Text Summarization with Large Language Models

Table of Contents

- Introduction (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#intro)
  - The Transformer Architecture (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#transformers)
- This Notebook (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#this_notebook)
  - The Task (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#task)
  - The Dataset (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#data)
  - The Model (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#model)
  - Evaluation Metrics (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#eval)
- Exploring the Dataset (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#eda)
  - Train Dataset (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#train)
  - Test Dataset (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#test)
  - Validation Dataset (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#val)

- Preprocessing Data (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#preprocess)
- Modeling (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#modeling)
- Evaluating and Saving Model (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#evaluating)
- Conclusion and Deployment (https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models#conclusion)

Introduction

November 30th, 2022, marks a significant chapter in the History of **machine learning**. It was the day OpenAI released ChatGPT, setting a new benchmark for chatbots powered by **Large Language Models** and offering the public an unparalleled conversational experience.

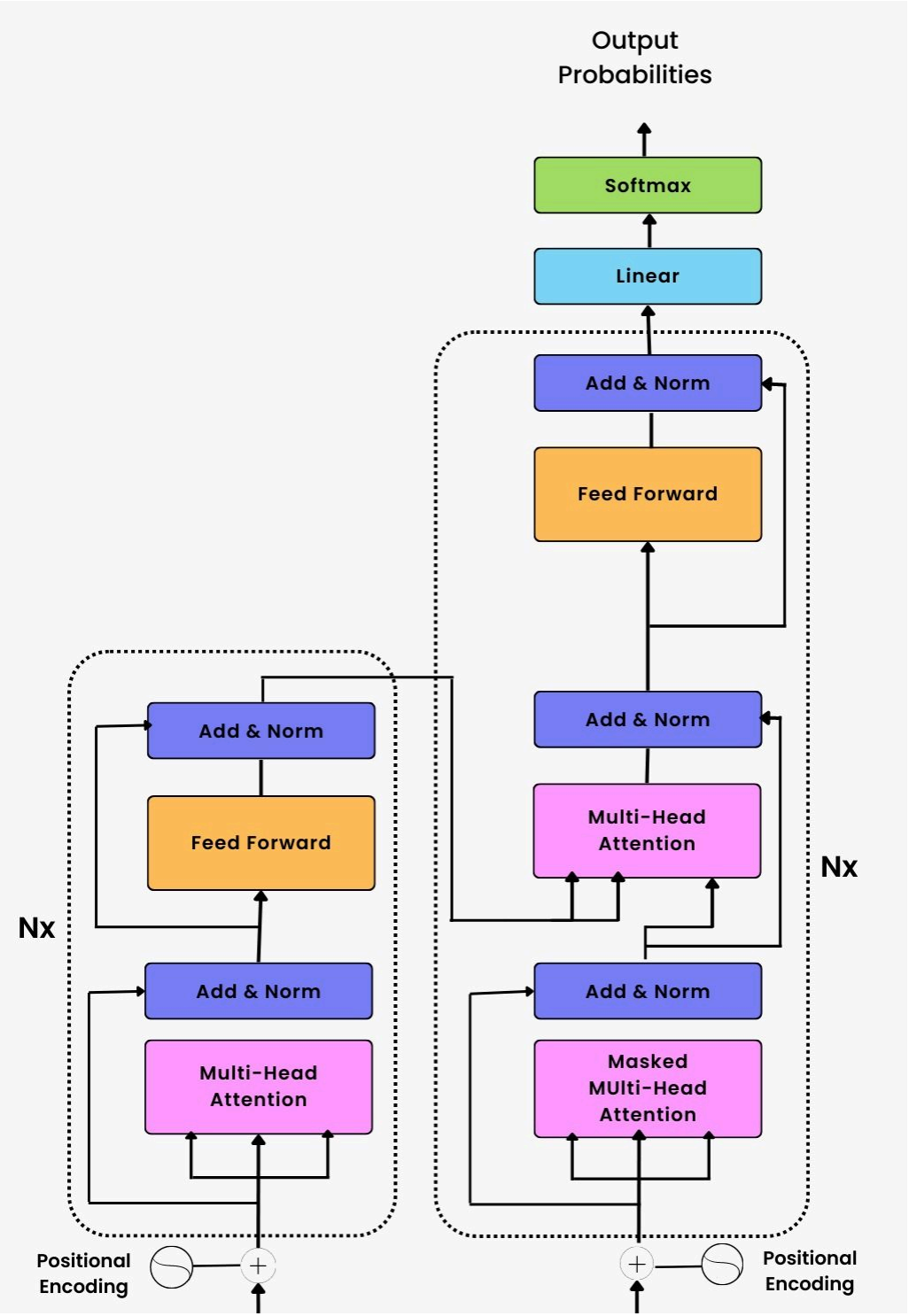
Ever since then, large language models — also referred to as **LLMs** —, have been in the public eye due to the extensive number of tasks they are able to perform. Examples include:

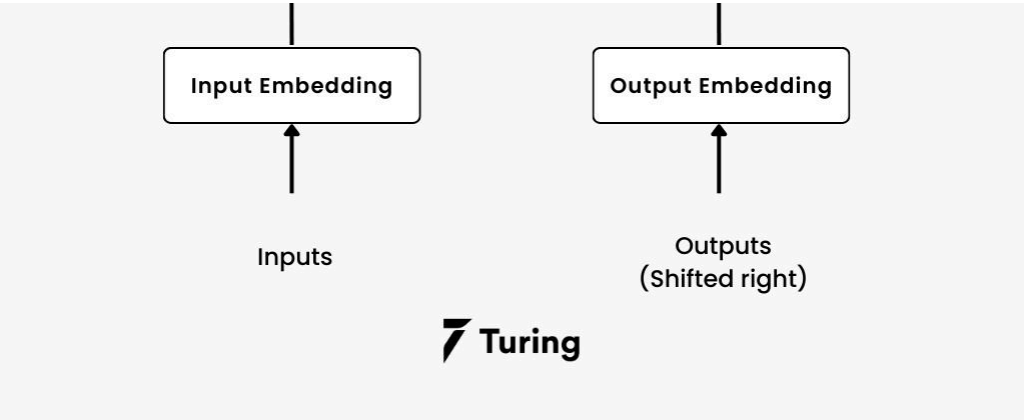
- **Text Summarization**: These models are able to perform a summarization of large texts, including legal texts, reviews, dialogues, among many others.
- **Sentiment Analysis**: They can read through reviews of products and services and classify them as positive, negative, or neutral. These can also be used in Finance to see if the general public feels *Bullish* or *Bearish* on certain securities.
- **Language Translation**: They can provide real-time translations from one language to another.
- **Text-based Recommender Systems**: They can also recommend new products for a client based on their reviews on previously bought products.

But how do these models actually work? 🤔

The Transformer Architecture

To understand the current state of LLMs, we must go back to Google's 2017 **Attention is All You Need**. In this paper, the **Transformer** architecture was introduced to the world, and it changed the industry forever.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked MUlti-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding



Transformer architecture.

Source: [Turing (https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power)](https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power)

While recurrent neural networks could be used to enable computers to comprehend text, these models were extremely limited due to the fact that they only allowed the machine to process one word at a time, which would result in the model not being able to acquire the full context of a text.

The **transformer architecture**, however, is based on the attention mechanism, which allows the model to process an entire sentence or paragraph at once, rather than each word at a time. This is the main secret behind the possibility of full context comprehension, which gives much more power to all these language processing models.

The processing of text input with the transformer architecture is based on **tokenization**, which is the process of transforming texts into smaller components called tokens. These can be words, subwords, characters, or many others.

The tokens are then mapped to numerical IDs, which are unique for each word or subword. Each ID is then transformed into an **embedding**: a dense, high-dimensional vector that contains numerical values. These values are designed to capture the original meaning of the tokens and serve as input for the transformer model.

It is important to note that these embeddings are high-dimensional, with each dimension capturing certain aspects of a token's meaning. Due to their high-dimensional nature, embeddings are not easily interpreted by humans, but transformer models readily use them to identify and group together tokens with similar meanings in the vector space.

Take the following example:

Original Text	**Tokenized Text**	**Numerical IDs**	**Embeddings (First 3 Dimensions)**
As she said this, she looked down at her hands, and was surprised to find that she had put on one of the rabbit's little gloves while she was talking.	['As', ' she', ' said', ' this', ',', ' she', ' looked', ' down', ' at', ' her', ' hands', ',', ' and', ' was', ' surprised', ' to', ' find', ' that', ' she', ' had', ' put', ' on', ' one', ' of', ' the', ' rabbit', "'s", ' little', ' gloves', ' while', ' she', ' was', ' talking', '.']	['7', ' 22', ' 258', ' 430', . . ., '589', ' 22', ' 78', ' 98', ' 5890']	['As': [1.12, -0.56, 0.07], ['she': [0.88, 0.45, -2.03], . . .]

By using this vector as input, the transformer model learns how to generate outputs based on the **probabilities of subsequent words that may naturally follow an input word**. This process gets repeated until the model creates an entire paragraph starting from an initial statement.

There is a very intriguing post on Andrej Karpathy's blog, ***The Unreasonable Effectiveness of Recurrent Neural Networks*** (http://karpathy.github.io/2015/05/21/rnn-effectiveness/), that explains why neural networks-based models are effective in predicting the next word of a text. One factor contributing to their effectiveness is the inherent *rules* in human languages, such as grammar, which constrain word usage in sentences.

When you feed your model with examples of written language — news articles, Twitter/X posts, product reviews, messages, dialogues, etc. — it implicitly acquires the rules of language through these examples, which helps it to predict sequences of words and generate human-like texts.

A large language model — such as *GPT*, *BERT*, *RoBERTa*, etc. — is a transformer model on a much larger scale. These models are built on an enormous amount of texts, so they learn and become experts in patterns and structures of language. The GPT-4, which is the

model behind the premium version of ChatGPT, was trained on massive amounts of text data from the internet, such as books, articles, websites, etc.

It is also relevant to note that different languages exhibit different patterns and structures. While Western European languages like English, French, German, Spanish, Portuguese, and Italian may share many structural similarities, other languages, such as Arabic and Japanese, are very distinct, posing unique challenges to modeling.

This Notebook

The goal of this notebook is to demonstrate how Large Language Models can be used for several tasks related to language processing. In this case, I am going to leverage the power of **transfer learning** to build a model capable of summarizing dialogues.

For those of you who may not be aware, transfer learning is a machine learning technique in which we use a *pre-trained model*—that is already knowledgeable in a wide domain—and tailor its expertise for a specific task by training it in a specific dataset we might have. This process may also be referred to as **fine-tuning**.

The 🤗 **Transformers** (https://huggingface.co/docs/transformers/index) library—which is one of the most popular libraries for working with deep learning tasks—offers the possibility of working with the following architectures:

Model Architectures
BART, BigBird-Pegasus, Blenderbot, BlenderbotSmall, Encoder decoder, FairSeq Machine-Translation, GPTSAN-japanese, LED, LongT5, M2M100, Marian, mBART, MT5, MVP, NLLB, NLLB-MOE, Pegasus, PEGASUS-X, PLBart, ProphetNet, SwitchTransformers, T5, UMT5, XLM-ProphetNet

The 🤗 **Transformers** library allows us to easily download and fine-tune state-of-the-art pre-trained models, and also allows us to easily work with both **TensorFlow** and **PyTorch** for several tasks related to Natural Language Preprocessing, Computer Vision, Audio, etc.

The Task

As previously mentioned, the task at hand is **Text Summarization**. From the documentation of the 🤗 Transformers library, summarization can be described as the creation of *a shorter version of a document or an article that captures all the important information*.

In this case, we are going to summarize dialogues by using a dataset containing chat texts.

For this task, we are going to use the **SamSum Dataset** (https://www.kaggle.com/datasets/nileshmalode1/samsum-dataset-text-summarization/versions/1), which contains three *csv* files for training, testing, and validation. All these files are structured into a specific `id`, a `dialogue`, and a `summary`. The SamSum dataset consists of chat texts, which is ideal for the summarization of dialogues.

The Model

As previously mentioned, we are going to harness the power of a pre-trained model for this task. In this case, I have decided to use the **BART** architecture, proposed in the 2019 paper BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (https://arxiv.org/abs/1910.13461). More specifically, I am going to fine-tune a version of BART that has been already trained to perform text summarization of news articles, which is the **facebook/bart-large-xsum** (https://huggingface.co/facebook/bart-large-xsum) version.

Briefly explaining, BART is a denoising autoencoder that employs the strategy of distorting the input text in many ways, such as blanking out some words and flipping them around, and then learning to reconstruct it. BART has outperformed established models like RoBERTa and BERT on multiple NLP benchmarks, and it is especially efficient in summarization tasks, due to its ability to generate text and learn the context of the input text.

For a deeper comprehension of BART, I highly suggest you read the research paper linked above, where this architecture was first introduced.

Evaluation Metrics

Evaluating performance for language models can be quite tricky, especially when it comes to text summarization. The goal of our model is to produce a short sentence describing the content of a dialogue, while maintaining all the important information within that dialogue.

One of the quantitative metrics we can employ to evaluate performance is the **ROUGE Score**. It is considered one of the best metrics for text summarization and it evaluates performance by comparing the quality of a machine-generated summary to a human-generated summary used for reference.

The similarities between both summaries are measured by analyzing the overlapping *n-grams*, either single words or sequences of words that are present in both summaries. These can be unigrams (ROUGE-1), where only the overlap of sole words is measured; bigrams (ROUGE-2), where we measure the overlap of two-word sequences; trigrams (ROUGE-3), where we measure the overlap of three-word sequences; etc. Besides that, we also have:

- **ROUGE-L**: It measures the *Longest Common Subsequence (LCS)* between the two summaries, which helps to capture content coverage of the machine-generated text. If both summaries have the sequence *"the apple is green"*, we have a match regardless of where they appear in both texts.

- **ROUGE-S**: It evaluates the overlap of skip-bigrams, which are bigrams that permit gaps between words. This helps to measure the coherence of a machine-generated summary. For example, in the phrase *"this apple is absolutely green"*, we find a match for the terms such as *"apple"* and *"green"*, if that is what we are looking for.

These scores might typically range from 0 to 100, where 0 indicates no match and 100 indicates a perfect match between both summaries.

Besides quantitative metrics, it is useful to use **human evaluation** to analyze the output of language models, since we are able to comprehend text in a way that a machine does not. So we might read the dialogue and then read the summary to check if it is an accurate summarization.

••• 7 hidden cells

In [11]:

```
# Transformers
from transformers import BartTokenizer, BartForConditionalGeneration      # BERT Tokenizer and architec
ture
from transformers import Seq2SeqTrainer, Seq2SeqTrainingArguments         # These will help us to fine-
tune our model
from transformers import pipeline                                         # Pipeline
from transformers import DataCollatorForSeq2Seq                           # DataCollator to batch the d
ata
import torch                                                              # PyTorch
import evaluate                                                           # Hugging Face's library for
model evaluation


# Other NLP libraries
from textblob import TextBlob                                             # This is going to help us fi
x spelling mistakes in texts
from sklearn.feature_extraction.text import TfidfVectorizer              # This is going to helps iden
tify the most common terms in the corpus
import re                                                                 # This library allows us to c
lean text data
import nltk                                                               # Natural Language Toolkit
nltk.download('punkt')                                                    # This divides a text into a
list of sentences
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[11]:

```
True
```

By observing the imports above, you can clearly note that I have choosen to work with **PyTorch** for this notebook.

In [9]:
```
# Configuring Pandas to exhibit larger columns
'''
This is going to allow us to fully read the dialogues and their summary
'''
pd.set_option('display.max_colwidth', 1000)
```

In [2]:
```
# Configuring notebook
seed = 42
#paper_color =
#bg_color =
colormap = 'cividis'
template = 'plotly_dark'
```

Hide code

In [11]:
```
# Checking if GPU is available
if torch.cuda.is_available():
    print("GPU is available. \nUsing GPU")
    device = torch.device('cuda')
else:
    print("GPU is not available. \nUsing CPU")
    device = torch.device('cpu')
```

```
GPU is available.
Using GPU
```

4 hidden cells

Exploring the Dataset

We can start our analysis of the dataset by loading all the three sets available, `train`, `test`, and `val`.

In [5]:
```
# Loading data
train = pd.read_csv('/kaggle/input/samsum-dataset-text-summarization/samsum-train.csv')
test = pd.read_csv('/kaggle/input/samsum-dataset-text-summarization/samsum-test.csv')
val = pd.read_csv('/kaggle/input/samsum-dataset-text-summarization/samsum-validation.csv')
```

I am now going to analyze each dataset separately.

Train Dataset

In [17]:

```
# Extracting info on the training Dataframe
describe_df(train)
```

```
DataFrame shape: (14732, 3)

14,732 samples

3 attributes

Missing Data:
id          0
dialogue    1
summary     0
dtype: int64

Duplicates: 0

Data Types:
id          object
dialogue    object
summary     object
dtype: object

Categorical Features:
id, dialogue, summary

Continuous Features:
None

Binary Features:
None
```

DataFrame Head:

	id	dialogue	summary
0	13818513	Amanda: I baked cookies. Do you want some?\r\nJerry: Sure!\r\nAmanda: I'll bring you tomorrow :-)	Amanda baked cookies and will bring Jerry some tomorrow.
1	13728867	Olivia: Who are you voting for in this election? \r\nOliver: Liberals as always.\r\nOlivia: Me too!!\r\nOliver: Great	Olivia and Olivier are voting for liberals in this election.
2	13681000	Tim: Hi, what's up?\r\nKim: Bad mood tbh, I was going to do lots of stuff but ended up procrastinating\r\nTim: What did you plan on doing?\r\nKim: Oh you know, uni stuff and unfucking my room\r\nKim: Maybe tomorrow I'll move my ass and do everything\r\nKim: We were going to defrost a fridge so instead of shopping I'll eat some defrosted veggies\r\nTim: For doing stuff I recommend Pomodoro technique where u use breaks for doing chores\r\nTim: It really helps\r\nKim: thanks, maybe I'll do that\r\nTim: I also like using post-its in kaban style	Kim may try the pomodoro technique recommended by Tim to get more stuff done.
3	13730747	Edward: Rachel, I think I'm in ove with Bella..\r\nrachel: Dont say anything else..\r\nEdward: What do you mean??\r\nrachel: Open your fu**ing door.. I'm outside	Edward thinks he is in love with Bella. Rachel wants Edward to open his door. Rachel is outside.
4	13728094	Sam: hey overheard rick say something\r\nSam: i don't know what to do :-/\r\nNaomi: what did he say??\r\nSam: he was talking on the phone with someone\r\nSam: i don't know who\r\nSam: and he was telling them that he wasn't very happy here\r\nNaomi: damn!!!\r\nSam: he was saying he doesn't like being my roommate\r\nNaomi: wow, how do you feel about it?\r\nSam: i thought i was a good rommate\r\nSam: and that we have a nice place\r\nNaomi: that's true man!!!\r\nNaomi: i used to love living with you before i moved in with me boyfriend\r\nNaomi: i don't know why he's saying that\r\nSam: what should i do???\r\nNaomi: honestly if it's bothering you that much you should talk to him\r\nNaomi: see what's going on\r\nSam: i don't want to get in any kind of confrontation though\r\nSam: maybe i'll just let it go\r\nSam: and see how it goes in the future\r\nNaomi: it's your choice sam\r\nNaomi: if i were you i would just talk to him and clear the air	Sam is confused, because he overheard Rick complaining about him as a roommate. Naomi thinks Sam should talk to Rick. Sam is not sure what to do.

DataFrame Tail:

	id	dialogue	summary
14727	13863028	Romeo: You are on my 'People you may know' list.\nGreta: Ah, maybe it is because of the changed number of somebody's?\nGreta: I don't know you?\nRomeo: This might be the beginning of a beautiful relationship\nRomeo: How about adding me on your friend list and talk a bit?\nGreta: No.\nRomeo: Okay I see.	Romeo is trying to get Greta to add him to her friend list but she refuses.
14728	13828570	Theresa: <file_photo>\r\nTheresa: <file_photo>\r\nTheresa: Hey Louise, how are u? \r\nTheresa: This is my workplace, they always give us so much food here 😊\r\nTheresa: Luckily they also offer us yoga classes, so all the food isn't much of a problem 😂\r\nLouise: Hey!! 🙂 \r\nLouise: Wow, that's awesome, seems great 😎 Haha\r\nLouise: I'm good! Are you coming to visit Stockholm this summer? 🙂\r\nTheresa: I don't think so :/ I need to prepare for Uni.. I will probably attend a few lessons this winter\r\nLouise: Nice! Do you already know which classes you will attend?\r\nTheresa: Yes, it will be psychology :) I want to complete a few modules that I missed :)\r\nLouise: Very good! Is it at the Uni in Prague? \r\nTheresa: No, it will be in my home town :)\r\nLouise: I have so much work right now, but I will continue to work until the end of summer, then I'm also back to Uni, on the 26th September!\r\nTheresa: You must send me some pictures, so I can see where you live :) \r\nLouise: I will,...	Theresa is at work. She gets free food and free yoga classes. Theresa won't go to visit Louise in Stockholm, because she will prepare for university psychology lessons. She'll be back at uni on 26th September.
14729	13819050	John: Every day some bad news. Japan will hunt whales again\r\nErica: Yes, I've read this. It's very upsetting\r\nJohn: Cruel Japanese\r\nFaith: I think this is a racist remark. Because Island and Norways has never joined this international whaling agreement\r\nErica: really? I haven't known, everybody is so outraged by Japan\r\nFaith: sure, European hypocrisy \r\nJohn: not entirely. Scandinavians don't use the nets that Japanese use, so Norway and Island kill much less specimens that Japan will\r\nFaith: oh, it's much more complex than one may expect\r\nJohn: True, but the truth is, that all of them should stop\r\nJohn: and this decision is a step back\r\nFaith: yes, this is worrying\r\nErica: And it seems that the most important whaling countries are out of the agreement right now\r\nFaith: yes, seems so\r\nJohn: Just like USA leaving the Paris Agreement	Japan is going to hunt whales again. Island and Norway never stopped hunting them. The Scandinavians kill fewer whales than the Japanese.
14730	13828395	Jennifer: Dear Celia! How are you doing?\r\nJennifer: The afternoon with the Collins was very pleasant, nice folks, but we missed you.\r\nJennifer: But I appreciate your consideration for Peter.\r\nCelia: My dear Jenny! It turns out that my decision not to come, though I wanted so much to see you again and Peter and the Collins, was right. Yesterday it all developed into a full bore cold. Sh.....\r\nCelia: All symptoms like in a text book.\r\nCelia: Luckily it's contagious only on the first 2, 3 days, so when we meet next week it should be alright.\r\nCelia: Thanks for asking! Somehow for all of us Peter comes first now.\r\nJennifer: That's too bad. Poor you...\r\nJennifer: I'll be driving to FR, do you want me to bring you sth? It's on my way.\r\nCelia: Thank you dear! I was at the pharmacy yesterday and had done my shopping the day before.\r\nCelia: You'd better still stay away from me in case I'm still contagious\r\nJennifer: Right. So I'll only leave a basket on your terrace. A...	Celia couldn't make it to the afternoon with the Collins and Jennifer as she is ill. She's working, but doesn't want to meet with Jennifer as it might be contagious. Jennifer will leave a basket with cookies on Celia's terrace.

	id	dialogue	summary
14731	13729017	Georgia: are you ready for hotel hunting? We need to book something finally for Lisbon\r\nJuliette: sure we can go on, show me what you found\r\nGeorgia: <file_photo>\r\nJuliette: nah... it looks like an old lady's room lol\r\nGeorgia: <file_photo>\r\nJuliette: that's better... but the bed doesn't look very comfortable\r\nGeorgia: i kind of like it and it's really close to the city center\r\nJuliette: show me the others please\r\nGeorgia: <file_photo>\r\nJuliette: nah... this one sucks too, look at those horrible curtains \r\nGeorgia: aff Julie you are such a princess\r\nJuliette: i just want to be comfortable\r\nGeorgia: come on, stop whining you know we are on a budget\r\nJuliette: well hopefully we can find something that's decent right?\r\nGeorgia: i did show you decent but you want a Marriott or something :/\r\nJuliette: ok ok don't get angry\r\nGeorgia: we need to decide today, the longer we wait the higher the prices get \r\nJuliette: ok how about we get the second one then?...	Georgia and Juliette are looking for a hotel in Lisbon. Juliette dislikes Georgia's choices. Juliette and Georgia decide on the second option presented by Georgia, but it has already been booked. Finally Georgia books the third hotel.

We have 14,732 pairs of dialogues and summaries. It also seems like one of the dialogues is empty, let's investigate it further.

Hide code

In [6]:
```
mask = train['dialogue'].isnull() # Creating mask with null dialogues
filtered_train = train[mask] # filtering dataframe
filtered_train # Visualizing
```

Out[6]:

	id	dialogue	summary
6054	13828807	NaN	problem with visualization of the content

It seems that sample **6054** does not really add anything to the dataset. We have a Null dialogue and the summary does not give us a clue on what this dialogue was supposed to be. We will remove this entry.

In [7]:
```
train = train.dropna() # removing null values
```

Hide code

In [20]:
```
# Removing 'Id' from categorical features list
categorical_features.remove('id')
```

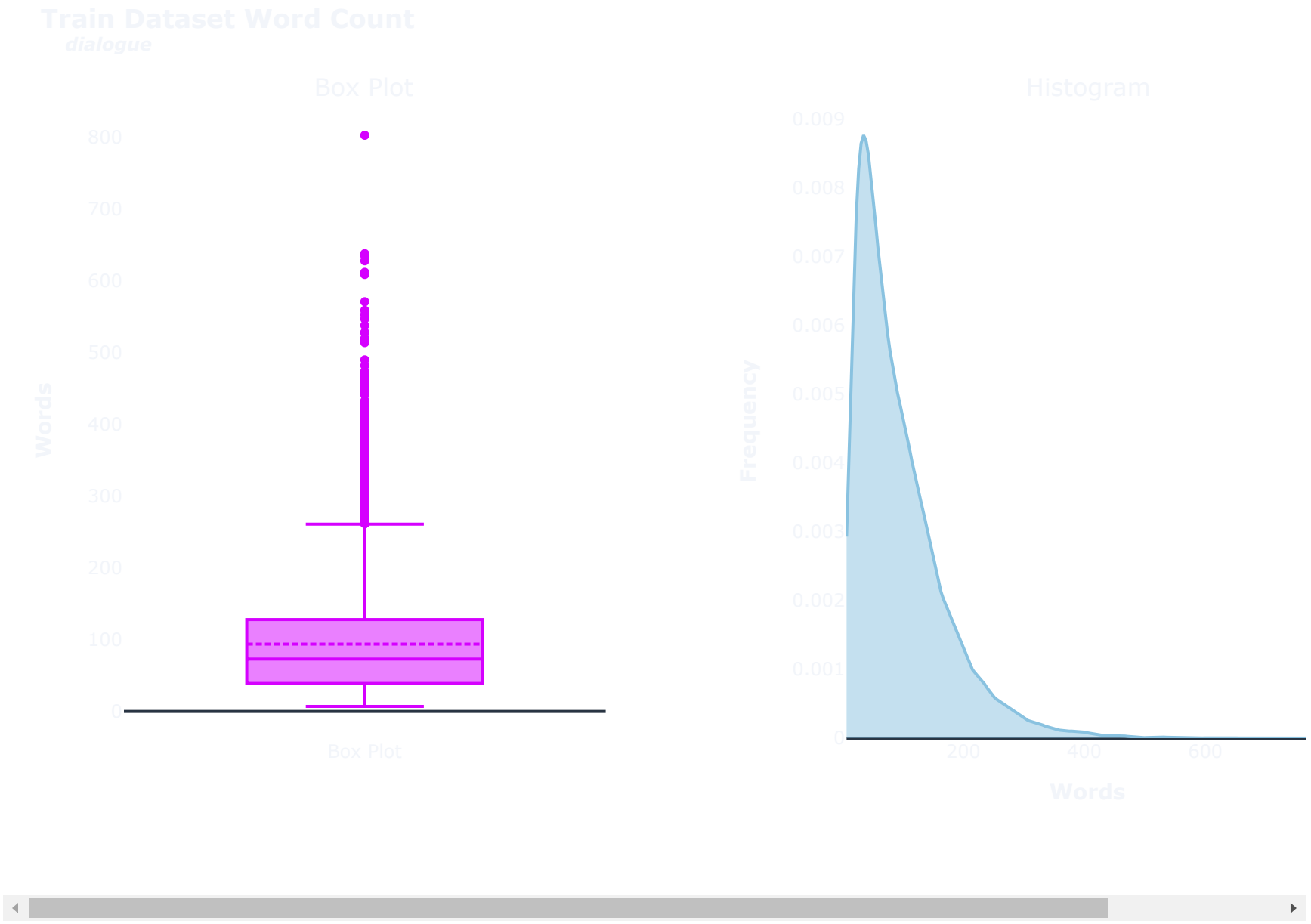
We can now analyze the length of both dialogues and summaries by counting the words in them. This might give us a clue about how these texts are structured.

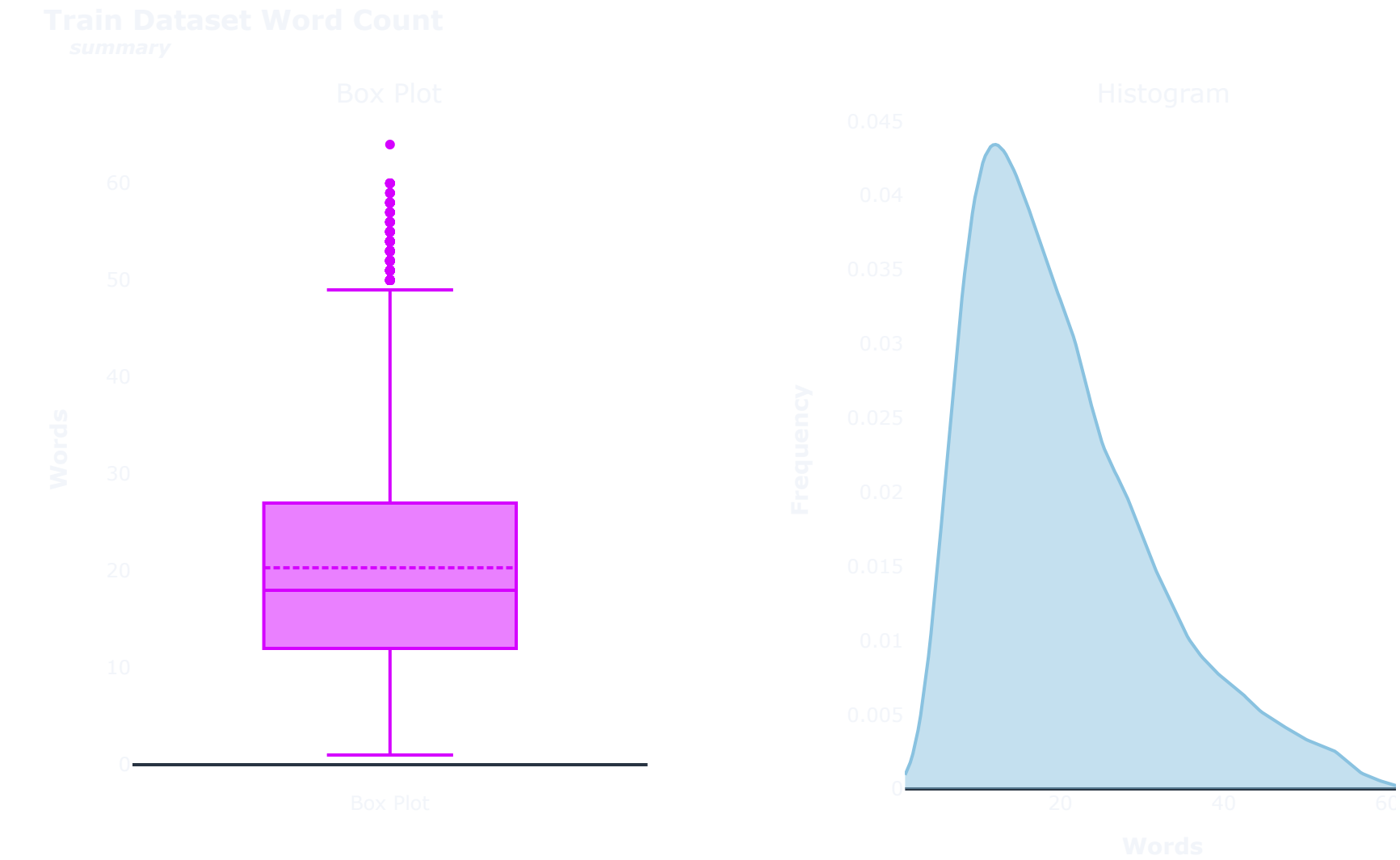
Hide code

In [21]:

```
df_text_lenght = pd.DataFrame() # Creating an empty dataframe
for feat in categorical_features: # Iterating through features --> Dialogue & Summary
    df_text_lenght[feat] = train[feat].apply(lambda x: len(str(x).split())) #  Counting words for each feature

# Plotting histogram-boxplot
histogram_boxplot(df_text_lenght,'#89c2e0', '#d500ff', 600, 1000, True, 'Train Dataset')
```


Train Dataset Word Count
dialogue
Box Plot
Words
800
700
600
500
400
300
200
100
0
Box Plot
Histogram
Frequency
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
200
400
600
Words


Train Dataset Word Count
summary
Box Plot
Words
60
50
40
30
20
10
0
Box Plot
Histogram
Frequency
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
20
40
60
Words

On average, dialogues consist of about 94 words. We do have some outliers with very extensive texts, going way over 300 words per dialogue.

Summaries are naturally shorter texts, consisting of about 20 words on average, although we also have some outliers with extensive summaries.

We can also use scikit-learn's `TfidfVectorizer` to extract more info on the dialogues and summaries available. This function will give us a dataframe with the top n most frequent terms in the corpus, which we select by using the `max_features` parameter.

In this dataframe, each column represents the n most frequent terms in the overall corpus, while each row represents one entry in the original dataframe, such as `train`. For each term in each entry, we will see the TF-IDF score associated with it, which quantifies the relevance of a term in a given dialogue — or summary — relative to its frequency across all other dialogues — or summaries.

We will also use the `ngram_range` parameter to select the most frequent words (unigrams), the most frequent sequence of two words (bigrams), and the most frequent sequence of three words (trigrams). The `stop_words = 'english'` parameter will help us filter out common stop-words of the English language, which are words that do not add up much to the overall context, such as *"and"*, *"of"*, etc.

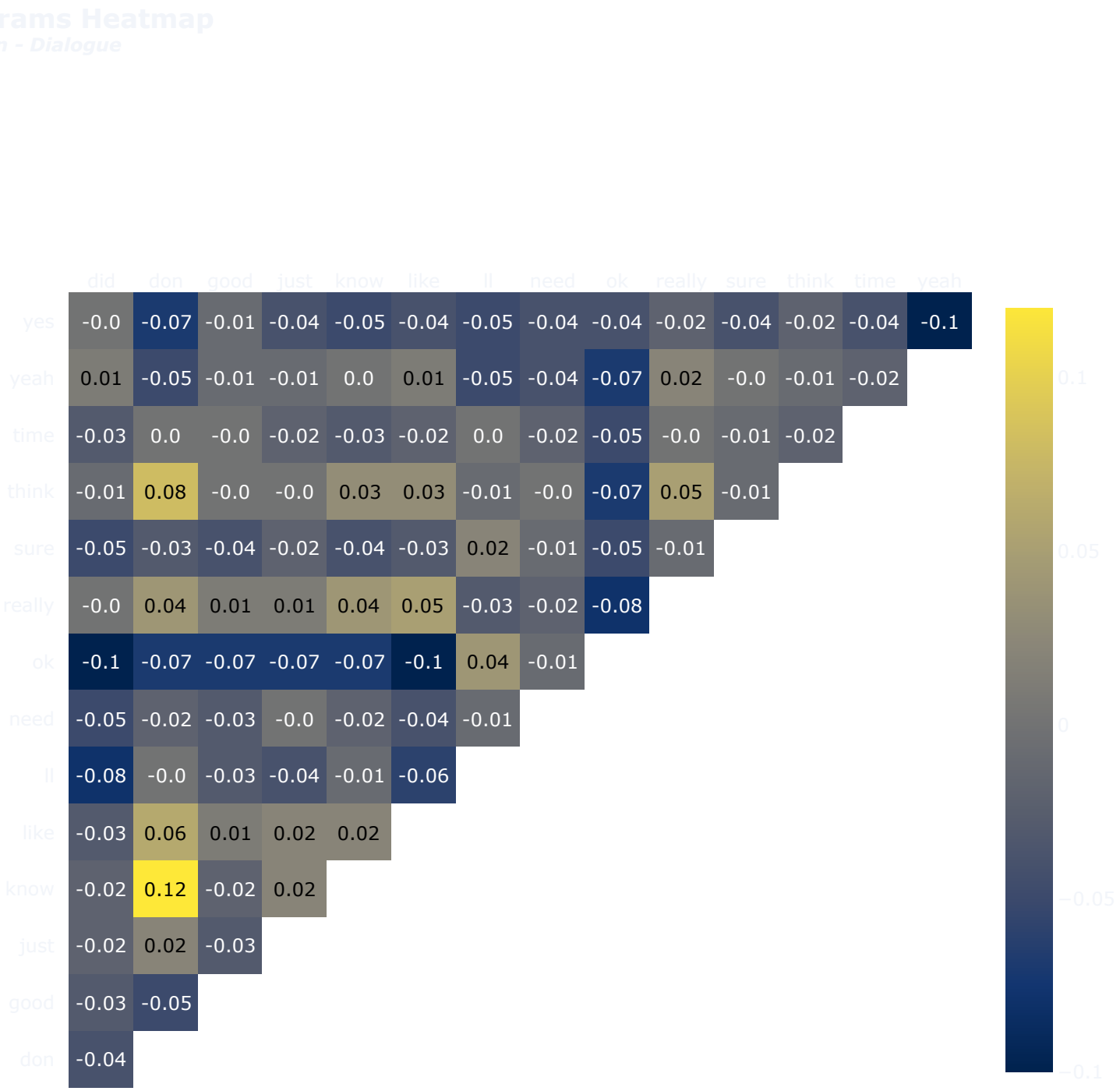
After measuring the most frequent terms, I will plot a heatmap displaying the correlations between these terms. This may help us understand how frequently they are used together in dialogues. For instance, how frequent is the occurrence of the word *"will"* when the

word *"we"* is present?

Hide code

In [12]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english') # Top 15 terms
x = vectorizer.fit_transform(train['dialogue'])
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Unigrams', 'Train - Dialogue', 800, 800, 12)
```


Unigrams Heatmap
Train - Dialogue
did don good just know like ll need ok really sure think time yeah
yes -0.0 -0.07 -0.01 -0.04 -0.05 -0.04 -0.05 -0.04 -0.04 -0.02 -0.04 -0.02 -0.04 -0.1
yeah 0.01 -0.05 -0.01 -0.01 0.0 0.01 -0.05 -0.04 -0.07 0.02 -0.0 -0.01 -0.02
time -0.03 0.0 -0.0 -0.02 -0.03 -0.02 0.0 -0.02 -0.05 -0.0 -0.01 -0.02
think -0.01 0.08 -0.0 -0.0 0.03 0.03 -0.01 -0.0 -0.07 0.05 -0.01
sure -0.05 -0.03 -0.04 -0.02 -0.04 -0.03 0.02 -0.01 -0.05 -0.01
really -0.0 0.04 0.01 0.01 0.04 0.05 -0.03 -0.02 -0.08
ok -0.1 -0.07 -0.07 -0.07 -0.07 -0.1 0.04 -0.01
need -0.05 -0.02 -0.03 -0.0 -0.02 -0.04 -0.01
ll -0.08 -0.0 -0.03 -0.04 -0.01 -0.06
like -0.03 0.06 0.01 0.02 0.02
know -0.02 0.12 -0.02 0.02
just -0.02 0.02 -0.03
good -0.03 -0.05
don -0.04
0.1
0.05
0
−0.05
−0.1

You can see that the correlations between these terms are neither strongly positive nor strongly negative. The most positively correlated terms are *"don"* and *"know"*, at 0.12. It is relevant to observe that the `TfidfVectorizer` function performs some changes to the text, such as removing contractions, which explains why the word *don't* appears without its apostrophe *'t*.

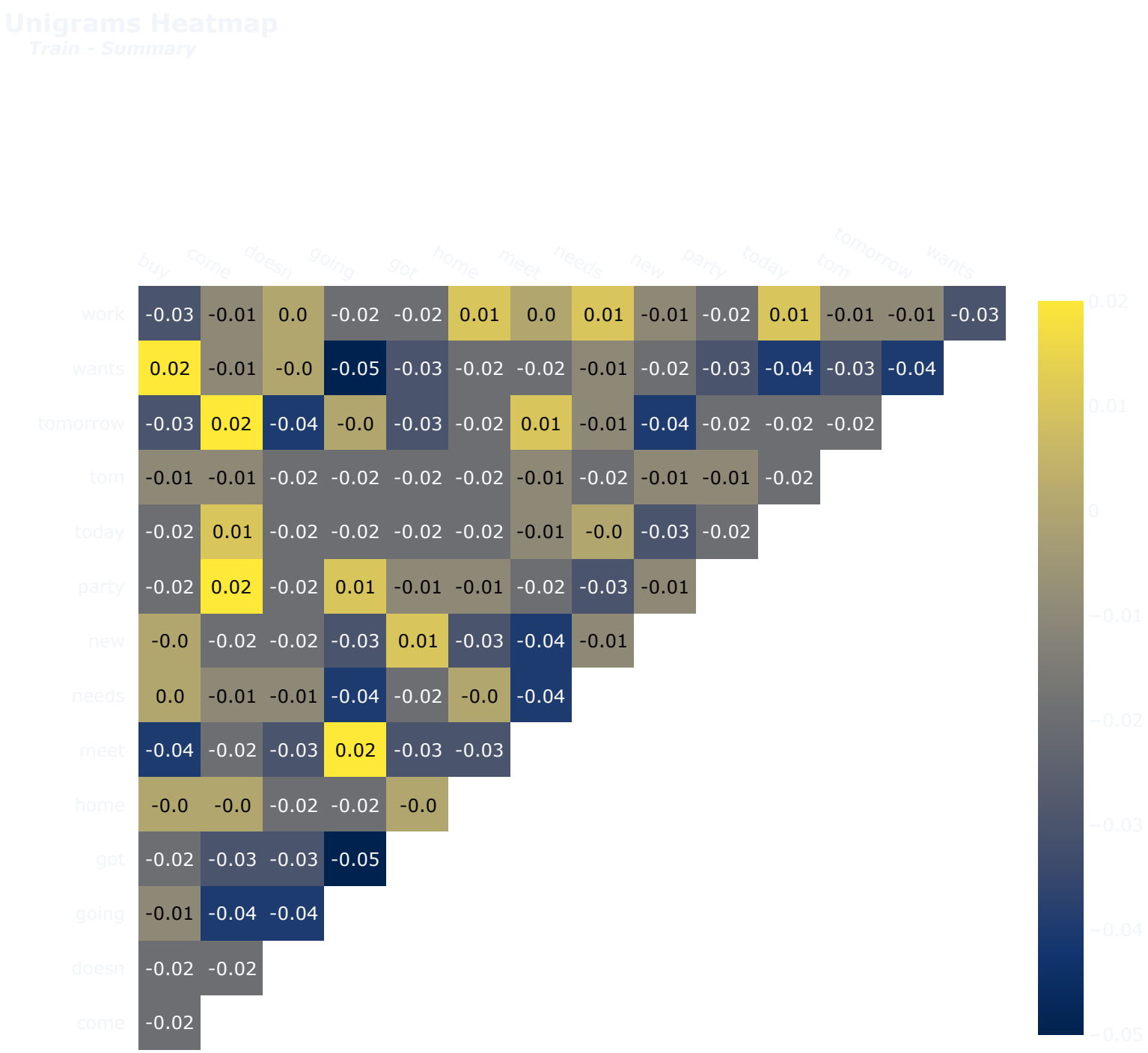
It is also interesting to notice a negative correlation — although still not extremely significant — between the terms *"yes"* and *"yeah"*. Maybe this happens because it would be redundant to include both in the same dialogue, or perhaps the data captures a tendency of individuals to use *"yeah"* instead of *"yes"* during conversations. These are some hypotheses we can consider when analyzing this type of heatmaps.

Let's perform the same analysis to the summaries.

Hide code

In [13]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english') # Top 15 terms
x = vectorizer.fit_transform(train['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Unigrams', 'Train - Summary', 800, 800, 12)
```


Unigrams Heatmap
Train - Summary
buy come doesn going got home meet needs new party today tom tomorrow wants
work -0.03 -0.01 0.0 -0.02 -0.02 0.01 0.0 0.01 -0.01 -0.02 0.01 -0.01 -0.01 -0.03
wants 0.02 -0.01 -0.0 -0.05 -0.03 -0.02 -0.02 -0.01 -0.02 -0.03 -0.04 -0.03 -0.04
tomorrow -0.03 0.02 -0.04 -0.0 -0.03 -0.02 0.01 -0.01 -0.04 -0.02 -0.02 -0.02
tom -0.01 -0.01 -0.02 -0.02 -0.02 -0.02 -0.01 -0.02 -0.01 -0.01 -0.02
today -0.02 0.01 -0.02 -0.02 -0.02 -0.02 -0.01 -0.0 -0.03 -0.02
party -0.02 0.02 -0.02 0.01 -0.01 -0.01 -0.02 -0.03 -0.01
new -0.0 -0.02 -0.02 -0.03 0.01 -0.03 -0.04 -0.01
needs 0.0 -0.01 -0.01 -0.04 -0.02 -0.0 -0.04
meet -0.04 -0.02 -0.03 0.02 -0.03 -0.03
home -0.0 -0.0 -0.02 -0.02 -0.0
got -0.02 -0.03 -0.03 -0.05
going -0.01 -0.04 -0.04
doesn -0.02 -0.02
come -0.02
0.02
0.01
0
−0.01
−0.02
−0.03
−0.04
−0.05

The correlations of terms in summaries seem to be more pronounced than those in dialogues, even though these correlations are still not strong. This suggests that summaries may convey relevant information more succinctly than full dialogues, which is exactly the idea behind a summary.

We have positively correlated pairs such as *"going"* and *"meet"*, *"come"* and *"party"*, as well as *"buy"* and *"wants"*. It makes perfect sense to see these unigrams appearing together across texts.

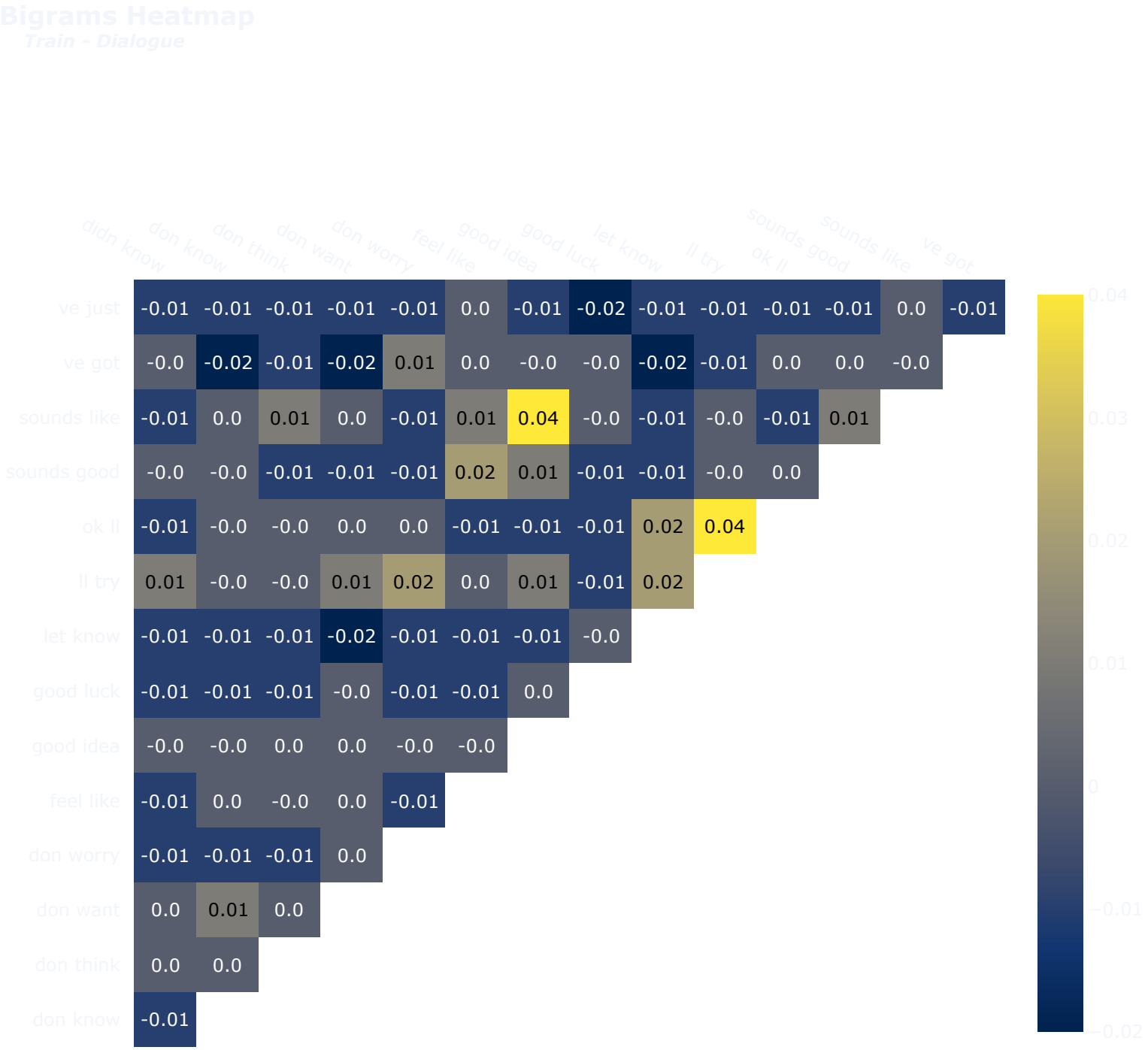
Conversely, it's reasonable for negatively correlated pairs **not** to co-occur frequently in texts, such as *"going"* and *"wants"*, and *"going"* and *"got"*.

Let's now analyze bigrams across dialogues and summaries.

Hide code

In [14]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (2,2)) # Top 15 te
rms
x = vectorizer.fit_transform(train['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Bigrams', 'Train - Dialogue', 800, 800, 12)
```

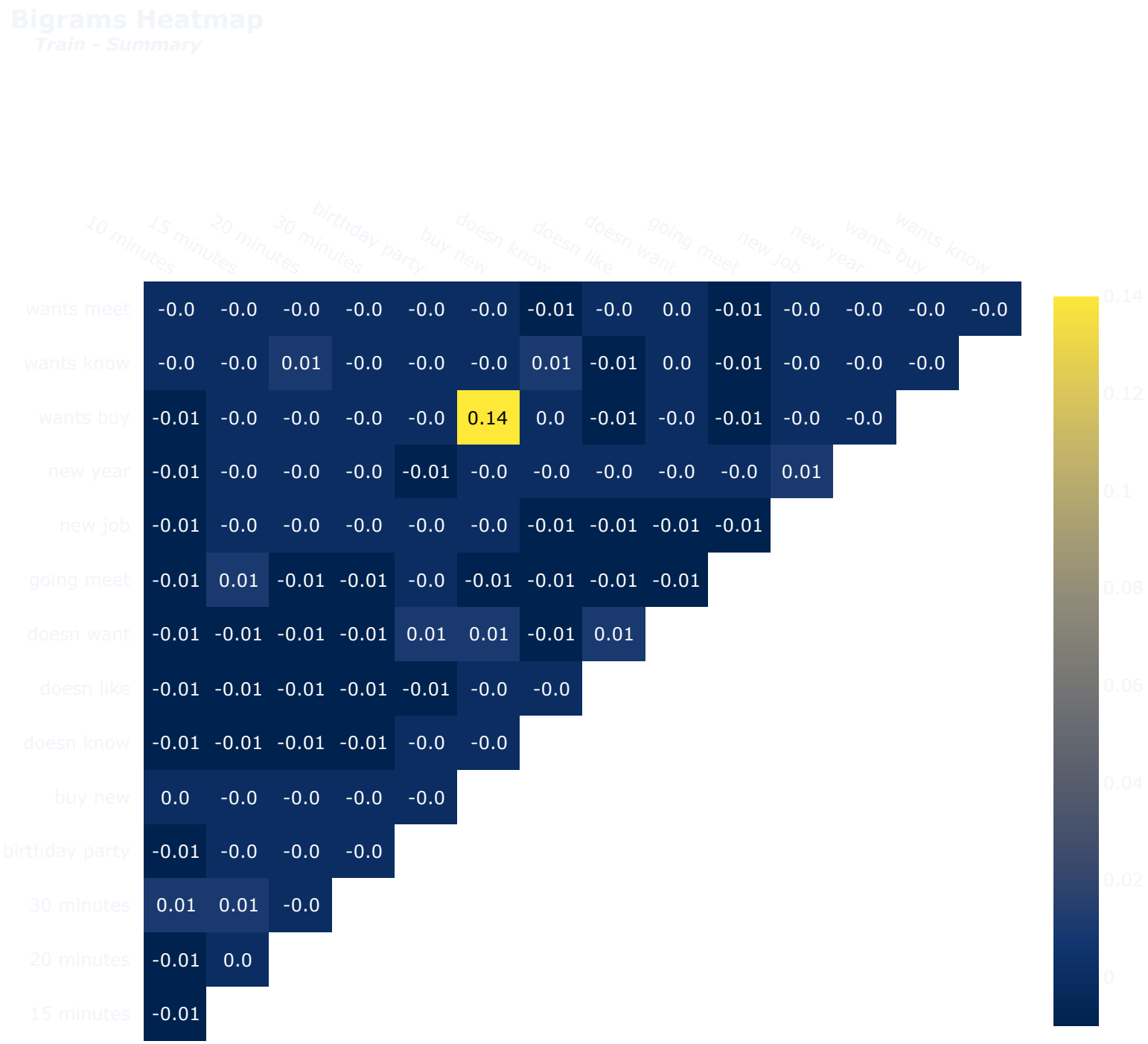

Bigrams Heatmap
Train - Dialogue
didn know
don know
don think
don want
don worry
feel like
good idea
good luck
let know
ll try
ok ll
sounds good
sounds like
ve got
ve just
-0.01 -0.01 -0.01 -0.01 -0.01 0.0 -0.01 -0.02 -0.01 -0.01 -0.01 -0.01 0.0 -0.01
-0.0 -0.02 -0.01 -0.02 0.01 0.0 -0.0 -0.0 -0.02 -0.01 0.0 0.0 -0.0
-0.01 0.0 0.01 0.0 -0.01 0.01 0.04 -0.0 -0.01 -0.0 -0.01 0.01
-0.0 -0.0 -0.01 -0.01 -0.01 0.02 0.01 -0.01 -0.01 -0.0 0.0
-0.01 -0.0 -0.0 0.0 0.0 -0.01 -0.01 -0.01 0.02 0.04
0.01 -0.0 -0.0 0.01 0.02 0.0 0.01 -0.01 0.02
-0.01 -0.01 -0.01 -0.02 -0.01 -0.01 -0.01 -0.0
-0.01 -0.01 -0.01 -0.0 -0.01 -0.01 0.0
-0.0 -0.0 0.0 0.0 -0.0 -0.0
-0.01 0.0 -0.0 0.0 -0.01
-0.01 -0.01 -0.01 0.0
0.0 0.01 0.0
0.0 0.0
-0.01
0.04
0.03
0.02
0.01
0
−0.01
−0.02

Once more, the correlations are not extremely strong. Still, we can see some pairs that seem reasonable to be together, such as *"good idea"* and *"sounds like"*.

Hide code

In [15]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (2,2)) # Top 15 te
rms
x = vectorizer.fit_transform(train['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Bigrams', 'Train - Summary', 800, 800, 12)
```


Bigrams Heatmap
Train - Summary
10 minutes
15 minutes
20 minutes
30 minutes
birthday party
buy new
doesn know
doesn like
doesn want
going meet
new job
new year
wants buy
wants know
wants meet
wants know
wants buy
new year
new job
going meet
doesn want
doesn like
doesn know
buy new
birthday party
30 minutes
20 minutes
15 minutes
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01 -0.0 0.0 -0.01 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 0.01 -0.0 -0.0 -0.0 0.01 -0.01 0.0 -0.01 -0.0 -0.0 -0.0
-0.01 -0.0 -0.0 -0.0 -0.0 0.14 0.0 -0.01 -0.0 -0.01 -0.0 -0.0
-0.01 -0.0 -0.0 -0.0 -0.01 -0.0 -0.0 -0.0 -0.0 -0.0 0.01
-0.01 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01 -0.01 -0.01 -0.01
-0.01 0.01 -0.01 -0.01 -0.0 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.01 0.01 0.01 -0.01 0.01
-0.01 -0.01 -0.01 -0.01 -0.01 -0.0 -0.0
-0.01 -0.01 -0.01 -0.01 -0.0 -0.0
0.0 -0.0 -0.0 -0.0 -0.0
-0.01 -0.0 -0.0 -0.0
0.01 0.01 -0.0
-0.01 0.0
-0.01
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0

We have only one correlation between the pairs *"wants buy"* and *"buy new"*. The other terms do not appear to have any kind of correlation at all.

It is interesting to see the tendency of the summaries to contain information on minutes, which does not seem to be present in the dialogues. We can even investigate further this relationship by querying some summaries where the bigram *15 minutes* appears in the summary.

Hide code

In [26]:

```
# Filtering dataset to see those containing the term '15 minutes' in the summary
filtered_train = train[train['summary'].str.contains('15 minutes', case=False, na=False)]
filtered_train.head()
```

Out[26]:

	id	dialogue	summary
136	13827893	Kate: I'm here <file_other>\r\nKate: there was no place in Red Lion\r\nSteven: hey! but it's quite far away\r\nKate: c'mon it's just 10 min by bike!\r\nSteven: yes, but I'm not by bike\r\nKate: car? \r\nSteven: nope\r\nSteven: by foot :P :P \r\nSteven: anyway google maps says 15 min and I'm there:D\r\nKate: ok, w8in ^^	Kate will meet with Steven in 15 minutes.
428	13811484-1	Jenny: Let's go out to eat.\r\nLucy: That sounds like fun.\r\nJenny: Where do you wanna go? \r\nLucy: Let me think a minute.\r\nJenny: I feel like Chinese.\r\nLucy: That sounds yummy.\r\nJenny: I know a good Chinese restaurant.\r\nLucy: How far away is it?\r\nJenny: It's only 10 minutes from my place.\r\nLucy: Do we have to book a table?\r\nJenny: Oh, no. We can walk right in.\r\nLucy: Cool. Will be in 15 minute. I'm really hungry!	Jenny and Lucy are going to a Chinese restaurant to eat. They do not need to book a table. Lucy will be at Jenny's in 15 minutes.
570	13818296	Danielle: hey where RU?\r\nJuan: I told u I'd be late!\r\nDanielle: but it's been almost 45 mins!\r\nDanielle: <file_gif>\r\nJuan: I'll be there in 15 minutes\r\nJuan: <file_gif>	Juan is almost 45 minutes late. He'll be there in 15 minutes.
1213	13682296-1	John: I know you will be outraged but I like to provoke you :P\r\nTyre: What is it?\r\nJohn: I talked to our neighbour today and I am really starting to think that religious people are just stupid.\r\nTyre: Gosh. You know it's a stupid claim.\r\nJohn: I know that there are some clever, religious individuals. But statistically religious people are stupid.\r\nTyre: It's not true. There are stupid religious people and clever ones, just like atheists.\r\nJohn: But most of academics are not religious.\r\nTyre: How do you know it?\r\nJohn: Experience but also some data I've seen.\r\nTyre: It's just not true.\r\nJohn: They are mostly people believing in things that have nothing to do with logic or reason: miracles, ghosts, witchcraft, just as our neighbour.\r\nTyre: I think it's only one part of them. There are theologians, people who actually know a lot about philosophy, logic etc.\r\nJohn: Yes, there are also people doing "scientifically" tarot, horoscopes and astrology.\r\nTyre: You ca...	John and Tyre's neighbour stopped John in the staircase and talked about some miracles for 15 minutes. John thinks that religious people are stupid. Tyre disagrees with this generalization.
1812	13820691	Madge: are you alive? xD\r\nDorothy: i'm still drunk\r\nMadge: xDDDDDDDD jeeez\r\nFelicia: I don't know...how much did i drink?\r\nMadge: like 10 rounds\r\nFelicia: SHIT \r\nFelicia: you gotta be kidding me ahahaha xDDDDDDDDDDDDDDDDDD\r\nDorothy: of course she is\r\nDorothy: it was at least 15\r\nFelicia: ;_____;\r\nFelicia: was nice to meet you girls...shame on me as always\r\nDorothy: oh stop talking\r\nDorothy: just live the moment B-)\r\nFelicia: how am i supossed to live the moment if i don't remember the half o the night XD\r\nDorothy: well it happens :p \r\nMadge: we gotta repeat it, i had a lot of fun :D\r\nDorothy: i'm in. in 15min?\r\nFelicia: you're crazy ;-;	Dorothy is still intoxicated after at least 15 rounds of drink yesterday and can't remember much of what happened. She would like to meet her friends for a drink again in 15 minutes.

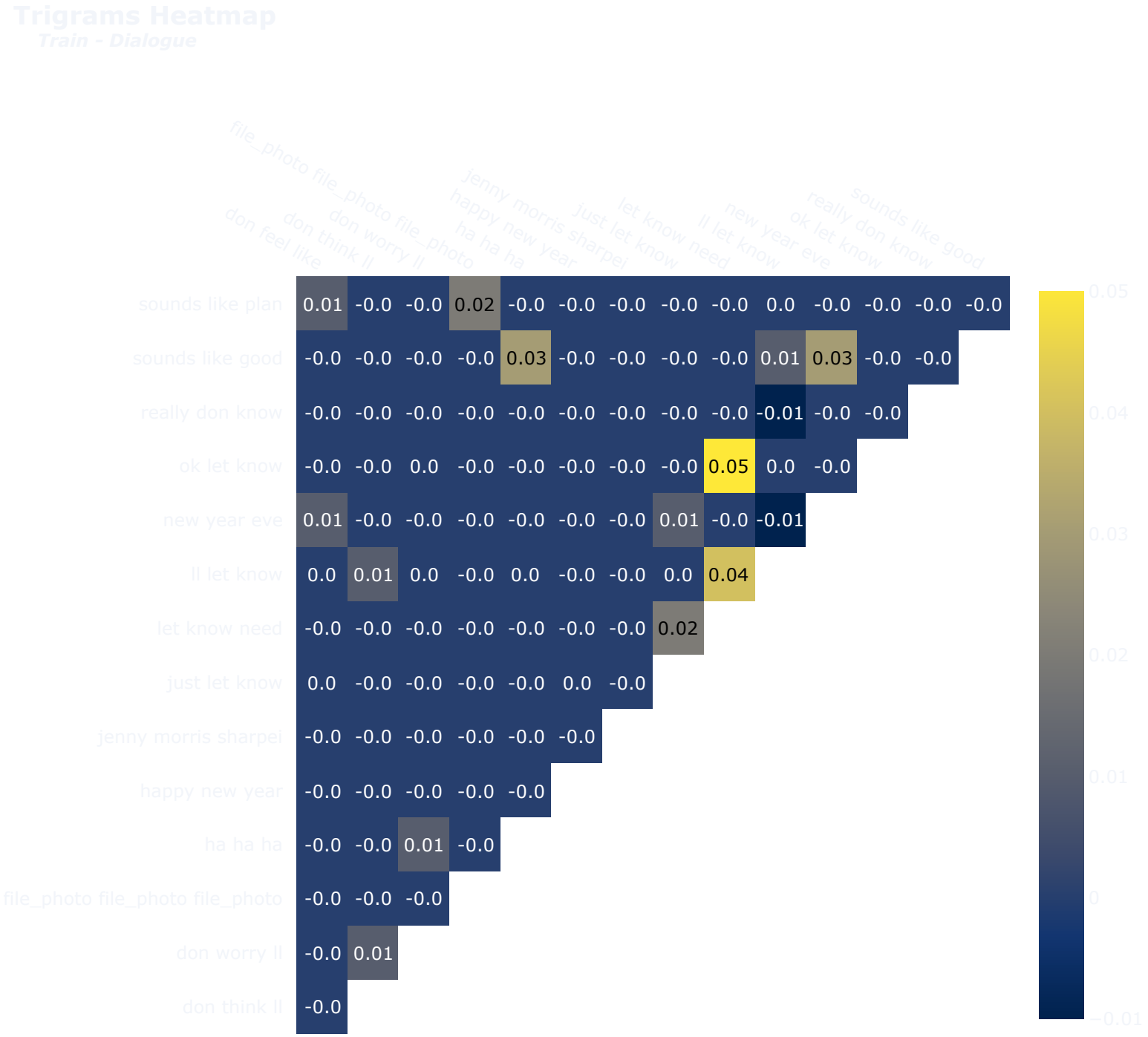
The last row gives us an idea of why we see so many terms related to minutes in summaries, but not in dialogues. In dialogues, people may write "15min" together or even other forms of it, such as "15m", whereas the summaries give us a patternized description, making it natural to be more prominent than other forms to describe time.

Let's now visualize the trigrams.

Hide code

In [16]:

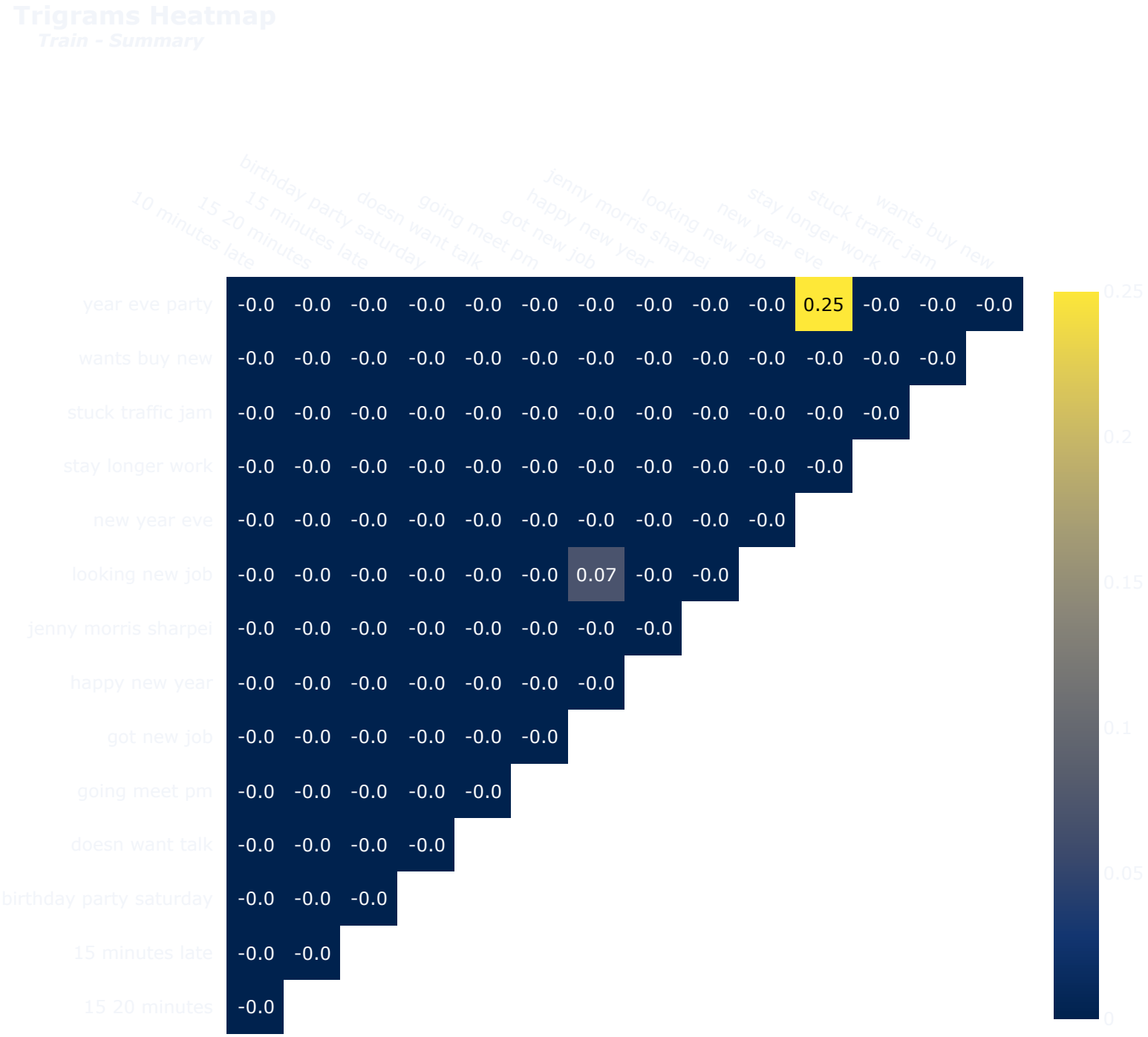
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (3,3)) # Top 15 te
rms
x = vectorizer.fit_transform(train['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Trigrams', 'Train - Dialogue', 800, 800, 12)
```


Trigrams Heatmap
Train - Dialogue
file_photo file_photo file_photo
don worry ll
don think ll
don feel like
happy new year
ha ha ha
jenny morris sharpei
just let know
let know need
ll let know
new year eve
ok let know
really don know
sounds like good
sounds like plan
0.01 -0.0 -0.0 0.02 -0.0 -0.0 -0.0 -0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 0.03 -0.0 -0.0 -0.0 -0.0 0.01 0.03 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01 -0.0 -0.0
-0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.05 0.0 -0.0
0.01 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.01 -0.0 -0.01
0.0 0.01 0.0 -0.0 0.0 -0.0 -0.0 0.0 0.04
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.02
0.0 -0.0 -0.0 -0.0 -0.0 0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 0.01 -0.0
-0.0 -0.0 -0.0
-0.0 0.01
-0.0
0.05
0.04
0.03
0.02
0.01
0
−0.01

Hide code

In [17]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (3,3)) # Top 15 te
rms
x = vectorizer.fit_transform(train['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Trigrams', 'Train - Summary', 800, 800, 12)
```


Trigrams Heatmap
Train - Summary
10 minutes late
15 20 minutes
15 minutes late
birthday party saturday
doesn want talk
going meet pm
got new job
happy new year
jenny morris sharpei
looking new job
new year eve
stay longer work
stuck traffic jam
wants buy new
year eve party
wants buy new
stuck traffic jam
stay longer work
new year eve
looking new job
jenny morris sharpei
happy new year
got new job
going meet pm
doesn want talk
birthday party saturday
15 minutes late
15 20 minutes
0.25
0.2
0.15
0.1
0.05
0

Once more, we can see that the terms are not strongly correlated. But still, it is possible to see pairs that seem logical to appear together in the corpus.

I will now perform the exact same analysis on the `test` and `val` datasets. We can expect the same behavior as the ones seen during the analysis of the training set, which is why I will refrain from commenting on the following plots to avoid redundancy. However, if something different appears, we will surely investigate further.

Test Dataset

In [29]:

```
# Extracting info on the test dataset
describe_df(test)
```

```
DataFrame shape: (819, 3)

819 samples

3 attributes

Missing Data:
id          0
dialogue    0
summary     0
dtype: int64

Duplicates: 0

Data Types:
id          object
dialogue    object
summary     object
dtype: object

Categorical Features:
id, dialogue, summary

Continuous Features:
None

Binary Features:
None
```

DataFrame Head:

	id	dialogue	summary
0	13862856	Hannah: Hey, do you have Betty's number?\nAmanda: Lemme check\nHannah: <file_gif>\nAmanda: Sorry, can't find it.\nAmanda: Ask Larry\nAmanda: He called her last time we were at the park together\nHannah: I don't know him well\nHannah: <file_gif>\nAmanda: Don't be shy, he's very nice\nHannah: If you say so..\nHannah: I'd rather you texted him\nAmanda: Just text him 🙂\nHannah: Urgh.. Alright\nHannah: Bye\nAmanda: Bye bye	Hannah needs Betty's number but Amanda doesn't have it. She needs to contact Larry.
1	13729565	Eric: MACHINE!\r\nRob: That's so gr8!\r\nEric: I know! And shows how Americans see Russian ;)\r\nRob: And it's really funny!\r\nEric: I know! I especially like the train part!\r\nRob: Hahaha! No one talks to the machine like that!\r\nEric: Is this his only stand-up?\r\nRob: Idk. I'll check.\r\nEric: Sure.\r\nRob: Turns out no! There are some of his stand-ups on youtube.\r\nEric: Gr8! I'll watch them now!\r\nRob: Me too!\r\nEric: MACHINE!\r\nRob: MACHINE!\r\nEric: TTYL?\r\nRob: Sure :)	Eric and Rob are going to watch a stand-up on youtube.
2	13680171	Lenny: Babe, can you help me with something?\r\nBob: Sure, what's up?\r\nLenny: Which one should I pick?\r\nBob: Send me photos\r\nLenny: <file_photo>\r\nLenny: <file_photo>\r\nLenny: <file_photo>\r\nBob: I like the first ones best\r\nLenny: But I already have purple trousers. Does it make sense to have two pairs?\r\nBob: I have four black pairs :D :D\r\nLenny: yeah, but shouldn't I pick a different color?\r\nBob: what matters is what you'll give you the most outfit options\r\nLenny: So I guess I'll buy the first or the third pair then\r\nBob: Pick the best quality then\r\nLenny: ur right, thx\r\nBob: no prob :)	Lenny can't decide which trousers to buy. Bob advised Lenny on that topic. Lenny goes with Bob's advice to pick the trousers that are of best quality.
3	13729438	Will: hey babe, what do you want for dinner tonight?\r\nEmma: gah, don't even worry about it tonight\r\nWill: what do you mean? everything ok?\r\nEmma: not really, but it's ok, don't worry about cooking though, I'm not hungry\r\nWill: Well what time will you be home?\r\nEmma: soon, hopefully\r\nWill: you sure? Maybe you want me to pick you up?\r\nEmma: no no it's alright. I'll be home soon, i'll tell you when I get home. \r\nWill: Alright, love you. \r\nEmma: love you too.	Emma will be home soon and she will let Will know.
4	13828600	Ollie: Hi , are you in Warsaw\r\nJane: yes, just back! Btw are you free for diner the 19th?\r\nOllie: nope!\r\nJane: and the 18th?\r\nOllie: nope, we have this party and you must be there, remember? \r\nJane: oh right! i lost my calendar.. thanks for reminding me\r\nOllie: we have lunch this week? \r\nJane: with pleasure!\r\nOllie: friday?\r\nJane: ok\r\nJane: what do you mean " we don't have any more whisky!" lol..\r\nOllie: what!!!\r\nJane: you just call me and the all thing i heard was that sentence about whisky... what's wrong with you?\r\nOllie: oh oh... very strange! i have to be carefull may be there is some spy in my mobile! lol\r\nJane: dont' worry, we'll check on friday.\r\nOllie: don't forget to bring some sun with you\r\nJane: I can't wait to be in Morocco..\r\nOllie: enjoy and see you friday\r\nJane: sorry Ollie, i'm very busy, i won't have time for lunch tomorrow, but may be at 6pm after my courses?this trip to Morocco was so nice, but time consuming!\r\nOllie: ok fo...	Jane is in Warsaw. Ollie and Jane has a party. Jane lost her calendar. They will get a lunch this week on Friday. Ollie accidentally called Jane and talked about whisky. Jane cancels lunch. They'll meet for a tea at 6 pm.

DataFrame Tail:

	id	dialogue	summary
814	13611902-1	Alex: Were you able to attend Friday night's basketball game?\r\nBenjamin: I was unable to make it.\r\nAlex: You should have been there. It was intense.\r\nBenjamin: Is that right. Who ended up winning?\r\nAlex: Our team was victorious.\r\nBenjamin: I wish I was free that night. I'm kind of mad that I didn't go.\r\nAlex: It was a great game. Everything alright tough? \r\nBenjamin: Yeah man thanks for asking, it's just that my mom is sick and I am taking care of her.\r\nAlex: Oh sorry to hear that. Hope she makes a fast recovery 💪\r\nBenjamin: She will, she just has a nasty flu but she will be alright :D\r\nAlex: Glad to hear that!\r\nBenjamin: What was the score at the end of the game?\r\nAlex: Our team won 101-98.\r\nBenjamin: Sounds like it was a close game then.\r\nAlex: That's the reason it was such a great game.\r\nBenjamin: I'll go to the next one for sure.\r\nAlex: It's next weekend so you better put on your calendar ahaha\r\nBenjamin: ahaha I will I will. Talk to you later!\...	Benjamin didn't come to see a basketball game on Friday's night. The team supported by Alex won 101-98. Benjamin's mom has a flu and he's looking after her. Benjamin declares to attend the next basketball match.
815	13820989	Jamilla: remember that the audition starts at 7.30 P.M.\r\nKiki: which station?\r\nJamilla: Antena 3\r\nYoyo: roger that	The audition starts at 7.30 P.M. in Antena 3.
816	13717193	Marta: <file_gif>\r\nMarta: Sorry girls, I clicked something by accident :D\r\nAgnieszka: No problem :p\r\nWeronika: Hahaha\r\nAgnieszka: Good thing you didn't send something from your gallery ;)	Marta sent a file accidentally,
817	13829115	Cora: Have you heard how much fuss British media made about meet and greet with James Charles in Birmingham?\r\nEllie: no...! what happened?\r\nCora: Well, there was a meet and greet with James Charles in one of the malls in Birmingham and about 8000 fans showed up for it.\r\nCora: It cause a gridlock around the mall and - of course - British media had to make some (quite negative) comments on it.\r\nEllie: they came for sister James?! >:(\r\nEllie: i sister snapped!! :p :D\r\nCora: Haha :D\r\nCora: You shouldn't watch so much youtube, you're getting weirder and weirder. :d\r\nEllie: sister shut up :P so, what did they say?\r\nCora: ;) :* "Daily Mail" was surprised that a meet and greet with a "virtually unknown" youtuber gathered 8000 people. :p\r\nCora: A host from LBC tried to find an answer to an unanswerable question: "Who is James Charles?". Eventually James called him and introduced himself. On air. :D\r\nEllie: there's something called google lol\r\nCora: Right? :p\r\nCora:...	There was a meet-and-greet with James Charles in Birmingham which gathered 8000 people.
818	13818810	Rachel: <file_other>\r\nRachel: Top 50 Best Films of 2018\r\nRachel: :)\r\nJanice: Omg, I've watched almost all 50... xDD\r\nSpencer: Hahah, Deadpool 2 also??\r\nJanice: Yep\r\nSpencer: Really??\r\nJanice: My bf forced me to watch it xD\r\nRachel: Hahah\r\nJanice: It wasn't that bad\r\nJanice: I thought it'd be worse\r\nRachel: And Avengers? :D\r\nJanice: 2 times\r\nRachel: Omg\r\nJanice: xP\r\nRachel: You are the best gf in the world\r\nRachel: Your bf should appreciate that ;-)\r\nJanice: He does\r\nJanice: x)	Rachel sends a list of Top 50 films of 2018. Janice watched almost half of them, Deadpool 2 and Avengers included.

Hide code

In [30]:

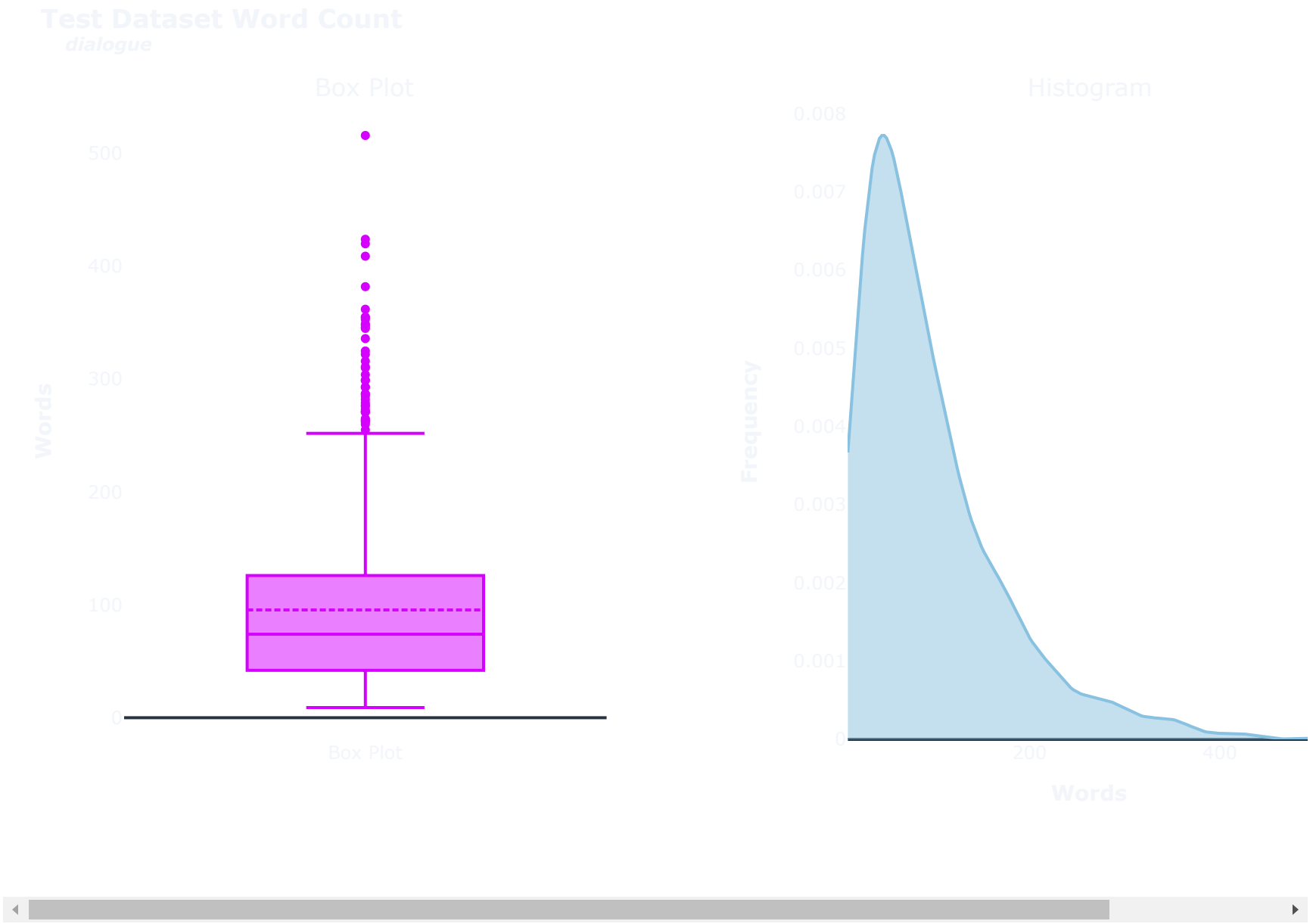
```
# Removing 'Id' from categorical features list
categorical_features.remove('id')
```

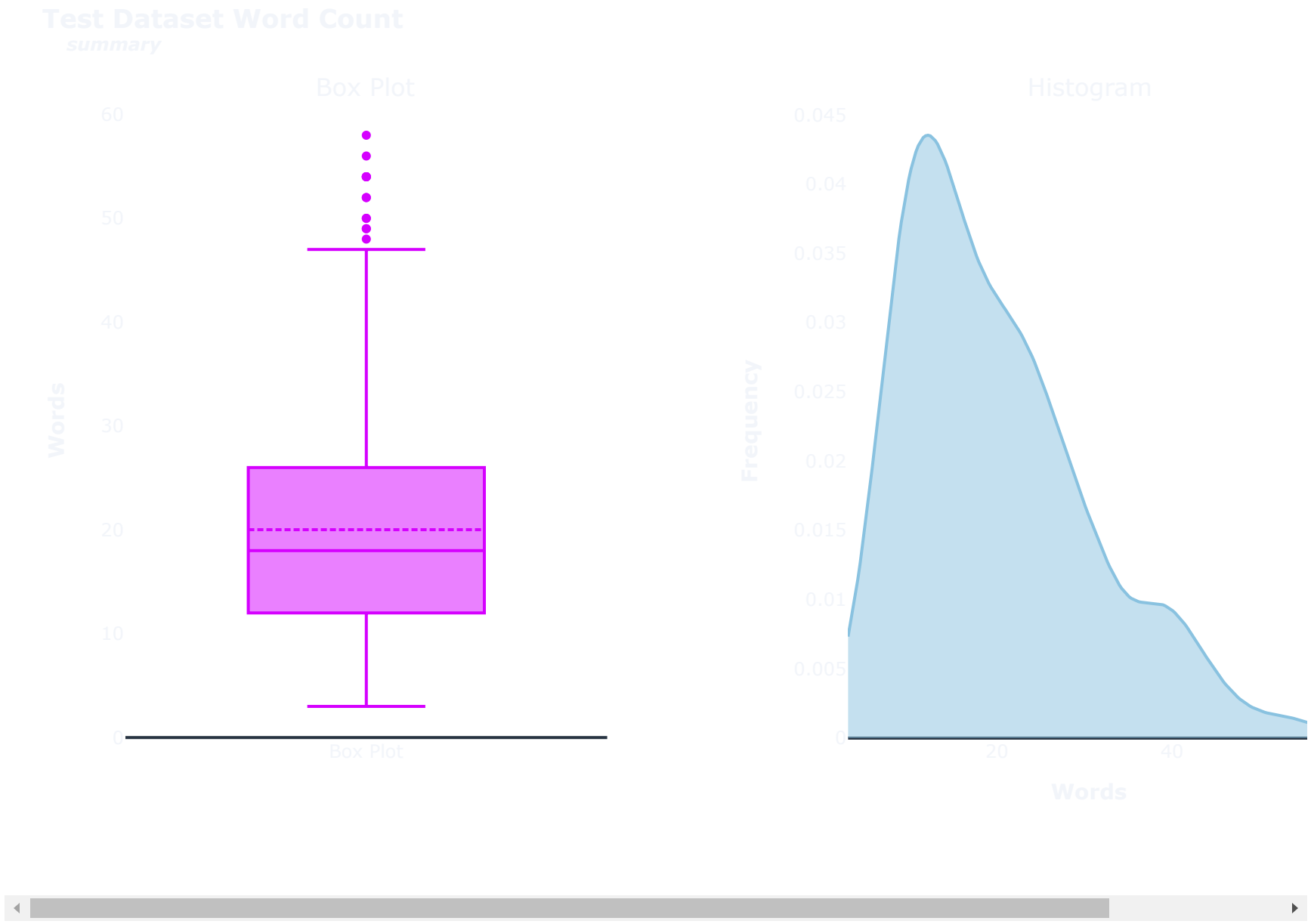
Hide code

In [31]:

```
df_text_lenght = pd.DataFrame()
for feat in categorical_features:
    df_text_lenght[feat] = test[feat].apply(lambda x: len(str(x).split()))

histogram_boxplot(df_text_lenght,'#89c2e0', '#d500ff', 600, 1000, True, 'Test Dataset')
```

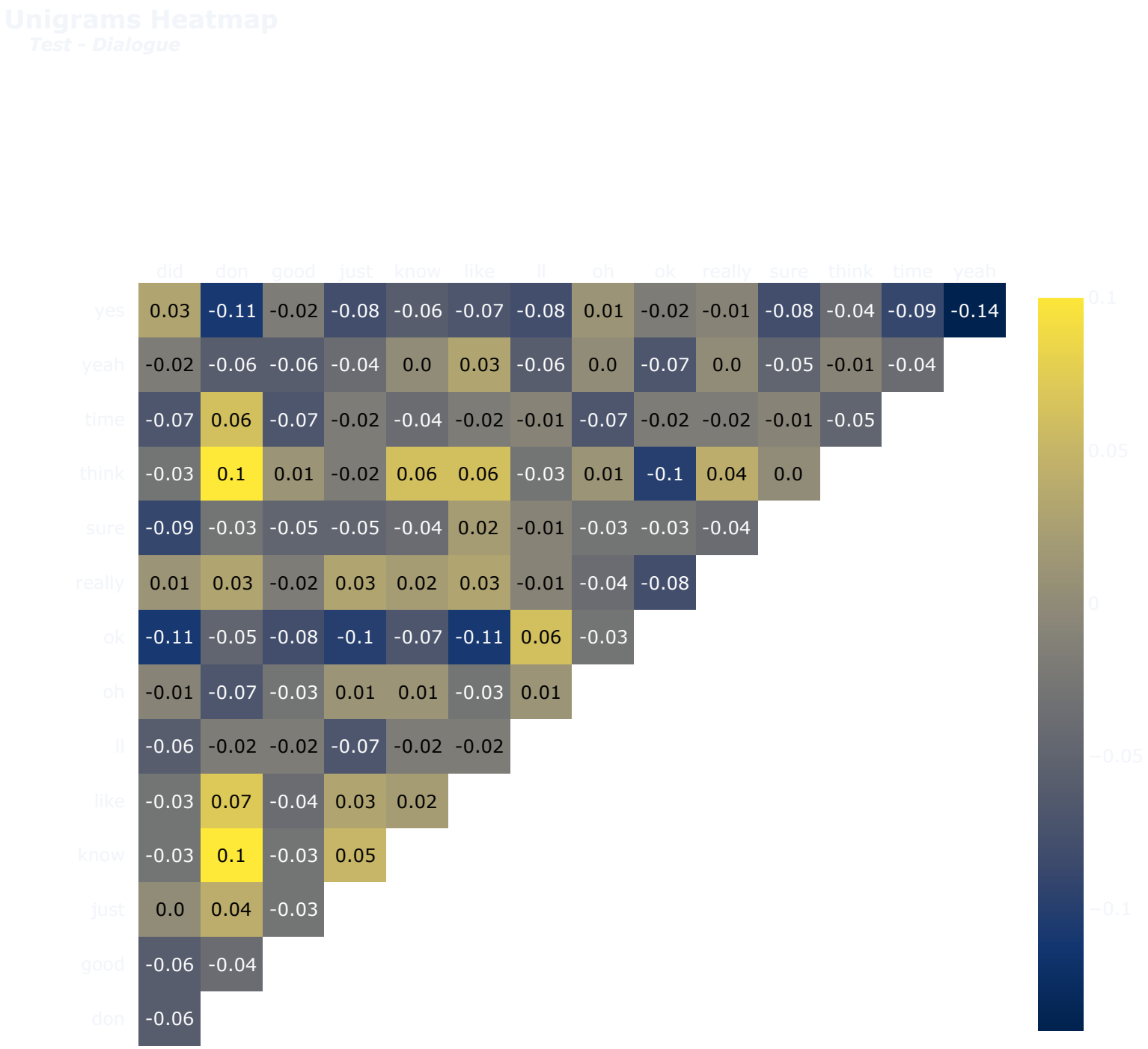

Test Dataset Word Count
dialogue
Box Plot
Words
500
400
300
200
100
0
Box Plot
Histogram
Frequency
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
200
400
Words


Test Dataset Word Count
summary
Box Plot
Words
60
50
40
30
20
10
0
Box Plot
Histogram
Frequency
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
20
40
Words

Hide code

In [32]:

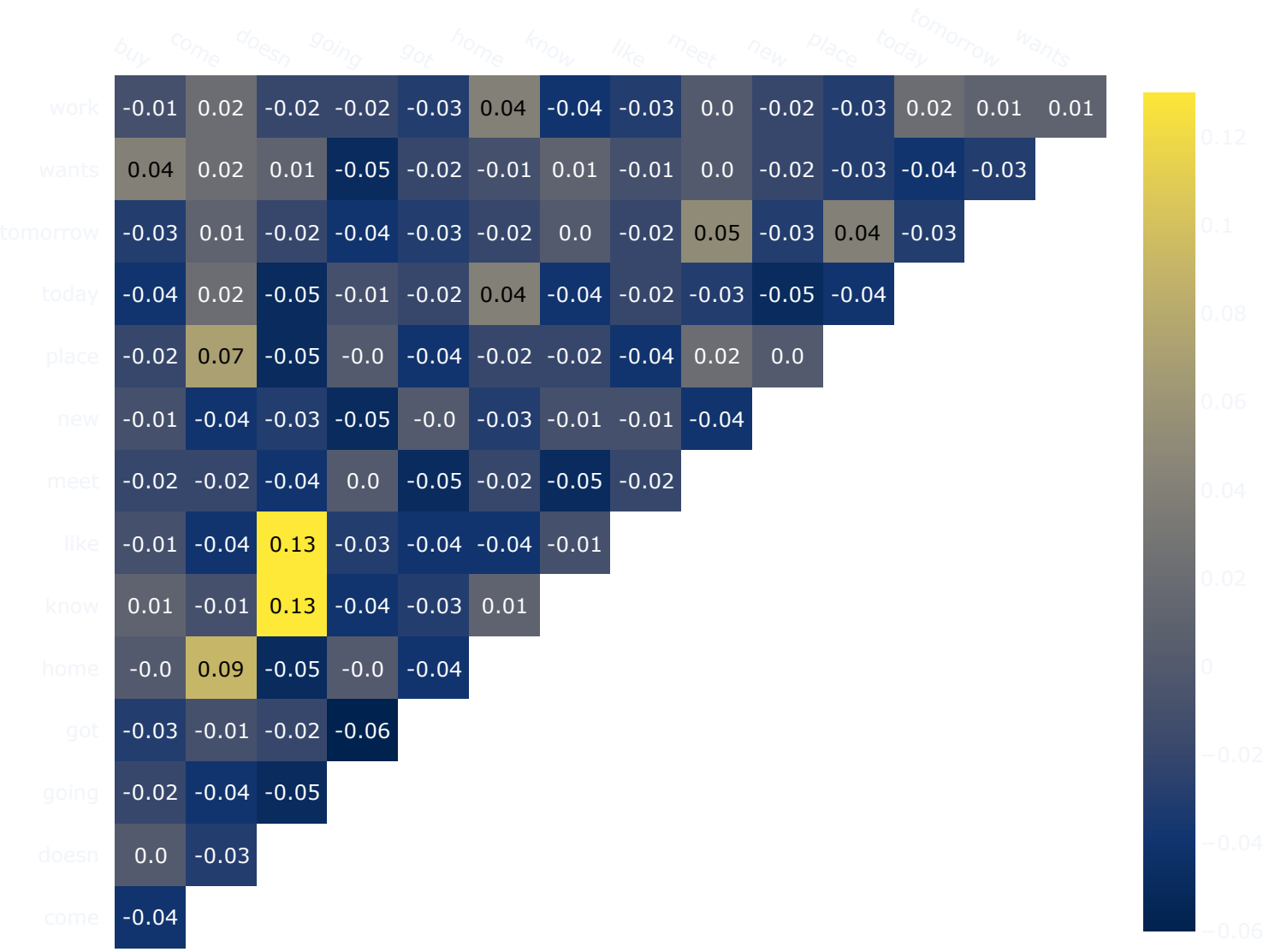
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english') # Top 15 terms
x = vectorizer.fit_transform(test['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Unigrams', 'Test - Dialogue', 800, 800, 12)
```


Unigrams Heatmap
Test - Dialogue
did don good just know like ll oh ok really sure think time yeah
yes 0.03 -0.11 -0.02 -0.08 -0.06 -0.07 -0.08 0.01 -0.02 -0.01 -0.08 -0.04 -0.09 -0.14
yeah -0.02 -0.06 -0.06 -0.04 0.0 0.03 -0.06 0.0 -0.07 0.0 -0.05 -0.01 -0.04
time -0.07 0.06 -0.07 -0.02 -0.04 -0.02 -0.01 -0.07 -0.02 -0.02 -0.01 -0.05
think -0.03 0.1 0.01 -0.02 0.06 0.06 -0.03 0.01 -0.1 0.04 0.0
sure -0.09 -0.03 -0.05 -0.05 -0.04 0.02 -0.01 -0.03 -0.03 -0.04
really 0.01 0.03 -0.02 0.03 0.02 0.03 -0.01 -0.04 -0.08
ok -0.11 -0.05 -0.08 -0.1 -0.07 -0.11 0.06 -0.03
oh -0.01 -0.07 -0.03 0.01 0.01 -0.03 0.01
ll -0.06 -0.02 -0.02 -0.07 -0.02 -0.02
like -0.03 0.07 -0.04 0.03 0.02
know -0.03 0.1 -0.03 0.05
just 0.0 0.04 -0.03
good -0.06 -0.04
don -0.06
0.1
0.05
0
−0.05
−0.1

Hide code

In [33]:

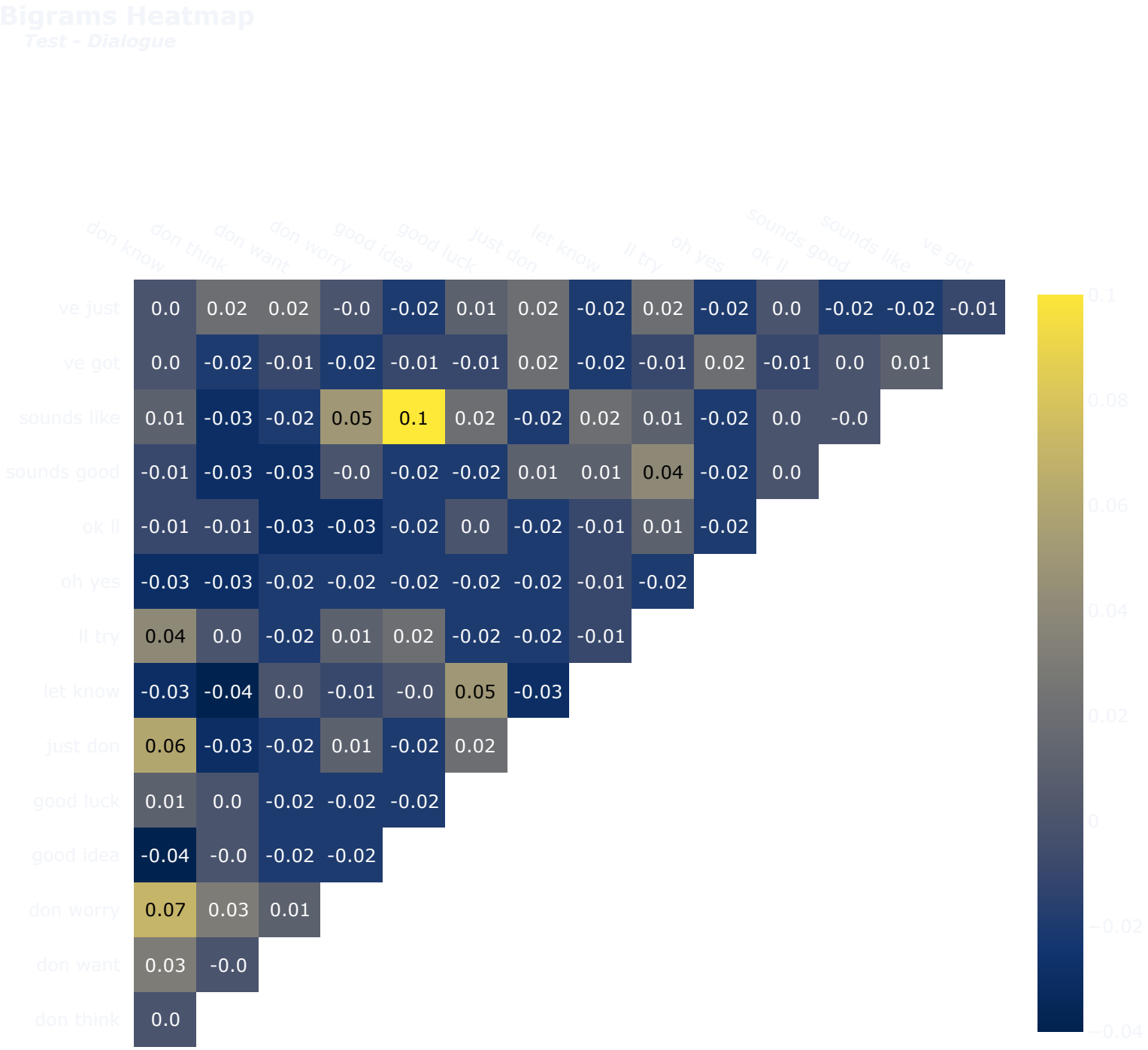
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english') # Top 15 terms
x = vectorizer.fit_transform(test['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Unigrams', 'Test - Summary', 800, 800, 12)
```


Unigrams Heatmap
Test - Summary
buy come doesn going got home know like meet new place today tomorrow wants
work -0.01 0.02 -0.02 -0.02 -0.03 0.04 -0.04 -0.03 0.0 -0.02 -0.03 0.02 0.01 0.01
wants 0.04 0.02 0.01 -0.05 -0.02 -0.01 0.01 -0.01 0.0 -0.02 -0.03 -0.04 -0.03
tomorrow -0.03 0.01 -0.02 -0.04 -0.03 -0.02 0.0 -0.02 0.05 -0.03 0.04 -0.03
today -0.04 0.02 -0.05 -0.01 -0.02 0.04 -0.04 -0.02 -0.03 -0.05 -0.04
place -0.02 0.07 -0.05 -0.0 -0.04 -0.02 -0.02 -0.04 0.02 0.0
new -0.01 -0.04 -0.03 -0.05 -0.0 -0.03 -0.01 -0.01 -0.04
meet -0.02 -0.02 -0.04 0.0 -0.05 -0.02 -0.05 -0.02
like -0.01 -0.04 0.13 -0.03 -0.04 -0.04 -0.01
know 0.01 -0.01 0.13 -0.04 -0.03 0.01
home -0.0 0.09 -0.05 -0.0 -0.04
got -0.03 -0.01 -0.02 -0.06
going -0.02 -0.04 -0.05
doesn 0.0 -0.03
come -0.04
0.12
0.1
0.08
0.06
0.04
0.02
0
−0.02
−0.04
−0.06

Hide code

In [18]:

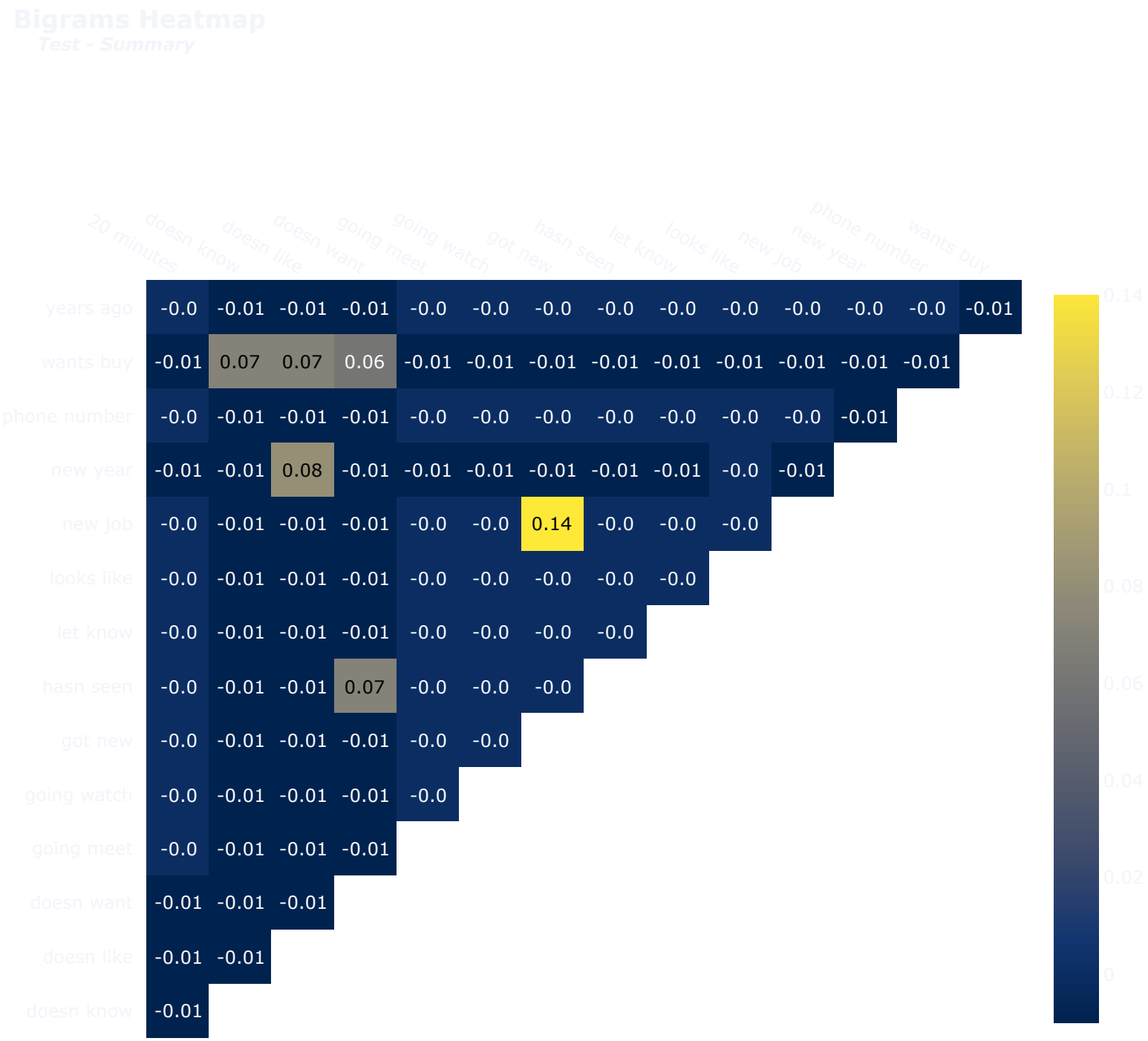
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (2,2)) # Top 15 te
rms
x = vectorizer.fit_transform(test['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Bigrams', 'Test - Dialogue', 800, 800, 12)
```


Bigrams Heatmap
Test - Dialogue
don know
don think
don want
don worry
good idea
good luck
just don
let know
ll try
oh yes
ok ll
sounds good
sounds like
ve got
ve just
ve got
sounds like
sounds good
ok ll
oh yes
ll try
let know
just don
good luck
good idea
don worry
don want
don think
0.0 0.02 0.02 -0.0 -0.02 0.01 0.02 -0.02 0.02 -0.02 0.0 -0.02 -0.02 -0.01
0.0 -0.02 -0.01 -0.02 -0.01 -0.01 0.02 -0.02 -0.01 0.02 -0.01 0.0 0.01
0.01 -0.03 -0.02 0.05 0.1 0.02 -0.02 0.02 0.01 -0.02 0.0 -0.0
-0.01 -0.03 -0.03 -0.0 -0.02 -0.02 0.01 0.01 0.04 -0.02 0.0
-0.01 -0.01 -0.03 -0.03 -0.02 0.0 -0.02 -0.01 0.01 -0.02
-0.03 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 -0.01 -0.02
0.04 0.0 -0.02 0.01 0.02 -0.02 -0.02 -0.01
-0.03 -0.04 0.0 -0.01 -0.0 0.05 -0.03
0.06 -0.03 -0.02 0.01 -0.02 0.02
0.01 0.0 -0.02 -0.02 -0.02
-0.04 -0.0 -0.02 -0.02
0.07 0.03 0.01
0.03 -0.0
0.0
0.1
0.08
0.06
0.04
0.02
0
−0.02
−0.04

Hide code

In [19]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (2,2)) # Top 15 te
rms
x = vectorizer.fit_transform(test['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Bigrams', 'Test - Summary', 800, 800, 12)
```


Bigrams Heatmap
Test - Summary
20 minutes
doesn know
doesn like
doesn want
going meet
going watch
got new
hasn seen
let know
looks like
new job
new year
phone number
wants buy
years ago
wants buy
phone number
new year
new job
looks like
let know
hasn seen
got new
going watch
going meet
doesn want
doesn like
doesn know
-0.0 -0.01 -0.01 -0.01 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01
-0.01 0.07 0.07 0.06 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.0 -0.01 -0.01 -0.01 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01
-0.01 -0.01 0.08 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.0 -0.01
-0.0 -0.01 -0.01 -0.01 -0.0 -0.0 0.14 -0.0 -0.0 -0.0
-0.0 -0.01 -0.01 -0.01 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.01 -0.01 -0.01 -0.0 -0.0 -0.0 -0.0
-0.0 -0.01 -0.01 0.07 -0.0 -0.0 -0.0
-0.0 -0.01 -0.01 -0.01 -0.0 -0.0
-0.0 -0.01 -0.01 -0.01 -0.0
-0.0 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01
-0.01 -0.01
-0.01
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0

Hide code

In [20]:

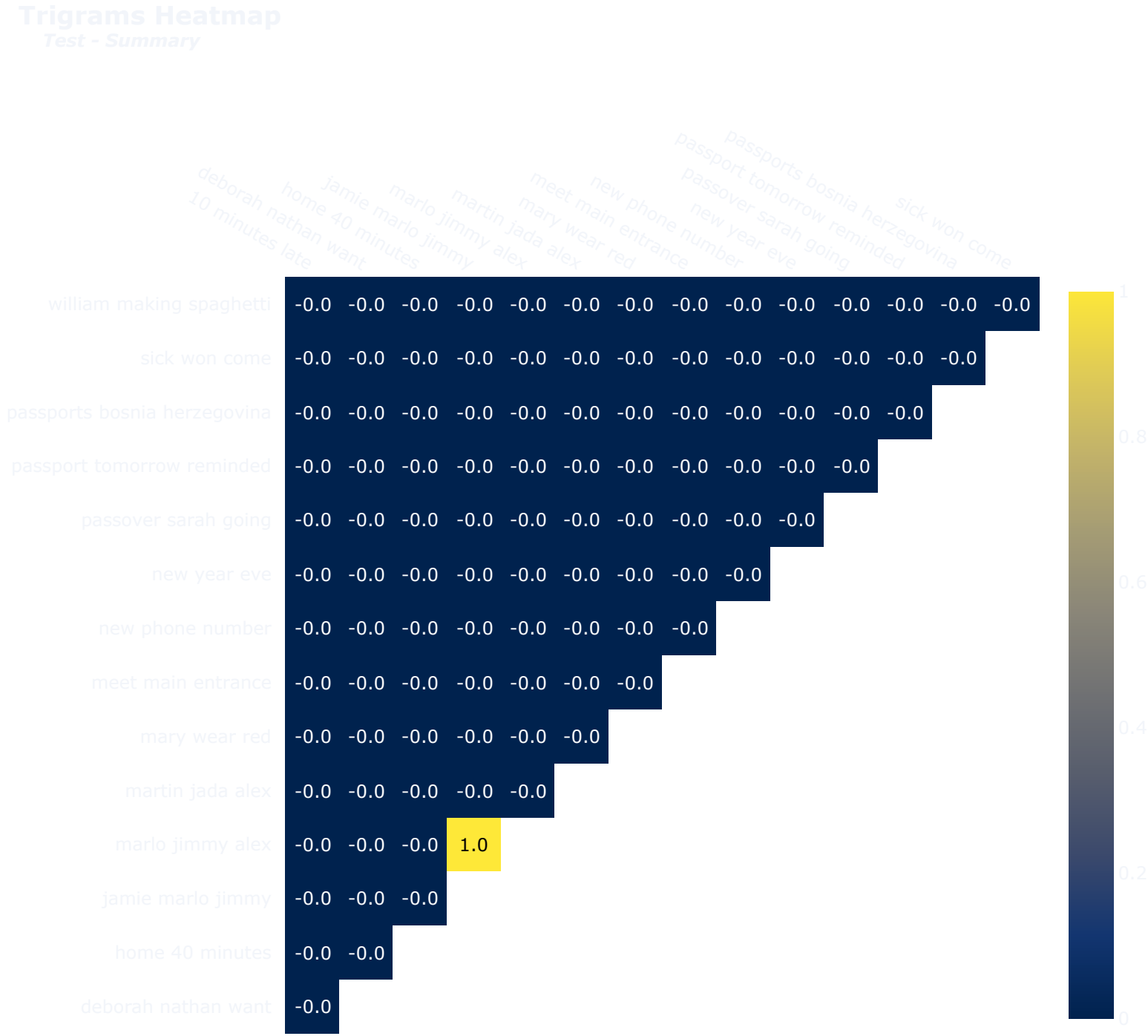
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (3,3)) # Top 15 te
rms
x = vectorizer.fit_transform(test['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Trigrams', 'Test - Dialogue', 800, 800, 12)
```


Trigrams Heatmap
Test - Dialogue
file_photo file_photo file_photo
file_other file_other file_other
aaron file_gif aaron
don know just
kathy file_picture kathy
just don know
laura file_photo laura
let know need
like good idea
ll let know
randal file_photo sidney
sounds good ll
sounds like good
sounds like plan
think let know
sounds like plan
sounds like good
sounds good ll
randal file_photo sidney
ll let know
like good idea
let know need
laura file_photo laura
kathy file_picture kathy
just don know
file_photo file_photo file_photo
file_other file_other file_other
don know just
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01 -0.0 0.17 -0.0
-0.0 -0.0 -0.0 0.14 -0.0 -0.0 -0.0 -0.0 0.7 -0.01 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.01 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.01 -0.01 -0.0 -0.01 -0.0 0.07 -0.01 -0.01
-0.0 -0.0 -0.0 0.1 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0
-0.0 -0.0
-0.0
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Hide code

In [21]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (3,3)) # Top 15 te
rms
x = vectorizer.fit_transform(test['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Trigrams', 'Test - Summary', 800, 800, 12)
```


Trigrams Heatmap
Test - Summary
deborah nathan want
10 minutes late
home 40 minutes
jamie marlo jimmy
marlo jimmy alex
martin jada alex
mary wear red
meet main entrance
new phone number
new year eve
passover sarah going
passport tomorrow reminded
passports bosnia herzegovina
sick won come
william making spaghetti
sick won come
passports bosnia herzegovina
passport tomorrow reminded
passover sarah going
new year eve
new phone number
meet main entrance
mary wear red
martin jada alex
marlo jimmy alex
jamie marlo jimmy
home 40 minutes
deborah nathan want
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 1.0
-0.0 -0.0 -0.0
-0.0 -0.0
-0.0
1
0.8
0.6
0.4
0.2
0

Validation Dataset

In [38]:

```
# Extracting info on the val dataset
describe_df(val)
```

```
DataFrame shape: (818, 3)

818 samples

3 attributes

Missing Data:
id          0
dialogue    0
summary     0
dtype: int64

Duplicates: 0

Data Types:
id          object
dialogue    object
summary     object
dtype: object

Categorical Features:
id, dialogue, summary

Continuous Features:
None

Binary Features:
None
```

```
DataFrame Head:
```

	id	dialogue	summary
0	13817023	A: Hi Tom, are you busy tomorrow’s afternoon?\r\nB: I’m pretty sure I am. What’s up?\r\nA: Can you go with me to the animal shelter?.\r\nB: What do you want to do?\r\nA: I want to get a puppy for my son.\r\nB: That will make him so happy.\r\nA: Yeah, we’ve discussed it many times. I think he’s ready now.\r\nB: That’s good. Raising a dog is a tough issue. Like having a baby ;-) \r\nA: I'll get him one of those little dogs.\r\nB: One that won't grow up too big;-)\r\nA: And eat too much;-)))\r\nB: Do you know which one he would like?\r\nA: Oh, yes, I took him there last Monday. He showed me one that he really liked.\r\nB: I bet you had to drag him away.\r\nA: He wanted to take it home right away ;-).\r\nB: I wonder what he'll name it.\r\nA: He said he’d name it after his dead hamster – Lemmy - he's a great Motorhead fan :-)))	A will go to the animal shelter tomorrow to get a puppy for her son. They already visited the shelter last Monday and the son chose the puppy.
1	13716628	Emma: I’ve just fallen in love with this advent calendar! Awesome! I wanna one for my kids!\r\nRob: I used to get one every year as a child! Loved them! \r\nEmma: Yeah, i remember! they were filled with chocolates!\r\nLauren: they are different these days! much more sophisticated! Haha!\r\nRob: yeah, they can be fabric/ wooden, shop bought/ homemade, filled with various stuff\r\nEmma: what do you fit inside?\r\nLauren: small toys, Christmas decorations, creative stuff, hair bands & clips, stickers, pencils & rubbers, small puzzles, sweets\r\nEmma: WOW! That’s brill! X\r\nLauren: i add one more very special thing as well- little notes asking my children to do something nice for someone else\r\nRob: i like that! My sister adds notes asking her kids questions about christmas such as What did the 3 wise men bring? etc\r\nLauren: i reckon it prepares them for Christmas \r\nEmma: and makes it more about traditions and being kind to other people\r\nLauren: my children get very excited eve...	Emma and Rob love the advent calendar. Lauren fits inside calendar various items, for instance, small toys and Christmas decorations. Her children are excited whenever they get the calendar.
2	13829420	Jackie: Madison is pregnant\r\nJackie: but she doesn't wanna talk about it\r\nIggy: why\r\nJackie: I don't know why because she doesn't wanna talk about it\r\nIggy: ok\r\nJackie: I wanted to prepare you for it because people get super excited and ask lots of questions\r\nJackie: and she looked way more anxious than excited\r\nIggy: she's probably worrying about it\r\nIggy: she's taking every commitment really seriously\r\nJackie: it could be money problems or relationship problems\r\nIggy: or maybe she wants an abortion\r\nJackie: it could be all of the above\r\nIggy: but you know what? \r\nIggy: once my friend was pregnant and I couldn't bring myself to be happy about it\r\nJackie: why?\r\nIggy: I felt they were immature and I couldn't picture this couple as parents\r\nJackie: I felt similar way on Patricia's wedding\r\nIggy: Patricia Stevens?\r\nJackie: yes\r\nIggy: so we're talking about the same person\r\nJackie: what a coincidence\r\nJackie: so she's pregnant?\r\nIggy: she thou...	Madison is pregnant but she doesn't want to talk about it. Patricia Stevens got married and she thought she was pregnant.
3	13819648	Marla: <file_photo>\r\nMarla: look what I found under my bed\r\nKiki: lol\r\nTamara: is that someone's underwear?\r\nMarla: it certainly isn't mine, my ass is big but it isn't huge\r\nKiki: it looks like male underwear\r\nTamara: not necessarily, maybe some butch had fun in your room while you were gone\r\nMarla: ok but how can you leave your underwear after hooking up? wtf is wrong with people\r\nKiki: she or he could be too wasted to notice\r\nTamara: or maybe someone put their pants there to piss you off\r\nMarla: that makes no sense\r\nMarla: it's so fucking childish\r\nKiki: if it's childish then it must have been your sister's idea\r\nMarla: she's 13, she doesn't have underwear that isn't pink\r\nTamara: maybe it belonged to one of your exes?\r\nKiki: she would have recognized it\r\nMarla: lol we're doing total CSI investigation on one pair of boxers :D\r\nKiki: <file_gif>\r\nTamara: lol\r\nTamara: I think your sister convinced someone to put their underwear in your room as a...	Marla found a pair of boxers under her bed.

	id	dialogue	summary
4	13728448	Robert: Hey give me the address of this music shop you mentioned before\r\nRobert: I have to buy guitar cable\r\nFred: <file_other>\r\nFred: Catch it on google maps\r\nRobert: thx m8\r\nFred: ur welcome	Robert wants Fred to send him the address of the music shop as he needs to buy guitar cable.

```
DataFrame Tail:
```

	id	dialogue	summary
813	13829423	Carla: I've got it...\r\nDiego: what?\r\nCarla: my date for graduation. Hope you're coming\r\nDiego: if you tell me when...\r\nCarla: oups sorry. June 4th\r\nDiego: we've got time.\r\nCarla: of course, but you have to book your plane\r\nDiego: i still don't know, and it's quite expensive\r\nCarla: that's why you have to book it right now. Please tell me you'll come\r\nDiego: i'd love to for sure\r\nCarla: come, come, please\r\nDiego: ok, i'll have a look and tell you.\r\nCarla: you could stay home for the week, my roommate won't be there.\r\nDiego: didn't you tell me your parents would come?\r\nCarla: yes they will, but they've got friends they could stay with.\r\nDiego: what was the company you flew with when you came last month?\r\nCarla: aeromexico was the cheapest at that time, but check with delta\r\nDiego: i think there is some flight comparison websites and also some apps.\r\nCarla: i only know the canadian one\r\nDiego: don't worry i'll find out \r\nCarla: ok ! i've to l...	Carla's date for graduation is on June 4th. Diego will try to come then.
814	13727710	Gita: Hello, this is Beti's Mum Gita, I wanted to ask if you were going on the school trip? \r\nBev: Hi Gita, yes, Milo wants me to come, he's a bit nervous going away from home or school still.\r\nGita: Yes, Beti is the same, they are still only 4 or 5 after all.\r\nBev: I know, still so young! It will help the teachers and TAs anyway, they have a lot to cope with!\r\nGita: I know, I could never do their job! I work part time as a music teacher, going round schools.\r\nBev: Oh really? I am in Marks, part time too, love it there! \r\nGita: Yes, it really helps to do some sort of work doesn't it! I could never manage full time, though.\r\nBev: Oh, I know, Gita. My sister's in management and she doesn't see her kids from 6.30am to 6.30pm every day! She is a high flier, but she does miss them. She does do lots with them on the weekend, though.\r\nGita: Yes, but children need time to just be at home and play or just be with family, not galavanting around all the time!\r\nBev: I agree 10...	Bev is going on the school trip with her son. Gita is going on the school trip with her daughter. Bev's sister rarely sees her children during the week because of her job. Gita has a few pets at home. The mothers with their children have to be at school at 7.45 to not miss the bus.
815	13829261	Julia: Greg just texted me\r\nRobert: ugh, delete him already\r\nJulia: He's saying he's sorry\r\nRobert: damn girl, delete the bastard\r\nJulia: it's not that simple, you know it\r\nRobert: No Julia, it is pretty simple\r\nRobert: go and delete him\r\nJulia: But he apologised, ok? He's never done it before\r\nRobert: srsly?\r\nRobert: do I need to remind you he cheated on you?\r\nRobert: Julia I'm not going through this again with you\r\nJulia: People change, I do believe it, maybe he changed. He apologised\r\nRobert: and that's it? That' ok? how's different from two other times?\r\nJulia: i told you - he apologised! he's sorry, he wants to meet\r\nRobert: don't, honey, really. We've been through this\r\nJulia: I know, but it's not easy. I think I love him\r\nRobert: i know you do, but you need to be strong. do you want to come over?\r\nJulia: no, thank you love, but i have to get up early tomorrow\r\nRobert: ok, you should go to sleep then\r\nJulia: what about Greg?\r\nRobert: do...	Greg cheated on Julia. He apologises to her. Robert tells Julia not to meet Greg.
816	13680226	Marry: I broke my nail ;(\r\nTina: oh, no!\r\nMarry: u know I have that party tomorrow!!!\r\nTina: I know, let me think...\r\nTina: I got it!. My sister friend is a cosmetitian, maybe she 'll help\r\nMarry: anyone will be good, I'm desperate!\r\nTina: I'll call her and let u know, ok?\r\nMarry: ok, I'll wait, but hurry!	Marry broke her nail and has a party tomorrow. Tina will call a cosmetician that she knows and let Marry know if she can help.

	id	dialogue	summary
817	13862383	Paige: I asked them to wait and send the declaration later\nPaige: Even end of March if it's possible\nMaddy: What did they say?\nPaige: They want to close it asap cause Lisa is afraid she forgets about it later\nPaige: But I can remind her in a couple of weeks\nPaige: It's my responsibility after all\nMaddy: But does it really matter? I mean the declaration\nMaddy: I think the deadline for payment is 31 March anyway\nPaige: I'm not sure, that's what I asked her\nPaige: Hope she confirms	Paige wants to have the declaration sent later. Lisa wants to send it soon. The deadline for payment is 31 March.

In [39]:

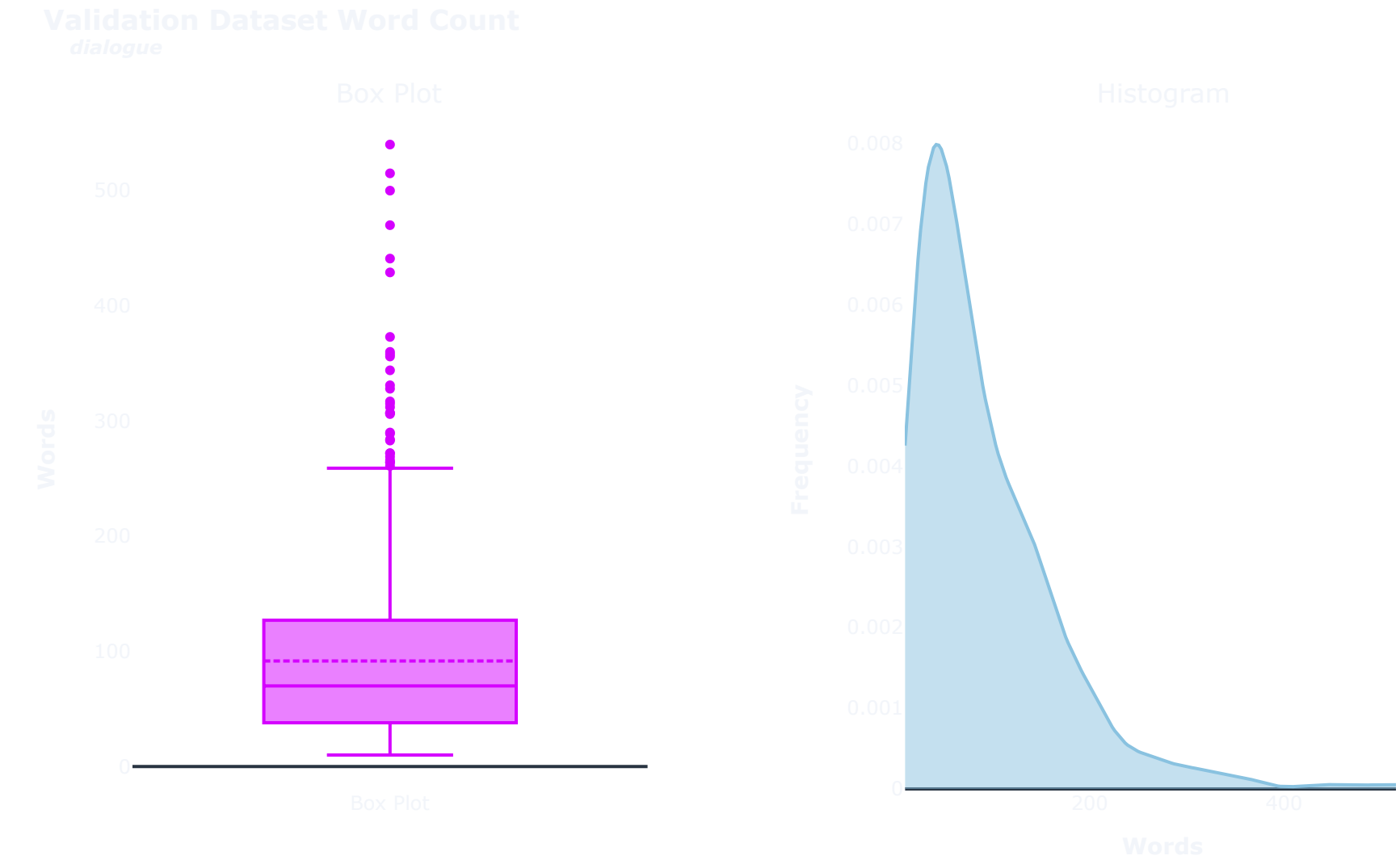
```
# Removing 'Id' from categorical features list
categorical_features.remove('id')
```

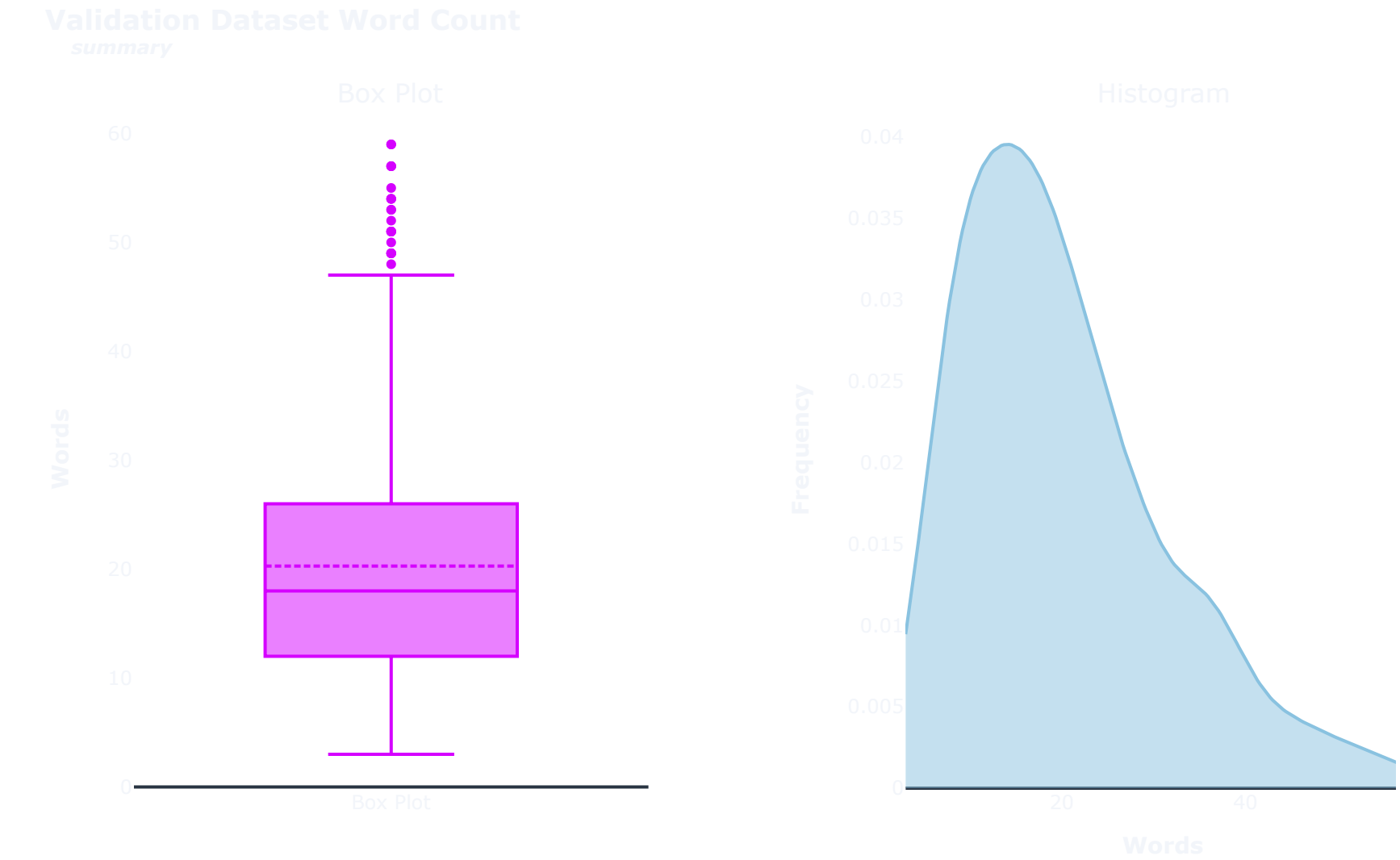
Hide code

In [40]:

```
df_text_lenght = pd.DataFrame()
for feat in categorical_features:
    df_text_lenght[feat] = val[feat].apply(lambda x: len(str(x).split()))

histogram_boxplot(df_text_lenght,'#89c2e0', '#d500ff', 600, 1000, True, 'Validation Dataset')
```

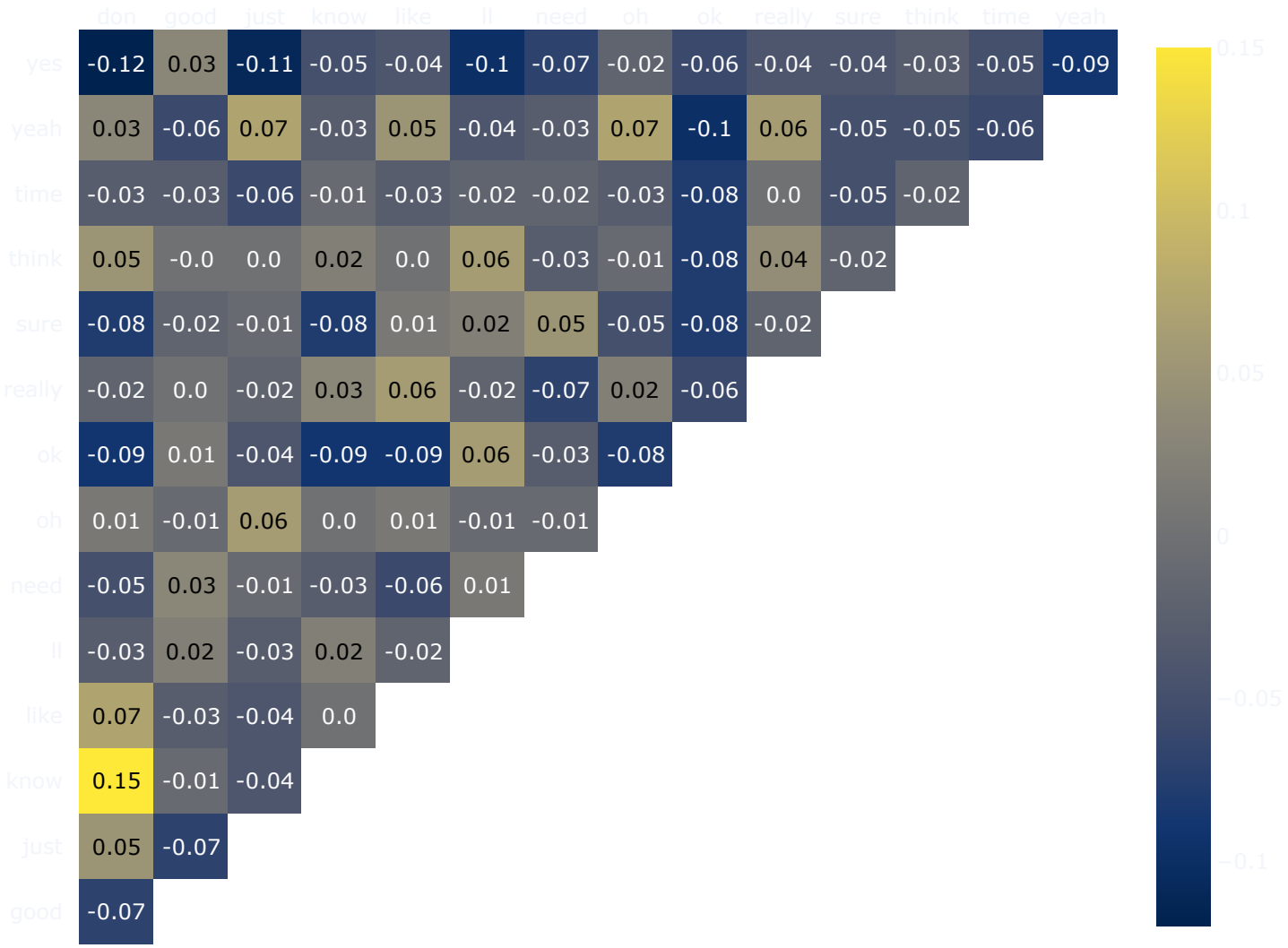

Validation Dataset Word Count
dialogue
Box Plot
Words
500
400
300
200
100
0
Box Plot
Histogram
Frequency
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
200
400
Words


Validation Dataset Word Count
summary
Box Plot
Words
60
50
40
30
20
10
0
Box Plot
Histogram
Frequency
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
20
40
Words

Hide code

In [41]:

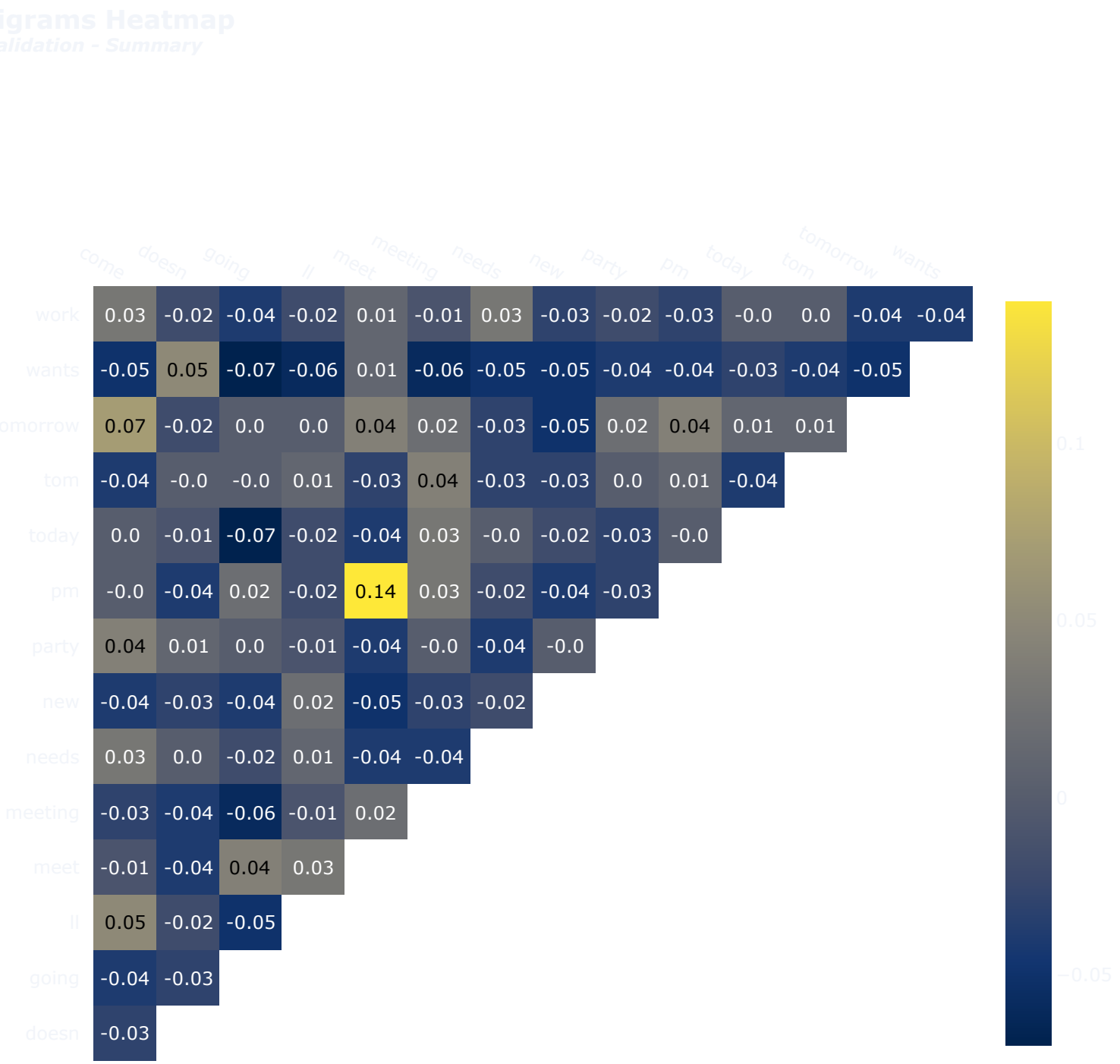
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english') # Top 15 terms
x = vectorizer.fit_transform(val['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Unigrams', 'Validation - Dialogue', 800, 800, 12)
```


Unigrams Heatmap
Validation - Dialogue
don good just know like ll need oh ok really sure think time yeah
yes -0.12 0.03 -0.11 -0.05 -0.04 -0.1 -0.07 -0.02 -0.06 -0.04 -0.04 -0.03 -0.05 -0.09
yeah 0.03 -0.06 0.07 -0.03 0.05 -0.04 -0.03 0.07 -0.1 0.06 -0.05 -0.05 -0.06
time -0.03 -0.03 -0.06 -0.01 -0.03 -0.02 -0.02 -0.03 -0.08 0.0 -0.05 -0.02
think 0.05 -0.0 0.0 0.02 0.0 0.06 -0.03 -0.01 -0.08 0.04 -0.02
sure -0.08 -0.02 -0.01 -0.08 0.01 0.02 0.05 -0.05 -0.08 -0.02
really -0.02 0.0 -0.02 0.03 0.06 -0.02 -0.07 0.02 -0.06
ok -0.09 0.01 -0.04 -0.09 -0.09 0.06 -0.03 -0.08
oh 0.01 -0.01 0.06 0.0 0.01 -0.01 -0.01
need -0.05 0.03 -0.01 -0.03 -0.06 0.01
ll -0.03 0.02 -0.03 0.02 -0.02
like 0.07 -0.03 -0.04 0.0
know 0.15 -0.01 -0.04
just 0.05 -0.07
good -0.07
0.15
0.1
0.05
0
−0.05
−0.1

Hide code

In [42]:

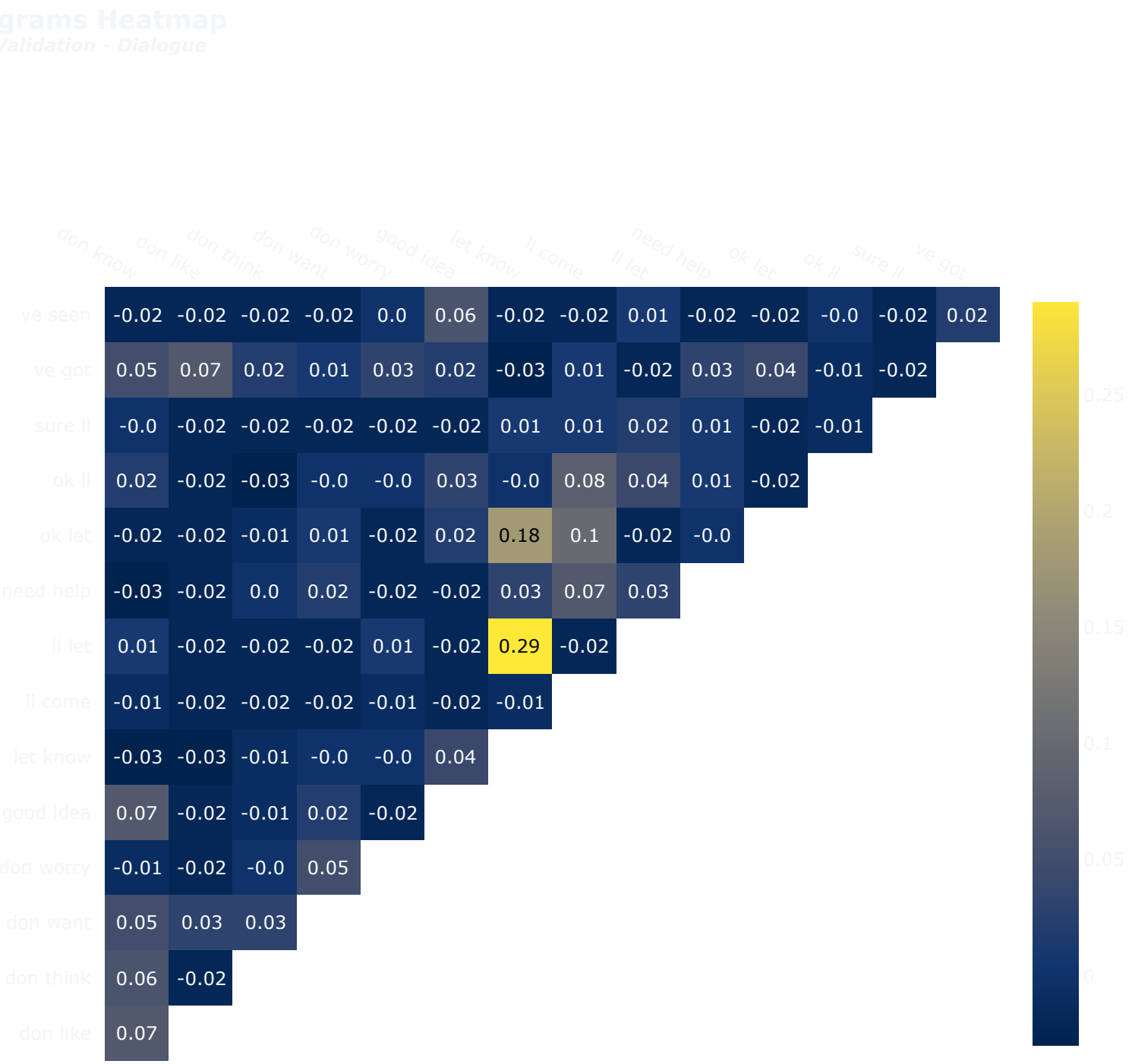
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english') # Top 15 terms
x = vectorizer.fit_transform(val['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Unigrams', 'Validation - Summary', 800, 800, 12)
```


Unigrams Heatmap
Validation - Summary
come doesn going ll meet meeting needs new party pm today tom tomorrow wants
work 0.03 -0.02 -0.04 -0.02 0.01 -0.01 0.03 -0.03 -0.02 -0.03 -0.0 0.0 -0.04 -0.04
wants -0.05 0.05 -0.07 -0.06 0.01 -0.06 -0.05 -0.05 -0.04 -0.04 -0.03 -0.04 -0.05
tomorrow 0.07 -0.02 0.0 0.0 0.04 0.02 -0.03 -0.05 0.02 0.04 0.01 0.01
tom -0.04 -0.0 -0.0 0.01 -0.03 0.04 -0.03 -0.03 0.0 0.01 -0.04
today 0.0 -0.01 -0.07 -0.02 -0.04 0.03 -0.0 -0.02 -0.03 -0.0
pm -0.0 -0.04 0.02 -0.02 0.14 0.03 -0.02 -0.04 -0.03
party 0.04 0.01 0.0 -0.01 -0.04 -0.0 -0.04 -0.0
new -0.04 -0.03 -0.04 0.02 -0.05 -0.03 -0.02
needs 0.03 0.0 -0.02 0.01 -0.04 -0.04
meeting -0.03 -0.04 -0.06 -0.01 0.02
meet -0.01 -0.04 0.04 0.03
ll 0.05 -0.02 -0.05
going -0.04 -0.03
doesn -0.03
0.1
0.05
0
−0.05

Hide code

In [22]:

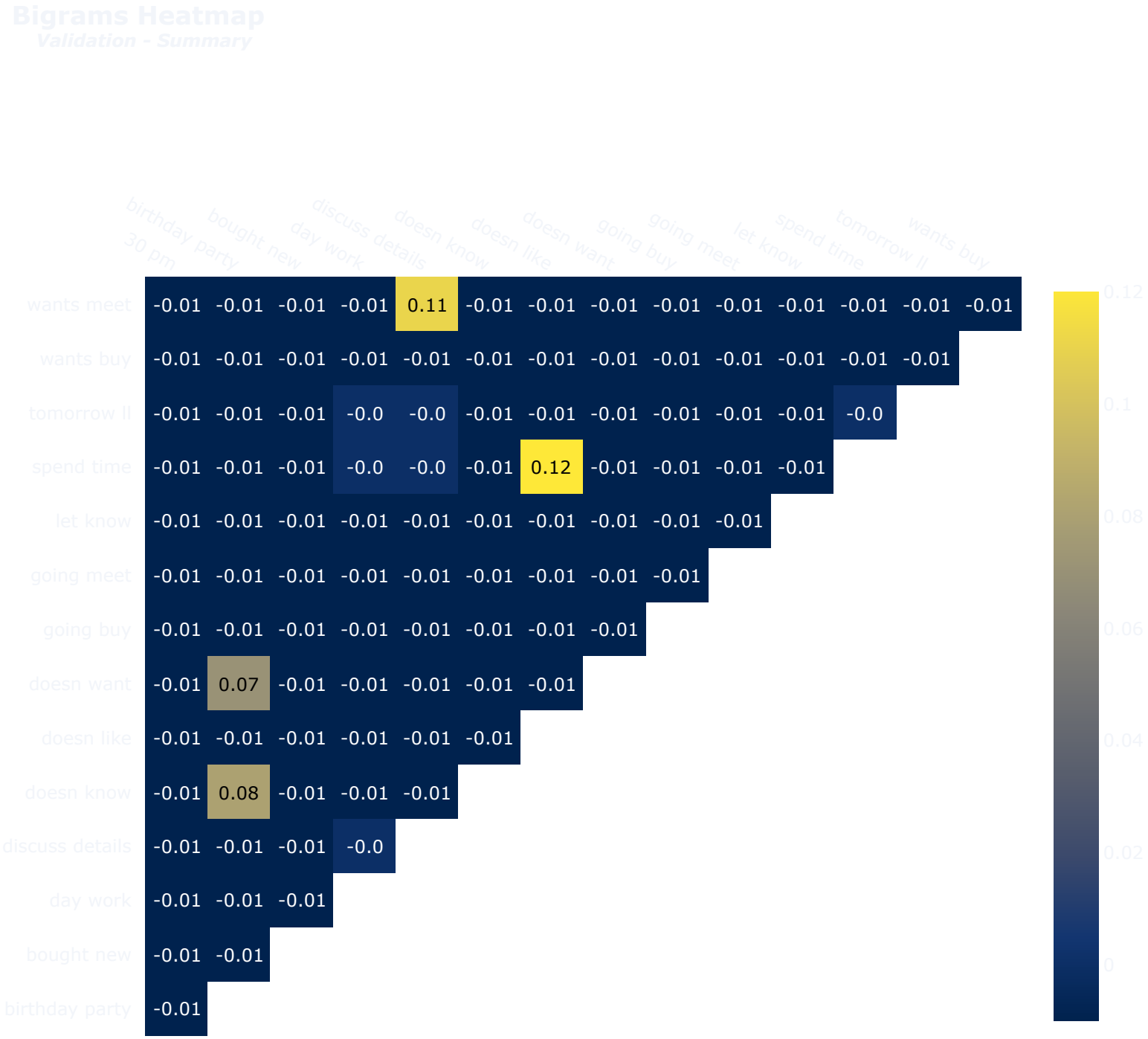
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (2,2)) # Top 15 te
rms
x = vectorizer.fit_transform(val['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Bigrams', 'Validation - Dialogue', 800, 800, 12)
```


Bigrams Heatmap
Validation - Dialogue
don know
don like
don think
don want
don worry
good idea
let know
ll come
ll let
need help
ok let
ok ll
sure ll
ve got
ve seen
ve got
sure ll
ok ll
ok let
need help
ll let
ll come
let know
good idea
don worry
don want
don think
don like
-0.02 -0.02 -0.02 -0.02 0.0 0.06 -0.02 -0.02 0.01 -0.02 -0.02 -0.0 -0.02 0.02
0.05 0.07 0.02 0.01 0.03 0.02 -0.03 0.01 -0.02 0.03 0.04 -0.01 -0.02
-0.0 -0.02 -0.02 -0.02 -0.02 -0.02 0.01 0.01 0.02 0.01 -0.02 -0.01
0.02 -0.02 -0.03 -0.0 -0.0 0.03 -0.0 0.08 0.04 0.01 -0.02
-0.02 -0.02 -0.01 0.01 -0.02 0.02 0.18 0.1 -0.02 -0.0
-0.03 -0.02 0.0 0.02 -0.02 -0.02 0.03 0.07 0.03
0.01 -0.02 -0.02 -0.02 0.01 -0.02 0.29 -0.02
-0.01 -0.02 -0.02 -0.02 -0.01 -0.02 -0.01
-0.03 -0.03 -0.01 -0.0 -0.0 0.04
0.07 -0.02 -0.01 0.02 -0.02
-0.01 -0.02 -0.0 0.05
0.05 0.03 0.03
0.06 -0.02
0.07
0.25
0.2
0.15
0.1
0.05
0

Hide code

In [23]:

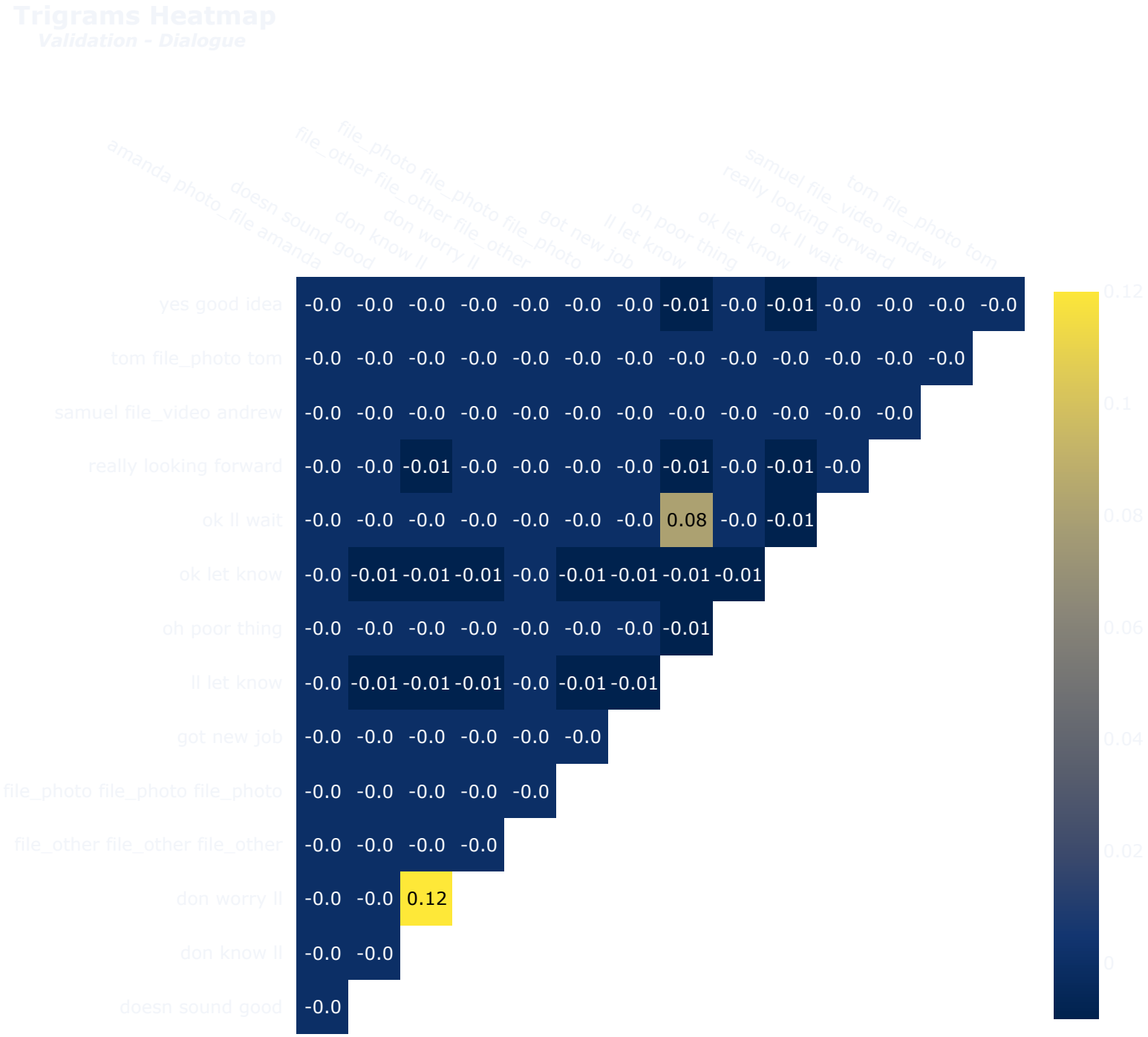
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (2,2)) # Top 15 te
rms
x = vectorizer.fit_transform(val['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Bigrams', 'Validation - Summary', 800, 800, 12)
```


Bigrams Heatmap
Validation - Summary
birthday party
bought new
day work
discuss details
doesn know
doesn like
doesn want
going buy
going meet
let know
spend time
tomorrow ll
wants buy
30 pm
wants meet
wants buy
tomorrow ll
spend time
let know
going meet
going buy
doesn want
doesn like
doesn know
discuss details
day work
bought new
birthday party
-0.01 -0.01 -0.01 -0.01 0.11 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.0 -0.0 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.0
-0.01 -0.01 -0.01 -0.0 -0.0 -0.01 0.12 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 0.07 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.01 -0.01 -0.01
-0.01 0.08 -0.01 -0.01 -0.01
-0.01 -0.01 -0.01 -0.0
-0.01 -0.01 -0.01
-0.01 -0.01
-0.01
0.12
0.1
0.08
0.06
0.04
0.02
0

Hide code

In [24]:

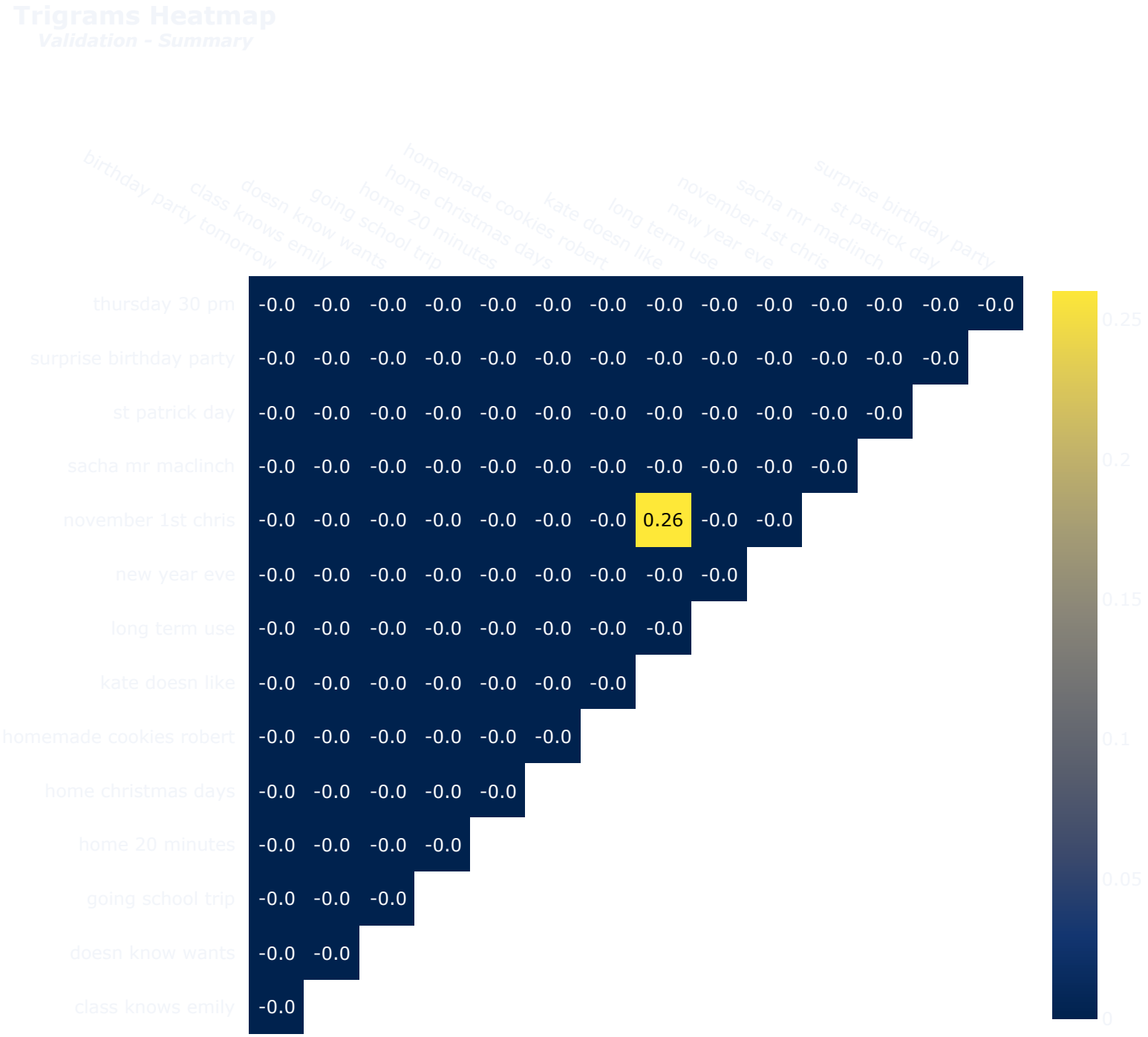
```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (3,3)) # Top 15 te
rms
x = vectorizer.fit_transform(val['dialogue'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Trigrams', 'Validation - Dialogue', 800, 800, 12)
```


Trigrams Heatmap
Validation - Dialogue
amanda photo_file amanda
doesn sound good
don know ll
don worry ll
file_other file_other file_other
file_photo file_photo file_photo
got new job
ll let know
oh poor thing
ok let know
really looking forward
ok ll wait
samuel file_video andrew
tom file_photo tom
yes good idea
tom file_photo tom
samuel file_video andrew
really looking forward
ok ll wait
ok let know
oh poor thing
ll let know
got new job
file_photo file_photo file_photo
file_other file_other file_other
don worry ll
don know ll
doesn sound good
0.12
0.1
0.08
0.06
0.04
0.02
0

Hide code

In [25]:

```
vectorizer = TfidfVectorizer(max_features = 15,stop_words = 'english',ngram_range = (3,3)) # Top 15 te
rms
x = vectorizer.fit_transform(val['summary'].fillna(''))
df_tfidfvect = pd.DataFrame(x.toarray(), columns=vectorizer.get_feature_names_out())
plot_correlation(df_tfidfvect, 'Trigrams', 'Validation - Summary', 800, 800, 12)
```


Trigrams Heatmap
Validation - Summary
birthday party tomorrow
class knows emily
doesn know wants
going school trip
home 20 minutes
homemade cookies robert
home christmas days
kate doesn like
long term use
new year eve
november 1st chris
sacha mr maclinch
st patrick day
surprise birthday party
thursday 30 pm
surprise birthday party
st patrick day
sacha mr maclinch
november 1st chris
new year eve
long term use
kate doesn like
homemade cookies robert
home christmas days
home 20 minutes
going school trip
doesn know wants
class knows emily
0.26
0.25
0.2
0.15
0.1
0.05
0

Overall, we have similar patterns across all the three datasets. Summaries are shorter in length than dialogues—as expected—and lots of terms that seem reasonable to be together have a higher degree of correlation.

By analyzing the *n-grams* heatmaps, it is also clear that this data consists of chat/dialogue texts, since we can see a lot of terms that would usuallly appear in conversations.

Preprocessing Data

One of the main advantages of working with pre-trained models, such as BART, is that these models are usually extremely robust and require very little data preprocessing.

While performing the EDA, I noticed that we have some tags in a few texts, such as `file_photo`. Let's take a look at a few examples.

Hide code

In [47]:
```
print(train['dialogue'].iloc[14727])
```

```
Theresa: <file_photo>
Theresa: <file_photo>
Theresa: Hey Louise, how are u?
Theresa: This is my workplace, they always give us so much food here 😊
Theresa: Luckily they also offer us yoga classes, so all the food isn't much of a problem 😂
Louise: Hey!! 🙂
Louise: Wow, that's awesome, seems great 😎 Haha
Louise: I'm good! Are you coming to visit Stockholm this summer? 🙂
Theresa: I don't think so :/ I need to prepare for Uni.. I will probably attend a few lessons this
winter
Louise: Nice! Do you already know which classes you will attend?
Theresa: Yes, it will be psychology :) I want to complete a few modules that I missed :)
Louise: Very good! Is it at the Uni in Prague?
Theresa: No, it will be in my home town :)
Louise: I have so much work right now, but I will continue to work until the end of summer, then
I'm also back to Uni, on the 26th September!
Theresa: You must send me some pictures, so I can see where you live :)
Louise: I will, and of my cat and dog too 🤗
Theresa: Yeeeesss pls :)))
Louise: 👌👌
Theresa: 🐱💖
```

Hide code

In [48]:
```
print(test['dialogue'].iloc[0])
```

```
Hannah: Hey, do you have Betty's number?
Amanda: Lemme check
Hannah: <file_gif>
Amanda: Sorry, can't find it.
Amanda: Ask Larry
Amanda: He called her last time we were at the park together
Hannah: I don't know him well
Hannah: <file_gif>
Amanda: Don't be shy, he's very nice
Hannah: If you say so..
Hannah: I'd rather you texted him
Amanda: Just text him 🙂
Hannah: Urgh.. Alright
Hannah: Bye
Amanda: Bye bye
```

I am going to use the `clean_tags` function defined below to remove these tags from the texts, so we can make them cleaner.

In [49]:

```
def clean_tags(text):
    clean = re.compile('<.*?>') # Compiling tags
    clean = re.sub(clean, '', text) # Replacing tags text by an empty string

    # Removing empty dialogues
    clean = '\n'.join([line for line in clean.split('\n') if not re.match('.*:\s*$', line)])

    return clean
```

Hide code

In [50]:

```
test1 = clean_tags(train['dialogue'].iloc[14727]) # Applying function to example text
test2 = clean_tags(test['dialogue'].iloc[0]) # Applying function to example text

# Printing results
print(test1)
print('\n' *3)
print(test2)
```

Theresa: Hey Louise, how are u?
Theresa: This is my workplace, they always give us so much food here 😊
Theresa: Luckily they also offer us yoga classes, so all the food isn't much of a problem 😂
Louise: Hey!! 🙂
Louise: Wow, that's awesome, seems great 😎 Haha
Louise: I'm good! Are you coming to visit Stockholm this summer? 🙂
Theresa: I don't think so :/ I need to prepare for Uni.. I will probably attend a few lessons this winter
Louise: Nice! Do you already know which classes you will attend?
Theresa: Yes, it will be psychology :) I want to complete a few modules that I missed :)
Louise: Very good! Is it at the Uni in Prague?
Theresa: No, it will be in my home town :)
Louise: I have so much work right now, but I will continue to work until the end of summer, then I'm also back to Uni, on the 26th September!
Theresa: You must send me some pictures, so I can see where you live :)
Louise: I will, and of my cat and dog too 🤗
Theresa: Yeeeesss pls :)))
Louise: 👌👌
Theresa: 🐱💖

Hannah: Hey, do you have Betty's number?
Amanda: Lemme check
Amanda: Sorry, can't find it.
Amanda: Ask Larry
Amanda: He called her last time we were at the park together
Hannah: I don't know him well
Amanda: Don't be shy, he's very nice

```
Hannah: If you say so..
Hannah: I'd rather you texted him
Amanda: Just text him 🙂
Hannah: Urgh.. Alright
Hannah: Bye
Amanda: Bye bye
```

You can see that we have successfully removed the tags from the texts. I am now going to define the `clean_df` function, in which we will apply the `clean_tags` to the entire datasets.

In [51]:
```
# Defining function to clean every text in the dataset.
def clean_df(df, cols):
    for col in cols:
        df[col] = df[col].fillna('').apply(clean_tags)
    return df
```

In [52]:
```
# Cleaning texts in all datasets
train = clean_df(train,['dialogue', 'summary'])
test = clean_df(test,['dialogue', 'summary'])
val = clean_df(val,['dialogue', 'summary'])
```

In [53]:
```
train.tail(3) # Visualizing results
```

Out[53]:

	id	dialogue	summary
14729	13819050	John: Every day some bad news. Japan will hunt whales again\r\nErica: Yes, I've read this. It's very upsetting\r\nJohn: Cruel Japanese\r\nFaith: I think this is a racist remark. Because Island and Norways has never joined this international whaling agreement\r\nErica: really? I haven't known, everybody is so outraged by Japan\r\nFaith: sure, European hypocrisy \r\nJohn: not entirely. Scandinavians don't use the nets that Japanese use, so Norway and Island kill much less specimens that Japan will\r\nFaith: oh, it's much more complex than one may expect\r\nJohn: True, but the truth is, that all of them should stop\r\nJohn: and this decision is a step back\r\nFaith: yes, this is worrying\r\nErica: And it seems that the most important whaling countries are out of the agreement right now\r\nFaith: yes, seems so\r\nJohn: Just like USA leaving the Paris Agreement	Japan is going to hunt whales again. Island and Norway never stopped hunting them. The Scandinavians kill fewer whales than the Japanese.
14730	13828395	Jennifer: Dear Celia! How are you doing?\r\nJennifer: The afternoon with the Collins was very pleasant, nice folks, but we missed you.\r\nJennifer: But I appreciate your consideration for Peter.\r\nCelia: My dear Jenny! It turns out that my decision not to come, though I wanted so much to see you again and Peter and the Collins, was right. Yesterday it all developed into a full bore cold. Sh.....\r\nCelia: All symptoms like in a text book.\r\nCelia: Luckily it's contagious only on the first 2, 3 days, so when we meet next week it should be alright.\r\nCelia: Thanks for asking! Somehow for all of us Peter comes first now.\r\nJennifer: That's too bad. Poor you...\r\nJennifer: I'll be driving to FR, do you want me to bring you sth? It's on my way.\r\nCelia: Thank you dear! I was at the pharmacy yesterday and had done my shopping the day before.\r\nCelia: You'd better still stay away from me in case I'm still contagious\r\nJennifer: Right. So I'll only leave a basket on your terrace. A...	Celia couldn't make it to the afternoon with the Collins and Jennifer as she is ill. She's working, but doesn't want to meet with Jennifer as it might be contagious. Jennifer will leave a basket with cookies on Celia's terrace.
14731	13729017	Georgia: are you ready for hotel hunting? We need to book something finally for Lisbon\r\nJuliette: sure we can go on, show me what you found\r\nJuliette: nah... it looks like an old lady's room lol\r\nJuliette: that's better... but the bed doesn't look very comfortable\r\nGeorgia: i kind of like it and it's really close to the city center\r\nJuliette: show me the others please\r\nJuliette: nah... this one sucks too, look at those horrible curtains \r\nGeorgia: aff Julie you are such a princess\r\nJuliette: i just want to be comfortable\r\nGeorgia: come on, stop whining you know we are on a budget\r\nJuliette: well hopefully we can find something that's decent right?\r\nGeorgia: i did show you decent but you want a Marriott or something :/\r\nJuliette: ok ok don't get angry\r\nGeorgia: we need to decide today, the longer we wait the higher the prices get \r\nJuliette: ok how about we get the second one then?\r\nGeorgia: ok give me a second\r\nJuliette: sure\r\nGeorgia: affff someon...	Georgia and Juliette are looking for a hotel in Lisbon. Juliette dislikes Georgia's choices. Juliette and Georgia decide on the second option presented by Georgia, but it has already been booked. Finally Georgia books the third hotel.

The tags have been removed from the texts. It's beneficial to conduct such data cleansing to eliminate noise—information that might not significantly contribute to the overall context and could potentially impair performance.

I am now going to perform some preprocessing that is necessary to prepare our data to serve as input to the pre-trained model and for fine-tuning. Most of what I'm doing here is a part of the tutorial on Text Summarization described in the 🤗 Transformers documentation, which you can see here (https://huggingface.co/docs/transformers/tasks/summarization).

First, I am going to use the 🤗 Datasets library to convert our Pandas Dataframes to Datasets. This is going to make our data ready to be processed across the whole Hugging Face ecosystem.

Hide code

In [54]:

```
# Transforming dataframes into datasets
train_ds = Dataset.from_pandas(train)
test_ds = Dataset.from_pandas(test)
val_ds = Dataset.from_pandas(val)

# Visualizing results
print(train_ds)
print('\n' * 2)
print(test_ds)
print('\n' * 2)
print(val_ds)
```

```
Dataset({
    features: ['id', 'dialogue', 'summary', '__index_level_0__'],
    num_rows: 14731
})
```

```
Dataset({
    features: ['id', 'dialogue', 'summary'],
    num_rows: 819
})
```

```
Dataset({
    features: ['id', 'dialogue', 'summary'],
    num_rows: 818
})
```

To see the cotent inside a 🤗 Dataset, we can select a specific row, as below.

In [55]:
```
train_ds[0] # Visualizing the first row
```

Out[55]:
```
{'id': '13818513',
 'dialogue': "Amanda: I baked  cookies. Do you want some?\r\nJerry: Sure!\r\nAmanda: I'll bring you
tomorrow :-)",
 'summary': 'Amanda baked cookies and will bring Jerry some tomorrow.',
 '__index_level_0__': 0}
```

This way, we can see the original ID, the dialogue, as well as the reference summary. `__index_level_0__` does not add anything to the data and will be removed further.

After successfully converting the pandas dataframes to 🤗Datasets, we can move on to the modeling process.

Modeling

As I have previously mentioned, we are going to fine-tune a version of BART that has been trained on several news articles for text summarization, **facebook/bart-large-xsum** (https://huggingface.co/facebook/bart-large-xsum).

I will briefly demonstrate this model by loading a summarization pipeline with it to show you how it works on news data.

In [56]:

```
# Loading summarization pipeline with the bart-large-cnn model
summarizer = pipeline('summarization', model = 'facebook/bart-large-xsum')
```

Hide output

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/1.51k [00:00<?, ?B/s]

Downloading pytorch_model.bin:   0%|          | 0.00/1.63G [00:00<?, ?B/s]

Downloading (…)neration_config.json:   0%|          | 0.00/309 [00:00<?, ?B/s]

Downloading (…)okenizer_config.json:   0%|          | 0.00/26.0 [00:00<?, ?B/s]

Downloading (…)olve/main/vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]

Downloading (…)olve/main/merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

As an example, I am going to use the following news article, published on CNN on October 24th, 2023, *Bobi, the world's oldest dog ever, dies aged 31 (https://edition.cnn.com/2023/10/24/europe/bobi-oldest-ever-dog-dies-intl-scli/index.html)*. Notice that this is a totally unseen news article that I'm passing to the model, so we can see how it performs.

In [57]:

```
news = '''Bobi, the world's oldest dog ever, has died after reaching the almost inconceivable age of 3
1 years and 165 days, said Guinness World Records (GWR) on Monday.
His death at an animal hospital on Friday was initially announced by veterinarian Dr. Karen Becker.
She wrote on Facebook that "despite outliving every dog in history, his 11,478 days on earth would nev
er be enough, for those who loved him."
There were many secrets to Bobi's extraordinary old age, his owner Leonel Costa told GWR in February.
He always roamed freely, without a leash or chain, lived in a "calm, peaceful" environment and ate hum
an food soaked in water to remove seasonings, Costa said.
He spent his whole life in Conqueiros, a small Portuguese village about 150 kilometers (93 miles) nort
h of the capital Lisbon, often wandering around with cats.
Bobi was a purebred Rafeiro do Alentejo – a breed of livestock guardian dog – according to his owner.
Rafeiro do Alentejos have a life expectancy of about 12-14 years, according to the American Kennel Clu
b.
But Bobi lived more than twice as long as that life expectancy, surpassing an almost century-old recor
d to become the oldest living dog and the oldest dog ever – a title which had previously been held by
Australian cattle-dog Bluey, who was born in 1910 and lived to be 29 years and five months old.
However, Bobi's story almost had a different ending.
When he and his three siblings were born in the family's woodshed, Costa's father decided they already
had too many animals at home.
Costa and his brothers thought their parents had taken all the puppies away to be destroyed. However,
a few sad days later, they found Bobi alive, safely hidden in a pile of logs.
The children hid the puppy from their parents and, by the time Bobi's existence became known, he was t
oo old to be put down and went on to live his record-breaking life.
His 31st birthday party in May was attended by more than 100 people and a performing dance troupe, GWR
said.
His eyesight deteriorated and walking became harder as Bobi grew older but he still spent time in the
backyard with the cats, rested more and napped by the fire.
"Bobi is special because looking at him is like remembering the people who were part of our family and
unfortunately are no longer here, like my father, my brother, or my grandparents who have already left
this world," Costa told GWR in May. "Bobi represents those generations."
```

```
'''
summarizer(news) # Using the pipeline to generate a summary of the text above
```

Out[57]:

```
[{'summary_text': 'The world’s oldest dog has died, Guinness World Records has confirmed.'}]
```

You can observe that the model is able to accurately produce a much shorter text consisting of the most relevant information present in the input text. This is a successful summarization.

However, this model has been trained mainly on datasets consisting of several news articles from CNN and the Daily Mail, not on much dialogue data. This is why I'm going to fine-tune it with the SamSum dataset.

Let's go ahead and load BartTokenizer and BartForConditionalGeneration using the ***facebook/bart-large-xsum*** checkpoint.

In [58]:

```
checkpoint = 'facebook/bart-large-xsum' # Model
tokenizer = BartTokenizer.from_pretrained(checkpoint) # Loading Tokenizer
```

In [59]:

```
model = BartForConditionalGeneration.from_pretrained(checkpoint) # Loading Model
```

We can also print below the architecture of the model.

Hide code

In [60]:

```
print(model) # Visualizing model's architecture
```

```
BartForConditionalGeneration(
  (model): BartModel(
    (shared): Embedding(50264, 1024, padding_idx=1)
    (encoder): BartEncoder(
      (embed_tokens): Embedding(50264, 1024, padding_idx=1)
      (embed_positions): BartLearnedPositionalEmbedding(1026, 1024)
      (layers): ModuleList(
        (0-11): 12 x BartEncoderLayer(
          (self_attn): BartAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (activation_fn): GELUActivation()
          (fc1): Linear(in_features=1024, out_features=4096, bias=True)
          (fc2): Linear(in_features=4096, out_features=1024, bias=True)
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    )
    (decoder): BartDecoder(
      (embed_tokens): Embedding(50264, 1024, padding_idx=1)
      (embed_positions): BartLearnedPositionalEmbedding(1026, 1024)
      (layers): ModuleList(
        (0-11): 12 x BartDecoderLayer(
          (self_attn): BartAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
```

```
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (activation_fn): GELUActivation()
          (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (encoder_attn): BartAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (fc1): Linear(in_features=1024, out_features=4096, bias=True)
          (fc2): Linear(in_features=4096, out_features=1024, bias=True)
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    )
  )
  (lm_head): Linear(in_features=1024, out_features=50264, bias=False)
)
```

It is possible to see that the models consist of an encoder and a decoder, we can see the Linear Layers, as well as the activation functions, which use $GeLU$, instead of the more typical $ReLU$.

It is also interesting to observe the output layer, **lm_head**, which shows us that this model is ideal for generating outputs with a vocabulary size— `out_features=50264` —this shows us that this architecture is adequate for summarization tasks, as well as other tasks, such as translation for example.

Now we must preprocess our datasets and use BartTokenizer so that our data is legible for the BART model.

The following `preprocess_function` can be directly copied from the 🤗 Transformers documentation, and it serves well to preprocess data for several NLP tasks. I am going to delve a bit deeper into how it preprocesses the data by explaining the steps it takes.

- `inputs = [doc for doc in examples["dialogue"]]:` In this line, we are iterating over every `dialogue` in the dataset and saving them as input to the model.

- `model_inputs = tokenizer(inputs, max_length=1024, truncation=True)` `:` Here, we are using the `tokenizer` to convert the input dialogues into tokens that can be easily understood by the BART model. The `truncation=True` parameter ensures that all dialogues have a maximum number of 1024 tokens, as defined by the `max_length` parameter.

- `labels = tokenizer(text_target=examples["summary"], max_length=128, truncation=True):` This line performs a very similar tokenization process as the one above. This time, however, it tokenizes the target variable, which is our summaries. Also, note that the max_length here is significantly lower, at 128. This implies that we expect summaries to be a much shorter text than that of dialogues.

- `model_inputs["labels"] = labels["input_ids"]:` This line is essentially adding the tokenized labels to the preprocessed dataset, alongside the tokenized inputs.

In [61]:

```
def preprocess_function(examples):
    inputs = [doc for doc in examples["dialogue"]]
    model_inputs = tokenizer(inputs, max_length=1024, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(examples["summary"], max_length=128, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

In [62]:

```
# Applying preprocess_function to the datasets
tokenized_train = train_ds.map(preprocess_function, batched=True,
                               remove_columns=['id', 'dialogue', 'summary', '__index_level_0__']) # Re
moving features

tokenized_test = test_ds.map(preprocess_function, batched=True,
                               remove_columns=['id', 'dialogue', 'summary']) # Removing features

tokenized_val = val_ds.map(preprocess_function, batched=True,
                               remove_columns=['id', 'dialogue', 'summary']) # Removing features

# Printing results
print('\n' * 3)
print('Preprocessed Training Dataset:\n')
print(tokenized_train)
print('\n' * 2)
print('Preprocessed Test Dataset:\n')
print(tokenized_test)
print('\n' * 2)
print('Preprocessed Validation Dataset:\n')
print(tokenized_val)
```

```
0%|          | 0/15 [00:00<?, ?ba/s]

0%|          | 0/1 [00:00<?, ?ba/s]

0%|          | 0/1 [00:00<?, ?ba/s]
```

```
Preprocessed Training Dataset:

Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 14731
})



Preprocessed Test Dataset:

Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 819
})



Preprocessed Validation Dataset:

Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 818
})
```

Our tokenized datasets consist now of only three features, `input_ids`, `attention_mask`, and `labels`. Let's print a sample from our tokenized train dataset to investigate further how the preprocess function altered the data.

Hide code

In [63]:

```
# Selecting a sample from the dataset
sample = tokenized_train[0]

# Printing its features
print("input_ids:")
print(sample['input_ids'])
print("\n")
print("attention_mask:")
print(sample['attention_mask'])
print("\n")
print("sample:")
print(sample['labels'])
print("\n")
```

```
input_ids:
[0, 10127, 5219, 35, 38, 17241, 1437, 15269, 4, 1832, 47, 236, 103, 116, 50121, 50118, 39237, 35, 9
136, 328, 50121, 50118, 10127, 5219, 35, 38, 581, 836, 47, 3859, 48433, 2]


attention_mask:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]


sample:
[0, 10127, 5219, 17241, 15269, 8, 40, 836, 6509, 103, 3859, 4, 2]
```

Let's dive a deep further into what each feature means.

- **input_ids**: These are the token IDs mapped to the dialogues. Each token represents a word or subword that can be perfectly understood by the BART model. For instance, the number ***5219*** could be a map to a word like *"hello"* in BART's vocabulary. Each word has its unique token in this context.
- **attention_mask**: This mask indicates which tokens the model should pay attention to and which tokens should be ignored. This is often used in the context of padding—when some tokens are used to equalize the lengths of sentences—but most of these padding tokens do not hold any meaningful information, so the attention mask ensures the model does not focus on them. In the case of this specific sample, all tokens are masked as '1', meaning they are all relevant and none of them are used for padding.
- **labels**: Similarly to the first feature, these are token IDs obtained from the words and subwords in the summaries. These are the tokens that the model will be trained on to give as output.

We must now use `DataCollatorForSeq2Seq` to batch the data. These data collators may also automatically apply some processing techniques, such as padding. They are important for the task of fine-tuning models and are also present in the 🤗 Transformers documentation for text summarization.

In [64]:
```
# Instantiating Data Collator
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)
```

Next, I am going to load the ROUGE metrics and define a new function to evaluate the model.

The `compute_metrics` function is also available in the documentation. In this function, we are basically extracting the model-generated summaries, as well as the human-generated summaries, and decoding them. We then use rouge to compare how similar they are to evaluate performance.

In [65]:
```
metric = load_metric('rouge') # Loading ROUGE Score
```

```
Downloading builder script:   0%|          | 0.00/2.16k [00:00<?, ?B/s]
```

In [66]:

```
def compute_metrics(eval_pred):
    predictions, labels = eval_pred# Obtaining predictions and true labels

    # Decoding predictions
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)

    # Obtaining the true labels tokens, while eliminating any possible masked token (i.e., label = -100)
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    # Rouge expects a newline after each sentence
    decoded_preds = ["\n".join(nltk.sent_tokenize(pred.strip())) for pred in decoded_preds]
    decoded_labels = ["\n".join(nltk.sent_tokenize(label.strip())) for label in decoded_labels]



    # Computing rouge score
    result = metric.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)
    result = {key: value.mid.fmeasure * 100 for key, value in result.items()} # Extracting some result
s


    # Add mean-generated length
    prediction_lens = [np.count_nonzero(pred != tokenizer.pad_token_id) for pred in predictions]
    result["gen_len"] = np.mean(prediction_lens)

    return {k: round(v, 4) for k, v in result.items()}
```

We now use the `Seq2SeqTrainingArguments` class to set some relevant settings for fine-tuning. I will first define a directory to serve as output, and then define the evaluation strategy, learning rate, etc.

This class can be quite extensive, with several different parameters. I highly suggest you take your time with the documentation (https://huggingface.co/docs/transformers/main_classes/trainer#transformers.Seq2SeqTrai to get familiar with them.

In [67]:

```
# Defining parameters for training
'''
Please don't forget to check the documentation.
Both the Seq2SeqTrainingArguments and Seq2SeqTrainer classes have quite an extensive list of parameters.

doc: https://huggingface.co/docs/transformers/v4.34.1/en/main_classes/trainer

'''
training_args = Seq2SeqTrainingArguments(
    output_dir = 'bart_samsum',
    evaluation_strategy = "epoch",
    save_strategy = 'epoch',
    load_best_model_at_end = True,
    metric_for_best_model = 'eval_loss',
    seed = seed,
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=2,
    weight_decay=0.01,
    save_total_limit=2,
    num_train_epochs=4,
    predict_with_generate=True,
    fp16=True,
    report_to="none"
)
```

Finally, the `Seq2SeqTrainer` class allows us to use **PyTorch** to fine-tune the model. In this class, we are basically defining the model, the training arguments, the datasets used for training and evaluation, the tokenizer, the data_collator, and the metrics.

In [68]:
```
# Defining Trainer
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

In [69]:
```
trainer.train() # Training model
```

[7364/7364 1:46:39, Epoch 3/4]

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	1.379400	1.484487	52.580300	27.111300	42.727600	48.367000	32.069600
2	1.074100	1.443861	52.815600	28.125900	43.714700	48.571200	29.147700
2	0.849400	1.525784	52.502500	27.853400	43.665300	48.442600	31.437100
3	0.681800	1.623999	52.693800	27.407100	43.325900	48.433200	30.183200

Out[69]:
```
TrainOutput(global_step=7364, training_loss=1.011966500186454, metrics={'train_runtime': 6402.2537,
'train_samples_per_second': 9.204, 'train_steps_per_second': 1.15, 'total_flos': 3.4981815168344064
e+16, 'train_loss': 1.011966500186454, 'epoch': 4.0})
```

We finally finished fine-tuning after 4 epochs. Since we had `load_best_model_at_end = True` in the training arguments, the Trainer automatically saves the model with the best performance, which in this case is the one with the lowest `Validation Loss`.

The second epoch was the one with the lowest validation loss, at `1.443861`. It also achieved the highest `Rouge1` and `Rouge2` scores, as well as the highest `Rougelsum` score.

I have not presented the `Rougelsum` score previously. According to [the documentation (https://pypi.org/project/rouge-score/)](https://pypi.org/project/rouge-score/) of the rouge-score library, we can conclude that this is similar to the RougeL score, but it measures content coverage at a sentence-by-sentence level, instead of the entire summary.

The `Gen Len` column gives us the average length of the model-generated summaries. It is relevant to remember that we want short, yet informative, texts. In this case, the second epoch also yielded the shortest summaries on average.

Evaluating and Saving Model

After training and testing the model, we can evaluate its performance on the `validation` dataset. We can use the `evaluate` method for that.

In [70]:
```
# Evaluating model performance on the tokenized validation dataset
validation = trainer.evaluate(eval_dataset = tokenized_val)
print(validation) # Printing results
```

[205/205 04:42]

```
{'eval_loss': 1.4104626178741455, 'eval_rouge1': 53.8804, 'eval_rouge2': 29.2329, 'eval_rougeL': 4
4.774, 'eval_rougeLsum': 49.8255, 'eval_gen_len': 28.8839, 'eval_runtime': 290.2637, 'eval_samples_
per_second': 2.818, 'eval_steps_per_second': 0.706, 'epoch': 4.0}
```

This outputs the same scores we have previously seen during training and testing. Here, we can notice that we have even **higher** performance in every metric compared to the performance in the testing set. When it comes to `Gen Len`, we also have more concise summaries in the validation set.

Considering that our results seem to be satisfactory at this point, we can go ahead and use the `save_model` method to save our fine-tuned model in the `bart_finetuned_samsum` directory. We can also use the `shutil` package to save the model in a *zip* file.

In [71]:
```
# Saving model to a custom directory
directory = "bart_finetuned_samsum"
trainer.save_model(directory)

# Saving model tokenizer
tokenizer.save_pretrained(directory)
```

Out[71]:
```
('bart_finetuned_samsum/tokenizer_config.json',
 'bart_finetuned_samsum/special_tokens_map.json',
 'bart_finetuned_samsum/vocab.json',
 'bart_finetuned_samsum/merges.txt',
 'bart_finetuned_samsum/added_tokens.json')
```

In [72]:
```
# Saving model in .zip format
shutil.make_archive('bart_finetuned_samsum', 'zip', '/kaggle/working/bart_finetuned_samsum')
shutil.move('bart_finetuned_samsum.zip', '/kaggle/working/bart_finetuned_samsum.zip')
```

Out[72]:
```
'/kaggle/working/bart_finetuned_samsum.zip'
```

After saving your model, you can easily upload it to Hugging Face Models (https://huggingface.co/docs/hub/models-uploading) and use it on new datasets and texts.

The fine-tuned model we trained here is now available for everyone on Hugging Face, and you can have access to it by clicking on luisotorres/bart-finetuned-samsum (https://huggingface.co/luisotorres/bart-finetuned-samsum).

Let's load the model, using the summarization pipeline, and generate some summaries for human evaluation, where we evaluate if the model-generated summaries are accurate or not.

In [73]:
```
# Loading summarization pipeline and model
summarizer = pipeline('summarization', model = 'luisotorres/bart-finetuned-samsum')
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/1.59k [00:00<?, ?B/s]

Downloading pytorch_model.bin:   0%|          | 0.00/1.63G [00:00<?, ?B/s]

Downloading (…)neration_config.json:   0%|          | 0.00/274 [00:00<?, ?B/s]

Downloading (…)okenizer_config.json:   0%|          | 0.00/1.31k [00:00<?, ?B/s]

Downloading (…)olve/main/vocab.json:   0%|          | 0.00/999k [00:00<?, ?B/s]

Downloading (…)olve/main/merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]

Downloading (…)cial_tokens_map.json:   0%|          | 0.00/957 [00:00<?, ?B/s]
```

After loading the pipeline, we can now produce some summaries. I'll first start by using examples from the validation dataset, so we can compare our model-generated summaries to the reference summaries.

Hide cell

In [74]:
```
# Obtaining a random example from the validation dataset
val_ds[35]
```

Out[74]:
```
{'id': '13821488',
 'dialogue': "John: doing anything special?\r\nAlex: watching 'Millionaires' on tvn\r\nSam: me too!
He has a chance to win a million!\r\nJohn: ok, fingers crossed then! :)",
 'summary': 'Alex and Sam are watching Millionaires.'}
```

Hide cell

In [75]:
```
text = "John: doing anything special?\r\nAlex: watching 'Millionaires' on tvn\r\nSam: me too! He has a
chance to win a million!\r\nJohn: ok, fingers crossed then! :)"
summary = "Alex and Sam are watching Millionaires."
generated_summary = summarizer(text)
```

```
Your max_length is set to 62, but your input_length is only 48. Since this is a summarization task,
where outputs shorter than the input are typically wanted, you might consider decreasing max_length
manually, e.g. summarizer('...', max_length=24)
```

Hide code

In [76]:
```
print('Original Dialogue:\n')
print(text)
print('\n' * 2)
print('Reference Summary:\n')
print(summary)
print('\n' * 2)
print('Model-generated Summary:\n')
print(generated_summary)
```

```
Original Dialogue:

John: doing anything special?
Alex: watching 'Millionaires' on tvn
Sam: me too! He has a chance to win a million!
John: ok, fingers crossed then! :)



Reference Summary:

Alex and Sam are watching Millionaires.



Model-generated Summary:

[{'summary_text': "Alex and Sam are watching 'Millionaires' on tvn."}]
```

The model-generated summary is just a bit longer than the reference summary, but it still captures quite well the content of the dialogue.

Let's see another example.

Hide cell

In [77]:
```
val_ds[22]
```

Out[77]:
```
{'id': '13727839',
 'dialogue': 'Madison: Hello Lawrence are you through with the article?\r\nLawrence: Not yet sir.
\r\nLawrence: But i will be in a few.\r\nMadison: Okay. But make it quick.\r\nMadison: The piece is
needed by today\r\nLawrence: Sure thing\r\nLawrence: I will get back to you once i am through.',
 'summary': 'Lawrence will finish writing the article soon.'}
```

Hide code

In [78]:

```
text = "Madison: Hello Lawrence are you through with the article?\r\nLawrence: Not yet sir. \r\nLawren
ce: But i will be in a few.\r\nMadison: Okay. But make it quick.\r\nMadison: The piece is needed by to
day\r\nLawrence: Sure thing\r\nLawrence: I will get back to you once i am through."
summary = "Lawrence will finish writing the article soon."
generated_summary = summarizer(text)

print('Original Dialogue:\n')
print(text)
print('\n' * 2)
print('Reference Summary:\n')
print(summary)
print('\n' * 2)
print('Model-generated Summary:\n')
print(generated_summary)
```

```
Original Dialogue:

Madison: Hello Lawrence are you through with the article?
Lawrence: Not yet sir.
Lawrence: But i will be in a few.
Madison: Okay. But make it quick.
Madison: The piece is needed by today
Lawrence: Sure thing
Lawrence: I will get back to you once i am through.



Reference Summary:

Lawrence will finish writing the article soon.



Model-generated Summary:

[{'summary_text': "Lawrence hasn't finished with the article yet, but he will be in a few minutes.
Madison needs the piece by today."}]
```

Once again, the model-generated summary is longer than the reference summary. However, I would definitely say that the model-generated summary is more informative than the reference one because it lets us know that there's a sense of urgency for Lawrence to finish the article since Madison needs it by today.

Let's see another example.

Hide cell

In [79]:
```
val_ds[4]
```

Out[79]:
```
{'id': '13728448',
 'dialogue': 'Robert: Hey give me the address of this music shop you mentioned before\r\nRobert: I have to buy guitar cable\r\nFred: Catch it on google maps\r\nRobert: thx m8\r\nFred: ur welcome',
 'summary': 'Robert wants Fred to send him the address of the music shop as he needs to buy guitar cable.'}
```

Hide cell

In [80]:

```
text = "Robert: Hey give me the address of this music shop you mentioned before\r\nRobert: I have to b
uy guitar cable\r\nFred: Catch it on google maps\r\nRobert: thx m8\r\nFred: ur welcome"
summary = "Robert wants Fred to send him the address of the music shop as he needs to buy guitar cabl
e."
generated_summary = summarizer(text)
```

```
Your max_length is set to 62, but your input_length is only 49. Since this is a summarization task,
where outputs shorter than the input are typically wanted, you might consider decreasing max_length
manually, e.g. summarizer('...', max_length=24)
```

Hide code

In [81]:

```
print('Original Dialogue:\n')
print(text)
print('\n' * 2)
print('Reference Summary:\n')
print(summary)
print('\n' * 2)
print('Model-generated Summary:\n')
print(generated_summary)
```

```
Original Dialogue:

Robert: Hey give me the address of this music shop you mentioned before
Robert: I have to buy guitar cable
Fred: Catch it on google maps
Robert: thx m8
Fred: ur welcome



Reference Summary:

Robert wants Fred to send him the address of the music shop as he needs to buy guitar cable.



Model-generated Summary:

[{'summary_text': 'Fred gives Robert the address of the music shop where he will buy guitar cabl
e.'}]
```

In this case, while the generated text captures the essence of the dialogue, it suffers from a lack of clarity due to ambiguity. Specifically, the pronoun *he* creates uncertainty about whether Fred or Robert intends to buy the guitar cable. In the original dialogue, it is clearly specified that it is Robert the one who has to buy the cable.

Now that we have been able to compare summaries, we can create some dialogues and input them into the model to check how it performs on them.

Hide code

In [82]:

```
# Creating new dialogues for evaluation
text = "John: Hey! I've been thinking about getting a PlayStation 5. Do you think it is worth it? \r\n
Dan: Idk man. R u sure ur going to have enough free time to play it? \r\nJohn: Yeah, that's why I'm no
t sure if I should buy one or not. I've been working so much lately idk if I'm gonna be able to play i
t as much as I'd like."
generated_summary = summarizer(text)
```

Hide code

In [83]:

```
print('Original Dialogue:\n')
print(text)
print('\n' * 2)
print('Model-generated Summary:\n')
print(generated_summary)
```

```
Original Dialogue:

John: Hey! I've been thinking about getting a PlayStation 5. Do you think it is worth it?
Dan: Idk man. R u sure ur going to have enough free time to play it?
John: Yeah, that's why I'm not sure if I should buy one or not. I've been working so much lately id
k if I'm gonna be able to play it as much as I'd like.



Model-generated Summary:

[{'summary_text': "John is thinking about getting a PlayStation 5, but he's not sure if it's worth
it as he doesn't have enough time to play it."}]
```

For this dialogue, I have decided to include some abbreviations such as *idk*—for *I don't know*—and *r u*—for *are you*— to observe how the model would interpret them.

We can see that the model has been able to successfully capture the essence of the dialogue and identify the main subject, which is John's uncertainty to buy a PlayStation 5 given the fact that he has so little time to play it.

Hide code

In [84]:

```
text = "Camilla: Who do you think is going to win the competition?\r\nMichelle: I believe Jonathan sho
uld win but I'm sure Mike is cheating!\r\nCamilla: Why do you say that? Can you prove Mike is really c
heating?\r\nMichelle: I can't! But I just know!\r\nCamilla: You shouldn't accuse him of cheating if yo
u don't have any evidence to support it."
generated_summary = summarizer(text)
```

Hide code

In [85]:

```
print('Original Dialogue:\n')
print(text)
print('\n' * 2)
print('Model-generated Summary:\n')
print(generated_summary)
```

```
Original Dialogue:

Camilla: Who do you think is going to win the competition?
Michelle: I believe Jonathan should win but I'm sure Mike is cheating!
Camilla: Why do you say that? Can you prove Mike is really cheating?
Michelle: I can't! But I just know!
Camilla: You shouldn't accuse him of cheating if you don't have any evidence to support it.



Model-generated Summary:

[{'summary_text': 'Jonathan should win the competition, but Michelle thinks Mike is cheating.'}]
```

Once more the model captures the main theme of the conversation, which is Michelle's belief that Jonathan should win the competition, but that Mike may be cheating. Some further improvements could be made, though, such as including the information that Michelle cannot really show any evidence to support her belief that Mike is cheating.

Conclusion and Deployment

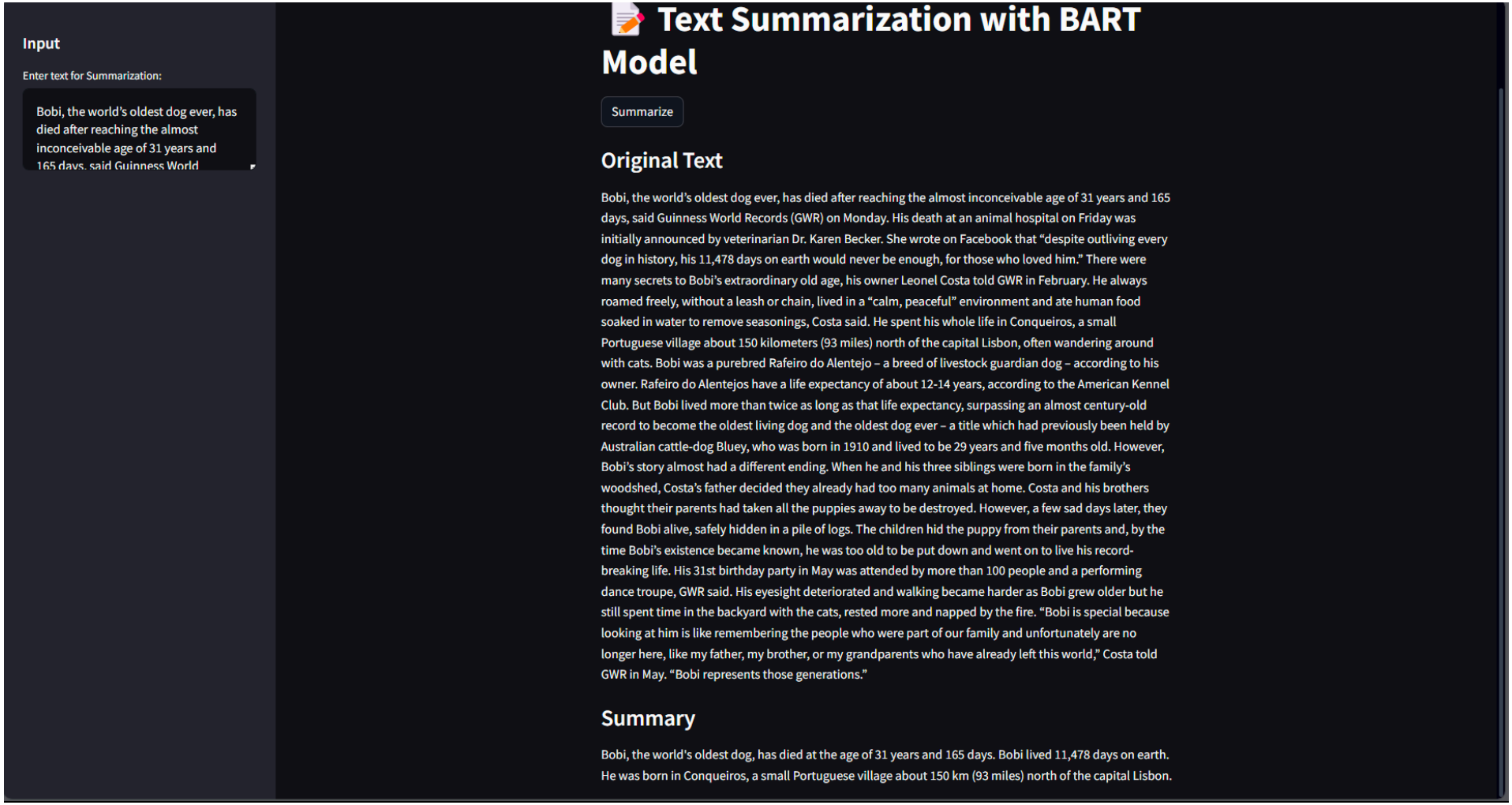
In this notebook, we have explored how we can use ***Large Language Models*** for several tasks involving Natural Language Processing, more specifically, Text Summarization tasks.

We delved into how Hugging Face's Transformers, Evaluate, and Datasets can be used to leverage frameworks such as PyTorch to fine-tune pre-trained models with a large number of parameters. This type of technique is usually referred to as **transfer learning**, which allows Data Scientists and Machine Learning Engineers to exploit the knowledge gained from previous tasks to improve generalization on a new task.

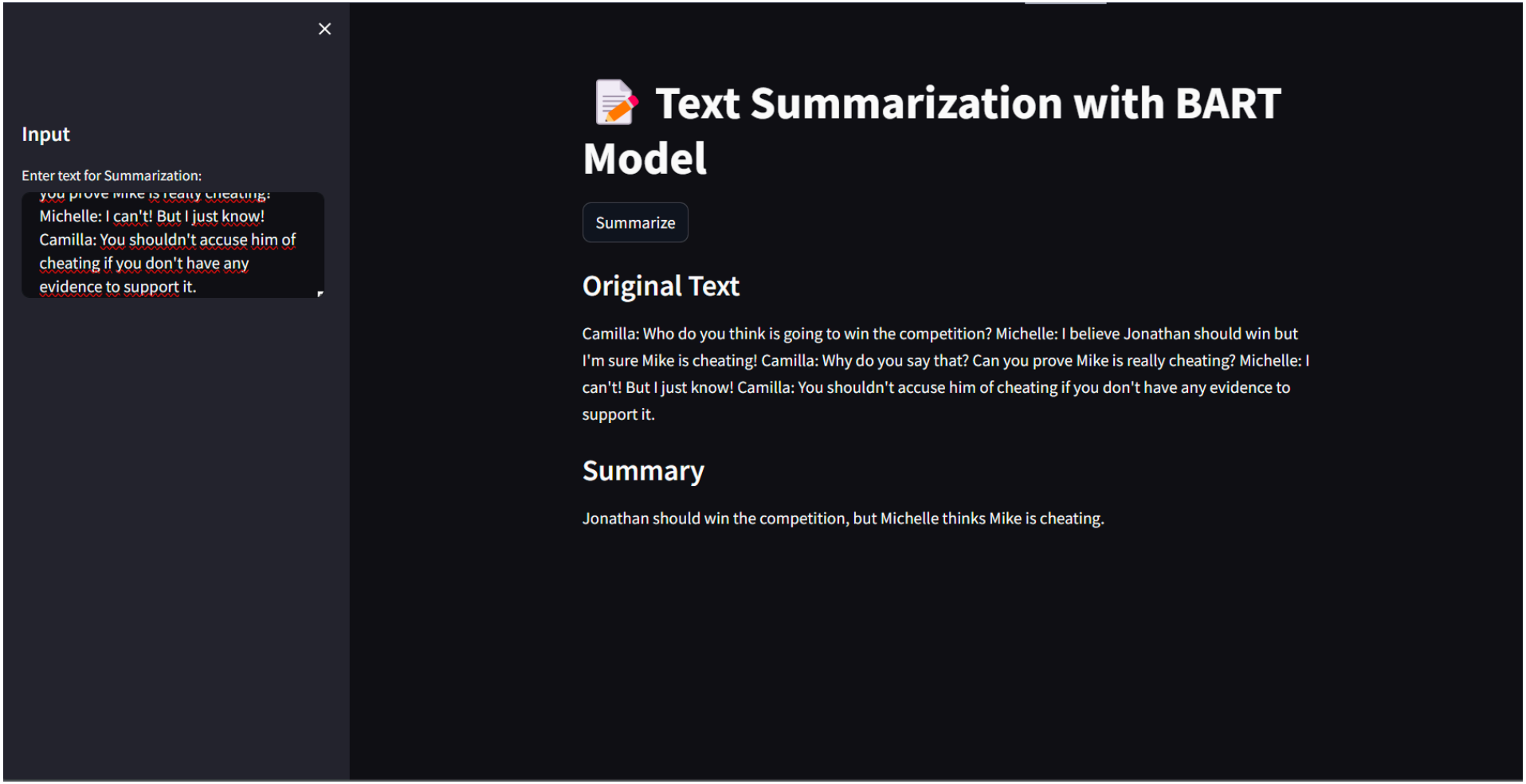
We used a BART model that has been already trained to perform summarization on news articles and fine-tuned it to perform summarizations of dialogues with the **SamSum** dataset.

Thanks to Hugging Face's Models and Spaces, I have uploaded this model online, and it is free for anyone to use on their own summarization tasks or further fine-tune it on other tasks. I highly suggest you visit the [luisotorres/bart-finetuned-samsum (https://huggingface.co/luisotorres/bart-finetuned-samsum)](https://huggingface.co/luisotorres/bart-finetuned-samsum) for more information on how to use this model.

I have also built a **web app** where you can use the model for the summarization of dialogues and news articles. Below, you can see some images of the web app, which is also available for free on [Bart Text Summarization (https://huggingface.co/spaces/luisotorres/bart-text-summarization)](https://huggingface.co/spaces/luisotorres/bart-text-summarization).



Example of Summarization of News Article



Example of Summarization of Dialogue

I hope that this notebook serves as a good introduction for those interested in the use of LLMs for Natural Language Processing tasks, as well as for those who already work with them and are in search of refining their knowledge on the subject.

This notebook took quite a while to be made and I highly appreciate your feedback on this work. Feel free to leave your comments, suggestions, and upvotes if you liked the content presented here.

Thank you very much!

Luis Fernando Torres, 2023

Let's connect!📎
LinkedIn (https://www.linkedin.com/in/luuisotorres/) • Medium (https://medium.com/@luuisotorres) • Hugging Face (https://huggingface.co/luisotorres)