**Project Title: Financing Used Cars**

**By:** [**Helen Amin**](https://github.com/helenamin) **and** [**FX Foo**](https://github.com/foofx88/)

**Motivation:**

Create an educational dashboard to guide potential buyers in making the right decision when it comes to buying a car with finance. Once data is scaped it can be used to analyse the comparison loans provided by car dealer and financial institutions.

We have chosen West Side Auto as it is a local business based in WA and they have been awarded MTAWA Automotive Excellence Awards – Used Car Dealer of the Year for 2021.

Finder.com.au was chosen for the Comparison rates data as it is 100% free and privately owned. Finder is also one of the most popular financial products advisors in Australia with 2 Million visitors each month.

**Extract:**

We used Beautiful Soup to scrape the data from two of the websites. Upon inspecting the HTML elements on the site, we were able to locate the HTML tags and CSS classes required to get the data.

As there were multiple pages for the used cars in West Side Auto’s site, we had to use Webdriver Manger to inspect the site and utilised Splinter to dynamically navigate to the next page.

As for the Finder.com.au, there was a slight challenge where the information that we require (Comparison Rates) were in a JavaScript Table. We were able to quickly identify the pattern required to scrape the data.

After transforming and cleaning the data with Pandas, the data is then exported to pgAdmin 4.

**Transform:**

We utilised Pandas to work on the scraped data. As the scraped data was in a list format and of object datatype, hence there were unwanted symbols such as ‘$’ and ‘,’ in the price.

As we were scraping the data, we cleaned up the data by removing any unwanted symbol from car prices and turn it into a Float data type. This was an important step as the transformed data is then used for the calculation on the required deposit amount and the monthly repayment rates according to the car dealer and financial institutions rates.

Once that was completed, we consolidate all the data from the list to the Vehicle dictionary which then transform into a usable DataFrame (vehicle\_df). After the vehicle\_df was created, we continue to add in more usable columns such as the Deposit and loan amount column. The deposit amount was calculated from 10% of the price of each vehicle whereas the loan amount is calculated from deducting the deposit from the price.

The monthly repayment was also calculated from the loan amount:

***Monthly\_repayment = (loan\_amou******nt)\*((r)\*((1+r)\*\*n))/(((1+r)\*\*n)-1)***

r = annual comparison rate (Calculated from comparison\_rate/term)\*0.01)

This was formulated as the comparison rate scraped was not in a percentage format, hence the multiplication of 0.01

n = t\* years (Where t is the term either Monthly t = 12, Weekly t = 52)

We did the same for the comparison rates from Finder.com.au where we cleaned the data as we scrape, putting the comparison rates into usable format. After generating the DataFrame (finance\_df), we had to remove some of the financial institutions because some only provided New Car Loans. In the finance\_df, there was a combination of financial institution where some provides New and Used car loans. Therefore, we had to use 2 conditions:

***(substring in finance\_df['finance\_name']) & (not(substring\_2 in finance\_df['finance\_name']))***

Where substring = “New”, substring\_2 = “Used”

As we planned to export the DataFrame into SQL later, we’ve noticed that there are some financial institutions with symbols in their name which would return an error during the export to SQL. We then proceeded to remove the unwanted symbol from the name.

We then transpose the finance\_df as we need the financial institution names to represent columns in the final dataframe. We had to concatenate the vehicle\_df and finance\_df as they do not have any column in common for merging.

Afterwards, we reused [the formula](#Formula) to calculate the monthly repayments for all the financial institutions using their corresponding comparison rates. For the final clean up, we dropped the interest\_rate that was carried over form the finance\_df during the concatenation using .dropna()

**Load:**

We have chosen postgres (pgadmin4) as it is a relational database. The main reason that this was chosen is because it is the simplest model and can be handled with simple SQL queries. Our car loan database will be used to compare monthly payments, the data accuracy (no duplication with use of Primary Key) and simplicity of the SQL queries of the relational database is just what we needed.

There are more than 60 pages from West Side Auto when it comes to the site listing, the scalability of relational database makes it feasible for any future modifications. We can continue to add more cars into the database and archive car data that have been sold.

**Post-mortem:**

If time permits, we will be using the stock number as an input. After visiting West Side Auto and obtaining the stock number, they can input it into the dashboard then the data will be scraped and display only relevant information for the user.

Once the data is scaped and transformed, we would prefer the data to be stored within mongodb instead of SQL as it requires less storage, and it is quicker.

All project screenshots can be viewed from [GitHub](https://github.com/foofx88/ETL-Financing-Used-Cars)