CLOUD COMPUTED WEB APPLICATION FOR CORE TRACKING AND RANGE PLANNING

A dissertation submitted in partial Fulfillment of the requirement for the award of Degree in

Master of Fashion Technology

Submitted By

Rhythm Sharma

Under the Guidance of

Dr. S. Angammal Santhi,
Associate professor

Department of Fashion Technology
National Institute of Fashion Technology, Bengaluru

May, 2017

ABSTRACT

PROJECT TITLE: Cloud computed web application for core tracking and range planning.

Background:

The department largely relies on Microsoft Excel for all the numerical tracking, formula based calculations and related activities like transfers, stock checks and allocations etc. The same set of processes are repeated every time, this takes up unnecessary time that could be invested in doing some other activity. In Kids Wear Brands manual activity was to be eliminated and in Kids Wear Private Labels immediate action after tracking of styles was required to be done. The planning department had no existing measures of IT integration and automation and wished to move towards it, so there arose a requirement to conceptualize, design, develop, implement and deploy softwares named as ‘Range Planner’ Web application (software) in Brands and ‘Core tracker’ in Private Labels Web application (software) on cloud.

Objective:

1. Range Planner (Brands)
   - Eliminating manual processes involved in making a Range Plan and replacing it with a software for forecasting.
2. Core Tracker (Private labels)
   - Tracking core styles according to sales of previous and current year along with growth and giving a visual representation for analysis.
   - Tracking buffer level and stock level for each core style.

Keywords:

IT(Information Technology), software, automation, planning, tracking, forecasting, analysis

CERTIFICATE

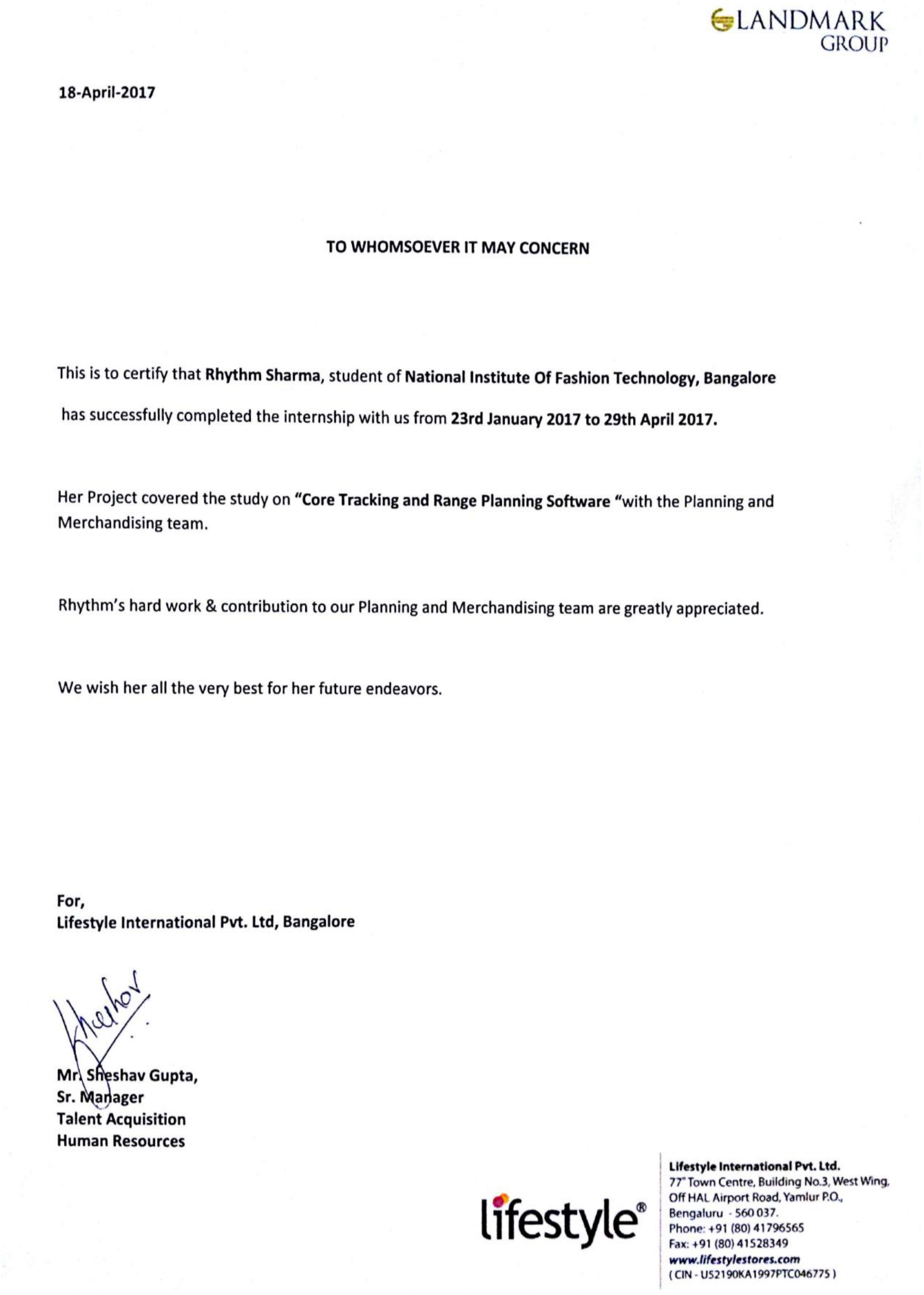
"This is to certify that this Project Report titled **"Cloud computed web application for core tracking and range planning"** *is based on my,* ***Rhythm Sharma's*** *original research work, conducted under the guidance of* ***Dr. S. Angammal Santhi, Associate professor, DFT, Bengaluru*** *towards partial fulfillment of the requirement for award of the Master's Degree in Fashion Technology (Apparel Production), of the National Institute of Fashion Technology, Campus.*

No part of this work has been copied from any other source. Material, wherever borrowed has been duly acknowledged."

Signature of Author/Researchers

Signature of Guide

COMPANY CERTIFICATE



LANDMARK
GROUP

18-April-2017

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Rhythm Sharma**, student of **National Institute Of Fashion Technology, Bangalore** has successfully completed the internship with us from **23rd January 2017 to 29th April 2017.**

Her Project covered the study on **"Core Tracking and Range Planning Software "**with the Planning and Merchandising team.

Rhythm's hard work & contribution to our Planning and Merchandising team are greatly appreciated.

We wish her all the very best for her future endeavors.

For,
Lifestyle International Pvt. Ltd, Bangalore

Mr. Sheshav Gupta,
Sr. Manager
Talent Acquisition
Human Resources

lifestyle®

Lifestyle International Pvt. Ltd.
77° Town Centre, Building No.3, West Wing,
Off HAL Airport Road, Yamlur P.O.,
Bengaluru - 560 037.
Phone: +91 (80) 41796565
Fax: +91 (80) 41528349
www.lifestylestores.com
(CIN - U52190KA1997PTC046775)

ACKNOWLEDGEMENT

This page expresses my honest and sincere gratitude towards all those who directly or indirectly encouraged me, helped me, criticized me in completion of this internship. The volume of the work would not have been possible without contribution in one form or the other by a few names to mention. I would like to express my sincere and grateful gratitude towards Mr. Kabir Lumba (MD, Lifestyle India) for providing me with an opportunity to do this momentous graduation project and internship in this prestigious organization.

I am thankful to Mr. Manish Sahu (Manager, Kids Wear- Brands), Mr. Amitesh Soni (Head, Kids wear – Brands and Private Labels) for mentoring me with a lot of support, help, patience, valuable suggestions and excellent guidance during the completion of this graduation project internship. They are the true guides who not only guided me professionally but also with moral values.

I would like to extend my sincere thanks to Mr. Aakash Aggarwal (developer, Practo) for his valuable guidance, suggestions and inputs during my graduation project.

I also take this opportunity to thank Ms. Sudha Singh(Course Coordinator, M.F.Tech, DFT, NIFT, Bangalore), my mentor Dr. Angammal Santhi (Associate professor, MFT, DFT, NIFT, Bangalore) and all the staff members of the Department of Fashion Technology for their cooperation and help during this internship.

I am very thankful to the honourable Director, Prof. V. Sivalingam and the Joint Director, Mr. M. Muthukumar, National Institute of Fashion Technology, Bengaluru, for their support.

I am also very thankful to my parents, friends and family for their unconditional support and faith.

CONTENTS

ANNEXURES

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. INTRODUCTION

The planning department largely relies on Microsoft Excel for all the numerical tracking, formula based calculations and related activities like transfers, stock checks and allocations etc. The same set of processes are repeated every time, this takes up unnecessary time that could be invested in doing some other activity.

In Kids Wear Brands manual activity was to be eliminated and in Kids Wear Private Labels immediate action after tracking of styles was required to be done. The planning department had no existing measures of IT integration and automation and wished to move towards it, so there arose a requirement to conceptualise, design, develop, implement and deploy softwares named as 'Range Planner' Web application (software) in Brands and 'Core tracker' in Private Labels Web application (software) on cloud.

1.1 Range planning

Merchandise range planning is part of the overall assortment planning process and involves making decisions about the amount of merchandise choice - the variety (or breadth) and the depth – that will be available to customers. The breadth of stock is the number of different types of merchandise and the depth of merchandise is defined as the number of SKU's per type.

Determining the right merchandise range is critical to meeting financial targets and satisfying customers. If the variety or amount of product aren't what the customer needs or what the customer expects from the retail brand, the retailer will likely lose the sale and possibly the customer.

It's a careful balancing act as overstocking merchandise will result in lower inventory turnover and reduced profits while understocking may result in lost sales and unhappy customers who take their business elsewhere.

In attempting to determine the ideal breadth and depth for a category, buyers should consider factors including:

Profitability of the merchandise mix

Corporate philosophy toward the assortment (broad and shallow or narrow and deep)

Physical characteristics of the stores or layout of the eCommerce site

Service level targets

Complementary merchandise

The ideal merchandise range plan results in the right merchandise, in the right quantity, at the right price and at the right time.

1.1.1 Objectives of a range planner

Eliminating manual processes involved in making a Range Plan and replacing it with a software for forecasting.

1.2 Core tracking

Core tracking aims at tracking all the sales dump data of core styles of private labels of kids wear.

This mainly focuses on availability of these core styles and stocks available in all the stores.

1.2.1 Objectives of core tracker

Tracking core styles according to sales of previous and current year along with growth and giving a visual representation for analysis.

Tracking buffer level and stock level for each core style.

CHAPTER 2

2. LITERATURE REVIEW

2.1 EXISTING RANGE PLANNING SOFTWARES IN THE MARKET:

a. JUST ENOUGH

Solutions offered:

1. Merchandise Financial Planning
2. Assortment & Space Planning
3. Clustering & Profiling
4. Allocation
5. Demand Forecasting
6. Inventory Planning
7. Replenishment & Order Planning
8. Promotion Management
9. Customer Insights
10. Price & Markdown Planning

b. TXTRETAIL

Solutions offered:

1. Merchandise Financial Planning
2. Assortment Planning & Buying - Building customer-focused collections
3. Collection Lifecycle Management

4. Forecasting, Allocation & Replenishment
5. Supply Chain Management

c. ISPIRA

Solutions offered:

1. Full price sales
2. Sale and markdowns
3. Purchase values and volumes
4. Open to buy

d. DARWIN

Solutions offered:

1. Financial planning
2. Merchandise planning
3. Assortment planning
4. Open to buy
5. Markdown planning

e. VISUALRETAILING

Solutions offered:

1. Optimized assortment per store cluster
2. Productivity increase for Planning & Merchandising Teams
3. Reduced costs by replacing physical Mock-up stores
4. Improved product photography
5. Reduced In-store labour costs
6. More accurate in-store execution
7. C-level & Management
8. Marketing & Planning
9. Retail operations & VM

f. RETAILSMART (not for fashion)

Solutions offered:

1. Store Planning Software
2. Planogram Software
3. Planogram Automation Software
4. Range Assortment Software
5. 3D Planograms
6. Consultancy Service
7. Product Photography

g. GALLERIA (not for fashion)

Solutions offered:

1. Master Data Management
2. Behavioral Clustering
3. Macro Space Optimization
4. Floor Planning
5. Virtual Store Design
6. Assortment Planning
7. Demand Transfer
8. Planograms
9. Promotion Optimization
10. Planogram Delivery and Compliance
11. Mobile Compliance
12. Analysis and Reporting

Collaboration Platform

2.2 EXISTING PRACTICES TO DEVELOP A RANGE PLAN IN LIFESTYLE:

2.2.1 Range plan

Right now Microsoft Excel is being used to develop range plans for Kids Wear Brands. (annexure 4)

Steps:

a. Sales dumps of previous seasons are taken. (annexure 1)

b. They are converted to pivots in a desired format, data is aggregated in a desired format. (annexure 2)

c. Assumptions are taken for the season for which the range plan is to be developed.(annexure 3)

d. This is put in a template in excel which has formula driven fields.(annexure 4)

e. Once a final range plan is made, a few fields are extracted to make a business plan.(annexure 5)

A total of 2 files are required and a total of 3 files are made manually.

2.2.2 Core tracker

Right now Microsoft Excel is being used to collect dumps of core styles for Kids Wear Private Labels. (annexure 5)

A core tracker template is available from which meaningful data was extracted for the software.

2.3 TOOLS REQUIRED FOR DEVELOPING A CUSTOMIZED SOFTWARE TO AUTOMATE THE MAKING OF A RANGE PLAN:

2.3.1 Back end or server side language

A server-side language is the programming language (also known as a code or a script) used on the web server to produce the website.

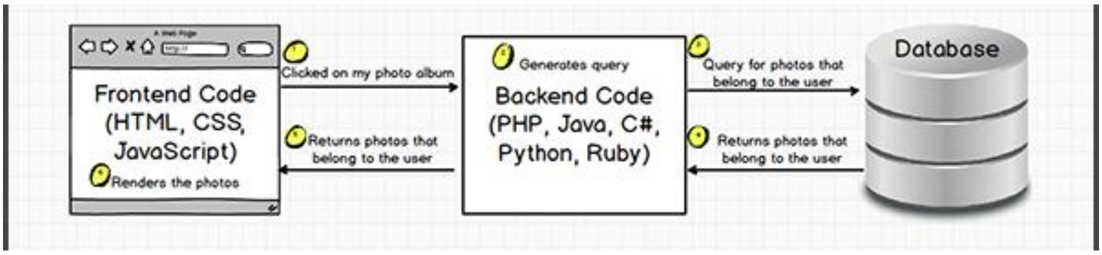

A Web Page
http://
Frontend Code (HTML, CSS, JavaScript)
Renders the photos
Clicked on my photo album
Returns photos that belong to the user
Generates query
Backend Code (PHP, Java, C#, Python, Ruby)
Query for photos that belong to the user
Returns photos that belong to the user
Database

FIGURE 1 FRONTEND, BACKEND DATABASE RELATION

2.3.1.1 Language trends of 2016

The programming languages in this list are ‘observable’ on GitHub and StackOverflow.

Chart showing the top programming languages of 2016:

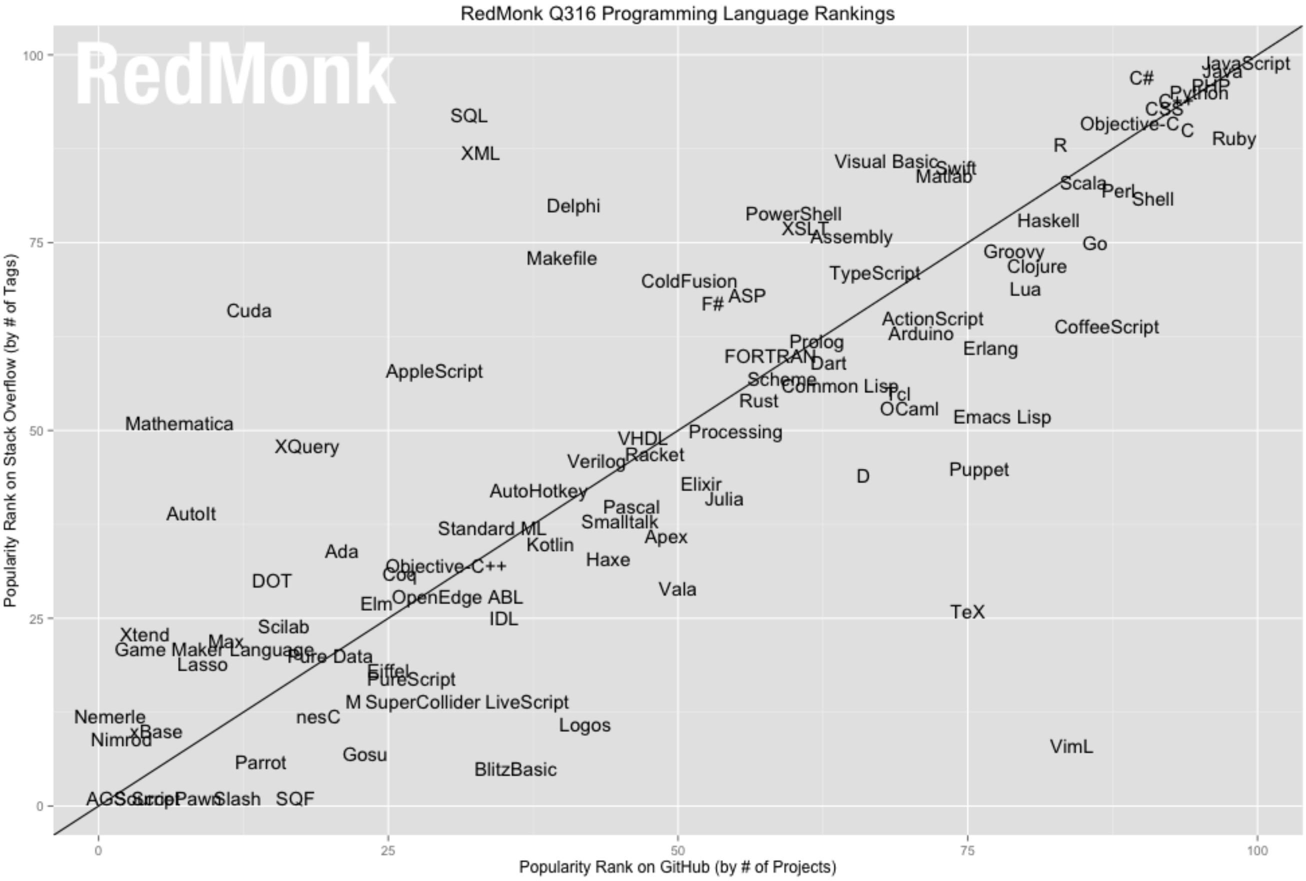


FIGURE 2 POPULARITY OF LANGUAGES IN 2016 ON GITHUB AND STACK OVERFLOW

List of top 21 programming languages of 2016:

#1. JavaScript

#2. Java

#3. PHP

#4. Python

#5. C#

#5. C++

#5. Ruby

#8. CSS

#9. C

#10. Objective-C

#11. Shell

#12. R

#13. Perl

#14. Scala

#15. Go

#16. Haskell

#17. Swift

#18. Matlab

#19. Visual Basic

#20. Clojure

#21. Groovy

Year wise trends of these languages:

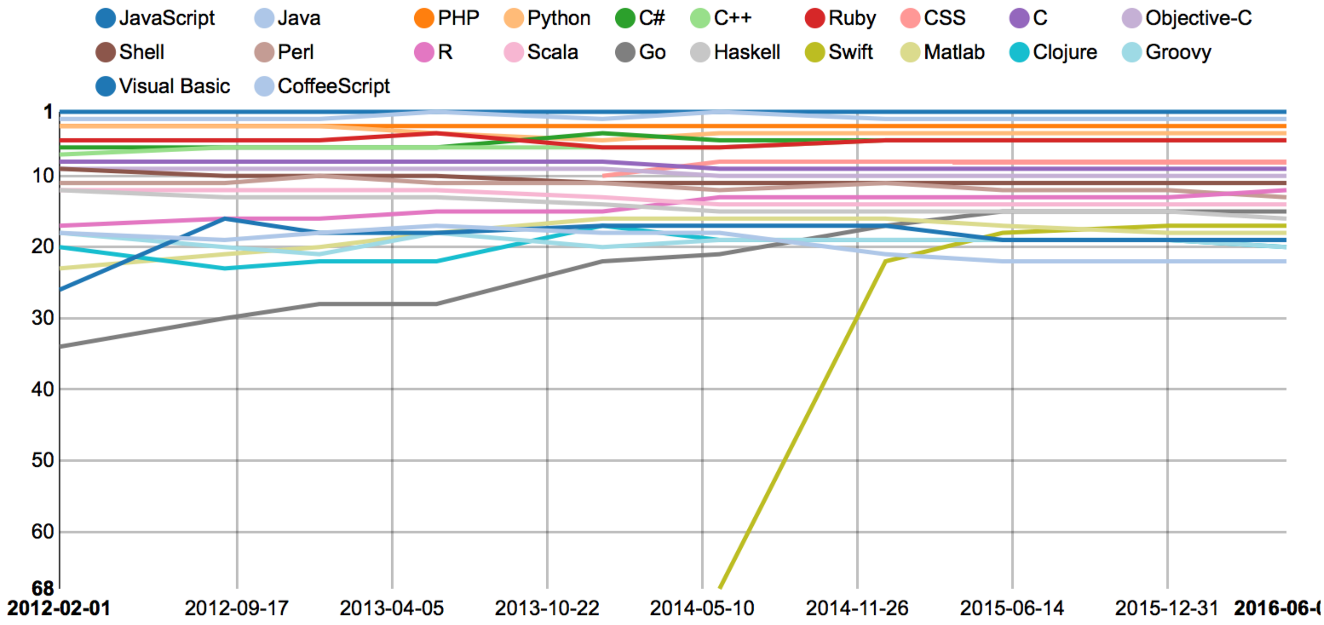


FIGURE 3 YEAR WISE LANGUAGE TRENDS

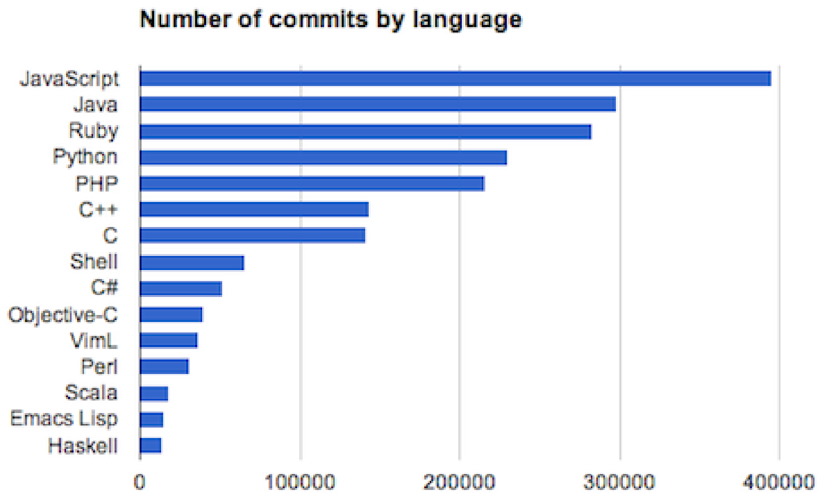


FIGURE 4 LANGUAGE WISE COMMITS ON GITHUB

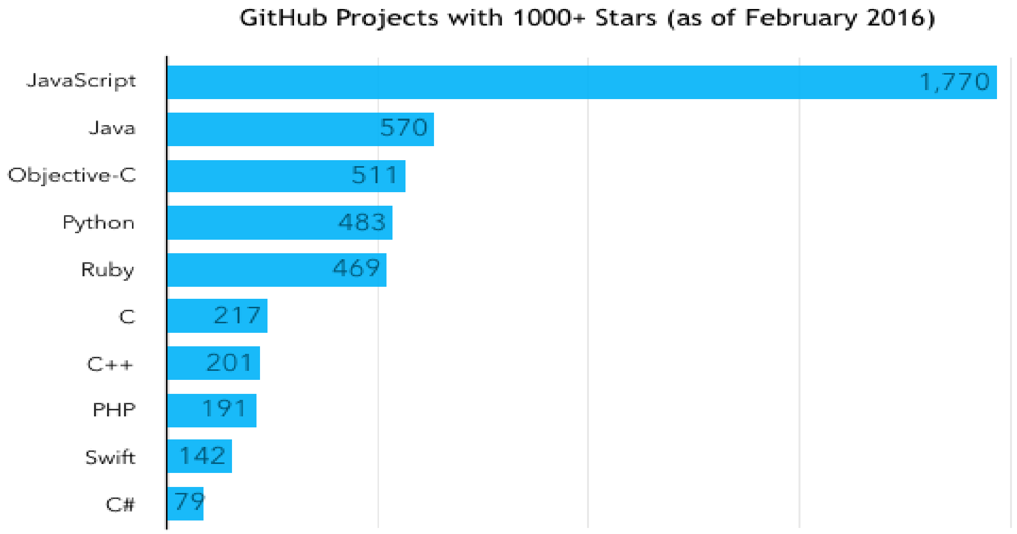


FIGURE 5 GITHUB PROJECTS WITH MORE THAN 1000 STARS

Top languages chosen:

1. .NET

Microsoft’s .NET platform is heavily used in business applications running the Microsoft Windows operating system. For Linux users, there is a port providing a subset of features via the Mono project. Although C# and various other parts of .NET are open standards, .NET is more or less a closed ecosystem controlled by Microsoft. Widely adopted by enterprises, the documentation and the support services for the language are comprehensive. Because .NET is a framework running on a virtualized platform known as the Common Language Infrastructure, or CLI, people have a choice of which language to use such as C#, VB.NET, C++ and even Python and Ruby. C# is the most popular language by far, followed by somewhat distant second VB.NET. In addition, .NET is also the technology used to create Windows mobile applications and integrates well with enterprise systems either running on Windows or through web services.

It's worth noting that although some free tools and libraries are available for .NET, the ecosystem is based on a licensed software model. This means that when third party tools are required, somebody somewhere will need to pay for a license and that eventually will fall upon you. Additionally, if you plan to use Windows as your OS for .NET, that will add costs (in contrast to the free license for most flavors of Linux), although Microsoft has programs that can offset or help to defer them. At the end of the day, choosing .NET is likely to cost more than choosing an open source platform.

Finally, many .NET programmers come from an enterprise background; developing internal systems and some may find adapting to the pace and the lifestyle of a start-up environment challenging.

2. Java

Java has been around since 1995 and is a core tool for creating enterprise applications since the advent of J2EE. The long commercial life and wide adoption of Java has created a robust ecosystem of documentation, libraries and frameworks many of which are aimed at e-commerce, security, and complex transactional architectures. There are experienced Java developers readily available regionally and globally both as contractors and full-time employees. The open source community embraced Java early on, creating an abundant free marketplace of Java solutions and tools.

The original goal of Java was to "write once, run everywhere", although there is some debate whether this was ever accomplished. Additional concern over licensing rights since the acquisition of Sun by Oracle has cast some doubt for the future of the language and given rise to the phrase "Java is dead, but the JVM lives on". This has fuelled the adoption of languages like Scala that runs on top of the JVM and can still take advantage of third party libraries. Java can also be resource-intensive, requiring more memory for example, as compared to other language. Java is used for Android devices and should be considered if that platform is part of your rollout strategy.

3. Python

Python is an open source interpretive language that has been embraced by many in the scientific community for its ease of learning and large set of scientific libraries. Django is a Python framework created for online newspaper publishers and powers popular start-ups such as Pinterest, Instagram, and EventBrite. Django and Python together are mostly platform independent, although developers must still design programs specifically for independence in order to run on both Windows and Linux platforms. There has been a trend in the Python community to improve code quality by dictating “one right way” to write Python, which is documented in the Python Enhancement Proposal (PEP) Index.

One caution is that Python is notoriously difficult to scale across multiple cores on a single machine. This is due to the limitations of the Global Interpreter Lock (GIL). However, it is very suited to applications that scale horizontally across stateless servers, making it a good solution for applications that take advantage of the cloud. Libraries such as Boto, IPython and Fabric also make it useful as an administrative scripting tool, allowing automation and infrastructure as code. This provides increased productivity for teams that have limited hands on keyboards.

4. Ruby

Ruby and particularly the web framework, Ruby on Rails, is also a popular interpreted language for start-ups. Ruby has some comprehensive training available online, both for beginners, with no coding experience, and experts in the field. Check out Try Ruby for newbies and Rails for Zombies for experienced programmers. The high adoption rate has meant the availability of many web-based libraries and tools to help web developers create applications rapidly. Ruby has repository of reusable libraries easily maintained and deployed in the form of RubyGems. Like Python, Ruby is suitable for automation with Puppet, which is an open source configuration management tool written natively in the language. Ruby powers popular web properties, including Airbnb, Github, and Groupon.

One valid concern is that Ruby does not scale up well on the server side for large numbers of requests to the application. As reported in InfoQ, Twitter famously made a migration from Ruby to Scala in 2010 in order to handle the back-end requirements of their explosive growth. Additionally, because Ruby is open source and community driven, quality documentation and support can be more difficult to find.

You may need to rely on Ruby experts to accomplish sophisticated solutions. There is some industry concern that Ruby becomes harder to maintain as the number of libraries and lines of code grow.

5. PHP

One of the early popular languages for Internet applications and websites, PHP has a vast ecosystem of developers, frameworks and libraries. Major companies including the likes of Facebook, WordPress, Twitpic, Flickr and Imgur are part of the PHP alumni. Because of the age of PHP and its long history, the quality of PHP code has a great variance. PHP doesn't have rules like compiled languages or strict standards as seen with Python, but rather guidelines available from the developer community. As a result, larger projects that do not have a strict structure can become difficult to read and maintain, a problem known as "Spaghetti Code."

6. Node.js

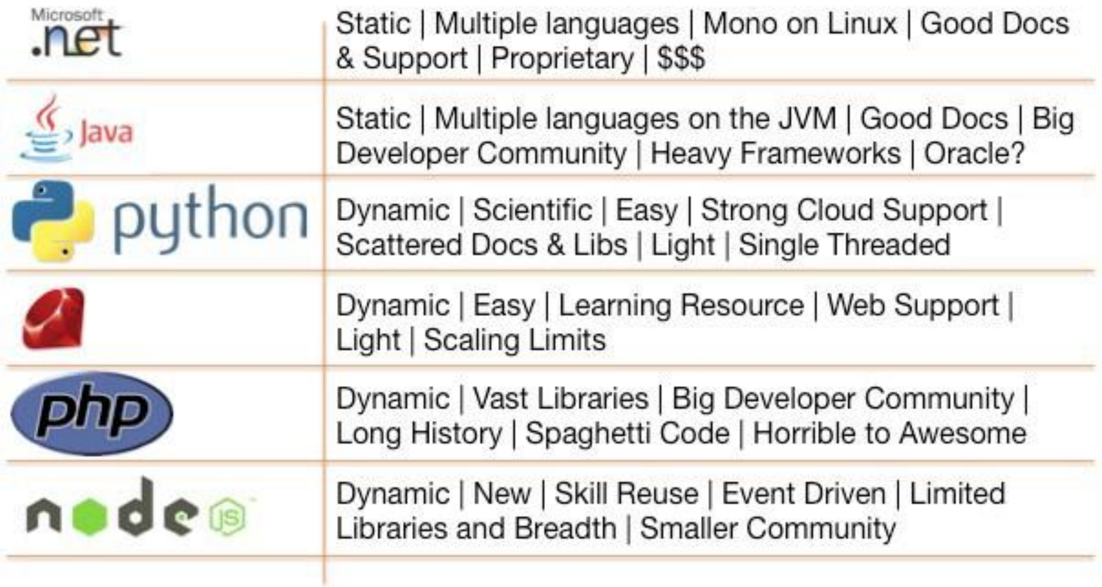
The newest entry in this list of programming frameworks is Node.js. Node.js is an event driven language and, having grown from JavaScript, the only one in this list designed from the start to serve web requests by taking advantage of JavaScript on the server side. The non-blocking I/O produces high performance server-side applications. Programmers will find that their client-side JavaScript skills are portable to server-side development tasks. Leaders such as Yahoo and LinkedIn have implemented portions of their applications on Node.js.

Because Node.js is newer, people with strong skills may be harder to find and more expensive to employ. Documentation and libraries are scarce, with additional challenges around the fact that projects tend to mix Node.js with other languages and frameworks to round out the feature set. However, this situation may improve as JavaScript developers' transition into back-end architectures.

Dynamic, Static, Interpretive and Compiled

All of the languages above can be put into two categories: Statically compiled and strongly typed on the one hand or dynamic and interpretive on the other. Older, traditional enterprise languages usually fit into the former group. Code created in this manner provides developers with compile-time errors rather than run time errors, ensuring safer, well-documented code with clean interfaces. However, applications using statically compiled code will take longer to build. This trade-off made sense when these languages were initially introduced. At that time compute power was difficult to obtain and costly. Today, however, the availability of cloud computing with AWS for instantly available, pay-as-you-go compute capacity has drastically changed the industry.

Now, compute power is faster and cheaper, while Integrated Development Environments (IDEs) are readily available to simplify development using interpretative languages and including support for sophisticated debugging and refactoring functionality. The advantages provided by compiled languages over dynamic languages have narrowed over the last few years, with dynamic languages proven to deliver high quality code and to be naturally suited to rapid development cycles. This is why many successful startups are gravitating towards the dynamic/interpretative side of the spectrum when choosing their development languages.



Microsoft .net	Static | Multiple languages | Mono on Linux | Good Docs & Support | Proprietary | $$$
Java	Static | Multiple languages on the JVM | Good Docs | Big Developer Community | Heavy Frameworks | Oracle?
python	Dynamic | Scientific | Easy | Strong Cloud Support | Scattered Docs & Libs | Light | Single Threaded
	Dynamic | Easy | Learning Resource | Web Support | Light | Scaling Limits
php	Dynamic | Vast Libraries | Big Developer Community | Long History | Spaghetti Code | Horrible to Awesome
node js	Dynamic | New | Skill Reuse | Event Driven | Limited Libraries and Breadth | Smaller Community

FIGURE 6 SUMMARY OF LANGUAGES CHOSEN

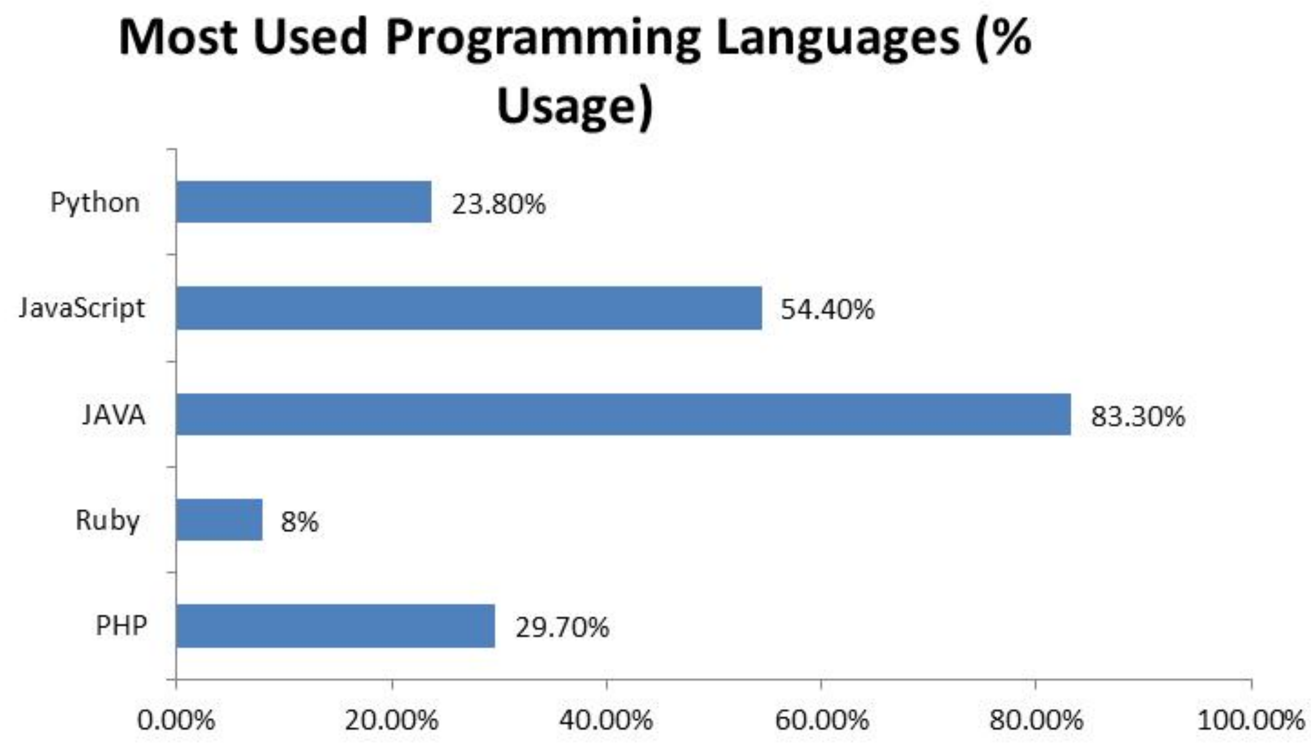


FIGURE 7 PERCENTAGE USAGE OF LANGUAGES CHOSEN

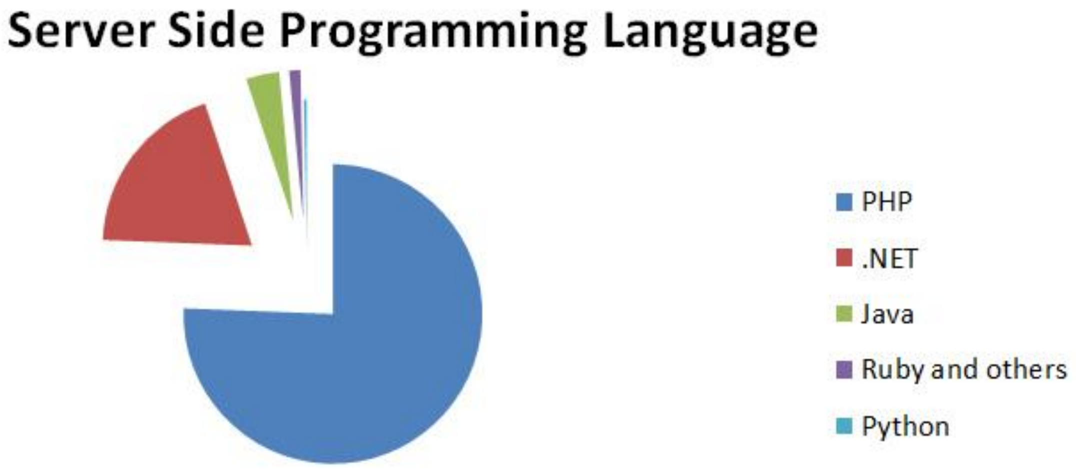
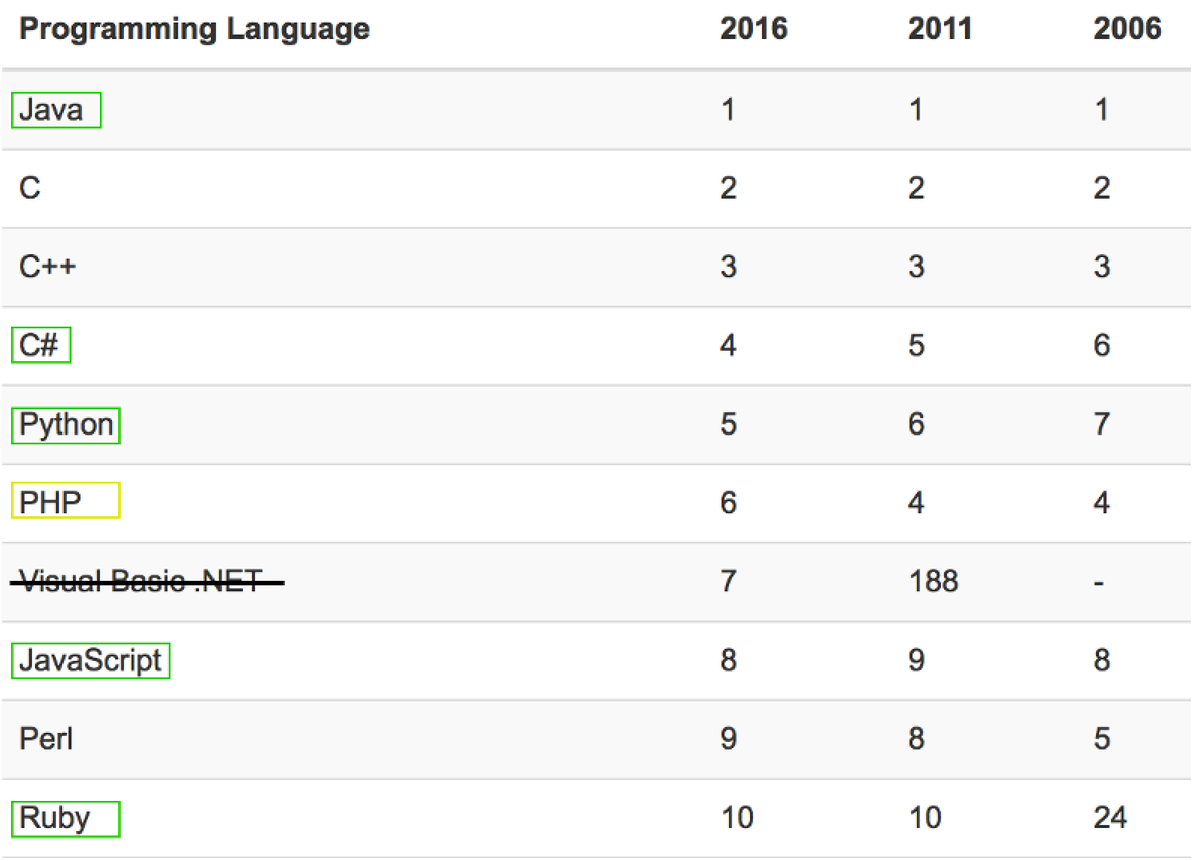


FIGURE 8 PIE CHART USAGE OF LANGUAGES CHOSEN



Programming Language	2016	2011	2006
Java	1	1	1
C	2	2	2
C++	3	3	3
C#	4	5	6
Python	5	6	7
PHP	6	4	4
~~Visual Basic .NET~~	7	188	-
JavaScript	8	9	8
Perl	9	8	5
Ruby	10	10	24

FIGURE 9 LANGUAGES TO FOCUS ON AND YEAR WISE RANKINGS

2.3.1.2 Comparisons and tests on various languages

This comparison consists of three parts:

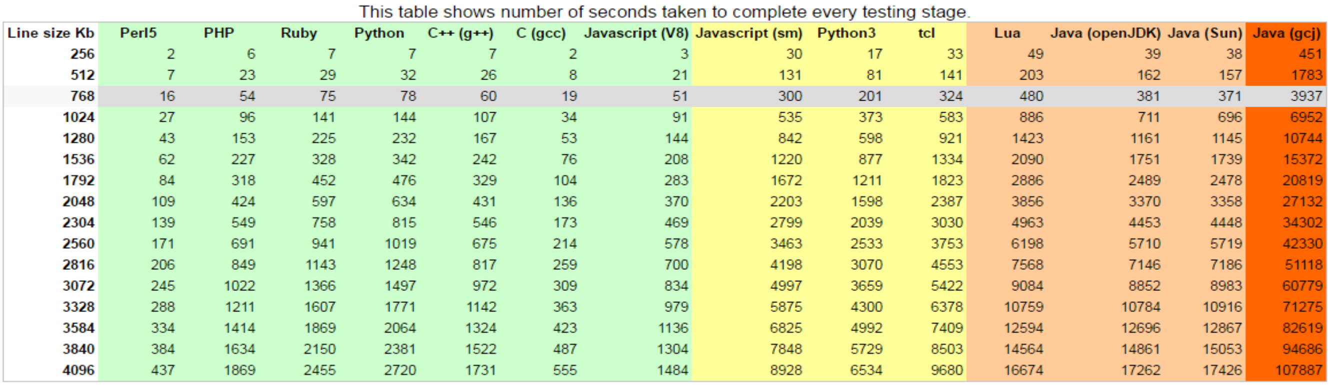
Part 1: Speed.

Part 2: Memory usage.

Part 3: Language features.

Speed

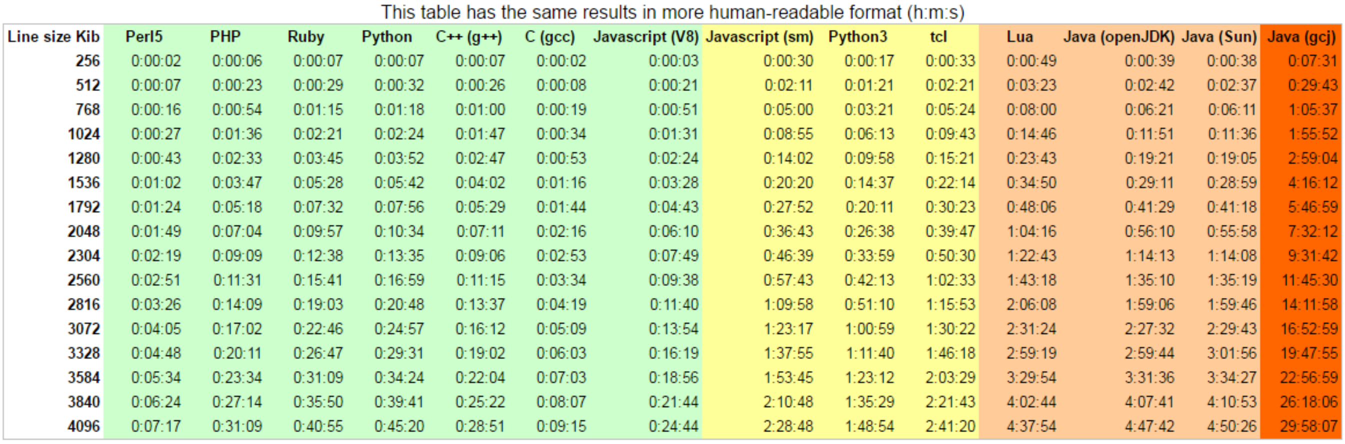
Execution speed is obviously important to understand the language. However performance alone is not the most important characteristic and therefore other aspects should be taken into consideration as well.



This table shows number of seconds taken to complete every testing stage.

Line size Kb	Perl5	PHP	Ruby	Python	C++ (g++)	C (gcc)	Javascript (V8)	Javascript (sm)	Python3	tcl	Lua	Java (openJDK)	Java (Sun)	Java (gcj)
256	2	6	7	7	7	2	3	30	17	33	49	39	38	451
512	7	23	29	32	26	8	21	131	81	141	203	162	157	1783
768	16	54	75	78	60	19	51	300	201	324	480	381	371	3937
1024	27	96	141	144	107	34	91	535	373	583	886	711	696	6952
1280	43	153	225	232	167	53	144	842	598	921	1423	1161	1145	10744
1536	62	227	328	342	242	76	208	1220	877	1334	2090	1751	1739	15372
1792	84	318	452	476	329	104	283	1672	1211	1823	2886	2489	2478	20819
2048	109	424	597	634	431	136	370	2203	1598	2387	3856	3370	3358	27132
2304	139	549	758	815	546	173	469	2799	2039	3030	4963	4453	4448	34302
2560	171	691	941	1019	675	214	578	3463	2533	3753	6198	5710	5719	42330
2816	206	849	1143	1248	817	259	700	4198	3070	4553	7568	7146	7186	51118
3072	245	1022	1366	1497	972	309	834	4997	3659	5422	9084	8852	8983	60779
3328	288	1211	1607	1771	1142	363	979	5875	4300	6378	10759	10784	10916	71275
3584	334	1414	1869	2064	1324	423	1136	6825	4992	7409	12594	12696	12867	82619
3840	384	1634	2150	2381	1522	487	1304	7848	5729	8503	14564	14861	15053	94686
4096	437	1869	2455	2720	1731	555	1484	8928	6534	9680	16674	17262	17426	107887

FIGURE 10 SPEED TEST OF VARIOUS LANGUAGES



This table has the same results in more human-readable format (h:m:s)

Line size Kib	Perl5	PHP	Ruby	Python	C++ (g++)	C (gcc)	Javascript (V8)	Javascript (sm)	Python3	tcl	Lua	Java (openJDK)	Java (Sun)	Java (gcj)
256	0:00:02	0:00:06	0:00:07	0:00:07	0:00:07	0:00:02	0:00:03	0:00:30	0:00:17	0:00:33	0:00:49	0:00:39	0:00:38	0:07:31
512	0:00:07	0:00:23	0:00:29	0:00:32	0:00:26	0:00:08	0:00:21	0:02:11	0:01:21	0:02:21	0:03:23	0:02:42	0:02:37	0:29:43
768	0:00:16	0:00:54	0:01:15	0:01:18	0:01:00	0:00:19	0:00:51	0:05:00	0:03:21	0:05:24	0:08:00	0:06:21	0:06:11	1:05:37
1024	0:00:27	0:01:36	0:02:21	0:02:24	0:01:47	0:00:34	0:01:31	0:08:55	0:06:13	0:09:43	0:14:46	0:11:51	0:11:36	1:55:52
1280	0:00:43	0:02:33	0:03:45	0:03:52	0:02:47	0:00:53	0:02:24	0:14:02	0:09:58	0:15:21	0:23:43	0:19:21	0:19:05	2:59:04
1536	0:01:02	0:03:47	0:05:28	0:05:42	0:04:02	0:01:16	0:03:28	0:20:20	0:14:37	0:22:14	0:34:50	0:29:11	0:28:59	4:16:12
1792	0:01:24	0:05:18	0:07:32	0:07:56	0:05:29	0:01:44	0:04:43	0:27:52	0:20:11	0:30:23	0:48:06	0:41:29	0:41:18	5:46:59
2048	0:01:49	0:07:04	0:09:57	0:10:34	0:07:11	0:02:16	0:06:10	0:36:43	0:26:38	0:39:47	1:04:16	0:56:10	0:55:58	7:32:12
2304	0:02:19	0:09:09	0:12:38	0:13:35	0:09:06	0:02:53	0:07:49	0:46:39	0:33:59	0:50:30	1:22:43	1:14:13	1:14:08	9:31:42
2560	0:02:51	0:11:31	0:15:41	0:16:59	0:11:15	0:03:34	0:09:38	0:57:43	0:42:13	1:02:33	1:43:18	1:35:10	1:35:19	11:45:30
2816	0:03:26	0:14:09	0:19:03	0:20:48	0:13:37	0:04:19	0:11:40	1:09:58	0:51:10	1:15:53	2:06:08	1:59:06	1:59:46	14:11:58
3072	0:04:05	0:17:02	0:22:46	0:24:57	0:16:12	0:05:09	0:13:54	1:23:17	1:00:59	1:30:22	2:31:24	2:27:32	2:29:43	16:52:59
3328	0:04:48	0:20:11	0:26:47	0:29:31	0:19:02	0:06:03	0:16:19	1:37:55	1:11:40	1:46:18	2:59:19	2:59:44	3:01:56	19:47:55
3584	0:05:34	0:23:34	0:31:09	0:34:24	0:22:04	0:07:03	0:18:56	1:53:45	1:23:12	2:03:29	3:29:54	3:31:36	3:34:27	22:56:59
3840	0:06:24	0:27:14	0:35:50	0:39:41	0:25:22	0:08:07	0:21:44	2:10:48	1:35:29	2:21:43	4:02:44	4:07:41	4:10:53	26:18:06
4096	0:07:17	0:31:09	0:40:55	0:45:20	0:28:51	0:09:15	0:24:44	2:28:48	1:48:54	2:41:20	4:37:54	4:47:42	4:50:26	29:58:07

FIGURE 11 SPEED TEST OF VARIOUS LANGUAGES FORMAT 2

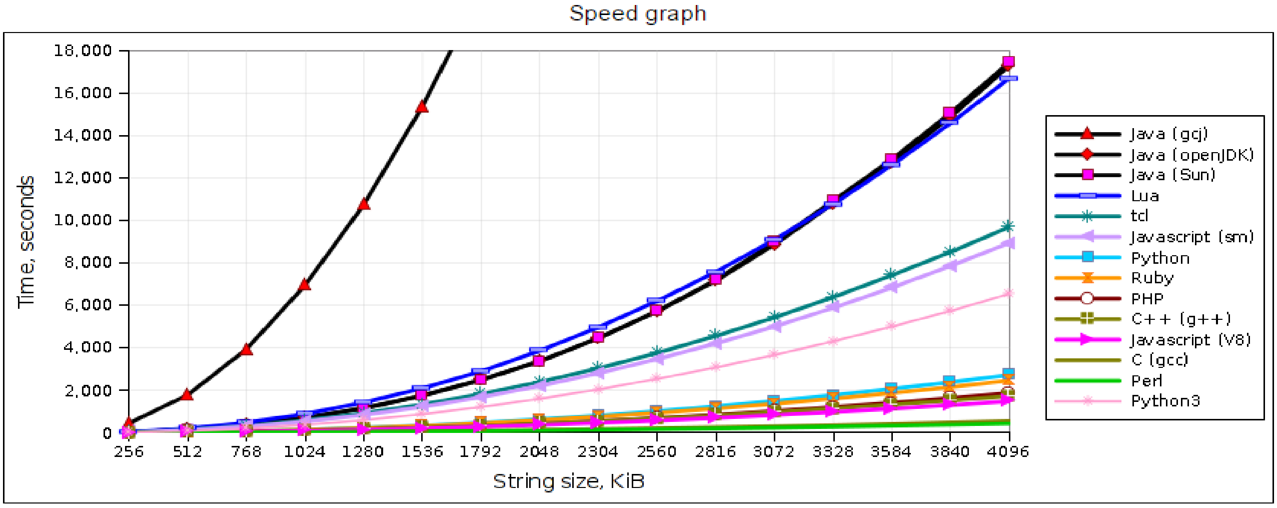


FIGURE 12 SPEED GRAPH

Speed tests fall into 4 categories:

Slowest: Java gcj (native executable)

Slow: Java (openJDK); Java (Sun); Lua

Not-so-fast: tcl; Javascript (spidermonkey)

Fastest: Python; Ruby; PHP; C++; Javascript V8; C; Perl5

As it can be seen from performance graph, processing speed slows down as the test string grow. The more graph curves up the more performance degrades. Graph reveals that performance of Java and Lua degrades dramatically.

All tested languages are good with manipulation of little strings but as the processed data grow the difference manifests itself.

Slow group [Java, Lua] suffer from severe performance degradation.

There are almost no difference in performance between OpenJDK Java and Sun Java. Lua's performance is very close to Java.

Initially GCJ Java interpreter crashed during the test, however GCJ Java can compile Java code to executable file which completed the test even though awfully slow. Here and below unqualified "Java" means only mainstream Sun/OpenJDK Java.

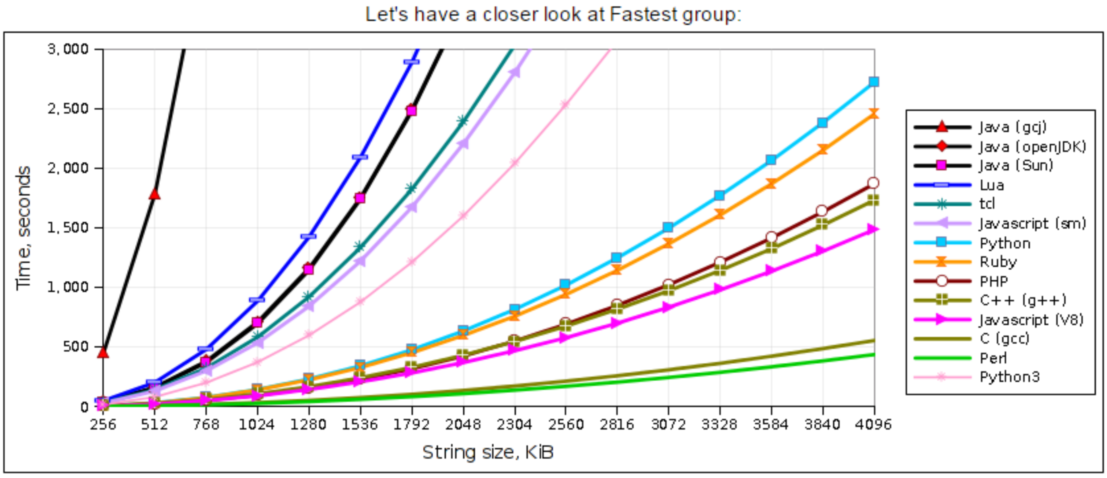


FIGURE 13 SPEED GRAPH ZOOMED IN

Pyhon, Ruby and PHP are slightly slower than than C++. This is not a surprise because those languages are optimised well enough.

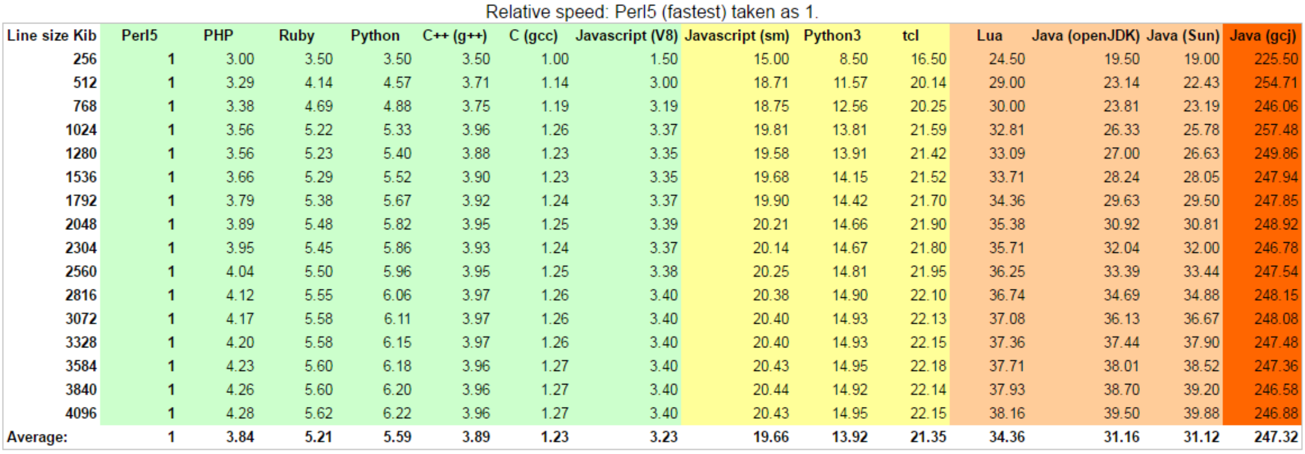
Javascript V8 completed test slightly faster than C++.

This group of languages shows average slow down while performance of C and Perl5 is almost a flat line on graph indicating very little degradation. It means that C and Perl5 process increasing amount of data at (almost) constant speed.

Perl5 is a clear winner with just a little more than 7 minutes needed to finish test against Java with worst result as big as nearly 5 hours to do the same. (Worst result of GCJ Java - almost 30 hours, doesn't worth comparing against)

Perl5 is not only superior in performance but it shows very little slow down on larger data. This is as close to C (compiled to machine code) as it can be for scripting language.

Interesting to note that with "use strict;" Perl completed the same test ~6 seconds quicker.In the table below Perl5 has been taken as 1 and other language's performance measured in Perls so it can be seen how many times slower a particular language comparing to Perl5 in this test. Because of performance degradation it will be incorrect to say something like "This is twice faster than That". Some language's performance degrade faster than others so in beginning of this test Java somewhat 20 times slower than Perl5 and in the end Java is about 40 times slower (for same amount of data). Clearly this is an important characteristic - size matters! This is correspond with observation of some Java applications which behave well under little load and degrade exponentially as the load increases.



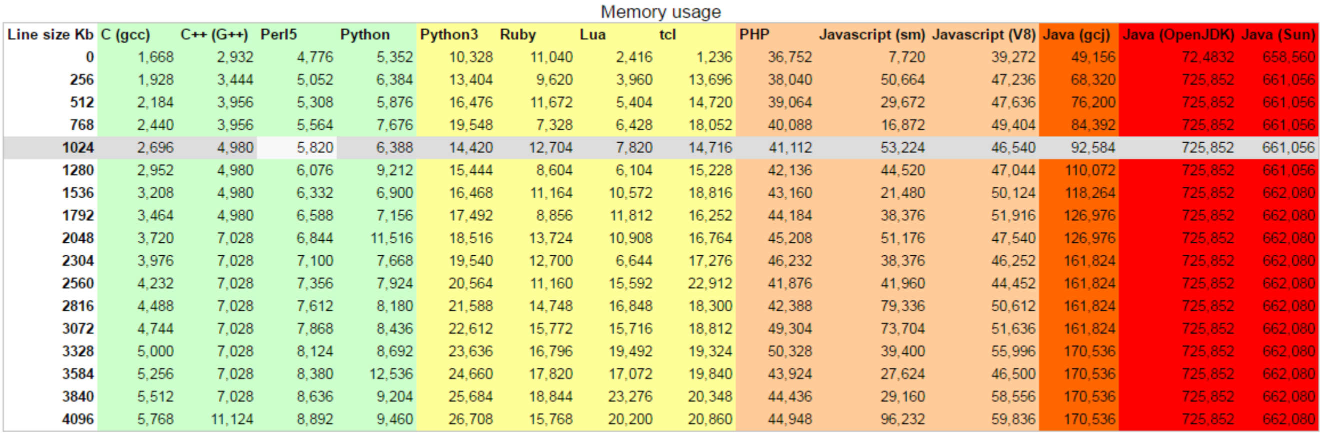
Relative speed: Perl5 (fastest) taken as 1.

Line size Kib	Perl5	PHP	Ruby	Python	C++ (g++)	C (gcc)	Javascript (V8)	Javascript (sm)	Python3	tcl	Lua	Java (openJDK)	Java (Sun)	Java (gcj)
256	1	3.00	3.50	3.50	3.50	1.00	1.50	15.00	8.50	16.50	24.50	19.50	19.00	225.50
512	1	3.29	4.14	4.57	3.71	1.14	3.00	18.71	11.57	20.14	29.00	23.14	22.43	254.71
768	1	3.38	4.69	4.88	3.75	1.19	3.19	18.75	12.56	20.25	30.00	23.81	23.19	246.06
1024	1	3.56	5.22	5.33	3.96	1.26	3.37	19.81	13.81	21.59	32.81	26.33	25.78	257.48
1280	1	3.56	5.23	5.40	3.88	1.23	3.35	19.58	13.91	21.42	33.09	27.00	26.63	249.86
1536	1	3.66	5.29	5.52	3.90	1.23	3.35	19.68	14.15	21.52	33.71	28.24	28.05	247.94
1792	1	3.79	5.38	5.67	3.92	1.24	3.37	19.90	14.42	21.70	34.36	29.63	29.50	247.85
2048	1	3.89	5.48	5.82	3.95	1.25	3.39	20.21	14.66	21.90	35.38	30.92	30.81	248.92
2304	1	3.95	5.45	5.86	3.93	1.24	3.37	20.14	14.67	21.80	35.71	32.04	32.00	246.78
2560	1	4.04	5.50	5.96	3.95	1.25	3.38	20.25	14.81	21.95	36.25	33.39	33.44	247.54
2816	1	4.12	5.55	6.06	3.97	1.26	3.40	20.38	14.90	22.10	36.74	34.69	34.88	248.15
3072	1	4.17	5.58	6.11	3.97	1.26	3.40	20.40	14.93	22.13	37.08	36.13	36.67	248.08
3328	1	4.20	5.58	6.15	3.97	1.26	3.40	20.40	14.93	22.15	37.36	37.44	37.90	247.48
3584	1	4.23	5.60	6.18	3.96	1.27	3.40	20.43	14.95	22.18	37.71	38.01	38.52	247.36
3840	1	4.26	5.60	6.20	3.96	1.27	3.40	20.44	14.92	22.14	37.93	38.70	39.20	246.58
4096	1	4.28	5.62	6.22	3.96	1.27	3.40	20.43	14.95	22.15	38.16	39.50	39.88	246.88
Average:	1	3.84	5.21	5.59	3.89	1.23	3.23	19.66	13.92	21.35	34.36	31.16	31.12	247.32

FIGURE 14 RELATIVE SPEED

Memory usage

During testing memory usage were captured as per every completed step.



Memory usage

Line size Kb	C (gcc)	C++ (G++)	Perl5	Python	Python3	Ruby	Lua	tcl	PHP	Javascript (sm)	Javascript (V8)	Java (gcj)	Java (OpenJDK)	Java (Sun)
0	1,668	2,932	4,776	5,352	10,328	11,040	2,416	1,236	36,752	7,720	39,272	49,156	72,4832	658,560
256	1,928	3,444	5,052	6,384	13,404	9,620	3,960	13,696	38,040	50,664	47,236	68,320	725,852	661,056
512	2,184	3,956	5,308	5,876	16,476	11,672	5,404	14,720	39,064	29,672	47,636	76,200	725,852	661,056
768	2,440	3,956	5,564	7,676	19,548	7,328	6,428	18,052	40,088	16,872	49,404	84,392	725,852	661,056
1024	2,696	4,980	5,820	6,388	14,420	12,704	7,820	14,716	41,112	53,224	46,540	92,584	725,852	661,056
1280	2,952	4,980	6,076	9,212	15,444	8,604	6,104	15,228	42,136	44,520	47,044	110,072	725,852	661,056
1536	3,208	4,980	6,332	6,900	16,468	11,164	10,572	18,816	43,160	21,480	50,124	118,264	725,852	662,080
1792	3,464	4,980	6,588	7,156	17,492	8,856	11,812	16,252	44,184	38,376	51,916	126,976	725,852	662,080
2048	3,720	7,028	6,844	11,516	18,516	13,724	10,908	16,764	45,208	51,176	47,540	126,976	725,852	662,080
2304	3,976	7,028	7,100	7,668	19,540	12,700	6,644	17,276	46,232	38,376	46,252	161,824	725,852	662,080
2560	4,232	7,028	7,356	7,924	20,564	11,160	15,592	22,912	41,876	41,960	44,452	161,824	725,852	662,080
2816	4,488	7,028	7,612	8,180	21,588	14,748	16,848	18,300	42,388	79,336	50,612	161,824	725,852	662,080
3072	4,744	7,028	7,868	8,436	22,612	15,772	15,716	18,812	49,304	73,704	51,636	161,824	725,852	662,080
3328	5,000	7,028	8,124	8,692	23,636	16,796	19,492	19,324	50,328	39,400	55,996	170,536	725,852	662,080
3584	5,256	7,028	8,380	12,536	24,660	17,820	17,072	19,840	43,924	27,624	46,500	170,536	725,852	662,080
3840	5,512	7,028	8,636	9,204	25,684	18,844	23,276	20,348	44,436	29,160	58,556	170,536	725,852	662,080
4096	5,768	11,124	8,892	9,460	26,708	15,768	20,200	20,860	44,948	96,232	59,836	170,536	725,852	662,080

FIGURE 15 MEMORY USAGE

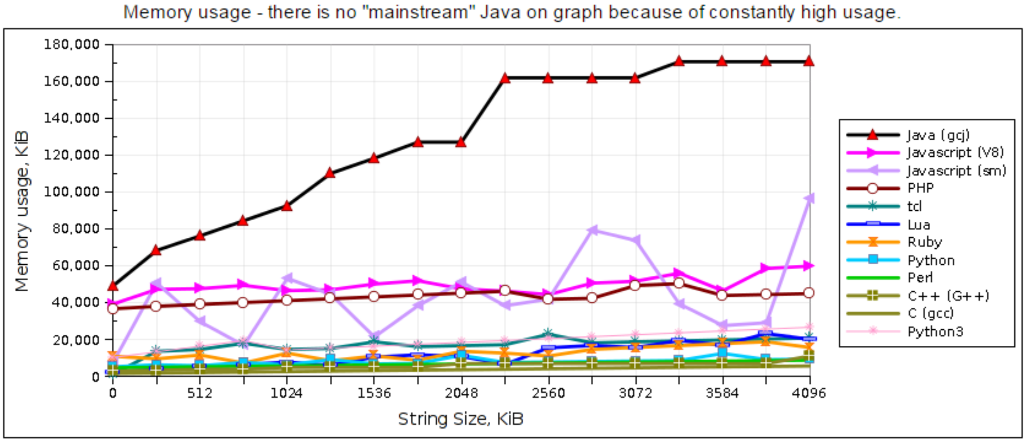


FIGURE 16 MEMORY USAGE GRAPH

Result fall into five categories:

Highest: Java OpenJDK, Java Sun

High:Java GCJ

Medium:Javascript V8, Javascript sm., PHP

Low:tcl, Lua, Ruby

Lowest: Python, Perl5, C++, C

Highest group - mainstream Java pre-allocates a fairly big chunk of memory (certain percentage) by default and does memory management inside this chunk. During this test memory usage hasn't change and was constantly high - so it is not present on graph: if included it makes all other results appear as flat lines well below.

To capture internal memory usage, print statements to Java code were introduced to show internal memory usage as per string growth. (It doesn't affect performance) Unfortunately printed numbers has no correspondence with string growth. This shows that Java garbage collection works completely independent from application code. Output numbers appeared to be random, sometimes as high as up to 95% of pre-allocated memory. Even if internal memory usage did not correspond with the string size it seems that sometimes Java is using nearly all of its memory before garbage collection (GC) releases some of it.

Java memory management appears to be extremely ineffective which seems to be the primary cause for poor performance.

High group - Java GCJ compiled to native executable. Thanks to this special feature GCJ Java demonstrated predictable behaviour when memory allocation grows together with data processed. Comparing with other non-Java runtimes memory utilisation is huge.

Medium group: Javascript demonstrate more or less consistent grow in memory usage as per data growth. PHP shows very little grow but its heavy runtime uses a lot of memory from very beginning. Despite initial requirements PHP uses memory pretty wise. High memory usage upon startup is not necessarily bad thing: if meant for continuous execution it may be OK to pre-load common libraries. However this may be a limitation for PHP usage on VPS server i.e. when available memory is limited.

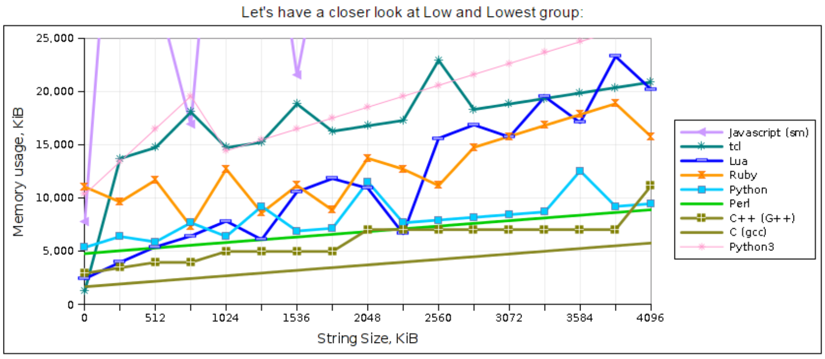
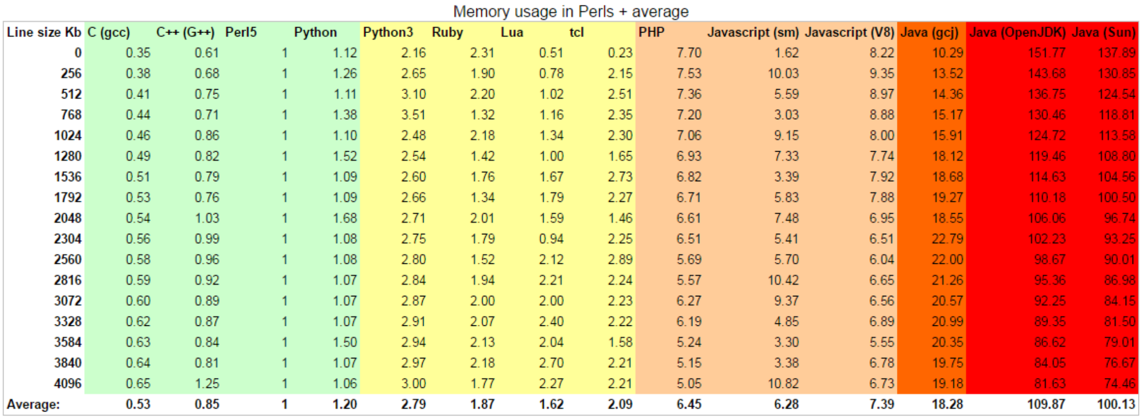


FIGURE 17 MEMORY USAGE GRAPH ZOOMED IN

Lua and tcl runtimes are tiny, but their memory management not very effective. Ruby used more memory than Python. Python utilises memory almost as good as Perl5 - perhaps their runtimes are almost the same size. Once again Perl5 performed amazingly well, demonstrating behaviour very similar to C - best among scripting languages. As expected C++ memory usage is roughly between C and Perl5.

As was done in speed test let's take Perl5 as 1 and see how other language's memory usage compares on every step and on average.



Memory usage in Perls + average

Line size Kb	C (gcc)	C++ (G++)	Perl5	Python	Python3	Ruby	Lua	tcl	PHP	Javascript (sm)	Javascript (V8)	Java (gcj)	Java (OpenJDK)	Java (Sun)
0	0.35	0.61	1	1.12	2.16	2.31	0.51	0.23	7.70	1.62	8.22	10.29	151.77	137.89
256	0.38	0.68	1	1.26	2.65	1.90	0.78	2.15	7.53	10.03	9.35	13.52	143.68	130.85
512	0.41	0.75	1	1.11	3.10	2.20	1.02	2.51	7.36	5.59	8.97	14.36	136.75	124.54
768	0.44	0.71	1	1.38	3.51	1.32	1.16	2.35	7.20	3.03	8.88	15.17	130.46	118.81
1024	0.46	0.86	1	1.10	2.48	2.18	1.34	2.30	7.06	9.15	8.00	15.91	124.72	113.58
1280	0.49	0.82	1	1.52	2.54	1.42	1.00	1.65	6.93	7.33	7.74	18.12	119.46	108.80
1536	0.51	0.79	1	1.09	2.60	1.76	1.67	2.73	6.82	3.39	7.92	18.68	114.63	104.56
1792	0.53	0.76	1	1.09	2.66	1.34	1.79	2.27	6.71	5.83	7.88	19.27	110.18	100.50
2048	0.54	1.03	1	1.68	2.71	2.01	1.59	1.46	6.61	7.48	6.95	18.55	106.06	96.74
2304	0.56	0.99	1	1.08	2.75	1.79	0.94	2.25	6.51	5.41	6.51	22.79	102.23	93.25
2560	0.58	0.96	1	1.08	2.80	1.52	2.12	2.89	5.69	5.70	6.04	22.00	98.67	90.01
2816	0.59	0.92	1	1.07	2.84	1.94	2.21	2.24	5.57	10.42	6.65	21.26	95.36	86.98
3072	0.60	0.89	1	1.07	2.87	2.00	2.00	2.23	6.27	9.37	6.56	20.57	92.25	84.15
3328	0.62	0.87	1	1.07	2.91	2.07	2.40	2.22	6.19	4.85	6.89	20.99	89.35	81.50
3584	0.63	0.84	1	1.50	2.94	2.13	2.04	1.58	5.24	3.30	5.55	20.35	86.62	79.01
3840	0.64	0.81	1	1.07	2.97	2.18	2.70	2.21	5.15	3.38	6.78	19.75	84.05	76.67
4096	0.65	1.25	1	1.06	3.00	1.77	2.27	2.21	5.05	10.82	6.73	19.18	81.63	74.46
Average:	0.53	0.85	1	1.20	2.79	1.87	1.62	2.09	6.45	6.28	7.39	18.28	109.87	100.13

FIGURE 18 RELATIVE MEMORY USAGE TABLE

Environment where applications work may have certain memory limits. It is true not only for popular Virtual Private Servers (VPS) where sometimes amount of RAM can be as little as 128 Mb for OS and all applications/services but also for embedded devices and heavily loaded servers. Good understanding of memory utilisation is equally important for consideration as speed.

Language features

Sometimes comfort and speed of development may outweigh performance and memory usage. Or in other words, perhaps sometimes performance and memory usage may be sacrificed in favour of quicker/easier development. For example, it is understandable if higher level language is chosen over C in order to benefit from automatic memory management. In this section I'm going to briefly scratch the surface of comparing language features.

Whilst it's quite a philosophical statement, language features play an important role in development. Let's see how easily an integer be parsed value from text string in popular languages. This task only looks straightforward. In fact there are plenty caveats.

In Java this is done:

```
//Java
   int val;
   val = Integer.parseInt("10000000000");
```

But there are problems. The example above will not only fail to parse correct value, but actually crash the entire application because of unhandled exception.

In this Java example

```
//Java
   val = Integer.parseInt("-10");   //this will work
   val = Integer.parseInt("+10");   //but not this - silly!
```

parsing integer from "+10" crashing application. To emulate this behaviour in PHP or Perl explicitly a point of failure has to be created:

```
$val=intval($str) or die("it didn't work");
```

In Java pretty much any call that does something can be a failure point unless enclosed within ugly try-catch statements. So to avoid crash wrap 'dangerous operations like this:

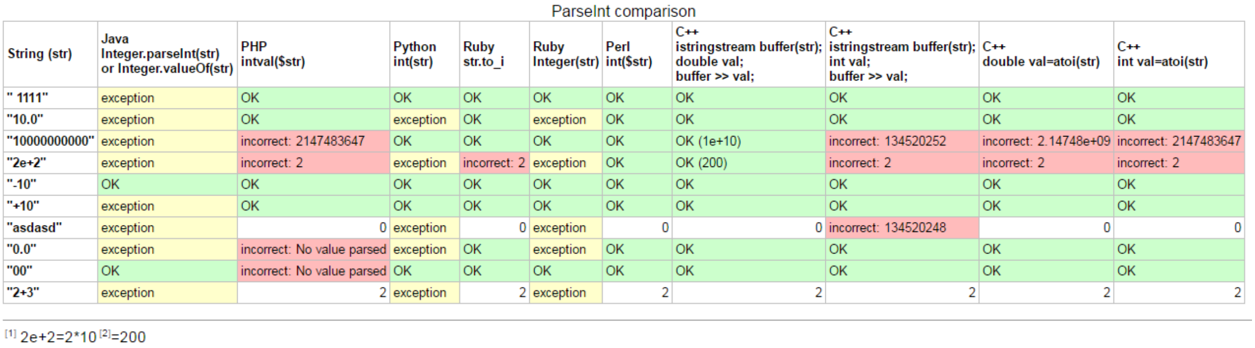
```
//Java
 try {
       val = Integer.parseInt(str);
 } catch (NumberFormatException nx) {
       //it didn't work, do something about it here
 }
```

In fact try-catch is a fancy syntax for if-else. Similar operation in PHP will not crash, but we can wrap it with if-else to make sure number parsed successfully.

```
#PHP
```

```
if($val=intval($str)){     # please note this has "zero case" caveat: in PHP and Perl 0 = 'false'
   print $val;             # so $val will not get 0 if input string is '0' (zero)
}
```

Python and Ruby use similar to Java fatal behaviour. Is that good? Perhaps sometimes. However in many cases returning something is better than nothing. Application may not do exactly what's expected but it may be considered to be better than crash. Perhaps for an application to keep running despite minor error instead of terminating. Maybe particular part of application is not too important to try-catch absolutely everything. Many examples of this are seen in web applications when seemingly innocent operation is in fact a fatal failure point leading to application crash. Several times Java and Python web-apps made by different teams had to be troubleshooted, in different companies, in different time but all of them used to crash on string transformations because of uncatched/unhandled exceptions when unexpected character came from database. Needless to say this was causing a great deal of frustration for users of those applications. One may argue that developers created those applications were incompetent. Could be. However development approach enforced by necessity of catching all possible exceptions is troublesome, difficult and slow. Obviously it clutters the code by generating 'noise' and implies a routine not strictly related to application's logic. Forgiving nature of Perl better match Test Driven Development when developer is not distracted with try-catch and therefore can concentrate on making code better, create more tests, check input values etc.



ParseInt comparison

String (str)	Java
	Integer.parseInt(str)
	or Integer.valueOf(str)
		PHP
		intval($str)
			Python
			int(str)
				Ruby
				str.to_i
					Ruby
					Integer(str)
						Perl
						int($str)
							C++
							istringstream buffer(str);
							double val;
							buffer >> val;
								C++
								istringstream buffer(str);
								int val;
								buffer >> val;
									C++
									double val=atoi(str)
										C++
										int val=atoi(str)

" 1111"	exception	OK	OK	OK	OK	OK	OK	OK	OK	OK
"10.0"	exception	OK	exception	OK	exception	OK	OK	OK	OK	OK
"10000000000"	exception	incorrect: 2147483647	OK	OK	OK	OK	OK (1e+10)	incorrect: 134520252	incorrect: 2.14748e+09	incorrect: 2147483647
"2e+2"	exception	incorrect: 2	exception	incorrect: 2	exception	OK	OK (200)	incorrect: 2	incorrect: 2	incorrect: 2
"-10"	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
"+10"	exception	OK	OK	OK	OK	OK	OK	OK	OK	OK
"asdasd"	exception	0	exception	0	exception	0	0	incorrect: 134520248	0	0
"0.0"	exception	incorrect: No value parsed	exception	OK	exception	OK	OK	OK	OK	OK
"00"	OK	incorrect: No value parsed	OK	OK	OK	OK	OK	OK	OK	OK
"2+3"	exception	2	exception	2	exception	2	2	2	2	2

[1] 2e+2=2*10 [2]=200

FIGURE 19 PARSE INT COMPARISON

Java has the most number of exceptions to handle - of course one may handle them as one but, as demonstrated in this example, a usable value can be parsed in most cases so if you want to do a good job you have to do it yourselves, for every case. Java is the only language which couldn't extract value from "+10".

Python is slightly smarter with recognising numbers in strings.

Ruby has two different methods to do the job - it is confusing which one is better.

PHP silently parses incorrect values.

Complexity and power of C++ vividly manifested in this example: you can choose from 4 different ways to parse a value from string but as soon you know which one of them is right, results are nearly perfect.
Since return value has to be a number, it returns 0 for non-numeric strings so it can be treated as exception to somehow determine if it was an error or an actual value.

Perl demonstrated *perfect* result. From the first look you may see that it's almost similar to C++: it returns 0 from non-numeric string. However with standard

```
use warnings;
```

a non-fatal warning will be issued: "*Argument "asdasd" isn't numeric in int at ./tst.pl line 8*." This warning can be converted to fatal with

```
use warnings FATAL=>'numeric';
```

Now we have an exception to catch like in the following example:

```
#!/usr/bin/perl
{ use warnings FATAL=>'numeric';
   my $str="asdasd";
   my $num=eval {int $str};
   if(defined $num){
      print "we got it - it's $num";
   }else{
      print "error: $@";
      # with "use English;" the line above could look like: print "error: $EVAL_ERROR";
   }
}
```

There are some important things to note:

- Fatal exception is enabled by developer's decision
  - Only for particular problem;
  - Only for particular block, so exception scope is strictly defined
- Only core language functionality used
- It works perfectly, including "zero case" and "2+3"
- It provides human-readable explanation of failure
- It extracts all usable values
- With minimal effort

So with just core Perl functionality it is possible to do the job a lot easier than with other languages. Not only this - the great flexibility of Perl is that one can use modules to introduce different styles of exception handling - you're not bound to the example above. With Try::Tiny (not the only module of such) you can use almost "traditional" Java's try-catch syntax in Perl:

```
#!/usr/bin/perl
use Try::Tiny;
use warnings FATAL=>'numeric';


   my $str="asdasd";
   my $num = try {
                int $str;
           } catch {
                die "error: $_";
           };
   print q{we got it - it's },$num;
```

Some links below might be interesting in order to compare languages' syntax:
Compare structure of Perl, Ruby, Python, Java and PHP
Wikipedia: Exception handling syntax

Notes (per language)

PHP

PHP is not a universal language. Perhaps it may be considered for web development only. Another problem with PHP is administration needed to configure runtime for different applications. Some PHP applications have different expectations regarding notorious "Magic quotes" runtime parameter.

Runtime is fast but not very compact. PHP has reputation of lightweight and fast language. While first happen to be false (PHP memory usage is quite big comparing with Python, Ruby and Perl5) it is a close second after Perl5 in Performance.

In some situations PHP functions cannot be trusted as demonstrated in "parsing integer from string" example.

Ruby

Ruby is universal but relatively young language. Its availability on different platforms is still limited and history of introducing backward incompatible changes makes development and maintenance unnecessary complicated. Performance and memory usage of Ruby and Python are close to each other. While Ruby is slightly faster, Python utilises memory better.

Python

Python is ripe and universal language. It stands strong enough during this test. However Python is interpreting white spaces and tabs. This particular 'feature looks unnecessary and silly especially after so much being said about importance of separation presentation from logic. Presentation *is* logic in Python. Python enforces certain way of formatting code in the rudest way one can imagine. Python is better than Java. Its "whitespace as constraint" could make reading/writing code harder. The only explanation for such strange Python's feature is that one can literally see the code flow pretty much the way interpreter see it. It is doubtful that good coding style can be effectively enforced - readable code formatting can be easily achieved with other languages through exercising best practice guidelines. In a way Python use military dress code - all applications should wear the same uniform. How this can make programming task easier? The more freedom programming language gives one - the better.

Perl5

Perl5 demonstrated amazing performance and memory usage far beyond all other languages tested. It proved to be most optimised, ripe and stable language. While some people believe it to be the most advanced programming language in the world it is clearly a very good choice.

Perl proved to be an extremely effective, highly optimised language.

Perl has a massive library of reusable code.

Perl is mature: it's 23 years old; (Perl5 is 17 years old).

Perl is very portable.

Perl is elegant and flexible.

Unfortunately Perl is often misunderstood because of widespread myths misrepresenting language capabilities. Typically those myths are product of ignorance and/or lack of knowledge. Some of those myths:

Myth: Perl is UNIX shell on steroids.

This is really an insult to Perl which is much more than this. In year 2010 Perl is a very mature and universal language with perhaps largest library of reusable code available. In Perl you can write GUI applications, web applications, systems daemons etc. It is possible to pack Perl's application, runtime and libraries to windows executable and distribute as single .EXE file. Perl's object oriented features and flexibility are far beyond perhaps any other language. Learn more about Modern Perl (presentation).

Myth: Perl is "write once - read never"

Perls often falsely accused of lack of readability. With certain discipline one can develop clear, understandable and maintainable code in any language. It's all a matter of learning good habits like commenting the code (especially if one is not the only developer) or choosing meaningful long names for variables etc. It comes with experience. One can't blame programming language for lack of clarity in one’s code just like one can’t blame natural language for its inappropriate use. If one’s Perl code is not beautiful they are doing it wrong - there is another, nice way.

Most people that complain about syntax have none or very little experience in Perl

There is a very good presentation Perl Myths 2009 where Tim Bunce is explaining some common Perl misunderstandings and revealing some of Perl's powers.

Perl is truly language of freedom. It gives amazing power and has features, non existing in other languages. Those powers can be used to create nice, tidy, clean and yet effective and concise code. Of course same powers can be used to write obfuscated code but, again, this is not a language problem because it is also possible with other languages. This is best explained the by creator of Perl himself (emphasis added):

The very fact that it's possible to write messy programs in Perl is also what makes it possible to write programs that are cleaner in Perl than they could ever be in a language that attempts to enforce cleanliness. The potential for greater good goes right along with the potential for greater evil. A little baby has little potential for good or evil, at least in the short term. A President of the United States has tremendous potential for both good and evil.

A good language is measured by how much freedom it gives, not by how much it can coerce others to do what one wants.

Java

Just like Perl, Java is a subject of numerous myths misrepresenting its real position. Despite commercial popularity there are multiple problems with the language:

* Poor memory management (garbage collection):

IMHO Java suffers from a garbage collection problem. If one doesn't allocate objects and maybe use only static methods, Java can be quite fast. But when you start creating huge amounts of objects (like required when working with Java's String class) its memory use and performance are getting worse and worse.

In theory GCs should be at least as fast as manual memory management or reference counting (which Python uses). Instead of wasting time for memory management while the program is working, it defers the memory management until the program is idle or it runs out of memory. Unfortunately on today's systems, memory is extremely slow and CPU cycles are cheap, and this is why the GC theory does not work. The Java VM constantly trashes the cache because it does not re-use memory fast enough. Instead it takes new (usually uncached) memory for new objects und defers freeing the unused memory of old objects (that are in the cache). This is probably the worst thing that one can do to the cache. A good VM would try to re-use memory as soon as possible, to increase the chances that it is still in cache (like Python's refcounter). Java does the opposite.

To make things worse, the VM seems to lack any coordination with the kernel. When the system is running out of RAM and needs to swap, the logical action for the VM would be to start the garbage collector. It doesn't however, and instead it starts allocating the new memory, forcing the kernel to move the old (unused) memory into the swap space! And when the VM finally decides to start the GC it will go through all the unused memory that is now in the swap, causing it be reloaded and possibly moving more frequently used memory back in the swap, only to re-load it again later. How much worse can it get?

Historically Java was successful partially because developers found it attractive comparing to C due to "automatic" memory management. It's turned to be a Java's greatest weakness. In C memory should me managed by developer to the contrast to Java where memory usually managed by systems administrator. In numerous papers explaining sophisticated garbage collection you may find dozens(!) parameters for memory tuning. And trust me, because Java developers usually cannot predict application's behaviour under load the only reliable way to configure memory management for particular application is to test, change parameter(s) and test again and again. Sometimes it helps. But defaults often not good enough, and it's too easy to make a mistake. Despite configuring Java "automatic" memory usage, developers can do very little. Java applications are handicapped by default.

Verbosity is bad because code is read more times than its written therefore verbosity increases effort needed to maintain code.

Java verbosity hurts both maintaining and development.

Usually Java developers claim Java code is easier to develop/maintain. I failed to discover any particular Java language feature to support that claim. Java's makes developers to do a lot of work even for simplest tasks.

Java's bad performance and memory usage are not compensated by any particular language feature(s). It is far behind other languages in both performance and memory usage/management.

Time needed to tweak and test memory management together with maintenance and troubleshooting efforts are horrifying.

2.3.1.3 Ruby backend features

Based on the information collected above ruby was chosen as the backend language.

1. **Free format** - One can start writing a program from any line and column.
2. **Case sensitive** - Lowercase letters and uppercase letters are distinct. The keyword **end**, for example, is completely different from the keyword **END**.
3. **Comments** - Anything following an unquoted **#**, to the end of the line on which it appears, is ignored by the interpreter. Also, to facilitate large comment blocks, the ruby interpreter also ignores anything between a line starting with **=begin** and another line starting with **=end**. This only works if the = signs are the first characters of each line.
4. **Statement delimiters** - Multiple statements on one line must be separated by semicolons, but they are not required at the end of a line; a linefeed is treated like a semicolon. If a line ends with a backslash (\), the linefeed following it is ignored; this allows you to have a single logical line that spans several lines.
5. **Keywords** - Also known as reserved words (around 42 of them) in Ruby typically cannot be used for other purposes. You may be used to thinking that a false value may be represented as a zero, a null string, a null character, or various other things. But in Ruby, all of these *values* are true; in fact, *everything is true* except the reserved words **false** and **nil**. Keywords would be called "reserved words" in most languages and they would never be allowed as identifiers. The Ruby parser is flexible and does not complain if you prefix these keywords with @, @@ or $ prefixes, or sigils, and use them as instance, class or global variable names, respectively. The best practice is to treat these **keywords** as reserved.
6. Pure OOP
7. Support for Reflection – can query Ruby objects about themselves
8. Duck Typing – Ruby “ducks” the issue of typing, letting the type of a variable be determined by its value.
9. **It's object-oriented.** What does that mean? Well, for every ten programmers, there are twelve opinions as to what OOP is. I will leave it user’s judgment. But for the record, Ruby does offer encapsulation of data and methods within objects and allows inheritance from one class to

another; and it allows polymorphism of objects. Unlike some languages (C++, Perl 5, etc.) Ruby was designed from the beginning to be object-oriented.

10. **It's a *pure* OOP language.** This means that everything, including primitive data types such as strings and integers, is represented as an object. There is no need for wrapper classes such as Java has. And in addition, even constants are treated as objects, so that a method may be invoked with, for example, a numeric constant as a receiver.
11. **It's a dynamic language.** For people only familiar with more static languages such as C++ and Java, this is a significant conceptual leap. It means that methods and variables may be added and redefined at runtime. It obviates the need for such features as C's conditional compilation (**#ifdef**), and makes possible a sophisticated reflection API. This in turn allows programs to become more "self-aware" -- enabling runtime type information, detection of missing methods, hooks for detecting added methods, and so on. Ruby is related to Lisp and Smalltalk in this respect.
12. **It's an interpreted language.** This is a complex issue, and deserves several comments. It can be argued that performance issues make this a negative rather than a positive. To this concern, I reply with these observations: 1. First and foremost: A rapid development cycle is a great benefit, and it is encouraged by the interpreted nature of Ruby. 2. How slow is too slow, anyway? Do some benchmarks before you call it slow. 3. Processors are getting faster every year. 4. If the speed is needed, one can write part of code in C. 5. Finally, in a sense, it is all a moot point, since no language is inherently interpreted. There is no law of the universe that says a Ruby compiler cannot be written.
13. **It understands regular expressions.** For years, this was considered the domain of UNIX weenies wielding clumsy tools such as **grep** and **sed**, or doing fancy search-and-replace operations in **vi**. Perl helped change that, and now Ruby is helping, too. More people than ever recognize the incredible power in the super-advanced string and text manipulation techniques.
14. **It's multi-platform.** It runs on Linux and other UNIX variants, the various Windows platforms, BeOS, and even MS-DOS. If my memory serves me, there's an Amiga version.
15. **It's derivative.** This is a good thing? Outside of the literary world, yes, it is. Isaac Newton said, "If I have seen farther than others, it is because I stood on the shoulders of giants." Ruby certainly has stood on the shoulders of giants. It borrows features from Smalltalk, CLU, Lisp, C, C++, Perl, Kornshell, and others. The principles I see at work are: 1. Don't reinvent the wheel. 2. Don't fix what isn't broken. 3. Finally, and especially: Leverage people's existing knowledge.

You understand files and pipes in UNIX? Fine, you can use that knowledge. You spent two years learning all the **printf** specifiers? Don't worry, you can still use **printf**. You know Perl's regex handling? Good, then you've almost learned Ruby's.

16. **It's innovative.** Is this in contradiction to #7 above? Well, partly; every coin has two sides. Some of Ruby's features are truly innovative, like the very useful concept of the mix-in. Maybe some of these features will be borrowed in turn by future languages. (**Note:** A reader has pointed out to me that LISP had mix-ins at least as far back as 1979. That's purely ignorance on my part; I shall have to find a better example, and make sure of it.)
17. **It's a Very High-Level Language (VHLL).** This is subject to debate, because this term is not in widespread use, and its meaning is even more disputable than that of OOP. When I say this, I mean that Ruby can handle complex data structures and complex operations on them with relatively few instructions, in accordance with what some call the Principle of Least Effort.
18. **It has a smart garbage collector.** Routines like **malloc** and **free** are only last night's bad dream. You don't even have to call destructors. Enough said.
19. **It's a scripting language.** Don't make the mistake of thinking it isn't powerful because of this. It's not a toy. It's a full-fledged language that happens to make it easy to do traditional scripting operations like running external programs, examining system resources, using pipes, capturing output, and so on.
20. **It's versatile.** It can do the things that Kornshell does well and the things that C does well. You want to write a quick ten-line hack to do a one-time task, or a wrapper for some legacy programs? Fine. You want to write a web server, a CGI, or a chess program? Again, fine.
21. **It's thread-capable.** You can write multi-threaded applications with a simple API. Yes, even on MS-DOS.
22. **It's open-source.** You want to look at the source code? Go ahead. Want to suggest a patch? Go ahead. You want to connect with a knowledgeable and helpful user community, including the language creator himself? You can.
23. **It's intuitive.** The learning curve is low, and once you get over the first hump, you start to "guess" how things work� and your guesses are often correct. Ruby endeavors to follow the Principle of Least Astonishment (or Surprise).
24. **It has an exception mechanism.** Like Java and C++, Ruby understands exceptions. This means less messing with return codes, fewer nested if statements, less spaghetti logic, and better error handling.

25. **It has an advanced Array class.** Arrays are dynamic; one doesn't have to declare their size at compile-time as in, say, Pascal. You don't have to allocate memory for them as in C, C++, or Java. They're objects, so you don't have to keep up with their length; it's virtually impossible to "walk off the end" of an array as you might in C. Want to process them by index? By element? Process them backwards? Print them? There are methods for all these. Want to use an array as a set, a stack, or a queue? There are methods for these operations, too. Want to use an array as a lookup table? That's a trick question; you don't have to, since we have hashes for that.
26. **It's extensible.** You can write external libraries in Ruby or in C. In addition, you can modify the existing classes and objects at will, on the fly.
27. **It encourages literate programming.** You can embed comments in your code which the Ruby documentation tool can extract and manipulate. (Real fans of literate programming may think this is pretty rudimentary.)
28. **It uses punctuation and capitalization creatively.** A method returning a Boolean result (though Ruby doesn't call it that) is typically ended with a question mark, and the more destructive, data-modifying methods are named with an exclamation point. Simple, informative, and intuitive. All constants, including class names, start with capital letters. All object attributes start with an **@** sign. This has the pragmatism of the old "Hungarian notation" without the eye-jarring ugliness.
29. **Reserved words aren't.** It's perfectly allowable to use an identifier that is a so-called "reserved word" as long as the parser doesn't perceive an amibiguity. This is a breath of fresh air.
30. **It allows iterators.** Among other things, this makes it possible to pass blocks of code to your objects in such a way that the block is called for each item in the array, list, tree, or whatever. This is a powerful technique that is worth exploring at great length.
31. **It has safety and security features.** Ruby borrows Perl's concept of tainting and allows different levels of control (levels of paranoia?) by means of the **$SAFE** variable. This is especially good for CGI programs that people will try to subvert in order to crack the web server.
32. **It has no pointers.** Like Java, and with a grudging nod to C++, Ruby does not have the concept of a pointer; there is no indirection, no pointer arithmetic, and none of the headaches that go with the syntax and the debugging of pointers. Of course, this means that real nuts-and-bolts system programming is more difficult, such as accessing a control-status register for a device; but that can always be done in a C library. (Just as C programmers drop into assembly when necessary, Ruby programmers drop into C when they have to!)

33. **It pays attention to detail.** Synonyms and aliases abound. You can't remember whether to say **size** or **length** for a string or an array? Either one works. For ranges, is it **begin** and **end**, or **first** and **last**? Take your pick. You spell it **indices**, and your evil twin spells it **indexes**? They both work.
34. **It has a flexible syntax.** Parentheses in method calls can usually be omitted, as can commas between parameters. Perl-style quotes allow arrays of strings without all the quotation marks and commas. The **return** keyword can be omitted.
35. **It has a rich set of libraries.** There is support for threads, sockets, limited object persistence, CGI programs, server-side executables, DB files, and more. There is some support for Tk, with more on the way.
36. **It has a debugger.** In a perfect world, we wouldn't need debuggers. This is not a perfect world.
37. **It can be used interactively.** Conceivably it could be used as a sort of "Kornshell squared." (This is the most contested item on this page, and I am forced to concede that Ruby is not really good as a shell. I still maintain, though, that a Ruby-based shell would be a good thing.)
38. **It is concise.** There are no superfluous keywords such as Pascal's **begin**, **then** after **if**, **do** after **while**. Variables need not be declared, as they do not have types. Return types need not be specified for methods. The return keyword is not needed; a method will return the last evaluated expression. On the other hand... it is not so cryptic as C or Perl.
39. **It is expression-oriented.** You can easily say things like **x = if a<0 then b else c end**.
40. **It is laced with syntax sugar.** (To paraphrase Mary Poppins: A spoonful of syntax sugar helps the semantic medicine go down.) If you want to iterate over an array **x** by saying **for a in x**, you can. If you want to say **a += b** instead of **a = a + b**, you can. Most operators are really just methods with short, intuitive names and a more convenient syntax.
41. **It has operator overloading.** If I am not mistaken, this originated long ago in SNOBOL, but was popularized more recently by C++. It can be overdone or misused, but it can be nice to have. Additionally, Ruby defines the assignment version of an operator automagically; if you define +, you get += as a bonus.
42. **It has infinite-precision integer arithmetic.** Who cares about **short**, **int**, **long**? Just use a **Bignum**. Admit it, you always wanted to find the factorial of 365. Now you can.
43. **It has an exponentiation operator.** In the old days, we used this in BASIC and FORTRAN. But then we learned Pascal and C, and learned how evil this operator was. (We were told we didn't

even know how the evaluation was done -- did it use logarithms? Iteration? How efficient was it?) But then, do we really care? If so, we can rewrite it ourselves. If not, Ruby has the good old ****** operator you loved as a child. Enjoy it.

44. **It has powerful string handling.** If you want to search, substitute, justify, format, trim, delimit, interpose, or tokenize, you can probably use one of the built-in methods. If not, you can build on them to produce what you need.
45. **It has few exceptions to its rules.** The syntax and semantics of Ruby are more self-consistent than most languages. Every language has oddities, and every rule has exceptions; but Ruby has fewer than you might expect.

2.3.1.4 RubyGems

RubyGems is a package manager for the Ruby programming language that provides a standard format for distributing Ruby programs and libraries (in a self-contained format called a "gem"), a tool designed to easily manage the installation of gems, and a server for distributing them.

The interface for RubyGems is a command-line tool called *gem* which can install libraries and manage RubyGems.[1] RubyGems integrates with Ruby run-time loader to help find and load installed gems from standardized library folders.Though it is possible to use a private RubyGems repository, the public repository is most commonly used for gem management. There are about 123,000 gems in the public repository with over 9.8 billion downloads.[2]

The public repository helps users find gems, resolve dependencies and install them. RubyGems is bundled with the standard Ruby package as of Ruby 1.9.

2.3.2 Framework

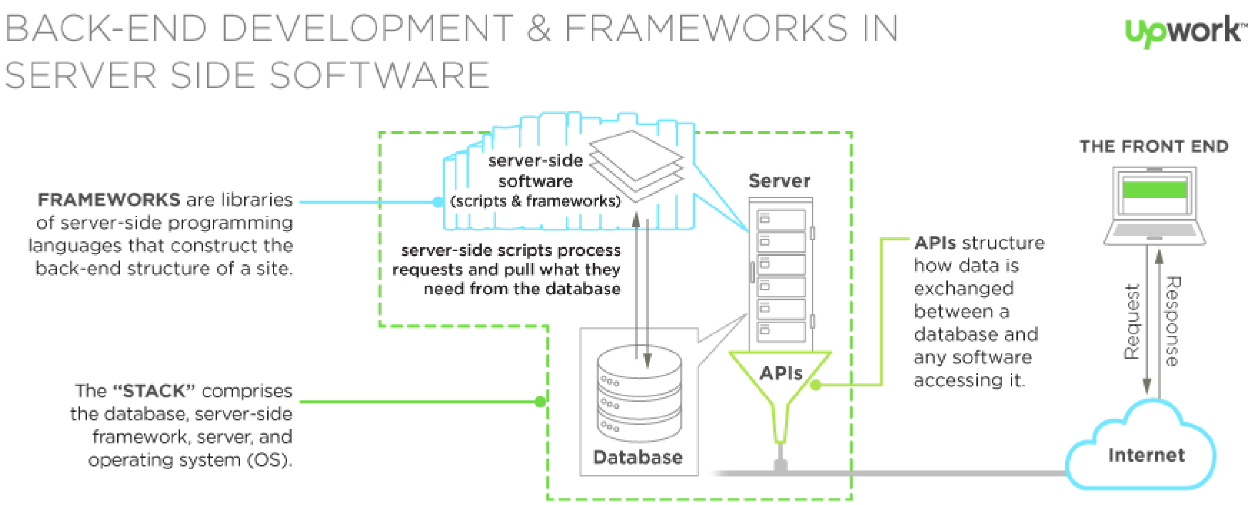

BACK-END DEVELOPMENT & FRAMEWORKS IN SERVER SIDE SOFTWARE
upwork
server-side software (scripts & frameworks)
FRAMEWORKS are libraries of server-side programming languages that construct the back-end structure of a site.
server-side scripts process requests and pull what they need from the database
Server
APIs
APIs structure how data is exchanged between a database and any software accessing it.
THE FRONT END
Request
Response
The "STACK" comprises the database, server-side framework, server, and operating system (OS).
Database
Internet

FIGURE 20 RELATION BETWEEN FRAMEWORK AND BACKEND

Rails is a web application development framework written in the Ruby language. It is designed to make programming web applications easier by making assumptions about what every developer needs to get started. It allows you to write less code while accomplishing more than many other languages and frameworks. Experienced Rails developers also report that it makes web application development more fun.

Rails is opinionated software. It makes the assumption that there is a "best" way to do things, and it's designed to encourage that way - and in some cases to discourage alternatives. If you learn "The Rails Way" you'll probably discover a tremendous increase in productivity. If you persist in bringing old habits from other languages to your Rails development, and trying to use patterns you learned elsewhere, you may have a less happy experience.

The Rails philosophy includes two major guiding principles:

- **Don't Repeat Yourself:** DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." By not writing the same information over and over again, our code is more maintainable, more extensible, and less buggy.

- **Convention Over Configuration:** Rails has opinions about the best way to do many things in a web application, and defaults to this set of conventions, rather than require that you specify minutiae through endless configuration files.

Features:

Ruby on Rails includes features that help in increasing developer productivity. Some of the main features include the following:

• MVC architecture: Ruby on Rails is based on the MVC (Model View Controller) architecture that enables the data to be separated from presentation.

• Database Access Library: Ruby on Rails includes a database access library - Active Record - that simplifies data handling in databases. Active Record automatically maps tables to classes and rows to objects.

• Libraries for common tasks: Ruby on Rails includes a host of libraries that simplify the coding of common programming tasks such as form validations, sessions management, etc.

• AJAX Library: An extensive library of AJAX functions is provided in the Rails framework. Ruby code can be used to generate AJAX code. The associated java scripting required for AJX gets generated automatically.

• Convention over configuration: Ruby on Rails does not have any XML configuration files. It includes simple programming conventions that can be used to specify the configuration parameters.

• Customized URL: Custom or Search Engine Friendly URLs can be developed using the Ruby on Rails framework.

• Debugging: Detailed error logs are provided, making it easier to debug applications.

• Components: Components can be used to store reusable code. Components can be included to modularize templates.

Ruby on Rails – The Good

Mature Framework

The more I develop on Rails, the more I really appreciate and love it. I've found that it enables us to create higher quality products for clients much faster, that are more maintainable. It's a mature and stable framework that many large companies are comfortable with introducing into their environments. Compare this with the PHP ecosystem that has many frameworks — there’s a risk of selecting a framework and finding that it’s just not that well supported several years from now (we made this mistake).

Speed and Development Joy

I absolutely love working with Rails because as a development platform, it is extremely automated. So many menial tasks have been automated so that you just focus entirely on solving the business problem instead of hacking your way around a framework. Some things really going for Rails in this regard are:

- **Generators/Scaffolding** – Provide a very good starting point for developing around. Some PHP frameworks now provide scaffolding features.
- **Gems/Plugins** – the Rails community provides a wealth of plugins as Ruby Gems that you simply add to your project Gemfile and install. This significantly accelerates development and maintenance time as you’re not trying to integrate disparate libraries, it’s already done for you.
- **Active Record ORM** – Of all the ORM’s I have used (for PHP I’ve used DataMapper DMZ, Fuel/Kohana, Doctrine), ActiveRecord in Ruby on Rails is simply the best. It actually works and is remarkably straightforward to use.
- **Integrated testing tools** – I love it that out of the gate, Rails has a testing framework that can be used. In PHP, many frameworks have only recently been trying to integrate PHPUnit, to varying degrees of success.

As a programming language, Ruby is really quite an amazing language. Unlike PHP, it really is Object Oriented from the ground up. Its code is very concise and powerful. Gems (extensions) enable you to bolt on needed functionality. After coding in Ruby, I find coding in PHP (or anything else really) rather tedious.

Ruby on Rails – The Bad

Steep Learning Curve

My main beef with Ruby on Rails is that it actually has a steep learning curve. Do not believe the hype that says that it is really easy. They will show you podcasts where you build a simple blog application using scaffolding and voila! Instant website. Nothing could be further from the truth. Rails is seen to be easy because they have automated many things in the framework — this does not make it easy to understand.

Developing a Rails app and deploying it actually requires you to know the full stack. With PHP, you can just cobble together some inline PHP code, FTP it to a server and off you go. In Rails, you really need to know what you are doing from the web server (Apache or NginX), setting up Phusion Passenger and database engine. Then you have to deal with the asset pipeline process to prepare your app to run in Production mode. It’s not as simple as running it in production mode — you have to precompile your assets and make sure files are actually there. If they are not, Rails will simply blow up and you have to find out why by accessing the Rails logs.

Compared to PHP, Rails is also unfriendly when it comes to errors. With PHP, it will spit out errors at you in development and the error messages actually make sense. Typically a page will render but the part with the error will show you which line the error occurred and the message is useful. In Rails, typically the whole app blows up.

One last thing to throw in is that good Ruby on Rails developers tend to be polyglots. They are able to pick up and learn many languages. While beginners are battling to just learn Ruby, Rails people are using CoffeeScript instead of Javascript, SCSS (or LESS), and Slim or HAML. For a newcomer to Rails, part of the steep curve is not just learning Ruby and the Rails framework, but all these other languages as well!

Ruby is not an easy language

I’m sorry to offend some people here, but Ruby is simply not as straightforward as PHP to learn. It is by all intents an extremely powerful language. I choose to use Ruby simply because as a developer I feel it is a much better language than PHP. But from a learning perspective, it is not.

Ruby has many features that are simply not straightforward for a beginner programmer to understand. One such concept are blocks, procs and lambdas, which Rails uses heavily. The classic Ruby on Rails example I will use is for creating a form:

```
<%= form_for @user do |f| %>
```

```
  <%= f.label :first_name %>
  <%= f.text_field :first_name %>
<% end %>
```

If you’re new to Ruby, you can be forgiven for saying, “Wait a minute….what’s f?” Yes sir. Welcome to blocks.

Here’s a bit of an extreme example:

```
(0...8).map{65.+(rand(25)).chr}.join
```

Even as an experienced programmer, I went crosseyed when seeing the above line of code. It’s very simple actually – generate an 8 character random string.

Another area is meta programming. Here’s an example:

```
class Client < ActiveRecord::Base
  has_one :address
  has_many :orders
  has_and_belongs_to_many :roles
end
```

I’ve taught Ruby on Rails to experienced developers and this always trips them up. What exactly is has_one, has_many and has_and_belongs_to_many? It looks like it’s some kind of reserved keyword or declaration as these are not encapsulated in a method. However, in Ruby, ALL code is executed. Every line of code is executed, so has_one, has_many and has_and_belongs_to_many are just methods that execute when the class is declared.

Finally another thing that makes Ruby challenging for beginners is its loose syntax. Let’s look again at the above code. It’s not obvious (to a beginner) that has_one :address is invoking a method because the brackets are missing from the method invocation. In PHP, the syntax is stricter and this makes it simpler for beginners to know what’s what.

As a language, especially if you’re coming in from others such as C/Java/PHP, Ruby is challenging and it will bend your mind. Once you’re up and running though, it’s fantastic and many who have taken the leap really enjoy coding with it.

2.3.3 Frontend

HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

What is HTML?
HTML is the language for describing the structure of Web pages. HTML gives authors the means to:
Publish online documents with headings, text, tables, lists, photos, etc.
Retrieve online information via hypertext links, at the click of a button.
Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
Include spread-sheets, video clips, sound clips, and other applications directly in their documents.
With HTML, authors describe the structure of pages using *markup*. The *elements* of the language label pieces of content such as “paragraph,” “list,” “table,” and so on.

What is CSS?
CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language. The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. This is referred to as the *separation of structure (or: content) from presentation.*

2.3.4 Database

A **database** is an organized collection of data. It is the collection of schemas, tables, queries, reports, views, and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information, such as modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

2.3.4.1 About SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced below 300KiB. SQLite can also be made to run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists and minute-by-minute chronologies of code changes.

The SQLite code base is supported by an international team of developers who work on SQLite full-time. The developers continue to expand the capabilities of SQLite and enhance its reliability and performance while maintaining backwards compatibility with the published interface spec, SQL syntax, and database file format. The source code is absolutely free to anybody who wants it, but professional support is also available.

The SQLite project was started on 2000-05-09. The future is always hard to predict, but the intent of the developers is to support SQLite through the year 2050. Design decisions are made with that objective in mind.

We the developers hope that you find SQLite useful and we entreat you to use it well: to make good and beautiful products that are fast, reliable, and simple to use. Seek forgiveness for yourself as you forgive others. And just as you have received SQLite for free, so also freely give, paying the debt forward.

2.3.4.2 About SQL

SQL (Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

SQL was one of the first commercial languages for Edgar F. Codd's relational model, as described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks." Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments.

2.3.4.3 ORM

Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to construct their own ORM tools.

In object-oriented programming, data-management tasks act on object-oriented (OO) objects that are almost always non-scalar values. For example, consider an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on. The address-book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as SQL database management systems (DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database(and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping implements the first approach.

The heart of the problem involves translating the logical representation of the objects into an atomized form that is capable of being stored in the database, while preserving the properties of the objects and their relationships so that they can be reloaded as objects when needed. If this storage and retrieval functionality is implemented, the objects are said to be persistent.

2.3.4.4 Active records

In software engineering, the **active record pattern** is an architectural pattern found in software that stores in-memory object data in relational databases. It was named by Martin Fowler in his 2003 book *Patterns of Enterprise Application Architecture*. The interface of an object conforming to this pattern would include functions such as Insert, Update, and Delete, plus properties that correspond more or less directly to the columns in the underlying database table.

The active record pattern is an approach to accessing data in a database. A database table or view is wrapped into a class. Thus, an object instance is tied to a single row in the table. After creation of an object, a new row is added to the table upon save. Any object loaded gets its information from the database. When an object is updated, the corresponding row in the table is also updated. The wrapper class implements accessor methods or properties for each column in the table or view.

This pattern is commonly used by object persistence tools and in object-relational mapping (ORM). Typically, foreign key relationships will be exposed as an object instance of the appropriate type via a property.

2.3.5 Operating system (OS)

An **operating system** (**OS**) is system software that manages computer hardware and software resources and provides common services for computer programs. All computer programs, excluding firmware, require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware,[1][2] although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers.

The dominant desktop operating system is Microsoft Windows with a market share of around 83.3%. macOS by Apple Inc. is in second place (11.2%), and the varieties of Linux is in third position (1.55%).[3] In the mobile (smartphone and tablet combined) sector, according to third quarter 2016 data, Android by Google is dominant with 87.5 percent and a growth rate 10.3 percent per year, followed by iOS by Apple with 12.1 percent and a per year decrease in market share of 5.2 percent, while other operating systems amount to just 0.3 percent.[4] Linux distributions are dominant in the server and supercomputing sectors. Other specialized classes of operating systems, such as embedded and real-time systems, exist for many applications.

2.3.5.1 Linux

The Linux kernel originated in 1991, as a project of Linus Torvalds, while a university student in Finland. He posted information about his project on a newsgroup for computer students and programmers, and received support and assistance from volunteers who succeeded in creating a complete and functional kernel.

Linux is Unix-like, but was developed without any Unix code, unlike BSD and its variants. Because of its open license model, the Linux kernel code is available for study and modification, which resulted in its use on a wide range of computing machinery from supercomputers to smart-watches. Although estimates suggest that Linux is used on only 1.82% of all "desktop" (or laptop) PCs, it has been widely adopted for use in servers and embedded systems such as cell phones. Linux has superseded Unix on many platforms and is used on most supercomputers including the top 385. Many of the same computers are also on Green500 (but in different order), and Linux runs on the top 10. Linux is also commonly used on other small energy-efficient computers, such as smartphones and smartwatches. The Linux kernel is used in some popular distributions, such as Red Hat, Debian, Ubuntu, Linux Mint and Google's Android. (annexure 6)

2.3.6 Server

In computing, a **server** is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients, and a single client can use multiple servers. A client process may run on the same device or may connect over a network to a server on a different device.[1] Typical servers are database servers, file servers, mail servers, print servers, web servers, game servers, and application servers.

Client–server systems are today most frequently implemented by (and often identified with) the request–response model: a client sends a request to the server, which performs some action and sends a response back to the client, typically with a result or acknowledgement. Designating a computer as "server-class hardware" implies that it is specialized for running servers on it. This often implies that it is more powerful and reliable than standard personal computers, but alternatively, large computing clusters may be composed of many relatively simple, replaceable server components.

2.3.6.1 Localhost

In computer networking, **localhost** is a hostname that means *this computer*. It is used to access the network services that are running on the host via its loopback network interface. Using the loopback interface bypasses any local network interface hardware.

The local loopback mechanism is useful for testing software during development, independently of any networking configurations. For example, if a computer has been configured to provide a website, directing a locally running web browser to http://localhost may display its home page.

On most computer systems, *localhost* resolves to the IP address 127.0.0.1, which is the most commonly used IPv4 loopback address, and to the IPv6 loopback address ::1.

The name *localhost* is also a reserved top-level domain name, set aside to avoid confusion with the definition as a hostname. The IETF standards restrict domain name registrars from assigning the name *localhost* in registration procedures, such as for second-level domains.

Localhost:3000 - This is the Ruby on Rails development default Port.

2.3.6.2 WEBrick server

WEBrick is a Ruby library providing simple HTTP web servers . WEBrick was primarily written by Masayoshi Takahashi and Yuuzou Gotou, with contributions from other developers via the open source model of software development. It uses basic access authentication and digest access authentication for different kinds of servers that it can create - HTTP based server, HTTPS server, proxy server and virtual-host server. Construction of several non-HTTP servers such as the Day Time Server which uses the Daytime Protocol rather than the HTTP is also facilitated by WEBrick. It is used by the Ruby on Rails and Padrino frameworks to test applications in a development environment as well as production mode for small loads. It is now a part of Ruby standard library. (annexure 7)

2.3.6.3 Amazon cloud

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate them from common failure scenarios.

Amazon EC2 Benefits

3. Elastic Web-Scale Computing:
   Amazon EC2 enables you to increase or decrease capacity within minutes, not hours or days. You can commission one, hundreds, or even thousands of server instances simultaneously. Because this is all controlled with web service APIs, your application can automatically scale itself up and down depending on its needs.
4. Completely Controlled:
   You have complete control of your instances including root access and the ability to interact with them as you would any machine. You can stop any instance while retaining the data on the boot partition, and then subsequently restart the same instance using web service APIs. Instances can be rebooted remotely using web service APIs, and you also have access to their console output.
5. Flexible Cloud Hosting Services:
   You have the choice of multiple instance types, operating systems, and software packages. Amazon EC2 allows you to select a configuration of memory, CPU, instance storage, and the boot partition size that is optimal for your choice of operating system and application. For example, choice of operating systems includes numerous Linux distributions and Microsoft Windows Server.
6. Integrated
   Amazon EC2 is integrated with most AWS services such as Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), and Amazon Virtual

Private Cloud (Amazon VPC) to provide a complete, secure solution for computing, query processing, and cloud storage across a wide range of applications.

7. Reliable

   Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned. The service runs within Amazon's proven network infrastructure and data centers. The Amazon EC2 Service Level Agreement commitment is 99.95% availability for each Amazon EC2 Region.

8. Secure:

   Cloud security at AWS is the highest priority. As an AWS customer, you will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations. Amazon EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality for your compute resources.

9. Inexpensive:

   Amazon EC2 passes on to you the financial benefits of Amazon's scale. You pay a very low rate for the compute capacity you actually consume. See Amazon EC2 Instance Purchasing Options for more details.

10. Easy to Start:

    There are several ways to get started with Amazon EC2. You can use the AWS Management Console, the AWS Command Line Tools (CLI), or AWS SDKs. AWS is free to get started. To learn more, please visit our tutorials.

2.3.7 Git repository hosting service

GitHub is a web-based Git or version control repository and Internet hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub offers both plans for private and free repositories on the same account which are commonly used to host open-source software projects.[5] As of April 2017, GitHub reports having almost 20 million users and 57 million repositories, making it the largest host of source code in the world.

GitHub is mostly used for code.

In addition to source code, GitHub supports the following formats and features:

- Documentation, including automatically rendered README files in a variety of Markdown-like file formats (see README files on GitHub)
- Issue tracking (including feature requests) with labels, milestones, assignees and a search engine
- Wikis
- Pull requests with code review and comments
- Commits history
- Graphs: pulse, contributors, commits, code frequency, punch card, network, members
- Integrations Directory
- Unified and split diffs
- Email notifications
- Option to subscribe someone to notifications by @ mentioning them.
- Emojis
- GitHub Pages: small websites can be hosted from public repositories on GitHub. The URL format is http://*username*.github.io.
- Nested task-lists within files
- Visualization of geospatial data
- 3D render files that can be previewed using a new integrated STL file viewer that displays the files on a "3D canvas". The viewer is powered by WebGL and Three.js.
- Photoshop's native PSD format can be previewed and compared to previous versions of the same file.

Licensing of repositories

GitHub's Terms of Service do not require public software projects hosted on GitHub to meet the Open Source Definition. For that reason, it is essential for users and developers intending to use a piece of software found on GitHub to read the software license in the repository (usually found in a top-level file called "LICENSE", "LICENSE.txt", or similar) to determine if it meets their needs. The Terms of Service state, "By setting your repositories to be viewed publicly, you agree to allow others to view and fork your repositories.”

2.3.7.1 Git

Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano.

As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, independent of network access or a central server.

Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2. (annexure 8)

CHAPTER 3

3. METHODOLOGY

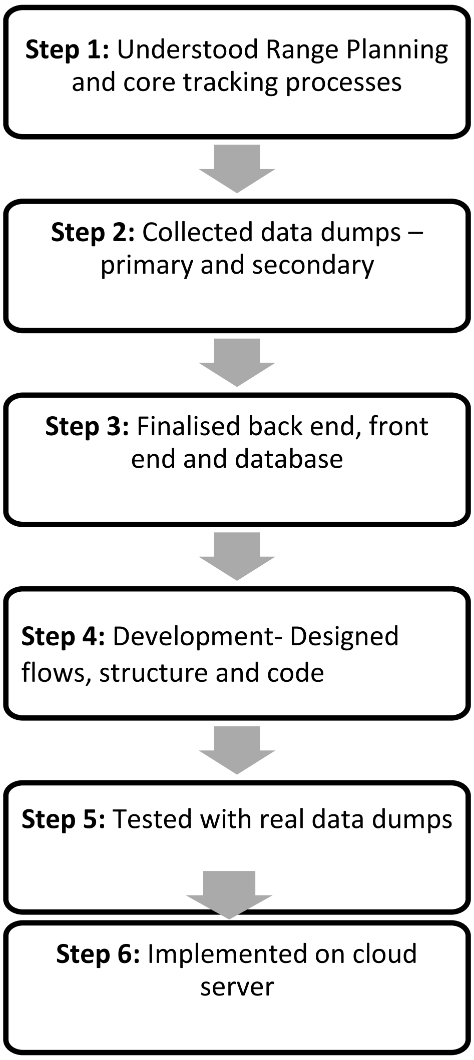

Step 1: Understood Range Planning and core tracking processes
Step 2: Collected data dumps – primary and secondary
Step 3: Finalised back end, front end and database
Step 4: Development- Designed flows, structure and code
Step 5: Tested with real data dumps
Step 6: Implemented on cloud server

FIGURE 21 METHODOLOGY FLOWCHART

3.1 Understanding of processes

1. To develop a range plan the previous season's sales data, projections and assumptions of the next season are taken and these are put in a template manually which has some formula driven fields.

2. For the core tracker, a core styles dump was received which is extracted weekly.

3.2 Collection of data

1. For the range planner Sales dumps of previous 2 seasons were obtained.(annexure 1)
   Assumptions and projections of next season were obtained.
   Lfl data was obtained for previous 2 years and current year (projected).
   All the fields required for the final range plan were marked.
   Formulae were obtained.
   Output desired was understood.

2. Sales data dump for core tracker was obtained.(annexure 4)
   Fields required for calculations were marked.
   Formulae were obtained.
   Output desired was understood.

3.3 Finalizing backend, frontend and database

Frontend, backend and database to be used for the web application were finalized based on the literature survey.
Frontend- HTML, CSS
Backend- Ruby on Rails (framework)
Database- SQLite

3.4 Development

1. **Wireframes**

a. Wireframes are low-fidelity, ‘bare-bones’ blueprints, usually presented with placeholders for final content, to be filled in at a later point in the design cycle. Wireframes aim to help represent what goes where in a design, without the design team having to spend too much time on the details.

In this stage wireframes were developed for the range planner. (annexure 10)

b. For the core tracker, templates with forms were made in Microsoft Access to test the queries to be used later, to understand how it would work and to give an idea of how it would look.

2. **Software**

Linux operating system was installed along with yakuake terminal and sublime text editor.

Ruby was installed, along with rvm 2.2.3, a rails was installed, a few ruby gems were installed and bundled. Gems are written in the gem file.

a. In the dump file a column with LFL flag of that year corresponding to the store was added in the end. In the software a database was created with the name range planner and 6 tables were created- Brands, Groups, category, seasons, new seasons and final plans.

Here brands in brand table, groups in group table, seasons in season table have their ids created during import.

From the previous season dump obtained the brand, group and category, seasons table were imported through some commands entered in the terminal.

From the projection and assumptions file, the new seasons table was populated.

All the tables are related to each other through brand, group and category tables which have their IDs for atomicity and faster execution.

- Fields in dump file:

- SALEDATE
- STORE_CODE
- STORE_NAME
- AREA_NAME
- REGION_NAME
- DISTRICT_NAME
- TSF_ENTITY_DESC

- DIVISION
- DIV_NAME
- GROUP_NO
- GROUP_NAME
- DEPT
- DEPT_NAME
- CLASS_ID
- CLASS_NAME
- SUBCLASS
- SUB_NAME
- PURCHASE_TYPE
- BRAND_ID
- BRAND_DESC
- BRAND_TYPE_DESC
- SUPPLIER
- SUP_NAME
- SEASON_ID
- SEASON_DESC
- PHASE_ID
- SALE_QTY
- CP
- SALE_MRP
- TAX
- DISC
- NET_SALES
- 15 16 lfl flag

1. Seasons table

- Fields chosen

- GROUP_NAME

- CLASS_NAME
- BRAND_DESC
- SALE_QTY
- CP
- SALE_MRP
- TAX
- DISC
- NET_SALES

- The aggregation is done on the basis of Brand, Group and Category.

1. Seasons table final fields to store and display

QTY	Assumption is taken a little higher than the previous season.
NOT	Assumption is taken a little higher than the previous season.
DMRP	Formula based calculation, DMRP = NET_SALES + TAXES
MRP	Assumption is taken a little higher than the previous season.
CP	Assumption is taken a little higher than the previous season.
AW16 **NOT MIX**	Formula, one particular BRAND, GROUP, and CATEGORY- NOT is taken, divided by the sum of the BRAND, GROUP including all categories, value and percentage is taken.
AW16 **LFL%**	From the sales dump only not values of lfl marked stores are added and then divided by 10^7. Where current year lfl is taken and the value of previous year lfl is subtracted from it and then divided by previous year lfl.

AW16 Overall%	From the sales dump only not values of lfl and nlfl marked stores are added and then divided by 10^7. Where current year overall lfl is taken and the value of previous year lfl is subtracted from it and then divided by previous year lfl.
AW16 LFL NOT	From the sales dump only not values of lfl marked stores are added and then divided by 10^7.

TABLE 1 SEASONS TABLE DESCRIPTION

2. New seasons table

Another dump is given for the assumption of the next season.

AW17 LFL Bud	Value is imported.
AW17 LFL%	Formula based calculation.
AW17 Overall%	Formula based calculation.
Brand Group Bud	Value is imported.
Class Mix	Value is imported.
Final Class Mix	Value is imported.
QTY	Formula based calculation.

NOT	Value is imported.
DMRP	Formula based calculation.
MRP	Formula based calculation.
CP	Formula based calculation.
MD%	Value is imported.
IM%	Formula based calculation.

TABLE 2NEW SEASONS TABLE DESCRIPTION

Formulae:-

--NewSeason.lfl = ((NewSeason.lfl_budget * NewSeason.final_class_mix) / Season.lfl_not)-100

--NewSeason.overall = ((NewSeason.not / Season.net_of_sales)-1)*100

--NewSeason.quantity = (NewSeason.mrp/ (Season.mrp * 1.05/Season.quantity))

-NewSeason.dmrp = NewSeason.not * 1.0525 # only 2 decimal places, round up

-NewSeason.mrp = (NewSeason.dmrp / (1- (NewSeason.md/100))) # only 2 decimal places, round up

*

NewSeason.cost_price = (NewSeason.mrp * (1 - (im/100))) # only 2 decimal places, round up

*

NewSeason.im = ((Season.mrp - Season.cost_price) / Season.mrp)*100 #no dec places, round up

-NewSeason.md = ((Season.mrp - Season.dmrp) / Season.mrp)*100 #no dec places, round up

3. Final Plans table:

Sell Thru Assumptions	Value is imported.
OTB Value	Formula based calculation.
OTB Qty	Formula based calculation.
DC Option	Value is imported.
DC Qty	Formula based calculation.
No Of Store	Value is imported.
Size Set	Value is imported.
Initial Allocation	Formula based calculation.

Replenishment Mix	Value is imported.
Dept/Option	Formula based calculation.
No of Option	Formula based calculation.

TABLE 3 FINAL PLANS TABLE DESCRIPTION

Formulae:-

FinalPlan.otb_value = (NewSeason.cost_price / (FinalPlan.sell_through_asumptions/100) / 0.95)

FinalPlan.otb_qty = (FinalPlan.otb_value / (NewSeason.cost_price / NewSeason.quantity)) #round up, no dec

FinalPlan.initial_alocation = FinalPlan.size_set * FinalPlan.number_of_stores

FinalPlan.dept_option = FinalPlan.initial_alocation * (1+ replenishment_mix)

FinalPlan.number_of_options = (FinalPlan.otb_qty/FinalPlan.dept_option) #round, no dec

b. A table was made in the web application which had the same fields as the core tracker template, all the data from the .csv file was imported into the application.

- Fields in core tracker

- LOCATION_CODE
- LOCATION_NAME
- LOCATION_GRADE
- LOCATION_TYPE
- REGION
- DEFAULT_WH
- BRAND_NAME
- GROUP_NAME
- DEPT_NAME
- CLASS
- SUB_CLASS
- MADE_UP_DESC
- STYLE
- COLOR
- SIZEE
- MRP
- ITEM_CODE
- EAN_CODE
- BASE_STOCK
- SOH
- ARP_MAX
- AVAILABLE_QTY
- INTRANSIT_QTY
- TSF_EXPECTED_QTY
- DISTRO_QTY
- ALLOC_NON_GRN
- OPEN_PO
- CY_SALEQTY
- PY_SALEQTY
- CY_SALEVALUE

- PY_SALEVALUE
- CY_MRP_SALEVALUE
- PY_MRP_SALEVALUE
- LAST_180DAYS_SALEQTY

A few fields were added in the core tracker which were formula based calculations:

a. ROS = LAST_180DAYS_SALEQTY/180
b. NOD(SOH) = SOH/ROS
c. NOD(SOH+OO) = SOH+OO/ROS

Where,
SOH = stock on hand
ROS= Return on sales
OO = open order

Core sales = SOH + Sales + NOD (b or c)

- Queries written for AVAILABILITY

1. Taking SOH values for ARP values > 0
2. Overall availability = ARP count for ARP > 0 / SOH count for SOH > 0 when ARP > 0
3. ARP values for ARP = 0
4. ARP < SOH
5. Bar Graph generation for sale quantity of each style for comparison graph indicating previous year and current year values of sale quantity.

All these values were displayed corresponding to STYLE, COLOUR, BRAND, GROUP, CATEGORY, SIZE, DESCRIPTION, SUB CLASS, DEPARTMENT NAME and MRP.

During the development stage:

- Database was made (coretracker and range_planner).
- Tables were made which are known as models.
- Migrations were written which are basically structures of a table.
- Views contains the css and html codes.
- Controllers are where the special functions are written.

-Then the data was imported and the output displayed.

Range planner and core tracker both had 5 templates which were to be displayed along with the admin controls.

- Options for sign up, login page (using Devise gem and warden) were made.
- Data can be imported from .csv files in the format already specified above.
- Output is displayed on the browser which has a particular theme.
- Options to edit and recalculate data are available for range planner.
- A pivot was developed for the core tracker.
- Filters were put in for filtering brand and group –single or multiple in the range planner.
- Totals sums are displayed.
- All the template data can be exported to .csv file.

a. Implementation

1. Testing

The softwares were tested on localhost, port 3000, and default port for ruby rails.

A plan for Autumn Winter 2017 was developed using the software and the results were compared which turned out to be same, no errors were found.

2. Training

The concerned people who will use the software were trained on how to use it. (annexure 16)

3. Use

The software has been handed over to the company for use.

4. The software was then implemented on Amazon cloud.

Ip address - http://139.59.125.254/pivot

b. Output

Web applications were developed.

Annexure 11 to Annexure 15 and annexure 17 to annexure 21

CHAPTER 4

4. RESULTS

Two web applications (softwares) were developed based on the procedure of range planning and the needs of core tracking, using Ruby as back end with Rails framework, HTML and CSS as front end and SQLite database. Testing was done using data from past two seasons and a range plan was developed which was compared with an already existing range plan, the results were compared and were found to have no discrepancies. This was implemented and deployed on a cloud server.

a. For the range planner-

- Need of eliminating manual work done to develop a range plan was identified
- A software was developed to replace those activities
- 4 types of reports were generated (annexure 11 to annexure 15)
- Time was reduced to develop a range plan from 2 to 3 days to a few hours

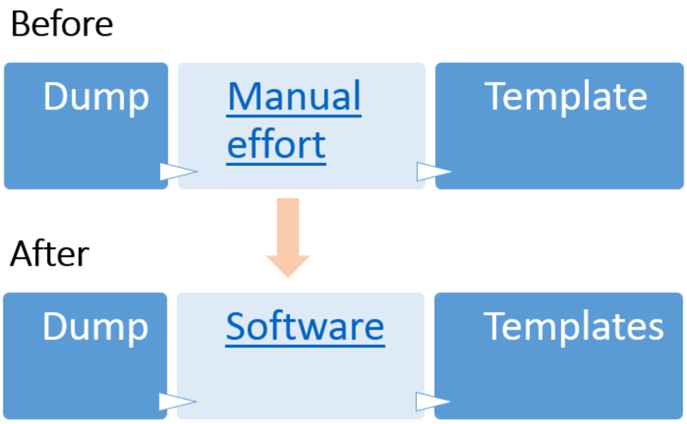


FIGURE 22RANGE PLANNER PROCESS BEFORE AND AFTER

- The software automated the following activities:
  - Collect last season sales data dump

- Calculate DMRP column
- Aggregate data according to brand, group and category
- Make pivot
- Paste it in a range planner template
- Replace values with assumptions
- Paste next season assumptions
- Formula driven fields get calculated

b. For the core tracker –

- No existing processes were in place so the available core sales data dump that was taken weekly was used to track core sales
- The availability was tracked overall
- The availability was tracked style wise
- Soh and arp were worked upon to develop 5 reports

ERDs (entity relationship diagram) generated (annexure 22)

CHAPTER 5

5. LIMITATIONS AND FUTURE SCOPE

- LIMITATIONS

1. These softwares cannot do anything else except for the purpose they are developed for.
2. The knowledge of ruby on rails is required to make any changes in the software.
3. Ruby is not an easy language and it has a steep learning curve.
4. Only the IT department will be able to make any changes if required, the users are laymen and therefore won't be able to make any changes in the software

- FUTURE SCOPE

1. The software is made for the kids department but will be extended to the women's wear department and the men's wear department.
2. The software can have many pivots, for core tracking.
3. Core tracker can have trend analysis graphs.
4. Imports can be done with import buttons instead of terminal.

CHAPTER 6

6. CONCLUSION

The two softwares developed helped in:

1. Eliminating manual work in case of the range planner
2. Saving time as the amount of time consumed was reduced from 2 – 3 days to 10 – 12 hours to develop range plans.
3. Eliminating human errors since all the calculations are done by the computer.
4. Eliminating redundancy as the same process is repeated everytime and automating it.
5. Centralising data on cloud so that it is available anywhere and anytime for the users.
6. Easier reading of data due to visual representation for the core tracker.
7. Tracking and forecasting.
8. Generating various reports for both.
9. Taking corrective actions quickly.
10. Tracking can now be done daily instead of on a weekly basis.
11. Knowing overall availability of core styles across all stores.

BIBLIOGRAPHY

[1]. http://www.justenough.com/

[2]. http://txtretail.txtgroup.com/solutions/assortment-planning-buying/

[3]. http://www.ispira.com/assortment-planning-software

[4]. http://www.darwinretail.com/

[5]. http://www.visualretailing.com/

[6]. http://www.visualretailing.com/how-we-help-retail

[7]. http://www.galleria-rts.com/index.php/solutions

[8]. http://www.retailsmart.com/store-planning-service

[9]. https://raid6.com.au/~onlyjob/posts/arena/

[10]. http://rubylearning.com/satishtalim/features.html

[11]. http://blog.stoneriverelearning.com/top-5-programming-languages-used-in-web-development/

[12]. http://guides.rubyonrails.org/getting_started.html

[13]. https://www.w3.org/standards/webdesign/htmlcss

[14]. https://www.sqlite.org/about.html

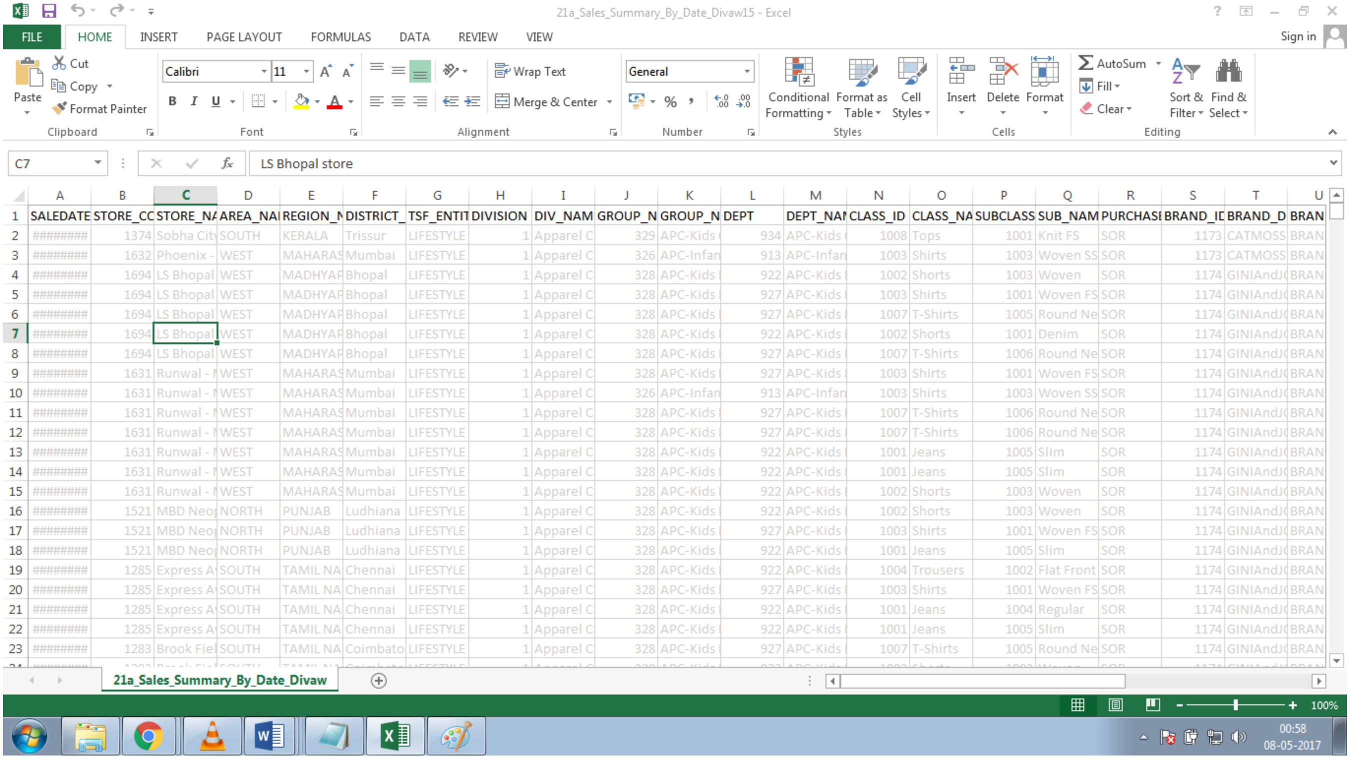
[15]. http://stackoverflow.com/questions/28555859/why-puma-rails-server-only-accepts-localhost3000-rather-than-127-0-0-13000

[16]. http://guides.rubygems.org/rubygems-basics/

[17]. http://www.justenough.com/merchandise-range-planning/

ANNEXURE

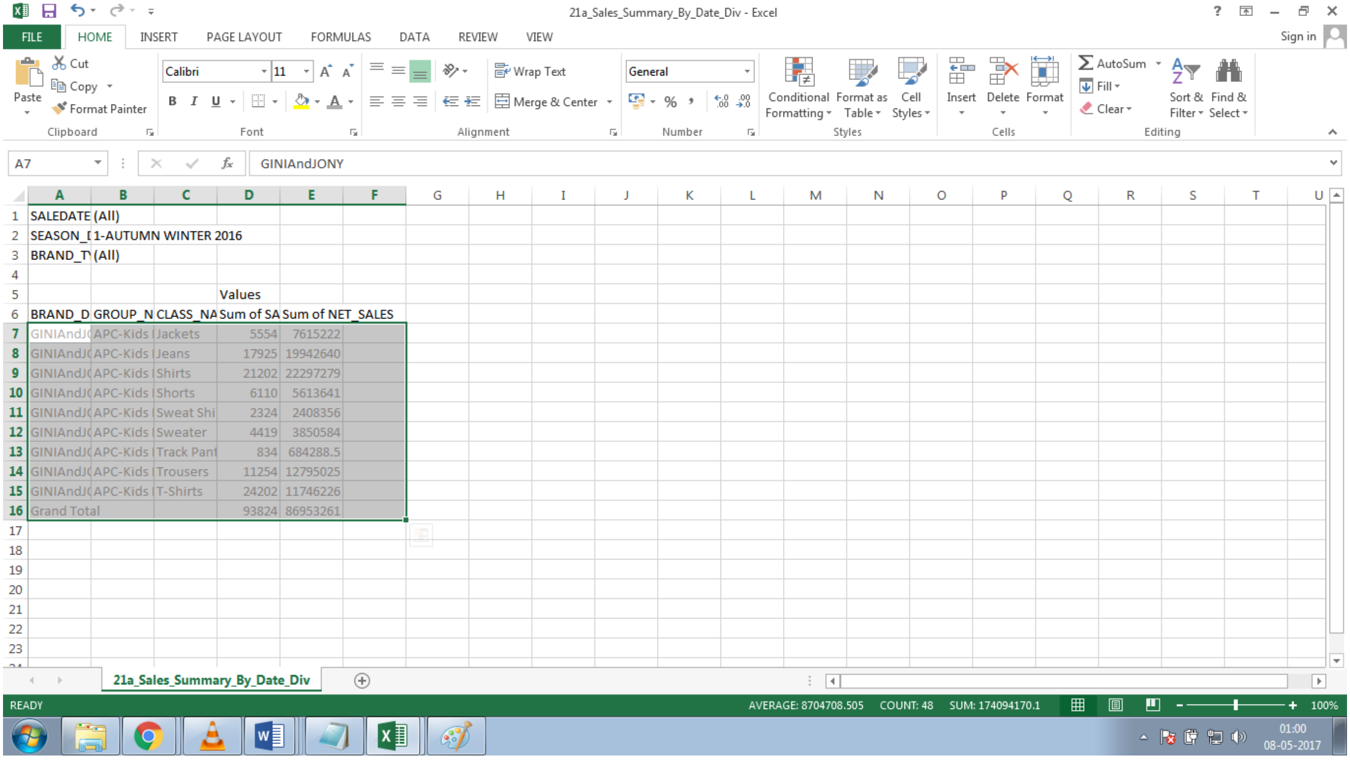
8.1 Annexure 1 – Previous season dump



C7 — LS Bhopal store

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	SALEDATE	STORE_CC	STORE_NA	AREA_NAI	REGION_N	DISTRICT_	TSF_ENTIT	DIVISION	DIV_NAM	GROUP_N	GROUP_N	DEPT	DEPT_NAI	CLASS_ID	CLASS_NA	SUBCLASS	SUB_NAM	PURCHASI	BRAND_ID	BRAND_D	BRAN
2	#########	1374	Sobha Cit	SOUTH	KERALA	Trissur	LIFESTYLE	1	Apparel C	329	APC-Kids (	934	APC-Kids (	1008	Tops	1001	Knit FS	SOR	1173	CATMOSS	BRAN
3	#########	1632	Phoenix -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	326	APC-Infan	913	APC-Infan	1003	Shirts	1003	Woven SS	SOR	1173	CATMOSS	BRAN
4	#########	1694	LS Bhopal	WEST	MADHYAP	Bhopal	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1002	Shorts	1003	Woven	SOR	1174	GINIAndJ	BRAN
5	#########	1694	LS Bhopal	WEST	MADHYAP	Bhopal	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1003	Shirts	1001	Woven FS	SOR	1174	GINIAndJ	BRAN
6	#########	1694	LS Bhopal	WEST	MADHYAP	Bhopal	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1007	T-Shirts	1005	Round Ne	SOR	1174	GINIAndJ	BRAN
7	#########	1694	LS Bhopal	WEST	MADHYAP	Bhopal	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1002	Shorts	1001	Denim	SOR	1174	GINIAndJ	BRAN
8	#########	1694	LS Bhopal	WEST	MADHYAP	Bhopal	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1007	T-Shirts	1006	Round Ne	SOR	1174	GINIAndJ	BRAN
9	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1003	Shirts	1001	Woven FS	SOR	1174	GINIAndJ	BRAN
10	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	326	APC-Infan	913	APC-Infan	1003	Shirts	1003	Woven SS	SOR	1174	GINIAndJ	BRAN
11	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1007	T-Shirts	1006	Round Ne	SOR	1174	GINIAndJ	BRAN
12	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1007	T-Shirts	1006	Round Ne	SOR	1174	GINIAndJ	BRAN
13	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1001	Jeans	1005	Slim	SOR	1174	GINIAndJ	BRAN
14	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1001	Jeans	1005	Slim	SOR	1174	GINIAndJ	BRAN
15	#########	1631	Runwal -	WEST	MAHARAS	Mumbai	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1002	Shorts	1003	Woven	SOR	1174	GINIAndJ	BRAN
16	#########	1521	MBD Neo	NORTH	PUNJAB	Ludhiana	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1002	Shorts	1003	Woven	SOR	1174	GINIAndJ	BRAN
17	#########	1521	MBD Neo	NORTH	PUNJAB	Ludhiana	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1003	Shirts	1001	Woven FS	SOR	1174	GINIAndJ	BRAN
18	#########	1521	MBD Neo	NORTH	PUNJAB	Ludhiana	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1001	Jeans	1005	Slim	SOR	1174	GINIAndJ	BRAN
19	#########	1285	Express A	SOUTH	TAMIL NA	Chennai	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1004	Trousers	1002	Flat Front	SOR	1174	GINIAndJ	BRAN
20	#########	1285	Express A	SOUTH	TAMIL NA	Chennai	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1003	Shirts	1001	Woven FS	SOR	1174	GINIAndJ	BRAN
21	#########	1285	Express A	SOUTH	TAMIL NA	Chennai	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1001	Jeans	1004	Regular	SOR	1174	GINIAndJ	BRAN
22	#########	1285	Express A	SOUTH	TAMIL NA	Chennai	LIFESTYLE	1	Apparel C	328	APC-Kids	922	APC-Kids	1001	Jeans	1005	Slim	SOR	1174	GINIAndJ	BRAN
23	#########	1283	Brook Fiel	SOUTH	TAMIL NA	Coimbato	LIFESTYLE	1	Apparel C	328	APC-Kids	927	APC-Kids	1007	T-Shirts	1005	Round Ne	SOR	1174	GINIAndJ	BRAN

8.2 Annexure 2 – Previous season dump

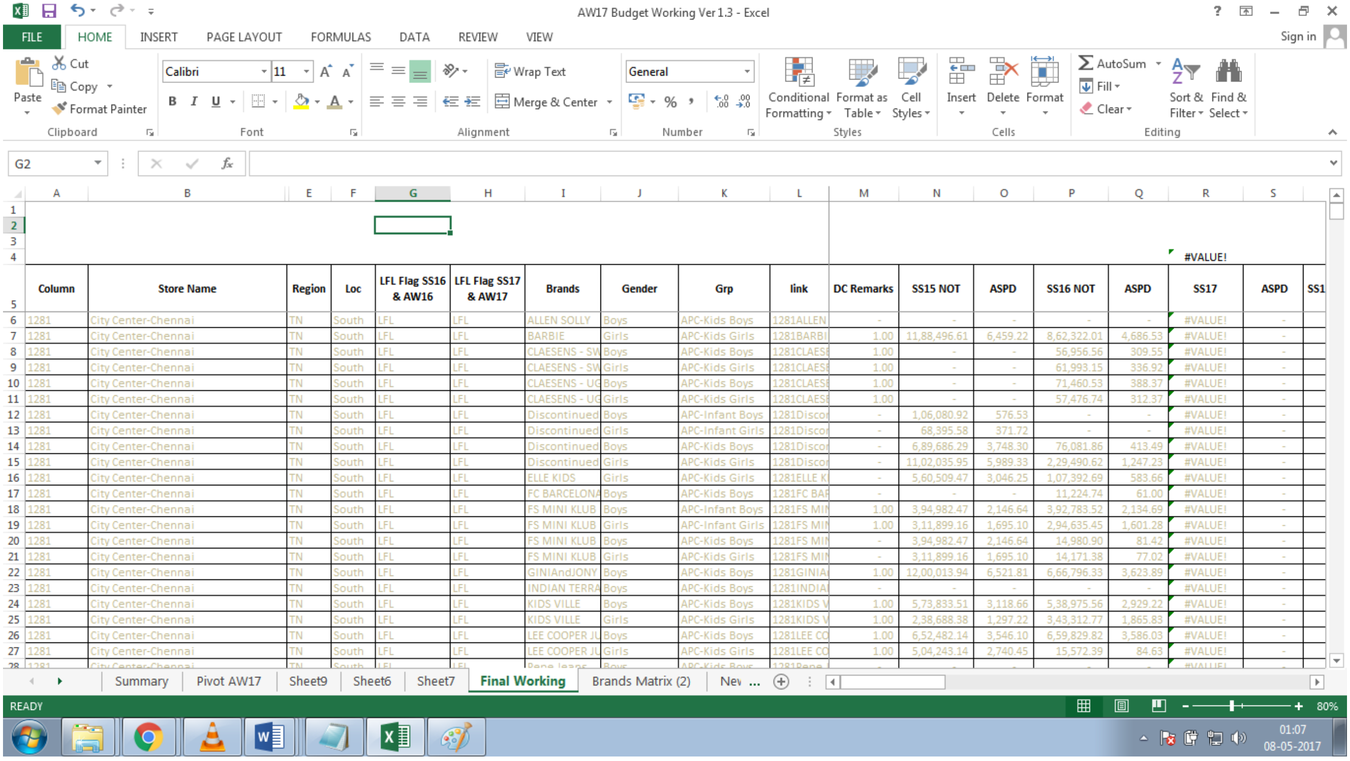


A7 — GINIAndJONY

	A	B	C	D	E	F
1	SALEDATE	(All)				
2	SEASON_I	1-AUTUMN WINTER 2016				
3	BRAND_T	(All)				
4						
5				Values		
6	BRAND_D	GROUP_N	CLASS_NA	Sum of SA	Sum of NET_SALES	
7	GINIAndJ	APC-Kids	Jackets	5554	7615222	
8	GINIAndJ	APC-Kids	Jeans	17925	19942640	
9	GINIAndJ	APC-Kids	Shirts	21202	22297279	
10	GINIAndJ	APC-Kids	Shorts	6110	5613641	
11	GINIAndJ	APC-Kids	Sweat Shi	2324	2408356	
12	GINIAndJ	APC-Kids	Sweater	4419	3850584	
13	GINIAndJ	APC-Kids	Track Pant	834	684288.5	
14	GINIAndJ	APC-Kids	Trousers	11254	12795025	
15	GINIAndJ	APC-Kids	T-Shirts	24202	11746226	
16	Grand Total			93824	86953261	

AVERAGE: 8704708.505 COUNT: 48 SUM: 174094170.1

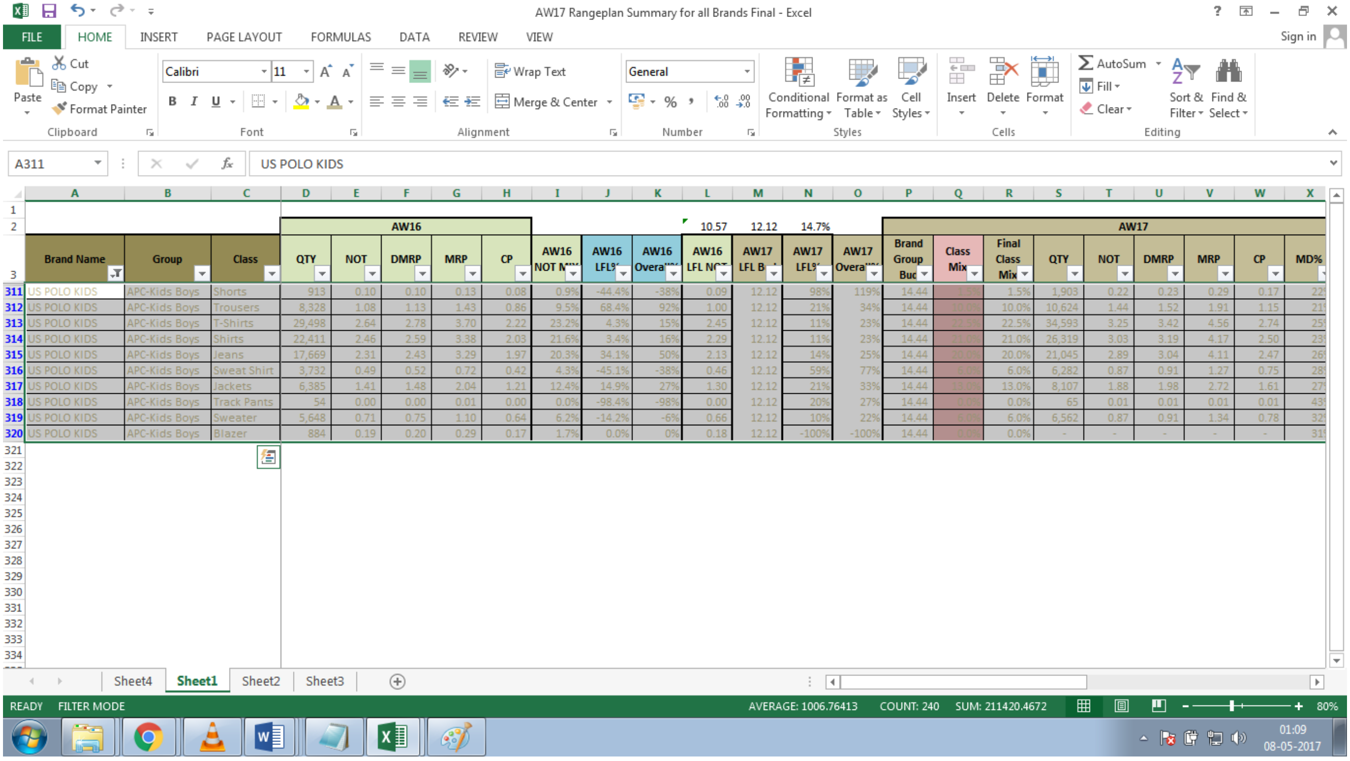
8.3 Annexure 3 – next season assumptions



	Column	Store Name	Region	Loc	LFL Flag SS16 & AW16	LFL Flag SS17 & AW17	Brands	Gender	Grp	link	DC Remarks	SS15 NOT	ASPD	SS16 NOT	ASPD	SS17	ASPD	SS1
4																#VALUE!		
6	1281	City Center-Chennai	TN	South	LFL	LFL	ALLEN SOLLY	Boys	APC-Kids Boys	1281ALLEN	-	-	-	-	-	#VALUE!	-	
7	1281	City Center-Chennai	TN	South	LFL	LFL	BARBIE	Girls	APC-Kids Girls	1281BARBI	1.00	11,88,496.61	6,459.22	8,62,322.01	4,686.53	#VALUE!	-	
8	1281	City Center-Chennai	TN	South	LFL	LFL	CLAESENS - SW	Boys	APC-Kids Boys	1281CLAESE	1.00	-	-	56,956.56	309.55	#VALUE!	-	
9	1281	City Center-Chennai	TN	South	LFL	LFL	CLAESENS - SW	Girls	APC-Kids Girls	1281CLAESE	1.00	-	-	61,993.15	336.92	#VALUE!	-	
10	1281	City Center-Chennai	TN	South	LFL	LFL	CLAESENS - UG	Boys	APC-Kids Boys	1281CLAESE	1.00	-	-	71,460.53	388.37	#VALUE!	-	
11	1281	City Center-Chennai	TN	South	LFL	LFL	CLAESENS - UG	Girls	APC-Kids Girls	1281CLAESE	1.00	-	-	57,476.74	312.37	#VALUE!	-	
12	1281	City Center-Chennai	TN	South	LFL	LFL	Discontinued	Boys	APC-Infant Boys	1281Discor	-	1,06,080.92	576.53	-	-	#VALUE!	-	
13	1281	City Center-Chennai	TN	South	LFL	LFL	Discontinued	Girls	APC-Infant Girls	1281Discor	-	68,395.58	371.72	-	-	#VALUE!	-	
14	1281	City Center-Chennai	TN	South	LFL	LFL	Discontinued	Boys	APC-Kids Boys	1281Discor	-	6,89,686.29	3,748.30	76,081.86	413.49	#VALUE!	-	
15	1281	City Center-Chennai	TN	South	LFL	LFL	Discontinued	Girls	APC-Kids Girls	1281Discor	-	11,02,035.95	5,989.33	2,29,490.62	1,247.23	#VALUE!	-	
16	1281	City Center-Chennai	TN	South	LFL	LFL	ELLE KIDS	Girls	APC-Kids Girls	1281ELLE KI	-	5,60,509.47	3,046.25	1,07,392.69	583.66	#VALUE!	-	
17	1281	City Center-Chennai	TN	South	LFL	LFL	FC BARCELONA	Boys	APC-Kids Boys	1281FC BAF	-	-	-	11,224.74	61.00	#VALUE!	-	
18	1281	City Center-Chennai	TN	South	LFL	LFL	FS MINI KLUB	Boys	APC-Infant Boys	1281FS MIN	1.00	3,94,982.47	2,146.64	3,92,783.52	2,134.69	#VALUE!	-	
19	1281	City Center-Chennai	TN	South	LFL	LFL	FS MINI KLUB	Girls	APC-Infant Girls	1281FS MIN	1.00	3,11,899.16	1,695.10	2,94,635.45	1,601.28	#VALUE!	-	
20	1281	City Center-Chennai	TN	South	LFL	LFL	FS MINI KLUB	Boys	APC-Kids Boys	1281FS MIN	-	3,94,982.47	2,146.64	14,980.90	81.42	#VALUE!	-	
21	1281	City Center-Chennai	TN	South	LFL	LFL	FS MINI KLUB	Girls	APC-Kids Girls	1281FS MIN	-	3,11,899.16	1,695.10	14,171.38	77.02	#VALUE!	-	
22	1281	City Center-Chennai	TN	South	LFL	LFL	GINIAndJONY	Boys	APC-Kids Boys	1281GINIA	1.00	12,00,013.94	6,521.81	6,66,796.33	3,623.89	#VALUE!	-	
23	1281	City Center-Chennai	TN	South	LFL	LFL	INDIAN TERRA	Boys	APC-Kids Boys	1281INDIAI	-	-	-	-	-	#VALUE!	-	
24	1281	City Center-Chennai	TN	South	LFL	LFL	KIDS VILLE	Boys	APC-Kids Boys	1281KIDS V	1.00	5,73,833.51	3,118.66	5,38,975.56	2,929.22	#VALUE!	-	
25	1281	City Center-Chennai	TN	South	LFL	LFL	KIDS VILLE	Girls	APC-Kids Girls	1281KIDS V	1.00	2,38,688.38	1,297.22	3,43,312.77	1,865.83	#VALUE!	-	
26	1281	City Center-Chennai	TN	South	LFL	LFL	LEE COOPER JU	Boys	APC-Kids Boys	1281LEE CO	1.00	6,52,482.14	3,546.10	6,59,829.82	3,586.03	#VALUE!	-	
27	1281	City Center-Chennai	TN	South	LFL	LFL	LEE COOPER JU	Girls	APC-Kids Girls	1281LEE CO	1.00	5,04,243.14	2,740.45	15,572.39	84.63	#VALUE!	-	
28	1281	City Center-Chennai	TN	South	LFL	LFL	Pepe Jeans	Boys	APC-Kids Boys	1281Pepe	-	-	-	-	-	#VALUE!	-	

Summary | Pivot AW17 | Sheet9 | Sheet6 | Sheet7 | **Final Working** | Brands Matrix (2) | Nev ...

8.4 Annexure 4 range plan



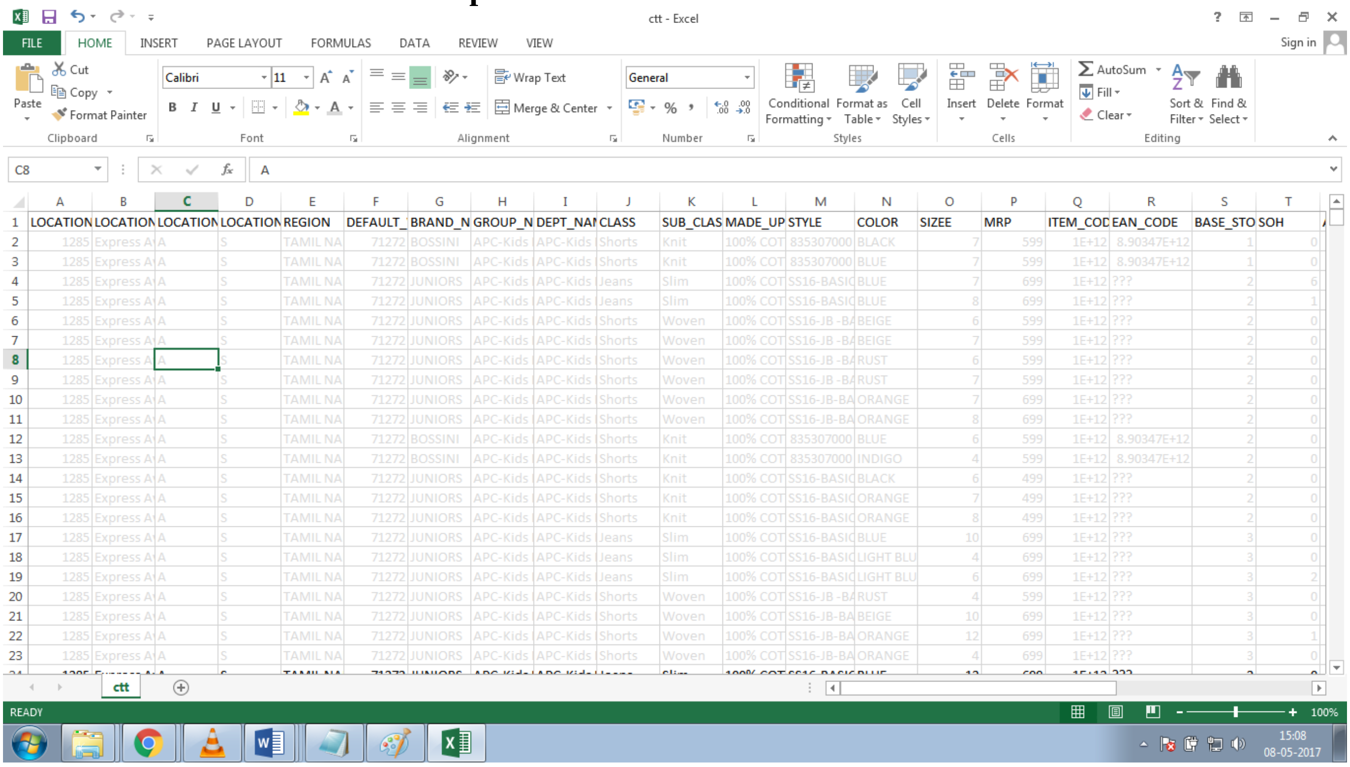
US POLO KIDS

				AW16								10.57	12.12	14.7%		AW17								
	Brand Name	Group	Class	QTY	NOT	DMRP	MRP	CP	AW16 NOT M	AW16 LFL%	AW16 Overa	AW16 LFL NOT	AW17 LFL B	AW17 LFL%	AW17 Overa	Brand Group Buc	Class Mix	Final Class Mix	QTY	NOT	DMRP	MRP	CP	MD%
311	US POLO KIDS	APC-Kids Boys	Shorts	913	0.10	0.10	0.13	0.08	0.9%	-44.4%	-38%	0.09	12.12	98%	119%	14.44	1.5%	1.5%	1,903	0.22	0.23	0.29	0.17	22
312	US POLO KIDS	APC-Kids Boys	Trousers	8,328	1.08	1.13	1.43	0.86	9.5%	68.4%	92%	1.00	12.12	21%	34%	14.44	10.0%	10.0%	10,624	1.44	1.52	1.91	1.15	21
313	US POLO KIDS	APC-Kids Boys	T-Shirts	29,498	2.64	2.78	3.70	2.22	23.2%	4.3%	15%	2.45	12.12	11%	23%	14.44	22.5%	22.5%	34,593	3.25	3.42	4.56	2.74	25
314	US POLO KIDS	APC-Kids Boys	Shirts	22,411	2.46	2.59	3.38	2.03	21.6%	3.4%	16%	2.29	12.12	11%	23%	14.44	21.0%	21.0%	26,319	3.03	3.19	4.17	2.50	23
315	US POLO KIDS	APC-Kids Boys	Jeans	17,669	2.31	2.43	3.29	1.97	20.3%	34.1%	50%	2.13	12.12	14%	25%	14.44	20.0%	20.0%	21,045	2.89	3.04	4.11	2.47	26
316	US POLO KIDS	APC-Kids Boys	Sweat Shirt	3,732	0.49	0.52	0.72	0.42	4.3%	-45.1%	-38%	0.46	12.12	59%	77%	14.44	6.0%	6.0%	6,282	0.87	0.91	1.27	0.75	28
317	US POLO KIDS	APC-Kids Boys	Jackets	6,385	1.41	1.48	2.04	1.21	12.4%	14.9%	27%	1.30	12.12	21%	33%	14.44	13.0%	13.0%	8,107	1.88	1.98	2.72	1.61	27
318	US POLO KIDS	APC-Kids Boys	Track Pants	54	0.00	0.00	0.01	0.00	0.0%	-98.4%	-98%	0.00	12.12	20%	27%	14.44	0.0%	0.0%	65	0.01	0.01	0.01	0.01	43
319	US POLO KIDS	APC-Kids Boys	Sweater	5,648	0.71	0.75	1.10	0.64	6.2%	-14.2%	-6%	0.66	12.12	10%	22%	14.44	6.0%	6.0%	6,562	0.87	0.91	1.34	0.78	32
320	US POLO KIDS	APC-Kids Boys	Blazer	884	0.19	0.20	0.29	0.17	1.7%	0.0%	0%	0.18	12.12	-100%	-100%	14.44	0.0%	0.0%	-	-	-	-	-	31

Sheet4 | **Sheet1** | Sheet2 | Sheet3

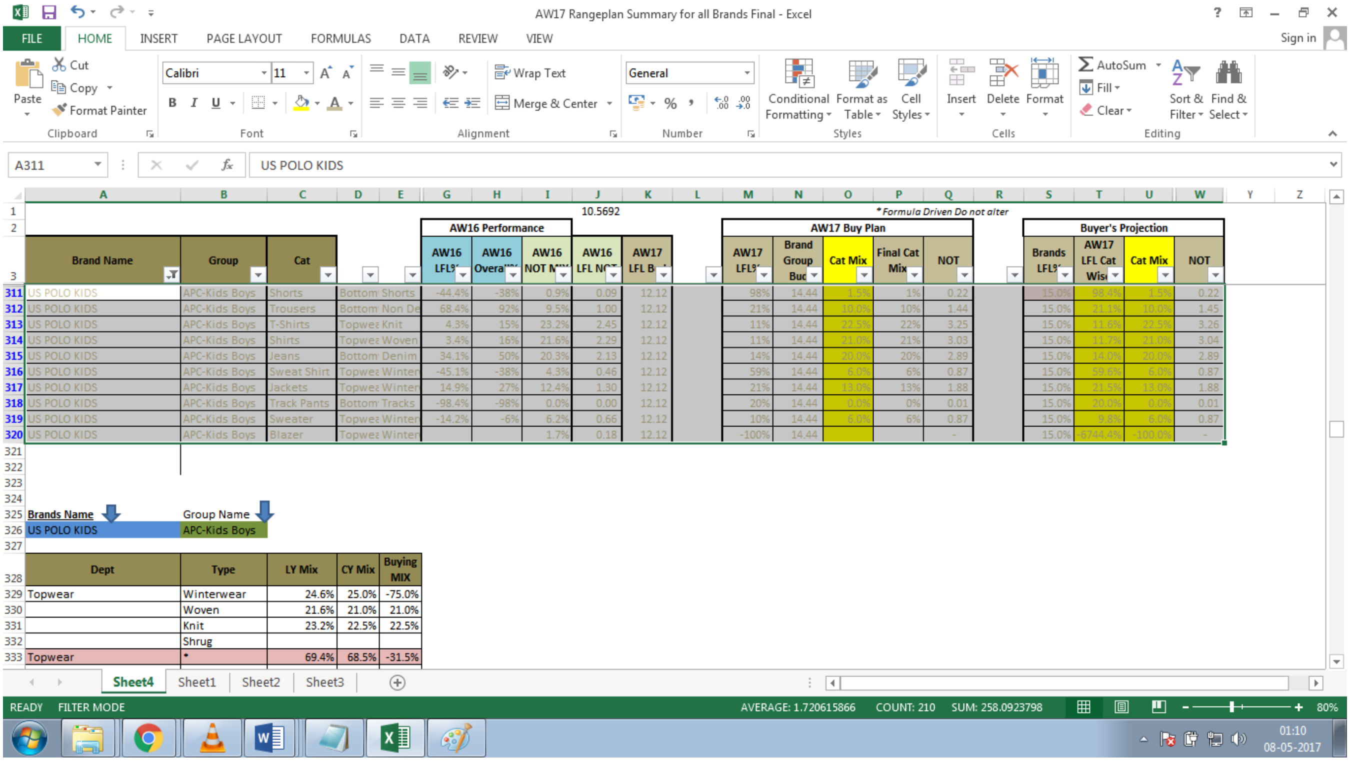
AVERAGE: 1006.76413 COUNT: 240 SUM: 211420.4672

8.5 Annexure 5 – core tracker template



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	LOCATION	LOCATION	LOCATION	LOCATION	REGION	DEFAULT_	BRAND_N	GROUP_N	DEPT_NAI	CLASS	SUB_CLAS	MADE_UP	STYLE	COLOR	SIZEE	MRP	ITEM_COD	EAN_CODE	BASE_STO	SOH
2	1285	Express A	A	S	TAMIL NA	71272	BOSSINI	APC-Kids	APC-Kids	Shorts	Knit	100% COT	835307000	BLACK	7	599	1E+12	8.90347E+12	1	0
3	1285	Express A	A	S	TAMIL NA	71272	BOSSINI	APC-Kids	APC-Kids	Shorts	Knit	100% COT	835307000	BLUE	7	599	1E+12	8.90347E+12	1	0
4	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Jeans	Slim	100% COT	SS16-BASIC	BLUE	7	699	1E+12	???	2	6
5	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Jeans	Slim	100% COT	SS16-BASIC	BLUE	8	699	1E+12	???	2	1
6	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB -BA	BEIGE	6	599	1E+12	???	2	0
7	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB -BA	BEIGE	7	599	1E+12	???	2	0
8	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB -BA	RUST	6	599	1E+12	???	2	0
9	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB -BA	RUST	7	599	1E+12	???	2	0
10	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB-BA	ORANGE	7	699	1E+12	???	2	0
11	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB-BA	ORANGE	8	699	1E+12	???	2	0
12	1285	Express A	A	S	TAMIL NA	71272	BOSSINI	APC-Kids	APC-Kids	Shorts	Knit	100% COT	835307000	BLUE	6	599	1E+12	8.90347E+12	2	0
13	1285	Express A	A	S	TAMIL NA	71272	BOSSINI	APC-Kids	APC-Kids	Shorts	Knit	100% COT	835307000	INDIGO	4	599	1E+12	8.90347E+12	2	0
14	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Knit	100% COT	SS16-BASIC	BLACK	6	499	1E+12	???	2	0
15	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Knit	100% COT	SS16-BASIC	ORANGE	7	499	1E+12	???	2	0
16	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Knit	100% COT	SS16-BASIC	ORANGE	8	499	1E+12	???	2	0
17	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Jeans	Slim	100% COT	SS16-BASIC	BLUE	10	699	1E+12	???	3	0
18	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Jeans	Slim	100% COT	SS16-BASIC	LIGHT BLU	4	699	1E+12	???	3	0
19	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Jeans	Slim	100% COT	SS16-BASIC	LIGHT BLU	6	699	1E+12	???	3	2
20	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB -BA	RUST	4	599	1E+12	???	3	0
21	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB-BA	BEIGE	10	699	1E+12	???	3	0
22	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB-BA	ORANGE	12	699	1E+12	???	3	1
23	1285	Express A	A	S	TAMIL NA	71272	JUNIORS	APC-Kids	APC-Kids	Shorts	Woven	100% COT	SS16-JB-BA	ORANGE	4	699	1E+12	???	3	0

8.6 Annexure 6 – range plan- business plan


AW17 Rangeplan Summary for all Brands Final - Excel
FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW
A311
US POLO KIDS
Sheet4 Sheet1 Sheet2 Sheet3
READY FILTER MODE
AVERAGE: 1.720615866 COUNT: 210 SUM: 258.0923798
80%
01:10
08-05-2017

10.5692

* Formula Driven Do not alter

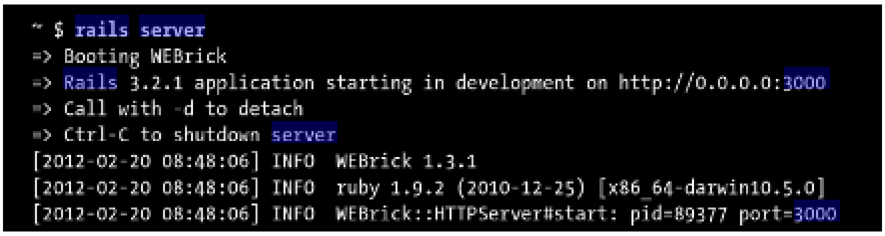
	A	B	C	D	E	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	W
2						AW16 Performance						AW17 Buy Plan						Buyer's Projection			
3	Brand Name	Group	Cat			AW16 LFL%	AW16 Overa	AW16 NOT M	AW16 LFL NO	AW17 LFL B		AW17 LFL%	Brand Group Buc	Cat Mix	Final Cat Mix	NOT		Brands LFL%	AW17 LFL Cat Wis	Cat Mix	NOT
311	US POLO KIDS	APC-Kids Boys	Shorts	Bottom	Shorts	-44.4%	-38%	0.9%	0.09	12.12		98%	14.44	1.5%	1%	0.22		15.0%	98.4%	1.5%	0.22
312	US POLO KIDS	APC-Kids Boys	Trousers	Bottom	Non De	68.4%	92%	9.5%	1.00	12.12		21%	14.44	10.0%	10%	1.44		15.0%	21.1%	10.0%	1.45
313	US POLO KIDS	APC-Kids Boys	T-Shirts	Topwea	Knit	4.3%	15%	23.2%	2.45	12.12		11%	14.44	22.5%	22%	3.25		15.0%	11.6%	22.5%	3.26
314	US POLO KIDS	APC-Kids Boys	Shirts	Topwea	Woven	3.4%	16%	21.6%	2.29	12.12		11%	14.44	21.0%	21%	3.03		15.0%	11.7%	21.0%	3.04
315	US POLO KIDS	APC-Kids Boys	Jeans	Bottom	Denim	34.1%	50%	20.3%	2.13	12.12		14%	14.44	20.0%	20%	2.89		15.0%	14.0%	20.0%	2.89
316	US POLO KIDS	APC-Kids Boys	Sweat Shirt	Topwea	Winter	-45.1%	-38%	4.3%	0.46	12.12		59%	14.44	6.0%	6%	0.87		15.0%	59.6%	6.0%	0.87
317	US POLO KIDS	APC-Kids Boys	Jackets	Topwea	Winter	14.9%	27%	12.4%	1.30	12.12		21%	14.44	13.0%	13%	1.88		15.0%	21.5%	13.0%	1.88
318	US POLO KIDS	APC-Kids Boys	Track Pants	Bottom	Tracks	-98.4%	-98%	0.0%	0.00	12.12		20%	14.44	0.0%	0%	0.01		15.0%	20.0%	0.0%	0.01
319	US POLO KIDS	APC-Kids Boys	Sweater	Topwea	Winter	-14.2%	-6%	6.2%	0.66	12.12		10%	14.44	6.0%	6%	0.87		15.0%	9.8%	6.0%	0.87
320	US POLO KIDS	APC-Kids Boys	Blazer	Topwea	Winter			1.7%	0.18	12.12		-100%	14.44			-		15.0%	-6744.4%	-100.0%	-

	A	B
325	Brands Name	Group Name
326	US POLO KIDS	APC-Kids Boys

	Dept	Type	LY Mix	CY Mix	Buying MIX
329	Topwear	Winterwear	24.6%	25.0%	-75.0%
330		Woven	21.6%	21.0%	21.0%
331		Knit	23.2%	22.5%	22.5%
332		Shrug			
333	Topwear	*	69.4%	68.5%	-31.5%

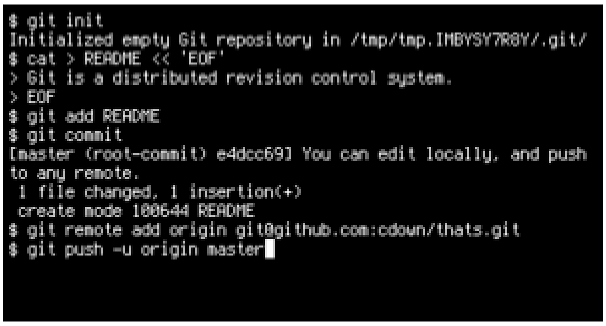
8.7 Annexure 7 – linux Ubuntu os

8.8 Annexure 8 – WEBrick server



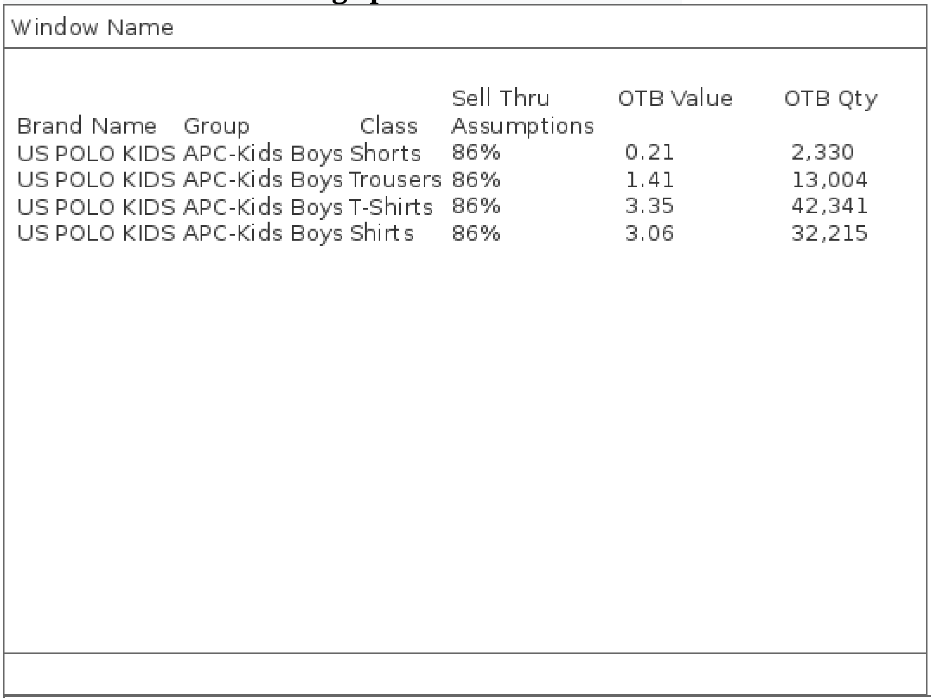
```
 ~ $ rails server
 => Booting WEBrick
 => Rails 3.2.1 application starting in development on http://0.0.0.0:3000
 => Call with -d to detach
 => Ctrl-C to shutdown server
 [2012-02-20 08:48:06] INFO  WEBrick 1.3.1
 [2012-02-20 08:48:06] INFO  ruby 1.9.2 (2010-12-25) [x86_64-darwin10.5.0]
 [2012-02-20 08:48:06] INFO  WEBrick::HTTPServer#start: pid=89377 port=3000
```

8.9 Annexure 9- A command-line session showing repository creation, addition of a file, and remote synchronization

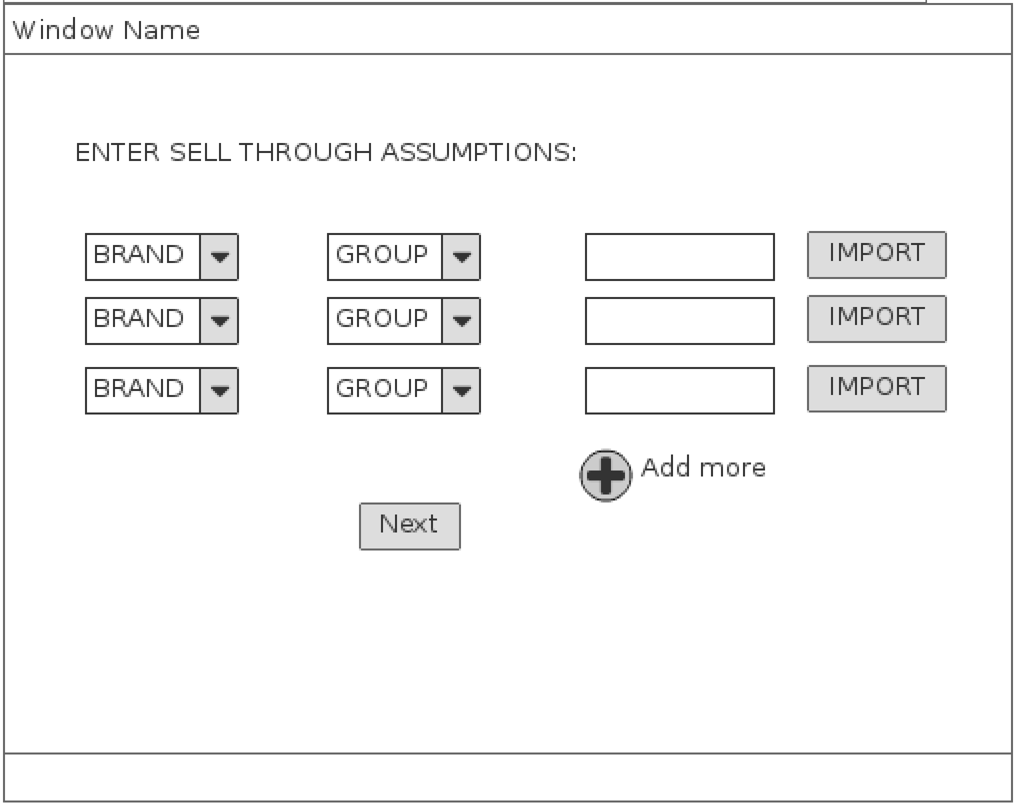


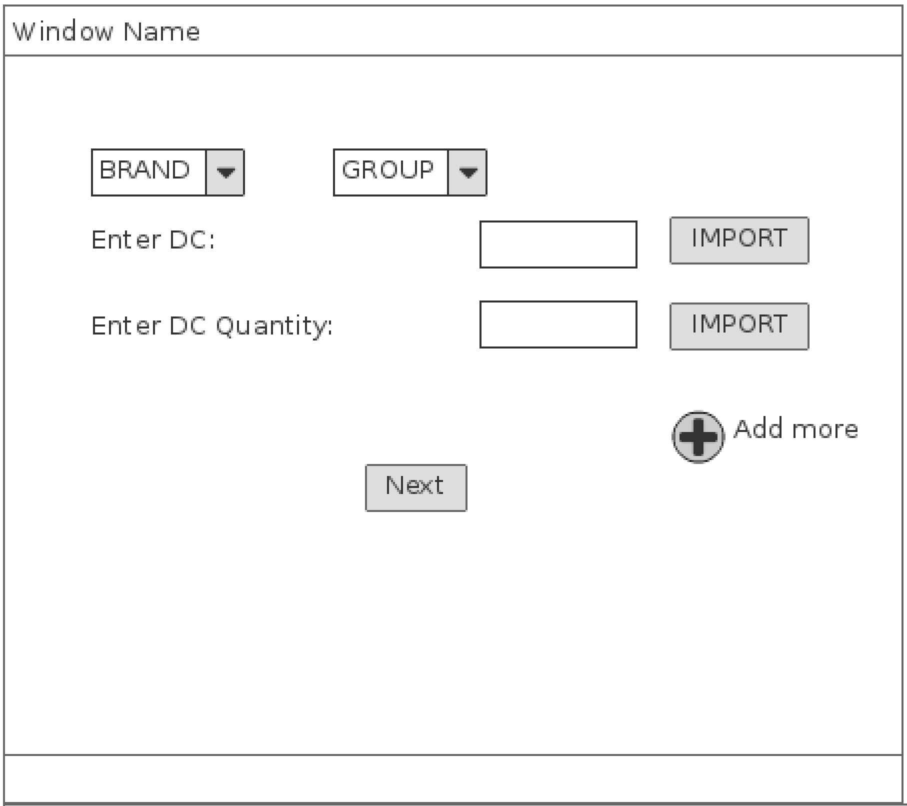
```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally, and push
to any remote.
 1 file changed, 1 insertion(+)
 create mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

8.10 Annexure 10 – Range plan initial wireframes



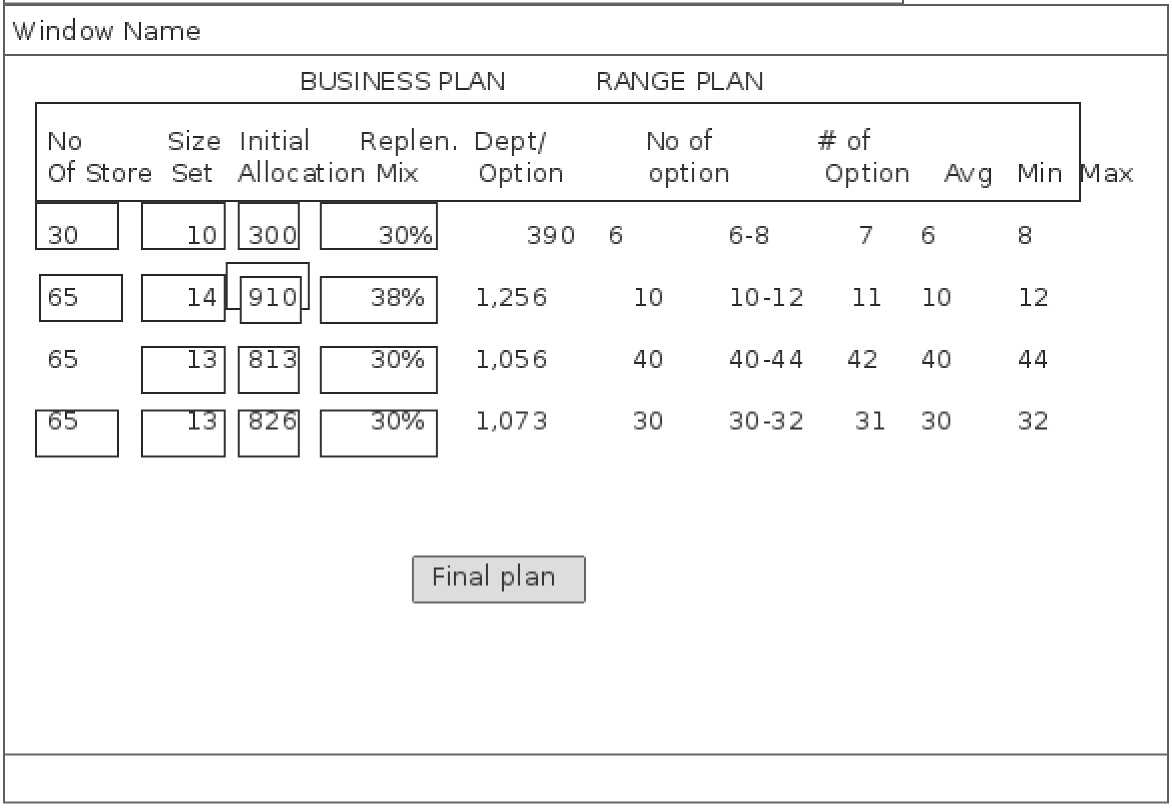
Brand Name	Group	Class	Sell Thru Assumptions	OTB Value	OTB Qty
US POLO KIDS	APC-Kids Boys	Shorts	86%	0.21	2,330
US POLO KIDS	APC-Kids Boys	Trousers	86%	1.41	13,004
US POLO KIDS	APC-Kids Boys	T-Shirts	86%	3.35	42,341
US POLO KIDS	APC-Kids Boys	Shirts	86%	3.06	32,215





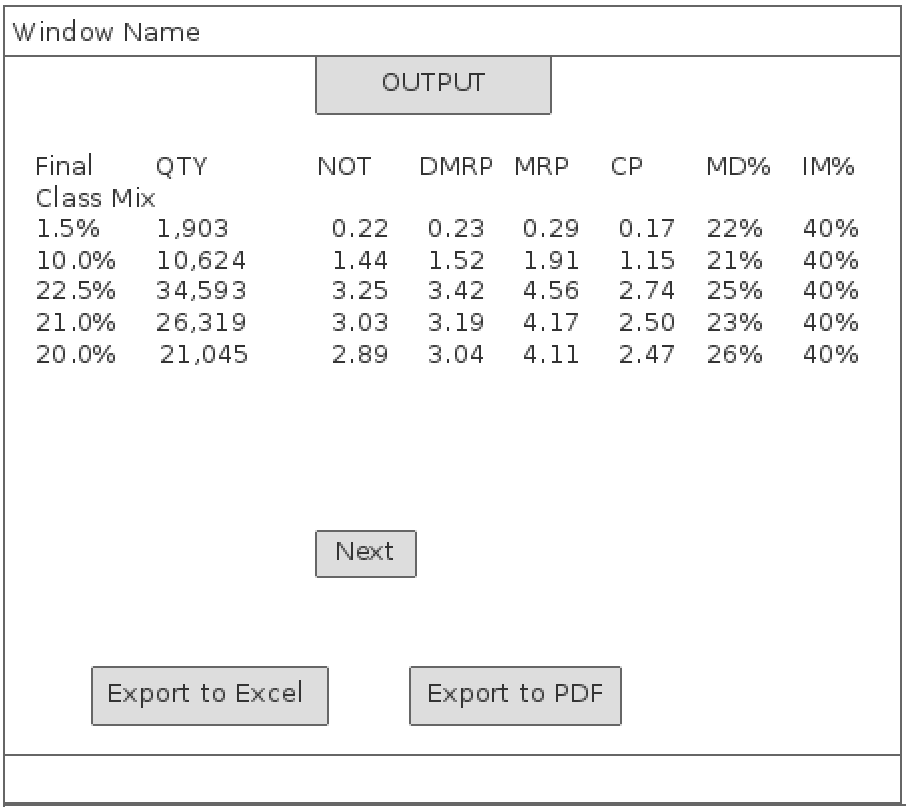
Enter DC:

Enter DC Quantity:

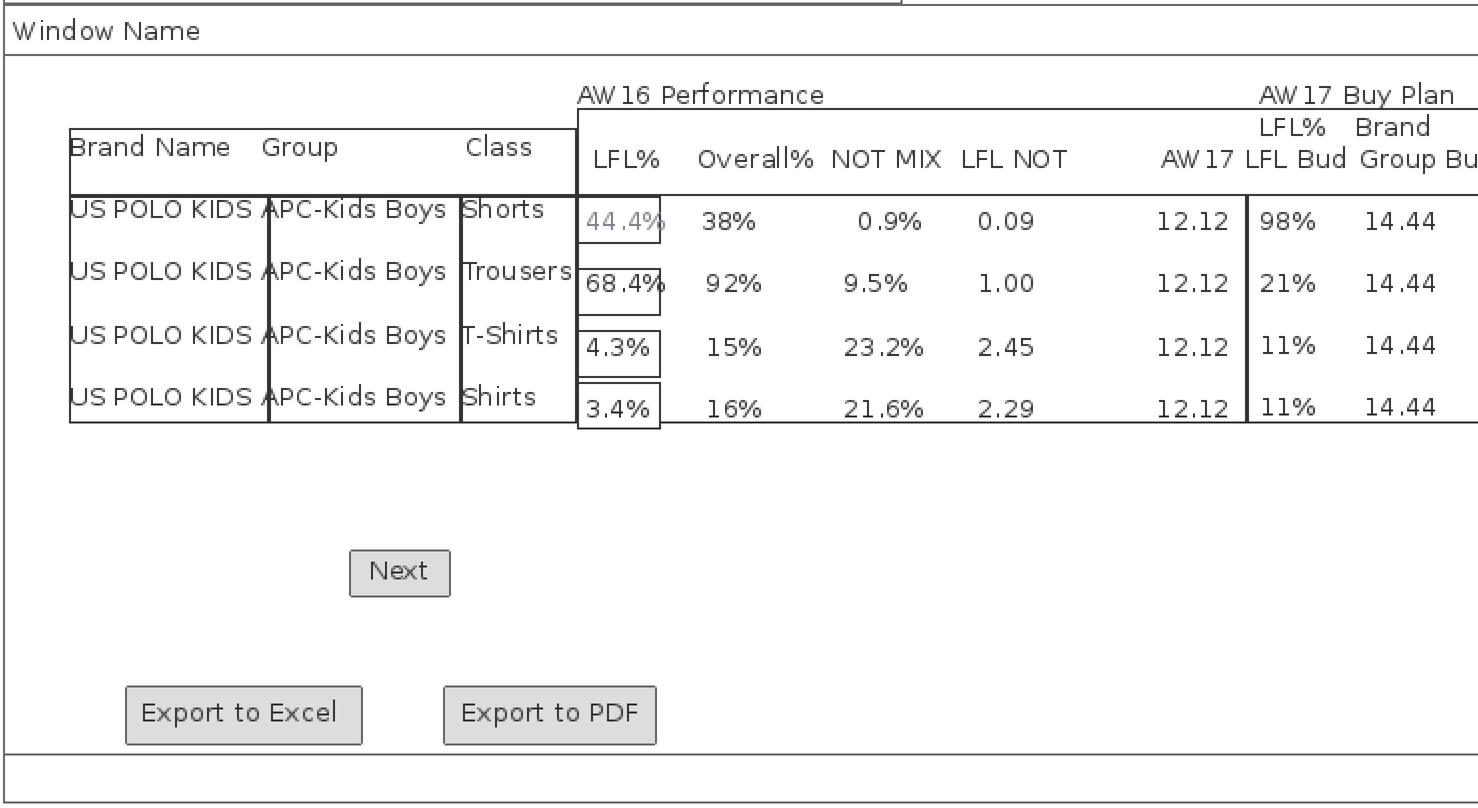


BUSINESS PLAN RANGE PLAN

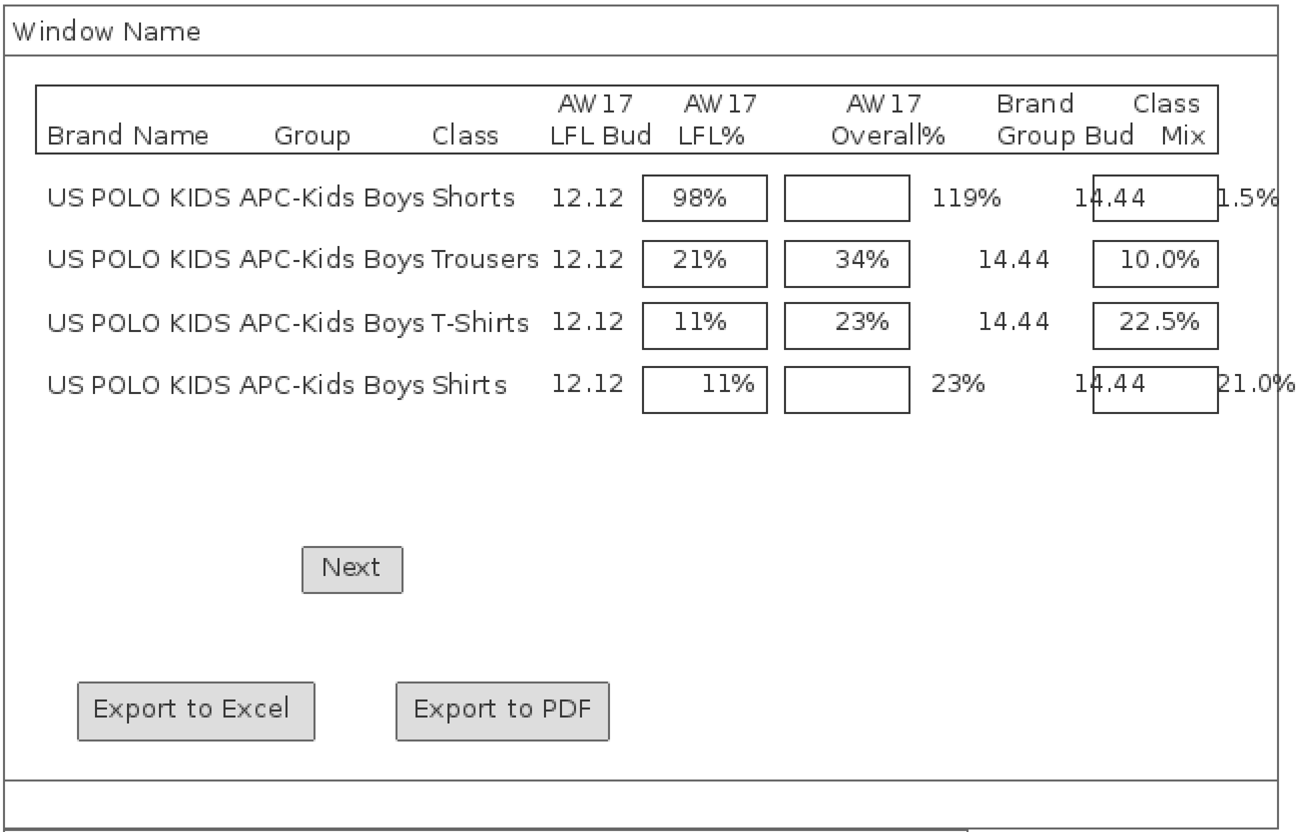
No Of Store	Size Set	Initial Allocation	Replen. Mix	Dept/ Option	No of option	# of Option	Avg	Min	Max
30	10	300	30%	390	6	6-8	7	6	8
65	14	910	38%	1,256	10	10-12	11	10	12
65	13	813	30%	1,056	40	40-44	42	40	44
65	13	826	30%	1,073	30	30-32	31	30	32



Final Class Mix	QTY	NOT	DMRP	MRP	CP	MD%	IM%
1.5%	1,903	0.22	0.23	0.29	0.17	22%	40%
10.0%	10,624	1.44	1.52	1.91	1.15	21%	40%
22.5%	34,593	3.25	3.42	4.56	2.74	25%	40%
21.0%	26,319	3.03	3.19	4.17	2.50	23%	40%
20.0%	21,045	2.89	3.04	4.11	2.47	26%	40%

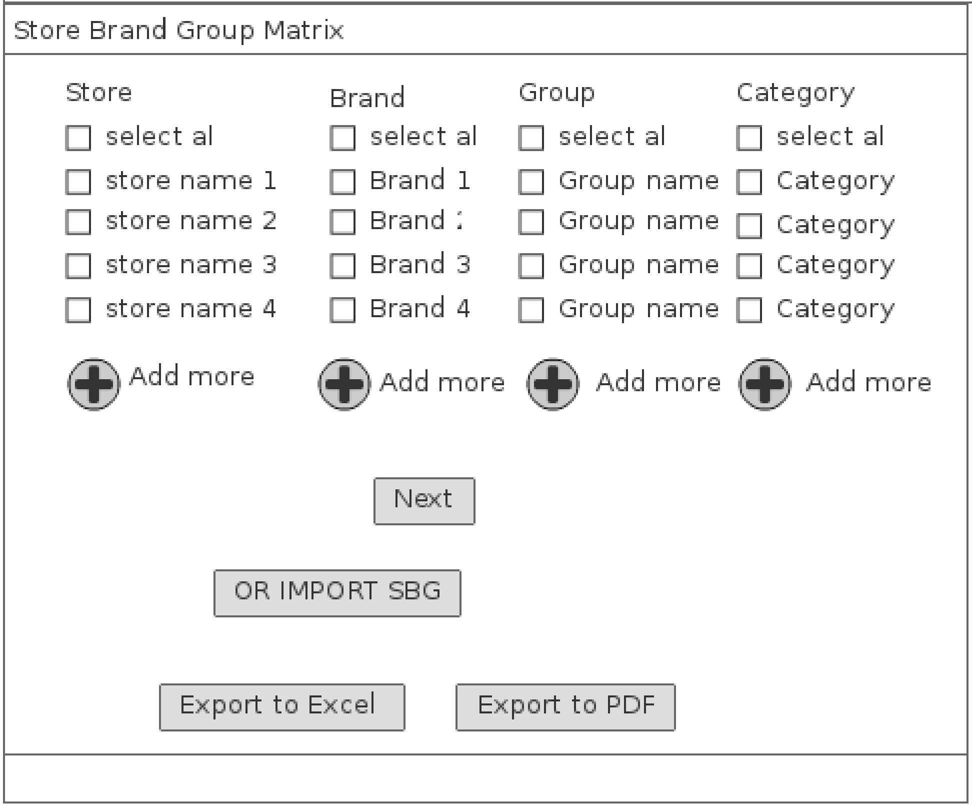


			AW16 Performance					AW17 Buy Plan	
								LFL%	Brand
Brand Name	Group	Class	LFL%	Overall%	NOT MIX	LFL NOT	AW17	LFL Bud	Group Bu
US POLO KIDS	APC-Kids Boys	Shorts	44.4%	38%	0.9%	0.09	12.12	98%	14.44
US POLO KIDS	APC-Kids Boys	Trousers	68.4%	92%	9.5%	1.00	12.12	21%	14.44
US POLO KIDS	APC-Kids Boys	T-Shirts	4.3%	15%	23.2%	2.45	12.12	11%	14.44
US POLO KIDS	APC-Kids Boys	Shirts	3.4%	16%	21.6%	2.29	12.12	11%	14.44



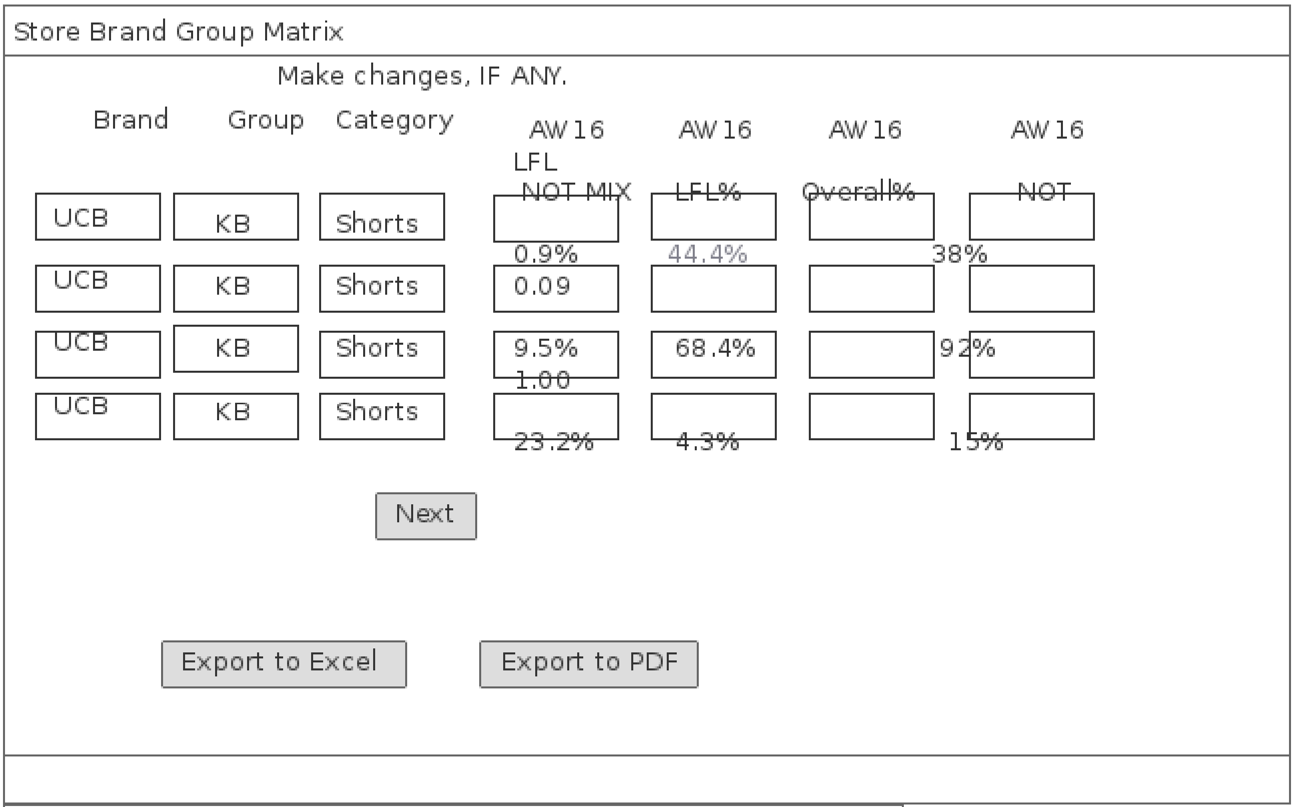
Window Name

Brand Name	Group	Class	AW 17 LFL Bud	AW 17 LFL%	AW 17 Overall%	Brand Group Bud	Class Mix
US POLO KIDS	APC-Kids Boys	Shorts	12.12	98%	119%	14.44	1.5%
US POLO KIDS	APC-Kids Boys	Trousers	12.12	21%	34%	14.44	10.0%
US POLO KIDS	APC-Kids Boys	T-Shirts	12.12	11%	23%	14.44	22.5%
US POLO KIDS	APC-Kids Boys	Shirts	12.12	11%	23%	14.44	21.0%



Store Brand Group Matrix

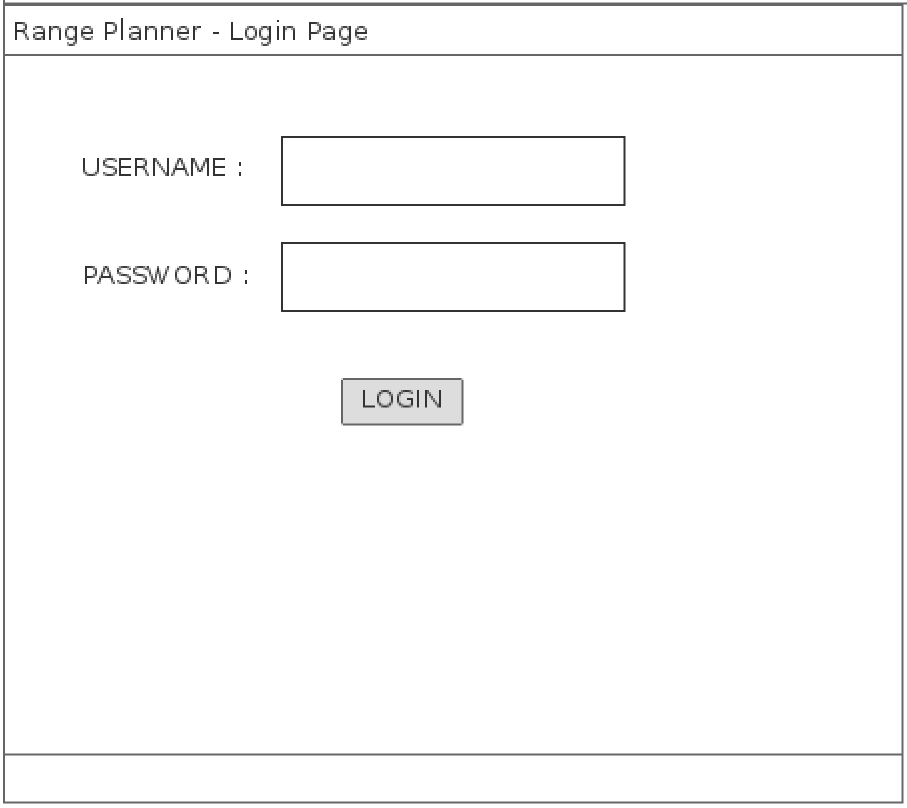
Store	Brand	Group	Category
□ select al	□ select al	□ select al	□ select al
□ store name 1	□ Brand 1	□ Group name	□ Category
□ store name 2	□ Brand :	□ Group name	□ Category
□ store name 3	□ Brand 3	□ Group name	□ Category
□ store name 4	□ Brand 4	□ Group name	□ Category
⊕ Add more	⊕ Add more	⊕ Add more	⊕ Add more



Store Brand Group Matrix

Make changes, IF ANY.

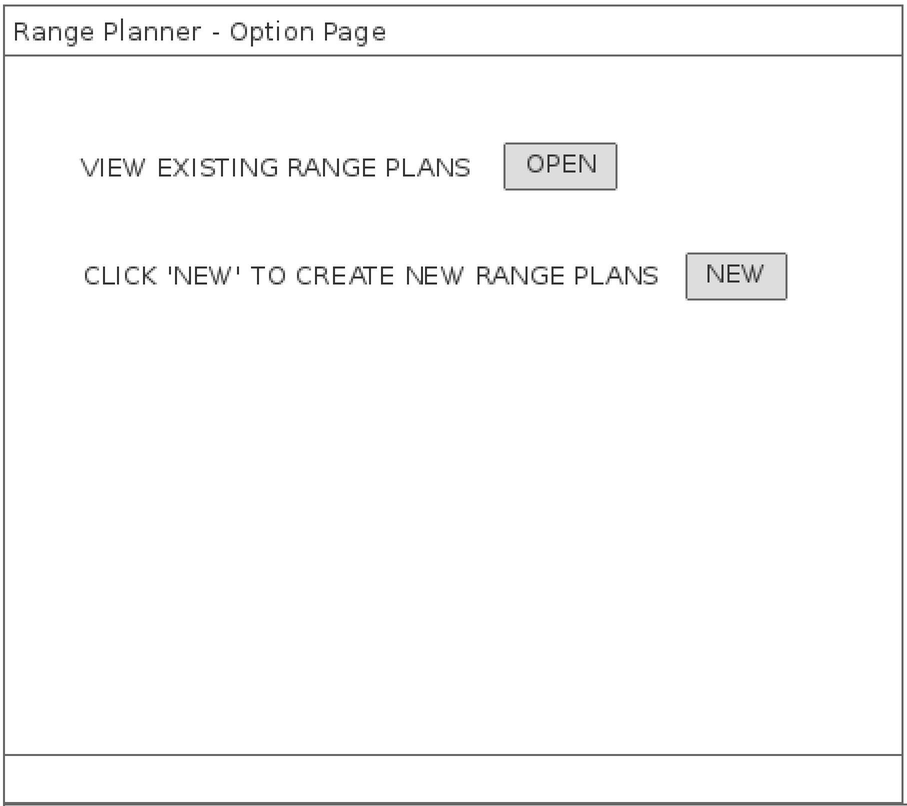
Brand	Group	Category	AW 16 LFL NOT MIX	AW 16 LFL%	AW 16 Overall%	AW 16 NOT
UCB	KB	Shorts	0.9%	44.4%	38%	
UCB	KB	Shorts	0.09			
UCB	KB	Shorts	9.5%	68.4%	92%	
UCB	KB	Shorts	1.00 23.2%	4.3%	15%	



Range Planner - Login Page

USERNAME :

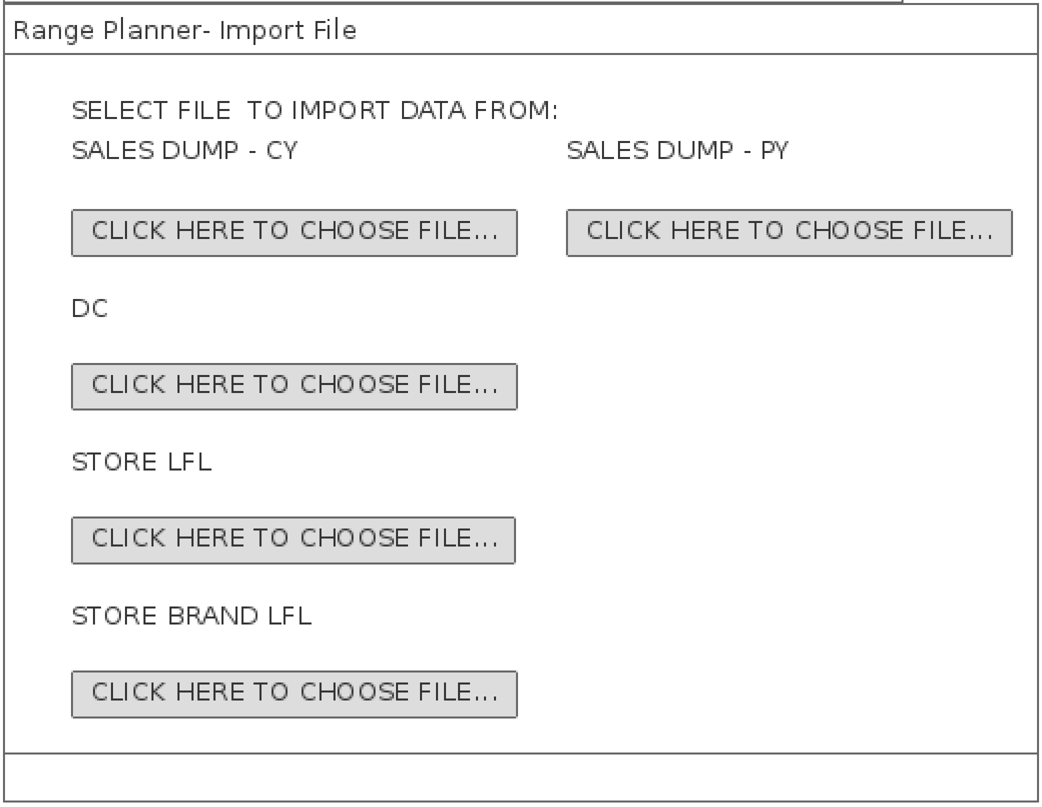
PASSWORD :



Range Planner - Option Page

VIEW EXISTING RANGE PLANS

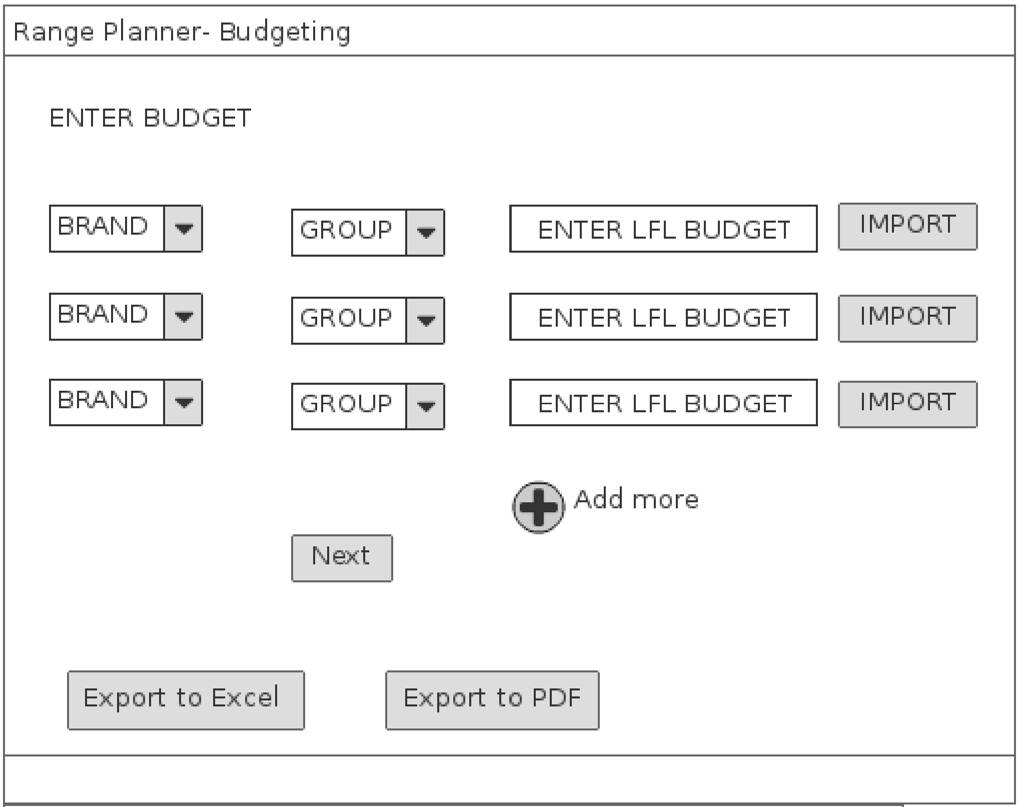
CLICK 'NEW' TO CREATE NEW RANGE PLANS

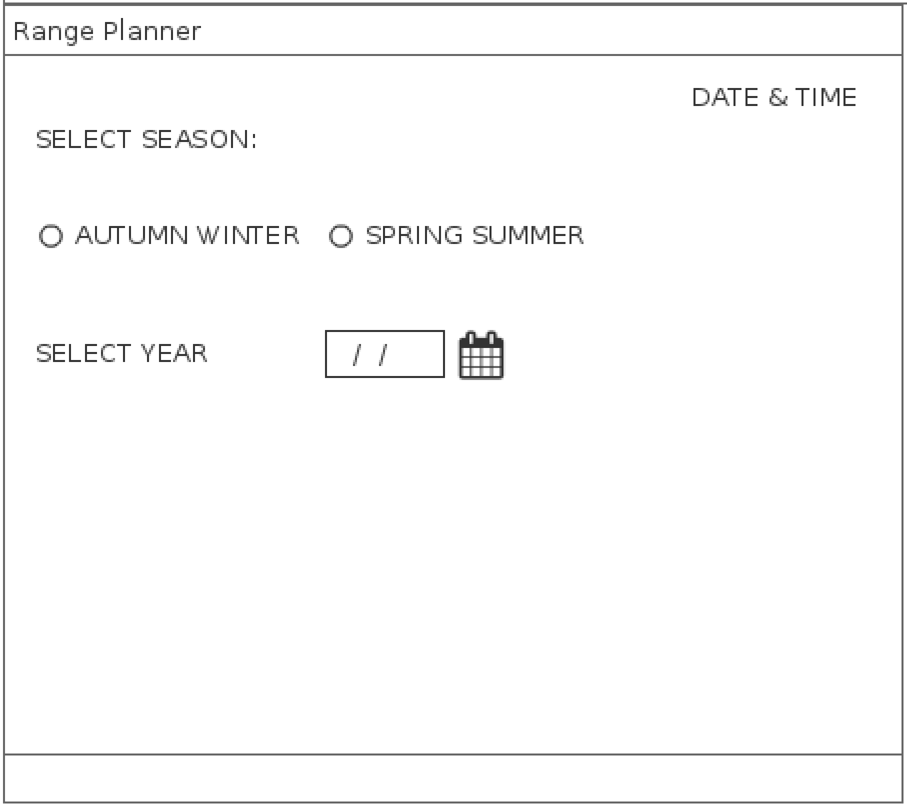


Range Planner- Import File

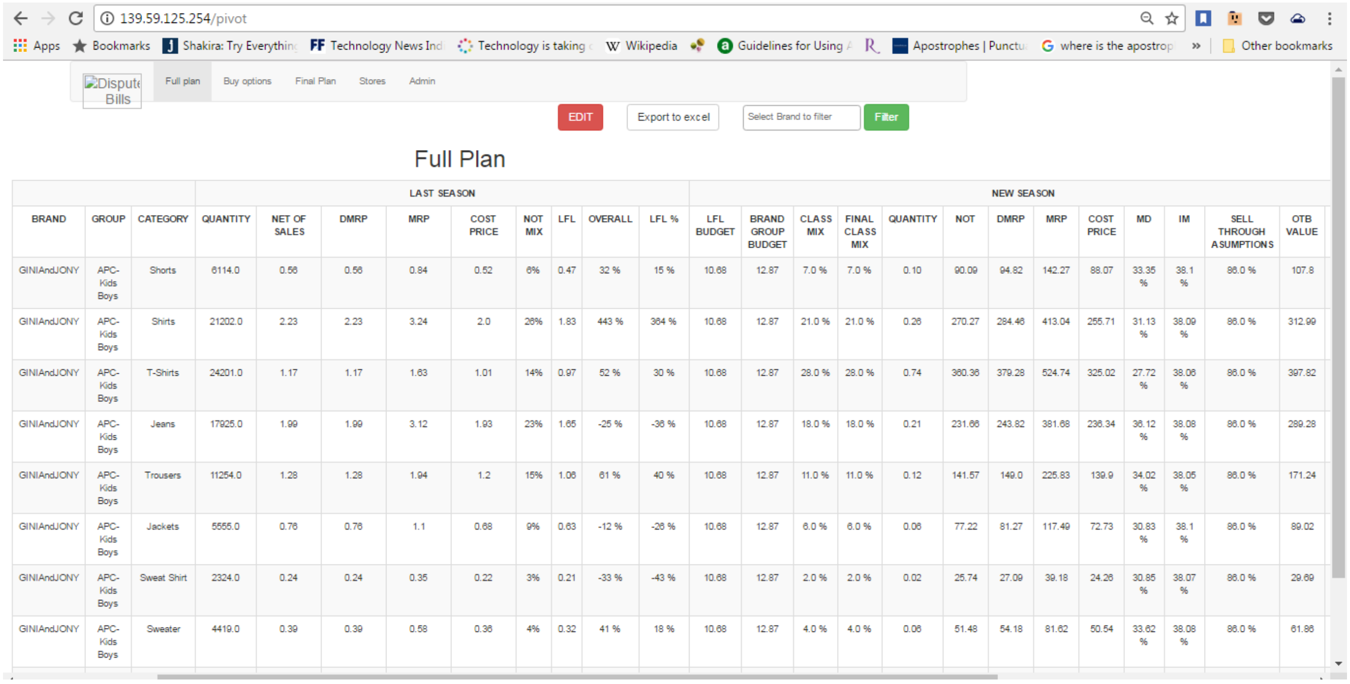
SELECT FILE TO IMPORT DATA FROM:

SALES DUMP - CY
SALES DUMP - PY
CLICK HERE TO CHOOSE FILE...
CLICK HERE TO CHOOSE FILE...
DC
CLICK HERE TO CHOOSE FILE...
STORE LFL
CLICK HERE TO CHOOSE FILE...
STORE BRAND LFL
CLICK HERE TO CHOOSE FILE...


Range Planner- Budgeting
ENTER BUDGET
BRAND
GROUP
ENTER LFL BUDGET
IMPORT
BRAND
GROUP
ENTER LFL BUDGET
IMPORT
BRAND
GROUP
ENTER LFL BUDGET
IMPORT
Add more
Next
Export to Excel
Export to PDF


Range Planner
DATE & TIME
SELECT SEASON:
○ AUTUMN WINTER
○ SPRING SUMMER
SELECT YEAR
/ /

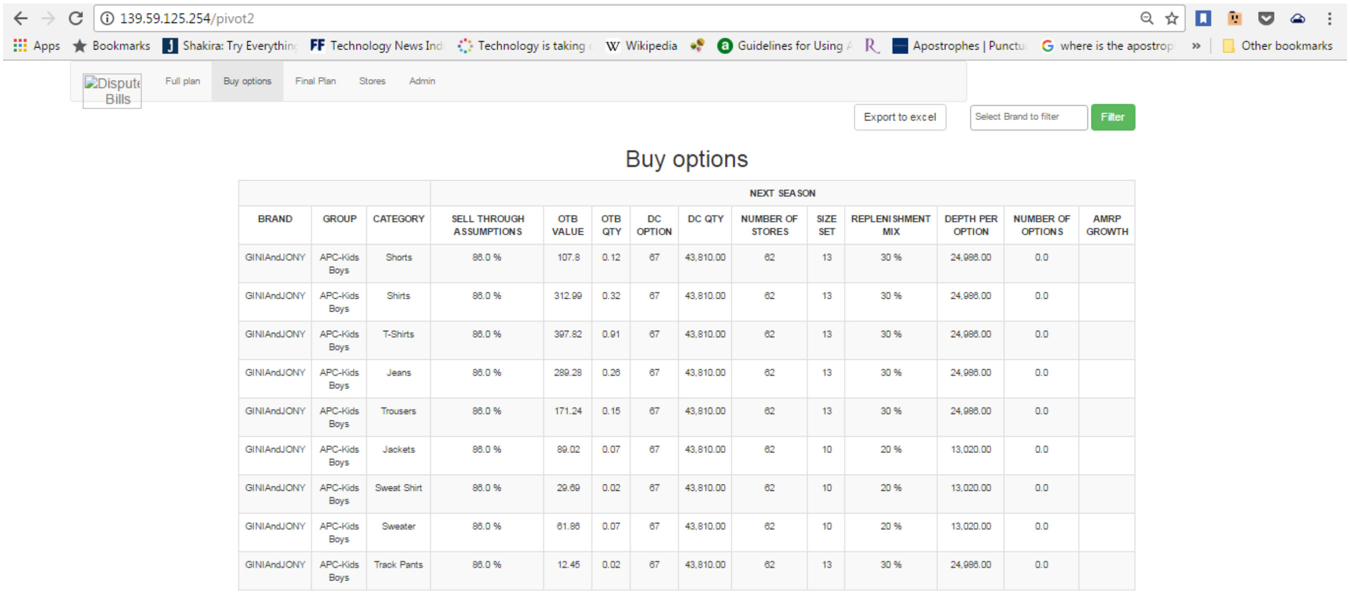
8.11 Annexure 11- range full plan



Full Plan

			LAST SEASON										NEW SEASON													
BRAND	GROUP	CATEGORY	QUANTITY	NET OF SALES	DMRP	MRP	COST PRICE	NOT MIX	LFL	OVERALL	LFL %	LFL BUDGET	BRAND GROUP BUDGET	CLASS MIX	FINAL CLASS MIX	QUANTITY	NOT	DMRP	MRP	COST PRICE	MD	IM	SELL THROUGH ASUMPTIONS	OTB VALUE
GINIAndJONY	APC-Kids Boys	Shorts	6114.0	0.56	0.56	0.84	0.52	6%	0.47	32 %	15 %	10.68	12.87	7.0 %	7.0 %	0.10	90.09	94.82	142.27	88.07	33.35 %	38.1 %	86.0 %	107.8
GINIAndJONY	APC-Kids Boys	Shirts	21202.0	2.23	2.23	3.24	2.0	26%	1.83	443 %	364 %	10.68	12.87	21.0 %	21.0 %	0.26	270.27	284.46	413.04	255.71	31.13 %	38.09 %	86.0 %	312.99
GINIAndJONY	APC-Kids Boys	T-Shirts	24201.0	1.17	1.17	1.63	1.01	14%	0.97	52 %	30 %	10.68	12.87	28.0 %	28.0 %	0.74	360.36	379.28	524.74	325.02	27.72 %	38.06 %	86.0 %	397.82
GINIAndJONY	APC-Kids Boys	Jeans	17925.0	1.99	1.99	3.12	1.93	23%	1.65	-25 %	-36 %	10.68	12.87	18.0 %	18.0 %	0.21	231.66	243.82	381.68	236.34	36.12 %	38.08 %	86.0 %	289.28
GINIAndJONY	APC-Kids Boys	Trousers	11254.0	1.28	1.28	1.94	1.2	15%	1.06	61 %	40 %	10.68	12.87	11.0 %	11.0 %	0.12	141.57	149.0	225.83	139.9	34.02 %	38.05 %	86.0 %	171.24
GINIAndJONY	APC-Kids Boys	Jackets	5555.0	0.76	0.76	1.1	0.68	9%	0.63	-12 %	-26 %	10.68	12.87	6.0 %	6.0 %	0.06	77.22	81.27	117.49	72.73	30.83 %	38.1 %	86.0 %	89.02
GINIAndJONY	APC-Kids Boys	Sweat Shirt	2324.0	0.24	0.24	0.35	0.22	3%	0.21	-33 %	-43 %	10.68	12.87	2.0 %	2.0 %	0.02	25.74	27.09	39.18	24.26	30.85 %	38.07 %	86.0 %	29.69
GINIAndJONY	APC-Kids Boys	Sweater	4419.0	0.39	0.39	0.58	0.36	4%	0.32	41 %	18 %	10.68	12.87	4.0 %	4.0 %	0.06	51.48	54.18	81.62	50.54	33.62 %	38.08 %	86.0 %	61.86

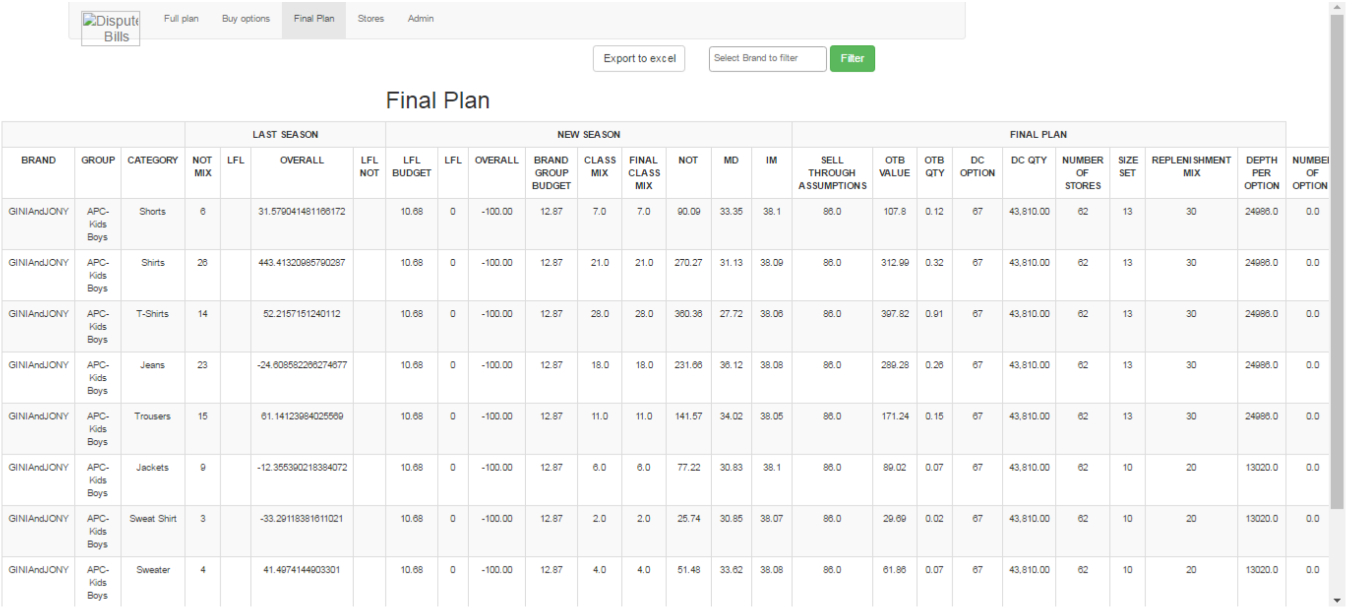
8.12 Annexure 12- range plan template buy options



Buy options

			NEXT SEASON										
BRAND	GROUP	CATEGORY	SELL THROUGH ASSUMPTIONS	OTB VALUE	OTB QTY	DC OPTION	DC QTY	NUMBER OF STORES	SIZE SET	REPLENISHMENT MIX	DEPTH PER OPTION	NUMBER OF OPTIONS	AMRP GROWTH
GINIAndJONY	APC-Kids Boys	Shorts	86.0 %	107.8	0.12	67	43,810.00	62	13	30 %	24,986.00	0.0	
GINIAndJONY	APC-Kids Boys	Shirts	86.0 %	312.99	0.32	67	43,810.00	62	13	30 %	24,986.00	0.0	
GINIAndJONY	APC-Kids Boys	T-Shirts	86.0 %	397.82	0.91	67	43,810.00	62	13	30 %	24,986.00	0.0	
GINIAndJONY	APC-Kids Boys	Jeans	86.0 %	289.28	0.26	67	43,810.00	62	13	30 %	24,986.00	0.0	
GINIAndJONY	APC-Kids Boys	Trousers	86.0 %	171.24	0.15	67	43,810.00	62	13	30 %	24,986.00	0.0	
GINIAndJONY	APC-Kids Boys	Jackets	86.0 %	89.02	0.07	67	43,810.00	62	10	20 %	13,020.00	0.0	
GINIAndJONY	APC-Kids Boys	Sweat Shirt	86.0 %	29.69	0.02	67	43,810.00	62	10	20 %	13,020.00	0.0	
GINIAndJONY	APC-Kids Boys	Sweater	86.0 %	61.86	0.07	67	43,810.00	62	10	20 %	13,020.00	0.0	
GINIAndJONY	APC-Kids Boys	Track Pants	86.0 %	12.45	0.02	67	43,810.00	62	13	30 %	24,986.00	0.0	

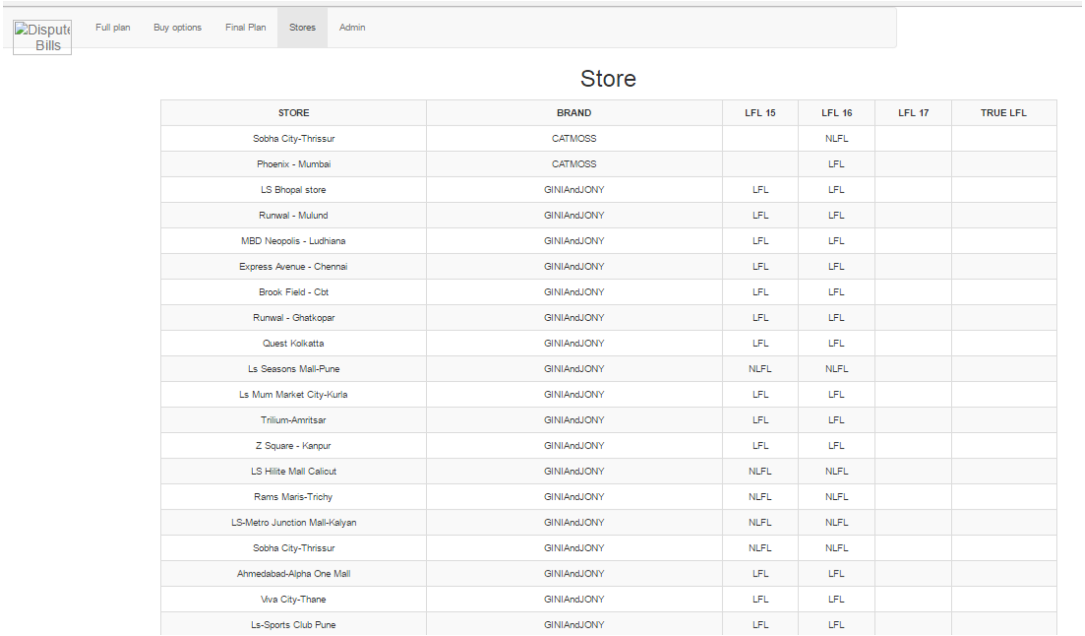
8.13 Annexure 13 – range plan final



Final Plan

			LAST SEASON				NEW SEASON									FINAL PLAN									
BRAND	GROUP	CATEGORY	NOT MIX	LFL	OVERALL	LFL NOT	LFL BUDGET	LFL	OVERALL	BRAND GROUP BUDGET	CLASS MIX	FINAL CLASS MIX	NOT	MD	IM	SELL THROUGH ASSUMPTIONS	OTB VALUE	OTB QTY	DC OPTION	DC QTY	NUMBER OF STORES	SIZE SET	REPLENISHMENT MIX	DEPTH PER OPTION	NUMBEI OF OPTION
GINIAndJONY	APC-Kids Boys	Shorts	6		31.579041481166172		10.68	0	-100.00	12.87	7.0	7.0	90.09	33.35	38.1	86.0	107.8	0.12	67	43,810.00	62	13	30	24986.0	0.0
GINIAndJONY	APC-Kids Boys	Shirts	26		443.41320985790287		10.68	0	-100.00	12.87	21.0	21.0	270.27	31.13	38.09	86.0	312.99	0.32	67	43,810.00	62	13	30	24986.0	0.0
GINIAndJONY	APC-Kids Boys	T-Shirts	14		52.2157151240112		10.68	0	-100.00	12.87	28.0	28.0	360.36	27.72	38.06	86.0	397.82	0.91	67	43,810.00	62	13	30	24986.0	0.0
GINIAndJONY	APC-Kids Boys	Jeans	23		-24.608582266274677		10.68	0	-100.00	12.87	18.0	18.0	231.66	36.12	38.08	86.0	289.28	0.26	67	43,810.00	62	13	30	24986.0	0.0
GINIAndJONY	APC-Kids Boys	Trousers	15		61.14123984025569		10.68	0	-100.00	12.87	11.0	11.0	141.57	34.02	38.05	86.0	171.24	0.15	67	43,810.00	62	13	30	24986.0	0.0
GINIAndJONY	APC-Kids Boys	Jackets	9		-12.355390218384072		10.68	0	-100.00	12.87	6.0	6.0	77.22	30.83	38.1	86.0	89.02	0.07	67	43,810.00	62	10	20	13020.0	0.0
GINIAndJONY	APC-Kids Boys	Sweat Shirt	3		-33.29118381611021		10.68	0	-100.00	12.87	2.0	2.0	25.74	30.85	38.07	86.0	29.69	0.02	67	43,810.00	62	10	20	13020.0	0.0
GINIAndJONY	APC-Kids Boys	Sweater	4		41.4974144903301		10.68	0	-100.00	12.87	4.0	4.0	51.48	33.62	38.08	86.0	61.86	0.07	67	43,810.00	62	10	20	13020.0	0.0

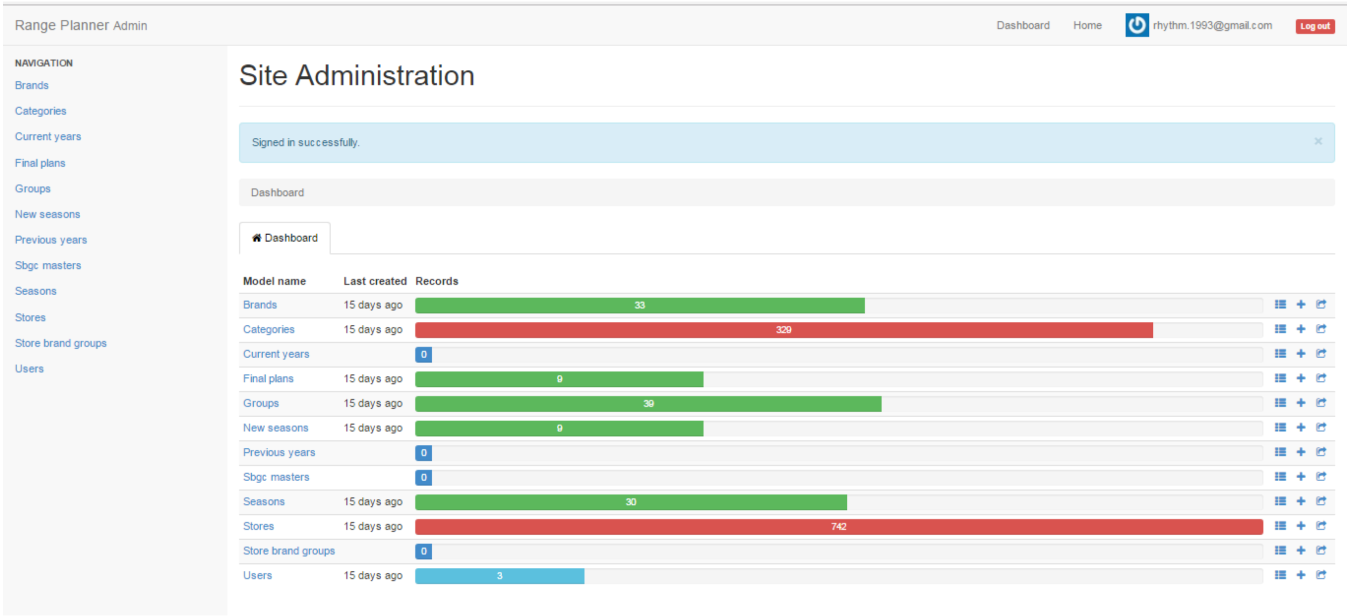
8.14 Annexure 14 – Range Plan- stores



Store

STORE	BRAND	LFL 15	LFL 16	LFL 17	TRUE LFL
Sobha City-Thrissur	CATMOSS		NLFL		
Phoenix - Mumbai	CATMOSS		LFL		
LS Bhopal store	GINIAndJONY	LFL	LFL		
Runwal - Mulund	GINIAndJONY	LFL	LFL		
MBD Neopolis - Ludhiana	GINIAndJONY	LFL	LFL		
Express Avenue - Chennai	GINIAndJONY	LFL	LFL		
Brook Field - Cbt	GINIAndJONY	LFL	LFL		
Runwal - Ghatkopar	GINIAndJONY	LFL	LFL		
Quest Kolkatta	GINIAndJONY	LFL	LFL		
Ls Seasons Mall-Pune	GINIAndJONY	NLFL	NLFL		
Ls Mum Market City-Kurla	GINIAndJONY	LFL	LFL		
Trilium-Amritsar	GINIAndJONY	LFL	LFL		
Z Square - Kanpur	GINIAndJONY	LFL	LFL		
LS Hilite Mall Calicut	GINIAndJONY	NLFL	NLFL		
Rams Maris-Trichy	GINIAndJONY	NLFL	NLFL		
LS-Metro Junction Mall-Kalyan	GINIAndJONY	NLFL	NLFL		
Sobha City-Thrissur	GINIAndJONY	NLFL	NLFL		
Ahmedabad-Alpha One Mall	GINIAndJONY	LFL	LFL		
Viva City-Thane	GINIAndJONY	LFL	LFL		
Ls-Sports Club Pune	GINIAndJONY	LFL	LFL		

8.15 Annexure 15 – range plan admin



Site Administration

Signed in successfully.

Model name	Last created	Records
Brands	15 days ago	33
Categories	15 days ago	329
Current years		0
Final plans	15 days ago	9
Groups	15 days ago	39
New seasons	15 days ago	9
Previous years		0
Sbgc masters		0
Seasons	15 days ago	30
Stores	15 days ago	742
Store brand groups		0
Users	15 days ago	3

8.16 Annexure 16 – manual

MANUAL

For range planner:

Navigating to the directory.

1. Open terminal/command prompt
2. git clone git@github.com:aakashaggarwal/range_planner.git
3. cd range_planner
4. Copy the csv you want to seed the data with in this folder with appropriate name.
5. git add -A
6. git commit -m "New csv."
7. git push origin master
8. ssh root@139.59.125.254 (When asked for password enter Qwerty123!)
9. Changing the context from your local machine to the server.
10. /home/rails/coretracker/
11. Navigating to the directory.
12. git stash
13. git pull origin master
14. git stash pop
15. ssh root@139.59.125.254 (When asked for password enter Qwerty123!)
16. Changing the context from your local machine to the server.
17. /home/rails/range_planner/

Following are the steps to run the range planner codebase.

1. bundle install

Make sure all dependencies in your Gemfile are available to your application.

2. rake db:create

This command will take all database configuration from config/database.yml file and create appropriate database of current environment’s database.

3. rake db:migrate

The creates tables in database. It takes all files under db/migrate/ directory and execute one by one from older to newer files.

4. rake seeder:seed_base_season_new

This command seeds the value of the current season with all the base factors.

5. rake seeder:seed_base_season_old

This command seeds the value of the last season with all the base factors.

6. rake seeder:seed_db_final_plans

This command seeds the value of the final plan being made for the current ongoing season.

7. rake seeder:seed_db_seasons

This command seeds the value of the base factors being made for all the seasons.

8. rake seeder:seed_updated_season_value_new

This command seeds the value of the new LFL sales for the current season.

9. rake seeder:seed_updated_season_value_old

This command seeds the value of the new LFL sales for the last season.

10. rails server

The rails server command launches a web server named Puma which comes bundled with Rails. You'll use this any time you want to access your application through a web browser.

Following are the steps to run the core tracker codebase.

1. Open terminal/command prompt
2. git clone git@github.com:aakashaggarwal/range_planner.git
3. cd range_planner
4. Copy the csv you want to seed the data with in this folder with appropriate name.
5. git add -A
6. git commit -m "New csv."
7. git push origin master
8. ssh root@139.59.125.254 (When asked for password enter Qwerty123!)
9. Changing the context from your local machine to the server.
10. /home/rails/coretracker/
11. Navigating to the directory.
12. git stash
13. git pull origin master
14. git stash pop
15. ssh root@139.59.125.254 (When asked for password enter Qwerty123!)
16. Changing the context from your local machine to the server.
17. /home/rails/coretracker/
18. Navigating to the directory.

19. undle install

20. Make sure all dependencies in your Gemfile are available to your application.

21. rake db:create

22. This command will take all database configuration from config/database.yml file and create appropriate database of current environment's database.

23. rake db:migrate

24. The creates tables in database. It takes all files under db/migrate/ directory and execute one by one from older to newer files.

25. rake seeder:seed_db_brands

26. This command seeds the value all the brands .

27. rake seeder:seed_db_lfl_nlfl_stores

28. This command seeds the value all the LFL and NON LFL stores.

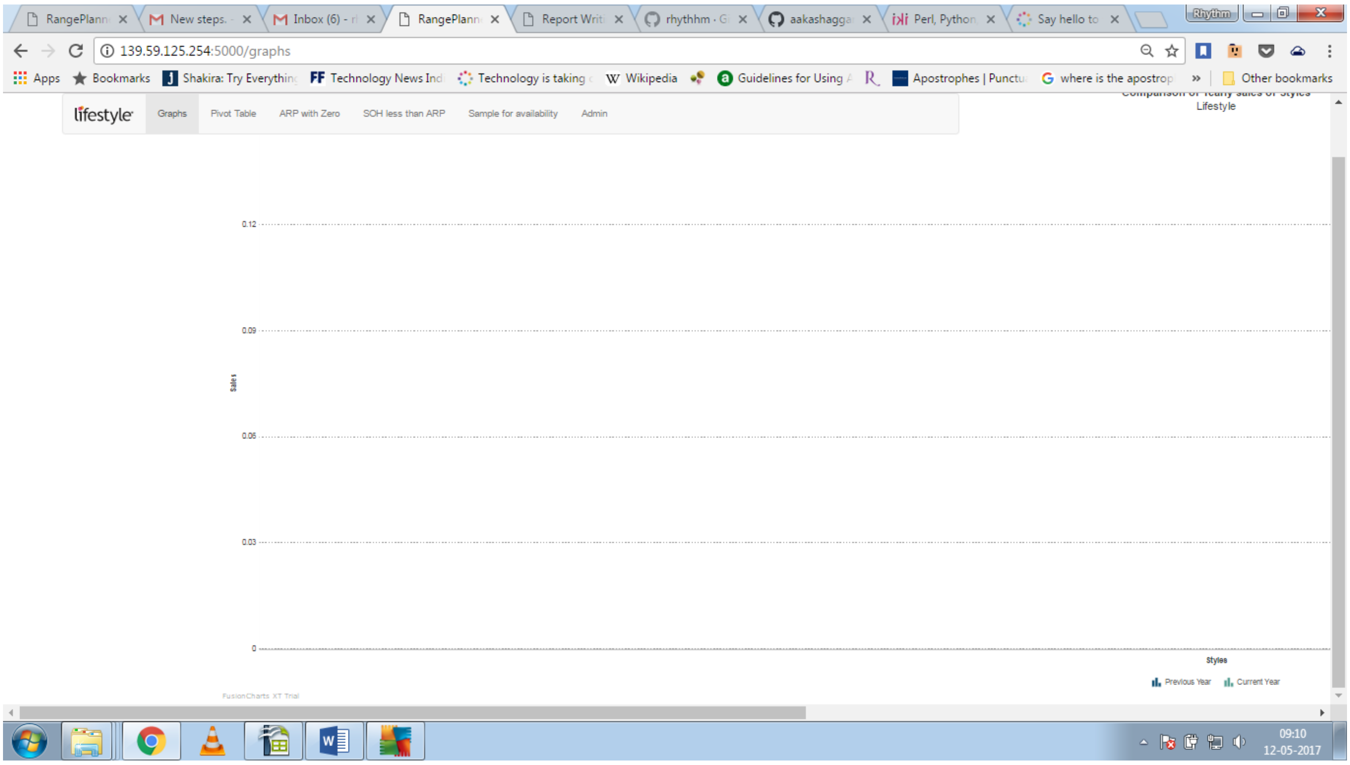
29. rake seeder:seed_db_core_tracker_templates

30. This command seeds the value of the entire template for core tracker.

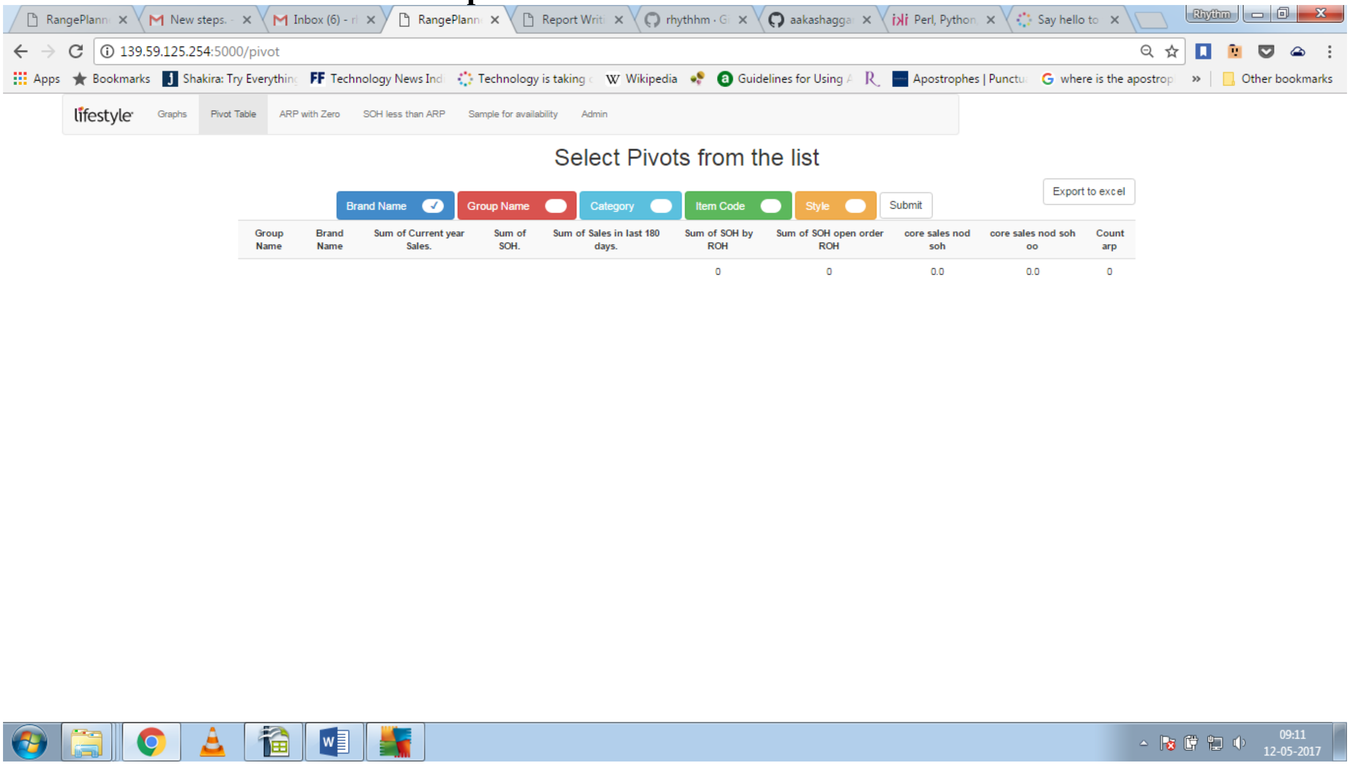
31. rails server

The rails server command launches a web server named Puma which comes bundled with Rails. You'll use this any time you want to access your application through a web browser.

8.17 Annexure 17 – core tracker – Graphs



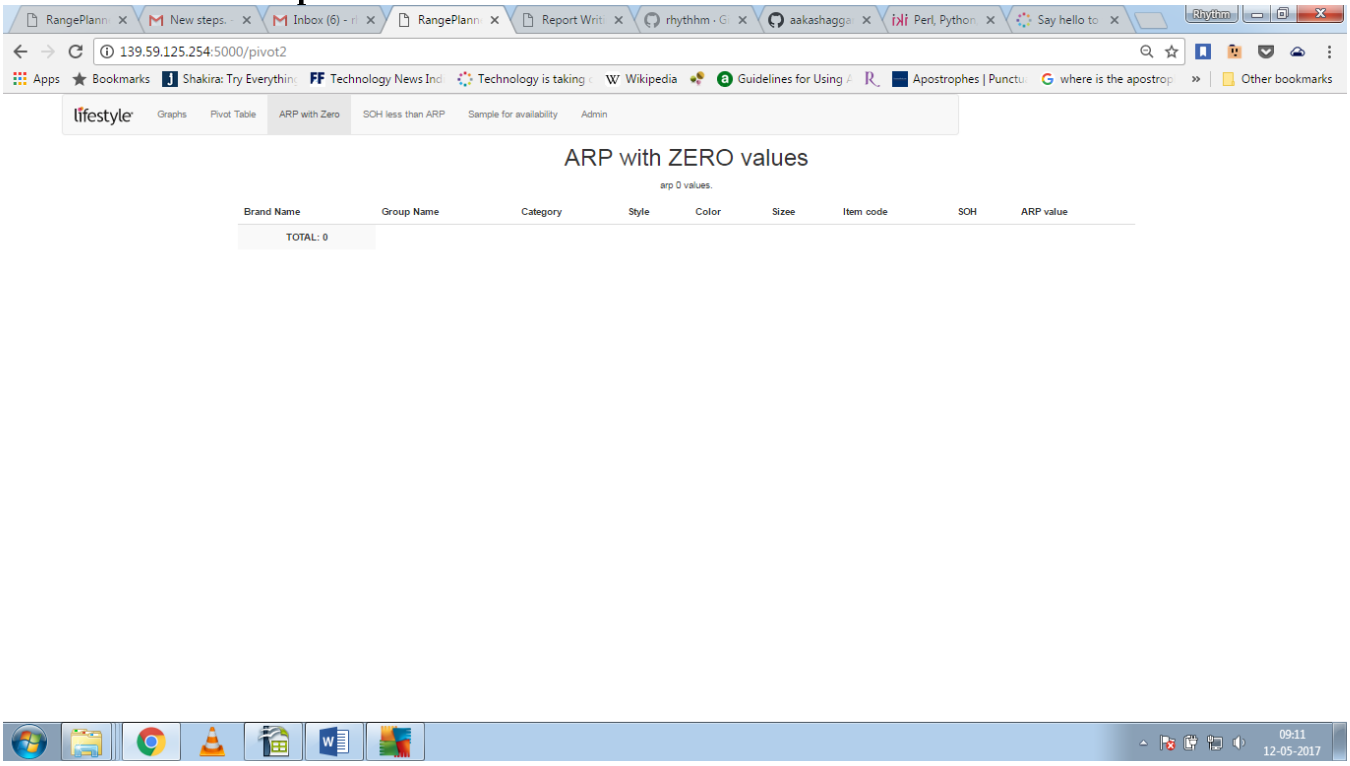
8.18 annexure 18 – core tracker – pivot



Select Pivots from the list

Group Name	Brand Name	Sum of Current year Sales.	Sum of SOH.	Sum of Sales in last 180 days.	Sum of SOH by ROH	Sum of SOH open order ROH	core sales nod soh	core sales nod soh oo	Count arp
					0	0	0.0	0.0	0

8.19 annexure 19 – arp zero

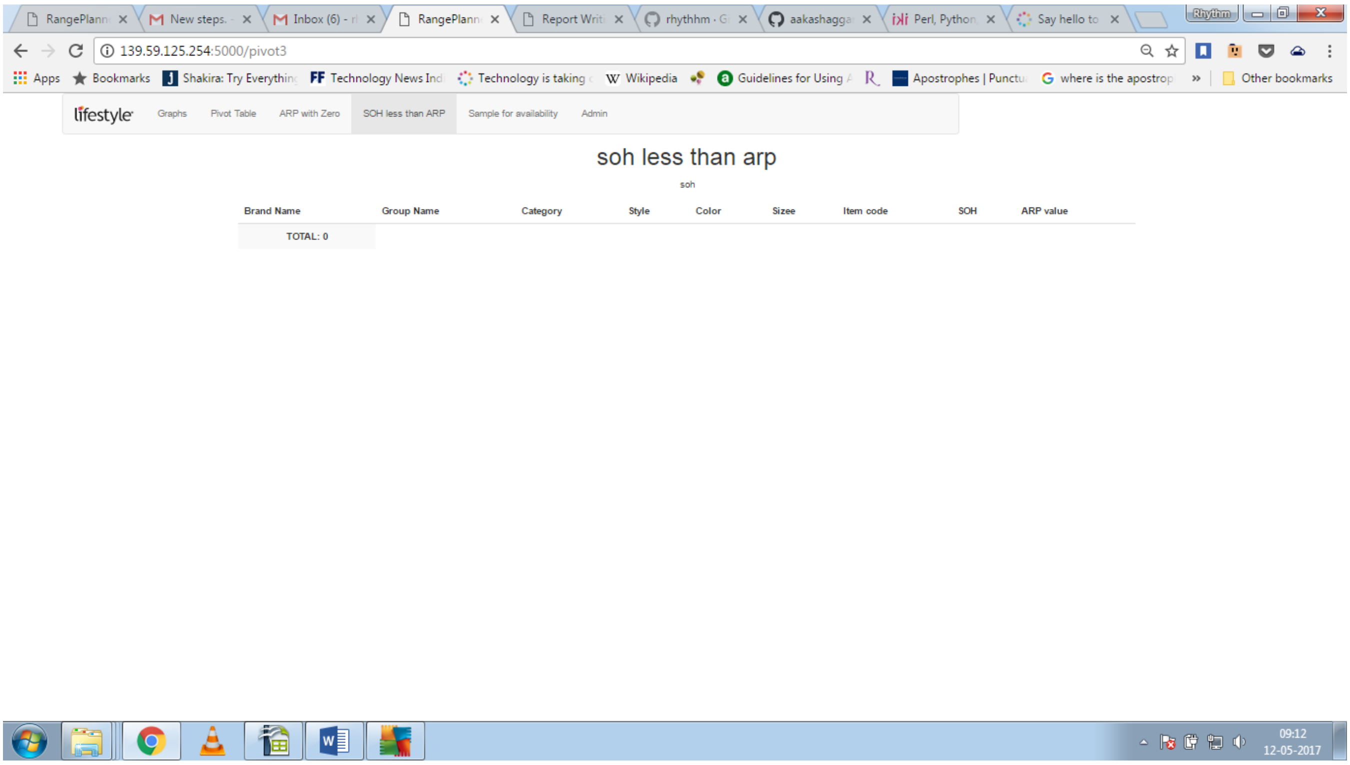


ARP with ZERO values

arp 0 values.

Brand Name	Group Name	Category	Style	Color	Sizee	Item code	SOH	ARP value
TOTAL: 0								

8.20 annexure 20 – core tracker – soh < arp



soh less than arp

soh

Brand Name	Group Name	Category	Style	Color	Sizee	Item code	SOH	ARP value
TOTAL: 0								

8.21 annexure 20 – core tracker – overall availability

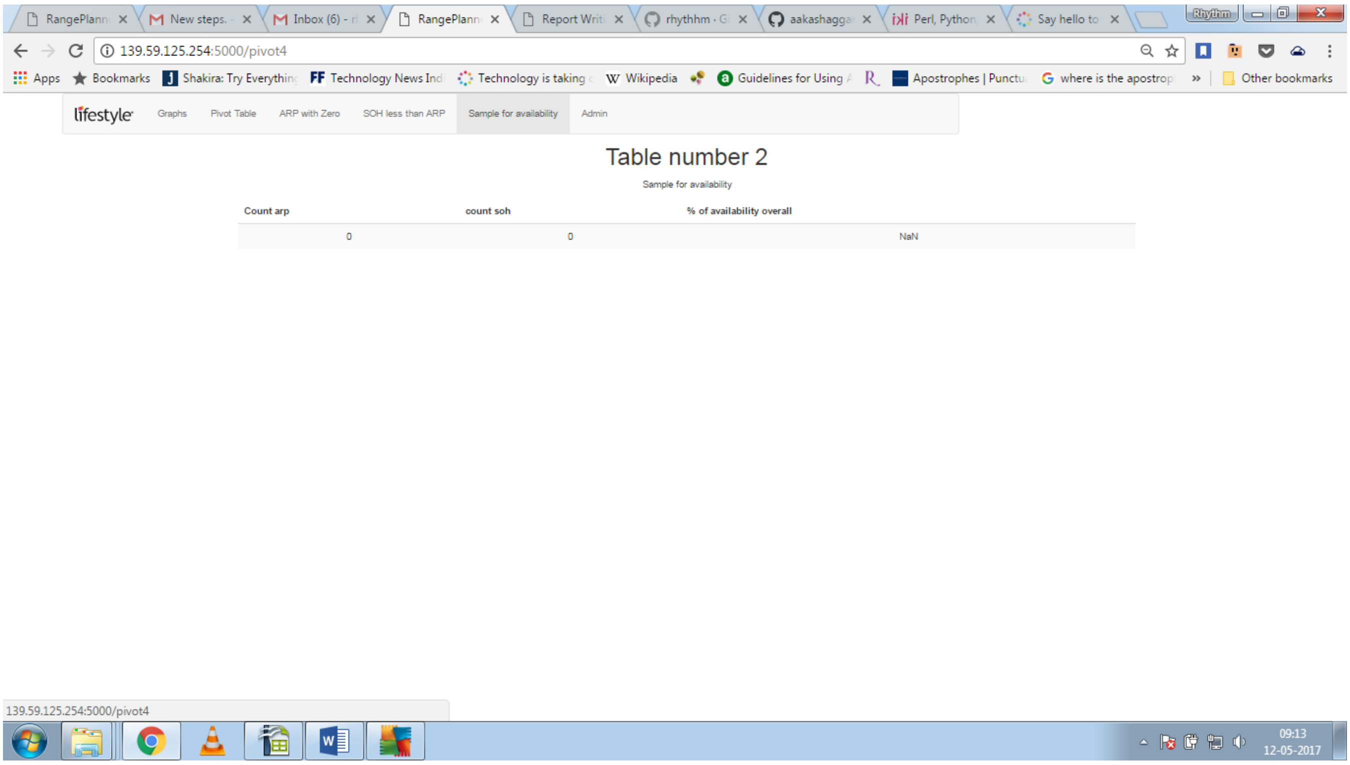
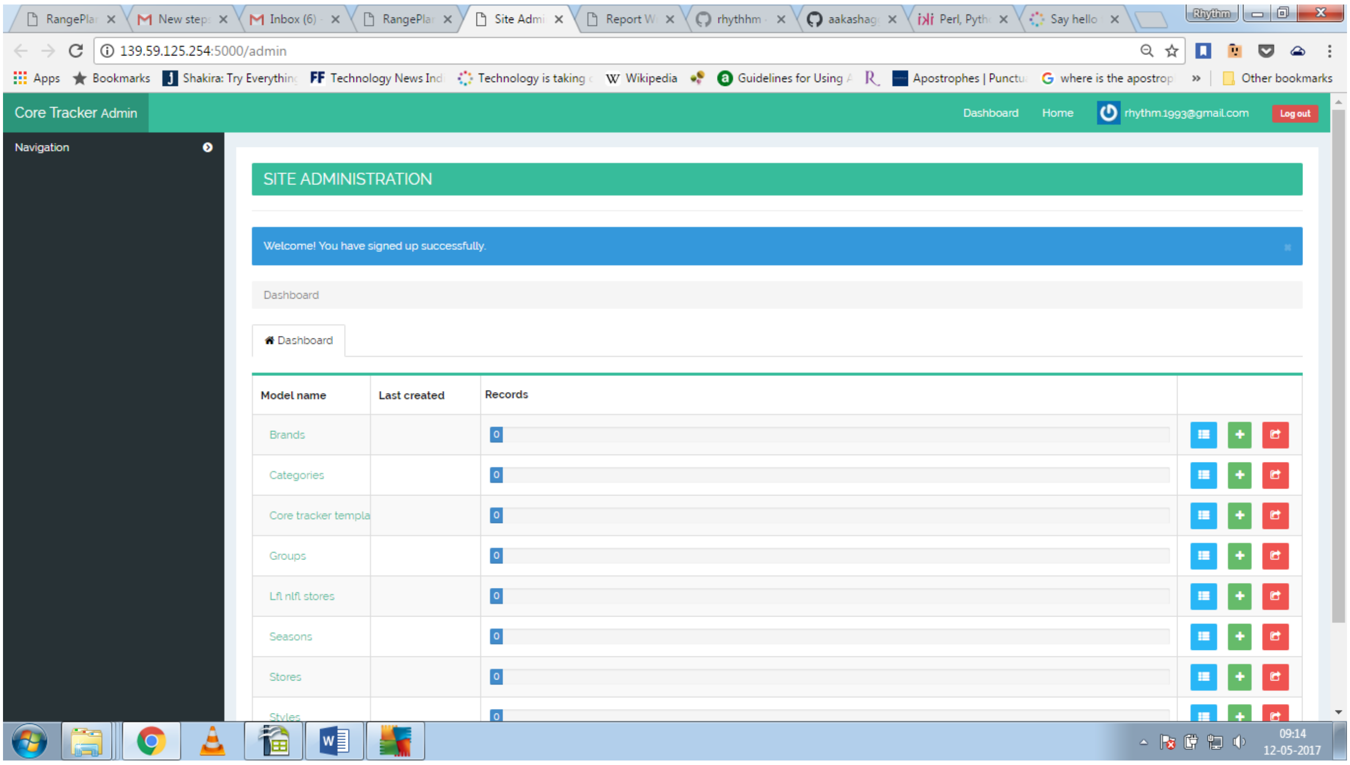


Table number 2

Sample for availability

Count arp	count soh	% of availability overall
0	0	NaN

8.22 annexure 21 – core tracker- admin



SITE ADMINISTRATION

Welcome! You have signed up successfully.

Dashboard

Dashboard

Model name	Last created	Records
Brands		0
Categories		0
Core tracker templa		0
Groups		0
Lfl nlfl stores		0
Seasons		0
Stores		0
Styles		0

GLOSSARY

March-June (Spring-Summer)	122 Days
July-August (Spring-Summer EOSS)	62 Days
September-December (Autumn-Winter)	122 Days
January-February (Autumn-Winter EOSS)	59 Days
LFL	Like for like, store that has completed 180 days
OTB	Open to buy, otb = decided amount – leftover stock
Last Year	Last Year means the fiscal year most recently completed.
Current Year	Current year means the running fiscal year.
Next Year	Next year means the following year.
Pre-Season	The fresh merchandise which has been sold before the actual season start.
Pre-EOSS	The merchandise which has been sold before the EOSS.
EOSS	The merchandise sold in EOSS.
Brands	It is the name of the company. Example: Allen Solly, United Colors Of Benetton.
Territory	Territory is the area of land on the larger scale. Example: North, Andhra Pradesh.
Store Name	It is the name given to the store.
Department	Department is the division of Apparel which share common characteristics. Example: Casual wear, Formal wear.
Category	It is a sub division of department. Example: Shirt, T-Shirt, Jackets.
Area	There are two types of Area: • **Retail Area-** it is the sum total of the area which a brand needs to keep the fixtures. • **Carpet Area-** it is the percentage of area which a brands requires to display the merchandise.
Trading Days	
Last Year	No. Of Days store traded last year.
Current Year	No. Of Days store will trade Current year.
Next Year	No. Of Days store will trade Next year.
PDQ	
Last Year	It is the Per Day Quantity of merchandise sold last year. PDQ= Sales QTY of Last Year Season/Trading Days in Last year.
Current Year	It is the Per Day Quantity of merchandise which is will be sold in current year. PDQ= Sales QTY of Current Year Season/Trading Days in Current year.
Next Year	It is the Per Day Quantity of merchandise which is will be selling in Next year. PDQ= Sales QTY of Next Year Season/Trading Days in Next year.

PDV	
Last Year	It is the Per Day Value of merchandise sold last year. PDQ= NOT Sales of Last Year Season/Trading Days in Last year.
Current Year	It is the Per Day Value of merchandise which is will be sold in current year. PDQ= NOT Sales of Current Year Season/Trading Days in Current year.
Next Year	It is the Per Day Value of merchandise which is will be selling in Next year. PDQ= NOT Sales of Next Year Season/Trading Days in Next year.
Intake Margin	
Last Year	It Shows the Intake Margin for the complete season of Last Year. Sale MRP-Sale CP/Sale MRP
Current Year	It Shows the Intake Margin for the complete season of Current Year. Sale MRP-Sale CP/Sale MRP
Next Year	It Shows the Intake Margin for the complete season of Next Year. Sale MRP-Sale CP/Sale MRP
Mark Down	
Last Year	It Shows the Mark Down for the complete season of Last Year. Sale MRP-DMRP/Sale MRP
Current Year	It Shows the Mark Down for the complete season of current Year. Sale MRP-DMRP/Sale MRP
Next Year	It Shows the Mark Down for the complete season of Next Year. Sale MRP-DMRP/Sale MRP
AMRP	
Last Year	It Shows the AMRP for the complete season of Last Year. Sale MRP/Sale Qty
Current Year	It Shows the AMRP for the complete season of Current Year. Sale MRP/Sale Qty
Next Year	It Shows the AMRP for the complete season of Current Year. Sale MRP/Sale Qty
Sale Quantity	This figure shows the total Quantity of merchandise sold.
NOT Sales	NOT stands for Net of Tax. This figure shows the total Net Sale of the merchandise. Net Sale=MRP-TAX-Discount
Sale Cost Price	This figure shows the total Cost Price of the merchandise sold.
Sale MRP	This figure shows the total Maximum Retail Price of the merchandise sold.
Sale Tax	This figure shows the total tax on the merchandise sold. TAX=DMRP-NOT
DMRP	This figure shows the total Discounted MRP of the merchandise sold. DMRP=Sale MRP-Discount
PDV	PDV stands for Per Day Value. PDV=NOT Sale/No. Of Trading Days
Intake Margin	Intake Margin=(MRP-CP)/MRP
MD%	MD% stands for Mark Down Percentage MD%=(Sale MRP-DMRP)/Sale MRP

Contribution of A in Total Item	Contribution= NOT sale of A/Sum of NOT Sale of total item
Elasticity	Elasticity=(PDV of Season EOSS-PDV of Pre-EOSS)/PDV of Pre-EOSS
Budget Achievement %	It the comparison between the projected budget of the year and the actual budget achieved in that year. Budget Achievement %= Sale Value/Budget
AMRP	Average MRP=Sale MRP/Qty
Department	Department is the division of Apparel which share common characteristics. Example: Casual wear, Formal wear.
Brand	It is the name of the company. Example: Allen Solly, United Colors Of Benetton.
Category	It is a sub division of department. Example: Shirt, T-Shirt, Jackets.
Style	It is a particular design or form of the merchandise.
Core/Fashion	**Core-** Those merchandise which will remain on the store for selling throughout the season. **Fashion-** Those Merchandise which ones sold there are chances of no replenishment.
Color	It is the property possessed by an object of producing different sensations on the eye as a result of the way it reflects or emits light
MRP	A **maximum retail price** is an upper limit on the price of a good set by the manufacturer or distributor of a good on retailers or final point-of-sale purchases
Store Code	Store Code is the system of numbers or letters which are assigned to the stores. Every store has its own code.
Location	Location is a particular place or position of the store.
Territory	Territory is the area of land on the larger scale. Example: North, Andhra Pradesh.
LFL/Others	LFL stands for **Like For Like.** Those stores which have completed one full cycle of financial year falls under LFL, rest falls under others.
Department	Department is the division of Apparel which share common characteristics. Example: Casual wear, Formal wear.
Brand	It is the name of the company. Example: Allen Solly, United Colors Of Benetton.
Category	It is a sub division of department. Example: Shirt, T-Shirt, Jackets.
Brand Type	For companys ease, brand has been classed into Private Label and Brands.
Purchase Type	It defines the method on which the company purchases its merchandise. Types of Purchase Type: • **SOR**-Sale Or Return is an arrangement by which a retailer pays only for goods sold, returning those that are unsold to the wholesaler or manufacturer. • **Outright**- To pay the full purchase price amount in cash. • **Consignment**- Two parties consignor and consignee, the ownership of the goods is not transferred to consignee, only the possession of the goods is transferred.
Month	It is the twelve named period in which the year is divided.
STD Last Year	

Quantity	It is the last year sale quantity for STD.
NOT	NOT stands for Net of Tax. It is the last year Net Sales for STD.
CP	CP stands for Cost Price. It is the last year Cost Price for STD.
MRP	MRP stands for Maximum Retail Price. It is the last year MRP for STD.
TAX	It is the last year Tax for STD.
STD Current Year	
Quantity	It is the current year sale quantity for STD.
NOT	NOT stands for Net of Tax. It is the current year Net Sales for STD.
CP	CP stands for Cost Price. It is the current year Cost Price for STD.
MRP	MRP stands for Maximum Retail Price. It is the current year MRP for STD.
TAX	It is the current year Tax for STD.
STD Budget	
Quantity	It is the season budgeted quantity for STD.
NOT	It is the season budgeted Net Sales for STD.
April-August Last Year	
Quantity	The sale quantity for the last year April to August.
NOT	The Net Sales for the last year April to August.
CP	The Cost Price for the last year April to August.
MRP	The Maximum Retail Price for the last year April to August.
Tax	The tax involved for the last year April to August.
April-August Current Year	
Quantity	The sale quantity for the last year April to August.
NOT	The Net Sales for the last year April to August.
CP	The Cost Price for the last year April to August.
MRP	The Maximum Retail Price for the last year April to August.
TAX	The tax involved for the last year April to August.
April-August Budget	
Quantity	The Budgeted quantity for April to August.
NOT	The Budgeted NOT for April to August.
Closing Stock	
Quantity	The quantity of the remaining stock at every month end. **Closing Stock=Stock on Hand(SOH)+open PO**
Cost Value	The Cost Value of the closing stock at every month end.
Order	
Quantity	It is the batch quantity of merchandise which the buyer request to the brands.
Cost Value	It is the cost value of the order quantity.
GRN	
Quantity	That order Quantity which has been realised by the company is called Goods Receipt Note (GRN) quantity.
Cost Value	It is the cost value of the order quantity which has been realised.
Open PO	
Quantity	It is the Purchase Order Quantity which has been raised by the Buyer. **Open PO QTY=Order QTY-GRN QTY**
Cost Value	It is the cost value of Open PO quantity.

RTV	
Quantity	Return-To-Vendor Quantity is the number of product which has been requested to be returned to the supplier.
Cost Value	It is the cost value of the quantity of the merchandise which has been requested to be returned to the supplier.
SHIPPED	
Quantity	It is the number of quantity of the merchandise which has been shipped back to the vendor.
Cost Value	It is the cost value of the merchandise which has been shipped.
PENDING PRN	
Quantity	The quantity of merchandise which is to be shipped and is requested for RTV. **Pending PRN=RTV Qty-Shipped Qty**
Cost Value	The cost value of the quantity of pending PRN.
Season Budget Month on month	It is the budget of the season month on month
Area	There are two types of Area: • **Retail Area**- it is the sum total of the area which a brand needs to keep the fixtures. • **Carpet Area**- it is the percentage of area which a brands requires to display the merchandise.
Display Capacity	It is the Product of the **Total Option, total Depth and Profile**
Forward Cover(FC)	It is the sale of merchandise at a fixed future date. = 3 months budget/no. Of Trading Days
NOD	NOD stands for No. Of Days. Two types of NOD. • Current Year NOD • Forward Cover NOD = Closing Stock/Rate of Sale(ROS)
YTD GROWTH NOT	It manifests the current year NOT growth against last year NOT growth. =(YTD CY NOT-YTD LY NOT)/YTD LY NOT
FC NOD ON ACH %	FC NOD on ACH% stands for Forward Cover No. Of Days on Achievement %. =NET CLOSING QTY/(STD ACH NOT*FC ROS)
CY AMRP	CY AMRP stands for Current year Average Maximum Retail Price. =STD CY MRP/STD CY QTY
LY AMRP	LY AMRP stands for Last year Average Maximum Retail Price. =STD LY MRP/STD LY QTY
CY ASP	CY ASP stands for Current Year Average Sale Per Day. =STD CY Not/STD CY Qty
LY ASP	LY ASP stands for Last Year Average Sale Per Day. =STD LY Not/STD LY Qty
CY MD	CY MD stands for Current Year Markdown. =(STD CY MRP-(STD CY Not+STD CY TAX))/STD CY MRP
LY MD	LY MD stands for Last Year Markdown. =(STD LY MRP-(STD LY Not+STD LY TAX))/STD LY MRP
DENSITY	No. Of Stock Per Unit Area.

	=(CLOSING Qty-PENDING PRN QTY)/AREA
YTD LY QTY	YTD LY QTY stands for Year till date Last year Quantity. =STD LY Qty+APR-AUG LY Qty
YTD LY NOT	YTD LY NOT stands for Year till date Last year Net Of Tax. =STD LY Not+APR-AUG LY Not
YTD LY CP	YTD LY CP stands for Year till date Last year Cost Price. =STD LY CP+ APR-AUG LY CP
YTD LY MRP	YTD LY MRP stands for Year till date Last year Maximum Retail Price. =STD LY MRP+ APR-AUG LY MRP
YTD LY TAX	YTD LY Tax stands for Year till date Last year Tax. =STD LY TAX+APR-AUG LY TAX
YTD CY QTY	YTD CY QTY stands for Year till date Current year quanity. = STD CY Qty+ APR-AUG CY Qty
YTD CY NOT	YTD CY NOT stands for Year till date Current year Net of Tax. =STD CY Not+ APR-AUG CY Not
YTD CY CP	YTD CY CP stands for Year till date Current year Cost Price. =STD CY CP+APR-AUG CY CP
YTD CY MRP	YTD CY MRP stands for Year till date Current year Maximum Retail Price. =APR-AUG CY MRP+STD CY MRP
YTD CY TAX	YTD CY Tax stands for Year till date Current year Tax. =STD CY TAX+APR-AUG CY TAX
YTD BUDGET QTY	YTD BUDGET QTY stands for Year till date Budget Quantity. =STD Budget Qty+APR-AUG Budget Qty
YTD BUDGET NOT	YTD BUDGET NOT stands for Year till date Budget Net Of Tax. = STD Budget NOT+ APR-AUG Budget NOT
STD GROWTH QTY	STD Growth QTY shows the growth in current year against last year at STD Level. =(STD CY Qty-STD LY Qty)/STD LY Qty
YTD GROWTH QTY	YTD Growth QTY shows the growth in current year against last year at YTD Level. =(YTD CY QTY-YTD LY QTY)/YTD LY QTY
NM CY STD	NM CY STD stands for Net Margin in Current Year at STD. =(STD CY Not-STD CY CP)/STD CY Not
NM CY YTD	NM CY YTD stands for Net Margin in Current Year at YTD. (YTD CY NOT-YTD CY CP)/YTD CY NOT
NM YTD LY	NM YTD LY stands for Net Margin Year Till Date Last year. =(YTD LY NOT-YTD LY CP)/YTD LY NOT
CY ROS	This shows the current year Rate of Sale. =STD CY Qty/153
FC NOD W/O OPEN PO	(NET CLOSING QTY-Open PO Qty)/FC ROS
YTD ACH NOT	YTD CY NOT/YTD BUDGET NOT
NET CLOSING VAL	CLOSING Val+Open PO Val-PENDING PRN VALUE
FC ROS	(MONTH1+MONTH2+MONTH3)/92
STD GROWTH NOT	(STD CY Not-STD LY Not)/STD LY Not
FC NOD NET CLOSING	NET CLOSING QTY/FC ROS

STD ACH NOT	STD CY Not/ STD Budget NOT
NET CLOSING QTY	CLOSING Qty+Open PO Qty-PENDING PRN QTY

CY SPF	('CY SPD' /'CY AREA')
PY SPF	= ('PY SPD' /'PY AREA')
CY MGN %	= ('CY MARGIN'/' YTD CY NOT')
PY MGN %	= ('PY MARGIN'/' YTD PY NOT')
CY MPF	= ('CY MGN %' *'CY SPF')
PY MPF	=('PY MGN %' *'PY SPF')
SPF GROWTH %	= ('CY SPF'-'PY SPF')/'PY SPF'
MPF GROWTH %	= ('CY MPF' -'PY MPF')/'PY MPF'
AREA GROWTH %	= ('CY AREA' -'PY AREA')/'PY AREA'
VALUE GROWTH %	= ('CY SPD' -'PY SPD')/'PY SPD'

Fixtures	They are the furniture's in different form which is used for displaying the product. • Wall • Promo • Gondola • T-Bar • Railing • Techno • Slat wall • Pigeon Hole
Net Retail Area	The area within the walls of the shop or store to which the public has access or from where sales are made, excluding fitting rooms, checkouts, the area in front of checkouts, serving counters and the area behind used by serving staff, customer services areas, and internal lobbies in which goods are displayed cafes and customer toilets. =area of the fixtures(in metres)/Loading factor
Carpet Area	Carpet area is the area enclosed within the walls, actual area to lay the carpet. This area does not include the thickness of the inner walls.
Efficiency	A level of performance that describes a process that uses the lowest amount of inputs to create the greatest amount of outputs. =Net Retail Area/Carpet Area