 users were on iOS devices;
* 459 users were on iPad devices;
* 468 users were on iPhone devices;
* 2036 users used Linux devices;
* 333 users used Macintosh devices;
* 2054 users were on MacOSX devices;
* 327 devices used were Other;
* 74 users were on SymbianOS;
* 1641 devices used were Unknown;
* 2399 devices used were Windows;
* 1315 devices used were WindowsPhone;
* 410 of the devices were blanks

***visits***

The *visits* are integers that represent the number of distinct website visits. Distinct visits mean the different pages or sites a user looks through in Zappos.com. For instance, a user could be viewing a product or several products under the Sortly site but another product(s) under the Widgetry site. These would be considered distinct visits. The number of visits in this data goes from 0 to 136057.

***distinct\_sessions***

The number of distinct session (integer) is how many times a user surfs through the website without leaving desk or in one sitting. One distinct session may have multiple sites. The data values range from 0 to 107104.

***orders***

This is the number of website orders (integer) made by each user (customer transactions). The values range from 0 to 4916.

***gross\_sales***

This field captures the total gross sales for the website orders. A user may not have placed order or bought anything, or they may have placed an order and did not buy it in that session: 9576 fields are blanks.

***bounces***

This is an integer column that refers to the number of visits that only viewed one page. The *bounces* in this data set ranges from 0 to 52598.

***add\_to\_cart***

This numerical column is the number of visits that added a product to cart. Ideally, all visits should end with a purchase or an item in the cart at least. The range of this data values are 0 to 3966.

***product\_page\_views***

This numerical data values are the number of product pages views per session and per visit. The values range from 0 to 146866.

***search\_page\_views***

This is the number of search pages viewed per visit per session.

***conversion\_rate***

This is a calculated measure =’orders’/’visits’

***bounce\_rate***

This is a calculated measure = ‘bounces’/’visits’

***add\_to\_cart\_rate***

This is a calculated measure =’add\_to\_cart’/’visits’

# 

# Appendix B: Data Analysis Using Jupyter Notebook

# Appendix C: Employing R Markdown for Data Analysis

# 

# Appendix D: Transact-SQL Queries and Explanation on e-Commerce Store Database

**Note**: *Unless used in a query, all column names are italicized while the table name is boldened.*

* The below script creates the **myData** table. Here, we specify the names of the fields, the type of values each column carries, the maximum amount characters for the data size and if it accepts null values:

/\*\*\*\*\*\* Object: Table [dbo].[myData] Script Date: 4/2/2018 6:23:44 PM \*\*\*\*\*\*/

CREATE TABLE [dbo].[myData](

[day\_time] [datetime] NULL, /\*equivalent to day in original table\*/

[visit\_site] [nvarchar](255) NULL, /\*equivalent to site in original table\*/

[new\_customer] [float] NULL,

[user\_platform] [nvarchar](255) NULL, /\*equivalent to platform in original table\*/

[visits] [float] NULL,

[distinct\_sessions] [float] NULL,

[orders] [float] NULL,

[gross\_sales] [float] NULL,

[bounces] [float] NULL,

[add\_to\_cart] [float] NULL,

[product\_page\_views] [float] NULL,

[search\_page\_views] [float] NULL,

[conversion\_rate] [float] NULL,

[bounce\_rate] [float] NULL,

[add\_to\_cart\_rate] [float] NULL

) ON [PRIMARY] GO

* Let’s say we’d like to know the average orders and gross sales by *new\_customer*, *user\_platform* and *visit\_site*. We can also filter out blanks in these fields:

USE [Zappos\_Data]

SELECT day\_time, visit\_site, new\_customer, user\_platform, AVG(orders) AS Average\_Orders,

AVG(gross\_sales) AS Average\_GrossSales

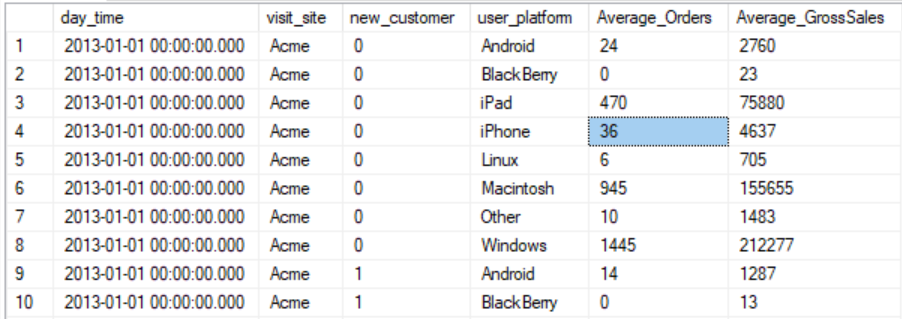
FROM dbo.myData

WHERE new\_customer IS NOT NULL AND

user\_platform IS NOT NULL

GROUP BY day\_time, visit\_site, new\_customer, user\_platform

ORDER BY day\_time, visit\_site, new\_customer, user\_platform

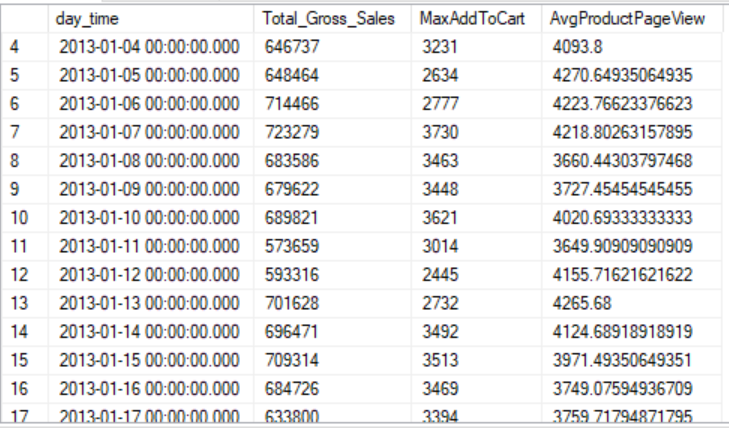
In the above query, we specified in the WHERE clause what we’d like to find first before grouping the aggregate function (average) by the three fields listed and then ordered in that order. This query indicates which platform and site is most popular in our data set. Though this may not necessarily have much impact in how we can improve sales, if we factor in the type of customer who bought items on certain sites using a specific platform, we can make recommendations on products that they customer may like based on where they shopped. The results of the query show that returning customers who used Windows and visited the Acme site had the greatest average gross sales ($707,642.00) and average orders of 4916. Acme also turns out to be the most popular site with 2357 transactions, while Widgetry and Tabular each had only 268 transactions.

* The total *gross\_sales*, maximum *add\_to\_cart* and the average *product\_page\_views* grouped and ordered by day in ascending order.

SELECT day\_time, SUM(gross\_sales) Total\_Gross\_Sales, MAX(add\_to\_cart) AS MaxAddToCart, Avg(product\_page\_views) AvgProductPageView

FROM [dbo].[myData]

GROUP BY day\_time

ORDER BY day\_time ASC /\*Ascending is the default order\*/

* We can specifically look at the number of sales in a month. For example, the below query extracts the total number of sales for the months of November and December as well as their respective total sales amount. Combined, the two months saw total of 2749 sale transactions with a gross amount of $58,388,210.00. Now, we can assume that December will see the higher sales of the two months. By filtering when the MONTH(day\_time) = 12 (or 11 for November) in our sub-query (sub), we find that there was a total of 1430 sales amounting to $34,235,990.00. Whereas in November, there was 1319 transactions which equated to $24,152,220.00 gross sales.

SELECT COUNT(\*) NumOfSales, SUM(gross\_sales) AS TotalSalesAmount

FROM (

SELECT day\_time, gross\_sales

FROM dbo.myData

WHERE (DATEPART(MONTH, day\_time) = 11 OR MONTH(day\_time) = 12) AND gross\_sales <> ''

) sub

* Let us look at the top 10 total bounces by the new\_customer, user\_platform and visit\_site. We will compare the total bounces to the total number of products viewed, how many of those products were added to cart and were ultimately ordered.

SELECT TOP 10 new\_customer, user\_platform, visit\_site,

SUM(product\_page\_views) TotalProductsViewed, SUM(orders) AS TotalOrder, SUM(bounces) TotalBounces, SUM(add\_to\_cart) TotalAdddToCart

FROM dbo.myData

WHERE new\_customer IS NOT NULL AND conversion\_rate <> '' AND bounce\_rate IS NOT NULL AND add\_to\_cart\_rate <> ''

GROUP BY new\_customer, user\_platform, visit\_site

ORDER BY TotalBounces DESC

From the above table result, not only did Acme have the most product views and the total number of bounces - the number of visits that only viewed one page - but it also had the most orders and products added to cart across all platforms used.

We can further look at the average *search\_page\_views*, *bounces*, *add\_to\_cart* and *distinct\_sessions* for each *site* and *platform* to see if there are any trends in the most popular site and, depending on the platform, which is more mobile-friendly.

SELECT visit\_site, user\_platform, CEILING(AVG(search\_page\_views)) AS Average\_Page\_Views, CEILING(AVG(bounces)) AS Average\_Bounces,

CEILING(AVG(add\_to\_cart)) AS Average\_Add\_to\_Cart

FROM dbo.myData

WHERE user\_platform IS NOT NULL

GROUP BY visit\_site, user\_platform

ORDER BY visit\_site, user\_platform

From the table below, we notice that the *site* Botly has only been visited using an Android *platform*, Widgetry was only visited using an iPhone or iOS device (laptop/computer) and most customers who visited Acme, Pinnacle, Widgetry were Windows (non-cellular) and iOS users. Another interesting thing to note is that though Widgetry had an estimated average of thirty-eight thousand users search the site, only less than four hundred items were added to cart. And though Pinnacle seems to have users across all platforms, its average search views, bounces and cart size is nothing in comparison to Tabular. We can also assume that Tabular, Botly and Widgetry are apps for the e-commerce store. In addition, from this query, we can conclude that Botly and Pinnacle are the least visited sites by all users in this database.



* Since our data is just one table in SQL Server, we can attempt a simple join to find aggregate calculations by month, day of week, or day of the month. This will help us looking at the data from a date view. In the below queries, we extracted the year, month, week, day of the week (1-7) and day (1-31) and added these new attributes/columns to the **myData** table.

--Add the new queries to the myData table

ALTER TABLE dbo.myData

ADD my\_year AS (YEAR(day\_time));

ALTER TABLE dbo.myData

ADD data\_month AS (MONTH(day\_time));

ALTER TABLE dbo.myData

ADD data\_week AS (DATEPART(WEEK, day\_time));

ALTER TABLE dbo.myData

ADD day\_of\_week AS (DATEPART(DW,day\_time));

ALTER TABLE dbo.myData

ADD data\_day AS (DATEPART(day, day\_time));

-- Let's check to see if we successfully added the new columns to the table

SELECT my\_year, data\_month, data\_week, data\_day, day\_of\_week

FROM dbo.myData

The SELECT statement outputs the new fields created and the values they hold. This data was simply obtained from the **myData** table. Now, we will create a new table called **Transaction\_Date** to be a small normalized table of the date of all transactions in **myData**. The reason we are creating this new table is to demonstrate that if we had a normalized data model, with multiple tables, we would have to perform some SQL functions that are higher-level, such as a JOIN or sub-querying. We will show examples of this very powerful query using the JOIN. But first, our new table is created:

SELECT my\_year, data\_month, data\_week, data\_day, day\_of\_week

INTO dbo.Transaction\_Date

FROM dbo.myData

GROUP BY my\_year, data\_month, data\_week, data\_day, day\_of\_week

ORDER BY my\_year, data\_month, data\_week, data\_day, day\_of\_week

SELECT \*

FROM dbo.Transaction\_Date

The table above is a snippet of the second SELECT statement that gets all the fields in the **Transaction\_Date** table. To successfully extract this data to populate our new data table, we found the date information, such as the year, month, week of the year, day of the week and day using the DATEPART functions.

* Here we will find the total and average sales and orders by month.

SELECT td.data\_month, SUM(orders) Total\_Orders, ROUND(AVG(orders), 2) Average\_Orders, SUM(gross\_sales) TotalGross\_Sales,

ROUND(AVG(gross\_sales), 2) AverageGross\_Sales

FROM dbo.myData m INNER JOIN dbo.Transaction\_Date td

ON m.data\_month = td.data\_month

WHERE orders IS NOT NULL AND gross\_sales IS NOT NULL

GROUP BY td.data\_month

ORDER BY td.data\_month DESC

It is apparent that December (followed by November) has the largest number of orders and sales than the other months, which makes a lot of sense as that would be when many users probably shop for holiday presents and gifts. Although August produced more orders and gross sales in totality, it did not beat the average sales and orders in July. This indicates that there was a very high volume of transactions for the month of August.

* Now let’s look at the same orders and gross sales (sum and average) by week, not including January and February:

SELECT md.data\_month, md.day\_of\_week, SUM(orders) Total\_Orders, CEILING(AVG(orders)) Average\_Orders, SUM(gross\_sales) TotalGross\_Sales,

CEILING(AVG(gross\_sales)) AverageGross\_Sales

FROM dbo.myData md

WHERE orders IS NOT NULL AND gross\_sales IS NOT NULL AND

md.data\_month IN (SELECT td.data\_month

FROM dbo.Transaction\_Date AS td

WHERE td.data\_month > 2)

GROUP BY md.day\_of\_week, md.data\_month

ORDER BY md.day\_of\_week, md.data\_month



Now let’s look at January and February

SELECT md.data\_month, md.day\_of\_week, SUM(orders) Total\_Orders, CEILING(AVG(orders)) Average\_Orders, SUM(gross\_sales) TotalGross\_Sales,

CEILING(AVG(gross\_sales)) AverageGross\_Sales

FROM dbo.myData md

WHERE orders IS NOT NULL AND gross\_sales IS NOT NULL AND

md.data\_month IN (SELECT td.data\_month

FROM dbo.Transaction\_Date AS td

WHERE td.data\_month <= 2)

GROUP BY md.day\_of\_week, md.data\_month

ORDER BY md.day\_of\_week, md.data\_month