

# Springboard Apps project - Tier 3 - Complete

Welcome to the Apps project! To give you a taste of your future career, we're going to walk through exactly the kind of notebook that you'd write as a data scientist. In the process, we'll be sure to signpost the general framework for our investigation - the Data Science Pipeline - as well as give reasons for why we're doing what we're doing. We're also going to apply some of the skills and knowledge you've built up in the previous unit when reading Professor Spiegelhalter's *The Art of Statistics* (hereinafter *AoS*).

So let's get cracking!

## Brief

Did Apple Store apps receive better reviews than Google Play apps?

## Stages of the project

1. Sourcing and loading
  - Load the two datasets
  - Pick the columns that we are going to work with
  - Subsetting the data on this basis
2. Cleaning, transforming and visualizing
  - Check the data types and fix them
  - Add a `platform` column to both the `Apple` and the `Google` dataframes
  - Changing the column names to prepare for a join
  - Join the two data sets
  - Eliminate the `NaN` values
  - Filter only those apps that have been reviewed at least once
  - Summarize the data visually and analytically (by the column `platform`)
3. Modelling
  - Hypothesis formulation
  - Getting the distribution of the data
  - Permutation test
4. Evaluating and concluding
  - What is our conclusion?
  - What is our decision?
  - Other models we could have used.

## Importing the libraries

In this case we are going to import `pandas`, `numpy`, `scipy`, `random` and `matplotlib.pyplot`

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# scipy is a library for statistical tests and visualizations
from scipy import stats
# random enables us to generate random numbers
import random
```

## Stage 1 - Sourcing and loading data

### 1a. Source and load the data

Let's download the data from Kaggle. Kaggle is a fantastic resource: a kind of social medium for data scientists, it boasts projects, datasets and news on the freshest libraries and technologies all in one place. The data from the Apple Store can be found [here](https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps) (<https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps>) and the data from Google Store can be found [here](https://www.kaggle.com/lava18/google-play-store-apps) (<https://www.kaggle.com/lava18/google-play-store-apps>). Download the datasets and save them in your working directory.

```
In [68]: # Now that the files are saved, we want to load them into Python using read_csv
path = "/Users/jasonzhou/Documents/GitHub/Portfolio/SpringBoard/Exercises/Apps Project/googleplaystore.csv"

# Create a variable called google, and store in it the path of the csv file
# If your dataset is in the same folder as this notebook, the path will simplify
google = 'Apps Project'

# Read the csv file into a data frame called Google using the read_csv() pandas method
Google = pd.DataFrame(pd.read_csv(googleplaystore.csv))

# Using the head() pandas method, observe the first three entries.
Google.head()
```

Out[68]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art and Design
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Design
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art and Design
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art and Design
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Design

```
In [69]: # Create a variable called apple, and store in it the path of the csv file
apple = 'Apps Project'

# Read the csv file into a pandas DataFrame object called Apple.
Apple = pd.DataFrame(pd.read_csv("AppleStore.csv"))

# Observe the first three entries like you did with your other data.
Apple.head()
```

Out[69]:

	Unnamed: 0	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ve
0	1	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	21
1	2	281796108	Evernote - stay organized	158578688	USD	0.00	161065	21
2	3	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.00	188583	282
3	4	282614216	eBay: Best App to Buy, Sell, Save! Online Shop...	128512000	USD	0.00	262241	64
4	5	282935706	Bible	92774400	USD	0.00	985920	532

## 1b. Pick the columns we'll work with

From the documentation of these datasets, we can infer that the most appropriate columns to answer the brief are:

1. Google:
  - Category # Do we need this?
  - Rating
  - Reviews
  - Price (maybe)
2. Apple:
  - prime\_genre # Do we need this?
  - user\_rating
  - rating\_count\_tot
  - price (maybe)

## 1c. Subsetting accordingly

Let's select only those columns that we want to work with from both datasets. We'll overwrite the subsets in the original variables.

```
In [70]: # Subset our DataFrame object Google by selecting just the variables ['Category', 'Rating', 'Reviews', 'Price']
Google = Google[['Category', 'Rating', 'Reviews', 'Price']]

# Check the first three entries
Google.head(3)
```

Out[70]:

	Category	Rating	Reviews	Price
0	ART_AND_DESIGN	4.1	159	0
1	ART_AND_DESIGN	3.9	967	0
2	ART_AND_DESIGN	4.7	87510	0

```
In [71]: # Do the same with our Apple object, selecting just the variables ['prime_genre', 'user_rating', 'rating_count_tot', 'price']
Apple = Apple[['prime_genre', 'user_rating', 'rating_count_tot', 'price']]

# Let's check the first three entries
Apple.head(3)
```

Out[71]:

	prime_genre	user_rating	rating_count_tot	price
0	Games	4.0	21292	3.99
1	Productivity	4.0	161065	0.00
2	Weather	3.5	188583	0.00

## Stage 2 - Cleaning, transforming and visualizing

### 2a. Check the data types for both Apple and Google, and fix them

Types are crucial for data science in Python. Let's determine whether the variables we selected in the previous section belong to the types they should do, or whether there are any errors here.

```
In [72]: # Using the dtypes feature of pandas DataFrame objects, check out the data
# Are they what you expect?
Apple.dtypes
```

```
Out[72]: prime_genre      object
user_rating      float64
rating_count_tot    int64
price            float64
dtype: object
```

This is looking healthy. But what about our Google data frame?

```
In [73]: # Using the same dtypes feature, check out the data types of our Google data
Google.dtypes
```

```
Out[73]: Category      object
Rating      float64
Reviews     object
Price       object
dtype: object
```

Weird. The data type for the column 'Price' is 'object', not a numeric data type like a float or an integer. Let's investigate the unique values of this column.

```
In [74]: # Use the unique() pandas method on the Price column to check its unique values
Google['Price'].unique()
```

```
Out[74]: array(['0', '$4.99', '$3.99', '$6.99', '$1.49', '$2.99', '$7.99', '$5.99',
 '$3.49', '$1.99', '$9.99', '$7.49', '$0.99', '$9.00', '$5.49',
 '$10.00', '$24.99', '$11.99', '$79.99', '$16.99', '$14.99',
 '$1.00', '$29.99', '$12.99', '$2.49', '$10.99', '$1.50', '$19.99',
 '$15.99', '$33.99', '$74.99', '$39.99', '$3.95', '$4.49', '$1.70',
 '$8.99', '$2.00', '$3.88', '$25.99', '$399.99', '$17.99',
 '$400.00', '$3.02', '$1.76', '$4.84', '$4.77', '$1.61', '$2.50',
 '$1.59', '$6.49', '$1.29', '$5.00', '$13.99', '$299.99', '$379.99',
 '$37.99', '$18.99', '$389.99', '$19.90', '$8.49', '$1.75',
 '$14.00', '$4.85', '$46.99', '$109.99', '$154.99', '$3.08',
 '$2.59', '$4.80', '$1.96', '$19.40', '$3.90', '$4.59', '$15.46',
 '$3.04', '$4.29', '$2.60', '$3.28', '$4.60', '$28.99', '$2.95',
 '$2.90', '$1.97', '$200.00', '$89.99', '$2.56', '$30.99', '$3.61',
 '$394.99', '$1.26', 'Everyone', '$1.20', '$1.04'], dtype=object)
```

Aha! Fascinating. There are actually two issues here.

- Firstly, there's a price called `Everyone`. That is a massive mistake!
- Secondly, there are dollar symbols everywhere!

Let's address the first issue first. Let's check the datapoints that have the price value `Everyone`

```
In [75]: # Let's check which data points have the value 'Everyone' for the 'Price' column
# Subset the Google dataframe on the price column.
# To be sure: you want to pick out just those rows whose value for the 'Price' column is 'Everyone'
Google['Price'][Google['Price'] == 'Everyone']
```

```
Out[75]: 10472    Everyone
Name: Price, dtype: object
```

Thankfully, it's just one row. We've gotta get rid of it.

```
In [76]: # Let's eliminate that row.

# Subset our Google dataframe to pick out just those rows whose value for t
# Reassign that subset to the Google variable.
# You can do this in two lines or one. Your choice!
Google = Google[Google['Price'] != 'Everyone']

# Check again the unique values of Google
Google['Price'].unique()
```

```
Out[76]: array(['0', '$4.99', '$3.99', '$6.99', '$1.49', '$2.99', '$7.99', '$5.99',
 '$3.49', '$1.99', '$9.99', '$7.49', '$0.99', '$9.00', '$5.49',
 '$10.00', '$24.99', '$11.99', '$79.99', '$16.99', '$14.99',
 '$1.00', '$29.99', '$12.99', '$2.49', '$10.99', '$1.50', '$19.99',
 '$15.99', '$33.99', '$74.99', '$39.99', '$3.95', '$4.49', '$1.70',
 '$8.99', '$2.00', '$3.88', '$25.99', '$399.99', '$17.99',
 '$400.00', '$3.02', '$1.76', '$4.84', '$4.77', '$1.61', '$2.50',
 '$1.59', '$6.49', '$1.29', '$5.00', '$13.99', '$299.99', '$379.99',
 '$37.99', '$18.99', '$389.99', '$19.90', '$8.49', '$1.75',
 '$14.00', '$4.85', '$46.99', '$109.99', '$154.99', '$3.08',
 '$2.59', '$4.80', '$1.96', '$19.40', '$3.90', '$4.59', '$15.46',
 '$3.04', '$4.29', '$2.60', '$3.28', '$4.60', '$28.99', '$2.95',
 '$2.90', '$1.97', '$200.00', '$89.99', '$2.56', '$30.99', '$3.61',
 '$394.99', '$1.26', '$1.20', '$1.04'], dtype=object)
```

Our second problem remains: I'm seeing dollar symbols when I close my eyes! (And not in a good way).

This is a problem because Python actually considers these values strings. So we can't do mathematical and statistical operations on them until we've made them into numbers.

```
In [78]: # Let's create a variable called nosymb.
# This variable will take the Price column of Google and apply the str.replace
nosymb = Google['Price'].str.replace('$', '')

# Now we need to do two things:
# i. Make the values in the nosymb variable numeric using the to_numeric()
# ii. Assign this new set of numeric, dollar-sign-less values to Google['Price']
# You can do this in one line if you wish.
Google['Price'] = pd.to_numeric(nosymb)
```

Now let's check the data types for our Google dataframe again, to verify that the 'Price' column really is numeric now.

```
In [79]: # Use the function dtypes.  
Google.dtypes
```

```
Out[79]: Category      object  
Rating      float64  
Reviews      object  
Price      float64  
dtype: object
```

Notice that the column `Reviews` is still an object column. We actually need this column to be a numeric column, too.

```
In [80]: # Convert the 'Reviews' column to a numeric data type.  
Google['Reviews'] = pd.to_numeric(Google['Reviews'])
```

```
In [81]: # Let's check the data types of Google again  
Google.dtypes
```

```
Out[81]: Category      object  
Rating      float64  
Reviews      int64  
Price      float64  
dtype: object
```

## 2b. Add a `platform` column to both the `Apple` and the `Google` dataframes

Let's add a new column to both dataframe objects called `platform`: all of its values in the `Google` dataframe will be just `'google'`, and all of its values for the `Apple` dataframe will be just `'apple'`.

The reason we're making this column is so that we can ultimately join our `Apple` and `Google` data together, and actually test out some hypotheses to solve the problem in our brief.

```
In [82]: # Create a column called 'platform' in both the Apple and Google dataframes  
# Add the value 'apple' and the value 'google' as appropriate.  
  
Apple.insert(loc=4, column='platform', value='apple')  
Google.insert(loc=4, column='platform', value='google')
```

## 2c. Changing the column names to prepare for our join of the two datasets

Since the easiest way to join two datasets is if they have both:

- the same number of columns
- the same column names we need to rename the columns of `Apple` so that they're the same as the ones of `Google`, or vice versa.

In this case, we're going to change the `Apple` columns names to the names of the `Google` columns.



This is an important step to unify the two datasets!

```
In [83]: # Create a variable called old_names where you'll store the column names of
# Use the feature .columns.
old_names = Apple.columns
print(old_names)

# Create a variable called new_names where you'll store the column names of
new_names = Google.columns
print(new_names)

# Use the rename() DataFrame method to change the columns names.
for i in range(4):
    Apple = Apple.rename(columns = {str(old_names[i]):str(new_names[i])})

Index(['prime_genre', 'user_rating', 'rating_count_tot', 'price', 'platform'], dtype='object')
Index(['Category', 'Rating', 'Reviews', 'Price', 'platform'], dtype='object')
```

## 2d. Join the two datasets

Let's combine the two datasets into a single data frame called df .

```
In [84]: # Let's use the append() method to append Apple to Google.
df = Google.append(Apple)

# Using the sample() method with the number 12 passed to it, check 12 random
df.sample(12)
```

Out[84]:

	Category	Rating	Reviews	Price	platform
450	Entertainment	3.0	5	2.99	apple
5977	Weather	4.5	14	0.00	apple
2896	PHOTOGRAPHY	4.0	157506	0.00	google
269	Photo & Video	3.5	12557	2.99	apple
5280	PERSONALIZATION	4.5	37	0.00	google
2176	FAMILY	4.7	185	4.99	google
1097	FINANCE	4.4	126431	0.00	google
1609	LIFESTYLE	4.6	69177	0.00	google
4935	TOOLS	2.2	32	0.00	google
64	AUTO_AND_VEHICLES	4.9	6090	0.00	google
5008	Games	3.5	166	2.99	apple
1683	GAME	4.6	2157930	0.00	google

## 2e. Eliminate the NaN values

As you can see there are some `NaN` values. We want to eliminate all these `NaN` values from the table.

```
In [85]: # Lets check first the dimesions of df before dropping `NaN` values. Use the
print(df.shape)

# Use the dropna() method to eliminate all the NaN values, and overwrite th
df = df.dropna()

# Check the new dimesions of our dataframe.
df.shape

(18037, 5)
```

Out[85]: (16563, 5)

## 2f. Filter the data so that we only see whose apps that have been reviewed at least once

Apps that haven't been reviewed yet can't help us solve our brief.

So let's check to see if any apps have no reviews at all.

```
In [87]: # Subset your df to pick out just those rows whose value for 'Reviews' is e
# Do a count() on the result.
df['Reviews'][df['Reviews'] == 0].count()
```

Out[87]: 929

929 apps do not have reviews, we need to eliminate these points!

```
In [88]: # Eliminate the points that have 0 reviews.
df.drop(df[df['Reviews'] == 0].index, inplace=True)

# New size should be 16563 - 929 = 15634
df.shape
```

Out[88]: (14830, 5)

## 2g. Summarize the data visually and analytically (by the column platform)

What we need to solve our brief is a summary of the `Rating` column, but separated by the different platforms.

```
In [89]: # To summarize analytically, let's use the groupby() method on our df.
df.groupby(['platform']).mean()
```

Out[89]:

	Rating	Reviews	Price
platform			
apple	4.049697	14803.805520	1.821977
google	4.191357	519211.371642	0.965035

Interesting! Our means of 4.049697 and 4.191757 don't **seem** all that different! Perhaps we've solved our brief already: there's no significant difference between Google Play app reviews and Apple Store app reviews. We have an **observed difference** here: which is simply  $(4.191757 - 4.049697) = 0.14206$ . This is just the actual difference that we observed between the mean rating for apps from Google Play, and the mean rating for apps from the Apple Store. Let's look at how we're going to use this observed difference to solve our problem using a statistical test.

#### Outline of our method:

1. We'll assume that platform (i.e, whether the app was Google or Apple) really doesn't impact on ratings.
2. Given this assumption, we should actually be able to get a difference in mean rating for Apple apps and mean rating for Google apps that's pretty similar to the one we actually got (0.14206) just by: a. shuffling the ratings column, b. keeping the platform column the same, c. calculating the difference between the mean rating for Apple and the mean rating for Google.
3. We can make the shuffle more useful by doing it many times, each time calculating the mean rating for Apple apps and the mean rating for Google apps, and the difference between these means.
4. We can then take the mean of all these differences, and this will be called our permutation difference. This permutation difference will be great indicator of what the difference would be if our initial assumption were true and platform really doesn't impact on ratings.
5. Now we do a comparison. If the observed difference looks just like the permutation difference, then we stick with the claim that actually, platform doesn't impact on ratings. If instead, however, the permutation difference differs significantly from the observed difference, we'll conclude: something's going on; the platform does in fact impact on ratings.
6. As for what the definition of *significantly* is, we'll get to that. But there's a brief summary of what we're going to do. Exciting!

If you want to look more deeply at the statistics behind this project, check out [this resource](https://www.springboard.com/archeo/download/4ea4d453b0b84014bcef287c50f47f00/) (https://www.springboard.com/archeo/download/4ea4d453b0b84014bcef287c50f47f00/).

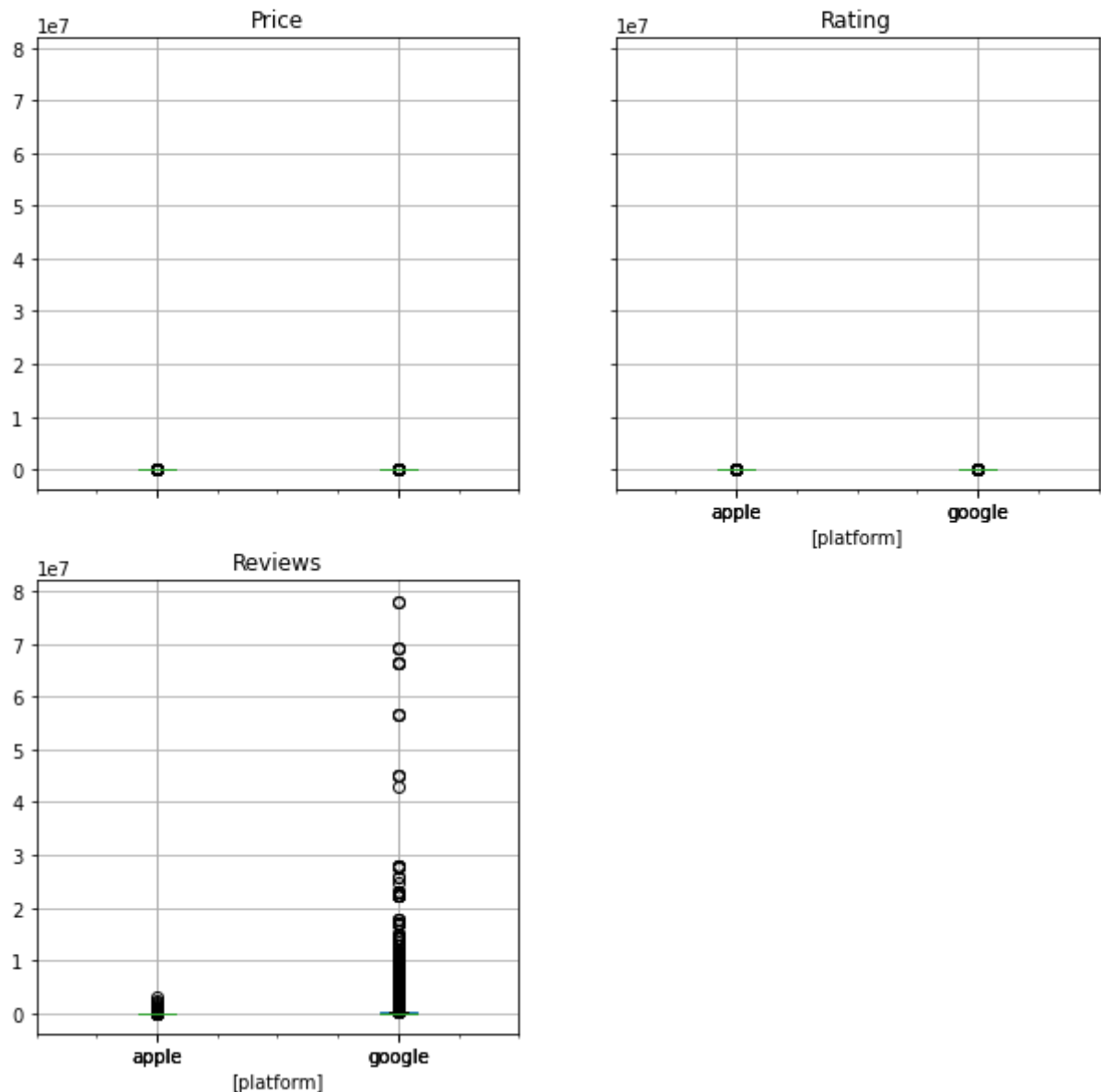
Let's also get a **visual summary** of the `Rating` column, separated by the different platforms.

A good tool to use here is the boxplot!

```
In [91]: # Call the boxplot() method on our df.
df.boxplot(by='platform',figsize = (10,10))
```

```
Out[91]: array([[<AxesSubplot:title={'center':'Price'}, xlabel='[platform] '>,
  <AxesSubplot:title={'center':'Rating'}, xlabel='[platform] '>],
  [<AxesSubplot:title={'center':'Reviews'}, xlabel='[platform] '>,
  <AxesSubplot:>]], dtype=object)
```

Boxplot grouped by platform



Here we see the same information as in the analytical summary, but with a boxplot. Can you see how the boxplot is working here? If you need to revise your boxplots, check out this [link](https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps) (<https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps>).

## Stage 3 - Modelling

### 3a. Hypothesis formulation

Our **Null hypothesis** is just:

**H<sub>null</sub>**: the observed difference in the mean rating of Apple Store and Google Play apps is due to chance (and thus not due to the platform).

The more interesting hypothesis is called the **Alternate hypothesis**:

**H<sub>alternative</sub>**: the observed difference in the average ratings of apple and google users is not due to chance (and is actually due to platform)

We're also going to pick a **significance level** of 0.05.

### 3b. Getting the distribution of the data

Now that the hypotheses and significance level are defined, we can select a statistical test to determine which hypothesis to accept.

There are many different statistical tests, all with different assumptions. You'll generate an excellent judgement about when to use which statistical tests over the Data Science Career Track course. But in general, one of the most important things to determine is the **distribution of the data**.

```
In [92]: # Create a subset of the column 'Rating' by the different platforms.
# Call the subsets 'apple' and 'google'
apple = df['Rating'][df['platform'] == 'apple']
google = df['Rating'][df['platform'] == 'google']
```

```
In [94]: # Using the stats.normaltest() method, get an indication of whether the app
# Save the result in a variable called apple_normal, and print it out
apple_normal = stats.normaltest(apple)
print(apple_normal)
```

```
NormaltestResult(statistic=1778.9974234584017, pvalue=0.0)
```

```
In [95]: # Do the same with the google data.
google_normal = stats.normaltest(google)
print(google_normal)
```

```
NormaltestResult(statistic=3397.6258711648793, pvalue=0.0)
```

Since the null hypothesis of the `normaltest()` is that the data are normally distributed, the lower the p-value in the result of this test, the more likely the data are to be non-normal.

Since the p-values is 0 for both tests, regardless of what we pick for the significance level, our conclusion is that the data are not normally distributed.

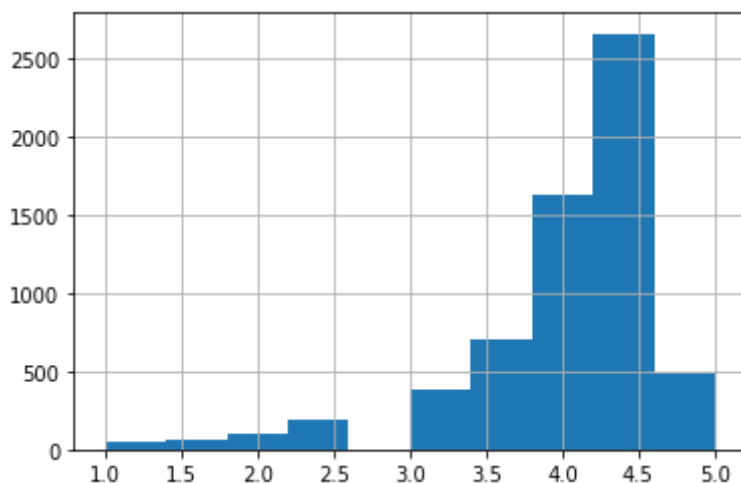
We can actually also check out the distribution of the data visually with a histogram. A normal distribution has the following visual characteristics:

- symmetric
- unimodal (one hump)

As well as a roughly identical mean, median and mode.

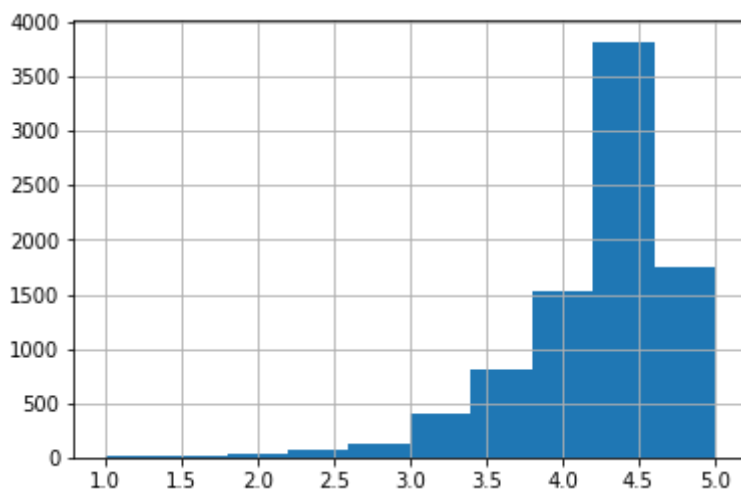
```
In [97]: # Create a histogram of the apple reviews distribution
apple.hist()
```

```
Out[97]: <AxesSubplot:>
```



```
In [98]: # Create a histogram of the google data
google.hist()
```

```
Out[98]: <AxesSubplot:>
```



### 3c. Permutation test

Since the data aren't normally distributed, we're using a *non-parametric* test here. This is simply a label for statistical tests used when the data aren't normally distributed. These tests are extraordinarily powerful due to how few assumptions we need to make.

Check out more about permutations [here](http://rasbt.github.io/mlxtend/user_guide/evaluate/permutation_test/).  
([http://rasbt.github.io/mlxtend/user\\_guide/evaluate/permutation\\_test/](http://rasbt.github.io/mlxtend/user_guide/evaluate/permutation_test/))

```
In [103]: # Create a column called `Permutation1`, and assign to it the result of per
# This assignment will use our numpy object's random.permutation() method
df['Permutation1'] = np.random.permutation(df['Rating'])

# Call the describe() method on our permutation grouped by 'platform'.
df.groupby(['platform']).mean().describe()
```

Out[103]:

	Rating	Reviews	Price	Permutation1
<b>count</b>	2.000000	2.000000	2.000000	2.000000
<b>mean</b>	4.120527	267007.588581	1.393506	4.131746
<b>std</b>	0.100169	356670.010487	0.605949	0.002396
<b>min</b>	4.049697	14803.805520	0.965035	4.130051
<b>25%</b>	4.085112	140905.697051	1.179270	4.130898
<b>50%</b>	4.120527	267007.588581	1.393506	4.131746
<b>75%</b>	4.155942	393109.480112	1.607741	4.132593
<b>max</b>	4.191357	519211.371642	1.821977	4.133440

```
In [104]: # Lets compare with the previous analytical summary:
df.groupby(['platform']).mean()
```

Out[104]:

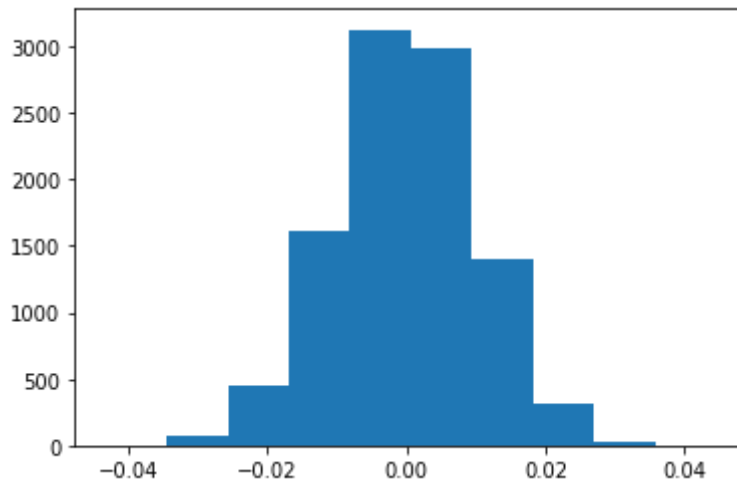
	Rating	Reviews	Price	Permutation1
<b>platform</b>				
<b>apple</b>	4.049697	14803.805520	1.821977	4.133440
<b>google</b>	4.191357	519211.371642	0.965035	4.130051

```
In [108]: # The difference in the means for Permutation1 (0.001103) now looks hugely
# It's sure starting to look like our observed difference is significant, a
# But to be sure, let's create 10,000 permutations, calculate the mean rati
# Let's create a vector with the differences - that will be the distribution

# First, make a list called difference.
difference = [0] * 10000
# Now make a for loop that does the following 10,000 times:
# 1. makes a permutation of the 'Rating' as you did above
# 2. calculates the difference in the mean rating for apple and the mean ra

for i in range(10000):
    df['Permutation1'] = np.random.permutation(df['Rating'])
    difference[i] = df.groupby(['platform']).mean()['Permutation1']['apple'
```

```
In [109]: # Make a variable called 'histo', and assign to it the result of plotting a
histo = plt.hist(difference)
```



```
In [110]: # Now make a variable called obs_difference, and assign it the result of the
obs_difference = apple.mean() - google.mean()

# Make this difference absolute with the built-in abs() function.
obs_difference = abs(obs_difference)

# Print out this value; it should be 0.1420605474512291.
print(obs_difference)

# I think my error is due to the part where I drop the rows where reviews =
0.14166028653643092
```

## Stage 4 - Evaluating and concluding

### 4a. What is our conclusion?

```
In [0]: '''
What do we know?

Recall: The p-value of our observed data is just the proportion of the data
As a result, we're going to count how many of the differences in our differ

If less than or equal to 5% of them are, then we will reject the Null.
'''
-- -- --
```



## 4b. What is our decision?

So actually, zero differences are at least as extreme as our observed difference!

So the p-value of our observed data is 0.

It doesn't matter which significance level we pick; our observed data is statistically significant, and we reject the Null.

We conclude that platform does impact on ratings. Specifically, we should advise our client to integrate **only Google Play** into their operating system interface.

## 4c. Other statistical tests, and next steps

The test we used here is the Permutation test. This was appropriate because our data were not normally distributed!

As we've seen in Professor Spiegelhalter's book, there are actually many different statistical tests, all with different assumptions. How many of these different statistical tests can you remember? How much do you remember about what the appropriate conditions are under which to use them?

Make a note of your answers to these questions, and discuss them with your mentor at your next call.