ASSESSMENT 3: WEBCRAWLER AND NLP SYSTEM

MA3831

Piper Rains

Task 1: Overview

When perusing through real estate property listings, regardless of a customer's intention to either rent or buy, the task can prove to be somewhat repetitive. As more listings are viewed, it may become increasingly hard to distinguish between different properties, their unique features, and descriptions. The difficulty of identifying a property that best suits a customer is more laborious by the sheer volume of different real estate agencies and sites to search through, with an abundance of information online.

By implementing a web scraper, the tedious task of property hunting can be made easy, by presenting all the desired information of a listing in one location. A web scraper can be used to extract all property listings across selected websites and store information such as property region or city, street address, postcode, listed price, property type, features such as bedroom and bathroom counts, as well as a detailed paragraph description of the property. The retrieved data will exist in a data frame, overall allowing for easier comprehension of property options available to a customer. In contrast to listings on a website, where only minimal filtering options exist to adjust the basics such as location, price range, or property type, by storing all property information in a dataset, users can filter to their preferences unrestricted, identifying a home most suited for them. Expanded filtering options could include an ideal number of bedrooms, as well as property ratings and reviews, making for more efficient and refined search results to examine.

Natural language processing (NLP), a technique employed to make sense of natural language computationally via algorithmic and semantic approaches, can be applied to the property dataset (Chowdhury, 2005). In turn, this would allow for more enhanced processing and refining of the data to take place, including a thorough analysis of the text fields such as property description, rather than just standard numerical or keyword statistical analyses. NLP introduces methods such as named-entity recognition, summarisation, text classification, sentiment analysis, and topic modelling.

For the case of real estate data, the two NLP tasks of summarisation and sentiment analysis when implemented could enhance a customer's property search experience and efficiency. By incorporating the task of summarisation, large chunks of text on property listings such as descriptions can be condensed to only encapsulate the most important information. In turn, this would reduce the time a customer takes to view each listing, as well as provide a brief and unique overview of each listing to help distinguish them for the customer. Implementing sentiment analysis to large chunks of text such as a property description or a review field, customers can view the score of each property, generated based on how positive or negative the language being used to describe the property is. With access to such a score, customers can additionally filter through listings depending on what score threshold they are interested in, thus further reducing search time.

Task 2: WebCrawler

To further investigate the deployment of these two NLP tasks, and in turn solve the issue faced when searching real estate property listings, as previously highlighted, a web scraper first needs to be implemented to accumulate property listings data. As a prototype, only a single real estate website will be considered for scraping, the *McGrath* real estate company site, in order to reduce the computational time and size of data collection, whilst still creating a dataset suitable for prototype analysis (McGrath, 2021).

The *McGrath* real estate site was selected for web scraping because of its wide variety of real estate listings across Australia, as well due to the terms and conditions confining the data, allowing for web scraping to occur. The *McGrath* site explicitly states in its terms and conditions that "*the information … on the McGrath website is provided for general information and/or educative purposes only*", and "*visitors to the site are granted permission to … the website material for private purposes [or research uses] only*" (McGrath, 2021). Due to the private, educative nature of this investigation, web scraping of the *McGrath* site is respectfully within the limits of its copyright.

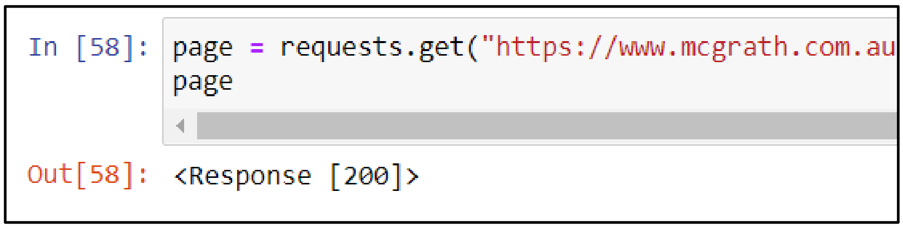
Other more popular Australian real estate sites such as *realestate.com.au* were disregarded as possible websites to scrape due to their terms and conditions, restricting the "*use any automated device, software, process or means to access, retrieve, scrape, or index our platform or any content"* (realestate.com.au, 2018). It was also due to this that no further data supplementation other than that of the *McGrath* website was collected for this prototype. In the instance where the outcome of this prototype would result in a greater study, these other real estate websites can be contacted to receive permission to web scrape data for private and educational purposes only. However, this is beyond the required scope of the prototype, and the data gathered from *McGrath* alone will be sufficient.

To extract the real estate data from the *McGrath* website, a web scraper will be coded in two parts – the extraction of URLs for property listings, and the extraction of property data from each of those property listing URLs.

The first section will focus on extracting property listing URLs from the *McGrath* website. The particular URL used has been specifically chosen as the initial URL to scrape, as it links to an already filtered property search. The applied filters confine the search results to only listings existing in the Greater Brisbane Region, QLD, allowing for property type and sale price range. The purchase type of the displayed listings (Buy, Rent or Sold) has been set as "*Sold*" to retrieve strictly prototype data, as this particular search query returns 1165 properties (at the time of this report), in comparison to "*Buy"* and "*Rent"* returning a combination of fewer than 200 results. For this prototype, over 1000 property entries are ideal, and thus, to save time when retrieving at least 1000 listings, the data will consist of sold properties (in this case, only one URL is required to be accessed/scraped from compared to multiple URLs if retrieving unsold properties) Overall, this should not cause any issues if unsold properties were to be assessed, as the HTML site layouts of both sold and unsold properties is identical, and hence data can be retrieved without any adaption to code.

To perform web scraping and future analyses, the language Python 3.7 will be used within the Jupyter notebook platform (Python 3, 2009), (Jupyter Notebooks, 2016). To construct the corpus through web scraping, the *get* command from the *Requests* library is called upon initially to be granted access to the HTML of the *McGrath* real estate site. This line of code returned the output

"*<Response [200]>*", indicating that the request was a success and web scraping can now be performed (refer to Figure 1).

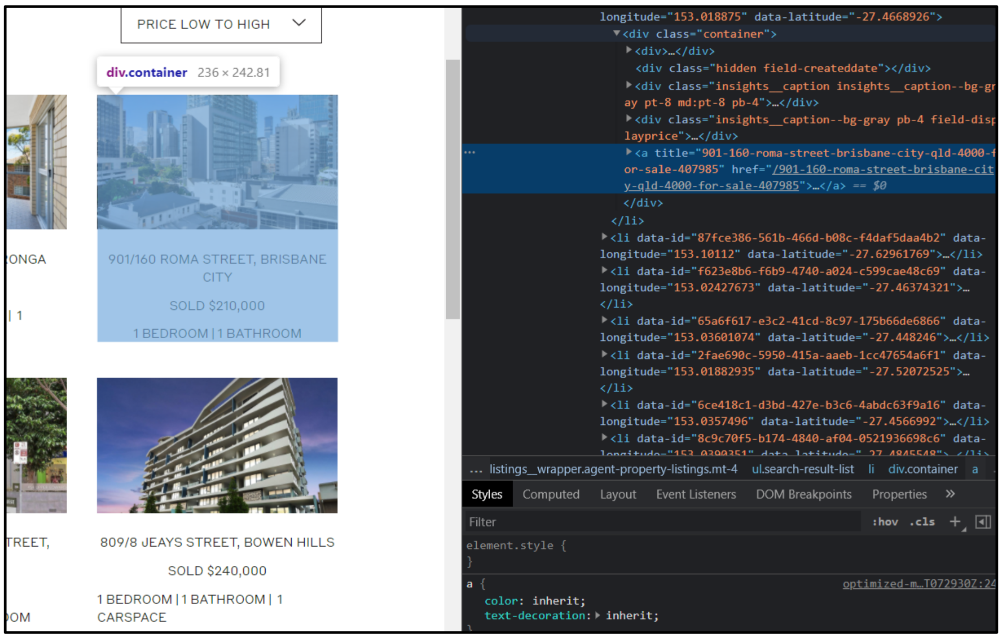


```
In [58]: page = requests.get("https://www.mcgrath.com.au
         page

Out[58]: <Response [200]>
```

Figure 1: Request for accessing URL for web scraping.

Once the site's HTML content is retrieved, the *BeautifulSoup* library is applied to parse the HTML content, in turn tidying up the layout of the content, and overall allowing for easier navigation throughout the HTML tree to retrieve all desired data. Now that the retrieved HTML from the *McGrath* site has been initialised, property listing URLs can now be located and extracted. An empty list called *urls* was created to store all retrieved URLs, and the constant variable, *head_link* was created to be attached to the beginning of each URL obtained. Investigating the HTML content, each property URL is located as a *href* object within an *<a>* tag. Each of these *<a>* tags are nested in a *<div>* tag, with a *class = container*. All property containers are nested within another *<div>* tag containing only search results (refer to Figure 2).



PRICE LOW TO HIGH
div.container 236 × 242.81
ONGA
901/160 ROMA STREET, BRISBANE CITY
SOLD $210,000
1 BEDROOM | 1 BATHROOM
TREET,
809/8 JEAYS STREET, BOWEN HILLS
SOLD $240,000
1 BEDROOM | 1 BATHROOM | 1 CARSPACE
OM
Styles Computed Layout Event Listeners DOM Breakpoints Properties
Filter
:hov .cls

```
longitude="153.018875" data-latitude="-27.4668926">
  ▼<div class="container">
    ▶<div>…</div>
     <div class="hidden field-createddate"></div>
    ▶<div class="insights__caption insights__caption--bg-gr
    ay pt-8 md:pt-8 pb-4">…</div>
    ▶<div class="insights__caption--bg-gray pb-4 field-disp
    layprice">…</div>
    ▶<a title="901-160-roma-street-brisbane-city-qld-4000-f
    or-sale-407985" href="/901-160-roma-street-brisbane-cit
    y-qld-4000-for-sale-407985">…</a> == $0
    </div>
  </li>
▶<li data-id="87fce386-561b-466d-b08c-f4daf5daa4b2" data-
longitude="153.10112" data-latitude="-27.62961769">…</li>
▶<li data-id="f623e8b6-f6b9-4740-a024-c599cae48c69" data-
longitude="153.02427673" data-latitude="-27.46374321">…
</li>
▶<li data-id="65a6f617-e3c2-41cd-8c97-175b66de6866" data-
longitude="153.03601074" data-latitude="-27.448246">…</li>
▶<li data-id="2fae690c-5950-415a-aaeb-1cc47654a6f1" data-
longitude="153.01882935" data-latitude="-27.52072525">…
</li>
▶<li data-id="6ce418c1-d3bd-427e-b3c6-4abdc63f9a16" data-
longitude="153.0357496" data-latitude="-27.4566992">…</li>
▶<li data-id="8c9c70f5-b174-4840-af04-0521936698c6" data-

… listings__wrapper.agent-property-listings.mt-4  ul.search-result-list  li  div.container  a

element.style {
}
a {                                        optimized-m…T072930Z:24
  color: inherit;
  text-decoration: ▶ inherit;
```

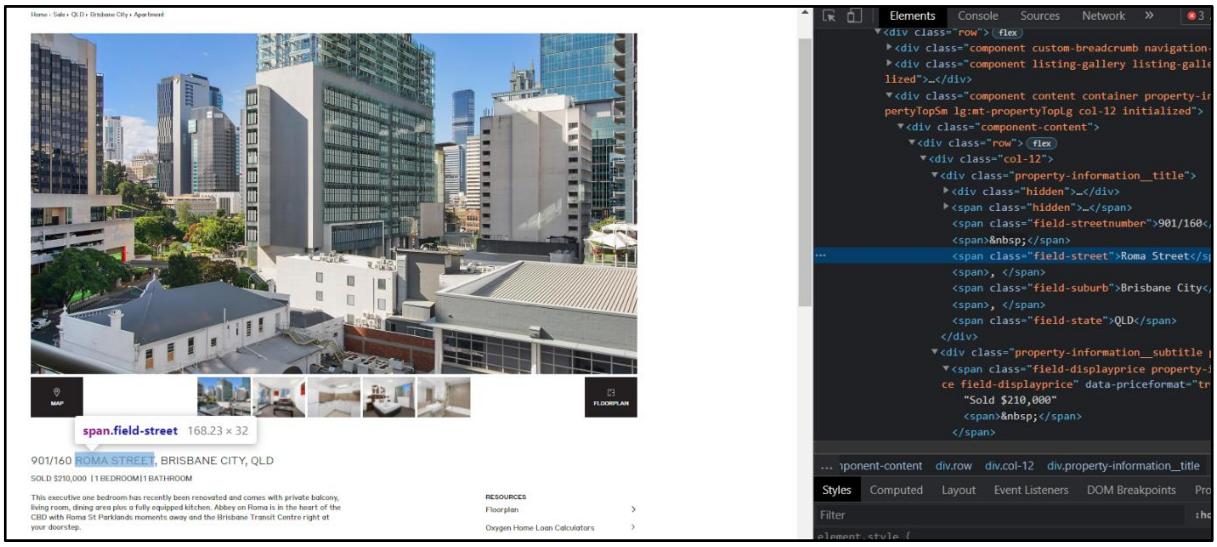
Figure 2: Location of property URLs within website HTML.

To extract all property URLs generated by this *McGrath* search, the *find_all* and *find* commands from the *BeautifulSoup* were used to return all the tags and strings that match the *<div>* filters for which the URLs are under. Once all the property container tags were located, a *for* loop was used to iterate through each container and extract the URL for each property, with each URL being appended to the *urls* list.

Despite executing the correct code to access each URL property tag, the website's HTML returned an automated text response in its HTML, "*Thanks for searching. We do not currently have any property* *listings in your search area. Please search again or contact us*". This response alludes that the initial URL used to access the *McGrath* site did not have any properties associated with it, although manual access to this link via a web browser returns 1165 property listings. Due to this unusual encounter and being unable to scrape any property URLs from *McGrath* links, the *urls* list was manually filled with the URLs of 24 property listings, to be fed into the second part of the web scraper.

The second section of the web scraper is designed to access and extract property data from each URL scraped previously, which will then go on to create the data frame for any statistical or NLP analyses to be performed. Each property listing presents multiple key features that are of interest for extraction. These include the type of property (such as apartment, house, or unit), the sale price of the listing, street number, address, suburb, and state, as well as a paragraph description of each property.

Before retrieval of this information, empty lists were initialised for each field to store scraped data in; *property_type, street_number, street_address, suburb, state, price,* and *description*. A *for* loop was created to loop through each URL stored in the *urls* list for scraping. For each URL, the *get* command from the *Requests* library was used to gain access to the contents of the page. If the request has a response of *200*, the content of the URL can be accessed, and *BeautifulSoup* is used to parse the HTML content received. Once the HTML content has been parsed, the desired tags for each field can be accessed using *BeautifulSoup*'s *find* command, along with the *text* command to extract only the text element from the tag. This data is then appended to the appropriate list. Property type can be located under the *<li>* tag, with a *class = breadcrumb-item breadcrumb-property-type*. Street number, address, suburb, state, and price are nested in a *<span>* tag, with corresponding classes *field-streetnumber, field-street, field-suburb, field-state,* and *field-displayprice property-information__price field-displayprice* (refer to Figure 3). The final data of interest, property description, is within a *<p>* tag where *class = property-information__summary field-description*.



901/160 ROMA STREET, BRISBANE CITY, QLD

SOLD $210,000 |1 BEDROOM|1 BATHROOM

This executive one bedroom has recently been renovated and comes with private balcony, living room, dining area plus a fully equipped kitchen. Abbey on Roma is in the heart of the CBD with Roma St Parklands moments away and the Brisbane Transit Centre right at your doorstep.

```
▼<div class="row"> flex
  ▶<div class="component custom-breadcrumb navigation-
  ▶<div class="component listing-gallery listing-galle
  lized">…</div>
  ▼<div class="component content container property-ir
  pertyTopSm lg:mt-propertyTopLg col-12 initialized">
    ▼<div class="component-content">
      ▼<div class="row"> flex
        ▼<div class="col-12">
          ▼<div class="property-information__title">
            ▶<div class="hidden">…</div>
            ▶<span class="hidden">…</span>
             <span class="field-streetnumber">901/160<
             <span>&nbsp;</span>
             <span class="field-street">Roma Street</sp
             <span>, </span>
             <span class="field-suburb">Brisbane City<
             <span>, </span>
             <span class="field-state">QLD</span>
           </div>
          ▼<div class="property-information__subtitle
            ▼<span class="field-displayprice property-i
           ce field-displayprice" data-priceformat="tr
               "Sold $210,000"
               <span>&nbsp;</span>
             </span>
… ıponent-content  div.row  div.col-12  div.property-information__title
element.style {
```

Figure 3: Location of extracted property data within HTML.

After all property listing URLs had been looped through and data scraped, a *Pandas* data frame was created, joining all eight lists formed during web scraping. Once in the data frame, the scraped data was tidied up, removing all unnecessary symbols with the *replace* command, such as new line breaks (*\n*) and dashes. The finalised and clean dataset was exported as a CSV file.

A brief analysis of the *properties* dataset shows its dimensions of 24 rows by 8 columns. With the property description being the biggest text section scraped, NLP tasks are to be performed on this. The average number of words per property description is 131.08, where a distribution of description word lengths can be viewed in Figure 4, created using the *MatplotLib* library.

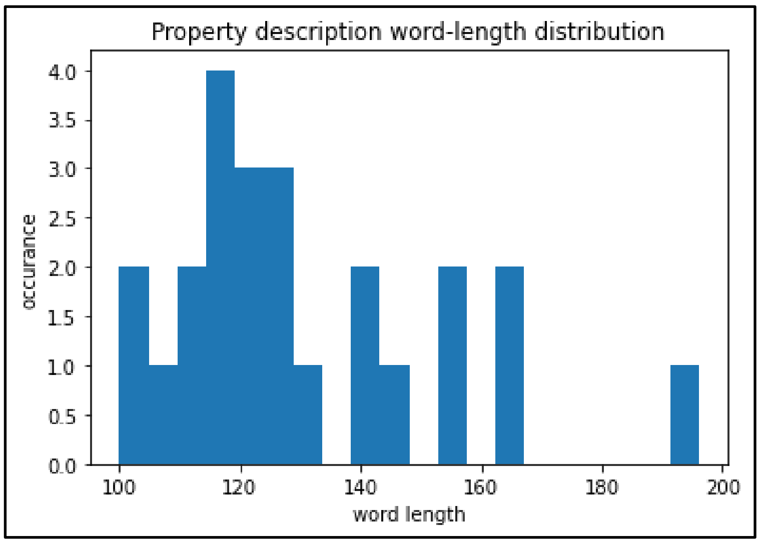


Figure 4: Distribution of word lengths for property descriptions.

After performing some NLP tasks, it is ideal to assess their performance by means of a dissimilarity matrix. To do this, each data entry needs some form of label or classification criteria. In the case of the *properties* dataset, the *property_type* field would be ideal for labelling, consisting of five unique classes.

Task 3: Prototype NLP Tasks

To receive the best understanding of the real estate data accumulated, multiple natural language processing tasks and techniques are implemented, being sentiment analysis and text summarisation of the property description field.

Sentiment analysis has been chosen as an ideal NLP task to perform on the gathered real estate data, where a sentiment score is computed on a large chunk of text, in this case, the property description. Customers can use the score generated for each property to assist with filtering through listings, depending on what score threshold they are interested in, thus further reducing search time.

To gain an understanding of the emotions conveyed through textual data, the analytical process of sentiment analysis, also referred to as opinion mining, can be undertaken. The role of sentiment analysis is to overall determine whether inputted textual information is of a positive, negative, or neutral nature about a specific topic or product. Sentiment analysis relies upon a combination of natural language processing, artificial intelligence, and machine learning approaches. Overall, sentiment analysis works as an automated method to generate an appropriate rating for a product or topic, thus making a customer’s need to consume reviews and descriptions redundant. There are two main approaches for implementing sentiment analysis, being either machine learning or lexicon-based. For the machine learning approach, supervised learning and text classification are at focus, relying on methods such as Naïve Bayes, K-Nearest Neighbours, Support Vector Machine, and neural networks. In contrast, a lexicon-based approach simply relies on a pre-existing dictionary and corpus of language and sentiment ratings (Hemamalini, Dr.S. Perumal, 2013). For implementing sentiment analysis to the *properties* data frame, the lexicon-based approach will be employed.

To perform a sentiment analysis on each property description within the *properties* data frame, python’s natural language tool kit library, *nltk* is implemented. As part of the NLTK library, VADER (Valence Aware Dictionary for Sentiment Reasoning) and its complementary lexicon are used specifically for text sentiment analysis tasks, a model that is sensitive to both positive and negative emotion in text, as well as that text’s emotional intensity (Hutto, C.J. & Gilbert, E.E., 2014). A sentiment analyser, *sid* was first coded using the *SentimentIntensityAnalyzer* command from *VADER*. The created sentiment analyser was implemented to every description entry in the *properties* data frame, using the *apply* command from the *pandas* library. Within the *apply* command, the *lambda* function is used to iterate over each row, *r*, of the *properties* data frame, before executing the sentiment analyser *sid* to calculate the polarity scores for each description. After each sentiment score is computed and stored in the *pandas* series array, *description_sent_*, the array was converted to a *pandas* data frame, which was then joined onto the original *properties* data frame. Viewing the new data frame, four new columns have been added, detailing the sentiment score for each property – *neg* (the individual negative score), *neu* (the individual neutral score), *pos* (the individual positive score), and *compound* (the combined and overall sentiment score).

After a review of the achieved sentiment scores for each property, it is obvious that the compound score for all properties is highly positive. By implementing a *for* loop, the average compound sentiment score is calculated, being 0.927. A sentiment score lies between 0 and 1, where the closer to 0, the more negative the text’s sentiment, and the closer to 1 the more positive sentiment. In retrospect, basing a sentiment score off property description provided by the real estate agent is bound to achieve a high sentiment, as real estate agents are likely to write positive descriptions on properties to sell them. Instead, in future analyses, further data supplementation could be implemented to scrape textual information on either property reviews or inspection reviews, which would provide a better, more reliable sentiment analysis output for a customer to assess.

The second NLP task, text summarisation, will be applied to the property descriptions with an aim to condense these to only encapsulate the most important information. As an effect, the time a customer takes to view each listing would be reduced due to being able to read a summary instead of the lengthy description. Also implementing text summarisation will help to provide a brief and unique overview of each listing enhancing a customer's ability to distinguish them.

Text summarisation systems work by taking a whole text document as input and outputting a summary - a shortened version of text covering only the most important information from the original input. To produce this summary based on the original document, one of two methods can be applied, the extractive or abstractive method. To implement an abstractive text summarisation method, a deep understanding of the text is required to create its summary based on the main ideas conveyed in the text, rather than simply producing a verbatim of the most important pre-existing sentences. The extractive method on the other hand relies upon linguistics and statistical approaches such as term frequency weighting to select these existing sentences in the text with the most influence as its summary (Saziyabegum & Sajja, 2016). For implementing text summarisation to the *properties* data frame, the extractive approach will be employed.

To perform text summarisation for property descriptions, the python libraries *heapq* and *nltk* are required. From the *nltk* library, the English *stopwords* dictionary and *punkt* package are employed. The list, *summary* was initialised to store generated description summaries. For each description in the *properties* data frame, *sent_tokenize* from *punkt* was used to tokenise each description into sentences. To compute a score weight for each sentence, weights for individual words need to be calculated prematurely. A dictionary was initialised to keep track of word frequencies within the definitions. The *word_tokenize* command was applied to the whole property description, before looping through and tallying the frequency of each word (excluding stop words). The weighted frequency of each word in the *word_frequencies* dictionary could then be calculated by dividing each word frequency by the maximum word frequency present for a description. Once the weighted frequency of each word has been calculated, a sentence weight is also able to be generated. A dictionary was initialised to store sentence scores, and for each tokenised sentence in the property description, a sum of the weighted frequencies of words in the sentence was calculated and assigned as the sentence weight. Finally, a hyperparameter of 2 was selected, as the number of sentences to retain from a description to become a summary. The two sentences with the largest weighting were retrieved using the *nlargest* command from the *heapq* library, before being appended to the official *summary* list. The complete *summary* list was joined to the *properties* data frame.

Evaluating the performance of the created text summarisation system appears to work effectively and execute as anticipated. A *for* loop was implemented to calculate the average number of words in each description summary, being 47.5. As calculated earlier from the initial data frame, each description holds on average 131.08 words, indicating that, the summaries generated are approximately a third of the size of the initial property description. This was the desired outcome, reducing the text size and thus a customer's reading time of a property description significantly, whilst still conveying the overall importance of the description.

In conclusion, this prototype has aimed to address the issue faced by a customer while perusing real estate property listings. The repetitive nature of this task can be time-consuming for customers as well as confusing at times when the information presented by many properties tends to blend together. Using a web scraper to obtain real estate property data from the *McGrath* website, property data was analysed and adapted by natural language processing. Sentiment analysis of property descriptions was implemented, assigning a score from 0 to 1 for each property based on the emotion in the text, allowing a customer to easily judge a property. Text summarisation was also performed, reducing the size of the property description by a third, providing the customer with only the most important information from the description, in turn reducing time spent reviewing each listing.

References

Chowdhury, G. G. (2005). Natural language processing. *Annual Review of Information Science and Technology*, 51.

Hemamalini, Dr.S. Perumal. (2013). *LITERATURE REVIEW ON SENTIMENT.* INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 9, ISSUE 04, APRIL 2020.

Hutto, C.J. & Gilbert, E.E. (2014). VADER: A *Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text.* Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

Kluyver, T., Ragan-Kelley, B., Fernando, P., Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), P*ositioning and Power in Academic Publishing: Players, Agents and Agendas* (pp. 87–90).

McGrath. (2021). *LEGAL INFORMATION*. Retrieved from McGrath: https://www.mcgrath.com.au/terms

McGrath. (2021). *McGrath*. Retrieved from McGrath: https://www.mcgrath.com.au/

realestate.com.au. (2018). *Terms of Use*. Retrieved from realestate.com.au: https://about.realestate.com.au/terms-use/

Saziyabegum, S., & Sajja, P. (2016). *Literature Review on Extractive Text Summarization.* Anand, India: International Journal of Computer Applications (0975 – 8887) Volume 156 – No 12, December 2016.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.