**TEAM: Jamie, Yasmin, Susov and Tas**

***INITIAL PROJECT BRIEF:***

To compare prices at three major retailers to obtain the data to where consumers should purchase TV’s. An additional retailer (Harvey Norman) was added after there were problems with JB Hifi (explained in the report).

The initial data sources that will be sourced:

* The Good Guys
* JB Hifi
* Bing Lee
* Harvey Norman (additional from the original brief)

***IDENTIFIED PROCESSES:***

**A. EXTRACT:**

Scrape the data from the retail outlets from their online stores obtaining the following information:

* Brand
* Size
* Product Name
* Model Number
* Currency
* Price
* Link to image
* Category

**B. TRANSFORM:**

* Complete ERD to explain the setup of the data table.
* Clean and store the data into a SQL database for querying.
* Amend and column types as required.
* Additional column names as required will be “retailer”.

**C. LOAD:**

* We will be putting the information into a SQL database, because we believe the structural integrity will be the same for the scraped and transformed data.

***ADDITIONAL - TIME PENDING***

**D. WEBSITE:**

* Build a basic website from a template to demonstrate who the data may be utilised by a consumer.
* Query the pricing sql database to extract the lowest price and display on a website that queries the database.

***FINAL:***

Report - As detailed below.

**FINAL REPORT**

***A. Extraction***

*Summary*

Goal: To extract, through scraping the retailer’s websites, the data on televisions available to purchase for the transformation phase.

Below is a more detailed summary of the extraction process and a table that summarises the process:

* Data sources and high level information.
* Dependencies used - these were chosen for the project:
  + Pandas: Python Pandas library for data manipulation, convert data into a format for easy export as a csv.
  + BeautifulSoup: Required for parsing and extracting the HTML code and attributes on the respective stores websites.
  + Splinter: Required to automate the browser actions, such as clicking “Next Page”, or “Load More”.
  + Selenium / Chrome WebDriver: Utilised to automate web browser interactions and work with BeautifulSoup and Splinter.
  + Sleep: Suspends and waits the execution of the “scrape”.
* Required Information - Summarises the success in scraping the information or if it was available.
* Attributes or tags scraped - a listing of the attributes for each item scraped.
* Additional information on work completed - convert to DataFrame, check for any duplicate values or missing values, any other minor cleaning as required and subsequent export to csv for transformation.

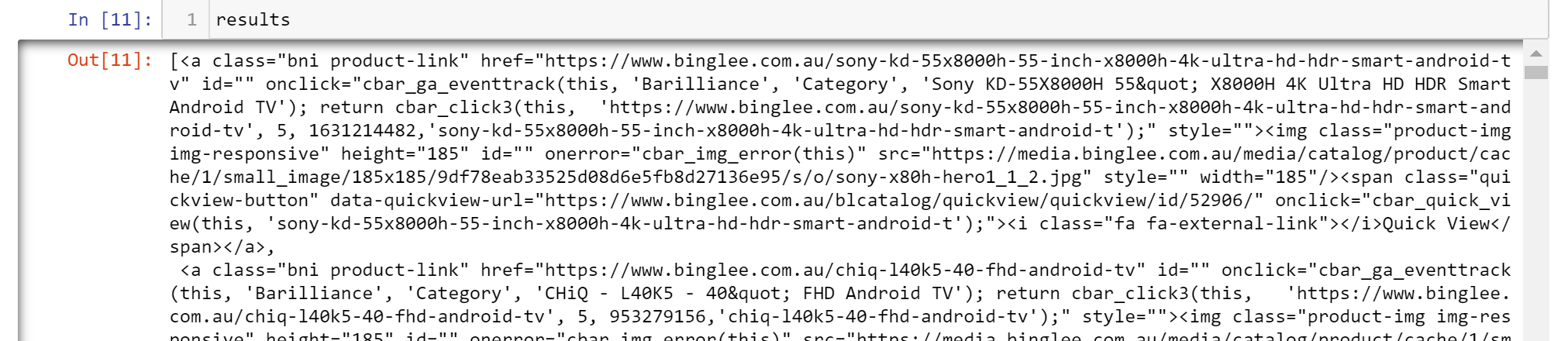
*Extraction comments and problems identified:*

***The Good Guys:***

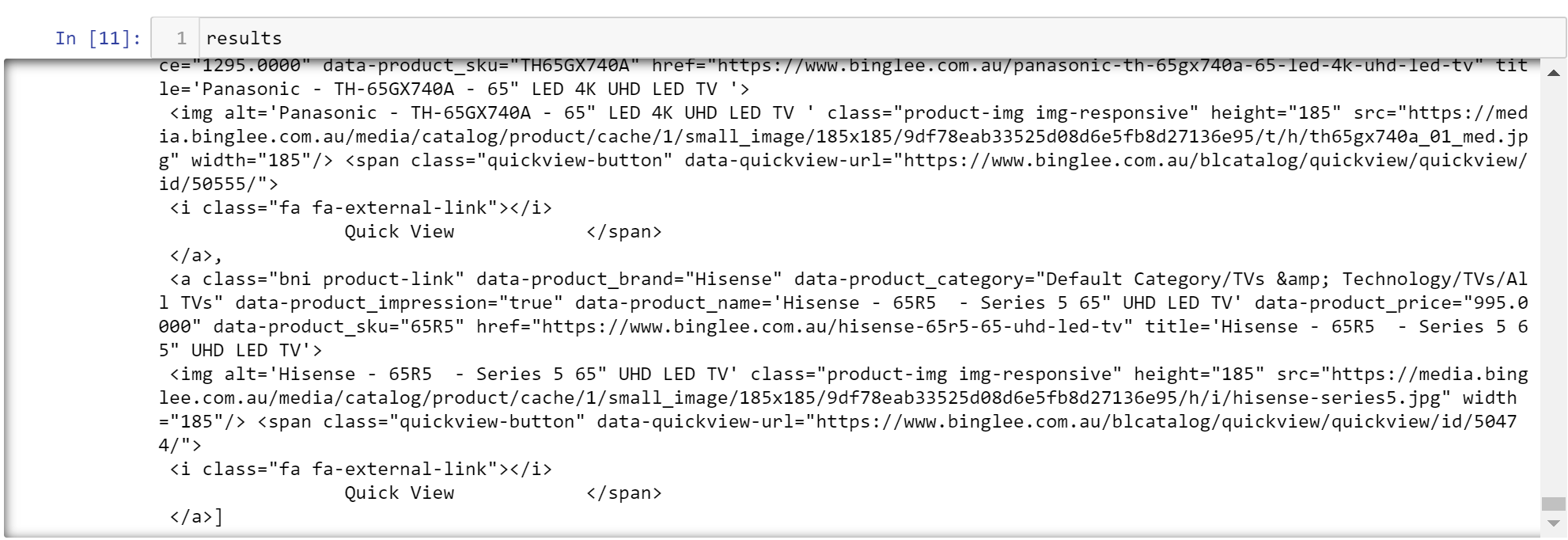
* Website serves up 60 product results per page in a <li> tag under a <div> tag with class “product\_listing\_tab”.
* **Challenge:** The html results presented in a page is not controlled by web address (using page index - see below in bold):
  + [https://www.thegoodguys.com.au/televisions/all-tvs#facet:&**productBeginIndex:60**&orderBy:&pageView:grid&minPrice:&maxPrice:&**pageSize:60**&](https://www.thegoodguys.com.au/televisions/all-tvs#facet:&productBeginIndex:0&orderBy:&pageView:grid&minPrice:&maxPrice:&pageSize:100&)
  + Therefore, we are not able to utilise Python requests module for each of the 3 pages by modifying the target url.
* The **solution** is to navigate through the pages using a combination of splinter, selenium and ChromeWebDriver (since the writer is using chrome).
* A function is created to automate the page navigation.
* Another function is created to obtain the web results using beautiful soup.
* The individual data points for each TV is extracted using beautiful soup and each of them are wrapped in a function:
  + **Product Category & Currency** are extracted using the Python rfind function to determine the location of the text within a string. Having determined the character location for the start and end positions within the string, the values are extracted by using the index to extract the substring.
  + **Brand, Name and Price** are extracted from <input> tags where the type=“hidden”. It is nested within the “values” element which was extracted using Beautiful Soup functions and observing the positions in the list which contains each information. Name is a combination of Brand and Name.
  + **Size** is extracted from the Name string from above. Upon checking, this has been found as imperfect and will be fixed during the Transformation phase.
  + **Model** is extracted from text contained within the <div> tag where class="product-tile-model".
  + **Image** is extracted from the <img> tag. The link to the image is contained within the “data-src” element within the <img> tag.
* After establishing all the necessary functions and defining the lists, a conditional loop within a loop is created using a combination of while, and forward statements:
  + Loop only runs while page number if the number of pages is 3 or less.
  + When the browser is on page 1, there is no need to call on the page navigation function. This is solved with a conditional statement.
  + We then call on the function to extract the web results using beautiful soup into a variable called “results”.
  + A second loop is used to extract all the data points and add them to lists.
  + When the browser is on page 2, we then call on the page navigation function. The **challenge** is to allow sufficient time for the results to load. Otherwise, we encountered duplicated results from previous pages and some missing information from the current page. We **resolved** this by using the sleep module to allow for the page to render.
  + We also utilised a try and except function to deal with errors. In particular, the website withholds price information when the product is temporarily unavailable. The information for each product is only captured when all data points are present.
* Data converted to pandas Dataframe to check for duplication and null values as quality control.
* Number of records reconciled with total number of TVs offered on the website and number of TVs that did not contain a price (and therefore omitted from being captured).
* Data frame exported as csv for further checking and transformation.

***Bing Lee:***

* The structure of the Bing Lee website was as follows:
  + **Header and Footer** - includes, menus, searches, shopping carts, contact details etc.
  + **Banner advertising** down the sides and a banner below the banner.
  + A **scroll banner** with the “top sellers” containing 12 items.
  + A “Shop By” **filter for the consumer** on the left of the page.
  + **Container** with the shopping items - with the first 36 items.
  + A **“Load 36 More” button,** which then changes to a **“Load More” button** after clicking on the first button.
* The key tags identified using BeautifulSoup to inspect the web address, as per the below table, to obtain the required information were <a> and the class being “bni product-link”.
* The two “loading” buttons were identified to find the splinter dependency *browser.click\_link\_by\_partial\_text*, being the text: “Load”.
* Data was then scraped, sleep was also used to break the code to not get blocked.
* Functions were completed - It was identified that the first 20 rows were redundant data that had **duplicate data** in different formatting and tags. This took some time to resolve the reason the code was not running without error. This information was not extracted below as the script would not run through those rows. This was discovered when doing some checking that the results were being obtained. The first 20 rows were giving an error, but subsequent rows were successful. An example of the bad data was:



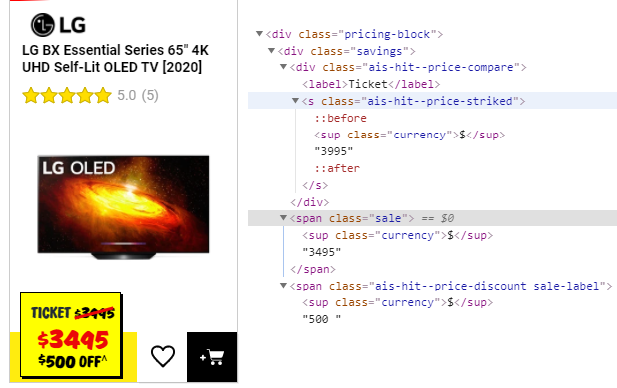
The required information was in the following format:



* To extract the size of the TV’s the tag utilised was the “title” tag as there was no specific size tag, except by clicking into each TV. It was determined that this can be dealt with at transformation.
* Loops were created to obtain the data and put into lists from row 20 to the end.
* Converted into a dataframe, dropped any duplicates and exported to csv.

***JB Hifi:***

* The key tags identified using BeautifulSoup to inspect the web address, as per the below table, to obtain the required information.
* The webpages were embedded with a Shopify plug in, which affected the scraping process.
* “Loading” buttons were identified to find the splinter dependencies.
* There was a problem scraping the information from the JB Hifi web pages due to pricing difficulties in the tags. The items that were on sale had 3 different prices, original price, discount amount and sale amount. Each of the prices were in different tags and each of them had different HTML classes associated with them. This was the most challenging aspect of scraping from JB Hifi. Please see the screenshot below.



* Due to the challenges in scraping this information, we determined to select another retailer.

***Harvey Norman:***

* Harvey Norman's website was simpler than JB Hifi and each of their TVs had just one price and all TVs were assigned the same HTML class. All the price values were inside the “span” tag and they were assigned to the “price” class inside span tag.
* Harvey Norman TV description did not have separate classes/tags for Brand, Size, etc. So, all the information needed to be extracted from names. All of that information was extracted by splitting the string based on different characters. All the item descriptions were under the “a” tag and they were assigned the “name fn l\_mgn-tb-sm l\_dsp-blc” class.
* There were 18 rows of data obtained for TV names which actually were ads for other products. They were not visible in the browser when initially searched. When those 18 rows of data were removed from the dataframe, all remaining entries were TV (120 entries). This was also supported by the fact that there were 120 image links scraped.
* All items had prices. Therefore, there was no need to filter items based on whether pricing was available.
* Harvey Norman's website did not list models of the TV when one gets to the search page. To access the model, each link needed to be clicked individually. The model was listed under the ‘small’ tag with class “product-id meta quiet p\_txt-sm”. So, a list with individual TV link was created first and then a loop was created so that each individual link would be opened and model names extracted into a new list. Both of these steps were completed using Splinter. Once the model name was extracted, it was added to the main dataframe.

*Work completed summary table:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Bing Lee** | **The Good Guys** | **Harvey Norman** | **JB Hifi** |
| ***Data Sources*** | | | | |
| Web source | <https://www.binglee.com.au/tech/tv-video/led-lcd-tvs> | <https://www.thegoodguys.com.au/televisions/all-tvs> | <https://www.harveynorman.com.au/tv-blu-ray-home-theatre/tvs-by-screen-size/all-tvs> | <https://www.jbhifi.com.au/collections/tvs> |
| VSB or Jupyter Notebook | Jupyter Notebook | Jupyter Notebook | Jupyter Notebook | Jupyter Notebook |
| ***High level info*** | | | | |
| Reviewed website: | 134 TVs | 136 TVs | 120 TVs | 110 TVs |
| Scraped data: | 134 TVs | 131 TVs | 120 TVs | 418 items |
| Structure - TV’s | Containers of items with a banner of “deals” across the top | Price of TVs not available on website when product is temporarily unavailable | Containers of items with a picture of TV and description (clickable) under the figure | Containers of items with various price offerings for each item |
| Structure - other | Header, footer, side bar (filter), comparison filter | Header, footer, side bar (filter), comparison filter | Header, footer, side bar (filter), banner advertisement | Header, footer, side bar (filter), sign up bar |
| One page or numerous pages | One page down with a “Load 36 more” and then a “Load all” button | One page with 60 items and ‘Next’ Button | One page with 40 items and ‘Next’ Button | Page numbers, each page with 36 items |
| Any 3rd party plugins? | ✖ | ✖ | ✖ | Shopify |
| Date attempted to scrape | 12/08/20 | 10/08/20 | 14/08/2020 | 11/08/20 |
| Successful scrape? | ✔ | ✔ | ✔ | ✖ |
| Date scraped | 13/08/20 | 11/08/20 | 16/08/2020 | Not successful |
| ***Dependencies*** | | | | |
| Pandas | ✔ | ✔ | ✔ | ✔ |
| Beautiful Soup | ✔ | ✔ | ✔ | ✔ |
| Splinter | ✔ | ✔ | ✔ | ✔ |
| Selenium | ✔ | ✔ |  |  |
| ChromeWebDriver | ✔ | ✔ | ✔ | ✔ |
| Sleep | ✔ | ✔ | ✔ | ✔ |
| ***Required info?*** | | | | |
| Category | ✖^ | ✔^ | ✔ | N/a |
| Brand | ✔ | ✔ | ✔ | N/a |
| Model / SKU | ✔ | ✔ | ✔^ | N/a |
| Size | ✔\* | ✔\* | ✔ | N/a |
| Details / Name | ✔ | ✔ | ✔\* | N/a |
| Price | ✔ | ✔ | ✔ | N/a |
| Currency | ✖ | ✔ | ✔ | N/a |
| Link to image | ✔ | ✔ | ✔ | N/a |
| ^ | No currency able to be scraped and category was a long string (see below) | Not all products has L1 category value | ^Listing page showing 40 entries did not have model number included | N/a |
| \* | Not in it’s own size tag, needs to be extracted from a longer string | Not in it’s own size tag, needs to be extracted from a longer string | \* equals Name of the TV. Extracted a lot of other categories from name | N/a |
| ***Attributes identified in inspecting the elements used to scrape*** | | | | |
| “Div” or “a” Attribute | a | div / li | a, href | N/a |
| Attribute / tag | bni product-link | product\_listing\_tab | name fn l\_mgn-tb-sm l\_dsp-blc | N/a |
| Category | N/a | Product Category L1 | N/a | N/a |
| Brand | data-product\_brand | input[type="hidden”] | Split string | N/a |
| Model / SKU | data-product\_sku | product-tile-model | product-id meta quiet p\_txt-sm | N/a |
| Size | title | input[type="hidden"] | Split string | N/a |
| Details / Name | data-product\_name | input[type="hidden"] | Split string | N/a |
| Price | data-product\_price | input[type="hidden"] | Price, split string | N/a |
| Currency | N/a | script/currencyCode | N/A | N/a |
| Link to image | src | img["data-src"] | src | N/a |
| ***Additional*** | | | | |
| Converted to Data Frame | ✔ | ✔ | ✔ | N/a |
| Added any required columns and data | ✔ | ✔ | ✔ |  |
| Checked for any duplicates | ✔ | ✔ | ✔ |  |
| Export to csv | ✔ | ✔ | ✔ |  |

***B. TRANSFORMATION***

*Goal:* In addition to data extracting, cleansing and checking for duplicate values, we will also be preparing our tables so it is suitable to be uploaded as a relational database. It was decided on using a relational database rather than a document-oriented database like MongoDB, as discussed in section C.

***Price:***

The price extracted from the Good Guys (csv) was in a string format and contained the “$” sign. The symbol was removed to convert that to an integer format to provide the ability to query by price range or calculate price differences by model between the retailers.

***Size:***

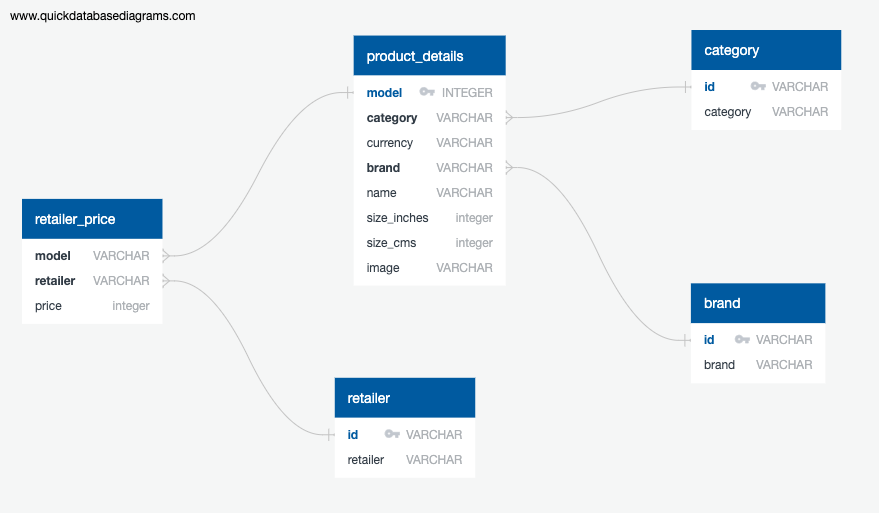
* Due to some of the differing formats, Python’s RegEx library was used to extract the television sizes.
* In the Bing Lee extraction, the title column contained a long string of brand name, model and the size. Additionally the inches notation (e.g. 60”) was within the string. The notation was also expressed in differing methods. These were extracted and removed using RegEx.
* The Good Guy’s extracted csv containing inch symbols, sizes in cms and brackets, that were removed by utilising RegEx.
* The size columns (inches & cm) are stored in numbers to help meet our project objectives - for example, to query by size range.
* Using regular expressions, screen sizes were extracted in inches from the product name. Columns were re-named from “Size” to “Size (in inches)”. Once populated, rows within the size in inches column, were examined for null values. Any null values are dropped.
* Summary of null values dropped by retailer
  + The Good Guys - none detected
  + Bing Lee - 5 to 6 rows
  + JB Hifi - not applicable
* The size was then calculated in cm in a new column called “Size (in cm)”.

***Structuring the Database:***

The database was structured as 5 tables though indexing repetitive variables such as; retailer, brand and category.

* Product\_details: contains all television details
* Retailer\_price: holds retailers id referencing retailer name in retailer table, price and model referencing model in product\_details table; it has a composite key of both the foriegn keys mentioned
* Retailer: holding retailer name and ID
* Brand: contains brand name and ID
* Category: contains category name and ID; although, it contains only one category at the moment but we decided on this indexing for a long term purpose of expansion to cover more categories other than televisions

For data types, size and price were converted to integers for ease of querying for the website in the long term. The rest of the variables were varchar yet, because of the size of the image the varchar was set to 200.

**

***Benefit of the Third Form Database:***

As the retailers and brands are contained in separate tables, it supports ease of maintenance in the long term so the database is kept uptodate. For example, if the name of the retailer has changed due to any rebranding, mergers or acquisitions, only one value would be required to update the table rather than updating all rows.

***C. LOAD***

*Goal:* To load the data into applicable formats so that a website is able to compare the information as required, based on the criteria set out as above.

A database was created using pgadmin; creating all five tables using the schema submitted mentioning all primary and foriegn keys. Then, a connection was created using a jupyter notebook, testing the connection by confirming the table names in the database.

Product\_details and retailer\_price tables were loaded using notebook, the index tables were imported using pgadmin.

***Why we chose relational SQL database:***

* **Data Integrity:**

One of the key reasons for choosing the relational database is the ability to ensure and maintain data integrity of our database. This is especially necessary as our data are scraped from websites and we need to ensure that numerical information like price and size are stored as numbers.

Going back to our project objective, we created this database for the purpose of having the ability to compare prices by product model by retailers, to retrieve product options by price range and size range, we will require these fields to be captured as numbers. A relational database will allow for accuracy and consistency in data typing and validity of our database.

* **Data Accuracy:**

Another crucial reason for selecting the relational database for our dataset is to ensure that there is no data repetition. By utilizing primary keys and foreign keys, we can make certain that data is not duplicated.

Also, we can validate and confirm the conformity of data captured (i.e. standardised naming convention). An example of this is that we are able to capture the brand Sony in the database as “Sony” and not “Sony” for some and “Sony Corporation” for others. We take advantage of the ability to capture data in a uniformed way as we move to a third form database.

* **High Security:**

In the event that the database can be expanded and monetized in the future, the relational database allows tag selected tables to be “confidential”. We will be able to control the level of access with the user ID.

***Limitation of relational database:***

We are aware that there are limitations to the relational database compared to an object-oriented database. A main limitation is that we lose an amount of **flexibility** in extending or adding new data sets that is inherent to an object-oriented database. However, given our main objective of this project and after weighing the trade-off between data integrity and data accuracy in the relational database with the flexibility of the object-oriented database, we feel that it is more important for us to be able to capture accurate and reliable data.

**D. ADDITIONAL WORK COMPLETED**

To demonstrate how such data may be utilised by a consumer, we built a template website to demonstrate how the scraped data could be used by a consumer to make a high level decision on price. In the interests of time, we did not link the website to the SQL database, but it does demonstrate the possible use case for scraping the data and making it available to a consumer.

Other information that would be useful to the consumer which was not extracted due to the timeliness of the project, would be features such as:

* Number of HDMI/output ports
* 8k / 4k / Full HD/ HD
* Smart TV
* Screen resolution
* Refresh rate
* Wifi
* Operating systems

Post the project the information will be connected to the database and loaded up as a GitHub hosted webpage.