

Profitable App Profiles for the App Store and Google Play Markets

- For this project, we'll pretend we're working as data analysts for a company that builds Android and iOS mobile apps. We make our apps available on Google Play and in the App Store.
- We only build apps that are free to download and install, and our main source of revenue consists of in-app ads. This means that the number of users of our apps determines our revenue for any given app — the more users who see and engage with the ads, the better.
- Our goal for this project is to analyze data to help our developers understand what type of apps are likely to attract more users.

Our aim is to help our developers understand what type of apps are likely to attract more users on Google Play and the App Store. To do this, we'll need to collect and analyze data about mobile apps available on Google Play and the App Store.

Here are two datasets that seem suitable for our goals:

1. [A dataset](#) containing data about approximately 10,000 Android apps from Google Play.
2. [A dataset](#) containing data about approximately 7,000 iOS apps from the App Store.

❖ Exploring Function:

We'll start by opening and exploring these two datasets.

```
def explore_data(dataset, start, end, rows_and_columns=False):
    dataset_slice = dataset[start:end]
    for row in dataset_slice:
        print(row)
        print('\n') # adds a new (empty) line after each row

    if rows_and_columns:
        print('Number of rows:', len(dataset))
        print('Number of columns:', len(dataset[0]))
```

❖ Loading Data:

```
from csv import reader

### The Google Play data set ###
opened_file = open('googleplaystore.csv')
read_file = reader(opened_file)
android = list(read_file)
android_header = android[0]
android = android[1:]
```

❖ Exploring Data

```
print(android_header)
print('\n')
explore_data(android, 0, 3, True)
```

❖ Data Cleaning

Before beginning our analysis, we need to make sure the data we analyze is accurate, or the results of our analysis will be wrong.

We call this process of preparing our data for analysis **data cleaning**.

We do data cleaning before the analysis; it includes removing or correcting wrong data, removing duplicate data, and modifying the data to fit the purpose of our analysis.

I. Deleting Wrong Data

The Google Play data set has a dedicated discussion section, and we can see that one of the discussions outlines an error for row 10472. Let's print this row and compare it against the header and another row that is correct.

```
print(android[10472]) # incorrect row
print('\n')
print(android_header) # header
print('\n')
print(android[0])     # correct row
```

```
['Life Made WI-Fi Touchscreen Photo Frame', '1.9', '19', '3.0M', '1,000+', 'Free', '0', 'Everyone', '', 'February 11, 2018', '1.0.19', '4.0 and up']
```

```
['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver', 'Android Ver']
```

```
['Photo Editor & Candy Camera & Grid & ScrapBook', 'ART_AND_DESIGN', '4.1', '159', '19M', '10,000+', 'Free', '0', 'Everyone', 'Art & Design', 'January 7, 2018', '1.0.0', '4.0.3 and up']
```

The row 10472 corresponds to the app *Life Made WI-Fi Touchscreen Photo Frame*, and we can see that the rating is 19. This is clearly off because the maximum rating for a Google Play app is 5.

this problem is caused by a missing value in the 'Category' column. Therefore, we must delete this row.

To Do no.1:
Delete the
row.

Note:
Don't run
the code of
deletion
more than
once

II. Removing Duplicate Entries

If we explore the Google Play data set long enough, we'll find that some apps have more than one entry.

We don't want to count certain apps more than once when we analyze data, so we need to remove the duplicate entries and keep only one entry per app.

```
duplicate_apps = []
unique_apps = []

for app in android:
    name = app[0]
    if name in unique_apps:
        duplicate_apps.append(name)
    else:
        unique_apps.append(name)

print('Number of duplicate apps:', len(duplicate_apps))
print('\n')
print('unique_app:', unique_apps)
print('\n')
print('Examples of duplicate apps:', duplicate_apps[:15])
```

Let's explore one of the duplicated data:

```
for app in android:
    name = app[0]
    if name == 'Instagram':
        print(app)
```

['Instagram', 'SOCIAL', '4.5', '66577313', 'Varies with device', '1,000,000,000+', 'Free', '0', 'Teen', 'Social', 'July 31, 2018', 'Varies with device', 'Varies with device']

['Instagram', 'SOCIAL', '4.5', '66577446', 'Varies with device', '1,000,000,000+', 'Free', '0', 'Teen', 'Social', 'July 31, 2018', 'Varies with device', 'Varies with device']

['Instagram', 'SOCIAL', '4.5', '66577313', 'Varies with device', '1,000,000,000+', 'Free', '0', 'Teen', 'Social', 'July 31, 2018', 'Varies with device', 'Varies with device']

['Instagram', 'SOCIAL', '4.5', '66509917', 'Varies with device', '1,000,000,000+', 'Free', '0', 'Teen', 'Social', 'July 31, 2018', 'Varies with device', 'Varies with device']

One thing we could do is remove the duplicate rows randomly, but we could probably find a better way.

If you examine the rows, the main difference happens on the fourth position of each row, which corresponds to the number of reviews. The different numbers show that the data was collected at different times. We can use this to build a criterion for keeping rows. We won't remove rows randomly, but rather we'll keep the rows that have the highest number of reviews because the higher the number of reviews, the more reliable the ratings.

To Do no.2:

1. Create a dictionary where each key is a unique app name, and the corresponding dictionary value is the highest number of reviews of that app.

- Start by creating an empty dictionary named `reviews_max`.
- Loop through the Google Play data set (don't include the header row). For each iteration, do the following:
 - Assign the app name to a variable named `name`.
 - Convert the number of reviews to `float`. Assign it to a variable named `n_reviews`.
 - If `name` already exists as a key in the `reviews_max` dictionary **and** `reviews_max[name] < n_reviews`, update the number of reviews for that entry in the `reviews_max` dictionary.
 - If `name` **is not in** the `reviews_max` dictionary as a key, create a new entry in the dictionary where the key is the app name, and the value is the number of reviews. Make sure you don't use an `else` clause here, or the number of reviews will be incorrectly updated whenever `reviews_max[name] < n_reviews` evaluates to `False`.
- Inspect the dictionary to make sure everything went as expected. Measure the length of the dictionary — remember that the expected length is 9,659 entries.

2. Use the dictionary you created above to remove the duplicate rows:

- Start by creating two empty lists: `android_clean` (which will store our new cleaned data set) and `already_added` (which will just store app names).
- Loop through the Google Play dataset (don't include the header row), and for each iteration, do the following:
 - Assign the app name to a variable named `name`.
 - Convert the number of reviews to `float` and assign it to a variable named `n_reviews`.
- If `n_reviews` is the same as the number of maximum reviews of the app `name` (the number can be found in the `reviews_max` dictionary) **and** `name` is not already in the list `already_`
 - Append the entire row to the `android_clean` list (which will eventually be a list of lists and store our cleaned dataset).
 - Append the name of the app `name` to the `already_added` list — this helps us to keep track of apps that we already added.

3. Explore the `android_clean` dataset to ensure everything went as expected. The dataset should have 9,659 rows.

III. Removing Non-English Apps

If you explore the data sets enough, you'll notice the names of some of the apps suggest they are not directed toward an English-speaking audience

爱奇艺PPS - 《欢乐颂2》电视剧热播

【脱出ゲーム】絶対に最後までプレイしないで ~謎解き&ブロックパズル~

中国語 AQリスニング

لعبة تقدر تريح DZ

One way to go about this is to remove each app whose name contains a symbol that is not commonly used in English text — English text usually includes letters from the English alphabet, numbers composed of digits from 0 to 9, punctuation marks (., !, ?, ,, etc.), and other symbols (+, *, /, etc.).


All these characters that are specific to English texts are encoded using the ASCII standard. Each ASCII character has a corresponding number between 0 and 127 associated with it, and we can take advantage of that to build a function that checks an app name and tells us whether it contains non-ASCII characters.

To Do no.3:

1. Write a function that takes in a string and returns `False` if there's any character in the string that doesn't belong to the set of common English characters; otherwise, the function returns `True`.

- Inside the function, iterate over the input string. For each iteration check whether the number associated with the character is greater than 127. When a character is greater than 127, the function should immediately `return False` — the app name is probably non-English since it contains a character that doesn't belong to the set of common English characters.
- If the loop finishes running without the `return` statement executing, then it means no character had a corresponding number over 127 — the app name is probably English, so the functions should return `True`.

2. Use your function to check whether these app names are detected as English or non-English:

- `'Instagram'`
- `'爱奇艺PPS - 《欢乐颂2》电视剧热播'`
- `'Docs To Go™ Free Office Suite'`
- `'Instachat' `

The function seems to work fine, but some English app names use emojis or other symbols that fall outside of the ASCII range. Because of this, we'll remove useful apps if we use the function in its current form.

To minimize the impact of data loss, we'll only remove an app if its name has more than three non-ASCII characters

To Do no.4:

1. Change the function you created on the previous screen. If the input string has more than three characters that fall outside the ASCII range (0 - 127), then the function should return `False` (identify the string as non-English), otherwise it should return `True`.
2. Use the new function to check whether these app names are detected as English or non-English:

- `'Docs To Go™ Free Office Suite'`
- `'Instachat 🌐'`
- `'爱奇艺PPS - 《欢乐颂2》电视剧热播'`

3. Use the new function to filter out non-English apps from both datasets. Loop through each dataset. If an app name is identified as English, append the whole row to a separate list.
4. Explore the datasets and see how many rows you have remaining for each dataset.

IV. Isolating the Free Apps

we only build apps that are free to download and install, and our main source of revenue consists of in-app ads.

Our data sets contain both free and non-free apps, and we'll need to isolate only the free apps for our analysis.

To Do no.5:

1. Loop through each dataset to isolate the free apps in separate lists.
Make sure you identify the columns describing the app price correctly.
2. After you isolate the free apps, check the length of each dataset to see how many apps you have remaining.

Thank you.

Good Luck.