Woolworths vs Coles

I wanted to compare the prices of groceries at Woolworths and Coles. The motivation for the project was simply to see if I could save money on groceries. To compare prices I would have to scrape product data from the web. Since Woolworths and Coles both have thousands and thousands of products it would be very time consuming to scrape their entire catalogues and the best way to compare prices would be to get a sample and use statistical tools to test hypotheses about the population.

Both Woolworths and Coles websites render their products dynamically using Javascript which means a library like urllib won't work. I chose to use Selenium which is a very powerful library that can be used for web scraping, bots and automation. Selenium lets you inject Javascript into a web page, extract data and return it to your Python program. Selenium opens a web browser on the device it is running on using a 'web driver' executable. There is no way for a web site to tell the difference between Selenium and a normal user. Injecting Javascript is very powerful for example you can inject Javascript code to fill out user credentials and click a log in button.

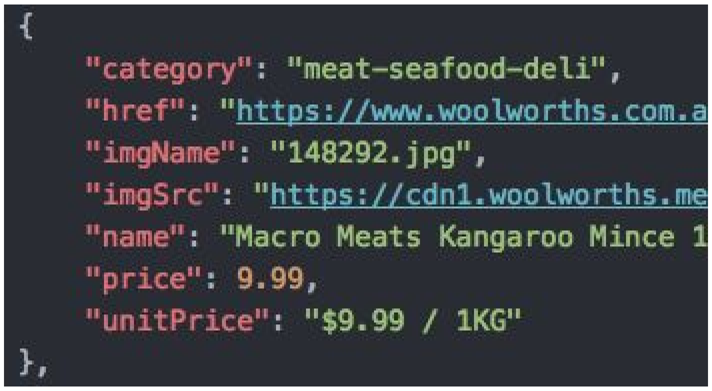
I used quite a limited subset of Javascript and if you are familiar with Java or any C style language most of the constructs should be very familiar to you. The main functions I used to search for data within HTML are listed below. The last three functions always return an array which may be empty if no matching HTML elements were found.

- getElementById('id') which searches for an HTML element with a specific ID
- getElementsByClassName('class') which searches for an HTML element with a specific class
- querySelectorAll('[attribute="value"]') which searches for an HTML element with a specific attribute and value pair
- getElementsByTagName('tag') which searches for an HTML element with a specific tag

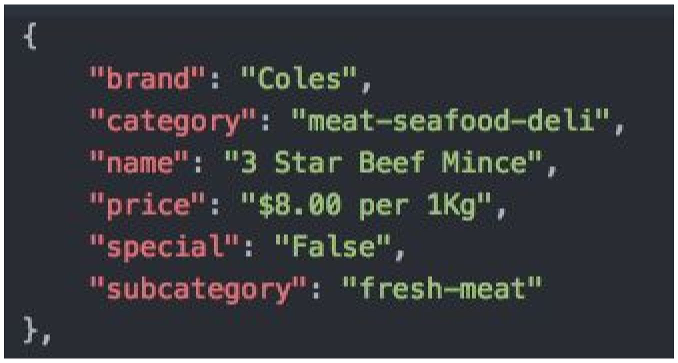
The main Python programs which scrape data from Woolworths and Coles, *scrape_woolworths.py* and *scrape_coles.py*, inject Javascript to extract product data and then save the information in JSON encoded files. The main Python program which analyses the data and visualises the results is *process.py*, this uses the JSON files.

When writing the web scraper I made it collect as much information as possible since it is very time consuming to run. In the end I believe I only ended up using the product name, unit price and whether the product was on special.

Below is an example of the JSON for a Woolworths product (left) and a Coles product (right).



```
{
    "category": "meat-seafood-deli",
    "href": "https://www.woolworths.com.a
    "imgName": "148292.jpg",
    "imgSrc": "https://cdn1.woolworths.me
    "name": "Macro Meats Kangaroo Mince 1
    "price": 9.99,
    "unitPrice": "$9.99 / 1KG"
},
```

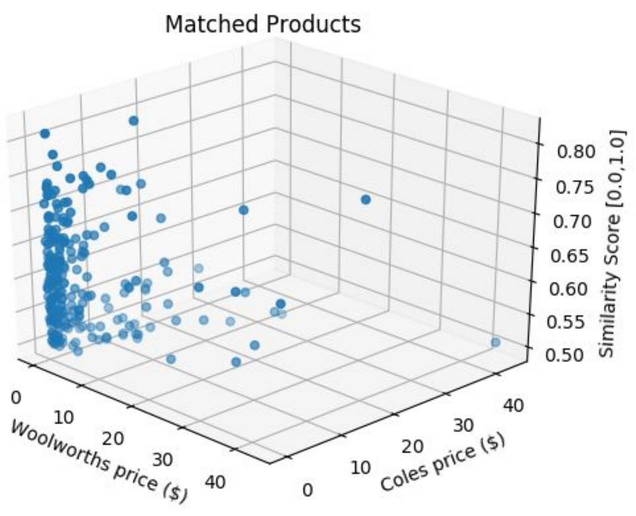


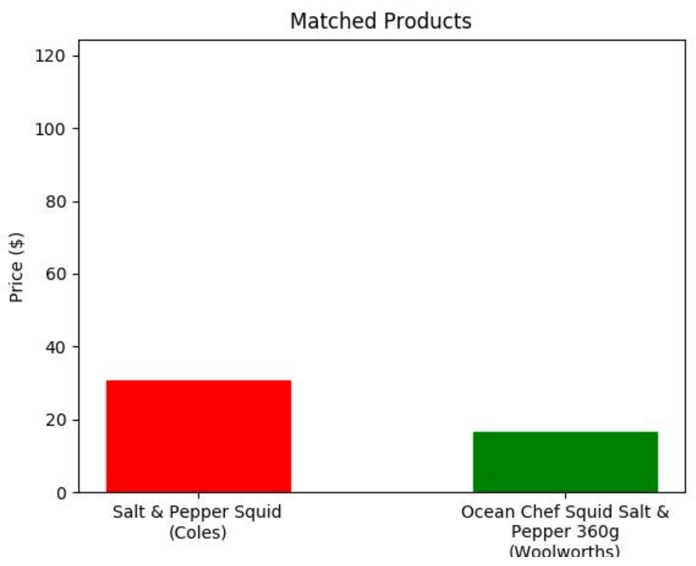
```
{
    "brand": "Coles",
    "category": "meat-seafood-deli",
    "name": "3 Star Beef Mince",
    "price": "$8.00 per 1Kg",
    "special": "False",
    "subcategory": "fresh-meat"
},
```

Once I had the data I needed to find 'similar' products to compare prices. To do this I found products with similar names using scikit-learn's text similarity features. Specifically scikit-learn has TFIDF, term frequency inverse document frequency, function which gives a similarity score for two documents by taking into account the number of terms they share in common and how common those words are in the overall corpus. The actual code I had to write for this was simple and quite short since almost everything is taken care of by scikit-learn, I only needed to do some preprocessing to get the data into the right shape.

The similarity score is between 0 and 1 where higher is more similar. Through trial and error I decided to use a threshold of 0.5, so I compare prices of products with a similarity greater than 0.5. Out of about 2000 products from both Woolworths and Coles this results in about 250 matches. The threshold is a parameter which can be easily adjusted.

In terms of more advanced visualisations I produced a 3D scatter plot and animated barplot. The 3D plot displays the price of matched products at Woolworths and Coles and their similarity score. When the similarity score is lower there seems to be more variance in the prices between Woolworths and Coles. The animated barplot iterates through all the matched products displaying the names and prices.





Analysis and Results

I did not know whether to expect Coles or Woolworths to be cheaper. I used the textbook OpenIntro Statistics, used in the course STAT1003, as a reference which I have included an excerpt from below.

"Two sets of observations are paired if each observation in one set has a special correspondence or connection with exactly one observation in the other data set… To analyze paired data, it is often useful to look at the difference in outcomes of each pair of observations... Using differences between paired observations is a common and useful way to analyze paired data... To analyze a paired data set, we simply analyze the differences. We can use the same t-distribution techniques we applied in the last section." - OpenIntro Statistics, Chapter 5.2, Page 228

The prices of matched products can be considered paired observations. For every matched product I have a pair of prices, one for Woolworths and for Coles. To analyse the prices I analyse the differences, the Woolworths price minus the Coles price. Specifically I performed a t-distribution hypothesis test using the scipy.stats.ttest_1samp() function.

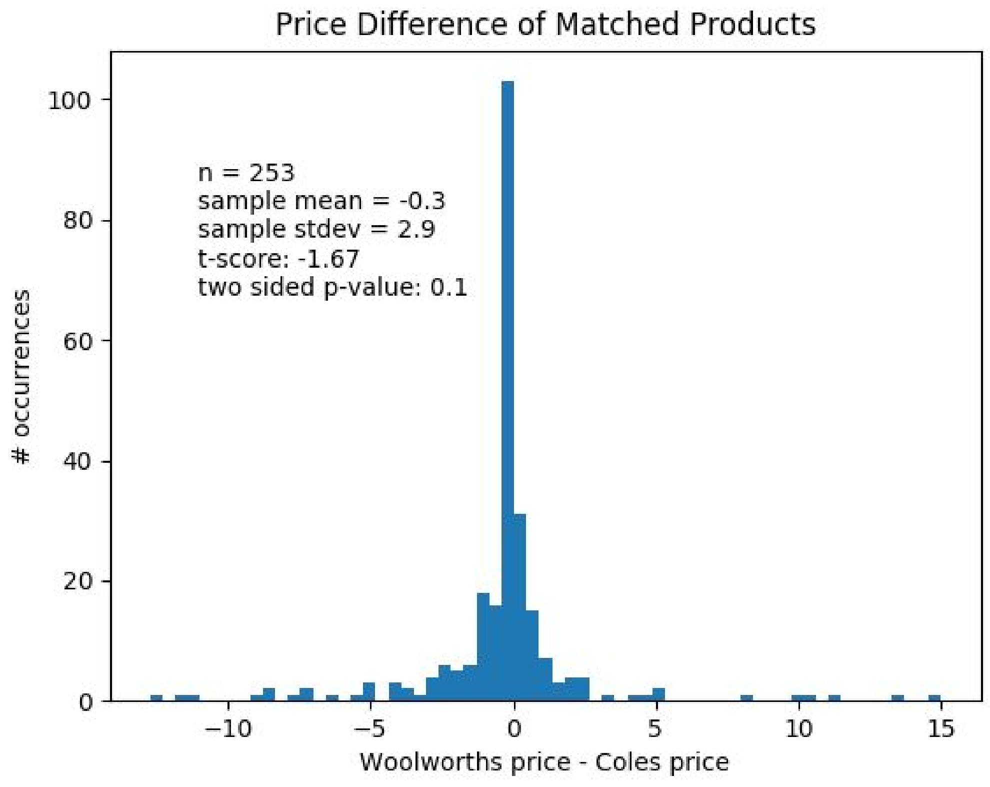
A t-distribution test requires the sample data to satisfy some conditions. Observations must be independent meaning they must be from a random sample of less than 10% of the population and the sample must be roughly normal or relatively large. I believe the sample of products I used satisfies both of these conditions.

The ttest_1samp() function takes the sample data as a numpy array and a population mean. It performs a test to determine the probability the actual mean of the population, i.e. the actual mean difference in price Woolworths minus Coles, is equal to the supplied population mean. I supplied a population mean of zero meaning the actual population mean difference is equal to zero or in other words there is no difference in price. This is our 'null hypothesis'.

For the sample data I used the ttest function returned a two-sided p-value of 0.96. Since the p-value is larger than the typical signifigance level used of 0.05, we fail to reject the null hypothesis. I did not find a statistically significant difference in price.

Interestingly if I had initially set out to perform a one sided test, the p-value would be half of 0.96 and I would have rejected the null hypothesis. However I think this would be considered bad practice to change from a two-sided to one-sided test after seeing the results.

Below is a histogram of the paired observations. The number of matched products is 253 which I think is less than 10% of all similar products at Coles and Woolworths. The data is not too strongly skewed either and the sample size is quite large so all of the conditions for using the t-distribution should be satisfied. I did not find a statistically significant difference in price however for this data set Woolworths was slightly cheaper. The tallest bar in the histogram below is to the left of zero meaning the Coles price was larger than the Woolworths price and the sample mean is also negative.



Extending the Project

I can think of lots of ways to extend or improve the project including:

- Improving the code in general by making it easier to read, more reusable and efficient
- Getting data from other sources e.g. Aldi or IGA
- Scraping the entire catalogue of both Coles and Woolworths which would allow new possibilities analysis
- Creating a web interface or GUI
- Given a shopping list of generic items e.g. chicken, potatoes, rice, find the cheapest shopping list of actual products from either Coles or Woolworths.
- Making use of nutrition data e.g. finding the cheapest source of protein