1 - Carrega informações do dataframe

2 - Criação do encoder e da matriz binária para cada coluna que iremos usar para classificação

3 - gera o corpus (bag of words) e vetoriza

4 - Faz a transformação TFIDF

5 - Cria bases de treinamento e validação

6 - Cria modelo

7 - Analisa a acurária e a perda de cada modelo

8 - Cria predições para cada modelo treinado

9 - Aplica o modelo na base completa e analisa resultados

In [1]:

```
import pandas as pd
import numpy as np
import random
import os
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder, LabelBinarizer, OneHotEncoder
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfTransformer
from collections import namedtuple
from typing import Dict
```

1 - Carrega informações do dataframe

1.1 - dados de descrição limpa (com stopwords e com afixos)

In [2]:

```
df_itens = pd.read_parquet('itens_desc_limpa_sem_stopwords_stemming.parquet')
df_tec = pd.read_parquet('2_tec_desc_limpa.parquet')
```

In [3]:

```
len(df_itens), len(df_tec)
```

Out[3]:

```
(26115, 10147)
```

In [4]:

```
df_itens[df_itens['capitulo'] == '00']
```

Out[4]:

descricao_limpa_sem_stopwords_stemming	capitulo	posicao	subposicao	item	subitem

In [5]:

```
# Duplicar linhas com somente 1 ou 2 exemplos
df_itens = df_itens.append(df_itens[df_itens['capitulo'].map(df_itens['capitulo'].value_cou
```

In [6]:

```
df_itens = df_itens.rename(columns={'descricao_limpa_sem_stopwords_stemming': 'descricao_li
```

In [7]:

```
df_itens.head()
```

Out[7]:

	descricao_limpa	capitulo	posicao	subposicao	item	subitem
0	masc fac hidrat embal 25ml day dre miracl vali...	33	04	99	1	0
1	dioctil ftalat flex bag d 20 tonel metr diocty...	29	17	32	0	0
2	sol calc borrach belfast mx	64	06	20	0	0
3	sol calc borrach lyon mx	64	06	20	0	0
4	sol calc borrach lyon mx	64	06	20	0	0

In [8]:

```
df_itens[df_itens['capitulo'] == '99']
```

Out[8]:

	descricao_limpa	capitulo	posicao	subposicao	item	subitem
4441889		99	99	99	9	9
4337496	seam dron import	99	99	99	9	9
4441889		99	99	99	9	9
4337496	seam dron import	99	99	99	9	9

In [9]:

```
len(df_itens)
```

Out[9]:

```
26117
```

In [10]:

```
# apaga linhas vazias
df_itens = df_itens.drop(df_itens[df_itens['descricao_limpa'] == ''].index)
```

In [11]:

```
len(df_itens)
```

Out[11]:

```
26104
```

In [12]:

```
df_tec.head()
```

Out[12]:

	descricao	ncm	ncm_str	capitulo	posicao	subposicao	item	subitem	descricao_
0	Reprodutores de raca pura Cavalos Cavalos as...	1012100.0	01012100	01	01	21	0	0	reproduto raca pura c cavalos
1	Outros Cavalos Cavalos asininos e muares vi...	1012900.0	01012900	01	01	29	0	0	outros c cavalos as e muares
2	Asininos Cavalos asininos e muares vivos	1013000.0	01013000	01	01	30	0	0	asininos c asir muares
3	Outros Cavalos asininos e muares vivos	1019000.0	01019000	01	01	90	0	0	outros c asir muares
4	Prenhes ou com cria ao pe Reprodutores de raca...	1022110.0	01022110	01	02	21	1	0	prenhes c cria reproduto

In [13]:

```
df_tec = df_tec[['capitulo', 'posicao', 'subposicao', 'item', 'subitem']]
```

In [14]:

```
df_tec.head()
```

Out[14]:

	capitulo	posicao	subposicao	item	subitem
0	01	01	21	0	0
1	01	01	29	0	0
2	01	01	30	0	0
3	01	01	90	0	0
4	01	02	21	1	0

2 - Criação do encoder e da matriz binária para cada coluna que iremos usar para classificação.

In [15]:

```
Encoders = namedtuple('Encoders', 'encoder binarizer')

def encode_fields(df, fields: list) -> Dict[str, Encoders]:
    result = {}
    for i, field in enumerate(fields):
        lblencoder = LabelEncoder()  # cria um número para cada categoria
        lblbinarizer = LabelBinarizer()  # one hot encoder (ex: 99categorias cria matriz co
        encoded = lblencoder.fit_transform(df[field].values)  # transforma os dados do arra
        # dessa forma, retorna um array com as categorias na forma numérica, começando em z
        print(f'field: {field} / encoded shape: {encoded.shape}')
        binarized = lblbinarizer.fit_transform(encoded)  # transforma os dados do array "fi
        # dados binários (zeros e uns) para cada categoria, então retorna matriz m x n, ond
        # de linhas do array de entrada e n é a quantidade de colunas de categorias tranfor
        # forma binária
        print(f'field: {field} / binarized shape: {binarized.shape}')
        encoders = Encoders(lblencoder, lblbinarizer)
        result[field] = encoders
    return result
```

In [16]:

```
encoders = encode_fields(df_itens, ['capitulo', 'posicao', 'subposicao', 'item', 'subitem']
```

```
field: capitulo / encoded shape: (26104,)
field: capitulo / binarized shape: (26104, 97)
field: posicao / encoded shape: (26104,)
field: posicao / binarized shape: (26104, 90)
field: subposicao / encoded shape: (26104,)
field: subposicao / binarized shape: (26104, 91)
field: item / encoded shape: (26104,)
field: item / binarized shape: (26104, 10)
field: subitem / encoded shape: (26104,)
field: subitem / binarized shape: (26104, 10)
```

2.1 - encode da coluna "capítulo"

In [18]:

```
y_encoded_cap = encoders['capitulo'].encoder.transform(df_itens.capitulo.values)
y_encoded_pos = encoders['posicao'].encoder.transform(df_itens.posicao.values)
y_encoded_subpos = encoders['subposicao'].encoder.transform(df_itens.subposicao.values)
y_encoded_item = encoders['item'].encoder.transform(df_itens.item.values)
y_encoded_subitem = encoders['subitem'].encoder.transform(df_itens.subitem.values)
```

In [19]:

```
print(f'formato do array "y_encoded_cap": {y_encoded_cap.shape} linhas, \nconteúdo: \n{y_en
print(f'formato do array "y_encoded_pos": {y_encoded_pos.shape} linhas, \nconteúdo: \n{y_en
print(f'formato do array "y_encoded_subpos": {y_encoded_subpos.shape} linhas, \nconteúdo: \
print(f'formato do array "y_encoded_item": {y_encoded_item.shape} linhas, \nconteúdo: \n{y_
print(f'formato do array "y_encoded_subitem": {y_encoded_subitem.shape} linhas, \nconteúdo:
```

```
formato do array "y_encoded_cap": (26104,) linhas,
conteúdo:
[32 28 63 ... 83 83 96]
formato do array "y_encoded_pos": (26104,) linhas,
conteúdo:
[ 3 16  5 ... 26 27 89]
formato do array "y_encoded_subpos": (26104,) linhas,
conteúdo:
[90 23 11 ... 12 62 90]
formato do array "y_encoded_item": (26104,) linhas,
conteúdo:
[1 0 0 ... 0 9 9]
formato do array "y_encoded_subitem": (26104,) linhas,
conteúdo:
[0 0 0 ... 0 0 9]
```

2.2 - encode binário da coluna "capítulo" - gera matriz

In [20]:

```
y_cap = encoders['capitulo'].binarizer.fit_transform(y_encoded_cap)
y_pos = encoders['posicao'].binarizer.fit_transform(y_encoded_pos)
y_subpos = encoders['subposicao'].binarizer.fit_transform(y_encoded_subpos)
y_item = encoders['item'].binarizer.fit_transform(y_encoded_item)
y_subitem = encoders['subitem'].binarizer.fit_transform(y_encoded_subitem)
y_todos = [y_cap, y_pos, y_subpos, y_item, y_subitem]
```

In [21]:

```
print(f'formato da matriz "y_cap": {y_cap.shape} (linhas, colunas),\nconteúdo: \n{y_cap}')
print(f'formato da matriz "y_pos": {y_pos.shape} (linhas, colunas),\nconteúdo: \n{y_pos}')
print(f'formato da matriz "y_subpos": {y_subpos.shape} (linhas, colunas),\nconteúdo: \n{y_s
print(f'formato da matriz "y_item": {y_item.shape} (linhas, colunas),\nconteúdo: \n{y_item}
print(f'formato da matriz "y_subitem": {y_subitem.shape} (linhas, colunas),\nconteúdo: \n{y
```

```
formato da matriz "y_cap": (26104, 97) (linhas, colunas),
conteúdo:
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]
formato da matriz "y_pos": (26104, 90) (linhas, colunas),
conteúdo:
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]
formato da matriz "y_subpos": (26104, 91) (linhas, colunas),
conteúdo:
[[0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]
formato da matriz "y_item": (26104, 10) (linhas, colunas),
conteúdo:
[[0 1 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 ...
 [1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]]
formato da matriz "y_subitem": (26104, 10) (linhas, colunas),
conteúdo:
[[1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 ...
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]
```

3 - gera o corpus (bag of words) e vetoriza

In [22]:

```
corpus = df_itens.descricao_limpa.values  # transforma todo o texto em bag of words
vectorizer = CountVectorizer(max_df=0.1, min_df=0.00001)  # elimina palavras mto ou pouco f
X_counts = vectorizer.fit_transform(corpus)  # aprende o dicionário de vocabulários e gera
X_counts.shape
```

Out[22]:

```
(26104, 19871)
```

In [23]:

```
X_pos = y_pos
X_pos.shape
```

Out[23]:

```
(26104, 90)
```

In [24]:

```
X_subpos = y_subpos
X_subpos.shape
```

Out[24]:

```
(26104, 91)
```

In [25]:

```
X_item = y_item
X_item.shape
```

Out[25]:

```
(26104, 10)
```

In [26]:

```
X_subitem = y_subitem
X_subitem.shape
```

Out[26]:

```
(26104, 10)
```

limpa memória excluindo variável que não será mais utilizada

In [27]:

```
# del vectorizer
del corpus  # apaga o corpus que gerou X_counts
```

4 - Faz a transformação TFIDF - realiza o cálculo da frequência relativa das palavras multiplicando por um peso, de forma a diminuir as palavras muito frequêntes e as raras.

In [28]:

```
transformer = TfidfTransformer()  # transforma em matrix TFIDF - faz freq relativa multipli
# peso nas palavras freq ou raras - ou seja, deixa de ser zero e 1.
X_tf = transformer.fit_transform(X_counts)
```

In [29]:

```
X_tf.shape
```

Out[29]:

```
(26104, 19871)
```

5 - Cria bases de treinamento e validação

5.1 - a base de teste representa 5% do dataset e está estratificada conforme os rótulos de "y" (matriz binária)

In [30]:

```
X_train_cap, X_val_cap, y_train_cap, y_val_cap = train_test_split(X_tf, y_cap, test_size=0.
```

In [31]:

```
# não funcionou stratify
X_train_pos, X_val_pos, y_train_pos, y_val_pos = train_test_split(X_tf, y_pos, test_size=0.
```

In [32]:

```
# não funcionou stratify
X_train_subpos, X_val_subpos, y_train_subpos, y_val_subpos = train_test_split(X_tf, y_subpo
```

In [33]:

```
X_train_item, X_val_item, y_train_item, y_val_item = train_test_split(X_tf, y_item, test_si
```

In [34]:

```
X_train_subitem, X_val_subitem, y_train_subitem, y_val_subitem = train_test_split(X_tf, y_s
```

In [35]:

```
X_train_todos = [X_train_cap, X_train_pos, X_train_subpos, X_train_item, X_train_subitem]
X_val_todos = [X_val_cap, X_val_pos, X_val_subpos, X_val_item, X_val_subitem]
y_train_todos = [y_train_cap, y_train_pos, y_train_subpos, y_train_item, y_train_subitem]
y_val_todos = [y_val_cap, y_val_pos, y_val_subpos, y_val_item, y_val_subitem]
```

In [36]:

```
for X_train in X_train_todos:
    print(f'Treinando com {X_train.shape[0]} exemplos da base e {X_train.shape} palavras di
```

```
Treinando com 24798 exemplos da base e (24798, 19871) palavras diferentes
Treinando com 24798 exemplos da base e (24798, 19871) palavras diferentes
Treinando com 24798 exemplos da base e (24798, 19871) palavras diferentes
Treinando com 24798 exemplos da base e (24798, 19871) palavras diferentes
Treinando com 24798 exemplos da base e (24798, 19871) palavras diferentes
```

In [37]:

```
for X_val in X_val_todos:
    print(f'Validando com {X_val.shape[0]} exemplos da base e {X_val.shape} palavras difere
```

```
Validando com 1306 exemplos da base e (1306, 19871) palavras diferentes
Validando com 1306 exemplos da base e (1306, 19871) palavras diferentes
Validando com 1306 exemplos da base e (1306, 19871) palavras diferentes
Validando com 1306 exemplos da base e (1306, 19871) palavras diferentes
Validando com 1306 exemplos da base e (1306, 19871) palavras diferentes
```

In [38]:

```
for y_train in y_train_todos:
    print(f'y_train com {y_train.shape[0]} exemplos da base e {y_train.shape} palavras dife
```

```
y_train com 24798 exemplos da base e (24798, 97) palavras diferentes
y_train com 24798 exemplos da base e (24798, 90) palavras diferentes
y_train com 24798 exemplos da base e (24798, 91) palavras diferentes
y_train com 24798 exemplos da base e (24798, 10) palavras diferentes
y_train com 24798 exemplos da base e (24798, 10) palavras diferentes
```

In [39]:

```
for y_val in y_val_todos:
    print(f'y_val com {y_val.shape[0]} exemplos da base e {y_val.shape} palavras diferentes
```

```
y_val com 1306 exemplos da base e (1306, 97) palavras diferentes
y_val com 1306 exemplos da base e (1306, 90) palavras diferentes
y_val com 1306 exemplos da base e (1306, 91) palavras diferentes
y_val com 1306 exemplos da base e (1306, 10) palavras diferentes
y_val com 1306 exemplos da base e (1306, 10) palavras diferentes
```

In [40]:

```
# cria dicionário com todos os modelos por coluna
setups = {}
colunas = ['cap', 'pos', 'subpos', 'item', 'subitem']
for i, coluna in enumerate(colunas):
    setups[coluna] = [X_train_todos[i], X_val_todos[i], y_train_todos[i], y_val_todos[i]]
```

In [41]:

```
#### limpa memória excluindo variavéis que não serão mais utilizadas
```

In [42]:

```
del X_tf  # apaga o resultado do TFIDF que foi utilizado para criar a base de teste e valid
```

In [43]:

```
del X_counts  # apaga o X_count que originou o X_tf
```

6 - Cria modelo

6.1 - Cria modelo classificador usando 2 camadas full connected (densidade passada por parâmetro 256 ou 512 neurônios e ativador "relu") com dropout passado por parâmetro sendo de 20% ou de 40%, para reduzir overfitting

obs: foram utilizadas 2 camadas pois com duas camadas é suficiente para identificamos relações não lineraes, mais de duas camadas teríamos que treinar o modelo muitas vezes o que tornaria mais complexo.

In [44]:

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import backend as K

def model1(input_size, output_size, optimizer='adam', dropout=0.4, dense=128):  # "adam" co
    # e tempo, é mais usado (SGD, SGD com momentum)
    model = tf.keras.Sequential()
    model.add(layers.Input(input_size))
    model.add(layers.Dense(dense, activation='relu'))  # ativador do neuronio função relu
    model.add(layers.Dropout(dropout))  # a cada passada ignora 40% dos neurônios
    model.add(layers.Dense(dense, activation='relu'))
    model.add(layers.Dropout(dropout))
    model.add(layers.Dense(output_size, activation='softmax'))  # coleção de 0, 1 do sigmoi
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',  # pega o softmax, onde tá zero penaliza
                  metrics=['accuracy'])

    return model
```

6.2 - Cria modelos com dropouts e densidades diferentes

In [45]:

```
# melhor configuração 256 neuronios e dropout 0.2
models = {}
dense = 256
dropout = 0.2
i = 0
for k, setup in setups.items():
    print(f'model {k}: Xtrain: {setup[0].shape[1]} e y_train: {setup[2].shape[1]} ')
    models[k] = model1(
            setup[0].shape[1],
            setup[2].shape[1],
            optimizer=tf.keras.optimizers.Adam(lr=0.001),
            dropout=dropout, dense=dense)
```

```
model cap: Xtrain: 19871 e y_train: 97
model pos: Xtrain: 19871 e y_train: 90
model subpos: Xtrain: 19871 e y_train: 91
model item: Xtrain: 19871 e y_train: 10
model subitem: Xtrain: 19871 e y_train: 10
```

6.3 - treina esses modelos sendo para cada modelo roda 40 epócas, dividindo a entrada em chunks de tamanho 512, para cada época e com um learning rate decrescente

In [46]:

```
from collections import defaultdict
import math

epochs = 40
# batch_size = 256
batch_size = 512
for key, setup in setups.items():
    rounds = setup[0].shape[0] // batch_size  +  1
    history = defaultdict(list)
    X_val_array = setup[1].toarray()

    print(f'\n\n modelo: {key} \n\n')
    for i in range(epochs):
        lr = 0.001 / (math.sqrt(i) + 1)
        print(f'Epoch {i} learning rate {lr}')
        K.set_value(models[key].optimizer.lr, lr)
        for batch_number in range(rounds):
            start = batch_number * batch_size
            X_chunk = setup[0][start: start + batch_size].toarray()
            y_chunk = setup[2][start: start + batch_size]
            models[key].train_on_batch(X_chunk, y_chunk)  # treina o modelo efetivamente
            if batch_number % 100 == 0.:
                print(f'Batch n.: {batch_number} de {rounds}')
                loss_acc = models[key].evaluate(X_chunk, y_chunk)
                history['train_loss'].append(loss_acc[0])
                history['train_acc'].append(loss_acc[1])
                val_loss_acc = models[key].evaluate(X_val_array, setup[3])
                history['val_loss'].append(val_loss_acc[0])
                history['val_acc'].append(val_loss_acc[1])
                # print('loss: {:0.2f} acc: {:0.2f}'.format(val_monitor[0], val_monitor[1])
        print('#################################################')
        print(f'Final da época {i}')
        models[key].evaluate(X_chunk, y_chunk)
        models[key].evaluate(setup[1].toarray(), setup[3])
        print('#################################################')
        del X_chunk
        del y_chunk
```

```
 modelo: cap


Epoch 0 learning rate 0.001
Batch n.: 0 de 49
16/16 [==============================] - 0s 5ms/step - loss: 4.5596 - accu
racy: 0.3809
41/41 [==============================] - 0s 6ms/step - loss: 4.5641 - accu
racy: 0.2626
#################################################
Final da época 0
7/7 [==============================] - 0s 6ms/step - loss: 2.3264 - accura
cy: 0.4144
41/41 [==============================] - 0s 6ms/step - loss: 2.1156 - accu
racy: 0.4908
#################################################
Epoch 1 learning rate 0.0005
```

7 - Analisa a acurária e a perda de cada modelo

In [47]:

```
for key, setup in setups.items():
    print(f'model: {key} - metrics: {models[key].metrics_names}')
    loss, acc = models[key].evaluate(setup[1].toarray(), setup[3])
    loss_teste, acc_teste = models[key].evaluate(setup[0][:1000].toarray(), setup[2][:1000]
    print(f'(acc_teste - acc): {(acc_teste - acc)*10_000:.2f}')
    print(f'(loss_teste - loss): {(loss - loss_teste)*100:.2f}\n')
```

```
model: cap - metrics: ['loss', 'accuracy']
41/41 [==============================] - 0s 6ms/step - loss: 0.2063 - accura
cy: 0.9426
32/32 [==============================] - 0s 6ms/step - loss: 0.0311 - accura
cy: 0.9930
(acc_teste - acc): 504.27
(loss_teste - loss): 17.52

model: pos - metrics: ['loss', 'accuracy']
41/41 [==============================] - 0s 6ms/step - loss: 0.4195 - accura
cy: 0.8905
32/32 [==============================] - 0s 6ms/step - loss: 0.0603 - accura
cy: 0.9840
(acc_teste - acc): 934.95
(loss_teste - loss): 35.92

model: subpos - metrics: ['loss', 'accuracy']
41/41 [==============================] - 0s 7ms/step - loss: 1.2774 - accura
cy: 0.6761
32/32 [==============================] - 0s 6ms/step - loss: 0.3879 - accura
cy: 0.9010
(acc_teste - acc): 2248.90
(loss_teste - loss): 88.95

model: item - metrics: ['loss', 'accuracy']
41/41 [==============================] - 0s 6ms/step - loss: 0.9540 - accura
cy: 0.7657
32/32 [==============================] - 0s 6ms/step - loss: 0.1294 - accura
cy: 0.9550
(acc_teste - acc): 1893.03
(loss_teste - loss): 82.45

model: subitem - metrics: ['loss', 'accuracy']
41/41 [==============================] - 0s 6ms/step - loss: 0.6159 - accura
cy: 0.8423
32/32 [==============================] - 0s 6ms/step - loss: 0.0764 - accura
cy: 0.9820
(acc_teste - acc): 1397.34
(loss_teste - loss): 53.95
```

8 - Cria predições para cada modelo treinado anteriormente e salva numa lista

In [48]:

```
preds_list = []
for key, setup in setups.items():
    teste = df_itens.descricao_limpa.values
    preds_list.append(models[key].predict(vectorizer.transform(teste)))
```

9 - Aplica o modelo na base completa e analisa resultados

9.1 - Com resultado da aplicação do modelo na base de dados completa, cria novas colunas para cada parte da NCM

In [49]:

```
i = 0
for key, model in models.items():
    name = key + '_resul'
    print(name)
    if key == 'cap':
        df_itens[name] = encoders['capitulo'].encoder.inverse_transform(encoders['capitulo'
    elif key == 'pos':
        df_itens[name] = encoders['posicao'].encoder.inverse_transform(encoders['posicao'].
    elif key == 'subpos':
        df_itens[name] = encoders['subposicao'].encoder.inverse_transform(encoders['subposi
    elif key == 'item':
        df_itens[name] = encoders['item'].encoder.inverse_transform(encoders['item'].binari
    elif key == 'subitem':
        df_itens[name] = encoders['subitem'].encoder.inverse_transform(encoders['subitem'].
    i += 1
```

```
cap_resul
pos_resul
subpos_resul
item_resul
subitem_resul
```

In [50]:

```
df_itens.head()
```

Out[50]:

	descricao_limpa	capitulo	posicao	subposicao	item	subitem	cap_resul	pos_resul	subpo
0	masc fac hidrat embal 25ml day dre miracl vali...	33	04	99	1	0	33	04	
1	dioctil ftalat flex bag d 20 tonel metr diocty...	29	17	32	0	0	29	17	
2	sol calc borrach belfast mx	64	06	20	0	0	64	06	
3	sol calc borrach lyon mx	64	06	20	0	0	64	06	
4	sol calc borrach lyon mx	64	06	20	0	0	64	06	

9.2 - Recria os campos de NCM e cria uma NCM_result com as colunas resultado da aplicação do modelo

In [51]:

```
df_itens['ncm'] = df_itens['capitulo'] + df_itens['posicao'] + df_itens['subposicao'] + df_
```

In [52]:

```
df_itens['ncm_resul'] = df_itens['cap_resul'] + df_itens['pos_resul'] + df_itens['subpos_re
```

In [53]:

```
df_itens.head()
```

Out[53]:

	descricao_limpa	capitulo	posicao	subposicao	item	subitem	cap_resul	pos_resul	subpo
0	masc fac hidrat embal 25ml day dre miracl vali...	33	04	99	1	0	33	04	
1	dioctil ftalat flex bag d 20 tonel metr diocty...	29	17	32	0	0	29	17	
2	sol calc borrach belfast mx	64	06	20	0	0	64	06	
3	sol calc borrach lyon mx	64	06	20	0	0	64	06	
4	sol calc borrach lyon mx	64	06	20	0	0	64	06	

9.3 - Cria dataframe erro com os valores de NCM_resultado errados

In [54]:

```
df_erros = df_itens[df_itens['ncm'] != df_itens.ncm_resul]
```

In [55]:

```
print(f'Tamanho do dataset: {len(df_itens)} registros')
print(f'Quantidade de erros: {len(df_erros)}, o que representa {(len(df_erros)/len(df_itens
```

```
Tamanho do dataset: 26104 registros
Quantidade de erros: 6814, o que representa 26.10%
```

In [56]:

```
df_erros.head()
```

Out[56]:

	descricao_limpa	capitulo	posicao	subposicao	item	subitem	cap_resul	pos_resul	subp
9	tambor metal d 25kg past pigment alumini stap ...	32	19	90	3	0	32	19	
11	bat recarrega liti ion 300 0653 14 8w 7 4v 200...	85	07	60	0	0	85	07	
47	cap telefon celul 7 imitaca	39	26	90	9	0	39	26	
48	cap telefon celul 7 plu imitaca	39	26	90	9	0	39	26	
49	cap telefon celul imitaca	39	26	90	9	0	39	26	

9.4 - Analisa os erros em cada parte da NCM

In [57]:

```
capitulos_err = list(df_erros[df_erros['capitulo'] != df_erros.cap_resul]['cap_resul'])

posicoes_err = list(df_erros[(df_erros['capitulo'] == df_erros.cap_resul) &
                             (df_erros['posicao'] != df_erros.pos_resul)]['pos_resul'])

subposicoes_err = list(df_erros[(df_erros['capitulo'] == df_erros.cap_resul) &
                                (df_erros['posicao'] == df_erros.pos_resul) &
                                (df_erros['subposicao'] != df_erros.subpos_resul)]['subpos_

itens_err = list(df_erros[(df_erros['capitulo'] == df_erros.cap_resul) &
                          (df_erros['posicao'] == df_erros.pos_resul) &
                          (df_erros['subposicao'] == df_erros.subpos_resul) &
                          (df_erros['item'] != df_erros.item_resul)]['item_resul'])

subitens_err = list(df_erros[(df_erros['capitulo'] == df_erros.cap_resul) &
                             (df_erros['posicao'] == df_erros.pos_resul) &
                             (df_erros['subposicao'] == df_erros.subpos_resul) &
                             (df_erros['item'] == df_erros.item_resul) &
                             (df_erros['subitem'] != df_erros.subitem_resul)]['subitem_resu
```

In [58]:

```
print(len(capitulos_err), len(posicoes_err), len(subposicoes_err), len(itens_err), len(subi
```

```
298 588 3484 1437 1007
```

9.5 - Erros em capítulo - detalha quantidade de capítulos errados e quantos erros por capítulo

In [59]:

```
total_err = {}
for capitulo in capitulos_err:
    if total_err.get(capitulo):
        total_err[capitulo] += 1
    else:
        total_err[capitulo] = 1

print(f'Total de capítulos errados: {len(total_err.keys())}')
print(f'Total de erros em capítulos: {len(capitulos_err)} erros\n')

for k, v in total_err.items():
    total_value = len(df_itens[df_itens['capitulo'] == str(k).zfill(2)])
    print(f'Capítulo com erro: {k} => {v} erros em {total_value} = {((v/total_value)*100):.
```

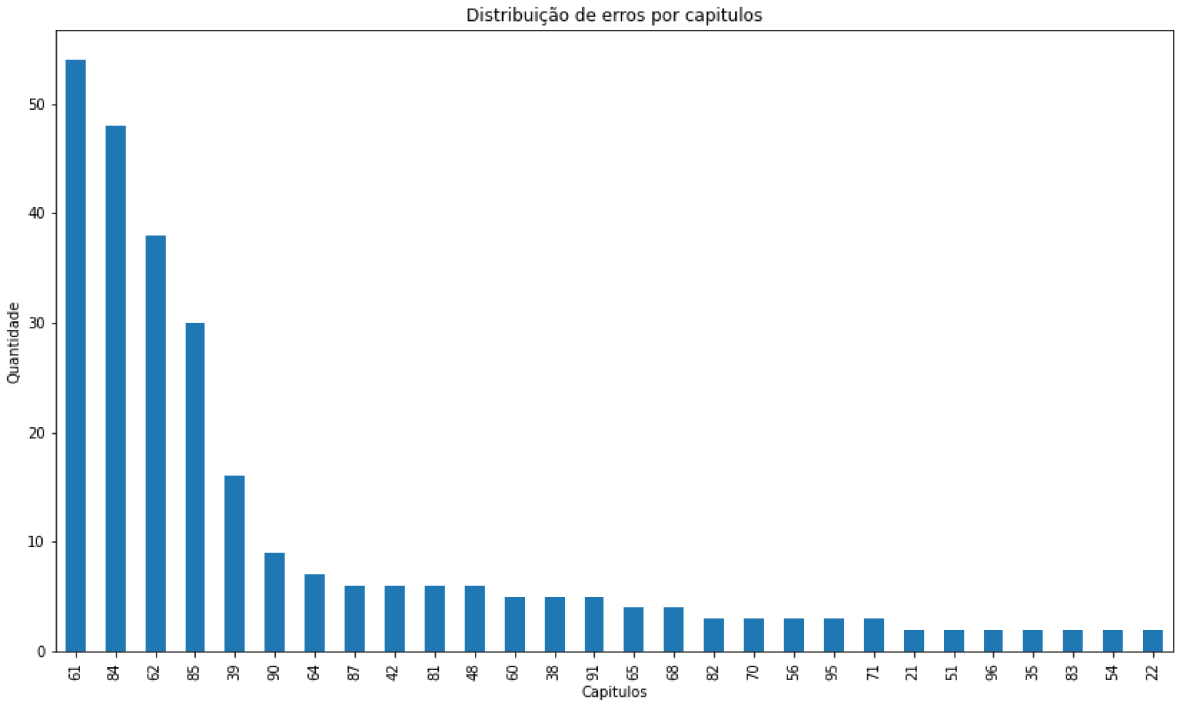
```
Total de capítulos errados: 48
Total de erros em capítulos: 298 erros

Capítulo com erro: 83 => 2 erros em 99 = 2.02%
Capítulo com erro: 91 => 5 erros em 189 = 2.65%
Capítulo com erro: 71 => 3 erros em 135 = 2.22%
Capítulo com erro: 62 => 38 erros em 1133 = 3.35%
Capítulo com erro: 39 => 16 erros em 596 = 2.68%
Capítulo com erro: 84 => 48 erros em 3640 = 1.32%
Capítulo com erro: 85 => 30 erros em 5178 = 0.58%
Capítulo com erro: 19 => 1 erros em 61 = 1.64%
Capítulo com erro: 48 => 6 erros em 290 = 2.07%
Capítulo com erro: 64 => 7 erros em 136 = 5.15%
Capítulo com erro: 61 => 54 erros em 2176 = 2.48%
Capítulo com erro: 94 => 1 erros em 100 = 1.00%
Capítulo com erro: 60 => 5 erros em 84 = 5.95%
Capítulo com erro: 42 => 6 erros em 357 = 1.68%
Capítulo com erro: 08 => 1 erros em 76 = 1.32%
Capítulo com erro: 35 => 2 erros em 54 = 3.70%
Capítulo com erro: 82 => 3 erros em 164 = 1.83%
Capítulo com erro: 90 => 9 erros em 737 = 1.22%
Capítulo com erro: 11 => 1 erros em 30 = 3.33%
Capítulo com erro: 21 => 2 erros em 109 = 1.83%
Capítulo com erro: 68 => 4 erros em 77 = 5.19%
Capítulo com erro: 38 => 5 erros em 310 = 1.61%
Capítulo com erro: 34 => 1 erros em 61 = 1.64%
Capítulo com erro: 59 => 1 erros em 27 = 3.70%
Capítulo com erro: 27 => 1 erros em 72 = 1.39%
Capítulo com erro: 63 => 1 erros em 133 = 0.75%
Capítulo com erro: 05 => 1 erros em 29 = 3.45%
Capítulo com erro: 25 => 1 erros em 87 = 1.15%
Capítulo com erro: 53 => 1 erros em 31 = 3.23%
Capítulo com erro: 51 => 2 erros em 51 = 3.92%
Capítulo com erro: 55 => 1 erros em 148 = 0.68%
Capítulo com erro: 54 => 2 erros em 98 = 2.04%
Capítulo com erro: 56 => 3 erros em 99 = 3.03%
Capítulo com erro: 58 => 1 erros em 44 = 2.27%
Capítulo com erro: 81 => 6 erros em 61 = 9.84%
Capítulo com erro: 78 => 1 erros em 12 = 8.33%
Capítulo com erro: 73 => 1 erros em 225 = 0.44%
Capítulo com erro: 87 => 6 erros em 371 = 1.62%
Capítulo com erro: 49 => 1 erros em 40 = 2.50%
Capítulo com erro: 96 => 2 erros em 162 = 1.23%
Capítulo com erro: 65 => 4 erros em 79 = 5.06%
Capítulo com erro: 70 => 3 erros em 112 = 2.68%
```

```
Capítulo com erro: 22 => 2 erros em 742 = 0.27%
Capítulo com erro: 95 => 3 erros em 1077 = 0.28%
Capítulo com erro: 16 => 1 erros em 78 = 1.28%
Capítulo com erro: 44 => 1 erros em 163 = 0.61%
Capítulo com erro: 33 => 1 erros em 1021 = 0.10%
Capítulo com erro: 88 => 1 erros em 36 = 2.78%
```

In [60]:

```
# Cria gráfico de barras
df_temp = pd.DataFrame()
df_temp = df_erros[df_erros['capitulo'] != df_erros.cap_resul]
values = df_temp['cap_resul'].value_counts()
threshold = 1  # define limite inferior para exibição no gráfico (exibir 10 primeiros )
mask = values > threshold
values = values.loc[mask] # pega os valores que devem ser exibidos

# informações do gráfico
ax = values.plot.bar(figsize=(14,8), title="Distribuição de erros por capitulos")
ax.set_xlabel("Capitulos")
ax.set_ylabel("Quantidade")
print(f"Quantidade de capítulos errados: {len(df_temp['cap_resul'].value_counts())}")
```

```
Quantidade de capítulos errados: 48
```



9.6 - Erros em posição - detalha quantidade de posições erradass e quantos erros por posição

In [61]:

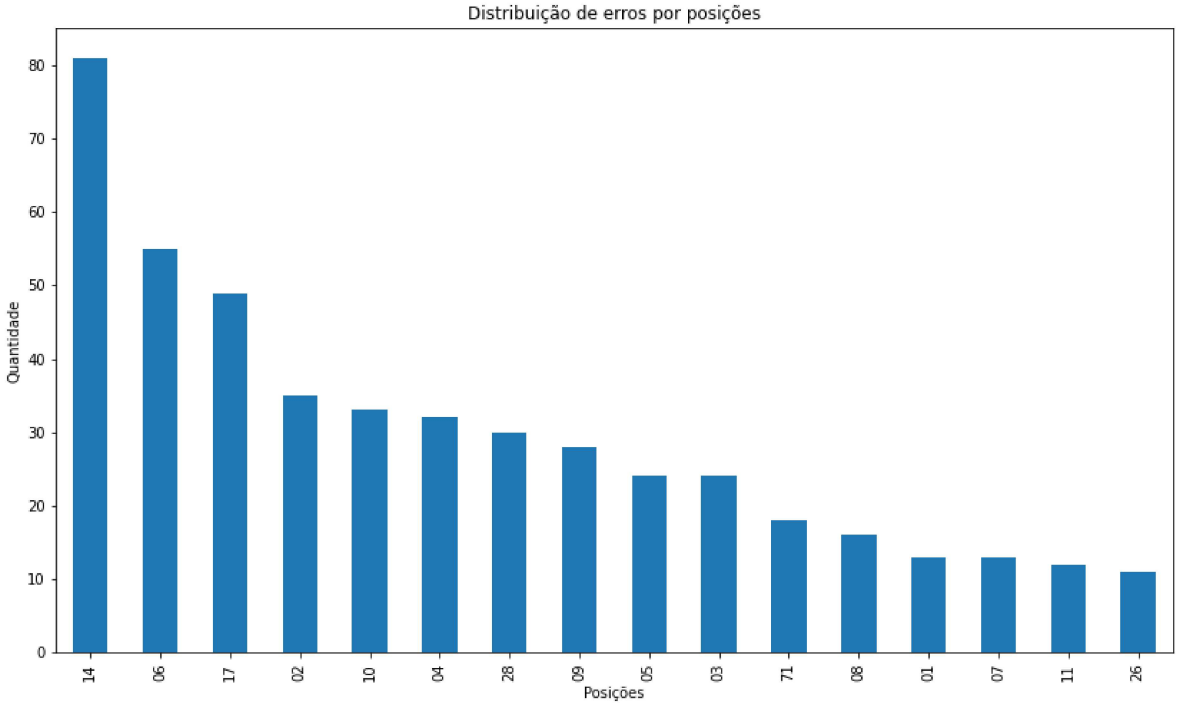
```
total_err = {}
for posicao in posicoes_err:
    if total_err.get(posicao):
        total_err[posicao] += 1
    else:
        total_err[posicao] = 1

print(f'Total de posições erradas: {len(total_err.keys())}')
print(f'Total de erros em posiçõss: {len(posicoes_err)} erros\n')

for k, v in total_err.items():
    total_value = len(df_itens[df_itens['posicao'] == str(k).zfill(2)])
    print(f'Posição com erro: {k} => {v} erro(s) em {total_value} = {((v/total_value)*100):
```

```
Total de posições erradas: 42
Total de erros em posiçõss: 588 erros

Posição com erro: 17 => 49 erro(s) em 2294 = 2.14%
Posição com erro: 26 => 11 erro(s) em 299 = 3.68%
Posição com erro: 23 => 6 erro(s) em 509 = 1.18%
Posição com erro: 36 => 4 erro(s) em 129 = 3.10%
Posição com erro: 21 => 2 erro(s) em 218 = 0.92%
Posição com erro: 03 => 24 erro(s) em 2393 = 1.00%
Posição com erro: 04 => 32 erro(s) em 2631 = 1.22%
Posição com erro: 19 => 7 erro(s) em 227 = 3.08%
Posição com erro: 11 => 12 erro(s) em 412 = 2.91%
Posição com erro: 82 => 7 erro(s) em 1258 = 0.56%
Posição com erro: 16 => 6 erro(s) em 242 = 2.48%
Posição com erro: 42 => 5 erro(s) em 69 = 7.25%
Posição com erro: 07 => 13 erro(s) em 905 = 1.44%
Posição com erro: 08 => 16 erro(s) em 744 = 2.15%
Posição com erro: 01 => 13 erro(s) em 662 = 1.96%
Posição com erro: 06 => 55 erro(s) em 1261 = 4.36%
Posição com erro: 43 => 4 erro(s) em 326 = 1.23%
Posição com erro: 14 => 81 erro(s) em 1513 = 5.35%
Posição com erro: 10 => 33 erro(s) em 584 = 5.65%
Posição com erro: 77 => 4 erro(s) em 20 = 20.00%
Posição com erro: 02 => 35 erro(s) em 1730 = 2.02%
Posição com erro: 18 => 3 erro(s) em 857 = 0.35%
Posição com erro: 05 => 24 erro(s) em 820 = 2.93%
Posição com erro: 09 => 28 erro(s) em 458 = 6.11%
Posição com erro: 15 => 6 erro(s) em 374 = 1.60%
Posição com erro: 31 => 3 erro(s) em 118 = 2.54%
Posição com erro: 13 => 9 erro(s) em 275 = 3.27%
Posição com erro: 12 => 3 erro(s) em 334 = 0.90%
Posição com erro: 25 => 5 erro(s) em 178 = 2.81%
Posição com erro: 28 => 30 erro(s) em 447 = 6.71%
Posição com erro: 39 => 6 erro(s) em 159 = 3.77%
Posição com erro: 65 => 2 erro(s) em 18 = 11.11%
Posição com erro: 51 => 2 erro(s) em 17 = 11.76%
Posição com erro: 61 => 7 erro(s) em 12 = 58.33%
Posição com erro: 50 => 1 erro(s) em 10 = 10.00%
Posição com erro: 40 => 1 erro(s) em 42 = 2.38%
Posição com erro: 79 => 9 erro(s) em 37 = 24.32%
Posição com erro: 44 => 4 erro(s) em 85 = 4.71%
Posição com erro: 73 => 3 erro(s) em 224 = 1.34%
Posição com erro: 71 => 18 erro(s) em 879 = 2.05%
Posição com erro: 29 => 2 erro(s) em 169 = 1.18%
Posição com erro: 24 => 3 erro(s) em 235 = 1.28%
```

In [62]:

```
# Cria gráfico de barras
df_temp = pd.DataFrame()
df_temp = df_erros[(df_erros['capitulo'] == df_erros.cap_resul) &
                   (df_erros['posicao'] != df_erros.pos_resul)]
df_temp.head()
values = df_temp['pos_resul'].value_counts()
threshold = 10  # define limite inferior para exibição no gráfico (exibir 10 primeiros )
mask = values > threshold
values = values.loc[mask] # pega os valores que devem ser exibidos

# informações do gráfico
ax = values.plot.bar(figsize=(14,8), title="Distribuição de erros por posições")
ax.set_xlabel("Posições")
ax.set_ylabel("Quantidade")
print(f"Quantidade de posições erradas: {len(df_temp['cap_resul'].value_counts())}")
```

```
Quantidade de posições erradas: 58
```



9.7 - Erros em subposição, item e subitem - calcula o erro por categoria e a quantidade de erros em cada categoria

In [63]:

```
total_err = {}
for subposicao in subposicoes_err:
    if total_err.get(subposicao):
        total_err[subposicao] += 1
    else:
        total_err[subposicao] = 1

print(f'Total de subposições erradas: {len(total_err.keys())}')
print(f'Total de erros em subposiçõss: {len(subposicoes_err)} erros\n')

for k, v in total_err.items():
    total_value = len(df_itens[df_itens['subposicao'] == str(k).zfill(2)])
    print(f'Subposição com erro: {k} => {v} erro(s) em {total_value} = {((v/total_value)*10
```

```
Total de subposições erradas: 49
Total de erros em subposiçõss: 3484 erros

Subposição com erro: 80 => 73 erro(s) em 357 = 20.45%
Subposição com erro: 12 => 150 erro(s) em 1280 = 11.72%
Subposição com erro: 90 => 244 erro(s) em 2599 = 9.39%
Subposição com erro: 51 => 31 erro(s) em 437 = 7.09%
Subposição com erro: 11 => 182 erro(s) em 507 = 35.90%
Subposição com erro: 40 => 112 erro(s) em 824 = 13.59%
Subposição com erro: 10 => 115 erro(s) em 2516 = 4.57%
Subposição com erro: 50 => 192 erro(s) em 1001 = 19.18%
Subposição com erro: 39 => 105 erro(s) em 393 = 26.72%
Subposição com erro: 21 => 141 erro(s) em 974 = 14.48%
Subposição com erro: 63 => 45 erro(s) em 209 = 21.53%
Subposição com erro: 32 => 75 erro(s) em 257 = 29.18%
Subposição com erro: 23 => 8 erro(s) em 106 = 7.55%
Subposição com erro: 53 => 46 erro(s) em 51 = 90.20%
Subposição com erro: 99 => 173 erro(s) em 899 = 19.24%
Subposição com erro: 14 => 43 erro(s) em 64 = 67.19%
Subposição com erro: 29 => 209 erro(s) em 628 = 33.28%
Subposição com erro: 43 => 49 erro(s) em 167 = 29.34%
Subposição com erro: 20 => 243 erro(s) em 1580 = 15.38%
Subposição com erro: 30 => 203 erro(s) em 2627 = 7.73%
Subposição com erro: 19 => 176 erro(s) em 794 = 22.17%
Subposição com erro: 00 => 26 erro(s) em 2007 = 1.30%
Subposição com erro: 13 => 12 erro(s) em 97 = 12.37%
Subposição com erro: 60 => 4 erro(s) em 246 = 1.63%
Subposição com erro: 70 => 18 erro(s) em 622 = 2.89%
Subposição com erro: 91 => 93 erro(s) em 312 = 29.81%
Subposição com erro: 22 => 60 erro(s) em 191 = 31.41%
Subposição com erro: 31 => 113 erro(s) em 244 = 46.31%
Subposição com erro: 41 => 34 erro(s) em 737 = 4.61%
Subposição com erro: 44 => 17 erro(s) em 63 = 26.98%
Subposição com erro: 89 => 83 erro(s) em 134 = 61.94%
Subposição com erro: 33 => 53 erro(s) em 141 = 37.59%
Subposição com erro: 49 => 95 erro(s) em 224 = 42.41%
Subposição com erro: 42 => 43 erro(s) em 156 = 27.56%
Subposição com erro: 69 => 20 erro(s) em 113 = 17.70%
Subposição com erro: 72 => 7 erro(s) em 19 = 36.84%
Subposição com erro: 93 => 48 erro(s) em 86 = 55.81%
Subposição com erro: 83 => 1 erro(s) em 13 = 7.69%
Subposição com erro: 59 => 32 erro(s) em 139 = 23.02%
Subposição com erro: 92 => 27 erro(s) em 276 = 9.78%
Subposição com erro: 52 => 6 erro(s) em 50 = 12.00%
Subposição com erro: 81 => 9 erro(s) em 44 = 20.45%
```

```
Subposição com erro: 71 => 16 erro(s) em 391 = 4.09%
Subposição com erro: 62 => 26 erro(s) em 900 = 2.89%
Subposição com erro: 61 => 6 erro(s) em 77 = 7.79%
Subposição com erro: 79 => 17 erro(s) em 50 = 34.00%
Subposição com erro: 77 => 1 erro(s) em 16 = 6.25%
Subposição com erro: 95 => 1 erro(s) em 24 = 4.17%
Subposição com erro: 94 => 1 erro(s) em 51 = 1.96%
```

In [64]:

```
total_err = {}
for item in itens_err:
    if total_err.get(item):
        total_err[item] += 1
    else:
        total_err[item] = 1

print(f'Total de itens erradas: {len(total_err.keys())}')
print(f'Total de erros em itens: {len(itens_err)} erros\n')

for k, v in total_err.items():
    total_value = len(df_itens[df_itens['item'] == str(k)])
    print(f'Itens com erro: {k} => {v} erro(s) em {total_value} = {((v/total_value)*100):.2
```

```
Total de itens erradas: 10
Total de erros em itens: 1437 erros

Itens com erro: 9 => 374 erro(s) em 5782 = 6.47%
Itens com erro: 0 => 111 erro(s) em 10731 = 1.03%
Itens com erro: 3 => 76 erro(s) em 1603 = 4.74%
Itens com erro: 1 => 454 erro(s) em 4628 = 9.81%
Itens com erro: 2 => 317 erro(s) em 1646 = 19.26%
Itens com erro: 4 => 57 erro(s) em 842 = 6.77%
Itens com erro: 7 => 22 erro(s) em 291 = 7.56%
Itens com erro: 6 => 4 erro(s) em 110 = 3.64%
Itens com erro: 5 => 21 erro(s) em 336 = 6.25%
Itens com erro: 8 => 1 erro(s) em 135 = 0.74%
```

In [65]:

```
total_err = {}
for subitem in subitens_err:
    if total_err.get(subitem):
        total_err[subitem] += 1
    else:
        total_err[subitem] = 1

print(f'Total de subitens erradas: {len(total_err.keys())}')
print(f'Total de erros em subitens: {len(subitens_err)} erros\n')

for k, v in total_err.items():
    total_value = len(df_itens[df_itens['subitem'] == str(k)])
    print(f'Subitens com erro: {k} => {v} erro(s) em {total_value} = {((v/total_value)*100)
```

```
Total de subitens erradas: 9
Total de erros em subitens: 1007 erros

Subitens com erro: 7 => 53 erro(s) em 258 = 20.54%
Subitens com erro: 0 => 201 erro(s) em 19432 = 1.03%
Subitens com erro: 1 => 267 erro(s) em 2109 = 12.66%
Subitens com erro: 9 => 188 erro(s) em 2595 = 7.24%
Subitens com erro: 4 => 102 erro(s) em 191 = 53.40%
Subitens com erro: 2 => 113 erro(s) em 910 = 12.42%
Subitens com erro: 3 => 49 erro(s) em 343 = 14.29%
Subitens com erro: 6 => 7 erro(s) em 76 = 9.21%
Subitens com erro: 5 => 27 erro(s) em 140 = 19.29%
```

In [66]:

```
for i, row in enumerate(df_erros.iloc[:,0]):
    if df_erros.iloc[i,1] != df_erros.iloc[i,6]:
        print(f'errou capítulo {df_erros.iloc[i,1]} - ncm: {df_erros.iloc[i, 11]} - ncm_res
    elif df_erros.iloc[i,2] != df_erros.iloc[i,7]:
        print(f'errou posicao {df_erros.iloc[i,2]} - ncm: {df_erros.iloc[i, 11]} - ncm_resu
    elif df_erros.iloc[i,3] != df_erros.iloc[i,8]:
        print(f'errou subposicao {df_erros.iloc[i,3]} - ncm: {df_erros.iloc[i, 11]} - ncm_r
    elif df_erros.iloc[i,4] != df_erros.iloc[i,9]:
        print(f'errou item {df_erros.iloc[i,4]} - ncm: {df_erros.iloc[i, 11]} - ncm_resul=
    elif df_erros.iloc[i,5] != df_erros.iloc[i,10]:
        print(f'errou subitem {df_erros.iloc[i,5]} - ncm: {df_erros.iloc[i, 11]} - ncm_resu
```

```
errou item 3 - ncm: 32199030 - ncm_resul= 32199090
errou subposicao 60 - ncm: 85076000 - ncm_resul= 85078000
errou subposicao 90 - ncm: 39269090 - ncm_resul= 39261290
errou subposicao 90 - ncm: 39269090 - ncm_resul= 39261231
errou subposicao 90 - ncm: 39269090 - ncm_resul= 39261290
errou subposicao 90 - ncm: 39269090 - ncm_resul= 39261290
errou subposicao 90 - ncm: 39269090 - ncm_resul= 39261290
errou posicao 04 - ncm: 85044010 - ncm_resul= 85174010
errou posicao 04 - ncm: 85044010 - ncm_resul= 85174010
errou subposicao 70 - ncm: 85177099 - ncm_resul= 85179099
errou item 0 - ncm: 90031100 - ncm_resul= 90031190
errou item 0 - ncm: 90031100 - ncm_resul= 90031190
errou item 0 - ncm: 90031100 - ncm_resul= 90031190
errou item 0 - ncm: 90031100 - ncm_resul= 90031190
errou item 0 - ncm: 90031100 - ncm_resul= 90031190
errou item 0 - ncm: 90031100 - ncm_resul= 90031190
errou capítulo 95 - ncm: 95030031 - ncm_resul= 83012030
errou posicao 04 - ncm: 85044010 - ncm_resul= 85264010
errou subposicao 10 - ncm: 39231090 - ncm_resul= 39235110
```

In [ ]: