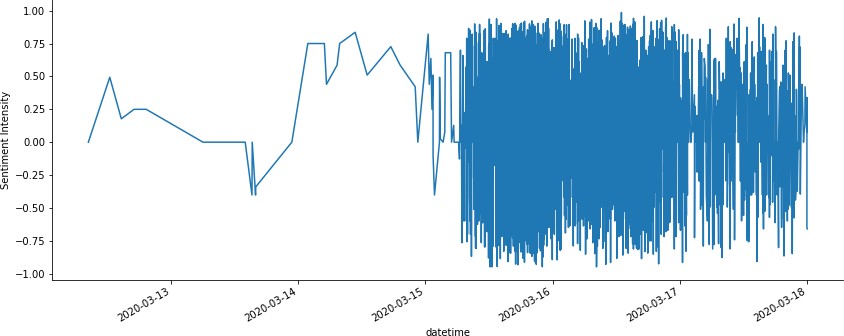
Emotional Sentiment on Twitter

# A coronavirus vaccine online firestorm



The ongoing competition for a viable vaccine against coronavirus is arguably the race of the century. With its hundred millions of users, Twitter is particularly well-suited for research into sentiment and emotions running in social media.

This is how it all begun: an exercise of 'real politiks' that is likely to change dramatically the way science, politics and business colide in a pos-covid19 world. As we will see below, the 15th March 2020 will go down in history as a shift of political tone that is at odds with the collaborative, responsible and ethical behaviour of scientific research.

I collected the data scraping tweets from Twitter’s application program inter-face (API), using TwitterScraping. Tweets were saved on a daily basis using the fol-lowing search term “Curevac”, the name of a German vaccine maker backed by Bill & Melinda Gates Foundation, and currently working on a Covid-19 vaccine. The post covers tweets from a 6-year period from March 3, 2014 to March 18, 2020 (N = 14,991).

The post covers tweets from a 6-year period from March 3, 2014 to March 18, 2020. Results include 15,036 tweets in a wide range of languages.

In this notebook you will find examples of some of the most common NLP (Natural Language Processing) techniques used to uncover patterns of sentiment and emotion in the kind of unstructured data that is predominant in Twitter. It is organized as follows:

Step 1: Exploratory analysis Step 2: Text processing

Step 3: Sentiment analysis Step 4: Word frequency Step 5: LDA topics extraction Step 6: Emotion analysis

# Step 1: EXPLORATORY ANALYSIS

After scrapping the Twitter API, the retained tweets were gathered in an excel file (tweets\_curevac.xlsx).

Below we have the major Python packages that are required for data handling (pandas), scientific computing (numpy) and data visualization (matplotlib and seaborn).

In [1]:

**import pandas as pd import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

%**matplotlib** inline

In [2]:

**from collections import** defaultdict

**from datetime import** date

**import re** *# for regular expressions*

**import string**

In [3]:

**import warnings**

warnings.filterwarnings("ignore", category=**DeprecationWarning**)

## Importing the data

In [4]:

tweets = pd.read\_csv('input/tweets.csv')

In [5]:

*# getting the date column ready for datetime operations*

tweets['datetime']= pd.to\_datetime(tweets['datetime'])

Here is a view of the first rows:

In [6]:

tweets.head()

|  |  |  |  |
| --- | --- | --- | --- |
| Out[6]: |  | | |
|  |  | **datetime** | **text** |
|  | **0** | 2014-03-12 18:26:59 | Robert-Jan Smits at Innovation Convention 2014... |
|  | **1** | 2014-03-13 09:50:54 | First #EU #vaccine prize awarded 2 CureVac ... |
|  | **2** | 2014-03-14 12:50:28 | Congrats 2 CureVac ! 4 #EU #vaccine prize #... |
|  | **3** | 2014-03-14 16:01:30 | MT @sanofiDE CureVac Wins Two Million EUR f... |

**4** 2014-03-14 17:44:32 CureVac wins EU's EUR2m inducement prize for ...

And here is a plot of the tweets with thw word "CureVac" over the past 6 years.

In [7]:

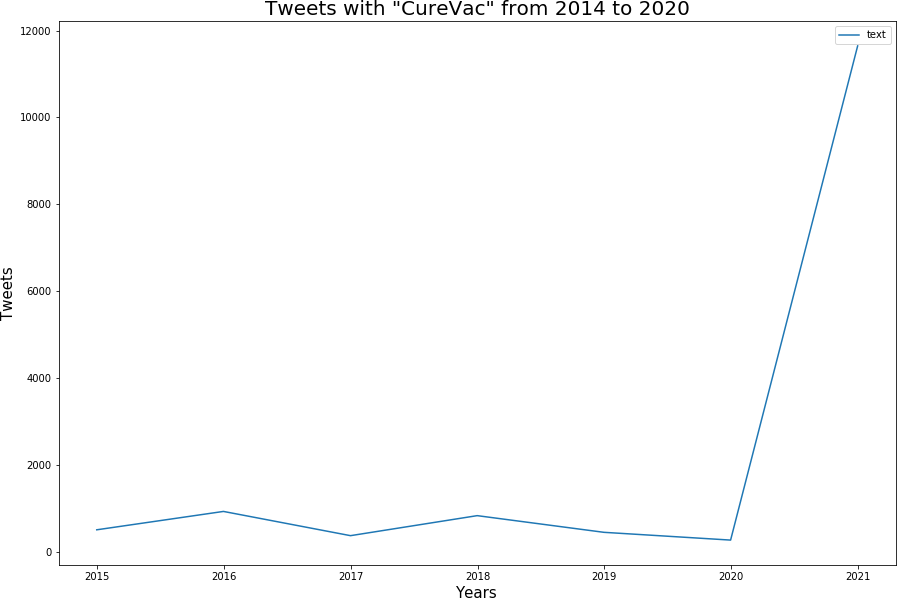
*# A simple timeseries plot*

fig = plt.figure(figsize=(15, 10))

ax = sns.lineplot(data=tweets.set\_index("datetime").groupby(pd.Grouper(fre q='Y')).count())

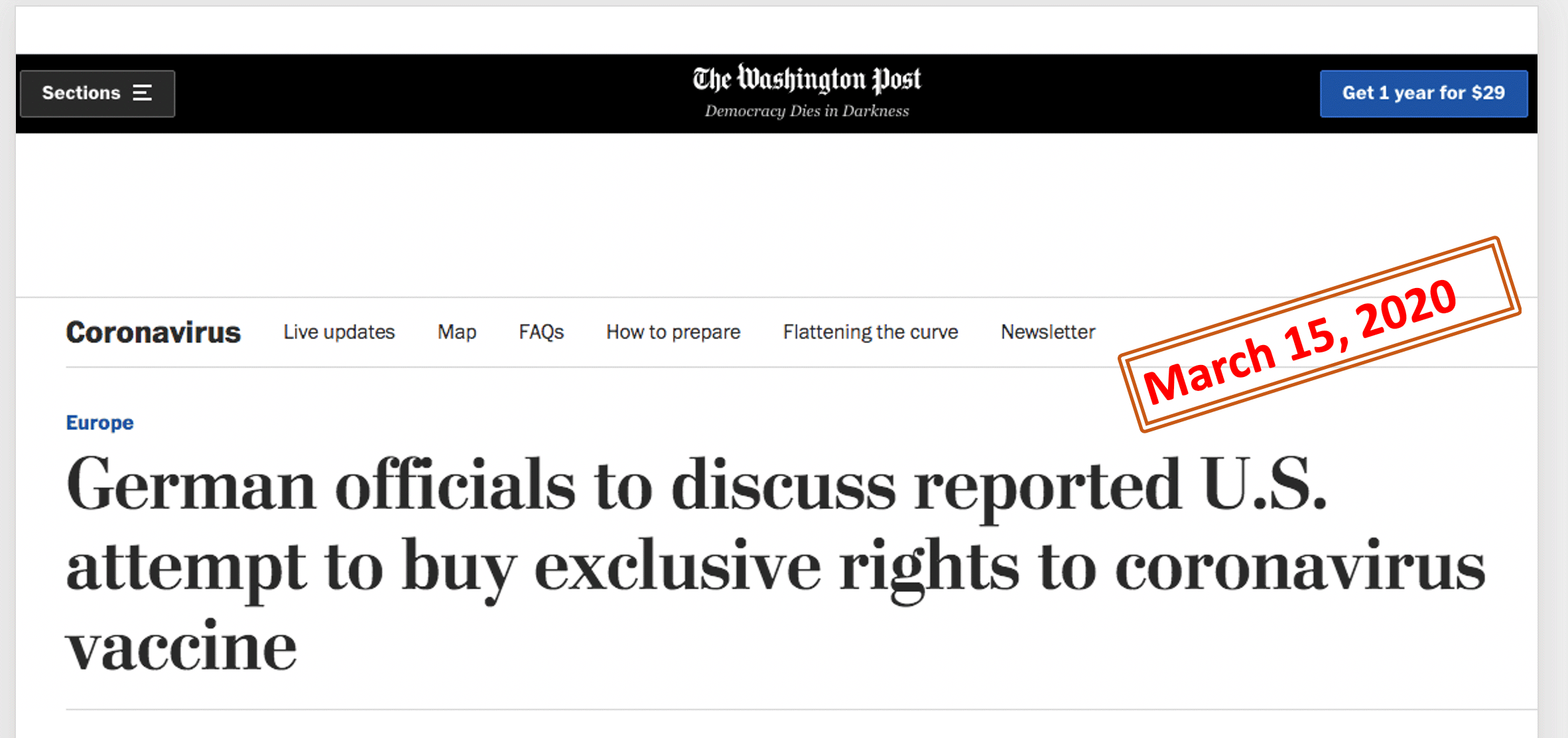
plt.title('Tweets with "CureVac" from 2014 to 2020', fontsize=20) plt.xlabel('Years', fontsize=15)

plt.ylabel('Tweets', fontsize=15) fig.savefig("images/All\_Tweets\_2014-2020.png")



For several years, the rate of tweets went on at a regular pace, until one day ... everything changed!

Digital marketing researchers call these events “online firestorms”, referring to negative word of mouth (eWOM) that suddenly attract thousands of expres-sions of support from other clients through social [1].



Let us create a column to identify this three-days event.

In [8]:

*# creating a column to filter the onlinestorm (from 15 and 18 March)*

**def** make\_onlinestorm\_field():

**for** i, row **in** tweets.iterrows():

**if** pd.to\_datetime(tweets.at[i, 'datetime']) > pd.Timestamp(date(20 20,3,15)):

tweets.at[i, 'onlinestorm'] = **True else**:

tweets.at[i, 'onlinestorm'] = **False**

make\_onlinestorm\_field()

In [9]:

*# count tweets during the three days online storm*

print('In three days, tweets went over **{}**, all around the world.'.format(t weets[tweets['onlinestorm']]['onlinestorm'].count()))

In three days, tweets went over 11364, all around the world.

Here we have them ..

In [10]:

tweets[tweets['onlinestorm']]

Out[10]:

**datetime text onlinestorm**

**3627**

2020-03-15

00:07:55

Germany's CureVac says low-dose coronavirus ...

True

**3628** 2020-03-15

00:14:42

ドイツのディ・ヴェルト紙（ネット版、3月15日）が伝えるところによる

と、米国政府がドイツでコ...

True

**3629**

2020-03-15

00:18:05

Germany's CureVac says low-dose coronavirus ...

True

**3630** 2020-03-15

00:30:29

A Bill & Melinda Gates funded, German Biotech ... True

**3631**

2020-03-15

00:42:51

.@BillGates .@gatesfoundation .@melindagates...

True

**...** ... ... ...

**14986**

2020-03-18

13:33:33

CureVac -Miteigentümer: Hopp macht Hoffnung au...

True

**14987** 2020-03-18

13:33:36

Non, c'était dans la presse allemande. L'actio... True

**14988**

2020-03-18

13:35:22

Rainer Hachfeld https:// cartoonmovement.sho...

True

**14989**

2020-03-18

13:36:14

ドイツのバイオテクノロジー企業である CureVac が新型コロナのワクチン

を秋までに開発す... True

**14990**

2020-03-18

13:38:00

Trad.: "Je n'ai pas parlé personnellement à M....

True

11364 rows × 3 columns

Let's have a look at the distribution of the tweets by the hour during the online storm.

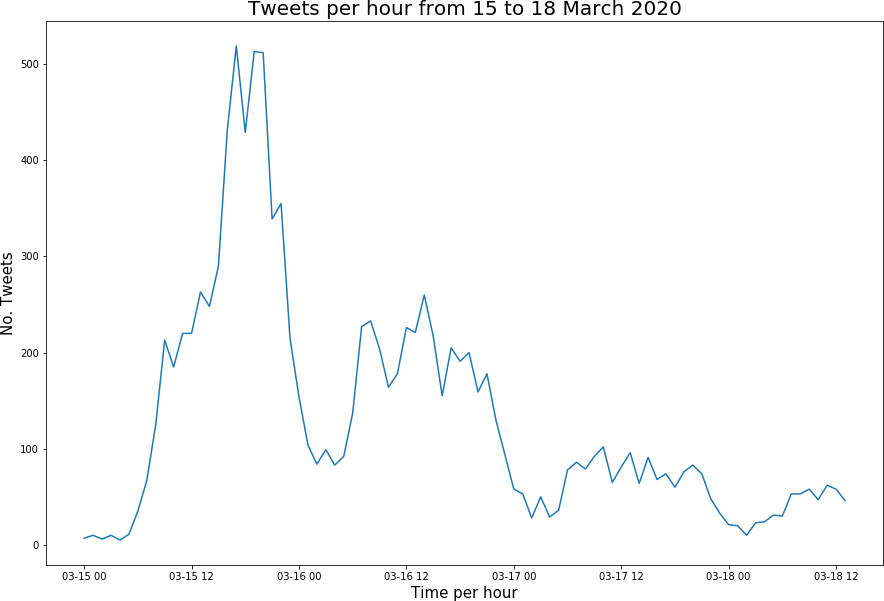
In [11]:

*# plot it*

fig = plt.figure(figsize=(15, 10))

ax = sns.lineplot(data=tweets[tweets['onlinestorm'] == **True**].set\_index("da tetime").groupby(pd.Grouper(freq='H')).onlinestorm.count()) plt.title('Tweets per hour from 15 to 18 March 2020', fontsize=20) plt.xlabel('Time per hour', fontsize=15)

plt.ylabel('No. Tweets', fontsize=15) fig.savefig("images/All\_Tweets\_Onlinestorm.png")



It is time to have a first look at the content of the tweets and do some descriptive statistics. For now, I will focus only on features like hastags, mentions, urls, capital words and words in general.

In [12]:

*# A function to count tweets based on regular expressions*

**def** count\_tweets(reg\_expression, tweet): tweets\_list = re.findall(reg\_expression, tweet) **return** len(tweets\_list)

In [13]:

*# Creating a dictionary to hold the counts*

content\_count = {

'words' : tweets['text'].apply(**lambda** x: count\_tweets(r'\w+', x)), 'mentions' : tweets['text'].apply(**lambda** x: count\_tweets(r'@\w+', x)), 'hashtags' : tweets['text'].apply(**lambda** x: count\_tweets(r'#\w+', x)), 'urls' : tweets['text'].apply(**lambda** x: count\_tweets(r'http.?://[^\s]+

[\s]?', x)),

}

In [14]:

df = pd.concat([tweets, pd.DataFrame(content\_count)], axis=1);df

Out[14]:

**datetime text onlinestorm words mentions hashtags urls**

**0**

2014-03-

12

18:26:59

Robert-Jan Smits at Innovation Convention

2014...

False

26

1

0

1

2014-03-

**1** 13

09:50:54

First #EU #vaccine prize awarded 2 False 23 0 4 0

CureVac ...

**2**

2014-03-

14

12:50:28

Congrats 2 CureVac ! 4 #EU #vaccine

prize #...

False

21

0

5

0

2014-03-

**3** 14

16:01:30

MT @sanofiDE CureVac Wins Two Million False 17 1 0 0

EUR f...

**4**

2014-03-

14

17:44:32

CureVac wins EU's EUR2m inducement

prize for ...

False

33

0

2

1

**...** ... ... ... ... ... ... ...

**14986**

2020-03-

18

13:33:33

CureVac -Miteigentümer: Hopp macht

Hoffnung au...

True

18

0

0

0

**14987**

2020-03-

18

13:33:36

**14988**

2020-03-

18

13:35:22

Rainer Hachfeld https:// cartoonmovement.sho...

True

42

4

2

0

Non, c'était dans la presse allemande. True 46 0 0 1

L'actio...

**14989**

2020-03-

18

13:36:14

ドイツのバイオテクノロジー企業である CureVac が新型コロナのワクチン

を秋までに開発す...

True 39 0 0 1

**14990**

2020-03-

18

13:38:00

Trad.: "Je n'ai pas parlé personnellement à

M....

True

62

0

0

0

14991 rows × 7 columns

## Tweets descriptive statistics

In [15]:

*# Display some descriptive statistics*

**for** key **in** content\_count.keys(): print()

print('Descriptive statistics for **{}**'.format(key)) print(df.groupby('onlinestorm')[key].describe())

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Descriptive | statistics for words  count mean | std | min | 25% | | 50% | 75% | max |
| onlinestorm  False | 3627.0 23.355390 | 10.731606 | 3.0 | 16.0 | | 20.0 | 29.0 | 113.0 |
| True | 11364.0 34.571718 | 15.048733 | 1.0 | 23.0 | | 36.0 | 45.0 | 147.0 |
| Descriptive | statistics for mentions count mean std | | min | 25% | 50% | 75% | max | |
| onlinestorm  False | 3627.0 0.304659 0.931811 | | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 | |
| True | 11364.0 0.205297 0.694837 | | 0.0 | 0.0 | 0.0 | 0.0 | 14.0 | |
| Descriptive | statistics for hashtags count mean std | | min | 25% | 50% | 75% | max | |
| onlinestorm  False | 3627.0 0.971602 1.611123 | | 0.0 | 0.0 | 0.0 | 2.0 | 13.0 | |
| True | 11364.0 1.071454 1.923787 | | 0.0 | 0.0 | 0.0 | 1.0 | 19.0 | |
| Descriptive | statistics for urls count mean | std | min | 25% | 50% | 75% | max | |
| onlinestorm  False | 3627.0 0.267990 | 0.480024 | 0.0 | 0.0 | 0.0 | 1.0 | 6.0 | |
| True | 11364.0 0.265488 | 0.449512 | 0.0 | 0.0 | 0.0 | 1.0 | 4.0 | |

In [16]:

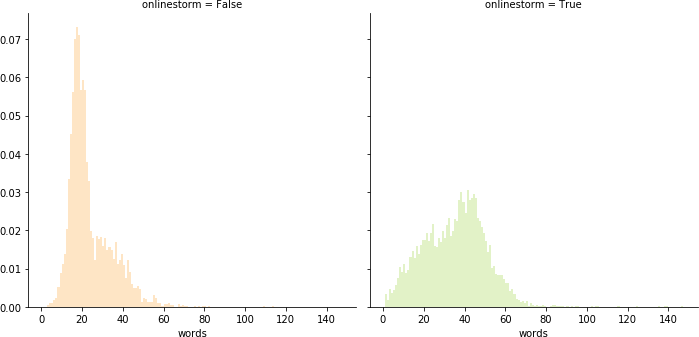
*# Now plot them*

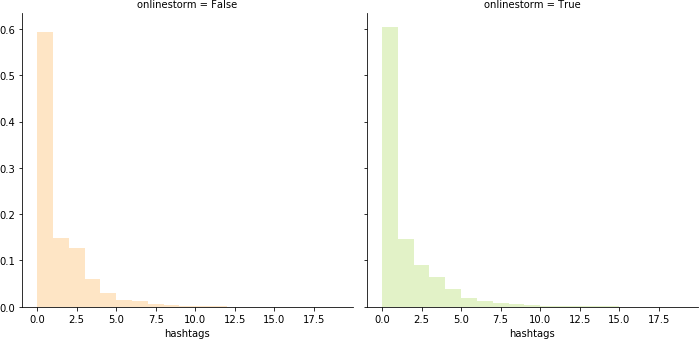
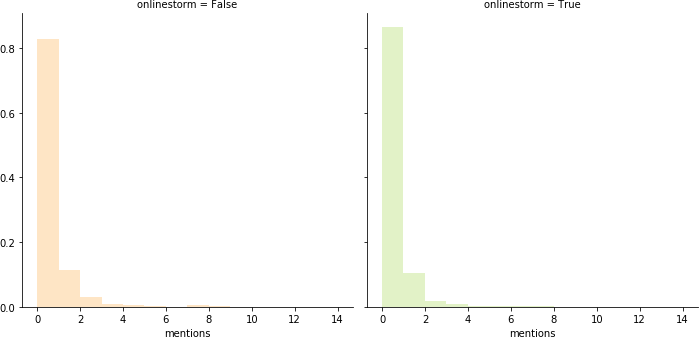
**for** key **in** content\_count.keys():

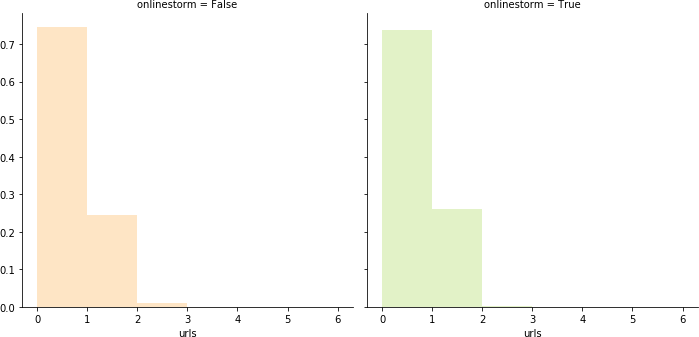
bins = np.arange(df[key].min(), df[key].max() + 1)

g = sns.FacetGrid(df, col='onlinestorm', height=5, hue='onlinestorm', palette="RdYlGn")

g = g.map(sns.distplot, key, kde=**False**, norm\_hist=**True**, bins=bins) plt.savefig('images/Descriptive\_stats\_for\_' + key + '.png')







From the above descriptive statistics, there are no noteworthy differences in terms of mentions, hashtags or urls during the online storm. Yet, the average number of words, per tweet, increased substantially during this period.

# Step 2: TEXT PROCESSING

For the next steps, I retained only the tweets in English, avoiding duplicates. These are contained in an excel file ('Tweets\_CureVac\_en.xlsx') with 6,546 tweets.

The second step of our analysis will look deeper into the content of these tweets. It is time to apply some of the basic NLP operations, such as cleaning, tokenizing and lemmatizing.

We will use NLTK (Natural Language Toolkit), one of the most popular NLP libraries for Python.

In [17]:

**import nltk**

**from nltk.corpus import** stopwords

**from nltk.stem.snowball import** SnowballStemmer

**from nltk.stem import** WordNetLemmatizer

**from nltk.tokenize import** sent\_tokenize, word\_tokenize

**from nltk import** pos\_tag

In [18]:

**import string**

**import re** *# for regular expressions*

In [19]:

*# I am adding my own stopwords list to the NLTK list.*

*# This way we can drop words that are irrelevant for text processing* MY\_STOPWORDS = ['curevac','vaccine','german','mrna','biotech','cancer','li lly','eli','ag','etherna\_immuno', 'translatebio', 'mooreorless62','boehrin ger', 'ingelheim','biopharmaceutical', 'company']

STOPLIST = set(stopwords.words('english') + list(MY\_STOPWORDS))

SYMBOLS = " ".join(string.punctuation).split(" ") + ["-", "...", "”", "``"

, ",", ".", ":", "''","#","@"]

In [20]:

*# The NLTK lemmatizer and stemmer classes*

lemmatizer = WordNetLemmatizer() stemmer = SnowballStemmer('english')

In [21]:

*# read english selected tweets, no duplicates*

tweets = pd.read\_csv('input/tweets\_en.csv')

In [22]: *# I use the POS tagging from NLTK to retain only adjectives, verbs, adverb s and nouns for lemmatization.*

**def** get\_lemmas(tweet):

*# A dictionary to help convert Treebank tags to WordNet*

treebank2wordnet = {'NN':'n', 'JJ':'a', 'VB':'v', 'RB':'r'}

postag = '' lemmas\_list = []

**for** word, tag **in** pos\_tag(word\_tokenize(tweet)):

**if** tag.startswith("JJ") \ **or** tag.startswith("RB") \ **or** tag.startswith("VB") \ **or** tag.startswith("NN"):

**try**:

postag = treebank2wordnet[tag[:2]]

### except:

postag = 'n' lemmas\_list.append(lemmatizer.lemmatize(word.lower(), postag))

**return** lemmas\_list

In [23]:

*# This function processes, cleans and filters the tokens for each tweet*

**def** clean\_tweet(tokens):

filtered = []

**for** token **in** tokens:

**if** re.search('[a-zA-Z]', token):

**if** token **not in** STOPLIST:

**if** token[0] **not in** SYMBOLS:

**if not** token.startswith('http'):

**if** '/' **not in** token:

**if** '-' **not in** token: filtered.append(token)

**return** filtered

We will now pre-process the tweets, following a pipeline of tokenization, filtering, case normalization and lemma extraction, including an overall cleaning of html and other codes.

Prior to lemmatization, I applied POS (part-of-speech) tagging to make sure that only the adjectives, verbs, adverbs and nouns were retained.

In [24]:

*# Starts the lemmatization process*

**def** get\_lemmatized(tweet):

all\_tokens\_string = '' filtered = []

tokens = []

*# lemmatize*

tokens = [token **for** token **in** get\_lemmas(tweet)]

*# filter*

filtered = clean\_tweet(tokens)

*# join everything into a single string*

all\_tokens\_string = ' '.join(filtered)

**return** all\_tokens\_string

In [25]:

*# get the lemmatized tweets and puts the result in an "edited" text column for future use*

edited = ''

**for** i, row **in** tweets.iterrows():

edited = get\_lemmatized(tweets.loc[i]['text'])

**if** len(edited) > 0: tweets.at[i,'edited'] = edited

**else**:

tweets.at[i,'edited'] = **None**

In [26]:

*# After lemmatization, some tweets may have as a result the same words # Let's make sure that we have no duplicates* tweets.drop\_duplicates(subset=['edited'], inplace=**True**) tweets.dropna(inplace=**True**)

With these text processing steps, and the removal of duplicates, the final sample counts 5,508 English-language tweets, with an average of 30 words (SD 12.5, ranging from 4 to 61 words).

In [27]:

*# Using apply/lambda to create a new column with the number of words in ea ch tweet*

tweets['word\_count'] = tweets.apply(**lambda** x: len(x['text'].split()),axis= 1)

t = pd.DataFrame(tweets['word\_count'].describe()).T t

Out[27]:

**count mean std min 25% 50% 75% max**

**word\_count** 5508.0 30.546659 12.499944 4.0 19.0 31.0 42.0 61.0

Here is the result of our pre-processing, showing the difference between the original tweet (column "text") and the lemmatized, cleaned, tweet (column "edited").

In [28]:

tweets.head()

Out[28]:

**datetime text edited word\_count**

**0**

2020-03-16

11:13:00

Interior Minister Horst Seehofer, when interior minister horst seehofer ask confirm

asked t...

r...

47

**1** 2020-03-16

03:34:02

**2**

2015-07-07

13:24:10

CureVac Announces Phase I/IIa Clinical

Study ...

announces phase clinical study data

immunother...

15

CureVac said it has been in contact with

many...

say contact many organization global

authority... 26

**3** 2015-11-02

13:21:08

CureVac alone has received >$100M

from US fou...

alone receive us foundation gate darpa

manufac... 21

**4**

2020-03-16

20:10:33

I hope to God CureVac doesn't cave.

Trump is...

hope god n't cave trump despicable po

planet

20

# Step 3: SENTIMENT ANALYSIS

For sentiment analysis -- a growing sub-field of Natural Language Processing (NLP) -- I used VADER (Valence Aware Dictionary for Sentiment Reasoning), a rule-based system that performs specially well on social media data.

VADER quantifies the sentiment of a tweet based on positive, neutral and neg-ative scores, and uses a Compound score to account for the intensity of the overall sentiment of a given text. It measures the strength of sentiments by giving scores ranging from +1 to -1 with positive numbers indicate favorable atti-tudes while negative numbers indicate negative attitudes.

In this section, I will focus on a simple comparison between the sentiment (Compound score) of a 6-year period with the score over the three days of the CureVac online firestorm, from March 15 to March 18.

Let us import the VADER analyser.

In [29]:

**from nltk.sentiment.vader import** SentimentIntensityAnalyzer

For the puropose of the timeseries analysis, we must make sure that the tweets are all correctly sorted.

In [30]:

tweets['datetime']=pd.to\_datetime(tweets['datetime']) tweets.sort\_values('datetime', inplace=**True**, ascending=**True**) tweets = tweets.reset\_index(drop=**True**)

In [ ]:

In [31]:

*# Creating a column to "filter" the online storm period.*

make\_onlinestorm\_field()

In [32]:

*# To avoid repetitions in our code, here are some plottimng functions # that will be called often ...*

**def** plot\_sentiment\_period(df, info):

*# Using the mean values of sentiment for each period*

df1 = df.groupby(df['datetime'].dt.to\_period(info['period'])).mean()

df1.reset\_index(inplace=**True**)

df1['datetime'] = pd.PeriodIndex(df1['datetime']).to\_timestamp() plot\_df = pd.DataFrame(df1, df1.index, info['cols'])

plt.figure(figsize=(15, 10))

ax = sns.lineplot(data=plot\_df, linewidth = 3, dashes = **False**) plt.legend(loc='best', fontsize=15)

plt.title(info['title'], fontsize=20) plt.xlabel(info['xlab'], fontsize=15) plt.ylabel(info['ylab'], fontsize=15) plt.tight\_layout() plt.savefig('images/' + info['fname']) **return**

**def** plot\_fractions(props, title, fname):

plt1 = props.plot(kind='bar', stacked=**False**, figsize=(16,5), colormap= 'Spectral')

plt.legend(bbox\_to\_anchor=(1.005, 1), loc=2, borderaxespad=0.) plt.xlabel('Online storm', fontweight='bold', fontsize=18) plt.xticks(rotation=0,fontsize=14)

*#plt.ylim(0, 0.5)*

plt.ylabel('Fraction of Tweets', fontweight='bold', fontsize=18) plt1.set\_title(label=title, fontweight='bold', size=20) plt.tight\_layout()

plt.savefig('images/' + fname + '.png')

### return

**def** plot\_frequency\_chart(info):

fig, ax = plt.subplots(figsize=(14, 8)) sns.set\_context("notebook", font\_scale=1)

ax = sns.barplot(x=info['x'], y=info['y'], data=info['data'], palette= (info['pal']))

ax.set\_title(label=info['title'], fontweight='bold', size=18) plt.ylabel(info['ylab'], fontsize=16) plt.xlabel(info['xlab'], fontsize=16) plt.xticks(rotation=info['angle'],fontsize=14) plt.yticks(fontsize=14)

plt.tight\_layout() plt.savefig('images/' + info['fname'])

### return

In [33]:

*# Calling VADER*

analyzer = SentimentIntensityAnalyzer()

In [34]:

*# get VADER Compound value for sentiment intensity*

tweets['sentiment\_intensity'] = [analyzer.polarity\_scores(v)['compound'] **f or** v **in** tweets['edited']]

The output of VADER are the positive, negative, and neutral ratios of sentiment. The most useful metric in VADER is the Compound score. Basically, it is calculated by a sum of the scores of each word, normalized to yeld values between -1, the most extreme negative score, and +1, the most extreme positive.

From this normalized score, I will then create a categorical variable ("sentiment"), with an output of positive, negative and neutral ratios of sentiment, using the following thresholds:

Positive sentiment : (compound score >= 0.05).

Neutral sentiment : (compound score > -0.05) and (compound score < 0.05). Negative sentiment : (compound score <= -0.05)

In [35]:

*# This function returns the sentiment category*

**def** get\_sentiment(intensity):

**if** intensity >= 0.05:

**return** 'Positive'

**elif** (intensity >= -0.05) **and** (intensity < 0.05):

**return** 'Neutral'

**else**:

**return** 'Negative'

*# Using pandas apply/lambda to speed up the process*

tweets['sentiment'] = tweets.apply(**lambda** x: get\_sentiment(x['sentiment\_in tensity']),axis=1)

**The Online Storm**

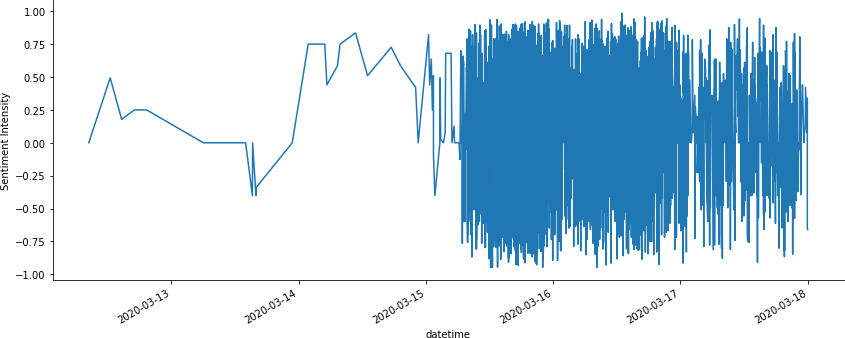
The next plot gives us a clear image of the “explosion” of contradictory sentiments in this period:

In [36]:

df=tweets.loc[:,['datetime','sentiment\_intensity']] df.set\_index('datetime',inplace=**True**) df=df[(df.index>='2020-03-12') & (df.index<'2020-03-18')] df.plot(figsize=(12,5));

plt.ylabel('Sentiment Intensity') plt.legend().set\_visible(**False**) plt.tight\_layout() sns.despine(top=**True**)

plt.savefig('images/Average\_sentiment\_during\_onlinestorm.png') plt.show();



And this one will shows us a comparison of the sentiments before and during the online strom.

In [37]:

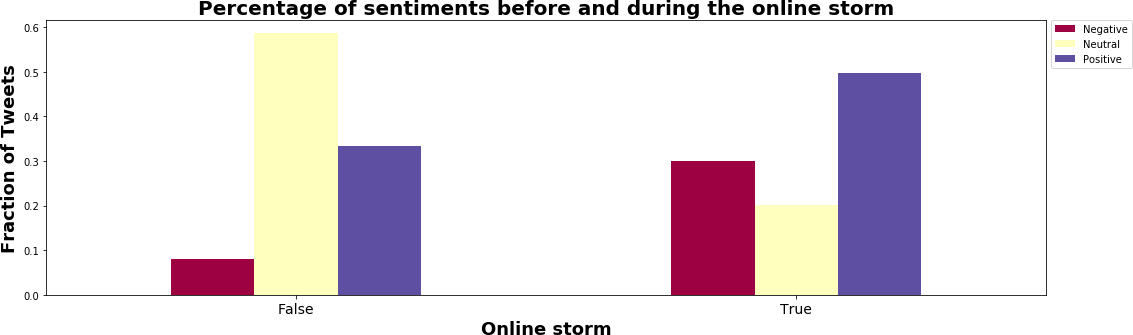
*# Values are normalized to take into account the number of tweets in each # of the two different periods*

props = tweets.groupby('onlinestorm')['sentiment'].value\_counts(normalize=

**True**).unstack()

plot\_fractions(props,'Percentage of sentiments before and during the onlin e storm',

'Fraction\_sentiments\_before\_and\_during\_onlinestorm')



In sentiment analysis and opinion mining, neutral tweets usually outnumber the negative or positive ones. This is what actually happened during the 6-year period in consideration. Moreover, research has been showing that scientists tend to use neutral language while communicating among peers, particularly in social media.

The picture clearly changed during the 3-days online storm. Sentiments become less neutral, as it is also likely that the majority of the authors come from a wider public. The percentage of positive tweets increased, suggesting increased expectations about a viable vaccine for coronavirus.

But the fraction of negative tweets increased even more during the online storm. This calls for a deeper look to the date. That is what we will do now.

# Step 4: Word frequency

Now that our text is pre-processed, it is time to examine key patterns of word frequency in tweets posted before and during the online storm.

In [ ]:

In [38]:

*# We need these imports for the wordcloud representation:*

**from PIL import** Image

**from wordcloud import** WordCloud, STOPWORDS, ImageColorGenerator

**from matplotlib.colors import** makeMappingArray

**from palettable.colorbrewer.diverging import** Spectral\_4

In [39]:

**from collections import** Counter

*st*

*# Look at the most common items in a li*

In [ ]:

In [40]:

**def** display\_wordcloud(tokens):

tokens\_upper = [token.upper() **for** token **in** tokens]

cloud\_mask = np.array(Image.open("images/cloud\_mask.png")) wordcloud = WordCloud(max\_font\_size=100,

max\_words=50, width=2500, height=1750,mask=cloud\_mask, background\_color="white").generate(" ".join(toke

ns\_upper))

plt.figure()

fig, ax = plt.subplots(figsize=(14, 8)) plt.imshow(wordcloud, interpolation="bilinear") plt.axis("off")

plt.show()

**return**

In [41]:

**def** join\_edited\_string(edited\_tweets):

edited\_string = ''

**for** row **in** edited\_tweets:

edited\_string = edited\_string + ' ' + row

**return** edited\_string

In [42]:

**def** get\_trigrams(trigrams, top\_grams):

grams\_str = [] data = []

gram\_counter = Counter(trigrams)

**for** grams **in** gram\_counter.most\_common(10): gram = ''

grams\_str = grams[0]

grams\_str\_count = []

**for** n **in** range(0,3):

gram = gram + grams\_str[n] + ' ' grams\_str\_count.append(gram) grams\_str\_count.append(grams[1]) data.append(grams\_str\_count) print(grams\_str\_count)

df = pd.DataFrame(data, columns = ['Grams', 'Count'])

**return** df

## Word frequency before the online storm

Let’s have a look at the 20 most frequent words in tweets before the online storm.

In [43]:

*# Filtering the tweets of the 6 years before the online storm*

df = tweets[tweets['onlinestorm'] == **False**]

*# Join all the edited tweets in one single string*

joined\_string = join\_edited\_string(df['edited'])

*# Get tokens*

tokens = joined\_string.split(' ')

*# get trigrams*

trigrams = nltk.trigrams(tokens)

In [44]:

*# plot word frequency during online storm*

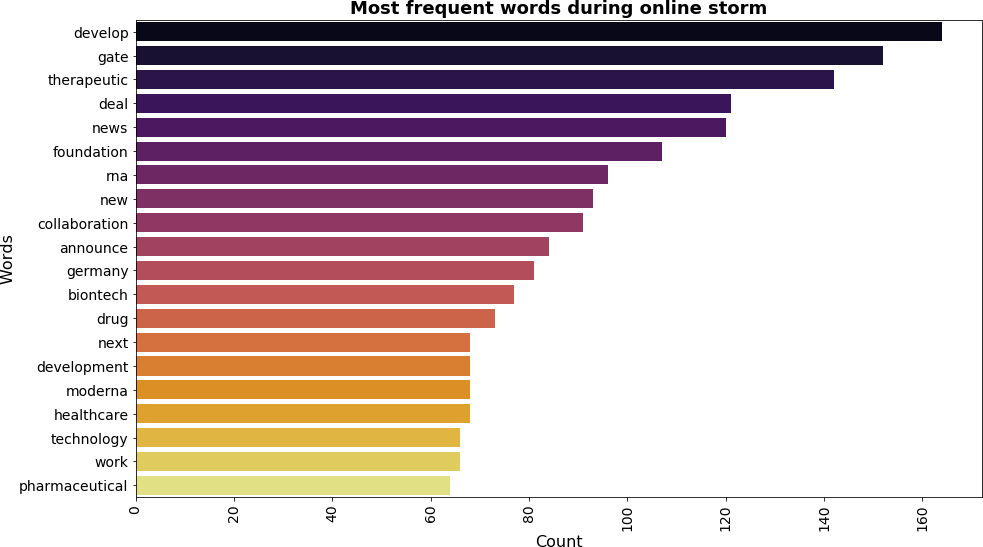
word\_counter = Counter(tokens)

df\_counter = pd.DataFrame(word\_counter.most\_common(20), columns = ['word', 'freq'])

info = {'data': df\_counter, 'x': 'freq', 'y': 'word',

'xlab': 'Count', 'ylab': 'Words', 'pal':'inferno', 'title': 'Most frequent words during online storm', 'fname':'word\_frequency\_during\_onlinestorm.png', 'angle': 90}

plot\_frequency\_chart(info)



And now the 10 most frequent trigrams (sequences of 3 consecutive words) ...

In [45]:

*# plot trigram frequency*

df\_trigrams = get\_trigrams(trigrams, 10)

info = {'data': df\_trigrams, 'x': 'Grams', 'y': 'Count',

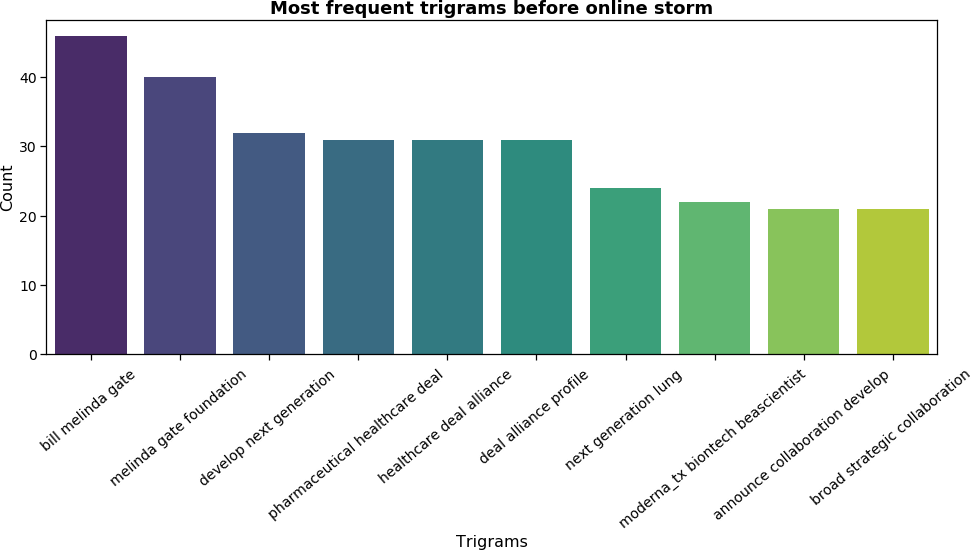
'xlab': 'Trigrams', 'ylab': 'Count', 'pal':'viridis', 'title': 'Most frequent trigrams before online storm', 'fname':'trigrams\_frequency\_before\_onlinestorm.png', 'angle': 40}

plot\_frequency\_chart(info)

['bill melinda gate ', 46] ['melinda gate foundation ', 40] ['develop next generation ', 32]

['pharmaceutical healthcare deal ', 31] ['healthcare deal alliance ', 31] ['deal alliance profile ', 31]

['next generation lung ', 24] ['moderna\_tx biontech beascientist ', 22] ['announce collaboration develop ', 21] ['broad strategic collaboration ', 21]

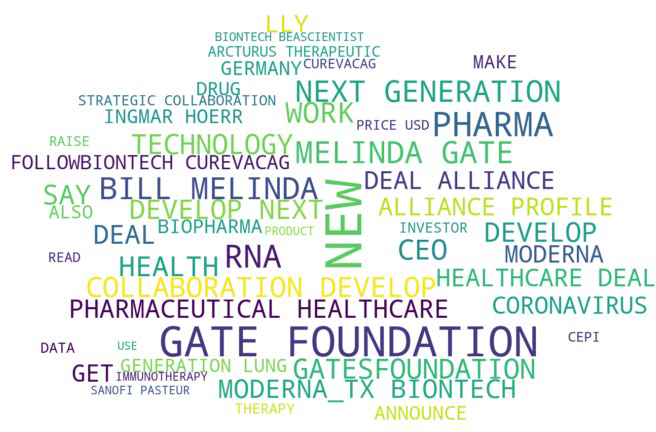


And the wordcloud ...

In [46]:

display\_wordcloud(tokens)

<Figure size 432x288 with 0 Axes>



There are some noteworthy features in these plots:

Along with ‘gate’ (ie., Bill Gates), the most frequent words in 6 years of tweets are ‘develop’, ‘therapeutic’, ‘deal’ and 'news'. Unsurprisingly, these were times when tweets were used mainly as public relations devices to communicate the core business of CureVac, a vaccine maker funded by the Melinda gate Foudation.

Immediatly follows ‘Collaboration’, the next most frequent word, reflecting in this way the key importance of partnerships in the strategy of the company, followed by ‘new’, as a evidence of CureVac's concern with innovation.

The trigrams reinforce these trends, and with a stronger focus on collaboration. These are mainly about 'next generation in health care' and 'pharmaceutical deals' carried out in ‘broad strategic collaborations’.

## Word frequency during the online storm

It is now time to examine what happened on those "stormy" three days, after the 15th March 2020 ...

In [47]:

*# Filtering the tweets of the 3 days of the online storm*

df =tweets[tweets['onlinestorm']]

*# Join all the edited tweets in one single string*

joined\_string = join\_edited\_string(df['edited'])

*# Get tokens*

tokens = joined\_string.split(' ')

*# get trigrams*

trigrams = nltk.trigrams(tokens)

In [48]:

*# plot word frequency during online storm*

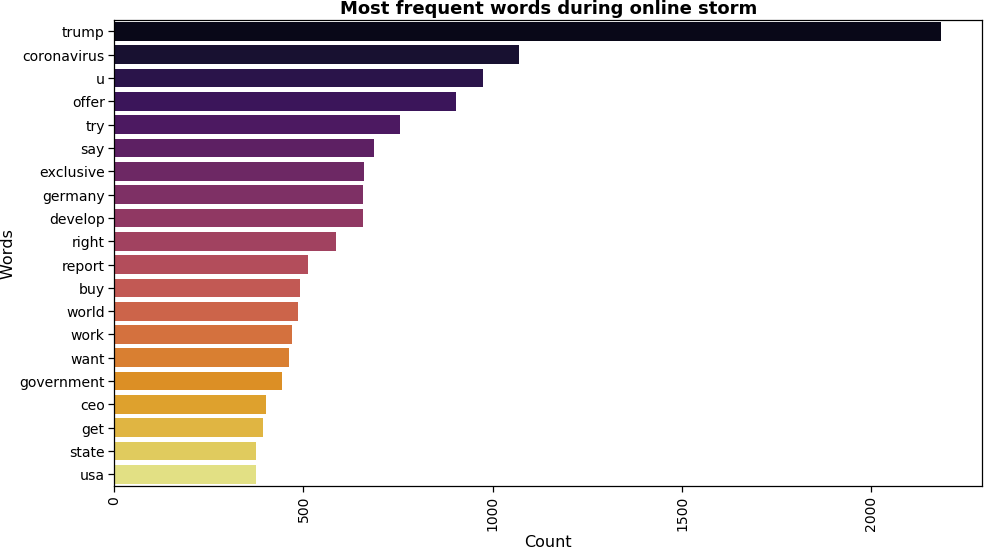
word\_counter = Counter(tokens)

df\_counter = pd.DataFrame(word\_counter.most\_common(20), columns = ['word', 'freq'])

info = {'data': df\_counter, 'x': 'freq', 'y': 'word',

'xlab': 'Count', 'ylab': 'Words', 'pal':'inferno', 'title': 'Most frequent words during online storm', 'fname':'word\_frequency\_during\_onlinestorm.png', 'angle': 90}

plot\_frequency\_chart(info)



In [49]:

*# plot trigrams frequency*

df\_trigrams = get\_trigrams(trigrams, 10)

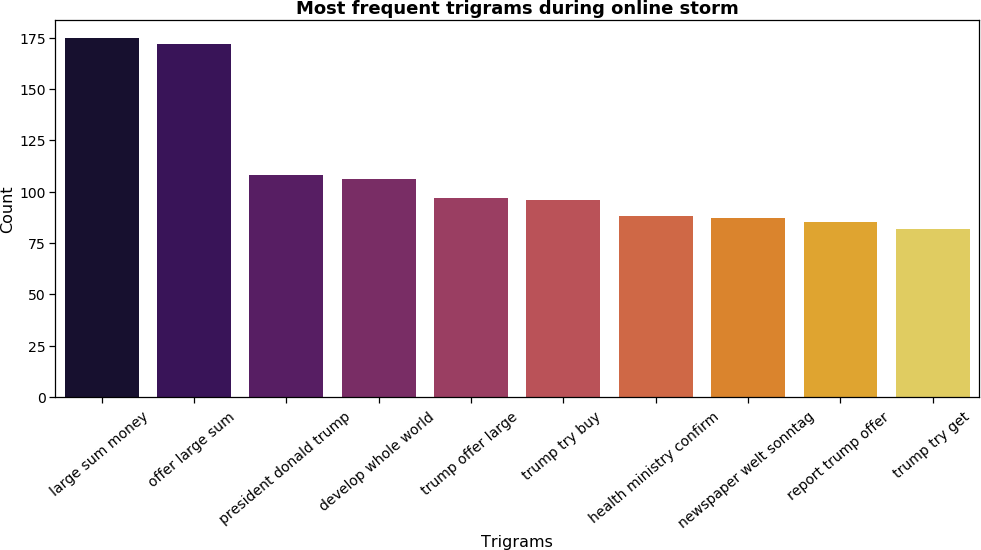
info = {'data': df\_trigrams, 'x': 'Grams', 'y': 'Count',

'xlab': 'Trigrams', 'ylab': 'Count', 'pal':'inferno', 'title': 'Most frequent trigrams during online storm', 'fname':'trigrams\_frequency\_during\_onlinestorm.png', 'angle': 40}

plot\_frequency\_chart(info)

['large sum money ', 175] ['offer large sum ', 172] ['president donald trump ', 108] ['develop whole world ', 106] ['trump offer large ', 97] ['trump try buy ', 96]

['health ministry confirm ', 88] ['newspaper welt sonntag ', 87] ['report trump offer ', 85] ['trump try get ', 82]



In [50]:

display\_wordcloud(tokens)

<Figure size 432x288 with 0 Axes>



What we've seen above shows obvious differences from the main stream life of CureVac on Twitter:

The top word is no longer ‘gate’ but ‘trump’ (ie., Donald Trump), immediately followed by 'coronavirus'. Gone are the days of collaboration for a next generation of new and innovative therapies. - - ‘Exclusive’

takes the lead, ‘collaboration’ is out of the league.

The most frequent trigram is ‘try buy exclusive’. These are now times for ‘exclusive large gain’.

‘Buy’ becames a new key word. ‘large sum money’ and ‘offer large sum’ are now the top trigrams in the chart.

# Step 5: LDA topics extraction

LDA (Latent Dirichlet Allocation) is an unsupervised machine learning technique that is increasingly popular in most text mining toolkits. You can find [here](https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158) a comprehensive article on the subject, published on Medium, covering extensively the assumptions and the math behind the algorithm.

I applied LDA to the two datasets (before and during the CureVac online fire-storm) to check whether the findings corroborate the trends that we have seen in our previous analysis of the word frequency.

In [51]:

**from sklearn.decomposition import** LatentDirichletAllocation

**from sklearn.feature\_extraction.text import** TfidfVectorizer

In [52]: *# I am using here Susan Li's functions to get the top words of a topic:*

**def** get\_keys(topic\_matrix):

*'''*

*returns an integer list of predicted topic categories for a given topic matrix*

*'''*

keys = topic\_matrix.argmax(axis=1).tolist()

**return** keys

**def** keys\_to\_counts(keys):

*'''*

*returns a tuple of topic categories and their accompanying magnitudes for a given list of keys '''*

count\_pairs = Counter(keys).items()

categories = [pair[0] **for** pair **in** count\_pairs] counts = [pair[1] **for** pair **in** count\_pairs] **return** (categories, counts)

**def** get\_top\_n\_words(n, n\_topics, keys, document\_term\_matrix, tfidf\_vectori zer):

*'''*

*returns a list of n\_topic strings, where each string contains the n mo st common*

*words in a predicted category, in order '''*

top\_word\_indices = []

**for** topic **in** range(n\_topics): temp\_vector\_sum = 0

**for** i **in** range(len(keys)):

**if** keys[i] == topic:

temp\_vector\_sum += document\_term\_matrix[i] temp\_vector\_sum = temp\_vector\_sum.toarray()

top\_n\_word\_indices = np.flip(np.argsort(temp\_vector\_sum)[0][-n:],0

)

top\_word\_indices.append(top\_n\_word\_indices) top\_words = []

**for** topic **in** top\_word\_indices: topic\_words = []

**for** index **in** topic:

temp\_word\_vector = np.zeros((1,document\_term\_matrix.shape[1])) temp\_word\_vector[:, index] = 1

the\_word = tfidf\_vectorizer.inverse\_transform(temp\_word\_vector

)[0][0]

**try**:

topic\_words.append(the\_word.encode('ascii').decode('utf-8'

))

**except**:

**pass**

top\_words.append(", ".join(topic\_words))

**return** top\_words

And here is a function for topics extraction using LDA, in which I end up creating a dataframe with the topics and their top words to facilitate the plotting that follows.

In [53]: *# LDA topics*

**def** get\_topics(edited, n\_topics, n\_words): eds = edited.values

vec = TfidfVectorizer(use\_idf=**True**, smooth\_idf=**True**) document\_term\_matrix = vec.fit\_transform(eds)

model = LatentDirichletAllocation(n\_components=n\_topics) topic\_matrix = model.fit\_transform(document\_term\_matrix)

keys = get\_keys(topic\_matrix) categories, counts = keys\_to\_counts(keys)

top\_n\_words = get\_top\_n\_words(n\_words, n\_topics, keys, document\_term\_m atrix, vec)

topics = ['Topic **{}**: **\n**'.format(i + 1) + top\_n\_words[i] **for** i **in** categ ories]

data=[]

**for** i, topic **in** enumerate(topics): tmp = []

tmp.append(topic) tmp.append(counts[i]) data.append(tmp)

df\_topics = pd.DataFrame(data, columns = ['Topics', 'Count'])

**return** df\_topics

In [ ]:

## Topics before the online storm

In [54]:

*# Filtering the tweets of the 6 years before the online storm*

df = tweets[tweets['onlinestorm'] == **False**]

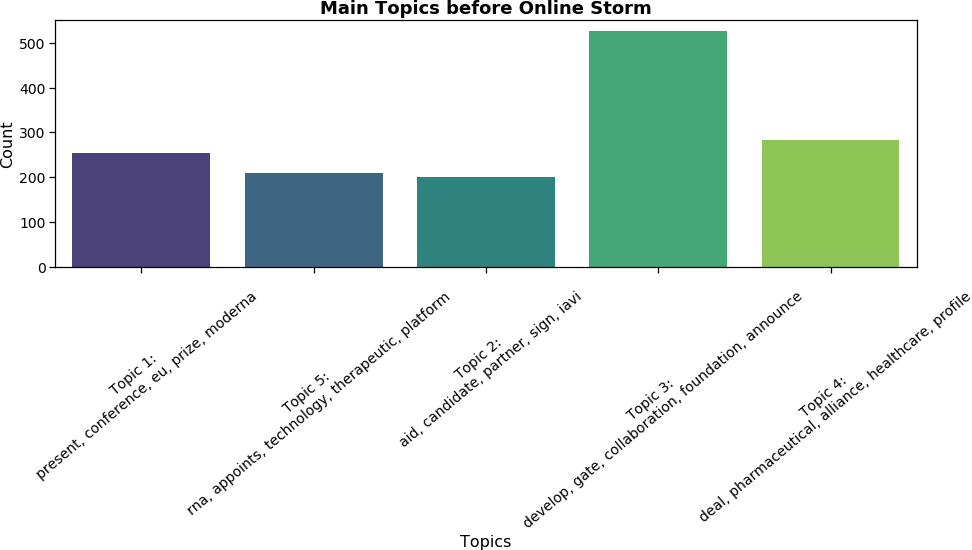
*# LDA topics*

df\_topics = get\_topics(df['edited'], 5, 5)

info = {'data': df\_topics, 'x': 'Topics', 'y': 'Count',

'xlab': 'Topics', 'ylab': 'Count', 'pal':'viridis', 'title': 'Main Topics before Online Storm', 'fname':'LDA\_Topics\_before\_onlinestorm.png', 'angle': 40}

plot\_frequency\_chart(info)



## Topics during the online storm

In [55]:

*# Filtering the tweets of the 3 days of the online storm*

df =tweets[tweets['onlinestorm']]

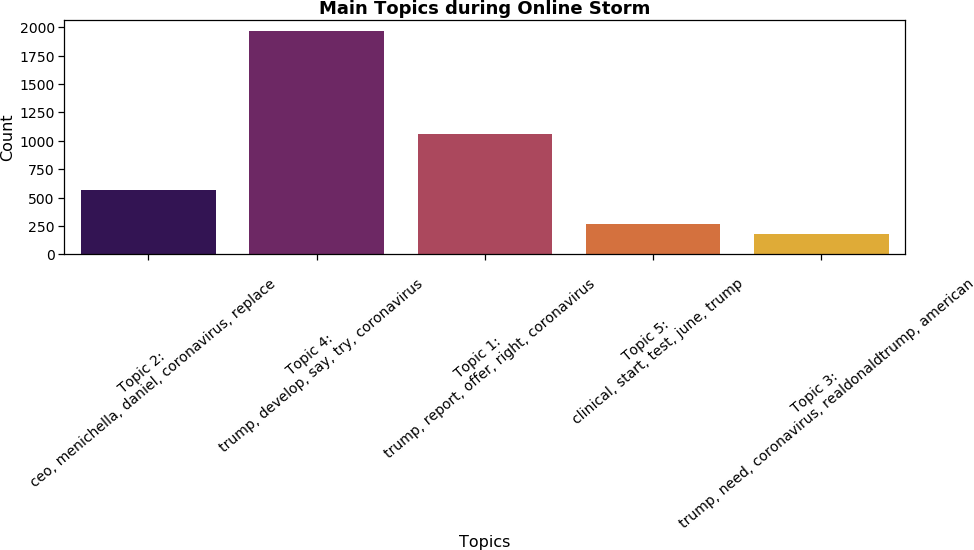
*# LDA topics*

df\_topics = get\_topics(df['edited'], 5, 5)

info = {'data': df\_topics, 'x': 'Topics', 'y': 'Count',

'xlab': 'Topics', 'ylab': 'Count', 'pal':'inferno', 'title': 'Main Topics during Online Storm', 'fname':'LDA\_Topics\_during\_onlinestorm.png', 'angle': 40}

plot\_frequency\_chart(info)

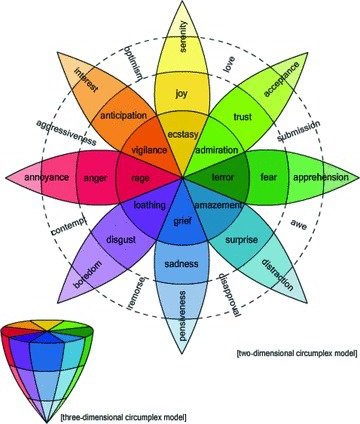


A word of caution must be exercised here. LDA topics are “imaginary” (latent) topics, frequently overlapping, and a clear distinction is not always achievable. Nevertheless, a comparison between topics before and during the online storm leaves no doubts about contrasting trends.

For a period of six years, the major topic emerging from tweets is about collaborative developments. In contrast, during the online storm, in a period of three days, the two topics that stand out are clearly about the alleged attempt of the USA president to ensure the exclusive rights for the coronavirus vaccine.

# Step 6: Emotion analysis

I drew from Robert Plutchik’s wheel of basic emotions an attempt to uncover the presence of the seven lexical units for anger, fear, sadness, disgust, anticipa-tion, joy and surprise [4].



In [56]:

**import termcolor import sys**

**from termcolor import** colored, cprint plt.style.use('fivethirtyeight')

The basic approach is to create a matrix of tweets and emotions to connect each word in the tweet to one or more emotions. I applied the National Research Council Canada (NRC) lexicon, a binary matrix with 14,182 words and 10 col-umns rows, each corresponding to positive and negative sentiment plus eight emotions. For a full understanding of the NRC lexicon read this article.[5]

In [57]:

ncr = pd.read\_csv('input/NCR-lexicon.csv', sep =';')

In [58]:

emotions = ['Anger', 'Anticipation','Disgust','Fear', 'Joy','Sadness', 'Su rprise', 'Trust']

In [ ]:

In [59]:

*# Join all the edited tweets in one single string*

joined\_string = join\_edited\_string(df['edited'])

*# Get tokens*

tokens = joined\_string.split(' ')

In [ ]:

In [60]:

*# We build now two dictionaries with indexes and unique words, for future reference*

unique\_words = set(tokens)

word\_to\_ind = dict((word, i) **for** i, word **in** enumerate(unique\_words)) ind\_to\_word = dict((i, word) **for** i, word **in** enumerate(unique\_words))

In [ ]:

In [61]:

**def** plot\_emotions\_period(df, cols, period = 'h' ):

df1 = df.groupby(df['datetime'].dt.to\_period(period)).mean() df1.reset\_index(inplace=**True**)

df1['datetime'] = pd.PeriodIndex(df1['datetime']).to\_timestamp() plot\_df = pd.DataFrame(df1, df1.index, cols)

plt.figure(figsize=(15, 10))

ax = sns.lineplot(data=plot\_df, linewidth = 3,dashes = **False**) plt.legend(loc='best', fontsize=15)

plt.title('Emotions in Tweets with CureVac during online storm', fonts ize=20)

plt.xlabel('Time by ythe hour', fontsize=15) plt.ylabel('Z-scored Emotions', fontsize=15) plt.savefig('images/Emotions\_during\_onlinestorm.png') **return**

In [62]: **def** get\_tweet\_emotions(df, emotions, col):

df\_base = df.copy()

df\_base.drop(['sentiment','sentiment\_intensity'], axis=1, inplace=**True**

)

emo\_info = {'emotion':'' , 'emo\_frq': defaultdict(int) } list\_emotion\_counts = []

**for** emotion **in** emotions: emo\_info = {} emo\_info['emotion'] = emotion

emo\_info['emo\_frq'] = defaultdict(int) list\_emotion\_counts.append(emo\_info)

*#criamos um dataframe de zeros com a dimensão de df*

df\_emotions = pd.DataFrame(0, index=df.index, columns=emotions)

stemmer = SnowballStemmer("english") x = 0

**for** i, row **in** df\_base.iterrows():

tweet = word\_tokenize(df\_base.loc[i][col])

**for** word **in** tweet:

word\_stemmed = stemmer.stem(word.lower()) result = ncr[ncr.English == word\_stemmed] **if not** result.empty:

**for** idx, emotion **in** enumerate(emotions): df\_emotions.at[i, emotion] += result[emotion]

**if** result[emotion].any():

**try**:

list\_emotion\_counts[idx]['emo\_frq'][word\_to\_in

d[word]] += 1

**except**:

**continue**

df\_base = pd.concat([df\_base, df\_emotions], axis=1)

**return** df\_base, list\_emotion\_counts

In [ ]:

In [63]:

**def** get\_words(word\_list, emotions): words\_emotion\_idx = []

**for** i, word **in** enumerate(word\_list): word = stemmer.stem(word.lower()) result = ncr[ncr.English == word] **if not** result.empty:

**for** emotion **in** emotions:

**if** result[emotion].any() > 0: words\_emotion\_idx.append(i)

**return** words\_emotion\_idx

In [ ]:

In [64]:

**def** get\_top\_emotion\_words(word\_counts, n = 5):

*# Passamos finalmente o dicionário para uma numpy array "words", com o indice da palavra e respectiva frequência*

words = np.zeros((len(word\_counts), 2), dtype=int)

**for** i, w **in** enumerate(word\_counts): words[i][0] = w

words[i][1] = word\_counts[w]

*# A partir dos indices gerados pela função argsort, sabemos a posição # das "n" palavras mais frequentes na array words*

top\_words\_indices = np.flip(np.argsort(words[:,1])[-n:],0)

*# Com estas posições (indices), obtemos os indices que funcionam como keys no dicionário ind\_to\_word,*

*# e nos devolvem, como "value", as palavras como strings*

top\_words = [words[ind][0] **for** ind **in** top\_words\_indices]

**return** words, top\_words, top\_words\_indices

In [ ]:

In [65]:

**def** print\_colored\_emotions(tweets, emotions, color, on\_color):

**for** tweet **in** tweets:

word\_list = word\_tokenize(tweet)

word\_emotion\_idx = get\_words(word\_list, emotions)

**for** i, w **in** enumerate(word\_list):

**if** i **in** word\_emotion\_idx:

w=colored(w, color=color, on\_color=on\_color) print(w, end='')

print(' ', end='')

print('**\n**')

**return**

In [ ]:

## Connecting words to emotions

In [66]:

*# We are using the NCR lexicon to associate words to emotions # Be patient, this will take some minutes ...*

df\_emo, list\_emotion\_counts = get\_tweet\_emotions(tweets, emotions, 'edited ')

*# Preparing for time series*

df\_emo['datetime']= pd.to\_datetime(df\_emo['datetime'])

For a better understanding of the word-emotions associations, we have here the charts showing what are the 10 words that contributed the most for each of the 8 emotions.

In [67]:

*# Plotting the 10 words that contribute the most for each of the 8 emotion s*

fig, axs = plt.subplots(figsize=(15, 25), frameon=**False**) plt.box(**False**)

plt.axis('off') plt.subplots\_adjust(hspace = 1.6)

counter = 0

**for** i, emotion **in** enumerate(emotions): *# for each emotioin # This is the dict that holds the top 10 words*

words, top\_words, top\_words\_indices = get\_top\_emotion\_words(list\_emoti on\_counts[i]['emo\_frq'], 10)

info = {'values' : [words[ind][1] **for** ind **in** top\_words\_indices], 'labels' : [ind\_to\_word[word] **for** word **in** top\_words]

}

sns.set(style="whitegrid") sns.set\_context("notebook", font\_scale=1.25)

ax = fig.add\_subplot(4, 2, counter+1) *# plot 2 charts in each of the 4 rows*

sns.despine()

ax = sns.barplot(x='labels', y='values', data=info, palette=("cividis"

))

plt.ylabel('Top words', fontsize=12)

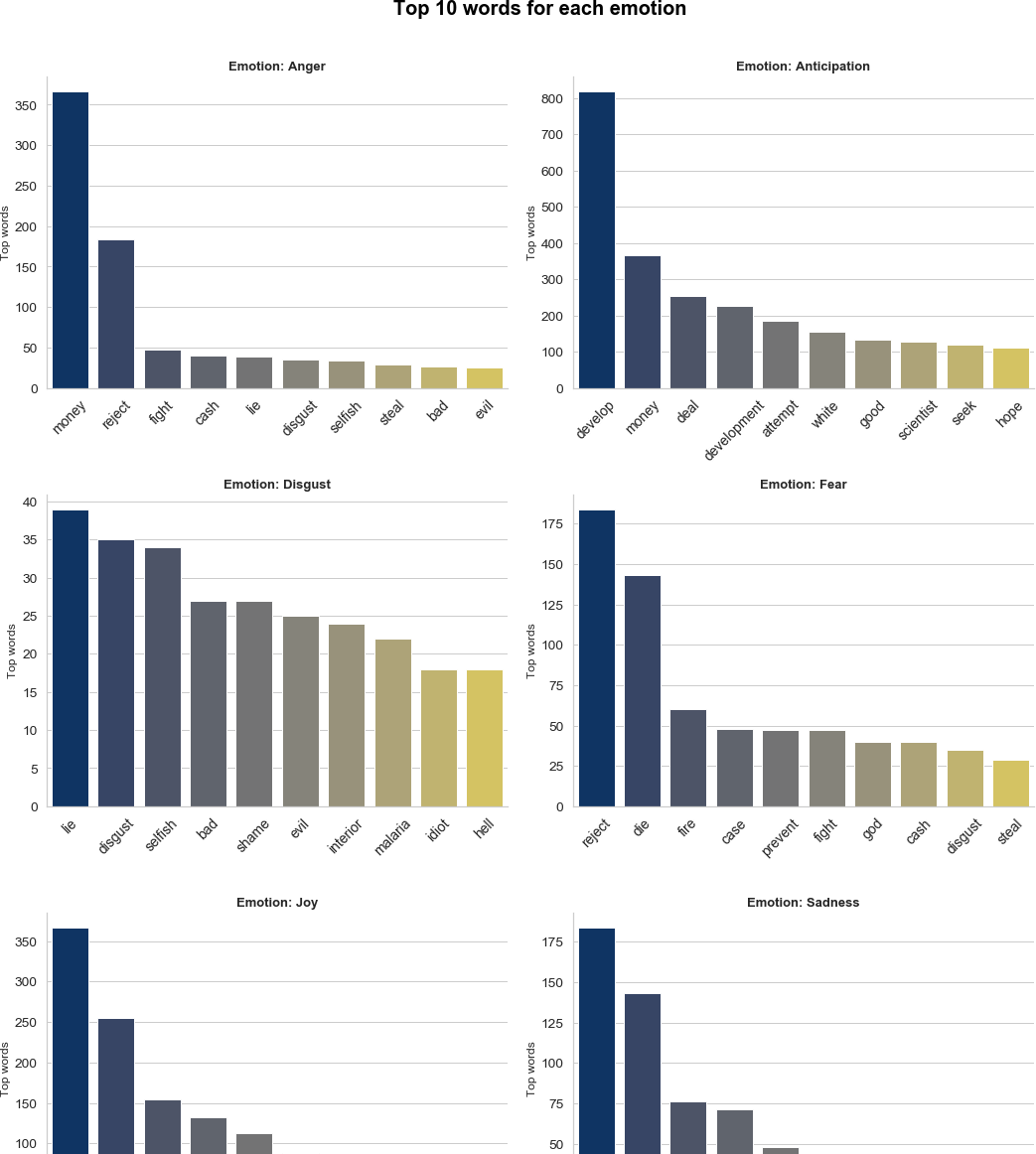
ax.set\_title(label=str('Emotion: ') + emotion, fontweight='bold', size

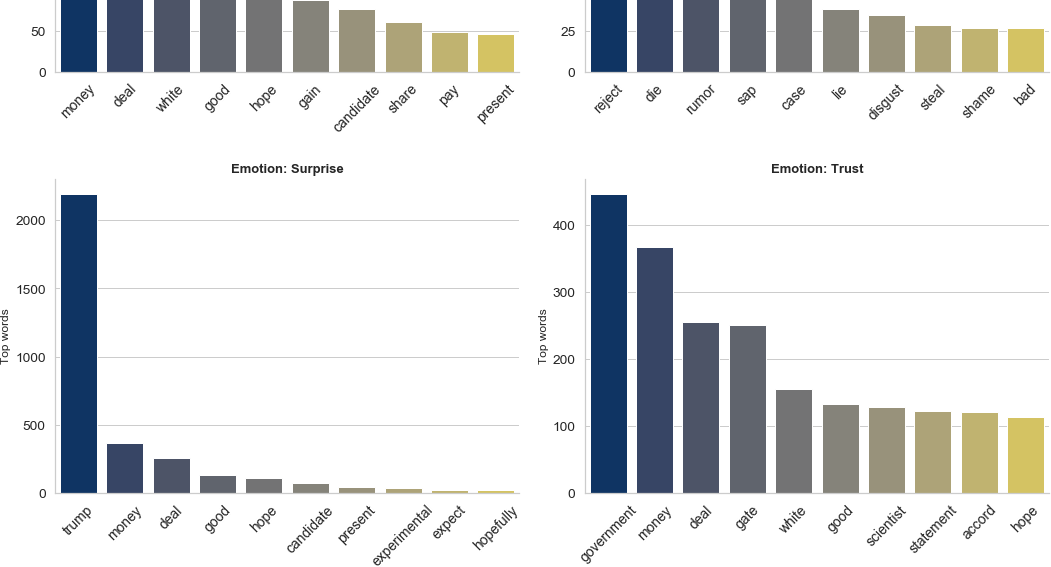
=13)

plt.xticks(rotation=45, fontsize=14) counter += 1

axs.set\_title(label='**\n**Top 10 words for each emotion**\n**', fontweight='bold', size=20, pad=40)

plt.tight\_layout() plt.savefig('images/Top10\_words\_per\_emotion.png')





For some authors, isolated emotions might not be the best granullarity for analysis. Skillicorn (2019) and colleagues prefer to aggregate emotions into positive and negative emotions [6]. Let's try it.

In [68]:

*# Aggregating negative and positive emotions*

df\_emo['neg\_emotions'] = df\_emo['Sadness'] + df\_emo['Fear'] + df\_emo['Disg ust'] + df\_emo['Anger']

df\_emo['pos\_emotions'] = df\_emo['Joy'] + df\_emo['Anticipation'] + df\_emo[' Trust'] + df\_emo['Surprise']

In [69]:

df\_emo['total\_neg\_emotions'] = df\_emo['neg\_emotions'].apply(**lambda** x: x > 0)

df\_emo['total\_pos\_emotions'] = df\_emo['pos\_emotions'].apply(**lambda** x: x > 0)

I use here pandas groupby feature to obtain a normalized account of the emotions as a proportion that takes into account the number of tweets in each of the two periods (before and during the online storm).

In [70]:

props = df\_emo.groupby('onlinestorm')['total\_neg\_emotions'].value\_counts(n ormalize=**True**).unstack()

props

|  |  |  |  |
| --- | --- | --- | --- |
| Out[70]: |  | | |
|  | **total\_neg\_emotions**  **onlinestorm** | **False** | **True** |
|  | **False** | 0.890699 | 0.109301 |
|  | **True** | 0.575713 | 0.424287 |

The results show that during the online storm period, negative emotions are present in 42 per cent of the tweets, whereas previously only 11% of the tweets included negative emotions.

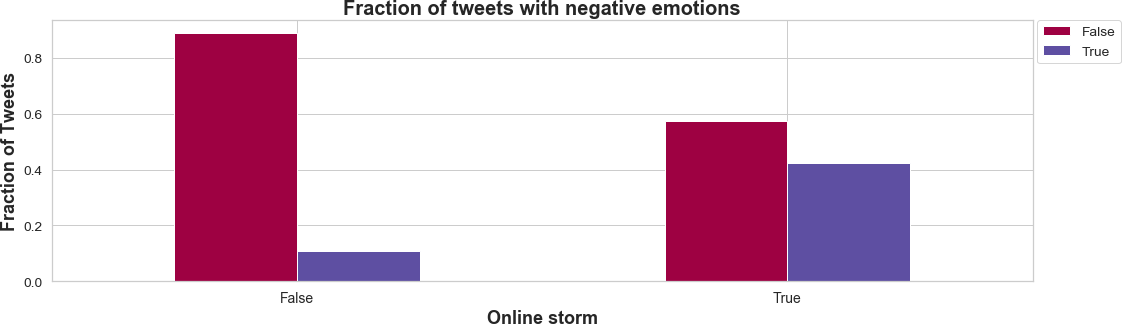
We can spot it more clearly in the following chart ...

In [71]:

*# plot it*

plot\_fractions(props,'Fraction of tweets with negative emotions','Fraction

\_of\_Tweets\_with\_negative\_emotions')



Converselly, when it comes to positive emotions, we witness also an increase in the proportion of tweets with these emotions during online storm (84 per cent). But it is not such a sharp change when compared to the 61 per cent before the online storm.

In [72]:

props = df\_emo.groupby('onlinestorm')['total\_pos\_emotions'].value\_counts(n ormalize=**True**).unstack()

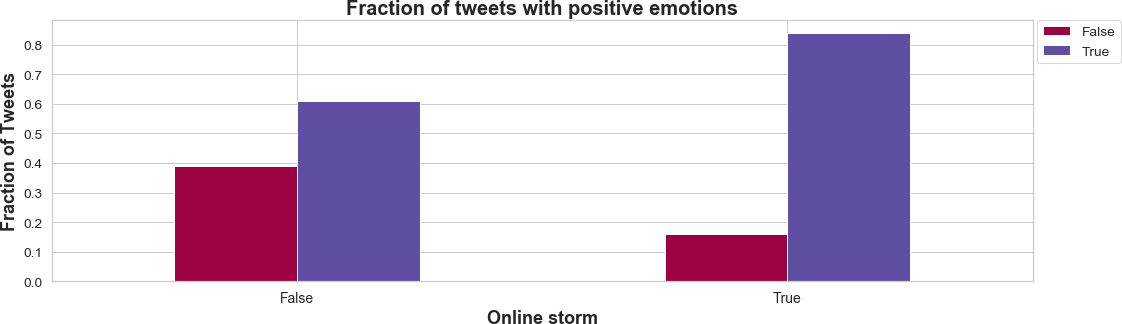
props

|  |  |  |  |
| --- | --- | --- | --- |
| Out[72]: |  |  |  |
|  | **total\_pos\_emotions**  **onlinestorm** | **False** | **True** |
|  | **False** | 0.390360 | 0.609640 |
|  | **True** | 0.159356 | 0.840644 |

In [73]:

plot\_fractions(props,'Fraction of tweets with positive emotions','Fraction

\_of\_Tweets\_with\_positive\_emotions')



## Word - emotion connections in the tweets

To help us have a feeling of how things work behind thew scenes, I wrote a function (print\_colored\_emotions) to

display the words connected to negative (red) and positive (green) emotions.

It is important to acknowledge that I am not giving any kind of emotion score to the tweets (that would be another undertaking all together). I am just locating the word-emotion connections within the tweets, since a tweet may depict more than one emotion (or cluster of emotions) – and they usually do.

Here are some negative emotions ...

In [74]:

df = df\_emo[df\_emo['Sadness'] > 3]

print\_colored\_emotions(df['text'], ['Disgust','Sadness','Anger','Fear'], ' white', 'on\_red')

Die Stiftung von Bill Gates investiert 52 Millionen Dollar in dt . Firma C ureVac , die an Impfstoff gegen # Coronavirus forscht . ( Das ist die Firm a , an der die US-Regierung großes Interesse hat aktuell , siehe : https :

// twitter.com/AscotBlack/sta tus/1239161218398670848 ? s=19 … ) https :

//www . forbes.com/sites/matthewh erper/2015/03/05/bill-melinda-gates-foun dation-makes-largest-ever-equity-investment-in-a-biotech-company/amp/ ? twitter\_impression=truehttps : [//www.forbes.com/sites/matthewherper/2015/0](http://www.forbes.com/sites/matthewherper/2015/0) 3/05/bill-melinda-gates-foundation-makes-largest-ever-equity-investment-in

-a-biotech-company/amp/ ? twitter\_impression=true …

You created crowded conditions @ airports corralling sick w/healthy . This will lead to further disease spreading . Your denials & delays , along w/

ill-prepared quarter measures are going to kill many . And stop trying to steal CureVac for U.S. only . # VaccinesForAll

@ KimStrassel @ kimguilfoyle @ seanmdav @ maggieNYT That wasn ’ t the lie put forth by Germany . The lies was that Curevac had a cure and Trump want ed a . Exclusive supply . You lied all about Trump/Russia and you are lyin g again .

You are full of it . Trump has put America in danger every single day and his mixed messages are causing confusion and eventually death . He has han dled this Corona situation so badly and tried to bribe a German called Cur eVac . This President is so shameful https : //www . tagesschau.de/inland/ corona- impfstoff-deutschland-usa-101.html … https : // twitter.com/GOP/st atus/123 9342159486164998 …

News ! # CureVac CureVac Rejects Rumors of US Acquisition : CureVac Reject

s Rumors of US Acquisition http : // dlvr.it/RRzMsN Visit our site ! pic.t witter.com/OLnwkX4Glu

Yes they have lied , but in this case no , USA Today and CureVac provide p roof it ’ s a false story.. pick a real lie to propagate , not a fake news

story , all that does is hurt your credibility

# COVID19 latest , 9pm GMT Cases 197,467 Deaths 7,953 Recoveries 81,691 # Belgium to enforce lockdown from 11am GMT Wed # US cities , states announc e piecemeal lockdowns # EU Commission Chief Ursula von der Leyen claims Ge rman company CureVac may have # vaccine ready “ towards fall '' pic.twitte r.com/jycIhEU0bg

And here some positive ones ...

In [75]:

df = df\_emo[df\_emo['Anticipation'] > 4] print\_colored\_emotions(df['text'], ['Joy','Trust','Anticipation'], 'white'

, 'on\_green')

UK pharma is equally likely to develop a vaccine . In fact , pharma all ov er the world are searching for a vaccine for SARS-CoV-2 . CureVac is unlik ely to develop one quickly enough alone . Now 's a good time for pharma to

cooperate , rather than compete for financial gains .

It works ! It worked after 9/11 . God Bless Lee Greenwood ! God Bless Cure

Vac ! God

Bless China ! God

Bless Italy ! God

Bless Iran ! God

Bless South

Korea ! God Bless Germany Too ! God Bless Asia , Europe , Africa , Americ

as & Australia . pic.twitter.com/dSKwPrLTeD

Glad researchers behind GER lab # CureVac have '' vetoed '' this , it s a sign of huge progress from the times of # WorldWar . Einstein & others gav e up their research to build d A-Bomb : if we do give up our values to mon ey we have nothing else but # death ahead . # EU stay strong @ vonderleyen

Take Precautions . Pray Harder . Pray that something good comes out of ' C urevac among others . Faith over Fear . This too shall pass in JESUS name

. Once again liked I posted earlier . `` Pls dont forget to lift up prayer s over the virus '' . We are not really praying . Lets Pray Saints

1/ Yes . Human vaccine trials are already underway in the US and China . C ureVac in Germany has two candidates and expect an experimental vaccine in June/July . Having said that , the expectation is , it will only be publi

cly available in about 18 months . Until then

In [ ]:

## Fraction of emotions in relation to number of tweets, before and during the online storm

In [76]:

df1 = df\_emo.groupby(df\_emo['onlinestorm'])[emotions].apply(**lambda** x:( x.s um()/x.count())\*100)

In [77]:

df1.index = ['before\_onlinestorm', 'during\_onlinestorm']

In [78]:

df1.head()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[78]: |  | | | | | | | | |
|  |  | **Anger** | **Anticipation** | **Disgust** | **Fear** | **Joy** | **Sadness** | **Surprise** |  |
|  | **before\_onlinestorm** | 5.091650 | 54.921928 | 2.511881 | 8.689749 | 26.408690 | 5.431093 | 22.471147 | 5 |
|  | **during\_onlinestorm** | 33.358116 | 86.270136 | 16.183395 | 31.945477 | 40.421314 | 28.178439 | 82.131351 | 8 |

In [79]:

df1.index

Out[79]: Index(['before\_onlinestorm', 'during\_onlinestorm'], dtype='object')

In [80]:

df\_ =df1.T

In [81]:

df\_.reset\_index()

Out[81]:

**index before\_onlinestorm during\_onlinestorm**

**0**

Anger

5.091650

33.358116

**1** Anticipation 54.921928 86.270136

**3** Fear 8.689749 31.945477

**2**

Disgust

2.511881

16.183395

**4**

Joy

26.408690

40.421314

**5** Sadness 5.431093 28.178439

**6**

Surprise

22.471147

82.131351

**7** Trust 54.989817 81.437423

In [82]:

fig, ax = plt.subplots(1, 1, figsize=(10, 6)) ax.set\_title(label='Comparing percentage of emotion-related words before a nd during online storm**\n**', fontweight='bold', size=18) df\_.reset\_index().plot(

x="index", y=["before\_onlinestorm", "during\_onlinestorm"], kind="bar", ax=ax

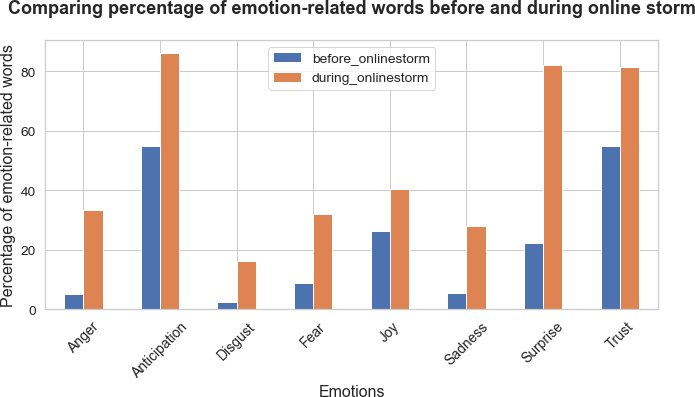
)

plt.xlabel("Emotions",fontsize = 16)

plt.ylabel("Percentage of emotion-related words",fontsize = 16) plt.xticks(rotation=45,fontsize=14)

plt.tight\_layout() plt.savefig('images/Percentage\_emotions\_before\_and\_during\_onlinestorm.png'

)

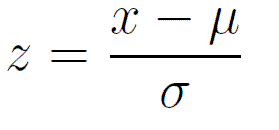


In [ ]:

## Applying a Z-score normalization

In another effort to normalize the emotion scores, I am using the Z-score, instead of the mere counts of word- emotion connections, because these are heavily affected by the number of tweets in each period in consideration.

The z-score tells us how many standard deviations an individual value is from the mean, and is calculated with following formula:



I use the pandas’ apply function to calculate the z-score of each individual value in all the 8 columns of emotions in the dataframe.

In [84]:

df\_zscore = df\_emo.groupby(df\_emo['onlinestorm'])[emotions].apply(**lambda** x

:(x - x.mean()) / x.std())

In [85]:

df\_emo = pd.concat([df\_emo[['datetime','text','edited', 'onlinestorm']], d f\_zscore], axis=1)

In [86]:

df\_emo.head()

Out[86]:

Million EUR european...

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **datetime text edited onlinestorm** | **Anger** | **Anticipation** | **Disgust** | **Fear** |  |
| Robert-Jan smits 2014-03- Smits at innovation  **0** 12 Innovation convention False 18:26:59 Convention win €2m  2014... inducement... | -0.211661 | -0.764009 | -0.156192 | -0.261235 | -0.515 |
| First #EU first eu prize 2014-03- #vaccine award  **1** 13 prize euic2014 False 09:50:54 awarded 2 check  CureVac ... complex jo... | -0.211661 | 2.018154 | -0.156192 | 2.745007 | 3.388 |
| Congrats 2 congrats eu 2014-03- CureVac ! prize  **2** 14 4 #EU euic2014 find False 12:50:28 #vaccine industry  prize #... contr... | -0.211661 | -0.764009 | -0.156192 | -0.261235 | -0.515 |
| MT  2014-03- @sanofiDE mt sanofide  **3** 14 CureVac win million eur False | -0.211661 | -0.764009 | -0.156192 | -0.261235 | -0.515 |
| 16:01:30 Wins Two inaugural  f... |  |  |  |  |  |

2014-03-

**4** 14

17:44:32

CureVac wins EU's EUR2m

inducement prize for ...

win eu eur2m inducement

prize rnactive technol...

False -0.211661

-0.764009 -0.156192 -0.261235 -0.515

In [ ]:

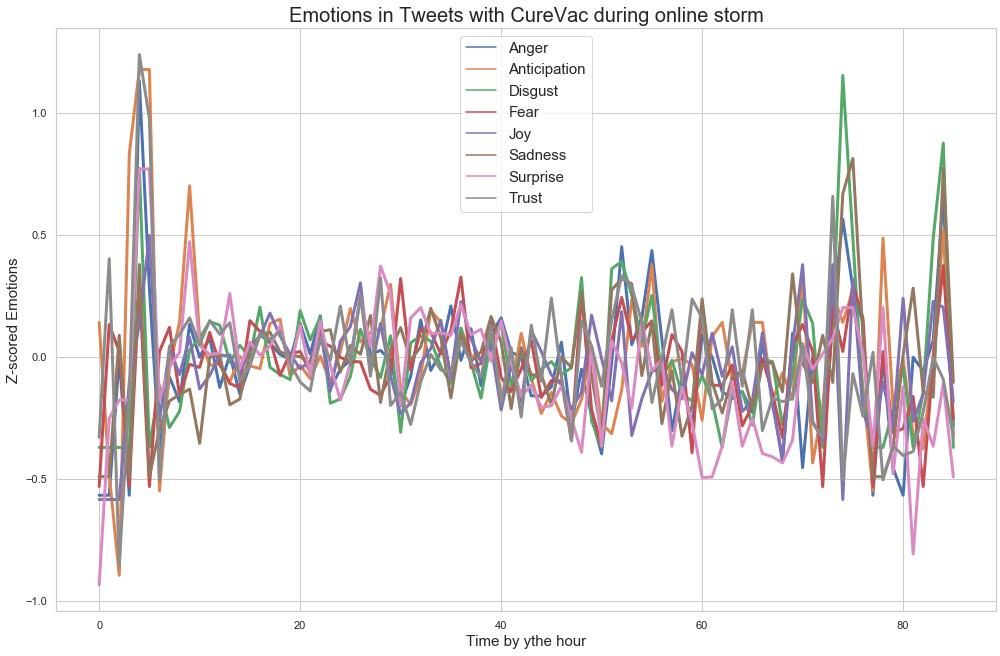
## The dynamics of emotions during the online storm

With our normalized values, we can now have a more precise view of the way emotions evolved, by the hour, during the 3 days of the online storm.

Here we have a mixture of all the emotions during online storm ...

In [87]:

plot\_emotions\_period(df\_emo[df\_emo['onlinestorm']], emotions)



Let's see how each of the emotions evolved during this period ...

In [88]: *# Plotting emotions during online storm*

fig, axs = plt.subplots(figsize=(15, 25), frameon=**False**) plt.box(**False**)

plt.axis('off') plt.subplots\_adjust(hspace = 1.6)

counter = 0

df = df\_emo[df\_emo['onlinestorm']]

df1 = df.groupby(df['datetime'].dt.to\_period('h')).mean() df1.reset\_index(inplace=**True**)

df1['datetime'] = pd.PeriodIndex(df1['datetime']).to\_timestamp()

**for** i, emotion **in** enumerate(emotions): *# for each emotion*

emo = [] emo.append(emotion)

plot\_df = pd.DataFrame(df1, df1.index, emo)

sns.set(style="whitegrid") sns.set\_context("notebook", font\_scale=1.25)

ax = fig.add\_subplot(4, 2, counter+1) *# plot 2 charts in each of the 4 rows*

sns.despine()

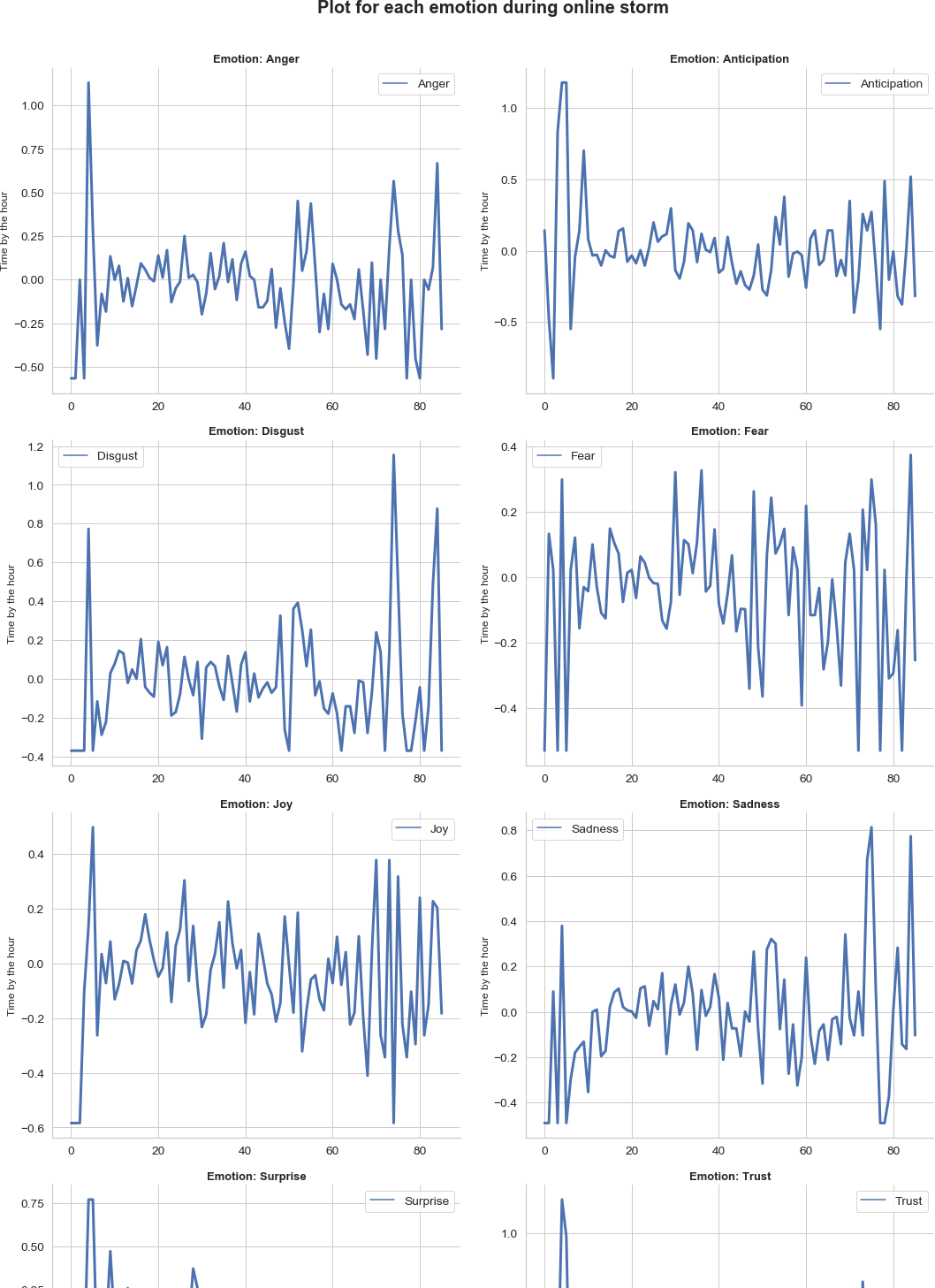
ax = sns.lineplot(data=plot\_df, linewidth = 3,dashes = **False**) plt.ylabel('Time by the hour', fontsize=12) ax.set\_title(label=str('Emotion: ') + emotion, fontweight='bold', size

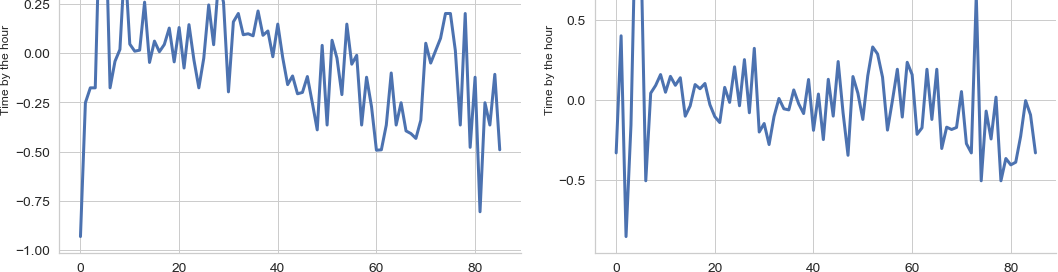
=13)

counter += 1

axs.set\_title(label='**\n**Plot for each emotion during online storm**\n**', fontweight='bold', size=20, pad=40)

plt.tight\_layout() plt.savefig('images/Emotions\_during\_onlinestorm.png')

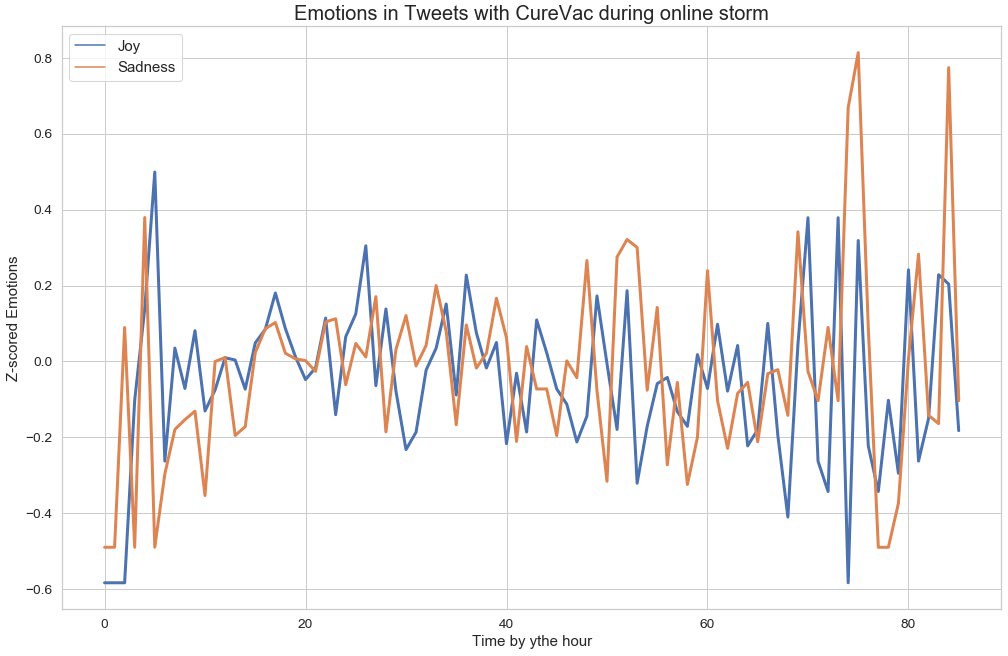




Another way of looking at it is by plotting contrasts of emotions, like joy and sadness ...

In [89]:

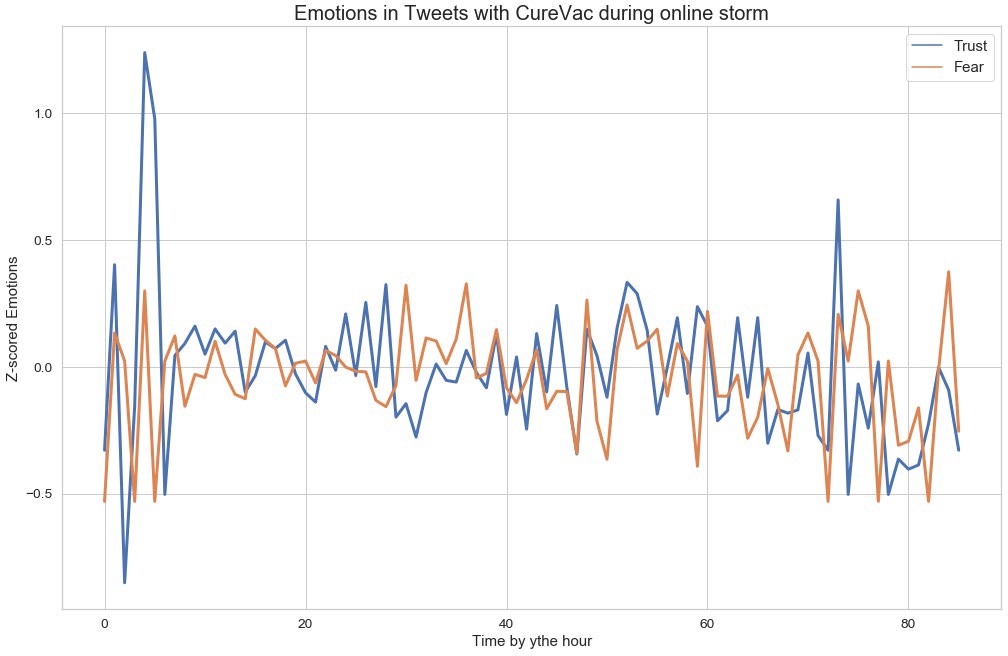
plot\_emotions\_period(df\_emo[df\_emo['onlinestorm']], ['Joy', 'Sadness'])



In [ ]:

In [91]:

plot\_emotions\_period(df\_emo[df\_emo['onlinestorm']], ['Trust', 'Fear'])



In [ ]:

In [ ]:

In [ ]: