Task 2: Web Crawler to scrape TripAdvisor using Selenium (1500 words) - 30% of overall

**2.1 Domains: (~325 words) 15%**

*2.1.1 Website URLs to be crawled*

TripAdvisor is one of the most influential websites for customers when booking their holiday accommodation. The website reaches 390 million unique visitors each month and listing 465 million reviews and opinions about more than 7 million accommodations, restaurants, and attractions in 49 markets worldwide. Because TripAdvisor has so many data, it has become extremely popular with both tourists and managers. Tourists can read the accumulated opinions of everyday tourists. They can also check the popularity index, which is computed using an algorithm that accounts user reviews and other published sources such as guidebooks and paper articles.

TripAdvisor has their own official hotel description and bubble score rating, but the website also offers thousands of hotel reviews from previous customers, and these are coupled with the users own bubble score rating. The bubble score rating is on a scale from 1 to 5, where one bubble represents a terrible experience, and five bubbles represents an excellent experience. Online customer opinion has become so valuable to travellers but understanding the true sentiment of online user reviews has posed a daunting task given the complexities of the human language.

TripAdvisor has enough standing to be used as a text source, storing numerous reviews of tourist businesses around the world.

*2.1.2 Coverage of the chosen domains*

To address the issue of ‘what attributes relate to excellent accommodation in Townsville’, the user (customer, travel agent or hotel manager) needs to understand the important topics and aspects of positive reviews. For example, the important topics when considering accommodation may include features like location, what facilities the accommodation has. Important aspects of customer reviews may include the friendliness of hotel staff and cleanliness of the rooms.

This investigation decided to extract the following information from the TripAdvisor website:

* URLs for all the hotels in Townsville.
* Name of the hotels.
* TripAdvisor description of the hotels.
* Customer reviews for the hotels.
* Customer rating for the hotels.
* Date of stay for each user who provided a review.

*2.1.3 The natural language data and how these data align to the issue*

The natural language data that was extracted from the TripAdvisor website was:

* Hotel descriptions - The hotel description was the official description of the hotel provided by TripAdvisor. Not all hotels had a description provided on TripAdvisor.
* Customer review - The review provided by the customer who had stayed at the accommodation. The reviews were full of sentiment, both positive and negative. Not all customers who stayed at the hotel provided a review and some hotels have more reviews than others.

The hotel descriptions and customer reviews give a mix of commercial and personal language. This scraped natural language data will be used to explore what attributes relate to excellent accommodation in Townsville by providing two different viewpoints of the accommodation in TripAdvisor and the many customer reviews.

*2.1.4 Copyright of chosen domain*

In copyright law, ‘fair use’ permits third parties to use copyright protected works but only in limited way. The more analytics performed on the scraped data, the more the scraped material tends towards fair use. The more ‘excerpting’ performed on scraped data the more likely one is liable of copyright infringement. An example of this is where one would not change anything significant about the scraped material and simply reposting snippets of the data as is. Creating your own classifications and fields of information will generally mean playing around with scraped data and organizing it in a different way than what you received from a robot will push your use of information towards the “transformative “, allowing you to create a stronger claim of “fair use” of information under copyright law.

TripAdvisor content is protected by copyright but as this is an academic exercise and we intend perform an analysis on the material, scraping review data from TripAdvisor does not violate copyright laws.

**2.2 WebCrawler workflow: (~750 words) 50%**

*2.2.1 Technology components used for the web crawler*

The TripAdvisor website uses lazy loading scripts which edit the HTML document object model (DOM) and can break the element refs. This can make it difficult to write a consistent web-crawler program to move through all the URLs and then scrape the required data from each site. At the time of writing this report, a web-crawler application was successfully developed using the **Selenium** package to scrape the required data for this investigation using WAIT\_FOR methods. Selenium works by automating web browser activity, replicating functions such as opening a browser, clicking on links. This automated activity mimics the movements of an actual human website user and was not identified by bot detection. The package requires installation of a driver to interface with a browser (Chrome was used in this case). Given the fact our program was able to scrape over 18,000 customer reviews suggests TripAdvisor may not have bot detection software.

*2.2.2 Complexity of the domains where the targeted data resides*

Scraping challenges – As highlighted in the screen shot below, TripAdvisor only loads part of the review initially and waits until the user clicks ‘Read more’ to load the rest.

Graphical user interface, text, application, letter

Description automatically generated

The following is screen of the inspect element tool before the ‘Read more’ button is clicked.

Text

Description automatically generated

Below is the same review once the ‘Read more’ button is clicked.

*Text, letter

Description automatically generated*

Below is a screen shot of the inspect element after the ‘Read more’ button is clicked.

Text

Description automatically generated

Once clicked, further Javascript is executed making it challenging to scrape the whole review. **Selenium** package launched the browser and simulated pressing the ‘Read more’ link which executed the JavaScript to reveal the whole review for scraping. It then needed to be waited for until the data populated successfully.

Another example of the complexity of the TripAdvisor website is the graphical score given by the user as part of their review. The image is made up of five circles, some of which are filled in green. The number of green circles indicates the score out of 5. There is no textual indication of the score, and this makes it difficult to scrape. This study identified that the CSS class of the SPAN element holds an indication of the score.

Graphical user interface, text, application

Description automatically generated

By performing a string manipulation on the class name and retaining a substring of the last two index positions, the code was able to extract the review rating.

*2.2.3 Sequencing of the crawler, using the complexity, data structures and website access restrictions to optimise the crawler*

This report will analyse language data from user reviews of hotels in Townsville, using **Python version 3.9.0**. Three web-crawler programs were used to extract the required data from TripAdvisor. The sequencing of the web-crawler programs is as follows:

* *‘urlCrawler’* – Takes the initial query of 'https://www.tripadvisor.com.au/Hotels-g255073-Townsville\_Queensland-Hotels.html'. Crawls through every Townsville hotel on TripAdvisor and collect the URLs. Save the file as hotelUrls.csv
* *‘hotelInfoScraper’* - Use the list of URLs, from hotelUrls.csv, to crawl through all web pages to scrape the TripAdvisor hotel descriptions and hotel names. Save the file as hotelData.csv.
* *‘hotelReviewScraper’* - Perform a third web crawl, using the list of URLs, to extract the customer reviews for each hotel, the bubble score rating for each review and the date of the stay for each review. Save the file as hotelReviews.csv.

The following code was used to crawl through all web pages and collect the URLs associated with hotels in Townsville on TripAdvisor:

#You must run them in the same folder as the .py file

chrome\_options = webdriver.ChromeOptions()

chrome\_options.add\_argument('--headless')

chrome\_options.add\_argument('--no-sandbox')

chrome\_options.add\_argument('--disable-dev-shm-usage')

driver = webdriver.Chrome('chromedriver', options=chrome\_options)

defaultTimeout = 30

defaultRetryWaitTime = 0.5

url = 'https://www.tripadvisor.com.au/Hotels-g255073-Townsville\_Queensland-Hotels.html'

#Creates a DataFrame for urls we can save later

hotels = pd.DataFrame(columns = ['url'])

hotels.index.name = "hotelId"

#stores the nextPage to parse

nextPage = url

def timedRetryGetAttribute(element, attribute):

startTime = time.time()

finishTime = startTime + defaultTimeout

firstTime = True

while time.time() < finishTime:

try:

if firstTime == False:

time.sleep(defaultRetryWaitTime)

firstTime = False

attr = element.get\_attribute(attribute)

return attr

except Exception:

continue

return ""

def getHotels(driver):

hotelsNew = []

listingTitles = WebDriverWait(driver, defaultTimeout).until(

EC.presence\_of\_all\_elements\_located((By.CLASS\_NAME, "listing\_title"))

)

for titleElement in listingTitles:

#Needed to catch exception as some titleElement were null

try:

childElements = WebDriverWait(titleElement, defaultTimeout).until(

EC.presence\_of\_all\_elements\_located((By.XPATH, './/\*'))

)

except TypeError:

continue

firstElement = childElements[0]

hrefValue = timedRetryGetAttribute(firstElement, 'href')

#Needed to catch exception as some values were null and showing as blank lines in the .csv

try:

if len(hrefValue) < 5:

continue

except TypeError:

continue

#Add the URL into our array that we'll save to CSV later

hotelsNew.append(hrefValue)

print(f"Found {len(hotelsNew)} hotels on this page")

return hotelsNew

def getNextPage(driver, currentPageNumber):

nextPageNumber = currentPageNumber + 1

pageNumbersElement = WebDriverWait(driver, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CLASS\_NAME, "pageNumbers"))

)

pageNumbersElementChildren = WebDriverWait(pageNumbersElement, defaultTimeout).until(

EC.presence\_of\_all\_elements\_located((By.XPATH, './/\*'))

)

for child in pageNumbersElementChildren:

pageNumberAttr = timedRetryGetAttribute(child, 'data-page-number')

#Needed to catch exception as some values are null, such as "..." option

try:

pageNumberAttr = pageNumberAttr + ""

except TypeError:

continue

if int(pageNumberAttr) == nextPageNumber:

return timedRetryGetAttribute(child, 'href')

return ""

#This tracks the current page number, useful for keeping track when moving through pages

#perhaps change to currentPageNumber?

currentPageNumber = 1

getPageRetryMax = 5

currentTry = 1

try:

while nextPage != "":

if currentTry > getPageRetryMax:

raise Exception("Failed too many times")

print(f"Downloading current page: {currentPageNumber}")

driver.get(nextPage)

print("Adding hotel URLs")

hotelsNew = None

try:

hotelsNew = getHotels(driver)

except Exception:

currentTry = currentTry + 1

continue

print("Getting next page link")

try:

nextPage = getNextPage(driver, currentPageNumber)

except Exception:

currentTry = currentTry + 1

continue

for hotelNew in hotelsNew:

hotels.loc[len(hotels)] = [hotelNew]

currentPageNumber = currentPageNumber + 1

except Exception:

print("Failed too many times")

finally:

print("Saving data to disk")

hotels.to\_csv("hotelUrls.csv", index = True, header = True)

driver.quit()

print("Program finished")

Elements were identified and extracted using the HTML inspect navigation panel corresponding to the page in the chrome browser. The following code was used to scrape the TripAdvisor hotel description, and hotel name from each URLs collected in the previous code:

chrome\_options = webdriver.ChromeOptions()

chrome\_options.add\_argument('--headless')

chrome\_options.add\_argument('--no-sandbox')

chrome\_options.add\_argument('--disable-dev-shm-usage')

driver = webdriver.Chrome('chromedriver', options=chrome\_options)

defaultTimeout = 30

hotelData = pd.read\_csv('hotelUrls.csv')

totalHotelsToScrape = len(hotelData)

print(f"Preparing to scrape {totalHotelsToScrape} hotels")

hotelNames = []

hotelDescriptions = []

getPageRetryMax = 5

currentTry = 1

for index, row in hotelData.iterrows():

currentTry = 1

while True:

if(currentTry > getPageRetryMax):

print(f"Failed to retrieve information from hotel at index {index}, skipping")

hotelNames.append("")

hotelDescriptions.append("")

break

try:

#There would be a nicer way to track this index, but this works

currentHotelIndex = len(hotelNames)

currentPageUrl = row['url']

driver.get(currentPageUrl)

#Get hotel name from page, it has an ID

hotelNameElement = WebDriverWait(driver, defaultTimeout).until(

EC.presence\_of\_element\_located((By.ID, "HEADING"))

)

hotelName = hotelNameElement.text

aboutElement = WebDriverWait(driver, defaultTimeout).until(

EC.presence\_of\_element\_located((By.ID, "ABOUT\_TAB"))

)

descriptionElement = WebDriverWait(aboutElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "div[class='pIRBV \_T']"))

)

descriptionText = descriptionElement.text.replace('\n', " ")

#If the name is blank/small just put a blank string in the array to keep our column aligned with hotelData

if(len(hotelName) < 5):

hotelNames.append("")

else:

hotelNames.append(hotelName)

#If the description is blank/small just put a blank string in the array to keep our column aligned with hotelData

if(len(descriptionText) < 5):

hotelDescriptions.append("")

else:

hotelDescriptions.append(descriptionText)

print(f"Scraped hotel {currentHotelIndex + 1} of {totalHotelsToScrape}, hotelName: {hotelName}, len(descriptionText): {len(descriptionText)}")

break

except Exception:

print(f"Failed to get data from page at index {index}, retrying...")

currentTry = currentTry + 1

continue

hotelData['hotelNames'] = hotelNames

hotelData['hotelDescriptions'] = hotelDescriptions

hotelData.to\_csv("hotelData.csv", index = False, header = True)

driver.quit()

The hotel descriptions were saved to a csv file, ‘hotelData.csv’. The following code was used to collect the customer reviews for every hotel, the customer rating, and the date of their stay by crawling through the pages by URL:

# hotelId, reviewId, stars, date, review

hotelData = pd.read\_csv('hotelUrls.csv')

hotelFlag = [0] \* len(hotelData)

hotelData['hotelFlag'] = hotelFlag

hotelReviews = pd.DataFrame(columns = ['hotelId', 'reviewId', 'stars', 'date', 'review'])

def timedRetryGetAttribute(element, attribute):

startTime = time.time()

finishTime = startTime + defaultTimeout

firstTime = True

while time.time() < finishTime:

try:

if firstTime == False:

time.sleep(defaultRetryWaitTime)

firstTime = False

attr = element.get\_attribute(attribute)

return attr

except Exception:

continue

return ""

def timedRetryGetAttributeContains(element, attribute, contains):

startTime = time.time()

finishTime = startTime + defaultTimeout

firstTime = True

while time.time() < finishTime:

try:

if firstTime == False:

time.sleep(defaultRetryWaitTime)

firstTime = False

attr = element.get\_attribute(attribute)

if("none" in attr):

return attr

except Exception:

continue

return ""

def timedRetryClickOnCssClass(element, cssClass):

startTime = time.time()

finishTime = startTime + defaultTimeout

firstTime = True

while time.time() < finishTime:

try:

if firstTime == False:

time.sleep(defaultRetryWaitTime)

firstTime = False

elementClickable = WebDriverWait(element, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, cssClass))

)

elementClickable.click()

return

except Exception:

continue

return ""

def getReviews(dataFrameIndex, hotelId, pageUrl, pageNumber):

#pageUrl = https://www.tripadvisor.com.au/Hotel\_Review-g255073-d255393-Reviews-Hotel\_Grand\_Chancellor\_Townsville-Townsville\_Queensland.html#REVIEWS

fixedUrl = pageUrl

if pageNumber != 1:

orNum = (pageNumber - 1) \* 5

reviewsKeyword = "Reviews-"

reviewKeywordIndex = pageUrl.index(reviewsKeyword) + len(reviewsKeyword) #68

urlFirstPart = pageUrl[:reviewKeywordIndex] #https://www.tripadvisor.com.au/Hotel\_Review-g255073-d255393-Reviews-

orStr = "or" + str(orNum) + "-" #or5-

urlSecondPart = pageUrl[reviewKeywordIndex:] #Hotel\_Grand\_Chancellor\_Townsville-Townsville\_Queensland.html

fixedUrl = urlFirstPart + orStr + urlSecondPart #https://www.tripadvisor.com.au/Hotel\_Review-g255073-d255393-Reviews-or5-Hotel\_Grand\_Chancellor\_Townsville-Townsville\_Queensland.html

getPageRetryMax = 5

currentTry = 1

while True:

try:

if(currentTry > getPageRetryMax):

print(f"Failed to retrieve reviews from hotelId {hotelId}, skipping")

break

print(f"Scraping reviews for hotelId {hotelId} on their page number {pageNumber}")

hotelReviewsNew = []

driver.get(fixedUrl)

try:

reviewElements = WebDriverWait(driver, defaultTimeout).until(

EC.presence\_of\_all\_elements\_located((By.CSS\_SELECTOR, "div[class='cWwQK MC R2 Gi z Z BB dXjiy']"))

)

except TimeoutException:

print(f"Could not see any reviews on hotelId {hotelId} on their {pageNumber} page")

hotelData.loc[dataFrameIndex, 'hotelFlag'] = 1

break

#This will only trigger if a hotel has ZERO reviews in total

if len(reviewElements) == 0:

print(f"Hotel {hotelId} never had any reviews at all")

hotelData.loc[dataFrameIndex, 'hotelFlag'] = 1

return

for idx, reviewElement in enumerate(reviewElements):

#Get review id

reviewId = (pageNumber - 1) \* 5 + idx

#Get review stars

reviewStarParentElement = WebDriverWait(reviewElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "div[class='emWez F1']"))

)

reviewStarChildElement = WebDriverWait(reviewStarParentElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.XPATH, './/\*'))

)

reviewStarClass = timedRetryGetAttribute(reviewStarChildElement, "class") #ui\_bubble\_rating bubble\_30

reviewStars = reviewStarClass[-2:] #30

#Get review date

reviewDateElement = WebDriverWait(reviewElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "span[class='euPKI \_R Me S4 H3']"))

)

reviewDateTextRaw = reviewDateElement.text #'Date of stay: November 2021'

reviewDateTextRawColonIndex = reviewDateTextRaw.index(":") + len(":") + 1

reviewDate = reviewDateTextRaw[reviewDateTextRawColonIndex:]

#readMore

reviewTextBlockElement = WebDriverWait(reviewElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "div[class='dovOW']"))

)

reviewTextBlockElementChildren = WebDriverWait(reviewTextBlockElement, defaultTimeout).until(

EC.presence\_of\_all\_elements\_located((By.XPATH, './\*'))

)

for child in reviewTextBlockElementChildren:

childsClass = timedRetryGetAttribute(child, 'class')

if "duhwe \_T bOlcm" not in childsClass:

continue

if "dMbup" in childsClass:

timedRetryClickOnCssClass(child, "span[class='eljVo \_S Z']")

break

#wait until the class 'duhwe \_T bOlcm' is directly below dovOW, but dovOW refreshes as well

expandedReviewTextBlockParentElement = WebDriverWait(reviewElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "div[class='dovOW']"))

)

try:

WebDriverWait(expandedReviewTextBlockParentElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "div[class='duhwe \_T bOlcm']"))

)

except Exception:

pass

reviewTextElement = WebDriverWait(expandedReviewTextBlockParentElement, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "q[class='XllAv H4 \_a']"))

)

reviewText = reviewTextElement.text.replace('\n', " ")

hotelReviewsNew.append([hotelId, reviewId, reviewStars, reviewDate, reviewText])

uiPaginationElement = None

try:

uiPaginationElement = WebDriverWait(driver, defaultTimeout).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, "div[class='ui\_pagination is-centered"))

)

except TimeoutException:

hotelData.loc[dataFrameIndex, 'hotelFlag'] = 1

print(f"Found {len(hotelReviewsNew)} new reviews for hotelId {hotelId}")

print(f"Hotel {hotelId} only has one page of reviews")

for hotelReviewNew in hotelReviewsNew:

hotelReviews.loc[len(hotelReviews)] = hotelReviewNew

break

uiPaginationElementChildren = WebDriverWait(uiPaginationElement, defaultTimeout).until(

EC.presence\_of\_all\_elements\_located((By.XPATH, './\*'))

)

for childElement in uiPaginationElementChildren:

childElementClass = timedRetryGetAttribute(childElement, "class")

isNextButton = "next" in childElementClass

if isNextButton == False:

continue

isNextDisabled = "disabled" in childElementClass

if isNextDisabled == True:

print(f"Hotel {hotelId} does not have another page of reviews")

hotelData.loc[dataFrameIndex, 'hotelFlag'] = 1

print(f"Found {len(hotelReviewsNew)} new reviews for hotelId {hotelId}")

for hotelReviewNew in hotelReviewsNew:

hotelReviews.loc[len(hotelReviews)] = hotelReviewNew

break

except Exception as error:

print(f"Failed to retrieve reviews from hotelId {hotelId} on pageNumber {pageNumber}")

currentTry = currentTry + 1

def getReviewsWithPageNumber(pageNumber):

for index, row in hotelData.iterrows():

hotelId = row['hotelId']

pageUrl = row['url']

hotelFlag = row['hotelFlag']

if hotelFlag == 0:

getReviews(index, hotelId, pageUrl, pageNumber)

hotelReviews.to\_csv("hotelReviews.csv", index = False, header = True)

hotelData.to\_csv("hotelReviews-flags.csv", index = False, header = True)

def anyHotelsToCheck():

return 0 in hotelData['hotelFlag'].values

pageNumber = 1

while anyHotelsToCheck():

print(f"Scraping each hotel page for reviews, looking at their page number {pageNumber}")

getReviewsWithPageNumber(pageNumber)

print(f"Finished scraping each hotel's page {pageNumber} for reviews, currently we have {len(hotelReviews)} reviews")

pageNumber = pageNumber + 1

print("No more hotel reviews to scan")

driver.quit()

A flag system was used to deal with how the ‘*hotelReviewScraper*’ program would only look for the appropriate amount of pages for customer reviews. The program detected if the hotel had no more reviews to scrape if the current page was ‘no more reviews’ or the ‘next’ button was not clickable. Fault tolerant coding practises were emphasised to ensure the code was able to run successfully on the first attempt and prevent being blocked by bot detection. An example of this is the significant amount of retry logic applied to code, based on exception throwing. Waiting loops were used to allow for the lazy loading scripts as mentioned in part *2.2.1.* Also, print statements were added to the loops to understand the efficiency of the code and potentially minimise bot detection with early termination if a bug was encountered.

*2.2.4 Data Storage*

All output from the three web crawler programs were saved as a comma-separated values file:

* hotelUrls.csv
* hotelData.csv
* hotelReviews.csv

*Demonstrate: Robust, fault tolerance coding practices / Illustration*

Reach your goal. Stop

Crawl the page for a new URL

Get the initial URL

Whether the stop condition is satisfied

Put the new URL in the URL queue

Put the captured address into the list that has been crawled

Read the new URL

**2.3 Harvested Corpus EDA: (~325 words) 35%**

*2.3.1 Data Wrangling*

Hotels that were missing the description were removed from the ‘hotelData’ data using the following code:

# Remove rows with missing descriptions

dfHotelData = dfHotelData.dropna( how='any', subset=['hotelDescriptions'])

In the ‘hotelReviews’ data, the review rating was changed to 1–5 (instead of 10-50) to make the data easier to interpret.

Most of the data wrangling was applied in the web crawler scripts which left little wrangling to perform following the corpus harvest.

*2.3.2 Summary of the generated corpus*

The hotelData.csv file is comprised of four columns: ‘hotelID’, ‘url’, ‘hotelNames’ and ‘hotelDescriptions’ (Table x). The data has 115 observations with 25 rows missing the hotel name and hotel description.

|  |  |  |
| --- | --- | --- |
| **Field ID** | **Description** | **Totals** |
| hotelId | ID for each hotel in Townsville. | 115 |
| url | URL of the hotel listed on TripAdvisor. | 115 |
| hotelNames | The name of the hotel. | 90 |
| hotelDescriptions | The TripAdvisor description of the hotel. | 90 |

**Table x.** Data table dictionary for hotelData.csv.

The hotelReviews.csv file is comprised of five columns: ‘hotelId’, ‘reviewId’, ‘stars’, ‘date’ and ‘hotelReviews’ (Table x). The data has 18242 observations with no missing data.

|  |  |  |
| --- | --- | --- |
| **Field ID** | **Description** | **Totals** |
| hotelId | ID for each hotel in Townsville. | 18242 |
| reviewId | ID for each review assigned to a hotel. | 18242 |
| stars | The user rating from each review. | 18242 |
| date | Date of stay from each user review. | 18242 |
| reviews | The review posted from the user review to TripAdvisor that describes their stay at the hotel. | 18242 |

**Table x.** Data table dictionary for hotelReviews.csv.

*2.3.3 Visualisation of the corpus*

To understand the harvest corpus of both the hotelData.csv and hotelReviews.csv, initial data exploration and analysis was performed.

*2.3.3.1 Visual representation of ‘hotelData.csv’ data.*

A bar plot was created to represent the number of words in each hotel description. Most of the hotel descriptions were between 50 – 100 words (Fig. x).

*Chart, histogram

Description automatically generated*

**Fig. x.** Word count distribution for hotel description from hotelData.csv.

A bigram was created to understand the frequency of two-word sequences in the hotel descriptions. The three most frequent two-word sequences in the hotel descriptions were ‘self contained’, ‘air conditioning’, and ‘The strand’ (Fig. x).

*Chart, bar chart

Description automatically generated*

**Fig. x.** Bigram distribution for hotel description from hotelData.csv.

A trigram was created to understand the frequency of three-word sequences in the hotel descriptions. The three most frequent three-word sequences in the hotel descriptions were ‘motel hall hire’, ‘fully self contained, and ‘offer air conditioning’ (Fig. x).

After exploring the bigram and trigram of hotel descriptions, the study identified that hotel descriptions most talked about the accommodation being self-contained and having air conditioning. The descriptions also frequently mentioned ‘The strand’ which is a seaside foreshore in Townsville which has views of the port of Townsville and Magnetic Island. This bigram speaks to the location of the accommodation.

*Chart

Description automatically generated*

**Fig. x.** Trigram distribution for hotel description from hotelData.csv.

A word cloud was created to visualise the frequent individual words in the hotel description (Fig. x). The most prominent words in the word cloud are ‘located’, ‘Townsville’, ‘apartment’ and ‘beachfront’.

*Text

Description automatically generated*

**Fig. x.** Word cloud for hotel description from hotelData.csv.

*2.3.3.2 Visual representation of ‘hotelReviews.csv’ data.*

A bar plot was created to represent the total word count for hotel reviews. Most of the reviews contained 50-100 words (Fig. x).

*Chart, histogram

Description automatically generated*

**Fig. x.** Word count distribution for hotel reviews from hotelReviews.csv.

A bigram was created to understand the frequency of two-word sequences in the hotel reviews. The three most frequent two-word sequences in the hotel reviews were ‘the room, ‘the staff, and ‘friendly helpful’ (Fig. x).

*Chart, bar chart

Description automatically generated*

**Fig. x.** Bigram distribution for hotel reviews from hotelReviews.csv.

A trigram was created to understand the frequency of three-word sequences in the hotel reviews. The three most frequent three-word sequences in the hotel reviews were ‘staff friendly helpful, ‘the staff friendly’, and ‘the room clean’ (Fig. x).

After exploring the bigram and trigram of hotel descriptions, the study identified that hotel reviews mostly talked about how friendly and helpful the staff, and how clean the room was. This is quite a contrast in comparison to the features of the room and location discussed in the hotel descriptions.

*Chart, bar chart

Description automatically generated*

**Fig. x.** Trigram distribution for hotel reviews from hotelReviews.csv.

A bar plot was created to visualise the distribution of the various rating in the hotel reviews. Most of the people who left reviews rated their stay as 4 and 5 stars (Fig. x).

*Chart, bar chart

Description automatically generated*

**Fig. x.** Distribution of ratings for hotel reviews from hotelReviews.csv.

A word cloud was created to visualise the frequent individual words in the hotel reviews (Fig. x). The most prominent words in the word cloud are ‘room’, ‘Townsville’, ‘breakfast’, ‘great’ and ‘close’.

*Text

Description automatically generated*

**Fig. x.** Word cloud for hotel reviews from hotelReviews.csv.

*2.3.4 Descriptive statistics of corpus*

The number of reviews per hotel ranges from 0 – 1179 in the *hotelData.csv*.