Installing dependencies

In [1]:
```
! pip install pandas plotly
```

```
Requirement already satisfied: pandas in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-packages (1.4.4)
Requirement already satisfied: plotly in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-packages (5.10.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-pa
ckages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.18.5 in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-packages (f
rom pandas) (1.23.2)
Requirement already satisfied: pytz>=2020.1 in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-packages (fr
om pandas) (2022.2.1)
Requirement already satisfied: tenacity>=6.2.0 in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-packages
(from plotly) (8.0.1)
Requirement already satisfied: six>=1.5 in /home/vageesh/miniconda3/envs/ht/lib/python3.9/site-packages (from p
ython-dateutil>=2.8.1->pandas) (1.16.0)
```

Importing libraries

In [2]:
```python
import os
import pickle
import operator

import pandas as pd

import plotly
import plotly.express as px
import plotly.graph_objects as go
```

Analyzing advertisements on grammatical and syntactical levels

In [3]:
```python
def load_data(mode="pos"):
    """
    :param mode can only be pos or dep. The "pos" mode loads the data to generate the anaylsis on grammatical l
    mode loads the data to generate the analysis on syntactical level. Both, pos and dep features are obtained
    for each token in the advertisements.
    """

    if mode == "pos":
        # Loading the pos data
        with open(os.path.join(os.getcwd(), 'data/eurekarebellionaus_pos2vocab.pickle'), 'rb') as handle:
            parent_vendor_vocab = pickle.load(handle)
        with open(os.path.join(os.getcwd(), 'data/eurekarebellionaus_pos2den.pickle'), 'rb') as handle:
            parent_vendor_density = pickle.load(handle)

        with open(os.path.join(os.getcwd(), 'data/eurekarebellion_pos2vocab.pickle'), 'rb') as handle:
            alias1_vocab = pickle.load(handle)
        with open(os.path.join(os.getcwd(), 'data/eurekarebellion_pos2den.pickle'), 'rb') as handle:
            alias1_density = pickle.load(handle)

        with open(os.path.join(os.getcwd(), 'data/stimz_pos2vocab.pickle'), 'rb') as handle:
            alias2_vocab = pickle.load(handle)
        with open(os.path.join(os.getcwd(), 'data/stimz_pos2den.pickle'), 'rb') as handle:
            alias2_density = pickle.load(handle)

    elif mode == "dep":
        # Loading the dep data
        with open(os.path.join(os.getcwd(), 'data/eurekarebellionaus_dep2vocab.pickle'), 'rb') as handle:
            parent_vendor_vocab = pickle.load(handle)
        with open(os.path.join(os.getcwd(), 'data/eurekarebellionaus_dep2den.pickle'), 'rb') as handle:
            parent_vendor_density = pickle.load(handle)

        with open(os.path.join(os.getcwd(), 'data/eurekarebellion_dep2vocab.pickle'), 'rb') as handle:
            alias1_vocab = pickle.load(handle)
        with open(os.path.join(os.getcwd(), 'data/eurekarebellion_dep2den.pickle'), 'rb') as handle:
            alias1_density = pickle.load(handle)

        with open(os.path.join(os.getcwd(), 'data/stimz_dep2vocab.pickle'), 'rb') as handle:
            alias2_vocab = pickle.load(handle)
        with open(os.path.join(os.getcwd(), 'data/stimz_dep2den.pickle'), 'rb') as handle:
            alias2_density = pickle.load(handle)

    else:
        raise NotImplementedError

    return parent_vendor_vocab, parent_vendor_density, alias1_vocab, alias1_density, alias2_vocab, alias2_densi
```

In [4]:
```python
def generate_scatter_plot(parent_vendor_vocab, parent_vendor_density, alias_vocab, alias_density, parent_vendor
        fig = go.Figure()

        dict_parent, dict_alias = ({} for i in range(2))

        # Plotting top-n tokens
        for pos, token_dict in parent_vendor_vocab.items():
            if token_dict != None:
                dict_parent[pos] = dict(sorted(parent_vendor_vocab[pos].items(), key=operator.itemgetter(1), re

        for pos, token_dict in alias_vocab.items():
            if token_dict != None:
                dict_alias[pos] = dict(sorted(alias_vocab[pos].items(), key=operator.itemgetter(1), reverse=Tru

        # Plotting the bar plot
        fig.add_trace(go.Bar(x=list(parent_vendor_density.keys()), y=list(parent_vendor_density.values()),
                             name=parent_vendor_name, marker_color="red", opacity=0.3))
        fig.add_trace(go.Bar(x=list(alias_density.keys()), y=list(alias_density.values()),
                             name=alias_name, marker_color="blue", opacity=0.3))

        for feature, token_freq in dict_parent.items():
            fig.add_trace(go.Scatter(x=[feature]*len(list(dict_parent.keys())),
                                     y=list(token_freq.values()), text=list(token_freq.keys()),
                                     mode='text', marker_color="red", showlegend=False,
                                     textfont={'color':"red"}))

        for feature, token_freq in dict_alias.items():
            fig.add_trace(go.Scatter(x=[feature]*len(list(dict_alias.keys())),
                                     y=list(token_freq.values()), text=list(token_freq.keys()),
                                     mode='text', marker_color="blue", showlegend=False,
                                     textfont={'color':"blue"}))


        fig.update_layout(
            title= mode.upper() + " distribution plot",
            xaxis_title = mode.upper(),
            yaxis_title="Frequency",
            font=dict(
                family="Courier New, monospace",
                size=16,
                color="RebeccaPurple"
            )
            )

        fig.show()
```

POS Distribution plot :- Analyses Grammatical patterns

The POS distribution plots analyze the preferred grammatical patterns and the n-most frequent words used within each POS-categories by the two authors. For the sake of visibility, we choose to plot the 5-most active and 5-least active POS tags for every vendor.

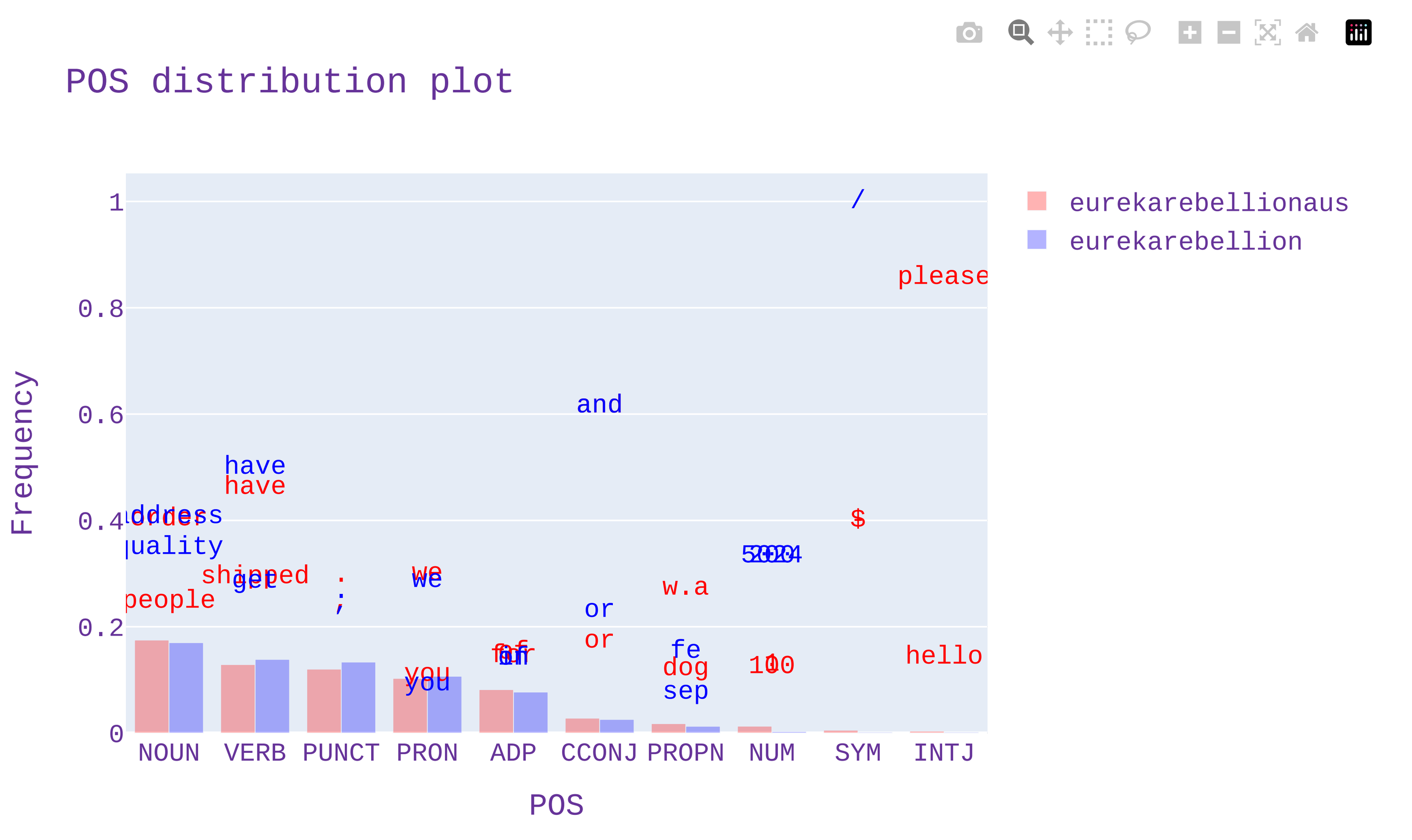
After analyzing multiple vendor-alias pair, we found a pattern that suggests that vendors with high representational similarity in vendor identification task often had similar grammatical structure in their advertisements.

Note: Both POS and DEP features are collected for advertisements of vendors in the same category through the spacy library. We apologize the reviewer's for the messy plots due to the time constraint between the release of reviews and end of rebuttal period. In the coming future, we will try to make these plots prettier and add them to our work in the final draft.

In [5]:
```python
parent_vendor_vocab, parent_vendor_density, alias1_vocab, alias1_density, alias2_vocab, alias2_density = load_d
```

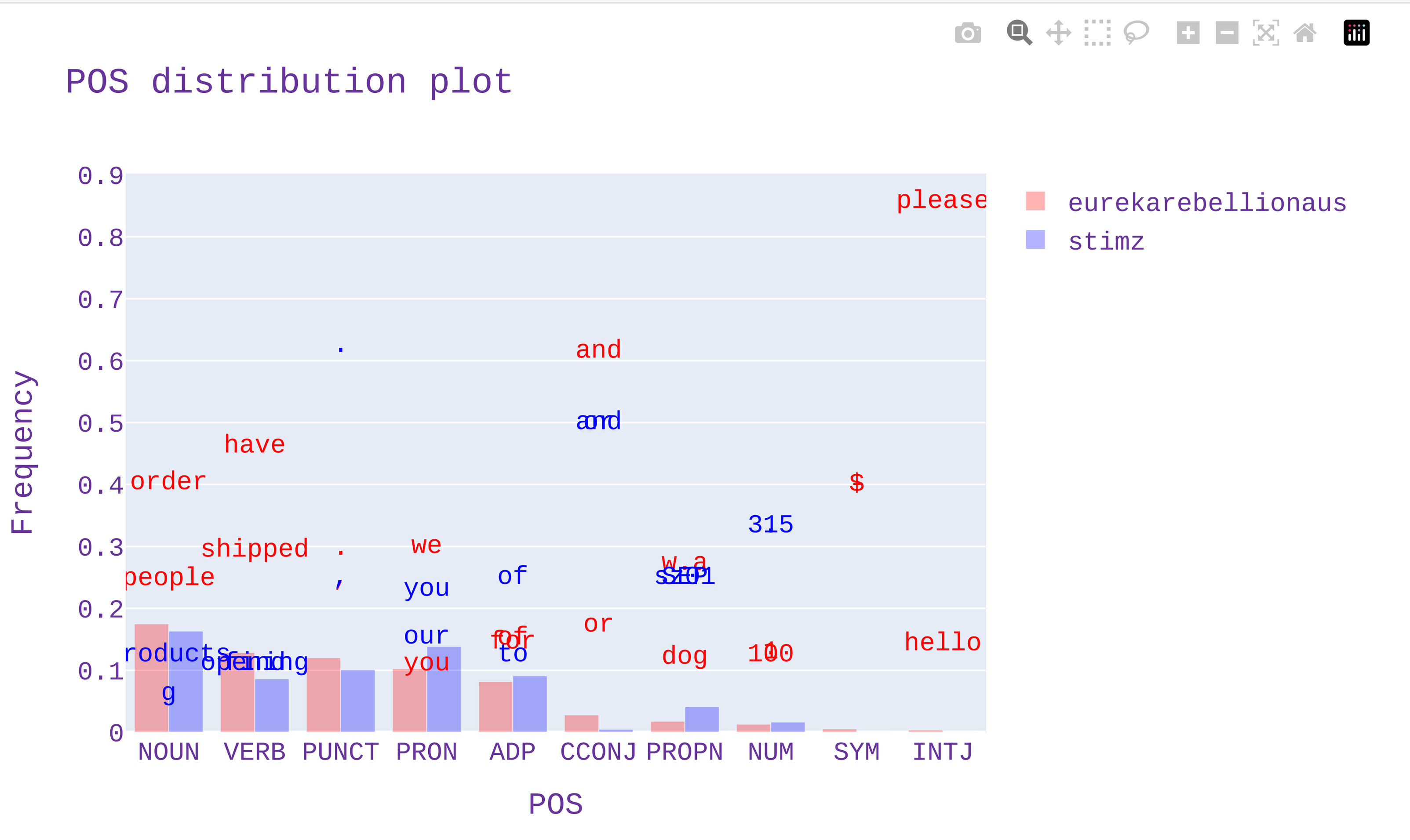
- As seen in the first plot, both vendors, eurekarebellionaus and eurekarebellion, prefer similar grammatical structures in their advertisements. The most activated POS tag in their advertisements is NOUN, followed by VERB, PUNT, PRON, and so on.
- Although the frequency of these tags changes, the preference of the POS tags remains the same.
- We plot the 2-most frequent tokens / POS tag for every vendor. Since we use Plotly, it provides the users an interface to zoom infinitely; therefore, in practice, all the tokens can be plotted and compared for any two vendors. However, for the sake of the visibility of reviewers, we decided to stick with just the 2-most frequent tokens for now.
- The 5-most frequent token analyses shows that both these vendors also prefer similar unigram vocabulary distribution.
- All in all, the first plot indicated high similarity in the advertisements of both the vendors on grammatical and vocabulary level.

In [6]:
```python
generate_scatter_plot(parent_vendor_vocab, parent_vendor_density, alias1_vocab, alias1_density, parent_vendor_n
```



- The second plot shows how differently the vendors eurekarebellionaus and stimz prefer different grammatical structures in their advertisements.
    - The most activate POS tags for eurekarebellionaus is NOUN, followed by VERB, PUNT, PRON, and ADP. Whereas, the vendor stimz prefer PRON more than PUNCT and VERB.
    - The n-most frequent token analyses points to inconclusive results as it is difficult to make out if the vendors are talking about the same thing.

In [7]:
```python
generate_scatter_plot(parent_vendor_vocab, parent_vendor_density, alias2_vocab, alias2_density, parent_vendor_n
```



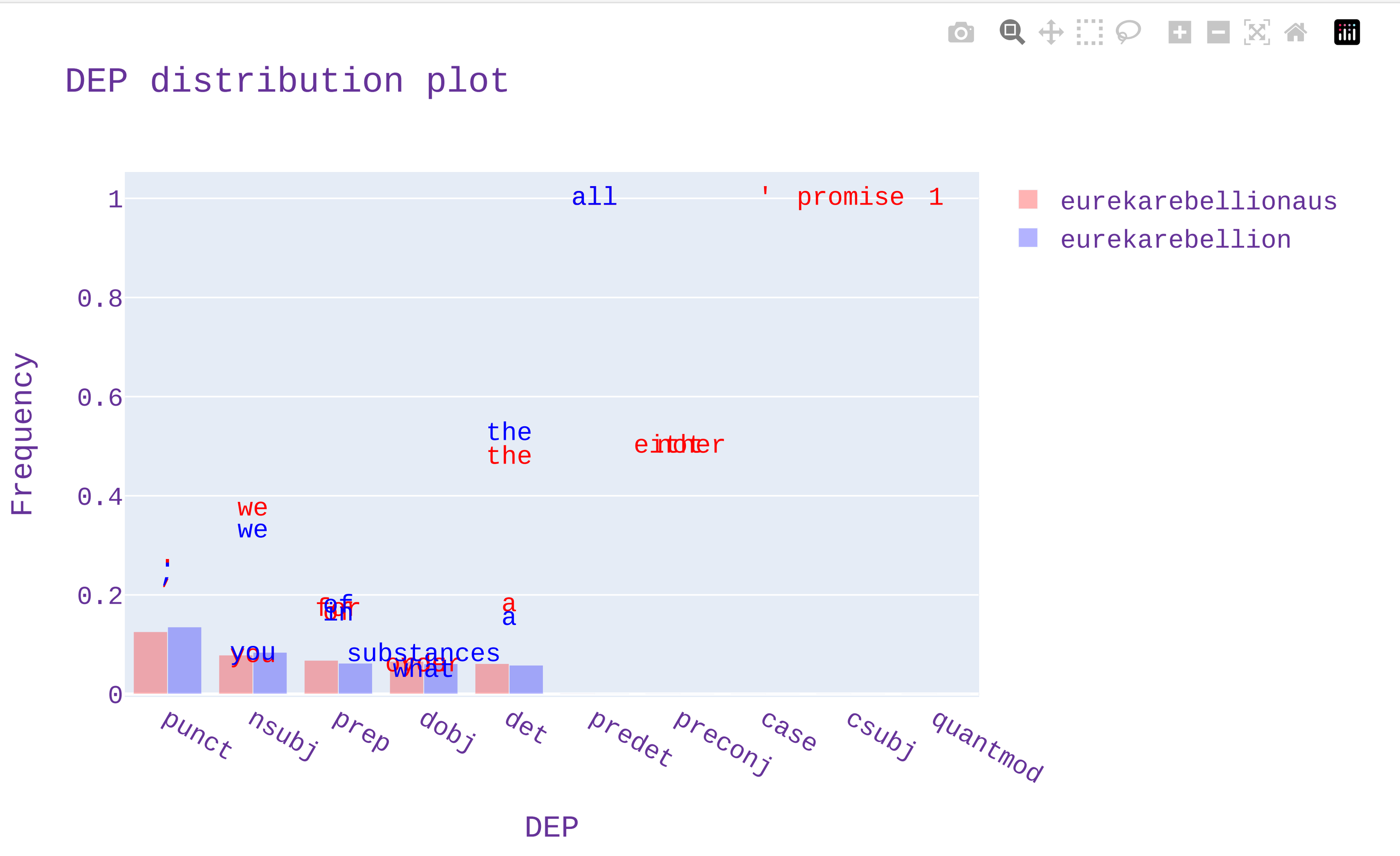
DEP Distribution plot :- Analyses syntactical patterns

Similar to the experiments above, the DEP distribution plot analyze the preferred syntactical patterns and the n-most frequent words used within each DEP-categories by the two authors. For the sake of visibility, we choose to plot the 5-most active and 5-least active DEP tags for every vendor.

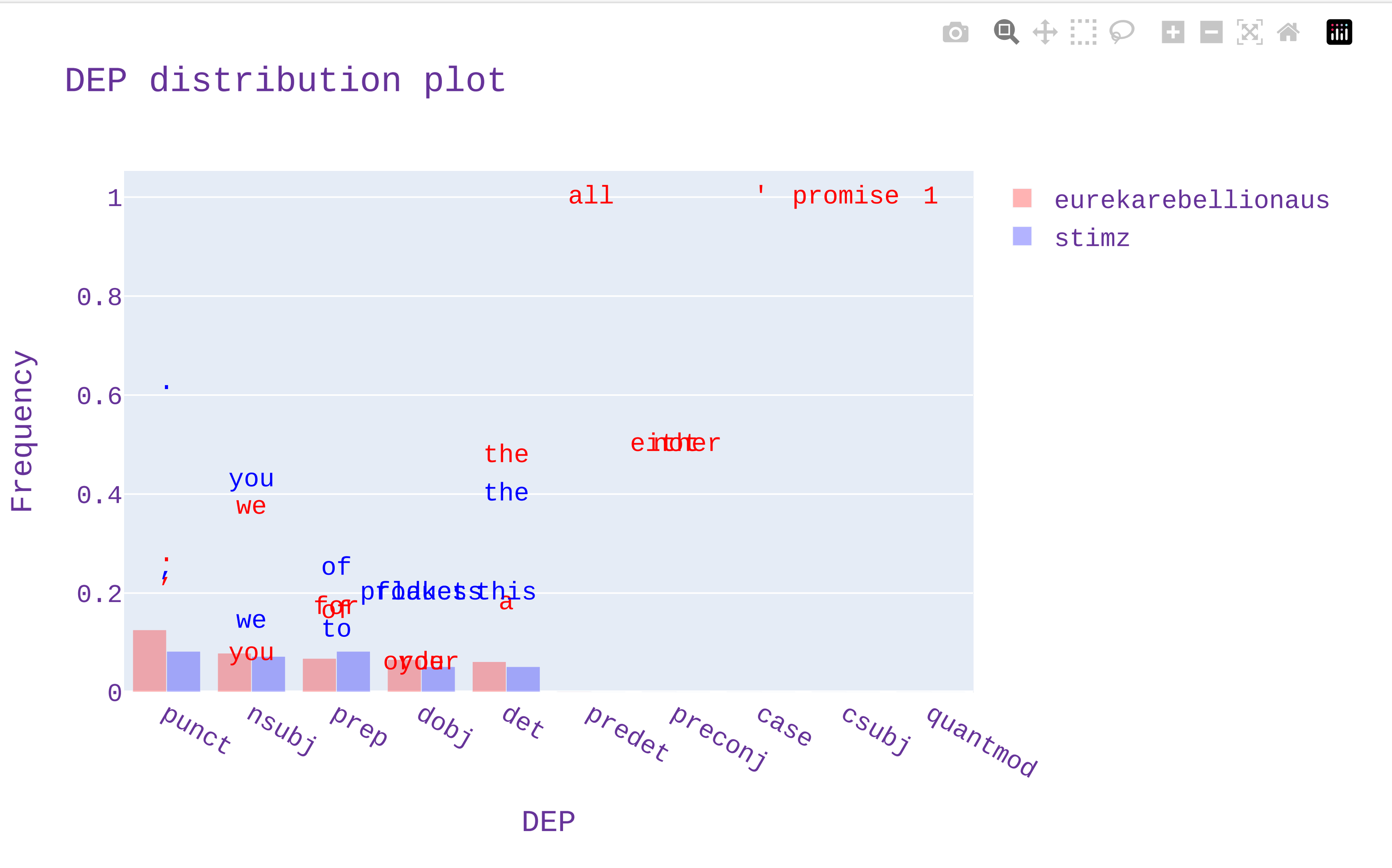
The analyses below show consistent pattern of similarity in the advertisements as mentioned above even on the DEP level.

In [8]:
```python
parent_vendor_vocab, parent_vendor_density, alias1_vocab, alias1_density, alias2_vocab, alias2_density = load_d
```

In [9]:
```python
generate_scatter_plot(parent_vendor_vocab, parent_vendor_density, alias1_vocab, alias1_density, parent_vendor_n
```



In [10]:
```python
generate_scatter_plot(parent_vendor_vocab, parent_vendor_density, alias2_vocab, alias2_density, parent_vendor_n
```



Loading Ngram data

In [11]:
```python
# unigram,  bigram, and trigram data
ngram_data = pd.read_csv("data/ngram_comp.csv").drop(['Unnamed: 0'], axis=1)
```

The dataframe below indicates the % of unigrams, bigrams, and trigrams features common to the advertisements of both vendors. To make the analysis fair, we only compare the advertisements across the same categories of the advertisements.

- As can be seen in the first example, both eurekarebellionaus-eurekarebellion share a heavy proportion of unigrams, bigrams, and trigrams features indicating a high similarity in their advertisements.
- In the third example, planet-pluto - planetpluto share completely similar unigrams, bigrams, and trigrams features indicating identical advertisements. Although, when checked for other categories, the results came out to be really low indicating that the n-gram feature technique cannot reliably comment of the similarity in the vendors.
- While the vendors houseofdank-houseofdank2.0 share a considerable proportion of unigram features, the bigrams and trigrams result indicates the inconclusiveness of the results for similarity analysis.

In [12]:
```python
ngram_data
```

Out[12]:

	vendors	unigram	bigrams	trigrams
0	eurekarebellionaus - eurekarebellion	0.9725	0.9314	0.9228
1	eurekarebellionaus - stimz	0.3783	0.0313	0.0046
2	planet-pluto - planetpluto	1.0000	1.0000	1.0000
3	houseofdank-houseofdank2.0	0.7003	0.2773	0.1136
4	topgear - topgear69	0.3899	0.0650	0.0044
5	aussieimportpills - aussieimportpillsv2	0.2618	0.0135	0.0000

In [ ]: