

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Diego de Souza Abreu

Bem-te-vi: Monitoramento e análise de dados de rede social

Duque de Caxias, Rio de Janeiro
2019

Diego de Souza Abreu

BEM-TE-VI: MONITORAMENTO E ANÁLISE DE DADOS DE REDE SOCIAL

Trabalho de Conclusão de Curso apresentado
ao Curso de Especialização em Ciência de
Dados e Big Data como requisito parcial à
obtenção do título de especialista.

Orientador: Rodrigo Richard Gomes

Duque de Caxias, Rio de Janeiro
2019

SUMÁRIO

1. Introdução	4
1.1. Contextualização	4
1.2. O problema proposto	7
1.3. Tecnologias utilizadas	9
1.4. Arquivos do projeto.....	9
2. Coleta de Dados.....	11
2.1. Tweets.....	11
2.2. Datasets Kaggle.....	14
2.2.1. Cidades brasileiras.....	14
2.2.2. Tweets em português com classificação de sentimentos.....	14
3. Tratamento de Dados	15
3.1. Tratamento de Tweets.....	15
3.2. Tratamento de datasets do Kaggle	16
3.2.1. Cidades brasileiras.....	16
3.2.2. Tweets em português com classificação de sentimentos.....	17
4. Análise exploratória dos dados	18
5. Modelo de Machine Learning: Classificação de sentimentos.....	24
6. Apresentação dos Resultados	29
6.1. Visualização em gráficos	29
6.2. Dashboard	39
6.3. Relatórios	41
6.4. Como utilizar o bem-te-vi.....	42
6.5. Estudos de caso	43
6.5.1. 02/10/19 – Grêmio x Flamengo	44
6.5.2. 06/10/19 – Último dia do Rock in Rio 2019	51
7. Links	59
REFERÊNCIAS	60

1. Introdução

1.1. Contextualização

Vivemos uma época de intensa produção e utilização de dados. De fato, eles não são novidades em nossas vidas. Desde os primórdios, registramos e passamos para as gerações seguintes, dados e informações (conjunto de dados) de diferentes formas. Passando pela tradição oral, escrita e chegando aos diversos formatos hoje possibilitados pela tecnologia. Um processo que evoluiu ao longo do tempo, e sem dúvida, podemos dizer que é um dos fatores responsáveis por onde estarmos hoje, como espécie e como sociedade.

O que torna o cenário atual diferenciado de outros momentos da história, é que nunca antes foi possível produzir, armazenar e analisar tantos dados. Segundo uma publicação [IBM](#) do segundo semestre de 2017, cerca de 90% dos dados existentes no mundo foram produzidos nos últimos 2 anos (2015-2016). O que nos leva a percentual ainda maior hoje em dia.

Esses números grandiosos são reflexos direto da evolução e popularização do uso da tecnologia nos últimos 25 anos. Conforme um artigo publicado pelo [Fórum Econômico Mundial](#), estima-se que todo o universo de dados digitais, tenha em 2020 cerca de 44 zettabytes. O que dá aproximadamente 40 vezes mais bytes, do que estrelas no universo observável. Não à toa, o termo “Big Data” é amplamente usado para denominar esse fenômeno.

A razão para estarmos continuamente gerando e capturando tantos dados, é porque sabemos que deles podemos extrair informações valiosas. O que nos leva a outros termos que costumam acompanhar o Big Data, como: ciência de dados, mineração de dados, análise de dados, Inteligência artificial, aprendizado de máquina e muitos outros, que correspondem as inúmeras técnicas, ferramentas, conceitos, processos e disciplinas que temos à disposição para realizar essa tarefa de extração de valor.

Por isso é cada vez mais presente o uso de aplicações baseadas em dados. As vemos na medicina onde é utilizada para melhorar diagnósticos, nas empresas para aumentar as vendas, e também no nosso dia-a-dia nos sugerindo um caminho melhor do que a avenida principal engarrafada.

Um outro exemplo, esse mais específico, de como se obter informações de valor em grandes conjuntos de dados e que vem sendo amplamente utilizado no mundo todo é a análise de dados redes sociais. Que inclusive, será nosso objeto de estudo.

As redes sociais hoje, podem ser consideradas uma extensão ou mesmo um complemento do “mundo real”, onde as pessoas interagem, compartilham suas opiniões, intenções, desejos, memes, angústias e interesses. Em muitos casos, até mais do que o fazem presencialmente. De acordo com relatórios publicados no início do ano de 2019, cerca 57% da [população mundial](#) está conectada as redes sociais. No [Brasil](#), esse número chega a 66%.

A análise de dados redes sociais consiste em utilizar dados fornecidos pelos usuários através de cadastros e publicações nas redes para obter informações. Esse tipo de análise é feito hoje pelas próprias redes sociais, onde são usadas como base para a melhoria da própria rede e também na criação de produtos de marketing para seus usuários, como por exemplo: publicações direcionadas a um determinado grupo de pessoas. As redes também permitem, com certas limitações e condições, que além deles, outros possam coletar esses dados e fazer suas próprias análises.

Um aspecto interessante desse tipo de atividade, é que um primeiro momento, os dados de forma individual, parece ter pouco ou nenhum valor. Mas quando em conjuntos, associados e comparados a outros, pode gerar uma fonte consistente de informações. De onde por exemplo, é possível identificar perfis de pessoas, seus comportamentos e suas preferências, fazer publicidade segmentada a públicos específicos, verificar a aceitação de produtos, a receptividade de campanhas, emitir retorno ao público com mais assertividade e velocidade, identificar tendências, construir base de conhecimento para ações futuras, enfim, uma infinidade de possibilidades.

Para não ficarmos só enumerando o que é possível fazer, podemos citar alguns casos concretos em que a análise de rede sociais gerou visões estratégicas e criativas em diferentes níveis.

O [Facebook](#) por exemplo, revelou que cerca de 90 milhões de micro e pequenas empresas utilizam suas ferramentas para se conectar com clientes e que cerca de [75%](#) dessas empresas, conseguiram aumentar as vendas com a ajuda da plataforma. Um estudo da [Comscore](#) sobre a Copa da Rússia em 2018, mostra que a América Latina foi responsável por mais de 19 milhões de menções sobre o evento.

Entre as publicações que citavam os patrocinadores, Coca-cola, Adidas e Visa foram os mais presentes. Um tipo de impacto que não era possível ser quantificado no passado.

Após divulgarem o trailer e o material promocional do filme Sonic, os fãs demonstraram toda sua [insatisfação](#) produzindo uma enxurrada de críticas a respeito da estética do protagonista feito em computação gráfica. Em pouco tempo, o estúdio e o diretor do filme se pronunciaram informando que o lançamento do filme seria adiado para que o visual fosse refeito.

Além do uso para fins comerciais, devemos também mencionar o seu uso em estudos que nos possibilitam compreender e quantificar melhor movimentos sociais. Como o da [Primavera Árabe](#), onde as redes sociais foram o principal meio de difusão de informações, influenciando a criação de movimentos contra regimes autoritários em países da região. Outro exemplo é o projeto [Me Too Rising](#) da Google, que monitora o uso da #metoo usada por mulheres em publicações relatando assédio sexual, mostrando o quanto frequente e grande é esse problema.

E é claro, falando sobre análise de redes sociais não podemos deixar de citar o escandaloso caso da [Cambridge Analytica](#), empresa que fornecia de forma combinada serviços de análise e marketing. Ela teve uma atuação ativa em processos políticos recentes como a eleição presidencial americana em 2016 e o Brexit. Com um modelo de aplicação que consistia em traçar perfis de eleitores e direcionar a eles publicações a votar em um determinado candidato.

O escândalo se deu após serem revelados que os dados usados pela empresa foram obtidos de forma ilegal, onde foi aproveitado uma falha no sistema de permissões do Facebook e usuários tiveram seus dados coletados conceder autorização prévia. Outro ponto, é que foi identificado que as publicações direcionadas continham informações inverídicas com o objetivo de manipular a opinião pública. A empresa fechou as portas em 2018, mas os processos judiciais e seus reflexos se estendem até hoje conforme é mostrado no documentário [“Privacidade violada”](#).

Este caso, trouxe à tona o debate sobre a ética no uso de dados e questões de permissões e propriedade. Que vem repercutindo em mudanças de políticas de privacidade nas plataformas e legislação nos países como a GDPR (Regulamento Geral sobre a Proteção de Dados) na União Europeia e a LGPD (Lei Geral de

Proteção de Dados Pessoais) no Brasil, afetando diretamente a forma de como trabalhar com esse tipo de dados.

1.2. O problema proposto

Conforme mencionado, nesse estudo abordaremos a análise de dados de redes sociais. Nossa foco será o Twitter. A plataforma é uma das mais usadas atualmente e tem como principal característica trabalhar em “tempo real”. Pois ela exibe as publicações a medida em que os usuários as criam. Diferente de outras redes sociais que usam algoritmos de recomendação que não levam em conta a ordem cronológica, mostrando aos usuários publicações mais direcionadas aos seus interesses e com maior probabilidade de agradá-lo.

O aspecto cronológico dá ao Twitter uma dinâmica única, onde os usuários através de suas publicações, seus tweets, falam e interagem sobre assuntos que estão acontecendo naquele exato momento. Pode ser uma notícia política, um programa de TV ou mesmo um evento musical. Tudo que está ocorrendo no mundo, aparece no Twitter.

O que faz da plataforma um tópico sempre presente nos planos de marketing de empresas que querem compreender o comportamento e se aproximar mais de seus clientes. Já que é um poderoso identificador de tendências e termômetro de receptividade e aprovação.

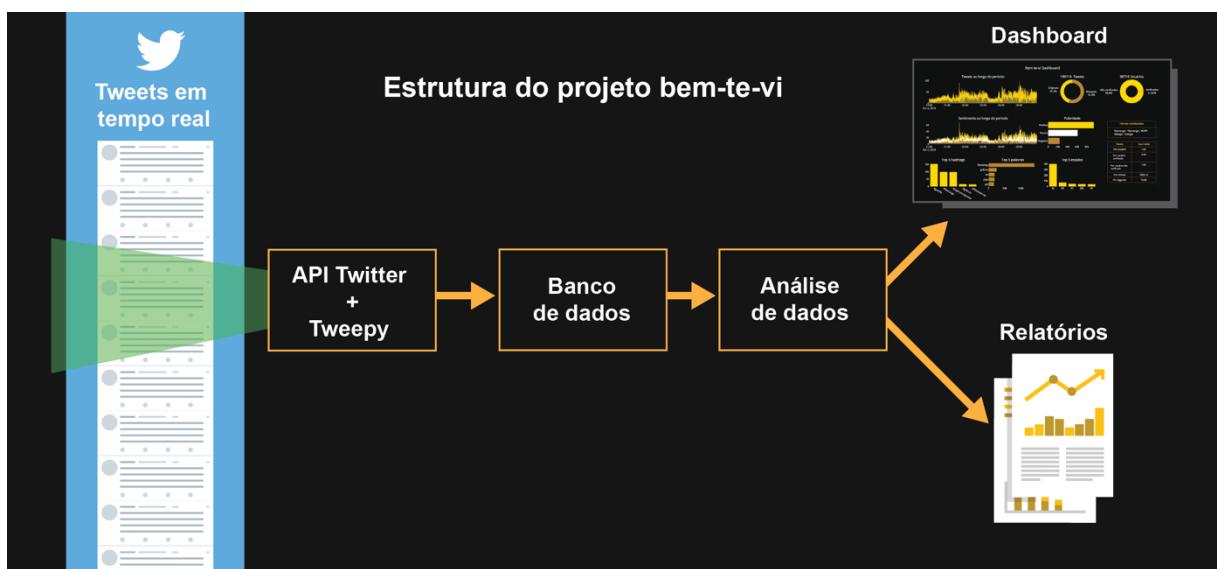
Segundo o artigo do Fórum Econômico Mundial citado anteriormente, por dia são feitos cerca de 500 milhões tweets. Ou seja, um volume imenso de dados. Para tentar compreender o que está acontecendo no Twitter e extrair dele apenas aquelas informações que são relevantes e de nosso interesse, abordaremos nesse estudo o desenvolvimento de uma ferramenta de monitoramento e análise de dados.

Há no mercado algumas opções desse tipo de ferramenta. O próprio Twitter oferece esse tipo de serviço. Porém, são opções pagas e com limitações quanto a personalização das análises. Por isso, a ferramenta aqui desenvolvida utiliza soluções open source, sem nenhum tipo de custo. A intenção é que qualquer pessoa que queira fazer esse tipo de análise e tenha um conhecimento básico de programação possa utilizá-la. Podendo também adaptar o código a sua necessidade.

O Twitter tem como logotipo um pássaro e significa gorpear, o cantar dos pássaros. Segundo os criadores, a ideia é que os usuários assim como os pássaros cantam pela rede. Já que é para contar o que está sendo cantado pelas aves, esse projeto não poderia ser batizado com outro nome, bem-te-vi.

Este projeto é composto resumidamente pelas etapas: coleta de dados, análises e apresentação. Os dados são os tweets públicos criados pelos usuários da rede sobre um determinado tema, definido por palavras-chave. Para isso, utilizamos a API do Twitter e a biblioteca Tweepy, que analisam os tweets e capturam aqueles que contêm as palavras-chave. Os dados capturados são armazenados em um banco de dados.

Os dados armazenados são acessados e passam por tratamentos e análises. Por fim, as análises são apresentadas em forma de um dashboard para visualização imediata e de relatórios para posterior.



As análises feitas são do tipo quantitativa, onde identificamos as informações numéricas como quantidades gerais, contagem de valores específicos e taxas de proporção. E também do tipo qualitativa, em que classificamos através de um modelo de aprendizagem de máquina, o sentimento presente no texto do tweet como positivo, neutro ou negativo.

O projeto tem foco nos usuários brasileiros, por isso a coleta é apenas de tweets em português. Os tratamentos e análises também são focados em nosso idioma. O tempo de coleta depende do tema e do objetivo do usuário. No capítulo seis serão apresentados alguns casos de uso.

1.3. Tecnologias utilizadas

[Linux Ubuntu](#): Sistema operacional Open-source;

[Python](#): Linguagem de programação. Bem como diversas bibliotecas e pacotes compatíveis ou criados para essa linguagem:

[JSON](#): Pacote nativo da linguagem python para manipulação de dados em formato JSON;

[Unidecode](#): Pacote para retirada de caracteres especiais do texto, como emojis e acentos;

[Its dangerous](#): Pacote para criptografia de dados;

[Pandas](#): Pacote para manipulação de dados em formato dataframe;

[PyMongo](#): Pacote para conexão com o mongoDB;

[Time](#): Pacote nativo da linguagem python para manipulação de dados em formato data/hora;

[Date time](#): Biblioteca para manipulação de dados em formato data/hora;

[Scikit-learn](#): Biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python;

[NLTK](#): Conjunto de bibliotecas para processamento de linguagem natural simbólica;

[Emoji](#): Pacote para manipulação de emojis. [Vader](#): Ferramenta de análise de sentimentos baseada regras de vocabulário;

[NBConvert](#): Pacote para exportar o Notebook em HTML;

[Tweepy](#): Biblioteca Open-source para acessar a API do Twitter;

[Jupyter notebook](#): IDE para linguagem python

[MongoDB](#): Banco de dados;

[Plotly](#): Framework para criação de gráficos e dashboards;

1.4. Arquivos do projeto

[assets/hidecode.tpl](#): Arquivo responsável por omitir retirar a codificação dos relatórios criados;

[assets/typography.css](#): Arquivo padrão para ajustes css do plotly. Ajusta a aparência final do dashboard;

`dados_base_analise_de_sentimentos/`: Pasta que contém os datasets de treino e teste do modelo de classificação de sentimentos;

`dados_base_analise_de_sentimentos/`: Pasta que contém o dataset de cidades brasileiras usado para criar o arquivo “dicionário_brasil.py”;

`dados_tweets_coletados/`: Pasta que contém os datasets coletados nos estudos de casos;

`imagens/`: Pasta com imagens utilizadas no README.md;

`relatorios/`: Pasta que armazena os relatórios criados;

`README.md`: Breve resumo textual do projeto bem-te-vi;

`atualiza_dashboard.py`: Código que atualiza periodicamente o conteúdo exibido no dashboard;

`chaves.py`: Arquivo onde são definidas as chaves de API, chave de criptografia, nome da base de dados e as palavras-chave para a busca;

`coleta_tweets.py`: Código-fonte para executar a coleta de tweets;

`dashboard.py`: Código-fonte da construção do dashboard;

`dicionario_brasil.py`: Arquivo com a relação de estados e cidades brasileiras;

`executa.todos.py`: Código que executa os arquivos coleta_tweets.py, dashboard.py e atualiza_dashboard.py de forma sincronizada;

`gera_relatorio.sh`: Código que executa o arquivo relatório.ipynb via prompt de comando;

`modelo_analise_de_sentimentos.py`: Modelo de classificação de sentimentos de tweets;

`passo_a_passo_bem-te-vi.ipynb`: Arquivo em formato notebook com o passo a passo das etapas do projeto;

`passo_a_passo_bem-te-vi.html`: Arquivo passo_a_passo_bem-te-vi.ipynb convertido em formato html;

`passo-a-passو_dashboard.html`: Protótipo do dashboard criado no arquivo passo_a_passo_bem-te-vi.ipynb;

`relatorio.ipynb`: Código responsável pela criação de relatórios;

`requirements.txt`: Lista de bibliotecas e pacotes que precisam estarem corretamente instalados para o funcionamento do projeto;

2. Coleta de Dados

Neste projeto, utilizamos dados obtidos de duas fontes. Como mencionado anteriormente, a rede social Twitter é a principal delas, mas durante as análises também usamos alguns conjuntos de dados que foram disponibilizados por usuários no Kaggle, uma plataforma de competições e compartilhamento de conhecimento sobre ciência de dados.

2.1. Tweets

Para realizar a coleta de dados no Twitter é necessário antes ter uma [conta de desenvolvedor](#) na rede social. Com a conta já criada, através do mesmo site, é preciso submeter uma aplicação para ter acesso as chaves de autenticação da API do Twitter. Após a aprovação, temos em mãos 4 chaves: Consumer key, Consumer secret, Access token e Access token secret. Que nos permitem acessar e coletar os tweets. Armazenamos essas chaves no arquivo “chaves.py”. Nesse arquivo também é definida a chave de criptografia, o nome da base que irá armazenar os dados e as palavras-chaves que serão usadas para filtrar os tweets.

```
# -----
# Chaves da API do Twitter:
# Consumer Key
consumer_key = "INSIRA_AQUI_SUA_CHAVE_DA_API_DO_TWITTER"
# Consumer Secret
consumer_secret = "INSIRA_AQUI_SUA_CHAVE_DA_API_DO_TWITTER"
# Access Token
access_token = "INSIRA_AQUI_SUA_CHAVE_DA_API_DO_TWITTER"
# Access Token Secret
access_token_secret = "INSIRA_AQUI_SUA_CHAVE_DA_API_DO_TWITTER"
# -----
# Chave para criptografia do campo nome do usuário:
cripto_key = 'INSIRA_AQUI_SUA_CHAVE_DE_CRIPTOGRAFIA'
# -----
# Lista de palavras chaves para a coleta dos tweets:
keywords = ['INSIRA_AQUI_SUAS_PALAVRAS_CHAVE_PARA_BUSCA']
# -----
# Nome da base de dados no MongoDB:
banco = 'INSIRA_AQUI_O_NOME_DA_BASE_DE_DADOS'
```

O acesso ao Twitter utilizando as chaves da API é feito via biblioteca tweepy. Há outras formas de se fazer esse acesso, como acessar diretamente [via HTTP](#). Porém o tweepy se mostrou uma opção mais prática por exigir menos configuração e linhas de código e atender perfeitamente a necessidade do projeto. O código desenvolvido para a coleta fica no arquivo “coleta_tweets.py”.

Note que é utilizado o pacote de criptografia it’s dangerous. Isso porque apesar de estarmos trabalhando com tweets públicos, optamos por não exibir a identificação do autor do tweet (a @ do usuário). O it’s dangerous recebe como parâmetro a chave de criptografia definida no arquivo “chaves.py”, que pode ser uma simples palavra ou um termo mais complexo.

O código de coleta não possui uma limitação de tempo ou de quantidade de tweets, porém há um limite da própria API do twitter, que é mencionado nas documentações do Twitter e do Tweepy. Não é informado um valor definido, mas trata-se de um grande volume de tweets em um curto período de tempo. O que pode gerar um erro e parar com a coleta. Por isso a coleta é feita através de uma função que ao se deparar com esse erro ou qualquer outro como falha de conexão, ele para a coleta, aguarda um período de tempo (no projeto definido como 10 segundos) e retoma a coleta novamente. Dessa forma, a coleta após iniciada só é encerrada pelo usuário.

```
# -----
# Importação de pacotes:
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
import json
from unidecode import unidecode
from itsdangerous import URLSafeSerializer
from pymongo import MongoClient
import time
# -----
# Importação do arquivo chaves.py:
import chaves
# -----
# Autenticação com API do twitter:
auth = OAuthHandler(chaves.consumer_key, chaves.consumer_secret)
auth.set_access_token(chaves.access_token, chaves.access_token_secret)
# Função de criptografia:
cripto = URLSafeSerializer(chaves.cripto_key)
# -----
# Função de filtragem de tweets:
```

```
class Filtratweets(StreamListener):
    def on_data(self, dados):
        tweet = json.loads(dados)
        created_at = tweet["created_at"]
        cp_screen_name = cripto.dumps(tweet["user"]["screen_name"])
        verified = tweet["user"]["verified"]
        text = tweet["text"]
        location = tweet["user"]["location"]
        obj = {"created_at":created_at,
                "cp_screen_name": cp_screen_name,
                "verified":verified,
                "text":text,
                "location":location}
        tweetind = col.insert_one(obj).inserted_id
        print (obj)
        return True
    def on_error(self, status_code):
        if status_code == 420:
            return True
# Objeto filtragem de tweets
filtratweets = Filtratweets()
# Objeto captura de tweets, faz a conexão via API do Twitter
capturatweets = Stream(auth, listener = filtratweets, wait_on_rate_limit=True)
# -----
# Criação da conexão ao MongoDB
client = MongoClient('localhost', 27017)
# Criação do banco de dados
db = client[chaves.banco]
# Criação da collection
col = db[chaves.banco]
# -----
# Função que inicia a coleta de tweets:
def inicia_coleta():
    while True:
        try:
            capturatweets.filter(languages=["pt"], track=chaves.keywords)
        except:
            time.sleep(10)
            continue
# -----
# Inicia a coleta dos tweets
inicia_coleta()
```

Há dezenas de dados em um único tweet, entretanto não coletamos todos, apenas os mais relevantes:

Variável coletada	Descrição	Tipo
_id	Identificação única de cada tweet publicado.	string
created-at	Data e hora da publicação do tweet.	string
screen_name	Identificação do usuário (@).	string
verified	Informação de que o usuário possui selo de verificação.	bool
location	Localização do usuário.	string

2.2. Datasets Kaggle

2.2.1. Cidades brasileiras

O Twitter não possui uma padronização para o preenchimento do campo “localition” e permite que o usuário o preencha de forma livre. Enquanto alguns usuários a preenchem com país, estado ou cidade, outros não preenchem ou até usam lugares fictícios. Por isso, foi criado o arquivo “dicionário_brasil.py” que contém cidades, siglas e estados brasileiros. Esse arquivo é uma adaptação do dataset [cidades brasileiras](#), arquivo “BrazilianCities.csv”, disponível no kaggle. Com ele podemos comparar o registro feito pelo usuário com a nossa lista de locais e determinar a validade da localização. A precisão dessa localização que será feita é de nível estadual.

2.2.2. Tweets em português com classificação de sentimentos

Os arquivos “Test3classes.csv” e “Train3Classes.csv” do dataset “[Portuguese Tweets for Sentiment Analysis](#)” disponível no kaggle, foram adaptados dando origem aos arquivos “df_teste.csv” e “df_treino.csv”. Ambos foram utilizados para a construção do modelo de classificação de sentimentos utilizado na etapa de análise. O dataset de treino possui 100 mil tweets em português já classificados em Negativo, Positivo e Neutro. O de teste possui 4999 registros classificados.

3. Tratamento de Dados

3.1. Tratamento de Tweets

Felizmente os dados do Twitter já vem com uma boa estrutura, praticamente pronto para iniciar a análise exploratória. Fazemos apenas dois tratamentos nele. O primeiro é a criptografia do nome do usuário que já é feito direto na captura do tweet, assim ele já é gravado no banco de dados criptografado. O trecho de código que consta no arquivo “coleta.py” e responsável por isso é:

```
from itsdangerous import URLSafeSerializer
...
# Função de criptografia:
cripto = URLSafeSerializer(chaves.cripto_key)
...
cp_screen_name = cripto.dumps(tweet["user"]["screen_name"])
```

O segundo tratamento é realizado em cima da variável de data/hora (created_at), pois ele não vem em nosso fuso horário. Para isso acessamos o banco de dados e criamos um dataframe Pandas com os tweets coletados através do código abaixo que está presente nos arquivos “dashboard.py” e “relatorio.ipynb”:

```
# Banco de dados
# Criação da conexão ao MongoDB
client = MongoClient('localhost', 27017)
# Criação do banco de dados
db = client[chaves.banco]
# Criação da collection
col = db[chaves.banco]
# -----
# criação de um dataset com dados retornados do MongoDB
dataset = [{"created_at": item["created_at"],
            "cp_screen_name": item["cp_screen_name"],
            "verified": item["verified"],
            "text": item["text"],
            "location": item["location"],
            } for item in col.find()]
# Criação do dataframe
df = pd.DataFrame(dataset)
# -----
def format_datetime(dt_series):
    def get_split_date(strdt):
        split_date = strdt.split()
        str_date = split_date[1] + ' ' + split_date[2] + ' ' + split_date[5] + ' '
        + split_date[3] + ' ' + split_date[4]
```

```

    return str_date
dt_series = pd.to_datetime(dt_series.apply(lambda x: get_split_date(x)),
    format = '%b %d %Y %H:%M:%S %z')
return dt_series
# Cria nova variável para o horário correto:
df['data_hora'] = format_datetime(df['created_at'])
# Converte para o nosso fuso horário:
df['data_hora'] = df['data_hora'].dt.tz_convert('America/Sao_Paulo')

```

Desta forma passamos a ter uma nova coluna com a data e hora em nosso fuso horário. Os demais tratamentos serão feitos a medida em que forem necessários na fase exploratória.

3.2. Tratamento de datasets do Kaggle

3.2.1. Cidades brasileiras

Mais a frente, iremos localizar os autores dos tweets comparando a variável “location” com nosso conjunto de localidades válidas, arquivo “dicionário_brasil.py”. Para criarmos esse conjunto, usamos as colunas “Cidade”, “Estado”, “Sigla” do dataset de cidades brasileiras, para gerar um dicionário python com a seguinte estrutura:

```

# Dicionário de locais:
dic = {
"SIGLA_UF":"SIGLA_UF",
...
"NOME_DO_ESTADO":"SIGLA_UF",
...
"NOME_DA_CIDADE":"SIGLA_UF ",
...
}

```

Para facilitar esse mapeamento entre registro e dicionário, os caracteres foram convertidos para maiúsculos e os espaços entre palavras foram removidos. O mesmo também será feito com a variável “location” durante a análise. Por fim, foi identificado que 240 cidades possuem nomes iguais, para evitarmos localizar incorretamente, essas cidades foram removidas do dicionário.

3.2.2. Tweets em português com classificação de sentimentos

Com o objetivo de analisar o sentimento contido em cada tweet, criaremos no tópico cinco desse projeto, um modelo multinomial Naive Bayes. Entretanto esse algoritmo precisa ser treinado, e para isso usaremos os arquivos com tweets já classificados que foram baixados do Kaggle.

Não precisamos de todos os dados que esses datasets possuem, por isso “df_teste.csv” e “df_treino.csv” que usaremos no modelo, possuem apenas as colunas “tweet_text” e “sentiment” dos arquivos “Test3classes.csv” e “Train3Classes.csv”. Os nomes dessas colunas nos novos arquivos foram alterados para “tweets” e “sentimento”. E os valores da variável ‘sentimento’, que originalmente são “0”, “1” e “2” foram transformado respectivamente para “Negativo”, “Positivo” e “Neutro”.

4. Análise exploratória dos dados

Temos definidos alguns tópicos a serem respondidos que nos ajudarão a transformar os dados em informações por meio de das análises quantitativas. São eles:

- Quantidade de tweets;
- Quantidade de usuários únicos;
- Quantidade de tweets ao longo do período de coleta;
- Top hashtags usadas;
- Top palavras mais presentes;
- Top estados com mais tweets;
- Taxas de tweets por usuário, por minuto e por segundo;

Nesta etapa é utilizado o pacote Pandas juntamente com algumas outras bibliotecas python como o scikitlearn. Os mesmos códigos utilizados aqui também são usados nos arquivos “dashboard.py” e “relatorio.ipynb”.

Quantidade de tweets:

Para obter a quantidade de tweets é fácil, basta contar a quantidade de registros no dataset. Porém podemos enriquecer essa informação identificando quais deles são originais e quais deles são retweets(tweets que foram replicados por outros usuários). Os retweets tem como característica ter a sigal "RT" antes do conteúdo. Portanto temos que identificar em quantos registros a variável "text" começa com "RT".

```
# Quantidade total de tweets:
total_tweets = len(df)
# Cria uma variável chamada "retweeted" que armazena se o conteúdo do tweet começa com RT ou não:
df['retweeted'] = df['text'].str.startswith('RT')
# Contagem dos retweets:
retweets = len(df[df['retweeted'] == True])
# Contagem dos originais:
originais = len(df[df['retweeted'] == False])
```

Quantidade de usuários únicos:

Mesmo de criptografados, podemos fazer a contagem do número de usuários. Pois o algoritmo de criptografia segue um padrão baseado pela palavra chave, então palavras iguais possuem correspondentes criptografados iguais. Portanto basta apenas fazer uma contagem simples. Assim como o tópico anterior, também podemos enriquecer a análise. Pois através da variável "verified" podemos identificar quantos usuários possuem o selo de verificação do twitter. Esse selo é dado para personalidades públicas, empresas e veículos de comunicação.

```
# Contagem de usuários únicos:
qtd_usuarios_unicos = df['cp_screen_name'].nunique()
# Contagem de usuários verificados:
usuarios_verificados = df[df['verified'] == True]
total_tt_verificados = len(usuarios_verificados)
qtd_usuarios_verificados = usuarios_verificados['cp_screen_name'].nunique()
# Contagem de usuários não verificados:
usuarios_nao_verificados = df[df['verified'] == False]
total_tt_nao_verificados = len(usuarios_nao_verificados)
qtd_usuarios_nao_verificados = usuarios_nao_verificados['cp_screen_name'].nunique()
```

Quantidade de tweets ao longo do período de coleta:

Para alcançar essa informação é necessário quantos tweets foram publicados no mesmo horário. Ou seja, quantos registros possuem a mesmo valor na variável `data_hora`.

```
# Cria um dataframe com a quantidade de tweets por horário:
tw_x_pd = df['data_hora'].value_counts().to_frame().reset_index()
tw_x_pd.columns = ['data_hora', 'qtd_tweets']
# Ordenar pelo horário:
tw_x_pd = tw_x_pd.sort_values(by=['data_hora'])
```

Top hashtags usadas:

Esta análise é em teoria similar à análise de retweets, mas com a diferença de que os "RT's" sempre estão presentes no início do conteúdo, já as hashtags(#) podem estar em qualquer posição dentro da variável "text". Por isso requer alguns

tratamentos adicionais, que também servirão para adiantar o trabalho na questão de top 5 palavras.

```
# Contagem de hashtags e palavras mais presentes:
verifica_hash = pd.DataFrame()
verifica_hash['text'] = df['text']
# Substituição dos caracteres # e @ antes do tratamento para que não sejam Excluídos no processo:
verifica_hash['text'] = verifica_hash['text'].str.replace('#', 'hashtag_vl',
    regex=True)
verifica_hash['text'] = verifica_hash['text'].str.replace('@', 'arroba_vl',
    regex=True)
# Utilização do método CountVectorizer para criar uma matriz de documentos:
cv = CountVectorizer(strip_accents = None)
count_matrix = cv.fit_transform(verifica_hash['text'])
# Criação de um dataframe com o número de ocorrências das principais palavras em nosso dataset:
contagem_palavras = pd.DataFrame(cv.get_feature_names(), columns=["palavra"])
contagem_palavras["count"] = count_matrix.sum(axis=0).tolist()[0]
contagem_palavras = contagem_palavras.sort_values("count",
    ascending=False).reset_index(drop=True)
# Retorno do caracteres # e @:
contagem_palavras['palavra'] = contagem_palavras['palavra'].str.replace(
    'hashtag_vl', '#', regex=True)
contagem_palavras['palavra'] = contagem_palavras['palavra'].str.replace(
    'arroba_vl', '@', regex=True)
# Contagem de Hashtags:
hashtags = contagem_palavras[contagem_palavras['palavra'].str.startswith('#')]
# Separação e exibição as mais presentes:
top5_hashtags = hashtags.head()
top_hashtags = hashtags.head(10)
```

Top palavras mais presentes:

Nesse tópico, vamos aproveitar o dataframe "contagem_palavras" criado durante a etapa anterior e refiná-lo com alguns tratamentos que incluem: remoção de hashtags, nomes de usuários (@), termos irrelevantes e stop words.

```
# Pacote NLTK
import nltk
from nltk.corpus import stopwords
# Importação de stop words em português do pacote NLTK:
portugues_stops = set(stopwords.words('portuguese'))
# Adição de novas stop words identificadas como faltantes em análises anteriores:
novas_stopwords = ['tá', 'ta', 'pra', 'pro']
portugues_stops = portugues_stops.union(novas_stopwords)
```

```

# Arrobas de usuários citadas nos tweets:
arrobas_citadas = contagem_palavras[
    contagem_palavras['palavra'].str.startswith('@')]
# Remoção hashtags e arrobas
contagem_palavras_relevantes = contagem_palavras[
    (contagem_palavras['palavra'].isin(
        hashtags['palavra'])==False)]
contagem_palavras_relevantes = contagem_palavras_relevantes[
    (contagem_palavras_relevantes['palavra'].isin(
        arrobas_citadas['palavra'])==False)]
# Remoção de termos irrelevantes:
contagem_palavras_relevantes = contagem_palavras_relevantes[
    contagem_palavras_relevantes['palavra'] != 'rt']
contagem_palavras_relevantes = contagem_palavras_relevantes[
    contagem_palavras_relevantes['palavra'] != 'https']
contagem_palavras_relevantes = contagem_palavras_relevantes[
    contagem_palavras_relevantes['palavra'] != 'http']
contagem_palavras_relevantes = contagem_palavras_relevantes[
    contagem_palavras_relevantes['palavra'] != 'co']
contagem_palavras_relevantes = contagem_palavras_relevantes[
    contagem_palavras_relevantes['palavra'] != '#']
# Remoção de stop words:
contagem_palavras_relevantes = contagem_palavras_relevantes[
    (contagem_palavras_relevantes['palavra'].isin(
        portugues_stops) == False)]
# Separação e exibição as mais presentes:
top5_contagem_palavras_relevantes = contagem_palavras_relevantes.head()
top_contagem_palavras_relevantes = contagem_palavras_relevantes.head(10)

```

Top estados com mais tweets:

A localização de um tweet pode ser obtida por duas variáveis a "location" e a "place". Com base em coletas anteriores, foi possível verificar que a "place" apresenta geralmente valor nulo. Isso porque é uma variável que só é preenchida quando o usuário opta por compartilhar sua localização exata no tweet, como em uma postagem de "check-in" em um restaurante, por exemplo. Esse recurso é pouco utilizado pelos usuários da rede, por isso não coletamos esse dado.

Já a "location" é uma informação que consta na bio do usuário, não é preciso como a place, mas nos dá uma localização a nível de cidade/estado/país. Além de ter uma taxa de preenchimento consideravelmente maior. Entre os que preenchem de forma minimamente adequada, é possível notar um certo padrão. A cidade ou estado geralmente aparecem no primeiro ou segundo termo, sendo separados por vírgulas ou hifens. O terceiro termo geralmente é o país. Veja alguns exemplos de preenchimento:

- Manaus, Brasil
- Porto Alegre

- Rio de Janeiro, Brasil
- Madureira - RJ
- Joinville, Santa Catarina - BR

Geralmente temos presentes nomes de cidades e estados. Então conseguimos garantir uma certa precisão em relação ao estado, visto que com o nome da cidade é possível saber o estado, já o inverso não.

É criado a variável “estado”. Fazemos então o tratamento dos caracteres especiais, colocamos tudo em maiúsculo, separamos os dois primeiros termos e mapeamos junto ao nosso dicionário de localidades. Aqueles que constarem no dicionário, terão o valor da variável “estado” preenchido com a sigla UF.

```
## Importação do dicionário de localidades:
import dicionario_brasil
# Tratamento da variável location:
df['localizacao'] = df['location']
df['localizacao'] = df['localizacao'].str.upper()
df['localizacao'] = df['localizacao'].fillna('NULO')
df['localizacao'] = df['localizacao'].apply(unidecode)
df['localizacao'] = df['localizacao'].str.replace('BRASIL', 'NULO', regex=True)
df['localizacao'] = df['localizacao'].str.replace('-', ' ', regex=True)
df['localizacao'] = df['localizacao'].str.replace('/', ' ', regex=True)
df['localizacao'] = df['localizacao'].str.replace('|', ' ', regex=True)
df['localizacao'] = df['localizacao'].str.replace(' ', ' ', regex=True)
# Divisão dos dois primeiros termos presentes:
df['loc_01'] = df['localizacao'].str.split(',').str[0]
df['loc_02'] = df['localizacao'].str.split(',').str[1]
# Mapeamento com dicionário para a criação da variável "estado":
df['estado'] = df['loc_01'].map(dicionario_brasil.dic)
df['estado_02'] = df['loc_02'].map(dicionario_brasil.dic)
df['estado'] = df['estado'].fillna(df['estado_02'])
# Verificação da quantidade de localizações estaduais obtidas:
tweets_localizados = sum(df['estado'].value_counts())
pct_tweets_localizados = round((tweets_localizados*100)/len(df),2)
# Dataframe com a contagem dos estados:
tweets_estados = df['estado'].value_counts().to_frame().reset_index()
tweets_estados.columns = ['estado', 'qtd_tweets']
# Separação e exibição dos mais presentes:
Top5_estados = tweets_estados.head()
top_estados = tweets_estados.head(10)
```

Taxas de tweets por usuário, por minuto e por segundo:

Concluímos então essa etapa de análise quantitativa com alguns cálculos que nos ajudaram a entender o volume e as proporções dos dados coletados:

```
# Cálculo de taxas de tweets:  
# Criação de um dataframe para a análise:  
data_hora_coleta = pd.DataFrame()  
# Separação de data e hora:  
data_hora_coleta['data'] = [d.date() for d in df['data_hora']]  
data_hora_coleta['hora'] = [d.time() for d in df['data_hora']]  
# Coletando o último e o primeiro horário:  
fim = data_hora_coleta['hora'].iloc[-1]  
inicio = data_hora_coleta['hora'].iloc[0]  
# Transformação dos dados:  
horario_fim = datetime.datetime.combine(datetime.date.today(), fim)  
horario_inicio = datetime.datetime.combine(datetime.date.today(), inicio)  
# Cálculo do período de coleta:  
periodo = horario_fim - horario_inicio  
# Taxa média de tweets por usuário:  
if qtd_usuarios_unicos > 0:  
    media_tweets_por_usuario = round((total_tweets/ qtd_usuarios_unicos),2)  
else:  
    media_tweets_por_usuario = 0  
# Taxa média de tweets por usuário verificado:  
if qtd_usuarios_verificados > 0:  
    media_tweets_por_usuario_verificado = round(  
        (total_tt_verificados/ qtd_usuarios_verificados),2)  
else:  
    media_tweets_por_usuario_verificado = 0  
# Taxa média de tweets por usuário não verificado:  
if qtd_usuarios_nao_verificados > 0:  
    media_tweets_por_usuario_nao_verificado = round(  
        (total_tt_nao_verificados/ qtd_usuarios_nao_verificados),2)  
else:  
    media_tweets_por_usuario_nao_verificado = 0  
# Taxa média de tweets por minuto:  
tweets_por_minuto = round((total_tweets/(periodo.seconds/60)),2)  
# Taxa média de tweets por segundo:  
tweets_por_segundo = round(tweets_por_minuto/60,2)
```

5. Modelo de Machine Learning: Classificação de sentimentos.

Finalizaremos nossas análises com a classificação de sentimento contido no texto do tweet. Em nosso projeto essa talvez seja a com maior grau de complexidade. Para essa análise usaremos dois algoritmos o Multinomial Naive Bayes e Vader.

Há diversos algoritmos de análise de sentimentos já prontos, disponíveis e com bom desempenho, o Vader é um exemplo. No projeto [iFell](#) feito pela UFMG que compara o desempenho de diversos algoritmos, o Vader obteve excelentes [resultados](#), ficando entre os mais precisos em diferentes teste. Porém, esse algoritmo e boa parte de outros que são similares, não fazem essa análise em língua portuguesa.

Devido a grande quantidade de tweets em algumas coletas, a tradução dos tweets para inglês para depois passarem por um desses algoritmos, se mostrou demorado demais. As API's de tradução como gloogletrans e translate também apresentaram erro devido ao alto número de requisições em um curto período de tempo. Sendo descartada a opção de utilizar um algoritmo pronto, a alternativa para nosso projeto foi criar um algoritmo próprio, utilizando Multinomial NB e treinado com os datasets obtidos no Kaggle. Esse código está presente no arquivo “modelo_analise_de_sentimentos.py”

```
# Importação de pacotes:
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
# Criação do Algoritmo de análise de sentimentos:
# Leitura dos dados de treino:
df_treino = pd.read_csv("dados_base_analise_de_sentimentos/df_treino.csv")
# Função de tratamento do texto:
tf_vectorizer = TfidfVectorizer(stop_words = portuguese_stops, analyzer='word',
                                ngram_range=(1, 1), lowercase=True, use_idf=True,
                                strip_accents='unicode')
# Definição de variáveis de treinamento e aplicação do tratamento de texto:
treino_x = tf_vectorizer.fit_transform(df_treino['tweets'])
treino_y = df_treino['sentimento']
# Criação do modelo de análise:
classificador_MNB = MultinomialNB()
# Treinamento do modelo:
classificador_MNB.fit(treino_x, treino_y)
```

Teste do modelo:

```
# Leitura dos dados de teste:
df_teste = pd.read_csv("dados_base_analise_de_sentimentos/df_teste.csv")
# Definição de variáveis de teste e aplicação da função de tratamento de texto:
teste_x = tf_vectorizer.transform(df_teste['tweets'])
teste_y = df_teste['sentimento']
# Aplicação do modelo nos dados de teste:
predicao_MNB_teste = classificador_MNB.predict(teste_x)
resultado_MNB_teste = pd.Series(predicao_MNB_teste)
# Resultados da análise nos dados de teste:
resultado_MNB_teste.value_counts()
```

Resultado do teste do modelo ao ser aplicado no dataset df_teste:

Neutro	1765
Negativo	1666
Positivo	1568
dtype: int64	

Avaliação da precisão do modelo:

```
# Matrix de confusão:
print (confusion_matrix(teste_y, resultado_MNB_teste))

precision    recall   f1-score   support
Negativo      0.72      0.72      0.72      1666
    Neutro      0.92      0.98      0.95      1666
    Positivo     0.72      0.67      0.70      1667

   micro avg     0.79      0.79      0.79      4999
   macro avg     0.79      0.79      0.79      4999
weighted avg     0.79      0.79      0.79      4999

# Taxa de acurácia do modelo:
metrics.accuracy_score(teste_y, resultado_MNB_teste)

0.7903580716143228
```

Nosso modelo apresentou uma taxa de 79% de acerto. Como nosso projeto faz análises de forma genérica e foi treinado com tweets também genéricos, essa taxa pode ser considerada um bom valor. Para melhora-lá pode-se treinar o modelo

com tweets mais similares aos que ele irá analisar. Por exemplo, Se o objetivo é analisar tweets sobre cinema, treinar com tweets sobre cinema.

Aplicando o modelo de predição de sentimentos nos tweets coletados:

```
# Aplicação da função de tratamento de texto:
analise_sent_tweets = tf_vectorizer.transform(df['text'])
# Aplicação do modelo
predicao_MNB = pd.Series(classificador_MNB.predict(analise_sent_tweets))
# Armazenamento dos resultados em uma nova variável chamada "analise_sentimento":
df['analise_sentimento'] = predicao_MNB
```

Foi mencionado anteriormente que também usariamos o algoritmo Vader. Inicialmente ele foi descartado por trabalharmos com tweets em português. Porém durante as análises surgiu uma oportunidade para o utilizarmos e melhorar nossas previsões.

O modelo Multinomial não trabalha com emojis, e durante o tratamento do texto eles são removidos. Entretanto, os emojis são cada vez mais usados e podem fazer total diferença no sentimento do conteúdo.

```
# frases com sentimentos Neutro, Positivo e Negativo
frases_para_teste = pd.Series(['Estou indo comprar um computador novo.',
                               'Estou indo comprar um computador novo. 😊',
                               'Estou indo comprar um computador novo. 😞'])
# Análise com modelo multinomialNB
resultado_teste_MNB = pd.Series(classificador_MNB.predict(
    tf_vectorizer.transform(frases_para_teste)))
# Resultado:
resultado_teste_MNB
0    Positivo
1    Positivo
2    Positivo
dtype: object
```

Podemos considerar que nosso modelo fez uma boa classificação, apesar da primeira frase possuir uma interpretação dúbia. Porém na última o emoji dar claramente uma conotação negativa a frase, e ela acaba não sendo captada.

O algoritmo Vader trabalha muito bem com emojis como veremos a seguir. Note que a primeira ele irá classificar como neutro por estar em um idioma em que ele não foi treinado:

```
# Vader
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
# Criação do modelo Vader:
analyser = SentimentIntensityAnalyzer()
def vader(text):
    score = analyser.polarity_scores(text)
    lb = score['compound']
    if lb >= 0.05:
        return 'Positivo'
    elif (lb > -0.05) and (lb < 0.05):
        return 'Neutro'
    else:
        return 'Negativo'
# Aplicação:
resultado_teste_Vader = frases_para_teste.apply(lambda x: vader(x))
# Resultado:
resultado_teste_Vader
0    Neutro
1    Positivo
2   Negativo
dtype: object
```

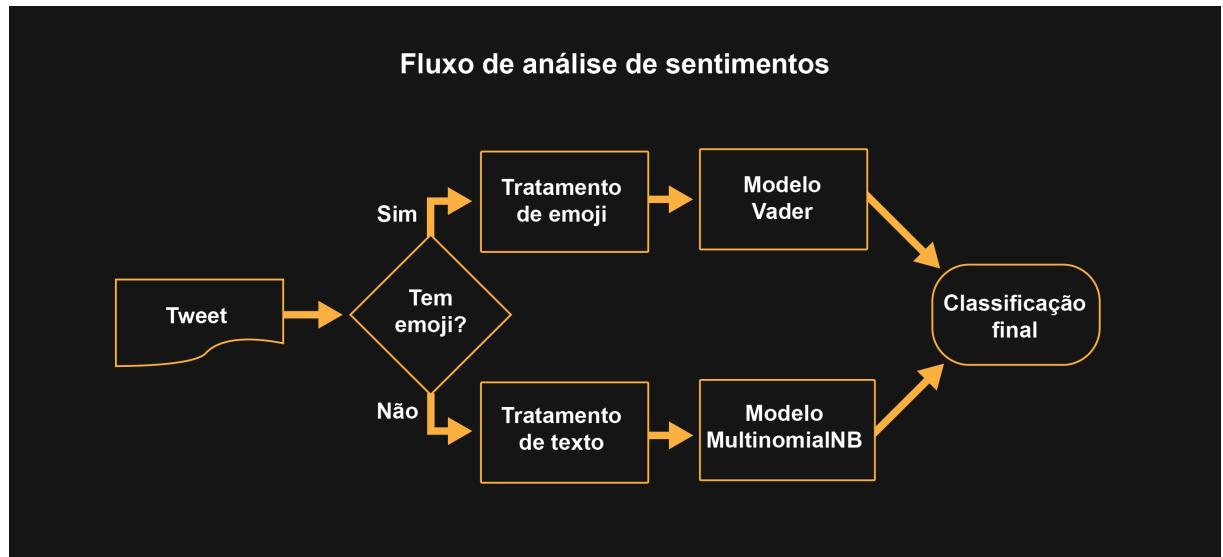
Outra amostra de como o Vader consegue fazer cálculos com emojis e determinar se eles são positivos ou não:

```
# Novo teste com Vader:
frases_para_teste_2 = pd.Series(['😊', '👍👎', '😃😄😁', '😔😢😟'])
# Aplicação:
resultado_teste_Vader = frases_para_teste_2.apply(lambda x: vader(x))
# Resultado:
resultado_teste_Vader
0    Positivo
1    Neutro
2    Positivo
3   Negativo
dtype: object
```

Emojis tem sido cada vez mais usados, então não podemos deixar de incluí-los em nossa análise. Por isso, vamos verificar quais são os tweets com emojis e extraí-los para uma nova variável, e classificá-los com o modelo Vader.

Como os emojis possuem um peso maior em relação a sentimentos do que as palavras, aqueles tweets em que o resultado do Vader for positivo ou negativo, a classificação será a do modelo Vader.

Nos casos onde os tweets não tenha emojis, ou tenha e o resultado do Vader seja neutro, será mantida a classificação feita pelo modelo MultinomialNB.



Código de classificação de tweets com emoji e substituição de análises::

```

# Emoji
import emoji
# Análise de tweets com emojis:
# Função de extração:
def extracao_de_emojis(str):
    return ''.join(c for c in str if c in emoji.UNICODE_EMOJI)
# Criação da nova variável contendo os emojis:
df['emojis_extraidos'] = df['text'].apply(lambda x: extracao_de_emojis(x))
# Aplicação do modelo Vader
df['analise_sent_emoji'] = df['emojis_extraidos'].apply(lambda x: vader(x))
# Substituição das análises dos tweets com emojis
df.loc[df['analise_sent_emoji'] == 'Positivo', 'analise_sentimento'] = 'Positivo'
df.loc[df['analise_sent_emoji'] == 'Negativo', 'analise_sentimento'] = 'Negativo'
  
```

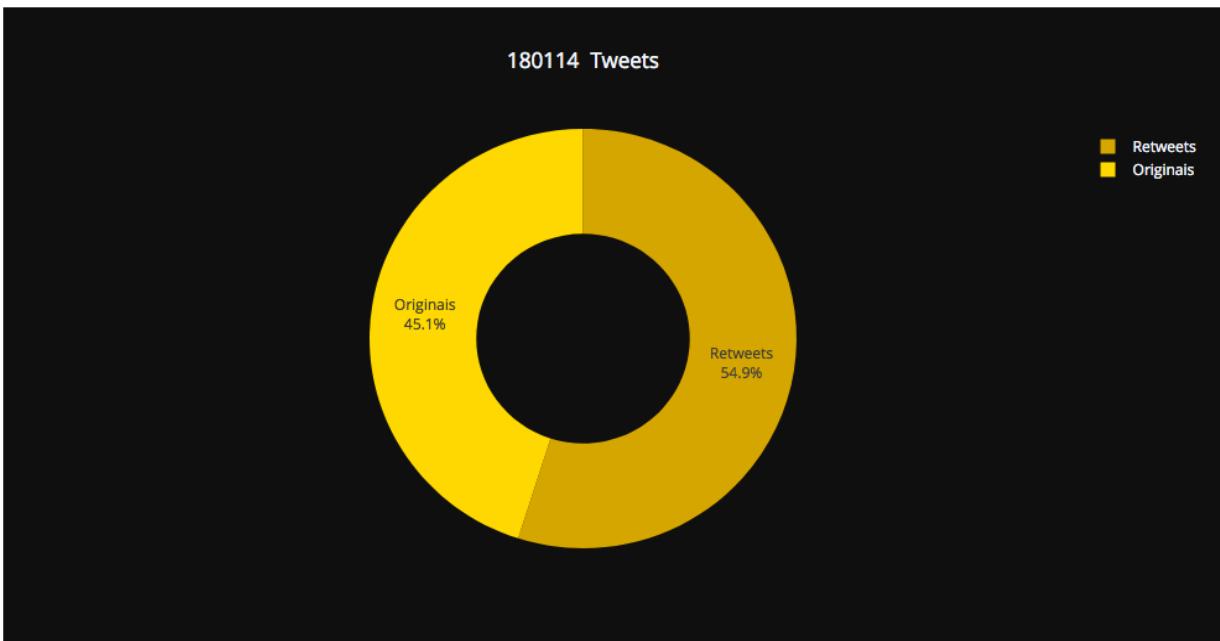
6. Apresentação dos Resultados

6.1. Visualização em gráficos

Para visualizar os resultados das análises feitas na etapa anterior, utilizamos em nosso projeto o Plotly. Um framework para a criação de gráficos e dashboard para a linguagem python. A seguir temos os códigos que estão presente nos arquivos de “relatorio.ipynb” e “dashboard.py” que possibilitam a visualização das análises:

Quantidade de tweets:

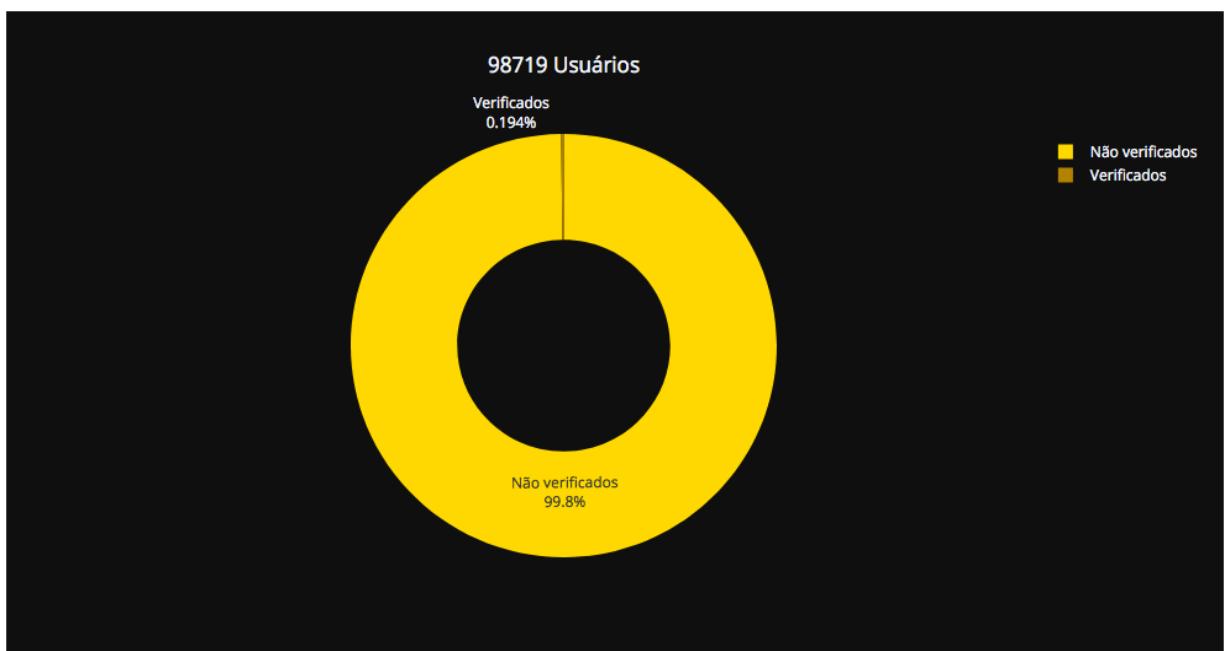
```
# Plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots
# Gráfico da quantidade total de tweets:
total_tt_st = str(total_tweets) + " Tweets"
fig = go.Figure(data = [go.Pie(labels = ['Originais','Retweets'],values = [
    originais, retweets],hole = .5, marker_colors = ['gold', 'goldenrod'],
    textinfo = 'label+percent', hoverinfo = 'value')])
fig.update_layout(template="plotly_dark",
                  title = go.layout.Title(text = total_tt_st,xref = "paper", x=0.5))
fig.show()
# Visualização dos resultados:
print("Quantidade de tweets originais: ", originais)
print("Quantidade de retweets: ", retweets)
print("Quantidade total dos tweets: ", total_tweets)
```



Quantidade de tweets originais: 81214
 Quantidade de retweets: 98900
 Quantidade total dos tweets: 180114

Quantidade de usuários únicos:

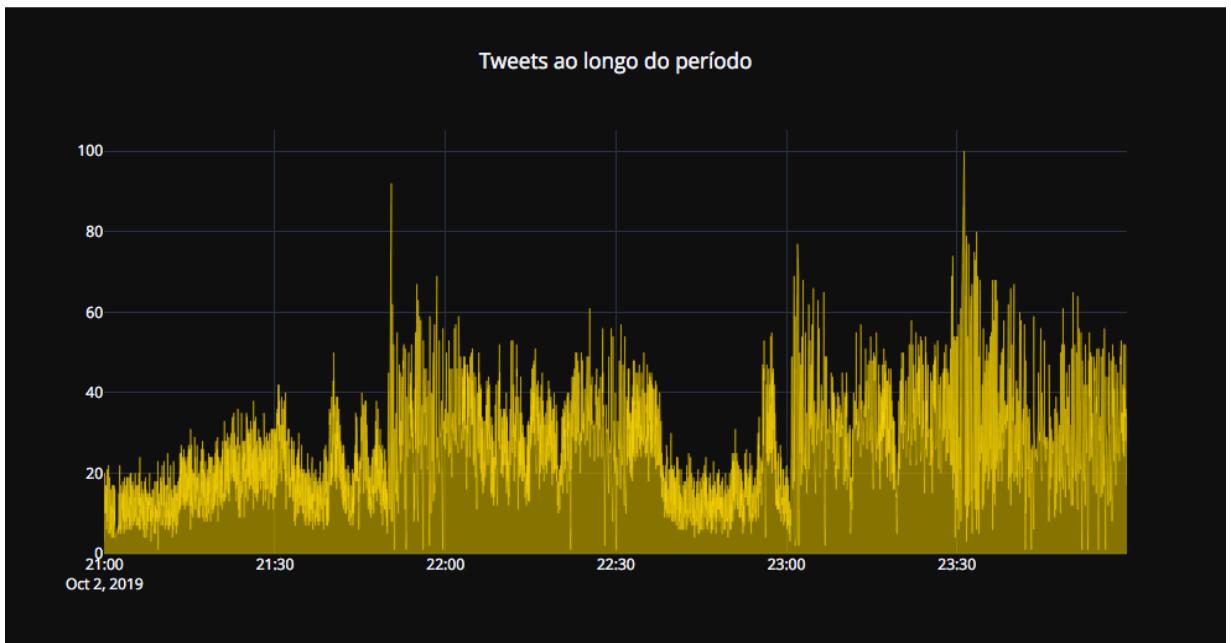
```
# Gráfico da quantidade total de usuários:
total_us_st = str(qtd_usuarios_unicos) + " Usuários"
fig = go.Figure(data = [go.Pie(labels = ['Verificados', 'Não verificados'],
                                 values = [qtd_usuarios_verificados, qtd_usuarios_nao_verificados],
                                 hole = .5, marker_colors =['darkgoldenrod','gold'],
                                 textinfo = 'label+percent', hoverinfo = 'value')])
fig.update_layout(template="plotly_dark",
                  title = go.layout.Title(text = total_us_st, xref = "paper", x=0.5))
fig.show()
# Visualização dos resultados:
print("Quantidade de usuários verificados: ", qtd_usuarios_verificados)
print("Quantidade de usuários não verificados: ", qtd_usuarios_nao_verificados)
print("Quantidade total de usuários: ", qtd_usuarios_unicos)
```



Quantidade de usuários verificados: 192
 Quantidade de usuários não verificados: 98527
 Quantidade total de usuários: 98719

Quantidade de tweets ao longo do período de coleta:

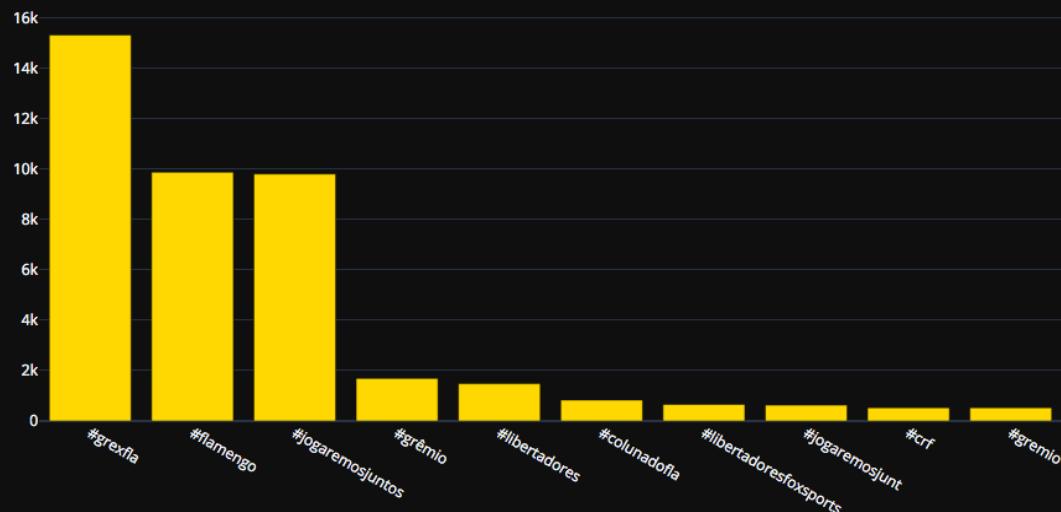
```
# Cria um dataframe com a quantidade de tweets por horário:  
tw_x_pd = df['data_hora'].value_counts().to_frame().reset_index()  
tw_x_pd.columns = ['data_hora', 'qtd_tweets']  
# Ordenar pelo horário:  
tw_x_pd = tw_x_pd.sort_values(by=['data_hora'])  
# Gráfico da quantidade ao longo do período:  
fig = go.Figure(data = [go.Scatter(x = tw_x_pd['data_hora'], y = tw_x_pd['qtd_tweets'],  
fill = 'tozero', mode = 'lines', line = dict(color = 'gold', width = 0.5))])  
fig.update_layout(template = "plotly_dark",  
title = go.layout.Title(text = "Tweets ao longo do período", xref  
= "paper", x = 0.5))  
fig.show()
```



Top hashtags usadas

```
# Gráfico de contagem de hashtags:
fig = go.Figure(data = [go.Bar(y = top_hashtags['count'],
                                x = top_hashtags['palavra'], marker_color='gold')])
fig.update_layout(template = "plotly_dark", title = go.layout.Title(
    text = "Top hashtags #", xref = "paper", x = 0.5))
fig.show()
# Visualização dos resultados:
print('1ºlugar: ', top_hashtags.iloc[0][0] + ' | Quantidade: ',
      top_hashtags.iloc[0][1])
print('2ºlugar: ', top_hashtags.iloc[1][0] + ' | Quantidade: ',
      top_hashtags.iloc[1][1])
print('3ºlugar: ', top_hashtags.iloc[2][0] + ' | Quantidade: ',
      top_hashtags.iloc[2][1])
print('4ºlugar: ', top_hashtags.iloc[3][0] + ' | Quantidade: ',
      top_hashtags.iloc[3][1])
print('5ºlugar: ', top_hashtags.iloc[4][0] + ' | Quantidade: ',
      top_hashtags.iloc[4][1])
print('6ºlugar: ', top_hashtags.iloc[5][0] + ' | Quantidade: ',
      top_hashtags.iloc[5][1])
print('7ºlugar: ', top_hashtags.iloc[6][0] + ' | Quantidade: ',
      top_hashtags.iloc[6][1])
print('8ºlugar: ', top_hashtags.iloc[7][0] + ' | Quantidade: ',
      top_hashtags.iloc[7][1])
print('9ºlugar: ', top_hashtags.iloc[8][0] + ' | Quantidade: ',
      top_hashtags.iloc[8][1])
print('10ºlugar: ', top_hashtags.iloc[9][0] + ' | Quantidade: ',
      top_hashtags.iloc[9][1])
```

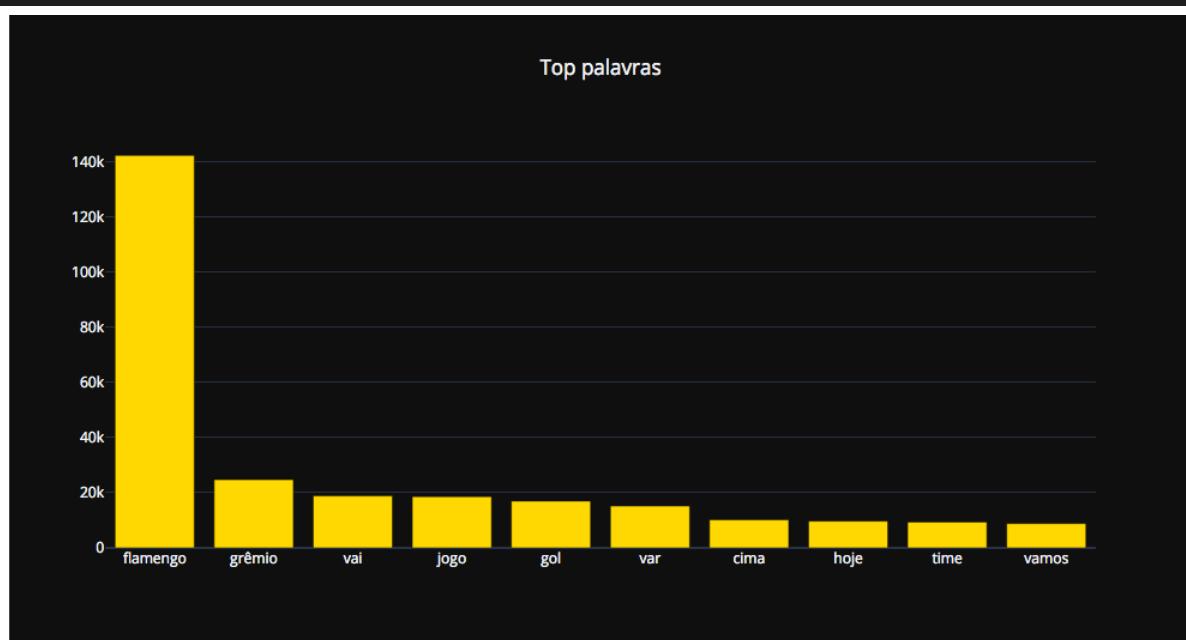
Top hashtags #



```
1ºlugar: #grexfla | Quantidade: 15329
2ºlugar: #flamengo | Quantidade: 9880
3ºlugar: #jogaremosjuntos | Quantidade: 9811
4ºlugar: #grêmio | Quantidade: 1691
5ºlugar: #libertadores | Quantidade: 1482
6ºlugar: #colunadofla | Quantidade: 834
7ºlugar: #libertadoresfoxsports | Quantidade: 660
8ºlugar: #jogaremosjunt | Quantidade: 630
9ºlugar: #crf | Quantidade: 530
10ºlugar: #gremio | Quantidade: 530
```

Top palavras mais presentes:

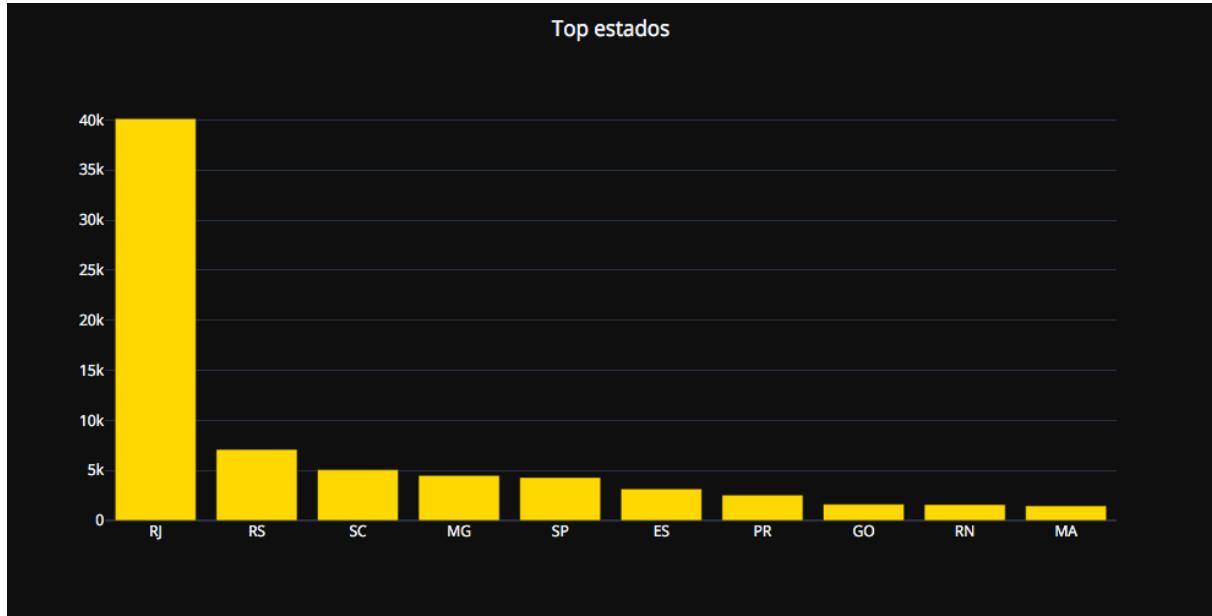
```
# Gráfico de contagem de palavras relevantes:
fig = go.Figure(data = [go.Bar(x = top_contagem_palavras_relevantes['palavra'],
                                y = top_contagem_palavras_relevantes['count'],
                                marker_color='gold')])
fig.update_layout(template = "plotly_dark", title = go.layout.Title(
    text = "Top palavras", xref = "paper", x = 0.5))
fig.show()
# Visualização dos resultados:
print('1ºlugar: ', top_contagem_palavras_relevantes.iloc[0][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[0][1])
print('2ºlugar: ', top_contagem_palavras_relevantes.iloc[1][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[1][1])
print('3ºlugar: ', top_contagem_palavras_relevantes.iloc[2][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[2][1])
print('4ºlugar: ', top_contagem_palavras_relevantes.iloc[3][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[3][1])
print('5ºlugar: ', top_contagem_palavras_relevantes.iloc[4][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[4][1])
print('6ºlugar: ', top_contagem_palavras_relevantes.iloc[5][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[5][1])
print('7ºlugar: ', top_contagem_palavras_relevantes.iloc[6][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[6][1])
print('8ºlugar: ', top_contagem_palavras_relevantes.iloc[7][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[7][1])
print('9ºlugar: ', top_contagem_palavras_relevantes.iloc[8][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[8][1])
print('10ºlugar: ', top_contagem_palavras_relevantes.iloc[9][0] + ' | Quantidade: ', top_contagem_palavras_relevantes.iloc[9][1])
```



```
1ºlugar: flamengo | Quantidade: 142322
2ºlugar: grêmio | Quantidade: 24717
3ºlugar: vai | Quantidade: 18844
4ºlugar: jogo | Quantidade: 18600
5ºlugar: gol | Quantidade: 16989
6ºlugar: var | Quantidade: 15171
7ºlugar: cima | Quantidade: 10177
8ºlugar: hoje | Quantidade: 9713
9ºlugar: time | Quantidade: 9356
10ºlugar: vamos | Quantidade: 8824
```

Top estados com mais tweets:

```
# Verificação da quantidade de localizações estaduais obtidas:  
tweets_localizados = sum(df['estado'].value_counts())  
pct_tweets_localizados = round((tweets_localizados*100)/len(df),2)  
# Gráfico de contagem de palavras relevantes:  
fig = go.Figure(data = [go.Bar(x = top_estados['estado'],  
                                y = top_estados['qtd_tweets'], marker_color = 'gold')])  
fig.update_layout(template = "plotly_dark", title = go.layout.Title(  
                    text = "Top estados", xref = "paper", x = 0.5))  
fig.show()  
# Visualização dos resultados:  
print("Tweets localizados: ", tweets_localizados)  
print("Percentual de tweets localizados: ", pct_tweets_localizados,"%")  
print('1ºlugar: ', top_estados.iloc[0][0] + ' | Quantidade: ',  
top_estados.iloc[0][1])  
print('2ºlugar: ', top_estados.iloc[1][0] + ' | Quantidade: ',  
top_estados.iloc[1][1])  
print('3ºlugar: ', top_estados.iloc[2][0] + ' | Quantidade: ',  
top_estados.iloc[2][1])  
print('4ºlugar: ', top_estados.iloc[3][0] + ' | Quantidade: ',  
top_estados.iloc[3][1])  
print('5ºlugar: ', top_estados.iloc[4][0] + ' | Quantidade: ',  
top_estados.iloc[4][1])  
print('6ºlugar: ', top_estados.iloc[5][0] + ' | Quantidade: ',  
top_estados.iloc[5][1])  
print('7ºlugar: ', top_estados.iloc[6][0] + ' | Quantidade: ',  
top_estados.iloc[6][1])  
print('8ºlugar: ', top_estados.iloc[7][0] + ' | Quantidade: ',  
top_estados.iloc[7][1])  
print('9ºlugar: ', top_estados.iloc[8][0] + ' | Quantidade: ',  
top_estados.iloc[8][1])  
print('10ºlugar: ', top_estados.iloc[9][0] + ' | Quantidade: ',  
top_estados.iloc[9][1])
```



```
Tweets localizados: 84071
Percentual de tweets localizados: 46.68 %
1ºlugar: RJ | Quantidade: 40141
2ºlugar: RS | Quantidade: 7093
3ºlugar: SC | Quantidade: 5073
4ºlugar: MG | Quantidade: 4497
5ºlugar: SP | Quantidade: 4295
6ºlugar: ES | Quantidade: 3165
7ºlugar: PR | Quantidade: 2547
8ºlugar: GO | Quantidade: 1633
9ºlugar: RN | Quantidade: 1588
10ºlugar: MA | Quantidade: 1457
```

Taxas de tweets por usuário, por minuto e por segundo:

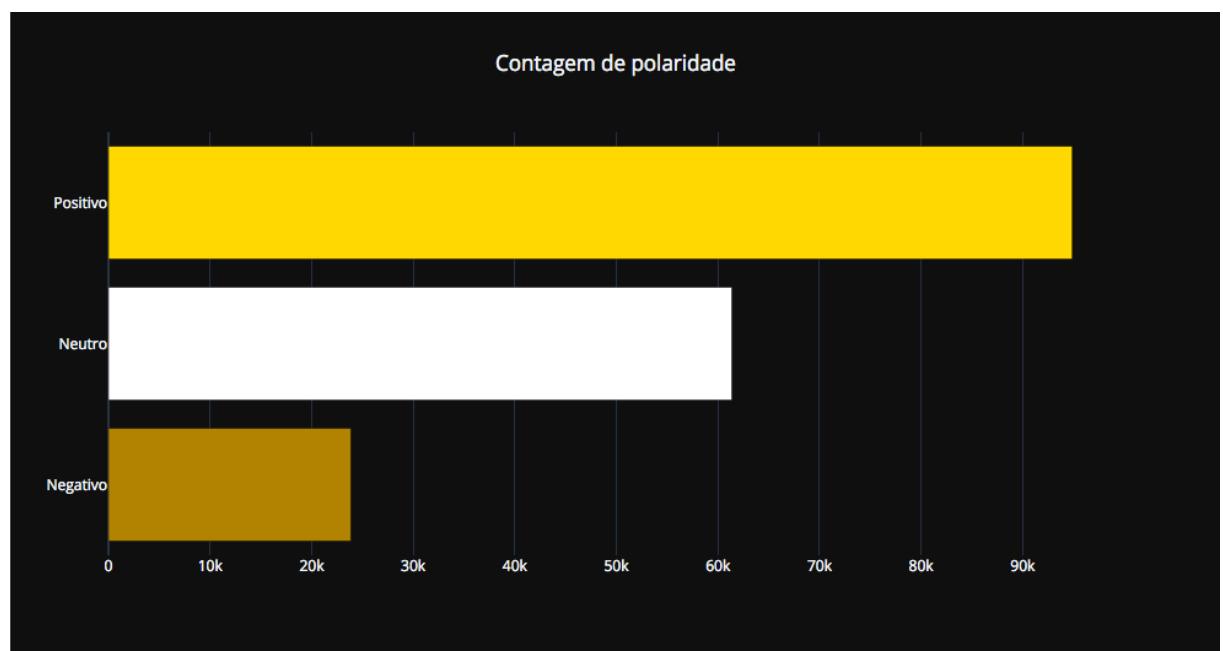
```
# Visualização do resultado:
print("Período de coleta: ", periodo)
print("Taxa média de tweets por usuário: ", media_tweets_por_usuario)
print("Taxa média de tweets por usuário verificado: ", media_tweets_por_usuario_verificado)
print("Taxa média de tweets por usuário não verificado: ", media_tweets_por_usuario_nao_verificado)
print("Taxa média de tweets por minuto: ", tweets_por_minuto)
print("Taxa média de tweets por segundo: ", tweets_por_segundo)
```

Taxas de tweets por usuário, por minuto e por segundo:

```
Período de coleta: 2:59:59
Taxa média de tweets por usuário: 1.82
Taxa média de tweets por usuário verificado: 3.44
Taxa média de tweets por usuário não verificado: 1.82
Taxa média de tweets por minuto: 1000.73
Taxa média de tweets por segundo: 16.68
```

Classificação de sentimentos do tweet como positivo, negativo ou neutro

```
# Contagem de sentimentos
polaridade_sentimentos = pd.DataFrame(df['analise_sentimento'].value_counts()
).reset_index()
polaridade_sentimentos.columns = ['sentimento', 'num_de_tweets']
# Gráfico com a contagem de polaridade
cores_dic = {'Positivo': 'gold', 'Neutro': 'white', 'Negativo': 'darkgoldenrod'}
cores= polaridade_sentimentos['sentimento'].map(cores_dic)
fig = go.Figure(data = [go.Bar( x = polaridade_sentimentos['num_de_tweets'],
                                y = polaridade_sentimentos['sentimento'],
                                orientation = 'h',
                                marker_color = cores)]).update_yaxes(
                                categoryorder = "total ascending")
fig.update_layout(template = "plotly_dark",
                   title = go.layout.Title(text = "Contagem de polaridade",
                                           xref = "paper", x = 0.5))
fig.show()
# Visualização dos resultados:
print("Total de tweets com sentimento positivo: ", polaridade_sentimentos[
    'num_de_tweets'][polaridade_sentimentos['sentimento'] == 'Positivo'].values[0])
print("Total de tweets com sentimento neutro: ", polaridade_sentimentos[
    'num_de_tweets'][polaridade_sentimentos['sentimento'] == 'Neutro'].values[0])
print("Total de tweets com sentimento negativo: ", polaridade_sentimentos[
    'num_de_tweets'][polaridade_sentimentos['sentimento'] == 'Negativo'].values[0])
```



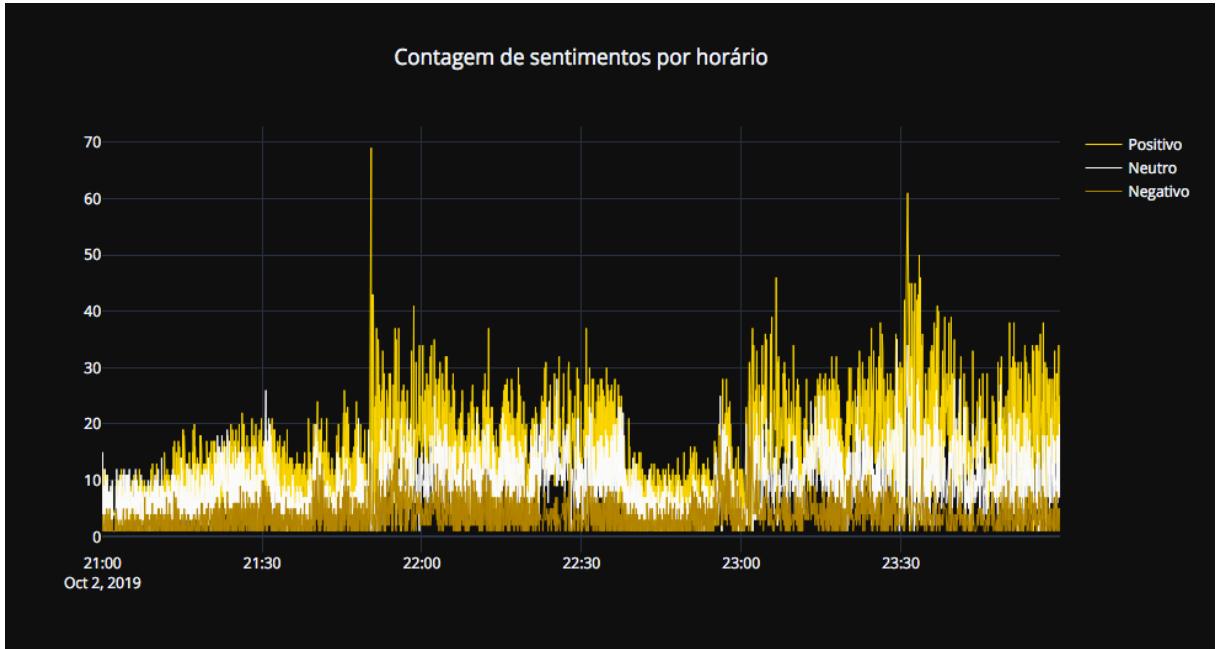
```
Total de tweets com sentimento positivo: 94882
Total de tweets com sentimento neutro: 61368
Total de tweets com sentimento negativo: 23864
```

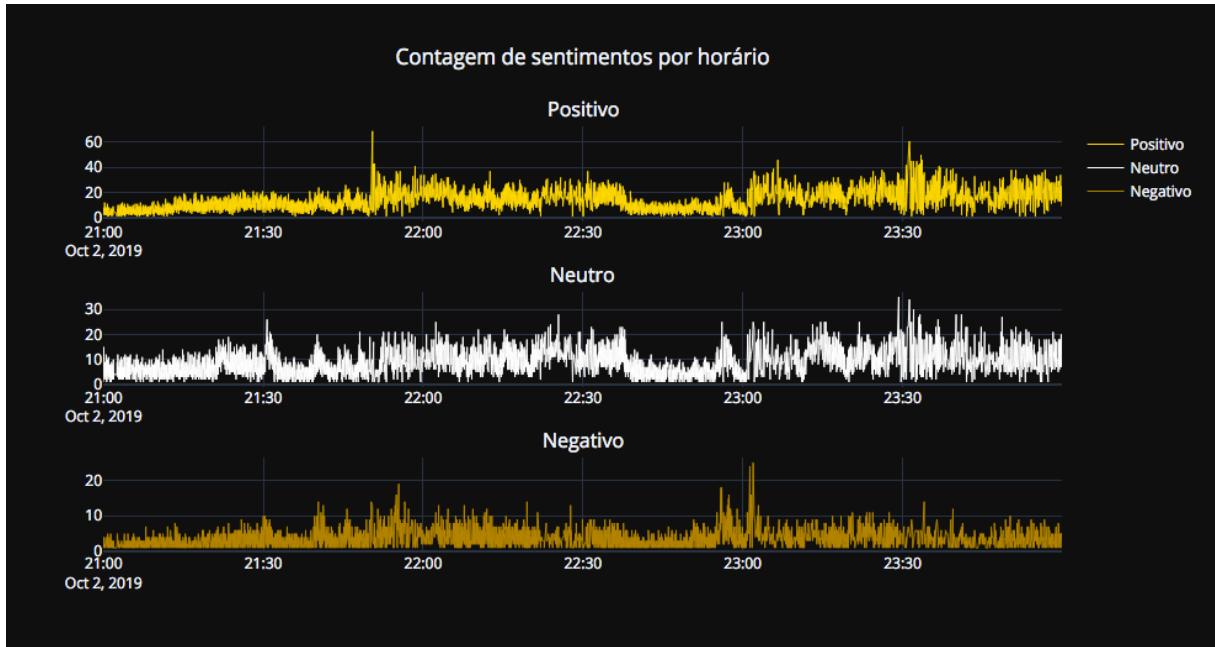
Sentimentos ao longo do período:

```

# Contagem de sentimentos por horário:
# Criação de um dataframe para essa análise:
total_sents = pd.DataFrame()
total_sents['horario'] = df['data_hora']
total_sents['sentimento'] = df['analise_sentimento']
# Contagem dos tweets positivos:
total_sents_pos = total_sents[total_sents['sentimento'] == 'Positivo']
total_sents_pos = total_sents_pos['horario'].value_counts().to_frame()
.reset_index()
total_sents_pos.columns = ['data_hora', 'qtd_tweets']
total_sents_pos = total_sents_pos.sort_values(by=['data_hora'])
# Contagem dos tweets neutros:
total_sents_neu = total_sents[total_sents['sentimento'] == 'Neutro']
total_sents_neu = total_sents_neu['horario'].value_counts().to_frame()
.reset_index()
total_sents_neu.columns = ['data_hora', 'qtd_tweets']
total_sents_neu = total_sents_neu.sort_values(by=['data_hora'])
# Contagem dos tweets negativos:
total_sents_neg = total_sents[total_sents['sentimento'] == 'Negativo']
total_sents_neg = total_sents_neg['horario'].value_counts().to_frame()
.reset_index()
total_sents_neg.columns = ['data_hora', 'qtd_tweets']
total_sents_neg = total_sents_neg.sort_values(by=['data_hora'])
# Gráfico com a contagem de sentimentos por horário:
fig = go.Figure()
fig.add_trace(go.Scatter(x=total_sents_pos['data_hora'],
y=total_sents_pos['qtd_tweets'], line=dict(color='gold', width=1), name="Positivo"))
fig.add_trace(go.Scatter(x=total_sents_neu['data_hora'],
y=total_sents_neu['qtd_tweets'], line=dict(color='white', width=1), name="Neutro"))
fig.add_trace(go.Scatter(x=total_sents_neg['data_hora'],
y=total_sents_neg['qtd_tweets'], line=dict(color='darkgoldenrod', width=1),
name="Negativo"))
fig.update_layout(template = "plotly_dark",
title = go.layout.Title(
text = "Contagem de sentimentos por horário", xref = "paper", x = 0.5))
fig.show()

```





6.2. Dashboard

Um dos objetivos do bem-te-vi é obter todas essas análises a medida em que os tweets são coletados. Para fazer esse acompanhamento, agrupamos boa parte dessas análises em um único dashboard. Por uma questão de espaço exibiremos a análise de sentimentos ao longo do tempo apenas e forma agrupada. E os 5 primeiros elementos de cada Top. Veja a estruturação do dashboard que é usada nos arquivos “relatorio.ipynb” e “dashboard.py”:

```
# Dashboard
# Estrutura:
fig = make_subplots(
    rows=3, cols=4, start_cell="top-left",
    specs=[[{"colspan": 2},None, {"type": "domain"}, {"type": "domain"}],
           [{"colspan": 2},None, {}, {"rowspan": 2,"type": "table"}],
           [{}],{}],None]],
    subplot_titles=("Tweets ao longo do período", total_tt_st, total_us_st,
                   "Sentimento ao longo do período", "Polaridade","", "",
                   "Top 5 hashtags", "Top 5 palavras", "Top 5 estados"))

# Tweets ao longo do período:
fig.add_trace(go.Scatter(x=tw_x_pd['data_hora'], y=tw_x_pd['qtd_tweets'],
                         fill='tozerooy', mode= 'lines',
                         line=dict(color='gold', width=1),name=""),row=1, col=1)

# Quantidade de tweets:
fig.add_trace(go.Pie(labels = ['Originais', 'Retweets'], values = [originais,
                                                               retweets], hole = .7,marker_colors=['gold',
                                                               'darkgoldenrod'],textinfo='label+percent',
```

```

# Quantidade de usuários:
fig.add_trace(go.Pie(labels = ['Verificados', 'Não verificados'],
                      values = [qtd_usuarios_verificados,
                                qtd_usuarios_nao_verificados], hole = .5,
                      marker_colors=['darkgoldenrod','gold'],
                      textinfo='label+percent',
                      hoverinfo='value', rotation=90), row=1, col=3)

# Sentimento ao longo do período:
fig.add_trace(go.Scatter(x=total_sents_pos['data_hora'], y=total_sents_pos['qtd_tweets'], line=dict(color='gold', width=1), name="Positivo"), row=2, col=1)
fig.add_trace(go.Scatter(x=total_sents_neu['data_hora'], y=total_sents_neu['qtd_tweets'], line=dict(color='white', width=1), name="Neutro"), row=2, col=1)
fig.add_trace(go.Scatter(x=total_sents_neg['data_hora'], y=total_sents_neg['qtd_tweets'], line=dict(color='darkgoldenrod', width=1), name="Negativo"), row=2, col=1)

# Polaridade:
cores_dic = {'Positivo': 'gold', 'Neutro': 'white', 'Negativo': 'darkgoldenrod'}
cores = polaridade_sentimentos['sentimento'].map(cores_dic)
fig.add_trace(go.Bar( x=polaridade_sentimentos['num_de_tweets'],
                      y=polaridade_sentimentos['sentimento'], orientation='h',
                      marker_color = cores, name=""),
               row=2, col=3).update_yaxes(categoryorder="total ascending")

# Tabela de taxas:
values_tx = [['', '', '', 'Tweets', 'Por usuário', 'Por usuário verificado',
              'Por usuário não verificado', 'Por minuto', 'Por segundo'],
              ['', '', '', 'Taxa média', media_tweets_por_usuario,
               media_tweets_por_usuario_verificado,
               media_tweets_por_usuario_nao_verificado,
               tweets_por_minuto, tweets_por_segundo ]]
palavras_chaves = str(chaves.keywords).strip('[]')
fig.add_trace(go.Table(columnorder = [1,2], columnwidth = [5,5],
                       header = dict( values = [['Tweets'], ['Taxa média']],
                                      line_color='rgb(122, 94, 8)', fill_color='black',
                                      align=['center', 'center'], font=dict(color='gold', size=11),
                                      height=25),
                       cells=dict(values=values_tx, line_color=[['rgb(122, 94, 8)' if (val != '') else 'rgb(17, 17, 17)']
                                                               for val in values_tx[0]], ['rgb(122, 94, 8)']
                                                               if (val != '') else 'rgb(17, 17, 17)' for val in values_tx[1]],
                                      fill_color='rgb(17, 17, 17)', align=['center', 'center'],
                                      font = dict(color =[['gold'if (val == 'Tweets') else 'white' for val in values_tx[0]]],
                                                   ['gold'if (val == 'Taxa média') else 'white' for val in values_tx[1]]], size = 11), height=25)), row=2, col=4)

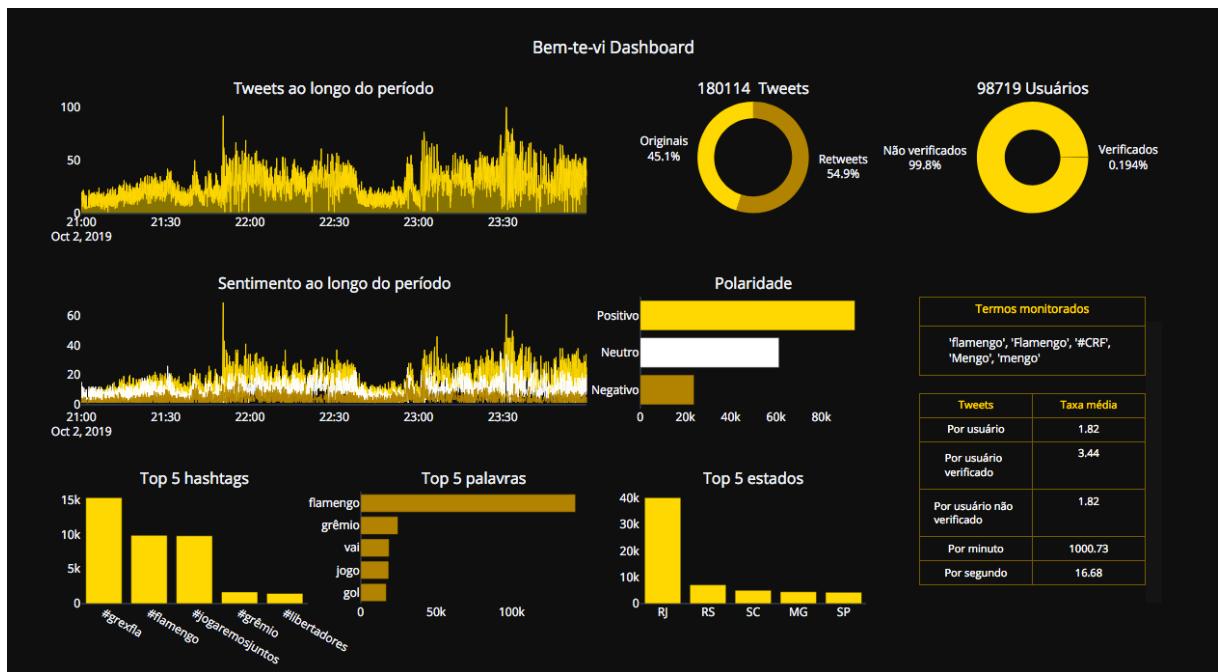
fig.add_trace(go.Table(header=dict(values=['Termos monitorados'],
                                   line_color='rgb(122, 94, 8)', fill_color='rgb(17, 17, 17)',
                                   align=['center', 'center'], font=dict(color='gold', size=12),
                                   height=30), cells=dict(values=[[palavras_chaves]]),
                                   line_color='rgb(122, 94, 8)', fill=dict(color=['rgb(17, 17, 17)', 'rgb(17, 17, 17)']), align=['center', 'center'], font_size=12),
               row=2, col=5)

```

```

        height=30)), row=2, col=4)
# Top 5 Hashtags:
fig.add_trace(go.Bar(y = top5_hashtags['count'], x = top5_hashtags['palavra'],
                      marker_color='gold', name=""), row=3, col=1)
# Top 5 Palavras:
fig.add_trace(go.Bar(y = top5_contagem_palavras_relevantes['palavra'],
                      x = top5_contagem_palavras_relevantes['count'],
                      marker_color='darkgoldenrod', orientation='h',
                      name=""), row=3, col=2)
# Top 5 Estados:
fig.add_trace(go.Bar(x = top5_estados['estado'].head(5),
                     y=top5_estados['qtd_tweets'].head(5),
                     marker_color='gold', name=""), row=3, col=3)
# Configura tema:
fig.update_layout(showlegend=False, title=go.layout.Title(
    text="Bem-te-vi Dashboard", xref="paper", x=0.5),
    template="plotly_dark").update_xaxes(
    showgrid=False).update_yaxes(showgrid=False)
# Cria um arquivo html com o dashboard:
base_de_dados = str(chaves.banco)
fig.write_html('relatorios/' + base_de_dados + '_dashboard.html', auto_open=False)

```



6.3. Relatórios

Pensando em também dar a possibilidade de aprofundar a análise posteriormente. O notebook “gera_relatorio.sh” exporta para dentro da pasta relatórios, três arquivos sobre os tweets coletados: um .html com todos os gráficos

das análises, um .csv com os dados coletados e os dados obtidos, por fim, um outro html contendo o dashboard no estado final da coleta.

6.4. Como utilizar o bem-te-vi

O projeto bem-te-vi está disponível github para ser baixado e também para receber futuras colaborações de usuários que queiram contribuir para melhorá-lo.

Pode ser usado de duas formas: Baixando os arquivos do Github e configurando o ambiente ou baixando uma imagem completa de um sistema operacional Linux com todo o ambiente preparado. O link para a imagem do Linux também está no Github.

Com o ambiente devidamente preparado e com os arquivos do bem-te-vi, o usuário deve abrir o arquivo “chaves.py” e preenchêlo adequadamente com suas chaves de API, palavras-chave, chave de criptografia e nome da base de dados.

Feito isso, é possível fazer as seguintes operações:

Coleta:

Executando o comando via terminal, no diretório do projeto:

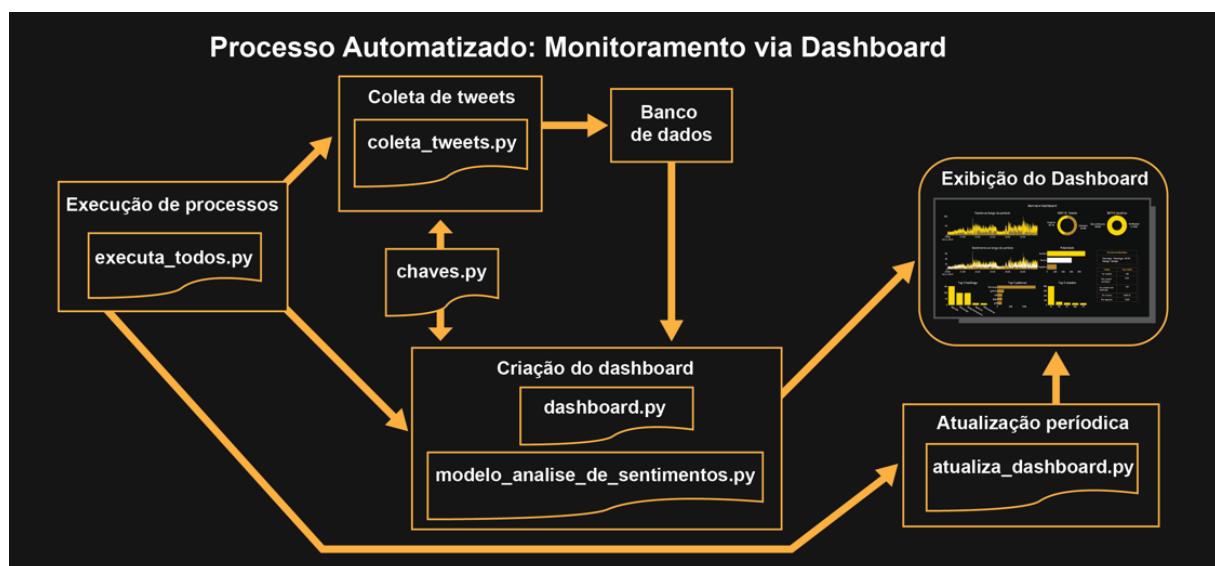
```
python coleta_tweets.py
```

Coleta + Monitoramento simultâneo:

Executando o comando via terminal, no diretório do projeto:

```
python executa.todos.py
```

Esse comando, executa os seguintes processos:



Gerar arquivos de relatório:

Executando o comando via terminal, no diretório do projeto:

```
bash gera_relatorio.sh
```

Esse comando, gera dois arquivos .html e .csv dentro da pasta “relatórios” com as análises obtidas.

6.5. Estudos de caso

Com base nas informações obtidas através dos processos descritos anteriormente, foi possível produzir os seguintes estudos de caso:

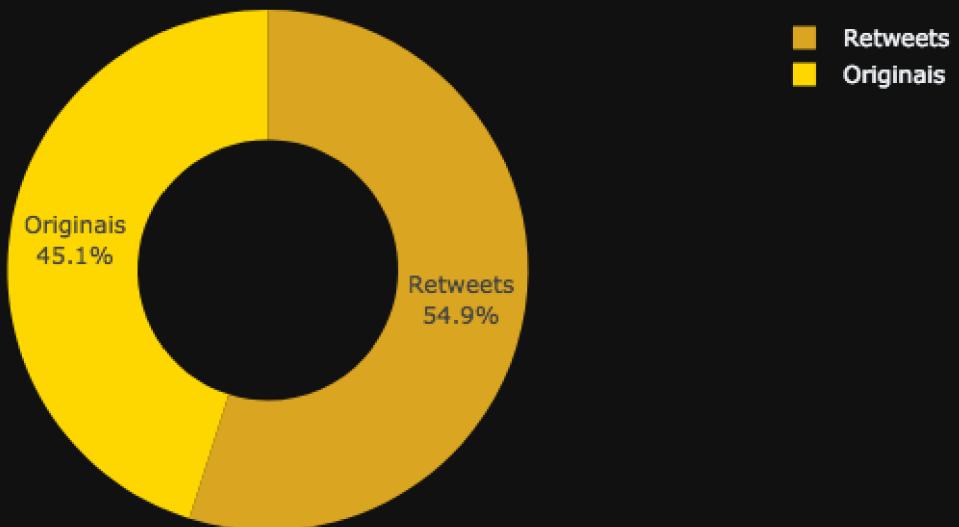
- Comportamento da torcida do Flamengo no twitter durante o jogo Grêmio x Flamengo;
- Comportamento da audiência dos shows do último dia do Rock in Rio;

02/10/2019 - Grêmio x Flamengo

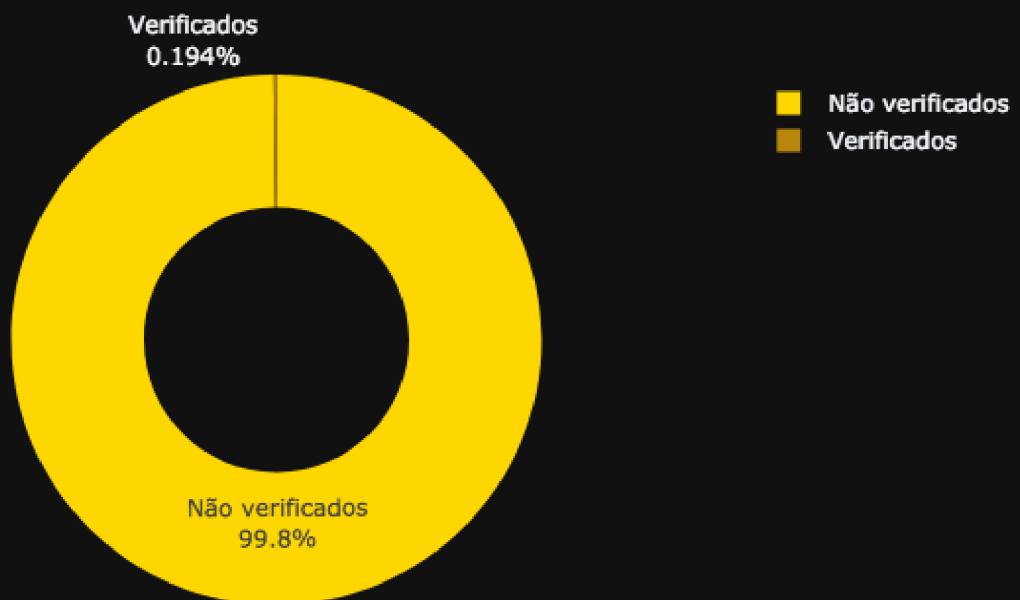
1º Jogo da semifinal da Copa Libertadores da América

Palavras-chave : 'flamengo', 'Flamengo', '#CRF',
'Mengo', 'mengo'

180114 Tweets



98719 Usuários



Tweets em números

Horário do primeiro tweet coletado: 20:59:56

Horário do último tweet coletado: 23:59:55

Tempo total de coleta: 2:59:59

Quantidade de tweets originais: 81214

Quantidade de retweets: 98900

Quantidade total dos tweets: 180114

Quantidade de usuários verificados: 192

Quantidade de usuários não verificados: 98527

Quantidade total de usuários: 98719

Taxa média de tweets por usuário: 1.82

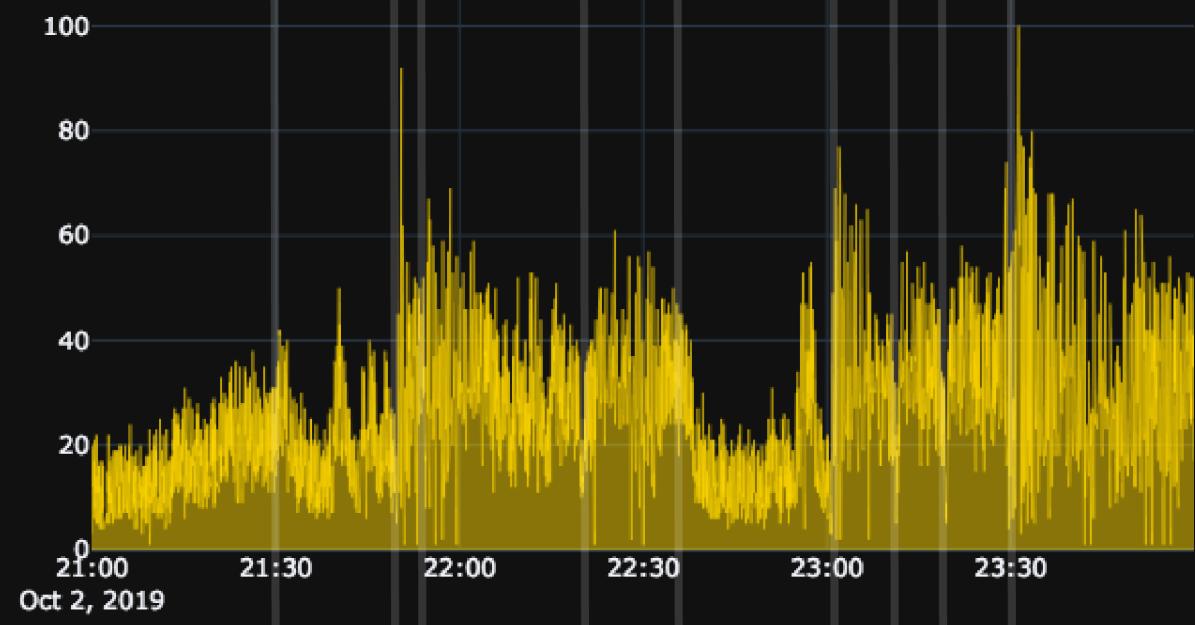
Taxa média de tweets por usuário verificado: 3.44

Taxa média de tweets por usuário verificado: 1.82

Taxa média de tweets por minuto: 1000.73

Taxa média de tweets por segundo: 16.68

Tweets ao longo do período



Oct 2, 2019 1 2 3 4 5 6 7 8 9

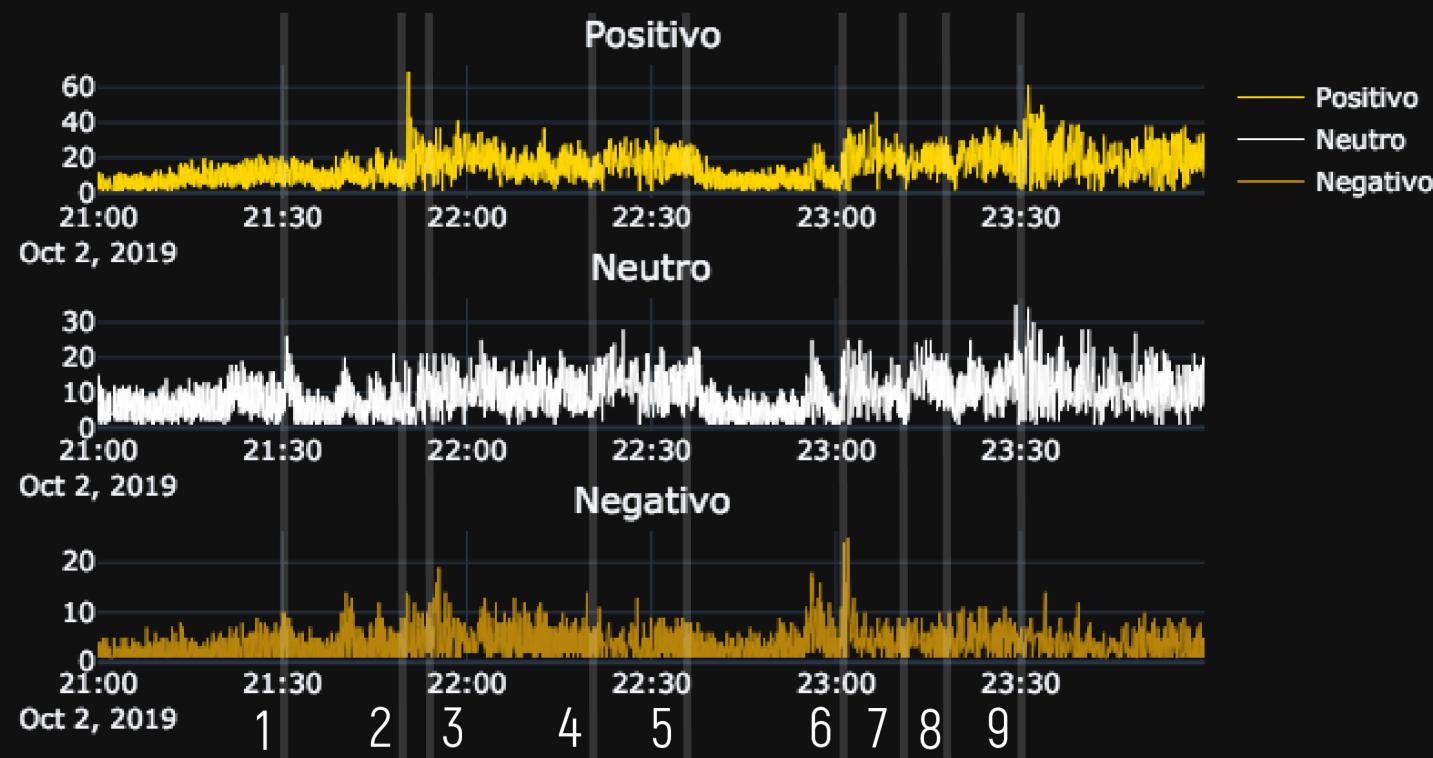
- Positivo
- Neutro
- Negativo

Oct 2, 2019 21:00 21:30 22:00 22:30 23:00 23:30

Momentos

- 1 - Início a partida;
- 2 - Gol anulado do Flamengo;
- 3 - Gol anulado do Flamengo;
- 4 - Fim do primeiro tempo;
- 5 - Início do segundo tempo;
- 6 - Gol do Flamengo;
- 7 - Gol anulado do Flamengo;
- 8 - Gol do Grêmio;
- 9 - Fim da partida.

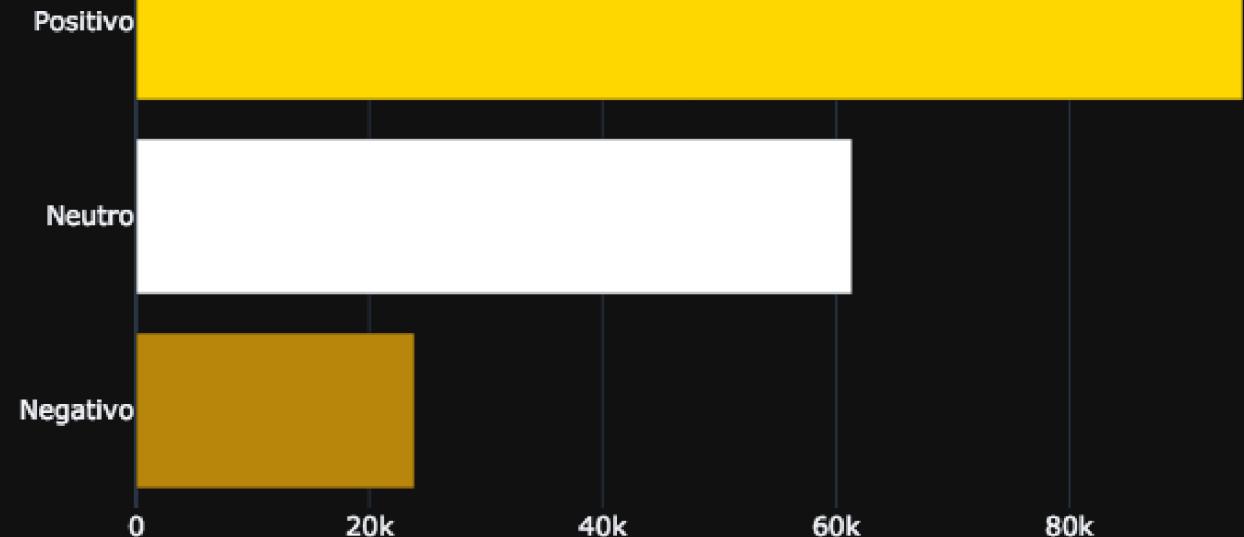
Tweets ao longo do período

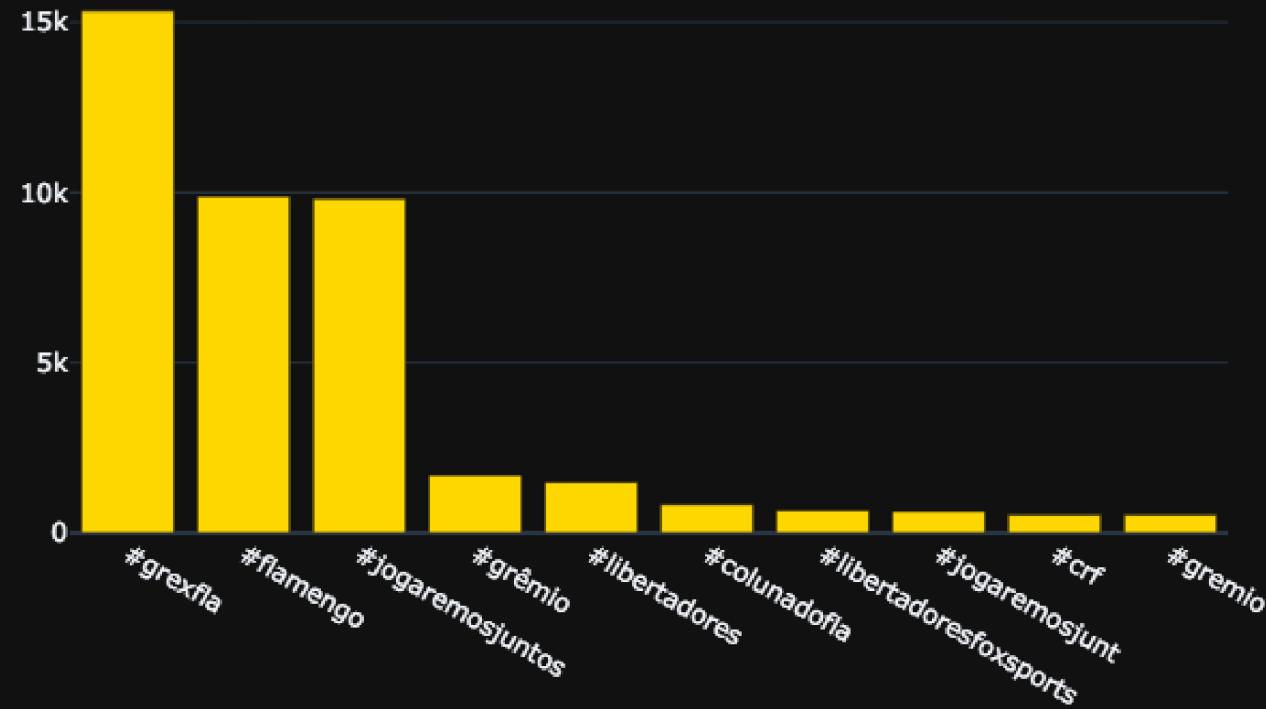


Momentos

- 1 - Início a partida;
- 2 - Gol anulado do Flamengo;
- 3 - Gol anulado do Flamengo;
- 4 - Fim do primeiro tempo;
- 5 - Início do segundo tempo;
- 6 - Gol do Flamengo;
- 7 - Gol anulado do Flamengo;
- 8 - Gol do Grêmio;
- 9 - Fim da partida.

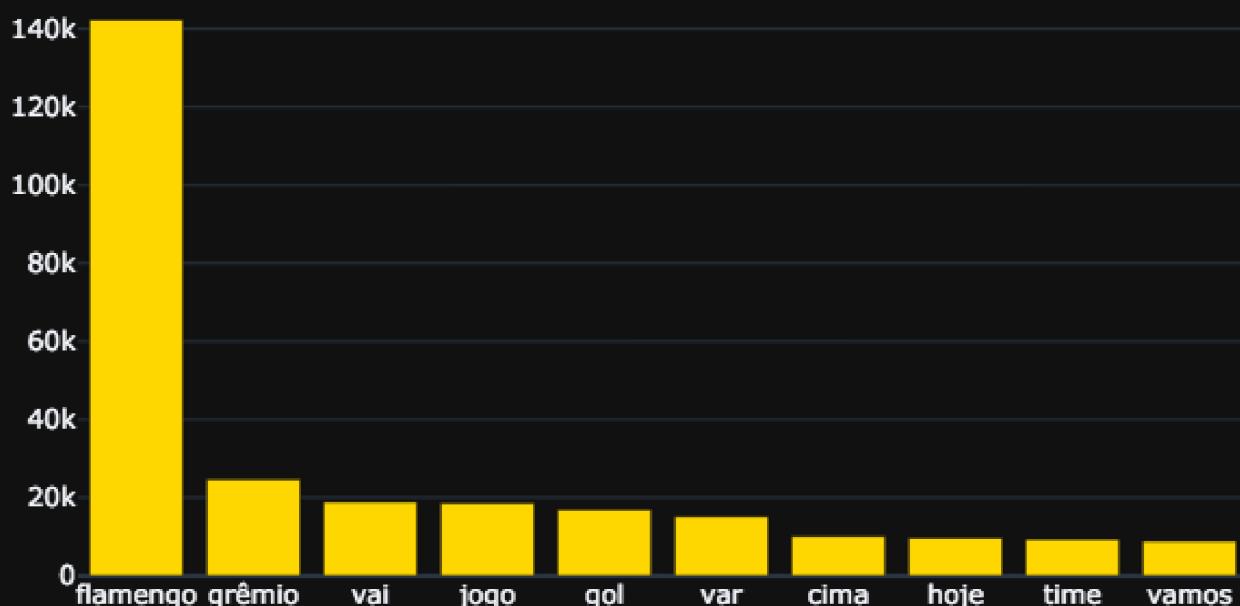
Tweets com sentimento positivo: 94882
 Tweets com sentimento neutro: 61368
 Tweets com sentimento negativo: 23864





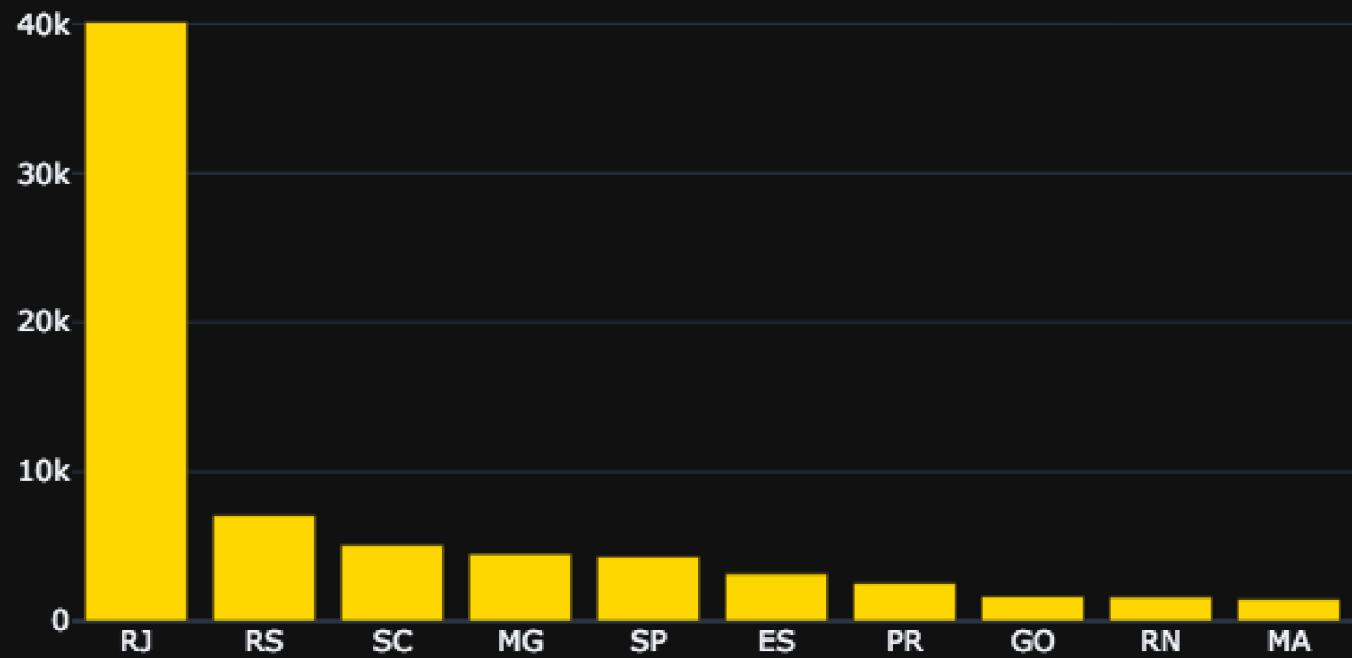
Top 10 Hashtags

- 1ºlugar: #grexfla: 15329
- 2ºlugar: #flamengo: 9880
- 3ºlugar: #jogaremosjuntos: 9811
- 4ºlugar: #grêmio: 1691
- 5ºlugar: #libertadores: 1482
- 6ºlugar: #colunadofla: 834
- 7ºlugar: #libertadoresfoxsports: 660
- 8ºlugar: #jogaremosjunt: 630
- 9ºlugar: #crf : 530
- 10ºlugar: #gremio: 530



Top 10 Palavras

- 1ºlugar: flamengo: 142322
- 2ºlugar: grêmio: 24717
- 3ºlugar: vai: 18844
- 4ºlugar: jogo: 18600
- 5ºlugar: gol: 16989
- 6ºlugar: var: 15171
- 7ºlugar: cima: 10177
- 8ºlugar: hoje: 9713
- 9ºlugar: time: 9356
- 10ºlugar: vamos: 8824



Top 10 estados

Tweets localizados: 84071

Percentual de tweets localizados: 46.68 %

- 1ºlugar: RJ: 40141
- 2ºlugar: RS: 7093
- 3ºlugar: SC: 5073
- 4ºlugar: MG: 4497
- 5ºlugar: SP: 4295
- 6ºlugar: ES: 3165
- 7ºlugar: PR: 2547
- 8ºlugar: GO: 1633
- 9ºlugar: RN: 1588
- 10ºlugar: MA: 1457

Conclusões

- O alto número de tweets em um curto período de tempo, vai de encontro a pesquisas recentes que apontam o Flamengo como um dos times mais comentados do mundo no twitter.
- Nas redes sociais hoje tem uma forte cultura de replicação de publicações. O número de tweets originais não tão distantes dos retweets, demonstra uma forte proximidade dos usuários com o tema, a ponto de sentirem liberdade de dar suas próprias opiniões.
- É possível destacar 3 picos de tweets durante a coleta: O primeiro referente a um gol do rubro-negro anulado no primeiro tempo; O segundo ao gol do Flamengo e o terceiro e maior de todos chegando a ter 100 tweets por segundo, ao apito final da partida.
- No momento do gol válido, há um comportamento diferenciado, primeiro um pico negativo e depois uma onda crescente positiva. Provavelmente a torcida começou utilizando palavras que o modelo de classificação de sentimentos considera negativas, como palavrões e depois comemorou de forma mais positiva. Isso indica que o modelo pode ser melhorado para tratar melhor essas expressões.
- Os tweets são majoritariamente positivos. Sendo o número desses maior que os de neutros e negativos somados.
- A #grexfla que foi a hashtag do jogo e também foi utilizada pela torcida do Grêmio foi a campeã. Mas podemos destacar também a #jogaremosjuntos levantada pela conta do time do Flamengo, e a #colunadofla que é um blog especializado em notícias sobre o clube que terminou a frente de uma das emissoras de tv da partida #libertadoresfoxsports.
- Já em relação as palavras, no meio das de incentivos, é possível notar a presença do VAR. O arbitro de vídeo que atuou bastante no jogo, foi a sexta palavra mais presente.
- Entre os estados, tivemos o esperado, com os locais dos clubes em primeiro e segundo. Seguidos por outros estados das regiões sul-sudeste, centro-oeste e nordeste. Com destaque para Santa Catarina que foi o terceiro colocado.

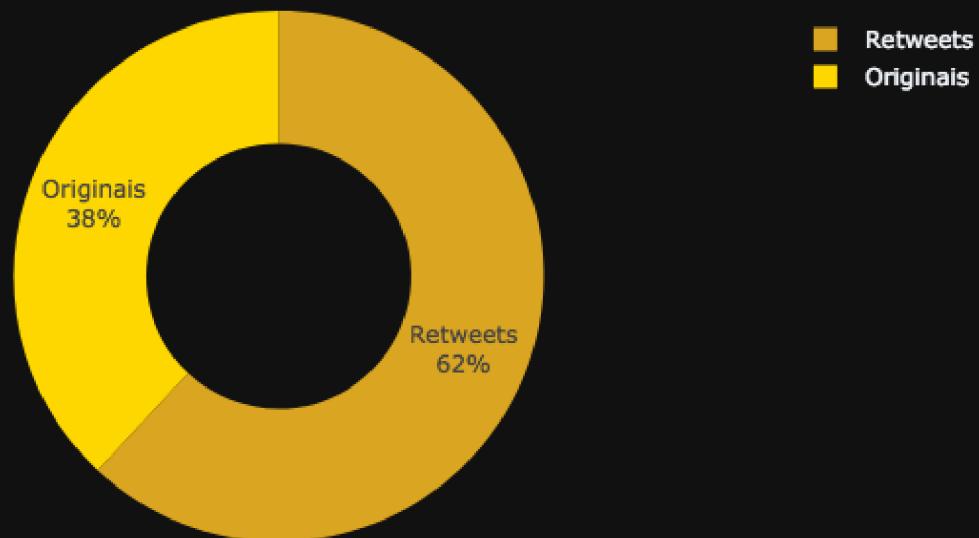


06/10/2019 - Último dia do Rock in Rio 2019

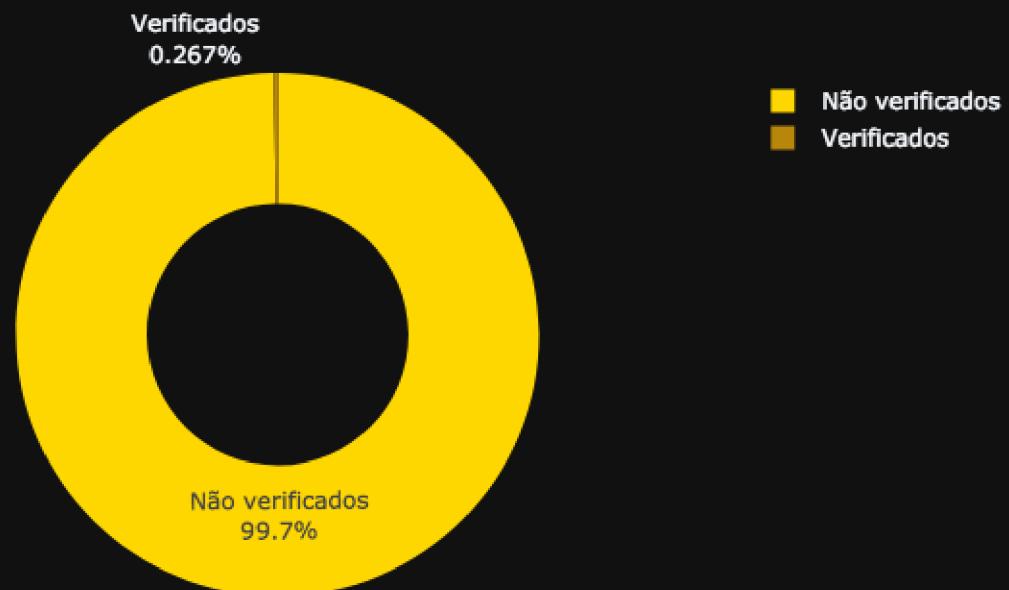
Line-up: Paralamas do sucesso, Nickelback,
Imagine Dragons e muse

Palavras-chave : '#RockinRio2019', 'RockinRio'#',
'Rock in Rio', 'rock in rio', 'ROCK IN RIO'

99327 Tweets



69199 Usuários



Tweets em números

Horário do primeiro tweet coletado: 17:59:58

Horário do último tweet coletado: 02:44:55

Tempo total de coleta: 8:44:57

Quantidade de tweets originais: 37786

Quantidade de retweets: 61541

Quantidade total dos tweets: 99327

Quantidade de usuários verificados: 185

Quantidade de usuários não verificados: 69014

Quantidade total de usuários: 69199

Taxa média de tweets por usuário: 1.44

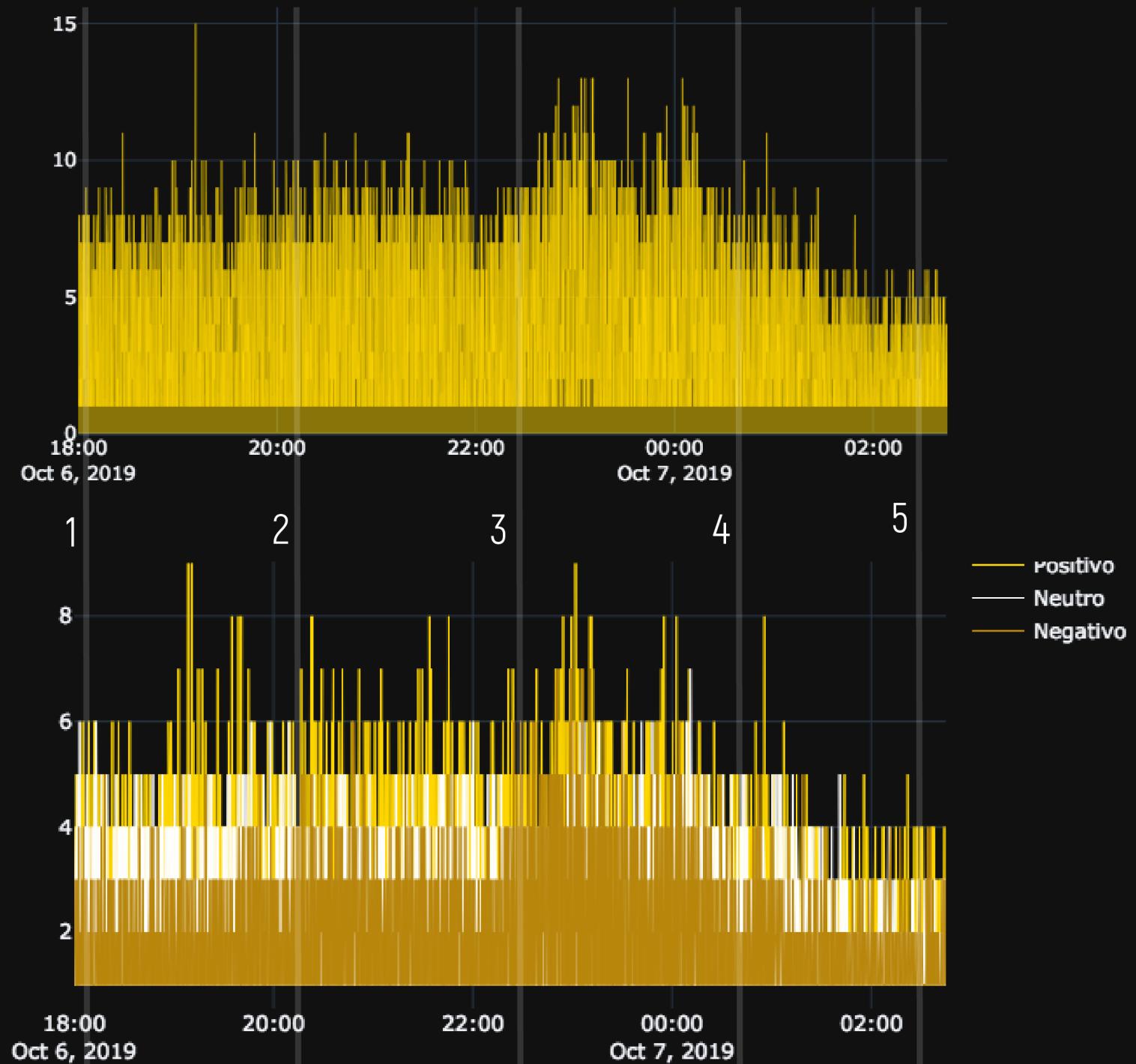
Taxa média de tweets por usuário verificado: 6.7

Taxa média de tweets por usuário verificado: 1.42

Taxa média de tweets por minuto: 189.21

Taxa média de tweets por segundo: 3.15

Tweets ao longo do período



Momentos

- 1 - Show Paralamas do Sucesso;
- 2 - Show Nickelback;
- 3 - Show Imagine Dragons;
- 4 - Show Muse;
- 5 - Fim do evento;

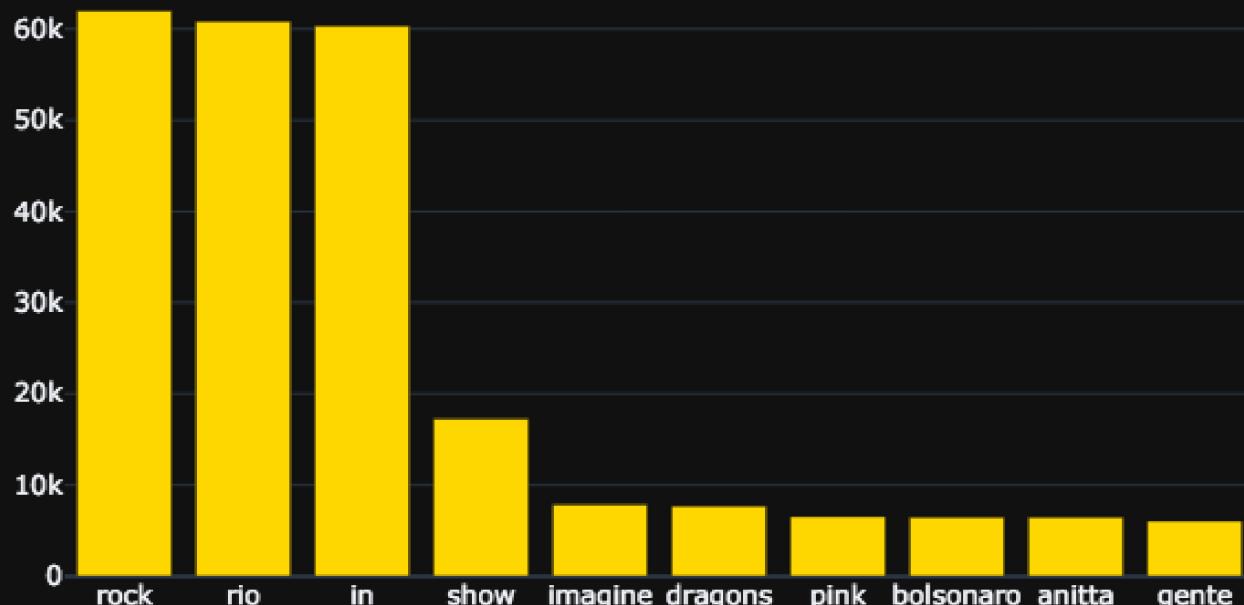
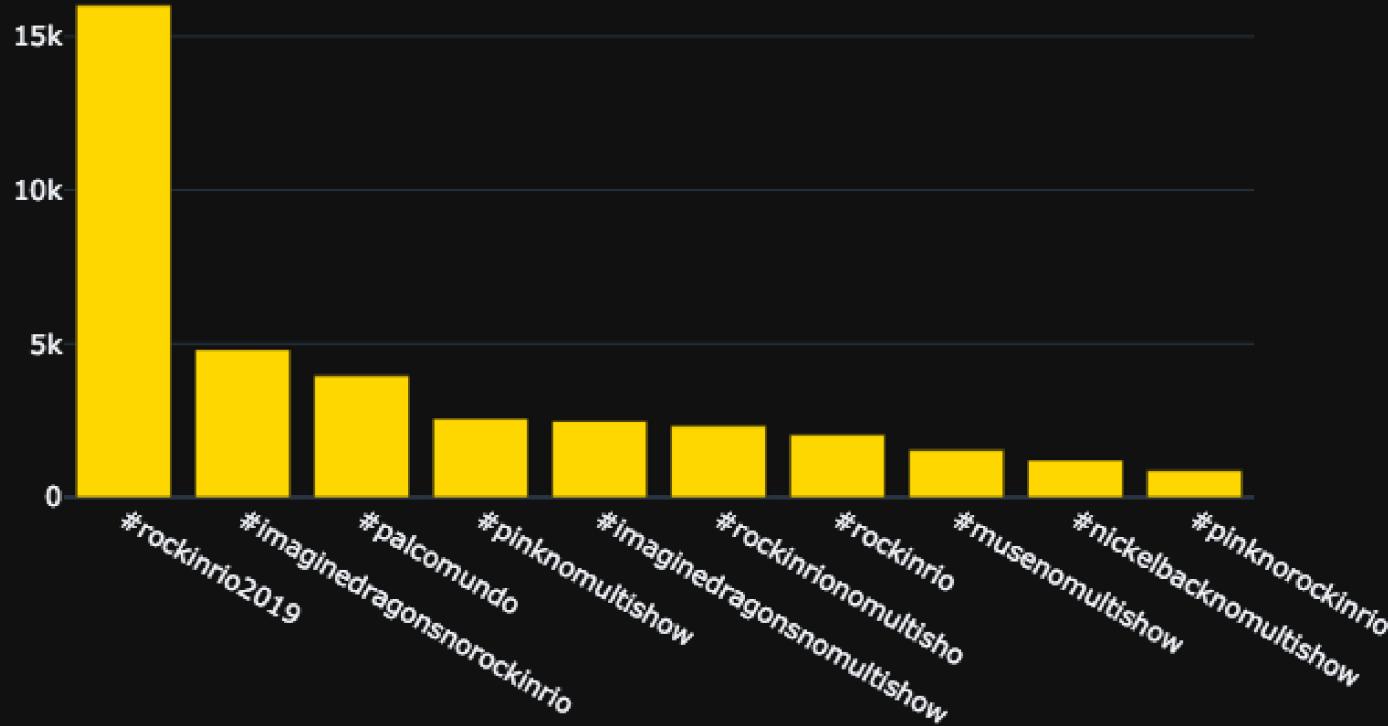
Tweets ao longo do período



Momentos

- 1 - Show Paralamas do Sucesso;
- 2 - Show Nickelback;
- 3 - Show Imagine Dragons;
- 4 - Show Muse;
- 5 - Fim do evento;

Tweets com sentimento positivo: 45206
 Tweets com sentimento neutro: 30644
 Tweets com sentimento negativo: 23477

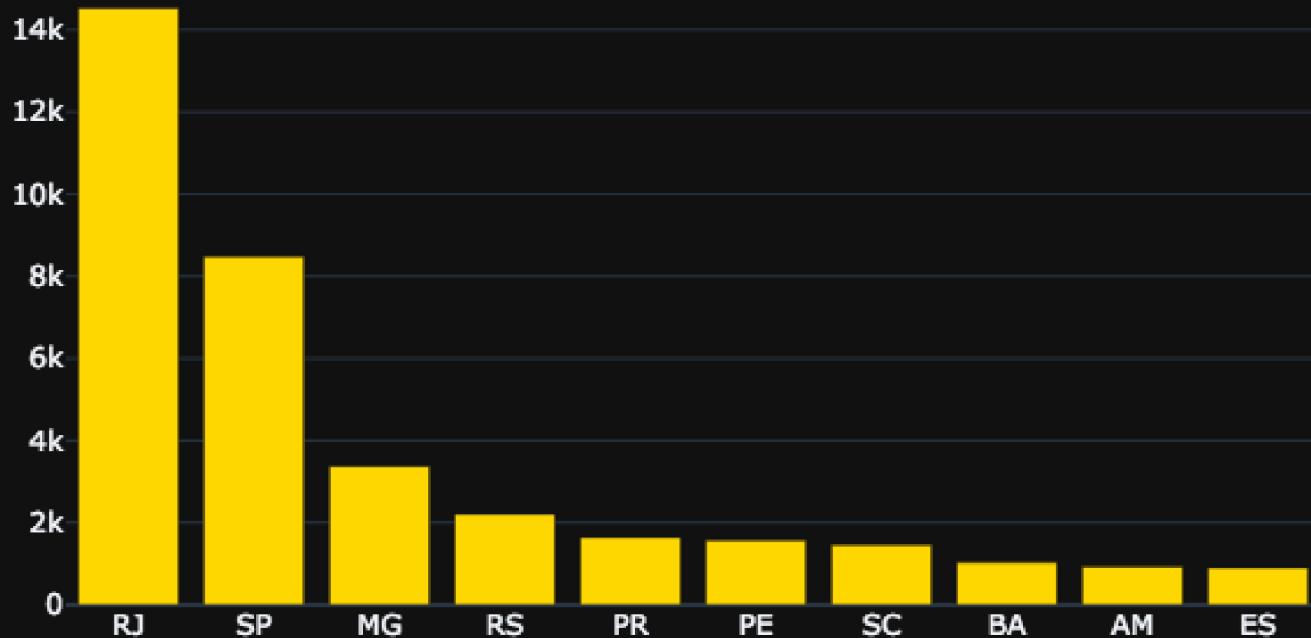


Top 10 Hashtags

- 1ºlugar: #rockinrio2019: 16016
- 2ºlugar: #Imaginedragonsnorockinrio: 4814
- 3ºlugar: #palcomundo: 3955
- 4ºlugar: #pinknomultishow: 2571
- 5ºlugar: #imaginedragonsnomultishow: 2474
- 6ºlugar: #rockinrionomultisho: 2323
- 7ºlugar: #rockinrio: 2037
- 8ºlugar: #museenomultishow: 1534
- 9ºlugar: #nickelbacknomultishow: 1214
- 10ºlugar: #pinknorockinrio: 881

Top 10 Palavras

- 1ºlugar: rock: 62000
- 2ºlugar: rio: 60836
- 3ºlugar: in: 60348
- 4ºlugar: show: 17284
- 5ºlugar: imagine: 7867
- 6ºlugar: dragons: 7701
- 7ºlugar: pink: 6550
- 8ºlugar: bolsonaro: 6451
- 9ºlugar: anitta: 6448
- 10ºlugar: gente: 6069



Top 10 estados

Tweets localizados: 43573

Percentual de tweets localizados: 43.87 %

- 1ºlugar: RJ: 14528
- 2ºlugar: SP: 8478
- 3ºlugar: MG: 3372
- 4ºlugar: RS: 2216
- 5ºlugar: PR: 1647
- 6ºlugar: PE: 1568
- 7ºlugar: SC: 1452
- 8ºlugar: BA: 1028
- 9ºlugar: AM: 935
- 10ºlugar: ES: 905



7. Links

Repositório do projeto:

https://github.com/DiegoAbreu/Projeto_Bem-te-vi_TCC_PUC-Minas

Vídeo explicativo:

<https://youtu.be/RP2IB-vTAOs>

REFERÊNCIAS

10 Key Marketing Trends for 2017 and Ideas for Exceeding Customer Expectations.

IBM, 2017. Disponível em:

<<https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf>>. Acesso em: 10 de out de 2019.

How much data is generated each day? **World Economic Forum**, 2019. Disponível em: <<https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/>>. Acesso em: 10 de out de 2019.

Digital 2019: Global Internet Use Accelerates. **We are Social**, 2019. Disponível em: <<https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates>>. Acesso em 10 de out de 2019

Digital 2019: Brazil. **Data Reportal**, 2019. Disponível em:

<<https://datareportal.com/reports/digital-2019-brazil>>. Acesso em 10 de out de 2019

Introducing New Messaging and Group Features for Businesses. **Facebook**, 2019. Disponível em: <<https://www.facebook.com/business/news/introducing-new-messaging-and-group-features-for-businesses>>. Acesso em 10 de out de 2019

Impulsionne com Facebook. **Facebook**, 2019. Disponível em:

<<https://www.facebook.com/boost/about>>. Acesso em 10 de out de 2019

A Copa do Mundo provocou mais de 19 milhões de menções nas redes sociais na América Latina. **Comscore**, 2019. Disponível em:

<<https://www.comscore.com/por/Insights/Blog/A-Copa-do-Mundo-provocou-mais-de-19-milhoes-de-mencoes-nas-redes-sociais-na-America-Latina>>. Acesso em 10 de out de 2019

Sonic: O Filme | Depois de críticas, diretor diz que visual do ouriço azul vai mudar.

Jovem Nerd, 2019. Disponível em: <<https://jovemnerd.com.br/nerdbunker/sonic-o-filme-depois-criticas-visual-ourico-azul-vai-mudar>>. Acesso em 10 de out de 2019

A PRIMAVERA ÁRABE E AS REDES SOCIAIS: Uso das redes sociais nas manifestações da Primavera Árabe nos países da Tunísia, Egito e Líbia. **Jaqueline Bartkowiak, Thatiane Fonseca, Gabriel Mattos e Vitor Henrique Souza**, 2017. Disponível em: <<https://www.maxwell.vrac.puc-rio.br/30432/30432.PDFXXvmi>>. Acesso em 10 de out de 2019

MeToo. **Google**, 2018. Disponível em: <<https://metoorising.withgoogle.com>>. Acesso em 10 de out de 2019

Vendaval Cambridge Analytica abala os EUA por fraudes com dados do Facebook.

El País Brasil, 2018. Disponível em:

<https://brasil.elpais.com/brasil/2018/03/20/internacional/1521574139_109464.html>. Acesso em 10 de out de 2019

Privacidade Hackeada. **Netflix**, 2019. Disponível em:
<<https://www.netflix.com/br/title/80117542>>. Acesso em 10 de out de 2019

Twitter developer home page. **Twitter**, 2019. Disponível em:
<<https://developer.twitter.com/>>. Acesso em 10 de out de 2019

Filter realtime Tweets. **Twitter**, 2019. Disponível em:
<<https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/connecting>>.
Acesso em 10 de out de 2019

Brazilian Cities. **Gilberto Trindade**, 2019. Disponível em:
<<https://www.kaggle.com/gilbertotrindade/cidades-brasileiras>>. Acesso em 10 de out de 2019

Portuguese Tweets for Sentiment Analysis. **Augustop**, 2019. Disponível em:
<<https://www.kaggle.com/augustop/portuguese-tweets-for-sentiment-analysis>>.
Acesso em 10 de out de 2019

iFeel - A Sentiment Analysis Framework. **DCC UFMG**, 2016. Disponível em:
<<http://blackbird.dcc.ufmg.br:1210>>. Acesso em 10 de out de 2019

SentiBench - a benchmark comparison of state-of-the-practice sentiment analysis methods. **Filipe Ribeiro, Matheus Araújo, Pollyanna Gonçalves, Marcos André Gonçalves e Fabrício Benevenuto**, 2016. Disponível em:
<<https://link.springer.com/article/10.1140/epjds/s13688-016-0085-1>>. Acesso em 10 de out de 2019