**Confluence Apps**

**A Rutgers 2019 Data Engineering and Visualization ETL Project**

**Brown Varghese, Krunal Panchal, Radha Mahalingam**

**Quick Summary**

The ever-changing mobile landscape is a challenging space to navigate. The percentage of mobile over desktop is only increasing. Android holds about 53.2% of the smartphone market, while iOS is 43%. To get more people to download various app, we need to make sure the consumers can easily find the apps that they need. We offer an ETL pipeline that assists young Adults and Parents to make an informed choice to buy/download free/paid mobile/tablet apps from Apple store or Google Play Store. The source data obtained are from two different data sources in Kaggle. At the end of the ETL process, we provide our data in a NoSQL database (Mongo DB) which exposes, for different categories/genres, type (free/paid) and content ratings, the KPIs to determine which app store for various categories will deliver the most value to the Mobile apps consumers. This current version is focused towards data from Google and Apple App stores but can be extended to include other app stores in future. This ETL pipeline will be the backbone to build a mobile app analytics platform in future – an intuitive and user-friendly front end to help consumers as well as producers

**Steps to run the Pipeline**

1. Make sure the downloaded files - googleplaystore.csv and appleStore.csv are stored in ‘Resources’ folder in the *pwd (current working directory).*
2. From pwd, run Jupyter Notebook.
3. Select Apps\_Data\_Extract.ipynb and run the script.
4. Select Apps\_Data\_transformation.ipynb and run the script
5. To load the CSV generated from programs above we will need to run following python script. This will require following steps
   1. Make sure MongoDB Compass is installed
   2. Launch mongod on Gitbash.
   3. Launch Terminal or command line to run the python file mentioned below:
      1. Confluence\_Apps.py
   4. Once script has run to success launch MongoDB Compass
   5. Now you should be able to see following DB and collections:

* **Database:**

Confluence\_Apps

* **Collections:**

Android\_Data.segment

Apple\_Data.segment

Apple\_Google\_Summary.segment

**Narrative / Motivation**

We are providing a Database for the Mobile apps consumers – especially young adults and parents to make informed decisions about what apps to buy / download for various categories of apps from Google Play Store and Apple store, based on apps price, user rating and number of users (installs).

**Final Schema / Data Model / How to use the data**

The process of data extraction and data transformation resulted in the following final data model, which was loaded on NoSQL database – Mongo DB. This data could have been loaded in a relational database like MySQL, as well. But we choose to go with Mongo DB, because of the following reasons:

* 1. We wanted to learn more Mongo DB as we did not have prior experience.
  2. We believe that in future, there will be document types (rows), which will hold unstructured data fields representing the mobile apps. NoSQL database will come handy in those situations.

The details are provided as follows.

**Google Play Store App Details(**Android\_Data.segment**):** This collection in Mongo DB represents the Google Play Store App Details, after data cleaning and data wrangling.

**Apple Store App Details(**Apple\_Data.segment**):** This collection in Mongo DB represents the Apple Store App Details, after data cleaning and data wrangling.

Data Table Structure for Android and Apple Data Collection:

|  |  |
| --- | --- |
| Serial\_number | Unique Numbers |
| App\_Name | Application Name |
| App\_Size | Size of the Applications |
| currency | Currency Code |
| App\_Price | Price relative to Currency Code |
| Type | Type of App |
| Review\_count | Total Reviews for the App |
| User\_Rating | User Ratings for given App |
| Content\_Rating | Application Content Rating |
| Category | Category Application falls under |

**Apps Summary Details(**Apple\_Google\_Summary.segment**):** This collection in Mongo DB represents the aggregated summary level details of apps, grouped at the level of App Store/Category / Type / Content Rating. This provides valuable analytics like average user ratings, average price and Average users at an aggregate level.

Data Table Structure for Android and Google Summary Data collection:

|  |  |
| --- | --- |
| App\_Store | Application Provider |
| Category | Application Category |
| Type | Application Type |
| Content\_Rating | Application Content Rating |
| Avg\_App\_Price | Avg App Price |
| Avg\_review\_count | Avg Review Count |
| Avg\_User\_Rating | Avg User Ratings |

**How to use the data?**

The users are expected to query the **Apps Summary Details(**Apple\_Google\_Summary.segment**)** collection in Mongo DB by providing Category / Type / Content rating. They will be presented with potentially two documents (rows) representing Apple store and Google Play Store. Based on the details provided like Average App Price, Average review count and Average review count, the users can elect to explore apps details either from Apple store (**Apple Store App Details (**Apple\_Data.segment**)** or Google Play Store (**Google Play Store App Details (**Android\_Data.segment**)**. In our view, this is supposed to provide a one-stop shop for comparing apps between Apple Store and Google Play Store based on apps popularity and cost effectiveness, which should aid the users to make determination to buy/download appropriate apps.

**Data Sources**

We considered two datasets in .csv form from Kaggle on mobile apps from Google Play Store and Apple Store, each of which captured top 7000 to 9000 mobile apps, covering various categories. In the Kaggle site, these data datasets are infrequently updated, the latest update happened in March 2019. These data sets allowed us to build KPIs around average user ratings and average number of users (KPIs representing popularity of apps) and average price (KPI representing cost effective apps) aggregated at various categories/content rating level.

The structure of the input source datasets are as follows:



**Transformation Step**

In order to build our final collections/database, we had to perform the following data cleaning and data transformation activities:

**Data Cleaning steps performed [Apps\_Data\_Extract.ipynb]:**

1. Visual inspection of data
2. Verified the file contents with the head () method.
3. Dealt with missing data
   1. Dropped columns that have a high incidence of missing data
   2. Added in default value for the missing columns
   3. Invoked Lambda function to replace any values other than Alphanumeric and special characters with SPACES. Application\_Name field contained nonstandard characters as some of the apps were intended for non-English speaking audience. This step allowed further processing of file in Transform and Load processes.
4. Removed incomplete rows
   1. Deleted any rows that have a missing value. This will avoid errors in the Transformation and Load process.
5. Based on the utility of the final database, the structures of Apple app dataset and Google app dataset are aligned by bringing only selective columns into the dataframe.
6. Normalized data types
   1. Formatting columns to appropriate data type for father processing in Transformation and Load process.
   2. Type ()
7. Renamed columns
   1. Rename columns to more user-friendly names. DataFrame.rename(columns = {})
8. Saved results
9. After data clean-up, exported data back into CSV format for further processing in another program

**Data transformation steps performed [Apps\_Data\_transformation.ipynb]:**

1. The Apps categorization between Apple Store and Google Play Store was different. Realigned Google Play Store ‘Category’ column based on Apple Store Category by executing .replace command. See an example as follows:

*gps\_data['Category'].replace('BEAUTY', 'LIFESTYLE',inplace=True)*

1. The Apps ‘Content Rating’ between Apple Store and Google Play Store was different. Realigned Google Play Store ‘Content Rating’ column based on Apple Store Content Rating by executing .replace command. See an example as follows:

*gps\_data['Content\_Rating'].replace('Adults only 18+', '17+', inplace=True)*

1. In order to align with Google Play Store data, using a Lambda function, updated the ‘Type’ column in the Apple Store Apps data as ‘Free or Paid’, based on the App Price = 0 or not respectively.
2. Executed an aggregation function using NumPy on both Apple Store and Google Play Store Apps data to arrive at summary level detail for Average Price, Average User Rating and Average Rating Count. This aggregation was done at Category / Type / Content Rating. Here is the example:

*gps\_data2=pd.pivot\_table(gps\_data1, index=['App\_Store','Category','Type', 'Content\_Rating'],values=['App\_Price','User\_Rating','Review\_count'],aggfunc = np.average)*

1. After these transformations, exported the transformed Apps Detail files as well as Apps Summary files (for both Apple Store and Google Play Store) back into a .csv file format. These files were processed as input part of the ‘Load’ process.

**Project Plan**

****