Exercício 1

A base Reviews.csv possui avaliações de produtos comprados numa grande empresa de e-commerce no Brasil. Além de informações categóricas, a base possui o campo `review_text`, com o texto da avaliação, e o campo `overall_rating`, com a nota dada pelo usuário para determinado produto. Crie um modelo para prever a nota da avaliação do cliente a partir do texto.

Etapas sugeridas:

1 - Faça a divisão da base de teste com pelo menos 20% das amostras (sugestão: out-of-time);

2 - Verifique a existência de dados duplicados ou faltantes;

3 - Crie uma nuvem de palavras para cada nota de avaliação; Verifique a variação das notas ao longo do tempo;

4 - Crie uma representação para a sua coluna de texto. Você pode tentar Bag Of Words, TF-IDF ou o Word2Vec;

5 - Faça o treinamento do modelo. Você pode utilizar uma busca de hiperparâmetros para otimizar seus resultados;

6 - Aplique seu modelo na base de teste para obter seu resultado.

Desafios:

1 - Faça um modelo considerando as demais variáveis no processo;

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('Reviews.csv')

print(df.shape)

df.head()
```

```
(132373, 14)
C:\Users\alexr\AppData\Local\Temp\ipykernel_23300\300059706.py:1: DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or set low_memory
=False.
  df = pd.read_csv('Reviews.csv')
```

Out[2]:

	submission_date	reviewer_id	product_id	product_name	product_brand	site_category_lv1	site_category_lv2	review_title	overall_rating	recomm
0	2018-01-01 00:11:28	d0fb1ca69422530334178f5c8624aa7a99da47907c44de...	132532965	Notebook Asus Vivobook Max X541NA-GO472T Intel...	NaN	Informática	Notebook	Bom	4	
1	2018-01-01 00:13:48	014d6dc5a10aed1ff1e6f349fb2b059a2d3de511c7538a...	22562178	Copo Acrílico Com Canudo 500ml Rocie	NaN	Utilidades Domésticas	Copos, Taças e Canecas	Preço imbatível, ótima qualidade	4	
2	2018-01-01 00:26:02	44f2c8edd93471926fff601274b8b2b5c4824e386ae4f2...	113022329	Panela de Pressão Elétrica Philips Walita Dail...	philips walita	Eletroportáteis	Panela Elétrica	ATENDE TODAS AS EXPECTATIVA.	4	
3	2018-01-01 00:35:54	ce741665c1764ab2d77539e18d0e4f66dde6213c9f0863...	113851581	Betoneira Columbus - Roma Brinquedos	roma jensen	Brinquedos	Veículos de Brinquedo	presente mais que desejado	4	
4	2018-01-01 01:00:28	7d7b6b18dda804a897359276cef0ca252f9932bf4b5c8e...	131788803	Smart TV LED 43" LG 43UJ6525 Ultra HD 4K com C...	lg	TV e Home Theater	TV	Sem duvidas, excelente	5	

In [3]:
```
from sklearn.model_selection import train_test_split
```

In [4]:
```
# Transformando a data em text num valor DateTime
df['submission_date'] = pd.to_datetime(df['submission_date'])
```

In [5]:
```
# out-of-sample

df_experimento, df_teste = train_test_split(df, test_size=.25)

print(df_experimento.shape, df_teste.shape)
```

```
(99279, 14) (33094, 14)
```

Passo 1

Faça a divisão da base de teste com pelo menos 20% das amostras (sugestão: out-of-time)

In [6]:
```
# out-of-time

df_experimento = df[df['submission_date'] < pd.to_datetime('2018-05-01')]
df_teste       = df[df['submission_date'] >= pd.to_datetime('2018-05-01')]

print(f'{len(df_teste)/len(df_experimento):.2f}')

print(df_experimento.shape, df_teste.shape)
```

```
0.25
(105939, 14) (26434, 14)
```

Fim do trecho fornecido pelo Helder

Passo 2

Verifique a existência de dados duplicados ou faltantes

Avaliação de Faltantes

In [7]:
```
df_experimento.count()
```

Out[7]:
```
submission_date          105939
reviewer_id              105939
product_id               105939
product_name             105891
product_brand             34603
site_category_lv1        105934
site_category_lv2        102919
review_title             105713
overall_rating           105939
recommend_to_a_friend    105921
review_text              103373
reviewer_birth_year      101340
reviewer_gender          102836
reviewer_state           102951
dtype: int64
```

Temos várias colunas com dados faltantes.

review_text: Vamos excluir todos. Esse é o dado que precisamos que esteja preenchido para o modelo funcionar.
product_name: Vamos preencher com 'N/I' para faltantes
product_brand: Vamos preencher com 'N/I'

site_category_lv1: Vamos preencher com 'N/I'

site_category_lv2: Vamos preencher com 'N/I'

review_title: Vamos preencher com ''

recommend_to_a_friend: Vamos preencher com 'N/I'

reviewer_birth_year: Vamos preencher com 0

reviewer_gender: Vamos preencher com 'X'

reviewer_state: Vamos preencher com 'XX'

In [8]:
```
def preencher_faltantes(df):
    df = df.copy()
    df = df.dropna(subset=['review_text'])
    df['product_name'].fillna('N/I',inplace=True)
    df['product_brand'].fillna('N/I',inplace=True)
    df['site_category_lv1'].fillna('N/I',inplace=True)
    df['site_category_lv2'].fillna('N/I',inplace=True)
    df['review_title'].fillna('',inplace=True)
    df['recommend_to_a_friend'].fillna('N/I',inplace=True)
    df['reviewer_birth_year'].fillna(0,inplace=True)
    df['reviewer_gender'].fillna('X',inplace=True)
    df['reviewer_state'].fillna('XX',inplace=True)
    return df
```

In [9]:
```
df_experimento = preencher_faltantes(df_experimento)
```

In [10]:
```
df_experimento.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 103373 entries, 0 to 105938
Data columns (total 14 columns):
 #   Column                Non-Null Count   Dtype         
---  ------                --------------   -----         
 0   submission_date       103373 non-null  datetime64[ns]
 1   reviewer_id           103373 non-null  object        
 2   product_id            103373 non-null  object        
 3   product_name          103373 non-null  object        
 4   product_brand         103373 non-null  object        
 5   site_category_lv1     103373 non-null  object        
 6   site_category_lv2     103373 non-null  object        
 7   review_title          103373 non-null  object        
 8   overall_rating        103373 non-null  int64         
 9   recommend_to_a_friend  103373 non-null  object        
 10  review_text           103373 non-null  object        
 11  reviewer_birth_year   103373 non-null  float64       
 12  reviewer_gender       103373 non-null  object        
 13  reviewer_state        103373 non-null  object        
dtypes: datetime64[ns](1), float64(1), int64(1), object(11)
memory usage: 11.8+ MB
```

Análise de duplicidades

In [11]:
```
def remove_dupl(df):
    df = df.copy()
    df = df.drop(axis=1, labels=['reviewer_id','reviewer_birth_year'])
    df = df.drop_duplicates()
    return df
```

In [12]:
```
# Verificação de duplicidades
df_sem_dupl = remove_dupl(df_experimento)
```

In [13]:
```
cd = len(df_experimento)
print('Com publicidades', cd)
sd = len(df_sem_dupl)
print('Sem publicidades', sd)
print('Duplicados:', cd - sd)
```

```
Com publicidades 103373
Sem publicidades 103213
Duplicados: 160
```

In [14]:
```
# Removendo duplicados da base de treino
df_experimento = df_sem_dupl
```

In [15]:
```
# Agora uma análise de duplicidades apenas entre o texto e o overall_rating
df_texto_nota = df_experimento[['review_text', 'overall_rating']]
cd = len(df_texto_nota)

print('Sem duplicidade de texto X rating:', cd)
df_texto_nota = df_texto_nota.drop_duplicates()
sd = len(df_texto_nota)

print('Sem duplicidade de texto X rating:', sd)
print('Duplicados:', cd - sd)
```

```
Sem duplicidade de texto X rating: 103213
Sem duplicidade de texto X rating: 102317
Duplicados: 896
```

Passo 3 - Análise Exploratória

Crie uma nuvem de palavras para cada nota de avaliação; Verifique a variação das notas ao longo do tempo

In [16]:
```
df_experimento['overall_rating'].value_counts()
```

Out[16]:
```
5    38401
4    25951
1    19730
3    12792
2     6339
Name: overall_rating, dtype: int64
```

Nota:

Constatamos que há desbalanceamento da base.

In [17]:
```
!pip install wordcloud
```

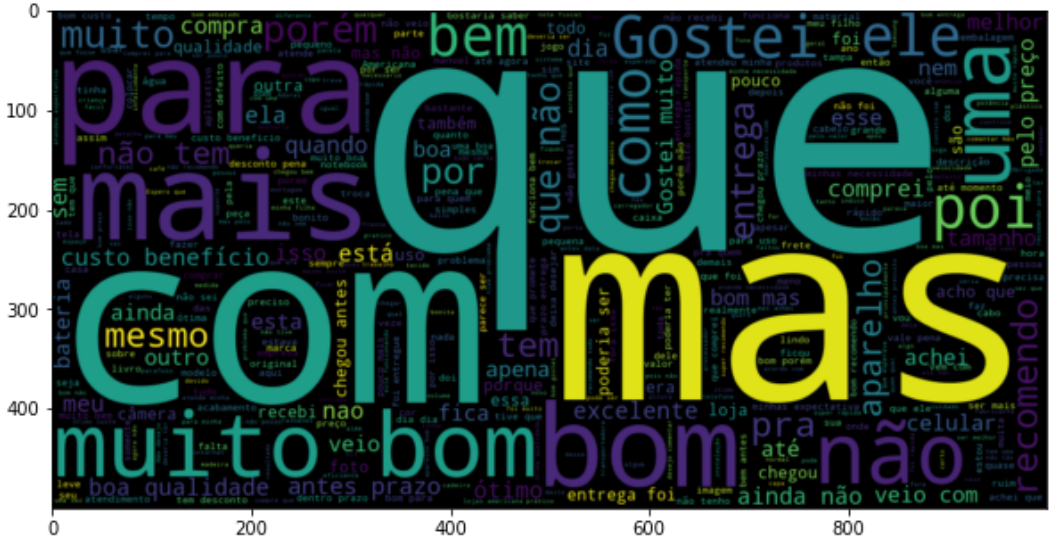
```
Requirement already satisfied: wordcloud in c:\python310\lib\site-packages (1.8.1)
Requirement already satisfied: numpy>=1.6.1 in c:\python310\lib\site-packages (from wordcloud) (1.22.3)
Requirement already satisfied: matplotlib in c:\python310\lib\site-packages (from wordcloud) (3.5.2)
Requirement already satisfied: pillow in c:\python310\lib\site-packages (from wordcloud) (9.1.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\python310\lib\site-packages (from matplotlib->wordcloud) (3.0.7)
Requirement already satisfied: packaging>=20.0 in c:\python310\lib\site-packages (from matplotlib->wordcloud) (21.3)
Requirement already satisfied: cycler>=0.10 in c:\python310\lib\site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\python310\lib\site-packages (from matplotlib->wordcloud) (4.33.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\python310\lib\site-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\python310\lib\site-packages (from matplotlib->wordcloud) (1.4.2)
Requirement already satisfied: six>=1.5 in c:\python310\lib\site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
```

In [18]:
```
from matplotlib import pyplot as plt
from wordcloud import WordCloud, STOPWORDS

def plot_wordcloud(texts):
    # Por observações anteriores, verificamos a presença da palavra "Produto" com ênfase em todos os resultados, bem como
    # dos termos o, a, do, da, um que não agregam muito valor. Configuramos então para remover esses elementos.

    plt.figure(figsize=(10, 10))
    text = ' '.join(texts.tolist())

    WC = WordCloud(width=1000, height=500, max_words=500, min_font_size=5, min_word_length=3, stopwords=['produto'])

    words = WC.generate(text)

    plt.imshow(words, interpolation='bilinear')
    plt.show()
```

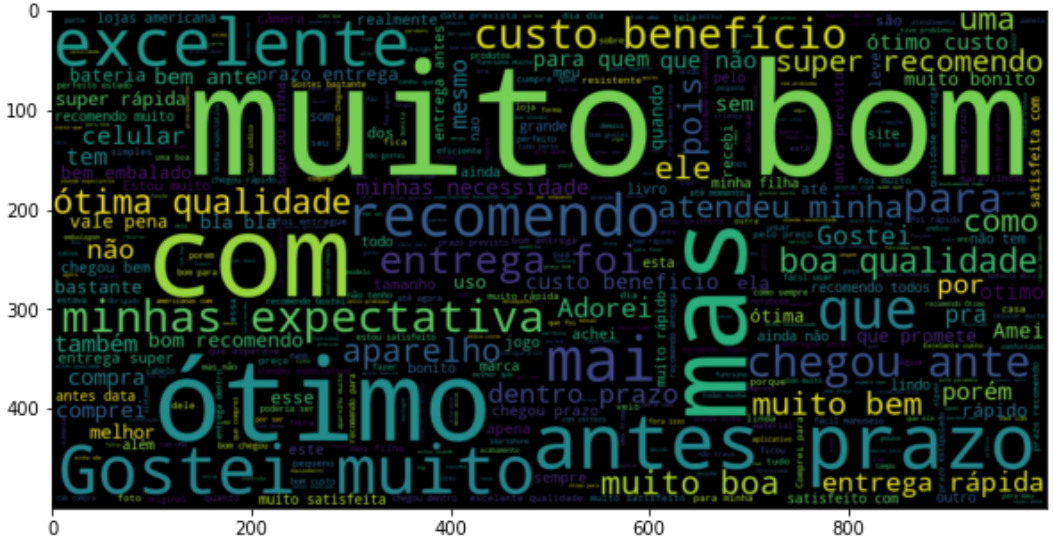
In [19]:
```
plot_wordcloud(df_experimento.loc[df_experimento['overall_rating'] == 1, 'review_text'])
```

In [20]:
```
plot_wordcloud(df_experimento.loc[df_experimento['overall_rating'] == 2, 'review_text'])
```

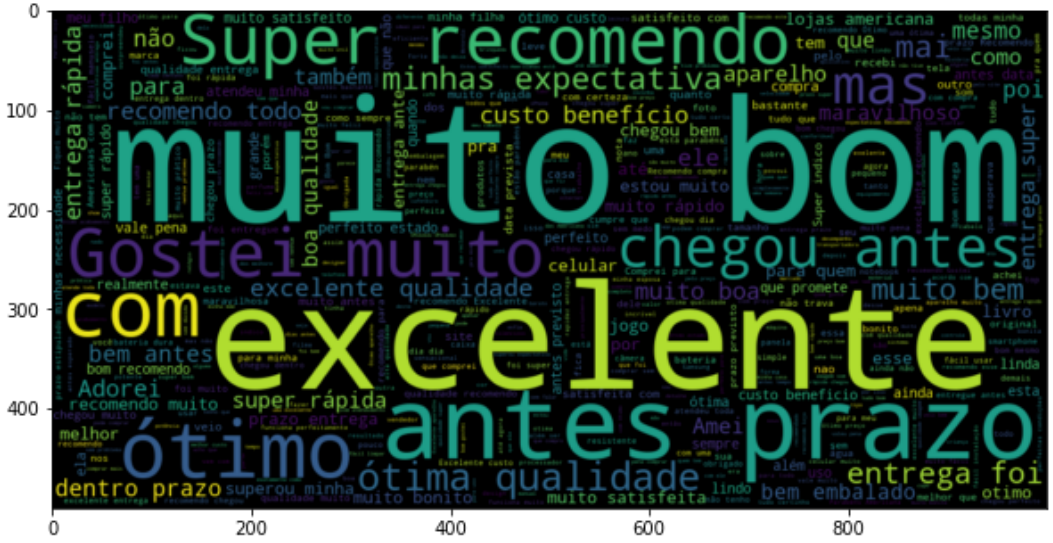
In [21]:
```
plot_wordcloud(df_experimento.loc[df_experimento['overall_rating'] == 3, 'review_text'])
```



In [22]:
```
plot_wordcloud(df_experimento.loc[df_experimento['overall_rating'] == 4, 'review_text'])
```



In [23]:
```
plot_wordcloud(df_experimento.loc[df_experimento['overall_rating'] == 5, 'review_text'])
```



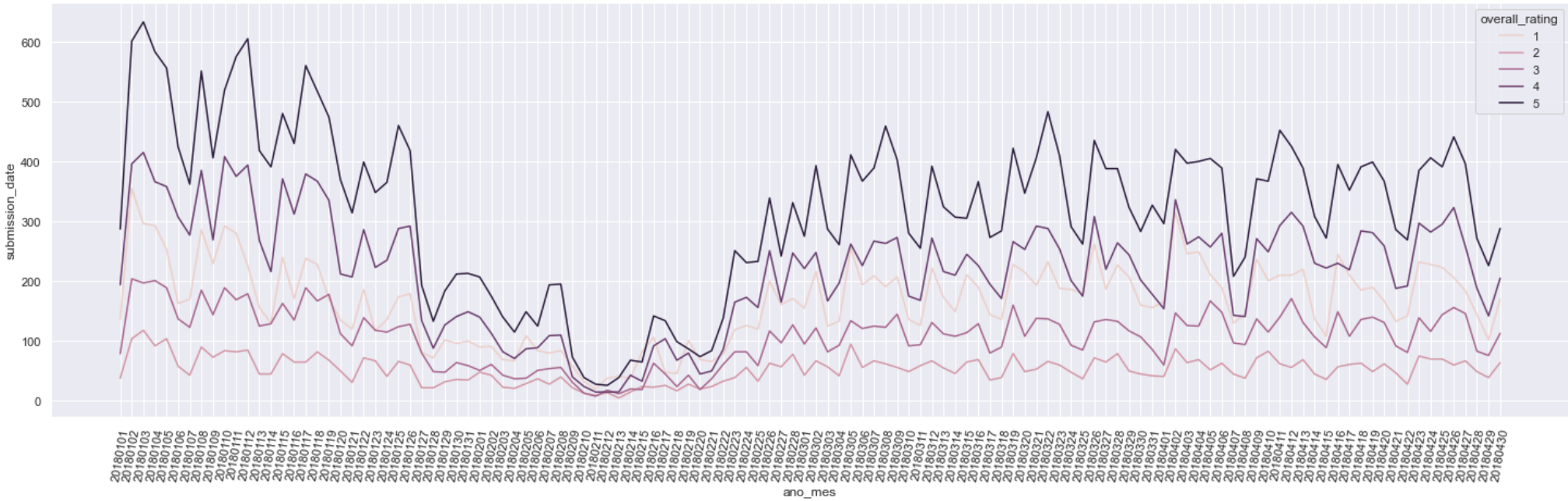
Análise ao longo do tempo

In [24]:
```
import seaborn as sns
sns.set_theme()
sns.set()
```

In [25]:
```
df_experimento['ano_mes'] = df_experimento['submission_date'].dt.strftime('%Y%m%d')
```

In [26]:
```
df_hist = df_experimento[['ano_mes','overall_rating','submission_date']]\
          .groupby(by=['ano_mes','overall_rating']).count().reset_index()
```

In [27]:
```
plt.figure(figsize=(25,7))
sns.lineplot(data=df_hist, x='ano_mes', y='submission_date', hue='overall_rating')
plt.xticks(rotation=80)
plt.show()
```



Passo 4

Crie uma representação para a sua coluna de texto. Você pode tentar Bag Of Words, TF-IDF ou o Word2Vec

In [28]:
```
df_train, df_valid = train_test_split(df_experimento, test_size=.2)

x_train, y_train = df_train['review_text'], df_train['overall_rating']
x_valid, y_valid = df_valid['review_text'], df_valid['overall_rating']

print(x_train.shape, y_train.shape)
print(x_valid.shape, y_valid.shape)
```

```
(82570,) (82570,)
(20643,) (20643,)
```

Função para exibir as métricas

In [29]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

In [30]:
```
def metricas(y_train, y_train_pred, y_valid=None, y_valid_pred=None):

    acc_train  = accuracy_score(y_train, y_train_pred)
    prec_train = precision_score(y_train, y_train_pred, average='weighted')
    rec_train  = recall_score(y_train, y_train_pred, average='weighted')
```

```
    if (y_valid is None):
        print(f'Teste:\nAcc: {acc_train:.3f}, Precision: {prec_train:.3f}, Recall: {rec_train:.3f}')
    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')

        print(f'Treino:\nAcc: {acc_train:.3f}, Precision: {prec_train:.3f}, Recall: {rec_train:.3f}')
        print(f'Validação:\nAcc: {acc_valid:.3f}, Precision: {prec_valid:.3f}, Recall: {rec_valid:.3f}')
```

Função para exibir o shape das bases de treino e validação

In [31]:
```
def shapes(treino, validacao):
    print('Treino...:', treino.shape)
    print('Validacao:', validacao.shape)
```

Função para exibir a matriz de confusão

In [32]:
```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
```

In [33]:
```
def matriz_conf(y, y_pred):
    cm = confusion_matrix(y, y_pred)
    plt.figure(figsize=(8,8))
    plot_confusion_matrix(conf_mat=cm)
    plt.show()
```

BagOfWords

In [34]:
```
# BagOfWords
from sklearn.feature_extraction.text import CountVectorizer

# Tamanho do vocabulário
vectorizer = CountVectorizer(max_features=10000, strip_accents='unicode',\
                             stop_words=['produto','a','e','i','o','u','da','de','do','as','os','se','um'])
vectorizer.fit(x_train)
```

Out[34]:
```
CountVectorizer(max_features=10000,
                stop_words=['produto', 'a', 'e', 'i', 'o', 'u', 'da', 'de',
                            'do', 'as', 'os', 'se', 'um'],
                strip_accents='unicode')
```

In [35]:
```
x_train_transformed = vectorizer.transform(x_train).toarray()
x_valid_transformed = vectorizer.transform(x_valid).toarray()
shapes(x_train_transformed, x_valid_transformed)
```

```
Treino...: (82570, 10000)
Validacao: (20643, 10000)
```

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
```

In [37]:
```
modelo_bow = MultinomialNB()
```

In [ ]:
```
# Treino do modelo
modelo_bow.fit(x_train_transformed, y_train)
```

In [ ]:
```
y_train_pred_bow = modelo_bow.predict(x_train_transformed)
y_valid_pred_bow = modelo_bow.predict(x_valid_transformed)

shapes(y_train_pred_bow, y_valid_pred_bow)
```

In [ ]:
```
metricas(y_train, y_train_pred_bow, y_valid, y_valid_pred_bow)
```

In [ ]:
```
matriz_conf(y_valid, y_valid_pred_bow)
```

In [ ]:
```
modelo_bow = None
x_train_transformed = None
x_valid_transformed = None
y_train_pred_bow = None
y_valid_pred_bow = None
```

TF-IDF

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=10000)
vectorizer.fit(x_train)
```

In [ ]:
```
x_train_transformed = vectorizer.transform(x_train).toarray()
x_valid_transformed = vectorizer.transform(x_valid).toarray()

shapes(x_train_transformed, x_valid_transformed)
```

In [ ]:
```
modelo_tf = MultinomialNB()
```

In [ ]:
```
modelo_tf.fit(x_train_transformed, y_train)
```

In [ ]:
```
y_train_pred_tidf = modelo_tf.predict(x_train_transformed)
y_valid_pred_tidf = modelo_tf.predict(x_valid_transformed)

shapes(y_train_pred_tidf, y_valid_pred_tidf)
```

In [ ]:
```
metricas(y_train, y_train_pred_tidf, y_valid, y_valid_pred_tidf)
```

In [ ]:
```
matriz_conf(y_valid, y_valid_pred_tidf)
```

In [ ]:
```
modelo_tf = None
x_train_transformed = None
x_valid_transformed = None
y_train_pred_tidf = None
y_valid_pred_tidf = None
```

Word Embeddings

In [ ]:
```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from gensim.utils import simple_preprocess
```

Função para quebrar texto em suas palavras (tokens)

In [ ]:
```
def tokenize(text):
    return simple_preprocess(text, min_len=1)
```

Normalizando as bases para minúsculas

In [ ]:
```
x_train = x_train.str.lower()
x_valid = x_valid.str.lower()

shapes(x_train, x_valid)
```

Criando os tokens

In [ ]:
```
from gensim.models import Word2Vec

modelo_w2v = Word2Vec(sentences=x_train.apply(lambda t: tokenize(t)),
                      vector_size=300,
                      window=5,
                      min_count=1,
                      workers=8)
```

In [ ]:
```
# Exibindo o tamanho do vocabulário criado
len(modelo_w2v.wv)
```

In [ ]:
```
# Plotando um gráfico das palavras

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

def plot_words(words, model_w2v):
    vocab = [word for word in model_w2v.wv.key_to_index]

    X = model_w2v.wv.vectors

    pca = PCA(n_components=2)
    X_reduct = pca.fit_transform(X)

    df = pd.DataFrame(X_reduct, index=vocab, columns=['x', 'y'])

    if type(words) == int:
        df = df.sample(min(words, len(df)))
    else:
        df = df.loc[words]

    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(1, 1, 1)

    ax.scatter(df['x'], df['y'])

    for word, pos in df.iterrows():
        ax.annotate(word, pos)

    plt.show()
```

In [ ]:
```
plot_words(10, modelo_w2v)
```

Indexando os tokens e criando um DataFrame com as features

In [ ]:
```
from tqdm import tqdm

''' Retorna um vetor para um conjunto de palavras em uma sentença '''
def to_word_vector(words, w2v):

    vectors = []
    for word in words:
        vectors.append(w2v.wv[word] if word in w2v.wv.key_to_index else np.zeros(w2v.vector_size))

    if not vectors:
        vectors.append(np.zeros(w2v.vector_size))
```

```
    return np.mean(vectors, axis=0)

''' Popula um array NumPy com os vetores de todas as sentenças '''
def embeddings(x, w2v):
    x = x.copy()
    x = x.apply(tokenize)
    x = np.vstack([[to_word_vector(sentences, w2v)] for sentences in tqdm(x)])

    return x
```

Criando os novos DataFrames de treino e validação

In [ ]:
```
x_train_trans = embeddings(x_train, modelo_w2v)
```

In [ ]:
```
x_valid_trans = embeddings(x_valid, modelo_w2v)
```

Utilizando o modelo escolhido

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

In [ ]:
```
modelo = LGBMClassifier(n_estimators=200, max_depth=3, learning_rate=0.05, random_state=12)
modelo
```

Treino do modelo

In [ ]:
```
modelo.fit(x_train_trans, y_train)
```

Obtendo os resultados

In [ ]:
```
y_train_pred = modelo.predict(x_train_trans)
y_valid_pred = modelo.predict(x_valid_trans)
```

Shapes e métricas

In [ ]:
```
shapes(y_train_pred, y_valid_pred)
```

In [ ]:
```
metricas(y_train, y_train_pred, y_valid, y_valid_pred)
```

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

Passo 5

Faça o treinamento do modelo. Você pode utilizar uma busca de hiperparâmetros para otimizar seus resultados.

Com o treinamento já realizado dos modelos nos passos anteriores, constatei que o Word Embeddings teve o melhor desempenho. Testando alguns hiperparâmetros para tentar melhor a performance.

In [ ]:
```
def hipers(w2v_sents, w2v_vector_size, w2v_window, w2v_min_count, w2v_alpha, lgbm_n_estims, lgbm_mxdepth, lgbm_lrate):

    modelo_w2v = Word2Vec(sentences=w2v_sents,
                          vector_size=w2v_vector_size,
                          window=w2v_window,
                          min_count=w2v_min_count,
                          alpha=w2v_alpha,
                          workers=8)

    x_train_trans = embeddings(x_train, modelo_w2v)
    x_valid_trans = embeddings(x_valid, modelo_w2v)

    modelo = LGBMClassifier(n_estimators=lgbm_n_estims, max_depth=lgbm_mxdepth, learning_rate=lgbm_lrate, random_state=12)
    print("Fit do LGBM em andamento...")
    modelo.fit(x_train_trans, y_train)

    y_train_pred = modelo.predict(x_train_trans)
    y_valid_pred = modelo.predict(x_valid_trans)

    metricas(y_train, y_train_pred, y_valid, y_valid_pred)

    return modelo_w2v, modelo
```

In [ ]:
```
tokens = x_train.apply(lambda t: tokenize(t))
```

In [ ]:
```
# Performance inicial
hipers(tokens, 300, 5, 1, 0.025, 200, 3, 0.05)
```

In [ ]:
```
# Teste 1 - reduzindo w2v_vector_size
hipers(tokens, 50, 5, 1, 0.025, 200, 3, 0.05)
```

Houve piora das métricas

In [ ]:
```
# Teste 3 - aumentando w2v_vector_size
hipers(tokens, 400, 5, 1, 0.025, 200, 3, 0.05)
```

Houve piora das métricas

In [ ]:
```
# Teste 3 - reduzindo w2v_alpha
```

```
hipers(tokens, 300, 5, 1, 0.015, 200, 3, 0.05)
```

Houve piora das métricas

In [ ]:
```
# Teste 4 - aumentando w2v_alpha
hipers(tokens, 300, 5, 1, 0.075, 200, 3, 0.05)
```

Houve melhora das métricas

In [ ]:
```
# Teste 5 - aumentando ainda mais w2v_alpha
hipers(tokens, 300, 5, 1, 0.1, 200, 3, 0.05)
```

Houve piora das métricas

In [ ]:
```
# Teste 6 - aumentando lgbm_n_estimators
hipers(tokens, 300, 5, 1, 0.075, 300, 3, 0.05)
```

Houve melhora das métricas

In [ ]:
```
# Teste 7 - aumentando lgbm_max_depth
hipers(tokens, 300, 5, 1, 0.075, 300, 5, 0.05)
```

Houve melhora das métricas

In [ ]:
```
# Teste 8 - aumentando lgbm_learning_rate
hipers(tokens, 300, 5, 1, 0.075, 300, 5, 0.10)
```

Houve melhora das métricas, mas também do overfitting

In [ ]:
```
# Teste 9 - diminuindo w2v_window
hipers(tokens, 300, 3, 1, 0.075, 300, 5, 0.10)
```

Houve piora das métricas

In [ ]:
```
# Teste 10 - aumentando um pouco mais alguns hiperparâmetros na direção em que houve melhora das métricas
hipers(tokens, 350, 7, 1, 0.080, 350, 6, 0.15)
```

Houve melhora das métricas, mas o overfitting cresceu muito

In [ ]:
```
# Teste 11 - aumentando um pouco mais alguns hiperparâmetros, buscando evitar o overfitting excessivo.
hipers(tokens, 400, 8, 1, 0.080, 350, 4, 0.075)
```

In [ ]:
```
# Teste 12 - aumentando um pouco mais alguns hiperparâmetros, buscando evitar o overfitting excessivo.
hipers(tokens, 400, 9, 1, 0.080, 300, 4, 0.075)
```

In [ ]:
```
# Teste 13 - equilibrando melhor o overfitting do modelo
modelo_w2v, modelo = hipers(tokens, 350, 8, 1, 0.075, 300, 3, 0.05)
```

Passo 6

Aplique seu modelo na base de teste para obter seu resultado

In [ ]:
```
# Preenchimento de dados faltantes de entrada
df_tst = preencher_faltantes(df_teste)
```

In [ ]:
```
# Separação da variável-alvo
x_tst, y_tst = df_tst['review_text'], df_tst['overall_rating']
```

In [ ]:
```
# Normalização das palavras
x_tst = x_tst.str.lower()
```

In [ ]:
```
# Word Embedding
x_tst_trans = embeddings(x_tst, modelo_w2v)
```

In [ ]:
```
# Predição
y_tst_pred = modelo.predict(x_tst_trans)
```

In [ ]:
```
# Métricas
metricas(y_tst, y_tst_pred)
```

In [ ]:
```
# Matriz de confusão
matriz_conf(y_tst, y_tst_pred)
```

Resultados finais:

```
Treino:
Acc: 0.616, Precision: 0.591, Recall: 0.616
Validação:
Acc: 0.583, Precision: 0.542, Recall: 0.583
Teste:
Acc: 0.579, Precision: 0.533, Recall: 0.579
```

In [ ]: