# databricks project_2_notebook

(https://databricks.com/)

**Below are some instructions to help you understand the scope of this project.**

**a. First choose a dataset and inform the instructor about it. Make sure that your dataset satisfies at least one of the conditions below:**

**More than one million records. If less, consult with the instructor to get permission based on volume. (Volume) Different data types and a complex structure. More than forty columns. (Variety) Changing in time. Streaming data. (Velocity) A clear indication that the quality and trustworthiness of the dataset is compromised. (Veracity) A significant number of inconsistencies in the dataset. (Variability)**

**b. Second, you should find the metadata and make a brief but concise description of the organization generating that data and its business rules. How using Big Data helps to improve results of the organization/mission that generated this dataset? Is this social data? Machine data? Transactional data?**

**c. Now it is your turn to choose which big data task this problem requires. There are two options:**

**This dataset needs to be heavily transformed and it will allow me to create a dashboard and share the business insights with my classmates. This activity will be completed using Databricks which is the unified analytics platform used in the course. I encourage you all to attach a Python notebook (using pyspark) to the cluster in order to complete this data preprocessing tasks. However, your notebook can include other interfaces such as R,SQL, and Scala.**

**My dataset is fairly structured and I want to build a linear regression model. This activity will be completed using Databricks which is the unified analytics platform used in the course. I encourage you all to attach a Python notebook (using pyspark) to the cluster in order to build and train your model. However, your notebook can include other interfaces such as R,SQL, and Scala.**

**d. The submission file must include the following:**

**The notebook you created in Databricks (published) A report in a word doc format (Title page, Abstract, Table of Contents, Introduction, Body, and Conclusions) Your power point presentation (A well designed dashboard may be accepted as well)**

Dennis and I are using a dataset from Kaggle that contains 3,460,842 unique values and 26 columns.

The data set is a collection of information regarding Google Play applications.

The metadata is as follows: "This dataset was collected in August 2022 and supplemented with 200 thousand applications at the end of October 2022. Data was collected for the US region."

My download of the data was on 11/27/2022.

**The pivotal question we asked ourselves was: "What makes an application within the Google Play store successful?"**

But how should we define success?

After plenty of discussion, we concluded that rating score and minimum number of instillations was a good metric as to determine an application as successful.

We also utilized both PySpark and Spark SQL throughout our notebook. While PySpark is a tremendously powerful tool, there are often times that performing a function in SQL is easier and faster.

### Load Libraries

```
from pyspark.sql.functions import col, isnan, when, count
from pyspark.sql.functions import expr
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *
from pandas import read_csv
import pandas as pd
```

### Create Spark Session

```
spark=SparkSession.builder.appName('structured_streaming').getOrCreate()
```

Although we don't have the resources nor a live dataset to justify the necessity of a SparkSession, we felt that under normal circumstance a dataset of applications that were live would necessitate a SparkSession to properly function.

### Load CSV

```
google_play_df = spark.read.csv("/FileStore/tables
/google_play_dataset_by_tapivedotcom.csv", inferSchema = "true", header = "true")
```

### Preprocessing

Lets establish a count of our dataframe to better understand the amount of data we are working with.

```
print((google_play_df.count(), len(google_play_df.columns)))
```

```
(3460966, 36)
```

We can also run the display function to view the records within our data.

```
google_play_df.display()
```

Table

| | _c0 | appId |
|---|---|---|
| 1 | 0 | com.cyou.cma.clauncher.theme.v542279ccb70366ba6adf1de2 |
| 2 | 1 | com.cyou.cma.clauncher.theme.v546307d9199ddff33502cbdb |
| 3 | 2 | com.kids.paint.free.app |
| 4 | 3 | com.touchpal.touchpal_skins_tropical_paradise |
| 5 | 4 | com.musmon.Prado |
| 6 | 5 | de.diewidmannbibel.appfull.v1 |
| 7 | 6 | freeappscollection.freeapps.wish.quote.sunrise.photoframe.selfie.goodmorning |

Truncated results, showing first 1,000 rows.

Now that we've read the CSV and turned it into a dataframe lets fix some type issues. This is a friendly reminder that CSVs load all their values as string.

```
google_play_df = (google_play_df.withColumn("_c0", expr("CAST(_c0 as INT) as _c0"))
      .withColumn("minInstalls", expr("CAST(minInstalls as INT) as minInstalls"))
      .withColumn("free", expr("CAST(free as INT) as free"))
      .withColumn("minInstalls", expr("CAST(minInstalls as INT) as minInstalls"))
      .withColumn("offersIAP", expr("CAST(offersIAP as INT) as offersIAP"))
      .withColumn("price", expr("CAST(offersIAP as DOUBLE) as offersIAP"))
      .withColumn("ratings", expr("CAST(ratings as INT) as ratings"))
      .withColumn("adSupported", expr("CAST(adSupported as INT) as adSupported"))
      .withColumn("containsAds", expr("CAST(containsAds as INT) as containsAds"))
      .withColumn("reviews", expr("CAST(reviews as INT) as reviews"))
      .withColumn("sale", expr("CAST(sale as INT) as sale"))
      .withColumn("score", expr("CAST(score as DOUBLE) as score"))
      .withColumn("updated", expr("CAST(updated as INT) as updated"))
      .withColumn("histogram1", expr("CAST(histogram1 as INT) as histogram1"))
      .withColumn("histogram2", expr("CAST(histogram2 as INT) as histogram2"))
      .withColumn("histogram3", expr("CAST(histogram3 as INT) as histogram3"))
      .withColumn("histogram4", expr("CAST(histogram4 as INT) as histogram4"))
      .withColumn("histogram5", expr("CAST(histogram5 as INT) as histogram5"))
      .withColumn("releasedDay", expr("CAST(releasedDay as INT) as releasedDay"))
      .withColumn("releasedYear", expr("CAST(releasedYear as INT) as releasedYear"))
      .withColumn("maxprice", expr("CAST(maxprice as DOUBLE) as maxprice")))
```

Let's rename some columns to better represent their values and purpose.

```
google_play_df = google_play_df.withColumnRenamed("_c0","count_key")
google_play_df = google_play_df.withColumnRenamed("histogram1","1_star_ratings")
google_play_df = google_play_df.withColumnRenamed("histogram2","2_star_ratings")
google_play_df = google_play_df.withColumnRenamed("histogram3","3_star_ratings")
google_play_df = google_play_df.withColumnRenamed("histogram4","4_star_ratings")
google_play_df = google_play_df.withColumnRenamed("histogram5","5_star_ratings")

google_play_df.printSchema()
```

```
root
 |-- count_key: integer (nullable = true)
 |-- appId: string (nullable = true)
 |-- developer: string (nullable = true)
 |-- developerId: string (nullable = true)
 |-- developerWebsite: string (nullable = true)
 |-- free: integer (nullable = true)
```

```
 |-- genre: string (nullable = true)
 |-- genreId: string (nullable = true)
 |-- inAppProductPrice: string (nullable = true)
 |-- minInstalls: integer (nullable = true)
 |-- offersIAP: integer (nullable = true)
 |-- originalPrice: string (nullable = true)
 |-- price: double (nullable = true)
 |-- ratings: integer (nullable = true)
 |-- len screenshots: string (nullable = true)
 |-- adSupported: integer (nullable = true)
 |-- containsAds: integer (nullable = true)
 |-- reviews: integer (nullable = true)
 |-- releasedDayYear: string (nullable = true)
 |-- sale: integer (nullable = true)
 |-- score: double (nullable = true)
 |-- summary: string (nullable = true)
 |-- title: string (nullable = true)
 |-- updated: integer (nullable = true)
 |-- 1_star_ratings: integer (nullable = true)
 |-- 2_star_ratings: integer (nullable = true)
 |-- 3_star_ratings: integer (nullable = true)
 |-- 4_star_ratings: integer (nullable = true)
 |-- 5_star_ratings: integer (nullable = true)
 |-- releasedDay: integer (nullable = true)
 |-- releasedYear: integer (nullable = true)
 |-- releasedMonth: string (nullable = true)
 |-- dateUpdated: string (nullable = true)
 |-- minprice: string (nullable = true)
 |-- maxprice: double (nullable = true)
 |-- ParseReleasedDayYear: string (nullable = true)
```

Lets create a table to be able to pull our data using Spark SQL.

```
google_play_df.createOrReplaceTempView("google_play_table")
```

### EDA and Analysis

Let's check for null values in all our columns.

```
# Handling null values
# Create a count of Null, None, NaN and Empty/Blank values from all our dataframe's
columns using a for loop
google_play_df.select([count(when(col(c).contains('None') | \
                            col(c).contains('NULL') | \
                            (col(c) == '' ) | \
                            col(c).isNull() | \
                            isnan(c), c
                           )).alias(c)
                    for c in google_play_df.columns]).display()
```

Table

| | count_key | appId | developer | developerId | developerWebsite | fr |
|---|---|---|---|---|---|---|
| 1 | 9903 | 51 | 67 | 62 | 1105843 | 99 |

Showing 1 row.

Since we found null values within the key lets display them using Spark SQL.

```
spark.sql("SELECT * FROM google_play_table where count_key='None' or count_key IS
NULL").groupBy("count_key").count().orderBy('count', ascending = False).show()

# ALTERNATE
# google_play_df.filter(google_play_df.count_key.isNull()).count()  <-- 9903
# google_play_df.filter("count_key is NULL").count()
# spark.sql("SELECT * FROM google_play_table where count_key IS NULL").count() <--
9903
# spark.sql("SELECT * FROM google_play_table where count_key='None'").count() <--0
```

```
+---------+-----+
|count_key|count|
+---------+-----+
|     null| 9903|
+---------+-----+
```

We can also view the null values within developerWebsite.

```
spark.sql("SELECT * FROM google_play_table where developerWebsite='None' or
developerWebsite IS NULL or
developerWebsite=0").groupBy("developerWebsite").count().orderBy('count', ascending =
False).show()

# ALTERNATE
#spark.sql("SELECT * FROM google_play_table where developerWebsite='None'").count()
<--1105788
#google_play_df.filter('developerWebsite == "None"').count()

#google_play_df.filter(google_play_df.developerWebsite.isNull()).count()  <--55
```

```
+----------------+-------+
|developerWebsite|  count|
+----------------+-------+
|            None|1105788|
|            null|     55|
|               0|     11|
+----------------+-------+
```

Most importantly lets verify the null values within our score and minInstalls columns.

```
spark.sql("SELECT * FROM google_play_table where score IS
NULL").groupBy("score").count().orderBy('count', ascending = False).show()
```

```
+-----+-----+
|score|count|
+-----+-----+
| null| 9597|
+-----+-----+
```

```
spark.sql("SELECT * FROM google_play_table where minInstalls='None' or minInstalls IS
NULL").groupBy("minInstalls").count().orderBy('count', ascending = False).show()
```

```
+-----------+-----+
|minInstalls|count|
+-----------+-----+
|       null|  416|
+-----------+-----+
```

Now that we've verified and viewed the null values lets ultimately drop them. The amount of null values we have for a dataset of this size means that dropping these records won't damage the overall integrity of our dataset. We'll proceed by dropping the null records for our most important columns.

```
google_play_df = google_play_df.na.drop(subset='count_key')
```

```
google_play_df.count()
```

```
Out[52]: 3451063
```

```
google_play_df = google_play_df.na.drop(subset='score')
```

```
google_play_df.count()
```

```
Out[53]: 3451063
```

```
google_play_df = google_play_df.na.drop(subset='minInstalls')
```

```
google_play_df.count()
```

```
Out[54]: 3451012
```

Lets check for duplicates values within our key column.

```
google_play_df.distinct().count()
```

```
Out[55]: 3451012
```

Lets create a query to view a count aggregated by our count key.

```
google_play_df.groupby('count_key').agg(F.count('count_key').alias('total_count')).ord
erBy('total_count', ascending = False).show()
```

```
+---------+-----------+
|count_key|total_count|
+---------+-----------+
|      148|          1|
|      463|          1|
|      471|          1|
|      496|          1|
```

```
|      833|          1|
|     1088|          1|
|     1238|          1|
|     1342|          1|
|     1580|          1|
|     1591|          1|
|     1645|          1|
|     1829|          1|
|     1959|          1|
|     2122|          1|
|     2142|          1|
|     2366|          1|
|     2659|          1|
|     2866|          1|
|     3175|          1|
|     3749|          1|
+---------+-----------+
only showing top 20 rows
```

Thankfully our dataset does not contain any duplicate values within our key. Lets check for irregularities within the score column.

```
spark.sql("select count(title) from google_play_table where score > 5").show()
```

```
+------------+
|count(title)|
+------------+
|          80|
+------------+
```

Score is a column composed of an average of the star ratings system where the sum of one-star ratings column is multiplied by one, and the sum of two-star ratings column is multiplied by two. Like the previously mentioned columns this repeats until the final five-star column is multiplied and summed, and then an average is performed on all values obtained. Therefore, the score cannot be greater than five and all records where score is greater than five will be dropped.

```
google_play_df = google_play_df.filter(google_play_df['score'] < 5)

google_play_df.display()
```

Table

| | count_key | appId |
|---|---|---|
| 1 | 0 | com.cyou.cma.clauncher.theme.v542279ccb70366ba6adf1de2 |
| 2 | 1 | com.cyou.cma.clauncher.theme.v546307d9199ddff33502cbdb |
| 3 | 2 | com.kids.paint.free.app |
| 4 | 3 | com.touchpal.touchpal_skins_tropical_paradise |
| 5 | 4 | com.musmon.Prado |
| 6 | 5 | de.diewidmannbibel.appfull.v1 |
| 7 | 6 | freeappscollection.freeapps.wish.quote.sunrise.photoframe.selfie.goodmorning |

Truncated results, showing first 1,000 rows.

Now that we've gone and altered our dataframe again lets ensure our table updates.

```
google_play_df.createOrReplaceTempView("google_play_table")
```

Lets continue our analysis this time taking into account genre.

```
google_play_df.groupBy('genre').agg(F.count('genre').alias('total_count')).orderBy('to
tal_count', ascending = False).show()
```

```
+-----------------+-----------+
|            genre|total_count|
+-----------------+-----------+
|        Education|     383986|
|         Business|     248299|
|            Tools|     234482|
|    Music & Audio|     214410|
|    Entertainment|     208454|
|        Lifestyle|     161976|
|         Shopping|     159493|
|     Productivity|     146997|
|Books & Reference|     140928|
|     Food & Drink|     123985|
|          Finance|     104456|
|  Personalization|      97886|
|   Travel & Local|      89513|
|           Casual|      85838|
|    Communication|      79651|
|           Arcade|      77397|
```

```
|           Puzzle|      75811|
|           Sports|      70001|
|           Social|      68723|
| News & Magazines|      52232|
+-----------------+-----------+
only showing top 20 rows
```

Here we have a total count per genre. Lets explore some other columns as well.

```
google_play_df.groupBy('developer').agg(F.sum('sale').alias('total_sales')).orderBy('t
otal_sales', ascending = False).show()
```

```
+--------------------+-----------+
|           developer|total_sales|
+--------------------+-----------+
|       silentmind.co|          0|
|              ITLOVE|          0|
|              lxapps|          0|
|Sub Dimension Stu...|          0|
|             A.W.A.S|          0|
|          OmniaVerse|          0|
|         didaxbussid|          0|
|     Person Infosoft|          0|
|         SoftService|          0|
|       ООО РокетСофт|          0|
|       SAMADBO310DEV|          0|
|             Truebil|          0|
|      AnalogSoftware|          0|
|        Sul Revendas|          0|
|     Mikheev Aleksey|          0|
|  Nexgen Enterprises|          0|
|Telematics and Tr...|          0|
|            Visionip|          0|
|       FingertipApps|          0|
|        Giant Killer|          0|
+--------------------+-----------+
only showing top 20 rows
```

Here we can see that there are some issues within our dataframe.

```
google_play_df.filter(google_play_df['sale'] > 0).count()
```

```
Out[62]: 0
```

There are no records where sale is greater than 0. It seems are dataset was hoping to quantify in-app purchases yet never imported the data. Lets move on and take a look at the free column (boolean).

```
google_play_df.groupBy('free').count().show()
```

```
+----+-------+
|free|  count|
+----+-------+
|   1|3319019|
|   0| 106307|
+----+-------+
```

The vast majority of applications are free. Remember that 0 is false and 1 is true. Let's explore free a bit more.

```
spark.sql("select * from google_play_table where free is not null").display()
# We've excluded records where it is unknown if the application listed was free
```

Table

| | count_key | appId |
|---|---|---|
| 1 | 0 | com.cyou.cma.clauncher.theme.v542279ccb70366ba6adf1de2 |
| 2 | 1 | com.cyou.cma.clauncher.theme.v546307d9199ddff33502cbdb |
| 3 | 2 | com.kids.paint.free.app |
| 4 | 3 | com.touchpal.touchpal_skins_tropical_paradise |
| 5 | 4 | com.musmon.Prado |
| 6 | 5 | de.diewidmannbibel.appfull.v1 |
| 7 | 6 | freeappscollection.freeapps.wish.quote.sunrise.photoframe.selfie.goodmorning |

Truncated results, showing first 1,000 rows.

We've used SQL to show our data where the column free (whether a application is free or not) is not null. Now, do free applications recieve more instillations than paid applications?

```
google_play_df.na.drop(subset =
'free').groupBy('free').agg(F.sum('minInstalls').alias('total_instillations')).orderBy
('total_instillations',ascending=False).display()
```

Table

| | free | total_instillations |
|---|---|---|
| 1 | 1 | 422618039294 |
| 2 | 0 | 463540792 |

Showing all 2 rows.

Here we can see that applications that are free typically have much more installations than applications that are not free. Lets explore the relationship between minInstillations and other columns.

```
spark.sql("select genre, round(avg(score),2) as average_score, sum(ratings) as
total_ratings from google_play_table where ratings > 0 group by genre order by
total_ratings desc").display()
# Since anyone can publish an application to the google play app store we've decided
to exclude applications without any ratings
# This should counter the thinning affect that due to market saturation
```

Table Total Ratings per Genre Score vs Ratings Scatterplot

| | genre | average_score | total_ratings |
|---|---|---|---|
| 1 | Action | 3.82 | 704910628 |
| 2 | Tools | 3.81 | 504054244 |
| 3 | Casual | 3.85 | 425416589 |
| 4 | Social | 3.93 | 398376411 |
| 5 | Finance | 3.96 | 319831158 |
| 6 | Arcade | 3.83 | 314639738 |
| 7 | Shopping | 4.01 | 314588969 |

Showing all 48 rows.

Here we can see the average rating and total amount of ratings per genre. The first visuilization we created displays the total amount of ratings per genre. Our second visuilization was created to explore the idea that score is connected to a genre's total amount of ratings. We can see that this is not the case. Lets find out if there is a relationship between average score and the year an application was created.

```
spark.sql("select releasedYear, round(avg(score),2) as average_score from
google_play_table where ratings > 0 and releasedYear <= 2022 and releasedYear >= 2008
group by releasedYear order by releasedYear").display()
```

Table Average Score per Year

| | releasedYear | average_score |
|---|---|---|
| 1 | 2008 | 3.87 |
| 2 | 2009 | 3.55 |
| 3 | 2010 | 3.9 |
| 4 | 2011 | 3.93 |
| 5 | 2012 | 3.96 |
| 6 | 2013 | 3.99 |
| 7 | 2014 | 4 |

Showing all 15 rows.

It seems that the average score of apps released in two-thousand-nineteen is much lower than the other years. Lets create a list of the most popular applications with columns score and releasedYear.

```
spark.sql("select title, minInstalls, score, releasedYear from google_play_table where
ratings > 0 and title is not null order by minInstalls desc, score
desc").limit(10).display()
```

Table

| | title | minInstalls | score | releasedYear |
|---|---|---|---|---|
| 1 | Samsung Notes | 1000000000 | 4.75 | 2016 |
| 2 | Files by Google | 1000000000 | 4.63 | 2017 |
| 3 | Candy Crush Saga | 1000000000 | 4.62 | 2012 |
| 4 | Subway Surfers | 1000000000 | 4.59 | 2012 |
| 5 | Microsoft OneDrive | 1000000000 | 4.59 | 2012 |
| 6 | Google Lens | 1000000000 | 4.59 | 2018 |
| 7 | File Manager | 1000000000 | 4.58 | 2017 |

Showing all 10 rows.

Here we have the most downloaded applications ordered by their minimum number of installs and their score in descending order. Let's explore how free applications relate to advertisers.

```
display(spark.sql("select count_key, title, free, adSupported from google_play_table
where free = 1"))
```

Table Ad Supported Free Applications

| | count_key | title | free |
|---|---|---|---|
| 1 | 0 | Cute Bear love honey with Pin | 1 |
| 2 | 1 | Stylish Romantic Theme: Neon N | 1 |
| 3 | 2 | Paint | 1 |
| 4 | 3 | Tropical paradise Keyboard Ani | 1 |
| 5 | 6 | Good Morning Wish Photo Frame | 1 |
| 6 | 7 | Nail Art and Design - Latest 2 | 1 |
| 7 | 8 | PLAY OF THE GAME - image meme | 1 |

Truncated results, showing first 1,000 rows.

We can see that most applications that are free are not supported by advertisers. Lets create a table for our dashboard where ratings, instillations, and score are visible.

```
display(spark.sql("select title, genre, minInstalls, ratings, score from
google_play_table where title is not null order by minInstalls desc, ratings desc"))
```

Table

| | title | genre | minInstalls |
|---|---|---|---|
| 1 | Instagram | Social | 1000000000 |
| 2 | Garena Free Fire: 5th Anniv. | Action | 1000000000 |
| 3 | TikTok | Social | 1000000000 |
| 4 | Subway Surfers | Arcade | 1000000000 |
| 5 | Candy Crush Saga | Casual | 1000000000 |
| 6 | Snapchat | Communication | 1000000000 |
| 7 | Spotify: Music and Podcasts | Music & Audio | 1000000000 |

Truncated results, showing first 1,000 rows.

Lets check the quantiles for minInstalls to determine the skew for minInstalls. This is another method for finding how common it is for applications to become successful.

```
google_play_df.approxQuantile("minInstalls", [0.25, 0.5, 0.75, 0.95], 0)
```

```
Out[72]: [10.0, 100.0, 1000.0, 100000.0]
```

Finally, lets take the previous quantiles and create a table with them as to place it in our dashboard. We previously attempted to make a histogram to show this data but the large skew of the data made the visulization not very pleasing to look at.

```
df = {'25%':[10.0], '50%':[100.0], '75%':[1000.0], '95%':[100000]}

quantiles_df = pd.DataFrame(data=df)

quantiles_df.display()
```

Table

| | 25% | 50% | 75% | 95% |
|---|---|---|---|---|
| 1 | 10 | 100 | 1000 | 100000 |

Showing 1 row.

After the notebook was published and completed the cluster (project 2) was terminated