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

Eduardo Henrique Vicensotti Berdugo

ANÁLISE DE DADOS TEXTUAIS DE ORDENS DE SERVIÇO DE UM SISTEMA DE GERENCIAMENTO E CONTROLE DA MANUTENÇÃO (CMMS)

Belo Horizonte
2020

Eduardo Henrique Vicensotti Berdugo

ANÁLISE DE DADOS TEXTUAIS DE ORDENS DE SERVIÇO DE UM SISTEMA DE GERENCIAMENTO E CONTROLE DA MANUTENÇÃO (CMMS)

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

Belo Horizonte
2020

SUMÁRIO

1. Introdução

As novas tecnologias têm possibilitado o armazenamento de uma quantidade cada vez maior de textos em forma digital, provenientes de interações humanas, suas comunicações e sua cultura. A informação codificada em forma de texto, é um rico complemento a tipos mais estruturados de dados, e nos anos recentes observa-se uma explosão de pesquisas impíricas usando textos como dados. (GENTZKOW, KELLY, e TADD, 2019)

Com a massificação do uso de computadores, diversas são as fontes disponíveis de dados em forma de texto, não somente a mais conhecida delas, a *internet*, como também fontes dentro das próprias empresas e instituições. Kremer, et al. (2013), cita por exemplo, textos de relatórios de analistas como fonte de dados textuais não estruturados em um banco, e Raja, et al. (2008), cita registros eletrônicos clínicos médicos em um hospital.

No entanto, apesar do avanço da tecnologia, atualmente ainda não estamos em um ponto onde a linguagem humana em todas as suas formas não processadas, podem ser entendidas por computadores. (RESHAMWALA, MISHRA, e PAWAR, 2013)

O processamento de linguagem natural (NLP, da sigla em inglês *Natural Language Processing*) é uma área da ciência da computação, inteligência artificial e línguistica que estuda as interações entre computadores e linguagem natural humana. Trata-se de uma coleção de técnicas utilizadas para extrair estrutura gramatical e significado de textos, por exemplo, com o objetivo de desempenhar tarefas úteis como resultado. (RESHAMWALA, MISHRA, e PAWAR, 2013)

Este trabalho visa utilizar algumas destas técnicas de processamento de linguagem natural para análise de textos armazenados em um sistema de controle e gerenciamento da área de manutenção em uma indústria.

1.1. Contextualização

Observa-se nas últimas décadas, a área de manutenção na indústria transformando-se de um simples setor responsável por atividades de reparo dos ativos, para uma área estratégica inserida no processo produtivo e essencial para o

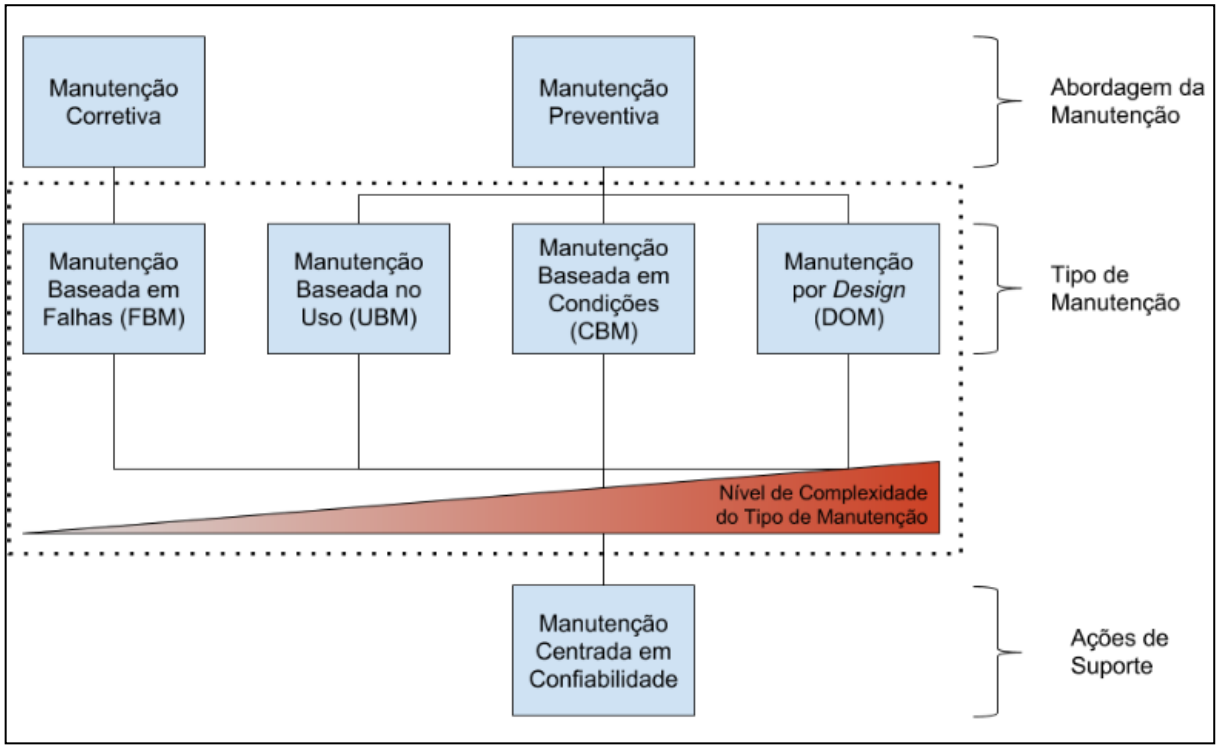
alcance dos objetivos e metas da organização. Com a evolução tecnológica, equipamentos industriais cada vez mais complexos em seus sistemas mecânicos, eletroeletrônicos e computacionais, foi assim necessário o desenvolvimento de técnicas e ferramentas mais avançadas e inovadoras de gestão estratégica da área de manutenção e suas atividades. (XAVIER, 2004, *apud* PERES, 2011).

> "A manutenção, para ser estratégica, precisa estar voltada para os resultados empresariais da organização. E, sobretudo, deixar de ser apenas eficiente para se tornar eficaz; em outras palavras, não basta apenas reparar o equipamento ou instalação tão rápido quanto possível, mas é necessário, principalmente, manter a função do equipamento disponível para a operação reduzindo a probabilidade de uma parada de produção ou o não fornecimento de um serviço." (PINTO, 2003, *apud* PERES, 2011).

Existem diversas classificações para as técnicas de gestão da manutenção, dentre elas, destacam-se três abordagens principais: manutenção corretiva, realizada após a falha; manutenção preventiva, realizada em intervalos pré-definidos; e manutenção preditiva, realizada através do monitoramento de condições ou da análise da confiabilidade dos ativos. (PRAJAPATI, BECHTEL, e GANESAN, 2012, *apud* NETO, 2019)

Segundo WAEYENBERGH e PINTELON (2004, *apud* NETO, 2019), existem ainda outras abordagens mais específicas, como por exemplo, manutenção baseada em condições (em inglês *CBM, Condition-Based Maintenance*); manutenção baseada em falhas (em inglês FBM, *Failure Based Maintenance*); manutenção por *design* (em inglês DOM, *Design-Out Maintenance*); manutenção baseada em detecção (em inglês DBM, *Detection-Based Maintenance*) e manutenção baseada no uso (em inglês UBM, *Use Based Maintenance*), demostradas na Figura 1.

Figura 1: Abordagens e Tipos de Manutenção



Fonte: NETO (2019, p. 30).

Em uma gestão de manutenção eficiente, todos os tipos de manutenção e as atividades podem ser rastreadas, de forma a permitir conhecer o histórico de manutenções em um determinado ativo. Para tal são utilizados sistemas computadorizados de gerenciamento da manutenção (em inglês, CMMS, *Computerized maintenance management system),* que devem permitir requisição e registro eletrônico de cada serviço realizado em um equipamento. (COHEN, 2014). Ordem de serviço (sigla OS) é a denominação normalmente utilizada para este registro eletrônico gerado a partir do sistema CMMS (NETO, 2011, *apud* VIANA, 2002)

Independente do tipo de sistema CMMS, ele deve documentar o histórico do serviço de forma compreensível, contendo minimamente, o ativo que sofreu a manutenção, tipo de manutenção, peças utilizadas, mão de obra executante e duração do serviço. (COHEN, 2014) A Tabela 1 mostra os campos que são tipicamente utilizados como registro em ordens de serviço de um sistema CMMS.

Tabela 1: Campos comuns em Ordens de Serviço

Nome do campo	**Descrição do campo**
ID do requisitante e informações de contato	Múltiplos campos indicando quem é o requisitante e suas informações de contato
Data e hora da requisição	Hora local e data da requisição de serviço
ID do Equipamento e/ou Descrição	Preferivelmente um identificador único do equipamento. Adicionalmente, uma descrição detalhada do equipamento.
Descrição do problema	Descrição do problema pelo requisitante.
Código de falha	Causa raíz do problema reportado.
Peças utilizadas	Descrição das peças utilizadas na atividade.
Mão de obra utilizada (Horas)	Quantidade de horas-homem utilizadas para realizar a atividade.
Custo da mão de obra	Custo de total das horas-homem utilizadas.

Fonte: Adaptado de COHEN (2014).

A integridade dos dados em um sistema CMMS começa pela responsabilidade da própria equipe de trabalho. Apesar da maioria dos *softwares* de CMMS do mercado possuir ferramentas básicas para minimizar a ocorrência de erros – por exemplo, a obrigatoriedade de escolher um ID de um equipamento de uma árvore de equipamentos previamente cadastrados –, a acuracidade das informações preenchidas depende da entrada do usuário e da pré-configuração realizada no sistema, por exemplo, utilizando padronização de nomenclatura no cadastro dos ativos. (COHEN, 2014)

1.2. O problema proposto

Dados históricos normalizados e íntegros em um sistema CMMS são a chave para geração de gráficos e relatórios precisos, com o objetivo de entender onde a organização deve focar seus esforços para tornar a manutenção mais eficaz; não apenas reparando o equipamento rapidamente em uma manutenção corretiva, mas principalmente agindo de forma preventiva, reduzindo a probabilidade de uma parada de produção.

Em uma primeira análise amostral do histórico de ordens de serviço em um sistema CMMS de uma indústria, a utilização de recursos para normalização e integridade dos dados é comum, por exemplo, através da obrigatoriedade de preenchimento ou seleção de dados mínimos normalizados em uma ordem de serviço, e desta forma, proporcionam um histórico eficiente para consulta, provendo as informações necessárias para estudos da melhoria da gestão da manutenção.

Em segunda análise, constata-se que outros dados não normalizados disponíveis no sistema CMMS também podem ser utilizados em estudos para fins de aumento de eficácia da área de manutenção, como por exemplo, os campos abertos para digitação do usuário. Informações como “título da ordem de serviço” e “descrição detalhada do serviço”, são campos textuais abertos presentes no CMMS em questão especialmente importantes para ordens de serviço do tipo manutenção corretiva.

Em sua maioria, campos abertos descrevem de forma detalhada o defeito, sua provável causa raíz ou ausência aparente da mesma, e a ação tomada pelo manutentor no momento do problema até sua resolução. No entanto, a análise individual destes campos em todas as ordens de serviço do sistema CMMS não é normalmente realizada, exceto em casos pontuais, por ser custosa devido ao grande volume de dados.

O trabalho em questão visa realizar esta análise em uma base histórica de forma automatizada através da utilização de algoritmos, bibliotecas e técnicas de processamento de linguagem natural implementadas em forma de scripts *python*.

Objetiva-se detectar quais ações são mais freqüentemente citadas pelos manutentores de forma mais granular possível, em ordens de serviço do tipo corretiva. Por exemplo, qual dispositivo está sofrendo mais intervenção dentro de uma máquina e qual ação está sendo realizada, permitindo entender padrões de atividades realizadas de forma mais específica.

Com este resultado, objetiva-se:

1. Entender os principais “modos de falha” da fábrica, de forma a potencialmente reconhecer padrões de atividades e re-ocorrência de

problemas e soluções, baseando-se na freqüência de aparecimento de termos ou conjunto de termos.

2. Entender quais tópicos são mais importantes e mais freqüentes, para intensificar atividades de melhoria. Por exemplo, objetiva-se entender quais atividades de fato estão sendo realizadas pela equipe de manutenção – atuação em quebras ou ajustes. Para estes dois exemplos, ajustes poderiam ser realizados pela equipe de operação, através de treinamentos, e quebras poderiam ser evitadas através de manutenções preventivas ou preditivas mais eficazes. Não serão sugeridas ações, como as citadas acima, porém serão indicados tópicos de relevância para futura atuação.

2. Coleta de Dados

Os dados adquiridos são provenientes da base de dados do sistema CMMS de uma indústria, exportados para formato *Microsoft Excel*®. Os dados representam o histórico de 2 anos de manutenções realizadas em ativos de uma fábrica pela equipe de elétrica e cada linha da tabela representa uma ordem de serviço realizada.

Tabela 2: *descriptions.xlsx*

Nome da coluna / campo	**Descrição**	**Tipo**
Work Order Number (index)	Número da ordem de serviço	Numérico(7), sequencial.
Description	Título da ordem de serviço	Texto.
WO LONG DESCRIPTION	Descrição da ordem de serviço.	Texto.
Asset Number	ID do ativo	Alfanumérico, normalizado.
Work Type	Tipo do serviço	Texto, normalizado. Exemplos: PM – manutenção preventiva, CM – manutenção corretiva, EM – manutenção corretiva emergencial, PRJ – projeto, OTHER – outro tipo.
Location	ID do local onde o ativo está instalado	Alfanumérico, normalizado.
Status Date	Data que a ordem de serviço foi fechada.	Data e hora.
Report Date	Data que a ordem de serviço foi reportada pelo	Data e hora.

Nome da coluna / campo	Descrição	Tipo
	manutentor.	
Act Lab Hrs	Duração da em horas da ordem de serviço	Numérico.

Fonte: Autor.

Na figura 2, é possível ver um *snapshot* de uma consulta a esta base, com alguns exemplos de dados.

Figura 2: *Snapshot* da tabela *descriptions*.

Index	Description	WO LONG DESCRIPTION	Asset Number	Work Type	Location	Status Date	Report Date	Act Lab Hrs
3316111	Instalar sensores de contra check no rejeito	nan	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-10 07:46:03.999…	10
3321059	Troca dos sensores para calibração	Troca dos sensores para calibração.Foram trocados os sensores e resistência, e acompan…	SAC-0002	CM	SAL-128	2017-10-04 17:02:44	2017-01-16 06:33:28	4.33333
3324231	Filtros do painel da Enchedora IMA F673 - L-…	efetuada a troca dos filtros	BJ1019	CM	SAL-098	2017-04-24 07:24:49.999…	2017-01-17 11:05:14.999…	1
3333521	Medição da rotação do motor do Tanque Auxilia…	Realizadas todas as medições de rotação conforme protoco…	TNQ-0032	CM	SAL-106	2017-04-24 07:24:49.999…	2017-01-23 08:48:58	11
3338319	Não mantém temperatura de 80°C durante SAQ	Não mantém temperatura de 80°C durante SAQ. desconhecida, possível baixa vazão na entrada…	SAP-0007	CM	PAP-002	2017-04-24 07:24:49.999…	2017-01-27 14:43:59	3.66667
3338320	Vazamento de água dentro da célula em cima do A1	Vazamento de água dentro da célula em cima do A1 vazamento já havia sido seco pelo pessoal …	nan	CM	SAL-102	2017-04-24 07:24:49.999…	2017-01-27 14:47:02	0.666667
3338321	Máquina não liga	Máquina não liga.Bug PLC.Apenas foi colocado em modo prog e retornado para run, ok	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-27 14:51:06.999…	0.166667
3338322	Resetar captador de pó no piso técnico	Resetar captador de pó no piso técnico.Possível limpeza.Foi resetado o captador, ok	CAP-0015	CM	SAL-256	2017-04-24 07:24:49.999…	2017-01-27 14:54:37	0.583333
3338324	Não imprime na B9	Não imprime na B9 desconhecida, foi retirada e usada na B7 e depois devolveram Verificado na…	INK-0037	OTHER	SAL-033	2017-10-04 17:02:44	2017-01-27 15:11:25	0.916667
3338329	Falha no sinal do pedal	Falha no sinal do pedal.cabo rompido.Feito reparo e acompanhamento, ok.	TNQ-0014	CM	SAL-103	2017-04-24 07:24:49.999…	2017-01-27 15:24:06	1.16667
3338330	Perdendo aquecimento	Perdendo aquecimento.cabo resistência rompido, feito isolação no cabo e religado, ok.	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-27 15:30:42.999…	0.416667
3338332	Alarme OV561	Alarme OV561.sensor com mau contato.Feito reaperto no cabo do sensor, ok.	TNQ-0014	CM	SAL-103	2017-04-24 07:24:49.999…	2017-01-27 15:34:08.999…	0.5
3338337	Montagens de equipamentos e treiname…	Montagens de equipamentos e treinamentos	nan	CM	JCP-001	2017-04-24 07:24:49.999…	2017-01-27 15:57:49.999…	24
3338361	Mordente dianteiro com cabo da resistência rom…	Mordente dianteiro com cabo da resistência rompido.Foi passado para realizar a troca da …	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-27 21:09:01.999…	3
3339432	Sensor queimado	Sensor queimado tempo de usoFoi trocado o sensor, ok.	VAL-0203	OTHER	CIP-002	2017-10-04 17:02:44	2017-01-30 18:18:55	0.5
3339673	Falha no sensor da válvula OV-561	Falha no sensor da válvula OV-561.Durante o setup foram trocados os conectores dos cabos …	VAL-0002	OTHER	SAL-103	2017-10-04 17:02:44	2017-01-31 01:25:36	1.33333
3339692	Treinamento geral	nan	nan	OTHER	JCP-001	2017-10-04 17:02:44	2017-01-31 02:44:38	16
3339693	Impressora Wolke não imprime	Impressora Wolke não imprime.Cabos das cabeças de impressão estavam invertidos ocorrido após…	INK-0037	OTHER	SAL-033	2017-10-04 17:02:44	2017-01-31 02:46:46	2.25
3339703	Treinamentos e organização para inicio…	Foi feito varios treinamentos e organização da planta para volta das operações.	nan	OTHER	JCP-001	2017-10-04 17:02:44	2017-01-31 03:46:12	10
3339705	Organização da planta e ligando as máquinas par…	Foi terminada a organização da planta e ligada as máquinas para voltar as atividades …	nan	CM	JCP-001	2017-04-24 07:24:49.999…	2017-01-31 03:50:49.999…	16
3339731	Ajustes e acompanhamento da Inkjet do RPA.	Foi feito ajustes e acompanhamento do funcionamento da Inkjet do RPA onde a mesma e…	INK-0005	CM	SAL-167	2017-04-24 07:24:49.999…	2017-01-31 07:02:09.999…	2
	Problemas com o EBOD	Foi feita uma verificação no painel eletrico				2017-10-04	2017-01-31	

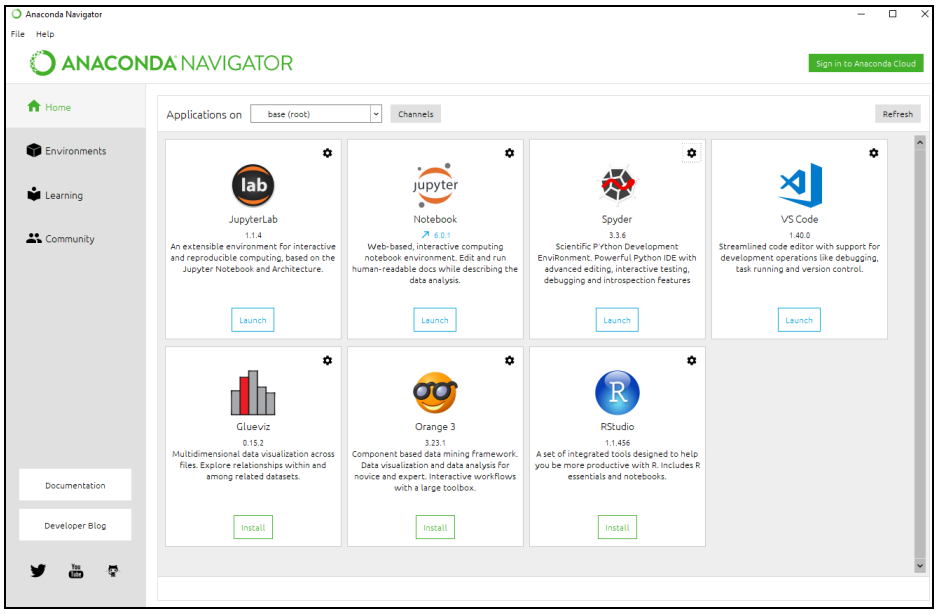
Fonte: Autor

3. Processamento/Tratamento de Dados

3.1 Ferramentas utilizadas

Como ferramenta para desenvolvimento dos *scripts* em *python*, foi escolhida a distribuição *Anaconda*, versão 2019.10 (figura 3), disponível em https://www.anaconda.com/distribution/.

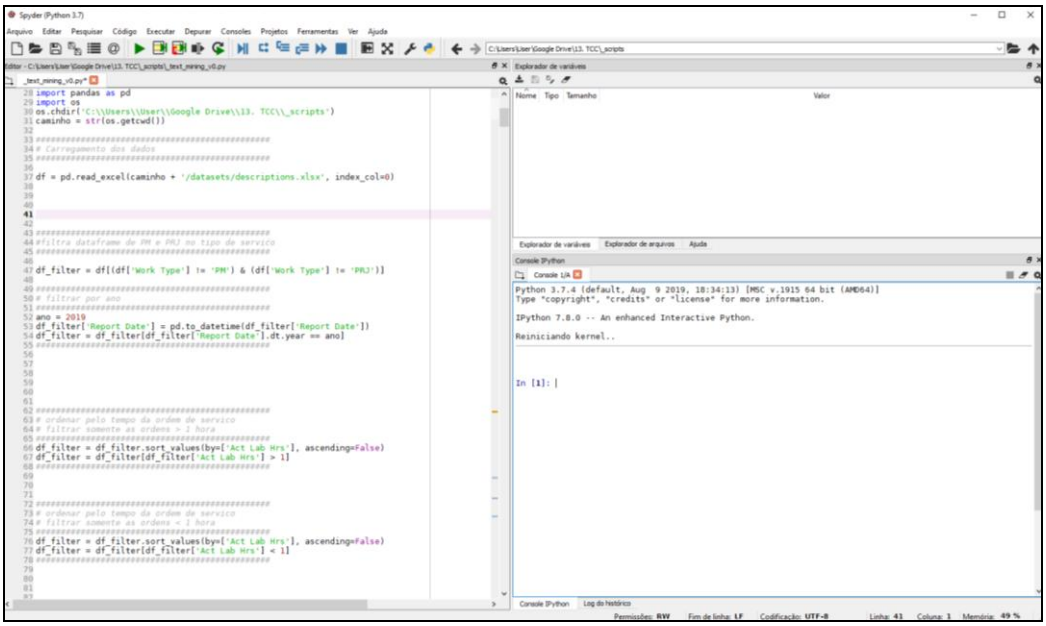
Figura 3: *Screenshot* do *Anaconda Navigator*, da distribuição *Anaconda*.



Fonte: Autor.

A distribuição *Anaconda* foi escolhida por possuir as principais ferramentas e bibliotecas para criação de scripts de análise de dados e aprendizado de máquinas em seu pacote, evitando assim a necessidade de instalação manual e configuração de cada biblioteca que seria necessária, optimizando o desenvolvimento dos *scripts*. O editor *Spyder*, incluso no pacote, foi utilizado como ferramenta de edição de código e depuração – figura 4.

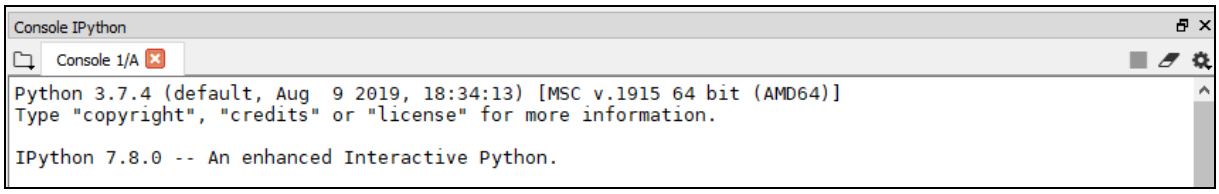
Figura 4: *Screenshot* do editor *Spyder*.



Fonte: Autor.

O editor *Spyder* versão 3.3.6 possui o ambiente configurado para trabalhar sob o *Python 3.7.4*, incluso no pacote da distribuição *Anaconda 2019.10*, assim como o *Ipython 7.8.0,* conforme figura 5.

Figura 5: *Screenshot* do *console do Spyder*.



Fonte: Autor.

3.2 Feature Selection

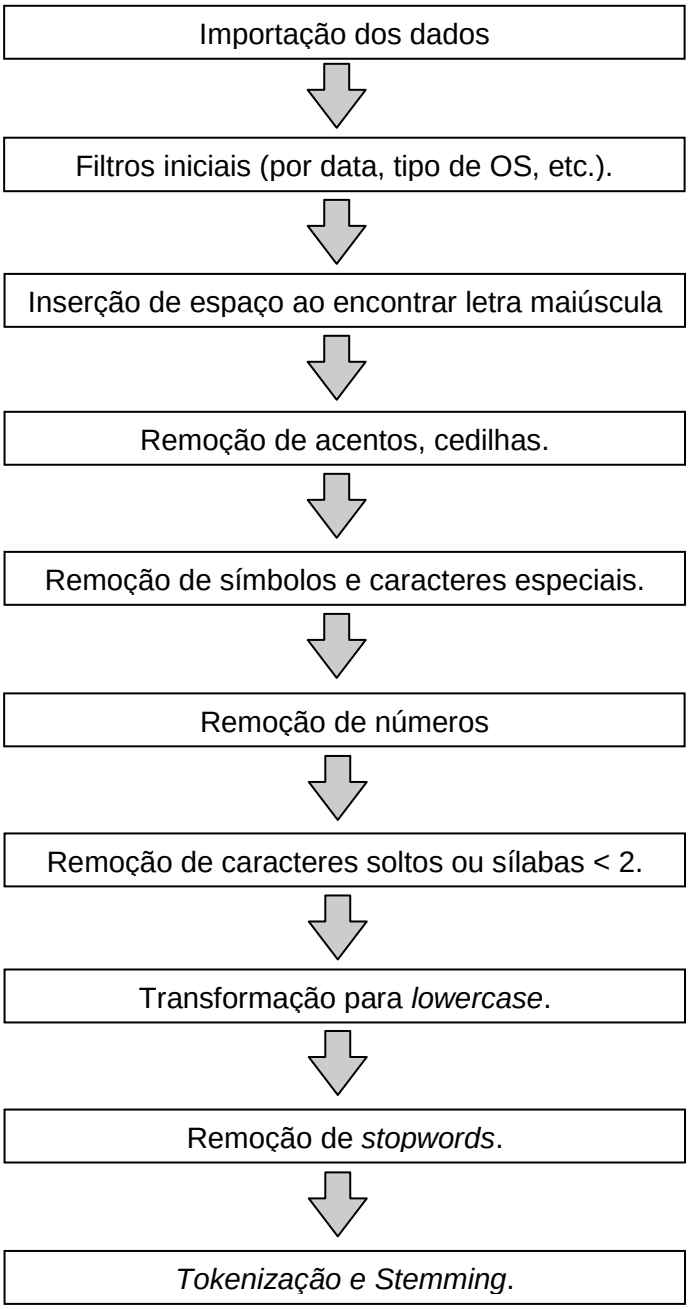
Com o objetivo de reduzir o número de elementos em um texto para uma representação mais simples e apropriada para uma análise computacional posterior, o primeiro passo comum é o de eliminar os elementos do texto que não sejam palavras. Isto inclui pontuações, números, nomes próprios, entre outros. Também é comum remover um conjunto de palavras muito comuns, ou muito raras, que são importantes para estrutura gramatical de sentenças porém transmitem pouco ou nenhum significado quando sozinhas. Estes termos são conhecidos como *stopwords*, e exemplos seriam artigos (“a”, “o”, “um”, “uma”), conjunções (“e”, “ou”), formas do verbo ser (“estar”, “estou”), entre outros. Uma prática comum é a de utilizar uma lista de termos *stopwords* previamente definidos. (GENTZKOW, KELLY, e TADD, 2019)

Um passo final que é normalmente usado é denominado *stemming*. O *stemmming* é o processo de remover sufixos de palavras, como por exemplo, “economia” e “economicamente” para “econom”. (GENTZKOW, KELLY, e TADD, 2019)

Estes passos de limpeza dos dados reduzem o número de elementos únicos de linguagem, e além de oferecer um ganho computacional no momento do processamento, são a chave para interpretação de alguns modelos de aprendizado de máquinas. (GENTZKOW, KELLY, e TADD, 2019)

A figura 6 apresenta a seqüência na qual estes processamentos foram realizados no algoritmo desenvolvido para tratamento dos dados, e os seus respectivos *scripts* serão apresentados ao decorrer do trabalho:

Figura 6: Seqüência de tratamento dos dados.



Fonte: Autor.

Importante ressaltar que alguns destes processos foram necessários para eliminar inclusive erros de digitação encontrados no texto. Por exemplo, sentenças sem espaço ou sem pontuação final (“a máquina parouFoi necessário...”) e erros de acentuação (“maquina”, “agregacao”, entre outros). Para o caso de acentuação, por

exemplo, foi decidido remover qualquer tipo com intuito de simplificar o processamento posterior do texto. Por exemplo, “é”, “É”, “ê”, foram transformados em “e” para todas as palavras, e o mesmo com demais vogais.

Além disso, muitos símbolos, códigos e numerações presentes nos textos referentes a códigos de peças, valores e parâmetros de máquinas, entre outros, foram removidos, pois eram pontuais, raros, e não enriqueciam a análise com informações relevantes, conforme direcionado na bibliografia consultada.

3.3 Bibliotecas utilizadas no processamento

Em linhas gerais, foram pesquisadas bibliotecas e ferramentas que poderiam ser utilizadas para executar os seguintes tratamentos, já citados anteriormente de forma mais detalhada:

1. Exploração e manipulação de dados: capacidade de importar arquivos de base de dados (exemplo: planilhas), e permitir sua manipulação e consulta.

2. Manipulação de textos: capacidade de buscar padrões em texto para substituição, remoção de caracteres, transformação de caracteres (exemplo, *lowercase*).

3. *Tokenização*, *stemming* em português brasileiro: capacidade de identificar palavras da lingua portuguesa em um texto, separá-las (*tokenização*), e posteriormente realizar o *stemming* (remoção de sufixos*)*.

4. *Stopwords*: capacidade de remover *stopwords* de um texto, baseando-se em uma listagem pré-definida de palavras em português brasileiro e permitir inclusão de novos termos.

Após longa pesquisa e testes, chegou-se a um conjunto de bibliotecas que seriam necessárias para o desenvolvimento do código, apresentadas na tabela 3. Muitas destas bibliotecas já se encontram presentes na distribuição *Anaconda*, e como já mencionado, este foi um dos motivos de sua escolha.

Tabela 3: *Bibliotecas utilizadas.*

biblioteca	**Descrição**	**Comando(s) utilizados**
Pandas	Pacote de ferramentas para análise de dados e manipulação, construída sobre a base da linguagem de programação *python*. Informações: https://pandas.pydata.org/	import pandas as pd
Regex	Linguagem de programação embutida no p*ython* especializada na criação de regras para busca de padrões em *strings* (variáveis do tipo texto). Informações: https://docs.python.org/3/howto/regex.html	import re
Spacy	Pacote de ferramentas para processamento de linguagem natural. Instalada à parte (não inclusa do pacote *Anaconda*). Necessário pacote de linguagem portuguesa (*download* e instalação manual): arquivo *pt_core_news_sm-2.2.5.tar.gz*. Disponível em: https://github.com/explosion/spacy-models/releases, acesso em 01 mar. 2020. Informações: https://spacy.io/	Pré-requisito: prompt: *pip install spacy* prompt: *pip install pt_core_news_sm-2.2.5.tar.gz* import spacy
NLTK	Pacote de ferramentas para processamento de linguagem natural. Necessário pacote *RSLPStemmer* (instalação manual). Informações: https://www.nltk.org/	Pré-requisito: import nltk nltk.download('rslp') from nltk.stem import RSLPStemmer

Fonte: Autor.

3.4 Importação dos dados e filtros iniciais

Os dados foram importados do formato planilha *.xlsx* de um diretório previamente definido, para um *dataframe pandas*, permitindo posterior manipulação – figura 7:

Figura 7: Importação dos dados.

```
28 import pandas as pd
29 import os
30 os.chdir('C:\\Users\\User\\Google Drive\\13. TCC\\_scripts')
31 caminho = str(os.getcwd())
32
33 ##############################################
34 # Carregamento dos dados
35 ##############################################
36
37 df = pd.read_excel(caminho + '/datasets/descriptions.xlsx', index_col=0)
38
```

Fonte: Autor.

O *dataframe "df"* foi utilizado como *raw data*, ou seja, dado original sem alterações, de forma que um novo *dataframe* filtrado "*df_filter*" foi criado para trabalho no código, eliminando ordens de serviço do tipo "PM" e "PRJ", manutenção preventiva e projetos respectivamente, conforme *script* apresentado na figura 8:

Figura 8: Criação do *dataframe* de trabalho filtrado.

```
43 ##############################################
44 #filtra dataframe de PM e PRJ no tipo de servico
45 ##############################################
46
47 df_filter = df[(df['Work Type'] != 'PM') & (df['Work Type'] != 'PRJ')]
48
```

Fonte: Autor.

O *dataframe* filtrado possui agora apenas registros com o tipo de ordem de serviço diferentes de "PM" e "PRJ" (figura 9), totalizando 8553 registros, contra os 11267 registros do *dataframe* original (figura 10):

Figura 9: *Snapshot* do dataframe filtrado. Coluna "*Work Type*" já filtrada.

Index	Description	LONG DESCRIPTI	Asset Number	Work Type	Location	Status Date	Report Date	Act Lab Hrs
3316111	Instalar sensores de …	nan	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-10 07:46:03.999…	10
3321059	Troca dos sensores par…	Troca dos sensores par…	SAC-0002	CM	SAL-128	2017-10-04 17:02:44	2017-01-16 06:33:28	4.33333
3324231	Filtros do painel da En…	efetuada a troca dos fi…	BJ1019	CM	SAL-098	2017-04-24 07:24:49.999…	2017-01-17 11:05:14.999…	1
3333521	Medição da rotação do m…	Realizadas todas as med…	TNQ-0032	CM	SAL-106	2017-04-24 07:24:49.999…	2017-01-23 08:48:58	11
3338319	Não mantém temperatura …	Não mantém temperatura …	SAP-0007	CM	PAP-002	2017-04-24 07:24:49.999…	2017-01-27 14:43:59	3.66667
3338320	Vazamento de água dentro …	Vazamento de água dentro …	nan	CM	SAL-102	2017-04-24 07:24:49.999…	2017-01-27 14:47:02	0.666667
3338321	Máquina não liga	Máquina não liga.Bug PLC…	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-27 14:51:06.999…	0.166667
3338322	Resetar captador de …	Resetar captador de …	CAP-0015	CM	SAL-256	2017-04-24 07:24:49.999…	2017-01-27 14:54:37	0.583333
3338324	Não imprime na B9	Não imprime na B9 desco…	INK-0037	OTHER	SAL-033	2017-10-04 17:02:44	2017-01-27 15:11:25	0.916667
3338329	Falha no sinal do ped…	Falha no sinal do ped…	TNQ-0014	CM	SAL-103	2017-04-24 07:24:49.999…	2017-01-27 15:24:06	1.16667
3338330	Perdendo aquecimento	Perdendo aquecimento.…	SAC-0002	CM	SAL-128	2017-04-24 07:24:49.999…	2017-01-27 15:30:42.999…	0.416667
3338332	Alarme OV561	Alarme OV561.sensor…	TNQ-0014	CM	SAL-103	2017-04-24 07:24:49.999…	2017-01-27 15:34:08.999…	0.5
3338337	Montagens de equipamentos…	Montagens de equipamentos…	nan	CM	JCP-001	2017-04-24 07:24:49.999…	2017-01-27 15:57:49.999…	24
3338361	Mordente	Mordente	SAC-0002	CM	SAL-128	2017-04-24	2017-01-27	3

Fonte: Autor.

Figura 10: Comparativo dos *dataframes* carregados.

df	DataFrame	(11267, 8)	Column names: Description, WO LONG DESCRIPTION, Asset Number, Work Typ ...
df_filter	DataFrame	(8553, 8)	Column names: Description, WO LONG DESCRIPTION, Asset Number, Work Typ ...

Fonte: Autor.

Durante o processamento e etapas de análise dos dados, foram feitos exercícios com os 2 campos textuais presentes na base de dados, são eles: "*Description*" e "*WO LONG DESCRIPTION*". O intuíto era o de entender qual dos campos trariam melhores informações para análise em questão. A principal diferença entre ambos diz respeito à quantidade e o detalhamento das informações no texto, visto que ambos são campos digitados pelo usuário, porém o campo "*Description*" é mais sucinto, tendo sido utilizado como "título" do registro, enquanto que o campo "*WO LONG DESCRIPTION*" é um texto detalhado, onde o manutentor coloca todas as informações entendidas por ele sobre o serviço realizado. Para tal, foram criadas duas linhas de código para escolha de qual dos campos seriam utilizados, bastando comentar (#) a linha em questão para desabilitar a utilização do campo, conforme figura 11:

Figura 11: Escolha dos campos a serem utilizados.

```
84 #define tamanho do processamento
85 l = len(df_filter)
86
87 nome_coluna = 'WO LONG DESCRIPTION'
88 #nome_coluna = 'Description'
89
```

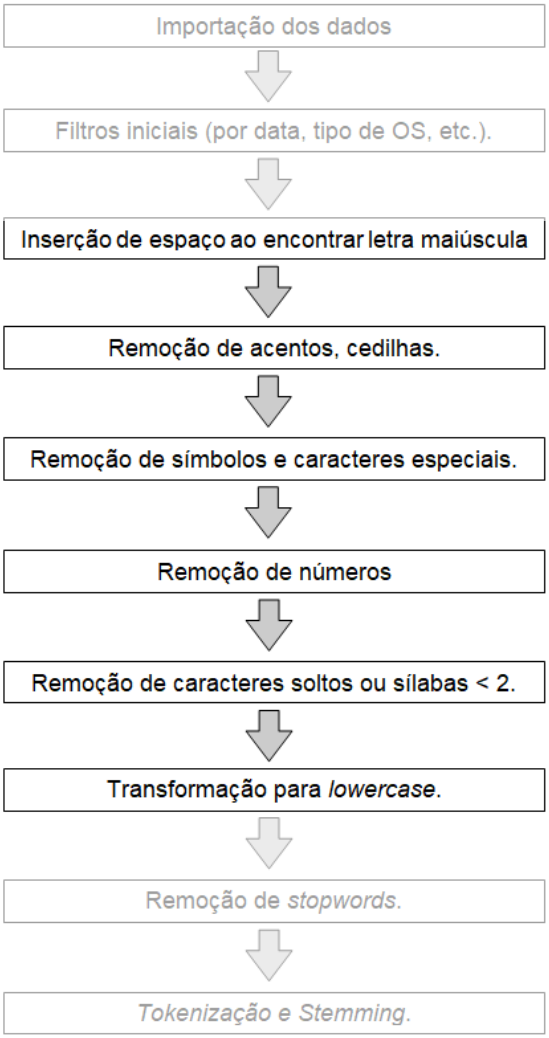
Fonte: Autor.

Os dados estão carregados para o início do processamento.

3.5 Substituição, remoção e inserção de caracteres

Os próximos passos no processamento seguem a seqüência apresentada anteriormente na figura 6, destacados desta vez na figura 12, seguindo orientações da bibliografia pesquisada. Para os passos em destaque, foram utilizadas funções da biblioteca *regex* entre outras nativas do *python*.

Figura 12: Passos que utilizaram funções *Regex* e funções nativas do *python*.



Fonte: Autor.

O primeiro passo foi o de importar a biblioteca *regex*, representada por ”*re”* no *python*, conforme figura 13.

Figura 13: Passos que utilizaram funções *Regex*.

```
95 import re
96
```

Fonte: Autor.

O processamento seguinte, apresentado na figura 14, foi o de utilizar a biblioteca *regex* para encontrar caracteres em letra maíuscula e inserir um espaço antes do mesmo. O objetivo desta etapa era o de corrigir erros de digitação encontrados ao longo do texto, após um análise preliminar amostral.

Figura 14: Rotina para encontrar caracteres maísculos.

```
99  ################################################################
100 # Encontra padrão letra maíscula
101 # se encontrar coloca espaco antes
102 ################################################################
103
104 coluna_tmp = [0] * len(df_filter)
105
106 for i in range(l):
107
108     coluna_tmp[i] = df_filter.iloc[i][nome_coluna]
109
110     coluna_tmp[i] = re.sub('([A-Z])', r' \1', str(coluna_tmp[i]))
111
112     printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)
113
114 # insere nova coluna filtrada do dataframe
115 nome_coluna = "New"
116 insertColumn(nome_coluna, coluna_tmp)
117 print("Remove palavras juntas completo.")
118
119 ################################################################
120
```

Fonte: Autor.

É possível notar o uso de duas funções customizadas no código, sendo elas: *printProgressBar*() e *insertColumn*(). A primeira foi criada para apresentar visualmente no console do *ipython* o *status* atual da execução, apenas a título orientativo, conforme apresentado na figura 15.

Figura 15: Apresentação da saída da função *printProgressBar()* .

```
Progress: |████████████████████████████████----------| 70.5% Complete
```

Fonte: Autor.

Esta função é chamada ao longo de todo código, dentro de cada *loop* para facilitar a visualização da execução, e seu código é apresentado na figura 16.

Figura 16: Código de fonte da função *printProgressBar()* .

```
# Print iterations progress
def printProgressBar (iteration, total, prefix = '', suffix = '', decimals = 1, length = 100, fill = '', printEnd = "\r"):
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent complete (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
        printEnd    - Optional  : end character (e.g. "\r", "\r\n") (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print('\r%s |%s| %s%% %s' % (prefix, bar, percent, suffix), end = printEnd)
    # Print New Line on Complete
    if iteration == total:
        print()
```

Fonte: Autor. Código disponível em:

<https://stackoverflow.com/questions/3173320/text-progress-bar-in-the-console >

A função *insertColumn*() foi criada para facilitar a inserção de dados no *dataframe*, pois este processo seria executado inúmeras vezes a cada camada de processamento, e se repete ao longo de todas as etapas. A função espera como parâmetro o nome da coluna que será adicionada/substituída e uma lista contendo os dados que devem ser inseridos na coluna. Seu código é apresentado na figura 17.

Figura 17: Código de fonte da função *insertColumn()* .

```
15 #verifica se coluna existe, deleta antes de inserí-la no dataframe
16 def insertColumn(nome, dados):
17
18     if nome in df_filter.columns:
19         df_filter.drop([nome], axis=1, inplace=True)
20
21     df_filter.insert(len(df_filter.columns),nome, dados, True)
22
23     return
24
```

Fonte: Autor.

No processamento seguinte, apresentado na figura 18, o objetivo foi o de normalizar o texto digitado através da substituição de todos os caracteres com acento e cedilha. Esta etapa mostrou-se importante para eliminar a possibilidade de um erro de digitação interferir na análise posterior das palavras do texto. De outra maneira, corrigir gramaticalmente todas as palavras, traria um esfoço computacional demasiado sem agregar conteúdo ao resultado da análise. Por exemplo, a palavra “maquina” ou “máquina” faria o mesmo sentido para interpretação posterior, ou “substituição” e “substituicao”, entre outros exemplos encontrados.

Figura 18: Rotina para substituir acentos e cedilhas.

```
123 ###############################################################
124 # Encontrando padrão á, é, ó, cedilha e variações
125 # e substitui
126 ###############################################################
127
128 coluna_tmp = [0] * len(df_filter)
129
130 for i in range(l):
131
132     coluna_tmp[i] = df_filter.iloc[i][nome_coluna]
133
134     novo_texto = "a"
135     coluna_tmp[i] = re.sub(r'([áãâ])', novo_texto, str(coluna_tmp[i]))
136
137     novo_texto = "A"
138     coluna_tmp[i] = re.sub(r'([ÁÃÂ])', novo_texto, str(coluna_tmp[i]))
139
140     novo_texto = "e"
141     coluna_tmp[i] = re.sub(r'([éê])', novo_texto, str(coluna_tmp[i]))
142
143     novo_texto = "E"
144     coluna_tmp[i] = re.sub(r'([ÉÊ])', novo_texto, str(coluna_tmp[i]))
145
146     novo_texto = "o"
147     coluna_tmp[i] = re.sub(r'([óõô])', novo_texto, str(coluna_tmp[i]))
148
149     novo_texto = "O"
150     coluna_tmp[i] = re.sub(r'([ÓÕÔ])', novo_texto, str(coluna_tmp[i]))
151
152     novo_texto = "c"
153     coluna_tmp[i] = re.sub(r'([ç])', novo_texto, str(coluna_tmp[i]))
154
155     novo_texto = "i"
156     coluna_tmp[i] = re.sub(r'([í])', novo_texto, str(coluna_tmp[i]))
157
158     novo_texto = "I"
159     coluna_tmp[i] = re.sub(r'([Í])', novo_texto, str(coluna_tmp[i]))
160
161
162     printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)
163
164 del novo_texto
165
166 # insere nova coluna filtrada do dataframe
167 nome_coluna = "New"
168 insertColumn(nome_coluna, coluna_tmp)
169 print("Substitui letras completo.")
170
171 ###############################################################
```

Fonte: Autor.

Após normalizado o texto em relação à diferenças na digitação entre um usuário e outro, foi necessário remover todos os símbolos e caracteres especiais de seu conteúdo. Diversos símbolos ao longo dos textos indicavam códigos de peças, temperatura, valores de parâmetros de máquina, graus, entre outros, e não traziam conhecimento significativo para o resultado da análise. Para tal, novamente contou-se com funções da biblioteca *regex*, e seu código é apresentado na figura 19.

Figura 19: Rotina para remover símbolos.

```
############################################################
# Encontrando padrão com símbolos
# se encontrar, coloca espaco
############################################################

coluna_tmp = [0] * len(df_filter)

for i in range(l):

    coluna_tmp[i] = df_filter.iloc[i][nome_coluna]

    novo_texto = " "
    coluna_tmp[i] = re.sub(r'[-!$%^&*()_+|#@~=`{}\[\]:ª°";<>?,.\/]', novo_texto, str(coluna_tmp[i]))

    novo_texto = " "
    coluna_tmp[i] = re.sub("(\d|\W)+", novo_texto, str(coluna_tmp[i]))

    printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)

del novo_texto

# insere nova coluna filtrada do dataframe
nome_coluna = "New"
insertColumn(nome_coluna, coluna_tmp)
print("Remove pontuação completo.")

############################################################
```

Fonte: Autor.

A próxima camada de processamento visava remover todos os dígitos numéricos ao longo dos textos. Seu código é apresentado na figura 20.

Figura 20: Rotina para remover números.

```
############################################################
# Encontrando padrão com 0-9
# se encontrar, coloca nada
############################################################

coluna_tmp = [0] * len(df_filter)

for i in range(l):

    coluna_tmp[i] = df_filter.iloc[i][nome_coluna]

    novo_texto = " "
    coluna_tmp[i] = re.sub(r'([0-9]){1}', novo_texto, str(coluna_tmp[i]))

    printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)

del novo_texto

# insere nova coluna filtrada do dataframe
nome_coluna = "New"
insertColumn(nome_coluna, coluna_tmp)
print("Remove números completo.")

############################################################
```

Fonte: Autor.

Por fim, e uma das etapas mais importantes, todos os textos deveriam ser convertidos para *lowercase,* e palavras contendo 2 caracteres ou menos poderiam ser removidas do conteúdo. A remoção de caracteres soltos ao longo do texto ou palavras muito pequenas, já faz parte de um pré-processamento para remoção de

stopwords, conforme definição apresentada anteriormente, e a configuração de todo o texto em *lowercase*, irá otimizar o algoritmo de agrupamento de palavras posteriormente – por exemplo, “Maquina” e “maquina” serão considerados o mesmo termo, após esta etapa. Seu código é apresentado na figura 21.

Figura 21: Rotina para conversão em *lowercase* e filtro por tamanho.

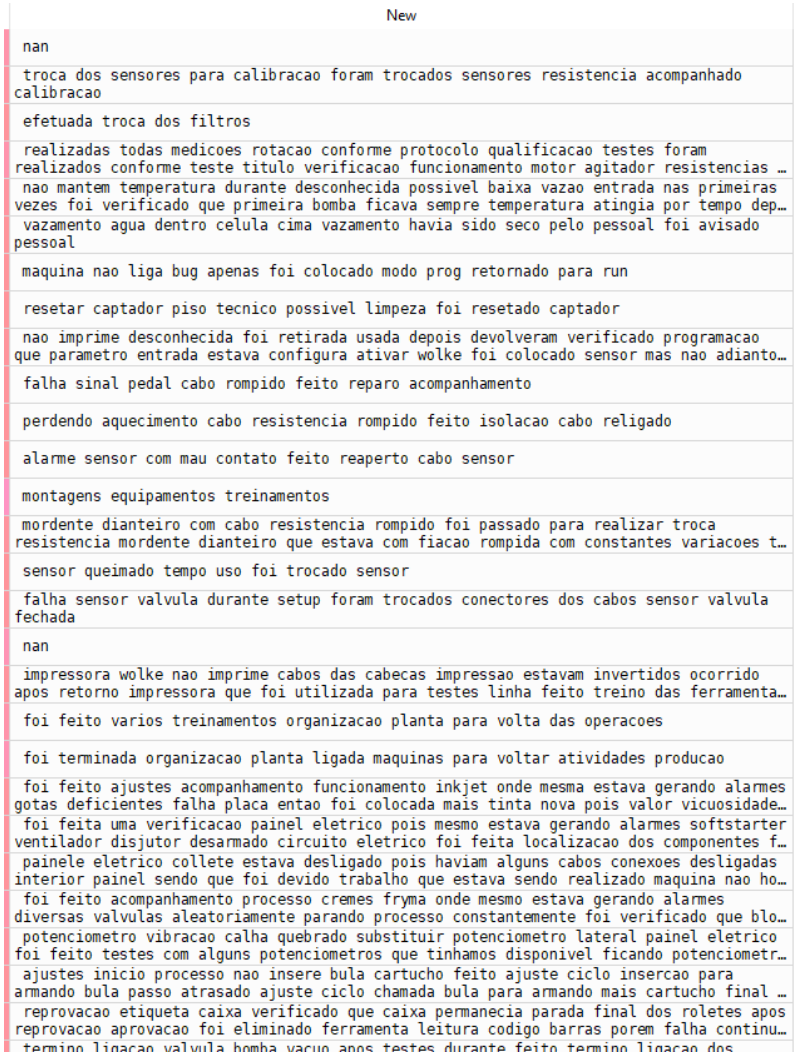
```
256 ################################################################
257 # coloca em lowercase,
258 # remove caracteres soltos.
259 ################################################################
260
261 coluna_tmp = [0] * len(df_filter)
262
263 for i in range(l):
264
265     tokens = str(df_filter.iloc[i][nome_coluna]).lower().split()
266
267     doc_temp = ""
268
269     for word in tokens:
270
271         if len(word)>2:
272             doc_temp = doc_temp + " " + word
273
274     coluna_tmp[i] = doc_temp
275     printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)
276
277 del doc_temp, word, tokens
278
279 # insere nova coluna filtrada do dataframe
280 nome_coluna = "New"
281 insertColumn(nome_coluna, coluna_tmp)
282 print("Remove caracteres soltos completo.")
283
284 ################################################################
285
```

Fonte: Autor.

É possível notar que neste momento utiliza-se a função *.lower*() e *.split*() para respectivamente transformar o texto da linha atual em *lowercase*, e logo separar cada palavra da setença em questão e popular em um vetor de palavras, representado pela variável *tokens*. Esta variável por sua vez é percorrida em todos os seus termos através do ponteiro *word* na linha “*for word in tokens:”* e o código subsequente faz a análise do tamanho de palavra por palavra desta lista de *tokens*.

Após execução deste código, a coluna “*New*” do *dataframe* “*df_filter*” apresenta-se razoavelmente normalizada para continuar com o processamento. Notar pela figura 22 que o texto pré-processado está conforme rotinas executadas até então.

Figura 22: *Snapshot* parcial da nova coluna “*New*” de texto pré-processado.



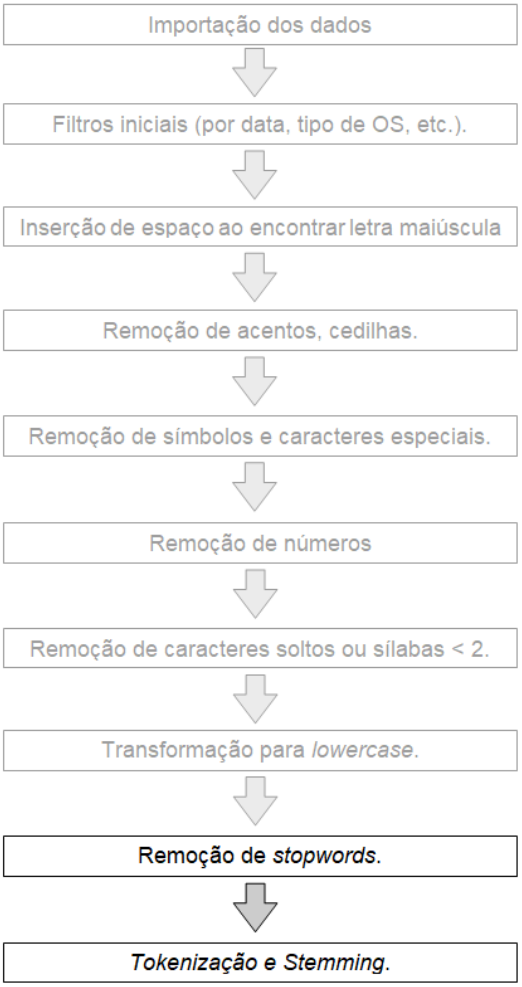
New
nan
troca dos sensores para calibracao foram trocados sensores resistencia acompanhado calibracao
efetuada troca dos filtros
realizadas todas medicoes rotacao conforme protocolo qualificacao testes foram realizados conforme teste titulo verificacao funcionamento motor agitador resistencias …
nao mantem temperatura durante desconhecida possivel baixa vazao entrada nas primeiras vezes foi verificado que primeira bomba ficava sempre temperatura atingia por tempo dep…
vazamento agua dentro celula cima vazamento havia sido seco pelo pessoal foi avisado pessoal
maquina nao liga bug apenas foi colocado modo prog retornado para run
resetar captador piso tecnico possivel limpeza foi resetado captador
nao imprime desconhecida foi retirada usada depois devolveram verificado programacao que parametro entrada estava configura ativar wolke foi colocado sensor mas nao adianto…
falha sinal pedal cabo rompido feito reparo acompanhamento
perdendo aquecimento cabo resistencia rompido feito isolacao cabo religado
alarme sensor com mau contato feito reaperto cabo sensor
montagens equipamentos treinamentos
mordente dianteiro com cabo resistencia rompido foi passado para realizar troca resistencia mordente dianteiro que estava com fiacao rompida com constantes variacoes t…
sensor queimado tempo uso foi trocado sensor
falha sensor valvula durante setup foram trocados conectores dos cabos sensor valvula fechada
nan
impressora wolke nao imprime cabos das cabecas impressao estavam invertidos ocorrido apos retorno impressora que foi utilizada para testes linha feito treino das ferramenta…
foi feito varios treinamentos organizacao planta para volta das operacoes
foi terminada organizacao planta ligada maquinas para voltar atividades producao
foi feito ajustes acompanhamento funcionamento inkjet onde mesma estava gerando alarmes gotas deficientes falha placa entao foi colocada mais tinta nova pois valor vicuosidade…
foi feita uma verificacao painel eletrico pois mesmo estava gerando alarmes softstarter ventilador disjutor desarmado circuito eletrico foi feita localizacao dos componentes f…
painele eletrico collete estava desligado pois haviam alguns cabos conexoes desligadas interior painel sendo que foi devido trabalho que estava sendo realizado maquina nao ho…
foi feito acompanhamento processo cremes fryma onde mesmo estava gerando alarmes diversas valvulas aleatoriamente parando processo constantemente foi verificado que blo…
potenciometro vibracao calha quebrado substituir potenciometro lateral painel eletrico foi feito testes com alguns potenciometros que tinhamos disponivel ficando potenciometr…
ajustes inicio processo nao insere bula cartucho feito ajuste ciclo insercao para armando bula passo atrasado ajuste ciclo chamada bula para armando mais cartucho final …
reprovacao etiqueta caixa verificado que caixa permanecia parada final dos roletes apos reprovacao aprovacao foi eliminado ferramenta leitura codigo barras porem falha continu…
termino ligacao valvula bomba vacuo apos testes durante feito termino ligacao dos

Fonte: Autor.

3.6 Stopwords, tokenização e stemming

Os passos seguintes, após etapas de limpeza do texto, envolvem um processamento mais aprofundado sobre o conteúdo do texto, e consequentemente, maior processamento e ferramentas mais avançadas de NLP. Estes passos estão destacados na figura 23, já apresentada anteriormente.

Figura 23: Passos que utilizaram bibliotecas de NLP.



Fonte: Autor.

A realização da *tokenização*, não é exatamente o processo de maior complexidade a ser realizado, visto que é possível separar as palavras em *tokens* levando em consideração simplesmente o espaço entre elas através de funções nativas do *python,* conforme apresentado anteriormente. Por exemplo, a frase “A máquina está parada”, após limpeza e *tokenização*, forma um vetor/lista de conteúdo [‘maquina’,’esta’,’parada’]. Após remoção de *stopwords,* este texto pode ser ainda mais simplificado para [‘maquina’,’parada’]. Após implementação de *stemming*, este texto pode ser apresentado como [‘maquin’,’par’].

Considerações sobre os algoritmos utilizados:

Para remoção de *stopwords* e *stemming*, o algoritmo deve possuir um *modelo* para sua execução, visto que estes processos demandam padrões pré-estabelecidos relacionados ao idioma que será processado. Em pesquisas

realizadas, o idioma inglês é facilmente encontrado, assim como seus modelos para *stemming, stopwords*, e *lemmatization* (não implementado neste trabalho), o que não necessariamente acontece para o idioma português brasileiro.

Foram realizados testes com algumas bibliotecas que declaravam possuir idioma português para tais funções, e ao final, foram escolhidas as bibliotecas NLTK e *Spacy*, e algumas funções específicas de cada uma.

Especialmente ressaltando que a biblioteca *Spacy* possui um algoritmo bem abrangente de remoção de *stopwords,* em comparação por exemplo, com o próprio algoritmo da biblioteca NLTK, e a função de *stemming* denominada *RSLPStemmer* da biblioteca NLTK, apresentou melhor resultado em comparação com outras funções de outras bibliotecas, como por exemplo, em comparação com a função *stemming* da biblioteca *Spacy*, ou a função *Snowball Stemmer* da biblioteca NLTK.

O objetivo deste trabalho não é o de comparar as diversas funções das bibliotecas disponíveis no mercado, porém vale ressaltar que algumas delas foram escolhidas após alguns testes utilizando alguns requisitos:

1. Menor quantidade de palavras geradas após *stemming*. Ou seja, algoritmo conseguiu remover sufixos do maior número de palavras, para o idioma português.

2. Maior remoção de *stopwords* em português.

3. Maior banco de *stopwords* em português.

Sobre a processo de *lemmatization*, notou-se que o algoritmo alterava em excesso o conteúdo do texto, fazendo o mesmo por muitas vezes perder seu significado. De forma prática sem aprofundar-se neste tipo de processamento, palavras como “bula” (bula de medicamento), presentes no texto, eram convertidas para “bulir” (verbo), entre outros exemplos. Optou-se por utilizar apenas o *stemming*, que manteve o resultado final mais significativo.

Seguindo o processo, a figura 24 apresenta então o *script* utilizado para remoção de *stopwords*, através da biblioteca *Spacy*. Notar que foi utilizado um arquivo adicional para inclusão de *stopwords* customizadas, denominado

"*stopwords.txt*", onde foi possível adicionar um vocabulário de termos que deveriam ser removidos, além dos presentes na biblioteca. Estes termos foram escolhidos após repetidos processamentos e análise dos resultados.

Figura 24: Rotina de remoção de *stopwords*.

```
############################################################
# remove STOPWORDS,
# biblioteca SPACY
# vide arquivo stopwords.txt
############################################################

#pip install spacy
#pip install pt_core_news_sm-2.2.5.tar.gz
import spacy

nlp = spacy.load("pt_core_news_sm")

from spacy.lang.pt.stop_words import STOP_WORDS

stopwords = [k for k in STOP_WORDS]

#configura stopwords:
with open(caminho + "/stopwords.txt", "r") as f:
    new_stopwords = []
    new_stopwords = f.read().splitlines()

#adiciona lista de stopwords
new_stopwords_list = stopwords.extend(new_stopwords)

coluna_tmp = [0] * len(df_filter)

for i in range(l):

    doc = nlp(str(df_filter.iloc[i][nome_coluna]))
    tokens = doc.text.split()

    doc_temp = ""

    for word in tokens:

        if (word in stopwords) == False:
            doc_temp = doc_temp + " " + word

    coluna_tmp[i] = doc_temp
    printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)

del doc, doc_temp, word, tokens, new_stopwords_list, new_stopwords

# insere nova coluna filtrada do dataframe
nome_coluna = "New"
insertColumn(nome_coluna, coluna_tmp)
print("STOPWORDS com SPACY completo.")

############################################################
```

Fonte: Autor.

Por fim, foi realizada rotina de *stemming*, conforme já mencionado, utilizando função *RSLPStemmer* da biblioteca NLTK. Cada *token* (palavra) do texto foi reduzida pelo algoritmo e uma nova coluna denominada "*Stemmer*", foi adicionada no *dataframe* original. A figura 25 apresenta o código executado:

Figura 25: Rotina de *stemming*, com algoritmo *RSLPStemmer*.

```
375 ################################################################
376 # RSLPStemmer stemming (resultado em nova coluna)
377 ################################################################
378
379 from nltk.stem import RSLPStemmer
380 #import nltk
381 #nltk.download('rslp')
382
383 stemmer = RSLPStemmer()
384
385 coluna_tmp = [0] * len(df_filter)
386
387 for i in range(l):
388
389     doc = nlp(str(df_filter.iloc[i][nome_coluna]))
390     tokens = doc.text.split()
391
392     temp = ""
393     for token in tokens:
394
395         if token != "nan":
396             temp = temp + " " + stemmer.stem(token)
397
398     coluna_tmp[i] = temp
399
400     printProgressBar(i + 1, l, prefix = 'Progress:', suffix = 'Complete', length = 50)
401
402 del doc, temp, tokens, stemmer
403
404 # insere nova coluna filtrada do dataframe
405 insertColumn("Stemmer", coluna_tmp)
406 print("Stemming completo.")
407
408 ################################################################
409
```

Fonte: Autor.

Ao final do processamento, o *dataframe* “*df_filter*” possui uma coluna com os termos que serão utilizados nos processos seguintes, apresentado na figura 26.

Figura 26: Snapshot da nova coluna “*Stemmer*” do *dataframe*, com termos finais. Coluna ao lado “*New*” apresenta o texto antes do processo de *stemming*.

New	Stemmer
troca sensores calibracao trocados sensores resistencia acompanhado calibracao	troc sensor calibraca troc sensor resistenc acompanh calibraca
efetuada troca filtros	efetu troc filtr
realizadas medicoes rotacao protocolo qualificacao testes realizados teste titulo …	realiz medico rotaca protocol qualificaca test realiz test titul verificaca funcion motor agit resistenc aq…
mantem temperatura desconhecida possivel baixa vazao entrada primeiras verificado bom…	mant temperat desconhec possi baix vaza entr prim verific bomb fic temperat ating comec cair faz bomb f…
vazamento agua celula vazamento sido seco pessoal avisado pessoal	vaz agu celul vaz sid sec pessoal avis pessoal
maquina liga bug colocado modo prog retornado	maquin lig bug coloc mod prog retorn
resetar captador piso tecnico possivel limpeza resetado captador	reset capt pis tecn possi limp reset capt
imprime desconhecida retirada usada devolveram verificado programacao parametro …	imprim desconhec retir us devolv verific programaca parametr entr config ativ wolk coloc sensor adiant pa…
falha sinal pedal cabo rompido feito reparo acompanhamento	falh sinal pedal cab romp feit repar acompanh
perdendo aquecimento cabo resistencia rompido feito isolacao cabo religado	perd aquec cab resistenc romp feit isolaca cab relig
alarme sensor mau contato feito reaperto cabo sensor	alarm sensor mau contat feit reapert cab sensor
montagens equipamentos treinamentos	mont equip trein
mordente dianteiro cabo resistencia rompido passado realizar troca resistencia mordente …	mord diant cab resistenc romp pass realiz troc resistenc mord diant fiaca romp const variaco tempera…
sensor queimado trocado sensor	sensor queim troc sensor
falha sensor valvula setup trocados conectores cabos sensor valvula fechada	falh sensor valvul setup troc conec cab sensor valvul fech
nan	
impressora wolke imprime cabos cabecas impressao estavam invertidos ocorrido retorn…	impres wolk imprim cab cabec impressa est invert ocorr retorn impres utiliz test linh feit trein ferra…
feito varios treinamentos organizacao planta volta operacoes	feit vari trein organizaca plant volt operaco
terminada organizacao planta ligada maquinas voltar atividades producao	termin organizaca plant lig maquin volt ativ produca
feito ajustes acompanhamento funcionamento inkjet gerando alarmes gotas deficientes fal…	feit ajust acompanh funcion inkjet ger alarm got defici falh plac coloc tint vicuos feit limp cabec im…
feita verificacao painel eletrico gerando alarmes softstarter ventilador disjutor desa…	feit verificaca painel eletr ger alarm softstart ventil disju desarm circuit eletr feit localizaca com…
painele eletrico collete desligado cabos conexoes desligadas interior painel devido t…	painel eletr collet deslig cab conexo deslig interi painel dev trabalh realiz maquin houv remont par ano …
feito acompanhamento processo cremes fryma gerando alarmes diversas valvulas aleatoriam…	feit acompanh process crem frym ger alarm divers valvul aleat par process const verific bloc eletroval…
potenciometro vibracao calha quebrado substituir potenciometro lateral painel elet…	potenciometr vibraca calh quebr substitu potenciometr later painel eletr feit test potenciometr tinh dispon…
ajustes processo insere bula cartucho feito ajuste ciclo insercao armando bula passo atr…	ajust process ins bul cartuch feit ajust cicl inserca arm bul pass atras ajust cicl cham bul arm cartuch se…
reprovacao etiqueta caixa verificado caixa permanecia parada roletes reprovacao aprovac…	reprovaca etiquet caix verific caix permanec par rolet reprovaca aprovaca elimin ferrament leit codig …
termino ligacao valvula bomba vacuo testes feito termino ligacao terminais valvula bomb…	termin ligaca valvul bomb vacu test feit termin ligaca term valvul bomb vacu
instalacao sensor valvula feito instalacao sensor testes colocado cabo terra sensor pos…	instalaca sensor valvul feit instalaca sensor test coloc cab terr sensor possu fio
linha veio apresentar problema gerava erro bomba envio ligando feito verificacao anorma…	linh vei apresent problem ger err bomb envi lig feit verificaca anorm encontr ped auxili pessoal engenh ve…
feita ligacao motor misturado fabsol desligado ano manutencao sala feito testes l…	feit ligaca motor mistur fabsol deslig ano manutenca sal feit test liber
varia velocidade knob solto realizado reaperto fixacao knob	var veloc knob solt realiz reapert fixaca knob
silverson reseta emergencia desconhecida contato micro mau fechado feito reaperto con…	silverson reset emergenc desconhec contat micr mau fech feit reapert contat micr acomp
verificacao funcionamento valvulas cip	verificaca funcion valvul cip crem constat acion

Fonte: Autor.

4. Análise e Exploração dos Dados

Ao analisar o texto normalizado, verificou-se a necessidade de utilizar algum recurso para o aprendizado do vocabulário dos textos, para possibilitar em seguida análise e exploração mais adequada. Em outras palavras, os termos deveriam ser condensados de alguma forma para facilitar compreensão.

Até então, entendia-se que a apresentação somente do conjunto de termos e suas freqüências em qualquer que fosse a ferramenta de visualização escolhida – por exemplo, *nuvem de palavras* -, não agregaria valor considerável à análise final, no entanto, o estudo das freqüencias dos termos deveria ser uma etapa inicial para agrupamento dos dados.

Para iniciar o cálculo das freqüências dos termos nos documentos, uma maneira simples e muito comum de representação é a chamada *bag-of-words* (bolsa de palavras, em tradução literal). A ordem das palavras é ignorada, e é criado um vetor onde seu tamanho é igual à quantidade de palavras no vocabulário, e seus elementos representam a quantidade de vezes que cada palavra ocorre em cada documento. (GENTZKOW, KELLY, e TADD, 2019)

Este esquema pode ser extendido para contagem de frases com um número limitado de termos dependentes, ao invés de apenas palavras únicas. Uma frase de lagura *n* pode ser referida como um *n-gram.* Um *bi-gram* e um *tri-gram* indicam frases com relação entre 2 termos e 3 termos respectivamente, e uma representação de *bag-of-words* corresponde tipicamente à contagem de *uni-grams*. (GENTZKOW, KELLY, e TADD, 2019)

Estes conceitos seriam especialmente importantes para o objetivo final da análise, pois além de entender quais termos eram mais citados nas ordens de serviço, seria necessário também entender a relação entre pelo menos 2 termos. Por exemplo, a palavra “falha”, observada freqüentemente nos documentos em amostragem inicial, geralmente vinha acompanhada de um objeto alvo, como por exemplo, “sensor”. Portanto, entender a freqüência dos termos “falha sensor” na mesma frase taria valor para o objetivo da análise.

As funções de criação de *n-grams* e vetor de palavras, além de outras funções de aprendizado de máquinas para NLP foram encontradas dentro da biblioteca *scikit-learn* contida na distribuição *Anaconda*. A tabela 4 demonstra a função utilizada.

Tabela 4: Biblioteca utilizada para criação do vetor de palavras e *n-grams*.

biblioteca	**Descrição**	**Comando(s) utilizados**
scikit-learn	Pacote de ferramentas para *machine-learning.* Informações: https://scikit-learn.org/stable/	from sklearn.feature_extraction.text import CountVectorizer

Fonte: Autor.

O código para criação do vetor de palavras utilizando a classe *CountVectorizer()* foi implementado em cima do texto processado, ou seja, após todas as camadas de tratamento realizadas anteriormente, e é apresentado na figura 27.

Figura 27: Código para criação do vetor de palavras e criação do vocabulário.

```
421
422 ############################################################
423 # Criar matrix de termos, da coluna Stemmer
424 ############################################################
425
426 corpus = df_filter['Stemmer']
427
428 from sklearn.feature_extraction.text import CountVectorizer
429
430 cv=CountVectorizer(max_df=0.8, max_features=10000, ngram_range=(1,3), tokenizer=lambda x: x.split(' '))
431
432 X = cv.fit(corpus)
433
```

Fonte: Autor.

É possível notar que no código apresentado, o parâmetro "*ngram_range=(1,3)"* configura a função para procurar relações freqüentes de até 3 palavras como *n-grams*, e utilizá-las no cálculo de sua saída.

Primeiramente é criado uma variável "*cv*" com o retorno da classe *CountVectorizer(),* e logo é chamada a função "*fit"* para aprender e construir o vocabulário de palavras, na linha "*X = cv.fit(corpus)"*. O resultado desta execução pode ser impresso, pelo código demonstrado na figura 28. O parâmetro "*max_df=0.8"* foi escolhido impiricamente através de sucessivos testes, pois tem o

objetivo de eliminar relações muito freqüentes entre termos, indicando assim uma possível *stopword* que não foi removida adequadamente.

Figura 28: Impressão dos 30 primeiros itens do vocabulário aprendido.

```
In [27]: list(cv.vocabulary_.keys())[:30]
Out[27]:
['troc',
 'sensor',
 'calibraca',
 'resistenc',
 'acompanh',
 'troc sensor',
 'acompanh calibraca',
 'efetu',
 'filtr',
 'efetu troc',
 'troc filtr',
 'realiz',
 'medico',
 'rotaca',
 'qualificaca',
 'test',
 'verificaca',
 'funcion',
 'motor',
 'agit',
 'aquec',
 'leit',
 'frequenc',
 'igual',
 'set',
 'inver',
 'real',
 'limit',
 'esper',
 'veloc']
```

Fonte: Autor.

Como esperado, o vocabulário aprendido apresenta não apenas *uni-grams*, porém já alguns *bi-grams* nas primeiras posições do vetor.

A conclusão encontrada é que a exploração dos dados deveria ter como base o *ranking individual* de *uni-grams*, *bi-grams* e *tri-grams* gerados a partir da fonte de dados textuais. Para tal, foram desenvolvidos 3 códigos para gerar cada um destes *dataframes*, de tamanho igual a 80. Este número se mostrou interessante após diversos testes, pois apresentava uma população de termos significativa para a interpretação e já ajudaria na tomada de algumas conclusões. Importante mencionar também que os dados textuais do campo “*WO LONG DESCRIPTION*” apresentou melhor riqueza de termos gerados em relação ao campo “*Description*”, pois este último era demasiadamente sucinto. Esta condição também foi concluída após diversos testes impíricos.

O *ranking* de *uni-grams* foi armazenado no *dataframe top_df*, conforme apresentado na figura 29. Este *dataframe* de resultado é apresentado parcialmente na figura 30. Notar que o *script* utiliza a mesma função de aprendizado de vocabulário sobre o vetor de palavras, *CountVectorizer().fit(),* presente na biblioteca *scikit-learn,* testada anteriormente.

Figura 29: Código para geração do *ranking* de *uni-grams*.

```
458 ##############################################################
459 # Palavras mais frequentes (variável CORPUS)
460 ##############################################################
461
462 #n = tamanho da lista
463 n = 80
464
465 from sklearn.feature_extraction.text import CountVectorizer
466
467 def get_top_n_words(corpus, n=None):
468
469     vec = CountVectorizer(max_df=0.8, max_features=10000).fit(corpus)
470     bag_of_words = vec.transform(corpus)
471     sum_words = bag_of_words.sum(axis=0)
472
473     words_freq = [(word, sum_words[0, idx]) for word, idx in
474                   vec.vocabulary_.items()]
475
476     words_freq =sorted(words_freq, key = lambda x: x[1],
477                        reverse=True)
478
479     return words_freq[:n]
480
481 top_words = get_top_n_words(corpus, n=n)
482
483 top_df = pd.DataFrame(top_words)
484 top_df.columns=["Word", "Freq"]
485
486 ##############################################################
```

Fonte: Autor.

Figura 30: *Snapshot* parcial do *dataframe* de *uni-grams.*

top_df - DataFrame

Index	Word	Freq
0	feit	7213
1	ajust	4729
2	maquin	4025
3	falh	3507
4	acompanh	2973
5	realiz	2968
6	test	2544
7	sensor	2483
8	cartuch	2394
9	fic	2118
10	process	2044
11	trein	1856
12	visa	1788
13	ferrament	1530
14	liber	1477
15	linh	1363
16	nosica	1253

Fonte: Autor.

O *ranking* de *bi-grams* foi armazenado no *dataframe top2_df*, conforme apresentado na figura 31. Este *dataframe* de resultado é apresentado parcialmente na figura 32.

Figura 31: Código para geração do *ranking* de *bi-grams*.

```
483
484 ############################################################
485 # Bi-grams mais frequentes (variável CORPUS)
486 ############################################################
487
488 from sklearn.feature_extraction.text import CountVectorizer
489
490 def get_top_n2_words(corpus, n=None):
491
492     vec2 = CountVectorizer(ngram_range=(2,2),max_features=10000, max_df=0.8).fit(corpus)
493     bag_of_words = vec2.transform(corpus)
494     sum_words = bag_of_words.sum(axis=0)
495
496     words_freq = [(word, sum_words[0, idx]) for word, idx in
497                   vec2.vocabulary_.items()]
498     words_freq =sorted(words_freq, key = lambda x: x[1],
499                 reverse=True)
500
501     return words_freq[:n]
502
503
504 top2_words = get_top_n2_words(corpus, n=n)
505 top2_df = pd.DataFrame(top2_words)
506 top2_df.columns=["Bi-gram", "Freq"]
507
508 ############################################################
509
```

Fonte: Autor.

Figura 32: *Snapshot* parcial do *dataframe* de *bi-grams.*

top2_df - DataFrame

Index	Bi-gram	Freq
0	feit ajust	1649
1	feit acompanh	1252
2	feit test	856
3	trein ferrament	750
4	acompanh process	730
5	realiz test	651
6	feit verificaca	614
7	ajust visa	496
8	liber maquin	420
9	feit reset	396
10	maquin liber	389
11	ajust posica	389
12	acompanh maquin	382
13	test fic	344
14	feit trein	340
15	ajust sensor	326
16	realiz ajust	322

Fonte: Autor.

Ainda não seria conclusivo saber se gerar o *ranking* de *tri-grams* enriqueceria a análise, no entanto, este processo foi implementado para fins de avaliação. Seu código é apresentado na figura 33. O *dataframe top3_df* que armazena o resultado é apresentado parcialmente na figura 34.

Figura 33: Código para geração do *ranking* de *tri-grams*.

```
517
518 ############################################################
519 # Tri-grams mais frequentes (variável CORPUS)
520 ############################################################
521
522 from sklearn.feature_extraction.text import CountVectorizer
523
524 def get_top_n3_words(corpus, n=None):
525
526     vec3 = CountVectorizer(ngram_range=(3,3),max_features=10000, max_df=0.8).fit(corpus)
527
528     bag_of_words = vec3.transform(corpus)
529
530     sum_words = bag_of_words.sum(axis=0)
531
532     words_freq = [(word, sum_words[0, idx]) for word, idx in
533                   vec3.vocabulary_.items()]
534     words_freq =sorted(words_freq, key = lambda x: x[1],
535                 reverse=True)
536
537     return words_freq[:n]
538
539 top3_words = get_top_n3_words(corpus, n=n)
540 top3_df = pd.DataFrame(top3_words)
541 top3_df.columns=["Tri-gram", "Freq"]
542
543
544 ############################################################
545
```

Fonte: Autor.

Figura 34: *Snapshot* parcial do *dataframe* de *tri-grams.*

top3_df - DataFrame

Index	Tri-gram	Freq
0	feit acompanh maquin	250
1	feit acompanh process	240
2	trein ferrament visa	203
3	feit test func	202
4	feit ajust posica	196
5	feit trein ferrament	172
6	realiz test fic	169
7	feit ajust ferrament	163
8	fic maquin liber	159
9	feit verificaca vist	157
10	maquin feit acompanh	156
11	feit acompanh fic	153
12	feit test acompanh	151
13	acompanh maquin fic	146
14	feit ajust sensor	144
15	desajust feit ajust	130
16	liber maquin feit	127
17	ajust visa cartuch	123

Fonte: Autor.

Após criação dos *dataframes* de resposta para os *uni-grams*, *bi-grams* e *tri-grams,* seria necessário apresentá-los de forma a facilitar a interpretação dos resultados. Para tal foi escolhida como forma de apresentação a utilização de gráficos, conforme conceitos estudados no decorrer do curso. A própria distribuição *Anaconda* disponibiliza ferramentas para este fim, e foi utilizada a biblioteca *seaborn* apresentada na tabela 5.

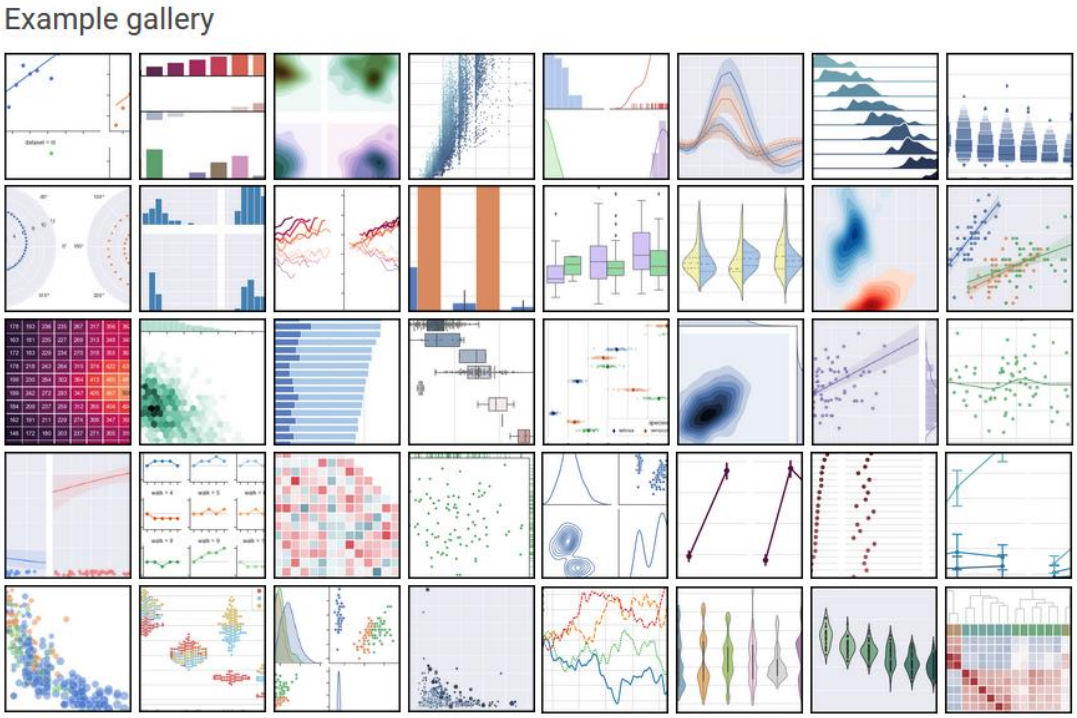
Tabela 5: Biblioteca utilizada para geração da visualização.

biblioteca	**Descrição**	**Comando(s) utilizados**
Seaborn	Pacote de ferramentas para visualização de dados. Informações: https://seaborn.pydata.org/	import seaborn as sns

Fonte: Autor.

A biblioteca *seaborn* possui implementação de diversos tipos de gráficos, e alguns exemplos estão apresentados na figura 35.

Figura 35: Exemplos de tipos de gráficos da biblioteca *seaborn*.



Fonte: Adaptado de https://seaborn.pydata.org/examples/index.html.

Para plotar o gráfico de barras para cada *dataframe*, foi desenvolvida uma função genérica que utiliza a biblioteca *seaborn* e o comando *barplot*. Esta função espera como parâmetros o nome das colunas e o *dataframe* em questão, e seu código é apresentado na figura 36.

Figura 36: Função genérica para plotar gráficos.

```
565
566 ##############################################################
567 # função para plotar gráficos de barras
568 ##############################################################
569
570 def plotar(nomeX, nomeY, dados):
571
572     import seaborn as sns
573     sns.set(rc={'figure.figsize':(20,15)})
574
575     g = sns.barplot(y=nomeY,x=nomeX, data=dados)
576
577     g.tick_params(labelsize=8)
578
579     g.set_xlabel(nomeY,fontsize=10)
580     g.set_ylabel(nomeX,fontsize=10)
581
582     return
583
584 ##############################################################
585
```

Fonte: Autor.

Desta maneira foi possível plotar os 3 gráficos através dos comandos customizados para cada *dataframe*, apresentado na figura 37.

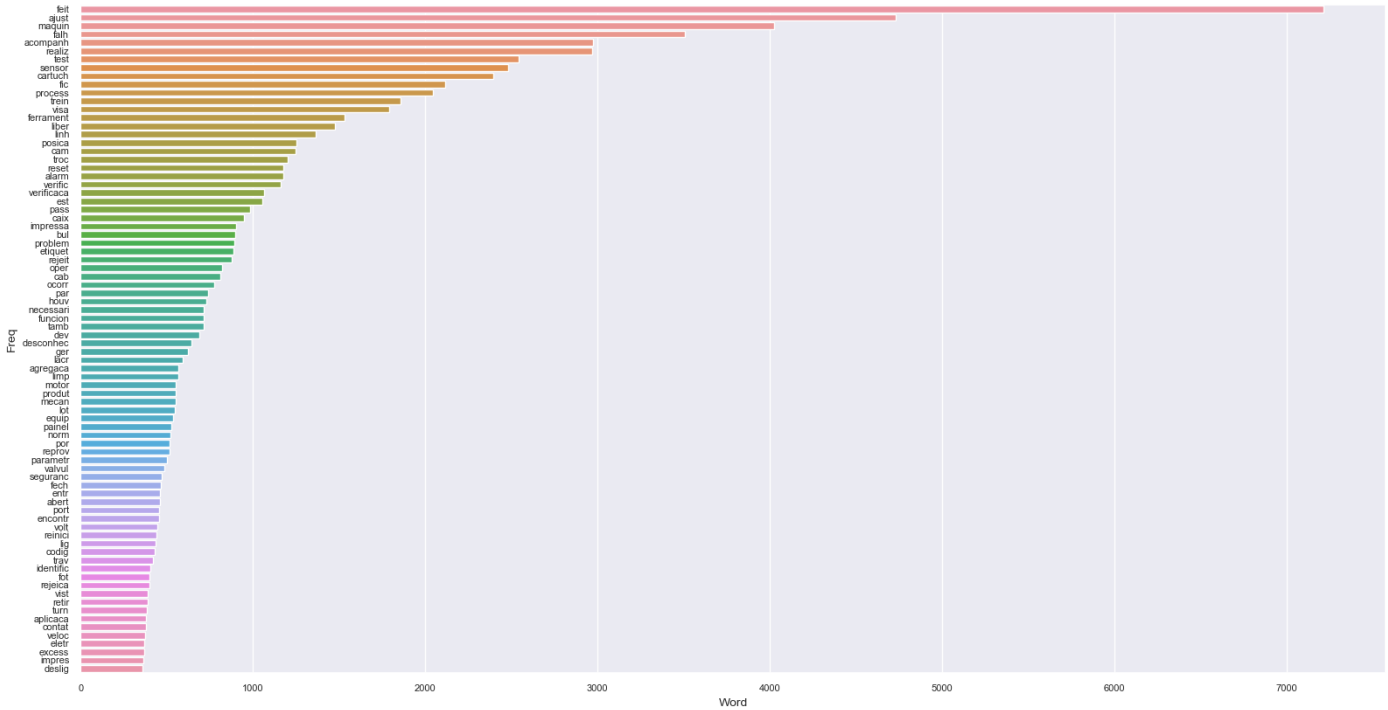
Figura 37: Código para chamar função para plotar o gráfico de cada *dataframe*.

```
588 ##############################################################
589 # plota gráfico de cada dataframe
590 ##############################################################
591
592 plotar(nomeX="Freq",nomeY ="Word",dados=top_df)
593 plotar(nomeX="Freq",nomeY ="Bi-gram",dados=top2_df)
594 plotar(nomeX="Freq",nomeY ="Tri-gram",dados=top3_df)
595
596 ##############################################################
```

Fonte: Autor.

O primeiro *ranking* de *uni-grams* apresentou o seguinte resultado, de acordo com figura 38:

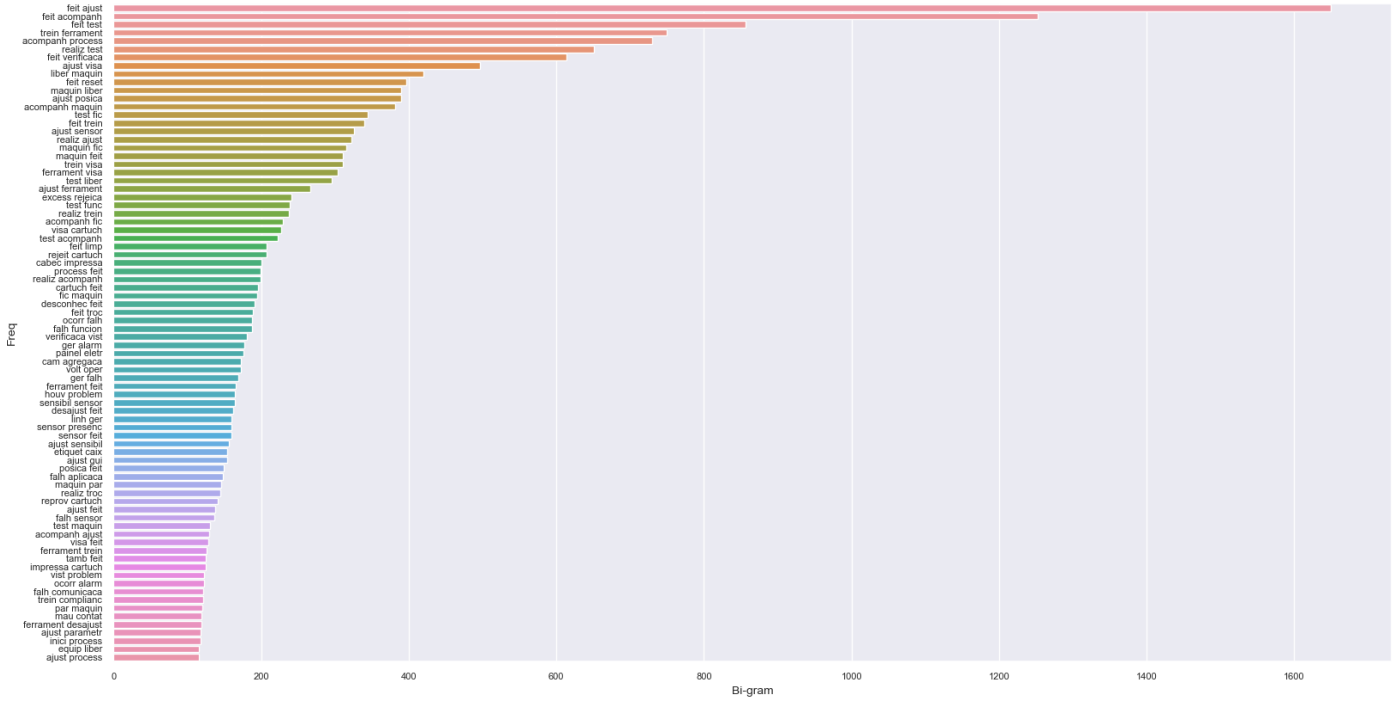
Figura 38: Gráfico do *ranking* de *uni-grams*.



Fonte: Autor.

O primeiro *ranking* de *bi-grams* apresentou o seguinte resultado, de acordo com figura 39:

Figura 39: Gráfico do *ranking* de *bi-grams*.

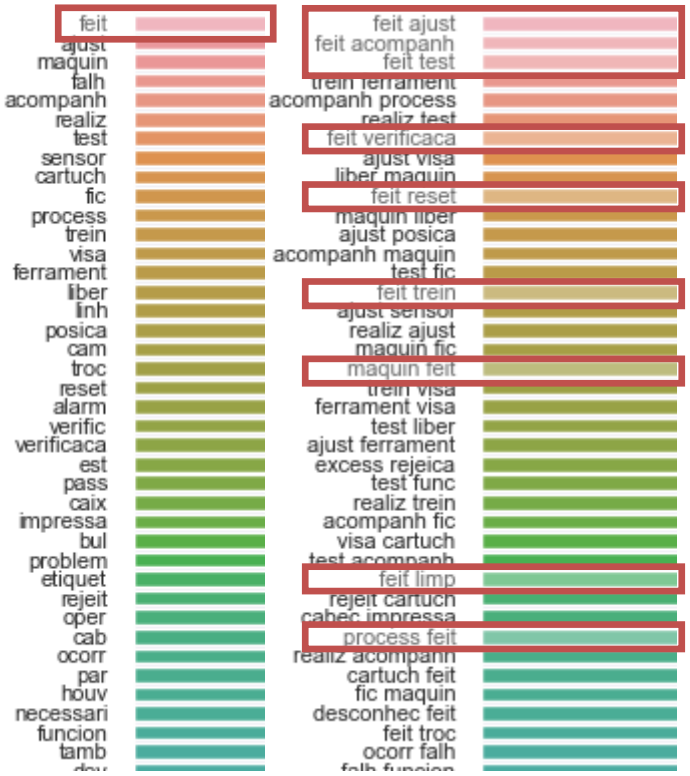


Fonte: Autor.

É possível claramente notar que houve um “estouro” de ambos os gráficos para alguns termos comuns. No resultado de *uni-grams* por exemplo, é visível o termo “feit” como primeiro colocado, e sua relação com “ajust” (“feit ajust”), “acompanh” (“feit acompanh”), “test” (“feit test”), “verificaca” (“feit verifica”) entre outros, no resultado de *bi-grams.*

Fazendo um comparativo entre os dois *rankings* lado a lado, levando em consideração a freqüência do termo “feit”, chega-se a seguinte observação, apresentada parcialmente na figura 40:

Figura 40: Comparação entre *uni-grams* e *bi-grams* e termo “feit”*.*



Fonte: Autor.

Isto leva a crer que o termo “feit” pode potencialmente ser categorizado como uma *stopword,* visto que além de sua alta freqüência, o mesmo não apresenta significado real para o resultado da análise. Foi decidido criar um vetor adicional de *stopwords* para o algoritmo *CountVectorizer()* utilizar, que deve ser aplicado no momento de aprendizado do vocabulário. Segundo documentação da biblioteca, apresentada na figura 41, é possível informar uma lista de termos para a função utilizar como *stopwords*, além do parâmetro “*max_df”* já mencionado anteriormente.

Figura 41: Parâmetros para limitar o aprendizado de termos muito freqüentes.

stop_words : *string {'english'}, list, or None (default)*

If 'english', a built-in stop word list for English is used. There are several known issues with 'english' and you should consider an alternative (see Using stop words).

If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens. Only applies if `analyzer == 'word'`.

If None, no stop words will be used. max_df can be set to a value in the range [0.7, 1.0) to automatically detect and filter stop words based on intra corpus document frequency of terms.

max_df : *float in range [0.0, 1.0] or int, default=1.0*

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

Fonte: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

A lista de *stopwords* definidas está apresentada na figura 42, e foram incluídos outros termos além do previamente citado "feit". Estes termos foram escolhidos após análise criteriosa dos gráficos resultantes.

Figura 42: *Stopwords* para o aprendizado, após análise inicial dos resultados.

```
444 ############################################################
445 # Stopwords complementares para CountVectorizer
446 ############################################################
447
448 stopwords_aprendizado = ['feit','liber', 'fic', 'tamb', 'ocorr']
449
450
451 ############################################################
452
```

Fonte: Autor.

Aplicou-se a lista "*stopwords_aprendizado"* como parâmetro para geração de cada *dataframe* novamente, conforme apresentado na figura 43.

Figura 43: *Stopwords* para o aprendizado, após análise inicial dos resultados.

```
...
469     vec = CountVectorizer(max_df=0.8, max_features=10000, stop_words = stopwords_aprendizado).fit(corpus)
...
506     vec2 = CountVectorizer(ngram_range=(2,2),max_features=10000, max_df=0.8, stop_words = stopwords_aprendizado).fit(corpus)
...
540     vec3 = CountVectorizer(ngram_range=(3,3),max_features=10000, max_df=0.8, stop_words = stopwords_aprendizado).fit(corpus)
...
```

Fonte: Autor.

A utilização de uma lista de *stopwords* neste momento do processo mostrou-se bastante efetiva, pois como o texto já encontra-se normalizado e com aplicação

de *stemming*, esta lista pôde ser criada de uma forma muito mais optimizada do que em comparação com a criação da mesma no início do processo de limpeza do texto. Por exemplo, o termo “liber”, já elimina a possível ocorrência dos termos anteriores “liberada”, “liberou”, “liberei”, “liberar”, entre outros. Além disso, qualquer termo que por ventura tenha passado pela limpeza sem eliminação, tem agora a oportunidade de ser removido neste pós-processamento, como aconteceu por exemplo com o termo “tamb” (“também”), que deveria ter sido eliminado anteriormente.

A primeira conclusão tomada é que o resultado do *ranking* de *uni-grams* não traz muito significado para a análise, e foi útil apenas para auxiliar na verificação de termos recorrentes para eliminação, conforme já discutido. Apesar disso, é possível entender a partir da análise destas freqüências que existe uma forte ocorrência de manutenções que são de fato apenas “ajustes”: muitas palavras indicam esta natureza de serviço, como a propria palavra “ajust” em primeiro lugar no *ranking*, e “trein”. O termo “trein” pode indicar treinamento de um sistema, treino de um sensor, treino de uma câmera, entre outros, e é muito utilizado para indicar o procedimento de *teaching* de um dispositivo eletrônico ou sistema, e não está relacionado somente com o sentido de “treinar a pessoa”, por exemplo.

Na análise dos *bi-grams*, esta conclusão se concretiza: o conjunto de palavras que ocorre é exatamente “trein ferrament”, em primeira posição do *ranking* com mais de 700 ocorrências, o que indica que a ação mais realizada pelos manutentores é o treino de ferramentas do sistema de visão da fábrica, ou seja, o *teaching* do sistema de visão computacional, presente em diversas máquinas desta fábrica em questão. Esta ação se repete de formas variadas nas posições seguintes do *ranking*, como sintetizado na tabela 6:

Tabela 6: Termos potencialmente relacionados com o mesmo assunto.

Termos	**Posição no *ranking***
trein ferrament	1
ajust visa	4
trein visa	9
ferrament visa	11

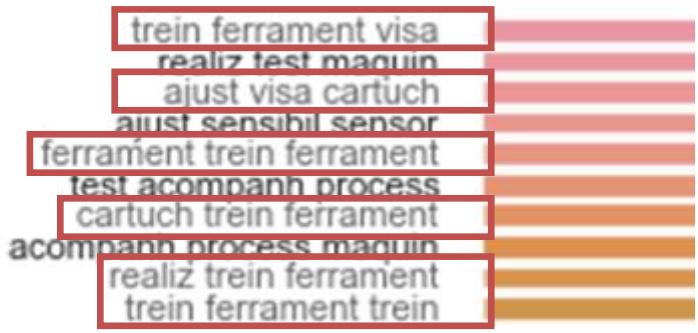
ajust ferrament	13
excess rejeic	15
realiz trein	16
visa cartuch	18

Fonte: Autor.

O termo “excess rejeic”, 15ª posição do *ranking*, é provavelmente um sintoma identificado, de forma que o ajuste de um sistema de visão é a ação para solução do problema de excesso de rejeitos, então foi considerado igualmente como um conjunto de termos relacionados com o mesmo tópico, na tabela 6. O item “realiz trein”, posição 16 do *ranking*, não é explícito para o treino do sistema, e pode indicar o treino de uma pessoa por exemplo – esta condição será conclusiva provavelmente na análise dos *tri-grams*. Já o termo “visa cartuch” referencia explicitamente o sistema de visão de cartuchos, que no caso, trata-se da inspeção por visão computacional dos textos impressos nos cartuchos – produto manufaturado nesta fábrica –, indicando que o foco dos ajustes em sua maioria está neste tipo de inspeção/sistema.

Partindo para interpretação dos *tri-grams*, novamente os termos "trein ferrament visa" indicam um pontecial problema crônico voltado para o ajuste dos sistemas de visão da fábrica, em colocação número 1 no *ranking*, conforme mostra a figura 44.

Figura 44: *Top* 10 de termos tri-grams.



Fonte: Autor.

Reforçando esta hipótese, logo na 3ª posição do *ranking*, os termos "ajust visa cartuch" indicam que o sistema de visão mais ajustado é realmente o de inspeção visual do cartucho. Isto se repete nas demais posições do gráfico de *tri-grams*, de forma que os 10 primeiros itens possuem relação com ajustes no sistema de visão, ou nas ferramentas do sistema de visão, usualmente na inspeção de cartuchos.

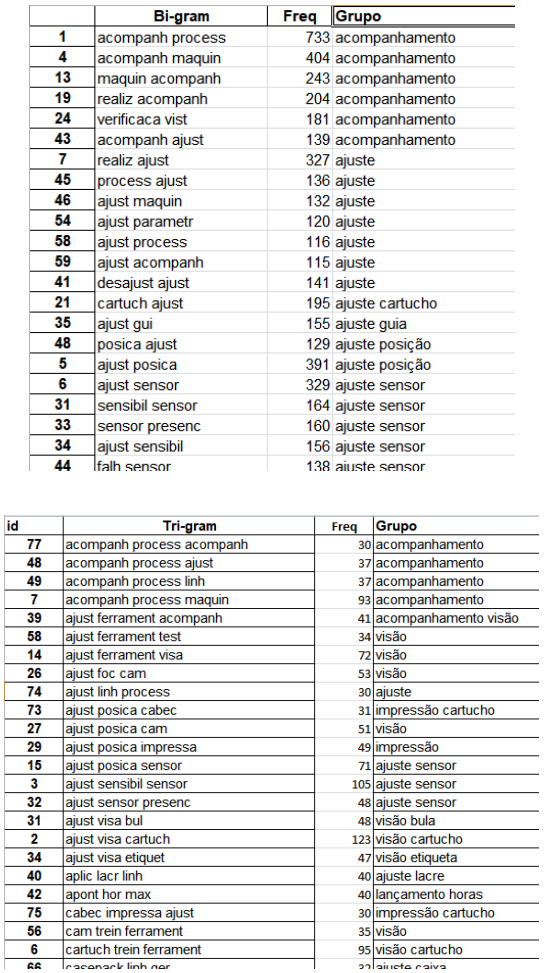
Como esperado, os 3 *rankings* gerados se relacionam entre si, pois aparentemente o *ranking* de maior número de *n-grams* agrega conteúdo para interpretação do *ranking* anterior de menor número de *n-grams*. Desta forma, conclui-se que o *ranking* mais significante para a análise foi o de *tri-grams*, que detalha melhor as ações realizadas, e os objetos alvo das manutenções.

5. Considerações sobre Modelos de Machine Learning

Através da interpretação destes *rankings*, chegou-se à conclusão de que os *n-grams* deveriam ser também classificados para aumentar ainda mais o agrupamento, e potencialmente auxiliar na posterior interpretação e apresentação dos resultados. O intuito agora seria o de classificar em qual categoria cada conjunto de *n-grams* poderia se encontrar.

Notar que para esta atividade de classificação foi necessária a interpretação dos dados, conhecimento dos equipamentos e atividades da área – ou seja, conhecimento do negócio – e posteriormente uma atividade manual de atribuição de cada conjunto de *tri-gram* ou *bi-gram* a uma categoria definida. Para tal, foram geradas tabelas de apoio, e um snapshot parcial de cada uma delas é apresentado na figura 45.

Figura 45: *Snapshot* parcial das tabelas com *n-grams* classificados.



	Bi-gram	Freq	Grupo
1	acompanh process	733	acompanhamento
4	acompanh maquin	404	acompanhamento
13	maquin acompanh	243	acompanhamento
19	realiz acompanh	204	acompanhamento
24	verificaca vist	181	acompanhamento
43	acompanh ajust	139	acompanhamento
7	realiz ajust	327	ajuste
45	process ajust	136	ajuste
46	ajust maquin	132	ajuste
54	ajust parametr	120	ajuste
58	ajust process	116	ajuste
59	ajust acompanh	115	ajuste
41	desajust ajust	141	ajuste
21	cartuch ajust	195	ajuste cartucho
35	ajust gui	155	ajuste guia
48	posica ajust	129	ajuste posição
5	ajust posica	391	ajuste posição
6	ajust sensor	329	ajuste sensor
31	sensibil sensor	164	ajuste sensor
33	sensor presenc	160	ajuste sensor
34	ajust sensibil	156	ajuste sensor
44	falh sensor	138	ajuste sensor

id	Tri-gram	Freq	Grupo
77	acompanh process acompanh	30	acompanhamento
48	acompanh process ajust	37	acompanhamento
49	acompanh process linh	37	acompanhamento
7	acompanh process maquin	93	acompanhamento
39	ajust ferrament acompanh	41	acompanhamento visão
58	ajust ferrament test	34	visão
14	ajust ferrament visa	72	visão
26	ajust foc cam	53	visão
74	ajust linh process	30	ajuste
73	ajust posica cabec	31	impressão cartucho
27	ajust posica cam	51	visão
29	ajust posica impressa	49	impressão
15	ajust posica sensor	71	ajuste sensor
3	ajust sensibil sensor	105	ajuste sensor
32	ajust sensor presenc	48	ajuste sensor
31	ajust visa bul	48	visão bula
2	ajust visa cartuch	123	visão cartucho
34	ajust visa etiquet	47	visão etiqueta
40	aplic lacr linh	40	ajuste lacre
42	apont hor max	40	lançamento horas
75	cabec impressa ajust	30	impressão cartucho
56	cam trein ferrament	35	visão
6	cartuch trein ferrament	95	visão cartucho
66	casepack linh ger	32	ajuste caixa

Fonte: Autor.

Vale ressaltar que para estas últimas classificações foram realizadas tentativas de automação utilizando biblioteca *scikit-learn* com o objetivo atribuir a cada documento ou ordem de serviço uma *keyword* (palavra-chave), porém sem sucesso. Após sucessivos testes, foi concluído que a interpretação humana seria importante neste passo, visto que existia a necessidade de entendimento do negócio para maximizar o agrupamento dos resultados encontrados, e de outra forma, os dados continuavam pouco aninhados pelo excesso de diferentes termos para o mesmo assunto.

Além disso, estudou-se a possibilidade de utilizar aprendizado supervisionado, porém esta atividade demandaria uma classificação prévia de dezenas de milhares de ordens de serviço para treino do modelo e posterior utilização. Não encontrou-se também disponível qualquer modelo pré-treinado que

atende-se ao idioma português e principalmente que possuísse os termos técnicos dos textos utilizados.

A partir destas conclusões, decidiu-se apenas classificar os *dataframes* finais de *n-grams* para condensar a apresentação final dos resultados, o que já atenderia os objetivos finais desta análise de dados. No entanto, vale enfatizar que este tópico poderá ser mais profundamente abordado em trabalhos posteriores.

Ao final, para esta classificação manual, cada *dataframe* de *n-grams* foi exportado em formato planilha editável, e seu código está apresentado na figura 46.

Figura 46: Código para exportar cada *dataframe* para edição em planilha externa.

```
565 ################################################################
566 # export de cada dataframe para classificação manual
567 ################################################################
568
569 top2_df.to_excel(caminho + "/groups_bigrams_raw.xlsx")
570 top3_df.to_excel(caminho + "/groups_trigrams_raw.xlsx")
571
572 ################################################################
```

Fonte: Autor.

Após geradas as respectivas classificações em formato tabela, estas foram novamente importadas para o ambiente *Spyder*, e utilizou-se da biblioteca *seaborn* mais uma vez para criação das novas visualizações mais condensadas, sendo necessário apenas agrupar os dados de cada *dataframe* através da função *groupby* da biblioteca *pandas*, conforme algoritmos demonstrados nas figuras 47 e 48 respectivamente, para os *bi-grams* e *tri-grams*.

Figura 47: Código para agrupar os dados e gerar o gráfico do resultado da classificação dos *bi-grams.*

```
606
607 ############################################################
608 # plota gráfico dos grupos de bi-grams
609 ############################################################
610
611 df_grupos_bigrams = pd.read_excel(caminho + '/groups_bigrams.xlsx', index_col=0)
612
613 df_grupos_filter = df_grupos_bigrams.groupby('Grupo').agg(
614         Total=pd.NamedAgg(column='Freq', aggfunc=sum),
615                                     ).sort_values('Total', ascending= False)
616
617 import seaborn as sns
618
619 y = df_grupos_filter['Total']
620 x = df_grupos_filter.index
621
622 ax = sns.barplot(y=y, x=x, data=df_grupos_filter)
623
624 ax.axes.set_title("Grupos de falhas e atividades, sobre bi-grams",fontsize=20)
625 ax.tick_params(labelsize=8, rotation=45)
626 ax.set_xlabel("Grupos",fontsize=10)
627 ax.set_ylabel("Freqüência",fontsize=10)
628
629 for p in ax.patches:
630     ax.text(p.get_x() + p.get_width()/2., p.get_height(), '%d' % int(p.get_height()),
631             fontsize=12, color='black', ha='center', va='bottom')
632
633 ############################################################
634
```

Fonte: Autor.

Figura 48: Código para agrupar os dados e gerar o gráfico do resultado da classificação dos *bi-grams.*

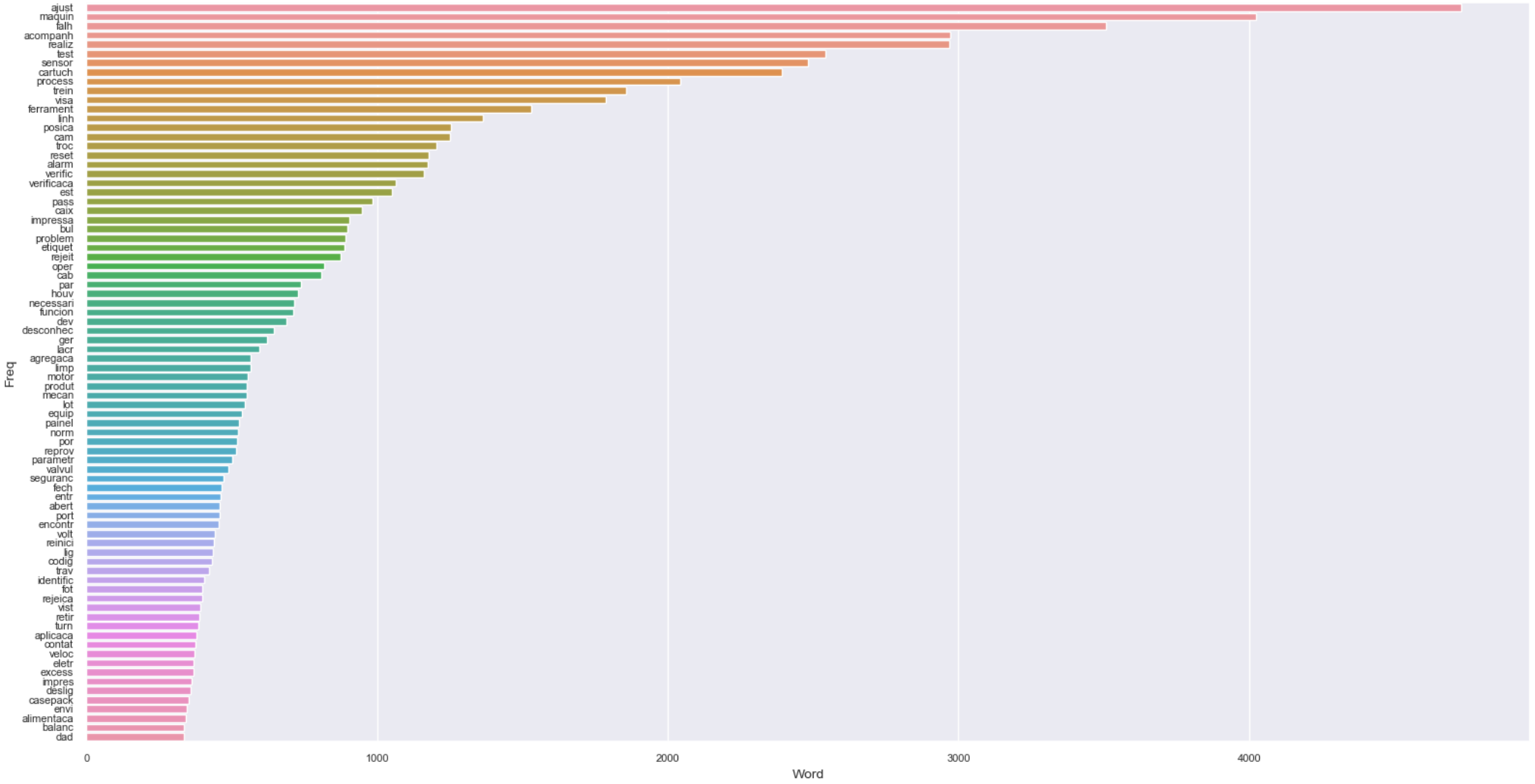
```
638
639 ############################################################
640 # plota gráfico dos grupos de tri-grams
641 ############################################################
642
643 df_grupos_trigrams = pd.read_excel(caminho + '/groups_trigrams.xlsx', index_col=0)
644
645 df_grupos_filter = df_grupos_trigrams.groupby('Grupo').agg(
646         Total=pd.NamedAgg(column='Freq', aggfunc=sum),
647                                     ).sort_values('Total', ascending= False)
648
649 import seaborn as sns
650
651 y = df_grupos_filter['Total']
652 x = df_grupos_filter.index
653
654 ax = sns.barplot(y=y, x=x, data=df_grupos_filter)
655
656 ax.axes.set_title("Grupos de falhas e atividades, sobre tri-grams",fontsize=20)
657 ax.tick_params(labelsize=8, rotation=45)
658 ax.set_xlabel("Grupos",fontsize=10)
659 ax.set_ylabel("Freqüência",fontsize=10)
660
661 for p in ax.patches:
662     ax.text(p.get_x() + p.get_width()/2., p.get_height(), '%d' % int(p.get_height()),
663             fontsize=12, color='black', ha='center', va='bottom')
664
665 ############################################################
666
```

Fonte: Autor.

6. Apresentação dos Resultados

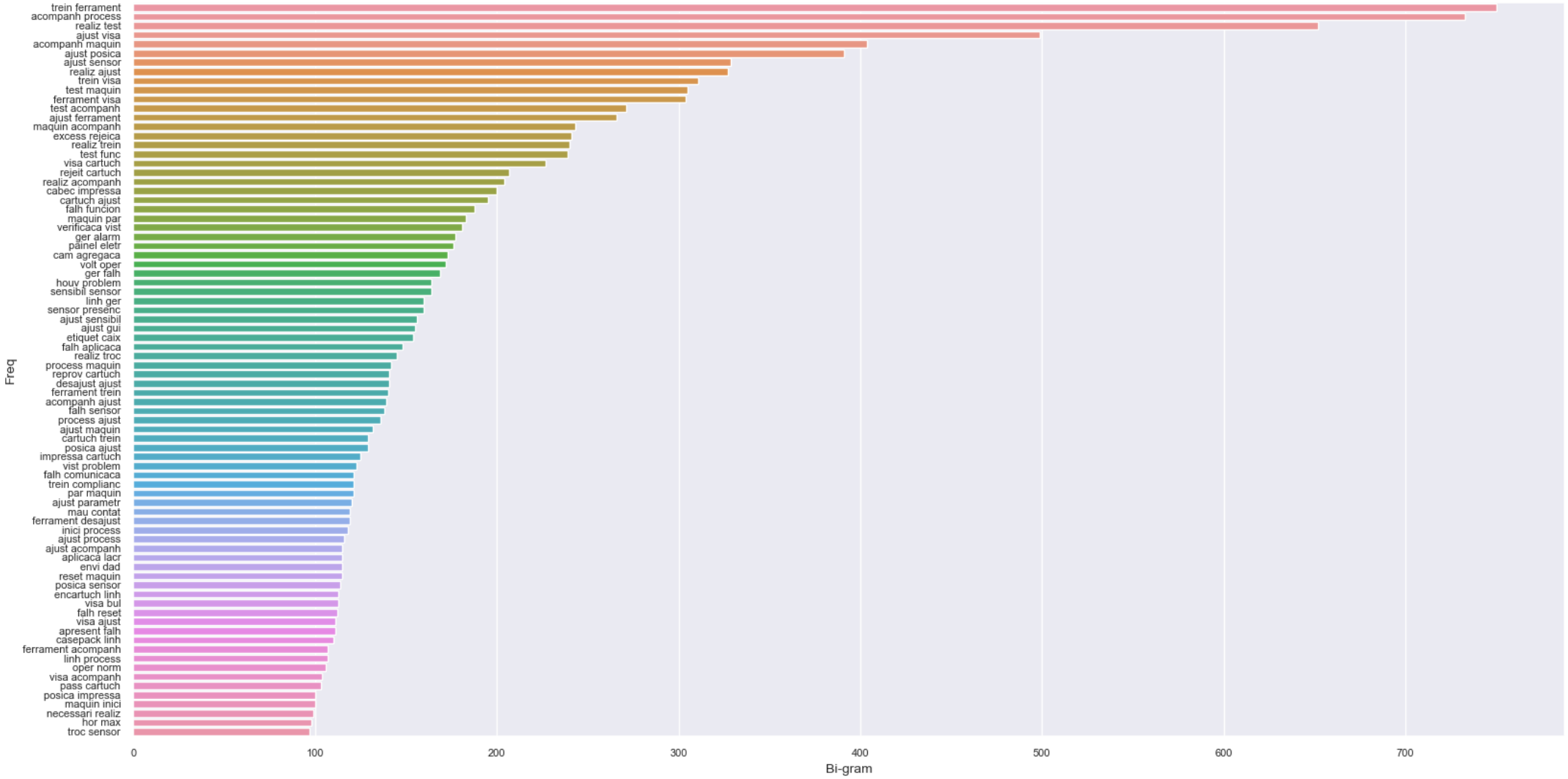
Finalmente, o resultado do gráfico final dos 3 *dataframes* de *uni-grams*, *bi-grams* e *tri-grams*, são apresentados respectivamente nas figuras 49, 50 e 51.

Figura 49: Gráfico final do *ranking* de *uni-grams.*



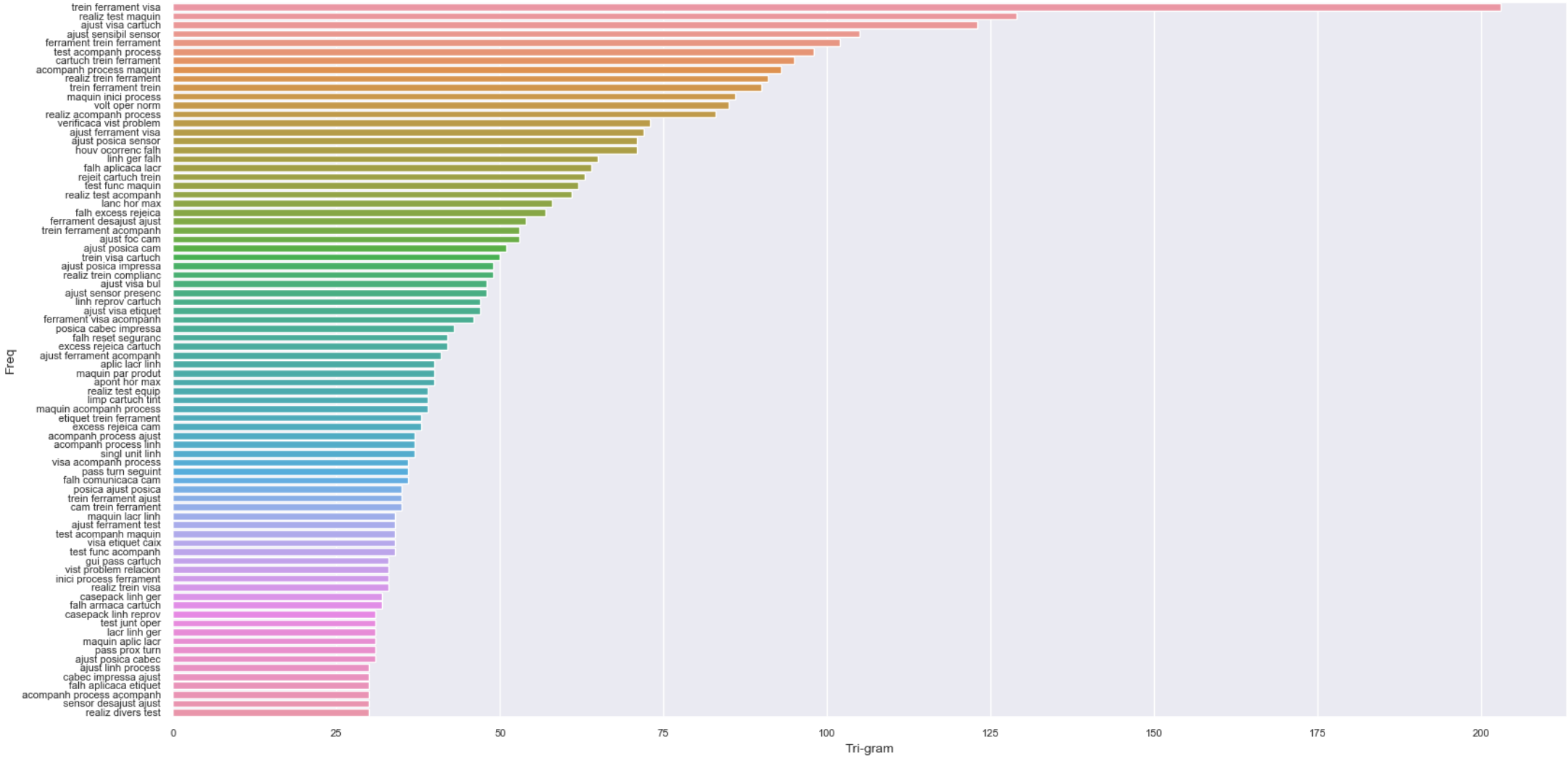
Fonte: Autor

Figura 50: Gráfico final do *ranking* de *bi-grams.*



Fonte: Autor

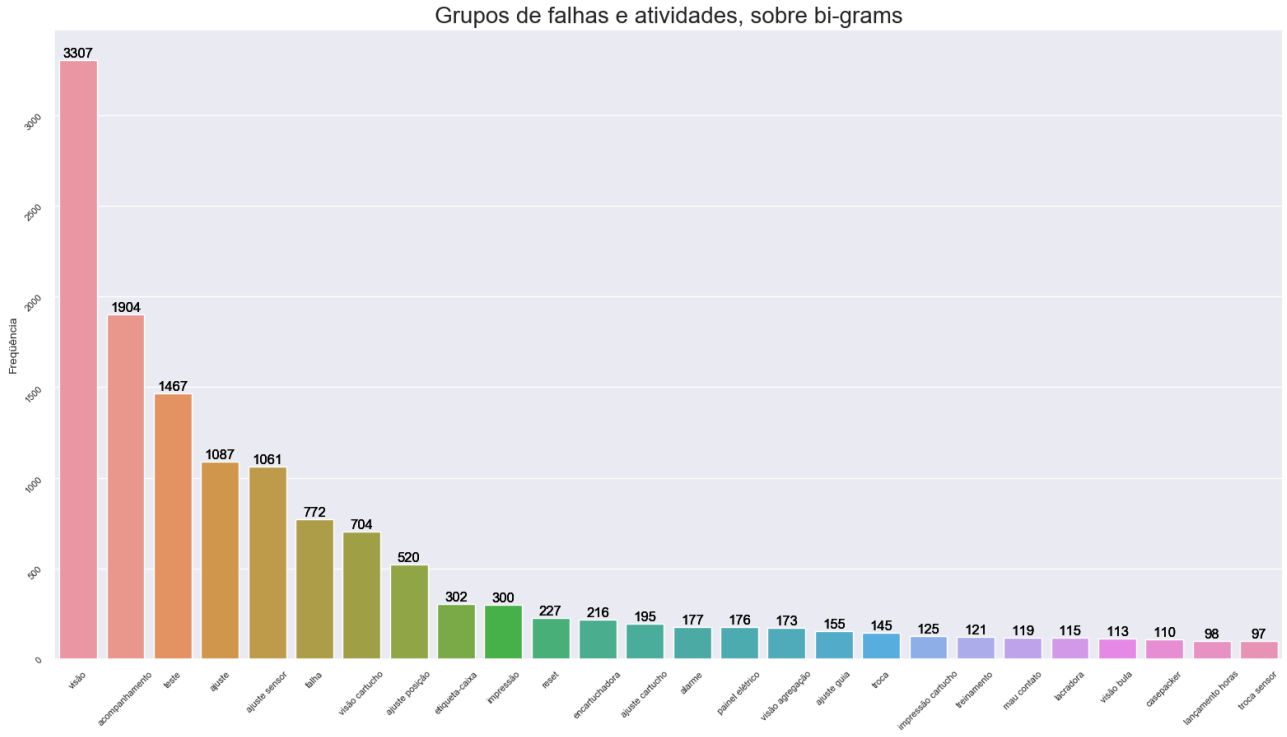
Figura 51: Gráfico final do *ranking* de *tri-grams.*



Fonte: Autor

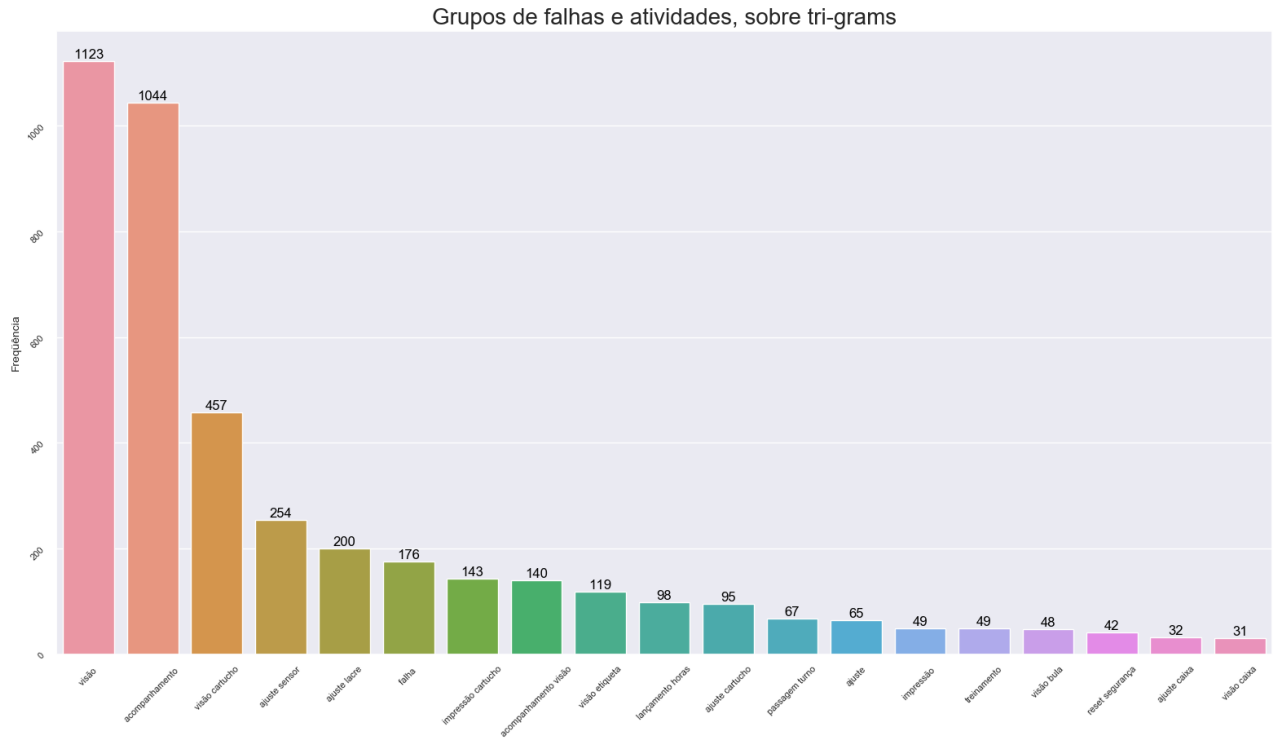
Os gráficos dos *n-grams* classificados em grupos conforme elucidado anteriormente estão apresentado nas figuras 52 e 53 respectivamente.

Figura 52: Gráfico da classificação gerada sobre os *bi-grams.*



Fonte: Autor.

Figura 53: Gráfico da classificação gerada sobre os *tri-grams.*



Fonte: Autor.

O resultado final das análises possibilita assim a a interpretação e criação de uma listagem de tópicos, apresentados na tabela 7, que podem ser avaliados e utilizados como base para planejamento de ações com o objetivo de aumento da eficácia da área de manutenção da fábrica.

Estes resultados em conjunto com demais resultados de análises sobre dados normalizados, já realizadas atualmente pelo negócio, formam uma rica base de conhecimento. A agregação das informações obtidas através da análise textual pode contribuir com a interpretação e prover uma fonte de informações mais granulares e específicas sobre onde o negócio deve intensificar seus esforços para melhoria contínua da gestão da manutenção.

Tabela 7: Tópicos finais de maior relevância, após interpretação dos dados.

Ranking proposto	Tópicos de maior relevância
1	*Ajuste em sistema de visão*
2	*Acompanhamento e testes*
3	*Ajuste em sistema de visão de cartucho*
4	*Ajuste em sensores*
5	*Ajuste na aplicação de lacre*
6	*Ajuste de posição*
7	*Ajuste de aplicação etiqueta*
8	*Ajuste em sistema de visão de etiquetas/caixa*
9	*Tempo para passagem de turno*
10	*Ajuste de guias*
11	*Ajuste de impressão de cartucho*
12	*Ajuste de sistema de visão de bula*
13	*Reset de sistemas de segurança*
14	*Tempo para lançamento de horas*
15	*Troca de sensores*

Fonte: Autor.

A conclusão evidente é que não existem quebras freqüentes de fato, e as manutenções em sua maioria são apenas ajustes e acompanhamentos, disparadamente com foco em sistemas de visão computacional e sistemas de impressão desta fábrica.

7. Links

O link abaixo disponibiliza todos os *scripts* e *datasets* utilizados neste trabalho.

https://drive.google.com/open?id=1ykzdJwlrKFzz7I8GvE6tg0ybUU1AdMza

REFERÊNCIAS

COHEN, Ted: Data Management, The Basics of CMMS, AAMI, Sacramento, CA, 2014. Disponível em: <https://www.aami-bit.org/doi/pdf/10.2345/0899-8205-48.2.117>. Acesso em: 16 fev. 2020.

GENTZKOW, Matthew & KELLY, Bryan & TADDY, Matt: Text as Data, Journal of Economic Literature, 57(3), 535–574, 2019, disponível em <https://web.stanford.edu/~gentzkow/research/text-as-data.pdf>. Acesso em: 1 mar. 2020.

KREMER, Andreas & MALZKORN, Wolfgang & STROBEL, Florian: Ratings revisited: Textual analysis for better risk management. McKinsey & Company, 2013. Disponível em: <https://www.mckinsey.com/business-functions/risk/our-insights/ratings-revisited-textual-analysis-for-better-risk-management>. Acesso em: 1 mar. 2020.

NETO, Edmon Darwich: Sistema de gestão de manutenção de ativos – Desenvolvimento de método de implantação em uma universidade pública, Universidade Federal do Rio Grande do Norte, Programa de pós-graduação em engenharia de produção, Natal, 2019. Disponível em: <https://repositorio.ufrn.br/jspui/bitstream/123456789/26979/1/Sistemagest%c3%a3o manuten%c3%a7%c3%a3o_Darwich_2019.pdf>. Acesso em: 16 fev. 2020.

PERES, Mayara Lima: Sistema de planejamento e controle de manutenção baseado nos índices de controle de processo numa empresa de telecomunicações, Universidade Federal do Amazonas, Programa de pós-graduação em engenharia de produção, Manaus, 2011. Disponível em: <https://tede.ufam.edu.br/bitstream/tede/5318/5/Disserta%c3%a7%c3%a3o%20-%20Mayara%20Lima%20Peres.pdf>. Acesso em: 16 fev. 2020.

RAJA, Uzma & MITCHELL, Tara & DAY, Timothy & HARDIN, J. Michael: Text Mining in Healthcare: Applications and Opportunities. Journal of healthcare information management (JHIM), 2008. Disponível em: <https://www.researchgate.net/publication/24182770_Text_mining_in_healthcare_Applications_and_opportunities>. Acesso em: 1 mar. 2020.

RESHAMWALA, Alpa & MISHRA, Dhirendra & PAWAR, Prajakta: Review on natural language processing, IRACST – Engineering Science and Technology: An International Journal (ESTIJ), 3, 113-116, 2013, Disponível em: < https://www.researchgate.net/publication/235788362_REVIEW_ON_NATURAL_LANGUAGE_PROCESSING >. Acesso em: 1 mar. 2020.

VIVEK, Sowmya. Automated Keyword Extraction from Articles using NLP. 2018. Disponível em: <https://medium.com/analytics-vidhya/automated-keyword-extraction-from-articles-using-nlp-bfd864f41b34>. Acesso em: 26 fev. 2020.