Data Science at SAMY

Table of contents

1 Welcome to the Data Science Handbook

Welcome to our Data Science Handbook, your comprehensive guide to the methods, case studies, and best practices that define our approach to data science here at SAMY. This handbook is designed to be a dynamic resource for our team, evolving with new insights, tools, and technologies.

Within these pages, you'll find detailed case studies showcasing our successful projects, high-level concepts that underpin our strategies, and practical coding examples to help you apply these techniques in your own work. Irrespective of your experience in data science, this handbook aims to provide valuable insights and practical guidance to enhance your skills and knowledge.

We believe that sharing knowledge and continuously learning are key to staying ahead in the fast-paced world of data science. As such, this handbook is not just a static document but a collaborative space where ideas are exchanged, and innovation thrives.

Happy data sciencing!

> **Note**
>
> If you have any questions at all, ask any member of the team. Whilst this Handbook aims to be a valuable resource for self-learning, it can often be more beneficial to spend 5 minutes talking through a concept with someone on the team who may be able to describe something in a different manner to this document.

Part I

Introduction

2 The Data Science team

2.1 Where we sit

The Data Science department are a fully global resource within the alliance

Capture Intelligence is our biggest internal "client" as there are plenty of opportunities to offer data science led services in the research offering of Capture. But also have our own core central pipe for development that supports *all* agency brands.

What this means is as a team we have responsibilities that range from continual development of our own tech stack to help answer specific research questions for external clients to helping empower members of the alliance to use mindful applications of emerging technologies.

2.2 Who we are

Mike Tapp: Data Director

Jack Penzer: Global Data Product Lead

Jamie Hudson: Senior Data Scientist

Aoife Ryan: Data Scientist

Sophie Thomas: Jr. Data Scientist

Part II

Case Studies

3 Peaks and Pits

"Peaks and Pits" is one of our fundamental project offerings and a workflow that is a solid representation of good data science work that we perform.

3.1 What is the concept/project background?

Strong memories associated to brands or products go deeper than simple positive or negative sentiment. Most of our experiences are not encoded in memory, rather what we remember about experiences are changes, significant moments, and endings. In their book "The Power of Moments", two psychologists (Chip and Dan Heath) define these core memories as Peak and Pits, impactful experiences in our lives.

Broadly, peak moments are experiences that stand our memorable in our lives in a positive sense, whereas pit moments are impactful negative experiences.

Microsoft tasked us with finding a way to identify these moments in social data- going beyond 'simple' positive and negative sentiment which does not tell the full story of consumer/user experience. The end goal is that by providing Microsoft with these peak and pit moments in the customer experience, they can design peak moments in addition to simply removing pit moments.

3.1.1 The end goal

With these projects the core final 'product' is a collection of different peaks and pits, with suitable representative verbatims and an explanation to understand the high-level intricacies of these different emotional moments.

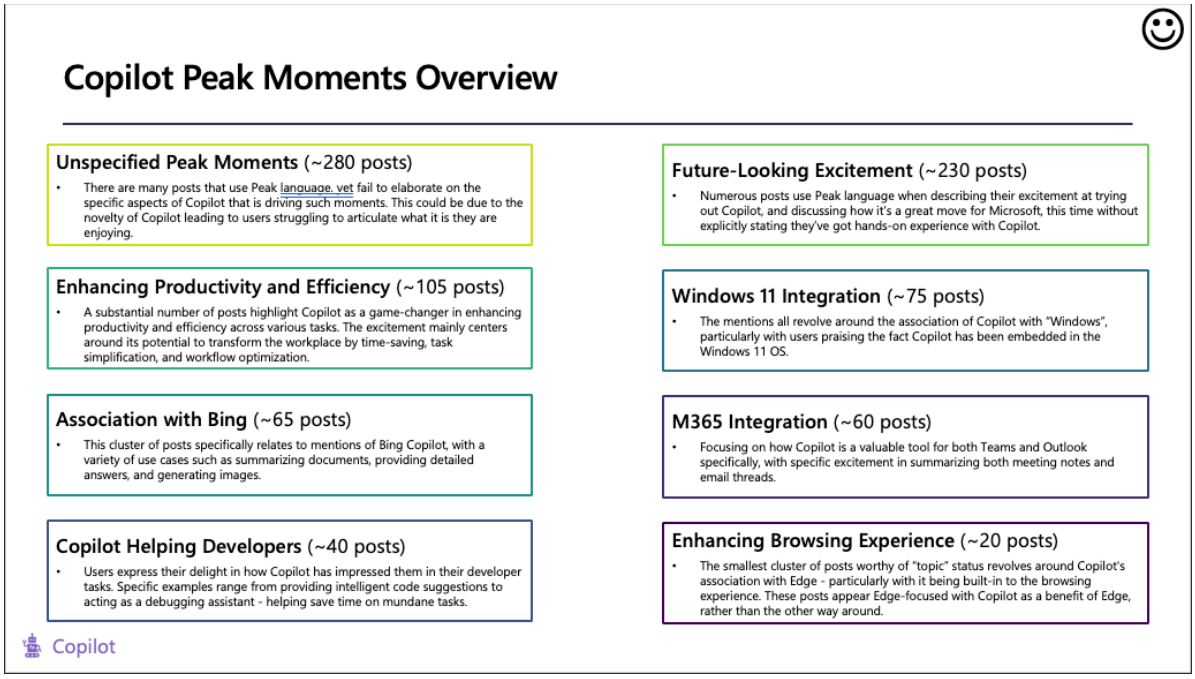


Figure 3.1: Screenshot from a Peaks and Pits project showcasing the identified Peak moments for a product at a high level

3.1.2 Key features of project

- There is no out-of-the-box ML model available whose purpose is to classify social media posts as either peaks or pits (i.e. we cannot use a ready-made solution, we must design our own bespoke solution).
- There is limited data available
  - Unlike the case of spam/ham or sentiment classification, there is not a bank of pre-labelled data available for us to leverage for 'traditional ML'.
- Despite these issues, the research problem itself is well defined (**what** are the core peak and pit moments for a brand/product), and because there are only three classes (peak, pit, or neither) which are based on extensive research, the classes themselves are well described (even if it is case of "you know a peak moment when you see it").

3.2 Overview of approach

Peaks and pits projects have gone through many iterations throughout the past year and a half. Currently, the general workflow is to use utilise a model framework known as SetFit to

efficiently train a text classification model with limited training data. This fine-tuned model is then able to run inference over large datasets to label posts as either peaks, pits, or neither. We then utilise the LLM capabilities to refine these peak and pit moments into a collection of posts we are extremely confident are peaks and/or pits. We then employ topic modelling to identify groups of similar peaks and pits to help us organise and discover hidden topics or themes within this collection of core moments.

This whole process can be split into six distinct steps:

1. Extract brand/product mentions from Sprinklr (the start of any project)
2. Obtain project-specific exemplar posts to help fine-tune a text classification model
3. Perform model fine-tuning through contrastive learning
4. Run inference over all of the project specific data
5. Use GPT-3.5 for an extra layer of classification on identified peaks and pits
6. Turn moments into something interpretable using topic modelling

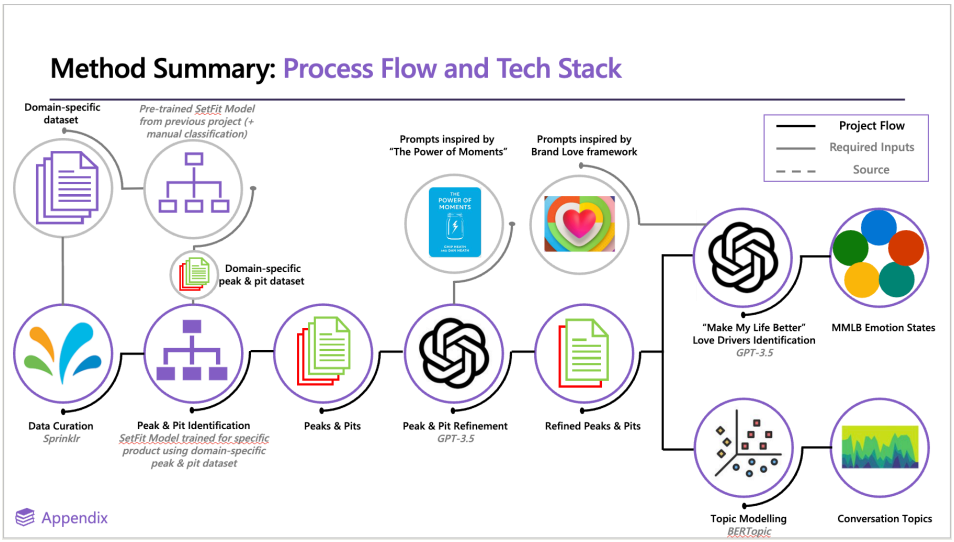


Figure 3.2: Schematic workflow from Project 706 - Peaks and Pits in M365 Apps

3.2.1 Obtain posts for the project (Step 1)

This step relies on the analysts to export relevant mentions from Sprinklr (one of the social listening tools that analysts utilise to obtain social data), and therefore is not detailed much here. What is required is one dataset for each of the brands/products, so they can be analysed separately.

3.2.2 Identify project-specific exemplar peaks and pits to fine-tune our ML model (Step 2)

This step is synonymous with data labelling required for any machine learning project where annotated data is not already available.

Whilst there is no one-size-fits-all for determining the amount of data required to train a machine learning model, for traditional models and tasks, the number of examples per label is often in the region of thousands, and often this isn't even enough for more complex problems.

The time required to accurately label thousands of peaks and pits to train a classification model in the traditional way is sadly beyond the scope of feasibility for our projects. As such we needed an approach that did not rely on copious amounts of pre-labelled data.

This is where SetFit comes in. As mentioned previously, SetFit is a framework that enables us to efficiently train a text classification model with limited training data.

> How does it do this?
>
> *Note the below is directly copied from the SetFit documentation. It is so succinctly written that trying to rewrite it would not do it justice.*
> Every SetFit model consists of two parts: a **sentence transformer** embedding model (the body) and a **classifier** (the head). These two parts are trained in two separate phases: the **embedding finetuning phase** and the **classifier training phase**. This conceptual guide will elaborate on the intuition between these phases, and why SetFit works so well.
> **Embedding finetuning phase**
> The first phase has one primary goal: finetune a sentence transformer embedding model to produce useful embeddings for our classification task. The Hugging Face Hub already has thousands of sentence transformer available, many of which have been trained to very accurately group the embeddings of texts with similar semantic meaning.
> However, models that are good at Semantic Textual Similarity (STS) are not necessarily immediately good at our classification task. For example, according to an embedding model, the sentence of 1) "`He biked to work.`" will be much more similar to 2) "`He drove his car to work.`" than to 3) "`Peter decided to take the bicycle to the beach party!`". But if our classification task involves classifying texts into transportation modes, then we want our embedding model to place sentences 1 and 3 closely together, and 2 further away.
> To do so, we can finetune the chosen sentence transformer embedding model. The goal here is to nudge the model to use its pretrained knowledge in a different way that better aligns with our classification task, rather than making the completely forget what it has learned.
> For finetuning, SetFit uses **contrastive learning**. This training approach involves

creating **positive and negative pairs** of sentences. A sentence pair will be positive if both of the sentences are of the same class, and negative otherwise. For example, in the case of binary "positive"-"negative" sentiment analysis, ("`The movie was awesome`", "`I loved it`") is a positive pair, and ("`The movie was awesome`", "`It was quite disappointing`") is a negative pair.
During training, the embedding model receives these pairs, and will convert the sentences to embeddings. If the pair is positive, then it will pull on the model weights such that the text embeddings will be more similar, and vice versa for a negative pair. Through this approach, sentences with the same label will be embedded more similarly, and sentences with different labels less similarly.
Conveniently, this contrastive learning works with pairs rather than individual samples, and we can create plenty of unique pairs from just a few samples. For example, given 8 positive sentences and 8 negative sentences, we can create 28 positive pairs and 64 negative pairs for 92 unique training pairs. This grows exponentially to the number of sentences and classes, and that is why SetFit can train with just a few examples and still correctly finetune the sentence transformer embedding model. However, we should still be wary of overfitting.
Classifier training phase
Once the sentence transformer embedding model has been finetuned for our task at hand, we can start training the classifier. This phase has one primary goal: create a good mapping from the sentence transformer embeddings to the classes.
Unlike with the first phase, training the classifier is done from scratch and using the labelled samples directly, rather than using pairs. By default, the classifier is a simple **logistic regression** classifier from scikit-learn. First, all training sentences are fed through the now-finetuned sentence transformer embedding model, and then the sentence embeddings and labels are used to fit the logistic regression classifier. The result is a strong and efficient classifier.
Using these two parts, SetFit models are efficient, performant and easy to train, even on CPU-only devices.

There is no perfect number of labelled examples to find per class (i.e. peak, pit, or neither). Whilst in general more exemplars (and hence more training data) is beneficial, having fewer but high quality labelled posts is far superior than more posts of poorer quality. This is extremely important due to the contrastive nature of SetFit where it's superpower is making the most of few, extremely good, labelled data.

Okay so now we know why we need labelled data, and we know what the model will do with it, *what is our approach* for obtaining the labelled data?

Broadly, we use our human judgement to read a post from the current project dataset, and manually label whether we think it is a peak, a pit, or neither. To avoid us just blindly reading through random posts in the dataset in the hope of finding good examples (this is not a good use of time), we can employ a couple of tricks to narrow our search region to posts that have

a reasonable likelihood of being suitable examples.

1) The first trick is to use the OpenAI API to access a GPT model. This involves taking a sample of posts (say ~1000) and running these through a GPT model, with a prompt that asks the model to classify each post into either a peak, pit, or neither. This is possible because GPT models have learned knowledge of peaks and pits from its training on large datasets. We can therefore get a human to only sense-check/label posts that GPT also believes are peaks or pits.
2) The second trick involves using a previously created SetFit model (i.e. from an older project), and running inference over a similar sample of posts (again, say ~1000).

We would tend to suggest the OpenAI route, as it is simpler to implement (in our opinion), and often the old SetFit model has been finetuned on data from a different domain so it might struggle to understand domain specific language in the current dataset. However, be aware if it not as scalable as using an old SetFit model and has the drawback of being a prompt based classification of a black-box model (as well as issues relating to cost and API stability).

Irrespective of which approach is taken, by the end of this step we need to have a list of example posts we are confident represent what a peak or pit moment looks like for each particular product we are researching, including posts that are "neither".

> **i** Why do we do this for each project?
>
> After so many projects now don't we already have a good idea of what a peak and pit moment for the purposes of model training?
> Each peak and pit project we work on has the potential to introduce 'domain' specific language, which a machine learning classifier (model) may not have seen before. By manually identifying exemplar peaks and pits that are project-specific, this gives our model the best chance to identify emotional moments appropriate to the project/data at hand.
> The obvious case for this is with gaming specific language, where terms that don't necessarily relate to an 'obvious' peak or pit moment could refer to one the gaming conversation, for example the terms/phrases "GG", "camping", "scrub", and "goat" all have very specific meanings in this domain that differ from their use in everyday language.

3.2.3 Train our model using our labelled examples (Step 3)

Before we begin training our SetFit model with our data, it's necessary to clean and wrangle the fine-tuning datasets. Specifically, we need to mask any mentions of brands or products to prevent bias. For instance, if a certain brand frequently appears in the training data within peak contexts, the model could erroneously link peak moments to that brand rather than learning the peak-language expressed in the text.

This precaution should extend to all aspects of our training data that might introduce biases. For example, as we now have examples from various projects, an overrepresentation of data from 'gaming projects' in our 'peak' posts within our training set (as opposed to the 'pit' posts) could skew the model into associating gaming-related language more with peaks than pits.

Broadly the cleaning steps that should be applied to our data for finetuning are:

- Mask brand/product mentions
- Remove hashtags #
- Remove mentions
- Remove URLs
- Remove emojis

> **What about other cleaning steps?**
>
> You will notice here we do not remove stop words-. As explained in the previous step, part of the finetuning process is to finetune a sentence embedding model, and we want to keep stop words so that we can use the full context of the post in order to finetune accurate embeddings.

At this step, we can split out our data into training, testing, and validation datasets. A good rule of thumb is to split the data 70% to training data, 15% to testing data, and 15% to validation data. By default, SetFit oversamples the minimum class within the training data, so we *shouldn't* have to worry too much about imbalanced datasets- though be aware if we have extreme imbalanced we will end up sampling the same contrastive pairs (normally positive pairs) multiple times. However, our experimentation has shown that class imbalance doesn't seem to have a significant effect to the training/output of the SetFit model for peaks and pits.

We are now at the stage where we can actually fine-tune the model. There are many different parameters we can change when fine-tuning the model, such as the specific embedding model used, the number of epochs to train for, the number of contrastive pairs of sentences to train on etc. For more details, please refer to the Peaks and Pits Playbook

We can assess model performance on the testing dataset by looking at accuracy, precision, recall, and F1 scores. For peaks and pits, the most important metric is actually **recall** because in step 4 we reclassify posts using GPT, so we want to make sure we are able to provide *as many true peak/pit moments as possible* to this step, even if it means we also provide a few false positives.

Click here for more info as to why recall is most important

As a refresher, `precision` is the *proportion of positive identifications* that are actually *correct* (it focuses on the correctness of positive predictions) whereas `recall` is the *proportion of actual positives* that are identified correctly (it focuses on capturing all relevant instances).
In cases where false positives need to be minimised (incorrectly predicting a non-event as an event) we need to prioritise `precision` - if you've built a model to identify hot dogs from regular ol' doggos, high precision ensures that normal dogs are not misclassified as hot dogs.
In cases where false negatives need to be minimised (failing to detect an actual event) we need to prioritise `recall` - in medical diagnoses we need to minimise the number of times a patient is incorrectly told they *do not* have a disease when in reality they *do* (or worded differently, we need to ensure that ***all*** patients with a disease are identified).
To apply this to our problem- we want to be sure that we capture all (or as many as possible) relevant instances of peaks or pits- even if a few false positives come in (neither posts that are incorrectly classified as peaks or pits). As we use GPT to make further peak/pit identifications, it's better to provide GPT with with a comprehensive set of potential peaks and pits, including some incorrect ones, than to miss out on critical data.

Visualise model separation

As a bonus, we can actually neatly visualise how well our finetuning of the sentence transformer embedding model has gone- by seeing how well the model is able to separate our different classes in embedding space.

We can do this by visualising the 2-D structure of the embeddings and see how they cluster:

This is what it looks the embeddings space looks like on an un-finetuned model:

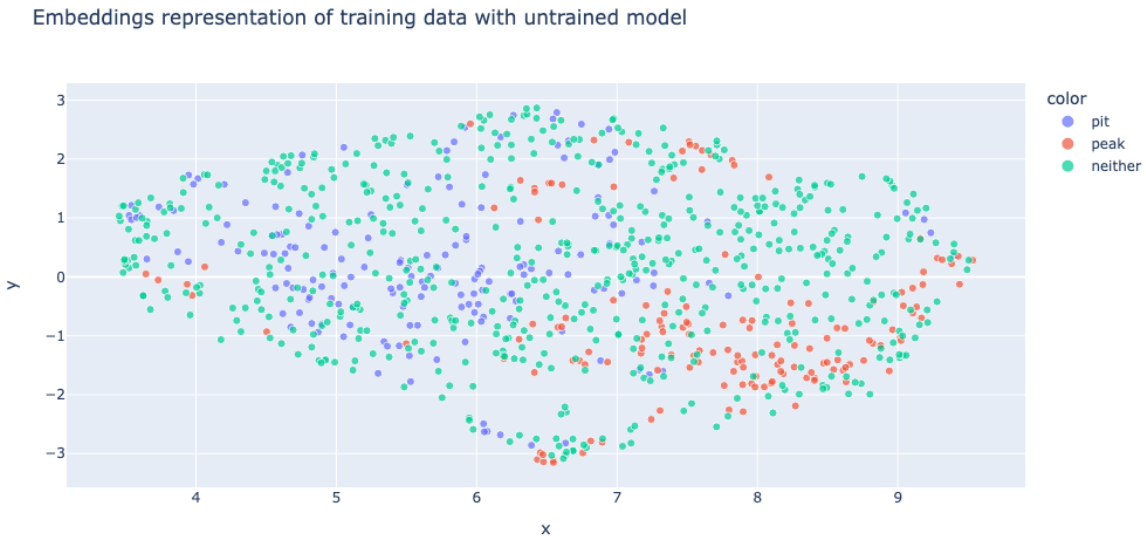


Figure 3.3: Un-finetuned embedding model

Here we can see that posts we know are peaks, pits, and neither all overlap and there is no real structure in the data. Any clustering of points observed are probably due to the posts' semantic similarity (c.f. the mode of transport example above). We would not be able to nicely use a classifier model to get a good mapping from this embedding space to our classes (i.e. we couldn't easily separate classes here).

By visualising the same posts after finetuning the embedding model, we get something more like this, where we can see that the embedding model now clearly separates posts based on their peak/pit classification (though we must be wary of overfitting!).

Figure 3.4: Finetuned embedding model

Finally, now we are happy with our model performance based on the training and validation datasets, we can evaluate the performance of this final model using our testing data. This is data that the model has never seen, and we are hoping that the accuracy and performance is similar to that of the validation data. This is Machine Learning 101 and if a refresher is needed for this there are plenty of resources online looking at the role of training, validation, and testing data.

3.2.4 Run inference over project data (Step 4)

It is finally time to infer whether the project data contain peaks or pits by using our fine-tuned SetFit model to classify the posts.

Before doing this again we need to make sure we do some data cleaning on the project specific data.

Broadly, this needs to match the high-level cleaning we did during fine-tuning stage:

- Mask brand/product mentions (using RoBERTa-based model [or similar] and `Rivendell` functions)
- Remove hashtags #
- Remove mentions
- Remove URLs
- Remove emojis

> ⚠ Note on social media sources
>
> Currently all peak and pit projects have been done on Twitter or Reddit data, but if a project includes web/forum data quirky special characters, numbered usernames, structured quotes etc. should also be removed.

Okay now we can *finally* run inference. This is extremely simple and only requires a couple of lines of code (again see the Peaks and Pits Playbook for code implementation)

3.2.5 The metal detector, GPT-3.5 (Step 5)

During step 4 we obtained peak and pit classification using few-shot classification with SetFit. The benefit of this approach (as outlined previously) is its speed and ability to classify with very few labelled samples due to contrastive learning.

However, during our iterations of peak and pit projects, we've realised that this step still classifies a fair amount of non-peak and pit posts incorrectly. This can cause noise in the downstream analyses and be very time consuming for us to further trudge through verbatims.

As such, the aim of this step is to further our confidence in our final list of peaks and pits to be *actually* peaks and pits. Remember before we explained that for SetFit, we focussed on **recall** being the most important measure in our business case? This is where we assume that GPT-3.5 enables us to remove the false positives due to it's incredibly high performance.

> ❗ Why not use GPT from the start?
>
> Using GPT-3.5 for inference, even over relatively few posts as in peaks and pits, is expensive both in terms of time and money. Preliminary tests have suggested it is in the order of magnitude of thousands of times slower than SetFit. It is for these reasons why we do not use GPT-x models from the get go, despite it's obvious incredible understanding of natural language.

Whilst prompt-based classification such as those with GPT-3.5 certainly has its drawbacks (dependency on prompt quality, prompt injections in posts, handling and version control of complex prompts, unexpected updates to the model weights rendering prompts ineffective), the benefits include increased flexibility in what we can ask the model to do. As such, in the absence of an accurate, cheap, and quick model to perform span detection, we have found that often posts identified as peaks/pits did indeed use peak/pit language, but the context of the moment was not related to the brand/product at the core of the research project.

For example, take the post that we identified in the project 706, looking for peaks and pits relating to PowerPoint:

> This brings me so much happiness! Being a non-binary graduate student in STEM academia can be challenging at times. Despite using my they/them pronouns during introductions, emails, powerpoint presentations, name tags, etc. my identity is continuously mistaken. Community is key!

This is clearly a 'peak', however it is not accurate or valid to attribute this memorable moment to PowerPoint. Indeed, PowerPoint is merely mentioned in the post, but is not a core driver of the Peak which relates to feeling connection and being part of a community. This is as much a PowerPoint Peak as it is a Peak for the use of emails.

Therefore, we can engineer our prompt to include a caveat to say that the specific peak or pit moment must relate directly to the brand/product usage (if relevant).

3.2.6 Topic modelling to make sense of our data (Step 6)

Now we have an extremely refined set of posts classified as either peak or pits. The next step is to identify what these moments actually relate to (i.e. identify the topics of these moments through statistical methods).

To do this, we employ topic modelling via BERTopic to identifying high-level topics that emerge within the peak and pit conversation. This is done separately for each product and peak/pit dataset (i.e. there will be one BERTopic model for product A peaks, another BERTopic model for product A pits, an additional BERTopic model for product B peaks etc).

We implement BERTopic using the R package BertopicR. As there is already good documentation on BertopicR this section will not go into any technical detail in regards to implementation.

From BertopicR. we end up with a topic label for each post in our dataset, meaning we can easily quantify the size of each topics and visualise temporal patterns of topic volume etc.

4 Conversation Landscape

The 'Conversation Landscape' method has proven to be an effective tool for querying, auditing, and analysing both broad concepts and finely grained topics across social conversations on all major platforms, as well as web pages and forums.

4.1 Project Background

Working with semi-structured or unstructured high-dimensional data, such as text (and in our case, social media posts), poses significant challenges in measuring or quantifying the language used to describe any specific phenomena. One common approach to quantifying language is topic modelling, where a corpus (or collection of documents) is processed and later represented in neater and simplified format. This often involves displaying top terms, verbatims, or threads highlighting any nuances or differences within the data. Traditional topic modelling or text analysis methods, such as Latent Dirichlet Allocation (LDA), operate on the probability or likelihood of terms or n-grams belonging to a set number of topics.

The Conversation Landscape workflow offers a slightly different solution and one that partitions text data without a specific need for burdening the user with sifting through rows of data in order to segment documents with hopes of understanding or recognising any differences in language, which would ideally be defined more simply as topics. The is mostly achieved through sentence transforming, where documents are converted from words to numerical values, which are often referred to as 'embeddings'. These values are calculated based on their content's semantic and syntactic properties. The transformed values are then processed again using dimension reduction techniques, making the data more suitable for visualisation. Typically, this involves reducing to two dimensions, though three dimensions may be used to introduce another layer of abstraction between our data points. The example provided throughout this chapter, represents some text data as nodes upon a two-dimensional space.

> **Note**
>
> *This documentation will delve deeper into the core concepts of sentence transforming and dimension reduction, along with the different methods used to cluster or group topics once the overall landscape is mapped out, referring back to our illustrated real-world business use case of these techniques. We will then later look at best practices and any downstream flourishes that will help us operate within this work-stream.*

4.2 Final output of project

An ideal output, like the one shown below should always showcase the positioning of our reduced data points onto the semantic space, along with any topic or subtopic explanations alongside, using color coding where appropriate. While we sometimes provide raw counts of documents per topic/subtopic, we always include the percentage of topic distribution across our data, occasionally referred to as Share of Voice (SOV).

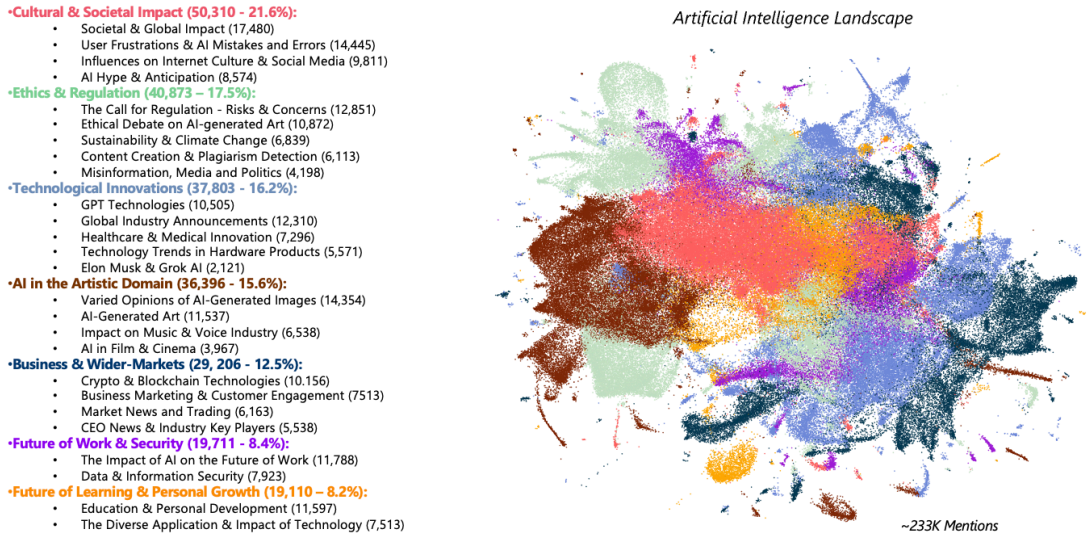


Figure 4.1: Screenshot Taken from the Final Output of an AI Landscape Microsoft Project - Q2 FY24

4.3 How to get there

As promised, we will provide some more context as well as the appropriate information surrounding the required steps taken, so that a reader may replicate and implement the methods mentioned throughout so far, providing an efficient analysis tool to use for any set of documents, regardless of domain specifics. While the example output provided displays a simplified means for visualising complex and multifaceted noisy data such as the 'Artificial Intelligence' conversation on social, there are a number of steps that one must take carefully and be mindful of throughout, in order to create the best fit model appropriate for a typical Conversation Landscape project.

The broad steps would include, and as one might find across many projects within the realms of Natural Language Processing (NLP):

- Initial Exploratory Data Analysis (EDA): Checking that the data is relevant and fit to answer the brief.
- Cleaning and Processing: Removal of spam, unhelpful or irrelevant data, and pre-processing of text variable for embedding.
- Transforming/Embedding: Turning our words into numbers which will later be transformed again before being visualised.
- Dimension Reduction: Reducing our representational values of documents to a manageable state in order to visualise.
- Topic Modelling/Clustering: Scientifically modelling and defining our data into a more digestible format.

4.3.1 Exploratory Data Analysis (EDA):

Whether the user is responsible for data querying/collection or not, the first steps in our workflow should always involve some high-level checks before we proceed with any of the following steps in order to save time downstream and give us confidence to carry over into the data cleaning and processing steps and beyond.

First, one should always check things like the existing variables and clean or rename any where necessary. This step requires a little forward thinking as to what columns are necessary to complete each stage of the project. Once happy with our variables, we can then check for things such as missing dates, and/or if there are any abnormal distributions across columns like 'Social Platform' that might skew any findings or help us understand or perhaps justify the resulting topic model. Next, we can do some bespoke or project specific checks like searching for likely-to-find terms or strings within our text variable to ensure that the data is relevant and query has captured the phenomena we are aiming to model.

4.3.2 Data Cleaning/Processing:

Again, as we may not always be responsible for data collection, we can expect that our data may contain unhelpful or even problematic information which is often the result of data being unwillingly bought in by the query. Our job at this stage is to minimize the amount of unhelpful data existing in our corpus to ensure our findings are accurate as well as appropriate for the data which we will be modelling.

Optimal procedures for spam detection and removal are covered in more detail [here]*will include link when data cleaning section is complete.* However, there are steps the user absolutely must take to ensure that the text variable which will be provided to the sentence transformer model is clean and concise so that an accurate embedding process can take place upon our documents. This includes the removal of:

- Hashtags #
- User/Account Mentions
- URLs or Links
- Emojis
- Non-English Characters

Often, we might also choose to remove punctuation and/or digits, however in our provided example, we have not done so. There are also things to beware of such as documents beginning with numbers that can influence the later processes, so unless we deem them necessary we should remove these where possible to ensure no inappropriate grouping of documents takes place based on these minor similarities. This is because when topic modelling, we aim to capture the pure essence of clusters which is ultimately defined by the underlying semantic meaning of documents, as apposed to any similarities across the chosen format of said documents.

4.3.3 Sentence Transforming/Embedding:

Once we are happy with the cleanliness and relevance of our data, including the processing steps we have taken with our chosen text variable, we can begin embedding our documents so that we have a numerical representation that can later be reduced and visualised for each. Typically, and in this case we have used already pre-trained sentence transformer models that are hosted on Hugging Face, such as `all-mpnet-base-v2` which is the specific model we had decided to use in our AI Conversation Landscape example. This is because during that time, the model had boasted great performance scores for how lightweight it was, however with models such as these being open-source, community-lead contributions are made to further train and improve model performance which means that these performance metrics are always increasing, so one may wish to consult the Hugging Face leaderboard, or simply do some desk research before settling on an ideal model appropriate for their own specific use case.

While the previous steps taken might have involved using R and Rstudio and making use of SHARE's suite of data cleaning, processing and parsing functionality, the embedding process will need to be completed using Google Colab. This is to take advantage of their premium GPUs and high RAM option, as embedding documents can require large amounts of compute, so much so that most fairly competent machines with standard tech specs will struggle. It is also worth noting that an embedding output may depend on the specific GPU being utilized as well as the version of Python that Colab is currently running, it's good practice to make note of both of these specifics, along with other modules and library versions that one may wish to use in the same session, such as `umap-learn` (you may thank yourself at a later stage for doing so). To get going with sentence transformers and for downloading/importing a model such as all-mpnet-bas-v2, there are step-by-step guides purposed to enable users with the know-how to use them and deal with model outputs upon the Hugging Face website.

4.3.4 Dimension Reduction:

At this stage, we would expect to have our data cleaned along with the representative embeddings for each document, which is output by the sentence transforming process. This next step, explains how we take this high-dimensional embeddings object and then simplify/reduce columns down enough to a more manageable size in order to map our documents onto a semantic space. Documents can then be easily represented as a node and are positioned within this abstract space based upon their nature, meaning that those more semantically similar will be situated closer together upon our two (or sometimes three-dimensional) plot, which then forms our landscape.

There are a number of ways the user can process an embeddings output. Each method has its own merits as well as appropriate use cases, which mostly depend whether the user intends to focus on either the local or global structure of their data. For more on the alternative dimension reduction techniques, the BERTopic documentation provides some further detail while staying relevant to the subject matter of Topic Modelling and NLP.

Once we have reduced our embeddings, and for the sake of staying consistent to the context of our given example, lets say we have decided to use Uniform Manifold Approximation and Projection (UMAP), a technique which is helpful for when we wish to represent both the local and global structures of our data. The output of this step should have resulted in taking our high dimensional embedding data (often 768 columns or sometimes more) and reduced these values down to just 2 columns so that we can plot them onto our semantic space (our conversation landscape plot), using these 2 reduced values as if to serve as X and Y coordinates to appropriately map each data point, we often name these two columns `V1` and `V2`.

At this stage, we can use the `LandscapeR` package to render a static visualisation of the entire landscape, and we can select the desired colour of our nodes by making use of the `fill_colour` parameter. In this instance, we've mapped our documents onto the semantic space represented as nodes using columns `V1` and `V2` and coloured them a dark grey.

```
data %>%
  LandscapeR::ls_plot_static(x_var = V1,
                             y_var = V2,
                             fill_colour = "#808080")
```

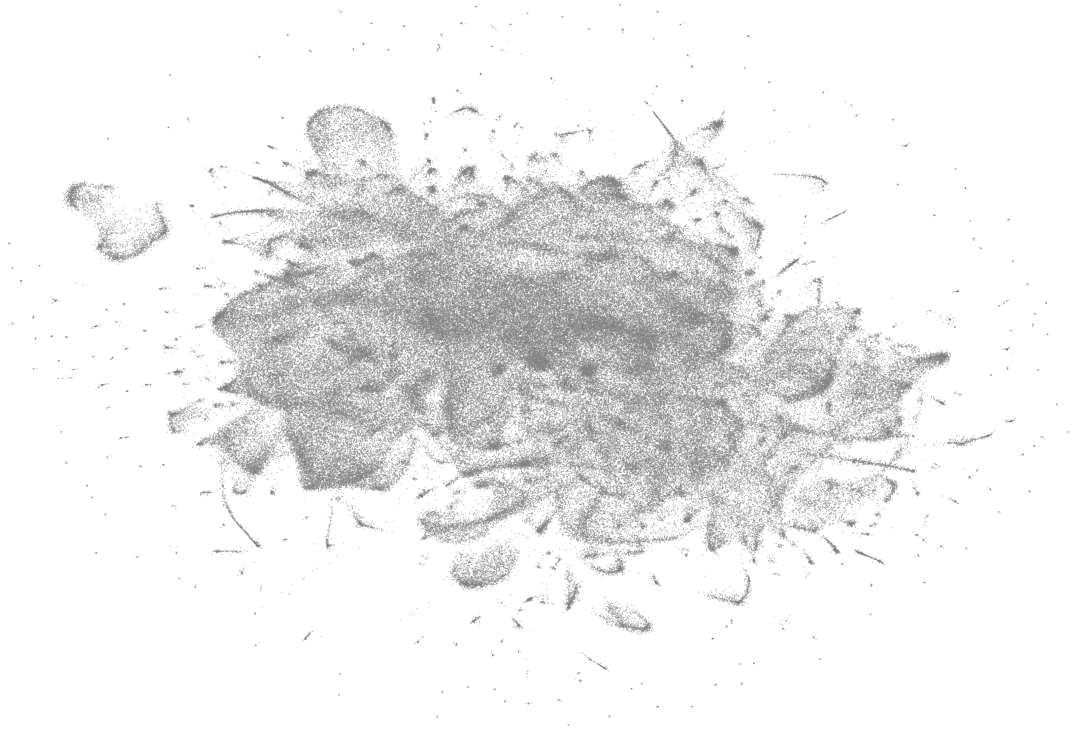


Figure 4.2: Grey Colourless Landscape Plot from an AI Landscape Microsoft Project - Q2 FY24

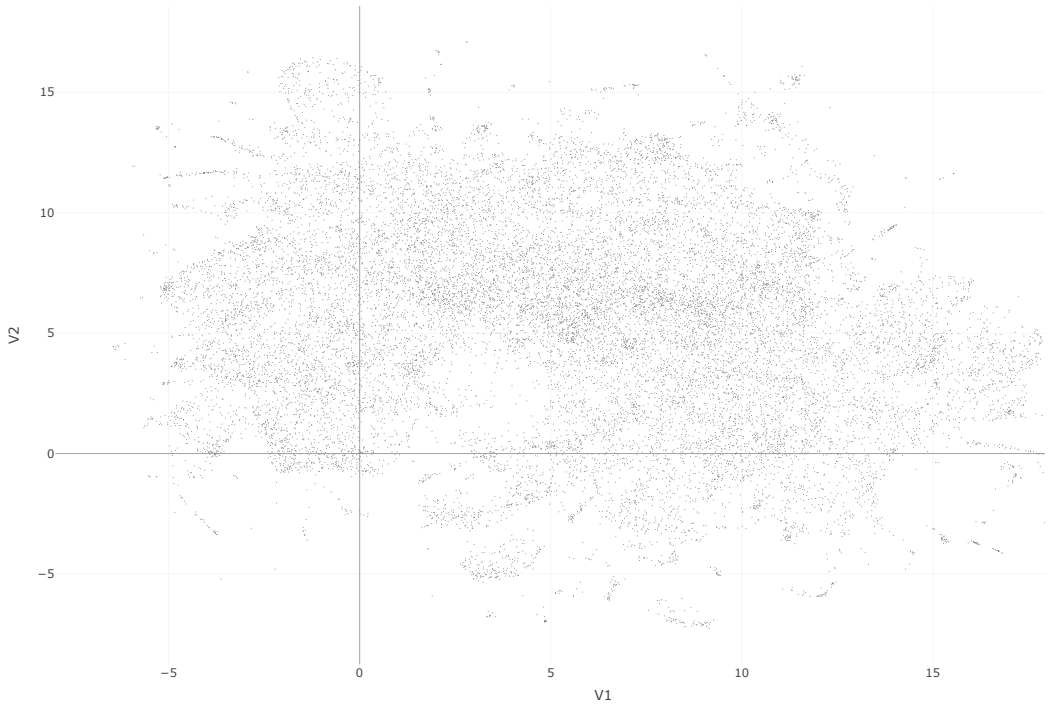
It's worth pointing out, that there are a number of ways for the user to interactively explore the landscape at this stage by scanning over each node, checking the documents contents. This helps the user to familiarise with each region of the landscape before clustering. The `plotly` package serves as a user friendly means for this purpose, helping us gather a 'lay of the land' and identify the dense and not so dense sections of our data.

> **Note**
>
> *This shows just a 20K sample from our data, which is done only to comply with data size limits on GitHub and to be more conservative with compute and memory usage. Here, we also use a message column with breaks every 10 words to ensure the visual is neater.*

```
data %>% plotly::plot_ly(
  x = ~V1,
  y = ~V2,
  type = 'scatter',
```

```
  mode = 'markers',
  marker = list( color = '#808080', size = 1),
  text = ~paste('Message: ', message_with_breaks),
  hoverinfo = 'text'
)
```

```
PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installe
```


15
10
5
0
−5
V2
−5
0
5
10
15
V1

4.3.5 Topic Modelling/Clustering:

The final steps taken are arguably the most important, this is where we will define our documents and simplify our findings byway of scientific means, in this case using Topic Modelling.

There are a number of algorithms that serve this purpose, but the more commonly used clustering techniques are KMeans and HDBSCAN. However, the example we have shown uses KMeans, where we define the number of clusters that we would expect to find beforehand and perform clustering on either the original embeddings object output by the sentence transformer model, or we can reduce those embeddings to something much smaller like 10 dimensions and cluster documents based on those. If we were to opt for HDBSCAN however, we would allow the model to determine how many clusters were formed based on some input parameters such as `min_cluster_size` which are provided by the user. For more on these two techniques and when/how to use them in a topic modelling setting, we can consult the BERTopic documentation once more.

It's also worth noting that this step requires a significant amount of human interpretation, so the user can definitely expect to partake in an iterative process of trial and error, trying out different values for the clustering parameters which determine the models output, with hopes of finding the model of best fit, which they feel accurately represents the given data.

In practise, this visualisation can be derived using our original data object with topic/cluster information appended, as well as the original `V1` and `V2` coordinates that we had used previously. To ensure our topics are coloured appropriately, we can create and use a named character vector and some additional `ggplot2` syntax to manually assign topics with specific hex codes or colours.

```
# assign colours to topics
topic_colours <- c("Ethics & Regulation" = "#C1E1C1",
                   "Technological Innovations" = "#6e88db",
                   "AI in an Artistic Domain" = "#7e2606",
                   "Cultural & Social Impact" = "#ff6361",
                   "Business & Wider-Markets" = "#063852",
                   "Future of Learning & Personal Growth" = "#ffa600",
                   "Future of Work & Security" = "#9e1ad6"
                   )
```

```
data %>%
  LandscapeR::ls_plot_group_static(x_var = V1,
                                   y_var = V2,
                                   group_var = topic_name) +
  ggplot2::scale_colour_manual(values = topic_colours) # colour nodes manually
```

Figure 4.3: Segmented Colourful Landscape Plot from an AI Landscape Microsoft Project - Q2 FY24

4.4 Downstream Flourishes

With the basics of each step covered, we will now touch on a few potentially beneficial concepts worth grasping that may help us overcome anything else that may occur when working within the Conversation Landscape project domain.

4.4.1 Model Saving & Reusability:

Occasionally, a client may want us to track topics over time or perform a landscape change analysis. In these cases, we need to save both our Dimension Reduction and Clustering models so that new data can be processed using these models, tp produce consistent and comparable results.

This requires careful planning. When we initially reduce embeddings and perform clustering, we use the `.fit()` method from `sklearn` when either reducing the dimensions of or clustering on the original embeddings. This ensures that the models are trained on the data they are intended to represent, making future outputs comparable.

We had earlier, mentioned, that it is crucial to document the versions of the modules and Python interpreter used. When we reduce or cluster new data using our pre-fitted models, it is essential to do so with the exact same versions of important libraries and Python. The reason is that the internal representations and binary structures of the models can differ between versions. If we attempt to load and apply previously saved models with different versions, we risk encountering incompatibility errors. By maintaining version control and documenting the environment in which the models were created, we can ensure the reusability of our models. Overall, this practice allows for us to be accurate when tracking and comparing topics and noting any landscape changes.

4.4.2 Efficient Parameter Tuning:

When we're performing certain steps within this workflow, more specifically the Dimension Reduction with likes of UMAP, or if we were to decide we'd want to cluster using HDBSCAN for example, being mindful of and efficient with tuning the different parameters at each step will definitely improve the outcome of our overall model. Therefore, understanding these key parameters and how they can interact will significantly enhance the performance of the techniques being used here.

4.4.2.1 Dimension Reduction with UMAP:

n_neighbors: This parameter controls the local neighborhood size used in UMAP. A smaller value focuses more on capturing the local structure, while a larger value considers more global aspects. Efficiently tuning this parameter involves considering the nature of your data and the scale at which you want to observe patterns.

min_dist: The min distance argument determines quite literally how tight our nodes are allowed to be positioned together within our semantic space, a lower value for this will mean nodes will be tightly packed together, whereas a higher number will ensure larger spacing of data points.

n_components: Here is where we decide how many dimensions we wish to reduce our high-dimensional embeddings object down to, for visualisation we will likely set this parameter to a value of 2.

4.4.2.2 KMeans CLustering

n_clusters: KMeans is a relatively simple algorithm compared to other methods and components, requiring very little input. Here we just provide a value for the number of clusters we wish to form, this will either be clusters in the embeddings or a smaller, more manageable reduced embeddings object as mentiioned previously.

4.4.2.3 HDBSCAN Clustering:

min_samples: This parameter defines the minimum number of points required to form a dense region. It helps determine the density threshold for clusters and can determine how conservative we want the clustering model to be. Put simply, a higher value can lead to fewer, larger clusters, while a lower value can result in more, smaller clusters.

min_cluster_size: This parameter sets the minimum size of clusters. Like `min_samples` it can directly influence the granularity of the clustering results. In this case, smaller values allow the formation of smaller clusters, while larger values prevent the algorithm from identifying any small clusters(or those below the size of the provided value). It's worth noting that the relationship between `min_samples` and `min_cluster_size` is crucial. `min_samples` should generally be less than or equal to `min_cluster_size`. Adjusting these parameters in tandem helps us to control the sensitivity of HDBSCAN, and for us to define what qualifies as a cluster.

4.4.2.4 Tip: Try starting with the default value for all of these parameters, and incrementally adjust based on the desired granularity or effect of any that we wish to amend.

4.4.3 Supporting Data Visualisation:

Once we have our landscape output, as shown in Final output of project, we will inevitably need to display some further information regarding our topics, most commonly; Volume over Time (VOT) and Sentiment Distribution for each.

When doing so, we would ideally keep some formatting consistencies when plotting, as we mentioned previously, the colouring of our topics must remain the same throughout so that they match up with any previous representations in existing visuals such as the landscape output. We would also want to ensure that any plot we create orders our topics by volume or

at least in the same order throughout our project. We can order our topics in terms of volume easily with just a few lines of code.

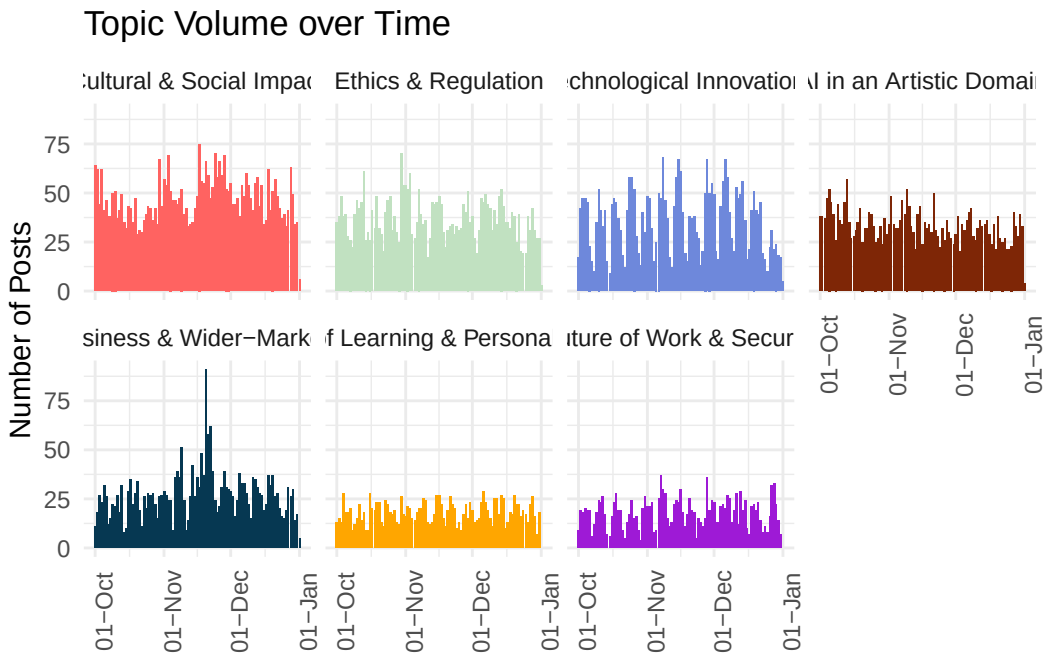
First, we'll make sure to set the factor levels of our topics by using `dplyr::count()` on the `topic_name` column, and setting the levels feature of the `factor()` base function based on the counted output.

```
# sort topics by order of volume
topic_order <- data %>% dplyr::count(topic_name, sort = TRUE)
# set levels determined by volume of topic, this orders the group variable for plotting
data <- data %>%
  dplyr::mutate(topic_name = factor(topic_name, levels = topic_order$topic_name))
```

4.4.3.1 Topic Volume over Time

Starting with volume over time, we often choose to render a faceted plot that includes all topics and their VOT for comparison. We can do so by using functionality found in packages such as `JPackage` for this.

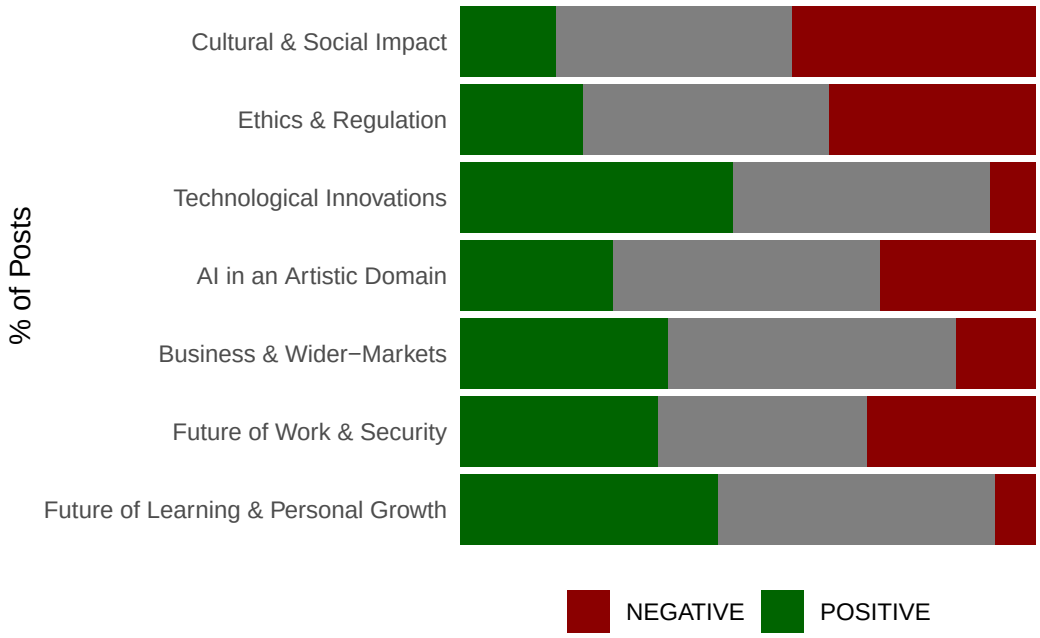
```
# plot topic volume over time using 'plot_group_vol_time()' function
data %>%
  JPackage::plot_group_vol_time(group_var = topic_name,
                                date_var = date,
                                unit = "day",
                                nrow = 2) +
  ggplot2::scale_fill_manual(values = topic_colours) # apply colours manually!
```



4.4.3.2 Topic Sentiment Distribution

Next, we might want/need to break each of our topics out by their sentiment distribution to help shine light on any of particular interest or to help us tell a more refined story using our topic model. This can be done by using the `dr_plot_sent_group()` function of the `DisplayR` package.

```
data %>%
  DisplayR::dr_plot_sent_group(group_var = topic_name,
                               sentiment_var = sentiment,
                               "percent", bar_labels = "none",
                               sentiment_colours = c("POSITIVE" = "darkgreen",
                                                     "NEGATIVE" = "darkred"))
```



4.4.3.3 Alternative Visualisations

While the two visuals we have displayed so far are relatively basic and commonly used, this does not mean that we won't require alternative methods to display topic-level information. Often, we may render n-grams per topic to display the relationships that exist between terms/phrases, and we may create plots to showcase things such as data source or social network/platform distributions across topics.

Finally, it's worth noting that the need for specific data visualisation methods entirely depends on the project domain and brief, as well as any outcomes/findings derived throughout. This means we ought to be flexible in our approach to utilising any technique that may assist with strengthening our understanding of the data and/or supporting our analyses.

Part III

Project work