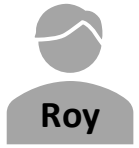
App-stracting Insights

EBA5006: Graduate Certificate in Big Data Analytics

Ang Mei Chi	(A0269733R)
Lew Kuan Teng Roy	(A0124354M)
Liu Wudi	(A0269809J)
Michael Wong Wait Kit	(A0269491N)
Ong Wee Yang	(A0017030A)

Presentation Outline



1.Introduction & Data Ingestion

2.Batch Data Processing & Storage

3.Machine Learning Model Integration (RCS)

4.Connectivity to Power BI & Dashboard Visualizations

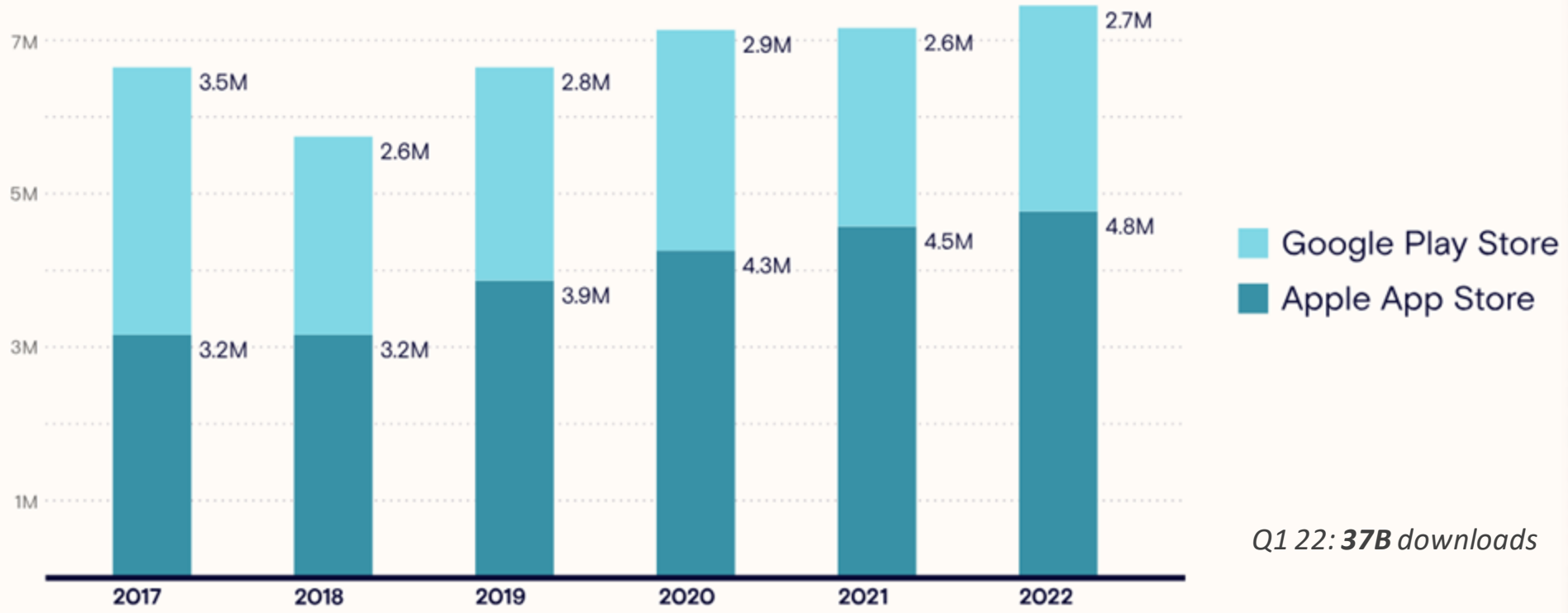
5.Monitoring, Logging, Conclusion/Retrospective

Introduction

Project Objectives

Number of available apps in Google Play Store and Apple App Store

Statista (2023)



Q1 22: ***37B*** *downloads*

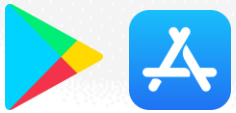
Project Objectives

Achievable and Scope

1. Conduct **trend analysis** across categories on the Apple App Store and Google Play Store **to understand app popularity and distribution**

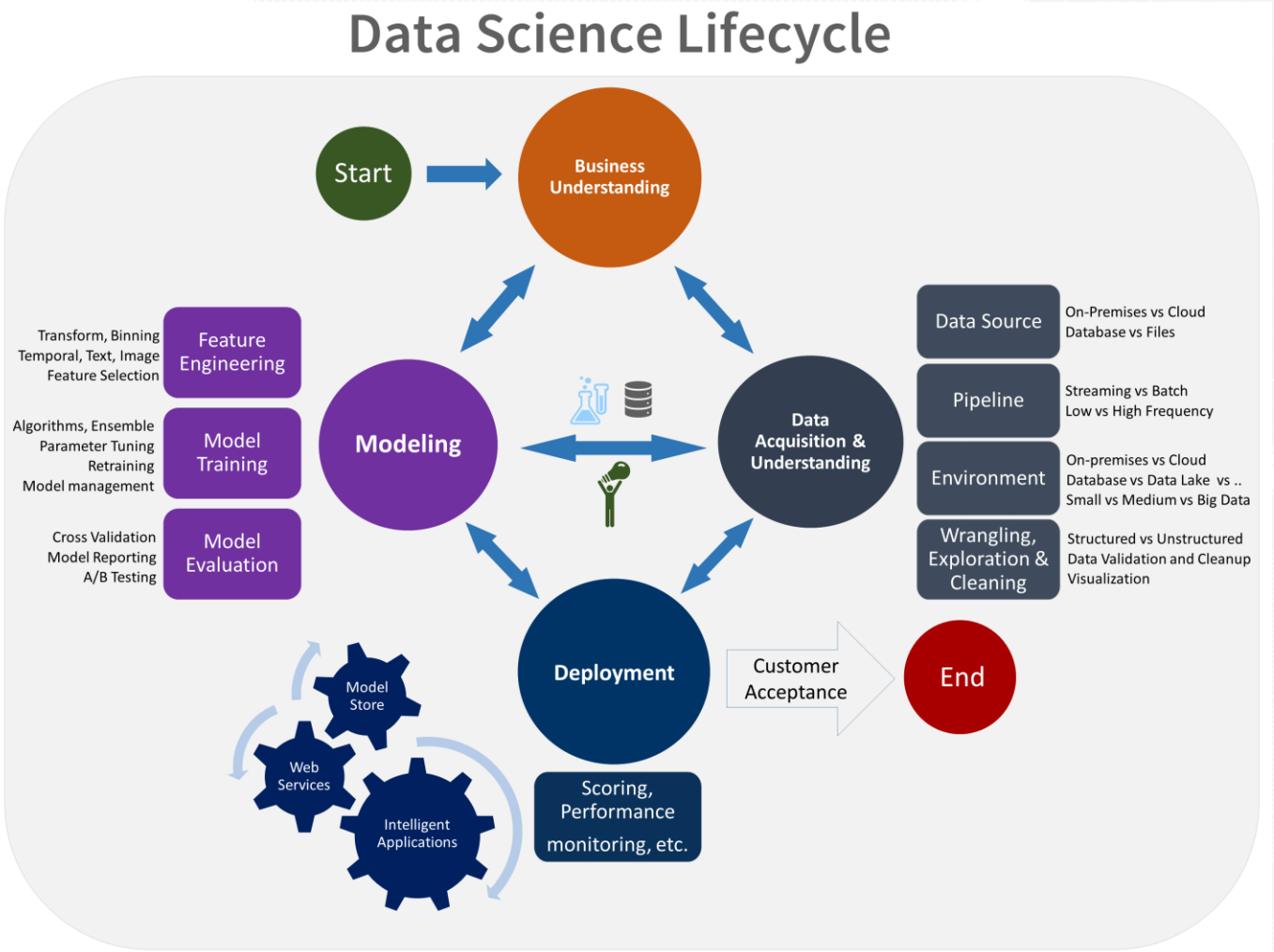
2. Perform **comparative market analysis** between Apple App Store and Google Play Store **to identify unique trends**

3. **Analyze key factors/features** that drive app installs and **explore predictive models** for apps installs



Methodology and Framework

CRISP-DM vs TDSP



Customers

Interested Parties. Stakeholders & Beneficiaries

App Developers

- Insights into app trends, user preferences and factors driving app installs
- App development strategies (user engagement)

App Development Companies

- Guide product roadmap
- Improve competitiveness of app in market

Digital Marketers

- Tailor advertising/promotion strategies
- Targeted audience, improve ROI

Data

Which data do you need?

App Feature Data

- Google Play Store
- Apple App Store

User Reviews Data

Skills

Which skills do you need for development?

- Web scraping
- API
- GCP
- GBQ
- Power BI
- PySpark
- GitHub

Output

Which key metric are you optimizing for?

- High recall and accuracy rate (> 70%)
- Revenue stream per app
- User satisfaction ratings

Value Proposition

What is the value added by your project?

- Empowers app developers with actionable insights for enhanced user engagement and optimized strategies
- Drives data-driven decisions in app development, enhancing performance and revenue
- Facilitates predictive modeling for app install forecasts, aiding strategic planning and investment

Integration

How will the project be integrated?

Stakeholders

Who are the key stakeholders?

- App Developers
- Marketing teams
- SME/Business Executive of App Company

Customers

Who are the end customers?

- App Developers

Cost

What costs will the project incur?

- Google Cloud Storage cost
- Data query and ETL pipeline cost
- Google Cloud Infrastructure cost

Revenue

How will the project generate revenue?

- Manhour savings from market app analysis
- Better market position (potential revenue and business growth)

Project

Stages and Data Architecture

Data Pipelines

Ingestion

Data Source

Data Extraction

Beautiful Soup

Storage

Main Storage

Google BigQuery

Temporary Storage

Google Cloud Storage

Code Versions

GitHub

Data Processing

Batch Processing

GitHub Action

EDA

Google BigQuery

Analytics Solution

Dashboard

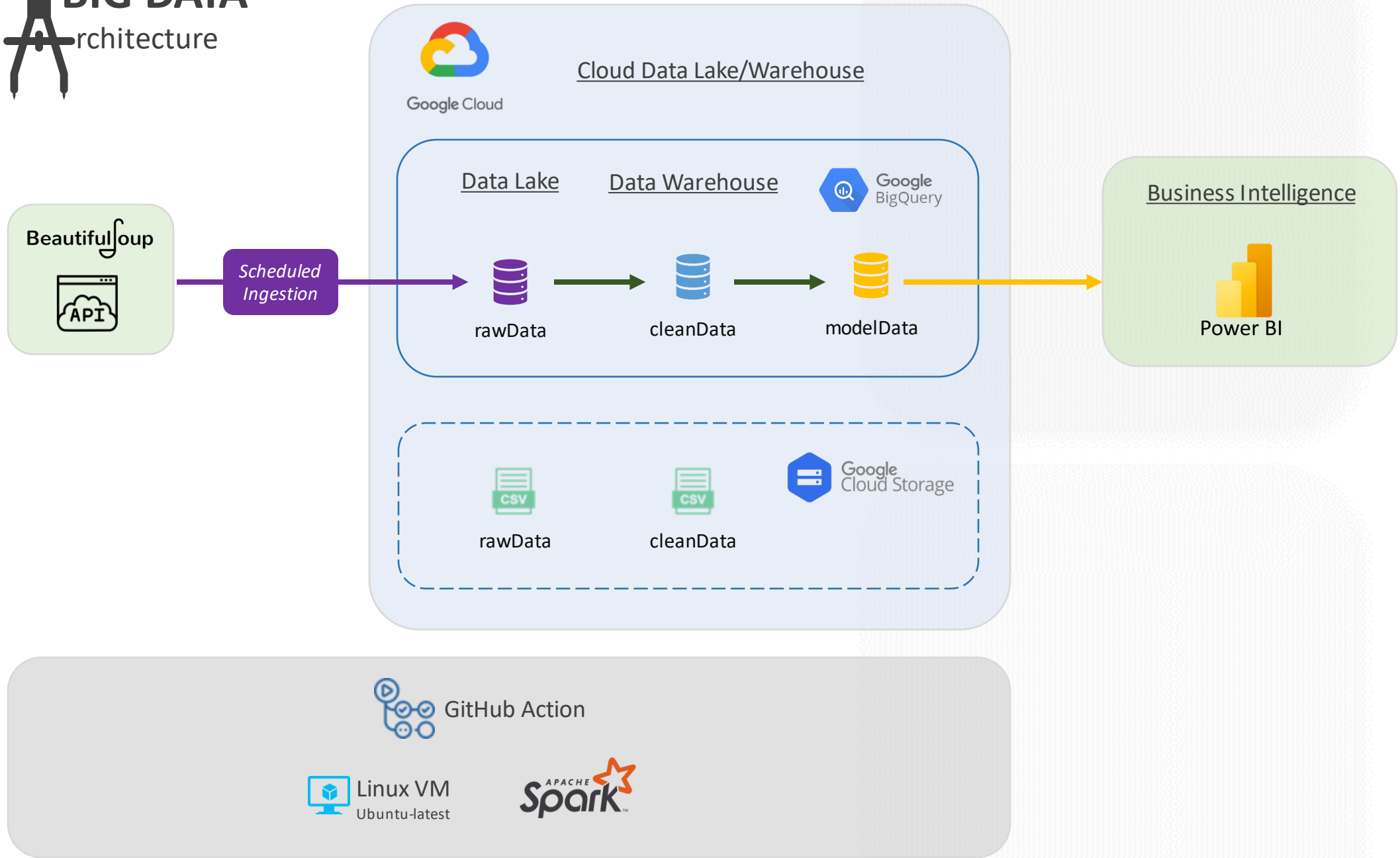
Power BI

Model Exploration

- Classification
- Recommenders

Spark MLlib

BIG DATA Architecture



Data Sources

Availability and Understanding

Web scraping

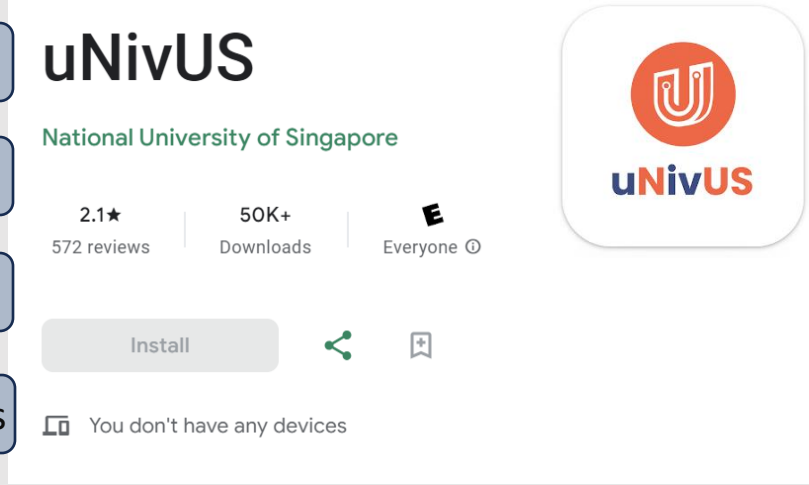
Extracting data from websites

App Name

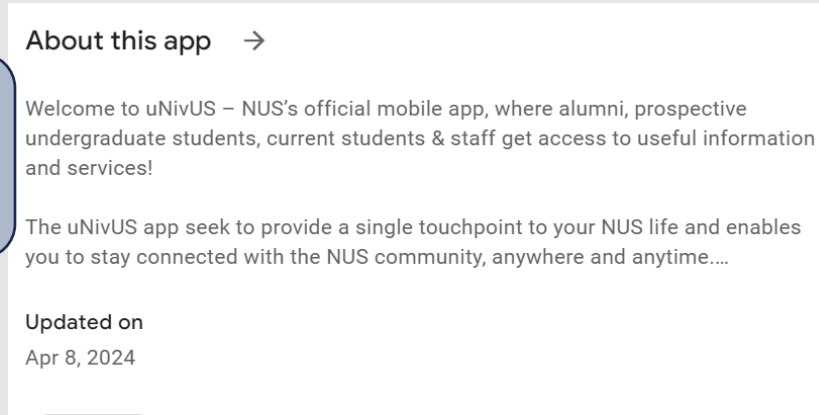
Genre

# of Reviews

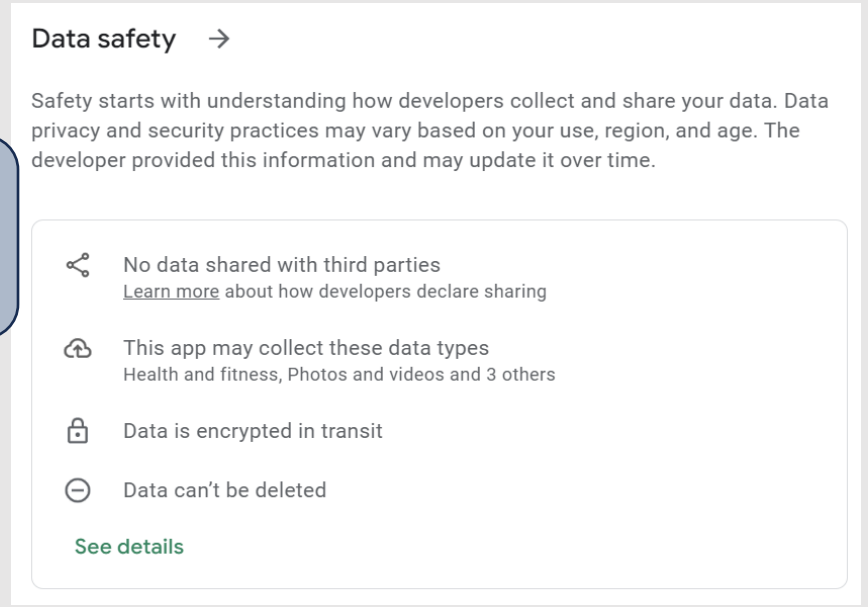
# of Downloads



App Description (updates)

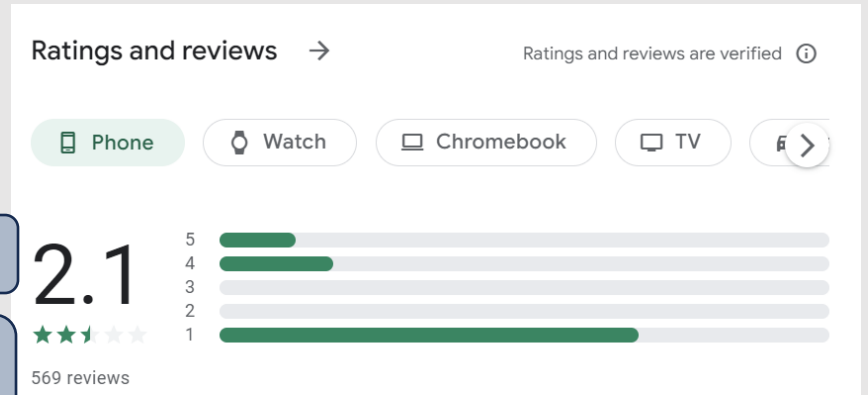


App Security Configuration

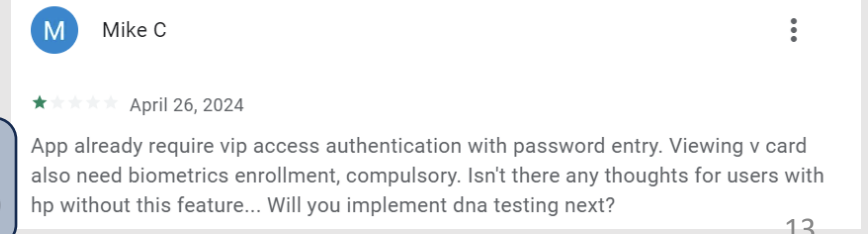


App Scores

Rating Distribution



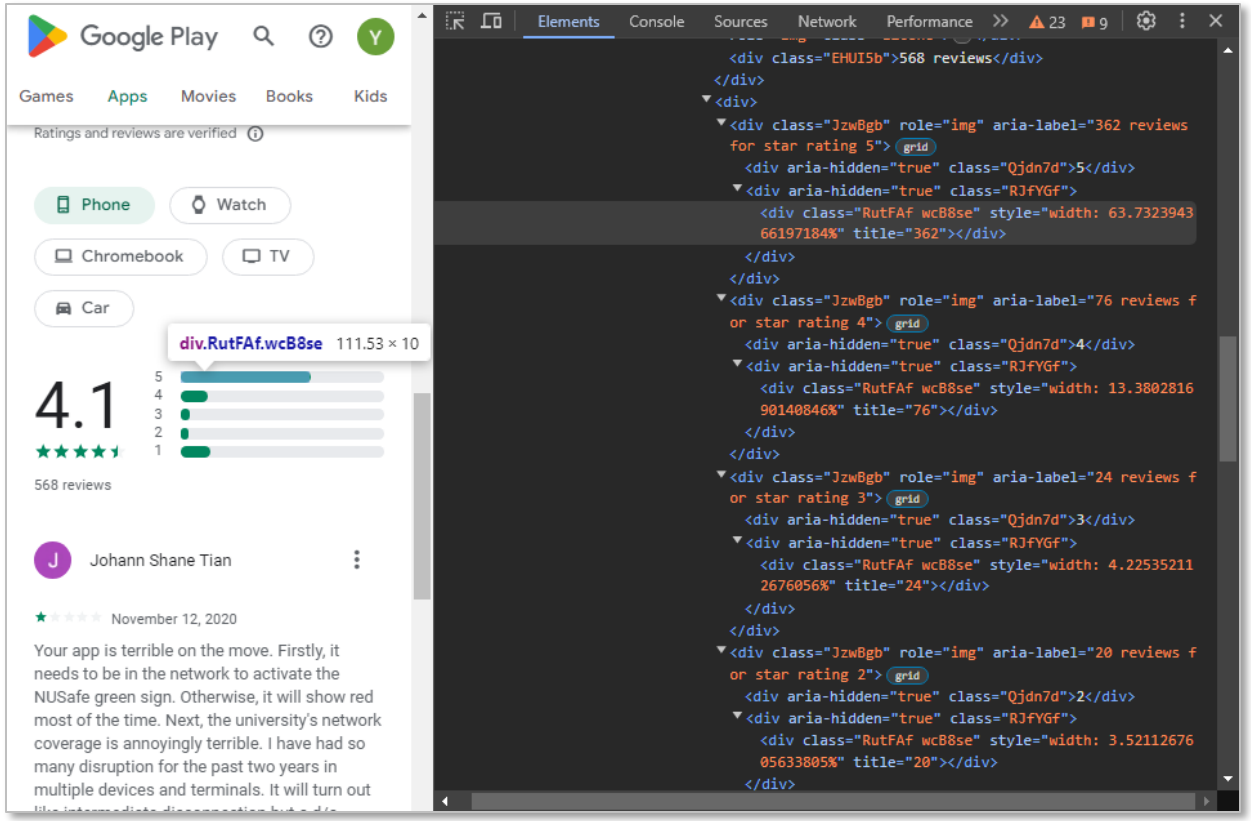
Reviews *(recent/rating)*



Web scraping & API

Extracting data from websites

Beautiful Soup



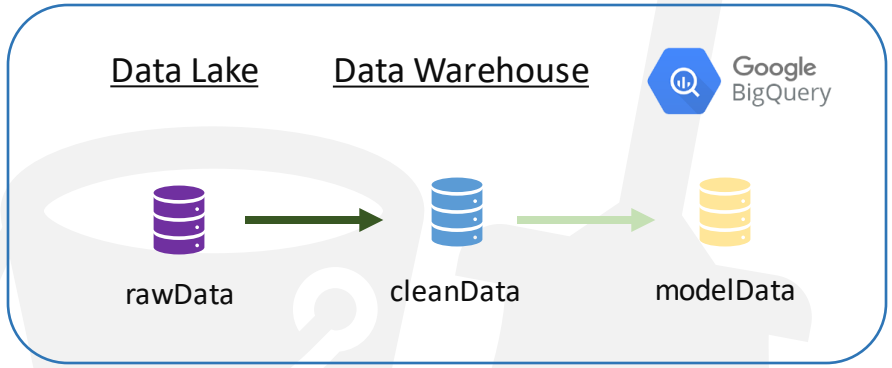

```python
def appWithThrottle(appId, lang = 'en', country = 'us', delay_between_requests = None):
    output = app(
                appId,
                lang=lang, # defaults to 'en'
                country=country # defaults to 'us'
                )
    if delay_between_requests != None:
        time.sleep(delay_between_requests)
    return output

def reviewsWithThrottle(appId, lang = 'en', country = 'us', count = 100, score = None, delay_between_requests = None):
    output = reviews(
                appId,
                lang=lang, # defaults to 'en'
                country=country, # defaults to 'us'
                sort=Sort.NEWEST, # defaults to Sort.NEWEST
                count=count, # defaults to 100
                filter_score_with=score # defaults to None(means all score)
                )
    if delay_between_requests != None:
        time.sleep(delay_between_requests)
    return output
```

Data Understanding

- No explicit explanation or metadata on the columns
- Systematic process to interpret and understand the data
  - ❑ Data type/structure analysis
  - ❑ Column name
  - ❑ Data inspection and profiling

S/N	Original Column	Data Type	Description
0	c0	integer	column number generated by spark
1	title	string	app title
2	description	string	app description
3	descriptionHTML	string	app description (copy)
4	summary	string	app summary
5	installs	string	app installs (categorical)
6	minInstalls	string	mininum installs following 'install' column
7	realInstalls	string	no. of real instals
8	score	string	score
9	ratings	string	no. of ratings, average is equivalent to score
10	reviews	string	no. of reviews
11	histogram	string	distribution of stared ratings
12	price	string	price of app
13	free	string	whether app is free for download
14	currency	string	currency of store (eg. USD)
15	sale	boolean	Sale (All false)
16	saleTime	string	time of sale (All None)
17	originalPrice	string	app original price (All None)
18	saleText	string	sale text (All None)
19	offersIAP	boolean	whether app offers In-App Purchases (IAP)
20	inAppProductPrice	string	IAP price range
21	developer	string	developer name
22	developerId	string	developer Id
23	developerAddress	string	developer Id address
24	genre	string	app genre
25	genreId	string	app genre id
26	categories	string	dictionary of app genre and genre id, may include genres besides in 'genre' column
27	contentRating	string	content rating eg. Everyone, Teen etc
28	contentRatingDescription	string	content rating description eg. Fantasy Violence, Simulated Gambling
29	adSupported	string	supports ads or not (True/False)
30	containsAds	boolean	contains ads or not (True/False)
31	released	string	App released date
32	lastUpdatedOn	string	App version last update release date
33	Updated	string	String of numbers, undecipherable
34	Version	string	App version
35	appId	string	App Id (PK)

Data Cleaning



- Row Reduction
  - ❑ Null value
  - ❑ <10 downloads
  - ❑ Reviews of prefiltered apps

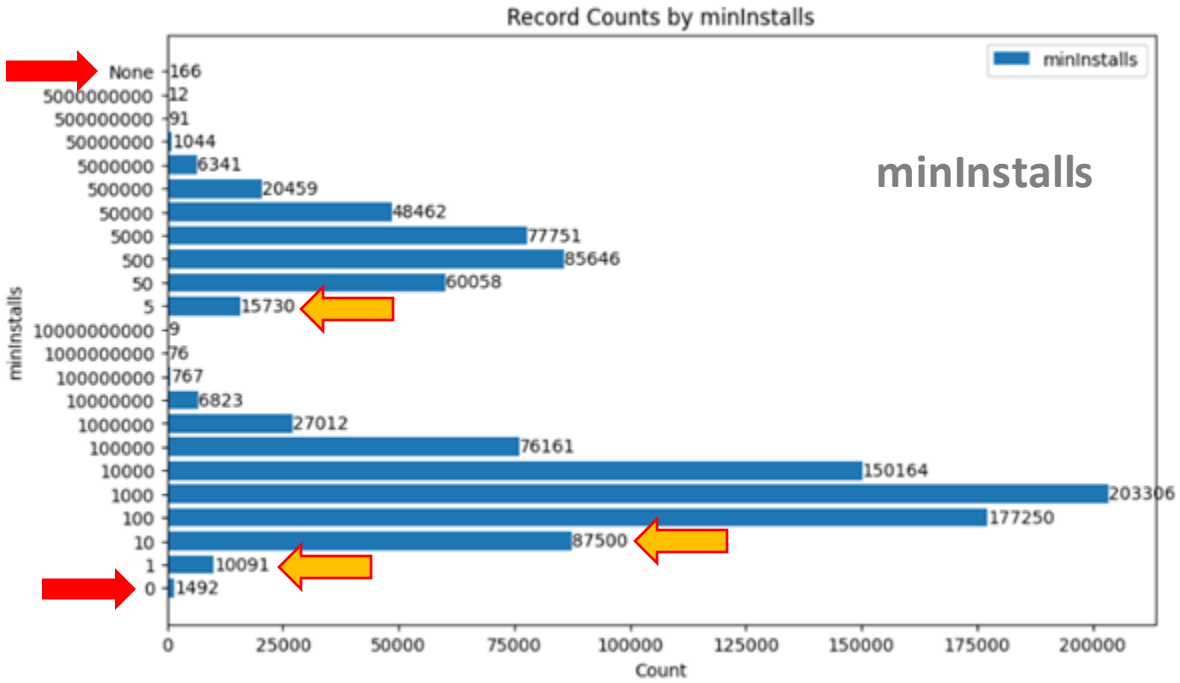
Eg. Google Apps

App Details	App Reviews
~1m rows x 36 columns	~1.5m rows x 12 columns
~540k rows x 31 columns	**~730k rows x 11 columns**

installs	minInstalls	realInstalls	score	ratings	reviews	histogram
500+	500	659	3.625	8	0	[1, 0, 3, 1, 3]
50,000+	50000	56925	3.4444444	172	5	[38, 9, 28, 28, 66]
10,000+	10000	10179	4.3767314	361	0	[39, 6, 14, 23, 279]
1,000,000+	1000000	1833331	4.353846	36224	535	[3328, 1386, 1941...
500+	500	849	0.0	0	0	[0, 0, 0, 0, 0]
100,000+	100000	302401	3.0	271	2	[108, 0, 54, 0, 108]
10,000+	10000	11306	4.402299	174	0	[11, 7, 4, 31, 121]
100+	100	245	nan	None	None	[0, 0, 0, 0, 0]
500,000+	500000	819193	3.94	15040	407	[1350, 1350, 1499...
1,000,000+	1000000	3100152	4.034483	20459	19	[3527, 352, 2116,...

nan – "not a number", missing or undefined numeric values

None – Null or absence of value



Detect Language using langdetect

Python library used for language detection and identification

- Total Review Count: **727,601**
- Distinct language identified: **56** *(1 unknown)*
- Top 10 languages:

S/N	language cod	Review Count	languages	% of tota
1	en	470,084	en: English	64.61%
2	so	40,557	so: Somali	5.57%
3	af	24,998	af: Afrikaans	3.44%
4	ro	16,671	ro: Romanian	2.29%
5	id	13,780	id: Indonesian	1.89%
6	unknown	11,185	unknown: Unknown	1.54%
7	de	10,760	de: German	1.48%
8	it	10,708	it: Italian	1.47%
9	ar	9,425	ar: Arabic	1.30%
10	fr	9,096	fr: French	1.25%

Other languages identified

11	ca	8,942	ca: Catalan	1.23%
12	es	7,746	es: Spanish	1.06%
13	tl	7,547	tl: Tagalog	1.04%
14	pl	7,401	pl: Polish	1.02%
15	no	7,208	no: Norwegian	0.99%
16	nl	5,571	nl: Dutch	0.77%
17	sl	5,151	sl: Slovenian	0.71%
18	sw	5,070	sw: Swahili	0.70%
19	et	4,648	et: Estonian	0.64%
20	pt	4,439	pt: Portuguese	0.61%
21	cy	3,828	cy: Welsh	0.53%
22	fa	3,519	fa: Persian	0.48%
23	da	3,378	da: Danish	0.46%
24	ru	3,110	ru: Russian	0.43%
25	tr	2,958	tr: Turkish	0.41%
26	hr	2,852	hr: Croatian	0.39%
27	sq	2,395	sq: Albanian	0.33%
28	sv	2,353	sv: Swedish	0.32%
29	fi	2,174	fi: Finnish	0.30%
30	ko	2,148	ko: Korean	0.30%
31	vi	2,129	vi: Vietnamese	0.29%
32	bn	1,989	bn: Bengali	0.27%
33	cs	1,640	cs: Czech	0.23%
34	sk	1,507	sk: Slovak	0.21%
35	ur	1,183	ur: Urdu	0.16%
36	hu	1,137	hu: Hungarian	0.16%
37	hi	1,136	hi: Hindi	0.16%
38	th	831	th: Thai	0.11%
39	he	797	he: Hebrew	0.11%
40	lt	764	lt: Lithuanian	0.11%
41	zh-cn	572	zh-cn: Chinese (Simplified)	0.08%
42	ja	544	ja: Japanese	0.07%
43	lv	516	lv: Latvian	0.07%
44	mr	486	mr: Marathi	0.07%
45	ta	383	ta: Tamil	0.05%
46	uk	348	uk: Ukrainian	0.05%
47	bg	317	bg: Bulgarian	0.04%
48	ne	288	ne: Nepali	0.04%
49	el	253	el: Greek	0.03%
50	gu	205	gu: Gujarati	0.03%
51	ml	196	ml: Malayalam	0.03%
52	zh-tw	171	zh-tw: Chinese (Traditional)	0.02%
53	mk	170	mk: Macedonian	0.02%
54	te	157	te: Telugu	0.02%
55	kn	116	kn: Kannada	0.02%
56	pa	64	pa: Punjabi	0.01%

Detect Language using [langdetect]

Python library used for language detection and identification

```
4  SELECT content, score, language FROM `big-data-analytics-415801.cleanData.cleanGoogleReview` where language in ('en')
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	content	score	language
1	Why this application didn't work to me 😡 😠	5	en
2	Yeah, this is def broken.	1	en
3	Whatsapp download	4	en
4	It works great	4	en
5	Its ok n just for my use	5	en
6	Great thanks	5	en
7	Have problem with my playstore	1	en
8	I have not really tested it yet but i think is nice	4	en
9	It is good to use, it has a beautiful color and i have loved it	4	en
10	Very floating	5	en
11	It's very perfect	5	en
12	I have'not use so far but ut good I like it	2	en
13	I really like it and pink is my favorite color and itmatches so well	5	en
14	Works gr8 on my old Note 8. Like the video clips tho I wish you could either adjust the speed or slow them down a notch. Also, some of the videos... muscles for instance... in particular the upper extremity, I believe flexion.... there were maybe 3 different muscles using the same clip to explain th...	5	en
15	One of the best app. Muscle action describe nicely. Add more actions of body part.	5	en
16	I love this app and would recommend it for anyone, whether you are studying, just interested or a natural therapist it's full of useful info	5	en
17	Great app for refresher of anatomy and as a visual reference for students and patients.	5	en

Detect Language using langdetect

Python library used for language detection and identification

- Total Review Count: **727,601**
- Distinct language identified: **56** *(1 unknown)*
- Top 10 languages:

S/N	language cod	Review Count	languages	% of tota
1	en	470,084	en: English	64.61%
2	so	40,557	so: Somali	5.57%
3	af	24,998	af: Afrikaans	3.44%
4	ro	16,671	ro: Romanian	2.29%
5	id	13,780	id: Indonesian	1.89%
6	unknown	11,185	unknown: Unknown	1.54%
7	de	10,760	de: German	1.48%
8	it	10,708	it: Italian	1.47%
9	ar	9,425	ar: Arabic	1.30%
10	fr	9,096	fr: French	1.25%

```
4  SELECT content, score, language FROM `big-data-analytics-415801.cleanData.
5  LIMIT 50
```

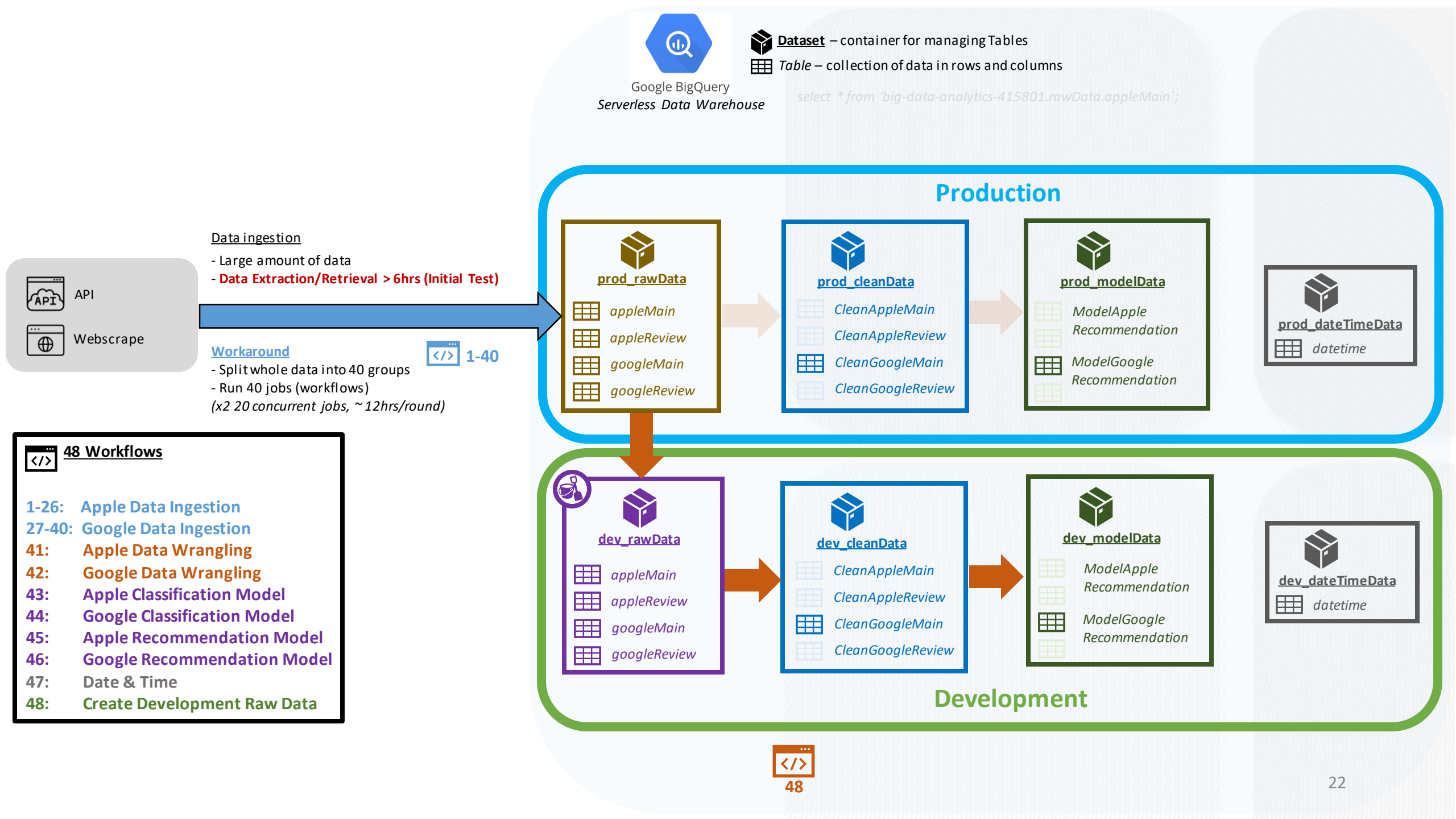
Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS

Row	content	score	language
14	😍😍	5	unknown
15	❤️❤️❤️❤️	5	unknown
16	👌🏻👌🏻👌🏻👌🏻👌🏻	5	unknown
17	👌🏻👌🏻	1	unknown
18	🚫🚫🚫	1	unknown
19	Abdul RAHim kham 😁🥶 😎🤓😎🤓🥹🥹🥹🥹 😍💔💔💔💔💔💔🖤🖤 🖤🖤🖤💪💪💪💪💪🦙 🦙🦙🦙🦙🦙🦙🦙	5	unknown
20	💜💜	5	unknown
21	👌🏻👌🏻👌🏻	5	unknown
22	🔥🔥🔥🔥🔥💣💣💣💣💣...	5	unknown
23	👎	1	unknown
24	☺	5	unknown
25	:)	5	unknown
26	🤑	5	unknown
27	👍🏻👍🏻	5	unknown
28	😀	5	unknown
29	🖖🖖🖖🖖🖖🖖💯💯💯👍...	5	unknown
30	👌🏻👌🏻👌🏻👌🏻👌🏻👌🏻	5	unknown
31	👍	5	unknown
32	សៀវភៅល្អ ខ្ញុំចូលចិត្តអានសៀវភៅ...	4	unknown
33	👍	5	unknown

Data Batch Processing and Storage

GitHub Scheduler, Workflow and everything in between


Google BigQuery
Serverless Data Warehouse
Dataset – container for managing Tables
Table – collection of data in rows and columns
select * from 'big-data-analytics-415801.rawData.appleMain';
Production
prod_rawData
appleMain
appleReview
googleMain
googleReview
prod_cleanData
CleanAppleMain
CleanAppleReview
CleanGoogleMain
CleanGoogleReview
prod_modelData
ModelApple Recommendation
ModelGoogle Recommendation
prod_dateTimeData
datetime
Data ingestion
- Large amount of data
- Data Extraction/Retrieval > 6hrs (Initial Test)
API
Webscrape
Workaround
- Split whole data into 40 groups
- Run 40 jobs (workflows)
(x2 20 concurrent jobs, ~12hrs/round)
1-40
48 Workflows
1-26: Apple Data Ingestion
27-40: Google Data Ingestion
41: Apple Data Wrangling
42: Google Data Wrangling
43: Apple Classification Model
44: Google Classification Model
45: Apple Recommendation Model
46: Google Recommendation Model
47: Date & Time
48: Create Development Raw Data
dev_rawData
appleMain
appleReview
googleMain
googleReview
dev_cleanData
CleanAppleMain
CleanAppleReview
CleanGoogleMain
CleanGoogleReview
dev_modelData
ModelApple Recommendation
ModelGoogle Recommendation
dev_dateTimeData
datetime
Development
48

GitHub repository structure

Runners ('virtual machines')

Top layer

actions1-48.yml files (workflows)

Middle layer

main.py

Bottom layer

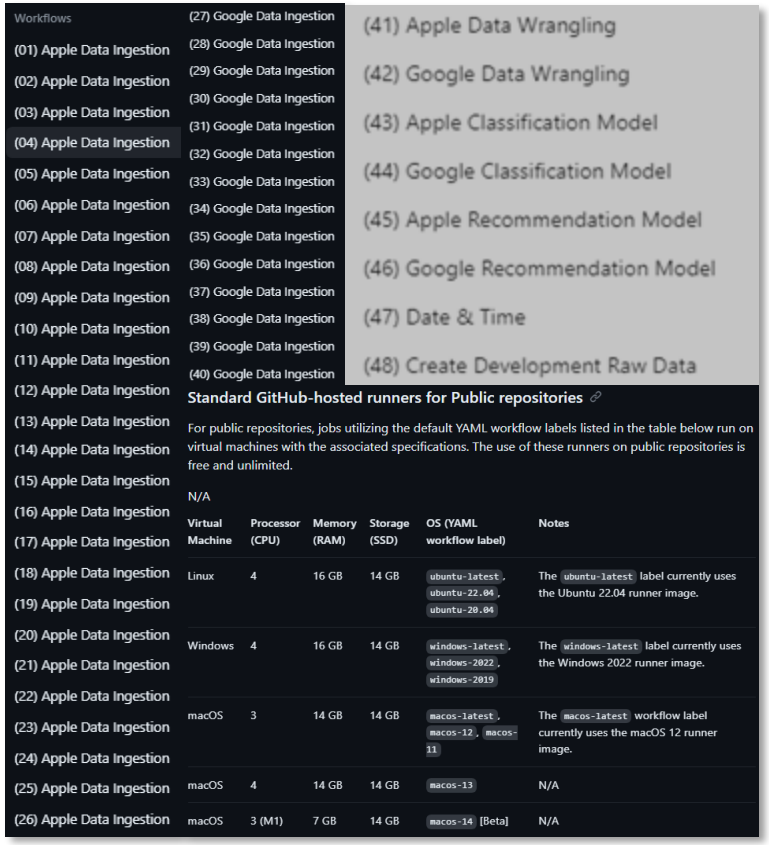
dataIngestionApple.py, dataIngestionGoogle.py,
dataWrangling.py, models.py

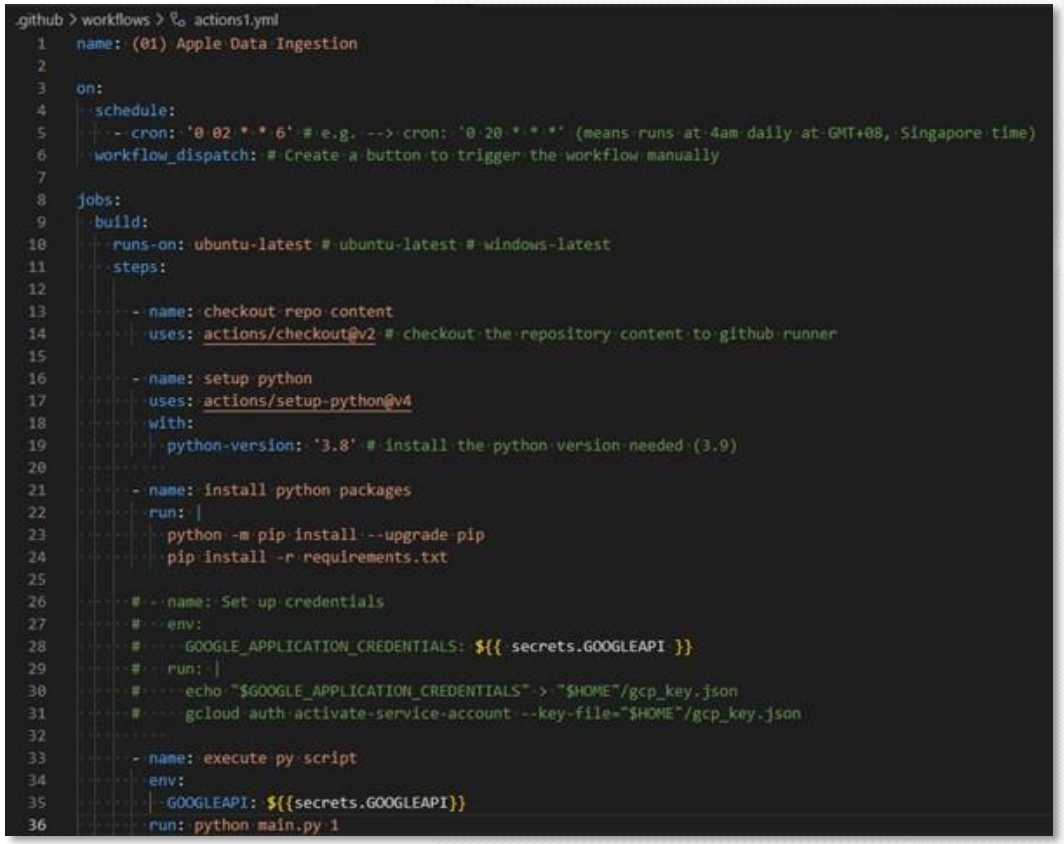
Supporting files

Common.py, requirements.txt, etc

Yml files

- Workflow files that instructs GitHub Actions to carry out a set of actions
- **40** for data ingestion, **2** for data wrangling, **4** for ML modelling, **1** to create training dataset, **1** to create date & time table
- A Runner will be assigned
  - ubuntu-latest (LINUX machine, quad-core processor)





Top layer

Middle layer

Bottom layer

Main.py

- To create a dictionary to store respective downstream functions
  - Data ingestions, data wrangling, ML modelling
- Read .yml variables under ‘execute Python script’ section to run the functions

(1. Define Ingestion function)

```
### Data Ingestion ###

currentAppleSubDf = 1
currentGoogleSubDf = 1

def dataIngestionFunction(appleSlices, currentAppleSubDf, googleSlices, currentGoogleSubDf, deleteRows = False, project_id = project_id, client = client):
    if deleteRows == True:
        deleteRowsAppleGoogle(project_id = project_id, client = client)
    dataIngestionApple(noOfSlices = appleSlices, subDf = currentAppleSubDf, client = client, project_id = project_id)
    dataIngestionGoogle(noOfSlices = googleSlices, subDf = currentGoogleSubDf, client = client, project_id = project_id)

def create_data_ingestion_function(apple_slices, current_apple_sub_df, google_slices, current_google_sub_df, delete_rows = False, project_id = project_id, client = client):
    return lambda: dataIngestionFunction(apple_slices, current_apple_sub_df, google_slices, current_google_sub_df, delete_rows, project_id, client)

for action_no in range(1, appleMaxSlice + googleMaxSlice + 1): # full range of YAML action no.s for data ingestion
    if action_no == 1:
        main_dict[action_no] = create_data_ingestion_function(appleMaxSlice, currentAppleSubDf, 0, 1, delete_rows = True)
        currentAppleSubDf += 1
    elif action_no in range(2, appleMaxSlice + 1): # range of YAML action no.s for Apple ONLY
        main_dict[action_no] = create_data_ingestion_function(appleMaxSlice, currentAppleSubDf, 0, 1)
        currentAppleSubDf += 1
    elif action_no in range(appleMaxSlice + 1, appleMaxSlice + googleMaxSlice + 1): # range of YAML action no.s for Google ONLY
        main_dict[action_no] = create_data_ingestion_function(0, 1, googleMaxSlice, currentGoogleSubDf)
        currentGoogleSubDf += 1
```

(2. Define Wrangling/ ML/ Date Time/ create training data functions)

```
### wrangling, ML, DateTime, devRawData ###

def wranglingMLDateTime_devRawData(devRawData = False, appleWrangling = False, googleWrangling = False,
                                   appleClassModel = False, googleClassModel = False,
                                   appleRecModel = False, googleRecModel = False, dateAndTime = False):

    # Start Spark session
    # spark = SparkSession.builder.master("local").appName("appStoreAnalytics").config('spark.ui.port', '4050').getOrCreate()
    spark = SparkSession.builder.appName("appStoreAnalytics").config("spark.executor.memory", "8g").getOrCreate()

    if devRawData == False:
        if appleWrangling == True:
            appleDataWrangling(spark, project_id, client)
            print('Apple Data wrangling step completed. Clean tables updated.')
        if googleWrangling == True:
            googleDataWrangling(spark, project_id, client)
            print('Google Data wrangling step completed. Clean tables updated.')
        if appleClassModel == True:
            appleClassificationModel(spark, project_id, client)
            print('Apple Classification Model step completed. Apple Classification Model tables updated.')
        if googleClassModel == True:
            googleClassificationModel(spark, project_id, client)
            print('Google Classification Model step completed. Google Classification Model tables updated.')
        if appleRecModel == True:
            appleRecommendationModel(spark, project_id, client)
            print('Apple Recommender Model step completed. Apple Recommender Model tables updated.')
        if googleRecModel == True:
            googleRecommendationModel(spark, project_id, client)
            print('Google Recommender Model step completed. Google Recommender Model tables updated.')
        if dateAndTime == True:
            dateTime(spark, project_id, client)
            print('Date & time updated.')
    else:
        AppleScraped_table_name = appleScraped_table_name
        AppleReview_table_name = appleReview_table_name
        GoogleScraped_table_name = googleScraped_table_name
        GoogleReview_table_name = googleReview_table_name
        table_names = [AppleScraped_table_name, AppleReview_table_name, GoogleScraped_table_name, GoogleReview_table_name]
        for table_name in table_names:
            devRawData_db_path = f"{project_id}.{dev_rawDataset}.{table_name}"
            sparkDf = read_gbq(spark, rawDataset, table_name)
            client.create_table(bigquery.Table(devRawData_db_path), exists_ok = True)
            to_gbq(sparkDf, dev_rawDataset, table_name, allDataTypes = False)
        print('Development raw data transfer step completed. Development raw data tables updated.')

    # Stop Spark session
    spark.stop()
```

```
def create_wranglingMLDateTime_devRawData(devRawData = False, appleWrangling = False, googleWrangling = False,
                                          appleClassModel = False, googleClassModel = False,
                                          appleRecModel = False, googleRecModel = False, dateAndTime = False):
    return lambda: wranglingMLDateTime_devRawData(devRawData, appleWrangling, googleWrangling,
                                                  appleClassModel, googleClassModel,
                                                  appleRecModel, googleRecModel, dateAndTime)

main_dict[appleWrangling_actionNo] = create_wranglingMLDateTime_devRawData(appleWrangling = True)
main_dict[googleWrangling_actionNo] = create_wranglingMLDateTime_devRawData(googleWrangling = True)
main_dict[appleClassificationModel_actionNo] = create_wranglingMLDateTime_devRawData(appleClassModel = True)
main_dict[googleClassificationModel_actionNo] = create_wranglingMLDateTime_devRawData(googleClassModel = True)
main_dict[appleRecommendationModel_actionNo] = create_wranglingMLDateTime_devRawData(appleRecModel = True)
main_dict[googleRecommendationModel_actionNo] = create_wranglingMLDateTime_devRawData(googleRecModel = True)
main_dict[dateTime_actionNo] = create_wranglingMLDateTime_devRawData(dateAndTime = True)
main_dict[dev_rawDataset_actionNo] = create_wranglingMLDateTime_devRawData(devRawData = True)
```

(3. Run functions)

```
### Run above functions conditionally depending on which YAML file is calling it ###
for action_inputNo in range(1, dev_rawDataset_actionNo+1):
    if sys.argv[1] == str(action_inputNo):
        main_dict[action_inputNo]()
```

Top layer

Middle layer

Bottom layer

Data Ingestion

- Done via reading a list of Apple & Google application IDs that is found on GitHub

[Limitations]

- API & web scrapping method runs too slowly
- GitHub Actions (free)
    - **Maximum of 6 hours** runtime per Runner's workflow
    - **Maximum of 20 Runners** deployable simultaneously

[Solutions]

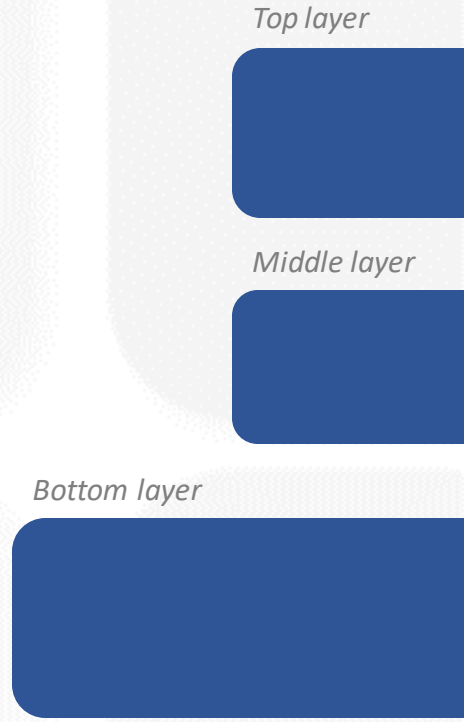
- Split into **40** separate workflows (**26** Apple & **14** Google), running 20 workflows simultaneously 2 times, back-to-back
    - API/ web scrapping function will iterate through sub list and append outputs to a data frame
    - Once completed, each of the runner will append the results to the respective GBQ tables
    - Split function created in Common.py
- Enabled multi-threading to run all **4** worker threads per runner in parallel, **increasing processing speed by 4x**

(Split function)

```python
def split_df(df, noOfSlices = 1, subDf = 1):

    if noOfSlices != 0:
        # Assuming df is your DataFrame
        num_parts = noOfSlices

        # Calculate the number of rows in each part
        num_rows = len(df)
        rows_per_part = num_rows // num_parts

        # Initialize a list to store the sub DataFrames
        sub_dfs = []

        # Split the DataFrame into parts
        for i in range(num_parts):
            start_idx = i * rows_per_part
            end_idx = start_idx + rows_per_part
            if i == num_parts - 1:  # For the last part, include the remaining rows
                end_idx = num_rows
            sub_df = df.iloc[start_idx:end_idx]
            sub_dfs.append(sub_df)

        # Select sub DataFrame
        indexOfSubDf = subDf - 1
        small_df = sub_dfs[indexOfSubDf]

    else:
        small_df = pd.DataFrame(columns = df.columns)

    return small_df
```

(Multi-threading)

```python
appsChecked = 0
# Use ThreadPoolExecutor for parallel processing
with ThreadPoolExecutor(max_workers=os.cpu_count()) as executor:
    print(f"No. of worker threads deployed: {os.cpu_count()}")

    for appId in apple.iloc[:, 2]:

        # Record the end time
        overall_end_time = time.time()
        # Calculate and print the overall elapsed time in seconds
        overall_elapsed_time = overall_end_time - overall_start_time

        if overall_elapsed_time < 21000: # 5 hour 50 mins

            appsChecked += 1

            try:
                # Record the start time
                start_time = time.time()

                executor.submit(process_app, appId)

                # Record the end time
                end_time = time.time()
                # Calculate and print the elapsed time in seconds
                elapsed_time = end_time - start_time

                # if appId in apple_main['appId'].to_list():
                print(f"Apple: {appId} -> Successfully saved in {elapsed_time} seconds. Total -> \
{appsChecked}/{len(apple)} ({round(appsChecked/len(apple)*100,1)}%) completed.")

            except Exception as e:
                print(f"Apple (Error): {appId} -> {e}")

        else:
            print("Exiting data ingestion prematurely ..")
            break

apple_reviews = pd.concat([apple_reviews_devResponse, apple_reviews_no_devResponse], ignore_index=True)
```



Data Wrangling

- Data wrangling script reads the raw data and process it to clean format
- Fully done via the Spark framework
- Cleaned datasets pushed to Google Big Query

```python
def appleDataWrangling(spark, project_id, client, local = False, sparkDf = None, sparkRvDf = None):

    # Apple functions
    def remove_string_from_column_apple(column, string):
        return regexp_replace(col(column), string, "")

    def helper_apple(df, string, column):
        return df.withColumn(column, remove_string_from_column_apple(column, string))

    def remove_strings_from_df_apple(df, column, strings):
        return reduce(lambda acc, string: helper_apple(acc, string, column), strings, df)

    def remove_strings_from_columns_apple(df, strings_to_remove):
        return reduce(lambda acc, column: remove_strings_from_df_apple(acc, column, strings_to_remove[column]), strings_to_remove, df)

    def remove_strings_apple(content, strings_to_remove):
        return reduce(lambda acc, s: acc.replace(s, ""), strings_to_remove, content)

    def detect_language_langdetect_apple(text):
        try:
            return detect(text)
        except:
            return "unknown"

    # Apple Data Wrangling
    def clean_data_appleMain(df):
        # Drop specific columns
        columns_to_drop = ['operatingSystem', 'authorurl']
        df = df.drop(*columns_to_drop)

        # Remove specified strings from specified columns
        strings_to_remove = {
        'description': ['<b>']
        }

        df = remove_strings_from_columns_apple(df, strings_to_remove)

        # Transform star_ratings column
        # Remove characters not needed [, ], (, ), ', %
        df = df.withColumn('cleaned_star_ratings', regexp_replace('star_ratings', r"[(|)|'|\]|\[|%]", ""))
```

(Apple data wrangling snippets)

```python
# Google Data Wrangling
def googleDataWrangling(spark, project_id, client):

    # googleMain

    cleanGoogleScraped_db_path = f"{project_id}.{cleanDataset}.{cleanGoogleMainScraped_table_name}" # Schema + Table

    sparkDf = read_gbq(spark, rawDataset, googleMain)
    # print(sparkDf.show())
    # print(sparkDf.count())

    # Code section for cleaning googleMain data
    def clean_data_googleMain(df):

        # Drop specific columns
        columns_to_drop = ['descriptionHTML', 'sale', 'saleTime', 'originalPrice', 'saleText', 'developerId', 'developerAddress', 'containsAds', 'updated']
        df = df.drop(*columns_to_drop)

        # Remove specified strings from specified columns
        strings_to_remove = {
        'description': ['<b>']
        }
        # for column, strings in strings_to_remove.items():
        #     for string in strings:
        #         df = df.withColumn(column, regexp_replace(col(column), string, ""))

        def remove_strings_from_columns(df, strings_to_remove):
            def remove_string_from_column(column, string):
                return regexp_replace(col(column), string, "")

            # def remove_strings_from_df(df, column, strings):
            #     for string in strings:
            #         df = df.withColumn(column, remove_string_from_column(column, string))
            #     return df

            def remove_strings_from_df(df, column, strings):
                def helper(df, string):
                    return df.withColumn(column, remove_string_from_column(column, string))

                return reduce(lambda acc, string: helper(acc, string), strings, df)


            return reduce(lambda acc, column: remove_strings_from_df(acc, column, strings_to_remove[column]), strings_to_remove, df)

        df = remove_strings_from_columns(df, strings_to_remove)
```

(Google data wrangling snippets)

Top layer

Middle layer

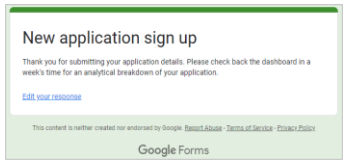
Bottom layer

ML modelling

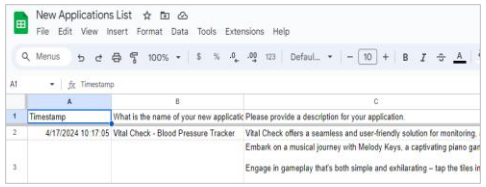
- Training
  - Models were initially trained using development datasets on Google Colabatory

- Deployment
  - Models subsequently deployed to Production
  - New data (model features) are introduced when developers enters their new application information here:
    - https://docs.google.com/forms/d/e/1FAIpQLSdcbV56ISkxl8vF-XLXpqkmeDCAAbBygcL5pYUKNY7Vd-0jcg/viewform
  - Pipeline:

1. Google Forms



2. Google Docs



3. Data frame in model.py

```
# Define the scope of the Google Sheets API
scope = ['https://spreadsheets.google.com/feeds', 'https://www.googleapis.com/auth/drive']

# Authenticate with Google Sheets API using the service account key
gspread_client = pygsheets.authorize(service_file = googleAPI_json_path)

# Read Google Sheet
spreadsheet = gspread_client.open_by_url(googleSheetURL)
worksheet = spreadsheet.sheet1
data = worksheet.get_all_records()
newApplications_df = spark.createDataFrame(data)
```

4. Model function codes in model.py

```
test_doc = word_tokenize(newData.lower())
test_vec = model.infer_vector(test_doc)
results = model.docvecs.most_similar(positive=[test_vec], topn=10)
```

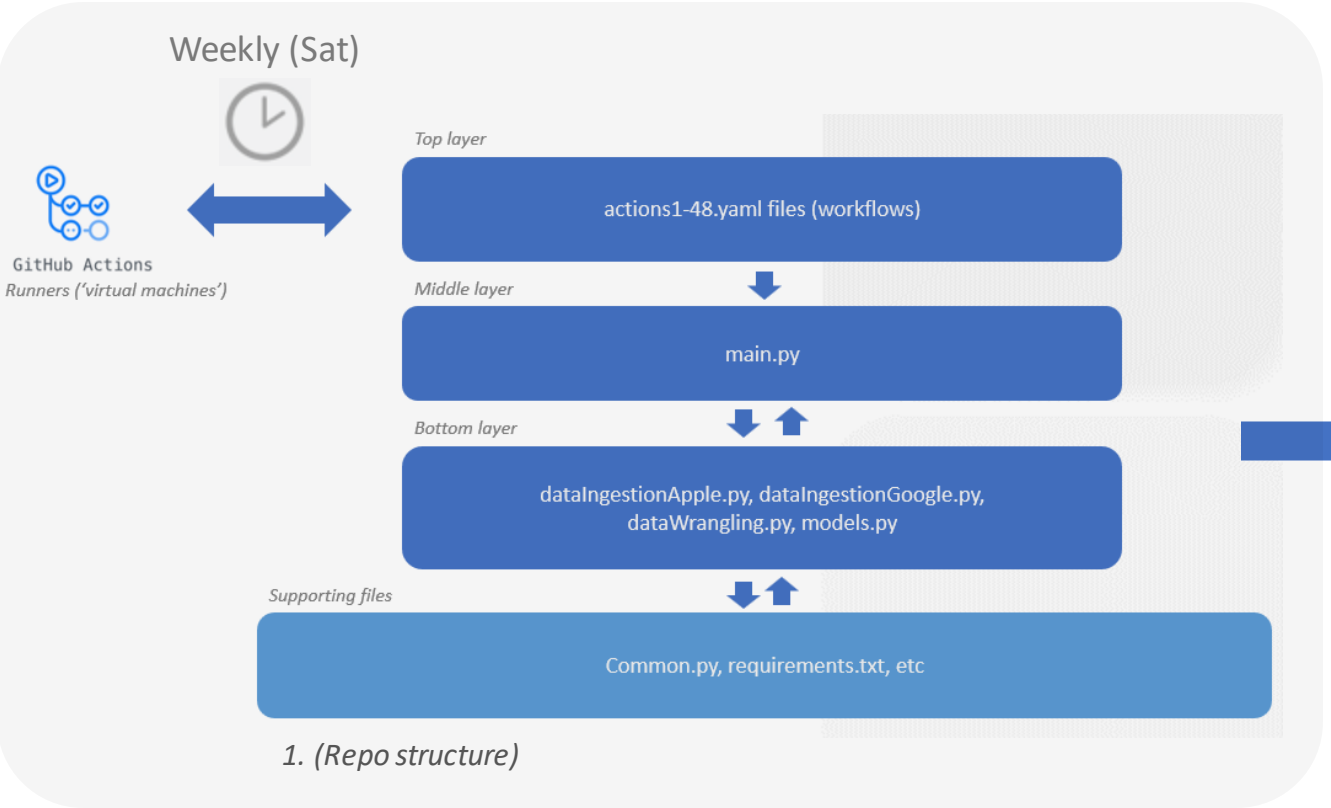
  - Model outputs pushed to Google Big Query

Top layer

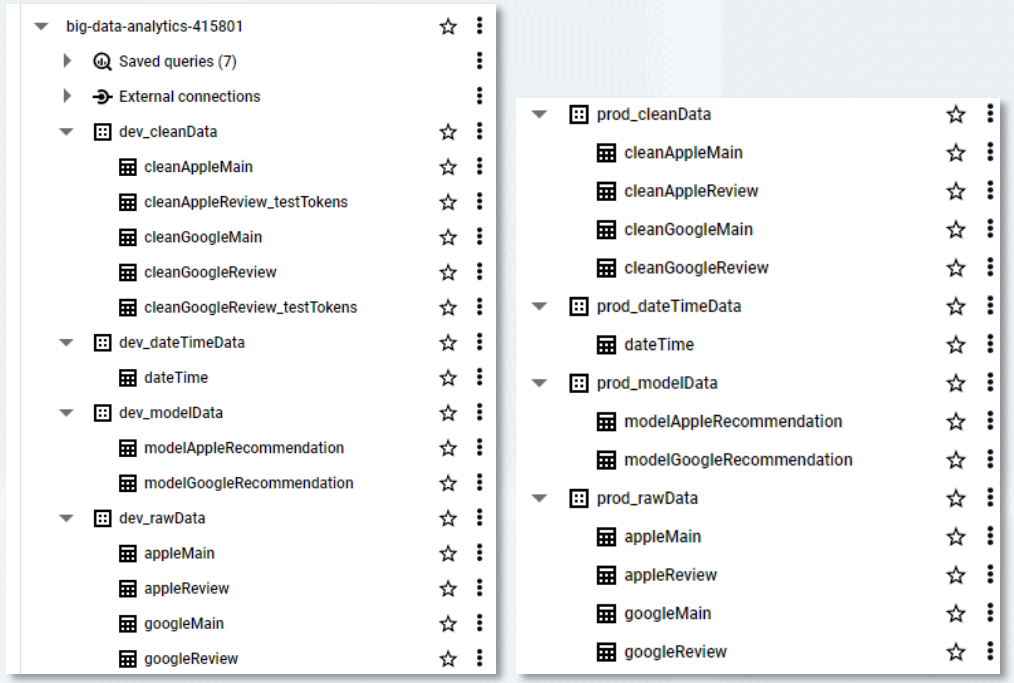
Middle layer

Bottom layer

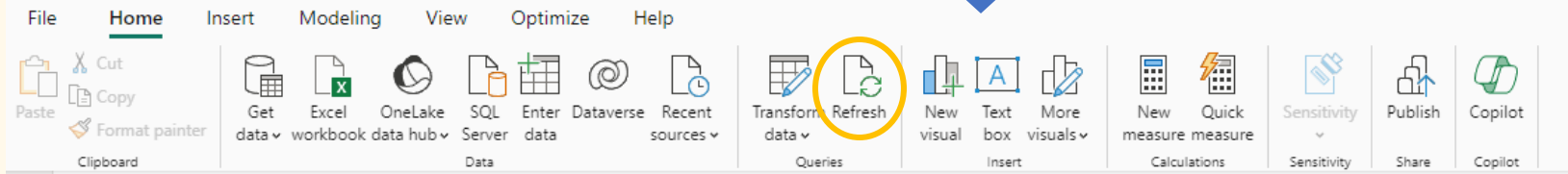
Overall Architecture Summary



1. (Repo structure)



2. (Google Big Query Data Warehouse)



3. (Power BI)

Exploratory ML and RCS

Machine Learning and Recommender System Exploration

Project Objectives

- *Analyze Key Factors/Features that drive app installs*
- *Explore predictive models on app installs*

Approach

- *Build various classification models*
- *If the model's prediction outcome is good, explore feature importance to derive insights*

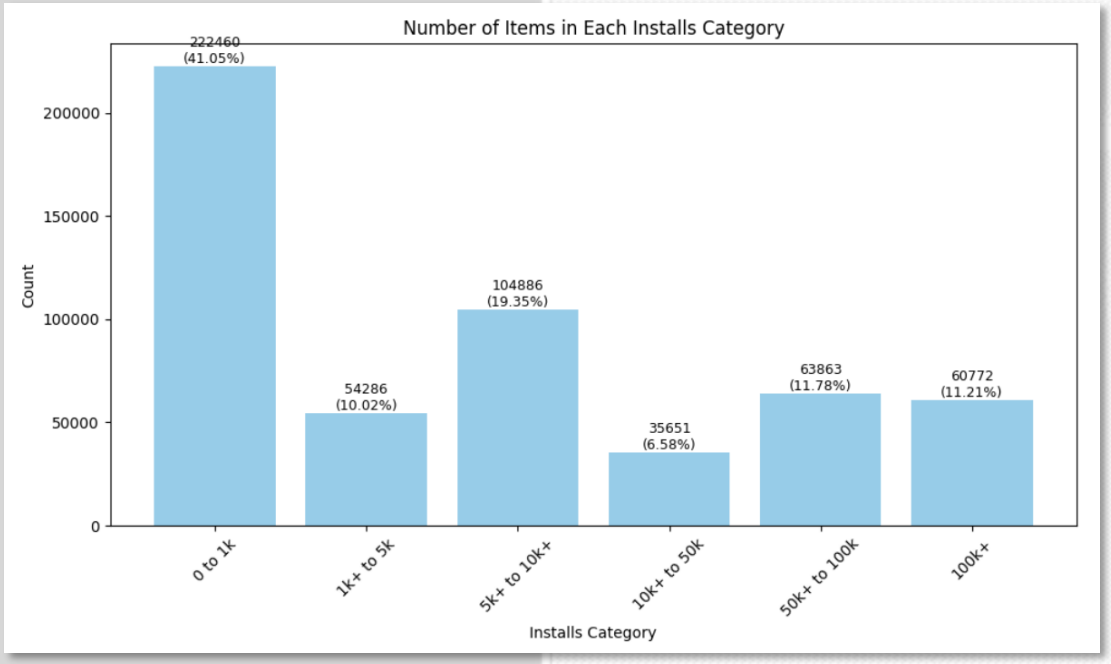
Categorical Variable Prediction

App Characteristics:

1. Genre
2. Content Rating
3. Ad Supported
4. Free
5. Offers In-App Purchases
6. Price
7. Score

App Performance

Number of installs



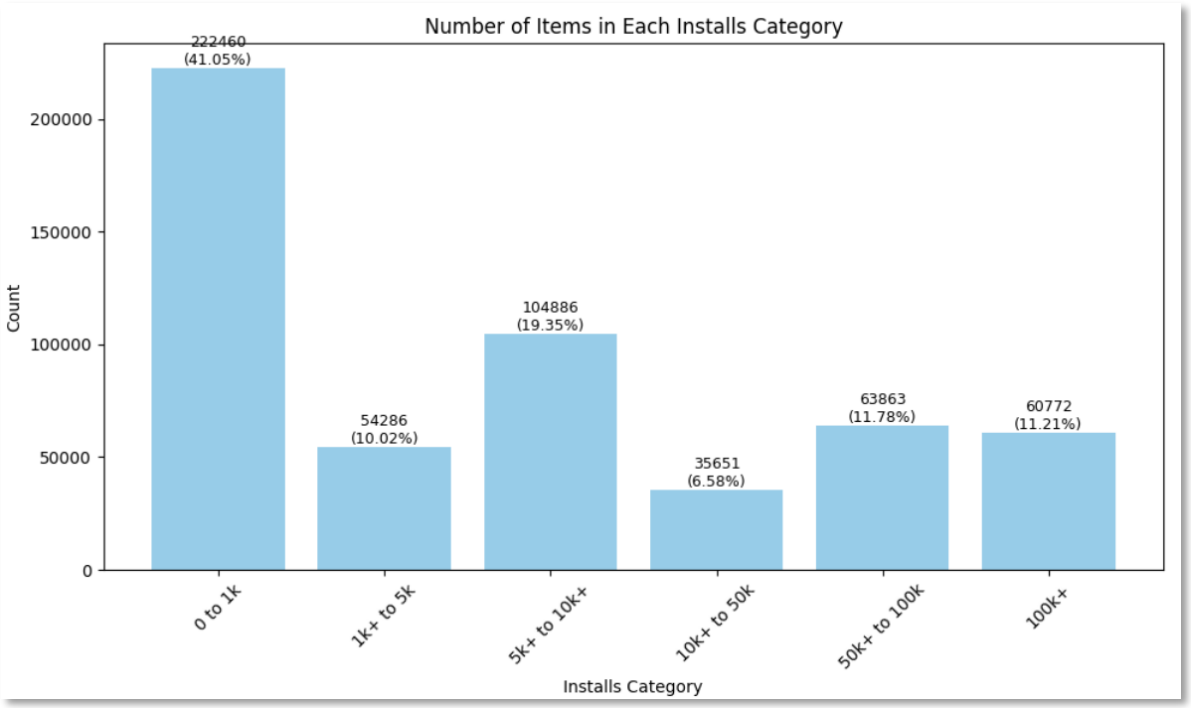
- 70% of features are categorical variables
- Decision tree and random forest models chosen for their ability to handle non-linear inputs

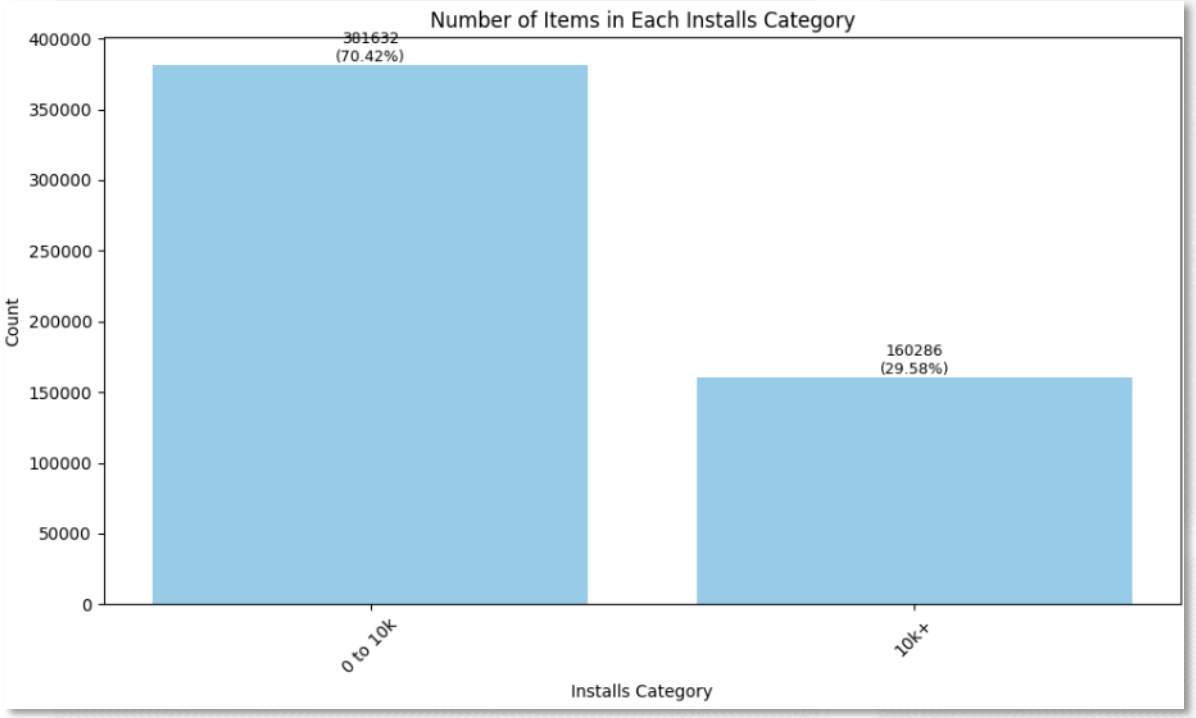
Initial Models

Performance Matrix	Decision Tree	Random Forest
Accuracy rate	45.62%	45.55%
Precision rate	35. 06%	32.74%
Recall rate	45.62%	45.55%
F1-score	36.67%	38.09%

** Evaluated using PySpark's Multiclass Classification Evaluator*

Output Variable Reduction





Bin the target variable into two categories based on the 70th percentile threshold for better model performance

Revised Models

		80% - 20% Sample Split			70% - 30% Sample Split		
		Decision Tree	Random Forest (50 trees; 10 max depth)	Random Forest (80 trees; 10 max depth)	Decision Tree	Random Forest (50 trees; 10 max depth)	Random Forest (80 trees; 10 max depth)
Overall Model Performance *(using PySpark's Multiclass Classification Evaluator)*	Accuracy rate	75.09%	74.99%	75.06%	74.96%	74.97%	74.99%
	Precision rate	73.61%	73.73%	73.93%	73.36%	73.54%	73.59%
	Recall rate	75.09%	74.99%	75.06%	74.96%	74.97%	74.99%
	F1-score	71.82%	74.36%	74.49%	71.64%	74.25%	74.28%

- Utilizing PySpark's Multiclass Classification Evaluator, the revised model demonstrates significantly improved performance, with all metrics above 70%
- To focus specifically on the model performance for the 70th percentile class, we recomputed the metrics

Revised Models

Exploratory ML and RCS

Machine Learning and Recommender System Exploration

		80% - 20% Sample Split			70% - 30% Sample Split		
		Decision Tree	Random Forest (50 trees; 10 max depth)	Random Forest (80 trees; 10 max depth)	Decision Tree	Random Forest (50 trees; 10 max depth)	Random Forest (80 trees; 10 max depth)
App Success Prediction	Accuracy rate	75.09%	74.99%	75.06%	74.96%	74.97%	74.99%
	Precision rate	67.06%	68.59%	69.41%	66.21%	67.54%	67.73%
	Recall rate	**31.31%**	**28.81%**	**28.33%**	**30.78%**	**29.07%**	**28.98%**
	F1-score	42.69%	40.58%	40.23%	42.03%	40.65%	40.59%

- Low recall rate indicates a high chance of missed opportunities, as the model fails to identify a significant portion of profitable investment opportunities
- This could lead to missed revenue and growth opportunities for app developers

Confusion Matrix

		Actual		
		0 - 10k	10k+	
Prediction	0 - 10k	72,271	22,958	95,229
	10k+	4,225	9,261	13,486
		76,496	32,219	108,715

Revised Models

	80% - 20% Sample Split			70% - 30% Sample Split		
	Decision Tree	**Random Forest (50 trees; 10 max depth)**	**Random Forest (80 trees; 10 max depth)**	**Decision Tree**	**Random Forest (50 trees; 10 max depth)**	**Random Forest (80 trees; 10 max depth)**
Top 3 Features	• Offer In-ad purchases - 45.19% • **Genre - 18.29%** • Ad supported – 17.57%	• **Genre – 43.71%** • Content rating – 24.48% • Score – 14.14%	• **Genre – 43.80%** • Content rating – 23.57% • Score – 14.89%	• Offer In-ad purchases – 45.47% • **Genre – 18.60%** • Ad supported – 17.55%	• **Genre – 43.01%** • Content rating – 24.50% • Score – 14.68%	• **Genre – 45.96%** • Content rating – 23.22% • Score – 14.35%

- App genre consistently ranks among the top three features in all the revised models, indicating its significant influence on app installs
- Focusing investments on these top genres can potentially lead to better app performance and higher user engagement
- These genres have been included in the monitoring dashboard for further analysis and focus

Competitor Analysis using
Content-Based Recommender System

- Implement **content-based recommender system** to identify **similar competitor apps**
- Perform **competitor analysis** to understand the performance of direct competitors
- **Gain insights** into **competitor strategies and market trends**
- Enhance app development and marketing strategies based on competitor analysis

Data Input

Option 1
Structured Data

- # of Installs
- Score
- Reviews
- Price
- Free
- Genre
- Content Rating
- Ad Supported

Option 2
Unstructured Data

- App title
- Description

Option 3
Mixed Data

- Features from Option 1
- Features from Option 2

Top 5 Similar Apps

Testing App: Password Guardian	Option 1: Structured Data (App Features)	Option 2: Unstructured Data (App Title and Description)	Option 3: Mixed Data
K-Nearest Neighbors (ball-tree algorithms)	1) Fun! CPU! : showing CPU usage 2) Astral Void - Sci-Fi Game 3) Slappy Paddles : arcade game 4) The Hera Hera Kingdom DX : puzzle game 5) SHUFFLE 2D Arcade	1) RevBits PAM 2) Security App : store password 3) Mistikee : generate & manage password 4) Roundshield password generator 5) Phone's ABI (Application Binary Interface (ABI)	1) Waste My Time – New Worlds : game 2) The Hera Hera Kingdom DX : puzzle game 3) Know, Made in : barcode scanner 4) Assortments : matching game 5) Twin Ships : game
	4 out of 5 Apps : Games	3 out of 5 Apps : Password Related App	4 out of 5 Apps : Games

Best Approach

Models Exploration

Testing App: Password Guardian	K-Nearest Neighbors (ball-tree algorithms)	Cosine-Similarity	Doc2Vec Embeddings
Top 5 Similar Apps	1) RevBits PAM 2) Security App : store password 3) Mistikee : generate & manage password 4) Roundshield password generator 5) Phone's ABI (Application Binary Interface (ABI)	1) Security App : store password 2) Roundshield password generator 3) Unmask – Your Privacy : store password 4) Masterkey : password manager 5) NEKTA Cloud : unique service for collecting and processing data in IoT	1) kpnPass Password Manager 2) Password Cerber : store password 3) KG Password Generator – Genera 4) MultiPass : secure identification & access code 5) TriGen : identify authenticator app
	Password related apps: **3 out of 5**	**Password related apps:** **4 out of 5**	**Password / Security related apps:** **5 out of 5**

Best Approach

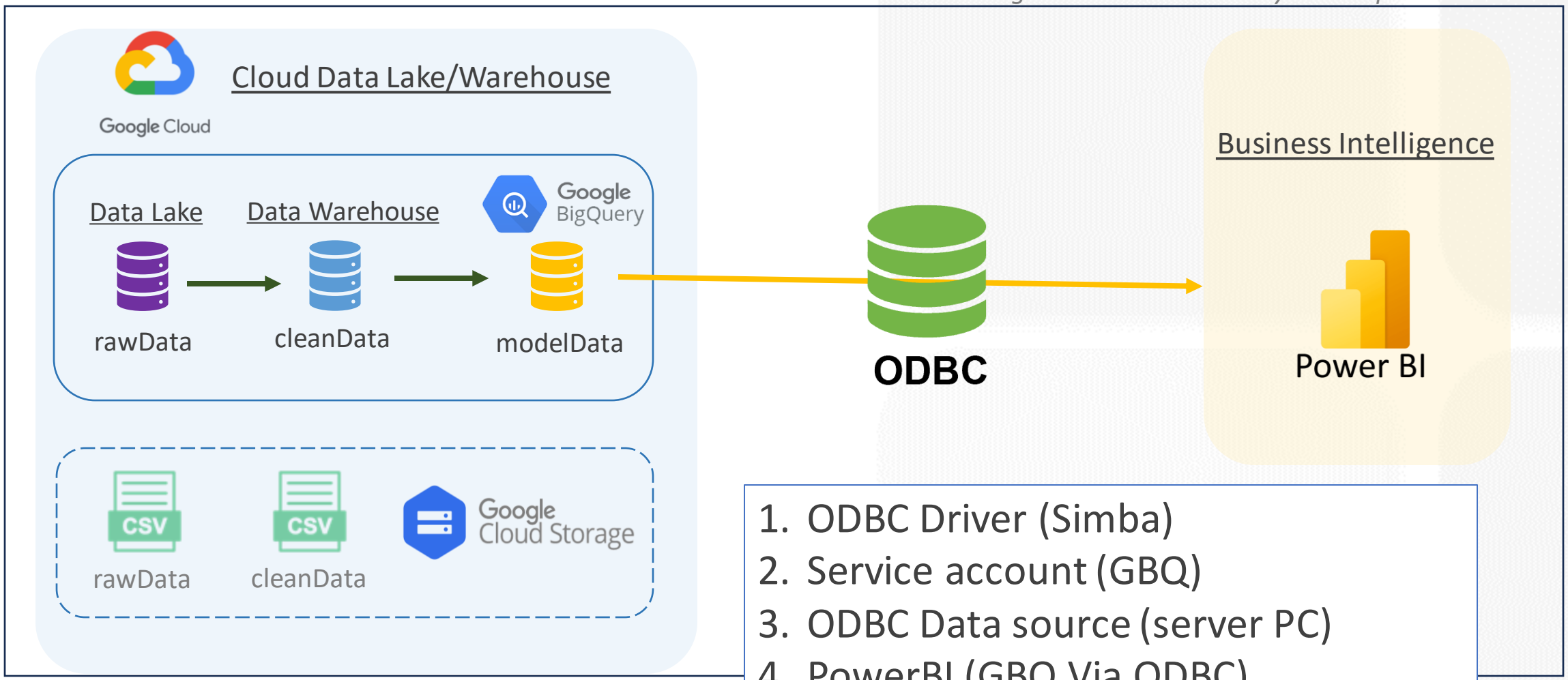
Dashboard

Power BI at a glance

BIG DATA
Architecture

Exploratory ML and RCS

Machine Learning and Recommender System Exploration



1. ODBC Driver (Simba)
2. Service account (GBQ)
3. ODBC Data source (server PC)
4. PowerBI (GBQ Via ODBC)

Dashboard Overview:
Insights into App Market Trends and Strategies

Trend Analysis:

- Descriptive analysis on Google and Apple apps
- Analyze **distribution of apps across different genres** and their **popularity** on both the Apple App Store and Google Play Store

Market Comparison:

- Comparative analysis between the Apple App Store and Google Play Store: uncovering **unique trends in each platform,** providing insights to developers and optimizing their app strategies

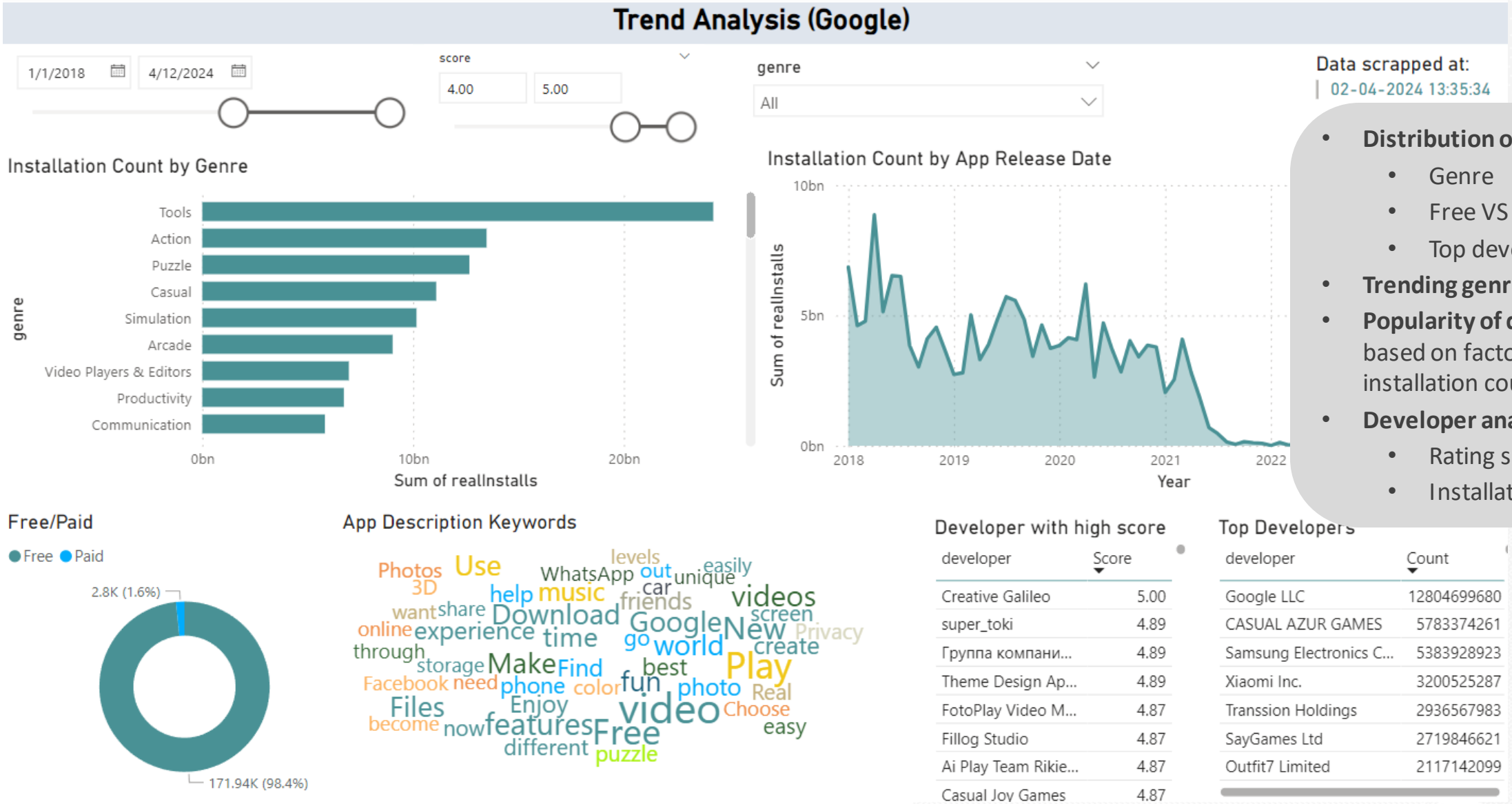
Recommender System:

- Recommendation to new app developers of existing applications in the market, their market share and similarities

Text Analytics:

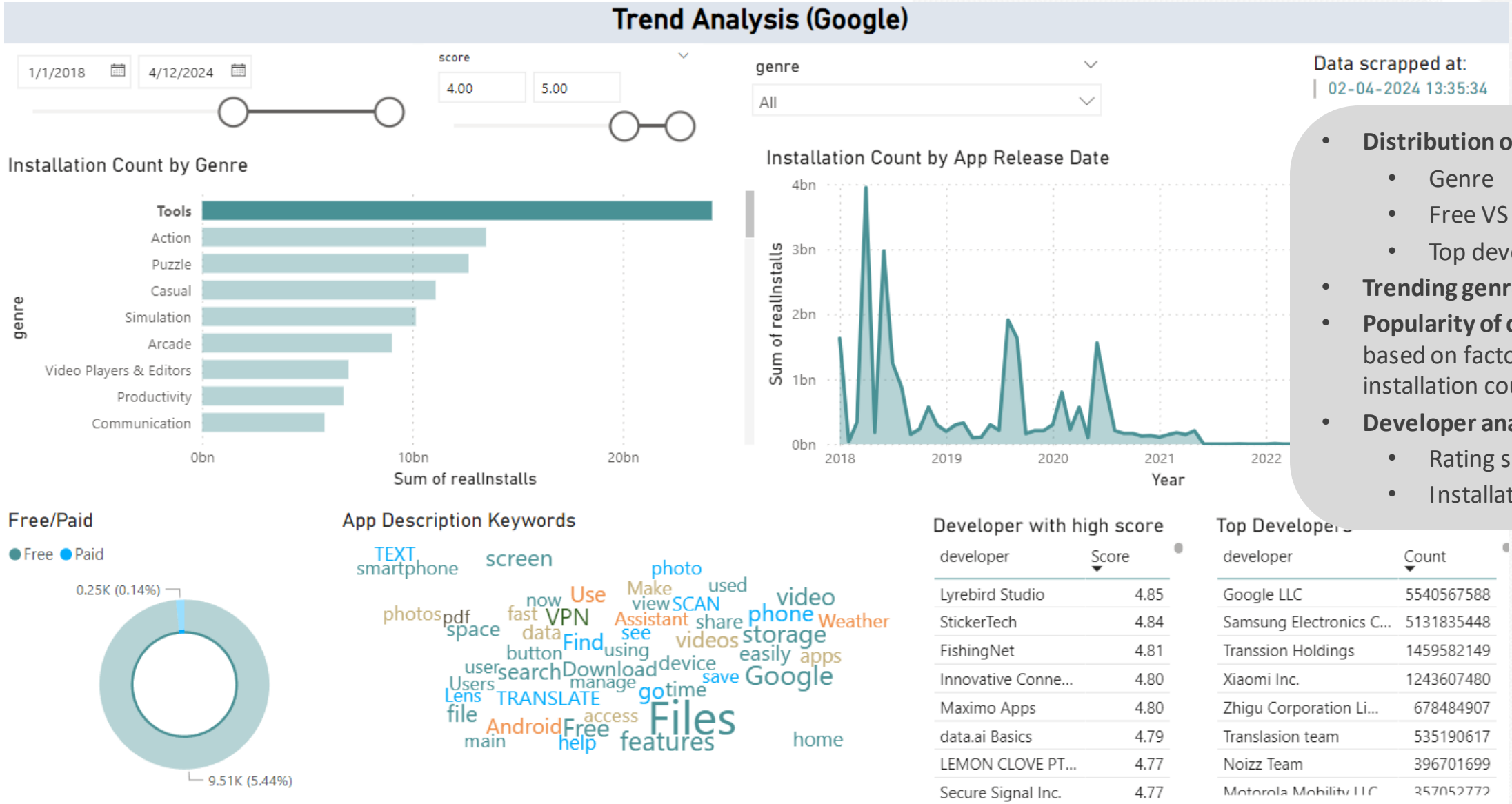
- Application description
- User reviews

Trend Analysis



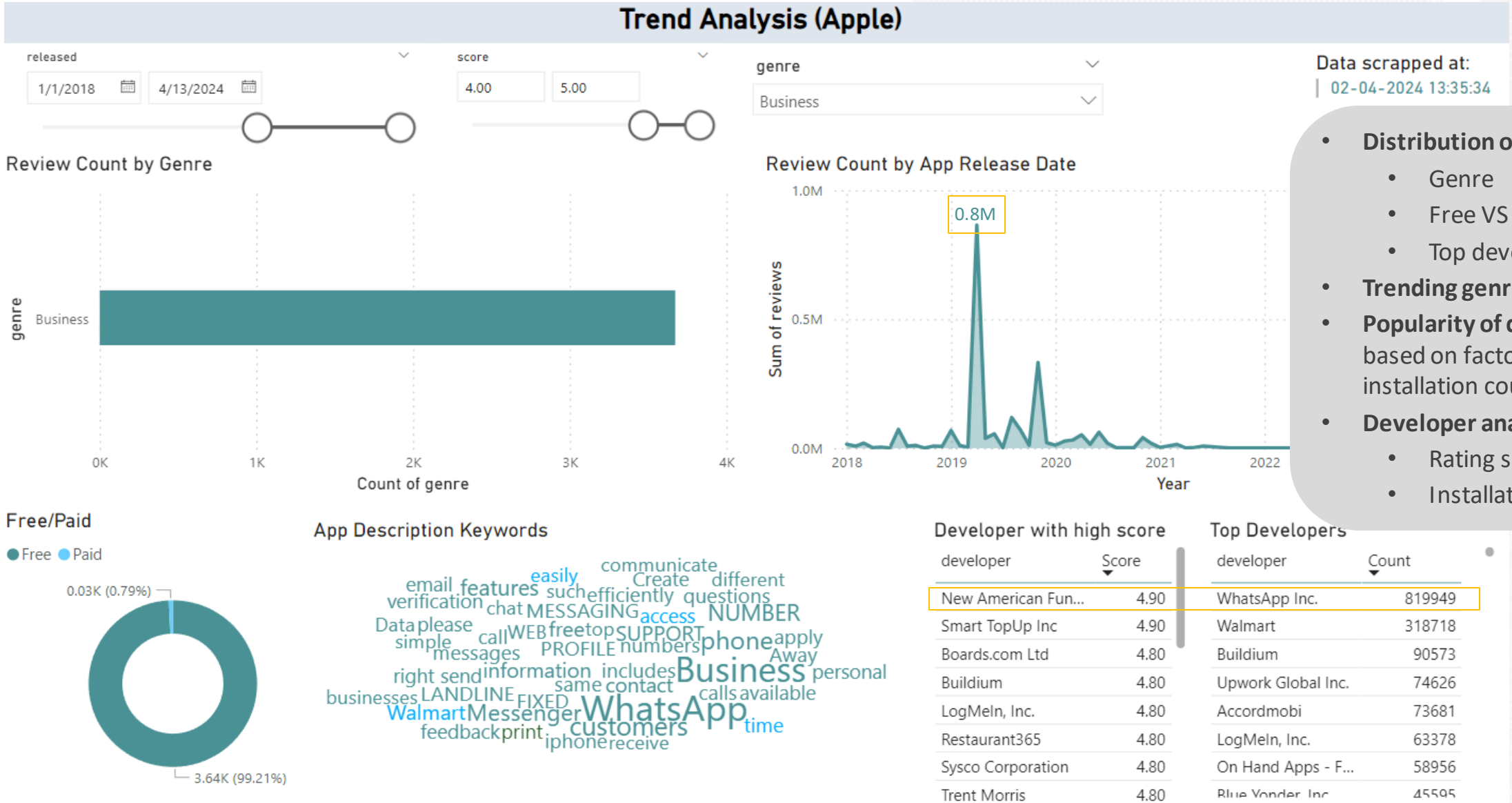
- **Distribution of**
  - Genre
  - Free VS Paid Apps
  - Top developers
- **Trending genre over time**
- **Popularity of different genres** based on factors e.g. installation counts
- **Developer analysis based on**
  - Rating score
  - Installation counts

Trend Analysis



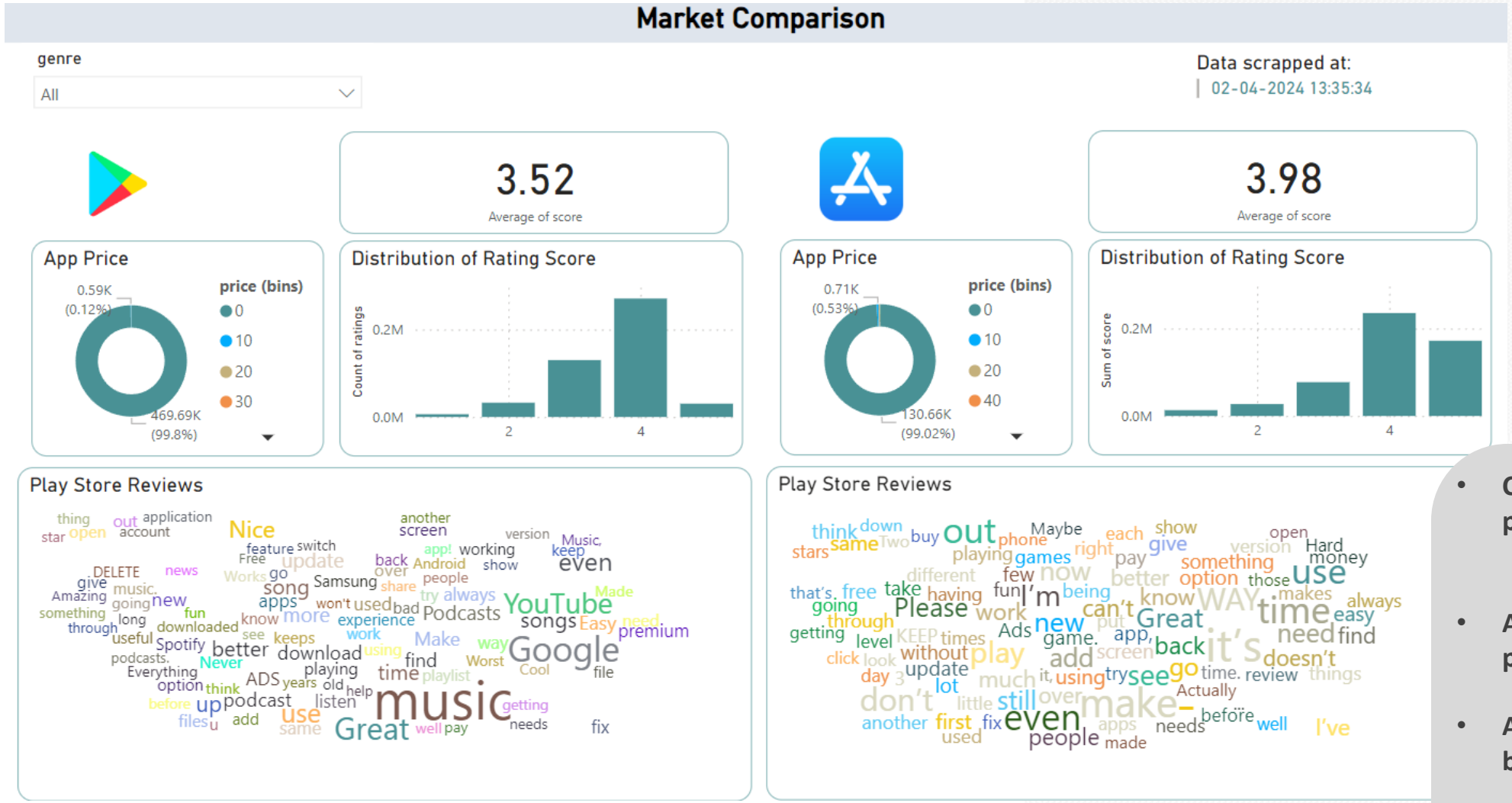
- **Distribution of**
  - Genre
  - Free VS Paid Apps
  - Top developers
- **Trending genre over time**
- **Popularity of different genres** based on factors e.g. installation counts
- **Developer analysis based on**
  - Rating score
  - Installation counts

Trend Analysis



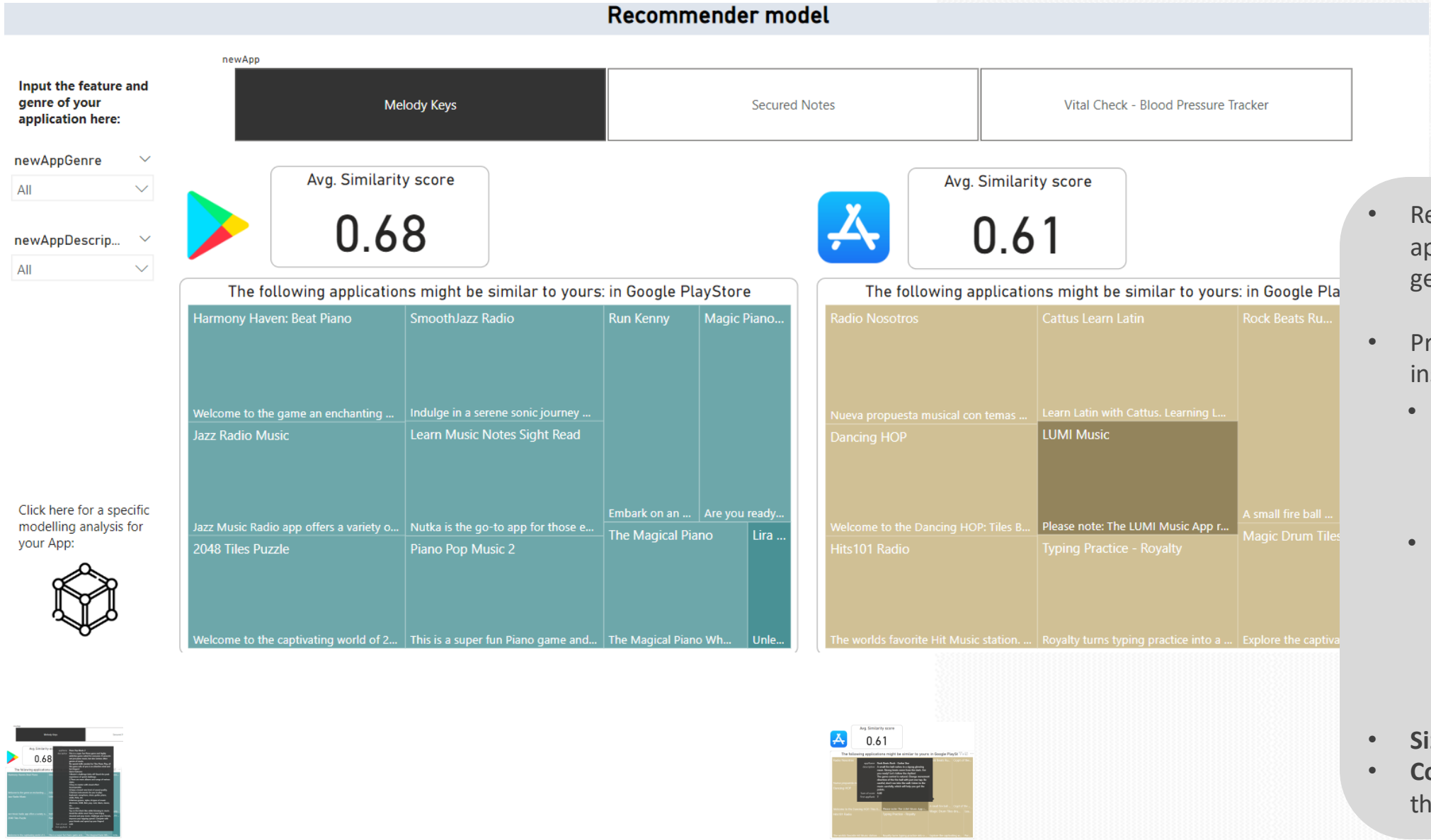
- **Distribution of**
  - Genre
  - Free VS Paid Apps
  - Top developers
- **Trending genre over time**
- **Popularity of different genres** based on factors e.g. installation counts
- **Developer analysis based on**
  - Rating score
  - Installation counts

Market Comparison



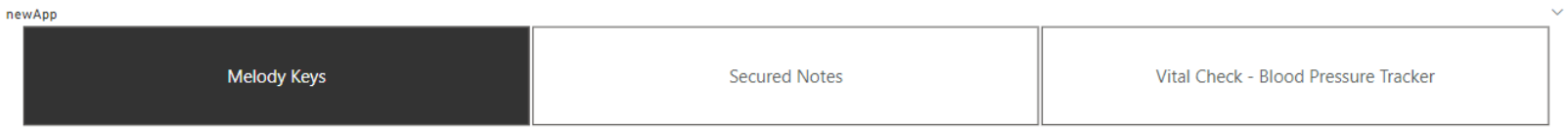
- **Compare key metrics of each platform:**
  - Average Rating scores
- **Analyze the different price points between two platforms.**
- **Analyze user feedback between different Genre**
  - Word cloud based on user reviews to identify common feedback themes and sentiments

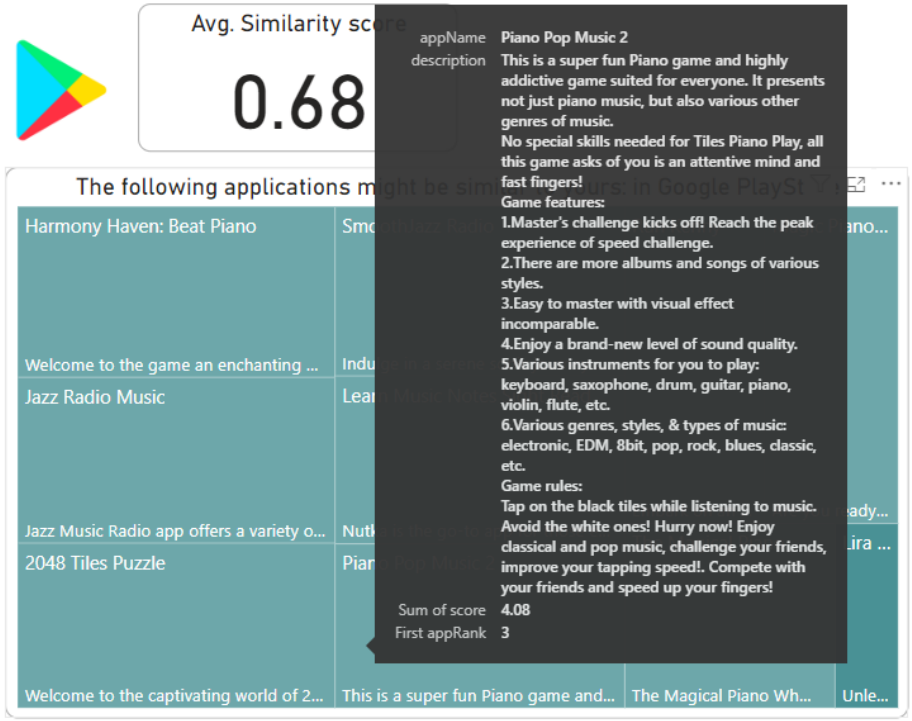
Recommender Model

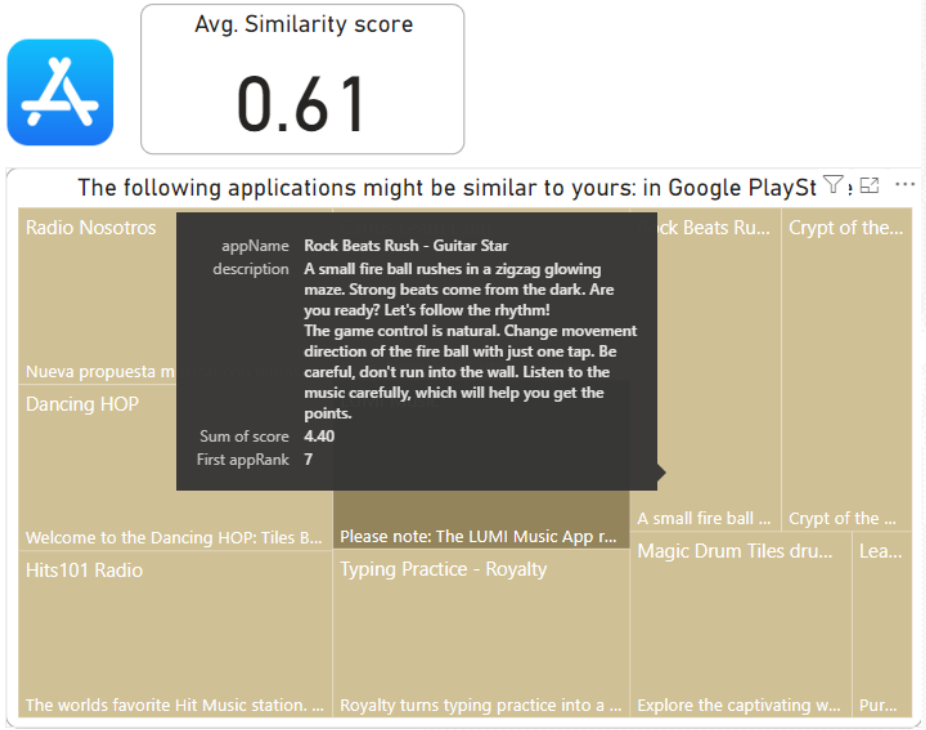


- Recommendation on similar apps based on New App's genre and description.
- Provide developers with insights on:
  - **Similarity Analysis:** Gauge how unique/creative the new app is comparing to existing apps.
  - **Market share analysis:** Gain insights on how the competitors performing currently in terms of review score.
- **Size of square:** review score
- **Color scheme:** Darkest with the highest similarity rank

Recommender Model







- Recommendation on similar apps based on New App's genre and description.
- Provide developers with insights on:
  - **Similarity Analysis:** Gauge how unique/creative the new app is comparing to existing apps.
  - **Market share analysis:** Gain insights on how the competitors performing currently in terms of review score.
- **Size of square:** review score
- **Color scheme:** Darkest with the highest similarity rank

Recommender Model – Explicit User Feedback Data Collection

New application sign up

This form collects information of new application details to provide customised analytical results for you.

What is the name of your new application? *

Your answer

Please provide a description for your application. *

Your answer

Please provide a short summary for your application.

Your answer

What genre will your application fall under? *

- Business
- Education
- Entertainment
- Finance
- Food & Drink
- Games
- Health & Fitness
- Lifestyle
- Medical
- Music
- Productivity
- Shopping
- Sports
- Weather
- Other:

- Customized modelling analysis for new app by getting the following inputs from user:
  - ❑ **Name**
  - ❑ **Description**
  - ❑ **Summary**

Conclusion

Monitoring, Logging, Conclusion & Retrospective

Challenges

Pre and Post Data Collections

Challenges

Pre-data collection

Data availability & Data quality

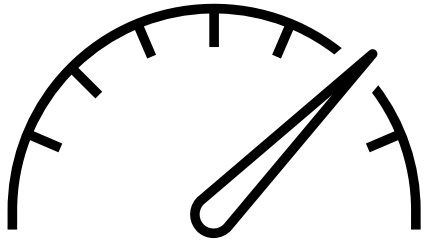
Data ingestion

Post data collection

Scalability

Model monitoring

Cost of Data Acquisition



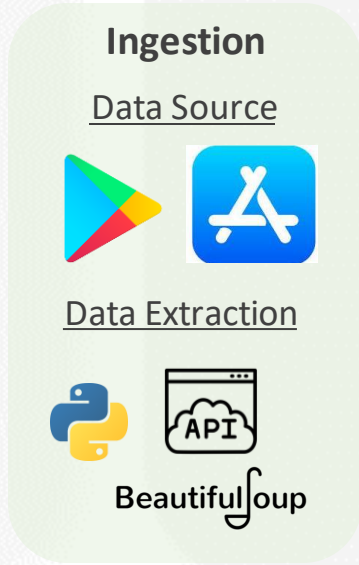
Challenges: Pre-Data collection

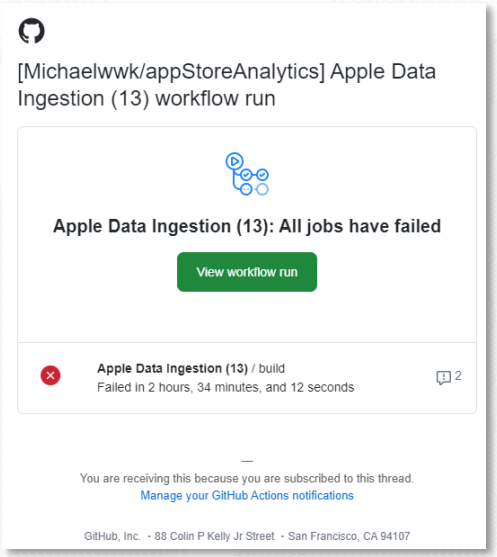
Web-scraping

- Noise and biases
- Inconsistent attributes between platforms
- Solution:
  - Data cleaning and preprocessing
  - Inconsistent attribute imputation and normalization

Ingestion failures

- Timeout and throttling issues – excessive API calls
- Solution:
  - Error logs
  - Iterative improvements





Strategic inserts of print() statements at key steps/functions to aid troubleshooting

Challenges: Post Data collection

Data source scalability

- Weekly data refreshes from different regions
- Solution:
  - Integration with APIs for comprehensive datasets

Model monitoring and evaluation

- Model drift and data drift
- Solution:
  - Drift detection and handling
  - Dashboard monitoring

Model monitoring & Evaluation

Detect drift

- ADWIN
- DDM
- EDDM

Handle drift

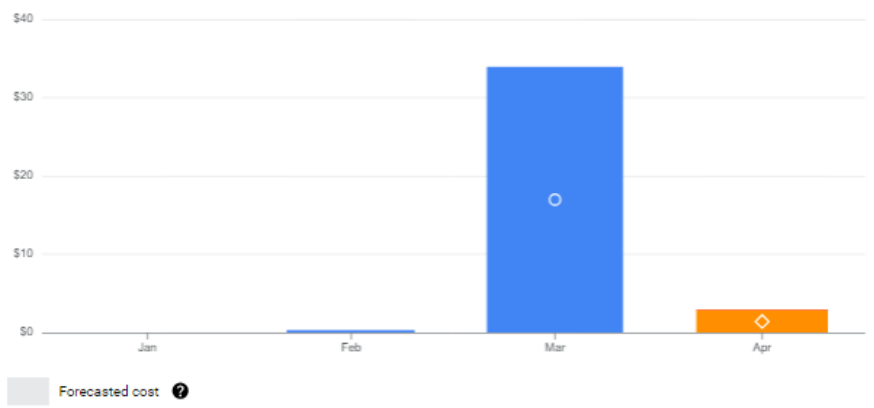
- Concept drift
- Data drift

Challenges: Cost of Data Acquisition

Cost - effective data acquisition process

- Future data acquisition costs VS benefits consideration

Services and usage	Subscription type	Price (USD)
Free tier	The BigQuery free tier gives customers 10 GiB storage, up to 1 TiB queries free per month, and other resources.	Free



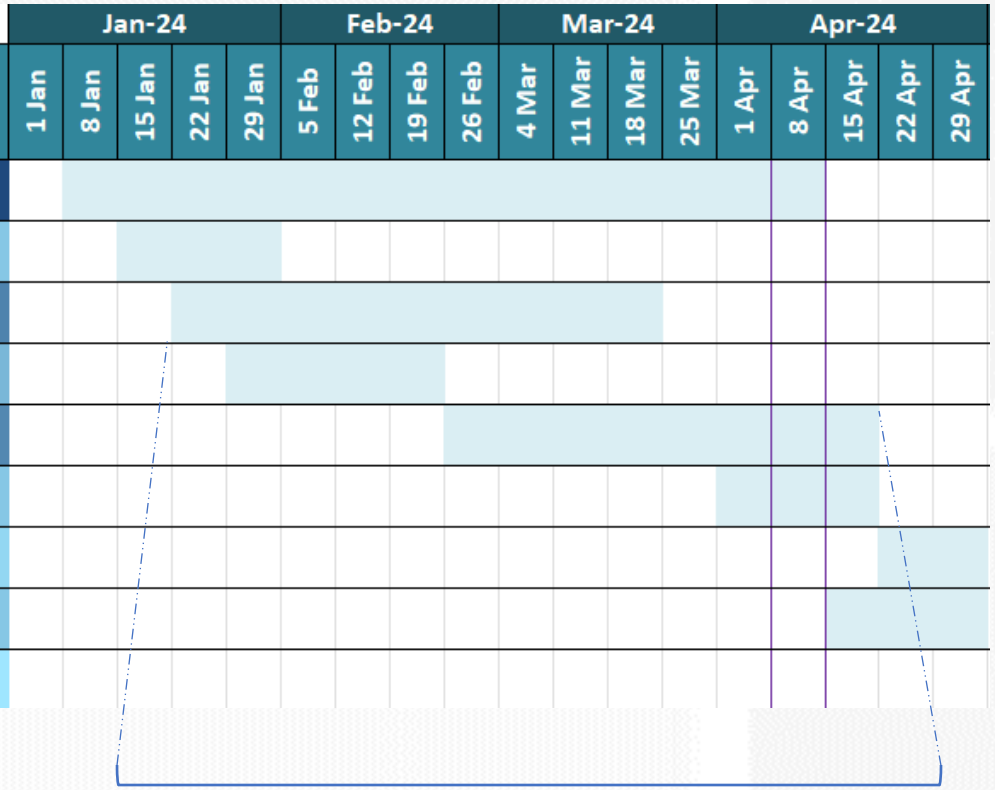
DOWNLOAD CSV

Service	Cost	Discounts	Promotions and others	Subtotal	% Change
Cloud Storage	$34.20	–	–	$34.20	New
BigQuery Storage API	$2.80	–	–	$2.80	New
BigQuery	$0.27	–	–	$0.27	New

Subtotal	$37.27
Tax	–
Filtered total	$37.27

Time Management

Item No.	Action Item	Assignees	Start Date	Initial Forecasted End Date	Est. Man Day efforts
1	Project topic research & gathering topics	All	6/1/2024	8/4/2024	31
2	Data collection/sourcing	All	13/1/2024	3/2/2024	7
3	Data preparation & transformation	Roy, Mich, WY	20/1/2024	19/3/2024	20
4	Exploratory Data Analysis	MC, Roy, Wudi	27/1/2024	24/2/2024	10
5	Predictive Model Development	Mich, Wudi, Wy, MC	24/2/2024	20/4/2024	19
6	Model Evaluation	MC, Mich, Wudi	30/3/2024	20/4/2024	7
7	Monitoring, maintenance and refinement	MC, WY, Roy	20/4/2024	30/4/2024	4
8	Project Demo Slide Preparation	MC, Roy, Wudi	11/4/2024	30/4/2024	7
0	Project Demo/Presentation	All	30/4/2024	30/4/2024	1



Data Ingestion, Prep and Modelling

Future Improvement Areas

Data quality enhancement

- Advance cleaning techniques
- Streamline feature engineering techniques

Enrich modelling

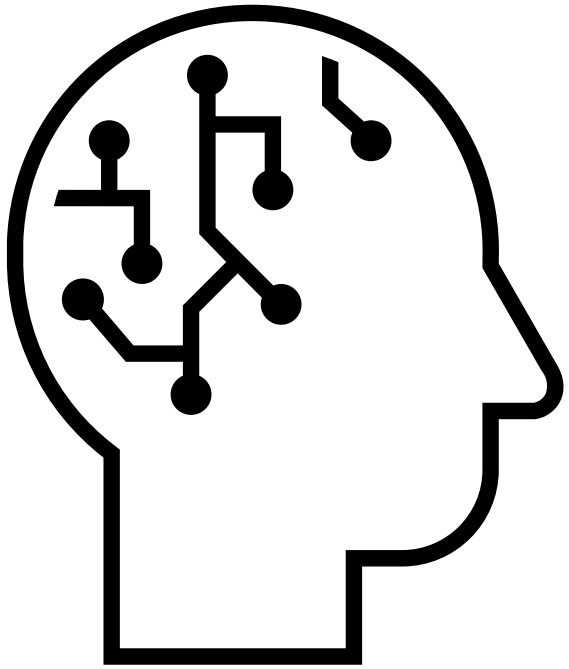
- Incorporate features such as user demographic data

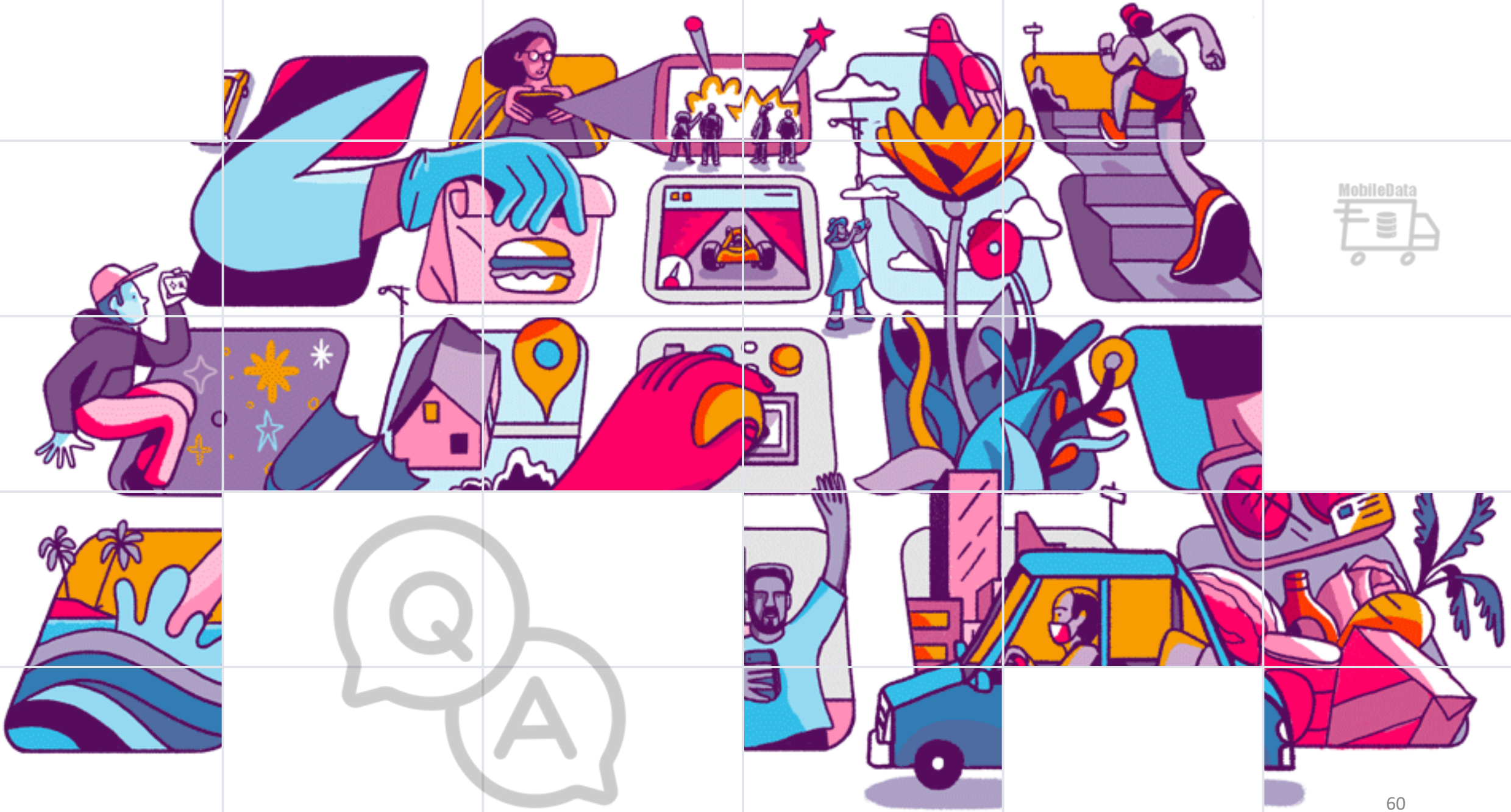
Optimization of Recommendations

- Provide additional insights to new app developers
- Enhance recommended app strategies

Scalability solution

- Refinement of current batch processing and API integrations for scalability
- Implementation of automated data processing pipeline using Kubernetes




MobileData
Q
A