MASTER OF SCIENCE (MSc)
AGRICULTURAL & FOOD DATA MANAGEMENT
Institut Polytechnique UniLaSalle – Campus de Rouen. France

Master´s Thesis

"To build an interactive NLP dashboard tool to understand prevalent user perspectives around the concept of #organic as inferred from Instagram posts. The tool allows us to explore word frequency, posts frequency by unique word, word co-occurrence counts, and word embeddings, which we use to deep dive into the concept of 'organic food' as a test case."

Student: Ponnappa Appanna Machimanda, MSc AgFood Data management 2022

Supervisors:
Salima Taibi – Program supervisor, MSc AgFood Data management
Alessandra Palmigiano – Full professor, VU Amsterdam

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors at the host institution, Dr Alessandra PALMIGIANO, and at the sending institution, Dr. Salima TAIBI, for giving me the opportunity to work on this research project. I have learned so much during this time which I believe will hold me in good stead throughout my future career.

Alessandra was gracious with her knowledge and expertise, providing me a direction to structure my research around. The discussions with her often opened my mind to new ways of looking at the data and capturing the intuition behind naturally occurring effects like the self-reinforcement effect, and the vector space semantics of lexical terms in language. She made me feel comfortable exploring unknown territory, which is common in a research project, while giving me a freedom to learn from my mistakes and deciding a suitable course of action at important junctures. I would also like to thank Nachoem Wijnberg for providing me an exposure to relevant projects in the field of NLP and offering more context into research in this field.

Now let me briefly describe my masters' program journey. After finding a passion for data analysis and introspecting into what domain I would like to work in, I realized that the idea of working with nature really appealed to me. I joined the masters' program at Unilasalle, Rouen, crafted by Salima Taibi and her team, which I found to have a unique and future relevant mix of data modules for agriculture. I am immensely grateful to the institution Unilasalle-Rouen, and Salima for accepting me as a student in this program.

Through the course of this masters' program, I was mentored by several notable professors who bolstered my understanding about statistical and programming concepts. I have mentioned a few people here but please note that there were also other professors and support staff in the program who significantly shaped my masters experience. Due to brevity, I do not mention them here. Needless to say, I whole heartedly appreciate the thought put into creating this masters' program and all the wonderful lecturers whose passion and expertise for their subject was shared with us students, eventually enabling a competence to carry out a project like this.

I would like to thank Jerome DANTAN, for providing guidance and coaching on the first python programs, and also being a responsible supervisor for our cohort. In all matters that involved administrative formalities during my stay in France, I would like to thank Emmanuelle GH for her tireless support despite having to attend to several international students. Rural sociology taught by Bill sparked interesting debates and a push to understanding perspectives of the different players in the Agricultural industry. The dashboard tool created in this thesis offers a window into the perspectives of the people and businesses as inferred from social media posts.

Along with an introduction to R programming, the foundational concepts about NLP (Natural Language Processing) methods using python, were taught by Redha MOULLA. It was indeed these notes and lectures on NLP that I first referred in the building of the tool.

I had the privilege of working with many (perhaps the most!) team-mates for the different projects in the program, all of whom have contributed to developing my skills in this field.

Hasnaa Latique, Uranie Jean-Louis, Robin Miranda, Elise BJ, Mouad Benkhouya, Yassine Ben-Mohamed, Marilena Charalambous, Fagr Gissan, and Malika Sekour. You guys were amazing to work with and I learned a lot from each of your unique strengths and personalities!

Last but not least, I would like to express gratitude to members of my family. I would like to thank my sister, Ashwini Ponnappa, and my parents, for encouraging me to pursue this masters' and supporting me throughout the program. I am deeply grateful and fortunate. During my time in the Netherlands, I was privileged to stay and explore the country with my cousin, Poonacha Medappa, who also gave me an immersive experience of Dutch culture. Much gratitude!

Table of Contents

Chapter 1. Introduction

Social media has made it possible for people to freely express their opinions and show their support or apprehension about certain topics. Groups of people that have similar opinions about a subject, tend to express themselves in similar ways using similar words. In this report we utilize social media data to understand how different groups of people use the word 'organic' in their relevant vocations. A dashboard tool is created for this purpose, and the entire code has been made available open source on the github repo: https://github.com/Appymp/Agridata_mining. The code has been developed in a block-wise, modular fashion and can be easily adapted for use in future projects.

This is not simply a positive or negative sentiment analysis but rather an approach to intuit a broader meaning. For example, as we will find out later in the results section, "organic" assumes certain connotations in the context of 'cosmetics' and a different one in the context of 'food'. The groups of people creating this definition are the ones posting on social media.

In this study we have used data specifically from Instagram for posts where the user has added a hashtag '#organic'. So it is implied that the post has something to do with organic but not necessarily based on the same literal meaning. The text description entered by the user contains valuable information about what the user thinks "organic" is about, or what it is being portrayed as 'organic' by the user.

Broadly there are 2 categories of people that post content on Instagram:

- users that create posts for business purposes,
- users that do not have a business interest but more of passionate association.

We do not explore classification of the users in depth, but we do create an option for detecting a possibly influential post through a likes per minute count.

The dashboard tool we have created is geared towards an analyst who not only relies on the standard graph analysis approach of making sense from the data (usually sequentially), but who can also use the multiple graphs in a back-and-forth manner, to confirm and refute the assumptions being made. There is expected to be a learning curve where one can develop their own approach of using the tool as well. Subject matter and marketing experts in the domain can use this as a discussion and exploration tool to draw more general and intuitive inferences about their target market and audience.

The tool consists of word clouds for studying the frequency of words and frequency of posts-by-word, bar charts for co-occurrences and post counts, and word2vec based vector space graphs for checking "similarity" of a particular word through the context expressed in the posts. The data rendered in these graphs can be dynamically selected from the data table filtering and graph options:

- row selection and single or multiple column combinations of description, hashtags, usernames, caption mentions, weblinks
- word filtering of multiple columns

- influential post filtering with z-score slider from 0 to 4. Higher value indicates a higher likes count in a unit of time.
- zoom-in and out of the graphs and saving the images to file
- Sliders for setting the number of results to display in the bar charts and vector space graphs.
- Input boxes for words displayed as vectors and to check co-occurrences

Chapter 2: Literature review

In order to obtain the meaning of the words we will create word vectors from the descriptions of the Instagram posts. Each word in the description is represented as a word embedding. Word embeddings are fundamentally a form of word representation that links the human understanding of knowledge meaningfully to the understanding of a machine. The representations can be a set of real numbers (a vector) (Selva Birunda & Kanniga Devi, 2021).

These vectors can be created using different methods ranging from the more transparent matrix factorization methods, to the more mysterious and better performing neural network models. In the neural network models like Word2vec, the actual algorithm is designed to locally predict the surrounding words and as a byproduct of this task, word embeddings are created which capture a lot of semantic information.

We explore some of these methods and the feasibility of implementing them in our use case. As pointed out by (Levy et al., 2015, p. 211) performance gains of word embeddings are due to certain system design choices and hyperparameter optimizations, rather than the embedding algorithms themselves. The embeddings hinge upon the distributional hypothesis by (Harris, 1954), where the degree of semantic similarity between two words (or other linguistic units) can be modeled as a function of the degree of overlap among their linguistic contexts.

The format of distributional representations vary depending on the specific aspects of meaning they are designed to model (Baroni & Lenci, 2010). At the core of these are the word co-occurrence counts. It is useful to identify these representations on the basis of what kind of similarity they are modelling. The concept of *relational similarity* and *attributional similarity* are significant in this regard (Gentner, 1983). As (Turney & Pantel, 2010) point out, the attributional similarity between two words a and b, depends on the degree of correspondence between the properties of a and b and the relational similarity the degree of correspondence depends on the relationship of the pairs a:b and c:d. Consider the example of *man* and *woman*. These have an attributional similarity because they are examples of humans. Whereas in the cases of human : speak, and dog : bark there is a relational similarity because they are examples of communication techniques. The relational similarity can be understood as “analogies”.

Distributional semantic models (DSMs) are particularly successful in tasks like synonym detection and concept categorization because these words share many properties (attributional similarity), such as synonyms and co-hyponyms (Baroni & Lenci, 2010). Their paper also describes how relational similarity is more suited to concepts related by hypernymy. Synonyms are words that have exactly or nearly the same meaning like

'intelligent' and 'clever'. A hyponym is a subconcept of a broader term, and co-hyponyms are subconcepts of the same broader term like 'rose' and 'daisy' are both 'flowers'. A hypernym is the broader meaning of a particular word, like 'flower' is the hypernym of 'rose'.

In the following section we discuss the techniques for creating embeddings:

1. LSA
2. Glove
3. Word2Vec

2.1 LSA

One of the techniques of creating vectors uses a term-document matrix and is called LSA (Latent semantic analysis). Here the main objective is to identify the similarity in documents but it can also be used to find similarity between terms (words). As the paper by (Dumais, 2005) points out, there are mainly 4 steps in the analysis:

2.1.1 Term-document matrix:

This a matrix of counts with words as rows and the document or sentence as the column. The count of the words within the document or sentence are populated in the matrix. The order of the words does not matter and hence the terms "bag of words" is used to describe this method.

2.1.2 Transformed term-document matrix:

The raw word frequency counts are transformed cumulatively in a sublinear (log) function and inversely with the overall occurrence of the term in the document (inverse document frequency).

2.1.3 Dimension reduction:

A singular value decomposition (SVD) is performed to reduce the matrix to smaller dimensions by retaining the largest 'k' singular values and setting the remaining to 0. The vector space derived by the SVD now contains vectors of the words and documents.

2.1.4 Retrieval in reduced space:

Similarities are computed between words and documents in the reduced dimensional space rather than the original matrix. If a search is made to retrieve documents, then a new vector is formed using the constituent term vectors, and this is compared with the document vectors to find a match.

In essence this model captures information about the document as a whole and through this process it also gathers information about the different words. Each document has a vector defined by the words which have maximum representation in the particular document. The drawback of this as pointed by (Landauer & Dumais, 1997) is that the context window for the words is the size of the document (large), and the best performing algorithm was found to have a small context window of 5 with the center word as the target. This is the only pure "count based" approach we explore, which does not involve a neural network under the hood. This is widely considered as a global matrix factorization method.

We do not implement this model because we are not interested in identifying what a post is specifically about. Rather, we want to understand what a specific word means as a representation of other words. So, in our case a word-word co-occurrence matrix would be apt. However, we carry forward the idea of using a context window of 5 words in our co-occurrence matrix.

2.2 Glove

In the previous method, LSA, word analogies are not possible due to the typically large document size and loss of local context information. The Glove method, short for global vectors, introduced by the team at Stanford (Pennington et al., 2014) tries to leverage the global document statistics like LSA but instead of term-document matrix it relies on the use of a word-word co-occurrence matrix:

- The method hinges upon the idea of comparative probabilities as in the paper by the Stanford team. Consider a word 'k'=[solid, gas, water] and its co-occurrence with 2 other words 'i'=steam and 'j'=ice. If a word is similar to both words like 'water':{'ice','steam'} or dissimilar to both words like 'water':{'dog','cat'} would result in both high probabilities or both low probabilities which would approximately equal to the value 1. Whereas for relations like 'solid':{'ice','steam'} or 'gas':{'ice','steam'} we can see that probability of one of the pairs in each of these relationships will be high resulting in a value greater than one or lesser than 1.
- Words with no discerning ability like the one which are used in all word contexts 'a','the' etc, called "stopwords", and also other words which are not relevant to the target word, are identified with the use of global statistics.
- The main issues in considering the co-occurrence matrix for both global and local statistics are:
    - There is no direct equation to represent the relation between the 3 word vectors and their relative probabilities of occurrence. Some function has to use the 3 words as arguments. This would be important in the calculation of analogies.
    - The word vectors are high dimensional vectors whereas the probabilities are scalars. This has issues in computation.
    - Since are 3 words involved, the loss function computation for this is tricky.

In comparison with the LSA model, this is similar in that it is also a matrix of counts. However, there are no transformations which takes place on the matrix. These counts are used as is. The complexity reduction is mainly executed by the logic in the weighting scheme. SVD as used in LSA sees decreased performance with a larger corpus.

In our project, some of the main benefits of the global document statistics like "stopwords" are addressed by their explicit treatment and removal using existing stopword indexes. Also, since the language use in social media tends to be more dynamic, the local statistics may be the foremost consideration rather than global. New posts are constantly coming in, and an ideal data pipeline must include a better alternative than repeatedly creating computationally expensive global co-occurrence matrixes.

2.3 Word2Vec

Finally, we explore Word2vec introduced by ex-google's (Mikolov et al., 2013) which is the model we implement in this project. We use the word2vec implementation from Gensim package (*Models.Word2vec – Word2vec Embeddings — Gensim*, n.d.). New posts (text information) can be added to an already trained model to update the weights and create new embeddings. Since this is based on a neural network architecture, the dimensions representing each vector are trained in a linear time complexity (very fast). Listed below are some of the significant details as relevant in our use case.

- We use the SGNS (skip gram negative sampling) algorithm, setting the window size to 5 and the vector size to 300 dimensions. In the study behind the glove method (Pennington et al., 2014), the vector size and window size were found optimal for values of 300 and 5 respectively (figure below).

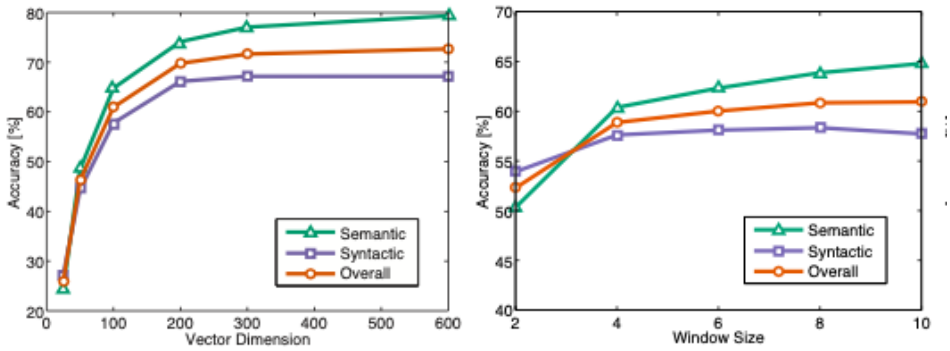

Figure 1: Vector dimension and window size selection (Pennington et al., 2014)

- The skipgram model is tuned to predict the context words from the center word. This is found to be better at creating word embeddings for similarity than the CBOW (Continuous bag of words) model which predicts the center word based on the context words.
- In this model, the words in a context window are weighted by probability of occurrence as opposed to the method in glove where it is harmonic function (Levy et al., 2015). It is also faster to train than glove.
- Words larger than some threshold are randomly removed through a process called subsampling. In this way, stopwords are also implicitly removed.

We will use the word2vec model to identify most similar words and also visualize the words in a 2-dimensional space. The closer the words are in this plot, the more semantically similar they are assumed to be. Here we will have to use our discretion in drawing conclusions from potentially "similar" words because they may be similar in aspects other than synonymity as well.

2.4 User types

Different groups of people use different language styles to describe their content. In real life we see an obvious distinction in the way a teenager speaks versus the way an individual in his late 20's and older would speak.

A consumer may express an interest or passion towards a certain idea through the usage of certain hashtags and the post description (text). Whereas an influencer or business may use the same hashtags along with a call to engage, like requests to follow, share etc.

An influencer could be influential purely due to celebrity status gained from another avenue like movies, sport etc. Parts of their fanbase can be quick to resonate with certain values and ideas expressed by them. An influencer could also be someone who offers credibility to the post because of a strong track record of selection through review and study, in the specific domain.

On reviewing the existing literature on user classification based on social media posts, we find an in-depth study based on health related social media post data. (Rivas et al., 2020) derive classification models for age, gender, ethnicity, and location based on data from twitter, google +, and specific patient drug forums. These involve manual classification techniques. We do not pursue classification of the posts as the main objective of this study, but we include an option in the dashboard tool to pursue manual classification in the eventuality that we train a model based on user classified data. However, an indicator metric of influential posts is created in this project called likes_per_minute, which is considered in a bivariate setting along with time elapsed, to filter out (influential) posts which are greater than a certain z-score threshold. Here it was found important to factor time_elapsed with the likes per minute because it seems to have an effect on it.

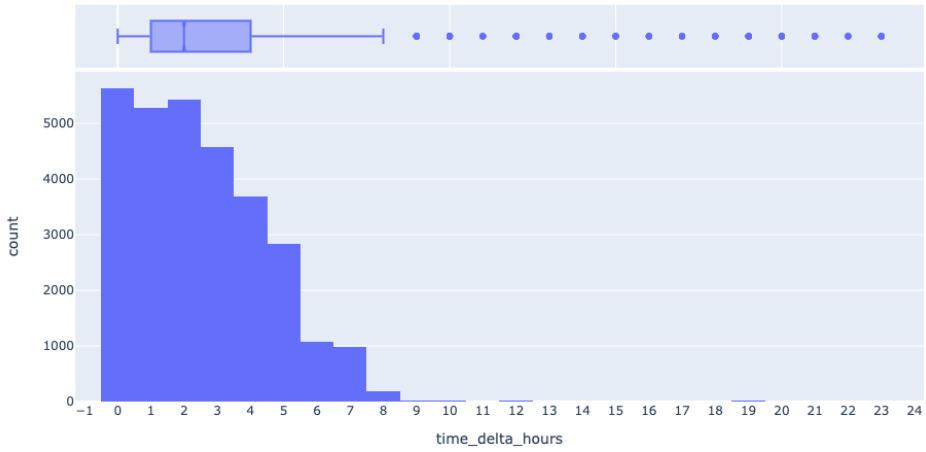

Figure 1: Count of posts by elapsed time delta

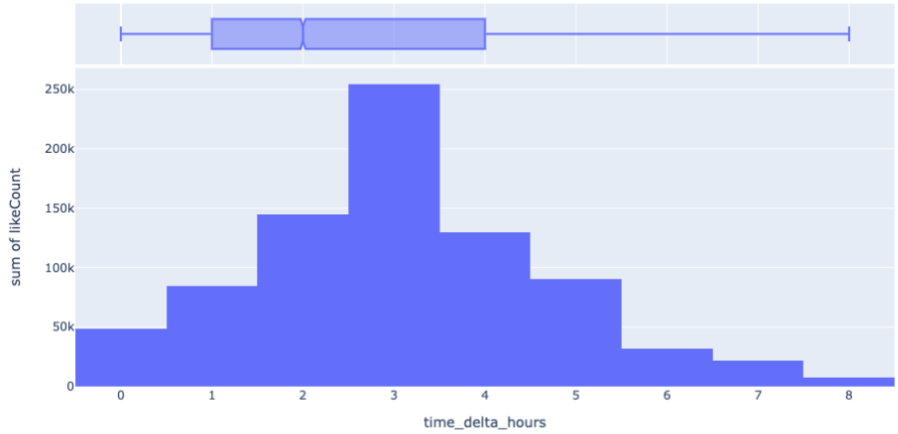

Figure 3: Sum of likes by elapsed time delta

In figure 3, we can see that the sum of likes increases till time_delta of 3 hours and after that it starts reducing. The reduction can be explained by decreased number of posts. The increase seems a little strange because the count of posts (figure 2) falls approximately in the same range till 3 hours, and in fact dipping in count of posts from the 3 hour time delta mark.

When we look at figure 4, we see that the sum of likes is pronounced for a time delta of 3 hours because of a very popular post which has over 150k likes. To reduce the effect of such a large outlier on the selection of "influential" posts in the entire corpus, we can factor the time delta in bivariate relationship. So this balances the effect of the scraping time on detecting influential posts. However, since we have categorized the posts in terms of hours, the time_elapsed resolution of a post for optimal segregation using the bivariate z-score, is 1 hour. Posts which are less than an hour may be more sensitive to time_elapsed rather than actual "influence".

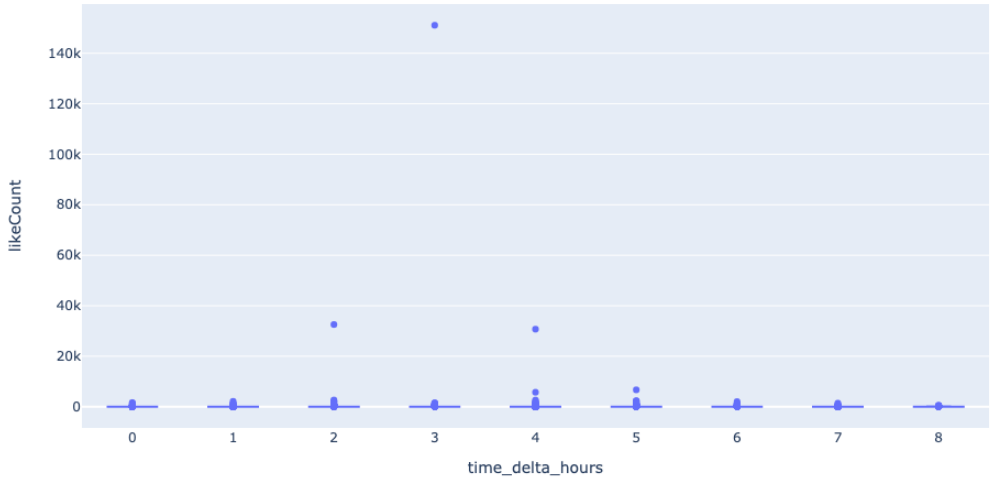

Figure 4: Outliers in like count by time delta

3: Methodology

In this section, we will discuss the bulk of the work done in terms of the python coding for cleaning the data, the data pre-processing steps, and the method of selecting and tuning the models.

3.1 Data scraping:

Sourcing the data was the first step of the research. Instagram currently does not have a publicly accessible data API from where we can query the data. So other methods of data collection were tried and the best method was chosen for the collecting the large data corpus. Also there were found to be no existing banks of data containing the Instagram post level data that we required.

In our project we wanted to obtain recent data containing the Instagram post descriptions for posts containing a particular hashtag (#organic). A post would normally contain this

hashtag if the image or video is relevant to what the #hashtag means. The meaning of the hashtag is subject to the users perception of what it means. Since language is constantly evolving, although a word might have a limited scope of understanding in its initial usage, the meaning can grow to accommodate a broader scope as the natural usage by the user evolves. Here we list out the different methods attempted and describe in more detail the method which was finally used.

3.1.1 Selenium:

First, coding a scraping bot using Selenium python package was considered. This would interact with the objects of the web page and execute the manual tasks of clicking and viewing the data (Python Simplified, 2020). The data from Instagram has to first load in the browser, by viewing it in page, and then the required objects can be populated into a data table. This process was found to be too time consuming to obtain a large text corpus for our research and would directly violate the terms of service of Instagram which forbids any bots on the platform.

3.1.2 Steviesie data platform:

Here the user has to access the pages manually on a web browser (like normal) and make sure to scroll through the required posts. Then, using the inspect tool of the web browser (Google Chrome in our case), we download the .HAR file which contains a mirror representation of all the data objects that has been displayed on the screen. This .HAR file is then uploaded to the Steviesie data platform (*Scrape Data Legally | No-Code Data API & Website Scraper | Stevesie Data*, n.d.), which parses the data and creates data tables out of similar types of objects. So if we require the “post” data, then we just need to select the relevant output from the parsed files on the platform, and then download the file in .csv format. However, even this method was found to be time consuming because of the same buffering bottlenecks of the previous method, but with an added element of manually having to scroll through the posts.

3.1.3 Phantombuster data platform:

Finally, a commercial method was found which provided a fully automated scraping solution for Instagram data. Selected hashtags were queried and on each run about 5000 posts were retrieved in a span of about 15 minutes. Observations were made with regard to the data scraping velocity of this tool.

Hashtag popularity on the platform (Instagram), affects the number of posts retrieved. For example, 3 hashtags were scraped at different run times to check the span of time covered by the retrieved posts. The table below describes the time of each of a few runs, which we then use to estimate the ideal automation parameter for the scraping bot.

Table 1: Data scraping time overlap

	Time stamp of first and last post scraped (hh:mm)				
Hashtag| Date	**11Aug_1220**	**11Aug_1038**	**11Aug_0020**	**10Aug_1435**	**9Aug_1914**
#organic	10:19 to 3:18	8:38 to 00:43	22:19 to 16:50	12:35 to 13:55	17:14 to 14:36
#sustainable	10:31 to 8:17	8:49 to 16:15(pd*)	22:31 to 19:05	12:43 to 20:21	17:16 to 7:01
#fairtrade	-	9:02 to 10:31(pd*)	22:31 to 10:33	12:55 to 10:33	9-17:26 to 6-15:53

*pd-previous day

In the above table the column headers refer to the date and time of running the query and the rows represent the different hashtags. From the above table we can infer the following:

- The more popular a hashtag is, i.e it has more posts, then there are more posts scraped within a certain unit of time. The spans of time for 5000 posts of each of the hashtags are:
  - #organic 7:00 hrs, 7:55 hrs, 5:29hrs
  - #sustainable 16:34 hrs, 16:23 hrs
  - #fairtrade :~106:00 hrs but only up to 3 days of previous data available.
- As seen above #organic is most popular of the selected hashtags. Since each run can pull 5000 posts, how can we automate the pull requests so that we can cover all times of the day:
  - If we set the bot to automatically query the data 4 times a day, then every 6 hours the new data is pulled. Since each query covers around 7 hrs of #organic posts, we will have a seamless pull of data across the day, with possibly a 1 hour duplication due to overlap.
- # sustainable contains 5000 posts across 16 hrs. So it is roughly 2.5 times less popular than #organic in terms of posts per unit time. #fairtrade however has the lowest number of posts per unit time and hits the Instagram backtrack query limit of 3 days before 5000 posts are pulled. Total posts containing hashtags at time of data collection:
  - #organic : 52.6M
  - #sustainable : 11.4M
  - #fairtrade : 3.4M

Conflict of Instagram session while data is being pulled:

While the phantombuster data scraping tool is carrying out a scheduled query, if the user (me) accesses Instagram on another device (like a smartphone), then instagram blocks the user ID with a warning. This is because Instagram does not encourage running a bot in a session. However, the data itself is openly available if a user chooses to manually access it. So it is technically not private data we are scraping. Instagram allows for an API to fetch the data if there is a commercial interest so phantom buster is a noted scraping player which is sometimes advertised on Instagram itself. Since the dashboard created here is not yet a commercial product, usage of this tool was halted after the first warning without pursuing any extended data collection.

Data fields contained in the queried data:

The below table describes the different fields available and provides an example of the data. Explanations of the field types are self explanatory but some clarifications are listed below.

The data table produced from the phantom buster tool consists of the following fields:

Table 2: Example data fields scraped from Phantombuster

Field	Example
profileUrl	https://www.instagram.com/myhomefarming
username	myhomefarming
fullName	Home Farming
commentCount	0
likeCount	1
pubDate	2021-08-10T12:34:56.000Z
description	Planting sweet potatoes using a rice bag. In the future, I will show you how I do it step by step. It is easy for home farming. #urbanfarming #gardening #hydroponics #urbangarden #urbanfarm #growyourownfood #organic #hydroponic #urbangardening #organicgardening #agriculture #microgreens #urbanfarmer #homegrown #vegetables #homefarming #sayurorganik #sayur #kebun #kebunrumah #healthyeating #sustainablelifestyle #vegetablegardens #sweetpotatoes
imgUrl	https://scontent-lhr8-1.cdninstagram.com/v/t51.2885-15/e15/234902054_1511876039156842_3394796396722981677_n.jpg?_nc_ht=scontent-lhr8-1.cdninstagram.com&_nc_cat=110&_nc_ohc=Jo3LSzWf7coAX-T5FL6&edm=ABZsPhsBAAAA&ccb=7-4&oh=e026bdfd0c9656b16fbc85b7e76b6fb3&oe=61186640&_nc_sid=4efc9f&ig_cache_key=MjYzNzIwMTA5NTEwNzAwMDQ0Ng%3D%3D.2-ccb7-4
postId	2637201095107000300
ownerId	48877898830
type	Video
query	#organic
timestamp	2021-08-10T12:35:19.268Z
videoUrl	https://scontent-lhr8-1.cdninstagram.com/v/t50.2886-16/234259757_428427111750827_6156703384781018693_n.mp4?efg=eyJxZV9ncm91cHMiOiJbXCJpZ19wcm9ncmVzc2l2ZV91cmxnZW4ucHJvZHVjdF90eXBlLmZlZWRcIl0ifQ&_nc_ht=scontent-lhr8-1.cdninstagram.com&_nc_cat=111&_nc_ohc=qitd3o_ju5YAX8wm9p0&edm=ABZsPhsBAAAA&ccb=7-4&oe=61149592&oh=073607c29be71b62078e62757cdfce00&_nc_sid=4efc9f
viewCount	5

- Each row of the data table refers to a post. The post can either be an image or a video. If it is a video, then there is an additional field called 'viewCount' which may be populated.
- 'commentCount' and 'likeCount' are engagement metrics. This can indicate the popularity of the post with other users. A user can either write a comment on the post or hit the "like" button for the post.
- 'pubDate' refers to the date and time that the post was posted.

- ‘description’ is the text field which is what we use to infer embeddings in our project.

3.2 Data preprocessing:

This section deals with all necessary preprocessing in order to get the data to a form in which we can carry out analysis.

3.2.1 Dataframe_1

The first dataframe is created by combining the outputs of the individual query runs from the phantombuster data tool. This dataframe includes all the different hashtags as well, which we had used for testing the time span of posts (including #fairtrade #sustainable). The python code (appendix python code line 22) takes these different files and concatenates them along the same columns. If there are no new individual queries and the old combined dataframe is the latest, then the old dataframe is loaded into the program.

- There were a total of 87,423 rows in this dataframe.
- Includes the hashtags #organic, #sustainability, #fairtrade.
- A total of 16 datasets were combined.
- This raw dataframe included 351 empty rows. So there are a total of 87072 non-empty rows in the raw dataframe.

3.2.2 Dataframe_2

From Dataframe_1 the description column is then treated to extract relevant data into its unique columns. This creates the Dataframe_2. The new columns created are:

- hashtags – consists of the hashtags. Some hashtags are not preceded by a space. So it is first treated to include a space so that text splitting functions can be properly applied.
- cap_mentions – this consists of the users that have been mentioned preceded by an ‘@’ character.
- web_links – any websites mentioned in the description will be populated in this field.
- clean_captions – this consists of the plain text after all the previous types of text have been extracted into their relevant columns.

3.2.3 Dataframe_3

Next, the clean_captions column which contained just the text data was processed with the following transformations:

- font_uniformity – the encoding of different fonts causes mismatch of word representations. So the fonts have to be made uniform. *'caption_processed’* column.
- convert_lower_case – Next all the words are converted to lower case. This ensures that the words starting with a capital letter are represented properly. *'caption_processed_2’* column.
- remove_punctuation – All special characters are then removed. This incude "!\"#$%&()*+-.•/:;<=>?@[\]^_`{|}~\n". *'caption_processed_3’* column.
- diff_encodings – Sometimes the punctuation marks like the triple dot (...) and inverted commas (‘) are of a different encoding. These had to be removed separately. 'caption_processed_4’ column.
- Emojis – after carrying out the above steps independently, emojis still remained. These were removed using the “preprocessor” library before removing stopwords.

For identifying the different treatments that were required on the data, the dataframe table had to be explored in the table form. Exemplars of a certain row which had unique fonts and punctuations were selected and viewed as test cases to check if a particular transformation was working. Shown below in figure 5, are the columns after the relevant preprocessing steps.

	clean_captions	caption_processed	caption_processed_2	caption_processed_3	caption_processed_4
11	Health is precious - protect it. For more information contact: Shatayu Ayurved - 9826092380. . . .	Health is precious - protect it. For more information contact: Shatayu Ayurved - 9826092380. . . .	health is precious - protect it. for more information contact: shatayu ayurved - 9826092380. . . .	health is precious protect it for more information contact shatayu ayurved 9826092380	health is precious protect it for more information contact shatayu ayurved 9826092380
13	Shatkratu is one of the 10 sacred flowers (Dashpushpam) in Kerala, also called "Love in a Puff". Enhances hair growth, thickening & reduces hair fall strengthening the hair roots. . .	Shatkratu is one of the 10 sacred flowers (Dashpushpam) in Kerala, also called "Love in a Puff". Enhances hair growth, thickening & reduces hair fall strengthening the hair roots. . .	shatkratu is one of the 10 sacred flowers (dashpushpam) in kerala, also called "love in a puff". enhances hair growth, thickening & reduces hair fall strengthening the hair roots. . .	shatkratu is one of the 10 sacred flowers dashpushpam in kerala also called love in a puff enhances hair growth thickening amp reduces hair fall strengthening the hair roots	shatkratu is one of the 10 sacred flowers dashpushpam in kerala also called love in a puff enhances hair growth thickening amp reduces hair fall strengthening the hair roots
23	She's here! Judith is here! She has cucumbers, MELONS, courgettes, herbs, chard, plums, beetroot, MELONS... did I mention MELONS!? I'm very excited about this little collab. She's a star and I can't wait to see what she brings to She's already sold a big bag full to a regular so you'd better get down here quickly!	She's here! Judith is here! She has cucumbers, MELONS, courgettes, herbs, chard, plums, beetroot, MELONS... did I mention MELONS!? I'm very excited about this little collab. She's a star and I can't wait to see what she brings to She's already sold a big bag full to a regular so you'd better get down here quickly!	she's here! judith is here! she has cucumbers, melons, courgettes, herbs, chard, plums, beetroot, melons... did i mention melons!? i'm very excited about this little collab. she's a star and i can't wait to see what she brings to she's already sold a big bag full to a regular so you'd better get down here quickly!	she's here judith is here she has cucumbers melons courgettes herbs chard plums beetroot melons did i mention melons i'm very excited about this little collab she's a star and i can't wait to see what she brings to she's already sold a big bag full to a regular so you'd better get down here quickly	shes here judith is here she has cucumbers melons courgettes herbs chard plums beetroot melons did i mention melons im very excited about this little collab shes a star and i cant wait to see what she brings to shes already sold a big bag full to a regular so youd better get down here quickly
106	Brownie balayage .	Brownie balayage .	brownie balayage .	brownie balayage	brownie balayage

Figure 5: Caption processing of exemplars

3.2.4 Language detection

A language detection library was used to identify the language of a particular caption. Here an offline model called "langdetect" (Danilak, n.d.) was used and it took 11:30 mins for the classification of 87072 posts captions on a macbook air 2017 laptop system. Other models involved an online api method of language classification (textblob which uses a google api) but seemed to throw a "too many requests" error.

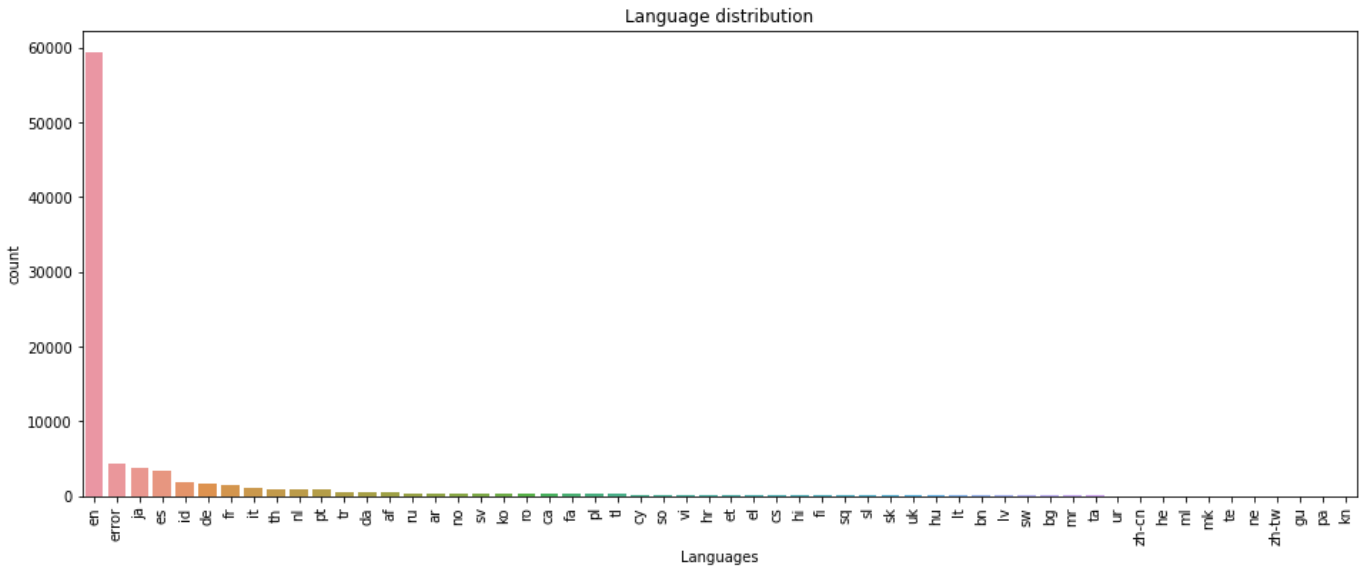

Figure 6: Language countplot of initial data

As seen in the above countplot, figure 6, most of the posts are in the English language. The second highest categorization are the errors which are mainly due to there being no data in 'clean_captions', after separating all other textual data like 'hashtags', 'web_links' etc. into their respective columns. The errors also arise if there are only *emojis* left in description after all other

text data has been separated into their columns. These error rows may contain useful hashtags, but since they do not make sense sequentially (like descriptions using a certain language), they can be safely discarded.

- The languages short forms are as per the list of ISO 639-1 codes ("List of ISO 639-1 Codes," 2022).
- We can see that the top languages on this dataset apart from English are: Japanese, Spanish, Indonesian, German, French, Italian, Thai, Dutch, Portugese, and Turkish.
- The languages gives us an indicator of which countries are more interested in the concept defined by these hashtags (#organic, #sustainability, #fairtrade). Since English can be considered as a global language, the only reliable inferences we can make are that
    - Users in Japan and Spain have high interest.
- For the purpose of using the text to interpret meanings, we will have to use a single language. The language selected was English which formed the bulk of posts. The dataframe was filtered to only English posts. This leaves us with 59374 rows of data. This is a drop of 33% from the preprocessed "Dataframe_3".

3.2.5 Further filtering

After the above transformations, it was decided to focus on only 1 type of hashtag to maintain a uniformity and depth of inference. The dataframe was filtered to only #organic which makes up most of the dataset.

- #organic posts made up 40198 posts. This is again a drop of 33% from the "English" only posts.
- Finally, duplicate posts (based on postId) were removed which consisted of about 5067 posts. After that duplicate posts based on same description were eliminated which amounted to another 4891 posts. The remaining dataset contains 30240 rows.

3.2.6 Stemming and Lemmatization

Stemming is the process by which a particular word is reduced to its root word ("Introduction to Stemming," 2018). For example the words "chocolates", "chocolatey", "choco", are reduced to the root word "chocolate. We should keep in mind the errors of stemming that can occur, namely over-stemming and under-stemming (Heidenreich, 2018).

- Over-stemming occurs when too much of a word is stemmed to a nonsensical term or different words being reduced to the same stem. Consider the words university, universal, universities, and universe; if these are reduced to the stem term "univers", then over stemming has occurred.
- Under stemming is the opposite issue where words which are actually forms of each other do not resolve to the same stem.
- A Porter stemmer from NLTK package was used on this dataset. This is considered as a gentler algorithm which offers good reproducibility and dates back to the 1980's.

Lemmatization is a more calculated process compared to stemming. Here the words are reduced based on their dictionary forms rather than by their common starting letters. For example, words like "is" and "are" can be reduced to "be".

- Lemmatizers require a lot more knowledge about the structure of the language and hence tends to be more intensive.
- The Wordnet lemmatizer from NLTK was used on this dataset.

3.2.7 Word Vocab

All the above steps were carried out to bring the text data into a tokenized form with which word co-occurrences and word embeddings can be created. The number of unique words (vocab) in each of the different processing steps are:

- 'cp4_no_emojis' contained 46066 unique words after general preprocessing of converting to lower case, removing punctuation, and removing emojis.
- 'cp4_rm_sw' is the same as the previous column with the added treatment of removing the stopwords. 45925 unique words remained.
- 'rm_sw_stem' , after running Porter stemming on 'cp4_rm_sw' reduced the vocabulary to 34051 unique words.
- 'rm_sw_lemt', after running Wordnet lemmatizer on 'cp4_rm_sw' resulted in an output of 41526 unique words.

Lemmatized text is more responsive to different part of speech, and hence is the more nuanced algorithm we have considered. Although the goal was to reduce the vocab to a saller size for easier computations, we have to still use the lemmatized text column (with more words than stemming algo) as the input to our Word2vec algorithm.

3.3 Word Embeddings

We explore 2 different methods of making word embeddings. The first method uses a traditional distributed semantic model (DSM), and in the second method "word2vec" they are formed using a neural network. We end up choose the word2vec model over the DSM because they significantly outperform the DSM model (*On Word Embeddings - Part 3*, 2016).

The *vocabulary* is a list of unique words of text from the dataset. This corpus of text is obtained from the lemmatized column 'rm_sw_lemt' which also omits stopwords and contains other preprocessing as described in the previous section.

Stopwords removal is an important aspect. *Stopwords* are a set of commonly used words in a language. The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead (Ganesan, 2014), and reduce computations.

Each word in the vocabulary (vocab), is represented as a vector and then reduced to 2 dimensions to give us a visual representation on a 2D plot.

3.3.1 DSM

3.3.1.1 Co-occurrence matrix

With the token word at the i^{th} position, a rolling bidirectional window is used to define the co-occurring words with 2 words in the front and 2 words behind the token word. The token word is iteratively selected from each word of the row, and is paired in a tuple with the surrounding words within a 5 word window. Then the counts of co-occurrence are updated as the value to the unique (word,word) tuple pair key of a dictionary.

- The vocab consists of 41526 words resulting in a co-occurrence array of shape 41526*41526. So each word is represented as a vector of 41526 dimensions.
- This is a sparse matrix (many redundant 0's) which makes it computationally very expensive.
- These vectors must be reduced to 2 dimensions while preserving the relationship of the words in the multidimensional space.

- For this an SVD matrix factorization can be performed to identify the most significant contributors to a vector's identity and represent it in 2 dimensions.

3.3.1.2 SVD matrix

To address the computational cost of working with sparse matrixes of very large dimensions, we can use the COO formatting method in sklearn library (*Scipy.Sparse.Coo_matrix — SciPy v1.7.1 Manual*, n.d.). Without this it is not feasible to perform matrix factorization operations. The advantages of using this format are:

- facilitates fast conversion among sparse formats
- permits duplicate entries
- very fast conversion to and from CSR/CSC formats (compressed sparse row/column)

This matrix then has to be upcast to a floating point format before carrying out the SVD.

We use the truncated SVD method from the sklearn library to decompose the multidimensional matrix into 2 dimensions. This method is also called LSA (latent semantic analysis). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently *(Sklearn.Decomposition.TruncatedSVD, n.d.)*. The fit_transform method is applied on the preprocessed co-occurrence array (COO formatting) and the output is an array with 2 columns and the number of rows defined by the vocab – shape (41526,2).

3.3.1.3 Plotting

The co-occurrence matrix contains the index and columns of the vocab in sorted order. This means that when we obtain the SVD matrix the index is in the sorted order here as well. Since the SVD matrix does not contain the actual word, the vocabulary to plot must be referenced by the index value. So for example if we want to plot a word which is in the 5th

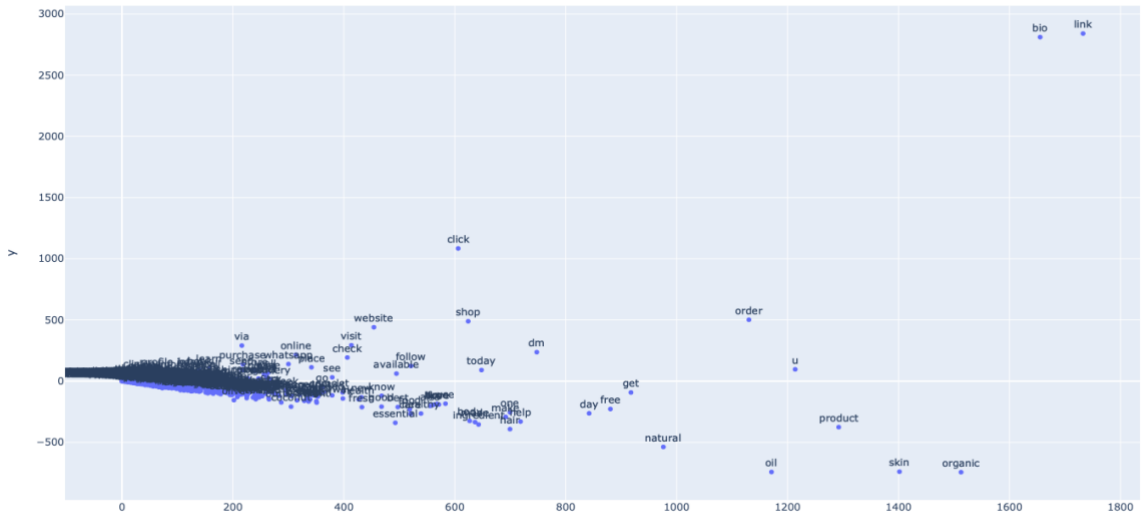

Figure 7: Co-occurrence SVD for window size 5

index location of the sorted vocab, we need to retrieve the 5th row of the SVD matrix. A dataframe is created of 3 columns with the word as one of the them and the 2 dimensions from the SVD as the x and y axis values. Then the dataframe is filtered to the words which are required to be plotted and this is passed to the plotting function. Shown below is the plot with all the words.

As seen in the below plot, most of the words are very densely concentrated on the left-hand side and it funnels out towards the right. The right hand side, which have words that are much further away in terms of distance, is indicative of the words which occur most from the co-occurrence values.

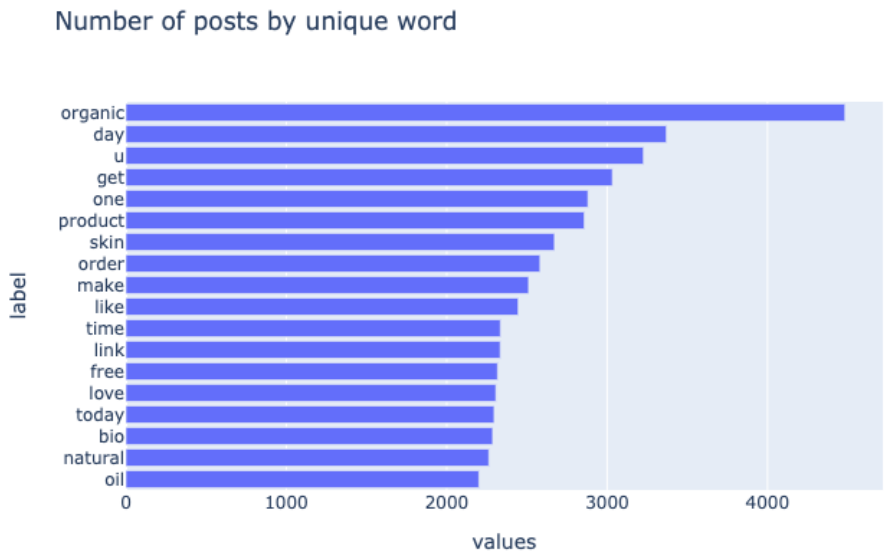

Figure 8: Count of posts by word

In the above bar chart, figure 8, the top results are the same words which are spread out towards the right-hand side of the plot in figure 7. The words seem to have enough information to assume a more specific position in the 2-dimensional representation.

Since most of the words are not adequately separate from each other, we cannot infer much from their 2D representations. It was noted that the model could have been improved by lowering the weights of words further away in the window, setting activation functions, and performing more mathematically heavy processing, in order for the vectors to gain better real-world meanings. However, we did not investigate further into this area because the method was computationally expensive.

3.3.2 Word2Vec model

To overcome the lack of context association between the words in the previous model, we use Word2Vec. Here the algorithm uses a large amount of unannotated plain text to learn the relationship between the words automatically (*Making Sense of Word2vec | RARE Technologies*, n.d., p. 2). The output are vectors, one vector per word, with remarkable linear relationships that allow us to do things like vec(“king”) – vec(“man”) + vec(“woman”) =~ vec(“queen”). This seems intuitive to us in our normal usage of these terms in everyday language. The relationships depend on the training corpus. The example mentioned here pertains to a dataset where the context of king and queen have been sufficiently represented with different context words so that it intuitively represents the relationship.

In our dataset however, the words king and queen are not used in normal contexts. So this does not yield the same result. This is the main idea we will leverage in using this model for the Instagram social media data. We use this to identify the underlying meaning of a

particular concept based on the usage of language in these contexts. Since our entire text corpus consists of small summarized pieces of text (descriptions) based on some notion of "organic", we can assume that the relationships and the implied meanings, apply specifically within a particular domain of selection.

3.3.2.1 Instantiation

We will use the word2vec implementation from the gensim package. One of the benefits of this implementation is that it uses optimized C routines, data streaming and Pythonic interfaces. The training is streamed, so ``sentences`` can be an iterable, reading input data from the disk or network on-the-fly, without loading your entire corpus into RAM (*Gensim*, n.d.).

The code was implemented in a Spyder python console and the model was set with the following hyperparameters:

- The skip gram model was used. Here the context words within a window are probabilistically predicted based on the center word.
- vector size is set to 300. This means that each word is represented as a vector with 300 dimensions.
- Window size was set to 5.

3.3.2.2 TSNE transformation

With the model instantiated, a TSNE transformation is performed on the vectors. TSNE stands for T-distributed stochastic neighbor embedding. The algorithm maps multi-dimensional data to two or more dimensions, where points which were initially far from each other are also located far away, and close points are also converted to close ones (Smetanin, 2018). This is analogous to the SVD transformation in the DSM method and benefits from smaller dimensions since vector size is 300 dimensions (2 powers of magnitude lesser) . The hyperparameters for the TSNE model were set to the following:

- Perplexity set to 40. This value is effectively the number of nearest neighbors (*Sklearn.Manifold.TSNE*, n.d.).
- n_components is set to 2. This is the number of dimensions we want to reduce to.
- intialization of the embedding is set to PCA.
- n_iterations is set to 2500. This is the maximum number of iterations for the optimization
- random_state set to an arbitrary number 23. Setting a value for this allows for reproducible results over multiple function calls.

3.3.2.3 Plotting

After carrying out the TSNE transformation, a dataframe is created and saved to file. The dataframe contains 3 columns, the word and columns representing the x and y axis. These dataframes are then readily loaded into the app during runtime, for exploring the word vectors on a graph. We can explore similar words and their placement from each other, and allows us to infer if a high-level concept is reflected by the directionality between the vectors.

Shown below is the plot of the full vocab. We can already notice a much more uniform distribution of the words, in a fairly symmetrical manner about the x and y axis. We can assume that the uniform and spread out representation, has maximized the information encoded in the 2D. Hence, we can use this to draw inference based on direction and distance.

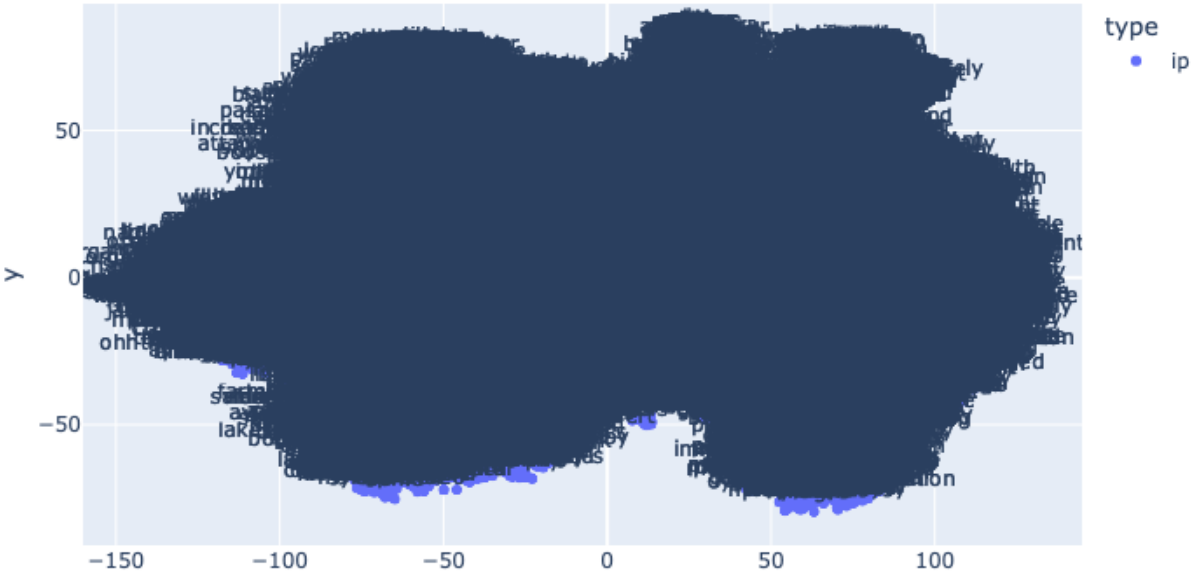

Figure 9: Word2vec represented in 2 dimensions (300 dimensions, windows size 5)

4: Results and discussion

To use the dashboard tool, we can start by filtering out the rows of the data table based on a target word. The description column and the hashtag column are the main columns with which we can filter by text. Often, the number of row results while filtering for a word, is more for hashtags rather than the descriptions. This can generally be attributed to the fact that although the word is not used in the description, the user may have used the hashtag because they believe that the post is related to the target word concept in some way. The filtered rows text can be rendered in a wordcloud to allow us to abstract a high level generalization about the text in the filtered selection.

Next, we can use any of the other graphs to gather our intuition about the particular concept. The available visualizations include barcharts for post count by word and co-occurences, and vector space graphs for checking the words which are close (most similar) to a particular word.

We can also perform simple mathematical operations like addition and subtraction on the word vectors, and they can point towards related terms or concepts. This allows us to play with different combinations of words relying on what feels intuitive to add or subtract or both.

- If word vectors are added it gives a resulting word which is similar to both words, like a middle word. It can be intuited as an average.

- Whereas if we subtract and add a word, then we can determine analagous relationships. For example in an analogy like: king is to man what woman is to '____', we can type it in the form 'king' - 'man'+ 'woman' = 'queen'.

We can use any domain of interest like "textile" or "pharma" and add "organic", then it should give the word which is closest to the meaning of the summed words, based on the text descriptions in the corpus. Example:

- Consumer + waste = footprint
- Plastic + organic = fairtade
- Textile + organic = fairtrade

The count of co-occurrences plot can be a good place to start for analysing generalizations about our target concept "organic". The bar chart will display the co-occurring words based on the word typed into the input box. If required, we can use the slider to adjust the number of words to display. Note that some of the words might be omitted because the sidebar of the plot cannot fit it. In these cases, we can zoom into that particular section using the controls on the top right corner of the plot till all the words are displayed on the sidebar.

4.1 The top co-occurrences with the word "organic"

If we type "**organic**" in the input box of the co-occurrence plot, we get the co-occurring words in descending order. The top words and their associated counts are described in the figure below various high-level insights can be derived.

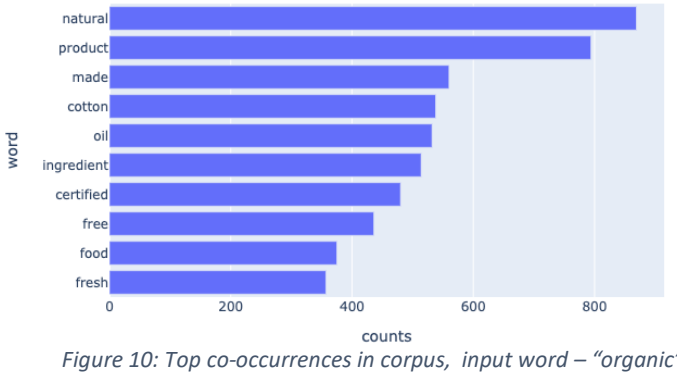

Figure 10: Top co-occurrences in corpus, input word – "organic"

- It has the highest co-occurrence with 'natural'. As opposed to 'artificial'. Synonymous.
- It is used most in the context of 'products'. And this is most associated with the verbs 'use' and 'made'; and the *domain* of 'skin care' which we will not further pursue at this point.
- The premier *material* in the ranks is 'cotton', which is associated the strongest (most number of co-occurrences) with 'certification'. The other associations of cotton are:
    - 'soft', 'fabric' – descriptive words. Cotton is soft, and it is a fabric.

        - Interestingly there are also high associations of 'fabric' usage with the cannabis industry 'pot' and 'hemp'. On filtering the dash table to "fabric pot" we can get a better picture of why; These are posts on indoor cannabis cultivation using fabric pots.
    - 'bag'- could this be cotton bags? Yes because when we look at the associations for 'bag', the foremost *material* is 'plastic' followed by 'cotton'.
    - Also we have 'free'. Filtering the "hashtags" column to #organiccotton and further filtering the "description" column to 'free' on the DashTable we can see the context of 'plastic' appear in a large number of posts (barchart on the left).
    - We could go deeper on the different associations with 'cotton' based on our business interests, using the other charts as well.
- Notice that till now we have only used the bar charts for ***co-occurrences*** and ***posts by unique word*** respectively.
- Let us now look at 'oil'. Here 'essential' seems to be the dominant co-occurrence. Let us sum these vectors in the ***vector math*** graph to understand more.
    - Oil + essential = jojoba, carrier, marula, frankincense etc. These are the types of essential oils.
    - On the unique word posts bar chart, 'skin' seems to be the premier use case followed by use as an 'ingredient'.
    - 'olive' and 'coconut' oils seem to be the most popular *types* in terms of co-occurrences. But these are not 'essential' oils because these do not feature in the previous equation.
        - From the dashtable filtering, we can see that 'coconut oil' is mainly mentioned in posts for skincare. Infact 'ingredient' is most co-occurring with 'skin' as a domain of use.
        - Whereas for 'olive oil' the wordcloud and bar chart of post mentions suggest application with other cooking ingredients and food.
    - We can similarly explore deeper for 'oils' using its other co-occurrences as a start point. Words like 'hair', 'skin', 'body' suggest applications in the beauty industry.
- For 'ingredient', if we enter the vector math formula 'organic' + 'ingredient', we get the resulting words that suggest concepts like 'purest', 'artisanal', 'nasties', 'leaping', and 'unrefined.
    - When exploring 'nasties' on the dashtable we notice that it is better represented with its prefix word 'no' as in 'no nasties'. We miss this in the visualizations because they focus on more significant words. 'leaping' is associated with 'leaping bunny' which is a certification against animal testing, used for cosmetic and household products.
- In the co-occurrences for 'certified', we see that 'cotton' has a strong association. When we execute the formula 'cotton' + 'organic' we get the results like 'gots', 'oeko', 'softest', 'fsc', 'ecocert'. These are all textile certifications, except 'softest' which is more of an implied characteristic of organic cotton.
- The word 'free' co-occurs most with 'gluten', 'shipping', 'cruelty', 'chemical', 'sugar', 'dairy', 'paraben', 'plastic'. The context of free seems pretty broad. For one it is a concept of something without charge, and on the other it indicates the lack of a

certain harmful substance. We can use the vector math graph to identify concepts that resonate in specific domains.

4.2 Deep analysis around the concept of “organic food”

4.2.1 Instagram DataTable (DT-1)

- We can use the first datatable on the dashboard “Instagram DataTable” and filter out the posts containing the keyword “food”. These should automatically include the idea of “organic” because the entire dataset consists only of “organic” posts.
    - Under the ‘description’ column header, there is a space to type an input query. If we type “food” and hit enter, the table gets filtered to post descriptions which contain the word.
    - On filtering the table, we see that the number of pages reduces to 185. Each page contains 10 posts, so we can infer that there are more than 1840 posts about food.
    - We can read the details about each post like ‘description’, ‘hashtags’, ‘cap_mentions’ – which means other users who have been tagged in the post, and ‘web_links’ – which are the websites mentioned in the post.
    - The tool is designed so that we can make general inferences about the data. So now with the table filtered to ‘food’, we can click on the *sel_all* button to select these filtered rows to be rendered in the wordcloud (WC-1).
    - By default, the ‘description’ column is highlighted. Using the dropdown, a different column or multiple columns can be selected. The selected rows and columns are visualized using WC-1 beside the Instagram DataTable. This DataTable only feeds into *WC-1*.

Figure 11: Dashtable DT-1. Allows for filtering and selection of rows and columns

4.2.2 Wordcloud (WC-1) – Word frequency

- The wordcloud offers an intuitive way to see significant words and themes in a large corpus of text (*WordCloud for Python Documentation — Wordcloud 1.8.1 Documentation*, n.d.). The parameters in the word frequency wordcloud were set to the following:
    - Meaningless words are not considered. English stopwords and irrelevant website link words like “https”, “http”, “www” etc were added to the ignore list and not rendered.
    - Collocations is set to true. This means that words which appear together “bi-grams” are also identified. These bigrams are only significant for the ‘description’ column since in the other columns there is no inherent meaning of words which appear successively together.
    - The columns that can be visualized apart from the ‘description’ and ‘hashtag’ columns are ‘cap_mentions’ and ‘web_links’.
- Let us explore the columns one at a time to see what trends emerge. It is useful to compare the wordclouds with a baseline of the entire corpus, figure 12, to identify what is changing.
    - The larger the word, the greater is its frequency of occurrence in the corpus.
    - Figure 13, on the right, is for posts filtered to contain “food”.
    - We notice that for ‘food’, some of the words are common from the full corpus WC on the left like: **‘organic’, ‘make’, ‘get’, ‘one’**.
    - Apart from these, the words which seem pronounced are ***‘body’, ‘also’, ‘health’***. So we know that the word ‘food’ is associated with the concept of ‘body’ and ‘health’.
    - Using the zoom option of WC-1 on the dashboard, we can explore the significance of other words at smaller sizes.
    - The insights drawn from this are broad and feel based. At this stage, the analysis is not empirical or conclusive, but gives us a direction to approach

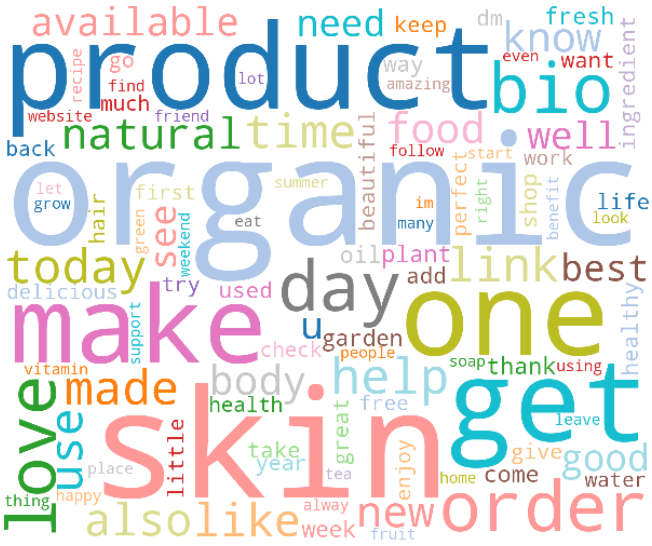

Figure 12: WC-1 based on entire corpus. Word frequency.

Figure 13: WC-1 based on "food" filtered posts. Word frequency.

4.2.3 Z-score DataTable (DT-2)

- This table looks and functions similar to the previous one with some additional features:
    - Z-score slider from 0-4. Default value is 0, in which case the table contains the same number of rows as DT-1(all rows). The Z-score is based on bivariate relationship between the 'likes_per_min' and 'time_delta_hours'. These columns are selected to normalize the effect of time elapsed on the 'likes_per_min' values. The outliers indicate high 'likes_per_min' values suggesting that they may be influential.
        - A value of 3 filters to the top 15 influential posts.
    - We can enter values in the 'Inf_non' and 'Domain' columns for manual classification of the posts as "influencer" or "not" and the respective domain like "textile", "pharma", "food" etc. This can then be exported for further processing. That is left for future work and we will not discuss it here.
    - The columns are different. Instead of 'cap_mentions' and 'web_links', we have 'username' and 'fullname'.
    - The row and column selections of this table feed into 2 plots: wordcloud *WC-2* and the barchart *BC-1*.

4.2.4 Wordcloud (WC-2) – Post frequency

- Next we can add the insights from the post frequency wordcloud. This is a wordcloud where the size of the word is dependent on the number of posts that a word occurs in. So this wordcloud controls for over emphasis of a word due to there being large amount of text about it.
- The wordcloud parameters are set to the following:
    - English stopwords are not displayed.
    - Only unique words are considered for each row. So that means collocations also become irrelevant and is set to False.
    - The special columns which can be visualized here are 'username' and 'fullname'.
    - We can render outliers in terms of the bivariate relationship between 'likes per minute' and 'time_delta_hours', using the 4 point z-score slider.

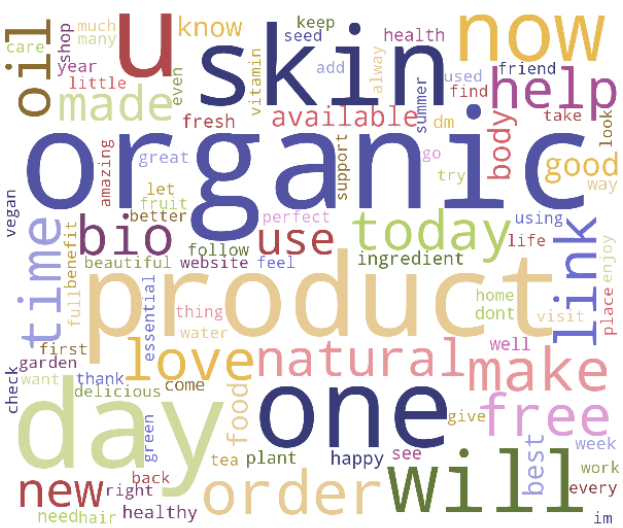

Figure 14: WC-2 based on entire corpus. Post frequency.

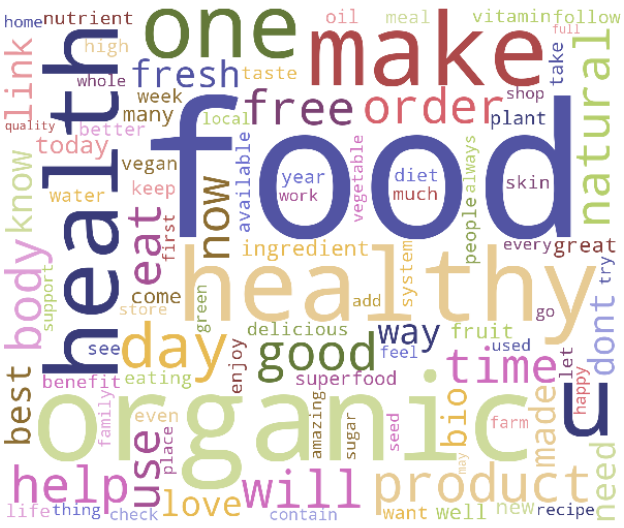

Figure 15: WC-2 based on "food". Post frequency.

- In the above figures 14 and 15, we can see the number of posts for each word of the 'food' filtered rows. We can see that ***'health' and 'make' are the pronounced words and 'body' does not feature as prominently (as in the word frequency WC)*** suggesting that there are few posts with lot of text containing 'body' in the context of 'food'.

4.2.5 Bar chart (BC-1) – Posts count

- From the table DT-2, we can select the 'hashtags' column along with 'description'. This gives us the distribution of words and hashtags in the posts for 'food'.
- We can see that hashtags offer more grouping possibilities because the users themselves try to connect the relevance of the content to standard concepts. There are much more posts which contain the hashtag "#organic", rather than the word.

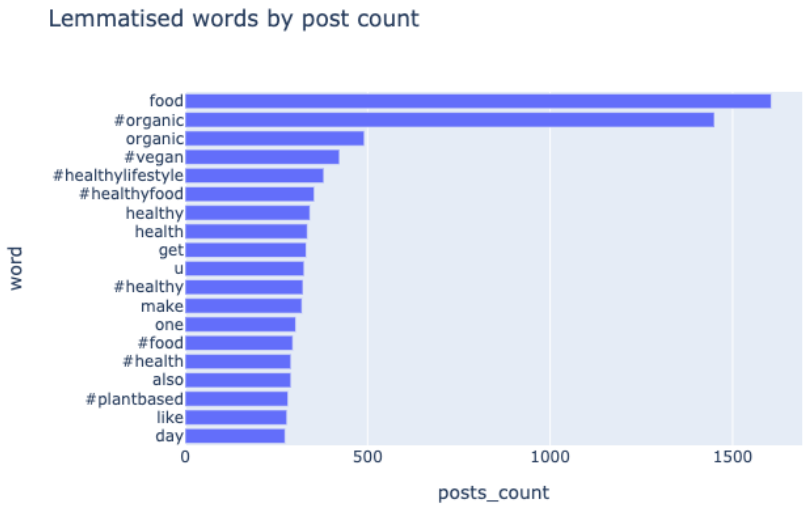

Figure 16: BC-1 number of posts by post count.

- In figure 16, apart from 'food' and 'organic', the concepts which stand out are: 'vegan', 'healthylifestyle', 'healthyfood', concepts around 'health', and 'plantbased'.

4.2.6 Bar chart (BC-2) – Co-occurrences count

- The co-occurrence count bar chart is particularly useful to understand in what specific context, within a window of 5 words, is the center word used. In the context of 'food' posts it is particularly useful in detecting verbs.
- For example, in the previous chart figure 16, we can see words like **'get' and 'u'**. When we type 'u' in the input box of BC-2 as seen in figure 17, the top co-occurrence is with 'dm'. 'dm' means direct message in Instagram. And the 2nd most co-occurring word is 'follow'. These are *terms used to solicit engagement with the customers, typically used by businesses and influencers*.

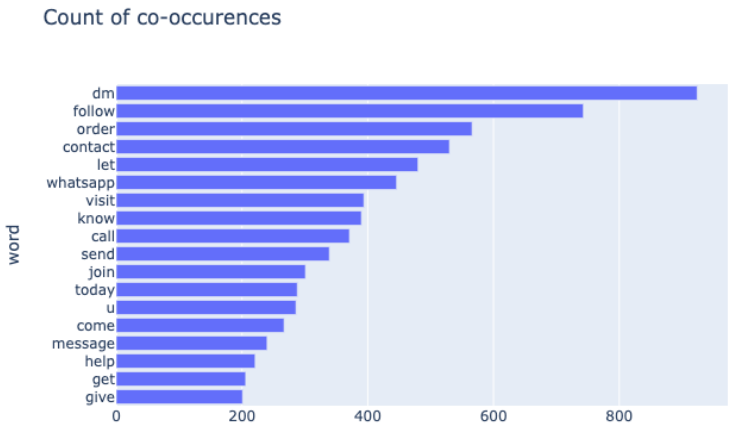

Figure 17: BC-2 word co-occurrences based on input word 'u'.

- On filtering out the description column of DT-2 to 'dm', we see that it is most used in the context of 'us'. Examples 'dm us', or 'follow us', 'call us' etc. Lemmatizing the word 'us' has resulted in it being cut short to 'u' because it thinks it is a plural.

4.2.7 Vector Exploration (VE)

- The closer the words are on this 2 dimensional space, the closer the points are in the 300 dimensional vector space. Starting with the slider at the 0 position we can increment it in steps of 1 to get the most similar words to the input word.
- The 'Clr' – clear button removes all input words. We can then type a word or multiple words (separated by comma ',') in the input box.

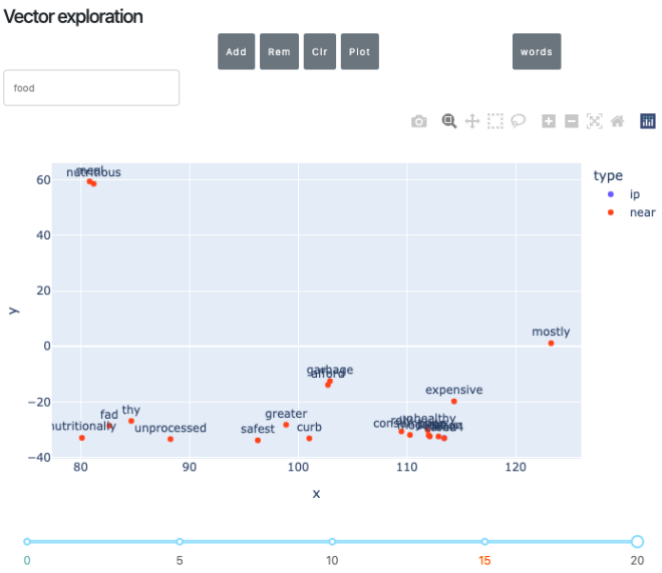

Figure 18: VE Vector exploration of word2vec words. Most similar to 'food' in the corpus

Figure 19: VE Vector exploration of word2vec words. Most similar to 'make' in the corpus

- The figure 18 contains the words which are most similar to ‘food’ from the word2vec vector space. The directions and grouping of the words are indicative of different contexts as well. These are not necessarily semantically similar, but they are used in the context of food. These words are themselves encoded with some meaning about food.
- The graph can be used to check which words are most similar to the word typed in the input box. If we type ‘get’ which is another significant word in ‘food’ posts, the top similar words suggest a sense of urgency and rush: ‘grab’, ‘asap’, ‘rush’. This is again indicative of usage in marketing strategies which seek to create a sense of urgency for sales.
- For the word ‘make’, figure 19, the most similar word is ‘swap’. This is followed after a word by ‘prepare’. So ‘make’ could be used in both the contexts, making food (prepare) and making changes(swap) to food consumed. In what context are the food related posts mostly about? We can use the Vector math graph to clarify.

4.2.8 Vector Math (VM)

- The Vector Math graph display requires that at least 3 out of the 4 input boxes contain a valid word from the vocab, in order to display a result.
- Let us type ‘food’ in one input and ‘make’ in the other input beside the ‘+’ sign to signify summation. Addition in the word2vec model outputs a word which is close to both concepts ie similar to both words.

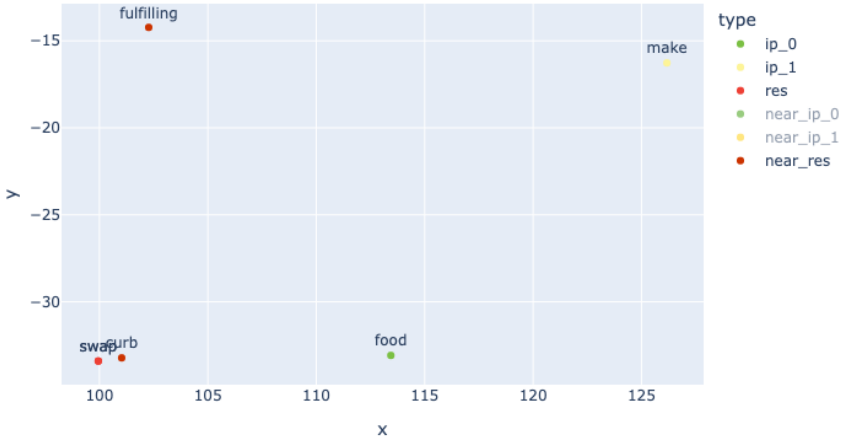

Figure 20: VM Vector Math graph for top 3 most similar words to ‘food’ and ‘make’.

  - As seen in figure 20, the most similar word is **‘curb’** followed by **‘swap’**. These suggest making changes.
  - The next most similar word is ‘fulfilling’ which is popular in garden grown pickles and salads as inferred by filtering DT-1 and observing the render of WC-1.

    - Exploring the co-occurences for ‘make’ in BC-2, we see how it is used in conjunction with different words which can assume totally different meanings. In this sense it can be considered as a stop word.
- Subtraction implies that the result must be similar to the first word and dissimilar to the negative word. This is useful for controlling against a certain concept.
    - For example, when we type ‘food’ + ‘organic’, the most similar word is ‘carna4’.
    - When we check ‘carna4’ using the wordclouds WC-1 and WC-2, we understand that this is a pet food brand which seems to use many words related to organic, in different contexts. So, the model has learned this as the most similar concept to ‘organic’ and ‘food’.
    - We can control for this by subtracting ‘pet’ in the input box. This gives us results which are likely to be related to human consumption. The results are **‘barley’, ‘hearty’, ‘raw’, ‘millet’**.

4.2.9 Synthesis

- From WC-1 we were able to identify more potential stopwords (does not add discerning meaning to the ‘food’ data) words like ‘organic’, ‘make’, ‘get’, ‘one’. Controlling for these stopwords, we find that words like ‘body’ and ‘health’ were used frequently.
- In WC-2 we saw that ‘health’ and ‘make’ is pronounced but ‘body’ is not emphasized. Since ‘make’ can be considered a stopword, ‘health’ is the major concept related to ‘food’ when it comes to ‘organic food’.
- Filtering DT-1 and DT-2 to ‘body’ under the description column and ‘food’ under the hashtag column:
    - From the hashtags, it appears that the word ‘body’ in the context of food is associated with ‘healthyfood’ and ‘healthylifestyle’.
    - From WC-1 and WC-2 we can see that ‘help’ is a prominent word in this selection of data. Reading some of the posts we can see that these are posts that ‘help’ the ‘body’ in some way or the other.
- Lateral thinking approaches:
    - Let us think about what organic food could also mean. It must relate to organic methods of growing right? But why should there be significance about organic methods of growing? **Ideas like ‘biodiversity’ and ‘sustainability’** come to mind.
    - Filtering DT-2 description column to ‘biodiversity’, only about 25% of the biodiversity posts speak about food, as observed in figure 21.

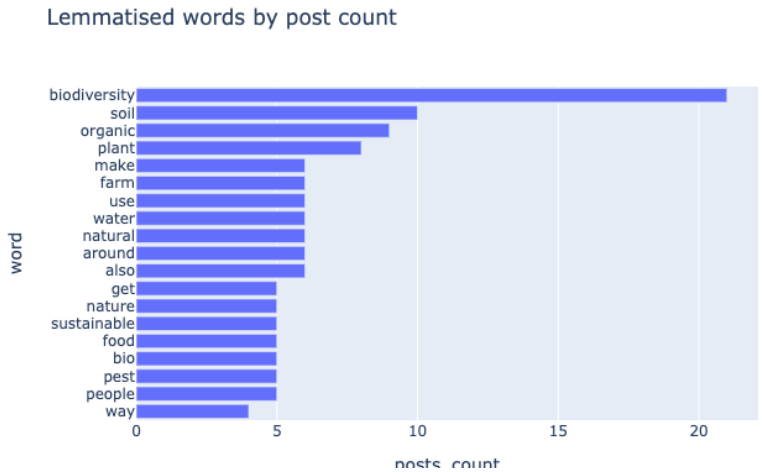

Figure 21: BC-1 where 'food' posts are 25% of the 'biodiversity' posts

- We can also see that terms related to farming are more prevalent in these posts: **'soil', 'plant', 'farm'**.
- It must also be noted that there are very few posts, only 21, about 'biodiversity'. So we must be cautious about drawing generalizations.
  - When we visualize the hashtags column for this same filtering, we see that they are mainly about **selling wines**.
  - This is a potential insight for organic food companies to hire biodiversity consultants who are working in the wine industry and the social media teams. They can enable a transformation in the meaning of 'organic food' not just being related to 'health' but also important for 'biodiversity' preservation.
- Filtering DT-2 to 'sustainable' and observing WC-2 and BR-1.
  - #sustainableliving and #sustainablefashion seem to be the notable themes.
  - When we visualize the hashtags specifically associated with 'food', we still see #sustainableliving to be in the top with other notable hashtags being #farmtotable, #sustainableliving, #smallbusinesses and #organicgardening.
  - From this we can say that the main idea of 'sustainable' within an organic and food context, is about a **way of life 'living'**.
- Filtering DT-2 to 'sustainability', we observe that #organicgardening is a prominent theme, which again suggests a way of living.

- Using the VE graph, we got a better understanding of what is meant by 'get' and 'make'
- In the VM graph we could clarify what meaning of 'make' is more prominent. And also we could get a list of what foods are most organic with terms like 'barley', 'hearty', 'raw', 'buckwheat'.
- To sum it up in a small paragraph:

"From the social media posts, we can infer that "Organic food" caters towards people interested in healthy lifestyle. The products generally claim to help the body in some way.

They encourage making changes to the consumed food towards healthier alternatives. The main foods which are organic include crops like ‘barley’ and ‘buckwheat’. Influencers and businesses presence is at least 16% with the posts containing a call to engage (‘get’, ‘u’)”

5: Conclusion

The aim of this project was to create a tool to infer word meanings using mathematical methods. Although mathematical operations can be performed with great success on numerical tasks, language cannot be fully reduced to numbers. Extracting insights with these models should be considered as an art. The task of translating words into numbers and making use of computers to analyze them, has only seen rapid progress since the past decade, and for this, implementing neural network models has been a game changer in the field. Statistics is still at the base of these methods, and most of the recent developments in the state of the art have been achieved by finetuning the parameters.

A word assumes different meanings depending on how they are used by the people. A word like “organic” assumes different meanings depending on which context it is used in. Although originally used in the context of farming, the sense of the word has assumed many connotations which intuitively make sense to us when used in those different contexts. In light of this, we had rather not relegate this intuitive sense making ability in the face of an empirical model, when it comes to language and word use.

Ideally, we should employ both an intuitive and empirical sense making approach. This tool attempts to offer such a functionality for the analyst or researcher. You may notice that the wordcloud immediately (perhaps “intuitively”) offers a deep insight into the nature of the posts without any numbers representing them. Whereas the bar charts add a numerical dimension to this which is relatively simple for most people to understand. The vector space graphs however, are a little more complex and may not be so intuitive for us to make sense of it right away. The results seem valid enough in most cases, but some results might not make sense, especially the ones with limited textual information.

In these vector space models, if the training dataset is a large corpus of text containing everyday use of language, then the results may make more sense because it has encoded most of the meaning behind it, in the way that we are used to using them. These results make intuitive sense to us only if we ourselves have been exposed to, and participate in, such everyday language use. I propose that this tool can uncover the evolving language use within a certain domain (example “organic”), which may not immediately feel intuitive, but over time gains its truth as such, as it is adopted by the society at large. Training the models on larger corpus of data will lead to better results.

The vector space graphs, might feel more useful for someone who has some (logical) understanding about the underlying statistics and the way it influences the results. The ideal candidate to use this tool could be a linguistics major with a statistical background and an interest in marketing.

It is worth noting that gaining insights from data depends on both, the methods, and the ability of the analyst to understand the methods. Hence, we have not considered more complex neural network models like transformers. This tool would be great for projects

involving social media market research and understanding the language use and motivations of the target users. In the coming months and years, we can expect to see further leaps in this field of abstracting information from words using computers, and it will become increasingly important for the data analysts to have some fundamental idea of the logic behind the methods in order to start using them, until they eventually feel intuitive.

6: Bibliography

Baroni, M., & Lenci, A. (2010). Distributional Memory: A General Framework for Corpus-Based Semantics. *Computational Linguistics*, *36*(4), 673–721. https://doi.org/10.1162/coli_a_00016

Danilak, M. M. (n.d.). *langdetect: Language detection library ported from Google's language-detection.* (1.0.9) [Python; OS Independent]. Retrieved January 11, 2022, from https://github.com/Mimino666/langdetect

Dumais, S. T. (2005). Latent semantic analysis. *Annual Review of Information Science and Technology*, *38*(1), 188–230. https://doi.org/10.1002/aris.1440380105

Ganesan, K. (2014, October 19). What are Stop Words? *Kavita Ganesan, PhD*. https://kavita-ganesan.com/what-are-stop-words/

Gensim: Topic modelling for humans. (n.d.). Retrieved January 17, 2022, from https://radimrehurek.com/gensim/models/word2vec.html

Gentner, D. (1983). Structure-Mapping: A Theoretical Framework for Analogy*. *Cognitive Science*, *7*(2), 155–170. https://doi.org/10.1207/s15516709cog0702_3

Harris, Z. S. (1954). Distributional Structure. WORD, *10*(2–3), 146–162. https://doi.org/10.1080/00437956.1954.11659520

Heidenreich, H. (2018, December 21). *Stemming? Lemmatization? What?* Medium. https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8

Introduction to Stemming. (2018, June 20). *GeeksforGeeks*. https://www.geeksforgeeks.org/introduction-to-stemming/

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, *104*(2), 211–240. https://doi.org/10.1037/0033-295X.104.2.211

Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, *3*, 211–225. https://doi.org/10.1162/tacl_a_00134

List of ISO 639-1 codes. (2022). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=List_of_ISO_639-1_codes&oldid=1064376433

Making sense of word2vec | RARE Technologies. (n.d.). Retrieved January 16, 2022, from https://rare-technologies.com/making-sense-of-word2vec/

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *ArXiv:1301.3781 [Cs]*. http://arxiv.org/abs/1301.3781

models.word2vec – Word2vec embeddings—Gensim. (n.d.). Retrieved January 30, 2022, from https://radimrehurek.com/gensim/models/word2vec.html

On word embeddings - Part 3: The secret ingredients of word2vec. (2016, September 24). Sebastian Ruder. https://ruder.io/secret-word2vec/

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Python Simplified. (2020, October 10). *Web Scraping Instagram with Selenium*. https://www.youtube.com/watch?v=iJGvYBH9mcY

Rivas, R., Sadah, S. A., Guo, Y., & Hristidis, V. (2020). Classification of Health-Related Social Media Posts: Evaluation of Post Content-Classifier Models and Analysis of User Demographics. *JMIR Public Health and Surveillance*, *6*(2), e14952. https://doi.org/10.2196/14952

scipy.sparse.coo_matrix—SciPy v1.7.1 Manual. (n.d.). Retrieved January 15, 2022, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html

Scrape Data Legally | No-Code Data API & Website Scraper | Stevesie Data. (n.d.). Retrieved January 9, 2022, from https://stevesie.com/

Selva Birunda, S., & Kanniga Devi, R. (2021). A Review on Word Embedding Techniques for Text Classification. In J. S. Raj, A. M. Iliyasu, R. Bestak, & Z. A. Baig (Eds.), *Innovative Data Communication Technologies and Application* (Vol. 59, pp. 267–281). Springer Singapore. https://doi.org/10.1007/978-981-15-9651-3_23

Sklearn.decomposition.TruncatedSVD. (n.d.). Scikit-Learn. Retrieved January 16, 2022, from https://scikit-learn/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

Sklearn.manifold.TSNE. (n.d.). Scikit-Learn. Retrieved January 19, 2022, from https://scikit-learn/stable/modules/generated/sklearn.manifold.TSNE.html

Smetanin, S. (2018, November 16). *Google News and Leo Tolstoy: Visualizing Word2Vec Word Embeddings with t-SNE*. Medium. https://towardsdatascience.com/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d

Turney, P. D., & Pantel, P. (2010). From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, *37*, 141–188. https://doi.org/10.1613/jair.2934

WordCloud for Python documentation—Wordcloud 1.8.1 documentation. (n.d.). Retrieved January 24, 2022, from https://amueller.github.io/word_cloud/

7: Appendix

7.1 Interactive Dashboard Tool

This is how the tool appears when scrolled down on the web page.

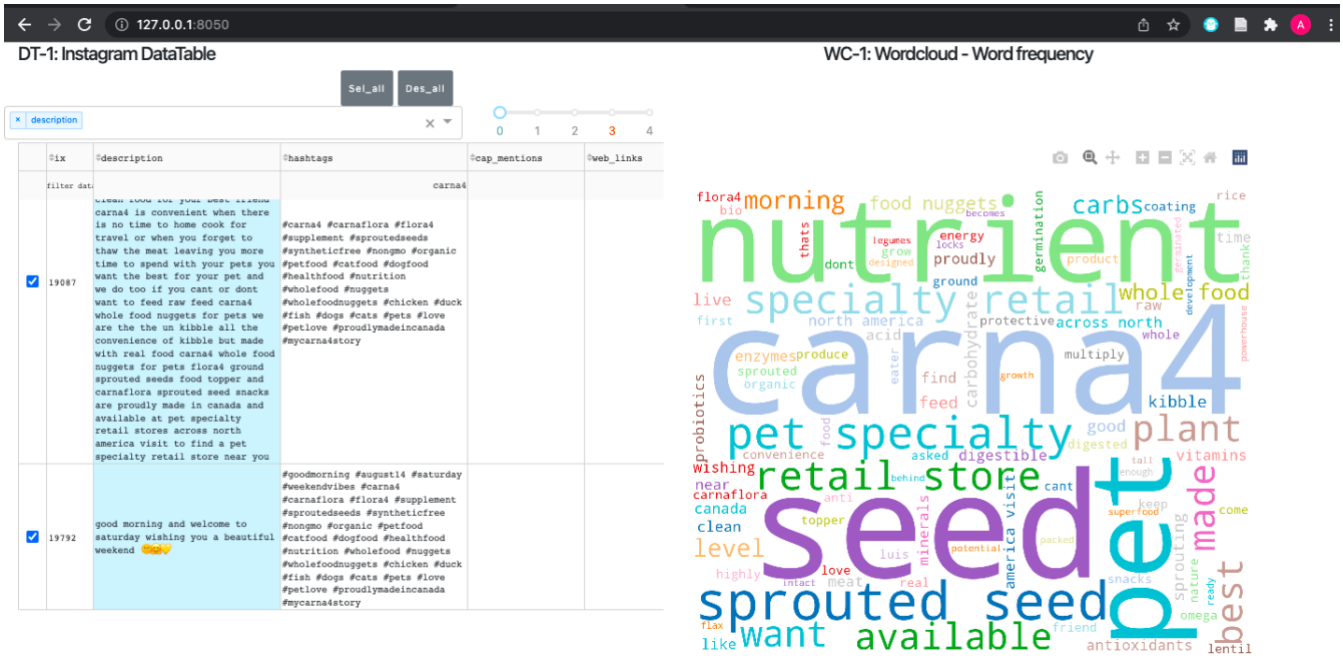

Figure 22: Dashtable DT-1, and Wordcloud WC-1 of the dashboard

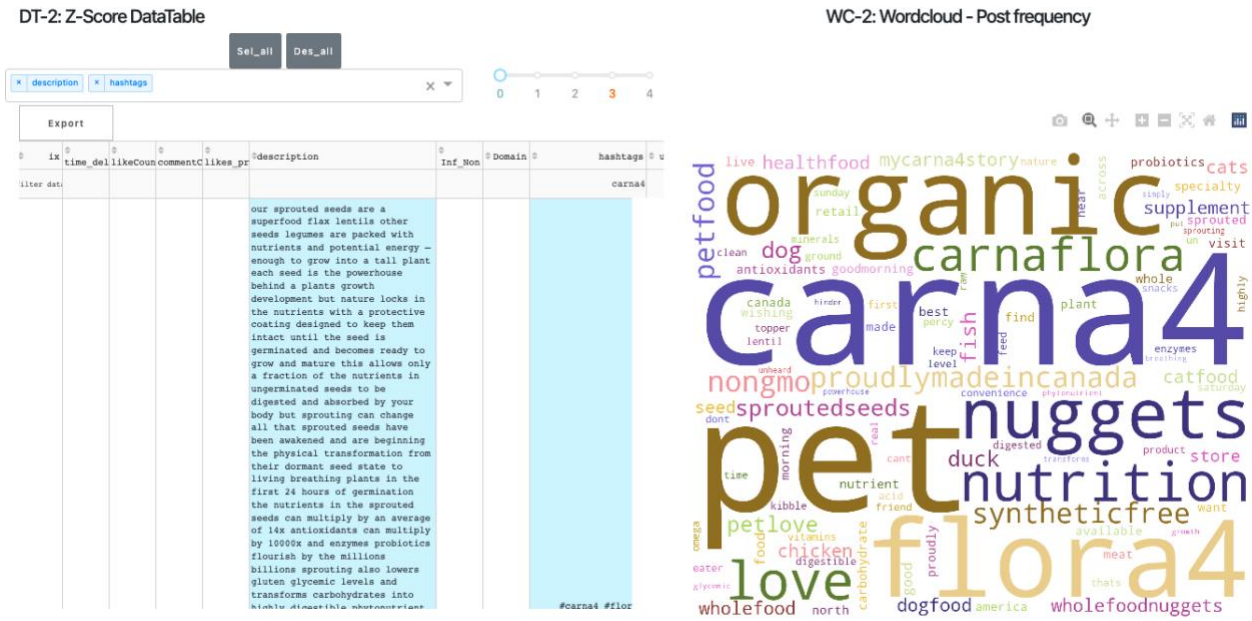

Figure 23: Dashtable DT-2, and Wordcloud WC-2 of the dashboard, on scrolling down

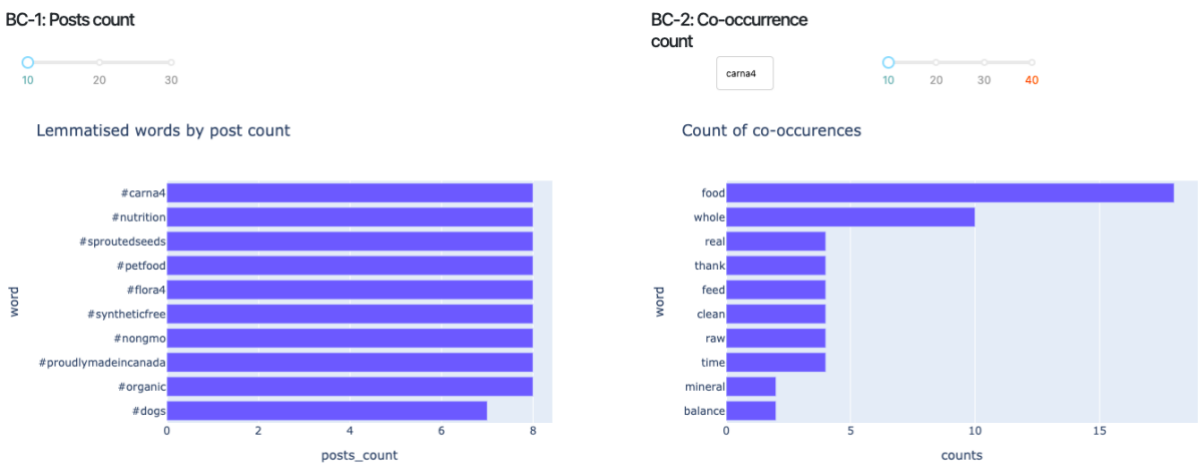

Figure 24: Barchart BC-1, and Barchart BC-2 of the dashboard, on scrolling down further

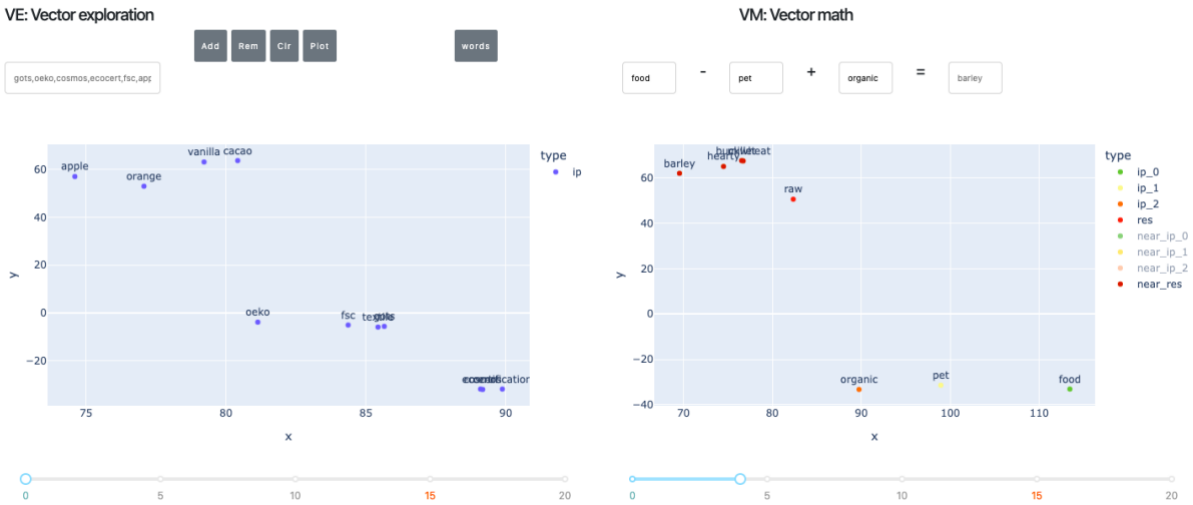

Figure 25: Vector Exploration VE graph, and Vector Math VM graph of the dashboard, towards the bottom end of the page

7.2 Development of code

SI No.	Development Version name	Date	Summary
1	Thesis_proj_v2.ipynb	20 July 2021	Exploring data sources. Data preprocessing.
2	Thesis_proj_v3.py	2 Sept 2021	App development started. Wordcloud.
3	Thesis_proj_v3.2.py	10 Sept 2021	Data table interactivity.
4	Thesis_proj_v3.3.py	16 Sept 2021	Stabilized app code. Code cleanup.
5	Thesis_proj_v3.4.py	27 Sept 2021	Word co-occurrence matrixes.
6	Thesis_proj_v3.5.py	26 Oct 2021	Word2vec math interactivity.
7	Thesis_proj_v4.py	10 Jan 2021	Final Explorations.

7.3 Python Code – preprocessing and creating support files

```
# In[1]:
print("Starting program..")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import unicodedata
from langdetect import detect

from wordcloud import WordCloud, STOPWORDS
from nltk.corpus import stopwords

import warnings
warnings.filterwarnings('ignore')

import os
import os.path

from datetime import datetime
app_launch_start=datetime.now() #Set start time for program start
# In[2]: Combine datasets; final_df.to_csv("combined_df.csv")
##Collect multiple datasets saved in "datasets" folder and create 1 dataframe

print("\nReading datasets in the folder :")
start=datetime.now()
working_dir = "datasets" #folder where the datasets exist
all_files=[]
for root, dirs, files in os.walk(working_dir):
    file_list = []
    for filename in files:
        if filename.endswith('.csv'):
            # print(filename)
            file_list.append(os.path.join(root, filename))
    all_files+=file_list
print(all_files) #list of file names
print("Number of datasets considered:", len(all_files))


read_df_sh=[]
df_list=[]
for file in all_files:
    read_df= pd.read_csv(file)
    read_df_sh.append(read_df.shape)
    df_list.append(read_df)
print("\nList of df shapes are: ",read_df_sh)

df_shape_sum=sum(i for i, j in read_df_sh)
print("\nSum of rows of multiple datasets", df_shape_sum)

#Check if the exisitng dataframe already includes all sub datasets
existing_comb_df=pd.read_csv('combined_df.csv')
existing_comb_df_rows=existing_comb_df.shape[0]
print("Sum of rows of existing combined dataset", existing_comb_df_rows)

if existing_comb_df_rows<df_shape_sum:
    print("\nNew dataframe exists in the dataset folder..")
    print("Create new combined dataframe")


    if df_list:
            final_df = pd.concat(df_list,ignore_index=True)
            final_df.to_csv("combined_df.csv")

```

```
else:
    print("\nAll datasets already included in combined dataset")


t=datetime.now() - start #datetime object
s=str(t) #string object
print("Execution time ", s[:-5])

# In[3]:
##Drop empty rows and create an SI number column
print("\nDrop empty rows and add SI number column..")
start=datetime.now()
comb_df=pd.read_csv('combined_df.csv')
print(comb_df.shape)
# comb_df.info() #Check where the empty rows are
comb_df = comb_df.dropna(axis=0, subset=['description']) #Drop empty rows in des
print("Redundant rows removed")
print(comb_df.shape)

#Hardcode index values
comb_df.reset_index(inplace= True,drop=True)
comb_df.reset_index(inplace= True)
comb_df.rename(columns={"index": "ix"},inplace=True) #to ref after dropping rows
comb_df['ix']=comb_df['ix'].astype(str) #convert to string type

t=datetime.now() - start
s=str(t)
print("Execution time ", s[:-5])


# In[4]:
##Functions for splitting into new columns
print("\nLoading column splitting functions..")

def desc_cleaning(org_str): #some hashtags are not separated by a space.
  orig_str=str(org_str)
  new_str = orig_str.replace('#'," #")
  new_str2 = new_str.replace('  '," ")
  return new_str2

def desc_splitting(df):
  df_upd=df
  df_upd['clean_captions']=" "
  df_upd['hashtags']=" "
  df_upd['cap_mentions']=" "
  df_upd['web_links']=" "

  for ind,row in df_upd.iterrows():
      X=str(row['new_desc']).split()#ensure description column is cleaned prior
      X_cc=[]
      X_hstgs=[]
      X_cms=[]
      X_http=[]

      for x in X:
          if not (x.startswith(('#','@','http')) or ('www.' in x) or ('.com' in x)) :
              X_cc.append(x)

          if x.startswith('#'):
              X_hstgs.append(x)

          if x.startswith('@'):
              X_cms.append(x)

          if (x.startswith('http') or ('www.' in x) or ('.com' in x)):
```

```
                X_http.append(x)

        df_upd.at[ind,'clean_captions'] = ' '.join(X_cc)
        df_upd.at[ind,'hashtags'] = ' '.join(X_hstgs)
        df_upd.at[ind,'cap_mentions'] = ' '.join(X_cms)
        df_upd.at[ind,'web_links'] = ' '.join(X_http)

    return df_upd


def new_cols(df): ##wrapping above functions
    df['new_desc'] = df['description'].apply(lambda x : desc_cleaning(x))
    df_updated=desc_splitting(df)
    return df_updated

print("Loaded")


# In[5]:
##New columns dataset "df_updated"
print("Applying splitting functions...")
start=datetime.now()

df_updated=new_cols(comb_df)
print("New columns created from splitting func")

t=datetime.now() - start
s=str(t)
print("Execution time ", s[:-5])


# In[6]: Text preprocessing; fonts, lower, punct, char_encodings
#Tried to make granular transformations for specificity of processing.

print("\nLoading text preprocessing functions..")

def font_uniformity(x):
    return unicodedata.normalize('NFKC', x)

def convert_lower_case(s):
    return np.char.lower(s)

def remove_punctuation(data):
    symbols = "!\"#$%&()*+-.•/:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')

    data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data

def diff_encodings(s):
    s = np.char.replace(s, "…", " ")
    s = np.char.replace(s, "’", "")
    return s

print("Loaded")
print("Applying text preprocessing...")
start=datetime.now()

df_updated['caption_processed']=df_updated['clean_captions'].apply(lambda x: font_uniformity(x))
df_updated['caption_processed_2']=df_updated['caption_processed'].apply(lambda x: convert_lower_ca
df_updated['caption_processed_3']=df_updated['caption_processed_2'].apply(lambda x: remove_punctua
df_updated['caption_processed_4']=df_updated['caption_processed_3'].apply(lambda x: diff_encodings

print("Text preprocessing done")
t=datetime.now() - start
s=str(t)
print("Execution time ", s[:-5])

# In[8]: "views" and "exemplars" - testing
##Define view frame to view previous processing steps with exemplars
view1=['description','new_desc','clean_captions' ]
view2=[]
view2=['clean_captions','caption_processed','caption_processed_2', 'caption_processed_3',
       'caption_processed_4']

#define the rows to display-exemplars of diff test cases
ex_row_list=[11,13,23,106] #subtract 2 because index is reset.

#df_updated[view2].loc[ex_row_list]
#df_updated[view2].head(30)

# In[9]: Language detection to_csv("App_dataframe.csv")
# Visualise how many languages are there:
print("\nLoading language detection function..") #Takes 11:30 mins to execute
```

```
start=datetime.now()

def lang_det(st):
    try:
        lang=detect(st)
        return lang

    except:
        lang="error"
        return lang

view3=['det_lang']
view3= view2+view3

print("Running language detection...")
df_updated['det_lang']= df_updated['caption_processed_2'].apply(lambda x: lang_det(x))
print("Language detection complete")
t=datetime.now() - start #datetime object
s=str(t) #string object
print("Execution time ", s[:-5])

#Export to app referenced df after this last transformation:
df_updated.to_csv("App_dataframe.csv")

# In[9]: Language countplot
##visualise the new transformations
df_updated = pd.read_csv('App_dataframe.csv')

print("Making countplot..")
plt.figure(figsize=(16,6))
ax= sns.countplot(x= 'det_lang', data=df_updated, order = df_updated['det_lang'].value_counts(
    ascending=False).index)
ax.set_title('Language distribution')
ax.set_xlabel('Languages')

for tick in ax.get_xticklabels():
    tick.set_rotation(90)

print("Language countplot graph loaded")

# In[10]: #Selective viewing of dataframe with "views" and "exemplars"
view_extracts=['new_desc','det_lang','clean_captions','caption_processed_4','hashtags',
               'cap_mentions','web_links' ]
df_updated[view_extracts].loc[ex_row_list]
df_updated[view_extracts].head(100)

#add new elemnts to exemplar list
ex_row_list.append(3) #add exemplar accents in differnt language
df_updated[view3][df_updated['det_lang']=='error'].head(10)
#Errors for descriptions which do not have readable text. Either blank or emojis.
#Fonts normalised but not much improvement.

# In[11]: ##Wordcloud with English and French stopwords

print("\nLoading test Word Cloud..")
start=datetime.now()

# df_updated['clean_captions'][10] #test individual rows
word_string = " ".join(df_updated['clean_captions'].dropna().astype(str)) #select column series
print(word_string[10:1000])


#Define stopwords
# type(STOPWORDS) #set
# len(STOPWORDS) #192
en_stopwords = stopwords.words('english') #179
fr_stopwords = stopwords.words('french') #157
web_links_sw = ['www','http','https','com']

combined_stopwords = en_stopwords + fr_stopwords

wordcloud = WordCloud(max_words=100,
                      stopwords= combined_stopwords,
                      collocations=False,
                      color_func=lambda *args, **kwargs: "orange",
                      background_color='white',
                      width=1200,
                      height=1000).generate(word_string)

fig_dims = (8, 8) #size of plot
fig, ax = plt.subplots(figsize=fig_dims)
plt.title("#Test word cloud")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
#wordcloud.to_file("static/images/en_fr_sw_wc.png")
```

```
print("Test Word Cloud loaded")
t=datetime.now() - start
s=str(t)
print("Execution time ", s[:-5])

#With only EN stopwords:  "u", "de", "la", "e", still present
#With EN and FR : 'u', 'e', 'o' still present

# In[12] Filter; read_csv("App_dataframe.csv") -> to_csv("App_dataframe_2.csv")
#Filter the dataframe to English, #organic, and remove duplicate posts
##Export to a new dataframe
print("Reading in dataframe for app with filters and duplicate removal..")
ad_1=pd.read_csv("App_dataframe.csv")
ad_1.shape
ad_2=ad_1[ad_1['det_lang']=='en'] #filter only english, removes appr 33%
print("Only English posts",len(ad_2))

ad_3=ad_2[ad_2['query']=='#organic'] #filter only #organic scrape query, removes 33%
print("Only organic posts: ", len(ad_3))

ad_4=ad_3.drop_duplicates(subset=['postId']) #remove dupl posts, removes 12.5%
print("After duplicate removal: ", len(ad_4))

ad_4.drop(['Unnamed: 0','ix','Unnamed: 0.1'],axis =1,inplace=True)

#Create new index for proper selection of the filtered table in appl
ad_4.reset_index(inplace= True,drop=True)
ad_4.reset_index(inplace= True)
ad_4.rename(columns={"index": "ix"},inplace=True) #replace index to use as row ref
ad_4['ix']=ad_4['ix'].astype(str) #convert to string type

ad_4.to_csv("App_dataframe_2.csv", index_label = False)


# In[13] More preprocessing; read_csv("App_dataframe_2.csv") --> to_pickle("App_dataframe_3.pkl")
# Implement Vecotrization.
# Tokenize the processed description

import ast
import preprocessor as p

ad_4= pd.read_csv("App_dataframe_2.csv")

#define function to remove newly identified punctuations/encodings
def addi_treatment(s): #Move this to the main preprocessing block if required
    s = np.char.replace(s, "”", "")
    s = np.char.replace(s, "“", "")
    s = np.char.replace(s, "'", "")
    s = np.char.replace(s,"–","") #from viewing bag of wrods in first 100 of cp4
    return s


def preprocess_tweet(row):
    text = p.clean(row) # removes hashtags,emojis,urls, mentions, smileys
    return text

def remove_stop_words(data): #takes 5:25 mins for the full dataset
    """
    Split/tokenize the words first. Create a list of tokens for each.
    Iterate each token removing stop words from text.
    """
    row_tokens = data.split()
    return [word for word in row_tokens if not word in stopwords.words("english")] # 25 s for 100r

start= datetime.now()

#Create new columns
ad_4['caption_processed_4'] = ad_4['caption_processed_4'].apply(lambda x : addi_treatment(x))
ad_4['cp4_no_emojis']=ad_4['caption_processed_4'].apply(lambda x: preprocess_tweet(x))
ad_4['cp4_rm_sw'] = ad_4['cp4_no_emojis'].apply(remove_stop_words)
ad_4['cp4_rm_sw'] = ad_4['cp4_rm_sw'].astype(str) #convert to string for literal eval
ad_4['cp4_rm_sw'] = ad_4['cp4_rm_sw'].apply(lambda x : ast.literal_eval(x)) #preserve lists.
al=datetime.now() - start #start time logged at start
sal=str(al) #string object
print("Execution time for all rows : ", sal[:-5])

print("Type of rows in cp4_rm_sw are ", type(ad_4['cp4_rm_sw'][32000]))
ch_list=['caption_processed_4', 'cp4_rm_sw']
view=ad_4[['caption_processed_4', 'cp4_rm_sw']] #index 35129
# print(view)

ad_4['cp4_rm_sw'][32000]

ad_4.to_pickle("App_dataframe_3.pkl", protocol=0) #preserves the list datatypes. prot=0 for colab

##App_dataframe_3 now has
```

```
#words minus stopwords
#treated for "addi_treatment
#tweet preprocessor removed emojis


# In[14]# Class for cooccurence embeddings initial- from prof Redha Moulla notes
# Finally used different co-occ method with window selection.
from sklearn.decomposition import TruncatedSVD

class CooccEmbedding:
    def __init__(self, corpus):
        """
        Takes in the corpus in the form of a list of lists (list of tweets,
        each tweet being a list of tokens)
        """
        self.corpus = corpus

    def vocabulary(self):
        """
        Returns the vocabulary associated with the corpus (a list of unique tokens from the corpus
        """
        self.vocab = []
        [self.vocab.append(word) for tweet in self.corpus for word in tweet if word not in self.vo
        self.len_vocab = len(self.vocab) #Initiliase this return because coocc module not used.
        return self.vocab #outputs single list of unique words

    def coocc_matrix(self): #Note that this is not looking at window size for cooccurence. Conside
        """
        Returns the co-occurrence matrix associated with the corpus.
        The co-occurrence matrix is first calculated as a list using comprehensive lists to speed
        the iteration process over the vocabulary and the corpus.
        """
        self.len_vocab = len(self.vocab)
        coocc_list = [sum([1 if (self.vocab[i] in tweet and self.vocab[j] in tweet and i != j) els
                      for i in range(self.len_vocab) for j in range(self.len_vocab)] #1st word at
        coocc_list = np.array(coocc_list)
        # transform the coocc_list into a matrix
        self.coocc_mat = coocc_list.reshape([self.len_vocab, self.len_vocab])
        return self.coocc_mat

    def svd_reduction(self, n_components = 2, n_iter = 10):
        """
        Performs a singular value decomposition on the co-occurrence matrix M and returns the trun
        matrix U, where M = UDV^T.
        """
        svd = TruncatedSVD(n_components = n_components, n_iter = n_iter)
        self.reduced_matrix = svd.fit_transform(self.coocc_mat)
        return self.reduced_matrix

    def vocab_ind_to_plot(self, vocab_to_plot):
        """
        Takes in the a list of vocabulary to plot and returns a dictionary in the form of
        {word:index} which is a subeset of the vocabulary.
        """
        vocab_dict = dict(zip(self.vocab, range(self.len_vocab)))
        self.dic_vocab_to_plot = {key:value for key, value in vocab_dict.items() if key in vocab_t
        return self.dic_vocab_to_plot


# In[15]: #Test bag of words sizes and instantiate Coocc_Embedding class
# Trim the vocab so that cooccurnce matrix can be computed.

ad_5=pd.read_pickle('App_dataframe_3.pkl')

def list_of_words(data): #takes 5:25 mins for the full dataset
    row_tokens = data.split()
    return [word for word in row_tokens] # 25 s for 100return the word only if not in the stop wor

#Prepare the list of lists for the bag of words input in Coocc class:
ad_5['caption_processed_4'] = ad_5['caption_processed_4'].apply(list_of_words)
words_cp4=[row_list for row_list in ad_5['caption_processed_4']]
words_cp4[0:3]

ad_5['cp4_no_emojis'] = ad_5['cp4_no_emojis'].apply(list_of_words)
words_cp4_no_emojis=[row_list for row_list in ad_5['cp4_no_emojis']]

words_cp4_rm_sw=[row_list for row_list in ad_5['cp4_rm_sw']] # List of lists

##Tests on the first 100 rows
inst = CooccEmbedding(words_cp4[:100])
print("Number of unique words: ",len(inst.vocabulary())) #1587 words in cp4

inst = CooccEmbedding(words_cp4_no_emojis[:100])
print("Number of unique words: ",len(inst.vocabulary())) #1418 words in words_cp4_no_emojis

inst = CooccEmbedding(words_cp4_rm_sw[:100])
```

```
print("Number of unique words: ",len(inst.vocabulary())) #1309 words in words_cp4_rm_sw

#For full dataframe:
inst_1 = CooccEmbedding(words_cp4)
inst_2 = CooccEmbedding(words_cp4_no_emojis)
inst_3 = CooccEmbedding(words_cp4_rm_sw)

#Takes time so time it
time_vocab=datetime.now()

print("cp4 unique words: ",len(inst_1.vocabulary())) #71616 words,#46066 words,#45925 words
t=datetime.now() - time_vocab
s=str(t)
print("cp4 counting time: ", s[:-5])

print("cp4_no_emojis unique words: ",len(inst_2.vocabulary())) #46066 words
t=datetime.now() - time_vocab
s=str(t)
print("cp4_no_emojis counting time: ", s[:-5])

print("cp4_rm_sw unique words: ",len(inst_3.vocabulary())) #45925 words
t=datetime.now() - time_vocab
s=str(t)
print("cp4_rm_sw counting time: ", s[:-5])


# In[16]: #Lemmatization and Stemming ->create new pkl App_dataframe_4

ad_5=pd.read_pickle('App_dataframe_3.pkl')

#Porter Stemming
from nltk.stem import PorterStemmer
porter = PorterStemmer()

def porter_stemming(row_list):
    new_word_list=[porter.stem(word) for word in row_list]
    return new_word_list

start = datetime.now()
ad_5['rm_sw_stem']=ad_5['cp4_rm_sw'].apply(porter_stemming)
t=str(datetime.now()-start)
print("Time for stemming full dataframe: ", t[:-5]) #27.2 seconds for full dataframe

words_rm_sw_stem=[row_list for row_list in ad_5['rm_sw_stem']]

print(words_cp4_rm_sw[:2])
print("\n",words_rm_sw_stem[:2])

inst_4=CooccEmbedding(words_rm_sw_stem)

start=datetime.now()
print("rm_sw_stem unique words: ",len(inst_4.vocabulary())) #34051 unique words
t=str(datetime.now() - start)
print("rm_sw_stem counting time: ", t[:-5]) #1:26 mins


#Lemmatization
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

def wordnet_lemmatizer_func(row_list):
    new_word_list=[wordnet_lemmatizer.lemmatize(word) for word in row_list]
    return new_word_list

ad_5['cp4_rm_sw'][3:5]

#For testing the functions
test_series= ad_5['cp4_rm_sw'][3:5].apply(wordnet_lemmatizer_func)
print(ad_5['cp4_rm_sw'][3:5])
print("\n Stemmed",ad_5['rm_sw_stem'][3:5])
print("\nLemmatized",test_series)

##Repeat as in previous step for stemming

start = datetime.now()
ad_5['rm_sw_lemt']=ad_5['cp4_rm_sw'].apply(wordnet_lemmatizer_func)
t=str(datetime.now()-start)
print("Time for lemmatizing full dataframe: ", t[:-5]) #5.5 seconds for full dataframe

words_rm_sw_lemt=[row_list for row_list in ad_5['rm_sw_lemt']]

inst_5=CooccEmbedding(words_rm_sw_lemt)

start=datetime.now()
print("rm_sw_lemt unique words: ",len(inst_5.vocabulary())) #41526 unique words
t=str(datetime.now() - start)
```

```
print("rm_sw_lemt counting time: ", t[:-5]) #1:48 mins

ad_5.to_pickle("App_dataframe_4.pkl", protocol=0) #with 2 new columns stemming and lemmatization

# In[17]: #More filtering of the dataframe App_dataframe_4->App_dataframe_5
##Filter out duplicate posts and create another pickle App_dataframe_5

ad_6=pd.read_pickle('App_dataframe_4.pkl')
len(ad_6) # 35131

ad_7 = ad_6.drop_duplicates(subset=['caption_processed_4'])
len(ad_7) # from 35131 of ad_6, ad_7 is reduced to 30240; Another 14.2% drop in the rows.

ad_7.drop(['ix'], axis=1, inplace = True) #drop any older 'ix' columns
ad_7.reset_index(inplace= True,drop=True) #reset and drop old index
ad_7.reset_index(inplace= True) #create new ix column
ad_7.rename(columns={"index": "ix"},inplace=True)

ad_7.to_pickle("App_dataframe_5.pkl", protocol=0)

len(ad_7) #29681

# In[18]: #Window Cooccurence array; save('co_occ_arr.npy'); choose 'rm_sw_lemt'
# Takes a long time. Run this only for new datasets.

from collections import defaultdict

ad_6=pd.read_pickle('App_dataframe_5.pkl') #duplicate posts removed.

words_rm_sw_lemt=[row_list for row_list in ad_6['rm_sw_lemt']] #row_list represents each words in
print("Bag of words created..")
len(words_rm_sw_lemt)

text=words_rm_sw_lemt #full dataset

def co_occ_windows(sentences, window_size):
    d = defaultdict(int) #gives default vaue for non-existing keys
    vocab = set()
    for text in sentences: #text is each post within full bag
        print(text)
        for i in range(len(text)):
            token = text[i]
            vocab.add(token)  #add to vocab
            # coocc_window = text[i+1 : i+1+window_size] #Rolling ahead window scope; sparse
            coocc_window = text[i-window_size : i] + text[i+1 : i+1+window_size]  #Rolling bidirec

            for t in coocc_window:
                key = tuple( sorted([t, token]) )
                # print("key is: ",key)
                d[key] += 1 #at the tuple key, increase the value by 1
                # print("default dict value at key is: ",d[key]) #Each key is a tuple and is uniqu

    # formulate the dictionary into dataframe
    vocab = sorted(vocab) # sort vocab
    global custom_vocab
    custom_vocab = vocab #test object

    # print("Sorted vocab at index is", vocab[0])
    df = pd.DataFrame(data=np.zeros((len(vocab), len(vocab)), dtype=np.int16),
                      index=vocab,
                      columns=vocab) #Initialise a dataframe of zeroes
    for key, value in d.items():
        df.at[key[0], key[1]] = value
        df.at[key[1], key[0]] = value
    return df

#Test out the function
start=datetime.now()
# co_occ_df = co_occ_windows(text,5) #full array w4 takes 24:25 mins rolling ahead
co_occ_df = co_occ_windows(text,2) #bidirectional. Effective window size is 5 takes  mins
print("shape of co_occ matrix is: ",co_occ_df.shape)
t=str(datetime.now() - start)
print("Time for execution: ", t[:-5])

print("Custom vocab shape: ", custom_vocab)
print("First 10 vocab shape: ", custom_vocab[:10])

co_occ_arr =co_occ_df.to_numpy() #convert to an array
co_occ_arr
from numpy import save #Have to import explicitly to save array as binary
save('ad5_co_occ_arr_w5_bi.npy', co_occ_arr)

# In[19]Perform Singular Value Decomposition on the array. SVD_matrix.npy

from numpy import load
# co_occ_arr = load('co_occ_arr_w2.npy') #41526
```

```
co_occ_arr = load('ad5_co_occ_arr_w5_bi.npy') #shape 41526

print("shape of co-occ matrix is: ", co_occ_arr.shape)

ad_6=pd.read_pickle('App_dataframe_5.pkl') #duplicate posts removed.

from sklearn.decomposition import TruncatedSVD

# co_occ_arr_sl=co_occ_arr[0:10000,0:10000] #sliced array
co_occ_arr_sl=co_occ_arr #Full cooccurence matrix

start = datetime.now()

from scipy.sparse import coo_matrix
co_occ_arr_sl_coo=coo_matrix(co_occ_arr_sl)
co_occ_arr_sl_coo = co_occ_arr_sl_coo.asfptype() #convert to COO format before arpack

svd = TruncatedSVD(n_components = 2, n_iter = 10, algorithm = "arpack")  #28.4 secs
# svd = TruncatedSVD(n_components = 2, n_iter = 10, algorithm = "randomized") #18.3 secs
Coocc_svd_matrix = svd.fit_transform(co_occ_arr_sl_coo)
Coocc_svd_matrix.shape

t = str(datetime.now()-start)
print("Time taken for svd: ",t[:-5])

print(type(Coocc_svd_matrix))

from numpy import save
save('ad5_svd_arpack_w5_bi.npy', Coocc_svd_matrix) #arpack algo used

# In[20]: svd array->plotting dataframe

coocc_svd_matrix = load('ad5_svd_arpack_w5_bi.npy') #Load arpack mode

ad_6=pd.read_pickle('App_dataframe_5.pkl') #df_8hrs is converted to string for col combining in ap

print("Custom vocab head 10: ",custom_vocab[:10])
custom_vocab[-10:-1]
cust_vocab = custom_vocab #entire word vocab. Common for all window sizes
# len(vocab_full) #41526
certs = ['gots','oeko','cosmos','ecocert','fsc'] #certifications
fruits = ['apple', 'orange']
rel_concs = ['fruits', 'certification','textile']

# vocab_to_plot = certs + fruits + rel_concs #specific plot list
vocab_to_plot = cust_vocab #full vocab plot

print("Making dict_to_plot..")
start=datetime.now()

#custom_vocab and co_occ vocab are indexed differently. coocc mat is based on custom(sorted) vocab
# dict_to_plot = inst.vocab_ind_to_plot(vocab_to_plot) #with above finding instantiating coocc cla

#dict_to_plot for the custom_vocab
def vocab_ind_to_plot(cust_vocab, vocab_to_plot):
        """
        Takes in the a list of full vocabulary and list to plot and returns a dictionary in the fo
        {word:index} which is a subeset of the vocabulary.
        """
        vocab_dict = dict(zip(cust_vocab, range(len(cust_vocab)))) #custom vocab is sorted
        dic_to_plot = {key:value for key, value in vocab_dict.items() if key in vocab_to_plot} #Re
        return dic_to_plot

dict_to_plot = vocab_ind_to_plot(cust_vocab, cust_vocab)

print("Dict_to_plot is ",dict_to_plot)

wl1=str(datetime.now()-start)
print("Time to make dict_to_plot: ", wl1[:-5])

data_list=[]
for word, ind in dict_to_plot.items():
    row_list=[word, coocc_svd_matrix[ind, 0], coocc_svd_matrix[ind, 1]]
    data_list.append(row_list)

# print(data_list)
type(coocc_svd_matrix)
print(coocc_svd_matrix.shape)

vocab_words_df= pd.DataFrame.from_records(data_list, columns=['word','x','y'])
wl2=str(datetime.now()-start)
print("Time till vocab_words_df ready: ", wl2[:-5]) #55secs for full vocab 41526 words
vocab_words_df
vocab_words_df.dtypes

vocab_words_df.to_pickle("ad5_vocab_words_svd_w5_bi.pkl", protocol=0)
```

```
# print(vocab_words_df)

# In[21]: Visualise plotly scatter  - Coocc SVD vectors
import plotly.express as px
import plotly.io as pio
pio.renderers.default='browser' #To plot in browser

vocab_words_df_full=pd.read_pickle('ad5_vocab_words_svd_w5_bi.pkl') #corrected word index

certs = ['gots','oeko','cosmos','ecocert','fsc'] #certifications
fruits = ['apple', 'orange', 'vanilla','cacao']
consc = ['sustainably', 'ethically']
rel_concs = ['fruits', 'certification','textile', 'cosmetics']
vocab_to_plot = certs + fruits + rel_concs

#filtered plot df
vocab_words_df = vocab_words_df_full[vocab_words_df_full['word'].isin(vocab_to_plot)]

fig = px.scatter(vocab_words_df_full, x="x", y="y", text="word", log_x=False, size_max=60,
                 # trendline = "rolling",
                 hover_name = 'word'
                 )
fig.update_traces(textposition='top center')
fig.show()


# In[22]: ## Word2vec model and saving .txt files
import os
import re
import time
from gensim.models import Word2Vec
from gensim.models import KeyedVectors #For loading the model
from sklearn.manifold import TSNE #for reducing the dimensions
from tqdm import tqdm
tqdm.pandas()

ad_7 = pd.read_pickle('App_dataframe_5.pkl')
words_rm_sw_lemt = [row_list for row_list in ad_7['rm_sw_lemt']]
print("number of sentences is: ",len(words_rm_sw_lemt)) #41526
train_sentences = words_rm_sw_lemt #list of lists sentences and words

start=datetime.now()
model = Word2Vec(sentences=train_sentences, sg=1,vector_size=300, window=5, min_count=1, workers=4
model.save("thesis_word2vec.model") #save a part of the training. avoid saving, is large
model = Word2Vec.load("thesis_word2vec.model") #load and continue training the sample
model.train([["hello", "world"]], total_examples=1, epochs=1) #train new samples if required
model.wv.most_similar('organic', topn = 20)
model.wv.save_word2vec_format('organic_glove_300d_w5.txt') #save all embeddings in this format

print("Time taken for creating model: ", str(datetime.now()-start)[:-5])#44 secs for 300d_w4

# In[23]: Create and Export Tsne dataframe by transforming w2c text files
# Note how before the saving the model, called as model.wv
# Takes 45-60 mins for full dataset.

w2v = KeyedVectors.load_word2vec_format('organic_glove_300d_w5.txt') # To load text file
# w2v = model.wv #use this if not loading from file and model is instantiated

def tsne_df(w2v): # dataframe conversion function
    "Creates and TSNE model and plots it"
    labels = []
    tokens = []

    words = list(w2v.index_to_key) #instead of model.wv.vocab
    len(words)
    words_cut=words[:1000] #To test time on a smaller df

    for word in words: #Test with 'words_cut' first
        tokens.append(w2v[word]) #Fetches the vector for the word. same as model.wv
        # print(model.wv[word])
        labels.append(word) #Word
        # print(word)

    tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)
    new_values = tsne_model.fit_transform(tokens) #fit_transform on vectors

    x = []
    y = []
    for value in new_values:
        x.append(value[0])
        y.append(value[1])

    tsne_df = pd.DataFrame(
    {'word': labels,
     'x': x,
     'y': y
```

```
    })

    tsne_df.to_pickle("tsne_300d_w2_df.pkl", protocol=0)
    print("Time taken for TSNE: ",str(datetime.now()-start)[:-5])

print("Creating new TSNE dataframe..")
start=datetime.now()
tsne_df(w2v) #call the function

# In[24]: Addition and subtraction logic word2vec
w2v = KeyedVectors.load_word2vec_format('organic_glove_300d_w5.txt') # To load text file
w2v.most_similar(positive=['king','woman'], negative= ['man'])
print(w2v.most_similar(positive=['oil', 'hempseed'])) #addition
print(w2v.most_similar(['oil', 'hempseed','free'])) #same as above

# In[25]: App simulation - w2v Scatter plot logic including hue.
# Toggle text dislay on and off.
# Bold word name in hover text
import plotly.express as px
import plotly.io as pio #To plot in browser
pio.renderers.default='browser'

vocab_to_plot = ['vanilla','organic','sustainable','cacao']
tsne_df_full=pd.read_pickle('tsne_100d_w5_df.pkl')
w2v = KeyedVectors.load_word2vec_format('organic_glove_300d.txt')
nearest_size = 5  #make input for nearest neighbor size

#index of input words
tsne_df_full['type'] = 'Sel'
ip_vocab_index=list(tsne_df_full[tsne_df_full['word'].isin(vocab_to_plot)].index.values)  #

#index of 10 words closest to inputs
clos_ten_out = []
for word in vocab_to_plot:
    a=w2v.most_similar(word, topn = nearest_size)
    clos_ten_in = [i[0] for i in a]
    clos_ten_out.append(clos_ten_in)

clos_ten_flat = [item for sublist in clos_ten_out for item in sublist]
clos_ten_index = list(tsne_df_full[tsne_df_full['word'].isin(clos_ten_flat)].index.values)  #

plot_index = ip_vocab_index + clos_ten_index
indices = [0,1,3,6,10,15]
indices = [0,1,3,6,10,15]
tsne_df_full.loc[ip_vocab_index,'type'] = 'ip'
tsne_df_full.loc[clos_ten_index,'type'] = 'near'

tsne_plot_df = tsne_df_full.loc[plot_index] #filter by index
tsne_plot_df
# tsne_plot_df = tsne_df_full[tsne_df_full['word'].isin(vocab_to_plot)] #filter by word

fig3 = px.scatter(tsne_plot_df, x="x", y="y", text="word",color='type', log_x=False, size_max=60,
                  # trendline = "rolling",
                  hover_name = 'word'
                  )
fig3.update_traces(textposition='top center')
fig3.show()

# In[26]: co-occurrence visualisations
#instantiate the sorted vocab in the co-occ code block
from numpy import load
co_occ_arr = load('ad5_co_occ_arr_w5_bi.npy') #shape 41526 * 41526
clf_table = pd.read_pickle("App_dataframe_5.pkl") #list col rm_sw_lemt

flat_list = [item for row_list in clf_table['rm_sw_lemt'] for item in row_list] #list col
len(flat_list) #total words in the corpus 799068 (df_8hr), 813922(App_dataframe_5)
len(set(flat_list)) #41526 (AD_5)

vocab = sorted(set(flat_list))

co_occ_df = pd.DataFrame(data = co_occ_arr,
                         index = vocab,
                         columns = vocab)
# co_occ_df.to_pickle("co_occ_df.pkl", protocol=0) #hangs the kernel. do not use!

#datastructure for bar chart
type(co_occ_df['organic'])
word = 'textile' #test
co_occ_plot = co_occ_df[word].sort_values(ascending=False).head(30)
co_occ_plot = co_occ_plot.reset_index() #make'words' a column
mapping = {co_occ_plot.columns[0]:'word', co_occ_plot.columns[1]: 'counts'}
co_occ_plot = co_occ_plot.rename(columns=mapping)
co_occ_plot

```

7.4 Python Code – Plotly dash web app

```
# In[27]: #App code block. To be run as a script. Can host on heroku app server.
print("\nStarting dashboard app...")
##############Import libraries and dataset
import pandas as pd      #(version 1.0.0)
import plotly            #(version 4.5.4) pip install plotly==4.5.4
import plotly.express as px
import dash              #(version 1.9.1) pip install dash==1.9.1
import dash_table  #interactive datatable
import dash_core_components as dcc #display elements
import dash_html_components as html #display elements
from dash.dependencies import Input, Output, State #for app callbacks
import dash_bootstrap_components as dbc #display elements
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
from nltk.corpus import stopwords
from datetime import datetime
from numpy import load
import os
import re
import time
from gensim.models import Word2Vec #for word2vec model
from gensim.models import KeyedVectors #For loading a saved model
from scipy import stats # for zscore
import numpy as np # for filtering zscore
from collections import Counter #co-occurence counts
import ast #literal evals for list types

############

app_launch_start=datetime.now() #Set start time for program start
print("Loading the dataframe..")

#Supporting files to load
tsne_300d_w5_df = pd.read_pickle('tsne_300d_w5_df.pkl')

try: #try and except for quick reloading of the w2v object.
    w2v
except:
    w2v = KeyedVectors.load_word2vec_format('organic_glove_300d_w5.txt') #initialise only once per

#Dashtables
df_updated = pd.read_pickle("df_8hrs.pkl") #filtered 8hr dataset
disp1=['ix','caption_processed_4','hashtags','cap_mentions','web_links'] #select columns to displa
df_disp_1 = df_updated[disp1]
df_disp_1.rename(columns={"caption_processed_4": "description"},inplace=True)

clf_table=pd.read_pickle("df_8hrs.pkl") #z-score table
clf_table['Inf_Non']='' #new entry columns
clf_table['Domain']='' #new entry columns

clf_view_6=['ix','time_delta_hours','likeCount','commentCount','likes_pr_min','caption_processed_4
clf_dashtable=clf_table[clf_view_6]
clf_dashtable.rename(columns={"caption_processed_4": "description"},inplace=True)

#------get vocab & initialise co-occurence dataframe
co_occ_table = pd.read_pickle("App_dataframe_5.pkl") #list col rm_sw_lemt
flat_list_co_occ = [item for row_list in co_occ_table['rm_sw_lemt'] for item in row_list] #list co
vocab = sorted(set(flat_list_co_occ))

co_occ_arr = load('ad5_co_occ_arr_w5_bi.npy') #shape 41526
co_occ_df = pd.DataFrame(data = co_occ_arr,
                         index = vocab,
                         columns = vocab)
#------


print("Dataframes loaded")
t=datetime.now() - app_launch_start
s=str(t)
print("Dataframes loaded in: ", s[:-5], "\n\n")

############

app = dash.Dash(__name__) #loads any CSS directly added in the "assets" folder

al=str(datetime.now() - app_launch_start) #start time logged at start of program code
print("Total time to load files and launch app", al[:-5])

#-------------------------------------------------------------
#Initialising variables outside the callbacks

#for dropdowns
col_sels = ['description','hashtags','cap_mentions','web_links']   #values for dropdown_1
col_sels_2 = ['description','hashtags','web_links','cap_mentions','username','fullName']   #values
```

```
input_boxs = ['text','text']
input_boxes1 = ('text','text','text')

#Default vocab to plot
certs = ['gots','oeko','cosmos','ecocert','fsc'] #certifications
fruits = ['apple', 'orange', 'vanilla','cacao']
consc = ['sustainably', 'ethically']
rel_concs = ['fruits', 'certification','textile', 'cosmetics']
vocab_plot_list = certs + fruits + rel_concs

#other arbitrary initialisations
z_init = 5
z_init2 = 5
wc_init = 0

#-----------------------------------------------------------

app.layout = dbc.Container([
################################## ROW1-Instagram Data, Sel-Desel button_1,Wordcloud ############
    dbc.Row([
        dbc.Col(html.H2("Instagram DataTable"), width={'size':3}),
        dbc.Col(
            dbc.Button(id='sel-button', n_clicks=0, children="Sel_all", className="mt-5 mr-2"),
            width={'size': 0.5}, style={'textAlign': "left"}), #another way to align
        dbc.Col(
            dbc.Button(id='desel-button', n_clicks=0, children="Des_all", className="mt-5"),
            width={'size': 0.5}, style={'textAlign':"left"}),
        dbc.Col(html.H2("Wordcloud - Word frequency"), width={'size':5, 'offset':2}, style={'textA
        ], no_gutters=False),

################################## ROW2-Dropdown ##########################
    dbc.Row([
        dbc.Col([
            dcc.Dropdown(id='my-dropdown', multi=True,
                      options=[{'label': x, 'value': x} for x in col_sels],
                      value=["description"], #initial values to pass
                      style={'width':'690px'}
                        )])
            ]),

################################## ROW3-Dashtable & Wordcloud ##########################
    dbc.Row([
        dbc.Col(
            dash_table.DataTable(
                id='datatable_id',
                data=df_disp_1.to_dict('records'),
                columns=[
                    {"name": i, "id": i, "deletable": False, "selectable": True} for i in df_disp_
                ],
                editable=False,
                filter_action="native",
                sort_action="native",
                sort_mode="multi",
                row_selectable="multi",
                row_deletable=False,
                selected_rows=[], #this parameter gets updated by interactive selection
                page_action="native",
                page_current= 0,
                page_size= 10,
                fixed_rows={ 'headers': True, 'data': 0 }, #if the header style is not defined and
                style_cell={ #creates a wrapping of the data to constrain column widths. applies t
                    'whiteSpace': 'normal',
                    'height': 'auto',
                    'minWidth': '50px', 'width' : '50px','maxwidth': '50px',  # minwidth of col.
                    },
                style_table={ #For parameters of the table container
                    'height': '300px',
                    'width': '690px',
                    'overflowY': 'auto'
                },
                style_cell_conditional=[
                    {'if': {'column_id': 'ix'},
                     'width': '50px', 'textAlign': 'left'}, #Setting the px values 1400px seems to
                    {'if': {'column_id': 'description'},
                     'width': '200px', 'textAlign': 'left'},
                    {'if': {'column_id': 'hashtags'},
                     'width': '200px', 'textAlign': 'left'},
                    {'if': {'column_id': 'cap_mentions'},
                     'width': '125px', 'textAlign': 'left'},
                    {'if': {'column_id': 'web_links'},
                     'width': '125px', 'textAlign': 'left'},
                ],
            ),
            width={'size': 6}),
        dbc.Col(
```

```
            dcc.Graph(id='wordcloud', figure={}, config={'displayModeBar': True},
                      style={'width':'600px' ,'height':'350px'}),
                )
            ],no_gutters=False),

    ################################## ROW4-Headers 2  ##########################
        dbc.Row([
            dbc.Col(html.H2("Vector exploration"), width={'size':2}),
            dbc.Col(
                dbc.Button(id='add1_button', n_clicks=0, children="Add", className="mt-5 mr-2"),
                width={'size': 0.5}, style={'textAlign': "left"}),
            dbc.Col(
                dbc.Button(id='rem1_button', n_clicks=0, children="Rem", className="mt-5 mr-2"),
                width={'size': 0.5}, style={'textAlign': "left"}),
            dbc.Col(
                dbc.Button(id='clr1_button', n_clicks=0, children="Clr", className="mt-5 mr-2"),
                width={'size': 0.5}, style={'textAlign': "left"}),
            dbc.Col(
                dbc.Button(id='plt1_button', n_clicks=0, children="Plot", className="mt-5 mr-2"),
                width={'size': 0.5}, style={'textAlign': "left"}),
            dbc.Col(
                dbc.Button(id='words_tog', n_clicks=0, children="words", className="mt-5 mr-2"),
                width={'size': 1, 'offset':1}, style={'textAlign': "left"}), #size 0.5
            dbc.Col(html.H2("Vector math"), width={'size':5}, style={'textAlign': "center"}), #If
            ], no_gutters=False),

    ################################## ROW5-Input_boxs ##########################
        dbc.Row([
            dbc.Col(
                dcc.Input(
                id='input_1',
                type='text',
                placeholder="",          #pass list vocab to display here
                debounce=False,           # changes to input are sent to Dash server only on enter or
                pattern=r"^[A-Za-z].*",  # Regex: string must start with letters only
                spellCheck=True,
                inputMode='latin',       # provides a hint to browser on type of data that might be en
                name='text',             # the name of the control, which is submitted with the form d
                list='browser',          # identifies a list of pre-defined options to suggest to the
                n_submit=0,              # number of times the Enter key was pressed while the input h
                n_submit_timestamp=-1,   # last time that Enter was pressed
                autoFocus=False,          # the element should be automatically focused after the page
                n_blur=0,                # number of times the input lost focus
                n_blur_timestamp=-1,     # last time the input lost focus.
                size="30"                # Width of box which can display placeholders
                    ), width={'size': 4}, style={'textAlign': "left"}   #size was 6
                ),
            #---------------Word2vec_2 math buttons------------
            dbc.Col(
                dcc.Input(
                    id='input_2',
                    type='text',
                    placeholder='',  # A hint to the user of what can be entered in the control
                    debounce=True,                          # Changes to input are sent to Dash server onl
                    min=2015, max=2019, step=1,             # Ranges of numeric value. Step refers to incr
                    minLength=0, maxLength=50,              # Ranges for character length inside input box
                    autoComplete='on',
                    disabled=False,                         # Disable input box
                    readOnly=False,                         # Make input box read only
                    required=False,                         # Require user to insert something into input
                    size="7",                              # Sets width of box. If exceeds col width then
                    ), width={'size': 1, 'offset' : 2}, style={'textAlign': "center"}  #text size 5 co
                ),
            dbc.Col(html.H3("-"), width={'size':0.5}, style={'textAlign' : "center"}), #total width to
            dbc.Col(
                dcc.Input(
                    id='input_3',
                    type='text',
                    placeholder='',  # A hint to the user of what can be entered in the control
                    debounce=True,                          # Changes to input are sent to Dash server onl
                    min=2015, max=2019, step=1,             # Ranges of numeric value. Step refers to incr
                    minLength=0, maxLength=50,              # Ranges for character length inside input box
                    autoComplete='on',
                    disabled=False,                         # Disable input box
                    readOnly=False,                         # Make input box read only
                    required=False,                         # Require user to insert something into input
                    size="7",                              # Number of characters that will be visible ins
                    ), width={'size': 1}, style={'textAlign': "center"}
                ),
            dbc.Col(html.H3("+"),width={'size': 0.5}, style={'textAlign' : "center"}),
            dbc.Col(
                dcc.Input(
                    id='input_4',
                    type='text',
                    placeholder='',  # A hint to the user of what can be entered in the control
                    debounce=True,                          # Changes to input are sent to Dash server onl
```

```
                min=2015, max=2019, step=1,          # Ranges of numeric value. Step refers to incr
                minLength=0, maxLength=50,           # Ranges for character length inside input box
                autoComplete='on',
                disabled=False,                      # Disable input box
                readOnly=False,                      # Make input box read only
                required=False,                      # Require user to insert something into input
                size="7",                           # Number of characters that will be visible ins
                ), width={'size': 1}, style={'textAlign': "center"}
            ),
        dbc.Col(html.H3("="),width={'size':0.5}, style={'textAlign' : "center"}),
        dbc.Col(
            dcc.Input(
                id='output_1',
                type='text',
                placeholder='',  # A hint to the user of what can be entered in the control
                debounce=True,                       # Changes to input are sent to Dash server onl
                min=2015, max=2019, step=1,          # Ranges of numeric value. Step refers to incr
                minLength=0, maxLength=50,           # Ranges for character length inside input box
                autoComplete='on',
                disabled=True,                      # Disable input box
                readOnly=False,                      # Make input box read only
                required=False,                      # Require user to insert something into input
                size="7",                           # Number of characters that will be visible ins
                ), width={'size': 1}, style={'textAlign': "center"}
            ),
        ],no_gutters=False),

################################### ROW6-word2vec_1  ##########################
    dbc.Row([
        dbc.Col(
            dcc.Graph(id='word2vec_1'),
            width={'size': 6}
                ),
        dbc.Col(
            dcc.Graph(id='word2vec_2'),
            width={'size': 6}
                )
            ],no_gutters=False),

################################### ROW7-range slider  ##########################
    dbc.Row([
        dbc.Col(
            dcc.Slider(
                id='my_slider',
                min=0,
                max=20,
                step=1,
                value=0,
                marks={
                    0: {'label': '0', 'style': {'color': '#77b0b1'}},
                    5: {'label': '5'},
                    10: {'label': '10'},
                    15: {'label': '15', 'style': {'color': '#f50'}},
                    20: {'label': '20'}
                    }),
            width={'size': 6}),
        dbc.Col(
            dcc.Slider(
                id='my_slider_2',
                min=0,
                max=20,
                step=1,
                value=0, #default value
                marks={
                    0: {'label': '0', 'style': {'color': '#77b0b1'}},
                    5: {'label': '5'},
                    10: {'label': '10'},
                    15: {'label': '15', 'style': {'color': '#f50'}},
                    20: {'label': '20'}
                    }),
            width={'size': 6})
            ],no_gutters=False),

############################## ROW8-"Classification Dashtable", Sel-Desel ######################
    dbc.Row([
        dbc.Col(html.H2("Z-Score adjustable DataTable"), width={'size':2}),
        dbc.Col(
            dbc.Button(id='sel-button_2', n_clicks=0, children="Sel_all", className="mt-5 mr-2"),
                    width={'size': 0.5}, style={'textAlign': "left"}),
        dbc.Col(
            dbc.Button(id='desel-button_2', n_clicks=0, children="Des_all", className="mt-5"),
            width={'size': 0.5}, style={'textAlign':"left"}),
        dbc.Col(html.H2("Wordcloud - Post frequency"), width={'size':5, 'offset':3}, style={'textA
            ],no_gutters=False),

############################## ROW9-Dropdown_2 ##############################
```

```
    dbc.Row([
        dbc.Col(
            dcc.Dropdown(id='my-dropdown_2', multi=True,
                      options=[{'label': x, 'value': x} for x in col_sels_2],
                      value=["description"], #initial values to pass
                      style={'width':'490px'}
                        ), width={ 'offset':0}),
        dbc.Col(
            dcc.Slider(
                id='my_slider_zscore',
                min=0,
                max=4,
                step=0.25,
                value=0,
                marks={
                    0: {'label': '0', 'style': {'color': '#77b0b1'}},
                    1: {'label': '1'},
                    2: {'label': '2'},
                    3: {'label': '3', 'style': {'color': '#f50'}},
                    4: {'label': '4'}
                    }),
            width={'size': 2}),
            ], no_gutters = False),

############################ ROW10-Clasification dashtable, WordCloud_2 ##########################
    dbc.Row([
            dbc.Col(
                dash_table.DataTable(
                    id='datatable_2',
                    data=clf_dashtable.to_dict('records'),
                    columns=[
                        {"name": i, "id": i, "deletable": False, "selectable": True} for i in clf_
                    ],
                    editable=True,
                    filter_action="native",
                    sort_action="native",
                    sort_mode="multi",
                    row_selectable="multi",
                    row_deletable=False,
                    selected_rows=[], #this parameter gets updated by interactive selection
                    page_action="native",
                    page_current= 0,
                    page_size= 10,
                    fixed_rows={ 'headers': True, 'data': 0 }, #if the header style is not defined
                    style_cell={ #creates a wrapping of the data to constrain column widths. appli
                        'whiteSpace': 'normal',
                        'height': 'auto',
                        'minWidth': '50px', 'width' : '50px','maxwidth': '50px',  # minwidth of co
                        },
                    style_table={ #For parameters of the table container
                        'height': '600px', #was 300px
                        'width': '690px',
                        'overflowY': 'auto'
                    },
                    style_cell_conditional=[
                        {'if': {'column_id': 'description'},
                         'width': '200px', 'textAlign': 'left'},
                    ],
                    export_format='xlsx',
                ),
                width={'size': 6}),
             dbc.Col(
                dcc.Graph(id='wordcloud_2', figure={}, config={'displayModeBar': True},
                          style={'width':'600px' ,'height':'600px'}), #ht was 350px
                )
            ],no_gutters=False),

############################ ROW11-Header "Posts count of unique words"  ##########################
    dbc.Row([
        dbc.Col(html.H2("Posts count of distinct words"), width={'size':2}),
        dbc.Col(html.H2("Co-occurrence counts"), width={'size':2, 'offset':4}),
            ],no_gutters=False),

############################ ROW12-Slider for most common ##############################
    dbc.Row([
        dbc.Col(
            dcc.Slider(
                id='my_slider_mc',
                min=10,
                max=30,
                step=1,
                value=10,
                marks={
                    10: {'label': '10', 'style': {'color': '#77b0b1'}},
                    20: {'label': '20'},
                    30: {'label': '30'},
```

```
                    40: {'label': '40', 'style': {'color': '#f50'}},
                    }),
            width={'size': 2}),
        dbc.Col(
            dcc.Input(
                id='co_occ_ip',
                type='text',
                placeholder='',  # A hint to the user of what can be entered in the control
                debounce=True,                      # Changes to input are sent to Dash server onl
                minLength=0, maxLength=50,          # Ranges for character length inside input box
                autoComplete='on',
                disabled=False,                     # Disable input box
                readOnly=False,                     # Make input box read only
                required=False,                     # Require user to insert something into input
                size="7",                          # Sets width of box. If exceeds col width then
                ), width={'size': 2, 'offset' : 4}, style={'textAlign': "center"}),  #text size 5
        dbc.Col(
            dcc.Slider(
                id='slider_co_occ',
                min=10,
                max=40,
                step=1,
                value=10,
                marks={
                    10: {'label': '10', 'style': {'color': '#77b0b1'}},
                    20: {'label': '20'},
                    30: {'label': '30'},
                    40: {'label': '40', 'style': {'color': '#f50'}},
                    }),
            width={'size': 2})
            ], no_gutters=False),

############################ ROW13-Bar_chart for distinct word count ##############################
    dbc.Row([
        dbc.Col(
            dcc.Graph(id='bar_chart_1'),
            width={'size': 6}
                ),
        dbc.Col(
            dcc.Graph(id='bar_chart_2'),
            width={'size': 6}
                ),
            ],no_gutters=False),
        ],fluid=True) #closes initial dbc container

################################ App Callbacks ##################################################
         ############           #CALLBACKS         ############
################################ App Callbacks ##################################################

################################ Barcharts co-occ counts#################################
@app.callback(
    Output(component_id='bar_chart_2', component_property='figure'),
    [Input(component_id='co_occ_ip', component_property='value'),
    Input(component_id='slider_co_occ', component_property='value')],
    prevent_initial_call=True,
)

def barchart_2(co_occ_ip,slider):
    word = co_occ_ip
    head_values = slider
    try:
        co_occ_plot = co_occ_df[word].sort_values(ascending=False).head(head_values)
        co_occ_plot = co_occ_plot.reset_index()
    except KeyError as e:
        print("\n word not in vocab: ", e)
    try:
        mapping = {co_occ_plot.columns[0]:'word', co_occ_plot.columns[1]: 'counts'}
        co_occ_plot = co_occ_plot.rename(columns=mapping)
        fig_br2 = px.bar(co_occ_plot, x="counts", y="word", title="Count of co-occurences", orient
        fig_br2.update_layout(yaxis=dict(autorange="reversed"))
        return fig_br2
    except UnboundLocalError as e:
        print(e)
        raise dash.exceptions.PreventUpdate

################################ Barcharts distinct words per post ###############
@app.callback(
    Output(component_id='bar_chart_1', component_property='figure'),
    [Input(component_id='datatable_2', component_property='selected_rows'),
    Input(component_id='my-dropdown_2', component_property='value'),
    Input(component_id='my_slider_zscore', component_property='value'),
    Input(component_id='my_slider_mc', component_property='value')],
    prevent_initial_call=False,
)

def barchart_1(chosen_rows, chosen_cols, zscore, mostcomm):
```

```
    global z_init2, flat_list, fig_br1
    ctx = dash.callback_context
    trigger = (ctx.triggered[0]['prop_id'].split('.')[0])

    z = np.abs(stats.zscore(clf_dashtable[['time_delta_hours','likes_pr_min']])) #assign z-score t
    view_rm_lm= ['rm_sw_lemt']
    view_rm_lm = clf_view_6 + view_rm_lm
    clf_dashtable_z=clf_table[view_rm_lm].iloc[np.unique(np.where(z > zscore)[0])] #table should h
    #clean up index for "select_all" functionality
    clf_dashtable_z.drop(['ix'], axis=1, inplace = True) #drop any older 'ix' columns
    clf_dashtable_z.reset_index(inplace= True,drop=True) #reset and drop old index
    clf_dashtable_z.reset_index(inplace= True) #create new ix column
    clf_dashtable_z.rename(columns={"index": "ix"},inplace=True)

    if len(chosen_cols) > 0: #consider rm_sw_lemt instead
        if 'description' in chosen_cols:
            chosen_cols.remove('description')
            chosen_cols.append('rm_sw_lemt')
            print("Chosen cols now contain",chosen_cols)#atleast 1 col to be selected
        if len(chosen_rows)==0: #if no rows selected consider all rows of z score filtered table-d
            #------fast render optimization
            if ((z_init2 == zscore) and (trigger not in ['my_slider_mc','my-dropdown_2'])): #if no
                print("\nz_init2 is equal to zscore, z_init2 value is: ", z_init2)
                return (fig_br1)
            if ((z_init2 == zscore) and (trigger in ['my_slider_mc'])): #uses flat_list global for
                counts = dict(Counter(flat_list).most_common(mostcomm))
                labels, values = zip(*counts.items())
                indSort = np.argsort(values)[::-1] # sort your values in descending order
                labels = np.array(labels)[indSort]
                values = np.array(values)[indSort]
                indexes = np.arange(len(labels))
                word_count_df = pd.DataFrame({'word': labels, 'posts_count': values}, columns=['wo
                fig_br_12 = px.bar(word_count_df, x="posts_count", y="word", title="Lemmatised wor
                fig_br_12.update_layout(yaxis=dict(autorange="reversed"))
                return (fig_br_12)
            else: #initialise object and set z_init value
                print("\nElse block reached, z_init2 value is; ", z_init2)
                z_init2 = zscore
                print("z_init2 value updated to: ", z_init2)
            #------
                df_filtered = clf_dashtable_z[chosen_cols]
                df_filtered['comb_cols'] = df_filtered[df_filtered.columns[0:]].apply(
                    lambda x: ' '.join(x.dropna().astype(str)),
                    axis=1)#Combine multiple columns into a single series
                df_filtered['comb_cols'] = df_filtered['comb_cols'].apply(
                    lambda x: ' '.join(set(x.split())))#Use only unique words per post in wordclou
                #Create word counts dataframe
                flat_list = [item for row_list in df_filtered['comb_cols'] for item in row_list.sp
                counts = dict(Counter(flat_list).most_common(mostcomm))
                labels, values = zip(*counts.items())
                indSort = np.argsort(values)[::-1] # sort your values in descending order
                labels = np.array(labels)[indSort]
                values = np.array(values)[indSort]
                indexes = np.arange(len(labels))
                word_count_df = pd.DataFrame({'word': labels, 'posts_count': values}, columns=['wo
                fig_br1 = px.bar(word_count_df, x="posts_count", y="word", title="Lemmatised words
                fig_br1.update_layout(yaxis=dict(autorange="reversed"))
                return (fig_br1)
        elif len(chosen_rows) > 0 : #filter to selected rows
            df_filtered = clf_dashtable_z[chosen_cols]
            df_filtered = df_filtered[df_filtered.index.isin(chosen_rows)]
    elif len(chosen_cols) == 0:
        raise dash.exceptions.PreventUpdate #no update if no cols selected

    ##figure logic repeated for row selections
    df_filtered['comb_cols'] = df_filtered[df_filtered.columns[0:]].apply(
        lambda x: ' '.join(x.dropna().astype(str)),
        axis=1)
    df_filtered['comb_cols'] = df_filtered['comb_cols'].apply(
        lambda x: ' '.join(set(x.split())))
    flat_list = [item for row_list in df_filtered['comb_cols'] for item in row_list.split()]
    counts = dict(Counter(flat_list).most_common(mostcomm))
    labels, values = zip(*counts.items())
    indSort = np.argsort(values)[::-1] # sort your values in descending order
    labels = np.array(labels)[indSort]
    values = np.array(values)[indSort]
    indexes = np.arange(len(labels))
    word_count_df = pd.DataFrame({'word': labels, 'posts_count': values}, columns=['word', 'posts_
    fig_br_1 = px.bar(word_count_df, x="posts_count", y="word", title="Lemmatised words by post co
    fig_br_1.update_layout(yaxis=dict(autorange="reversed"))
    return (fig_br_1) #note change in return object name

################################ Column highlighting Dashtable2 ################################
@app.callback(
    Output('datatable_2', 'style_data_conditional'),
    Input(component_id='my-dropdown_2', component_property='value')
```

```
)

def update_styles(selected_columns):
    return [{
        'if': { 'column_id': i },
        'background_color': '#D2F3FF'
    } for i in selected_columns]

############################### Dashtable_2 Zscore virtual table###############################
@app.callback(
    Output('datatable_2', 'data'), #virtual_data is displayed table. Even after filtering. #refere
    Input('my_slider_zscore','value'),
    prevent_initial_call=False
)

def zscore_outliers(zscore):
    z = np.abs(stats.zscore(clf_dashtable[['time_delta_hours','likes_pr_min']])) #assign z-score t
    clf_dashtable_z=clf_dashtable.iloc[np.unique(np.where(z > zscore)[0])] #create outlier table
    #clean up index for "select_all" functionality
    clf_dashtable_z.drop(['ix'], axis=1, inplace = True) #drop any older 'ix' columns
    clf_dashtable_z.reset_index(inplace= True,drop=True) #reset and drop old index
    clf_dashtable_z.reset_index(inplace= True) #create new ix column
    clf_dashtable_z.rename(columns={"index": "ix"},inplace=True)
    print("\nZscore slider value is: ", zscore)
    print("No of rows are: ", len(clf_dashtable_z))
    return clf_dashtable_z.to_dict('records')

############################### Dashtable_2 Sel_Desel button_2 ###############################
@app.callback(
    [Output('datatable_2', 'selected_rows')], #references ordered 0 to n index of larger table irr
    [Input('sel-button_2', 'n_clicks'),
    Input('desel-button_2', 'n_clicks'),
    Input('my_slider_zscore','value')],
    [State('datatable_2', 'derived_virtual_selected_rows'), #virtual selected row is what is selec
     State('datatable_2', 'derived_virtual_data')], #virtual_data is displayed table. Even after f
    prevent_initial_call=True
)

def select_deselect(selbtn, deselbtn, z_score_trigger,selected_rows,filtered_table):
    ctx = dash.callback_context
    if ctx.triggered:
        print(ctx.triggered)
        trigger = (ctx.triggered[0]['prop_id'].split('.')[0])
        if trigger == 'sel-button_2':
            print("\n\nSelect button clicked")
            print("Length of Selected_rows is: ", len(selected_rows))
            wc_list= []
            wc_list=[[row['ix'] for row in filtered_table]]
            wc_list = [[int(i) for i in wc_list[0]]] #convert to int for index reference
            print("Displayed table has:", len(filtered_table), 'rows')
            print("Wordcloud list contains:", len(wc_list[0]), "elements")
            return wc_list
        else:
            print("\n\nDeselect button clicked")
            print("Length of Selected_rows is: ", len(selected_rows))
            return [[]]

############################### Wordcloud_2  ###############################
@app.callback(
    Output('wordcloud_2','figure'),
    [Input(component_id='datatable_2',component_property='selected_rows'),
    Input(component_id='my-dropdown_2', component_property='value'),
    Input('my_slider_zscore','value')],
    prevent_initial_call=False
)

def ren_wordcloud(chosen_rows, chosen_cols, zscore):
    global z_init, fig_wordcloud2 #for try and except. otherwise destroyed at end of callback.
    ctx = dash.callback_context
    trigger = (ctx.triggered[0]['prop_id'].split('.')[0])

    z = np.abs(stats.zscore(clf_dashtable[['time_delta_hours','likes_pr_min']])) #assign z-score t
    clf_dashtable_z=clf_dashtable.iloc[np.unique(np.where(z > zscore)[0])] #create outlier table
    #clean up index for "select_all" functionality
    clf_dashtable_z.drop(['ix'], axis=1, inplace = True) #drop any older 'ix' columns
    clf_dashtable_z.reset_index(inplace= True,drop=True) #reset and drop old index
    clf_dashtable_z.reset_index(inplace= True) #create new ix column
    clf_dashtable_z.rename(columns={"index": "ix"},inplace=True)

    if len(chosen_cols) > 0: #atleast 1 col to be selected
        if len(chosen_rows)==0:
            #-----fast render optimization
            if ((z_init == zscore) and (trigger != 'my-dropdown_2')): #if no change in the score r
                print("\nz_init is equal to zscore, z_init value is: ", z_init)
                return fig_wordcloud2
            else: #initialise object and set z_init value
```

```
                print("\nElse block reached, z_init value is; ", z_init)
                z_init=zscore
                print("z_init value updated to: ", z_init)
            #-----
                #if no rows selected consider all rows of z score filtered table-defined again
                df_filtered = clf_dashtable_z[chosen_cols]
                print("Initialising full word cloud..")
                df_filtered['comb_cols'] = df_filtered[df_filtered.columns[0:]].apply( #Combine mu
                    lambda x: ' '.join(x.dropna().astype(str)),
                    axis=1)
                df_filtered['comb_cols'] = df_filtered['comb_cols'].apply(
                    lambda x: ' '.join(set(x.split())))#Use only unique words per post in wordclou

                en_stopwords = stopwords.words('english')
                fr_stopwords = stopwords.words('french')
                web_links_sw = ['www','http','https','com']
                combined_stopwords = en_stopwords + fr_stopwords + web_links_sw
                print("\nNo of rows in WC are: ",len(df_filtered))
                wordcloud = WordCloud(max_words=100,
                                      # stopwords= combined_stopwords,
                                      colormap='tab20b',
                                      background_color='white',
                                      width=1200, #1200,1700
                                      height=1000, #1000
                                      random_state=1).generate_from_text(' '.join(df_filtered['com

                fig_wordcloud2 = px.imshow(wordcloud, template='ggplot2')#title="Post frequency"
                fig_wordcloud2.update_layout(margin=dict(l=0, r=0,b=0,t=0))
                fig_wordcloud2.update_xaxes(visible=False)
                fig_wordcloud2.update_yaxes(visible=False)
                return fig_wordcloud2
        elif len(chosen_rows) > 0 :
            df_filtered = clf_dashtable_z[chosen_cols]
            df_filtered = df_filtered[df_filtered.index.isin(chosen_rows)]
    elif len(chosen_cols) == 0:
        raise dash.exceptions.PreventUpdate
## load wordcloud only once
    df_filtered['comb_cols'] = df_filtered[df_filtered.columns[0:]].apply( #Combine multiple colum
        lambda x: ' '.join(x.dropna().astype(str)),
        axis=1)
    df_filtered['comb_cols'] = df_filtered['comb_cols'].apply(
        lambda x: ' '.join(set(x.split()))) #because no order, bigrams become irrelevant

    en_stopwords = stopwords.words('english')
    fr_stopwords = stopwords.words('french')
    web_links_sw = ['www','http','https','com']
    combined_stopwords = en_stopwords + fr_stopwords + web_links_sw
    print("\nNo of rows in WC are: ",len(df_filtered))
    wordcloud = WordCloud(max_words=100,
                          # stopwords= combined_stopwords,
                          colormap='tab20b',
                          background_color='white',
                          width=1200, #1200,1700
                          height=1000, #1000
                          random_state=1).generate_from_text(' '.join(df_filtered['comb_cols'])) #

    fig_wordcloud_2 = px.imshow(wordcloud, template='ggplot2')#title="Post frequency"
    fig_wordcloud_2.update_layout(margin=dict(l=0, r=0,b=0,t=0))
    fig_wordcloud_2.update_xaxes(visible=False)
    fig_wordcloud_2.update_yaxes(visible=False)
    return fig_wordcloud_2

################################ word2vec_2 math graph ################################
@app.callback(
    [Output(component_id='word2vec_2', component_property='figure'),
    Output(component_id='output_1', component_property='placeholder')],
     [Input('input_2','value'),
      Input('input_3','value'),
      Input('input_4','value'),
      Input('my_slider_2','value'),
      Input('words_tog','n_clicks')],
    prevent_initial_call=False
    )

def word2vec_math(ip1,ip2,ip3,slider2_val,word_tog):
    tsne_df_full = tsne_300d_w5_df
    vocab_plot_list_2 = [ip1,ip2,ip3]
    print("\n\nvocab_plot_list is: ", vocab_plot_list_2)
    vocab_to_plot = list(filter(None, vocab_plot_list_2))
    print("Filtered vocab_to_plot is: ", vocab_to_plot) #Only non-empty inputs
    print("Slider2 val: ",slider2_val)
    nearest_size = slider2_val #nearest size defined
    tsne_df_full['type'] = 'Sel'
    ip_vocab_index=list(tsne_df_full[tsne_df_full['word'].isin(vocab_to_plot)].index.values)
    clos_ten_out = [] #index of closest 10 words
    try:
```

```
        for i,word in enumerate(vocab_to_plot):
            ix_word = tsne_df_full[tsne_df_full['word'].isin([word])].index.values
            tsne_df_full.loc[ix_word,'type'] = "ip_{}".format(i)
            if nearest_size != 0:
                a=w2v.most_similar(word, topn = nearest_size)
                clos_ten_in = [i[0] for i in a] #grab vectors and not similarity probability
                clos_ten_out.append(clos_ten_in) #gather words in a single list
            else:
                 clos_ten_out = []
    except ValueError as e:#Empty word None input
            print(e)
            pass
    except KeyError as e: #Word not in vocab
            print(e)
            pass

    ip_nearest=[]
    clos_ten_dict={}
    try:
        for i,sub_list in enumerate(clos_ten_out):
            clos_ten_dict[i] = list(tsne_df_full[tsne_df_full['word'].isin(sub_list)].index.values
            tsne_df_full.loc[clos_ten_dict[i],'type'] = "near_ip_{}".format(i)
            ip_nearest.append(clos_ten_dict[i]) #index values
    except Exception as e:
            print("clos_ten_out is empty: ", e)
            pass
    ip_nearest_ix = [item for sublist in ip_nearest for item in sublist]
    try: #Try catches no input case
        r=w2v.most_similar(positive=[ip1,ip3], negative= [ip2], topn = nearest_size+1)
        res_ten_in = [i[0] for i in r]
        print("Res words are: ", res_ten_in)
        res_ten_index = list(tsne_df_full[tsne_df_full['word'].isin(res_ten_in)].index.values)
    except Exception as e:      #TypeError
        print("Trying addition ")
        try:
            r=w2v.most_similar(positive=[ip1,ip3], topn = nearest_size+1)
            res_ten_in = [i[0] for i in r]
            print("Res words of addition: ", res_ten_in)
            res_ten_index = list(tsne_df_full[tsne_df_full['word'].isin(res_ten_in)].index.values)
        except Exception as e:
            print("Trying subtraction: ")
            try:
                r=w2v.most_similar(positive=[ip1], negative= [ip2], topn = nearest_size+1)
                res_ten_in = [i[0] for i in r]
                print("Res words of subtraction: ", res_ten_in)
                res_ten_index = list(tsne_df_full[tsne_df_full['word'].isin(res_ten_in)].index.val
            except Exception as e:
                print("Insufficient inputs",e)
                pass
    try:
        tsne_df_full.loc[res_ten_index[0],'type'] = 'res'
        tsne_df_full.loc[res_ten_index[1:],'type'] = 'near_res'
    except Exception as e:
        print("res nearest not defined: ", e)
        pass
    try:
        plot_index = ip_vocab_index + ip_nearest_ix + res_ten_index
    except Exception as e:
        print("plot index not fully defined: ",e)
        pass
    try:
        print("res_ten_index, word, type is: ", res_ten_index,tsne_df_full.loc[res_ten_index,'word
    except UnboundLocalError as e:
        print(e)
        pass
    try:
        tsne_plot_df = tsne_df_full.loc[plot_index]
    except UnboundLocalError as e:
        print(e)
        pass
    ctx = dash.callback_context
    if ctx.triggered:
        # print(ctx.triggered)
        trigger = (ctx.triggered[0]['prop_id'].split('.')[0])
        print("trigger is: ", trigger)
        print("word_tog nclick value is: ", word_tog)
        if trigger == 'words_tog':
            if (word_tog % 2) == 0:
                print("Words display toggled")
                text_disp = 'word'
            else:
                text_disp = None
    if nearest_size != 0 :
        order  = ["ip_0", "ip_1", "ip_2", "res","near_ip_0", "near_ip_1", "near_ip_2", "near_res"]
    else:
        order  = ["ip_0", "ip_1", "ip_2", "res"]
```

```
    #accomodate edge cases where input ex "near_ip_0" does not exist etc.
    pop_list_flag = 1
    while pop_list_flag == 1: #To iteratively remove undefined inputs
        try:
            df = tsne_plot_df.set_index('type') #set as index so that order can be specified
            df_ordered = df.T[order].T.reset_index()
        except KeyError as e:
            print("Exception in df_ordered dataframe: ",e)
            print("Entering while loop to remove items from order list")
            print("Initial order is: ", order)
            s=str(e)
            rem_val=s[s.find("["):s.find("]")+1]
            rem_val = ast.literal_eval(rem_val)
            order.remove(rem_val[0])
            print("order is now: ", order)
        except UnboundLocalError as e:
            print(e)
            pop_list_flag =0 #Else it continues in an endless while loop
            break
        else: #if try is succesful, end the while loop.s
            pop_list_flag =0
    try:    #if fig_1 is in the main program, "referenced before assignment" error.
        fig_2 = px.scatter(df_ordered, x="x", y="y", text= text_disp, color='type',
                           color_discrete_map={
                                "ip_0": "#7bc043", #green- most natural word by essence "concept1"
                                "ip_1": "#fdf498", #yellow- noticeable contrast from "concept1"; "
                                "ip_2": "#f37736", #orange- "concept2b"lighter contrast from "con
                                "res": "#ee4035", #red- attracts first attention; strange that it
                                "near_ip_0": "#339900",
                                "near_ip_1": "#ffcc00",
                                "near_ip_2": "#ff9966",
                                "near_res": "#cc3300"},
                           log_x=False, hover_name = 'word')
    except UnboundLocalError as e:
        print (e)
        try:
            fig_2 = px.scatter(df_ordered, x="x", y="y", text= 'word', color='type',
                               color_discrete_map={
                                    "ip_0": "#7bc043", #green- most natural word by essence "conce
                                    "ip_1": "#fdf498", #yellow- noticeable contrast from "concept
                                    "ip_2": "#f37736", #orange- "concept2b"lighter contrast from
                                    "res": "#ee4035", #red- attracts first attention; strange tha
                                    "near_ip_0": "#339900",
                                    "near_ip_1": "#ffcc00",
                                    "near_ip_2": "#ff9966",
                                    "near_res": "#cc3300"},
                               hover_name = 'word', log_x=False)
        except UnboundLocalError as e:
            print(e)
            pass
    try:
        fig_2.update_traces(textposition='top center')
        fig_2.update_layout(legend_traceorder="normal")
        fig_2.update_layout(margin=dict(l=0, r=0), uirevision = True)
        return (fig_2), res_ten_in[0]
    except Exception as e:
        print("No return values. Error: ",e)
        raise dash.exceptions.PreventUpdate
        pass

################################ Arbitrary_graphing_inputbox ################################
@app.callback(
    [Output(component_id='input_1', component_property='placeholder'),
     Output(component_id='input_1', component_property='value')],
    [Input('add1_button','n_clicks'),
     Input('rem1_button','n_clicks'),
     Input('clr1_button','n_clicks')],
    [State(component_id='input_1', component_property='value')],
    prevent_initial_call=False
    )

def vocab_list(add,rem,clr,inp_1):
    ctx = dash.callback_context
    if ctx.triggered:
        print(ctx.triggered)
        trigger = (ctx.triggered[0]['prop_id'].split('.')[0])
        if trigger == 'add1_button':
            if ',' in inp_1: #if typing a list directly split into its words
                inp_1=inp_1.replace(" ", "") #remove whitespace
                split_words=inp_1.split(',')
                print("Input words added are: ",split_words)
                vocab_plot_list.extend(split_words)
                value=''
                return vocab_plot_list, value
            else:
                print("Input word added: ",inp_1)
```

```
                vocab_plot_list.append(inp_1)
                print("vocab_list is: ",vocab_plot_list )
                value=''
                return vocab_plot_list, value
        if trigger == 'rem1_button':
            print("Last word removed")
            vocab_plot_list.pop()
            print("vocab_list is now: ",vocab_plot_list)
            value=''
            return vocab_plot_list, value
        if trigger == 'clr1_button':
            print("Clr button clicked")
            vocab_plot_list.clear()
            print("vocab_list is now: ",vocab_plot_list )
            value=''
            return vocab_plot_list, value

    else:
        vocab_plot_list.append(inp_1)
        value=''
        return vocab_plot_list, value

################################ Arb_Word2vec_1 graph ################################
@app.callback(
    Output(component_id='word2vec_1', component_property='figure'),
    [Input('add1_button','n_clicks'),
     Input('rem1_button','n_clicks'),
     Input('clr1_button','n_clicks'),
     Input('plt1_button','n_clicks'),
     Input('my_slider','value'),
     Input('words_tog','n_clicks'),
     Input('input_1', 'value')], #debounce of input box triggers callback of figure
    prevent_initial_call=False
    )

def svd_user_inputs(ad,rem,clr,plot_butt,slider_val,word_tog,ip_tog):

    vocab_to_plot = vocab_plot_list #initialised at the start of program
    tsne_df_full = tsne_300d_w5_df
    print("Slider val: ",slider_val)
    nearest_size = slider_val

    tsne_df_full['type'] = 'Sel'
    ip_vocab_index=list(tsne_df_full[tsne_df_full['word'].isin(vocab_to_plot)].index.values)
    print("vocab to plot of slider graph is: ",vocab_to_plot )

    clos_ten_out = [] #index of closest 10 words
    for word in vocab_to_plot:
        try:
            a=w2v.most_similar(word, topn = nearest_size)
            clos_ten_in = [i[0] for i in a]
            clos_ten_out.append(clos_ten_in)
        except ValueError as e:#Empty word None input
            print(e)
            pass
        except KeyError as e: #Word not in vocab
            print(e)
            pass
    print("close_ten_words are: ",clos_ten_out)
    clos_ten_flat = [item for sublist in clos_ten_out for item in sublist]
    clos_ten_index = list(tsne_df_full[tsne_df_full['word'].isin(clos_ten_flat)].index.values)

    plot_index = ip_vocab_index + clos_ten_index
    tsne_df_full.loc[ip_vocab_index,'type'] = 'ip' # indices = [0,1,3,6,10,15]
    tsne_df_full.loc[clos_ten_index,'type'] = 'near'
    tsne_plot_df = tsne_df_full.loc[plot_index]

    ctx = dash.callback_context
    if ctx.triggered:
        trigger = (ctx.triggered[0]['prop_id'].split('.')[0])
        print("trigger is: ", trigger)
        print("word_tog nclick value is: ", word_tog)
        if trigger == 'words_tog':
            if (word_tog % 2) == 0:
                print("Words display toggled")
                text_disp = 'word'
            else:
                text_disp = None
    try:    #if fig_1 is in the main program, "referenced before assignment" error.
        fig_1 = px.scatter(tsne_plot_df, x="x", y="y", text= text_disp, color='type',
                           log_x=False, hover_name = 'word')
    except UnboundLocalError as e:
        print (e)
        fig_1 = px.scatter(tsne_plot_df, x="x", y="y", text= 'word', color='type',
                           hover_name = 'word', log_x=False)
        pass
```

```
1933
1934        fig_1.update_traces(textposition='top center')
1935        fig_1.update_layout(margin=dict(l=0, r=0), uirevision = True)
1936        return (fig_1)
1937
1938    ############################### Select all button ##############################
1939    @app.callback(
1940        [Output('datatable_id', 'selected_rows')], #references ordered 0 to n index of larger table ir
1941        [Input('sel-button', 'n_clicks'),
1942        Input('desel-button', 'n_clicks')],
1943        [State('datatable_id', 'derived_virtual_selected_rows'), #virtual selected row is what is sele
1944         State('datatable_id', 'derived_virtual_data')], #virtual_data is displayed table. Even after
1945        prevent_initial_call=True
1946    )
1947
1948    def select_deselect(selbtn, deselbtn, selected_rows,filtered_table):
1949        ctx = dash.callback_context
1950        if ctx.triggered:
1951            print(ctx.triggered)
1952            trigger = (ctx.triggered[0]['prop_id'].split('.')[0])
1953            if trigger == 'sel-button':
1954                print("\n\nSelect button clicked")
1955                print("Length of Selected_rows is: ", len(selected_rows))
1956                wc_list= []
1957                wc_list=[[row['ix'] for row in filtered_table]]
1958                wc_list = [[int(i) for i in wc_list[0]]] #convert to int for index reference
1959                print("Displayed table has:", len(filtered_table), 'rows')
1960                print("Wordcloud list contains:", len(wc_list[0]), "elements")
1961                return wc_list
1962            else:
1963                print("\n\nDeselect button clicked")
1964                print("Length of Selected_rows is: ", len(selected_rows))
1965                return [[]]
1966
1967    ############################### Wordcloud callback ##############################
1968    @app.callback(
1969        Output('wordcloud','figure'),
1970        [Input(component_id='datatable_id',component_property='selected_rows'),
1971        Input(component_id='my-dropdown', component_property='value')],
1972        prevent_initial_call=False
1973    )
1974
1975    def ren_wordcloud(chosen_rows, chosen_cols):
1976        global wc_init, fig_wordcloud1 #global for pre-app saving of the object
1977        if len(chosen_cols) > 0: #atleast 1 col to be selected
1978            if len(chosen_rows)==0:
1979                df_filtered = df_disp_1[chosen_cols] #if no rows selected consider all rows
1980                if wc_init == 1: #load once.
1981                    return fig_wordcloud1
1982                else:
1983                    wc_init = 1 #object initialised flag set
1984
1985                    print("Full Word cloud not initialised")
1986                    print("Initialising full word cloud..")
1987                    df_filtered['comb_cols'] = df_filtered[df_filtered.columns[0:]].apply( #Combine mu
1988                        lambda x: ' '.join(x.dropna().astype(str)),
1989                        axis=1)
1990
1991                    en_stopwords = stopwords.words('english')
1992                    fr_stopwords = stopwords.words('french')
1993                    web_links_sw = ['www','http','https','com']
1994                    combined_stopwords = en_stopwords + fr_stopwords + web_links_sw
1995                    print("\nNo of rows in WC are: ",len(df_filtered))
1996                    wordcloud = WordCloud(max_words=100,
1997                                          # stopwords= combined_stopwords,
1998                                          colormap='tab20c',
1999                                          background_color='white',
2000                                          width=1700, #1200,1700
2001                                          height=1000, #1000
2002                                          random_state=1).generate(' '.join(df_filtered['comb_cols']))
2003
2004                    fig_wordcloud1 = px.imshow(wordcloud, template='ggplot2')#title="Word frequency"
2005                    fig_wordcloud1.update_layout(margin=dict(l=0, r=0,b=0,t=0))
2006                    fig_wordcloud1.update_xaxes(visible=False)
2007                    fig_wordcloud1.update_yaxes(visible=False)
2008                    return fig_wordcloud1
2009            elif len(chosen_rows) > 0 :
2010                df_filtered = df_disp_1[chosen_cols]
2011                df_filtered = df_filtered[df_filtered.index.isin(chosen_rows)]
2012        elif len(chosen_cols) == 0:
2013            raise dash.exceptions.PreventUpdate
2014    ## load wordcloud only once
2015        df_filtered['comb_cols'] = df_filtered[df_filtered.columns[0:]].apply( #Combine multiple colum
2016            lambda x: ' '.join(x.dropna().astype(str)),
2017            axis=1)
2018
```

```
    en_stopwords = stopwords.words('english')
    fr_stopwords = stopwords.words('french')
    web_links_sw = ['www','http','https','com']
    combined_stopwords = en_stopwords + fr_stopwords + web_links_sw
    print("\nNo of rows in WC are: ",len(df_filtered))
    wordcloud = WordCloud(max_words=100,
                          # stopwords= combined_stopwords,
                          colormap='tab20c',
                          background_color='white',
                          width=1700, #1200,1700
                          height=1000, #1000
                          random_state=1).generate(' '.join(df_filtered['comb_cols'])) #df_filtere
    fig_wordcloud = px.imshow(wordcloud, template='ggplot2')#title="Word frequency"
    fig_wordcloud.update_layout(margin=dict(l=0, r=0,b=0,t=0))
    fig_wordcloud.update_xaxes(visible=False)
    fig_wordcloud.update_yaxes(visible=False)
    return fig_wordcloud

############################### Column highlighting ##############################
@app.callback(
    Output('datatable_id', 'style_data_conditional'),
    Input(component_id='my-dropdown', component_property='value')
)

def update_styles(selected_columns):
    return [{
        'if': { 'column_id': i },
        'background_color': '#D2F3FF'
    } for i in selected_columns]

if __name__ == '__main__':
    app.run_server(debug=False)

```

Summary – English and French

Keywords: NLP, dashboard, wordembeddings, socialmedia, instagram, word2vec

Paying attention to language use is very useful in understanding what people think about a certain subject. The way a Particular word is used in different contexts, defines the meaning of the word. As such, the meaning of a word evolves over time and new words may often come into existence. In this project we build an interactive tool which uses natural language processing (NLP) techniques, to analyze text and infer the current meaning of the word 'organic' as applied to different vocations, from Instagram post descriptions. When a person or company uses a particular hashtag for a post in social media (ex. #organic), it is implied that the text descriptions they write for the posts, must be related to this hashtag. We use visualizations of word frequency counts, co-occurrences, and word embeddings from the word2vec skip gram method, to gather insights about the subject.

As for the insights, the premise is that the word 'organic' assumes different meanings in different contexts. For example, the meaning of 'organic' in the context of textiles, food, and cosmetics, at the highest level, carries connotations of 'health', 'certifications', and 'unadulterated' respectively. This interactive dashboard tool can be useful for companies and individuals trying to understand how the meaning of a particular word/concept is being perceived by their target audience and inform their marketing campaigns. An approach for using the tool is described in detail, using "organic food" as an example context of study.

Résumé: NLP, tableaudebord, wordembeddings, socialmedia, Instagram, word2vec

Prêter attention à l'utilisation de la langue est très utile pour comprendre ce que les gens pensent d'un certain sujet. La façon dont un mot particulier est utilisé dans différents contextes définit le sens du mot. Ainsi, le sens d'un mot évolue avec le temps et de nouveaux mots peuvent souvent voir le jour. Dans ce projet, nous construisons un outil interactif qui utilise des techniques de traitement du langage naturel (PNL) pour analyser le texte et déduire la signification actuelle du mot « biologique » tel qu'il est appliqué à différentes vocations, à partir des descriptions de publications Instagram. Lorsqu'une personne ou une entreprise utilise un hashtag particulier pour une publication sur les réseaux sociaux (par exemple, #organique), il est sous-entendu que les descriptions textuelles qu'elle rédige pour les publications doivent être liées à ce hashtag. Nous utilisons des visualisations du nombre de fréquences de mots, des cooccurrences et des incorporations de mots de la méthode word2vec skip gram, pour recueillir des informations sur le sujet. En ce qui concerne les idées, la prémisse est que le mot « biologique » prend différentes significations dans différents contextes. Par exemple, la signification de « biologique » dans le contexte des textiles, des aliments et des cosmétiques, au plus haut niveau, a des connotations de « santé », de « certifications » et de « pur » respectivement. Une approche d'utilisation de l'outil est décrite en détail, en utilisant « l'alimentation biologique » comme exemple de contexte d'étude.