High Level Design (HLD)

Analyzing Google Apps Store dataset

Revision Number: 1.0
Last date of revision: 10/7/22
Anwesha Das

Document Version Control

Date Issued	Version	Description	Author
10th July 2022	1.0	First Version of Complete HLD	Anwesha Das

Contents

Abstract

Google play store is engulfed with a few thousands of new applications regularly with a progressively huge number of designers working freely or on the other hand in a group to make them successful, with the enormous challenge from everywhere throughout the globe. Since most Play Store applications are free, the income model is very obscure and inaccessible regarding how the in-application buys, adverts and memberships add to the achievement of an application. In this way, an application's prosperity is normally dictated by the quantity of installation of the application and the client appraisals that it has gotten over its lifetime instead of the income created. This project aims to predict the ratings of Google Play Store apps using Python libraries. I have performed Data Analysis into the Google Play store application dataset that has been collected from Kaggle. I discovered the relationships among various attributes present in my dataset such as which application is free or paid, about the user reviews, rating of the application, etc.

1 Introduction

1.1 Why this High-Level Design Document?

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The HLD will:

- Present all of the design aspects and define them in detail
- Describe the user interface being implemented
- Describe the hardware and software interfaces
- Describe the performance requirements
- Include design features and the architecture of the project
- List and describe the non-functional attributes like:
    - Security
    - Reliability
    - Maintainability
    - Portability
    - Reusability
    - Application compatibility
    - Resource utilization
    - Serviceability

1.2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

2 General Description

2.1 Product Perspective & Problem Statement

.Android is expanding as an operating system. It has captured around 74% of the total market which is a true indicator of the huge amount of population using android. The goal is to help android developers to know what is the motivating factor for people to download an app. It will also help to find out the factors that affect someone's decision to download an app. I would like to analyze category, reviews, price, ratings and installs for this purpose and find out how they are interrelated. The aim of my analysis is to provide insights about android applications and their categories. I want to dive deep in data for the factors of influences on an application, to know why and how certain applications succeed others. Also, what is required for an application to be considered as successfully topping the charts

2.2 Tools used

Business Intelligence tools and libraries works such as Numpy, Pandas, Excel used to build the whole framework.

3 Design Details

3.1 Functional Architecture

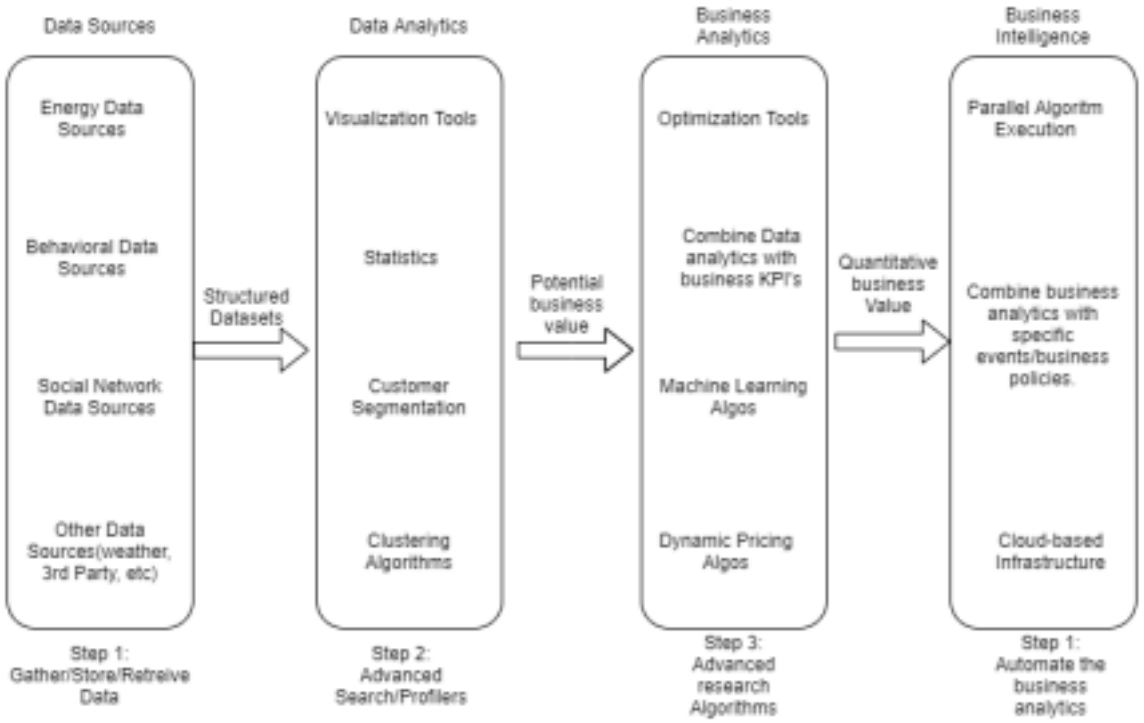
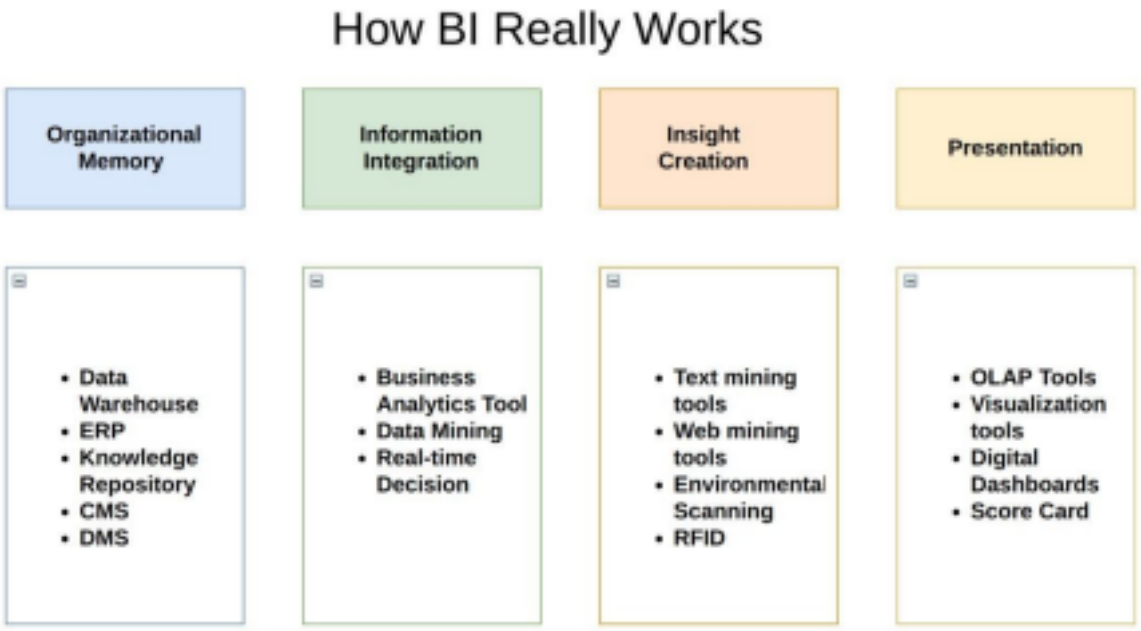


Figure 1: Functional Architecture of Business Intelligence



7

8 High Level Design (HLD)

3.2 Optimization

data strategy drives performance

- Minimize the number of fields
- Minimize the number of records
- Optimize extracts to speed up future queries by materializing calculations, removing columns and the use of accelerated views

Reduce the marks (data points) in your view

- Practice guided analytics. There's no need to fit everything you plan to show in a single view. Compile related views and connect them with action filters to travel from overview to highly-granular views at the speed of thought.
- Remove unneeded dimensions from the detail shelf.
- Explore. Try displaying your data in different types of views.

Limit your filters by number and type

- Reduce the number of filters in use. Excessive filters on a view will create a more complex query, which takes longer to return results. Double-check your filters and remove any that aren't necessary.

- Use an include filter. Exclude filters load the entire domain of a dimension, while include filters do not. An include filter runs much faster than an exclude filter, especially for dimensions with many members.

- Use a continuous date filter. Continuous date filters (relative and range-of-date filters) can take advantage of the indexing properties in your database and are faster than discrete date filters.

- Use Boolean or numeric filters. Computers process integers and Booleans (t/f) much faster than strings.

- Use parameters and action filters. These reduce the query load (and work across data sources).

Optimize and materialize your calculations

- Perform calculations in the database
- Reduce the number of nested calculations.

- Reduce the granularity of LOD or table calculations in the view. The more granular the calculation, the longer it takes.
    - LODs - Look at the number of unique dimension members in the calculation.
    - Table Calculations - the more marks in the view, the longer it will take to calculate.

- Where possible, use MIN or MAX instead of AVG. AVG requires more processing than MIN or MAX. Often rows will be duplicated and display the same result with MIN, MAX, or AVG.

- Make groups with calculations. Like include filters, calculated groups load only named members of the domain, whereas Tableau's group function loads the entire domain.

- Use Booleans or numeric calculations instead of string calculations. Computers can process integers and Booleans (t/f) much faster than strings.
  Boolean>Int>Float>Date>DateTime>String