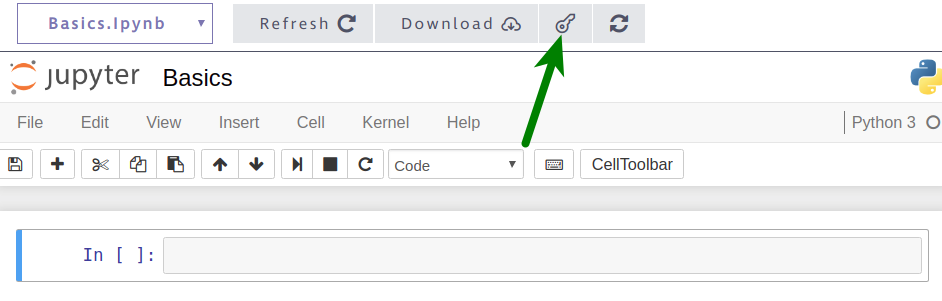
[**Source**](https://github.com/alexnisilev/Profitable-App-Profiles-for-the-App-Store-and-Google-Play-Markets) **by Alex Nisilev (From DataQuest)  
Edited by: Simon X Camilo**

**Guided Project: Profitable App Profiles for the App Store and Google Play Markets**

**Requirements:  
Python (**[**Download**](https://www.python.org/downloads/)**)  
Jupyter Notebook ([path] -m pip install jupyterlab) (**[**instructions**](https://jupyter.org/install.html)**)  
IOS & Android store apps sample (Found on Moodle)**

You are a data analysts for a company that makes mobile apps. All the apps target an English-speaking audience, are free, and contain ads, how much money we make is mostly influenced by how many people use our app. You are supposed to gather data to show the devs what app they should make to make the most profit.

If you are stuck, the solution notebook can be found [here](https://github.com/alexnisilev/Profitable-App-Profiles-for-the-App-Store-and-Google-Play-Markets), and through the key icon at the top of our interface. You can also look at the original version of this doc or perform a web search

  
Sidenote:  
-You can use this on your portfolio  
-What you need to do is below the “instructions” heading, use the outline or the navigator to go through the document headings

# **STEP 1: Intro**

First, let’s tell your readers what your project is about (Could be your employer or the devs). Use the first Markdown cell of the notebook to:

### Instructions

1. Add a title.
2. Explain in 1-2 paragraphs (no more):
   * What this is about
   * What your goal is

Don’t spend too much time here, this is a draft (an idea). You can come back here later and improve it.

# STEP 2: **Finding, exploring, and picking data**

Now, we need to analyze data about the mobile apps available on Google Play and App Store.

### **Finding** data to use

[This has already been done for you, below are the obstacles]  
There are millions of apps in these stores, and collecting this data requires a significant amount of time and money, so we'll try to analyze a sample of the data instead. To avoid spending resources on collecting new data ourselves, we should try to find any relevant existing data at no cost. This data can be found on Moodle.

### Getting familiar with exploring data sets

To make the data sets easier for you to explore, we created a function named explore\_data() that you can repeatedly use to print rows in a readable way.

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]))

The explore\_data() function:

* dataset is a list of lists
* Prints a row from the data sets
* Prints the number of rows and columns if rows\_and\_columns is True.
  + dataset shouldn't have a header row, otherwise the function will print the wrong number of rows (one more row compared to the actual length).

### Instructions

Now let's open the two data sets and explore them.

1. Open the two data sets we mentioned above, and save both as lists of lists.
   * Both CSV files can be opened directly in the Jupyter Notebook interface
   * If you run into an error named UnicodeDecodeError....  
     add encoding="utf8" to the open() function (for instance, use open('AppleStore.csv', encoding='utf8')).
2. Explore both data sets using the explore\_data() function above.
   * Print the first few rows of each data set.
   * Find the number of rows and columns of each data set.
3. Print the column names and **try to identify the columns that could help us with our analysis.**
   * If you don’t understand a column or if a column is not descriptive enough, add a link to the documentation so the readers can understand.

# **STEP 3: Data Cleaning**

Before beginning our analysis, we need to make sure the data we analyze is accurate, otherwise the results of our analysis will be wrong. This means that we need to:

* Correct or remove inaccurate data.
* Remove duplicated data

Remember that all our apps are free, and target a english-speaking audience, this meants that we need to remove:

* non-English apps like *爱奇艺PPS -《欢乐颂2》电视剧热播*.
* Paid apps

This process is called **data cleaning**. Data Analysts spend most of their time during this process. For this guided project, we'll guide you throughout the entire data cleaning process. In the future you will learn how to do it on your own

## Part **I:** Looking at online reports of wrong data

Let’s begin by looking at dedicated discussions online of our datasets, where people report wrong data found on these sets that we can fix.

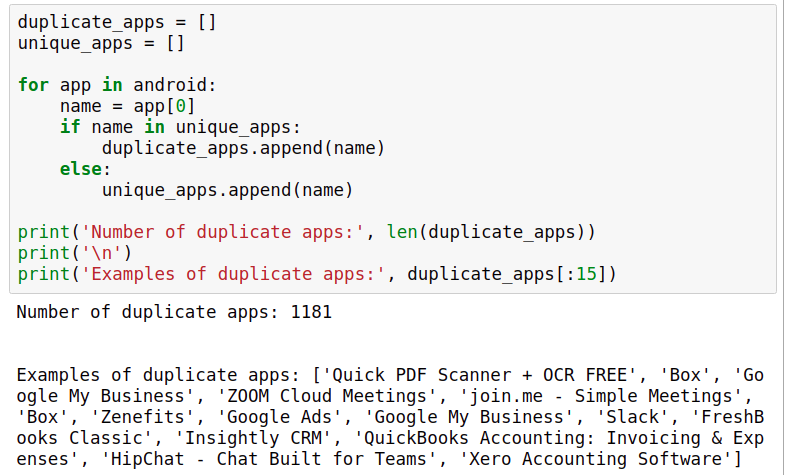
### Instructions

The Google Play data set has a dedicated [discussion section](https://www.kaggle.com/lava18/google-play-store-apps/discussion), and we can see that [one of the discussions](https://www.kaggle.com/lava18/google-play-store-apps/discussion/66015) describes an error for a certain row.

1. Read the discussion and find out what the index of the row is.
2. Print the row at that index to check whether it's indeed incorrect. Take into account the user reporting the error might or might have not removed the header row, so the index number might vary.
3. If the row has an error, remove the row using the [del](_blank) statement. For instance, to remove the row with the index 149 from a data set data that is stored as a list of list, you can use the code del data[149].
4. Make sure you don't run the del statement more than once, otherwise you'll delete more than one row.
5. Read the [discussion section](https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps/discussion) for the App Store data set, and see whether you can find any reports of wrong data.

## Part II: Duplicated apps

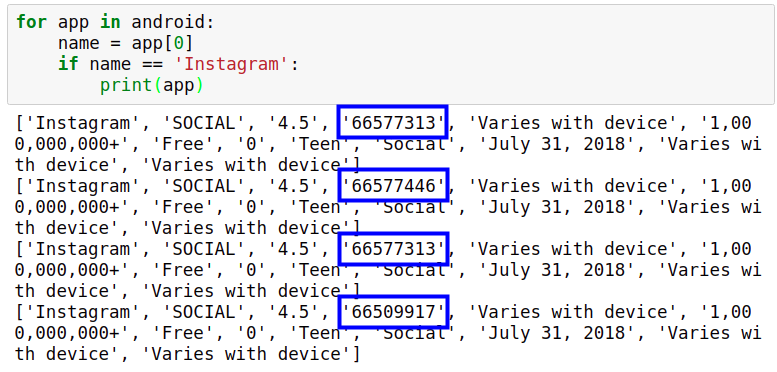
There are over 1000 duplicated apps on the Google Play Store data set. This is what we need to do to remove the duplicated apps from the data sets.



Above, we:

* Created two lists: one for storing the name of duplicate apps, and one for storing the name of unique apps.
* Looped through the android data set (the Google Play data set), and for each iteration:
  + We saved the app name to a variable named name.
  + If name was already in the unique\_apps list, we appended name to the duplicate\_apps list.
  + Else (if name wasn't already in the unique\_apps list), we appended name to the unique\_apps list.

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



If you examine the rows we printed for the Instagram app, the main difference happens on the fourth position of each row, which corresponds to the number of reviews. The different numbers show the data was collected at different times.

We can use this information to build a criterion for removing the duplicates. The higher the number of reviews, the more recent the data should be. Instead of removing duplicates randomly, with this criterion we can remove only the older duplicates.

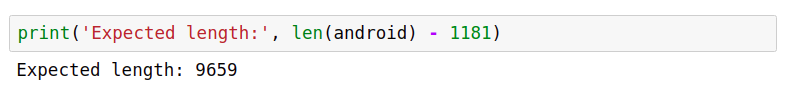
We'll remove the rows on the next part. Now it's your turn to write some code and confirm the data has duplicate entries.

**Instructions**

1. Using a combination of narrative and code, explain to the reader that the Google Play data set has duplicate entries. Print a few duplicate rows to confirm.
2. Count the number of duplicates using the technique we learned above.
3. Explain that you won't remove the duplicates randomly. Describe the criterion you're going to use to remove the duplicates.
   * The criterion above works, but you can make your own criterion. You need to support why this is a good criterion.

**Part III: Removing the duplicates**

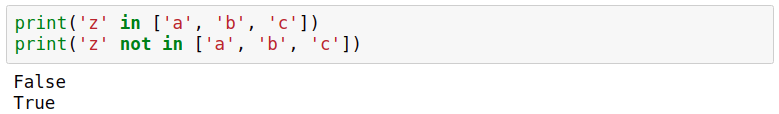
We looped through the Google Play data set and found that there are 1,181 duplicates. After we remove the duplicates, we should be left with 9,659 rows:



To remove the duplicates, we will:

* Create a dictionary, where each dictionary key is a unique app name and the corresponding dictionary value is the highest number of reviews of that app.
* Use the information stored in the dictionary and create a new data set, which will have only one entry per app (and for each app, we'll only select the entry with the highest number of reviews).

To turn the steps above into code, we'll need to use the [not in](_blank) operator. The not in operator is the opposite of the in operator. For instance:



Now let's write the code to remove the duplicate entries.

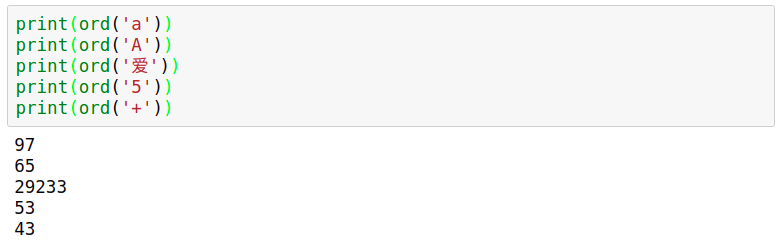
### Instructions

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 (make sure you don't include the header row). For each iteration:
     + 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, otherwise 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 data set (make sure you don't include the header row), and for each iteration:
     + 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\_added (read the solution notebook to find out why we need this supplementary condition):
     + Append the entire row to the android\_clean list (which will eventually be a list of list and store our cleaned data set).
     + 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 data set to ensure everything went as expected. The data set should have 9,659 rows. The two steps above are a bit more involved, so make sure you use Markdown to explain the readers the steps you took.

## **Part III: Removing non-english speaking apps**

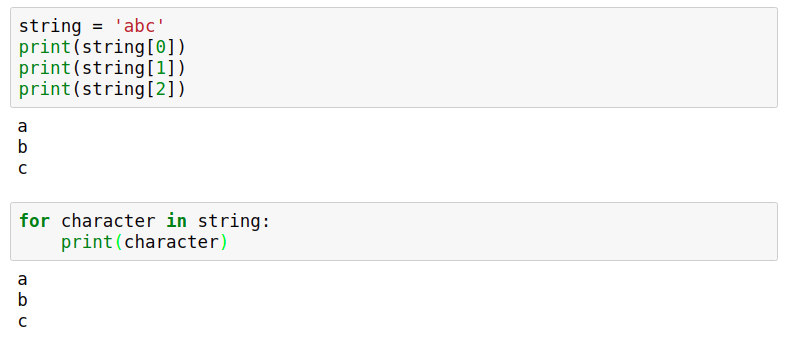
We're not interested in keeping these apps, so we'll remove them.

Behind the scenes, each character we use in a string has a corresponding number associated with it. We can get the corresponding number of each character using the [ord()](_blank) built-in function.



The numbers corresponding to the characters we commonly use in an English text are all in the range 0 to 127, according to the [ASCII](https://en.wikipedia.org/wiki/ASCII) (American Standard Code for Information Interchange) system. Based on this number range, we can build a function that detects whether a character belongs to the set of common English characters or not. If the number is equal to or less than 127, then the character belongs to the set of common English characters.

If an app name contains a character that is greater than 127, then it means that the app has a non-English name. Our app names, however, are stored as strings, so how could we take each individual character of a string and check its corresponding number?



Let's first try to write the function we talked about above, and in the next part of step 3 we'll remove the rows corresponding to the non-English apps.

### Instructions

Disclaimer: Don’t use this with your dataset yet

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 it 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 being executed, 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 😜'

## **Part IV**: **Part III has false positives.**

This is because emojis and characters like ™ fall outside the ASCII range and have corresponding numbers over 127.



If we're going to use the function we've created, we'll lose useful data since many English apps will be incorrectly labeled as non-English. To fix this, we'll only remove an app if its name has more than three characters outside the ASCII range. Our filter function is still not perfect, but it will be good enough.

Let’s improve our function, and then use it to filter out the non-English apps.

### Instructions

1. 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 data sets. Loop through each data set. If an app name is identified as English, append the whole row to a separate list.
4. Explore the data sets and see how many rows you have remaining for each data set.

## **Part V: Removing paid apps.**

Our data sets contain both free and non-free apps; we'll need to isolate only the free apps for our analysis. This is the last part of our cleaning process. On the next step, we're going to start analyzing the data.

### Instructions

1. Remove paid apps from the data sets. Make sure you identify the columns describing the app price correctly.
2. Check the length of each data set to see how many apps you have remaining.

# **STEP 4: Data Analysis**

## Part I: Profiling

Our aim is to determine the kinds of apps that are likely to attract more users because our revenue is highly influenced by the number of people using our apps.

To minimize risks and overhead, our validation strategy for an app idea is comprised of three steps:

1. Build a minimal Android version of the app, and add it to Google Play.
2. If the app has a good response from users, we develop it further.
3. If the app is profitable after six months, we build an iOS version of the app and add it to the App Store.

Because our end goal is to add the app on both Google Play and the App Store, we need to find app profiles that are successful on both markets.

### Instructions

Let's begin the analysis by getting a sense of what are the most common genres for each market. For this, we'll need to build frequency tables for a few columns in our data sets.

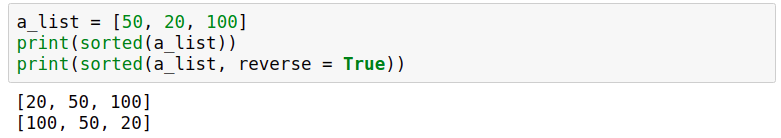
1. Give readers more context into why we want to find an app profile that fits both the App Store and Google Play. Explain our validation strategy for an app idea.
2. Identify the columns you could use to figure out what are the most common genres in each market.

## **Part II: Sorting genre**

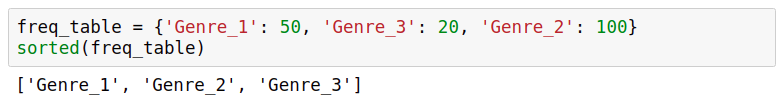
Two things you need to do

* Generate percentages from the tables.
* Display the percentages in descending order (in a second function).

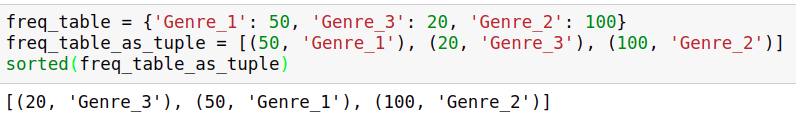
To do that, we'll need to make use of [sorted()](_blank) function. (the reverse parameter controls whether the order is ascending or descending).



The sorted() function doesn't work too well with dictionaries because it only considers and returns the dictionary keys.



However, it works on a list of tuples. To ensure the sorting works right, the dictionary value comes first, and the dictionary key comes second:



Using the workaround above, we wrote a helper function for you below named display\_table(), which you'll be able to combine with the function you're going to write in the next exercise. The display\_table() function you see below:

def display\_table(dataset, index):

table = freq\_table(dataset, index)

table\_display = []

for key in table:

key\_val\_as\_tuple = (table[key], key)

table\_display.append(key\_val\_as\_tuple)

​

table\_sorted = sorted(table\_display, reverse = True)

for entry in table\_sorted:

print(entry[1], ':', entry[0])

Let's now create a function for generating frequency tables, and use it in combination with the display\_table() function.

### Instructions

1. Create a function named freq\_table() that takes in two inputs: dataset (expected to be a list of lists) and index (expected to be an integer).
   * The function should return the frequency table (as a dictionary) for any column we want. The frequencies should also be expressed as percentages.
2. Copy the display\_table() function we wrote above. Use it to display the frequency table of the columns prime\_genre, Genres, and Category. We'll analyze the resulting tables on the next part of step 4.

## **Part III: Studying analysis**

### Instructions

1. Analyze the frequency table you generated for the prime\_genre column of the App Store data set.
   * What is the most common genre? What is the runner-up?
   * What other patterns do you see?
   * What is the general impression — are most of the apps designed for practical purposes (education, shopping, utilities, productivity, lifestyle) or more for entertainment (games, photo and video, social networking, sports, music)?
   * Can you recommend an app profile for the App Store market based on this frequency table alone? If there's a large number of apps for a particular genre, does that also imply that apps of that genre generally have a large number of users?
2. Analyze the frequency table you generated for the Category and Genres column of the Google Play data set.
   * What are the most common genres?
   * What other patterns do you see?
   * Compare the patterns you see for the Google Play market with those you saw for the App Store market.
   * Can you recommend an app profile based on what you found so far? Do the frequency tables you generated reveal the most frequent app genres or what genres have the most users?

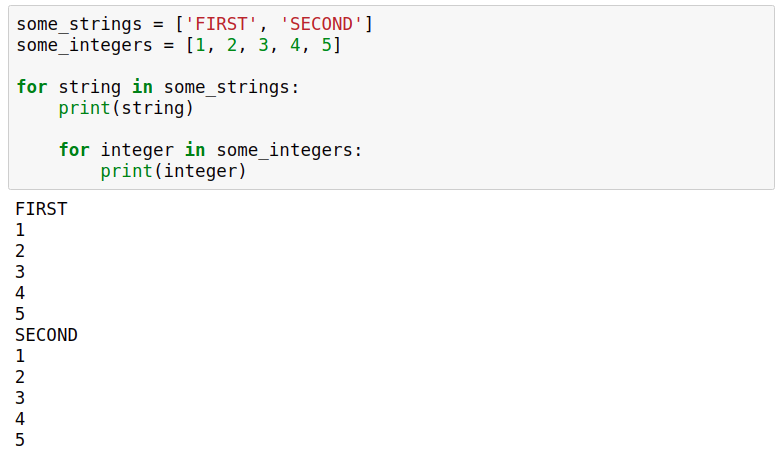
## **Part IV: Sorting by popularity (App Store)**

We found out from the tables we analyzed that the App Store is dominated by apps made for fun, while Google Play is dominated by fun but also practical apps. Now we want to know how many people install these apps, this is shown on Google Play, but not on the App Store. We can workaround the app store by using the number of user ratings instead (found on the rating\_count\_tot column)

Let's start with calculating the average number of user ratings per app genre on the App Store. To do that, we'll need to:

* Isolate the apps of each genre.
* Sum up the user ratings for the apps of that genre.
* Divide the sum by the number of apps belonging to that genre (not by the total number of apps).

To calculate the average number of user ratings for each genre, we'll use a for loop inside of another for loop, this is called a **nested loop**. This is an example of a for loop used inside another for loop:



### Instructions

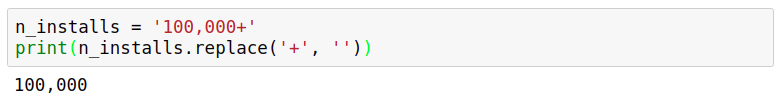
1. Start by generating a frequency table for the prime\_genre column to get the unique app genres (below, we'll need to loop over the unique genres). You can use the freq\_table() function you wrote previously on part II
2. Loop over the unique genres of the App Store data set. For each iteration (below, we'll assume that the iteration variable is named genre):
   * Initiate a variable named total with a value of 0. This variable will store the sum of the number of user ratings specific to each genre.
   * Initiate a variable named len\_genre with a value of 0. This variable will store the number of apps specific to each genre.
   * Loop over the App Store data set, and for each iteration:
     + Save the app genre to a variable named genre\_app.
     + If genre\_app is the same as genre (the iteration variable of the main loop), then:
       - Save the number of user ratings of the app as a float.
       - Add up the number of user ratings to the total variable.
       - Increment the len\_genre variable by 1.
   * Compute the average number of user ratings by dividing total by len\_genre. This should be done outside the nested loop.
   * Print the app genre and the average number of user ratings. This should also be done outside the nested loop.
3. Analyze the results and try to come up with at least one app profile recommendation for the App Store. (Basically, tell the devs what kind of app would be more profitable)

## **Part V: Sorting by popularity (Play Store)**

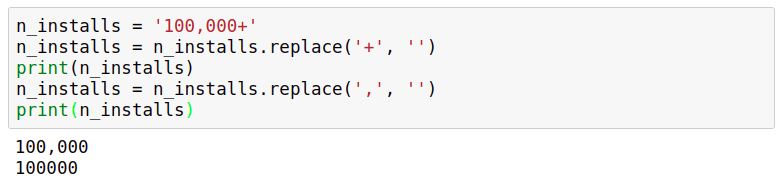
The install numbers are not precise, it has symbols and commas, but that’s fine. The only problem is that we need to convert each install number from string to float. This means we need to remove the commas and the plus characters, otherwise the conversion will fail.

To remove characters from strings, we can use [str.replace(old, new)](_blank) method. str.replace() takes in two parameters, old and new, and replaces all occurrences of old within a string with new:

To remove certain characters, we can replace them with the empty string '':



Note that we'll need to reassign to n\_installs if we want our changes saved:



Now let's calculate the average number of installs per app genre for the Google Play data set. We'll need to use a nested loop, just like in the previous screen.

### Instructions

1. Do the exact same thing as before but for google play, literally just copy paste and change variable names and change the index.
2. Come up with an app profile recommendation for Google Play. Remember, your aim is to recommend an app genre that shows potential for being profitable on both the App Store and Google Play.

# CONCLUSION

That’s it, you did it, wooo, celebration, that’s all, submit this to your professor and good job. Remember that you need to share and explain what app do you suggest the devs to make.

When in doubt of your project, remember to visit the solution notebook (and also, that your profile recommendations from Step 4 Part IV and V are completely up to you, doesn’t have to be the same as the solution notebook.)

Revise your work, **submit as proj03.ipynb**, remember to have your name and the project name on the first cell, and that’s all.

# ORIGINAL CONCLUSION

In this project, we went through a complete data science workflow:

* We started by clarifying the goal of our project.
* We collected relevant data.
* We cleaned the data to prepare it for analysis.
* We analyzed the cleaned data.

In the solution notebook, we concluded that taking a very popular book (perhaps a more recent book) and turning it into an app could be profitable for both the Google Play and the App Store market. The markets are already full of libraries, so we need to add some special features besides the raw version of the book. This might include daily quotes from the book, an audio version of the book, quizzes on the book, a forum where people can discuss the book, etc. You might have reached a different conclusion, which is perfectly fine as long as you managed to build a data-driven argumentation for your recommendation.

Looking for feedback on your project? Or maybe you'd like to show it off? Head over to our Community to [share your finished Guided Project](https://community.dataquest.io/tag/350-14)! You can also view your peers' completed notebooks to see how they approached the project.

These are a few next steps you could take:

* Analyze the frequency table for the Genre column of the Google Play data set, and see whether you can find useful patterns.
* Assume we could also make revenue via in-app purchases and subscriptions, and try to find out which genres seem to be liked the most by users — you could examine app ratings here.
* Refine your project using our data science project [style guide](https://www.dataquest.io/blog/data-science-project-style-guide/).

If you're going to work on the next steps above independently, you'll almost inevitably face some problems like not knowing how to fix an error, or not knowing what code to write to perform a certain task. In situations like these, the best thing to do is to start with a Google search (or any other search engine). In most situations, there will always be people who already ran into the same kind of problem, and you'll be able to piggyback on the solution they came up with.

As you search for solutions to your problems, you'll notice that one particular site will constantly show up in the first few results of your query — [Stack Overflow](https://stackoverflow.com/). The community on Stack Overflow is very active, and the answers you'll find there are almost always accurate and up-to-date. One important tip when you're searching on Google is to start with the word "python". For instance, if you want to find out how to remove the characters from a string, search for "**python** how to remove a character from a string" (not just "how to remove a character from a string") — otherwise you'll most likely get results for other programming languages.