

Projeto 2 - NLP

Nome: **Alexandre Rozante**

Turma: **782**

Os segundo projeto do módulo de Machine Learning será focado no processamento de linguagem natural! Usaremos os algoritmos aprendidos e as técnicas vistas na segunda parte do curso para extraímos informações relevantes de texto. Mais precisamente, de publicações no Twitter.

Os Dados

Utilizaremos um Dataset obtido do Twitter com 100K postagens entre os dias 01/08/2018 e 20/10/2018. Cada postagem é classificada como **positiva**, **negativa** ou **neutra**.

Dois arquivos serão disponilizados para o desenvolvimento dos modelos, um para treino/validação e outro para submissão. Os arquivos se encontram na pasta `/Dados/train` e `/Dados/subm`, respectivamente.

Descrição das colunas:

- **id**: ID único para o tweet
- **tweet_text**: Texto da publicação no Twitter
- **tweet_date**: Data da publicação no Twitter
- **sentiment**: 0, se negativo; 1, se positivo; 2, se neutro
- **query_used**: Filtro utilizado para buscar a publicação

O Problema

Você deverá desenvolver um modelo para detectar o sentimento de uma publicação do Twitter a classificando em uma das três categorias: **positiva**, **negativa** ou **neutra**. O texto da publicação está disponível na coluna "tweet_text". Teste pelo menos 3 técnicas de NLP diferentes e escolha a métrica de avaliação que julgar mais pertinente.

Escolha o melhor modelo e gere uma base a partir dos dados de submissão, que estão no caminho `Dados/subm/Subm3Classes.csv`, com o seguinte formato:

id	sentiment_predict
12123232	0
323212	1
342235	2

Salve essa tabela como um arquivo csv com o nome `<nome>_<sobrenome>_nlp_degree.csv` e submeta-o como parte da entrega final do projeto.

Para ajudar no desenvolvimento, é possível dividir o projeto em algumas fases:

- **Análise de consistência dos dados:** analise se os dados estão fazendo sentido, se os campos estão completos e se há dados duplicados ou faltantes. Se julgar necessário, trate-os.
- **Análise exploratória:** analise a sua base como um todo, verifique o balanceamento entre as classes e foque, principalmente, na coluna `tweet_text`.
- **Pré-processamento e transformações:** projetos de NLP exigem um considerável pré-processamento. Foque no tratamento da string do texto. Procure começar com tratamentos simples e adicione complexidade gradualmente. Nessa etapa você testará diferentes técnicas de transformações, como o Bag Of Words e o TF-IDF.
- **Treinamento do modelo:** depois das transformações, você poderá executar o treinamento do modelo classificador. Nessa etapa o problema se torna semelhante aos abordados na primeira parte do módulo. Você pode testar diversos classificadores como RandomForest, AdaBoost, entre outros. Otimize os hiperparâmetros do modelo com técnicas como a GridSearch e a RandomizedSearch.
- **Conclusões:** descreva, em texto, as conclusões sobre os seus estudos. O modelo é capaz de identificar o sentimento das publicações? É possível extrapolar o modelo para outros contextos, como a análise de sentimento de uma frase qualquer? Pense em questões pertinentes e relevantes que você tenha obtido durante o desenvolvimento do projeto!

Critérios de avaliação

Os seguintes itens serão avaliados:

1. Desenvolvimento das etapas descritas acima;
1. Reprodutibilidade do código: seu código será executado e precisa gerar os mesmos resultados apresentados por você;
1. Clareza: seu código precisa ser claro e deve existir uma linha de raciocínio direta. Comente o código em pontos que julgar necessário para o entendimento total;
1. Justificativa das conclusões obtidas: não existirá certo ou errado, mas as decisões e as conclusões precisam ser bem justificadas com base nos resultados obtidos.

O desempenho do modelo **não** será considerado como critério de avaliação.

Informações gerais

- O projeto deve ser desenvolvido individualmente;
- Data de divulgação: 11/01/2022;
- Aula de monitoria: 19/01/2022;

- Data de entrega: 26/01/2022;
- Entrega através do Class: Árvore de Decisão -> Exercícios -> Projeto 2

Anexar, na entrega, o notebook de desenvolvimento e o arquivo .csv de submissão, da seguinte forma:

notebook: <nome>_<sobrenome>_<númeroTurma>_projeto_2.ipynb
csv: <nome>_<sobrenome>_<númeroTurma>_projeto_2_submissao.csv

Dicas

Base de treino e submissão

A base de submissão não possui a variável de saída, portanto ela será utilizada **apenas** para gerar o arquivo que acompanha a submissão do projeto.

Tente encontrar possíveis vieses

É muito comum que modelos de NLP possuam fortes vieses, como a tendência de relacionar palavras específicas com alguma classe de saída. Tente encontrar vieses no seu estudo, isso pode ajudar a tirar boas conclusões. o campo "query_used" pode ser útil para essa análise.

O pré-processamento é a chave para um bom desempenho

Essa é a etapa que mais vai contribuir para o desempenho do seu modelo. Seja criativo e desenvolva essa etapa de uma maneira que seja fácil de aplicar o mesmo processamento para uma nova base, você terá que fazer isso para gerar a base de submissão.

Um termômetro para o seu desenvolvimento

Após a correção do seu projeto, o professor irá disponibilizar a sua acurácia obtida na base de submissão. Você pode interpretar esse resultado como a simulação do resultado do seu modelo em produção. Uma diferença entre o resultado do estudo e o resultado de submissão indica um grau de **overfitting** no seu modelo.

Desenvolvimento do projeto

In [1]: # Carregamento do Dataframe de dados

```
import pandas as pd

df_twitter = pd.read_csv('dados\\train\\Train3Classes.csv')
df_twitter.head()
```

Out[1]:

	id	tweet_text	tweet_date	sentiment	query_used
0	1049721159292346368	Rio elege maior bancada policial de sua história...	Tue Oct 09 18:00:01 +0000 2018	2	folha
1	1046251157025423360	fiquei tão triste quando eu vi o preço da câmera...	Sun Sep 30 04:11:28 +0000 2018	0	:("
2	1041744620206653440	Para Theresa May, seu plano para o Brexit é a ...	Mon Sep 17 17:44:06 +0000 2018	2	exame
3	1046937084727107589	caralho eu quero proteger a danielly em um pot...	Tue Oct 02 01:37:06 +0000 2018	0	:("
4	1047326854229778432	@SiCaetano_ viva o caos :)	Wed Oct 03 03:25:55 +0000 2018	1	:)

In [2]:

df_twitter.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95000 entries, 0 to 94999
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   id          95000 non-null   int64  
 1   tweet_text  95000 non-null   object 
 2   tweet_date  95000 non-null   object 
 3   sentiment   95000 non-null   int64  
 4   query_used  95000 non-null   object 
dtypes: int64(2), object(3)
memory usage: 3.6+ MB
```

In [3]:

df_twitter.sentiment.value_counts()

Out[3]:

```
0    31696
1    31678
2    31626
Name: sentiment, dtype: int64
```

Nota: Numa primeira análise, não há dados faltantes e a base é balanceada.

Separação de Treino e Validação

Há uma base de submissão a ser processada com o modelo. Mesmo assim, separaremos uma parte como base de Testes aqui (10%).

In [4]:

from sklearn.model_selection import train_test_split

In [5]:

Adotada a divisão 20%/80% arbitrariamente
df_analise, df teste = train_test_split(df_twitter, test_size=0.15)

In [6]:

```
print('Treino:', df_analise.shape)
print('Teste:', df teste.shape)
```

```
Treino: (80750, 5)
Teste: (14250, 5)
```

```
In [7]: df_analise.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 80750 entries, 26477 to 33463
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          80750 non-null   int64  
 1   tweet_text  80750 non-null   object  
 2   tweet_date  80750 non-null   object  
 3   sentiment   80750 non-null   int64  
 4   query_used  80750 non-null   object  
dtypes: int64(2), object(3)
memory usage: 3.7+ MB
```

Tratamento dos dados:

Não necessitamos da coluna **id**, então removê-la economizará memória.

Convém tratar **tweet_date** como uma data e hora e não como texto.

Convém reduzir o tamanho da coluna **sentiment** pois os valores armazenados são apenas 0, 1 ou 2.

```
In [8]: # Um pequeno artifício para evitar erros por falhas de digitação:
true  = True
false = False
```

```
'''
Versão inicial da função de tratamento dos dados
'''

def tratamento(df):
    df = df.copy()
    df = df.drop('id', axis=1)
    df['tweet_date'] = pd.to_datetime(df['tweet_date'])
    df['sentiment'] = pd.to_numeric(df['sentiment'], downcast='integer')
    return df
```

```
In [10]: df_analise = tratamento(df_analise)
df_analise.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 80750 entries, 26477 to 33463
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   tweet_text    80750 non-null   object  
 1   tweet_date    80750 non-null   datetime64[ns, UTC]
 2   sentiment     80750 non-null   int8    
 3   query_used    80750 non-null   object  
dtypes: datetime64[ns, UTC](1), int8(1), object(2)
memory usage: 2.5+ MB
```

Nota: Reduzimos a base em ~1mb (30%). A remoção do Id nos permite agora avaliar duplicidades.

```
In [11]: tam_anterior = len(df_analise)
df_analise = df_analise.drop_duplicates()
print(tam_anterior - len(df_analise), 'duplicidades removidas.')

9 duplicidades removidas.
```

```
In [12]: df_analise['query_used'].value_counts()
```

```
Out[12]: :(
:(
folha          27022
:)
estadao        26927
#fato          4214
jornaloglobo   3295
g1              2938
exame          2913
#trabalho      2894
#oportunidade  2597
#noticia        2110
veja            1991
#novidade       1788
#curiosidade   939
#noticia        793
#curiosidade   320
Name: query_used, dtype: int64
```

Nota: Sobre dados faltantes

Não há dados faltantes, mas convém definir um tratamento:

tweet_text: Se não estiver preenchido, o registro deve ser removido. Não faz sentido analisar isso.

tweet_date: Em princípio não é relevante para o modelo. Não será tratada.

sentiment: É a variável-alvo do modelo, então, o modelo em si não a utilizará. Não será tratada.

query_used: É a fonte da informação. Não será tratada.

```
In [13]: tam_anterior = len(df_analise)
df_analise.dropna(inplace=True, subset=['tweet_text'])
```

```
print(tam_anterior - len(df_analise), 'registros eliminados.')
```

0 registros eliminados.

In [14]:

```
'''  
Nova versão da função de tratamento dos dados  
'''  
  
def tratamento(df):  
    df = df.copy()  
    df = df.drop('id', axis=1)  
    df['tweet_date'] = pd.to_datetime(df['tweet_date'])  
    df['sentiment'] = pd.to_numeric(df['sentiment'], downcast='integer')  
    df.dropna(inplace=True, subset=['tweet_text'])  
    return df
```

Análise Exploratória

Vamos analisar as palavras pequenas (1 e 2 caracteres apenas)

In [15]:

```
# Armazenará as palavras  
pequenas = []  
  
'''  
Função para preencher a lista de pequenas palavras sem intervir do dado de entrada  
'''  
  
def extrai_palavras(texto):  
    for palavra in texto.split(' '):  
        if len(palavra) < 3:  
            pequenas.append(palavra)  
    return texto
```

In [16]:

```
# Cria a lista de palavras  
df_analise['tweet_text'].apply(extrai_palavras)
```

Out[16]:

```
26477 Gostei de um vídeo @YouTube https://t.co/TBrLM...  
9195 Mais uma luta, mais uma Vitória. 🤟 #Capitão201...  
50030 Candidatos intensificam campanha nas redes soc...  
66552 eu fiz dois desenhos hj :D mas não sei se posto.  
5862 @peekabruwu desculpa por falar tanto disso ami...  
      ...  
52428 @Kizat_Kdust @springwdayy Ela falava do relaci...  
4909 Mara Gabrilli rebate Doria e diz que vota em A...  
23688 &gt;@brdezoito Bolsonaro repete slogan de Coll...  
64356 será q um dia vou conseguir beijar algum dos m...  
33463 E que comece o feriadão! 🎉 #Work #WorKING #Job...  
Name: tweet_text, Length: 80741, dtype: object
```

In [17]:

```
# Transforma num DataFrame, remove duplicidades e ordena
```

```
pd_pequenas = pd.DataFrame(pequenas, columns=['palavra'])
pd_pequenas.drop_duplicates(inplace=True)
pd_pequenas.sort_values(by='palavra', inplace=True)
pequenas = pd_pequenas['palavra'].to_list()
```

```
In [18]: pd_pequenas = None # Não é mais necessário
```

```
'''
Função para exibir lista de pequenas palavras de forma concisa
'''
def exibir_pequenas(lista):
    col = 1
    idx = 0
    for palavra in lista:
        p = palavra + ' '
        print('{:04}=[{}]'.format(idx, p[0:2]), end=' ')
        idx += 1
        col += 7
        if col >= 70:
            print()
            col = 1
```

```
In [20]: exibir_pequenas(pequenas)
```

0000=[!] 0001=[!!] 0002=[!?] 0003=[!] 0004=[!😊] 0005=["] 0006=[-] 0007=[0] 0008=["5] 0009=[A]
0010=[E] 0011=[I] 0012=[N] 0013=[O] 0014=[Y] 0015=[a] 0016=[e] 0017=[n] 0018=[o] 0019=[q]
0020=[A] 0021=[É] 0022=[é] 0023=[ó] 0024=[😊] 0025=[#] 0026=[#1] 0027=[#2] 0028=[#3] 0029=[#7]
0030=[#8] 0031=[#a] 0032=[#e] 0033=[#o] 0034=[#é] 0035=[#맘] 0036=[#🤖] 0037=[#😊] 0038=[#😊] 0039=[#👉]
0040=[#Χ] 0041=[\\$] 0042=[\$\$] 0043=[\$,] 0044=[\$2] 0045=[\$5] 0046=[%] 0047=[%;] 0048=['] 0049=['']
0050=[',] 0051=['7] 0052=['A] 0053=['I] 0054=['0] 0055=['a] 0056=['e] 0057=['o] 0058=['É] 0059=['à]
0060=['é] 0061=[(] 0062=[((] 0063=[(,) 0064=[(.) 0065=[(/] 0066=[(1] 0067=[(3] 0068=[(7] 0069=[(:]
0070=[(@] 0071=[(A] 0072=[(E] 0073=[(O] 0074=[(Q] 0075=[(a] 0076=[(b] 0077=[(e] 0078=[(n] 0079=[(o]
0080=[(q] 0081=[(É] 0082=[(é] 0083=[(👉] 0084=[() 0085=[!) 0086=[)))] 0087=[(),] 0088=[().] 0089=[():]
0090=[();] 0091=[()👉] 0092=[*] 0093=[**] 0094=[*-] 0095=[*A] 0096=[*s] 0097=[+] 0098=[++] 0099=[+,]
0100=[+.] 0101=[+1] 0102=[+5] 0103=[+9] 0104=[+:] 0105=[+o] 0106=[,,] 0107=[,,?] 0108=[,,?] 0109=[,,e]
0110=[,n] 0111=[,o] 0112=[,...] 0113=[-] 0114=[-]) 0115=[--] 0116=[-1] 0117=[-5] 0118=[-:] 0119=[-B]
0120=[-e] 0121=[-f] 0122=[-p] 0123=[-q] 0124=[-👉] 0125=[-😊] 0126=[.] 0127=[.!] 0128=[..] 0129=[.g]
0130=[.+] 0131=[/] 0132=[//] 0133=[/:] 0134=[/p] 0135=[0] 0136=[0%] 0137=[0)] 0138=[0,,] 0139=[0,.]
0140=[0/] 0141=[01] 0142=[02] 0143=[03] 0144=[04] 0145=[05] 0146=[06] 0147=[07] 0148=[08] 0149=[0€]
0150=[1] 0151=[1%] 0152=[1)] 0153=[1,] 0154=[1-] 0155=[1.] 0156=[10] 0157=[11] 0158=[12] 0159=[13]
0160=[14] 0161=[15] 0162=[16] 0163=[17] 0164=[18] 0165=[19] 0166=[1:] 0167=[1?] 0168=[1K] 0169=[1M]
0170=[1T] 0171=[1a] 0172=[1d] 0173=[1h] 0174=[1m] 0175=[1o] 0176=[1p] 0177=[1s] 0178=[1x] 0179=[1ª]
0180=[1°] 0181=[1°] 0182=[1•] 0183=[1€] 0184=[2] 0185=[2!] 0186=[2%] 0187=[2'] 0188=[2)] 0189=[2,,]
0190=[2-] 0191=[2.] 0192=[20] 0193=[21] 0194=[22] 0195=[23] 0196=[24] 0197=[25] 0198=[26] 0199=[27]
0200=[28] 0201=[29] 0202=[2:] 0203=[2?] 0204=[2L] 0205=[2T] 0206=[2a] 0207=[2d] 0208=[2h] 0209=[2k]
0210=[2o] 0211=[2t] 0212=[2x] 0213=[2ª] 0214=[2°] 0215=[2º] 0216=[2•] 0217=[2€] 0218=[3] 0219=[3!]
0220=[3%] 0221=[3)] 0222=[3,] 0223=[3-] 0224=[3.] 0225=[30] 0226=[31] 0227=[32] 0228=[33] 0229=[34]
0230=[35] 0231=[36] 0232=[37] 0233=[38] 0234=[39] 0235=[3:] 0236=[3;] 0237=[3?] 0238=[3D] 0239=[3G]
0240=[3U] 0241=[3a] 0242=[3g] 0243=[3h] 0244=[3k] 0245=[3x] 0246=[3ª] 0247=[3°] 0248=[3º] 0249=[3']
0250=[3..] 0251=[4] 0252=[4!] 0253=[4%] 0254=[4'] 0255=[4)] 0256=[4+] 0257=[4,,] 0258=[4-] 0259=[4.]
0260=[40] 0261=[41] 0262=[42] 0263=[43] 0264=[44] 0265=[45] 0266=[46] 0267=[47] 0268=[48] 0269=[49]
0270=[4;] 0271=[4?] 0272=[4G] 0273=[4M] 0274=[4a] 0275=[4h] 0276=[4k] 0277=[4x] 0278=[4ª] 0279=[4°]
0280=[4º] 0281=[4'] 0282=[4€] 0283=[5] 0284=[5!] 0285=[5\$] 0286=[5%] 0287=[5'] 0288=[5)] 0289=[5+]
0290=[5,,] 0291=[5.] 0292=[50] 0293=[51] 0294=[52] 0295=[53] 0296=[54] 0297=[55] 0298=[56] 0299=[57]
0300=[58] 0301=[59] 0302=[5:] 0303=[5?] 0304=[5G] 0305=[5a] 0306=[5e] 0307=[5h] 0308=[5k] 0309=[5x]
0310=[5ª] 0311=[5°] 0312=[5º] 0313=[5€] 0314=[6] 0315=[6!] 0316=[6%] 0317=[6,] 0318=[6-] 0319=[6.]
0320=[60] 0321=[61] 0322=[62] 0323=[63] 0324=[64] 0325=[65] 0326=[66] 0327=[67] 0328=[68] 0329=[69]
0330=[6E] 0331=[6S] 0332=[6b] 0333=[6h] 0334=[6m] 0335=[6ª] 0336=[6º] 0337=[7] 0338=[7%] 0339=[7,,]
0340=[7-] 0341=[7.] 0342=[70] 0343=[71] 0344=[72] 0345=[73] 0346=[74] 0347=[75] 0348=[76] 0349=[77]
0350=[78] 0351=[79] 0352=[7a] 0353=[7h] 0354=[7k] 0355=[7m] 0356=[7x] 0357=[7ª] 0358=[7º] 0359=[7º]
0360=[7"] 0361=[8] 0362=[8\$] 0363=[8%] 0364=[8,,] 0365=[8-] 0366=[8.] 0367=[80] 0368=[81] 0369=[82]
0370=[83] 0371=[84] 0372=[85] 0373=[86] 0374=[87] 0375=[88] 0376=[89] 0377=[8h] 0378=[8k] 0379=[8ª]
0380=[8°] 0381=[8°] 0382=[8°] 0383=[9] 0384=[9!] 0385=[9%] 0386=[9)] 0387=[9,,] 0388=[9-] 0389=[9,.]
0390=[90] 0391=[91] 0392=[92] 0393=[93] 0394=[94] 0395=[95] 0396=[96] 0397=[97] 0398=[98] 0399=[99]
0400=[9;;] 0401=[9h] 0402=[9ª] 0403=[9º] 0404=[9º] 0405=[::] 0406=[:#] 0407=[:(;) 0408=[:]]) 0409=[*:]
0410=[:-] 0411=[:/] 0412=[:0] 0413=[:3] 0414=[::] 0415=[:C] 0416=[:D] 0417=[:0] 0418=[:P] 0419=[:T]
0420=[:V] 0421=[:\`] 0422=[:]]) 0423=[:c] 0424=[:o] 0425=[:p] 0426=[:u] 0427=[:v] 0428=[:x] 0429=[:]])
0430=[:]]) 0431=[:]]) 0432=[;] 0433=[;\$] 0434=[;();] 0435=[;();] 0436=[;*] 0437=[;-] 0438=[;3] 0439=[;;]
0440=[;D] 0441=[;0] 0442=[;P] 0443=[;();] 0444=[;p] 0445=[;x] 0446=[;~] 0447=[=] 0448=[=(;) 0449=[=)]
0450=[=/] 0451=[=D] 0452=[=0] 0453=[=T] 0454=[=]) 0455=[?] 0456=[?!] 0457=[?"] 0458=[?)] 0459=[??]
0460=[?😊] 0461=[@] 0462=[@,] 0463=[@,] 0464=[@?] 0465=[@g] 0466=[@s] 0467=[@...] 0468=[A] 0469=[A)]
0470=[A,,] 0471=[A..] 0472=[A2] 0473=[A9] 0474=[AA] 0475=[AB] 0476=[AC] 0477=[AE] 0478=[AF] 0479=[AH]
0480=[AI] 0481=[AK] 0482=[AL] 0483=[AM] 0484=[AN] 0485=[AO] 0486=[AP] 0487=[AQ] 0488=[AR] 0489=[AS]
0490=[AT] 0491=[AU] 0492=[Aa] 0493=[Ab] 0494=[Ae] 0495=[Af] 0496=[Ah] 0497=[Ai] 0498=[A1] 0499=[Am]
0500=[An] 0501=[Ao] 0502=[Aq] 0503=[Ar] 0504=[As] 0505=[Aw] 0506=[Az] 0507=[AÍ] 0508=[AÍ] 0509=[B]
0510=[B!] 0511=[B'] 0512=[B,,] 0513=[B-] 0514=[B,] 0515=[B1] 0516=[B3] 0517=[BA] 0518=[BB] 0519=[BC]

0520=[BD] 0521=[BE] 0522=[BF] 0523=[BG] 0524=[BH] 0525=[BI] 0526=[BK] 0527=[BL] 0528=[BM] 0529=[BN]
0530=[BP] 0531=[BR] 0532=[BS] 0533=[BT] 0534=[BU] 0535=[Ba] 0536=[Be] 0537=[Bi] 0538=[Bj] 0539=[Bo]
0540=[Br] 0541=[By] 0542=[C] 0543=[C#] 0544=[C*] 0545=[C,] 0546=[C.] 0547=[C9] 0548=[C?] 0549=[CA]
0550=[CB] 0551=[CC] 0552=[CD] 0553=[CE] 0554=[CL] 0555=[CM] 0556=[CP] 0557=[CS] 0558=[CT] 0559=[CU]
0560=[CV] 0561=[CZ] 0562=[Ca] 0563=[Cc] 0564=[Ce] 0565=[Cm] 0566=[Ct] 0567=[CÁ] 0568=[CÚ] 0569=[Cá]
0570=[Cê] 0571=[Cú] 0572=[C”] 0573=[D] 0574=[D,] 0575=[D.] 0576=[D2] 0577=[D9] 0578=[D:] 0579=[DA]
0580=[DC] 0581=[DE] 0582=[DF] 0583=[DJ] 0584=[DM] 0585=[DO] 0586=[DR] 0587=[DS] 0588=[DT] 0589=[DU]
0590=[DV] 0591=[DW] 0592=[Da] 0593=[Dc] 0594=[De] 0595=[Di] 0596=[Dj] 0597=[Dm] 0598=[Do] 0599=[Dr]
0600=[Du] 0601=[DÁ] 0602=[DÓ] 0603=[Dá] 0604=[Dê] 0605=[Dó] 0606=[E] 0607=[E'] 0608=[E,] 0609=[E3]
0610=[EA] 0611=[EC] 0612=[EF] 0613=[EH] 0614=[EI] 0615=[EJ] 0616=[EL] 0617=[EM] 0618=[EN] 0619=[EP]
0620=[ES] 0621=[ET] 0622=[EU] 0623=[EX] 0624=[Ec] 0625=[Ed] 0626=[Ee] 0627=[Ef] 0628=[Eh] 0629=[Ei]
0630=[E1] 0631=[Em] 0632=[En] 0633=[Es] 0634=[Eu] 0635=[Ex] 0636=[Ez] 0637=[E...] 0638=[F] 0639=[F"]
0640=[F.] 0641=[F1] 0642=[F3] 0643=[F5] 0644=[FB] 0645=[FC] 0646=[FF] 0647=[FH] 0648=[FL] 0649=[FM]
0650=[FS] 0651=[FT] 0652=[FX] 0653=[Fg] 0654=[FÉ] 0655=[Fã] 0656=[Fé] 0657=[Fê] 0658=[G] 0659=[G']
0660=[G-] 0661=[G.] 0662=[G1] 0663=[G2] 0664=[G4] 0665=[GB] 0666=[GE] 0667=[GG] 0668=[GJ] 0669=[GM]
0670=[GO] 0671=[GP] 0672=[GS] 0673=[GT] 0674=[GW] 0675=[Gn] 0676=[Go] 0677=[Gu] 0678=[H] 0679=[H.]
0680=[HA] 0681=[HB] 0682=[HC] 0683=[HD] 0684=[HG] 0685=[HJ] 0686=[HM] 0687=[HP] 0688=[HQ] 0689=[HR]
0690=[Ha] 0691=[He] 0692=[Hi] 0693=[Hj] 0694=[Hl] 0695=[Hm] 0696=[Hn] 0697=[Hs] 0698=[Há] 0699=[I]
0700=[I"] 0701=[I,] 0702=[I.] 0703=[I5] 0704=[IA] 0705=[IC] 0706=[ID] 0707=[IF] 0708=[IG] 0709=[IH]
0710=[II] 0711=[IM] 0712=[IN] 0713=[IR] 0714=[IS] 0715=[IT] 0716=[IU] 0717=[IV] 0718=[IX] 0719=[Ia]
0720=[Ig] 0721=[Ih] 0722=[Ii] 0723=[Im] 0724=[In] 0725=[Ir] 0726=[Is] 0727=[It] 0728=[Iã] 0729=[J]
0730=[J.] 0731=[JA] 0732=[JB] 0733=[JF] 0734=[JG] 0735=[JK] 0736=[JM] 0737=[JN] 0738=[JP] 0739=[JR]
0740=[Ja] 0741=[Ji] 0742=[Jk] 0743=[Jo] 0744=[Jp] 0745=[Jr] 0746=[Ju] 0747=[JÁ] 0748=[Jà] 0749=[Já]
0750=[Jã] 0751=[Jô] 0752=[Jú] 0753=[K] 0754=[K.] 0755=[K9] 0756=[KD] 0757=[KI] 0758=[KK] 0759=[KW]
0760=[Kd] 0761=[Ke] 0762=[Kg] 0763=[Ki] 0764=[Kk] 0765=[Km] 0766=[Ko] 0767=[Ku] 0768=[K"] 0769=[L]
0770=[L.] 0771=[L?] 0772=[LA] 0773=[LC] 0774=[LE] 0775=[LG] 0776=[LI] 0777=[LP] 0778=[LS] 0779=[LU]
0780=[LW] 0781=[La] 0782=[Le] 0783=[Li] 0784=[Lo] 0785=[Lu] 0786=[Lx] 0787=[Ly] 0788=[LÁ] 0789=[Lá]
0790=[Lê] 0791=[M] 0792=[M,] 0793=[MA] 0794=[MC] 0795=[ME] 0796=[MG] 0797=[MI] 0798=[MJ] 0799=[MK]
0800=[MP] 0801=[MS] 0802=[MT] 0803=[MV] 0804=[MY] 0805=[Ma] 0806=[Mc] 0807=[Me] 0808=[Mi] 0809=[Mo]
0810=[Mr] 0811=[Mt] 0812=[Mv] 0813=[My] 0814=[Mã] 0815=[MÁ] 0816=[Má] 0817=[Mó] 0818=[N] 0819=[N,]
0820=[N.] 0821=[NA] 0822=[ND] 0823=[NE] 0824=[NI] 0825=[NO] 0826=[NR] 0827=[NT] 0828=[NX] 0829=[NY]
0830=[Na] 0831=[Nc] 0832=[Nd] 0833=[Nf] 0834=[Nn] 0835=[No] 0836=[Nr] 0837=[Ns] 0838=[Nu] 0839=[Nw]
0840=[Nº] 0841=[NÉ] 0842=[Né] 0843=[Nó] 0844=[O] 0845=[O'] 0846=[O,] 0847=[OC] 0848=[OF] 0849=[OH]
0850=[OI] 0851=[OJ] 0852=[OK] 0853=[ON] 0854=[OQ] 0855=[OR] 0856=[OS] 0857=[OU] 0858=[Oc] 0859=[Of]
0860=[Oh] 0861=[Oi] 0862=[Ok] 0863=[On] 0864=[Oo] 0865=[Oq] 0866=[Or] 0867=[Os] 0868=[Ou] 0869=[Ow]
0870=[...] 0871=[P] 0872=[P.] 0873=[P/] 0874=[P1] 0875=[PA] 0876=[PB] 0877=[PC] 0878=[PE] 0879=[PF]
0880=[PG] 0881=[PL] 0882=[PM] 0883=[PN] 0884=[PP] 0885=[PQ] 0886=[PR] 0887=[PS] 0888=[PT] 0889=[PV]
0890=[Pa] 0891=[Pc] 0892=[Pf] 0893=[Po] 0894=[Pq] 0895=[Ps] 0896=[Pt] 0897=[Pv] 0898=[PÉ] 0899=[Pá]
0900=[Pé] 0901=[Pó] 0902=[Pô] 0903=[Q] 0904=[Q+] 0905=[QB] 0906=[QG] 0907=[QI] 0908=[QQ] 0909=[QR]
0910=[Qd] 0911=[Qm] 0912=[Qq] 0913=[Qr] 0914=[R] 0915=[R#] 0916=[R\$] 0917=[R.] 0918=[R6] 0919=[RB]
0920=[RC] 0921=[RE] 0922=[RG] 0923=[RH] 0924=[RJ] 0925=[RL] 0926=[RM] 0927=[RN] 0928=[RO] 0929=[RP]
0930=[RR] 0931=[RS] 0932=[RT] 0933=[RU] 0934=[RV] 0935=[Rh] 0936=[Rs] 0937=[Rt] 0938=[Ré] 0939=[S]
0940=[S.] 0941=[S2] 0942=[SA] 0943=[SC] 0944=[SE] 0945=[SH] 0946=[SI] 0947=[SL] 0948=[SM] 0949=[SO]
0950=[SP] 0951=[SS] 0952=[SU] 0953=[SV] 0954=[Sa] 0955=[Se] 0956=[Si] 0957=[So] 0958=[Sp] 0959=[Sr]
0960=[St] 0961=[Sò] 0962=[SÓ] 0963=[Sá] 0964=[Sé] 0965=[Só] 0966=[T] 0967=[T,] 0968=[T1] 0969=[T3]
0970=[TA] 0971=[TB] 0972=[TD] 0973=[TE] 0974=[TF] 0975=[TG] 0976=[TH] 0977=[TI] 0978=[TJ] 0979=[TL]
0980=[TO] 0981=[TR] 0982=[TT] 0983=[TU] 0984=[TV] 0985=[TW] 0986=[TY] 0987=[Ta] 0988=[Tb] 0989=[Td]
0990=[Te] 0991=[Ti] 0992=[To] 0993=[Tp] 0994=[Tt] 0995=[Tu] 0996=[Tv] 0997=[Ty] 0998=[TÁ] 0999=[TÔ]
1000=[Tá] 1001=[Tó] 1002=[Tô] 1003=[T...] 1004=[U] 1005=[U\$] 1006=[U,] 1007=[U2] 1008=[UB] 1009=[UE]
1010=[UF] 1011=[UI] 1012=[UK] 1013=[UM] 1014=[UP] 1015=[US] 1016=[UT] 1017=[UX] 1018=[Ua] 1019=[Ue]
1020=[Uh] 1021=[Ui] 1022=[Um] 1023=[Un] 1024=[Up] 1025=[Uq] 1026=[Us] 1027=[Ué] 1028=[V] 1029=[V,]
1030=[V.] 1031=[VA] 1032=[VC] 1033=[VE] 1034=[VI] 1035=[VO] 1036=[VR] 1037=[VS] 1038=[VT] 1039=[VW]

1040=[Va] 1041=[Vc] 1042=[Ve] 1043=[Vi] 1044=[Vo] 1045=[VÁ] 1046=[VÊ] 1047=[Vá] 1048=[Vé] 1049=[Vê]
1050=[Ví] 1051=[Vó] 1052=[Vô] 1053=[W] 1054=[WC] 1055=[WD] 1056=[WM] 1057=[WP] 1058=[WT] 1059=[WU]
1060=[WW] 1061=[We] 1062=[Ws] 1063=[Wu] 1064=[X] 1065=[X'] 1066=[X,] 1067=[X.] 1068=[XA] 1069=[XD]
1070=[XI] 1071=[XP] 1072=[XS] 1073=[XX] 1074=[XY] 1075=[Xs] 1076=[Xô] 1077=[YG] 1078=[YT] 1079=[Ya]
1080=[Yi] 1081=[Yo] 1082=[Yê] 1083=[Z] 1084=[ZL] 1085=[ZN] 1086=[Ze] 1087=[Zs] 1088=[Zé]
1090=[[] 1091=[;] 1092=[[]] 1093=[[a]] 1094=[\] 1095=[\o] 1096=[\õ] 1097=[[]] 1098=[[,]] 1099=[^]
1100=[^ .] 1101=[^ ^] 1102=[_ .] 1103=[_.] 1104=[a] 1105=[a)] 1106=[a,] 1107=[a.] 1108=[aa] 1109=[ab]
1110=[ad] 1111=[ae] 1112=[af] 1113=[ah] 1114=[ai] 1115=[ak] 1116=[al] 1117=[am] 1118=[an] 1119=[ao]
1120=[ap] 1121=[aq] 1122=[ar] 1123=[as] 1124=[at] 1125=[au] 1126=[av] 1127=[aw] 1128=[ay] 1129=[aí]
1130=[aí] 1131=[a...] 1132=[b] 1133=[b)] 1134=[b*] 1135=[b,] 1136=[b?] 1137=[ba] 1138=[bb] 1139=[bc]
1140=[bd] 1141=[be] 1142=[bf] 1143=[bg] 1144=[bh] 1145=[bi] 1146=[bj] 1147=[bk] 1148=[bo] 1149=[bp]
1150=[br] 1151=[bs] 1152=[bu] 1153=[bv] 1154=[by] 1155=[bê] 1156=[c] 1157=[c#] 1158=[c)] 1159=[c*]
1160=[c,] 1161=[c/] 1162=[c2] 1163=[c9] 1164=[c:] 1165=[ca] 1166=[cb] 1167=[cc] 1168=[cd] 1169=[ce]
1170=[cf] 1171=[cg] 1172=[ci] 1173=[cm] 1174=[co] 1175=[cp] 1176=[cr] 1177=[cs] 1178=[cu] 1179=[cv]
1180=[cy] 1181=[cá] 1182=[cê] 1183=[cú] 1184=[d] 1185=[d*] 1186=[d+] 1187=[dA] 1188=[dE] 1189=[da]
1190=[dc] 1191=[dd] 1192=[de] 1193=[dg] 1194=[di] 1195=[dj] 1196=[dk] 1197=[dm] 1198=[do] 1199=[dq]
1200=[dr] 1201=[dt] 1202=[du] 1203=[dx] 1204=[dá] 1205=[dê] 1206=[dó] 1207=[e] 1208=[e'] 1209=[e,]
1210=[e.] 1211=[e:] 1212=[eU] 1213=[ea] 1214=[ed] 1215=[ef] 1216=[eh] 1217=[ei] 1218=[el] 1219=[em]
1220=[en] 1221=[eo] 1222=[ep] 1223=[eq] 1224=[er] 1225=[es] 1226=[et] 1227=[eu] 1228=[ev] 1229=[ew]
1230=[ex] 1231=[ey] 1232=[ez] 1233=[eñ] 1234=[e...] 1235=[f] 1236=[f*] 1237=[f1] 1238=[f3] 1239=[fa]
1240=[fb] 1241=[fc] 1242=[fd] 1243=[fe] 1244=[ff] 1245=[fi] 1246=[fk] 1247=[fl] 1248=[fp] 1249=[fq]
1250=[ft] 1251=[fu] 1252=[fx] 1253=[fá] 1254=[fâ] 1255=[fâ] 1256=[fê] 1257=[g] 1258=[g!] 1259=[g1]
1260=[g4] 1261=[ga] 1262=[gc] 1263=[gd] 1264=[gg] 1265=[gi] 1266=[gl] 1267=[gn] 1268=[go] 1269=[gt]
1270=[gu] 1271=[gv] 1272=[gy] 1273=[h] 1274=[h,] 1275=[h.] 1276=[ha] 1277=[hb] 1278=[hc] 1279=[hd]
1280=[he] 1281=[hi] 1282=[hj] 1283=[hm] 1284=[ho] 1285=[hp] 1286=[hq] 1287=[hr] 1288=[hs] 1289=[há]
1290=[hê] 1291=[i] 1292=[iM] 1293=[ia] 1294=[ic] 1295=[id] 1296=[if] 1297=[ig] 1298=[ih] 1299=[im]
1300=[in] 1301=[io] 1302=[ip] 1303=[iq] 1304=[ir] 1305=[is] 1306=[it] 1307=[iz] 1308=[j] 1309=[j7]
1310=[ja] 1311=[jb] 1312=[je] 1313=[jg] 1314=[ji] 1315=[jk] 1316=[jm] 1317=[jo] 1318=[jp] 1319=[jr]
1320=[ju] 1321=[já] 1322=[jê] 1323=[jó] 1324=[jô] 1325=[k] 1326=[k)] 1327=[ka] 1328=[kd] 1329=[kg]
1330=[ki] 1331=[kk] 1332=[kl] 1333=[km] 1334=[ko] 1335=[ku] 1336=[l] 1337=[la] 1338=[le] 1339=[lg]
1340=[li] 1341=[l1] 1342=[lo] 1343=[lu] 1344=[ly] 1345=[lz] 1346=[lá] 1347=[lã] 1348=[lê] 1349=[lí]
1350=[m] 1351=[m,] 1352=[m9] 1353=[ma] 1354=[mb] 1355=[mc] 1356=[md] 1357=[me] 1358=[mh] 1359=[mi]
1360=[ml] 1361=[mm] 1362=[mn] 1363=[mo] 1364=[mp] 1365=[mr] 1366=[ms] 1367=[mt] 1368=[mv] 1369=[mw]
1370=[mx] 1371=[my] 1372=[m²] 1373=[má] 1374=[mó] 1375=[mô] 1376=[n] 1377=[n!] 1378=[n'] 1379=[n,]
1380=[n1] 1381=[n?] 1382=[nA] 1383=[na] 1384=[nc] 1385=[nd] 1386=[ne] 1387=[nf] 1388=[ng] 1389=[ni]
1390=[n1] 1391=[nm] 1392=[nn] 1393=[no] 1394=[np] 1395=[nq] 1396=[ns] 1397=[nt] 1398=[nu] 1399=[nv]
1400=[nx] 1401=[ny] 1402=[n°] 1403=[nº] 1404=[ná] 1405=[né] 1406=[nó] 1407=[nô] 1408=[n'] 1409=[o]
1410=[o!] 1411=[o.] 1412=[o/] 1413=[o:] 1414=[o?] 1415=[oO] 1416=[oS] 1417=[oa] 1418=[ob] 1419=[oc]
1420=[of] 1421=[oh] 1422=[oi] 1423=[ok] 1424=[om] 1425=[on] 1426=[oo] 1427=[op] 1428=[oq] 1429=[or]
1430=[os] 1431=[ou] 1432=[ow] 1433=[o...] 1434=[p] 1435=[p/] 1436=[p1] 1437=[p2] 1438=[pQ] 1439=[pa]
1440=[pc] 1441=[pd] 1442=[pe] 1443=[pf] 1444=[pg] 1445=[ph] 1446=[pj] 1447=[pk] 1448=[pl] 1449=[pm]
1450=[pn] 1451=[po] 1452=[pp] 1453=[pq] 1454=[pr] 1455=[ps] 1456=[pt] 1457=[pv] 1458=[pw] 1459=[px]
1460=[pá] 1461=[pé] 1462=[pê] 1463=[pó] 1464=[pô] 1465=[q] 1466=[q)] 1467=[q,] 1468=[q?] 1469=[qa]
1470=[qd] 1471=[qe] 1472=[qi] 1473=[ql] 1474=[qm] 1475=[qn] 1476=[qq] 1477=[qr] 1478=[qs] 1479=[qt]
1480=[qu] 1481=[r] 1482=[r,] 1483=[r.] 1484=[r6] 1485=[ra] 1486=[re] 1487=[rg] 1488=[ri] 1489=[rj]
1490=[rm] 1491=[rn] 1492=[ro] 1493=[rp] 1494=[rs] 1495=[rt] 1496=[ru] 1497=[rv] 1498=[rw] 1499=[rã]
1500=[ré] 1501=[rê] 1502=[rí] 1503=[rô] 1504=[s] 1505=[s,] 1506=[s.] 1507=[s/] 1508=[s2] 1509=[s3]
1510=[s8] 1511=[sa] 1512=[sb] 1513=[sc] 1514=[sd] 1515=[se] 1516=[sf] 1517=[sg] 1518=[si] 1519=[sj]
1520=[s1] 1521=[sm] 1522=[so] 1523=[sp] 1524=[sr] 1525=[ss] 1526=[su] 1527=[sv] 1528=[sz] 1529=[sã]
1530=[sé] 1531=[sê] 1532=[sí] 1533=[só] 1534=[sô] 1535=[t] 1536=[tA] 1537=[ta] 1538=[tb] 1539=[tc]
1540=[td] 1541=[te] 1542=[th] 1543=[ti] 1544=[tl] 1545=[tm] 1546=[to] 1547=[tp] 1548=[tr] 1549=[tt]
1550=[tu] 1551=[tv] 1552=[tw] 1553=[ty] 1554=[tá] 1555=[té] 1556=[tô] 1557=[tú] 1558=[u] 1559=[u!]

1560=[u,] 1561=[u@] 1562=[ua] 1563=[ue] 1564=[ui] 1565=[uk] 1566=[um] 1567=[un] 1568=[up] 1569=[ur]
1570=[us] 1571=[uu] 1572=[ué] 1573=[u”] 1574=[u❤] 1575=[v] 1576=[v] 1577=[v.] 1578=[v2] 1579=[va]
1580=[vc] 1581=[vd] 1582=[ve] 1583=[vg] 1584=[vi] 1585=[vm] 1586=[vo] 1587=[vr] 1588=[vs] 1589=[vv]
1590=[vy] 1591=[vz] 1592=[vÊ] 1593=[vá] 1594=[vã] 1595=[vê] 1596=[ví] 1597=[vó] 1598=[vô] 1599=[w]
1600=[w,] 1601=[w/] 1602=[w1] 1603=[wb] 1604=[wc] 1605=[we] 1606=[wf] 1607=[wl] 1608=[wn] 1609=[wp]
1610=[ws] 1611=[ww] 1612=[wê] 1613=[wî] 1614=[x] 1615=[x] 1616=[xD] 1617=[xd] 1618=[xx] 1619=[xá]
1620=[xé] 1621=[y] 1622=[y?] 1623=[ya] 1624=[ye] 1625=[yh] 1626=[yo] 1627=[yt] 1628=[yê] 1629=[yi]
1630=[z] 1631=[z1] 1632=[z2] 1633=[z3] 1634=[za] 1635=[ze] 1636=[zu] 1637=[zé] 1638=[{ } 1639=[|]
1640=[||] 1641=[|👉] 1642=[{ }] 1643=[~] 1644=[~o] 1645=[~a] 1646=[~✿] 1647=[~🍒] 1648=[£] 1649=[©]
1650=[«] 1651=[«o] 1652=[~-] 1653=[®] 1654=[°] 1655=[·] 1656=[»] 1657=[%] 1658=[%] 1659=[¿?]
1660=[À] 1661=[ÀS] 1662=[Às] 1663=[Á] 1664=[Ã] 1665=[ÃØ] 1666=[È] 1667=[É] 1668=[É,] 1669=[É.]
1670=[ÉS] 1671=[És] 1672=[Ê] 1673=[Ñ] 1674=[Ó] 1675=[Ó] 1676=[Ó,] 1677=[Ô] 1678=[Ô,] 1679=[Ôô]
1680=[à] 1681=[às] 1682=[á] 1683=[ás] 1684=[ão] 1685=[ça] 1686=[è] 1687=[é] 1688=[é!] 1689=[é])
1690=[é,] 1691=[é.] 1692=[é:] 1693=[é?] 1694=[éá] 1695=[éq] 1697=[és] 1698=[é~] 1699=[ê]
1700=[íá] 1701=[ñ] 1702=[ñ,] 1703=[ññ] 1704=[ó] 1705=[ó!] 1706=[ó,] 1707=[ó.] 1708=[ó:] 1709=[óh]
1710=[ô] 1711=[ô,] 1712=[ôh] 1713=[û] 1714=[α] 1715=[αê] 1716=[€U] 1717=[ø] 1718=[é] 1719=[]
1720=[-] 1721=[-e] 1722=[-] 1723=[--] 1724=[-a] 1725=[‘] 1726=[‘A] 1727=[‘0] 1728=[‘o] 1729=[‘É]
1730=[‘é] 1731=[‘] 1732=[“] 1733=[“] 1734=[“A] 1735=[“E] 1736=[“I] 1737=[“0] 1738=[“a] 1739=[“o]
1740=[“É] 1741=[“é] 1742=[“...] 1743=[“] 1744=[•] 1745=[...] 1746=[...] 1747=[....] 1748=[>] 1749=[※]
1750=[!!] 1751=[] 1752=[] 1753=[c] 1754=[,] 1755=[€] 1756=[] 1757=[→] 1758=[↔] 1759=[↳]
1760=[↔] 1761=[≡] 1762=[Σ] 1763=[·] 1764=[⌚] 1765=[💻] 1766=[➡➡] 1767=[⬆⬇] 1768=[🕒] 1769=[▣]
1770=[▪] 1771=[▶] 1772=[▷] 1773=[▷🎧] 1774=[▶] 1775=[●🎥] 1776=[✳️] 1777=[🟡] 1778=[☎] 1779=[📞]
1780=[☒] 1781=[✓] 1782=[☔] 1783=[◉] 1784=[◉] 1785=[☛] 1786=[👉] 1787=[☞] 1788=[⊗] 1789=[✖]
1790=[⊕🎵] 1791=[😊] 1792=[😢] 1793=[Θ] 1794=[Θ😢] 1795=[♡] 1796=[♡,] 1797=[♡:] 1798=[♡!] 1799=[♡♡]
1800=[♡+] 1801=[♥] 1802=[♥]) 1803=[♥☔] 1804=[♥♥] 1805=[♥] 1806=[♥😊] 1807=[♦] 1808=[♪] 1809=[♫]
1810=[♣] 1811=[×♣] 1812=[▲] 1813=[⚡] 1814=[⚽] 1815=[🏈] 1816=[🏈🎤] 1817=[⋮:] 1818=[✉] 1819=[✓]
1820=[✈] 1821=[✉] 1822=[✉] 1823=[👉] 1824=[👉,] 1825=[👉♥] 1826=[👉] 1827=[👉] 1828=[👉] 1829=[👉😊]
1830=[👉] 1831=[☛♥] 1832=[✓] 1833=[✓...] 1834=[✓] 1835=[†] 1836=[◆] 1837=[◆] 1838=[◆◆] 1839=[◆♥]
1840=[◆♥] 1841=[◆😊] 1842=[◆😊] 1843=[◆] 1844=[*] 1845=[*✉] 1846=[?] 1847=[!] 1848=[! ✓] 1849=[”]
1850=[?] 1851=[??] 1852=[?] 1853=[?] 1854=[?] 1855=[?] 1856=[?] 1857=[?] 1858=[?] 1859=[?] 1860=[?] 1861=[?] 1862=[?] 1863=[?] 1864=[?] 1865=[?] 1866=[?] 1867=[...] 1868=[?] 1869=[→ 10]
1870=[▶] 1871=[↔] 1872=[↔] 1873=[] 1874=[] 1875=[] 1876=[] 1877=[] 1878=[] 1879=[]
1880=[»] 1881=[ππ] 1882=[—] 1883=[] 1884=[ξ] 1885=[Ξ] 1886=[Ξ.] 1887=[Η] 1888=[사랑] 1889=[]
1890=[NO] 1891=[my] 1892=[to] 1893=[cl] 1894=[ng] 1895=[OK] 1896=[BR] 1897=[CL] 1898=[FR] 1899=[GB]
1900=[HN] 1901=[JP] 1902=[KR] 1903=[PY] 1904=[TR] 1905=[VE] 1906=[ZA] 1907=[RAINBOW] 1908=[RAINBOW] 1909=[RAINBOW]
1910=[PLANET] 1911=[PLANETMOON] 1912=[PLANETBIRD] 1913=[PLANETSMILE] 1914=[PLANETSMILE] 1915=[PLANETSMILE] 1916=[PLANETSMILE!] 1917=[PLANETPALM] 1918=[PLANETPALM] 1919=[PLANETPALM:]
1920=[PLANETSTAR] 1921=[FLASH] 1922=[FLASH] 1923=[FLASH] 1924=[FLASH:] 1925=[FLASHFLASH] 1926=[FLASHFLASH] 1927=[FLASHFLASH] 1928=[FLASHFLASH] 1929=[FLASHFLASH:]
1930=[FLASHFLASH] 1931=[FLASHFLASH] 1932=[FLASHFLASH] 1933=[FLASHFLASH] 1934=[FLASHFLASH] 1935=[FLASHFLASH] 1936=[FLASHFLASH] 1937=[FLASHFLASH] 1938=[FLASHFLASH] 1939=[FLASHFLASH]
1940=[FLASHFLASH] 1941=[FLASHFLASH] 1942=[FLASHFLASH] 1943=[FLASHFLASH] 1944=[FLASHFLASH] 1945=[FLASHFLASH] 1946=[FLASHFLASH] 1947=[FLASHFLASH] 1948=[FLASHFLASH] 1949=[FLASHFLASH]
1950=[FLASHFLASH] 1951=[FLASHFLASH] 1952=[FLASHFLASH] 1953=[FLASHFLASH:] 1954=[FLASHFLASH] 1955=[FLASHFLASH] 1956=[FLASHFLASH] 1957=[FLASHFLASH] 1958=[FLASHFLASH] 1959=[FLASHFLASH]
1960=[FLASHFLASH] 1961=[FLASHFLASH] 1962=[FLASHFLASH] 1963=[FLASHFLASH] 1964=[FLASHFLASH] 1965=[FLASHFLASH] 1966=[FLASHFLASH] 1967=[FLASHFLASH] 1968=[FLASHFLASH] 1969=[FLASHFLASH]
1970=[FLASHFLASH] 1971=[FLASHFLASH] 1972=[FLASHFLASH] 1973=[FLASHFLASH] 1974=[FLASHFLASH] 1975=[FLASHFLASH] 1976=[FLASHFLASH] 1977=[FLASHFLASH] 1978=[FLASHFLASH] 1979=[FLASHFLASH]
1980=[FLASHFLASH] 1981=[FLASHFLASH] 1982=[FLASHFLASH] 1983=[FLASHFLASH] 1984=[FLASHFLASH] 1985=[FLASHFLASH] 1986=[FLASHFLASH] 1987=[FLASHFLASH] 1988=[FLASHFLASH] 1989=[FLASHFLASH]
1990=[FLASHFLASH] 1991=[FLASHFLASH] 1992=[FLASHFLASH] 1993=[FLASHFLASH] 1994=[FLASHFLASH] 1995=[FLASHFLASH] 1996=[FLASHFLASH] 1997=[FLASHFLASH] 1998=[FLASHFLASH] 1999=[FLASHFLASH]
2000=[FLASHFLASH] 2001=[FLASHFLASH] 2002=[FLASHFLASH] 2003=[FLASHFLASH] 2004=[FLASHFLASH] 2005=[FLASHFLASH] 2006=[FLASHFLASH] 2007=[FLASHFLASH] 2008=[FLASHFLASH] 2009=[FLASHFLASH]
2010=[FLASHFLASH] 2011=[FLASHFLASH] 2012=[FLASHFLASH] 2013=[FLASHFLASH] 2014=[FLASHFLASH] 2015=[FLASHFLASH] 2016=[FLASHFLASH] 2017=[FLASHFLASH] 2018=[FLASHFLASH] 2019=[FLASHFLASH]
2020=[FLASHFLASH] 2021=[FLASHFLASH] 2022=[FLASHFLASH] 2023=[FLASHFLASH] 2024=[FLASHFLASH] 2025=[FLASHFLASH] 2026=[FLASHFLASH] 2027=[FLASHFLASH] 2028=[FLASHFLASH] 2029=[FLASHFLASH]
2030=[FLASHFLASH] 2031=[FLASHFLASH] 2032=[FLASHFLASH] 2033=[FLASHFLASH] 2034=[FLASHFLASH] 2035=[FLASHFLASH] 2036=[FLASHFLASH] 2037=[FLASHFLASH] 2038=[FLASHFLASH] 2039=[FLASHFLASH]
2040=[FLASHFLASH] 2041=[FLASHFLASH] 2042=[FLASHFLASH] 2043=[FLASHFLASH] 2044=[FLASHFLASH] 2045=[FLASHFLASH] 2046=[FLASHFLASH] 2047=[FLASHFLASH] 2048=[FLASHFLASH] 2049=[FLASHFLASH]
2050=[FLASHFLASH] 2051=[FLASHFLASH] 2052=[FLASHFLASH] 2053=[FLASHFLASH] 2054=[FLASHFLASH] 2055=[FLASHFLASH] 2056=[FLASHFLASH] 2057=[FLASHFLASH] 2058=[FLASHFLASH] 2059=[FLASHFLASH]
2060=[FLASHFLASH] 2061=[FLASHFLASH] 2062=[FLASHFLASH] 2063=[FLASHFLASH] 2064=[FLASHFLASH] 2065=[FLASHFLASH] 2066=[FLASHFLASH] 2067=[FLASHFLASH] 2068=[FLASHFLASH] 2069=[FLASHFLASH]
2070=[FLASHFLASH] 2071=[FLASHFLASH] 2072=[FLASHFLASH] 2073=[FLASHFLASH] 2074=[FLASHFLASH] 2075=[FLASHFLASH] 2076=[FLASHFLASH] 2077=[FLASHFLASH] 2078=[FLASHFLASH] 2079=[FLASHFLASH]

2080=[👤] 2081=[👨‍🟡] 2082=[👳] 2083=[👳‍肤色] 2084=[👳‍肤色] 2085=[👳‍肤色] 2086=[👳‍肤色] 2087=[👳‍肤色] 2088=[👳‍肤色] 2089=[👳‍肤色]
 2090=[👽‍👉] 2091=[👽‍👈] 2092=[💀‍👻] 2093=[👼] 2094=[👼] 2095=[👼] 2096=[👼] 2097=[👼‍👻] 2098=[CallCheck] 2099=[CallCheck]
 2100=[❤️.] 2101=[❤️❤️] 2102=[❤️] 2103=[❤️:] 2104=[❤️❤️] 2105=[❤️❤️] 2106=[❤️⬆️] 2107=[❤️] 2108=[❤️❤️] 2109=[❤️👉]
 2110=[❤️] 2111=[❤️❤️] 2112=[❤️👉] 2113=[❤️❤️] 2114=[❤️❤️] 2115=[❤️❤️] 2116=[❤️] 2117=[❤️⭐] 2118=[❤️❤️] 2119=[❤️❤️]
 2120=[❤️❤️] 2121=[❤️👉] 2122=[❤️👉] 2123=[❤️] 2124=[❤️❤️] 2125=[❤️❤️] 2126=[❤️] 2127=[❤️❤️] 2128=[❤️] 2129=[❤️🌿]
 2130=[❤️☀️] 2131=[❤️❤️] 2132=[❤️😊] 2133=[❤️👉] 2134=[❤️] 2135=[❤️❤️] 2136=[❤️❤️] 2137=[❤️❤️] 2138=[❤️] 2139=[❤️:]
 2140=[❤️👉] 2141=[❤️❤️] 2142=[❤️❤️] 2143=[❤️❤️] 2144=[❤️] 2145=[❤️❤️] 2146=[❤️❤️] 2147=[❤️👉] 2148=[❤️] 2149=[❤️]
 2150=[❤️❤️] 2151=[❤️] 2152=[diamond 0] 2153=[💡] 2154=[💡.] 2155=[💥] 2156=[💥💥] 2157=[⚡] 2158=[✖️] 2159=[👉]
 2160=[👉,] 2161=[👉.] 2162=[👉] 2163=[👉😊] 2164=[👉😊] 2165=[👉😊] 2166=[👉😊] 2167=[👉] 2168=[📞] 2169=[📞]
 2170=[🔟] 2171=[💰] 2172=[💰] 2173=[🌐] 2174=[🌐] 2175=[🌐] 2176=[🌐] 2177=[🌐] 2178=[🌐] 2179=[🌐]
 2180=[💻💡] 2181=[💻⌚] 2182=[💼⚽] 2183=[💼🎶] 2184=[👤🎶] 2185=[👤🏆] 2186=[📝] 2187=[📝] 2188=[📈🎵] 2189=[🎵]
 2190=[☎️] 2191=[☎️] 2192=[📠] 2193=[📠0] 2194=[📠] 2195=[📠] 2196=[📠] 2197=[📠✓] 2198=[📠] 2199=[📞]
 2200=[-Calculator] 2201=[📠] 2202=[📠0] 2203=[👉] 2204=[⬇️] 2205=[📠] 2206=[📠] 2207=[📠] 2208=[📠e] 2209=[📠]
 2210=[📷] 2211=[📷:] 2212=[📷] 2213=[📷.] 2214=[📺] 2215=[📺] 2216=[📺] 2217=[📺] 2218=[📺] 2219=[📺]
 2220=[🕒] 2221=[🕒END] 2222=[🕒] 2223=[🕒🎶] 2224=[🕒] 2225=[🕒🔔] 2226=[🕒] 2227=[🕒ONI] 2228=[🕒soon] 2229=[🕒soonsoon]
 2230=[🕒TOP] 2231=[🕒18] 2232=[🕒10] 2233=[🕒] 2234=[🕒:] 2235=[🕒🕒] 2236=[🕒🕒] 2237=[🕒] 2238=[🕒●] 2239=[🕒♦]
 2240=[◆] 2241=[▼] 2242=[🕒] 2243=[🕒] 2244=[🕒] 2245=[🕒] 2246=[🕒💥] 2247=[🕒❤️] 2248=[🕒❤️] 2249=[🕒]
 2250=[🕒] 2251=[🕒] 2252=[🕒] 2253=[🕒🎃] 2254=[🕒] 2255=[🕒,] 2256=[🕒🐶] 2257=[🕒] 2258=[🕒] 2259=[🕒🎃]
 2260=[😊] 2261=[😊,] 2262=[😊.] 2263=[😊0] 2264=[😊👉] 2265=[😊😊] 2266=[😊😊] 2267=[😊😊] 2268=[😊😊] 2269=[😊😊]
 2270=[😊] 2271=[😊!] 2272=[😊☀️] 2273=[😊] 2274=[😊!] 2275=[😊] 2276=[😊😊] 2277=[😊] 2278=[😊❤️] 2279=[😊]
 2280=[😺] 2281=[😺] 2282=[😺❤️] 2283=[😺] 2284=[😺👉] 2285=[😺❤️] 2286=[😺❤️] 2287=[😺] 2288=[😺,] 2289=[😺❤️]
 2290=[😊😊] 2291=[😊😊] 2292=[😊👉] 2293=[😊] 2294=[😊😊] 2295=[😊😊] 2296=[😊] 2297=[😊] 2298=[😊,] 2299=[😊❤️]
 2300=[😊🎶] 2301=[😊🎶] 2302=[😊🐱] 2303=[😊🐶] 2304=[😊❤️] 2305=[😊❤️] 2306=[😊😊] 2307=[😊] 2308=[😊😊] 2309=[😊]
 2310=[😊] 2311=[😊🎸] 2312=[😊👉] 2313=[😊💻] 2314=[😊⬆️] 2315=[😊😊] 2316=[😊] 2317=[😊] 2318=[😊😊] 2319=[😊😊]
 2320=[😊😊] 2321=[😊] 2322=[😊] 2323=[😊] 2324=[😊] 2325=[😊] 2326=[😊👉] 2327=[😊❤️] 2328=[😊❤️] 2329=[😊❤️]
 2330=[😊😊] 2331=[😊😊] 2332=[😊👉] 2333=[😊] 2334=[😊,] 2335=[😊] 2336=[😊❤️] 2337=[😊] 2338=[😊😊] 2339=[😊]
 2340=[😊!] 2341=[😊.] 2342=[😊❤️] 2343=[😊❤️] 2344=[😊] 2345=[😊] 2346=[😊] 2347=[😊] 2348=[😊] 2349=[😊]
 2350=[😊] 2351=[😊👉] 2352=[😊😊] 2353=[😊] 2354=[😊] 2355=[😊] 2356=[😊] 2357=[😊] 2358=[😊] 2359=[😊]
 2360=[😊] 2361=[😊,] 2362=[😊😊] 2363=[😊] 2364=[😊❤️] 2365=[😊] 2366=[😊] 2367=[😊] 2368=[😊] 2369=[😊]
 2370=[😊😊] 2371=[😊] 2372=[😊] 2373=[😊] 2374=[😊] 2375=[😊❤️] 2376=[😊] 2377=[😊] 2378=[😊] 2379=[😊]
 2380=[😊] 2381=[😊❤️] 2382=[😊❤️] 2383=[😊⬆️] 2384=[😊] 2385=[😊] 2386=[😊😊] 2387=[😊] 2388=[😊] 2389=[😊😊]
 2390=[😊] 2391=[owl] 2392=[owl] 2393=[owl] 2394=[owl] 2395=[owl] 2396=[😊❤️] 2397=[😊] 2398=[😊,] 2399=[😊💻]
 2400=[😊] 2401=[😊😊] 2402=[😊😊] 2403=[😊] 2404=[😊] 2405=[😊] 2406=[😊] 2407=[😊] 2408=[😊] 2409=[👉]
 2410=[👉] 2411=[👉] 2412=[👉] 2413=[👉] 2414=[👉] 2415=[👉] 2416=[👉] 2417=[👉!] 2418=[👉,] 2419=[👉.]
 2420=[👉É] 2421=[👉] 2422=[👉] 2423=[👉] 2424=[👉TOP] 2425=[👉] 2426=[👉] 2427=[👉❤️] 2428=[👉] 2429=[👉]
 2430=[👉] 2431=[👉] 2432=[👉?] 2433=[👉] 2434=[👉!] 2435=[👉] 2436=[👉] 2437=[👉] 2438=[👉] 2439=[👉]
 2440=[👉] 2441=[👉] 2442=[👉] 2443=[👉] 2444=[👉] 2445=[👉] 2446=[👉É] 2447=[👉] 2448=[👉] 2449=[👉]
 2450=[👉] 2451=[👉] 2452=[👉] 2453=[👉] 2454=[👉] 2455=[👉] 2456=[👉] 2457=[👉] 2458=[👉] 2459=[👉]
 2460=[👉] 2461=[👉] 2462=[👉] 2463=[👉] 2464=[👉] 2465=[👉] 2466=[👉] 2467=[👉] 2468=[👉] 2469=[👉]
 2470=[👉] 2471=[👉] 2472=[👉] 2473=[👉] 2474=[👉] 2475=[👉] 2476=[👉] 2477=[👉] 2478=[👉] 2479=[👉]
 2480=[👉] 2481=[👉] 2482=[👉] 2483=[👉] 2484=[👉] 2485=[👉] 2486=[👉] 2487=[👉] 2488=[👉] 2489=[👉]
 2490=[👉] 2491=[👉] 2492=[👉] 2493=[👉] 2494=[👉] 2495=[👉] 2496=[👉] 2497=[👉] 2498=[👉] 2499=[👉]
 2500=[👉] 2501=[👉]

Temos vários emoticons, que podem influenciar o significado do texto, mas também vários termos que não parecem incluir nenhum valor.
 Vamos criar uma lista seletiva, sem os emoticons, parar atuar como stop-words.

Temos que extrair as palavras da base original para não existir variação nos índices dos dados pelo split.

In [21]: pequenas = []

```
df_twitter['tweet_text'].apply(extrai_palavras)

pd_pequenas = pd.DataFrame(pequenas, columns=['palavra'])
pd_pequenas.drop_duplicates(inplace=True)
pd_pequenas.sort_values(by='palavra', inplace=True)
pequenas = pd_pequenas['palavra'].to_list()

pd_pequenas = None

exibir_pequenas(pequenas)
```

0000=[!] 0001=[!!] 0002=[!?] 0003=[!] 0004=[!♪] 0005=[! 😊] 0006=["] 0007=[",] 0008=[-] 0009=["0]
0010=["3] 0011=["5] 0012=["A] 0013=["E] 0014=["I] 0015=["N] 0016=["0] 0017=["Y] 0018=["a] 0019=["e]
0020=["n] 0021=["o] 0022=["q] 0023=["À] 0024=["É] 0025=["é] 0026=["ó] 0027=["😊] 0028=[#] 0029=[#1]
0030=[#2] 0031=[#3] 0032=[#4] 0033=[#7] 0034=[#8] 0035=[#R] 0036=[#a] 0037=[#e] 0038=[#o] 0039=[#p]
0040=[#é] 0041=[#맘] 0042=[# 📱] 0043=[# 😊] 0044=[# 😊] 0045=[# 🎤] 0046=[# 😊] 0047=[#] 0048=[\$\$] 0049=[\$,]
0050=[\$2] 0051=[\$5] 0052=[%] 0053=[%;] 0054=['] 0055=['] 0056=[',] 0057=['3] 0058=['7] 0059=['A]
0060=['I] 0061=['0] 0062=['a] 0063=['e] 0064=['o] 0065=['É] 0066=['à] 0067=['é] 0068=[()] 0069=[(())]
0070=[(,)] 0071=[(.)] 0072=[(/)] 0073=[(1)] 0074=[(3)] 0075=[(7)] 0076=[(9)] 0077=[(:) 0078=[(@)] 0079=[(A)]
0080=[(E)] 0081=[(0)] 0082=[(Q)] 0083=[(a)] 0084=[(b)] 0085=[(e)] 0086=[(i)] 0087=[(n)] 0088=[(o)] 0089=[(q)]
0090=[(É)] 0091=[(é)] 0092=[(📲)] 0093=[()] 0094=[() !] 0095=[())] 0096=[(),] 0097=[().] 0098=[():] 0099=[();]
0100=[() 🎉] 0101=[*] 0102=[**] 0103=[*-] 0104=[*A] 0105=[*E] 0106=[*s] 0107=[+] 0108=[++] 0109=[+,]
0110=[+.] 0111=[+1] 0112=[+5] 0113=[+9] 0114=[+:] 0115=[+o] 0116=[, ,] 0117=[,, ,] 0118=[, ,?] 0119=[, ,e]
0120=[, ,n] 0121=[, ,o] 0122=[, ,...] 0123=[- ,] 0124=[- ()] 0125=[- -] 0126=[--] 0127=[-1] 0128=[-5] 0129=[- :]
0130=[-B] 0131=[-e] 0132=[-f] 0133=[-p] 0134=[-q] 0135=[-É] 0136=[-...] 0137=[- 🌄] 0138=[- 😊] 0139=[.]
0140=[.!] 0141=[..] 0142=[.g] 0143=[.] 0144=[.+] 0145=[/] 0146=[//] 0147=[/ :] 0148=[/ p] 0149=[0]
0150=[0%] 0151=[0)] 0152=[0,] 0153=[0.] 0154=[0/] 0155=[01] 0156=[02] 0157=[03] 0158=[04] 0159=[05]
0160=[06] 0161=[07] 0162=[08] 0163=[09] 0164=[0€] 0165=[1] 0166=[1%] 0167=[1)] 0168=[1,] 0169=[1-]
0170=[1.] 0171=[10] 0172=[11] 0173=[12] 0174=[13] 0175=[14] 0176=[15] 0177=[16] 0178=[17] 0179=[18]
0180=[19] 0181=[1:] 0182=[1?] 0183=[1K] 0184=[1M] 0185=[1T] 0186=[1a] 0187=[1d] 0188=[1h] 0189=[1m]
0190=[1o] 0191=[1p] 0192=[1s] 0193=[1x] 0194=[1x] 0195=[1ª] 0196=[1°] 0197=[1°] 0198=[1•] 0199=[1€]
0200=[2] 0201=[2!] 0202=[2%] 0203=[2'] 0204=[2)] 0205=[2,] 0206=[2-] 0207=[2.] 0208=[20] 0209=[21]
0210=[22] 0211=[23] 0212=[24] 0213=[25] 0214=[26] 0215=[27] 0216=[28] 0217=[29] 0218=[2:] 0219=[2?]
0220=[2L] 0221=[2T] 0222=[2a] 0223=[2d] 0224=[2h] 0225=[2k] 0226=[2o] 0227=[2t] 0228=[2x] 0229=[2])
0230=[2ª] 0231=[2°] 0232=[2º] 0233=[2•] 0234=[2€] 0235=[3] 0236=[3!] 0237=[3%] 0238=[3)] 0239=[3,]
0240=[3-] 0241=[3.] 0242=[30] 0243=[31] 0244=[32] 0245=[33] 0246=[34] 0247=[35] 0248=[36] 0249=[37]
0250=[38] 0251=[39] 0252=[3:] 0253=[3;] 0254=[3?] 0255=[3D] 0256=[3G] 0257=[3U] 0258=[3a] 0259=[3g]
0260=[3h] 0261=[3k] 0262=[3x] 0263=[3ª] 0264=[3°] 0265=[3º] 0266=[3'] 0267=[3...] 0268=[4] 0269=[4!]
0270=[4"] 0271=[4%] 0272=[4'] 0273=[4)] 0274=[4+] 0275=[4,] 0276=[4-] 0277=[4.] 0278=[40] 0279=[41]
0280=[42] 0281=[43] 0282=[44] 0283=[45] 0284=[46] 0285=[47] 0286=[48] 0287=[49] 0288=[4:] 0289=[4;]
0290=[4?] 0291=[4G] 0292=[4K] 0293=[4M] 0294=[4a] 0295=[4h] 0296=[4k] 0297=[4x] 0298=[4ª] 0299=[4°]
0300=[4°] 0301=[4'] 0302=[4€] 0303=[5] 0304=[5!] 0305=[5\$] 0306=[5%] 0307=[5'] 0308=[5)] 0309=[5+]
0310=[5,] 0311=[5.] 0312=[50] 0313=[51] 0314=[52] 0315=[53] 0316=[54] 0317=[55] 0318=[56] 0319=[57]
0320=[58] 0321=[59] 0322=[5:] 0323=[5?] 0324=[5D] 0325=[5G] 0326=[5M] 0327=[5a] 0328=[5e] 0329=[5h]
0330=[5k] 0331=[5o] 0332=[5x] 0333=[5ª] 0334=[5°] 0335=[5º] 0336=[5€] 0337=[6] 0338=[6!] 0339=[6%]
0340=[6] 0341=[6,] 0342=[6-] 0343=[6.] 0344=[60] 0345=[61] 0346=[62] 0347=[63] 0348=[64] 0349=[65]
0350=[66] 0351=[67] 0352=[68] 0353=[69] 0354=[6E] 0355=[6S] 0356=[6b] 0357=[6h] 0358=[6m] 0359=[6ª]
0360=[6°] 0361=[7] 0362=[7%] 0363=[7,] 0364=[7-] 0365=[7.] 0366=[70] 0367=[71] 0368=[72] 0369=[73]
0370=[74] 0371=[75] 0372=[76] 0373=[77] 0374=[78] 0375=[79] 0376=[7a] 0377=[7h] 0378=[7k] 0379=[7m]
0380=[7o] 0381=[7x] 0382=[7ª] 0383=[7°] 0384=[7"] 0385=[7"] 0386=[8] 0387=[8!] 0388=[8\$] 0389=[8%]
0390=[8,] 0391=[8-] 0392=[8.] 0393=[80] 0394=[81] 0395=[82] 0396=[83] 0397=[84] 0398=[85] 0399=[86]
0400=[87] 0401=[88] 0402=[89] 0403=[8h] 0404=[8k] 0405=[8ª] 0406=[8°] 0407=[8°] 0408=[8°] 0409=[9]
0410=[9!] 0411=[9%] 0412=[9)] 0413=[9,] 0414=[9-] 0415=[9.] 0416=[90] 0417=[91] 0418=[92] 0419=[93]
0420=[94] 0421=[95] 0422=[96] 0423=[97] 0424=[98] 0425=[99] 0426=[9;] 0427=[9h] 0428=[9ª] 0429=[9°]
0430=[9°] 0431=[:] 0432=[:#] 0433=[:'] 0434=[:(:) 0435=[(:) 0436=[(:*) 0437=[(: -] 0438=[(: /] 0439=[(: 0]
0440=[:3] 0441=[::] 0442=[:C] 0443=[:D] 0444=[:0] 0445=[:P] 0446=[:T] 0447=[:V] 0448=[:X] 0449=[:\]
0450=[:] 0451=[:c] 0452=[:o] 0453=[:p] 0454=[:u] 0455=[:v] 0456=[:x] 0457=[:] 0458=[:] 0459=[: ~]
0460=[;] 0461=[;\$] 0462=[; () 0463=[; () 0464=[; *] 0465=[;-] 0466=[;3] 0467=[;;] 0468=[;D] 0469=[;O]
0470=[;P] 0471=[;] 0472=[;p] 0473=[;x] 0474=[;~] 0475=[=] 0476=[=() 0477=[= =] 0478=[=/] 0479=[=D]
0480=[=0] 0481=[=T] 0482=[=)] 0483=[?] 0484=[?!] 0485=[?"] 0486=[? ?] 0487=[??] 0488=[? O] 0489=[? 😊]
0490=[@] 0491=[@,] 0492=[@.] 0493=[@?] 0494=[@g] 0495=[@s] 0496=[@...] 0497=[A] 0498=[A!] 0499=[A)]
0500=[A,] 0501=[A.] 0502=[A2] 0503=[A9] 0504=[AA] 0505=[AB] 0506=[AC] 0507=[AE] 0508=[AF] 0509=[AH]
0510=[AI] 0511=[AK] 0512=[AL] 0513=[AM] 0514=[AN] 0515=[AO] 0516=[AP] 0517=[AQ] 0518=[AR] 0519=[AS]

0520=[AT] 0521=[AU] 0522=[Aa] 0523=[Ab] 0524=[Ae] 0525=[Af] 0526=[Ah] 0527=[Ai] 0528=[A1] 0529=[Am]
0530=[An] 0531=[Ao] 0532=[Aq] 0533=[Ar] 0534=[As] 0535=[Av] 0536=[Aw] 0537=[Az] 0538=[AÍ] 0539=[Aí]
0540=[A..] 0541=[B] 0542=[B!] 0543=[B'] 0544=[B] 0545=[B,] 0546=[B-] 0547=[B.] 0548=[B1] 0549=[B3]
0550=[BA] 0551=[BB] 0552=[BC] 0553=[BD] 0554=[BE] 0555=[BF] 0556=[BG] 0557=[BH] 0558=[BI] 0559=[BK]
0560=[BL] 0561=[BM] 0562=[BN] 0563=[BP] 0564=[BR] 0565=[BS] 0566=[BT] 0567=[BU] 0568=[Ba] 0569=[Bc]
0570=[Be] 0571=[Bi] 0572=[Bj] 0573=[Bo] 0574=[Br] 0575=[By] 0576=[C] 0577=[C#] 0578=[C*] 0579=[C,]
0580=[C.] 0581=[C9] 0582=[C?] 0583=[CA] 0584=[CB] 0585=[CC] 0586=[CD] 0587=[CE] 0588=[CF] 0589=[CL]
0590=[CM] 0591=[CP] 0592=[CS] 0593=[CT] 0594=[CU] 0595=[CV] 0596=[CZ] 0597=[Ca] 0598=[Cc] 0599=[Ce]
0600=[Cm] 0601=[Ct] 0602=[CÁ] 0603=[CÚ] 0604=[Cá] 0605=[Cê] 0606=[Cú] 0607=[C"] 0608=[D] 0609=[D])
0610=[D,] 0611=[D.] 0612=[D2] 0613=[D9] 0614=[D:] 0615=[DA] 0616=[DC] 0617=[DE] 0618=[DF] 0619=[DH]
0620=[DJ] 0621=[DM] 0622=[DO] 0623=[DR] 0624=[DS] 0625=[DT] 0626=[DU] 0627=[DV] 0628=[DW] 0629=[Da]
0630=[Dc] 0631=[De] 0632=[Di] 0633=[Dj] 0634=[Dm] 0635=[Do] 0636=[Dr] 0637=[Du] 0638=[DÁ] 0639=[DÓ]
0640=[Dá] 0641=[Dè] 0642=[Dó] 0643=[E] 0644=[E'] 0645=[E,] 0646=[E3] 0647=[EA] 0648=[EC] 0649=[EF]
0650=[EH] 0651=[EI] 0652=[EJ] 0653=[EL] 0654=[EM] 0655=[EN] 0656=[EP] 0657=[ES] 0658=[ET] 0659=[EU]
0660=[EX] 0661=[Ec] 0662=[Ed] 0663=[Ee] 0664=[Ef] 0665=[Eh] 0666=[Ei] 0667=[E1] 0668=[Em] 0669=[En]
0670=[Es] 0671=[Eu] 0672=[Ex] 0673=[Ez] 0674=[E..] 0675=[F] 0676=[F"] 0677=[F.] 0678=[F1] 0679=[F3]
0680=[F5] 0681=[FB] 0682=[FC] 0683=[FF] 0684=[FH] 0685=[FL] 0686=[FM] 0687=[FS] 0688=[FT] 0689=[FX]
0690=[Ff] 0691=[Fg] 0692=[FÃ] 0693=[FÉ] 0694=[Fã] 0695=[Fé] 0696=[Fê] 0697=[G] 0698=[G'] 0699=[G-]
0700=[G.] 0701=[G1] 0702=[G2] 0703=[G4] 0704=[GB] 0705=[GD] 0706=[GE] 0707=[GG] 0708=[GJ] 0709=[GM]
0710=[GO] 0711=[GP] 0712=[GS] 0713=[GT] 0714=[GV] 0715=[GW] 0716=[Ga] 0717=[Gd] 0718=[Gi] 0719=[Gn]
0720=[Go] 0721=[Gu] 0722=[H] 0723=[H.] 0724=[HA] 0725=[HB] 0726=[HC] 0727=[HD] 0728=[HG] 0729=[HJ]
0730=[HM] 0731=[HP] 0732=[HQ] 0733=[HR] 0734=[Ha] 0735=[He] 0736=[Hi] 0737=[Hj] 0738=[H1] 0739=[Hm]
0740=[Hn] 0741=[Hs] 0742=[HÁ] 0743=[Há] 0744=[I] 0745=[I"] 0746=[I,] 0747=[I.] 0748=[I5] 0749=[I:]
0750=[IA] 0751=[IC] 0752=[ID] 0753=[IF] 0754=[IG] 0755=[IH] 0756=[II] 0757=[IM] 0758=[IN] 0759=[IP]
0760=[IR] 0761=[IS] 0762=[IT] 0763=[IU] 0764=[IV] 0765=[IX] 0766=[Ia] 0767=[Ig] 0768=[Ih] 0769=[Ii]
0770=[Im] 0771=[In] 0772=[Ir] 0773=[Is] 0774=[It] 0775=[Iä] 0776=[I 🧑] 0777=[J] 0778=[J.] 0779=[J2]
0780=[JA] 0781=[JB] 0782=[JF] 0783=[JG] 0784=[JK] 0785=[JM] 0786=[JN] 0787=[JP] 0788=[JR] 0789=[JU]
0790=[JW] 0791=[JZ] 0792=[Ja] 0793=[Ji] 0794=[Jk] 0795=[Jo] 0796=[Jp] 0797=[Jr] 0798=[Ju] 0799=[JÁ]
0800=[Jà] 0801=[Já] 0802=[Jã] 0803=[Jô] 0804=[Jú] 0805=[K] 0806=[K.] 0807=[K9] 0808=[KD] 0809=[KI]
0810=[KK] 0811=[KM] 0812=[KW] 0813=[Kd] 0814=[Ke] 0815=[Kg] 0816=[Ki] 0817=[KK] 0818=[Km] 0819=[Ko]
0820=[Ku] 0821=[K"] 0822=[L] 0823=[L..] 0824=[L?] 0825=[LA] 0826=[LC] 0827=[LE] 0828=[LG] 0829=[LI]
0830=[LP] 0831=[LS] 0832=[LU] 0833=[LW] 0834=[La] 0835=[Le] 0836=[Li] 0837=[Lo] 0838=[Lu] 0839=[Lx]
0840=[Ly] 0841=[LÁ] 0842=[Lá] 0843=[Lê] 0844=[M] 0845=[M,] 0846=[MA] 0847=[MC] 0848=[ME] 0849=[MG]
0850=[MI] 0851=[MJ] 0852=[MK] 0853=[ML] 0854=[MP] 0855=[MS] 0856=[MT] 0857=[MV] 0858=[MY] 0859=[Ma]
0860=[Mc] 0861=[Me] 0862=[Mi] 0863=[Mo] 0864=[Mr] 0865=[Mt] 0866=[Mv] 0867=[My] 0868=[Má] 0869=[MÁ]
0870=[Má] 0871=[Mó] 0872=[N] 0873=[N,] 0874=[N.] 0875=[NA] 0876=[ND] 0877=[NE] 0878=[NF] 0879=[NI]
0880=[NO] 0881=[NR] 0882=[NT] 0883=[NX] 0884=[NY] 0885=[Na] 0886=[Nc] 0887=[Nd] 0888=[Nf] 0889=[Ni]
0890=[Nn] 0891=[No] 0892=[Nr] 0893=[Ns] 0894=[Nu] 0895=[Nw] 0896=[Nº] 0897=[NÉ] 0898=[Né] 0899=[Nó]
0900=[O] 0901=[O'] 0902=[O,] 0903=[O..] 0904=[OC] 0905=[OF] 0906=[OH] 0907=[OI] 0908=[OJ] 0909=[OK]
0910=[ON] 0911=[OP] 0912=[OQ] 0913=[OR] 0914=[OS] 0915=[OT] 0916=[OU] 0917=[OW] 0918=[Oa] 0919=[Oc]
0920=[Of] 0921=[Oh] 0922=[Oi] 0923=[Ok] 0924=[On] 0925=[Oo] 0926=[Oq] 0927=[Or] 0928=[Os] 0929=[Ou]
0930=[Ow] 0931=[O...] 0932=[P] 0933=[P,] 0934=[P.] 0935=[P/] 0936=[P1] 0937=[PA] 0938=[PB] 0939=[PC]
0940=[PE] 0941=[PF] 0942=[PG] 0943=[PL] 0944=[PM] 0945=[PN] 0946=[PP] 0947=[PQ] 0948=[PR] 0949=[PS]
0950=[PT] 0951=[PV] 0952=[Pa] 0953=[Pc] 0954=[Pf] 0955=[Po] 0956=[Pq] 0957=[Ps] 0958=[Pt] 0959=[Pv]
0960=[PÉ] 0961=[Pá] 0962=[Pé] 0963=[Pó] 0964=[Pô] 0965=[Q] 0966=[Q+] 0967=[QB] 0968=[QG] 0969=[QI]
0970=[QQ] 0971=[QR] 0972=[Qd] 0973=[Qm] 0974=[Qq] 0975=[Qr] 0976=[R] 0977=[R#] 0978=[R\$] 0979=[R.]
0980=[R6] 0981=[RB] 0982=[RC] 0983=[RD] 0984=[RE] 0985=[RG] 0986=[RH] 0987=[RJ] 0988=[RL] 0989=[RM]
0990=[RN] 0991=[RO] 0992=[RP] 0993=[RR] 0994=[RS] 0995=[RT] 0996=[RU] 0997=[RV] 0998=[Rh] 0999=[Ri]
1000=[Rs] 1001=[Rt] 1002=[Ru] 1003=[Ré] 1004=[S] 1005=[S.] 1006=[S2] 1007=[SA] 1008=[SC] 1009=[SD]
1010=[SE] 1011=[SH] 1012=[SI] 1013=[SL] 1014=[SM] 1015=[SO] 1016=[SP] 1017=[SS] 1018=[SU] 1019=[SV]
1020=[Sa] 1021=[Se] 1022=[Si] 1023=[So] 1024=[Sp] 1025=[Sr] 1026=[St] 1027=[Sv] 1028=[Sò] 1029=[SÓ]
1030=[Sá] 1031=[Sé] 1032=[Só] 1033=[Sô] 1034=[T] 1035=[T,] 1036=[T1] 1037=[T3] 1038=[TA] 1039=[TB]

1040=[TD] 1041=[TE] 1042=[TF] 1043=[TG] 1044=[TH] 1045=[TI] 1046=[TJ] 1047=[TL] 1048=[TO] 1049=[TR]
1050=[TS] 1051=[TT] 1052=[TU] 1053=[TV] 1054=[TW] 1055=[TY] 1056=[Ta] 1057=[Tb] 1058=[Td] 1059=[Te]
1060=[Ti] 1061=[To] 1062=[Tp] 1063=[Tt] 1064=[Tu] 1065=[Tv] 1066=[Ty] 1067=[TÁ] 1068=[TÔ] 1069=[Tá]
1070=[Tó] 1071=[Tô] 1072=[TÚ] 1073=[T...] 1074=[U] 1075=[U\$] 1076=[U,] 1077=[U2] 1078=[UB] 1079=[UE]
1080=[UF] 1081=[UI] 1082=[UK] 1083=[UM] 1084=[UP] 1085=[US] 1086=[UT] 1087=[UV] 1088=[UX] 1089=[Ua]
1090=[Ue] 1091=[Uh] 1092=[Ui] 1093=[Um] 1094=[Un] 1095=[Uo] 1096=[Up] 1097=[Uq] 1098=[Us] 1099=[Ué]
1100=[V] 1101=[V!] 1102=[V,] 1103=[V.] 1104=[VA] 1105=[VC] 1106=[VD] 1107=[VE] 1108=[VI] 1109=[VM]
1110=[VO] 1111=[VP] 1112=[VR] 1113=[VS] 1114=[VT] 1115=[VV] 1116=[VW] 1117=[Va] 1118=[Vc] 1119=[Ve]
1120=[Vi] 1121=[Vo] 1122=[VÁ] 1123=[VÊ] 1124=[Vá] 1125=[Vé] 1126=[Vê] 1127=[Ví] 1128=[Vó] 1129=[Vô]
1130=[W] 1131=[W1] 1132=[WC] 1133=[WD] 1134=[WM] 1135=[WO] 1136=[WP] 1137=[WT] 1138=[WU] 1139=[WW]
1140=[We] 1141=[Ws] 1142=[Wu] 1143=[X] 1144=[X'] 1145=[X,] 1146=[X.] 1147=[XA] 1148=[XD] 1149=[XI]
1150=[XP] 1151=[XS] 1152=[XX] 1153=[XY] 1154=[Xs] 1155=[Xô] 1156=[Y] 1157=[YG] 1158=[YT] 1159=[Ya]
1160=[Yi] 1161=[Yo] 1162=[Yê] 1163=[Z] 1164=[Z6] 1165=[ZL] 1166=[ZN] 1167=[Ze] 1168=[Zs] 1169=[Zé]
1170=[[]] 1171=[[];] 1172=[[]] 1173=[[]a] 1174=[[]p] 1175=[\] 1176=[\o] 1177=[\ö] 1178=[[]] 1179=[[],]
1180=[^] 1181=[^.] 1182=[^~] 1183=[_] 1184=[_.] 1185=[_q] 1186=[a] 1187=[a)] 1188=[a,] 1189=[a.]
1190=[aH] 1191=[aI] 1192=[aa] 1193=[ab] 1194=[ad] 1195=[ae] 1196=[af] 1197=[ah] 1198=[ai] 1199=[ak]
1200=[aI] 1201=[am] 1202=[an] 1203=[ao] 1204=[ap] 1205=[aq] 1206=[ar] 1207=[as] 1208=[at] 1209=[au]
1210=[av] 1211=[aw] 1212=[ay] 1213=[aí] 1214=[aê] 1215=[aí] 1216=[a...] 1217=[b] 1218=[b)] 1219=[b*]
1220=[b,] 1221=[b4] 1222=[b?] 1223=[ba] 1224=[bb] 1225=[bc] 1226=[bd] 1227=[be] 1228=[bf] 1229=[bg]
1230=[bh] 1231=[bi] 1232=[bj] 1233=[bk] 1234=[bo] 1235=[bp] 1236=[br] 1237=[bs] 1238=[bu] 1239=[bv]
1240=[by] 1241=[bê] 1242=[c] 1243=[c#] 1244=[c)] 1245=[c*] 1246=[c,] 1247=[c/] 1248=[c2] 1249=[c9]
1250=[c:] 1251=[ca] 1252=[cb] 1253=[cc] 1254=[cd] 1255=[ce] 1256=[cf] 1257=[cg] 1258=[ci] 1259=[cm]
1260=[cn] 1261=[co] 1262=[cp] 1263=[cr] 1264=[cs] 1265=[cu] 1266=[cv] 1267=[cy] 1268=[cá] 1269=[cê]
1270=[cú] 1271=[d] 1272=[d*] 1273=[d+] 1274=[d4] 1275=[dA] 1276=[dE] 1277=[da] 1278=[db] 1279=[dc]
1280=[dd] 1281=[de] 1282=[dg] 1283=[di] 1284=[dj] 1285=[dk] 1286=[dm] 1287=[do] 1288=[dq] 1289=[dr]
1290=[dt] 1291=[du] 1292=[dw] 1293=[dx] 1294=[dá] 1295=[dê] 1296=[dó] 1297=[e] 1298=[e'] 1299=[e,]
1300=[e.] 1301=[e:] 1302=[eU] 1303=[ea] 1304=[ed] 1305=[ef] 1306=[eh] 1307=[ei] 1308=[ej] 1309=[el]
1310=[em] 1311=[en] 1312=[eo] 1313=[ep] 1314=[eq] 1315=[er] 1316=[es] 1317=[et] 1318=[eu] 1319=[ev]
1320=[ew] 1321=[ex] 1322=[ey] 1323=[ez] 1324=[eñ] 1325=[e...] 1326=[f] 1327=[f*] 1328=[f1] 1329=[f3]
1330=[fa] 1331=[fb] 1332=[fc] 1333=[fd] 1334=[fe] 1335=[ff] 1336=[fg] 1337=[fi] 1338=[fk] 1339=[f1]
1340=[fp] 1341=[fq] 1342=[ft] 1343=[fu] 1344=[fx] 1345=[fá] 1346=[fâ] 1347=[fé] 1348=[fê] 1349=[g]
1350=[g!] 1351=[g1] 1352=[g4] 1353=[ga] 1354=[gc] 1355=[gd] 1356=[gg] 1357=[gi] 1358=[gl] 1359=[gn]
1360=[go] 1361=[gt] 1362=[gu] 1363=[gv] 1364=[gy] 1365=[h] 1366=[h,] 1367=[h.] 1368=[ha] 1369=[hb]
1370=[hc] 1371=[hd] 1372=[he] 1373=[hi] 1374=[hj] 1375=[hm] 1376=[ho] 1377=[hp] 1378=[hq] 1379=[hr]
1380=[hs] 1381=[há] 1382=[hê] 1383=[i] 1384=[i5] 1385=[i7] 1386=[iM] 1387=[ia] 1388=[ic] 1389=[id]
1390=[if] 1391=[ig] 1392=[ih] 1393=[im] 1394=[in] 1395=[io] 1396=[ip] 1397=[iq] 1398=[ir] 1399=[is]
1400=[it] 1401=[iz] 1402=[j] 1403=[j7] 1404=[ja] 1405=[jb] 1406=[je] 1407=[jg] 1408=[ji] 1409=[jk]
1410=[jm] 1411=[jo] 1412=[jp] 1413=[jr] 1414=[ju] 1415=[já] 1416=[jê] 1417=[jó] 1418=[jô] 1419=[k]
1420=[k)] 1421=[ka] 1422=[kd] 1423=[kg] 1424=[ki] 1425=[kk] 1426=[kl] 1427=[km] 1428=[ko] 1429=[ku]
1430=[l] 1431=[la] 1432=[le] 1433=[lg] 1434=[li] 1435=[ll] 1436=[lo] 1437=[lu] 1438=[lx] 1439=[ly]
1440=[lz] 1441=[lá] 1442=[lã] 1443=[lê] 1444=[lí] 1445=[m] 1446=[m,] 1447=[m9] 1448=[ma] 1449=[mb]
1450=[mc] 1451=[md] 1452=[me] 1453=[mh] 1454=[mi] 1455=[ml] 1456=[mm] 1457=[mn] 1458=[mo] 1459=[mp]
1460=[mr] 1461=[ms] 1462=[mt] 1463=[mv] 1464=[mw] 1465=[mx] 1466=[my] 1467=[m²] 1468=[má] 1469=[mó]
1470=[mô] 1471=[n] 1472=[n!] 1473=[n'] 1474=[n,] 1475=[n1] 1476=[n?] 1477=[nA] 1478=[na] 1479=[nc]
1480=[nd] 1481=[ne] 1482=[nf] 1483=[ng] 1484=[ni] 1485=[nl] 1486=[nm] 1487=[nn] 1488=[no] 1489=[np]
1490=[nq] 1491=[ns] 1492=[nt] 1493=[nu] 1494=[nv] 1495=[nx] 1496=[ny] 1497=[nº] 1498=[nº] 1499=[ná]
1500=[né] 1501=[nó] 1502=[nô] 1503=[n'] 1504=[o] 1505=[o!] 1506=[o..] 1507=[o/] 1508=[o:] 1509=[o?]
1510=[oo] 1511=[oS] 1512=[oa] 1513=[ob] 1514=[oc] 1515=[of] 1516=[oh] 1517=[oi] 1518=[ok] 1519=[om]
1520=[on] 1521=[oo] 1522=[op] 1523=[oq] 1524=[or] 1525=[os] 1526=[ou] 1527=[ow] 1528=[o...] 1529=[p]
1530=[p.] 1531=[p/] 1532=[p1] 1533=[p2] 1534=[pQ] 1535=[pa] 1536=[pc] 1537=[pd] 1538=[pe] 1539=[pf]
1540=[pg] 1541=[ph] 1542=[pj] 1543=[pk] 1544=[p1] 1545=[pm] 1546=[pn] 1547=[po] 1548=[pp] 1549=[pq]
1550=[pr] 1551=[ps] 1552=[pt] 1553=[pv] 1554=[pw] 1555=[px] 1556=[pá] 1557=[pé] 1558=[pê] 1559=[pó]

1560=[pô] 1561=[q] 1562=[q]) 1563=[q,] 1564=[q:] 1565=[q?] 1566=[qa] 1567=[qd] 1568=[qe] 1569=[qi]
1570=[q1] 1571=[qm] 1572=[qn] 1573=[qq] 1574=[qr] 1575=[qs] 1576=[qt] 1577=[qu] 1578=[r] 1579=[r,]
1580=[r.] 1581=[r6] 1582=[ra] 1583=[re] 1584=[rg] 1585=[ri] 1586=[rj] 1587=[rm] 1588=[rn] 1589=[ro]
1590=[rp] 1591=[rs] 1592=[rt] 1593=[ru] 1594=[rv] 1595=[rw] 1596=[rã] 1597=[ré] 1598=[rê] 1599=[rí]
1600=[rô] 1601=[s] 1602=[s,] 1603=[s.] 1604=[s/] 1605=[s2] 1606=[s3] 1607=[s7] 1608=[s8] 1609=[sa]
1610=[sb] 1611=[sc] 1612=[sd] 1613=[se] 1614=[sf] 1615=[sg] 1616=[si] 1617=[sj] 1618=[sl] 1619=[sm]
1620=[so] 1621=[sp] 1622=[sr] 1623=[ss] 1624=[su] 1625=[sv] 1626=[sz] 1627=[sã] 1628=[sé] 1629=[sê]
1630=[sí] 1631=[só] 1632=[sô] 1633=[t] 1634=[t,] 1635=[t2] 1636=[t3] 1637=[tA] 1638=[ta] 1639=[tb]
1640=[tc] 1641=[td] 1642=[te] 1643=[tg] 1644=[th] 1645=[ti] 1646=[tl] 1647=[tm] 1648=[to] 1649=[tp]
1650=[tr] 1651=[tt] 1652=[tu] 1653=[tv] 1654=[tw] 1655=[ty] 1656=[tá] 1657=[té] 1658=[tô] 1659=[tú]
1660=[u] 1661=[u!] 1662=[u,] 1663=[u@] 1664=[ua] 1665=[ue] 1666=[ui] 1667=[uk] 1668=[um] 1669=[un]
1670=[up] 1671=[ur] 1672=[us] 1673=[uu] 1674=[ué] 1675=[u”] 1676=[u♥] 1677=[v] 1678=[v)] 1679=[v.]
1680=[v2] 1681=[va] 1682=[vc] 1683=[vd] 1684=[ve] 1685=[vg] 1686=[vi] 1687=[vm] 1688=[vo] 1689=[vr]
1690=[vs] 1691=[vv] 1692=[vy] 1693=[vz] 1694=[vÊ] 1695=[vá] 1696=[vã] 1697=[vê] 1698=[ví] 1699=[vó]
1700=[vô] 1701=[w] 1702=[w,] 1703=[w/] 1704=[w1] 1705=[wb] 1706=[wc] 1707=[we] 1708=[wf] 1709=[wl]
1710=[wn] 1711=[wp] 1712=[ws] 1713=[ww] 1714=[wê] 1715=[wí] 1716=[x] 1717=[x)] 1718=[xD] 1719=[xd]
1720=[xs] 1721=[xx] 1722=[xá] 1723=[xé] 1724=[y] 1725=[y?] 1726=[ya] 1727=[ye] 1728=[yg] 1729=[yh]
1730=[yi] 1731=[yo] 1732=[yt] 1733=[yê] 1734=[yi] 1735=[z] 1736=[z1] 1737=[z2] 1738=[z3] 1739=[za]
1740=[ze] 1741=[zu] 1742=[zé] 1743=[{ }] 1744=[|] 1745=[||] 1746=[|👉|] 1747=[~] 1748=[~] 1749=[~o]
1750=[~a] 1751=[~🌸] 1752=[~🍒] 1753=[£] 1754=[€] 1755=[¤] 1756=[«o] 1757=[~~] 1758=[®] 1759=[°]
1760=[·] 1761=[«] 1762=[¼] 1763=[½] 1764=[ξ?] 1765=[À] 1766=[ÀS] 1767=[Às] 1768=[Á] 1769=[Ás]
1770=[Ã] 1771=[Ãø] 1772=[È] 1773=[É] 1774=[É,] 1775=[É.] 1776=[ÉS] 1777=[És] 1778=[Ê] 1779=[Ê,]
1780=[Ñ] 1781=[Ò] 1782=[Ó] 1783=[Ó,] 1784=[Ô] 1785=[Ô,] 1786=[Ôô] 1787=[à] 1788=[às] 1789=[á]
1790=[ás] 1791=[ão] 1792=[ça] 1793=[è] 1794=[é] 1795=[é!] 1796=[é)] 1797=[é,] 1798=[é.] 1799=[é:]
1800=[é;] 1801=[é?] 1802=[éa] 1803=[éh] 1804=[éq] 1805=[és] 1806=[é~] 1807=[ê] 1808=[íá] 1809=[ñ]
1810=[ñ,] 1811=[ññ] 1812=[ó] 1813=[ó!] 1814=[ó,] 1815=[ó.] 1816=[ó:] 1817=[óh] 1818=[ô] 1819=[ô,]
1820=[ôh] 1821=[û] 1822=[é] 1823=[α] 1824=[αê] 1825=[€U] 1826=[ø] 1827=[é] 1828=[] 1829=[-]
1830=[-e] 1831=[-] 1832=[--] 1833=[-a] 1834=[‘] 1835=[‘A] 1836=[‘0] 1837=[‘o] 1838=[‘É] 1839=[‘é]
1840=[‘] 1841=[“] 1842=[“,] 1843=[“A] 1844=[“E] 1845=[“I] 1846=[“0] 1847=[“a] 1848=[“o] 1849=[“É]
1850=[“é] 1851=[“..] 1852=[“] 1853=[•] 1854=[•...] 1855=[...] 1856=[....] 1857=[>] 1858=[※] 1859=[!!]
1860=[] 1861=[] 1862=[⟨] 1863=[⟨,] 1864=[€] 1865=[⟨] 1866=[→] 1867=[↔] 1868=[↳] 1869=[↔...]
1870=[=] 1871=[>] 1872=[·] 1873=[⌚] 1874=[🕒🕒] 1875=[🕒🕒] 1876=[🕒🕒] 1877=[🕒] 1878=[🕒] 1879=[▪]
1880=[▶] 1881=[▶] 1882=[▶🎧] 1883=[▶] 1884=[●🎥] 1885=[*] 1886=[🟡] 1887=[📞] 1888=[📞] 1889=[▣]
1890=[▢] 1891=[▢] 1892=[●] 1893=[●] 1894=[●] 1895=[●] 1896=[●] 1897=[●] 1898=[●] 1899=[▢]
1900=[ⓘ] 1901=[ⓘ] 1902=[ⓘ] 1903=[ⓘ] 1904=[ⓘ] 1905=[ⓘ] 1906=[ⓘ] 1907=[ⓘ] 1908=[ⓘ] 1909=[ⓘ]
1910=[ⓘ] 1911=[ⓘ] 1912=[ⓘ] 1913=[ⓘ] 1914=[ⓘ] 1915=[ⓘ] 1916=[ⓘ] 1917=[ⓘ] 1918=[ⓘ] 1919=[ⓘ]
1920=[✨] 1921=[✨] 1922=[✨] 1923=[✨] 1924=[✨] 1925=[✨] 1926=[✨] 1927=[✨] 1928=[✨] 1929=[✨]
1930=[✨] 1931=[✨] 1932=[✨] 1933=[✨] 1934=[✨] 1935=[✨,] 1936=[✨] 1937=[✨] 1938=[✨] 1939=[✨]
1940=[✨] 1941=[✨] 1942=[✨] 1943=[✨] 1944=[✨] 1945=[✨] 1946=[✨] 1947=[✨] 1948=[✨] 1949=[✨]
1950=[✨] 1951=[✨] 1952=[✨] 1953=[✨] 1954=[✨] 1955=[✨] 1956=[✨] 1957=[✨] 1958=[✨] 1959=[?]
1960=[!] 1961=[!] 1962=[!] 1963=[!] 1964=[!] 1965=[!] 1966=[!] 1967=[!] 1968=[!] 1969=[!]
1970=[❤] 1971=[❤] 1972=[❤] 1973=[❤] 1974=[❤] 1975=[❤] 1976=[❤] 1977=[❤] 1978=[—] 1979=[→]
1980=[→...] 1981=[→] 1982=[→] 1983=[→] 1984=[→] 1985=[→] 1986=[→] 1987=[→] 1988=[→] 1989=[→]
1990=[★] 1991=[★] 1992=[★] 1993=[★] 1994=[★] 1995=[—] 1996=[—] 1997=[ε] 1998=[3] 1999=[3.]
2000=[¶] 2001=[사랑] 2002=[] 2003=[NO] 2004=[my] 2005=[to] 2006=[🎉] 2007=[CL] 2008=[NG] 2009=[OK]
2010=[BR] 2011=[CL] 2012=[FR] 2013=[GB] 2014=[HN] 2015=[JP] 2016=[KR] 2017=[PY] 2018=[TR] 2019=[US]
2020=[VE] 2021=[ZA] 2022=[🌈] 2023=[🌈] 2024=[🌈] 2025=[🌈] 2026=[🌙] 2027=[🌙] 2028=[🌙] 2029=[🌙]
2030=[☀] 2031=[☀!] 2032=[☀] 2033=[☀] 2034=[☀:] 2035=[☀] 2036=[☀] 2037=[☀] 2038=[☀] 2039=[☀]
2040=[☀] 2041=[☀] 2042=[☀] 2043=[☀] 2044=[☀] 2045=[☀:] 2046=[☀] 2047=[☀] 2048=[☀] 2049=[☀]
2050=[☀] 2051=[☀] 2052=[☀] 2053=[☀] 2054=[☀] 2055=[☀] 2056=[☀] 2057=[☀] 2058=[☀] 2059=[☀]
2060=[☀] 2061=[☀] 2062=[☀] 2063=[☀] 2064=[☀] 2065=[☀] 2066=[☀] 2067=[☀] 2068=[☀] 2069=[☀]
2070=[☀] 2071=[☀] 2072=[☀] 2073=[☀] 2074=[☀] 2075=[☀] 2076=[☀] 2077=[☀] 2078=[☀] 2079=[☀]

2080=[🎁 .] 2081=[🎉 🎈] 2082=[🎃 🎃] 2083=[🎊] 2084=[🎀] 2085=[🎉] 2086=[🎈] 2087=[🎉] 2088=[🎤] 2089=[🎵 🎵]
2090=[🎤] 2091=[🎤] 2092=[🎤 🎥] 2093=[🎤] 2094=[🎤 📸] 2095=[🎤 📺] 2096=[🎤] 2097=[🎤 😊] 2098=[🎤 🎵] 2099=[🎤]
2100=[🎤 🎵] 2101=[🎤 🎤] 2102=[🎵 🎵] 2103=[🎅] 2104=[🎕] 2105=[🎕 A] 2106=[🎕 🎕] 2107=[🎕] 2108=[🎕 A] 2109=[🎕 🎕]
2110=[🎕 🎵] 2111=[🎵] 2112=[🎵 🎵] 2113=[🎕 🎕] 2114=[🎕] 2115=[🎕] 2116=[🎕 🎤] 2117=[🎕 🎤] 2118=[🎕 🎵] 2119=[🎕 🎵]
2120=[🎵] 2121=[🎵 🏆] 2122=[🚗 🚗] 2123=[🚗] 2124=[🚗] 2125=[🏠] 2126=[🚗] 2127=[🚗] 2128=[🎵] 2129=[🎵]
2130=[🐾] 2131=[🐾] 2132=[🐾] 2133=[🐾 /] 2134=[🐾 /] 2135=[🐾] 2136=[🐿] 2137=[🐿] 2138=[🐿] 2139=[🐿]
2140=[🐿] 2141=[🐿] 2142=[🐿 🖤] 2143=[🐿 🖤] 2144=[🐿] 2145=[🐿 🖤] 2146=[🐿 🖤] 2147=[🐿] 2148=[🐿] 2149=[🐿]
2150=[🖤] 2151=[🖤] 2152=[🖤] 2153=[🖤] 2154=[🖤] 2155=[🖤] 2156=[🖤 🖤] 2157=[🖤 🚨] 2158=[🖤] 2159=[🖤 🖤]
2160=[🖤 ❤️] 2161=[🖤] 2162=[🖤 🖤] 2163=[🖤 🖤] 2164=[🖤] 2165=[🖤 0] 2166=[🖤 ...] 2167=[🖤 🖤] 2168=[🖤 🖤] 2169=[🖤]
2170=[🖤 ❤️] 2171=[🖤] 2172=[🖤] 2173=[🖤] 2174=[🖤 🖤] 2175=[🖤 🚨] 2176=[🖤 😊] 2177=[🖤] 2178=[🖤] 2179=[🖤 🖤]
2180=[🖤] 2181=[🖤 !] 2182=[🖤] 2183=[🖤] 2184=[🖤] 2185=[🖤] 2186=[🖤] 2187=[🖤] 2188=[🖤 🖤] 2189=[🖤 🖤]
2190=[🖤 😊] 2191=[🖤 😊] 2192=[🖤] 2193=[🖤] 2194=[🖤] 2195=[🖤] 2196=[🖤] 2197=[🖤 😊] 2198=[🖤 🎵] 2199=[🖤 🖤]
2200=[🎭] 2201=[🎭 -] 2202=[🎭 🎭] 2203=[🎭] 2204=[🎭] 2205=[🎭] 2206=[🎭] 2207=[🎭] 2208=[🎭 ❤️] 2209=[🎭 🎭]
2210=[🎭] 2211=[🎭 ❤️] 2212=[🎭] 2213=[🎭] 2214=[🎭] 2215=[🎭] 2216=[🎭] 2217=[🎭] 2218=[🎭] 2219=[🎭]
2220=[🐱 🐱] 2221=[🐱] 2222=[🐱] 2223=[🐱 🐱] 2224=[🐱] 2225=[🐱 🐱] 2226=[🐱] 2227=[🐱] 2228=[🐱] 2229=[🐱]
2230=[🐱] 2231=[🐱 😊] 2232=[🐱] 2233=[🐱] 2234=[🐱 .] 2235=[🐱 ❤️] 2236=[🐱] 2237=[🐱 :] 2238=[🐱 ❤️] 2239=[🐱 ❤️]
2240=[🖤 ↑TOP] 2241=[🖤] 2242=[🖤 🖤] 2243=[🖤 🖤] 2244=[🖤] 2245=[🖤 :] 2246=[🖤 ❤️] 2247=[🖤 🐦] 2248=[🖤 🐱] 2249=[🖤 ❤️]
2250=[🖤 ❤️] 2251=[🖤 ❤️] 2252=[🖤 ❤️] 2253=[🖤] 2254=[🖤 ★] 2255=[🖤 ❤️] 2256=[🖤 ❤️] 2257=[🖤 ❤️] 2258=[🖤 ❤️] 2259=[🖤 ❤️]
2260=[🖤 🌸] 2261=[🖤] 2262=[🖤 🌸] 2263=[🖤 🌸] 2264=[🖤] 2265=[🖤 🌸] 2266=[🖤 😊] 2267=[🖤] 2268=[🖤 🌸] 2269=[🖤 🌻]
2270=[🖤 🖤] 2271=[🖤 🖤] 2272=[🖤 😊] 2273=[🖤 🖤] 2274=[🖤] 2275=[🖤 🌸] 2276=[🖤 🖤] 2277=[🖤 🌻] 2278=[🖤] 2279=[🖤 :]
2280=[🖤 🎮] 2281=[🖤 🖤] 2282=[🖤 🖤] 2283=[🖤 🖤] 2284=[🖤] 2285=[🖤 🖤] 2286=[🖤 🖤] 2287=[🖤 😊] 2288=[🖤 😊] 2289=[🖤]
2290=[🖤] 2291=[🖤 ❤️] 2292=[🖤 ❤️] 2293=[🖤] 2294=[🖤 0] 2295=[🖤 .] 2296=[🖤 .] 2297=[🖤] 2298=[🖤] 2299=[🖤 🚨]
2300=[🖤] 2301=[🖤] 2302=[🖤] 2303=[🖤 ,] 2304=[🖤 .] 2305=[🖤] 2306=[🖤] 2307=[🖤 ❤️] 2308=[🖤 😊] 2309=[🖤 😊]
2310=[🖤 ✘] 2311=[🖤 😊] 2312=[🖤] 2313=[🖤] 2314=[🖤] 2315=[🖤 100] 2316=[🖤] 2317=[🖤 💰] 2318=[🖤] 2319=[🖤]
2320=[🖤] 2321=[🖤] 2322=[🖤] 2323=[🖤] 2324=[🖤] 2325=[🖤] 2326=[🖤] 2327=[🖤 ⏳] 2328=[🖤 ⚽] 2329=[🖤 🎵]
2330=[🖤 🎵] 2331=[🖤 🎵] 2332=[🖤] 2333=[🖤] 2334=[🖤 ❤️] 2335=[🖤 🎵] 2336=[🖤] 2337=[🖤] 2338=[🖤] 2339=[🖤]
2340=[🖤 0] 2341=[🖤 🎵] 2342=[🖤] 2343=[🖤] 2344=[🖤] 2345=[🖤] 2346=[🖤] 2347=[🖤] 2348=[🖤] 2349=[🖤 0]
2350=[🖤] 2351=[🖤] 2352=[🖤] 2353=[🖤] 2354=[🖤] 2355=[🖤 e] 2356=[🖤] 2357=[🖤] 2358=[🖤 :] 2359=[🖤]
2360=[🖤 .] 2361=[🖤] 2362=[🖤] 2363=[🖤] 2364=[🖤] 2365=[🖤] 2366=[🖤] 2367=[🖤] 2368=[🔍] 2369=[🖤]
2370=[🖤 🎵] 2371=[🖤] 2372=[🖤] 2373=[🖤] 2374=[🔍] 2375=[🔍] 2376=[🔍] 2377=[🔍] 2378=[🔍] 2379=[🔍 II]
2380=[🖤] 2381=[🖤 :] 2382=[🖤 🖤] 2383=[🖤] 2384=[🖤] 2385=[🖤] 2386=[🖤] 2387=[🖤 ●] 2388=[🔶] 2389=[🔶]
2390=[🖤] 2391=[🖤] 2392=[🖤] 2393=[🖤] 2394=[🖤] 2395=[🖤 🚨] 2396=[🖤] 2397=[🖤 ❤️] 2398=[🖤] 2399=[🖤]
2400=[🖤] 2401=[🖤] 2402=[🖤 🎵] 2403=[🖤 -] 2404=[🖤] 2405=[🖤 ,] 2406=[🖤] 2407=[🖤] 2408=[🖤 😊] 2409=[🖤 😊]
2410=[😊] 2411=[😊 ,] 2412=[😊 .] 2413=[😊 0] 2414=[😊] 2415=[😊] 2416=[😊] 2417=[😊 😊] 2418=[😊 😊] 2419=[😊 😊]
2420=[😊] 2421=[😊 !] 2422=[😊 -] 2423=[😊 ☀️] 2424=[😊] 2425=[😊 !] 2426=[😊] 2427=[😊 😊] 2428=[😊] 2429=[😊 ❤️]
2430=[😊] 2431=[🐱] 2432=[😊] 2433=[😊 ❤️] 2434=[😊] 2435=[😊] 2436=[😊] 2437=[😊 ❤️] 2438=[😊 😊] 2439=[😊]
2440=[😊 ,] 2441=[😊 ❤️] 2442=[😊 😊] 2443=[😊 😊] 2444=[😊 😊] 2445=[😊] 2446=[😊] 2447=[😊 😊] 2448=[😊 😊] 2449=[😊]
2450=[😊 ❤️] 2451=[😊] 2452=[😊 ,] 2453=[😊 ❤️] 2454=[😊] 2455=[😊] 2456=[😊] 2457=[😊] 2458=[😊 ❤️] 2459=[😊 ❤️]
2460=[😊] 2461=[😊 🎵] 2462=[😊] 2463=[😊] 2464=[😊] 2465=[😊] 2466=[😊] 2467=[😊] 2468=[😊] 2469=[😊 ↑TOP]
2470=[😊 😊] 2471=[😊] 2472=[😊] 2473=[😊 😊] 2474=[😊 😊] 2475=[😊 😊] 2476=[😊] 2477=[😊] 2478=[😊] 2479=[😊]
2480=[😊] 2481=[😊] 2482=[😊] 2483=[😊] 2484=[😊] 2485=[😊] 2486=[😊] 2487=[😊] 2488=[😊] 2489=[😊]
2490=[😊] 2491=[😊] 2492=[😊 ,] 2493=[😊] 2494=[😊] 2495=[😊] 2496=[😊] 2497=[😊] 2498=[😊 !] 2499=[😊 .]
2500=[😊 ❤️] 2501=[😊 😊] 2502=[😊] 2503=[😊] 2504=[😊] 2505=[😊] 2506=[😊] 2507=[😊] 2508=[😊] 2509=[😊 🖤]
2510=[😊 😊] 2511=[😊] 2512=[😊] 2513=[😊] 2514=[😊] 2515=[😊] 2516=[😊] 2517=[😊] 2518=[😊] 2519=[😊 ,]
2520=[😊 😊] 2521=[😊] 2522=[😊] 2523=[😊] 2524=[😊] 2525=[😊] 2526=[😊] 2527=[😊] 2528=[😊] 2529=[😊]
2530=[😊 😊] 2531=[😊] 2532=[😊] 2533=[😊 😊] 2534=[😊] 2535=[😊] 2536=[😊] 2537=[😊] 2538=[😊] 2539=[😊 😊]
2540=[😊] 2541=[😊] 2542=[😊 ❤️] 2543=[😊] 2544=[😊 ↑TOP] 2545=[😊] 2546=[😊] 2547=[😊] 2548=[😊] 2549=[😊]
2550=[😊] 2551=[😊 😊] 2552=[😊] 2553=[😊] 2554=[😊] 2555=[😊] 2556=[😊] 2557=[😊] 2558=[😊] 2559=[😊]
2560=[😊 ,] 2561=[😊] 2562=[😊] 2563=[😊] 2564=[😊] 2565=[😊] 2566=[😊] 2567=[😊] 2568=[😊] 2569=[😊]
2570=[😊] 2571=[😊] 2572=[😊] 2573=[😊] 2574=[😊] 2575=[😊] 2576=[😊] 2577=[😊] 2578=[😊] 2579=[😊]
2580=[😊] 2581=[😊 !] 2582=[😊 ,] 2583=[😊 .] 2584=[😊 É] 2585=[😊] 2586=[😊] 2587=[😊] 2588=[😊] 2589=[😊 ↑TOP]
2590=[😊] 2591=[😊] 2592=[🚶 ❤️] 2593=[🚶] 2594=[🚶] 2595=[🚶] 2596=[🚶] 2597=[🚶 ?] 2598=[🚶] 2599=[🚶]

```
2600=[🚧!] 2601=[⚠️] 2602=[🚫] 2603=[🚲] 2604=[🔴] 2605=[🛒] 2606=[🚗] 2607=[🏎️😊] 2608=[ §§ ] 2609=[ §§ ]  
2610=[❗] 2611=[警示教育] 2612=[警示教育] 2613=[警示教育] 2614=[警示教育] 2615=[警示教育] 2616=[警示教育] 2617=[警示教育] 2618=[警示教育] 2619=[警示教育]  
2620=[警示教育] 2621=[警示教育] 2622=[警示教育] 2623=[警示教育] 2624=[警示教育] 2625=[警示教育] 2626=[警示教育] 2627=[警示教育] 2628=[警示教育] 2629=[警示教育]  
2630=[警示教育] 2631=[警示教育] 2632=[警示教育] 2633=[警示教育] 2634=[警示教育] 2635=[警示教育] 2636=[警示教育] 2637=[警示教育] 2638=[警示教育] 2639=[警示教育]  
2640=[警示教育] 2641=[警示教育] 2642=[警示教育] 2643=[警示教育] 2644=[警示教育] 2645=[警示教育] 2646=[警示教育] 2647=[警示教育] 2648=[警示教育] 2649=[警示教育]  
2650=[警示教育] 2651=[警示教育] 2652=[警示教育] 2653=[警示教育] 2654=[警示教育] 2655=[警示教育] 2656=[警示教育] 2657=[警示教育] 2658=[警示教育] 2659=[警示教育]  
2660=[警示教育] 2661=[警示教育] 2662=[警示教育] 2663=[警示教育] 2664=[警示教育] 2665=[警示教育] 2666=[警示教育] 2667=[警示教育] 2668=[警示教育] 2669=[警示教育]  
2670=[警示教育] 2671=[警示教育] 2672=[警示教育] 2673=[警示教育] 2674=[警示教育]
```

```
In [22]: df_twitter = None # Não é mais necessária
```

```
'''  
Função que avalia se um texto contém um Emoticon ou não  
'''  
  
def eh_emoticon(palavra2letras):  
  
    caracs = palavra2letras.strip()  
    for carac in caracs:  
        if ord(carac) > 0xff:  
            return True  
  
    return len(palavra2letras) >= 2 and caracs[0] in '[:8(){}' and caracs[1] in '[:8(){}DP'
```

```
# Filtrando os emoticons efetivos  
emoticons = []  
i = 0  
q = 0  
for peq in pequenas:  
    if eh_emoticon(peq):  
        emoticons.append(paq)  
        print('[{:03} {}]'.format(i, (peq+' ' )[0:2]), end=' ')  
        i += 1  
        q += 1  
        if q == 10:  
            print()  
            q = 0
```

[000 !"] [001 !♫] [002 !😊] [003 "😊"] [004 #맙] [005 #🎥] [006 #😊] [007 #😊] [008 #↗] [009 #😊]
[010 (()) [011 (:)] [012 (📸) [013 ()]) [014 (:)] [015 ();] [016 🎉] [017 ,..] [018 -..] [019 -👉]
[020 -👉] [021 .] [022 ..+] [023 €€] [024 1•] [025 1€] [026 2•] [027 2€] [028 3'] [029 3..]
[030 4'] [031 4€] [032 5€] [033 7"] [034 88] [035 8°] [036 :() [037 :)] [038 ::] [039 :D]
[040 :P] [041 :)] [042 ;() [043 ;)] [044 ;;) [045 ;D] [046 ;P] [047 ?😊] [048 @..] [049 A...]
[050 C"] [051 E..] [052 I👤] [053 K"] [054 0..] [055 T..] [056 a..] [057 e..] [058 n'] [059 o..]
[060 u"] [061 u♥] [062 |👉] [063 ~✿] [064 ~♦] [065 è] [066 a] [067 aé] [068 EU] [069 ♀]
[070 é] [071] [072 -] [073 -e] [074 -] [075 --] [076 -a] [077 '] [078 'A] [079 'O]
[080 'o] [081 'É] [082 'é] [083 '] [084 "] [085 ",] [086 "A] [087 "E] [088 "I] [089 "O]
[090 "a] [091 "o] [092 "É] [093 "é] [094 "...] [095 "] [096 •] [097 •..] [098 ...] [099]
[100 >] [101 *] [102 !!] [103] [104] [105 <] [106 >] [107 €] [108 ^] [109 →]
[110 →] [111 ↴] [112 ↵] [113 =] [114 >] [115 ·] [116 🔍] [117 📊] [118 ➡➡] [119 ⬆⬇]
[120 🕒] [121 ■] [122 ▪] [123 ►] [124 🔍] [125 ►🎧] [126 ►] [127 ●🎥] [128 *] [129 🌟]
[130 📲] [131 📞] [132 ✅] [133 ✅] [134 🌂] [135 🌂] [136 🌂] [137 ➡] [138 🤝] [139 ➡]
[140 💋] [141 💋] [142 🎵] [143 😊] [144 😢] [145 ☺] [146 ☺] [147 ❤] [148 ❤,] [149 ❤:]
[150 ❤!] [151 ❤!] [152 ❤*] [153 ❤] [154 ❤]) [155 ❤️] [156 ❤️] [157 ❤] [158 ❤️] [159 ♦]
[160 ⚽] [161 ⚽] [162 ⚽] [163 ⚽] [164 ⚽] [165 ⚽] [166 ⚽] [167 ⚽] [168 ⚽] [169 ⚽:]
[170 ✅] [171 ✅] [172 ✅] [173 ✅] [174 ✅] [175 ✅] [176 ✅] [177 ✅] [178 ✅,] [179 ✅❤]
[180 🤗] [181 🤗] [182 🤗] [183 🤗] [184 🤗] [185 🤗] [186 🤗] [187 🤗] [188 ✅] [189 ✅...]
[190 ✅] [191 ✅] [192 ✅] [193 ✨] [194 ✨] [195 ✨] [196 ✨] [197 ✨] [198 ✨] [199 ✅]
[200 *] [201 *] [202 ?] [203 !] [204 !] [205 "] [206 !] [207 ??] [208 ❤] [209 ❤]
[210 ❤,] [211 ❤:] [212 ❤️] [213 ❤] [214 ❤✿] [215 ❤️] [216 ❤️] [217 ❤️] [218 ❤️] [219 ❤️]
[220 ❤️] [221 —] [222 →] [223 →...] [224 ➡] [225 → 10] [226 ►] [227 ⇌] [228 ➡] [229 —]
[230] [231 🔍] [232 🔍] [233 🔍] [234 🔍] [235 🔍] [236 »] [237 ππ] [238 —] [239 —]
[240 ε] [241 3] [242 3.] [243 내] [244 사랑] [245] [246 NO] [247 my] [248 to] [249 🎉]
[250 CL] [251 NG] [252 OK] [253 BR] [254 cl] [255 FR] [256 GB] [257 HN] [258 JP] [259 KR]
[260 PY] [261 TR] [262 US] [263 VE] [264 ZA] [265 🌈] [266 🌈] [267 🌈] [268 🌈] [269 🌙]
[270 🌙] [271 🌙] [272 🌙] [273 🌙] [274 🌙!] [275 🌙] [276 🌙] [277 🌙:] [278 🌙] [279 🌙⚡]
[280 🌸] [281 🌸] [282 🌸] [283 🌸:] [284 🌸] [285 🌸] [286 🌸] [287 🌸] [288 🌸:] [289 🌸]
[290 🌸] [291 🌸] [292 🌸] [293 🌸] [294 🌸] [295 🌸] [296 🌸] [297 🌸] [298 🌸] [299 🌸]
[300 🌸] [301 🌸] [302 🌸] [303 🌸] [304 🌸] [305 🌸] [306 🌸] [307 🌸] [308 🌸] [309 🌸]
[310 🎂] [311 🎂] [312 🎂] [313 🎂] [314 🎂] [315 🎂] [316 🎂] [317 🎂] [318 🎂] [319 🎂]
[320 🎂] [321 🎂] [322 🎂] [323 🎂] [324 🎂] [325 🎂] [326 🎂] [327 🎂] [328 🎂] [329 🎂]
[330 🎂] [331 🎂] [332 🎂] [333 🎂] [334 🎂] [335 🎂] [336 🎂] [337 🎂] [338 🎂] [339 🎂]
[340 🎂] [341 🎂] [342 🎂] [343 🎂] [344 🎂] [345 🎂] [346 🎂] [347 🎂] [348 🎂] [349 🎂]
[350 🎂] [351 🎂] [352 🎂] [353 🎂] [354 🎂] [355 🎂] [356 🎂] [357 🎂] [358 🎂] [359 🎂]
[360 🎂] [361 🎂] [362 🎂] [363 🎂] [364 🎂] [365 🎂] [366 🎂] [367 🎂] [368 🎂] [369 🎂]
[370 🎂] [371 🎂] [372 🎂] [373 🎂] [374 🎂] [375 🎂] [376 🎂] [377 🎂/] [378 🎂] [379 🎂]
[380 🎂] [381 🎂] [382 🎂] [383 🎂] [384 🎂] [385 🎂] [386 🎂] [387 🎂] [388 🎂] [389 🎂]
[390 🎂] [391 🎂] [392 🎂] [393 🎂] [394 🎂] [395 🎂] [396 🎂] [397 🎂] [398 🎂] [399 🎂]
[400 🎂] [401 🎂] [402 🎂] [403 🎂] [404 🎂] [405 🎂] [406 🎂] [407 🎂] [408 🎂] [409 🎂...]
[410 🎂] [411 🎂] [412 🎂] [413 🎂] [414 🎂] [415 🎂] [416 🎂] [417 🎂] [418 🎂] [419 🎂]
[420 🎂] [421 🎂] [422 🎂] [423 🎂] [424 🎂!] [425 🎂] [426 🎂] [427 🎂] [428 🎂] [429 🎂]
[430 🎂] [431 🎂] [432 🎂] [433 🎂] [434 🎂] [435 🎂] [436 🎂] [437 🎂] [438 🎂] [439 🎂]
[440 🎂] [441 🎂] [442 🎂] [443 🎂] [444 🎂] [445 🎂] [446 🎂] [447 🎂] [448 🎂] [449 🎂]
[450 🎂] [451 🎂] [452 🎂] [453 🎂] [454 🎂] [455 🎂] [456 🎂] [457 🎂] [458 🎂] [459 🎂]
[460 🎂] [461 🎂] [462 🎂] [463 🎂] [464 🎂] [465 🎂] [466 🎂] [467 🎂] [468 🎂] [469 🎂]
[470 🎂] [471 🎂] [472 🎂] [473 🎂] [474 🎂] [475 🎂] [476 🎂] [477 🎂.] [478 🎂] [479 🎂]
[480 🎂] [481 🎂] [482 🎂] [483 🎂] [484 🎂] [485 🎂] [486 🎂] [487 🎂] [488 🎂] [489 🎂]
[490 🎂] [491 🎂] [492 🎂] [493 🎂] [494 🎂] [495 🎂] [496 🎂] [497 🎂] [498 🎂] [499 🎂]
[500 🎂] [501 🎂] [502 🎂] [503 🎂] [504 🎂] [505 🎂] [506 🎂] [507 🎂] [508 🎂] [509 🎂]
[510 🎂] [511 🎂] [512 🎂] [513 🎂] [514 🎂] [515 🎂] [516 🎂] [517 🎂] [518 🎂] [519 🎂]

```

[520 [hev] [521 [hev] [522 [hev:] [523 [hev:m] [524 [hev:hev] [525 [hev:hev] [526 [hev:hev] [527 [hev:] [528 [hev:hev] [529 [hev:hev]
[530 [hev:hev] [531 [hev:hev] [532 [hev:hev] [533 [hev:hev] [534 [hev:hev] [535 [hev:hev] [536 [hev:hev] [537 [hev:hev] [538 [hev:hev] [539 [hev:hev]
[540 [hev:hev] [541 [hev:hev] [542 [hev:hev] [543 [hev:hev] [544 [hev:hev] [545 [hev:hev] [546 [hev:hev] [547 [hev:hev] [548 [hev:hev] [549 [hev:hev]
[550 [hev:hev] [551 [hev:hev] [552 [hev:hev] [553 [hev:hev] [554 [hev:hev] [555 [hev:hev] [556 [hev:hev] [557 [hev:hev] [558 [hev:hev] [559 [hev:hev]
[560 [hev:hev] [561 [hev:hev] [562 [hev:hev] [563 [hev:hev] [564 [hev:hev] [565 [hev:hev] [566 [hev:hev] [567 [hev:hev] [568 [hev:hev] [569 [hev:hev]
[570 [hev:hev] [571 [hev:hev] [572 [hev:hev] [573 [hev:hev] [574 [hev:hev] [575 [hev:hev] [576 [hev:hev] [577 [hev:hev] [578 [hev:hev] [579 [hev:hev]
[580 [hev:hev] [581 [hev:hev] [582 [hev:hev] [583 [hev:hev] [584 [hev:hev] [585 [hev:hev] [586 [hev:hev] [587 [hev:hev] [588 [hev:hev] [589 [hev:hev]
[590 [hev:hev] [591 [hev:hev] [592 [hev:hev] [593 [hev:hev] [594 [hev:hev] [595 [hev:hev] [596 [hev:hev] [597 [hev:hev] [598 [hev:hev] [599 [hev:hev]
[600 [hev:hev] [601 [hev:hev] [602 [hev:hev] [603 [hev:hev] [604 [hev:hev] [605 [hev:hev] [606 [hev:hev] [607 [hev:hev] [608 [hev:hev] [609 [hev:hev]
[610 [hev:hev] [611 [hev:hev] [612 [hev:hev] [613 [hev:hev] [614 [hev:hev] [615 [hev:hev] [616 [hev:hev] [617 [hev:hev] [618 [hev:hev] [619 [hev:hev]
[620 [hev:hev] [621 [hev:hev] [622 [hev:hev] [623 [hev:hev] [624 [hev:hev] [625 [hev:hev] [626 [hev:hev] [627 [hev:hev] [628 [hev:hev] [629 [hev:hev]
[630 [hev:hev] [631 [hev:hev] [632 [hev:hev] [633 [hev:hev] [634 [hev:hev] [635 [hev:hev] [636 [hev:hev] [637 [hev:hev] [638 [hev:hev] [639 [hev:hev]
[640 [hev:hev] [641 [hev:hev] [642 [hev:hev] [643 [hev:hev] [644 [hev:hev] [645 [hev:hev] [646 [hev:hev] [647 [hev:hev] [648 [hev:hev] [649 [hev:hev]
[650 [hev:hev] [651 [hev:hev] [652 [hev:hev] [653 [hev:hev] [654 [hev:hev] [655 [hev:hev] [656 [hev:hev] [657 [hev:hev] [658 [hev:hev] [659 [hev:hev]
[660 [hev:hev] [661 [hev:hev] [662 [hev:hev] [663 [hev:hev] [664 [hev:hev] [665 [hev:hev] [666 [hev:hev] [667 [hev:hev] [668 [hev:hev] [669 [hev:hev]
[670 [hev:hev] [671 [hev:hev] [672 [hev:hev] [673 [hev:hev] [674 [hev:hev] [675 [hev:hev] [676 [hev:hev] [677 [hev:hev] [678 [hev:hev] [679 [hev:hev]
[680 [hev:hev] [681 [hev:hev] [682 [hev:hev] [683 [hev:hev] [684 [hev:hev] [685 [hev:hev] [686 [hev:hev] [687 [hev:hev] [688 [hev:hev] [689 [hev:hev]
[690 [hev:hev] [691 [hev:hev] [692 [hev:hev] [693 [hev:hev] [694 [hev:hev] [695 [hev:hev] [696 [hev:hev] [697 [hev:hev] [698 [hev:hev] [699 [hev:hev]
[700 [hev:hev] [701 [hev:hev] [702 [hev:hev] [703 [hev:hev] [704 [hev:hev] [705 [hev:hev] [706 [hev:hev] [707 [hev:hev] [708 [hev:hev] [709 [hev:hev]
[710 [hev:hev] [711 [hev:hev] [712 [hev:hev] [713 [hev:hev] [714 [hev:hev] [715 [hev:hev] [716 [hev:hev] [717 [hev:hev] [718 [hev:hev] [719 [hev:hev]
[720 [hev:hev] [721 [hev:hev] [722 [hev:hev] [723 [hev:hev] [724 [hev:hev] [725 [hev:hev] [726 [hev:hev] [727 [hev:hev] [728 [hev:hev] [729 [hev:hev]
[730 [hev:hev] [731 [hev:hev] [732 [hev:hev] [733 [hev:hev] [734 [hev:hev] [735 [hev:hev] [736 [hev:hev] [737 [hev:hev] [738 [hev:hev] [739 [hev:hev]
[740 [hev:hev] [741 [hev:hev] [742 [hev:hev] [743 [hev:hev] [744 [hev:hev] [745 [hev:hev] [746 [hev:hev] [747 [hev:hev] [748 [hev:hev] [749 [hev:hev]
[750 [hev:hev] [751 [hev:hev] [752 [hev:hev] [753 [hev:hev] [754 [hev:hev] [755 [hev:hev] [756 [hev:hev] [757 [hev:hev] [758 [hev:hev] [759 [hev:hev]
[760 [hev:hev] [761 [hev:hev] [762 [hev:hev] [763 [hev:hev] [764 [hev:hev] [765 [hev:hev] [766 [hev:hev] [767 [hev:hev] [768 [hev:hev] [769 [hev:hev]
[770 [hev:hev] [771 [hev:hev] [772 [hev:hev] [773 [hev:hev] [774 [hev:hev] [775 [hev:hev] [776 [hev:hev] [777 [hev:hev] [778 [hev:hev] [779 [hev:hev]
[780 [hev:hev] [781 [hev:hev] [782 [hev:hev] [783 [hev:hev] [784 [hev:hev] [785 [hev:hev] [786 [hev:hev] [787 [hev:hev] [788 [hev:hev] [789 [hev:hev]
[790 [hev:hev] [791 [hev:hev] [792 [hev:hev] [793 [hev:hev] [794 [hev:hev] [795 [hev:hev] [796 [hev:hev] [797 [hev:hev] [798 [hev:hev] [799 [hev:hev]
[800 [hev:hev] [801 [hev:hev] [802 [hev:hev] [803 [hev:hev] [804 [hev:hev] [805 [hev:hev] [806 [hev:hev] [807 [hev:hev] [808 [hev:hev] [809 [hev:hev]
[810 [hev:hev] [811 [hev:hev] [812 [hev:hev] [813 [hev:hev] [814 [hev:hev] [815 [hev:hev] [816 [hev:hev] [817 [hev:hev] [818 [hev:hev] [819 [hev:hev]
[820 [hev:hev] [821 [hev:hev] [822 [hev:hev] [823 [hev:hev] [824 [hev:hev] [825 [hev:hev] [826 [hev:hev] [827 [hev:hev] [828 [hev:hev] [829 [hev:hev]
[830 [hev:hev] [831 [hev:hev] [832 [hev:hev] [833 [hev:hev] [834 [hev:hev] [835 [hev:hev] [836 [hev:hev] [837 [hev:hev] [838 [hev:hev] [839 [hev:hev]
[840 [hev:hev] [841 [hev:hev] [842 [hev:hev] [843 [hev:hev] [844 [hev:hev] [845 [hev:hev] [846 [hev:hev] [847 [hev:hev] [848 [hev:hev] [849 [hev:hev]
[850 [hev:hev] [851 [hev:hev] [852 [hev:hev] [853 [hev:hev] [854 [hev:hev] [855 [hev:hev] [856 [hev:hev] [857 [hev:hev] [858 [hev:hev] [859 [hev:hev]
[860 [hev:hev] [861 [hev:hev] [862 [hev:hev] [863 [hev:hev] [864 [hev:hev] [865 [hev:hev] [866 [hev:hev] [867 [hev:hev] [868 [hev:hev] [869 [hev:hev]
[870 [hev:hev] [871 [hev:hev] [872 [hev:hev] [873 [hev:hev] [874 [hev:hev] [875 [hev:hev] [876 [hev:hev] [877 [hev:hev] [878 [hev:hev] [879 [hev:hev]
[880 [hev:hev] [881 [hev:hev] [882 [hev:hev] [883 [hev:hev] [884 [hev:hev] [885 [hev:hev] [886 [hev:hev] [887 [hev:hev] [888 [hev:hev] [889 [hev:hev]
[890 [hev:hev] [891 [hev:hev] [892 [hev:hev] [893 [hev:hev] [894 [hev:hev] [895 [hev:hev] [896 [hev:hev] [897 [hev:hev] [898 [hev:hev] [899 [hev:hev]
[900 [hev:hev] [901 [hev:hev] [902 [hev:hev] [903 [hev:hev] [904 [hev:hev] [905 [hev:hev] [906 [hev:hev] [907 [hev:hev] [908 [hev:hev] [909 [hev:hev]
[910 [hev:hev] [911 [hev:hev] [912 [hev:hev] [913 [hev:hev] [914 [hev:hev] [915 [hev:hev] [916 [hev:hev] [917 [hev:hev]

```

Agora, manualmente, vamos definir os emoticons positivos e negativos

```
In [25]: # Uma função para popular a lista a partir de ranges
def adic_emoticons(dest, lista):
    for item in lista:
        dest.append(item)
    return
```

```
In [26]: emoticons_positivos = []
adic_emoticons(emoticons_positivos, emoticons[6:9])
```

```
adic_emoticons(emoticons_positivos, emoticons[11:12])
adic_emoticons(emoticons_positivos, emoticons[37:38])
adic_emoticons(emoticons_positivos, emoticons[39:42])
adic_emoticons(emoticons_positivos, emoticons[43:48])
adic_emoticons(emoticons_positivos, emoticons[145:159])
adic_emoticons(emoticons_positivos, emoticons[177:186])
adic_emoticons(emoticons_positivos, emoticons[192:200])
adic_emoticons(emoticons_positivos, emoticons[210:220])
adic_emoticons(emoticons_positivos, emoticons[347:365])
adic_emoticons(emoticons_positivos, emoticons[385:394])
adic_emoticons(emoticons_positivos, emoticons[405:407])
adic_emoticons(emoticons_positivos, emoticons[412:435])
adic_emoticons(emoticons_positivos, emoticons[436:444])
adic_emoticons(emoticons_positivos, emoticons[450:455])
adic_emoticons(emoticons_positivos, emoticons[474:509])
adic_emoticons(emoticons_positivos, emoticons[510:515])
adic_emoticons(emoticons_positivos, emoticons[517:531])
adic_emoticons(emoticons_positivos, emoticons[532:537])
adic_emoticons(emoticons_positivos, emoticons[545:555])
adic_emoticons(emoticons_positivos, emoticons[558:561])
adic_emoticons(emoticons_positivos, emoticons[577:579])
adic_emoticons(emoticons_positivos, emoticons[647:719])
adic_emoticons(emoticons_positivos, emoticons[738:755])
adic_emoticons(emoticons_positivos, emoticons[796:801])
adic_emoticons(emoticons_positivos, emoticons[802:808])
adic_emoticons(emoticons_positivos, emoticons[816:836])
adic_emoticons(emoticons_positivos, emoticons[850:851])
adic_emoticons(emoticons_positivos, emoticons[865:884])
adic_emoticons(emoticons_positivos, emoticons[893:897])
adic_emoticons(emoticons_positivos, emoticons[909:910])
adic_emoticons(emoticons_positivos, emoticons[916:917])
```

In [27]: `exibir_pequenas(emoticons_positivos)`

```

0000=[#😊 ] 0001=[#😊 ] 0002=[#👉 ] 0003=[(:] 0004=[:]) 0005=[:D] 0006=[:P] 0007=[:] 0008=[;) 0009=[;;]
0010=[;D] 0011=[;P] 0012=[?😊] 0013=[@] 0014=[@👉] 0015=[♥ ] 0016=[♥,] 0017=[♥:] 0018=[♥!] 0019=[♥♥]
0020=[♥+] 0021=[♥ ] 0022=[♥] 0023=[♥👉] 0024=[♥♥] 0025=[♥ ] 0026=[♥😊] 0027=[👉 ] 0028=[👉,] 0029=[👉♥]
0030=[👉 ] 0031=[👉 ] 0032=[👉 ] 0033=[👉 ] 0034=[👉👉] 0035=[👉😊] 0036=[👉 ] 0037=[👉 ] 0038=[👉👉] 0039=[👉👉]
0040=[👉♥] 0041=[👉😊] 0042=[👉😊] 0043=[👉] 0044=[♥,] 0045=[♥:] 0046=[♥♥] 0047=[♥ ] 0048=[♥✿] 0049=[♥✿]
0050=[♥✿] 0051=[♥✿] 0052=[♥✿] 0053=[♥✿] 0054=[✿] 0055=[✿A] 0056=[✿✿] 0057=[✿ ] 0058=[✿✿A] 0059=[✿✿✿]
0060=[✿✿] 0061=[✿] 0062=[✿✿] 0063=[🎹✿] 0064=[✿✿] 0065=[✿✿] 0066=[✿✿] 0067=[✿✿] 0068=[✿✿] 0069=[✿✿✿]
0070=[✿✿] 0071=[✿✿] 0072=[✿✿] 0073=[✿✿] 0074=[✿✿] 0075=[✿✿] 0076=[✿✿] 0077=[✿✿] 0078=[✿✿] 0079=[✿✿]
0080=[✿✿] 0081=[✿✿] 0082=[✿✿] 0083=[✿✿] 0084=[✿✿] 0085=[✿✿] 0086=[✿✿] 0087=[✿✿] 0088=[✿✿] 0089=[✿✿✿]
0090=[✿✿] 0091=[✿✿] 0092=[✿✿] 0093=[✿✿] 0094=[✿✿] 0095=[✿✿!] 0096=[✿✿] 0097=[✿✿] 0098=[✿✿] 0099=[✿✿✿]
0100=[✿✿] 0101=[✿✿] 0102=[✿✿] 0103=[✿✿] 0104=[✿✿] 0105=[✿✿] 0106=[✿✿] 0107=[✿✿] 0108=[✿✿] 0109=[✿✿✿]
0110=[✿✿] 0111=[✿✿] 0112=[✿✿] 0113=[✿✿] 0114=[✿✿] 0115=[✿✿] 0116=[✿✿] 0117=[✿✿] 0118=[✿✿] 0119=[✿✿✿]
0120=[✿✿] 0121=[✿✿] 0122=[✿✿] 0123=[✿✿] 0124=[✿✿] 0125=[✿✿] 0126=[✿✿] 0127=[✿✿] 0128=[✿✿] 0129=[✿✿]
0130=[✿✿] 0131=[✿✿] 0132=[✿✿] 0133=[✿✿] 0134=[✿✿] 0135=[✿✿] 0136=[✿✿] 0137=[✿✿] 0138=[✿✿] 0139=[✿✿]
0140=[✿✿] 0141=[✿✿] 0142=[✿✿] 0143=[✿✿] 0144=[✿✿] 0145=[✿✿] 0146=[✿✿] 0147=[✿✿] 0148=[✿✿] 0149=[✿✿]
0150=[✿✿] 0151=[✿✿] 0152=[✿✿] 0153=[✿✿] 0154=[✿✿] 0155=[✿✿] 0156=[✿✿] 0157=[✿✿] 0158=[✿✿] 0159=[✿✿]
0160=[✿✿] 0161=[✿✿] 0162=[✿✿] 0163=[✿✿] 0164=[✿✿] 0165=[✿✿] 0166=[✿✿] 0167=[✿✿] 0168=[✿✿] 0169=[✿✿]
0170=[✿✿] 0171=[✿✿] 0172=[✿✿] 0173=[✿✿] 0174=[✿✿] 0175=[✿✿] 0176=[✿✿] 0177=[✿✿] 0178=[✿✿] 0179=[✿✿✿]
0180=[✿✿] 0181=[✿✿] 0182=[✿✿] 0183=[✿✿] 0184=[✿✿] 0185=[✿✿] 0186=[✿✿] 0187=[✿✿] 0188=[✿✿!] 0189=[✿✿]
0190=[✿✿] 0191=[✿✿] 0192=[✿✿] 0193=[✿✿] 0194=[✿✿] 0195=[✿✿] 0196=[✿✿] 0197=[✿✿] 0198=[✿✿] 0199=[✿✿]
0200=[✿✿] 0201=[✿✿] 0202=[✿✿] 0203=[✿✿] 0204=[✿✿] 0205=[✿✿] 0206=[✿✿] 0207=[✿✿] 0208=[✿✿] 0209=[✿✿]
0210=[✿✿] 0211=[✿✿] 0212=[✿✿] 0213=[✿✿] 0214=[✿✿!] 0215=[✿✿] 0216=[✿✿] 0217=[✿✿] 0218=[✿✿] 0219=[✿✿]
0220=[✿✿] 0221=[✿✿] 0222=[✿✿] 0223=[✿✿] 0224=[✿✿] 0225=[✿✿] 0226=[✿✿] 0227=[✿✿] 0228=[✿✿] 0229=[✿✿]
0230=[✿✿] 0231=[✿✿] 0232=[✿✿] 0233=[✿✿] 0234=[✿✿] 0235=[✿✿] 0236=[✿✿] 0237=[✿✿] 0238=[✿✿] 0239=[✿✿]
0240=[✿✿] 0241=[✿✿] 0242=[✿✿] 0243=[✿✿] 0244=[✿✿] 0245=[✿✿] 0246=[✿✿] 0247=[✿✿] 0248=[✿✿] 0249=[✿✿]
0250=[✿✿] 0251=[✿✿] 0252=[✿✿] 0253=[✿✿] 0254=[✿✿] 0255=[✿✿] 0256=[✿✿] 0257=[✿✿] 0258=[✿✿] 0259=[✿✿]
0260=[✿✿] 0261=[✿✿] 0262=[✿✿] 0263=[✿✿] 0264=[✿✿] 0265=[✿✿] 0266=[✿✿] 0267=[✿✿!] 0268=[✿✿!] 0269=[✿✿]
0270=[✿✿] 0271=[✿✿] 0272=[✿✿] 0273=[✿✿] 0274=[✿✿] 0275=[✿✿] 0276=[✿✿] 0277=[✿✿] 0278=[✿✿] 0279=[✿✿]
0280=[✿✿] 0281=[✿✿] 0282=[✿✿] 0283=[✿✿] 0284=[✿✿] 0285=[✿✿] 0286=[✿✿] 0287=[✿✿] 0288=[✿✿] 0289=[✿✿]
0290=[✿✿] 0291=[✿✿] 0292=[✿✿] 0293=[✿✿] 0294=[✿✿] 0295=[✿✿] 0296=[✿✿] 0297=[✿✿] 0298=[✿✿] 0299=[✿✿]
0300=[👉!] 0301=[👉!] 0302=[👉,] 0303=[👉,] 0304=[👉É] 0305=[👉] 0306=[👉] 0307=[👉] 0308=[👉] 0309=[👉↑]
0310=[👉] 0311=[👉] 0312=[👉!] 0313=[👉!] 0314=[👉!] 0315=[👉!] 0316=[👉!] 0317=[👉!] 0318=[👉!] 0319=[👉]
0320=[👉] 0321=[👉] 0322=[👉!] 0323=[👉!] 0324=[👉!] 0325=[👉!] 0326=[👉!] 0327=[👉!] 0328=[👉!] 0329=[👉]
0330=[👉!] 0331=[👉!] 0332=[👉!] 0333=[👉!] 0334=[👉!] 0335=[👉!] 0336=[👉!] 0337=[👉!] 0338=[👉!]

```

In [28]:

```

emoticons_negativos = []

adic_emoticons(emoticons_negativos, emoticons[2:3])
adic_emoticons(emoticons_negativos, emoticons[14:17])
adic_emoticons(emoticons_negativos, emoticons[36:37])
adic_emoticons(emoticons_negativos, emoticons[42:43])
adic_emoticons(emoticons_negativos, emoticons[143:145])
adic_emoticons(emoticons_negativos, emoticons[220:221])
adic_emoticons(emoticons_negativos, emoticons[411:412])
adic_emoticons(emoticons_negativos, emoticons[435:436])
adic_emoticons(emoticons_negativos, emoticons[461:464])
adic_emoticons(emoticons_negativos, emoticons[467:468])
adic_emoticons(emoticons_negativos, emoticons[509:510])
adic_emoticons(emoticons_negativos, emoticons[515:517])
adic_emoticons(emoticons_negativos, emoticons[531:532])
adic_emoticons(emoticons_negativos, emoticons[544:545])
adic_emoticons(emoticons_negativos, emoticons[579:580])

```

```
adic_emoticons(emoticons_negativos, emoticons[719:738])
adic_emoticons(emoticons_negativos, emoticons[755:784])
adic_emoticons(emoticons_negativos, emoticons[793:796])
adic_emoticons(emoticons_negativos, emoticons[801:802])
adic_emoticons(emoticons_negativos, emoticons[851:857])
adic_emoticons(emoticons_negativos, emoticons[859:860])
adic_emoticons(emoticons_negativos, emoticons[884:885])
adic_emoticons(emoticons_negativos, emoticons[887:893])
adic_emoticons(emoticons_negativos, emoticons[897:899])
```

In [29]: `exibir_pequenas(emoticons_negativos)`

```
0000=[!😊] 0001=[:) :] 0002=[;) ] 0003=[()!] 0004=[(:) ] 0005=[;() ] 0006=[😊] 0007=[😊] 0008=[❤️!] 0009=[👉!] 
0010=[👉] 0011=[😊] 0012=[😡] 0013=[😡😡] 0014=[😡] 0015=[❤️😊] 0016=[❤️😊] 0017=[❤️!] 0018=[❤️😊] 0019=[😊]
0020=[❗] 0021=[😊] 0022=[😊] 0023=[😊] 0024=[😊] 0025=[😊] 0026=[😊!] 0027=[😊!] 0028=[😊!] 0029=[😊!] 
0030=[😊!] 0031=[😊!] 0032=[😊!] 0033=[😊!] 0034=[😊!] 0035=[😊!] 0036=[😊!] 0037=[😊,] 0038=[😊,] 0039=[😊!]
0040=[😊!] 0041=[😊!] 0042=[😊!] 0043=[😊!] 0044=[😊!] 0045=[😊!] 0046=[😊!] 0047=[😊,] 0048=[😊!] 0049=[😊!]
0050=[😊!] 0051=[😊!] 0052=[😊!] 0053=[😊!] 0054=[😊!] 0055=[😊!] 0056=[😊!] 0057=[😊!] 0058=[😊!] 0059=[😊!]
0060=[😊!] 0061=[😊!] 0062=[😊!] 0063=[!] 0064=[!] 0065=[!] 0066=[!] 0067=[!] 0068=[!] 0069=[!]
0070=[😊!] 0071=[😊!] 0072=[😊!] 0073=[😊!] 0074=[😊!] 0075=[😊!] 0076=[😊!] 0077=[😊!] 0078=[😊!] 0079=[😊!]
0080=[😊!] 0081=[😊!] 0082=[😊!] 0083=[😊!] 0084=[😊!] 0085=[😊!] 0086=[😊!] 0087=[😊!] 0088=[😊!]
```

Agora, vamos definir como stop words todas as palavras pequenas que não são os emoticons que classificamos/

In [30]:

```
# Preenchendo a lista de stop-words
stop_words = []

for texto in pequenas:
    if not (texto in emoticons_positivos or texto in emoticons_negativos):
        stop_words.append(texto)

exibir_pequenas(stop_words[:200])
```

```

0000=[!] 0001=[!!] 0002=[!?] 0003=[!] 0004=[!♫] 0005=["] 0006=[",] 0007=[-] 0008=["0] 0009=["3]
0010=["5] 0011=["A] 0012=["E] 0013=["I] 0014=["N] 0015=["O] 0016=["Y] 0017=["a] 0018=["e] 0019=["n"]
0020=["o] 0021=["q] 0022=["À] 0023=[É] 0024=[é] 0025=[ó] 0026=[º] 0027=[#] 0028=[#1] 0029=[#2]
0030=[#3] 0031=[#4] 0032=[#7] 0033=[#8] 0034=[#R] 0035=[#a] 0036=[#e] 0037=[#o] 0038=[#p] 0039=[#é]
0040=[#¤] 0041=[#¤] 0042=[#¤] 0043=[\$] 0044=[$$] 0045=[$,] 0046=[$2] 0047=[$5] 0048=[%] 0049=[%;]
0050=[' ] 0051=[''] 0052=[''] 0053=['3] 0054=['7] 0055=['A] 0056=['I] 0057=['0] 0058=['a] 0059=['e]
0060=['o] 0061=['É] 0062=['à] 0063=['é] 0064=[( ] 0065=[((] 0066=[(,) 0067=[(.) 0068=[(/] 0069=[(1]
0070=[(3] 0071=[(7] 0072=[(9] 0073=[(@] 0074=[(A] 0075=[(E] 0076=[(O] 0077=[(Q] 0078=[(a] 0079=[(b]
0080=[(e] 0081=[(i] 0082=[(n] 0083=[(o] 0084=[(q] 0085=[(É] 0086=[(é] 0087=[(¤] 0088=[( )] 0089=[( )]
0090=[))] 0091=[,),] 0092=[.).] 0093=[*] 0094=[**] 0095=[*-] 0096=[*A] 0097=[*E] 0098=[*s] 0099=[+]
0100=[+++] 0101=[+,] 0102=[+.] 0103=[+1] 0104=[+5] 0105=[+9] 0106=[+:] 0107=[+o] 0108=[, ,] 0109=[,,]
0110=[,?] 0111=[,e] 0112=[,n] 0113=[,o] 0114=[,...] 0115=[-] 0116=[-( )] 0117=[-(-) 0118=[--] 0119=[-1]
0120=[-5] 0121=[-:] 0122=[-B] 0123=[-e] 0124=[-f] 0125=[-p] 0126=[-q] 0127=[-È] 0128=[-... ] 0129=[- ¤]
0130=[- ¤] 0131=[. ] 0132=[!.] 0133=[..] 0134=[.g] 0135=[. ] 0136=[.*] 0137=[/ ] 0138=[//] 0139=[/:]
0140=[/p] 0141=[0 ] 0142=[0%] 0143=[0)] 0144=[0,] 0145=[0.] 0146=[0/] 0147=[01] 0148=[02] 0149=[03]
0150=[04] 0151=[05] 0152=[06] 0153=[07] 0154=[08] 0155=[09] 0156=[0€] 0157=[1 ] 0158=[1%] 0159=[1)]
0160=[1,] 0161=[1-] 0162=[1.] 0163=[10] 0164=[11] 0165=[12] 0166=[13] 0167=[14] 0168=[15] 0169=[16]
0170=[17] 0171=[18] 0172=[19] 0173=[1:] 0174=[1?] 0175=[1K] 0176=[1M] 0177=[1T] 0178=[1a] 0179=[1d]
0180=[1h] 0181=[1m] 0182=[1o] 0183=[1p] 0184=[1s] 0185=[1x] 0186=[1¤] 0187=[1ª] 0188=[1°] 0189=[1º]
0190=[1•] 0191=[1€] 0192=[2 ] 0193=[2!] 0194=[2%] 0195=[2'] 0196=[2)] 0197=[2,] 0198=[2-] 0199=[2.]

```

Vamos redefinir a função que retorna se um token é emoticon ou não para utilizar as duas listas

```
In [31]: pequenas = None # Não é mais necessário
emoticons = None # Não é mais necessário
```

```
In [32]: '''
Retorna se um token é um emoticon pesquisando nas listas de emoticons positivos e negativos
'''

def eh_emoticon(palavra):
    return palavra in emoticons_positivos or palavra in emoticons_negativos
```

```
In [33]: # Pequeno teste

print(':' ,->,eh_emoticon(':'))
print(':( ,->,eh_emoticon(':('))
print('?? ,->,eh_emoticon('??'))

:) -> True
:( -> True
?? -> False
```

De fato, podemos até testar o grau de acerto ao avaliar os textos apenas pela presença de emoticons positivos ou negativos...

```
In [34]: '''
Criando nossa função de classificação customizada...
'''

def previsao_emoticons(texto):

    palavras = texto.split()
```

```

positivos = 0
negativos = 0
for palavra in palavras:
    if palavra in emoticons_positivos:
        positivos += 1
    elif palavra in emoticons_negativos:
        negativos += 1

resultado = positivos - negativos

if resultado < 0:
    return 0
elif resultado > 0:
    return 1
return 2

```

In [35]: df_teste = df_analise.copy()
df_teste['predito'] = df_teste['tweet_text'].apply(lambda x : previsao_emoticons(x))

In [36]: df_teste[['sentiment','predito']].value_counts(normalize=True) * 100

Out[36]:

sentiment	predito	value
2	2	32.522510
1	1	27.231518
0	0	26.733630
	2	6.612502
1	2	6.091081
2	1	0.609356
0	1	0.121376
2	0	0.050780
1	0	0.027248

dtype: float64

In [37]: print('% de acerto:', len(df_teste.loc[df_teste['predito'] == df_teste['sentiment']]) / len(df_teste) * 100)
% de acerto: 86.48765806715299

In [38]: df_teste = None # Não é mais necessário

Visualização por gráficos

Vamos criar uma função para apresentar o significado de 'sentiment' para melhor visualização nos gráficos.

In [39]:

```

'''
Retorna o texto correspondente ao código
'''

descr_sentim = {0:'negativo', 1:'positivo', 2:'neutro'}

```

```
def descr_sentimento(codigo):
    return descr_sentim[codigo]
```

```
In [40]: descr_sentimento(2)
```

```
Out[40]: 'neutro'
```

```
In [41]: df_analise['descr_sentim'] = df_analise['sentiment'].apply(descr_sentimento)
```

```
In [42]: df_analise.sample(5)
```

```
Out[42]:
```

	tweet_text	tweet_date	sentiment	query_used	descr_sentim
1930	#Fato 🤪 Sofrer por amor é o erro né, mas quando...	2018-09-22 10:49:42+00:00	2	#fato	neutro
69284	@coxxa Vou te falar que tá quase igual, com a ...	2018-10-03 17:35:18+00:00	1	:)	positivo
26237	Suplente de Alvaro Dias, empresário Joel Maluc...	2018-09-14 16:28:02+00:00	2	folha	neutro
84083	https://t.co/VznxpRT25w → No dia 19/09 a Sala ...	2018-09-11 20:50:10+00:00	2	#novidade	neutro
32862	eu e a @brunalopesribau às 4 da manhã no carro...	2018-10-01 15:07:26+00:00	1	:)	positivo

Nota: Temos várias referências de pessoas/empresas/grupos

Em postagens no Twitter é comum referenciar ou mencionar pessoas ou grupos na forma @\<destinatário>.

Podemos criar uma função para "limpar" essas referências, e se alguma posteriormente for importante manter, bastará adaptar a função.

Como processaremos texto nesse modelo, esse tratamento reduzirá a quantidade de palavras, sem influenciar negativamente a avaliação de sentimento.

```
In [43]: import re
```

```
def remove_referencias(texto):
    return re.sub(r'@\w+', '', texto)
```

```
In [44]: remove_referencias('Este é um teste @referencia de remoção de @qualquer_pessoa em tweets.')
```

```
Out[44]: 'Este é um teste de remoção de em tweets.'
```

```
In [45]: from matplotlib import pyplot as plt
from wordcloud import WordCloud, STOPWORDS
```

```
In [46]: '''
Plota o gráfico de nuvem de palavras de um texto
'''
def plot_wordcloud(textos):

    plt.figure(figsize=(12, 10))
```

```

texto = ' '.join(textos.tolist())

WC = WordCloud(width=1000, height=500, max_words=500, min_font_size=5, min_word_length=3)
tokens = WC.generate(texto)

plt.imshow(tokens, interpolation='bilinear')
plt.show()

```

Vamos plotar o gráfico de apenas um sentimento para avaliar

In [47]: `plot_wordcloud(df_analise.loc[df_analise['sentiment'] == 2, 'tweet_text'])`



NOTA: A presença de https pode indicar trechos HTML nos tweets (BeautifulSoup?) ou não... Vamos avaliar.

In [48]:

```

# Analisando HTTPS
c = 1
for texto in df_analise.loc[df_analise['tweet_text'].str.contains('https'), 'tweet_text']:
    print(texto)
    c += 1
    if c == 10:
        break

```

Gostei de um vídeo @YouTube <https://t.co/TBrLMJQnt8> dia vai sentir minha falta :(
Mais uma luta, mais uma Vitória. 🎉 #Capitão20123 #Trabalho #AlutaContinua #Eleições #DeputadoEstadual <https://t.co/57mQzChM3C>
Candidatos intensificam campanha nas redes sociais <https://t.co/Dyjd0c5cjM>
Gostei de um vídeo @YouTube <https://t.co/BjfEnuT1Wk> QUE VAMO TESTAR ESSA LIVE :)
Quem mata as pessoas não são as armas, diz Flávio Bolsonaro <https://t.co/60jbf6GXFG>
PT entra com reclamação no CNJ contra Moro após divulgação da delação de Palocci <https://t.co/TRyDlUX7Pa>
Veja o que é #FATO ou #FAKE na entrevista de Ana Amélia à GloboNews <https://t.co/cAYD7nGlFP>
#Vaga para Auxiliar de Expedição detalhes e envio de cv em: <https://t.co/3S3KfjR7CR> #oportunidade #emprego
Explosões de gás causam incêndios em dezenas de casas na região de Boston <https://t.co/h2leRmDhIE>

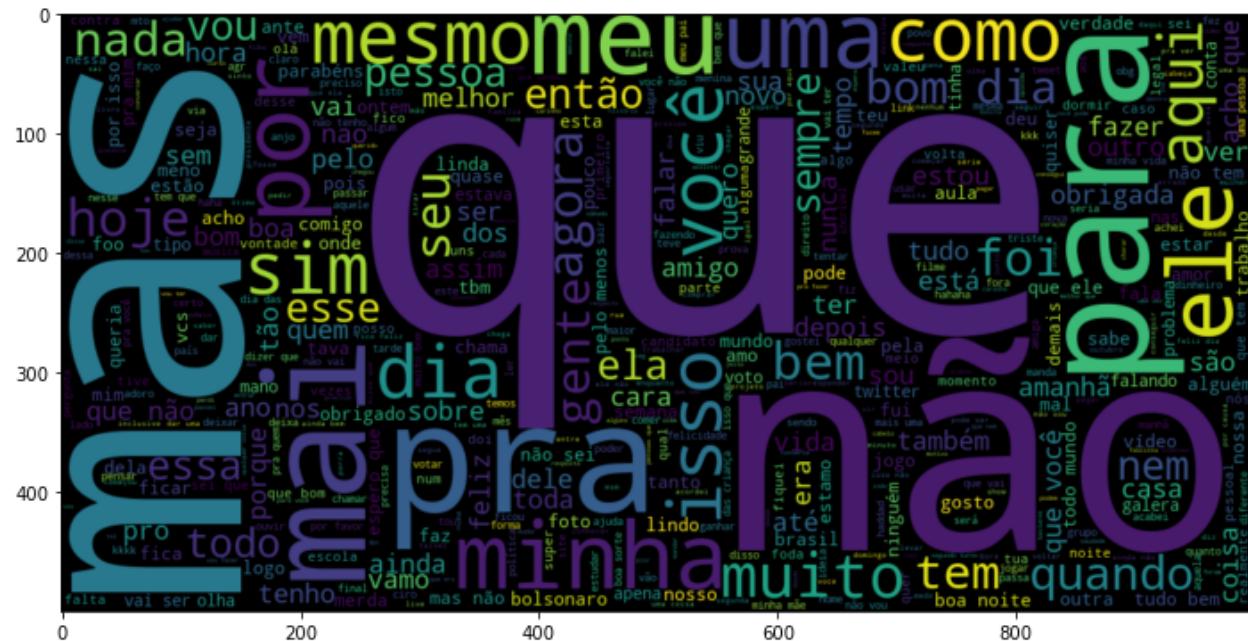
Nota: Não temos tags HTML, mas referências para sites. Vamos programar sua remoção, junto com a remoção de stop words e a normalização para minúsculas.

In [49]:

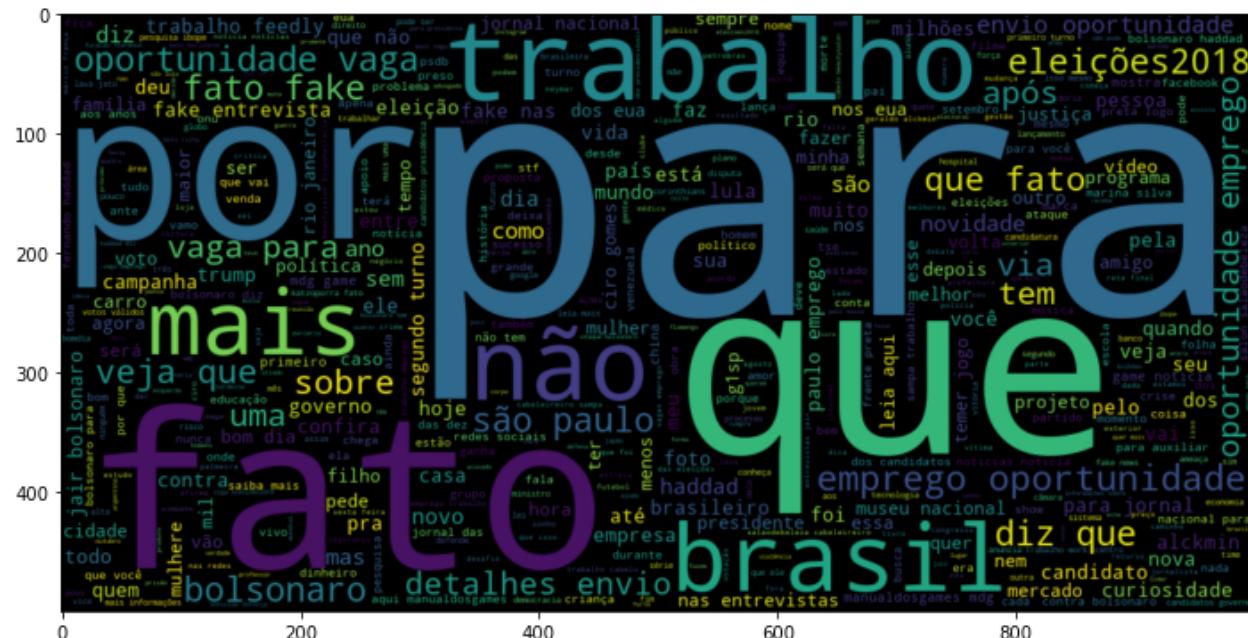
```
'''  
Limpa referências para sites e referências @... dos tweets  
def limpar_tweet(texto):  
  
    # Converte palavras em minúsculas  
    # Remove stop words  
    # Remove as referências @... dos tweets  
    # Remove sites http, https e www  
    palavras = texto.split(' ')  
    novo_texto = []  
    for palavra in palavras:  
        if not palavra in stop_words:  
            palavra = palavra.lower()  
            if palavra.find('http') != 0 and palavra.find('www') != 0:  
                novo_texto.append(re.sub(r'@\w+', '', palavra))  
  
    return ' '.join(novo_texto)
```

In [50]:

```
'''  
Nova versão da função de tratamento dos dados, aplicando a função limpar_tweet()  
def tratamento(df):  
    df = df.copy()  
    try:  
        # Remove a coluna 'id'  
        df = df.drop('id', axis=1)  
    except:  
        # Ignorar: O DataFrame pode já ter sido tratado  
        pass  
  
    # Elimina registros sem texto na variável 'tweet_text'  
    try:  
        df.dropna(inplace=True, subset=['tweet_text'])  
    except:  
        # Ignorar: O DataFrame pode já ter sido tratado  
        pass
```

*** neutro ***

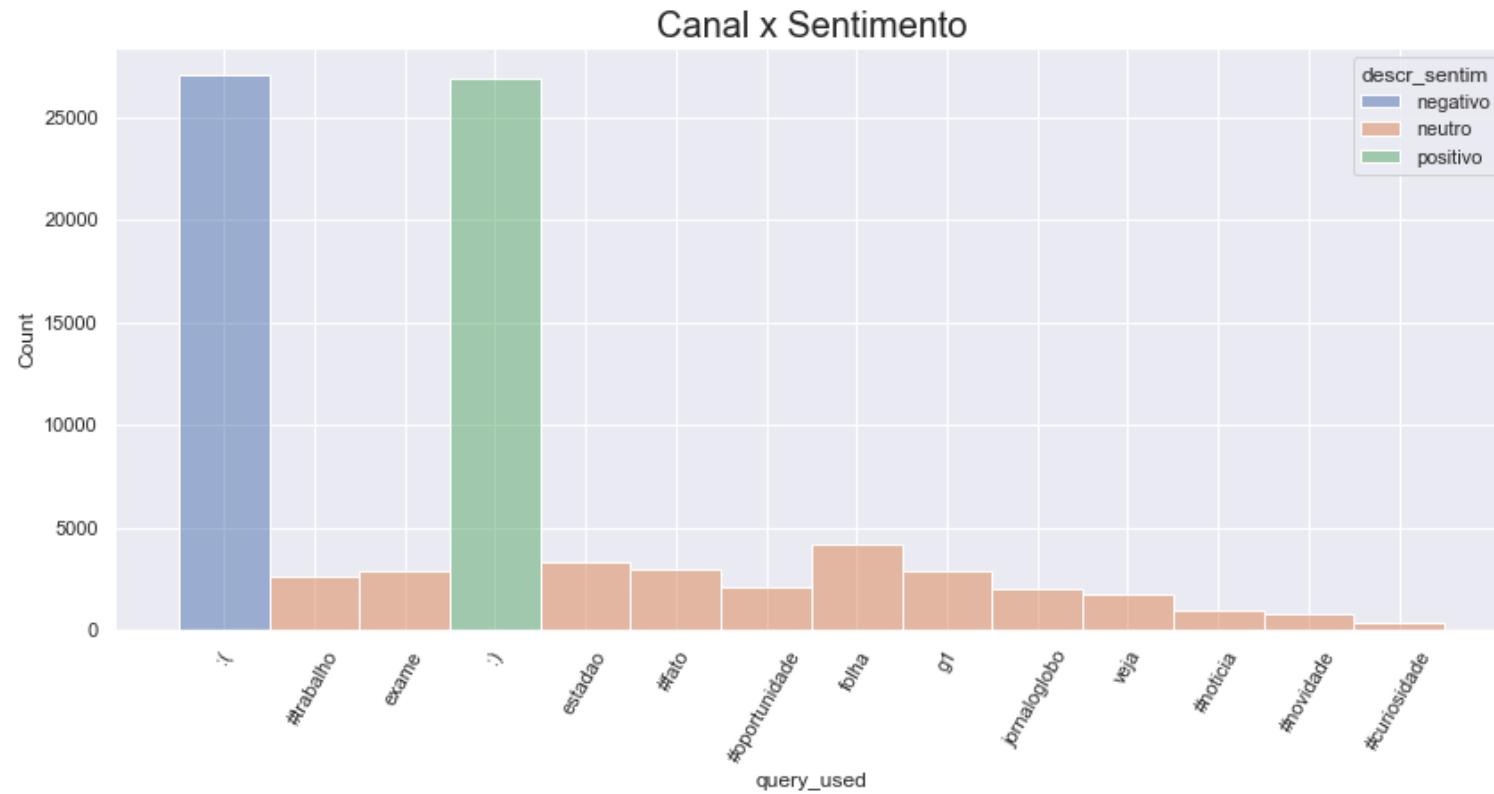


Algumas análises adicionais de exploração.

```
In [53]: import seaborn as sns
```

```
In [54]: sns.set_theme()  
sns.set()
```

```
In [55]: plt.figure(figsize=(14,6))  
sns.histplot(data=df_analise[['query_used', 'descr_sentim', 'tweet_date']], x='query_used', hue='descr_sentim')  
plt.title('Canal x Sentimento', fontsize=20)  
plt.xticks(rotation=60)  
plt.show()
```



```
In [56]: df_analise.groupby(by=['query_used', 'sentiment']).count()
```

Out[56]:

		tweet_text	tweet_date	descr_sentim
query_used	sentiment			
#curiosidade	2	320	320	320
#fato	2	2938	2938	2938
#noticia	2	939	939	939
#novidade	2	793	793	793
#oportunidade	2	2110	2110	2110
#trabalho	2	2597	2597	2597
:)	0	27022	27022	27022
:)	1	26927	26927	26927
estadao	2	3295	3295	3295
exame	2	2894	2894	2894
folha	2	4214	4214	4214
g1	2	2913	2913	2913
jornaloglobo	2	1991	1991	1991
veja	2	1788	1788	1788

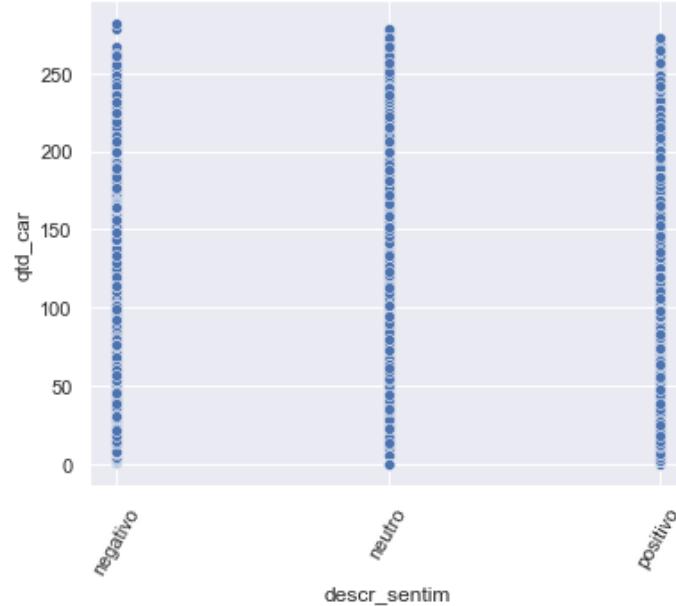
Temos uma variável que já determina o sentimento resultante: query_used, conforme constatado.

Como a proposta é construir um modelo que deduza o sentimento através do texto, ignoramos isso.

```
In [57]: df_analise['qtd_car'] = df_analise['tweet_text'].str.len()
df_analise['qtd_pal'] = df_analise['tweet_text'].str.split().str.len()
```

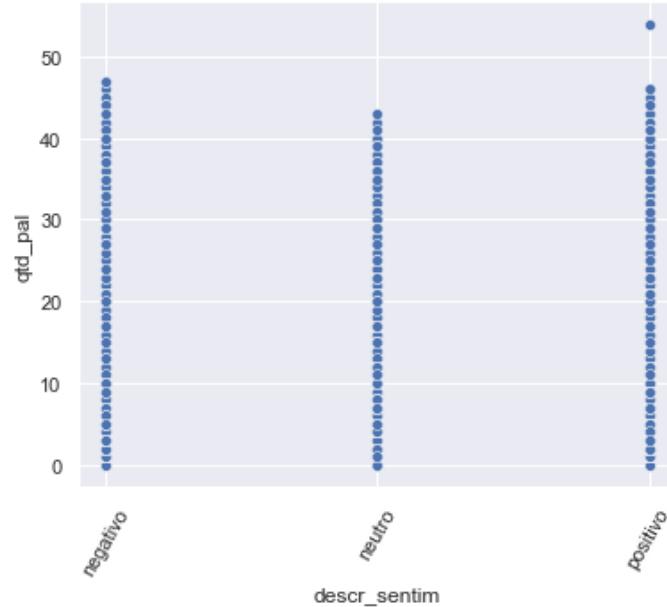
```
In [58]: plt.figure(figsize=(6,5))
sns.scatterplot(data=df_analise[['descr_sentim','qtd_car']], x='descr_sentim', y='qtd_car')
plt.title('Sentimento X Qtd.Caracteres', fontsize=20)
plt.xticks(rotation=60)
plt.show()
```

Sentimento X Qtd.Caracteres



```
In [59]: plt.figure(figsize=(6,5))
sns.scatterplot(data=df_analise[['descr_sentim','qtd_pal']], x='descr_sentim', y='qtd_pal')
plt.title('Sentimento X Qtd.Palavras', fontsize=20)
plt.xticks(rotation=60)
plt.show()
```

Sentimento X Qtd.Palavras



Nota: Contagem de letras e palavras não dizem nada sobre o sentimento resultante.

```
In [60]: df_analise[['qtd_car','qtd_pal']].describe(percentiles=[.1, .2, .3, .4, .5, .75, .9, .99]).T
```

```
Out[60]:      count      mean       std    min   10%   20%   30%   40%   50%   75%   90%   99%   max
qtd_car  80741.0  69.248907  52.152350  0.0  18.0  27.0  38.0  48.0  57.0  85.0  146.0  240.0  282.0
qtd_pal  80741.0  10.555752   7.382888  0.0   3.0   5.0   6.0   7.0   9.0  13.0  21.0  36.0   54.0
```

A contagem de caracteres e palavras reflete a natureza de textos curtos do Twitter.

```
In [61]: from datetime import datetime
```

```
In [62]: df_analise['ano_mes'] = df_analise['tweet_date'].apply(lambda x : x.strftime('%Y-%m-%W'))
```

```
In [63]: df_analise.sample(10)
```

Out[63]:

	tweet_text	tweet_date	sentiment	query_used	descr_sentim	qtd_car	qtd_pal	ano_mes
44783	ahaam!!! importante pro meu irmão irmos tbm ...	2018-10-01 14:06:57+00:00	1	:)	positivo	83	15	2018-10-40
61173	pedidos elas voltaram capa cadeira praia atoal...	2018-10-11 16:20:31+00:00	2	#novidade	neutro	230	28	2018-10-41
43855	hora aventura último episódio vem esse domingo...	2018-09-22 21:00:13+00:00	2	#noticia	neutro	102	14	2018-09-38
83757	aquele momento apega alguém pessoa afasta :(2018-10-02 00:09:33+00:00	0	:(negativo	44	7	2018-10-40
29702	tartaruga ferida reaprende nadar após 'intensi...	2018-10-02 21:15:00+00:00	2	g1	neutro	70	10	2018-10-40
52914	diz que profundo como oceano, mas fato que voc...	2018-10-03 02:32:09+00:00	1	:)	positivo	66	13	2018-10-40
76314	datafolha df: ibaneis (mdb) chega 32%; eliana ...	2018-10-04 22:43:35+00:00	2	veja	neutro	60	10	2018-10-40
69026	ainda solução, cara. você tranquilizou vários...	2018-10-03 17:58:04+00:00	1	:)	positivo	203	28	2018-10-40
44810	✿✿✿terça dia #lançamentos nova coleção está li...	2018-09-19 21:10:12+00:00	2	#novidade	neutro	227	34	2018-09-38
54959	bloqueia aqui também! :d	2018-10-08 11:50:47+00:00	1	:)	positivo	25	4	2018-10-41

In [64]:

```
gb = df_analise.groupby(by=['ano_mes','descr_sentim']).count()
df_gb = gb.reset_index()
df_gb.head(3)
```

Out[64]:

	ano_mes	descr_sentim	tweet_text	tweet_date	sentiment	query_used	qtd_car	qtd_pal
0	2018-08-32	neutro	1	1	1	1	1	1
1	2018-08-33	neutro	513	513	513	513	513	513
2	2018-08-34	neutro	1648	1648	1648	1648	1648	1648

In [65]:

```
plt.figure(figsize=(18,5))
sns.lineplot(data=df_gb, x='ano_mes', hue='descr_sentim', y='sentiment')
plt.title('Semana X Sentimento', fontsize=20)
plt.xticks(rotation=45)
plt.show()
```



Nota: Observamos que há uma questão temporal nos dados

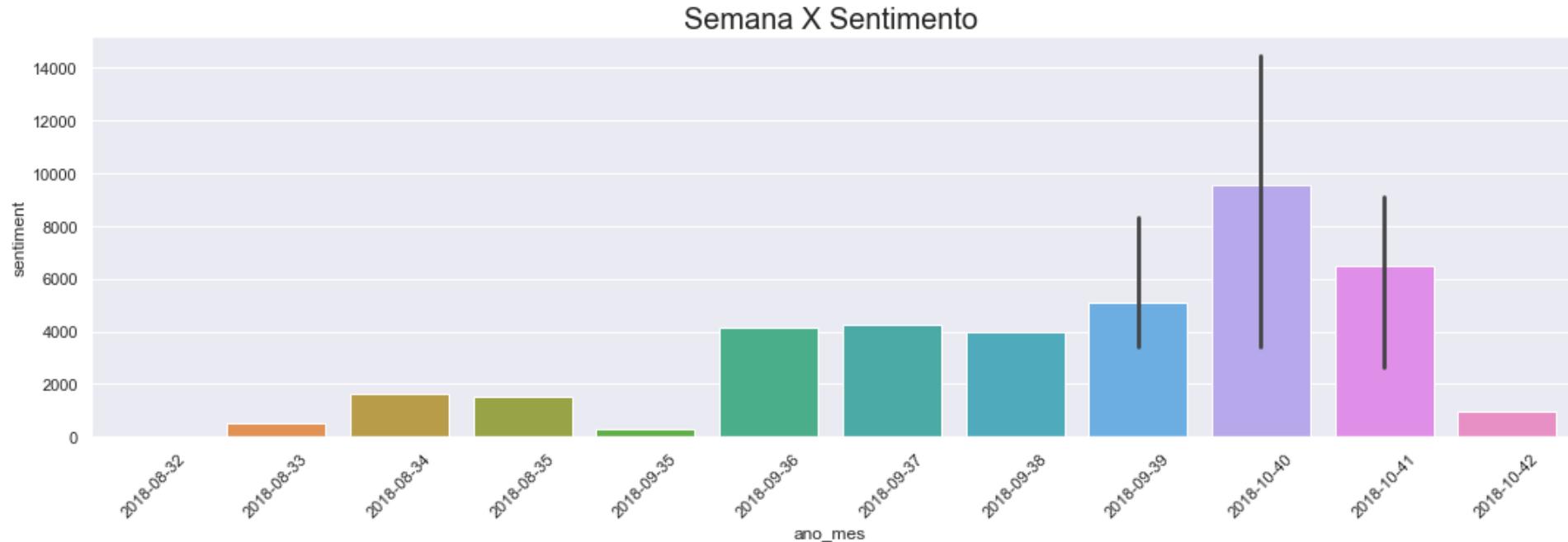
Até a quarta semana de Setembro, todos os tweets estão classificados como neutros. Somente a partir da última semana de setembro/2018 as classificações positivo e negativo passaram a ocorrer. E já na última semana de outubro/2018 só constam tweets neutros.

Uma eventual seleção Treino x Validação out-of-time pode produzir uma base de validação contendo apenas tweets neutros.

O gráfico a seguir comprova...

...a concentração de tweets entre o final de setembro e outubro.

```
In [66]: plt.figure(figsize=(18,5))
sns.barplot(data=df_gb, x='ano_mes', y='sentiment')
plt.title('Semana X Sentimento', fontsize=20)
plt.xticks(rotation=45)
plt.show()
```



```
In [67]: df_gb = None      # Não é mais necessária
```

Testes e Seleção do Modelo NLP

Primeiro Passo: Separação das bases de Treino e Validação

```
In [68]: df_treino, df_valid = train_test_split(df_analise, test_size=0.1)

X_treino, y_treino = df_treino['tweet_text'], df_treino['sentiment']
X_valid, y_valid = df_valid['tweet_text'], df_valid['sentiment']

print('Shape Treino - X:', X_treino.shape, 'Y:', y_treino.shape)
print('Shape Valid. - X:', X_valid.shape, 'Y:', y_valid.shape)

Shape Treino - X: (72666,) Y: (72666,)
Shape Valid. - X: (8075,) Y: (8075,)
```

```
In [69]: df_analise = None      # Não é mais necessária
```

Algumas funções utilitárias

Função para exibir métricas

```
In [70]: from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
In [71]: '''
Calcula as métricas de uma base de teste ou de bases de treino e validação.
'''
```

```
def metricas(y, y_pred, y_valid=None, y_valid_pred=None):

    acc = accuracy_score(y, y_pred)
    prec = precision_score(y, y_pred, average='weighted')
    rec = recall_score(y, y_pred, average='weighted')

    # Retornará os resultados calculados para uso posterior
    str_teste = ''
    str_treino = ''
    str_valid = ''
    if (y_valid is None):
        str_teste = f'Teste:\nAcc: {acc:.3f}, Precision: {prec:.3f}, Recall: {rec:.3f}'
        print(str_teste)
    else:
        acc_valid = accuracy_score(y_valid, y_valid_pred)
        prec_valid = precision_score(y_valid, y_valid_pred, average='weighted')
        rec_valid = recall_score(y_valid, y_valid_pred, average='weighted')

        str_treino = f'Treino...:\nAcc: {acc:.3f}, Precision: {prec:.3f}, Recall: {rec:.3f}'
        print(str_treino)
        str_valid = f'Validação:\nAcc: {acc_valid:.3f}, Precision: {prec_valid:.3f}, Recall: {rec_valid:.3f}'
        print(str_valid)

    return str_treino, str_valid, str_teste
```

Função para exibir a matriz de confusão

```
In [72]: from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
```

```
In [73]: '''
Plota o gráfico matriz de confusão de uma base Y e Y predita
'''
```

```
def matriz_conf(y, y_pred):
    cm = confusion_matrix(y, y_pred)
    plt.figure(figsize=(12,12))
    plot_confusion_matrix(conf_mat=cm)
    plt.show()
```

Função para executar um modelo de predição e apresentar métricas e matriz de confusão

```
In [74]: import time
...
Executa um modelo de predição com as bases especificadas.
Se X_Valid e y_valid forem especificados, entao a função entende que foi executada na fase de treino e faz o FIT
do modelo antes das predições. Caso contrário, o modelo entende ser uma predição de teste e não faz o FIT
...
def exec_modelo(modelo, X, y=None, X_valid=None, y_valid=None, zerar_valid=True):

    str_tempo = ''

    if X_valid is not None:
        # Fase de treino: FIT do modelo e predição para a base de validação
        print("Realizando FIT do modelo...")
        inicio = time.time()
        modelo.fit(X, y)
        fim = time.time()
        str_tempo = 'Tempo de treinamento do modelo: {}s'.format(fim - inicio)
        print(str_tempo)
        print()
        print('Realizando Predições da base de validacao...')
        y_valid_pred = modelo.predict(X_valid)

    # Predição da base X, que pode ser a base de treino ou a base de teste
    print('Realizando Predições da base de treino/teste...')
    y_pred = modelo.predict(X)

    # Apresenta as métricas e matriz de confusão somente de "y's" forem fornecidos
    str_treino = ''
    str_valid = ''
    str_teste = ''
    if y is not None:
        if y_valid_pred is not None:
            print("Resultados do Treinamento:")
            str_treino, str_valid, str_teste = metricas(y, y_pred, y_valid, y_valid_pred)
            matriz_conf(y_valid, y_valid_pred)
            if zerar_valid:
                y_valid_pred = None
        else:
            print("Resultados da Validação:")
            str_treino, str_valid, str_teste = metricas(y, y_pred)
            matriz_conf(y, y_pred)

    print("Modelo executado.")

    return y_pred, str_tempo, str_treino, str_valid, str_teste
```

BagOfWords

```
In [75]: from sklearn.feature_extraction.text import CountVectorizer
```

```
vectr = CountVectorizer(max_features=10000, stop_words=stop_words)
vectr.fit(X_treino)
```

```
D:\LetsCode\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['1k', '1t', '2l', '3d', '3u', '4g', '4m', '5d', '5g', '5m', '6e', '6s', 'a2', 'a9', 'ac', 'az', 'b1', 'b3', 'bl', 'bm', 'bn', 'bt', 'cl', 'ct', 'cz', 'd2', 'd9', 'df', 'dh', 'ds', 'dv', 'e3', 'ec', 'ee', 'f5', 'fh', 'fm', 'fs', 'g2', 'gb', 'ge', 'gj', 'gm', 'gp', 'gs', 'gw', 'hg', 'hl', 'hn', 'ii', 'iu', 'iv', 'ix', 'iá', 'j2', 'jf', 'jn', 'jw', 'jz', 'jà', 'jã', 'jú', 'k9', 'ke', 'kw', 'lc', 'lp', 'ls', 'lw', 'mg', 'mj', 'mk', 'ma', 'nr', 'nw', 'oj', 'ot', 'pb', 'qb', 'qg', 'rb', 'rc', 'rd', 'rh', 'rl', 'rr', 'sh', 'st', 'sá', 'sò', 't1', 'tf', 'tj', 'ts', 'tó', 'u2', 'ub', 'uf', 'uh', 'uo', 'uq', 'ut', 'uv', 'ux', 'vp', 'vt', 'vw', 'vé', 'wd', 'wm', 'wo', 'wt', 'wu', 'xa', 'xi', 'xp', 'xy', 'xô', 'z6', 'z1', 'zn', 'zs', 'ôô', 'eu', 'nó'] not in stop_words.
```

```
warnings.warn('Your stop_words may be inconsistent with '
```

```
Out[75]: CountVectorizer(max_features=10000,
    stop_words=['!', '!!', '!?', '!"', '!đđ', '",', "", ',', '"-', '"0',
    '"3', '"5', '"A', '"E', '"I', '"N', '"0', '"Y',
    '"a', '"e', '"n', '"o', '"q', '"A', '"É', '"é',
    '"ô', '"☺', '#', '#1', '#2', ...])
```

```
In [76]: X_treino_bow, X_valid_bow = vectr.transform(X_treino).toarray(), vectr.transform(X_valid).toarray()
```

```
print('Treino:', X_treino_bow.shape)
print('Valid.:', X_valid_bow.shape)
```

```
Treino: (72666, 10000)
Valid.: (8075, 10000)
```

Testando alguns modelos

```
In [77]: from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
```

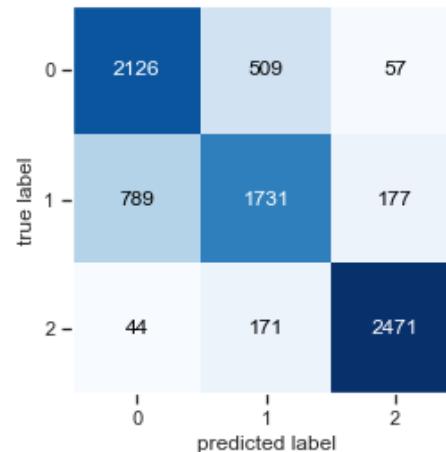
Naive Bayes

```
In [78]: modelo = MultinomialNB()
```

```
In [79]: # exec_modelo() retorna o y_pred, tempo de treinamento, métricas de treino, validação e teste.
# vamos salvar de cada modelo testado para facilitar a revisão ao final.
_, tempo_bow_nb, treino_bow_nb, valid_bow_nb, _ = exec_modelo(modelo, X_treino_bow, y_treino, X_valid_bow, y_valid)
```

```
Realizando FIT do modelo...
Tempo de treinamento do modelo: 88.24023842811584s
```

```
Realizando Predições da base de validacao...
Realizando Predições da base de treino/teste...
Resultados do Treinamento:
Treino...:
Acc: 0.819, Precision: 0.820, Recall: 0.819
Validação:
Acc: 0.784, Precision: 0.783, Recall: 0.784
<Figure size 864x864 with 0 Axes>
```



```
Modelo executado.
```

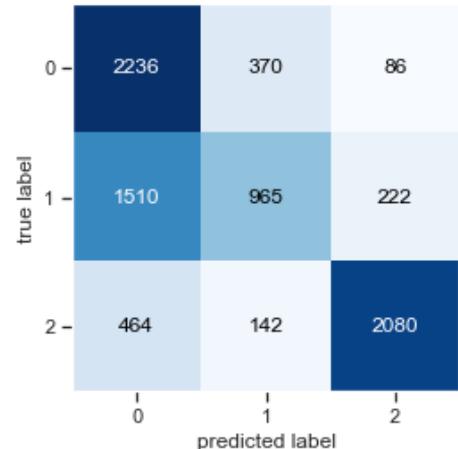
Random Forest

```
In [80]: modelo = RandomForestClassifier(n_estimators=200, max_depth=3, n_jobs=8, random_state=12)
```

```
In [81]: _, tempo_bow_rf, treino_bow_rf, valid_bow_rf, _ = exec_modelo(modelo, X_treino_bow, y_treino, X_valid_bow, y_valid)
```

```
Realizando FIT do modelo...
Tempo de treinamento do modelo: 28.386947870254517s
```

```
Realizando Predições da base de validacao...
Realizando Predições da base de treino/teste...
Resultados do Treinamento:
Treino...:
Acc: 0.667, Precision: 0.696, Recall: 0.667
Validação:
Acc: 0.654, Precision: 0.685, Recall: 0.654
<Figure size 864x864 with 0 Axes>
```



Modelo executado.

LightGBM

```
In [82]: modelo = LGBMClassifier(n_estimators=200, max_depth=3, n_jobs=8, random_state=12)
```

```
In [83]: _, tempo_bow_lg, treino_bow_lg, valid_bow_lg, _ = exec_modelo(modelo, X_treino_bow, y_treino, X_valid_bow, y_valid)
```

Realizando FIT do modelo...

Tempo de treinamento do modelo: 9.173457622528076s

Realizando Predições da base de validacao...

Realizando Predições da base de treino/teste...

Resultados do Treinamento:

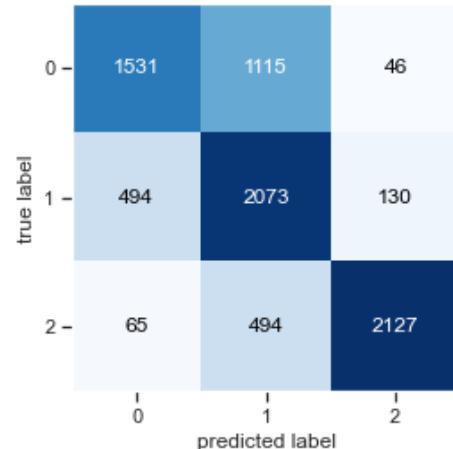
Treino...:

Acc: 0.720, Precision: 0.748, Recall: 0.720

Validação:

Acc: 0.710, Precision: 0.739, Recall: 0.710

<Figure size 864x864 with 0 Axes>



Modelo executado.

Vamos definir nossa classe classificadora customizada

Baseado na função de classificação por emoticons criada anteriormente

```
In [84]: import numpy as np
...
Classificador customizado (bastante simples) que retorna o sentimento com base na presença de
emoticons positivos e/ou negativos.
...
class EmoticonClassifier:

    def __init__(self):

        self._X = None
        self._y = None
        self._fitted = False

    def fit(self, x, y):

        self._X = x
        self._y = y
        self._fitted = True

    def predict(self, x):

        if not self._fitted:
            print('Model not fitted.')
            return None
        else:
```

```

y_pred = np.array([], dtype='int8')
for texto in x:
    y_pred = np.append(y_pred, previsao_emoticons(texto))

return y_pred

```

Executando o modelo customizado...

In [85]: `modelo = EmoticonClassifier()`

In [86]: `_, tempo_bow_cm, treino_bow_cm, valid_bow_cm, _ = exec_modelo(modelo, X_treino, y_treino, X_valid, y_valid)`

Realizando FIT do modelo...

Tempo de treinamento do modelo: 0.0s

Realizando Predições da base de validacao...

Realizando Predições da base de treino/teste...

Resultados do Treinamento:

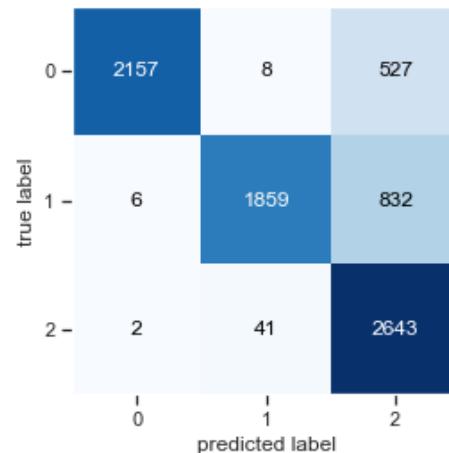
Treino...

Acc: 0.819, Precision: 0.873, Recall: 0.819

Validação:

Acc: 0.825, Precision: 0.877, Recall: 0.825

<Figure size 864x864 with 0 Axes>



Modelo executado.

In [87]: `vectr = None
modelo = None
X_treino_bow = None
X_valid_bow = None`

TF-IDF

```
In [88]: from sklearn.feature_extraction.text import TfidfVectorizer  
  
vectr_idf = TfidfVectorizer(max_features=10000, stop_words=stop_words)  
vectr_idf.fit(X_treino)
```

```
D:\LetsCode\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['1k', '1t', '2l', '3d', '3u', '4g', '4m', '5d', '5g', '5m', '6e', '6s', 'a2', 'a9', 'ac', 'az', 'b1', 'b3', 'bl', 'bm', 'bn', 'bt', 'c1', 'ct', 'cz', 'd2', 'd9', 'df', 'dh', 'ds', 'dv', 'e3', 'ec', 'ee', 'f5', 'fh', 'fm', 'fs', 'g2', 'gb', 'ge', 'gj', 'gm', 'gp', 'gs', 'gw', 'hg', 'hl', 'hn', 'ii', 'iu', 'iv', 'ix', 'ia', 'j2', 'jf', 'jn', 'jw', 'jz', 'já', 'jú', 'jã', 'k9', 'ke', 'kw', 'lc', 'lp', 'ls', 'lw', 'mg', 'mj', 'mk', 'ma', 'nr', 'nw', 'oj', 'ot', 'pb', 'qb', 'qg', 'rb', 'rc', 'rd', 'rh', 'rl', 'rn', 'sh', 'st', 'sá', 'sò', 't1', 'tf', 'tj', 'ts', 'tó', 'u2', 'ub', 'uf', 'uh', 'uo', 'uq', 'ut', 'uv', 'ux', 'vp', 'vt', 'vw', 'vé', 'wd', 'wm', 'wo', 'wt', 'wu', 'xa', 'xi', 'xp', 'xy', 'xô', 'z6', 'z1', 'zn', 'zs', 'ôô', 'eu', 'n o'] not in stop_words.
```

```
warnings.warn('Your stop_words may be inconsistent with '
```

```
Out[88]: TfidfVectorizer(max_features=10000,  
                         stop_words=['!', '!!', '!?', '!?"', '!ﷺ', '",', '",', '"-', '"0',  
                         '"3', '"5', '"A', '"E', '"I', '"N', '"O', '"Y',  
                         '"a', '"e', '"n', '"o', '"q', '"À', '"É', '"é',  
                         '"ô', '"߻', '#', '#1', '#2', ...])
```

```
In [89]: X_treino_idf, X_valid_idf = vectr_idf.transform(X_treino).toarray(), vectr_idf.transform(X_valid).toarray()  
  
print('Treino:', X_treino_idf.shape)  
print('Valid.:', X_valid_idf.shape )
```

```
Treino: (72666, 10000)  
Valid.: (8075, 10000)
```

Naive Bayes

```
In [90]: modelo = MultinomialNB()
```

```
In [91]: _, tempo_idf_rf, treino_idf_nb, valid_idf_nb, _ = exec_modelo(modelo, X_treino_idf, y_treino, X_valid_idf, y_valid)
```

```
Realizando FIT do modelo...
```

```
Tempo de treinamento do modelo: 2.796049118041992s
```

```
Realizando Predições da base de validacao...
```

```
Realizando Predições da base de treino/teste...
```

```
Resultados do Treinamento:
```

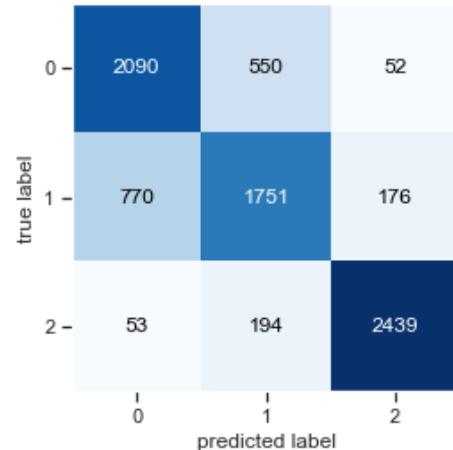
```
Treino...:
```

```
Acc: 0.820, Precision: 0.821, Recall: 0.820
```

```
Validação:
```

```
Acc: 0.778, Precision: 0.778, Recall: 0.778
```

```
<Figure size 864x864 with 0 Axes>
```



Modelo executado.

Random Forest

```
In [92]: modelo = RandomForestClassifier(n_estimators=200, max_depth=3, n_jobs=8, random_state=12)
```

```
In [93]: _, tempo_idf_nb, treino_idf_rf, valid_idf_rf, _ = exec_modelo(modelo, X_treino_idf, y_treino, X_valid_idf, y_valid)
```

Realizando FIT do modelo...

Tempo de treinamento do modelo: 26.47352170944214s

Realizando Predições da base de validacao...

Realizando Predições da base de treino/teste...

Resultados do Treinamento:

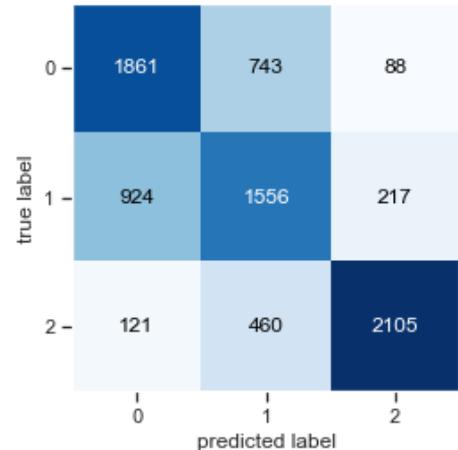
Treino...:

Acc: 0.688, Precision: 0.696, Recall: 0.688

Validação:

Acc: 0.684, Precision: 0.692, Recall: 0.684

<Figure size 864x864 with 0 Axes>



Modelo executado.

LightGBM

```
In [94]: modelo = LGBMClassifier(n_estimators=200, max_depth=3, n_jobs=8, random_state=12)
```

```
In [95]: _, tempo_idf_lg, treino_idf_lg, valid_idf_lg, _ = exec_modelo(modelo, X_treino_idf, y_treino, X_valid_idf, y_valid)
```

Realizando FIT do modelo...

Tempo de treinamento do modelo: 15.049388647079468s

Realizando Predições da base de validacao...

Realizando Predições da base de treino/teste...

Resultados do Treinamento:

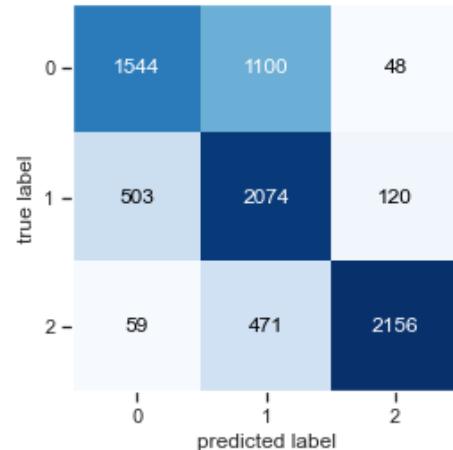
Treino...:

Acc: 0.728, Precision: 0.755, Recall: 0.728

Validação:

Acc: 0.715, Precision: 0.743, Recall: 0.715

<Figure size 864x864 with 0 Axes>



Modelo executado.

```
In [96]: vectr_idf = None
modelo = None
X_treino_idf = None
X_valid_idf = None
```

Word Embedding

Criação dos vetores

```
In [97]: from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
```

```
In [98]: '''
Quebra textos em palavras
'''

def tokenize(texto, m_len=3):
    novo_texto = simple_preprocess(texto, min_len=m_len)

    return novo_texto
```

Cálculo de similaridade das diferentes palavras via Word2Vec

```
In [99]: # Definindo o tamanho do vetor utilizado no modelo de embedding e na rede neural
TAM_VETOR=80
```

```
In [100...]: modelo_w2v = Word2Vec(sentences = X_treino.apply(lambda t: tokenize(t)),
                           vector_size = TAM_VETOR,
                           window = 7,
```

```
min_count = 1,  
workers = 3)
```

Alguns testes para avaliar a performance dos agrupamentos

```
In [101]: modelo_w2v.wv.most_similar('ótimo')[:5]
```

```
Out[101]: [('booom', 0.9392982125282288),  
 ('bom', 0.9340139627456665),  
 ('começando', 0.9256890416145325),  
 ('cansativo', 0.9221374988555908),  
 ('inteiro', 0.9202582836151123)]
```

```
In [102]: modelo_w2v.wv.most_similar('ruim')[:5]
```

```
Out[102]: [('legal', 0.9665572643280029),  
 ('difícil', 0.9635748267173767),  
 ('complicado', 0.9629555344581604),  
 ('longe', 0.9622583389282227),  
 ('foda', 0.9552310109138489)]
```

```
In [103]: modelo_w2v.wv.most_similar('não')[:5]
```

```
Out[103]: [('nem', 0.9258705377578735),  
 ('nunca', 0.8844177722930908),  
 ('absolutamente', 0.8841944336891174),  
 ('nao', 0.8839237689971924),  
 ('pior', 0.8763203024864197)]
```

Visualização da distribuição das palavras num plano cartesiano

De: <https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92>

```
In [104]: from sklearn.manifold import TSNE
```

```
In [105]: ...  
Exibe um gráfico de distribuição de palavras num plano cartesiano  
...  
def grafico_palavras(modelo, palavra):  
  
    # Array de dados float  
    arr = np.empty((0, TAM_VETOR), dtype='f')  
    # Uma Lista de palavras  
    palavras = [palavra]  
    # Obtem as palavras mais similares de uma dada amostra  
    proximas = modelo.similar_by_word(palavra)  
    # Constrói um array de arrays com as palavras e suas próximas  
    arr = np.append(arr, np.array([modelo[palavra]]), axis=0)  
    # Constrói o array de vetores  
    for score in proximas:  
        arr = np.append(arr, np.array([modelo[score]]), axis=0)
```

```

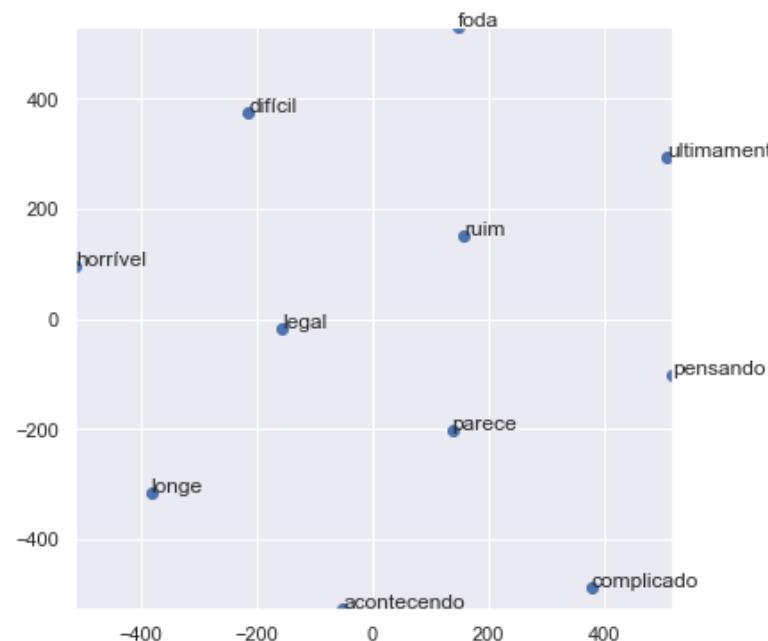
vetor_palavras = modelo[score[0]]
palavras.append(score[0])
arr = np.append(arr, np.array([vetor_palavras]), axis=0)

# Instancia o objeto produtor de dados para o gráfico
tsne = TSNE(n_components=2, random_state=0)
np.set_printoptions(suppress=True)
# Cálculo dos pontos do gráfico para os vetores obtidos
Y = tsne.fit_transform(arr)
coords_x = Y[:, 0]
coords_y = Y[:, 1]

# Gera o gráfico
plt.figure(figsize=(6,6))
plt.scatter(coords_x, coords_y)
# Adiciona a palavra relacionada com cada ponto
for palavra, x, y in zip(palavras, coords_x, coords_y):
    plt.annotate(palavra, xy=(x, y), xytext=(0, 0), textcoords='offset points')
# Garante uma pequena distância em cada ponto
plt.xlim(coords_x.min()+0.0005, coords_x.max()-0.0005)
plt.ylim(coords_y.min()+0.0005, coords_y.max()-0.0005)
plt.show()

```

In [106...]: grafico_palavras(modelo_w2v.wv, 'ruim')



Keras

```
In [107...]: from keras.preprocessing.sequence import pad_sequences
from tensorflow import keras
from tensorflow.keras import layers
from tqdm import tqdm
```

```
In [108...]: """
Retorna duas lista: tokens e índices correspondentes
"""

def get_vocab(modelo_w2v):
    word2id = {}
    id2word = {}

    for index, word in enumerate(tqdm(modelo_w2v.wv.key_to_index)):
        word2id[word] = index + 1
        id2word[index + 1] = word

    return word2id, id2word
```

```
In [109...]: """
Retorna um array NumPy contendo ids dos tokens informados
"""

def seq_to_id(sequence, vocabulary):

    ids = []
    nao_enc = len(vocabulary)      # Evita repetidas chamadas para Len()
    for word in sequence:
        ids.append(vocabulary[word] if word in vocabulary else nao_enc)

    return np.array(ids)
```

```
In [110...]: vocab, id2word = get_vocab(modelo_w2v)
```

100% |██████████| 51941/51941 [00:00<00:00, 1402529.72it/s]

```
In [111...]: X_treino_seq = X_treino.apply(tokenize).apply(lambda s: seq_to_id(s, vocab))
X_valid_seq = X_valid.apply(tokenize).apply(lambda s: seq_to_id(s, vocab))
```

```
In [112...]: X_treino_seq.head()
```

```
Out[112]: 77620    [17584, 16149, 10, 5933, 341, 341, 111, 5, 43, ...]
3015     [7, 3017, 101, 56, 607, 1, 7132, 146, 56, 2544]
43295    [327, 188, 167, 92, 11, 79, 11, 119, 3957, 5, ...]
38007    [3950, 4, 14, 933, 2, 67, 1187, 67, 1, 899]
51533    [203, 48, 10, 15, 208, 29, 2, 31, 4590, 1385, ...]
Name: tweet_text, dtype: object
```

```
In [113...]: # Parâmetro para truncamento e padding
MAX_LENGTH = 100
```

Padronização do tamanho das amostras

```
In [114... X_treino_seq = pad_sequences(X_treino_seq, padding='post', truncating='post', maxlen=MAX_LENGTH)
X_valid_seq = pad_sequences(X_valid_seq, padding='post', truncating='post', maxlen=MAX_LENGTH)
```

```
In [115... X_treino_seq.shape
```

```
Out[115]: (72666, 100)
```

```
In [116... X_treino_seq
```

```
Out[116]: array([[17584, 16149,    10, ...,     0,     0,     0],
   [    7, 3017,   101, ...,     0,     0,     0],
   [  327,   188,   167, ...,     0,     0,     0],
   ...,
   [   86, 19589, 1547, ...,     0,     0,     0],
   [   48,    94,   156, ...,     0,     0,     0],
   [   55,    48,   123, ...,     0,     0,     0]])
```

Encoding dos valores de validação: O modelo espera que os resultados estejam entre 0 e 1

```
In [117... """
Transforma 0, 1 e 2 em 0, 0.5 e 1, respectivamente, como esperado pelo modelo
"""
def encode_y(y):

    return y.copy().apply(lambda x: float(x) / 2)
```

```
In [118... y_treino_enc = encode_y(y_treino)
y_valid_enc = encode_y(y_valid)
```

```
In [119... print('y_treino_enc:')
print(y_treino_enc)
print()
print('y_valid_enc:')
print(y_valid_enc)
```

```
y_treino_enc:  
77620    0.5  
3015     0.0  
43295    0.0  
38007    0.5  
51533    0.0  
...  
74524    0.5  
43905    1.0  
28408    0.0  
50187    0.5  
60531    0.0  
Name: sentiment, Length: 72666, dtype: float64
```

```
y_valid_enc:  
47225    1.0  
28334    0.0  
63335    0.5  
70788    1.0  
33002    1.0  
...  
69442    0.0  
8275     0.5  
15049    1.0  
16797    0.5  
58220    0.0  
Name: sentiment, Length: 8075, dtype: float64
```

Rede Neural

```
In [120]: modelo = keras.Sequential(  
        [  
            layers.Embedding(len(vocab)+1, TAM_VETOR, trainable=True, input_length=MAX_LENGTH, mask_zero=True, name="emb"),  
            layers.Lambda(lambda x: keras.backend.mean(x, axis=1)),  
            layers.Dense(TAM_VETOR, activation="relu", name="layer2"),  
            layers.Dense(1, activation="sigmoid"),  
        ]  
)
```

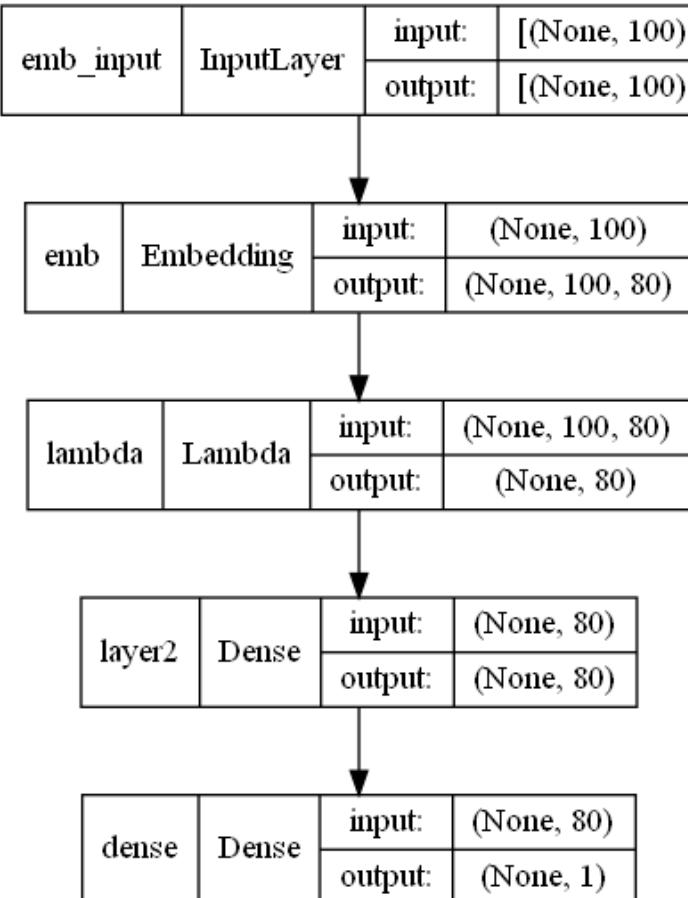
```
In [121]: modelo.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
emb (Embedding)	(None, 100, 80)	4155360
lambda (Lambda)	(None, 80)	0
layer2 (Dense)	(None, 80)	6480
dense (Dense)	(None, 1)	81
<hr/>		
Total params: 4,161,921		
Trainable params: 4,161,921		
Non-trainable params: 0		

In [122]: `keras.utils.plot_model(modelo, show_shapes=True)`

Out[122]:



Compilação do modelo

```
In [123...]: modelo.compile(  
    optimizer=keras.optimizers.Adam(learning_rate=0.001),  
    loss=keras.losses.BinaryCrossentropy(),  
    metrics=[keras.metrics.BinaryAccuracy()],  
)
```

Treino do modelo

```
In [124...]: print("Treino do modelo")  
inicio = time.time()  
modelo.fit(  
    X_treino_seq,  
    y_treino_enc,  
    batch_size=100,  
    epochs=1,  
    validation_data=(X_valid_seq, y_valid_enc),  
    use_multiprocessing=True  
)  
fim = time.time()  
tempo_we = 'Tempo de treinamento do modelo: {}s'.format(fim - inicio)
```

Treino do modelo

727/727 [=====] - 33s 43ms/step - loss: 0.4923 - binary_accuracy: 0.5990 - val_loss: 0.4031 - val_binary_accuracy: 0.6484- ET

Execução do modelo

Os resultados estarão na forma do "hot encoding", então devemos decodificar.

```
In [125...]:  
    '''  
    Decodifica as previsões para os valores 0, 1 e 2.  
    Como a previsão vem como um float entre 0 e 1, utilizamos a divisão customizada para decodificar:  
    Entre 0.00 e 0.39 -> 0  
    Entre 0.40 e 0.59 -> 1  
    Entre 0.60 e 1.00 -> 2  
    '''  
  
def decode_y(y):  
  
    novo_y = []  
    for x in y:  
        novo_y.append(0 if x < 0.40 else 1 if x < 0.60 else 2)  
    return np.array(novo_y)
```

```
In [126...]: y_treino_pred = decode_y(modelo.predict(X_treino_seq).reshape(-1))  
y_valid_pred = decode_y(modelo.predict(X_valid_seq).reshape(-1))  
  
print('Treino....:', y_treino_pred.shape)  
print('Validacao:', y_valid_pred.shape )
```

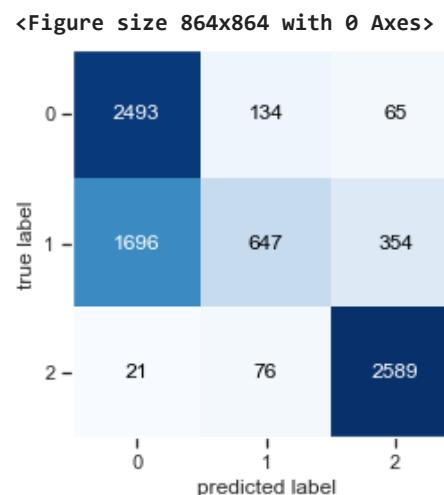
```
Treino...: (72666,)  
Validacao: (8075,)
```

```
In [127]: print(y_treino_pred)  
print(y_valid_pred)  
  
[0 0 0 ... 0 0 0]  
[2 0 0 ... 2 1 0]
```

Resultados

```
In [128]: treino_we, valid_we, _ = metricas(y_treino, y_treino_pred, y_valid, y_valid_pred)  
  
Treino...:  
Acc: 0.730, Precision: 0.770, Recall: 0.730  
Validação:  
Acc: 0.709, Precision: 0.736, Recall: 0.709
```

```
In [129]: matriz_conf(y_valid, y_valid_pred)
```



```
In [130]: modelo_w2v = None  
vocab = None  
id2word = None  
X_treino_seq = None  
X_valid_seq = None  
y_treino_enc = None  
y_valid_enc = None  
modelo = None  
y_treino_pred = None  
y_valid_pred = None
```

Conclusão sobre os modelos:

Levando em conta os resultados sem otimização dos hiperparâmetros dos modelos:

BagOfWords com Naive Bayes:

```
In [131...]: print(tempo_bow_nb, '\n', treino_bow_nb, '\n', valid_bow_nb)

Tempo de treinamento do modelo: 88.24023842811584s
Treino...:
Acc: 0.819, Precision: 0.820, Recall: 0.819
Validação:
Acc: 0.784, Precision: 0.783, Recall: 0.784
```

BagOfWords com Random Forest:

```
In [132...]: print(tempo_bow_rf, '\n', treino_bow_rf, '\n', valid_bow_rf)

Tempo de treinamento do modelo: 28.386947870254517s
Treino...:
Acc: 0.667, Precision: 0.696, Recall: 0.667
Validação:
Acc: 0.654, Precision: 0.685, Recall: 0.654
```

BagOfWords com LightGBM:

```
In [133...]: print(tempo_bow_lg, '\n', treino_bow_lg, '\n', valid_bow_lg)

Tempo de treinamento do modelo: 9.173457622528076s
Treino...:
Acc: 0.720, Precision: 0.748, Recall: 0.720
Validação:
Acc: 0.710, Precision: 0.739, Recall: 0.710
```

BagOfWords com Classificador Customizado:

```
In [134...]: print(tempo_bow_cm, '\n', treino_bow_cm, '\n', valid_bow_cm)

Tempo de treinamento do modelo: 0.0s
Treino...:
Acc: 0.819, Precision: 0.873, Recall: 0.819
Validação:
Acc: 0.825, Precision: 0.877, Recall: 0.825
```

TF-IDF com Naive Bayes:

```
In [135...]: print(tempo_idf_nb, '\n', treino_idf_nb, '\n', valid_idf_nb)
```

```
Tempo de treinamento do modelo: 26.47352170944214s
```

```
Treino...:
```

```
Acc: 0.820, Precision: 0.821, Recall: 0.820
```

```
Validação:
```

```
Acc: 0.778, Precision: 0.778, Recall: 0.778
```

TF-IDF com Random Forest:

```
In [136]: print(tempo_idf_rf, '\n', treino_idf_rf, '\n', valid_idf_rf)
```

```
Tempo de treinamento do modelo: 2.796049118041992s
```

```
Treino...:
```

```
Acc: 0.688, Precision: 0.696, Recall: 0.688
```

```
Validação:
```

```
Acc: 0.684, Precision: 0.692, Recall: 0.684
```

TF-IDF com LightGBM:

```
In [137]: print(tempo_idf_lg, '\n', treino_idf_lg, '\n', valid_idf_lg)
```

```
Tempo de treinamento do modelo: 15.049388647079468s
```

```
Treino...:
```

```
Acc: 0.728, Precision: 0.755, Recall: 0.728
```

```
Validação:
```

```
Acc: 0.715, Precision: 0.743, Recall: 0.715
```

Word Embedding com Keras:

```
In [138]: print(tempo_we, '\n', treino_we, '\n', valid_we)
```

```
Tempo de treinamento do modelo: 35.20295763015747s
```

```
Treino...:
```

```
Acc: 0.730, Precision: 0.770, Recall: 0.730
```

```
Validação:
```

```
Acc: 0.709, Precision: 0.736, Recall: 0.709
```

Verifica-se que BagOfWords com nosso modelo customizado obteve os melhores resultados, seguido de BagOfWords com Naive Bayes.

São modelos com poucas opções de customização de hiperparâmetros. De fato, nosso classificador customizado nem os têm.

Vamos explorar um pouco BOW com NB, e algumas tentativas com Word Embeddings.

Otimização do modelo BagOfWords com NB

Antes de executar, forçamos a limpeza do "lixo"

```
In [139]: import gc  
gc.collect()
```

Out[139]: 6163

GridSearch

In [140...]: `from sklearn.model_selection import GridSearchCV`

Recriando os modelos e dados necessários

In [141...]: `# Para o GridSearch vamos limitar o número de features devido o consumo de memória
vectr = CountVectorizer(max_features=4000, stop_words=stop_words)
vectr.fit(X_treino)`

```
X_treino_bow, X_valid_bow = vectr.transform(X_treino).toarray(), vectr.transform(X_valid).toarray()  
print('Treino:', X_treino_bow.shape)  
print('Valid.:', X_valid_bow.shape)
```

D:\LetsCode\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['1k', '1t', '2l', '3d', '3u', '4g', '4m', '5d', '5g', '5m', '6e', '6s', 'a2', 'a9', 'ac', 'az', 'b1', 'b3', 'bl', 'bm', 'bn', 'bt', 'c1', 'ct', 'cz', 'd2', 'd9', 'df', 'dh', 'ds', 'dv', 'e3', 'ec', 'ee', 'f5', 'fh', 'fm', 'fs', 'g2', 'gb', 'ge', 'gj', 'gm', 'gp', 'gs', 'gw', 'hg', 'hl', 'hn', 'ii', 'iu', 'iv', 'ix', 'ia', 'j2', 'jf', 'jn', 'jw', 'jz', 'jà', 'jã', 'jú', 'k9', 'ke', 'kw', 'lc', 'lp', 'ls', 'lw', 'mg', 'mj', 'mk', 'ma', 'nr', 'nw', 'oj', 'ot', 'pb', 'qb', 'qg', 'rb', 'rc', 'rd', 'rh', 'rl', 'rr', 'sh', 'st', 'sá', 'sò', 't1', 'tf', 'tj', 'ts', 'tó', 'u2', 'ub', 'uf', 'uh', 'uo', 'uq', 'ut', 'uv', 'ux', 'vp', 'vt', 'vw', 'vé', 'wd', 'wm', 'wo', 'wt', 'wu', 'xa', 'xi', 'xp', 'xy', 'xô', 'z6', 'zl', 'zn', 'zs', 'ôô', 'eu', 'no'] not in stop_words.
warnings.warn('Your stop_words may be inconsistent with '
Treino: (72666, 4000)
Valid.: (8075, 4000)

In [142...]: `# Transformação com o novo limite de fatures
X_treino_bow = vectr.transform(X_treino).toarray()`

In [143...]: `# Instância para usar com o GridSearch
modelo = MultinomialNB()`

In [144...]: `# Parâmetros a testar
parameters = {
 'alpha': [1.0, 0.8, 0.6, 0.4, 0.2],
 'fit_prior': [True, False]
}`

In [145...]: `# Execução dos testes
gs = GridSearchCV(modelo, parameters, scoring='accuracy', verbose=3, n_jobs=-1, cv=2)
gs.fit(X_treino_bow[:20000], y_treino[:20000])`

Fitting 2 folds for each of 10 candidates, totalling 20 fits
GridSearchCV(cv=2, estimator=MultinomialNB(), n_jobs=-1,
param_grid={'alpha': [1.0, 0.8, 0.6, 0.4, 0.2],
 'fit_prior': [True, False]},
scoring='accuracy', verbose=3)

Out[145]:

```
In [146... # Melhores parâmetros
```

```
gs.best_params_
```

```
Out[146]: {'alpha': 0.8, 'fit_prior': True}
```

```
In [147... # Modelo com os melhores parâmetros definidos por GridSearchCV
```

```
modelo = gs.best_estimator_
```

```
In [148... # Recriando o modelo com a quantidade de features original (utilizada na comparação com outros modelos)
```

```
vectr = CountVectorizer(max_features=10000, stop_words=stop_words)  
vectr.fit(X_treino)
```

```
D:\LetsCode\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['1k', '1t', '2l', '3d', '3u', '4g', '4m', '5d', '5g', '5m', '6e', '6s', 'a2', 'a9', 'ac', 'az', 'b1', 'b3', 'bl', 'bm', 'bn', 'bt', 'cl', 'ct', 'cz', 'd2', 'd9', 'df', 'dh', 'ds', 'dv', 'e3', 'ec', 'ee', 'f5', 'fh', 'fm', 'fs', 'g2', 'gb', 'ge', 'gj', 'gm', 'gp', 'gs', 'gw', 'hg', 'hl', 'hn', 'ii', 'iu', 'iv', 'ix', 'ia', 'j2', 'jf', 'jn', 'jw', 'jz', 'jà', 'jã', 'jú', 'k9', 'ke', 'kw', 'lc', 'lp', 'ls', 'lw', 'mg', 'mj', 'mk', 'ma', 'nr', 'nw', 'oj', 'ot', 'pb', 'qb', 'qg', 'rb', 'rc', 'rd', 'rh', 'rl', 'rr', 'sh', 'st', 'sá', 'sò', 't1', 'tf', 'tj', 'ts', 'tó', 'u2', 'ub', 'uf', 'uh', 'uo', 'uq', 'ut', 'uv', 'ux', 'vp', 'vt', 'vw', 'vé', 'wd', 'wm', 'wo', 'wt', 'wu', 'xa', 'xi', 'xp', 'xy', 'xô', 'z6', 'zl', 'zn', 'zs', 'ôô', 'eu', 'no'] not in stop_words.
```

```
warnings.warn('Your stop_words may be inconsistent with '
```

```
Out[148]: CountVectorizer(max_features=10000,  
                         stop_words=['!', '!!', '!?', '!"', '!$', "'", ",", "-", "0",  
                         "'3", "'5", "'A", "'E", "'I", "'N", "'O", "'Y",  
                         "'a", "'e", "'n", "'o", "'q", "'À", "'É", "'é",  
                         "'ó", "'ó'", '#', '#1', '#2', ...])
```

```
In [149... # Recriando as bases de treino transformadas
```

```
X_treino_bow = vectr.transform(X_treino).toarray()  
X_valid_bow = vectr.transform(X_valid).toarray()
```

```
In [150... # Execução com os dados de Treino e Validação
```

```
_, tempo_bow_nb, treino_bow_nb, valid_bow_nb, _ = exec_modelo(modelo, X_treino_bow, y_treino, X_valid_bow, y_valid)
```

```
Realizando FIT do modelo...
```

```
Tempo de treinamento do modelo: 80.5056164264679s
```

```
Realizando Predições da base de validacao...
```

```
Realizando Predições da base de treino/teste...
```

```
Resultados do Treinamento:
```

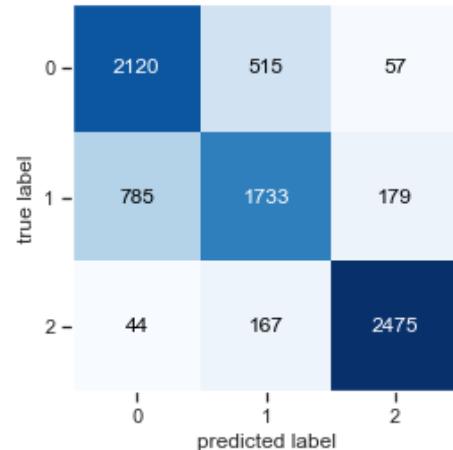
```
Treino...:
```

```
Acc: 0.820, Precision: 0.821, Recall: 0.820
```

```
Validação:
```

```
Acc: 0.784, Precision: 0.783, Recall: 0.784
```

```
<Figure size 864x864 with 0 Axes>
```



Modelo executado.

Não houve melhora nos valores já obtidos.

Explorando um pouco Word Embeddings

```
In [151...]: # As sentenças tokenizadas não precisam ser recriadas a cada teste
sentences = X_treino.apply(lambda t: tokenize(t))

In [152...]: # Não é necessário recodificar os Y's
y_treino_enc = encode_y(y_treino)
y_valid_enc = encode_y(y_valid)

In [153...]: # Definindo o tamanho default do vetor utilizado no modelo de embedding e na rede neural
TAM_VETOR=50

In [154...]: # Parâmetro para truncamento e padding
MAX_LENGTH=400

In [155...]: '''
Função para customizar o modelo W2V e vocabulário, modelo Keras, realizar o fit e predições de treino e validação.
Apresenta as métricas e matriz de confusão ao final
Retorna o tempo de treino, métricas de treino e de validação
'''
def w2v(window=7, min_count=1, alpha=0.025, epochs=5, workers=3, batch_size=100, train_epochs=1):

    print('Inicializando modelo...')
    modelo_w2v = Word2Vec(sentences = sentences,
                          vector_size = TAM_VETOR,
                          window = window,
```

```

        min_count = min_count,
        alpha = alpha,
        epochs = epochs,
        workers = workers)

vocab, id2word = get_vocab(modelo_w2v)

X_treino_seq = X_treino.apply(tokenize).apply(lambda s: seq_to_id(s, vocab))
X_valid_seq = X_valid.apply( tokenize).apply(lambda s: seq_to_id(s, vocab))

X_treino_seq = pad_sequences(X_treino_seq, padding='post', truncating='post', maxlen=MAX_LENGTH)
X_valid_seq = pad_sequences(X_valid_seq , padding='post', truncating='post', maxlen=MAX_LENGTH)

modelo = keras.Sequential(
    [
        layers.Embedding(len(vocab)+1, TAM_VETOR, trainable=True, input_length=MAX_LENGTH, mask_zero=True, name="emb"),
        layers.Lambda(lambda x: keras.backend.mean(x, axis=1)),
        layers.Dense(TAM_VETOR, activation="relu", name="layer2"),
        layers.Dense(1, activation="sigmoid"),
    ]
)
modelo.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss=keras.losses.BinaryCrossentropy(),
    metrics=[keras.metrics.BinaryAccuracy()],
)
inicio = time.time()
modelo.fit(
    X_treino_seq,
    y_treino_enc,
    batch_size=batch_size,
    epochs=train_epochs,
    validation_data=(X_valid_seq, y_valid_enc),
    use_multiprocessing=True
)
fim = time.time()

y_treino_pred = decode_y(modelo.predict(X_treino_seq).reshape(-1))
y_valid_pred = decode_y(modelo.predict(X_valid_seq ).reshape(-1))
treino_we, valid_we, _ = metricas(y_treino, y_treino_pred, y_valid, y_valid_pred)
print('Modelo executado.')

return tempo_we, treino_we, valid_we

```

In [156]:

```
# Teste 1
_, _, _ = w2v(window=5)

Inicializando modelo...
100% | 51941/51941 [00:00<00:00, 584471.10it/s]
```

```
727/727 [=====] - 20s 24ms/step - loss: 0.6065 - binary_accuracy: 0.5097 - val_loss: 0.4669 - val_binary_accuracy: 0.6343
Treino...:
Acc: 0.666, Precision: 0.668, Recall: 0.666
Validação:
Acc: 0.653, Precision: 0.651, Recall: 0.653
Modelo executado.
```

In [157...]

```
# Teste 2
# Acréscimo de 1 época de treinamento do Keras (train_epochs) em relação ao teste 1
_, _, _ = w2v(window=15)
```

Inicializando modelo...

```
100%|██████████| 51941/51941 [00:00<00:00, 1172695.56it/s]
727/727 [=====] - 18s 25ms/step - loss: 0.6274 - binary_accuracy: 0.4887 - val_loss: 0.4925 - val_binary_accuracy: 0.6243
Treino...:
Acc: 0.635, Precision: 0.635, Recall: 0.635
Validação:
Acc: 0.623, Precision: 0.623, Recall: 0.623
Modelo executado.
```

In [158...]

```
# Teste 3
# Diminuição da janela (window) em relação ao teste 1
_, _, _ = w2v(alpha=0.005)
```

Inicializando modelo...

```
100%|██████████| 51941/51941 [00:00<00:00, 1070315.73it/s]
727/727 [=====] - 18s 25ms/step - loss: 0.6023 - binary_accuracy: 0.5185 - val_loss: 0.4595 - val_binary_accuracy: 0.6281
Treino...:
Acc: 0.717, Precision: 0.705, Recall: 0.717
Validação:
Acc: 0.705, Precision: 0.693, Recall: 0.705
Modelo executado.
```

In [159...]

```
# Teste 4
# Aumento da janela (window) em relação ao teste 1
_, _, _ = w2v(batch_size=200)
```

Inicializando modelo...

```
100%|██████████| 51941/51941 [00:00<00:00, 1130728.58it/s]
364/364 [=====] - 13s 31ms/step - loss: 0.6695 - binary_accuracy: 0.4457 - val_loss: 0.5771 - val_binary_accuracy: 0.6183
Treino...:
Acc: 0.578, Precision: 0.669, Recall: 0.578
Validação:
Acc: 0.570, Precision: 0.664, Recall: 0.570
Modelo executado.
```

In [160...]

```
# Teste 5
# Aumento da janela (window) em relação ao teste 1
_, _, _ = w2v(train_epochs=2)
```

Inicializando modelo...

```
100%|██████████| 51941/51941 [00:00<00:00, 1144925.08it/s]
```

```
Epoch 1/2
727/727 [=====] - 19s 25ms/step - loss: 0.6284 - binary_accuracy: 0.4863 - val_loss: 0.4772 - val_binary_accuracy: 0.6348
Epoch 2/2
727/727 [=====] - 19s 26ms/step - loss: 0.4345 - binary_accuracy: 0.6427 - val_loss: 0.4181 - val_binary_accuracy: 0.6451
Treino...
Acc: 0.710, Precision: 0.744, Recall: 0.710
Validação:
Acc: 0.694, Precision: 0.713, Recall: 0.694
Modelo executado.
```

Conclusão:

Mesmo com bastantes esforço, não alcancei melhor performance com Word Embeddings em relação ao Bag Of Words e meu Custom Classifier.
Dados podem variar ligeiramente pela alteração da amostragem em diferentes execuções

Custom Classifier:

```
Treino...
Acc: 0.819, Precision: 0.874, Recall: 0.819
Validação:
Acc: 0.825, Precision: 0.875, Recall: 0.825
```

Bag Of Words Naive Bayes

```
Treino...
Acc: 0.818, Precision: 0.819, Recall: 0.818
Validação:
Acc: 0.793, Precision: 0.793, Recall: 0.793
```

Word Embeddings com Keras

```
Treino...
Acc: 0.746, Precision: 0.762, Recall: 0.746
Validação:
Acc: 0.734, Precision: 0.738, Recall: 0.734
```

A solução com algoritmo mais simples (que decide pela presença de emoticons) alcançou os melhores resultados com os dados do problema.

Execução com a base de Teste

```
df_subm = pd.read_csv('dados\\subm\\Subm3Classes.csv')
df_subm.head(3)
```

```
Out[161]:
```

	id	tweet_text	tweet_date	query_used
0	1046764676707753987	Apartamento Vila Mariana Praça Monteiro dos Sa...	Mon Oct 01 14:12:01 +0000 2018	:)
1	1047329264943751169	@FalleNCS @BrasilGameShow quero 1x1 de scout. ...	Wed Oct 03 03:35:29 +0000 2018	:)
2	1045443874947313665	mais uma analógica no correio à minha espera :...:	Thu Sep 27 22:43:37 +0000 2018	:)

```
In [162...]: # Tratamento dos dados
X_teste = tratamento(df_subm)[ 'tweet_text' ]
```

Aviso: dataframe não contém feature "sentiment": ignorando tratamentos.

```
In [163...]: # Confirmando shape
print(X_teste.shape)
X_teste.head(3)
```

```
(5000,)
0    apartamento vila mariana praça monteiro dos sa...
1    quero 1x1 scout. dizem que dou muita bala ca...
2    mais uma analógica correio minha espera :d fal...
Name: tweet_text, dtype: object
```

```
In [164...]: # Bag of Words com Custom Classifier
modelo = EmoticonClassifier()
```

```
In [165...]: # Fit novamente, para recriar a instância "modelo" sobrescrita nos testes anteriores
modelo.fit(X_treino, y_treino)
```

```
In [166...]: # Execução do modelo
y_pred = modelo.predict(X_teste)
y_pred.shape
```

```
Out[166]:
```

(5000,)

```
In [167...]: # Transformação dos resultados em DataFrame, que facilita inclusive o processo de gravação em arquivo.
df_y_pred = pd.DataFrame(y_pred, columns=[ 'y_pred' ])
```

```
In [168...]: # Confirmando quantidade de resultados
df_y_pred.shape
```

```
Out[168]:
```

(5000, 1)

```
In [169...]: # Dados obtidos
df_y_pred.value_counts()
```

```
Out[169]: y_pred  
2      2524  
0      1304  
1      1172  
dtype: int64
```

```
In [170...]: # Dados obtidos em percentuais  
df_y_pred.value_counts(normalize=True).apply(lambda x : x * 100)
```

```
Out[170]: y_pred  
2      50.48  
0      26.08  
1      23.44  
dtype: float64
```

Salvando o arquivo de resultados

```
In [171...]: arq_dest = 'alexandre_rozante_782_projeto_2_submissao.csv'
```

```
In [172...]: print('Salvando resultados em {}'.format(arq_dest))  
print()  
df_y_pred.to_csv(arq_dest)  
  
print('Arquivo {} criado com sucesso.'.format(arq_dest))  
  
Salvando resultados em alexandre_rozante_782_projeto_2_submissao.csv  
Arquivo alexandre_rozante_782_projeto_2_submissao.csv criado com sucesso.
```

CONCLUSÕES

O modelo tem bom potencial de acerto para o sentimento expressado pelos tweets analisados.

Considerando o modelo com *Custom Classifier*, é uma solução específica. Pode ser adaptada para textos similares, desde que a base de emoticons seja conhecida, e ocorra a devida parametrização. Já considerando BagOfWords, TF-IDF, sim é possível aplicar para outros textos tranquilamente.

Um ponto de atenção é a proximidade de resultados entre BagOfWords e TF-IDF. Dependendo da variação da amostra, os melhores resultados ocorrem em um ou outro.

Independente do modelo, o processamento NLP consome muita CPU e memória. Explorar os textos para minimizar tokens desnecessários é importante. Conhecer a cultura que os envolve é essencial para customizar o tratamento.

Cuidar do consumo de memória, evitando copiar DataFrames e apagar o que for ficando obsoleto é praticamente obrigatório.

Dependendo do cenário, não se deve fazer o que tive que aplicar aqui, reduzindo *features* para conseguir executar o GridSearch. Certamente, contar com CPUs mais potentes é o correto.

Talvez não tenha explorado todo o potencial dos modelos, mas foi interessante constatar a performance do Bag of Words e Naive Bayes. Como antecipou o Helder, é muito difícil superar os resultados que ele produz.

Espero ter alcançado os objetivos propostos pelo projeto.

Obrigado por ter chegado até aqui! Um grande abraço!

Alexandre Rozante

In []: