

**FACULDADE ANGLO-AMERICANO DE FOZ DO IGUAÇU**

**THIAGO MEDEIROS DE SOUZA**

**MINERAÇÃO DE DADOS APLICADO À REDE SOCIAL *TWITTER*  
UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON**

FOZ DO IGUAÇU

2016

THIAGO MEDEIROS DE SOUZA

**MINERAÇÃO DE DADOS APLICADO À REDE SOCIAL *TWITTER*  
UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON**

Trabalho de conclusão de curso apresentado como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação da Faculdade Anglo-American de Foz do Iguaçu.

Orientador: Prof. Msc. Valmei Abreu Júnior

Coorientador: Prof. Esp. João Paulo de Lima Barbosa

FOZ DO IGUAÇU

2016

## **TERMO DE APROVAÇÃO**

**THIAGO MEDEIROS DE SOUZA**

### **MINERAÇÃO DE DADOS APLICADO À REDE SOCIAL *TWITTER* UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON**

Trabalho de conclusão de curso apresentado como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação da Faculdade Anglo-American de Foz do Iguaçu, pela seguinte banca examinadora:

---

Prof. Msc. Valmei Abreu Júnior  
Faculdade Anglo-American  
(Orientador)

---

Prof. Banca 2  
Faculdade Anglo-Americano

---

Prof. Banca 3  
Faculdade Anglo-Americano

Foz do Iguaçu, 24 de junho de 2016

*Dedico este trabalho a meus pais,  
Valmir M. de Souza e Maria Emilia M. de Souza  
que, com muito amor, me ensinaram os valores da vida.*

## **AGRADECIMENTOS**

Primeiramente agradeço a Deus por sua graça e salvação.

À minha família, por terem me proporcionado oportunidades únicas e as melhores condições de estudo.

À Elyn Hsu, por me mostrar o caminho da disciplina e amor que me incentivaram durante esta jornada.

Aos meus grandes amigos, Daniel Gonzalez Maciel e Jann Claude Mousquer, por me acompanharem no caminho da vida profissional.

A todos os professores que fizeram parte desta importante etapa da minha vida.

Aos meus orientadores, Valmei Abreu Júnior e João Paulo de Lima Barbosa, por toda a disponibilidade e orientação.

*"Diggin deep for eternal treasure  
Stay away from quicksand and false pleasure."  
(Matthew Paul Miller - Matisyahu)*

## RESUMO

A rede social *Twitter* é utilizado para troca de mensagens imediatas, onde uma enorme quantidade de dados é gerado devido as interações de seus usuários. A grande proporcionalidade destes dados é disponibilizado pela rede social permitindo, então, a sua análise com o intuito de gerar conhecimento útil. Este trabalho apresenta técnicas e algoritmos de *data mining* para análise e mineração dos dados coletados no dia 17 de abril de 2016, em que foi realizado a votação no Congresso brasileiro para a continuidade do processo de Impeachment da presidente Dilma Rousseff e se beneficia dos recursos e bibliotecas que a linguagem de programação Python possui para a coleta e apresentação de gráficos e planilhas para a visualização e interpretação dos dados minerados.

**Palavras-chaves:** Dados. Data Mining. Twitter. Python.

## **ABSTRACT**

traduzindo...

**Keywords:** Data. Data Mining. Twitter. Python.

## LISTA DE ILUSTRAÇÕES

FIGURA 1 – Etapas do processo de KDD . . . . .	25
FIGURA 2 – Exemplo de uma <i>Series</i> . . . . .	32
FIGURA 3 – Criação de um <i>DataFrame</i> . . . . .	33
FIGURA 4 – Conteúdo de um <i>DataFrame</i> pelo interpretador <i>IPython</i> . . . . .	33
FIGURA 5 – Exemplo de um gráfico gerado pelo <i>matplotlib</i> . . . . .	34
FIGURA 6 – Exemplo de uma página web do <i>IPython Notebook</i> . . . . .	35
FIGURA 7 – Exemplo de um <i>tweet</i> . . . . .	42
FIGURA 8 – Aplicação <i>TweetDeck</i> apresentando várias <i>timelines</i> . . . . .	43
FIGURA 9 – Execução do <i>script</i> para coleta de dados . . . . .	48
FIGURA 10 – <i>Dirty Data</i> presente no arquivo coletado . . . . .	53
FIGURA 11 – Utilizando o comando <i>grep</i> para gerar um novo arquivo sem <i>dirty data</i> . . . . .	53
FIGURA 12 – Exemplo de <i>links</i> extraídos de <i>tweets</i> . . . . .	58
FIGURA 13 – <i>Word Cloud</i> das 500 palavras mais mencionadas . . . . .	69
FIGURA 14 – <i>Word Cloud</i> utilizando imagem como modelo . . . . .	70
FIGURA 15 – Distribuição geográfica de <i>tweets</i> . . . . .	71

## **LISTA DE TABELAS**

TABELA 1 – Cronograma de execução . . . . .	20
---	----

## LISTA DE CÓDIGOS

CÓDIGO 1 – Acesso à API do <i>Twitter</i> . . . . .	44
CÓDIGO 2 – <i>Script</i> coletar-hashtags.py . . . . .	47
CÓDIGO 3 – Exemplo de um <i>tweet</i> no formato JSON . . . . .	48
CÓDIGO 4 – Importação de bibliotecas Python . . . . .	54
CÓDIGO 5 – Leitura do arquivo JSON . . . . .	55
CÓDIGO 6 – Mapeamento de variáveis para um <i>DataFrame</i> . . . . .	55
CÓDIGO 7 – Contabilização do número de <i>tweets</i> por países . . . . .	56
CÓDIGO 8 – Buscar <i>hashtags</i> mais publicadas . . . . .	56
CÓDIGO 9 – Buscar informações sobre votação . . . . .	57
CÓDIGO 10 – Extração de <i>links</i> provenientes de <i>tweets</i> . . . . .	58
CÓDIGO 11 – Filtro para coletar nomes com mais <i>tweets</i> . . . . .	59
CÓDIGO 12 – Extração de outras entidades dos <i>tweets</i> . . . . .	60
CÓDIGO 13 – Filtro de usuários através de categorias de entidades . . . . .	61
CÓDIGO 14 – Implementação da <i>Word Cloud</i> . . . . .	61
CÓDIGO 15 – Implementação de coordenadas dos <i>tweets</i> . . . . .	62
CÓDIGO 16 – Data da publicação dos <i>tweets</i> . . . . .	63

## **LISTA DE GRÁFICOS**

GRÁFICO 1 – Idiomas que mais realizaram <i>tweets</i> . . . . .	64
GRÁFICO 2 – Países que mais realizaram <i>tweets</i> . . . . .	65
GRÁFICO 3 – Número de menções de SIM e NÃO em <i>tweets</i> . . . . .	66
GRÁFICO 4 – <i>Hashtags</i> com o maior número de <i>tweets</i> . . . . .	67
GRÁFICO 5 – Gráfico em setores para figuras importantes . . . . .	68
GRÁFICO 6 – Publicação de <i>tweets</i> por hora . . . . .	71

## LISTA DE ABREVIATURAS

API	<i>Application Programming Interface</i> - Interface de Programação de Aplicação
BMP	<i>Windows Bitmap</i>
CGI	<i>Common Gateway Interface</i> - Interface Comum de Entrada <sup>1</sup>
CSV	<i>Comma-Separated Values</i> - Valores Separados Por Vírgula <sup>1</sup>
DBA	<i>Database Administrator</i> - Administrador de Banco de Dados
FTP	<i>File Transfer Protocol</i> - Protocolo de Transferência de Arquivos
GIF	<i>Graphics Interchange Format</i> - Formato Para Intercâmbio de Gráficos <sup>1</sup>
GUI	<i>Graphical User Interface</i> - Interface Gráfica do Usuário
HTTP	<i>Hypertext Transfer Protocol</i> - Protocolo de Transferência de Hipertexto
HTTPS	<i>Hyper Text Transfer Protocol Secure</i> - Protocolo de Transferência de Hipertexto Seguro
IETF	<i>Internet Engineering Task Force</i>
IMAP	<i>Internet Message Access Protocol</i> - Protocolo de Acesso a Mensagem da Internet
IP	<i>Internet Protocol</i> - Protocolo de Internet
JPG	<i>Joint Photographic Experts Group</i>
JSON	<i>JavaScript Object Notation</i> - Notação de Objeto JavaScript <sup>1</sup>
KDD	<i>Knowledge Discovery From Data</i> - Descoberta de Conhecimento por Dados
NLP	<i>Natural Language Processing</i> - Processamento de Linguagem Natural
NLTK	<i>Natural Language Toolkit</i> - Ferramentas de Linguagem Natural <sup>1</sup>
PDF	<i>Portable Document Format</i> - Formato de Documento Portátil <sup>1</sup>

<sup>1</sup> Tradução do autor

PNG	<i>Portable Network Graphics</i> - Rede Portável de Gráficos <sup>1</sup>
POP	<i>Post Office Protocol</i> - Protocolo dos Correios
RFC	<i>Request for Comments</i> - Pedido Para Comentários
RPC	<i>Remote Procedure Call</i> - Chamada Remota de Procedimento <sup>1</sup>
RUP	<i>Rational Unified Process</i> - Processo Unificado da Rational
SMTP	<i>Simple Mail Transfer Protocol</i> - Protocolo de Transferência de Correio Simples
SSL	<i>Secure Sockets Layer</i> - Camada Segura de Sockets
SVG	<i>Scalable Vector Graphics</i> - Gráficos Vetoriais Escaláveis
TCP	<i>Transmission Control Protocol</i> - Protocolo de Controle de Transmissão
URI	<i>Uniform Resource Identifier</i> - Identificador Uniforme de Recursos
URL	<i>Uniform Resource Locator</i> - Localizador Padrão de Recursos
UTC	<i>Universal Time Coordinated</i> - Tempo Universal Coordenado
XHTML	<i>eXtensible Hypertext Markup Language</i> - Linguagem de Marcação de Hipertexto Extensiva
XML	<i>eXtensible Markup Language</i> - Linguagem de Marcação Extensiva
YML	<i>Yet Another Markup Language</i> - Uma Outra Linguagem de Marcação <sup>1</sup>

---

<sup>1</sup> Tradução do autor

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>16</b>
1.1	JUSTIFICATIVA	18
1.2	OBJETIVOS	19
1.2.1	Objetivo Geral	19
1.2.2	Objetivos Específicos	19
1.3	CRONOGRAMA DE ATIVIDADES	20
1.4	ORGANIZAÇÃO DO TRABALHO	20
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA</b>	<b>22</b>
<b>3</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>24</b>
3.1	DESCOBERTA DE CONHECIMENTO EM BASE DE DADOS E <i>DATA MINING</i>	24
3.2	LINGUAGEM PYTHON	28
<b>4</b>	<b>MATERIAIS E MÉTODOS</b>	<b>30</b>
4.1	TECNOLOGIAS E FERRAMENTAS	30
4.1.1	Bibliotecas da Linguagem Python	30
4.1.1.1	Biblioteca <i>NumPy</i>	30
4.1.1.2	Biblioteca <i>pandas</i>	32
4.1.1.3	Biblioteca <i>matplotlib</i>	33
4.1.1.4	Biblioteca <i>SciPy</i>	34
4.1.1.5	Interpretador <i>IPython</i>	35
4.1.1.6	Biblioteca <i>Folium</i>	36
4.1.1.7	Biblioteca <i>NLTK</i>	36
4.1.1.8	Biblioteca <i>Word Cloud</i>	36
4.1.2	Interface de Programação de Aplicações - API	37
4.1.2.1	Arquitetura REST	37
4.1.3	Protocolo de Autenticação - OAuth	38
4.1.3.1	Protocolo OAuth 1.0a	38
4.1.3.2	Protocolo OAuth 2.0	39
4.1.4	Rede Social <i>Twitter</i>	40
4.1.4.1	API do <i>Twitter</i>	40
4.1.4.2	Bibliotecas Para o Consumo de Dados da API do <i>Twitter</i>	44
4.2	METODOLOGIA E DESENVOLVIMENTO	45

<b>5</b>	<b>IMPLEMENTAÇÃO DAS TÉCNICAS . . . . .</b>	<b>47</b>
5.1	COLETA DE DADOS . . . . .	47
5.2	ANÁLISE DE DADOS . . . . .	52
<b>6</b>	<b>ANÁLISE DOS RESULTADOS . . . . .</b>	<b>64</b>
6.1	APRESENTAÇÃO DOS RESULTADOS . . . . .	64
<b>7</b>	<b>CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS . .</b>	<b>72</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>73</b>

## 1 INTRODUÇÃO

Redes sociais se tornaram um termo comum e uma chave fundamental para o estilo de vida moderno. Hoje em dia, a maioria das pessoas, independente de idade, sexo, crença, utilizam uma ou mais redes sociais. A princípio, esses ambientes *online* focavam-se na comunicação, por exemplo; a possibilidade de se comunicar com alguém distante e tornar esse diálogo pessoal, seguro e, de alguma forma, próximo, ajudando na popularização desse tipo de tecnologia. No decorrer dos anos e com o avanço tecnológico, diferentes tipos de redes sociais surgiram com ideias semelhantes ou extremamente diferentes, não sendo apenas para a comunicação, mas para outros fins como o compartilhamento de mídias, localização, críticas, *mini-blogs*, perguntas e respostas, negócios, profissão, música, artes, venda e troca de produtos, entre outros.

*Facebook*, *Twitter*, *LinkedIn*, *Google+* e, muito comum entre desenvolvedores, o *GitHub* são exemplos populares de redes sociais. Logo, possuem grande número de usuários e diversas interações que estes realizam a cada momento, gerando uma quantidade gigantesca de dados. Devido a diversidade e a vasta quantidade de informação gerada, algumas redes sociais podem utilizá-las para o aprimoramento de conteúdo ou, então, para o comércio de dados para empresas, por exemplo; publicidade e marketing, que fazem a mineração desses dados para encontrar padrões de seus usuários e, assim, conseguir aumentar suas vendas, reduzir riscos e, até mesmo, gerar novas tendências.

Nesse sentido, dados é um termo deliberadamente vago, que agrupa várias formas comuns de informações, como matrizes (vetores multidimensionais), planilhas ou tabelas, em que cada coluna pode ter um tipo diferente de informação: caracteres, numéricos, data, entre outros. As tabelas, por sua vez, também podem se relacionar através de colunas apresentadas no tradicional modelo de bancos de dados relacionais. Todavia, não é sempre que um percentual de um conjunto de dados pode ser transformado em uma forma estruturada, em que é possível ser analisado e modelado. Dessa forma, cabe aos cientistas de dados visualizá-los com o objetivo de produzir resultados claros e serem capazes de informar ao seus mantenedores sobre a situação atual e a qualquer momento. Este é o verdadeiro valor que um cientista nessa área precisa prover.

A mineração de dados, também conhecida como *data mining*, é o processo de analisar dados em diferentes perspectivas e transformá-los em informação útil. Hoje em dia, o *data mining* é usado por companhias com grande foco em varejo, finanças, comunicação e marketing, para conseguirem determinar as relações de fatores internos como preço, posição de produto, ou habilidade de recurso humano, e fato-

res externos como indicadores econômicos, competições e população demográfica de clientes (RUSSELL, 2013).

Essa análise de dados consiste em visualizar informações de diferentes maneiras e formas, plotando gráficos e planilhas. Com isso, novas informações aparecerão permitindo alguma previsão ou predição desse conteúdo. As observações levarão a uma reflexão que resultará em possibilidades ou probabilidades concretas para se exercer uma atividade. No primeiro momento, essas informações são amorfas e, após a análise, se transformarão em ideias (HAN et al., 2012).

Para que essas ideias se tornem um trabalho futuro é preciso capturá-las e interpretá-las através de um modelo de extração de conhecimento. Esse modelo, geralmente, é um processo que apresenta etapas que vão desde o armazenamento dos dados em estudos, até a aplicação de processos matemáticos, estatísticos e computacionais, com o objetivo de extrair informações úteis. Um modelo então, é muito mais que apenas a descrição dos dados, incorpora o entendimento de todo o processo da origem dos dados até a competência deles. Logo, ele consegue fazer previsões sobre os conhecimentos analisados (HAN et al., 2012).

Para conseguir fazer melhores previsões é preciso desenvolver métodos mais sofisticados antes de formular um modelo relevante. Com isso, a dificuldade aumenta e, então, é necessário implementar um modelo computacional que consiga obter possíveis resultados através do reconhecimento desses dados.

Para a análise e a interação de dados, computação exploratória e visualização de dados, a linguagem de programação Python vai, inevitavelmente, ser comparada a muitas outras, tanto no domínio de software livre, como também, com linguagens e ferramentas comerciais, como R, MATLAB, SAS, Stata e outros. Atualmente, o Python possui bibliotecas que se tornaram fortes alternativas para a tarefa de manipulação de dados. Combinado com o poder de programação que a linguagem tem, é uma excelente escolha como linguagem para a construção de aplicações centradas em dados (MCKINNEY, 2013).

Em muitas organizações é comum realizar pesquisas, prototipar e testar novas ideias utilizando mais de um domínio específico de linguagem computacional, como MATLAB ou R e, posteriormente, fazer com que estas ideias virem parte de um sistema de produção maior, escrito, por exemplo, em Java, C#, ou C++. Kaldero (2015), afirma que Python não é somente uma linguagem adequada para a pesquisa e protótipagem, mas também para o desenvolvimento de sistemas.

Devido a esta solução de apenas uma única linguagem, as organizações podem se beneficiar, tendo cientistas e tecnólogos usando o mesmo conjunto de ferramentas programáticas. Portanto, Python é a ferramenta escolhida pela maioria desses profissionais. Essa escolha se deve, não somente a alta produtividade que a lin-

guagem fornece, mas também por ela ser uma ferramenta comum a diferentes times e organizações (KALDERO, 2015).

Python é uma linguagem de programação livre e multiplataforma, possui uma excelente documentação e está sobre os cuidados de uma enorme comunidade, onde é possível obter ajuda e melhores soluções para problemas durante a codificação. Tem como grande vantagem a facilidade de aprendizado, porque foi desenvolvida para ser simples e descomplicada. É uma linguagem interpretada, dinamicamente tipada, com grande precisão e sintaxe eficiente. Tem grande popularidade para analisar dados, devido ao enorme poder que suas bibliotecas possuem (*NumPy*, *SciPy*, *pandas*, *matplotlib*, *IPython*). Apresenta alta produtividade para prototipação, desenvolvimento de sistemas menores e reaproveitáveis.

A mineração de dados busca então, extrair dos dados o conhecimento útil para algum objetivo específico. Entretanto, a tarefa de extração de conhecimento é complexa devido a multidisciplinaridade envolvida no seu processo de extração e, também, por não ter um modelo de mineração genérico para a busca de informação útil. Para isso, é necessário o uso de ferramentas que viabilizam essas tarefas. Logo, Python dispõe de um conjunto de bibliotecas para a análise e mineração de dados extremamente poderosas e com uma curva de aprendizado curta, graças a sintaxe clara e descomplicada que a linguagem fornece.

## 1.1 JUSTIFICATIVA

A rede social *Twitter* é um excelente ponto de partida para a mineração de dados em redes sociais, devido a sua abertura ao público para consulta de informações, por ser uma API bem documentada e por possuir uma vasta quantidade de dados resultante da interação dos usuários a todo instante. Os dados do *Twitter* são particularmente interessante, porque *tweets*, frases postadas nesta rede social, acontecem, segundo Russell (2013), na "velocidade do pensamento" e logo estão disponíveis na Internet.

Esta disponibilidade de dados em tempo real permite que os usuários possam comentar e interagir com outras pessoas sobre acontecimentos atuais e de uma forma instantânea. Logo, é possível acompanhar o *Twitter* com a finalidade de obter informações sobre qualquer tipo de notícia que esteja acontecendo no mundo, desde que estes fatos sejam publicados na rede social.

Atualmente, o Brasil está passando por alguns acontecimentos importantes que, consequentemente, tem gerado muitos *tweets* devido à sua relevância. Em virtude de vários episódios de crimes fiscais, corrupção, lavagem de dinheiro, a falta de atenção em alguns setores públicos, e como resultado, diversas manifestações populares, um processo de Impeachment da presidente Dilma Rousseff foi iniciado.

Muitos brasileiros têm se manifestado em diversas redes sociais, e o *Twitter* é uma ótima escolha, graças a sua velocidade de comunicação e propagação de notícias. Os *tweets* sobre a atual situação política, apresentam comentários de ambos os lados do processo de Impeachment, mostrando aqueles que são a favor e, também, os que são contrários a esta decisão.

É de interesse então, a mineração destes dados, já que a rede social *Twitter* dispõe de uma vasta quantidade de informações. A cada segundo, 9.100 *tweets* são publicados, e atingem um bilhão a cada 5 dias. A rede social possui um total de 289 milhões de usuários ativos no mundo inteiro, totalizando 58 milhões de *tweets* por dia (BRAIN, 2016).

Nesse sentido, a atual conjuntura do Brasil mostra-se propícia à mineração da base de dados do *Twitter*, visto que é possível coletar e analisar *tweets* a fim de buscar informações e compreender as preferências e ideias que a população brasileira possui a respeito dos acontecimentos políticos do país. A linguagem de programação Python, por sua vez, possibilita a coleta e análise destes dados através de bibliotecas específicas para manipulação e consumo de informações.

## 1.2 OBJETIVOS

### 1.2.1 Objetivo Geral

Este trabalho tem como objetivo principal utilizar técnicas e algoritmos de *data mining*, para a análise e mineração de dados provenientes da rede social *Twitter*, utilizando os recursos e bibliotecas que a linguagem de programação Python possui.

### 1.2.2 Objetivos Específicos

- Identificar os conceitos sobre KDD e *data mining*;
- Descrever as técnicas de *data mining*;
- Explorar as funcionalidades das bibliotecas de mineração e visualização da linguagem Python;
- Examinar e utilizar a API da rede social *Twitter* para a coleta de dados;
- Encontrar padrões em dados provenientes do *Twitter*;
- Compreender e aplicar técnicas para apresentação e visualização de informações geográficas encontradas nos dados coletados;
- Apresentar testes e resultados obtidos da análise e mineração dos dados.

### 1.3 CRONOGRAMA DE ATIVIDADES

As atividades a serem executadas no decorrer do projeto visando o êxito do mesmo, estão listados a seguir e especificados em meses na Tabela 1:

- Estudo e Pesquisa: aquisição dos conhecimentos pertinentes e necessários para o desenvolvimento do projeto;
- Análise de Requisitos: levantamento dos requisitos do projeto;
- Geração do Documento: desenvolvimento das documentações para especificação do projeto;
- Implementação: desenvolvimento dos códigos para a análise de dados;
- Testes: execução dos testes que irão garantir a qualidade das informações a serem geradas;
- Elaboração de Artigos: parte do tempo destinado ao projeto será para desenvolver artigos visando a publicação em eventos da área;
- Apresentação de Resultados: etapas destinadas à apresentação dos resultados parciais e finais.

TABELA 1: Cronograma de execução

Mês - Ano	08/15	09/15	10/15	11/15	12/15	02/16	03/16	04/16	05/16
Estudo e Pesquisa	X	X	X	X	X	X	X	X	
Análise de Requisitos	X	X	X	X	X	X	X	X	
Geração do Documento	X	X	X	X	X	X	X	X	X
Implementação				X	X	X	X	X	X
Testes				X	X	X	X	X	X
Elaboração de Artigos			X	X	X			X	X
Apresentação de Resultados					X				X

FONTE: Elaborado pelo autor

### 1.4 ORGANIZAÇÃO DO TRABALHO

Além deste capítulo introdutório, este trabalho é composto de mais seis capítulos.

O Capítulo 2 apresenta os trabalhos que são referências para este estudo.

Os fundamentos teóricos, como os conceitos de *data mining* e base para o entendimento do tema proposto, estão descritos no Capítulo 3.

As bibliotecas da linguagem Python utilizadas para a mineração de dados são expostas no Capítulo 4. Também, são apresentadas neste capítulo a API do *Twitter*, as etapas de *data mining* e outros materiais e metodologias utilizados para execução deste trabalho.

No Capítulo 5 são demonstradas as fases do desenvolvimento de *scripts* para a coleta e obtenção de resultados, assim como a utilização de técnicas de *machine learning*.

Os resultados obtidos e a apresentação de planilhas e gráficos das soluções desenvolvidas são apresentados no Capítulo 6.

Por fim, a conclusão deste trabalho se dá no Capítulo 7, onde são abordadas e analisadas as dificuldades, além de determinar as possibilidades para trabalhos futuros.

## 2 REVISÃO BIBLIOGRÁFICA

Alguns trabalhos serviram como ajuda e inspiração para este estudo. Porém durante o período de busca por bibliografias a respeito de *data mining*, pouco material foi encontrado quanto a mineração em redes sociais. Ainda sim, alguns estudos possuem um destaque e é necessário citá-los, devido a utilização de ferramentas específicas em *data mining* e, também, aos resultados obtidos neste processo.

De acordo com Lemos (2003), um dado se transforma em informação quando ganha um significado para seu utilizador, caso contrário, continua sendo simplesmente um dado.

Em seu estudo, Lemos (2003) aborda duas técnicas de *data mining*: Árvores de Decisão e Redes Neurais, para realizar a análise de crédito bancário. Estas técnicas permitem fazer o reconhecimento de padrões e também diagnosticar novos casos. Através deste modo, os analistas têm condições de diagnosticar os novos clientes, quanto ao merecimento de crédito ou não. Estas técnicas de mineração de dados são ferramentas que podem ser utilizadas pelo especialista para auxiliá-lo nas tomadas de decisão, nunca porém poderão por si só, substituir a figura do especialista no contexto da análise de crédito.

Semelhante as técnicas abordadas para a análise de crédito bancário, Steiner et al. (2004) realizaram um estudo na área médica, onde a posse e uso de ferramentas que auxiliem na tarefa de classificação de pacientes em prováveis ictéricos com câncer ou ictéricos com cálculo, pode ser crucial. Em uma tentativa de otimizar todo o processo do diagnóstico, minimizando riscos e custos e, por outro lado, maximizando a eficácia nos resultados, utilizaram técnicas de *data mining* como um processo para extração de informações valiosas. O trabalho consistiu na análise de 118 históricos de pacientes utilizando Árvores de Decisão e Regras de Classificação como ferramenta. Os autores afirmam que os métodos de *data mining* apresentam a vantagem de deixar claro ao usuário quais são os atributos que estão discriminando os padrões e de que forma a mesma está ocorrendo, isso é uma característica altamente desejável em qualquer técnica de reconhecimento de padrões.

O reconhecimento de padrões permite a obtenção de conhecimento da frequência com que determinadas seções de uma página *web* são acessadas e quais são os serviços mais procurados. Isso possibilita que empresas consigam descobrir o perfil de seus usuários e, com base nesse acontecimento, ofertar serviços e atendimento personalizado. Essa afirmação foi resultado do estudo de Silva, Boscaroli e Peres (2003), onde realizaram a mineração de dados em *logs* de acesso a servidores *web*. Para o desenvolvimento do trabalho utilizaram regras de associação para a desco-

bera e representação de padrões frequentes em conjuntos de dados. Isso propiciou a identificação de padrões de comportamento de usuários da Internet ao navegarem por *websites*. Também concluem que outras ferramentas de mineração podem ser aplicadas, visando aumentar a flexibilidade de manipulação dos atributos específicos para o ambiente *Web*.

Como visto, o tema sobre *data mining* possui uma abordagem comum a setores e áreas diferentes, o que caracteriza a enorme quantidade de dados que ainda não se tem informação útil para seus utilizadores. Por mais que as técnicas utilizadas dentre essa diversidade de áreas sejam semelhantes, o desenvolvimento e aplicação das técnicas são particulares para cada caso. Neste contexto, esta dissertação modela, de uma forma mais simplificada, o reconhecimento de padrões para a predição dos dados extraídos da rede social *Twitter*.

### 3 FUNDAMENTAÇÃO TEÓRICA

A mineração de dados é um assunto totalmente interdisciplinar, podendo ser definido de diversas maneiras. Até mesmo o termo *data mining* não representa realmente todos os componentes desta área. Han et al. (2012) exemplificam esta questão comentando sobre a mineração de ouro através da extração de rocha e areia, que é chamado de mineração de ouro e não mineração de rochas ou mineração de areia. Analogamente, a mineração de dados deveria se chamar "mineração de conhecimento através de dados" que, infelizmente, é um termo um tanto longo. Entretanto, uma referência mais curta, como "mineração de conhecimento", pode não enfatizar a mineração de uma enorme quantidade de dados. Apesar disso, a mineração é um termo que caracteriza o processo de encontrar uma pequena quantia de uma preciosa pepita em uma grande quantidade de matéria bruta. Nesse sentido, um termo impróprio contendo ambos "*data*" e "*mining*" se tornou popular e, como consequência, muitos outros nomes similares surgiram: *knowledge mining from data*, *knowledge extraction*, *data/pattern analysis*, *data archaeology* e *data dredging*.

Na seção 3.1 será abordado os conceitos de descoberta de conhecimento em base de dados e a sua diferença em relação ao *data mining*. Logo após a explicação desta distinção, será apresentado os métodos e a concepção de *data mining*. A seção 3.2 irá caracterizar a linguagem de programação Python e qual a sua vantagem em utilizá-la para a mineração de dados.

#### 3.1 DESCOBERTA DE CONHECIMENTO EM BASE DE DADOS E *DATA MINING*

Muitas pessoas tratam a mineração de dados como um sinônimo para outro termo muito popular, descoberta de conhecimento em base de dados (*knowledge discovery from data*) - KDD, enquanto outros referenciam *data mining* como apenas uma etapa no processo de descoberta de conhecimento em base de dados. O processo de KDD é demonstrado através da Figura 1 e, posteriormente, listada como uma sequência interativa e iterativa dos seguintes passos:

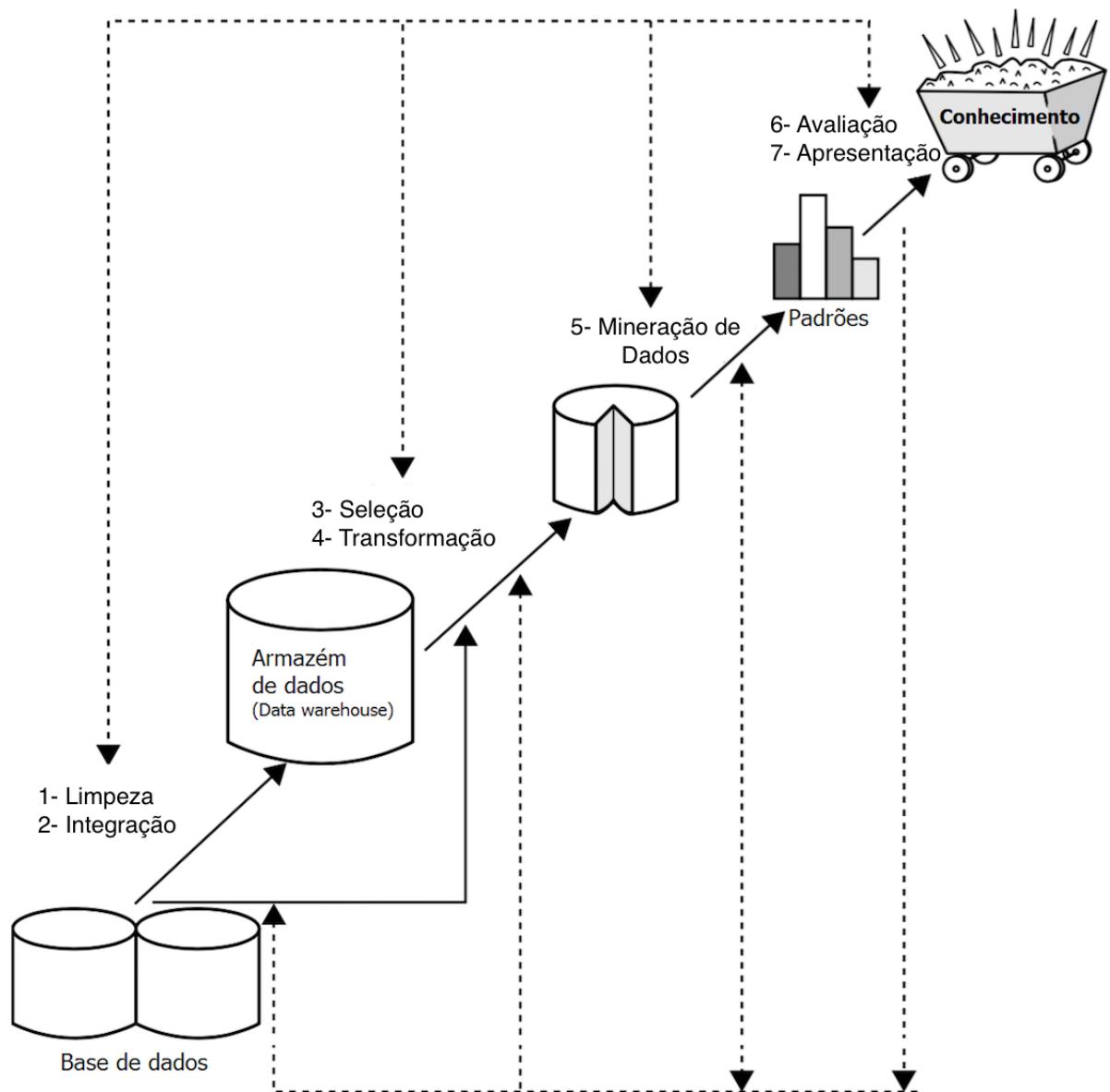


FIGURA 1: Etapas do processo de KDD

FONTE: Adaptado de Han et al. (2012)

1. *Data cleaning* (Limpeza de dados);
2. *Data integration* (Integração de dados);
3. *Data selection* (Seleção de dados);
4. *Data transformation* (Transformação de dados);
5. *Data mining* (Mineração de dados);
6. *Pattern evaluation* (Avaliação de padrões);
7. *Knowledge presentation* (Apresentação de conhecimento).

É importante notar que algum dos processos acontecem na mesma etapa: Limpeza e integração; Seleção e transformação; Avaliação e apresentação.

De acordo com Brachman et al. (1996 apud FAYYAD et al., 1996-b), as etapas são interativas porque envolvem a cooperação da pessoa responsável pela análise de dados, cujo conhecimento sobre o domínio orientará a execução do processo. Por sua vez, a interação deve-se ao fato de que, com frequência, esse processo não é executado de forma sequencial, mas envolve repetidas seleções de parâmetros e conjuntos de dados, aplicações das técnicas de *data mining* e posterior análise dos resultados obtidos, a fim de refinar os conhecimentos extraídos.

KDD refere-se ao processo global de descobrimento de conhecimento útil em bases de dados. *Data mining* é um passo particular neste processo de aplicação de algoritmos específicos para extrair padrões (modelos) de dados. Os passos adicionais no processo KDD, como integração de dados, limpeza, seleção e transformação dos dados, assim como a interpretação e a apresentação dos resultados, asseguram que o conhecimento útil (informação) foi descoberto provenientes da etapa de mineração (HAN et al., 2012). A aplicação cega de métodos de *data mining*, conforme alertado por Navega (2002), pode ser uma atividade perigosa que conduz a descoberta de padrões sem sentido.

O KDD evoluiu e continua evoluindo da interseção de pesquisas em campos como bancos de dados, aprendizado de máquinas (*machine learning*), reconhecimento de padrões, estatísticas, inteligência artificial, aquisição de conhecimento para sistemas especialistas, visualização de dados, descoberta científica, recuperação de informação e computação de alto-desempenho. Aplicações de KDD incorporam teorias, algoritmos e métodos de todos estes campos (LEMOS, 2003).

Apesar do conceito de *data mining*, na maioria das vezes, ser utilizado pelas indústrias, mídias e centros de pesquisa para se referir ao processo de descoberta de conhecimento considerado em sua globalidade, o termo *data mining* pode ser usado também para indicar o quinto estágio do KDD, sendo um processo essencial na descoberta e extração de padrões de dados. Han et al. (2012), adotam uma visão mais abrangente para a funcionalidade de mineração de dados: *data mining* é o processo de descoberta de padrões interessantes e conhecimentos de um vasto conjunto de dados. A fonte dos dados pode ser banco de dados, *data warehouses*, a Internet, outros repositórios de informações, ou dados correntes em sistemas dinâmicos.

Uma das definições, talvez, mais importante de *data mining* foi elaborada por Fayyad et al. (1996-a) "...o processo não-trivial de identificar, em dados, padrões válidos, novos, potencialmente úteis e ultimamente comprehensíveis".

*Data mining* ou mineração de dados, pode ser entendido então, como o processo de extrair informação, ou conhecimento útil, de algum conjunto de dados e uti-

lizar este conhecimento adquirido para tomada de decisões ou descrever características e padrões descobertos (SFERRA; CORREA, 2003).

Diversos métodos são usados em *data mining* para encontrar respostas ou extrair conhecimento interessante. Esses podem ser obtidos através dos seguintes métodos:

- Classificação: associa ou classifica um item a uma ou várias classes. Os objetivos dessa técnica envolvem a descrição gráfica ou algébrica das características diferenciais das observações de várias populações. A ideia principal é derivar uma regra que possa ser usada para classificar, de forma otimizada, uma nova observação a uma classe já rotulada;
- Modelos de Relacionamento entre Variáveis: associa um item a uma ou mais variáveis de predição de valores reais, conhecidas como variáveis independentes ou exploratórias. Nesta etapa se destacam algumas técnicas estatísticas como regressão linear simples, múltipla e modelos lineares por transformações, com o objetivo de verificar o relacionamento funcional entre duas variáveis quantitativas, ou seja, constatar se há uma relação funcional entre X e Y;
- Análise de Agrupamento (*Cluster*): associa um item a uma ou várias classes (ou *clusters*). Os *clusters* são definidos por meio do agrupamento de dados baseados em modelos probabilísticos ou medidas de similaridade. Analisar *clusters* é uma técnica com o objetivo de detectar a existência de diferentes grupos dentro de um determinado conjunto de dados e, caso exista, determinar quais são eles;
- Sumarização: determina uma descrição compacta para um determinado subconjunto, por exemplos; medidas de posição e variabilidade. Nesta etapa se aplica algumas funções mais sofisticadas envolvendo técnicas de visualização e a determinação de relações funcionais entre variáveis. Estas funções são usadas para a geração automatizada de relatórios, sendo responsáveis pela descrição compacta de um conjunto de dados;
- Modelo de Dependência: descreve dependências significativas entre variáveis. Estes modelos existem em dois níveis: estruturado e quantitativo. O nível estruturado demonstra, através de gráficos, quais variáveis são localmente dependentes. O nível quantitativo especifica o grau de dependência utilizando alguma escala numérica;
- Regras de Associação: determinam relações entre campos de um banco de dados. Esta relação é a derivação de correlações multivariadas que permitam auxiliar as tomadas de decisão. Medidas estatísticas, como correlação e testes de

hipóteses apropriados, revelam a frequência de uma regra no universo dos dados minerados;

- Análise de Séries Temporais: determina características sequênciais, como dados com dependência no tempo. Tem como objetivo modelar o estado do processo extraindo e registrando desvios e tendências no tempo. As séries são compostas por quatro padrões: tendência, variações cíclicas, variações sazonais e variações irregulares. Existem vários modelos estatísticos que podem ser aplicados a essas situações.

A maioria destes métodos são baseados em técnicas de aprendizado de máquina (*machine learning*), reconhecimento de padrões e estatística. Essas técnicas vão desde estatística multivariada, como análise de agrupamentos e regressões, até modelos mais atuais de aprendizagem, como redes neurais, lógica difusa e algoritmos genéticos (SFERRA; CORREA, 2003).

Devido aos vários métodos estatísticos que são aplicados no processo de *data mining*, Fayyad et al. (1996-a) mostram uma relevância da estatística para o processo de extração de conhecimentos ao afirmar que essa ciência provê uma linguagem e uma estrutura para quantificar a incerteza resultante quando se tenta deduzir padrões de uma amostra a partir de uma população.

### 3.2 LINGUAGEM PYTHON

Python é uma linguagem de programação orientada a objetos, interpretada e interativa. Incorpora módulos, exceções e de tipagem dinâmica alta. Possui uma sintaxe clara e simples, o que facilita o aprendizado para novos desenvolvedores, assim como a rápida leitura e interpretação para usuários mais experientes. Dispõe de interfaces para várias chamadas de sistemas (*system calls*) e bibliotecas, também para vários sistemas de janelas, e é extensível a outras linguagens de programação como C ou C++. É também usada como uma linguagem de extensão para aplicações que precisam de uma interface programática (PYTHON, 2015).

Outra característica da linguagem Python é a portabilidade, podendo ser utilizada em diversos sistemas operacionais como variantes do Unix, em sistemas Mac e também em PCs sob MS-DOS, Windows, Windows-NT, e OS/2.

É uma linguagem de programação de alto-nível que pode ser aplicada em soluções para diversas classes diferentes. Possui uma vasta quantidade de bibliotecas que atende a áreas como o processamento de *strings* (expressões regulares, Unicode, cálculo de diferença entre arquivos), protocolos de Internet (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI *programming*), engenharia de software (testes unitários,

registro de logs, *profiling*, análise de código Python), e interfaces para sistemas operacionais (*system calls*, sistemas de arquivos, TCP/IP sockets) (PYTHON, 2015).

A sintaxe bastante expressiva e a abundância de suas bibliotecas tornam Python uma ótima linguagem para se obter resultados em várias questões. Algumas de suas utilidades são apresentadas conforme a seguinte lista:

- Escrita de *scripts*: Python é uma ótima linguagem para a criação de *scripts*. É possível usar *scripts* para analisar arquivos de texto, gerar amostra de entradas para testar programas, coletar conteúdos de páginas *web* utilizando a biblioteca *Beautiful Soup*, dentre outras atividades;
- Desenvolvimento *backend* para aplicações *web*: É possível criar APIs (*Application Programming Interface*, apresentado no capítulo 4) e interagir com banco de dados. *Frameworks* mais utilizados inclui *Django*, *Flask* e *Pyramid*;
- Análise e visualização de dados: Conforme o foco deste trabalho, bibliotecas como *pandas*, *NumPy* e recursos semelhantes a outras ferramentas como R e MATLAB estão dispostas através da biblioteca *SciPy*;
- *matplotlib* e *Seaborn*: são mecanismos que possibilitam a visualização dos dados.

A utilização dessa linguagem como ferramenta principal para este trabalho se justifica na utilização dos pacotes que facilitam a análise e interpretação de dados. Como exemplo, *NumPy*, *SciPy*, *pandas*, *matplotlib* e *IPython* são os mecanismos indispensáveis para a mineração de informações juntamente com as estruturas de dados já presentes em Python como os *dictionaries* (estruturas similares ao JSON), que permitem ordenar dados através de um modelo chave-valor. Devido então a sintaxe intuitiva que a linguagem possui e seu excelente ecossistema de bibliotecas, é possível acessar APIs e manipular dados com mais facilidade (RUSSELL, 2013).

## 4 MATERIAIS E MÉTODOS

Após a revisão bibliográfica de outros estudos e os fundamentos teóricos necessários para a mineração de dados utilizando Python, torna-se importante definir as ferramentas, tecnologias e procedimentos necessários para o desenvolvimento do projeto.

Este capítulo apresenta os materiais e métodos utilizados para a realização do processo de *data mining*, onde, na seção 4.1 são apresentadas as tecnologias e ferramentas que serão utilizadas durante o estudo. Serão abordados quais as bibliotecas que a linguagem Python disponibiliza para a análise e mineração de dados e, também, como acessar a API do *Twitter*. Esta será esclarecida também neste capítulo, após a explicação do conceito de API e o protocolo OAuth.

A seção 4.2 irá concluir o capítulo apresentando as etapas de *data mining* com o intuito de evidenciar o processo para a obtenção de conhecimento útil.

### 4.1 TECNOLOGIAS E FERRAMENTAS

Tecnologias e ferramentas para a implementação de *scripts* e utilização dos algoritmos.

#### 4.1.1 Bibliotecas da Linguagem Python

Um dos grandes diferenciais da linguagem Python é o seu enorme conjunto de bibliotecas para soluções de diversos problemas.

A seguir serão apresentadas as bibliotecas necessárias para a mineração de dados, através das quais é possível coletar, limpar, transformar, realizar operações e apresentar resultados proveniente dos dados da rede social *Twitter*. Para evitar repetições da palavra "biblioteca", o termo "pacote" também será utilizado com o mesmo significado no restante desta dissertação.

##### 4.1.1.1 Biblioteca *NumPy*

*NumPy* é o pacote fundamental para computação científica em Python. É o acrônico para *Numerical Python*. Esta biblioteca provê:

- *ndarray* que é um objeto de matriz multidimensional;
- Funções que permitem realizar operações vetoriais ou operações matemáticas entre matrizes sem a necessidade de programar *loops*;

- Ferramentas para a leitura e escrita em conjuntos de dados matriciais;
- Operações de álgebra linear, transformada de Fourier e geração de números aleatórios;
- Ferramentas para a integração em outras linguagens de programação como C, C++ e Fortran.

Além da capacidade de rápido processamento em matrizes que o *NumPy* oferece ao Python, um dos principais objetivos em relação a análise de dados é que serve como um "container" para os dados serem passado por algoritmos. Para dados numéricos, as matrizes de *NumPy* são muito mais eficientes para a ordenação e manipulação de dados do que qualquer outra estrutura embutida em Python. Igualmente, bibliotecas escritas em linguagens de baixo nível, como C ou Fortran, podem operar dados gravados em matrizes da *NumPy* sem precisar da cópia de qualquer dado (MCKINNEY, 2013).

A biblioteca *NumPy* por si só, não provê uma funcionalidade de alto-nível para a análise de dados. Tendo um conhecimento sobre as matrizes de *NumPy* e matrizes orientadas a computação (*array-oriented computing*) irá facilitar o uso de outras ferramentas, como *pandas*, com mais efetividade.

Para aplicações voltadas para a análise de dados, esta biblioteca possui grande funcionalidade em setores como:

- Criação rápida de matrizes para a interação e limpeza de dados, separação e filtragem, transformação e outros tipos de operações computacionais;
- Algoritmos comuns para matrizes como ordenação, operações únicas e definidas;
- Eficiente descrição estatística e agregação/sumarização de dados;
- Alinhamento de dados e manipulação de dados relacionais para operações de junção e imerção (*join* e *merge*) de conjuntos de dados heterogêneos;
- Expressar lógicas de condições através de expressões matriciais ao invés de laços de repetições e condições como *while*, *for*, *if-elif-else*;
- Agrupamento de manipulação de dados (agregação, transformação, aplicação de funções).

Enquanto *NumPy* oferece o fundamento computacional para essas operações, é preferível utilizar a biblioteca *pandas* como base para a mineração de dados (especialmente de dados estruturados ou dados tabulados), devido a sua interface rica e de alto-nível no qual permite as tarefas com dados mais concisas e simples.

#### 4.1.1.2 Biblioteca *pandas*

A biblioteca *pandas* atrai o maior interesse de cientistas quanto a mineração de dados. Ela possui estruturas de dados de alto-nível e ferramentas de manipulação desenvolvidas para facilitar e agilizar a análise de dados em Python. *pandas* é desenvolvida sob a biblioteca *NumPy* e viabiliza o uso em aplicações centradas nesta. A seguir são expostas algumas soluções que a biblioteca disponibiliza (MCKINNEY, 2013):

- Estrutura de dados com eixos rotulados suportam o alinhamento de dados automáticos ou explícitos. Isso evita erros comuns resultantes de dados desalinhados e dados indexados de formas diferentes provenientes de outras fontes de dados;
- A mesma estrutura de dados consegue manusear tanto dados de séries temporais como dados não-temporais;
- Operações e reduções aritméticas é passado para metadados (eixos rotulados);
- Manipulação flexível de dados em falta;
- *Merge* (fundir) e outras operações relacionais encontradas em bancos de dados relacional.

Esta biblioteca possui duas estrutura de dados principais: *Series* e *DataFrame*. Estas estruturas não são uma solução universal para todos os problemas, mas provê uma base sólida e de fácil manipulação para a maioria das aplicações com mineração de dados.

Uma *Series* é um tipo de *array* ou uma matriz unidimensional, similar a um *array* que possui uma matriz de dados (qualquer tipo de dado da biblioteca *NumPy*) e um outro vetor associado a dados rotulados, chamados de *index* (índice). Uma simples *Series* é formado por uma única matriz de dados, conforme a Figura 2.

```
[In [5]: obj = Series([4, 7, -5, 3])
[In [6]: obj
Out[6]:
0    4
1    7
2   -5
3    3
dtype: int64]
```

FIGURA 2: Exemplo de uma *Series*

FONTE: McKinney (2013)

*DataFrame* representa uma tabela, uma estrutura de dados do tipo planilha, que possui uma coleção ordenada de colunas, onde cada uma delas pode ter um tipo de valor diferente (numérico, *string*, *boolean*, etc.). O *DataFrame* possui um índice para linhas e também para colunas. Pode ser interpretado como um dicionário de *Series*. De uma maneira geral, o dado é armazenado como um ou mais blocos bidimensionais ao invés de uma lista, dicionário, ou outro tipo de coleção de matriz unidimensional (MCKINNEY, 2013).

Existem várias maneiras diferentes de se criar um *DataFrame*, entretanto uma forma comum é um dicionário de dimensões iguais, conforme a Figura 3 e a Figura 4, ou uma matriz *NumPy*.

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

frame = DataFrame(data)
```

FIGURA 3: Criação de um *DataFrame*

FONTE: McKinney (2013)

	pop	state	year
0	1.5	Ohio	2000
1	1.7	Ohio	2001
2	3.6	Ohio	2002
3	2.4	Nevada	2001
4	2.9	Nevada	2002

FIGURA 4: Conteúdo de um *DataFrame* pelo interpretador *IPython*

FONTE: McKinney (2013)

#### 4.1.1.3 Biblioteca *matplotlib*

O pacote *matplotlib* é desenvolvido para a geração de gráficos bidimensionais a partir de *arrays*. Gráficos comuns podem ser criados com alta qualidade a partir de simples comandos, inspirados nos comandos gráficos do MATLAB, exemplo ilustrado na Figura 5.

Quando usado em conjunto com ferramentas GUI (*IPython*, por exemplo), esta biblioteca possui recursos interativos como zoom e visão panorâmica. Além disto, suporta várias ferramentas GUI *backend*, nos diversos sistemas operacionais suportados pelo Python, e permitem exportar gráficos em diversos formatos: PDF, SVG, JPG, PNG, BMP, GIF, etc.

*matplotlib* também possui várias ferramentas adicionais, como o *mplot3d* para plotar gráficos tridimensionais, e o *basemap* para mapeamentos e projeções.



FIGURA 5: Exemplo de um gráfico gerado pelo *matplotlib*

FONTE: Wiener (2014)

#### 4.1.1.4 Biblioteca *SciPy*

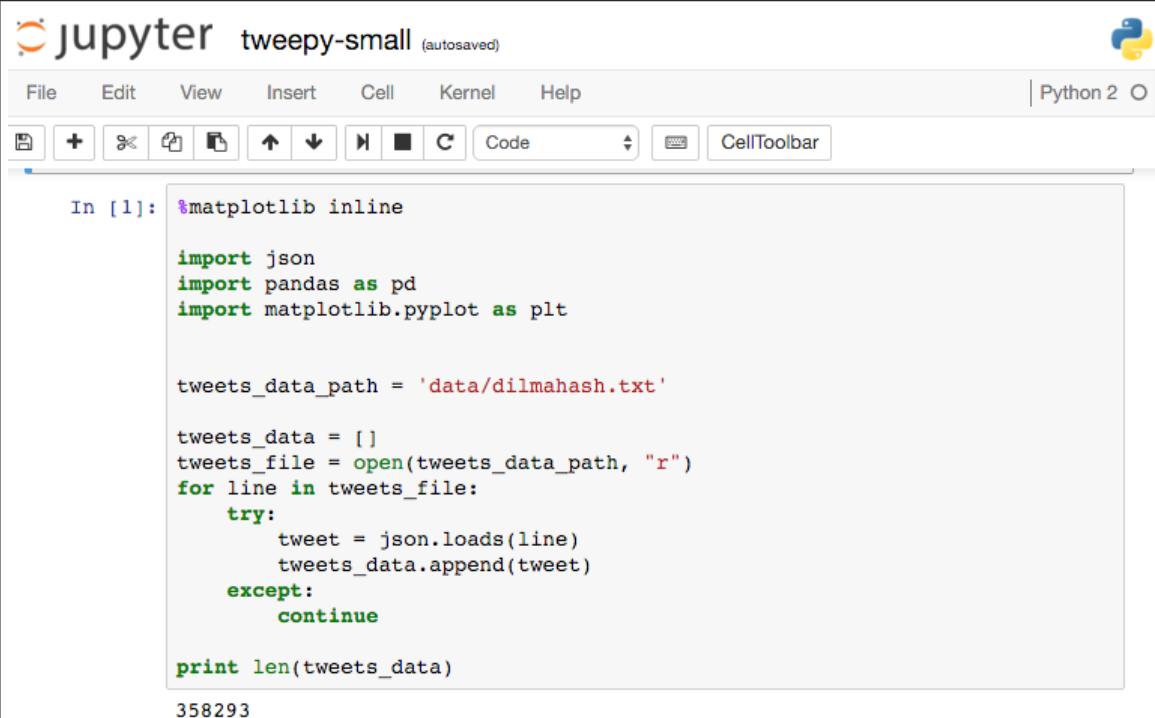
*SciPy* é uma coleção de pacotes que abordam uma série de soluções para diferentes domínios na computação científica. Na lista a seguir são apresentados exemplos desses pacotes (MCKINNEY, 2013):

- *scipy.integrate*: rotinas de integração numéricas e soluções de equações diferenciais;
- *scipy.linalg*: rotinas de álgebra linear e decomposição de matrizes;
- *scipy.optimize*: funções otimizadoras (minimizadoras) e algoritmos de busca em raíz;
- *scipy.signal*: ferramentas para processamento de sinais;
- *scipy.sparse*: matrizes esparsas e soluções de sistemas lineares esparsos;

- *scipy.special*: agregador do *SPECFUN*, uma biblioteca do Fortran que implementa várias funções matemáticas, como exemplo, a função gama;
- *scipy.stats*: funções estatísticas, variáveis contínuas e discretas, testes estatísticos e outros modelos estatísticos;
- *scipy.weave*: ferramenta para usar códigos *inline* de C++ para acelerar a computação de matrizes.

#### 4.1.1.5 Interpretador *IPython*

O interpretador *IPython* teve seu desenvolvimento iniciado em 2001, com o intuito de ser um interpretador interativo para a linguagem Python. Desde a sua criação o *IPython* evoluiu grandemente, ao ponto de ser considerada uma das mais importantes ferramentas para computação científica em Python. Essa biblioteca não oferece nenhuma ferramenta para análise de dados ou análise computacional em si, sendo designada para maximizar a produtividade, tanto na interação computacional como no desenvolvimento de softwares. Oferece um fluxo de visualização de um modo *execute-explore* ao invés do típico modelo *edit-compile-run* de muitas outras linguagens de programação. Ela também provê uma pequena integração com o *shell* e sistemas de arquivos. Como a maior parte da programação focada na mineração de dados envolve exploração, tentativa, erro e iteração, *IPython*, em quase todos os casos, irá facilitar este tipo de trabalho (MCKINNEY, 2013).



The screenshot shows the Jupyter Notebook interface with the title "jupyter tweepy-small (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. A toolbar below the menu contains various icons for file operations. The main area displays a code cell labeled "In [1]". The code is as follows:

```

In [1]: %matplotlib inline

import json
import pandas as pd
import matplotlib.pyplot as plt

tweets_data_path = 'data/dilmahash.txt'

tweets_data = []
tweets_file = open(tweets_data_path, "r")
for line in tweets_file:
    try:
        tweet = json.loads(line)
        tweets_data.append(tweet)
    except:
        continue

print len(tweets_data)

```

The output of the cell is "358293".

FIGURA 6: Exemplo de uma página web do *IPython Notebook*  
FONTE: Elaborado pelo autor

Hoje, o projeto *IPython*, mantido pela empresa *Jupyter*, engloba muito mais do que apenas um interpretador *shell* para Python. Ele também inclui um console gráfico interativo, o *IPython Notebook*, que provê ao usuário uma experiência de caderno (*notebook-like*) através de um navegador *web*, conforme Figura 6, e dispõe de um mecanismo de processamento paralelo. Assim como muitas outras ferramentas desenvolvidas para programadores, é extremamente customizável (RUSSELL, 2013).

#### 4.1.1.6 Biblioteca *Folium*

A linguagem Python possui a biblioteca *Folium* que permite a visualização de dados em um mapa interativo desenvolvido em JavaScript o *Leaflet*. *Folium* permite que os dados de coordenadas sejam combinados com marcadores para a atribuição de informações ou notas em um mapa (DIENER, 2015).

A biblioteca possui integração com várias tecnologias que disponibilizam dados para a criação de mapas interativos, como *OpenStreetmap*, *MapQuest Open*, *Mapbox* e *Stamen* (DIENER, 2015).

#### 4.1.1.7 Biblioteca *NLTK*

*Natural Language Toolkit* - NLTK, é um conjunto de bibliotecas e programas para a representação simbólica e estatística do processamento de linguagem natural (NLP) para o Python (BIRD, 2006).

A biblioteca NLTK se destina a apoiar a pesquisa e ensino em NLP ou áreas correlativas incluindo linguagem empírica, ciência cognitiva, inteligência artificial, recuperação de informação e aprendizado de máquina. Permite o suporte a classificação, utilização de *tokens*, *stemming*, *tagging*, análise (*parsing*) e funcionalidades de raciocínio semântico (BIRD, 2006).

Para a implementação de alguns códigos, esta biblioteca possibilitará a exclusão ou inclusão de certas palavras e símbolos durante a análise dos dados.

#### 4.1.1.8 Biblioteca *Word Cloud*

*Word Clouds*, conhecidas também como *Tag Clouds*, são nuvens de palavras ou imagens formadas por um conjunto de palavras, utilizadas para apresentação e interpretação de palavras em um determinado texto.

Python possui uma biblioteca para a criação de *Word Clouds*, em que, através de uma visualização, cada palavra tem seu tamanho regido pela relevância em determinado texto. Geralmente se trata de uma contagem simples das ocorrências de determinada palavra no texto (YANG; KAVANAUGH, 2011).

#### 4.1.2 Interface de Programação de Aplicações - API

API é uma sigla para *Application Programming Interface* e basicamente é uma tecnologia que permite um pedaço de *software* se comunicar com outro pedaço de *software*. Existem vários tipos de API e é comumente referenciado a outras tecnologias. Por exemplo, para o desenvolvimento deste trabalho será utilizado a API do *Twitter*.

Uma API é composta por uma série de funções acessíveis somente por programação, e que permitem utilizar características do *software* menos evidentes ao usuário tradicional. Neste caso, a rede social *Twitter* possui vasta quantidade de dados, aonde através de sua API, permite que desenvolvedores externos possam implementar tecnologias que utilizam os seus dados.

##### 4.1.2.1 Arquitetura REST

Abreviação para Transferência de Estado Representacional (REST), é um estilo arquitetural baseado em recursos e nas representações desses recursos. Enfatiza a escalabilidade na interação entre componentes, a generalidade de interfaces, a implantação independente dos componentes de um sistema, o uso de componentes intermediários visando a redução na latência de interações, o reforço na segurança e o encapsulamento de sistemas legados. A REST ignora os detalhes da implementação de componente e a sintaxe de protocolo com o objetivo de focar nos papéis dos componentes, nas restrições sobre sua interação com outros componentes e na sua interpretação de elementos de dados significantes (FIELDING, 2000).

REST foi um termo criado por Fielding (2000), onde ele modela um estilo de arquitetura para a construção de serviços *web* consistentes e coesos. O estilo da arquitetura REST é baseado em recursos e nos estados desses recursos.

A funcionalidade de uma REST API é similar ao funcionamento de uma página *web*, onde o usuário efetua uma requisição a um servidor *web*, utilizando o protocolo HTTP, e recebe dados como resposta.

Um recurso é qualquer conteúdo ou informação que é exposto na Internet, podendo ser um documento, vídeo clip, até processos de negócio ou dispositivos. Para utilizar um recurso é necessário ser capaz de identificá-lo na rede e de ter meios para manipulá-lo. Tem-se então, o *Uniform Resource Identifiers* (URI) para este propósito. Um URI unicamente identifica um recurso e, ao mesmo tempo, o torna endereçável ou capaz de ser manipulado utilizando um protocolo, como o HTTP. O URI de um recurso se distingue dos de qualquer outro recurso e é através do próprio URI que ocorrem as interações com o recurso (WEBBER; PARASTATIDIS; ROBINSON, 2010).

Recursos devem possuir pelo menos um identificador para ser endereçável, e

cada identificador é associado com uma ou mais representações, que é uma transformação ou uma visão do estado do recurso em um instante de tempo. Essa visão é codificada em um ou mais formatos transferíveis, tal como XHTML, Atom, texto simples, XML, YML, JSON, JPG, MP3, entre outros (WEBBER; PARASTATIDIS; ROBINSON, 2010).

Os recursos provêm o conteúdo ou objeto com o qual se quer interagir e para atuar sobre eles é utilizado os métodos de HTTP. Os métodos HTTP na arquitetura REST podem ser referenciados como Verbos, uma vez que representam ações sobre os recursos (WEBBER; PARASTATIDIS; ROBINSON, 2010).

#### 4.1.3 Protocolo de Autenticação - OAuth

Protocolos de autenticação são capazes de, simplesmente, autenticar a parte que está se conectando, ou ainda de autenticar a parte que está conectando, assim como se autenticar para ele.

Neste trabalho será utilizado apenas o protocolo OAuth 1.0 para o acesso aos dados do *Twitter*. É possível também, realizar a autenticação utilizando a versão mais atual, OAuth 2.0, mas será apenas referenciado, neste trabalho, para a melhor compreensão do funcionamento do protocolo.

OAuth é uma sigla para "*open authorization*", ou autorização aberta, e provê um meio para que usuários autorizem uma aplicação acessar dados, com alguma finalidade, através de uma API, sem que os usuários precisem passar credenciais como nome de usuário e senha. De um modo geral, usuários são capazes de controlar o nível de acesso para estas aplicações e revogar este controle a qualquer momento (RUSSELL, 2013).

##### 4.1.3.1 Protocolo OAuth 1.0a

OAuth 1.0a é um protocolo que permite que um cliente (*client*) *web* tenha acesso a um recurso protegido pelo seu dono em um servidor. Esta definição se dá através da RFC 5849. Que são documentos técnicos desenvolvidos e mantidos pelo Internet Engineering Task Force (IETF), instituição que especifica os padrões que serão implementados e utilizados em toda a Internet.

A razão para a existência dessa tecnologia é para evitar problemas de usuários (donos dos recursos) compartilhar suas senhas com aplicações *web*.

A versão OAuth 1.0a não permite que credenciais sejam trocadas utilizando uma conexão *Secure Socket Layer* (SSL) através de um protocolo HTTPS. Por esse motivo, muitos desenvolvedores achavam tedioso o trabalho devido aos vários detalhes envolvidos em encriptação.

SSL é um padrão global para tecnologia de segurança. Tem como função principal criar um canal criptografado entre um servidor *web* e um navegador (*browser*) para garantir que todos os dados transmitidos sejam seguros e sigilosos.

Uma aplicação que está requerindo acesso é conhecida como *client*, em alguns momentos chamado de *consumer*, a rede social ou o serviço que contém os recursos protegidas é nomeado como *server* (também chamado de *provider*) e o usuário que concede o acesso é o *resource owner* (dono do recurso, tradução livre). Com estes elementos, as três participações que envolvem o processo e a interação que estes elementos possuem é conhecida como "*three-legged-flow*" ou de uma maneira mais coloquial, a *OAuth dance*. Estas são as etapas fundamentais que envolvem a *OAuth dance* que, como resultado, permite ao *client* o acesso a recursos protegidos, conforme listado a seguir (RUSSELL, 2013):

1. O *client* obtém um *token* de requisição do servidor de serviço (aplicação);
2. O dono do recurso autoriza o *token* de requisição;
3. O *client* troca o *token* de requisição por um *token* de acesso;
4. O *client* usa o *token* de acesso para acessar os recursos protegidos com a consideração do dono do recurso.

Para credenciais particulares, um *client* começa com uma *consumer key* e um *consumer secret* e no fim do processo de *OAuth dance*, termina com um *token* de acesso e *token* de acesso secreto que pode ser usado para acessar recursos protegidos.

#### 4.1.3.2 Protocolo OAuth 2.0

Enquanto o protocolo OAuth 1.0a permite uma autorização útil para o acesso a aplicações *web*, o OAuth 2.0 foi originalmente destinado a simplificar, significamente, a implementação detalhada para desenvolvedores de aplicações *web*, sendo fundamentado completamente no SSL para aspectos de segurança e para satisfazer uma vasta quantidade de casos de uso. Esses casos de uso variaram desde suporte para dispositivos móveis à necessidades empresariais e, consequentemente, às necessidades de um termo mais futuro, da "Internet das Coisas" (RUSSELL, 2013).

Diferentemente da implementação OAuth 1.0a, que consiste de um rígido conjunto de etapas, a implementação do OAuth 2.0, definido através do RFC 6749, pode variar de acordo com a particularidade do caso de uso. Um decorrer típico da execução do OAuth 2.0 tem a vantagem do SSL e, essencialmente, contém apenas poucos redirecionamentos que, acompanhada de em alto-nível, não possui tanta diferença em relação ao processo anterior que envolvem um ciclo do OAuth 1.0a.

#### 4.1.4 Rede Social *Twitter*

Para definir o que seria o *Twitter*, Russell (2013) aborda as seguintes necessidades que uma tecnologia social precisa disponibilizar à uma pessoa:

- Permitir que a pessoa seja ouvida;
- Permitir que a pessoa satisfaça suas curiosidades;
- Permitir de um modo fácil e acessível;
- Permitir agora.

Essas observações são, de um modo geral, verdadeiramente humanas. Pessoas possuem o desejo de compartilhar ideias e experiências, que as permitam se conectar com outras pessoas, para serem ouvidas, e se sentirem parte ou digna de importância (HUBERMAN; ROMERO; WU, 2008).

Os dois últimos itens que Russell (2013) aborda ao comentar sobre as necessidades de uma tecnologia social, enfatizam a vontade de não ter alguma dificuldade para satisfazer as curiosidades ou realizar algum trabalho específico.

Uma maneira de descrever o *Twitter* é, então, como um serviço de *microblog* que permite uma breve comunicação entre pessoas. Mensagens de no máximo 140 caracteres explicam a "breve comunicação", que normalmente correspondem à pensamentos ou ideias sobre um determinado assunto, em outras palavras, *Twitter* é um serviço global de trocas de mensagens, extremamente rápido e gratuito (KWAK et al., 2010).

Outro elemento interessante para a utilização dessa rede social é a assimetria ao "seguir" uma pessoa. Esta liberdade permite aos usuários atender as suas curiosidades. Diferentemente de outras redes sociais, como *Facebook* e *LinkedIn*, que requer uma aceitação mútua da conexão entre os usuários, o modelo de relacionamento no *Twitter* viabiliza os acontecimentos de qualquer outro cliente na rede, podendo ser alguma especulação sobre celebridades, atualizações sobre times de esportes, algum tópico particular político ou o simples desejo de se conectar a alguém (HUBERMAN; ROMERO; WU, 2008).

##### 4.1.4.1 API do *Twitter*

*Twitter* é caracterizado como um serviço de *microblog* em tempo real, que permite os usuários postarem pequenas atualizações de qualquer situação que desejar, essas postagens são chamados de *tweets*, que aparecem em *timelines* (linhas de

tempo). *Tweets* podem incluir uma ou mais entidades em seus 140 caracteres de conteúdo e referenciar um ou mais lugares que mapeiam localizações do mundo real (RUSSELL, 2013).

A essência do *Twitter* são os *tweets* e, apesar deles serem representados como 140 caracteres de conteúdo textual associado a uma atualização de um usuário, existe muito mais informação através dos metadados. Em adição ao conteúdo textual de um *tweet*, eles vêm empacotados em dois pedaços de metadados: entidades e lugares. Entidades de um *tweet* são, basicamente, menções de usuários, *hashtags*, URLs e mídias que são associadas a um *tweet*. Lugares são localizações no mundo real, que podem estar adjunto a um *tweet*. Um lugar pode ser uma localização atual aonde o *tweet* foi composto, mas também pode ser uma referência a algum lugar descrito no *tweet* (TWITTER, 2016b).

Para exemplificar, considere o *tweet* apresentado pela Figura 7. O *tweet* possui 83 caracteres (incluindo espaços e quebra de linhas) e contém quatro entidades: duas *hashtags* #GameOfThrones e #Linux, uma menção ao usuário @itsfoss e uma imagem como conteúdo de mídia. Não existe porém, uma localidade exposta nesta postagem, porém existe muito mais informações no metadado deste *tweet* que demonstra a quantidade de dados que se pode coletar.

*Timelines* são coleções de *tweets* ordenadas cronologicamente. É possível afirmar, que *timeline* é qualquer coleção específica de *tweets* apresentadas em uma ordem cronológica. Através da perspectiva de um usuário do *Twitter*, a *timeline* inicial (*home*) é a visualização principal quando este usuário acessa a sua conta no serviço do *Twitter* (RUSSELL, 2013). Quando este acesso é realizado, normalmente acontece através do endereço <https://twitter.com>. A URL para a *timeline* de algum usuário em particular, no entanto, precisa ser sufixado com o contexto que identifica este usuário específico, por exemplo; <https://twitter.com/unixstickers>.

A figura 8 ilustra a aplicação *TweetDeck* que disponibiliza a customização da visualização de várias *timelines*.

*Twitter* possui três tipos de APIs que permitem que desenvolvedores accessem seus dados, isto é, são capazes de coletar, ou consumir, *tweets* completos, entidades específicas e também informações dos usuários ou réplicas de mensagens.

A API de busca do *Twitter* (*Twitter's Search API*), tem a funcionalidade de capturar dados através de uma busca ou um usuário. Esta API permite o acesso a dados que já existem em *tweets* previamente publicados. Através da API de busca, o usuário solicita *tweets* que correspondem a algum critério de busca, podendo ser nome de usuários, localizações, nome de lugares, palavras-chaves e *hashtags* (TWITTER, 2016b).



FIGURA 7: Exemplo de um *tweet*

FONTE: Twitter (2016a)

Através da API de busca, desenvolvedores podem coletar *tweets* que ocorreram e são limitados a um resultado de 3200 publicações dependendo do critério de busca. Utilizando uma *hashtag* específica a API permite coletar até 5000 *tweets* por *hashtag*. Estas limitações acontecem em um determinado período de tempo, onde hoje, são permitidos 180 requisições a cada 15 minutos (TWITTER, 2016b).

A segunda tecnologia, e também a tecnologia utilizada neste trabalho para a



FIGURA 8: Aplicação *TweetDeck* apresentando várias *timelines*

FONTE: Elaborado pelo autor

coleta dos dados, é a API de *streaming* do *Twitter* (*Twitter's Streaming API*) onde se comporta de maneira diferente da API de busca. Nesta, é recebido dados que estão acontecendo em tempo real. Através da API de *streaming*, é possível registrar um conjunto de critérios de busca, para que cada vez que um *tweet* corresponder ao critério salvo, irá ser enviado diretamente ao seu solicitante (TWITTER, 2016b).

A principal desvantagem da API de *streaming* é que o *Twitter* provê apenas uma parte das publicações que estão acontecendo. E então, a porcentagem dos *tweets* coletados por esta API variam de acordo com o número de requisições de usuários durante o tráfego de dados naquele exato momento. A razão por não receber todas as publicações é porque o *Twitter* precisa dispor de uma infraestrutura que suporte todas as requisições dos *tweets* que estão sendo publicados (KWAK et al., 2010).

O *Twitter Firehose* é a última API disponibilizada e funciona semelhante a API de *streaming*, onde é enviado a usuários finais os dados de *tweets* que são publicados em tempo real. A diferença é que o *Twitter Firehose* garante o consumo de 100% dos *tweets* que correspondem ao critério de busca (TWITTER, 2016b).

A API de *Firehose* é mantida por dois provedores de dados, GNIP e DataSift, que possuem relações empresariais com o *Twitter*. Devido a esta relação e a disponibilidade de infraestrutura destas empresas, a utilização do *Firehose* não é gratuita mas também exclui várias restrições de usos impostas pelo *Twitter*, diferentemente da API de *streaming* onde é possível realizar a coleta sem custos (KWAK et al., 2010).

#### 4.1.4.2 Bibliotecas Para o Consumo de Dados da API do *Twitter*

O acesso a API acontece através da criação de uma aplicação pela página *web* de desenvolvimento do *Twitter*, <https://apps.twitter.com>. Após a criação desta aplicação é fornecido ao usuário informações para o acesso utilizando o protocolo OAuth. Será informado uma chave exclusiva da API da aplicação, uma chave secreta, o *token* de usuário OAuth e credenciais secretas do usuário OAuth (RUSSELL, 2013).

Com estas credencias é possível, então, ter acesso as APIs do *Twitter* e, consequentemente, o que a mesma disponibiliza para os usuários. Para o consumo desta API, a linguagem de programação Python apresenta algumas bibliotecas que facilita este tipo de serviço, listadas a seguir:

- *requests-oauthlib*;
- *tweepy*;
- *python-twitter*;
- *oauthlib*.

A biblioteca *tweepy*, será a mais utilizada para as implementações deste trabalho devido a fácil interação com a APIs do *Twitter*. Esta biblioteca permite o acesso a todos os métodos presentes na API, cada método pode aceitar parâmetros diversos e retornar respostas quando invocados (TWEETPY, 2009).

O Código 1 exemplifica o consumo da API segundo Tweepy (2009).

```

1 import tweepy
2
3 auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
4 auth.set_access_token(access_token, access_token_secret)
5
6 api = tweepy.API(auth)
7
8 public_tweets = api.home_timeline()
9 for tweet in public_tweets:
10     print tweet.text

```

CÓDIGO 1: Acesso à API do *Twitter*

Quando um método da API é invocado, quase todos os seus retornos serão uma instância de um modelo *tweepy*. Este modelo irá conter dados retornados do *Twitter* onde é possível manipulá-los da maneira como o programador preferir (TWEETPY, 2009).

## 4.2 METODOLOGIA E DESENVOLVIMENTO

Um trabalho científico, segundo Demo (1991), pode ser avaliado pela sua qualidade política e pela sua qualidade formal. A qualidade política é referenciada pelos seus conteúdos, relevância e utilidade prática. Já a qualidade formal, diz respeito aos meios e formas usados na produção do trabalho, referindo-se ao domínio de técnicas de coleta e interpretação de dados, manipulação de fontes de informação, conhecimento demonstrado na apresentação do referencial teórico e apresentação escrita ou oral em conformidade com os ritos acadêmicos (SILVA; MENEZES, 2001).

A característica política deste trabalho se dá com a demonstração da possibilidade de utilização da linguagem Python para a mineração de dados em redes sociais, especificamente neste presente trabalho o *Twitter*, através das APIs disponibilizadas por este.

Para o cumprimento das qualidades citadas, desempenhou-se uma pesquisa científica em que segundo Goldenberg (2002), precisa satisfazer os seguintes requisitos:

1. a existência de uma pergunta que se deja responder;
2. a elaboração de um conjunto de passos que permitam chegar à resposta;
3. a indicação do grau de confiabilidade na resposta obtida.

A pergunta a ser respondida diz respeito da viabilidade de utilizar Python como uma ferramenta útil para a mineração de dados e, consequentemente, a apresentação dos resultados obtidos. Sendo necessário o conhecimento teórico sobre técnicas de mineração de dados e das funcionalidades e bibliotecas que a linguagem em estudo dispõe.

Para o cumprimento do segundo requisito de Goldenberg (2002), foram desenvolvidos *scripts* para a autenticação, coleta e limpeza de dados utilizando o modelo Iterativo e Incremental como técnica de Engenharia de *Software*.

O desenvolvimento Iterativo e Incremental é um modelo clássico para o processo de criação de *softwares*, que utiliza padrões do modelo *Rational Unified Process* - RUP e também de desenvolvimentos ágeis, tem como característica o retrabalho em que o tempo de revisão e melhorias de partes dos *scripts* é pré-definido e o planejamento estagiado em que várias partes do *script* são desenvolvidos em paralelo e integrados quando completos (RUP, 2003).

A apresentação dos resultados e, também, o indicativo da resposta alcançada pela mineração dos dados, ocorreu através da implementação de códigos Python que resultaram em informações interpretadas através de gráficos, imagens e mapa.

O presente trabalho se caracteriza, também, segundo Gil (2002), como uma pesquisa exploratória, em que visa proporcionar familiaridade com problema ou necessidade e torná-lo explícito ou também a construção de hipóteses. Através da interpretação dos dados minerados é possível a explicitação e o desenvolvimento de hipóteses apresentadas pelas publicações na rede social *Twitter*.

## 5 IMPLEMENTAÇÃO DAS TÉCNICAS

Este capítulo tem como finalidade apresentar, com um maior nível de detalhamento, as técnicas utilizadas neste trabalho, com o objetivo de atingir as metas propostas na seção 1.2.

### 5.1 COLETA DE DADOS

Uma característica comentada anteriormente sobre a rede social *Twitter* é a disponibilidade de informações de eventos em tempo real. Os *tweets* podem ser postados com o intuito de comentar sobre a final de um campeonato de futebol, datas importantes, acontecimentos internacionais e políticos, entre outros.

Para este trabalho foi aproveitado o dia 17 de abril de 2016, onde foi realizado a votação no Congresso Brasileiro pela continuação do processo de Impeachment do cargo de presidente da senhora Dilma Rousseff. Neste dia, milhares de *tweets* foram publicados utilizando a *hashtag* #ImpeachmentDay, a fim de comentar sobre o evento e, também, a atual situação política do Brasil.

Com a finalidade de coletar todos os dados da referida *hashtag*, foi desenvolvido um *script* que utiliza o serviço da API de *streaming* do *Twitter*.

As primeiras linhas mostradas no Código 2 servem para importar as bibliotecas e objetos necessários para a utilização da API do *Twitter*, assim como as informações das chaves de acesso e *tokens* para o protocolo OAuth, que são fundamentais para a comunicação com o serviço de *Stream*.

```

1 from tweepy.streaming import StreamListener
2 from tweepy import OAuthHandler
3 from tweepy import Stream
4
5 access_token = "131556934-LrYRiXzAL3QcRyFN0fdN53EDWhNGfZFnVX59NCnT"
6 access_token_secret = "JraMtps5IB98d8XoeIAF71KHn8ZQ4nshdoSKiTz6OHd"
7 consumer_key = "P4XZ2GUkeqdhIQMOredBuW05"
8 consumer_secret = "r5TPb2UcM8bzq7t5zfIRPMHUrCfwNG4GRuVPXypowrpHhTmue"
9
10
11 class StdOutListener(StreamListener):
12
13     def on_data(self, data):
14         print data
15         return True
16
17     def on_error(self, status):

```

```

18     print status
19
20
21 if __name__ == '__main__':
22
23     l = StdOutListener()
24     auth = OAuthHandler(consumer_key, consumer_secret)
25     auth.set_access_token(access_token, access_token_secret)
26     stream = Stream(auth, l)
27
28     stream.filter(track=['ImpeachmentDay', 'NaoVaiTerGolpe', 'ForaDilma'])

```

CÓDIGO 2: *Script* coletar-hashtags.py

Destaca-se neste código que a partir da linha 11 é criado uma classe que instancia um serviço de escuta para este *Stream*. O serviço de escuta é responsável por observar todos os *tweets* que são publicados e, após identificar no *tweet* alguma palavra ou a *hashtag* semelhante às palavras declaradas na linha 28, captura e imprime na tela de execução do *script*.

Quando este *script* está em execução é mostrado na tela todos os *tweets*, já filtrados, que estão sendo coletados. Porém, estes não estão persistindo em nenhum arquivo ou banco de dados. Com o intuito de salvar todos estes dados, foi utilizado o comando *stdout* (>), que está presente em sistemas operacionais de base *Unix*. Este comando permite redirecionar a saída do código anterior, no caso a execução do *script* coletar-hashtags.py, para um novo arquivo ou um arquivo já existente, conforme ilustrado pela Figura 9.

```

scripts git:(master) ✘
> python coletar-hashtags.py > ../data/coleta-impeachment.json

```

FIGURA 9: Execução do *script* para coleta de dados

FONTE: Elaborado pelo autor

O *script* apresentado no Código 2 permaneceu em execução durante 12 horas (das 10:30 às 22:30) do dia 17 de abril, resultando em um arquivo de 2.6 gigabytes. O formato deste arquivo é um JSON, onde apresenta todas as informações presentes em um *tweet*, como demonstrado através do Código 3.

```

1 {
2     "created_at": "Sun Apr 17 15:15:21 +0000 2016",
3     "id": 721718699418202112,
4     "id_str": "721718699418202112",

```

```

5   "text": "RT @GarotaCiume: N\u00e3o sou petista , s\u00f3 n\u00e3o sou cega. Impeachment sem crime de responsabilidade \u00e9 golpe , e voc\u2019\u00e1 u00e9as est\u00e1 traindo a democracia. #\u2026",
6   "source": "\u003ca href=\"http://twitter.com/download/android\" rel=\"nofollow\"\u003eTwitter for Android\u003c/a\u003e",
7   "truncated": false,
8   "in_reply_to_status_id": null,
9   "in_reply_to_status_id_str": null,
10  "in_reply_to_user_id": null,
11  "in_reply_to_user_id_str": null,
12  "in_reply_to_screen_name": null,
13  "user": {
14    "id": 1876843824,
15    "id_str": "1876843824",
16    "name": "Millena Souza",
17    "screen_name": "Souzaa_mih1",
18    "location": "Brasil , Vit\u00f3ria - ES",
19    "url": "https://www.facebook.com/myllena.souzaa",
20    "description": "Brasileira 15 , paulista e capixaba , feminista , Taurina e n\u00e3o sou obrigada a nada \u2764\u2764 BORN TO DIE !!",
21    "protected": false,
22    "verified": false,
23    "followers_count": 433,
24    "friends_count": 431,
25    "listed_count": 0,
26    "favourites_count": 613,
27    "statuses_count": 28370,
28    "created_at": "Tue Sep 17 20:52:04 +0000 2013",
29    "utc_offset": -10800,
30    "time_zone": "Brasilia",
31    "geo_enabled": true,
32    "lang": "pt",
33    "contributors_enabled": false,
34    "is_translator": false,
35    "profile_background_color": "611994",
36    "profile_background_image_url": "http://pbs.twimg.com/profile_background_images/378800000076733988/60df12d5296d8d8081e19c306c15c859.jpeg",
37    "profile_background_image_url_https": "https://pbs.twimg.com/profile_background_images/378800000076733988/60df12d5296d8d8081e19c306c15c859.jpeg",
38    "profile_background_tile": true,
39    "profile_link_color": "6B25C7",
40    "profile_sidebar_border_color": "FFFFFF",
41    "profile_sidebar_fill_color": "7AC3EE",
42    "profile_text_color": "3D1957",
43    "profile_use_background_image": true,
44    "profile_image_url": "http://pbs.twimg.com/profile_images
```



```

85         "utc_offset": -10800,
86         "time_zone": "Brasilia",
87         "geo_enabled": true,
88         "lang": "pt",
89         "contributors_enabled": false,
90         "is_translator": false,
91         "profile_background_color": "FFFFFF",
92         "profile_background_image_url": "http://pbs.twimg.com/←
profile_background_images/879103282/2454b176f3fffc4b100fb2bc64b1f2b5←
.png",
93         "profile_background_image_url_https": "https://pbs.twimg.←
com/profile_background_images/879103282/2454←
b176f3fffc4b100fb2bc64b1f2b5.png",
94         "profile_background_tile": false,
95         "profile_link_color": "383838",
96         "profile_sidebar_border_color": "FFFFFF",
97         "profile_sidebar_fill_color": "F6F6F6",
98         "profile_text_color": "000000",
99         "profile_use_background_image": true,
100        "profile_image_url": "http://pbs.twimg.com/profile_images/←
3719404815/b868778f39b4b087561e3444b3093e20_normal.gif",
101        "profile_image_url_https": "https://pbs.twimg.com/←
profile_images/3719404815/b868778f39b4b087561e3444b3093e20_normal.←
gif",
102        "profile_banner_url": "https://pbs.twimg.com/←
profile_banners/206181302/1403658723",
103        "default_profile": false,
104        "default_profile_image": false,
105        "following": null,
106        "follow_request_sent": null,
107        "notifications": null
108    },
109    "geo": null,
110    "coordinates": null,
111    "place": null,
112    "contributors": null,
113    "is_quote_status": false,
114    "retweet_count": 153,
115    "favorite_count": 130,
116    "entities": {
117        "hashtags": [
118            "text": "ImpeachmentDay",
119            "indices": [121, 136]
120        ],
121        "urls": [],
122        "user_mentions": [],
123        "symbols": []
124    },

```

```

125     "favorited": false ,
126     "retweeted": false ,
127     "filter_level": "low" ,
128     "lang": "pt"
129 },
130     "is_quote_status": false ,
131     "retweet_count": 0,
132     "favorite_count": 0,
133     "entities": {
134         "hashtags": [{{
135             "text": "ImpeachmentDay",
136             "indices": [139, 140]
137         }],
138         "urls": [],
139         "user_mentions": [{{
140             "screen_name": "GarotaCiume",
141             "name": "Garota Ci\u00fame",
142             "id": 206181302,
143             "id_str": "206181302",
144             "indices": [3, 15]
145         }],
146         "symbols": []
147 },
148     "favorited": false ,
149     "retweeted": false ,
150     "filter_level": "low" ,
151     "lang": "pt",
152     "timestamp_ms": "1460906121483"
153 }

```

CÓDIGO 3: Exemplo de um *tweet* no formato JSON

Os *tweets* coletados, mediante a execução do *script* apresentado no Código 2, foram redirecionados para o arquivo *coleta-impeachment.json*. Este, foi gerado através do comando *stdout*, conforme ilustrado, previamente, pela Figura 9 possibilitando a conclusão da etapa de obtenção de dados.

## 5.2 ANÁLISE DE DADOS

Após a coleta dos dados foi gerado, então, um arquivo JSON de 2.6 gigabytes. Baseando-se nele, a etapa seguinte foi de analisar estes dados para extrair as informações consideradas úteis.

Foram realizados testes no arquivo JSON para verificar se existia algum tipo de *dirty data*, que são informações quebradas, dados irrelevantes ou códigos que impedem a execução de *scripts* de mineração (PIPINO; KOPCSO, 2004). O único *dirty data* encontrado apresenta informações referentes ao limite da API de *streaming* que

possui o padrão "*limit*": seguido de outras informações adicionais (TWITTER, 2016b). A Figura 10 demonstra o único padrão de *dirty data* encontrado dentro do arquivo.

```
{"created_at": "Mon Apr 18 02:46:16 +0000 2016", "id": 721891460947577319}
{"created_at": "Mon Apr 18 02:46:16 +0000 2016", "id": 721891460947577319}
{"created_at": "Mon Apr 18 02:46:17 +0000 2016", "id": 721891460947577319}
{"limit": {"track": 11138, "timestamp_ms": "1460947577319"}}
{"created_at": "Mon Apr 18 02:46:17 +0000 2016", "id": 721891460947577319}
{"created_at": "Mon Apr 18 02:46:17 +0000 2016", "id": 721891460947577319}
{"created_at": "Mon Apr 18 02:46:17 +0000 2016", "id": 721891460947577319}
```

FIGURA 10: *Dirty Data* presente no arquivo coletado

FONTE: Elaborado pelo autor

Dentro das milhares linhas do arquivo, existiam algumas linhas contendo este mesmo padrão ("*limit*:"), que foram removidos utilizando outra ferramenta de sistemas baseados em Unix, o *grep*.

A funcionalidade *grep* é considerada um "canivete suíço" para o uso de expressões regulares e possui uma funcionalidade que permite realizar uma consulta inversa, ou seja, o usuário passa um padrão ou uma palavra que deseja encontrar dentro de um determinado arquivo e, após encontrar todas as ocorrências, o *grep* seleciona tudo o que não contém este padrão ou palavra.

Logo, foi possível utilizar a consulta inversa de *grep* para gerar um arquivo sem *dirty data*, apenas informando o padrão "*limit*": combinado com o comando de *stdout* (>). Todos os *tweets* que não apresentaram esse padrão foram redirecionados para um novo arquivo, chamado de *small-data.json*, conforme apresenta a Figura 11.

```
tcc git:(master) ✘
> grep --invert-match '{"limit":' coleta-impeachment.json > small-data.json
```

FIGURA 11: Utilizando o comando *grep* para gerar um novo arquivo sem *dirty data*

FONTE: Elaborado pelo autor

Após obter o arquivo JSON, sem a presença de *dirty data*, foi possível iniciar o processo de extração de conhecimento utilizando o interpretador *IPython*.

Nesta primeira etapa foram importadas as bibliotecas necessárias para se trabalhar com arquivos do tipo JSON e, também, informado ao interpretador *IPython* para apresentar gráficos em sua funcionalidade de *notebook*, conforme a primeira linha do Código 4.

Os comandos de importação, apresentados no Código 4, foram diferenciados por sua precisão, uma vez que entre as linhas 3 e 15 carregaram-se todas as funcionalidades de cada biblioteca, enquanto a partir da linha 17, houve uma especificação na importação de funcionalidades nestas bibliotecas, permitindo o uso desta aplicação através do nome importado.

```

1 %matplotlib inline
2
3 import folium
4 import json
5 import geojsonio
6 import matplotlib as mpl
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import nltk
10 import pandas as pd
11 import re
12 import string
13 import sys
14 import time
15 import vincent
16
17 from collections import Counter
18 from collections import defaultdict
19 from datetime import datetime
20 from matplotlib import dates
21 from matplotlib import rcParams
22 from matplotlib.ticker import MaxNLocator
23 from os import path
24 from pandas.tseries.resample import TimeGrouper
25 from pandas.tseries.offsets import DateOffset
26 from scipy.misc import imread
27 from textblob import TextBlob
28 from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

```

#### CÓDIGO 4: Importação de bibliotecas Python

A primeira linha do Código 5 atribuiu a variável, "tweets\_data\_path", a localização do arquivo para a leitura dos dados, onde, através de um laço de repetição, o interpretador leu cada linha do arquivo JSON, ou seja, cada *tweet* foi adicionado a

variável "tweets\_data", em que, na linha 3, declarou-se como uma estrutura de dados do tipo lista em Python.

```

1 tweets_data_path = 'data/small-data.json'
2
3 tweets_data = []
4 tweets_file = open(tweets_data_path, "r")
5 for line in tweets_file:
6     try:
7         tweet = json.loads(line)
8         tweets_data.append(tweet)
9     except:
10        continue
11
12 tweets = pd.DataFrame()
13 print len(tweets_data)

```

CÓDIGO 5: Leitura do arquivo JSON

O Código 5 também apresenta a construção de um *DataFrame*, definido na seção 4.1.1.2, que foi alocado à variável "tweets" na linha 14. Esses *DataFrames* possuem uma arquitetura semelhante ao formato de tabelas, em que é possível adicionar uma coluna ou uma tabela a uma variável em Python.

Através de uma variável do tipo *DataFrame* foi possível determinar uma condição específica em que um *DataFrame* consiga mapear informações de uma lista Python, neste caso a variável "tweet\_data". O Código 6 demonstra esta funcionalidade para mapear o conteúdo de texto, idioma e país de um *tweet* da lista "tweets\_data" para o *DataFrame* "tweets".

```

1 tweets[ 'text' ] = map(lambda tweet: tweet[ 'text' ], tweets_data)
2 tweets[ 'lang' ] = map(lambda tweet: tweet[ 'lang' ], tweets_data)
3 tweets[ 'country' ] = map(lambda tweet: tweet[ 'place' ][ 'country' ],
4                            if tweet[ 'place' ] != None else None, tweets_data)
5
6 tweets_by_lang = tweets[ 'lang' ].value_counts()
7
8 fig, ax = plt.subplots(figsize=(20,10))
9 ax.tick_params(axis='x', labelsize=20)
10 ax.tick_params(axis='y', labelsize=15)
11 ax.set_xlabel('Idiomas'.decode('utf-8'), fontsize=20)
12 ax.set_ylabel('Número de tweets'.decode('utf-8'), fontsize=20)
13 ax.set_title('Top 4 Idiomas'.decode('utf-8'),
14               fontsize=20, fontweight='bold')
15 tweets_by_lang[:4].plot(ax=ax, kind='bar', color='mediumspringgreen')
16 plt.grid()

```

CÓDIGO 6: Mapeamento de variáveis para um *DataFrame*

A linha 6 do Código 6 demonstra a soma de *tweets* com o mesmo idioma através da função "value\_counts()".

Semelhante ao que foi realizado no Código 6, foi possível verificar quais os cinco países que mais realizaram *tweets* utilizando a hashtag #ImpeachmentDay através do Código 7.

```

1 tweets_by_country = tweets[ 'country' ].value_counts()
2
3 fig , ax = plt.subplots(figsize=(20,10))
4 ax.tick_params(axis='x' , labelsize=15)
5 ax.tick_params(axis='y' , labelsize=15)
6 ax.set_xlabel('Paises'.decode('utf-8') , fontsize=20)
7 ax.set_ylabel('Numero de tweets'.decode('utf-8') , fontsize=20)
8 ax.set_title('Top 5 Paises'.decode('utf-8') , fontsize=20, fontweight='←
    bold')
9 tweets_by_country[:5].plot(ax=ax , kind='bar' , color='lightskyblue')
10 plt.grid()
```

CÓDIGO 7: Contabilização do número de *tweets* por países

Utilizando a biblioteca para expressões regulares (*import re*, linha 11, Código 4), foi possível procurar por *tweets* de palavras específicas, ou *hashtags* independentemente da forma digitada pelo usuário, isto é, com letras maiúsculas ou minúsculas.

O Código 8, apresenta uma função que procura determinadas palavras dentro do *DataFrame*. Estas palavras então, preenchem uma coluna com os *tweets* relacionados com a palavra buscada. As palavras específicas buscadas são referentes às *hashtags* que mais foram publicadas na data de coleta dos dados.

```

1 def word_in_text(word , text):
2     word = word.lower()
3     text = text.lower()
4     match = re.search(word, text)
5     if match:
6         return True
7     return False
8
9 tweets[ 'NaoVaiTerGolpe' ] = tweets[ 'text' ].apply(lambda tweet: ←
    word_in_text('NaoVaiTerGolpe' , tweet))
10 tweets[ 'TchauQuerida' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text(←
    'TchauQuerida' , tweet))
11 tweets[ 'ForaDilma' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text('←
    ForaDilma' , tweet))
12 tweets[ 'BrasilContraOGolpe' ] = tweets[ 'text' ].apply(lambda tweet: ←
    word_in_text('BrasilContraOGolpe' , tweet))
13 tweets[ 'ForaCunha' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text('←
    ForaCunha' , tweet))
```

```

15 hashtags = [ 'ForaDilma' , 'NaoVaiTerGolpe' , 'TchauQuerida' , '←
    BrasilContraOGolpe' , 'ForaCunha' ]
16 tweets_by_hashtags = [tweets[ 'ForaDilma' ].value_counts()[True] ,
17                         tweets[ 'NaoVaiTerGolpe' ].value_counts()[True] ,
18                         tweets[ 'TchauQuerida' ].value_counts()[True] ,
19                         tweets[ 'BrasilContraOGolpe' ].value_counts()[True] ,
20                         tweets[ 'ForaCunha' ].value_counts()[True] ]
21
22 plt.subplots(figsize=(10,10))
23 colors = [ 'gold' , 'yellowgreen' , 'lightcoral' , 'lightskyblue' , 'peachpuff←
    ']
24 explode = (0.03, 0.03, 0.03, 0.03, 0.03)
25 plt.pie(tweets_by_hashtags, explode=explode, labels=hashtags, colors=←
    colors,
26           autopct='%.1f%%', shadow=True, startangle=140)
27 plt.rcParams[ 'font.size' ] = 15
28 plt.legend(tweets_by_hashtags, loc=(.95,.6), title='Número de Tweets:'.←
    decode( 'utf-8' ), fontsize=15)
29 plt.axis('equal')
30 plt.show()

```

CÓDIGO 8: Buscar *hashtags* mais publicadas

Ainda utilizando a função declarada no Código 8 e aproveitando a temática da votação no congresso, o Código 9 implementa um filtro para buscar informações com a palavra "SIM" em *tweets* com a *hashtag* #ForaDilma e a palavra "NÃO" para as publicações realizadas com a *hashtag* #NaoVaiTerGolpe.

```

1 tweets[ 'nao' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( ' nao ' , ←
    tweet))
2 tweets[ 'sim' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( ' sim ' , ←
    tweet))
3
4 tweets[ 'ImpeachmentDay' ] = tweets[ 'text' ].apply(lambda tweet: ←
    word_in_text( ' sim ' , tweet)
5                                     or word_in_text( ' nao ' , tweet)←
    )
6
7 hashtags = [ 'ForaDilma' , 'NaoVaiTerGolpe' ]
8 tweets_by_hashtags = [tweets[tweets[ 'ImpeachmentDay' ] == True][ 'ForaDilma←
    '].value_counts()[True],
9                         tweets[tweets[ 'ImpeachmentDay' ] == True][ '←
    NaoVaiTerGolpe' ].value_counts()[True]]
10
11
12 ind = np.arange(2)
13 width = 0.5
14 width2 = 0.3

```

```

15 x_pos = list(range(len(hashtags)))
16 fig, ax = plt.subplots(figsize=(12,10))
17 ax.bar(ind + width2, tweets_by_hashtags, width, color='yellowgreen')
18 ax.tick_params(axis='x', labelsize=15)
19 ax.tick_params(axis='y', labelsize=15)
20 ax.set_ylabel('Numero de tweets'.decode('utf-8'), fontsize=20)
21 ax.set_title('Ranking: ForaDilma vs. NaoVaiTerGolpe (Votacao SIM x NAO)'.decode('utf-8'),
22           fontsize=15, fontweight='bold')
23 ax.set_xticks([p + 1.1 * width for p in x_pos])
24 ax.set_xticklabels(hashtags)
25 plt.grid()

```

CÓDIGO 9: Buscar informações sobre votação

Dando continuidade ao uso da biblioteca para trabalhar com expressões regulares, o Código 10 permite encontrar *links* ou entidades de vídeo e foto que são publicados pelos usuários através de seus *tweets*. Estes *links* podem ser filtrados de acordo com a *hashtag* utilizada. A Figura 12 apresenta um pedaço desta lista com o intuito de exemplificar o que está sendo exposto.

8871	<a href="https://t.co/rtOavI9uQP">https://t.co/rtOavI9uQP</a>
8906	<a href="https://t.co/VswJPAt5ul">https://t.co/VswJPAt5ul</a>
9619	<a href="https://t.co/IiXU4ICRrH">https://t.co/IiXU4ICRrH</a>
9685	<a href="https://t.co/tlt3wTQ969">https://t.co/tlt3wTQ969</a>
9904	<a href="https://t.co/y2MILsrv6I">https://t.co/y2MILsrv6I</a>
45728	<a href="https://t.co/crggdefh02">https://t.co/crggdefh02</a>
53250	<a href="https://t.co/Y50qDJkPXp">https://t.co/Y50qDJkPXp</a>
62948	<a href="https://t.co/5mqswBDIHh">https://t.co/5mqswBDIHh</a>
63335	<a href="https://t.co/Kck2W7uc9B">https://t.co/Kck2W7uc9B</a>
64182	<a href="https://t.co/Odd6HZPlJU">https://t.co/Odd6HZPlJU</a>
67891	<a href="https://t.co/yJ1BGtHGHd">https://t.co/yJ1BGtHGHd</a>
68558	<a href="https://t.co/rTvrUyAHA9">https://t.co/rTvrUyAHA9</a>
83968	<a href="https://t.co/XinE7SVm1F">https://t.co/XinE7SVm1F</a>
90123	<a href="https://t.co/gAraHA4Fdt">https://t.co/gAraHA4Fdt</a>
90129	<a href="https://t.co/XinE7SVm1F">https://t.co/XinE7SVm1F</a>
91107	<a href="https://t.co/XinE7SVm1F">https://t.co/XinE7SVm1F</a>
91160	<a href="https://t.co/XinE7SVm1F">https://t.co/XinE7SVm1F</a>
101356	<a href="https://t.co/6hOBQnYHhW">https://t.co/6hOBQnYHhW</a>

FIGURA 12: Exemplo de *links* extraídos de *tweets*

FONTE: Elaborado pelo autor

```

1 def extract_link(text):
2     regex = r'https?://[^<>]+|www\.[^<>]+'
3     match = re.search(regex, text)
4     if match:
5         return match.group()
6     return ''
7

```

```

8 tweets[ 'link' ] = tweets[ 'text' ].apply(lambda tweet: extract_link(tweet))
9
10 tweets_relevant = tweets[tweets[ 'ImpeachmentDay' ] == True]
11 tweets_relevant_with_link = tweets_relevant[tweets_relevant[ 'link' ] != '←
    ]
12
13 print tweets_relevant_with_link[tweets_relevant_with_link[ 'TchauQuerida' ]←
    == True][ 'link' ]
14 print tweets_relevant_with_link[tweets_relevant_with_link[ 'ForaDilma' ] ==←
    True][ 'link' ]
15 print tweets_relevant_with_link[tweets_relevant_with_link[ 'ForaCunha' ] ==←
    True][ 'link' ]
16 print tweets_relevant_with_link[tweets_relevant_with_link[ 'NaoVaiTerGolpe←
    ' ] == True][ 'link' ]

```

CÓDIGO 10: Extração de *links* provenientes de *tweets*

Semelhante ao Código 8, é possível aplicar o filtro das *hashtags* para encontrar nomes com maior popularidade através do Código 11. Como pode ser observado nas linhas 8 até 14, em que os nomes dos principais personagens do momento político nacional eram citados.

```

1 tweets[ 'moro' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( 'moro' , ←
    tweet))
2 tweets[ 'cunha' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( 'cunha' ←
    , tweet))
3 tweets[ 'bolsonaro' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( '←
    bolsonaro' , tweet))
4 tweets[ 'lula' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( 'lula' , ←
    tweet))
5 tweets[ 'temer' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( 'temer' ←
    , tweet))
6 tweets[ 'feliciano' ] = tweets[ 'text' ].apply(lambda tweet: word_in_text( '←
    feliciano' , tweet))
7
8 hashtags = [ 'Sergio Moro'.decode( 'utf-8' ), 'Eduardo Cunha', 'Jair ←
    Bolsonaro', 'Lula', 'Marcos Feliciano', 'Michel Temer' ]
9 tweets_by_hashtags = [tweets[ 'moro' ].value_counts()[True], ←
10                 tweets[ 'cunha' ].value_counts()[True], ←
11                 tweets[ 'bolsonaro' ].value_counts()[True], ←
12                 tweets[ 'lula' ].value_counts()[True], ←
13                 tweets[ 'feliciano' ].value_counts()[True], ←
14                 tweets[ 'temer' ].value_counts()[True]]
15
16 plt.subplots(figsize=(10,10))
17 colors = [ 'gold' , 'yellowgreen' , 'lightskyblue' , 'lightcoral' , 'peachpuff←
    , 'mediumturquoise' ]
18 explode = (0.03, 0.03, 0.03, 0.05, 0.03, 0.03)

```

```

19 plt.pie(tweets_by_hashtags, explode=explode, labels=hashtags, colors=←
20         colors,
21         autopct='%.1f%%', shadow=True, startangle=90)
22 plt.rcParams[ 'font.size' ] = 15
23 # plt.legend(tweets_by_hashtags, loc='best')
24 plt.legend(tweets_by_hashtags, loc=(-.22,.6), title='Número de Tweets:'.←
25             decode( 'utf-8' ), fontsize=15)
26 plt.axis('equal')
27 plt.show()

```

CÓDIGO 11: Filtro para coletar nomes com mais *tweets*

A partir da linha 16 do Código 11 é implementado as informações para a apresentação de um gráfico em setores para a melhor interpretação dos resultados.

Através da utilização do *DataFrame* com a variável "tweets", é possível ainda extrair algumas outras entidades dos *tweets* como a data de criação, nome de usuário, quantidade de seguidores do usuário que publicou o *tweet*, número de *retweets* daquela publicação entre outros. O Código 12 apresenta alguma dessas formas de mapeamento e, também, imprime o número de *tweets* originais e o número de *retweets* que representam a totalidade das publicações coletadas.

```

1 tweets[ 'created_at' ] = map(lambda tweet: time.strptime( '%Y-%m-%d %H:%M:%S'←
2     , time.strptime(tweet[ 'created_at' ], '%a %b %d %H:%M:%S +0000 %Y')) , ←
3     tweets_data)
4 tweets[ 'user' ] = map(lambda tweet: tweet[ 'user' ][ 'screen_name' ], ←
5     tweets_data)
6 tweets[ 'user_followers_count' ] = map(lambda tweet: tweet[ 'user' ][ '←
7     followers_count' ], tweets_data)
8 tweets[ 'retweet_count' ] = map(lambda tweet: tweet[ 'retweet_count' ], ←
9     tweets_data)
10 tweets[ 'favorite_count' ] = map(lambda tweet: tweet[ 'favorite_count' ], ←
11     tweets_data)
12
13 tweets[ 'text' ] = map(lambda tweet: tweet[ 'text' ].encode('utf-8'), ←
14     tweets_data)
15 tweets[ 'lang' ] = map(lambda tweet: tweet[ 'lang' ], tweets_data)
16 tweets[ 'Location' ] = map(lambda tweet: tweet[ 'place' ][ 'country' ] if tweet[ ←
17     'place' ] != None else None, tweets_data)
18
19 list_of_original_tweets = [element for element in tweets[ 'text' ].values ←
20     if not element.startswith('RT')]
21
22 print "Número de Tweets originais : " + str(len(list_of_original_tweets))
23
24 list_of_retweets = [element for element in tweets[ 'text' ].values if ←
25     element.startswith('RT')]

```

```
16 print "Numero de Retweets : " + str(len(list_of_retweets))
```

### CÓDIGO 12: Extração de outras entidades dos tweets

O Código 13 aproveita o mapeamento das entidades e define uma função para gerar um gráfico em barras que representa o número de *tweets* que alguns usuários realizaram dentro da *hashtag* utilizada para a coleta de dados, ou seja, a *hashtag* #ImpeachmentDay.

```
1 def plot_tweets_per_category(category , title , x_title , y_title , top_n=5 , ←
2     output_filename="plot.png") :
3     tweets_by_cat = category.value_counts()
4     fig , ax = plt.subplots(figsize=(20,10))
5     ax.tick_params(axis='x')
6     ax.tick_params(axis='y')
7     ax.set_xlabel(x_title)
8     ax.set_ylabel(y_title)
9     ax.set_title(title)
10    tweets_by_cat[:top_n].plot(ax=ax , kind='bar' , color='mediumturquoise'←
11    )
12    fig.savefig(output_filename)
13    fig.show()
14
15    plot_tweets_per_category(tweets[ 'user '],
16                             "#ImpeachmentDay usuarios ativos",
17                             "Usuarios".decode('utf-8'),
18                             "Numero de Tweets".decode('utf-8') , 20)
```

### CÓDIGO 13: Filtro de usuários através de categorias de entidades

Uma outra maneira de apresentar os dados coletados foi através do *Word Cloud* ou nuvem de palavras. O Código 14 apresenta o desenvolvimento de uma *Word Cloud*, onde contabilizou-se o número de aparições de todas as palavras do conteúdo de texto publicado na rede social *Twitter*.

Para gerar uma *Word Cloud* foi necessário realizar um laço de repetição, conforme demonstrado na linha 3, que verificou cada palavra e removeu algumas exceções, por exemplo palavras iniciadas com "@" ou com "RT" em seu conteúdo.

A linha 12 apresenta o uso da biblioteca *nltk* para definir algumas palavras que foram bloqueadas como padrão, por exemplo, artigos definidos e indefinidos, que não são necessários a sua contabilização.

```
1 text = " ".join(tweets[ 'text '].values.astype(str))
2
3 no_urls_no_tags = " ".join([word for word in text.split()
4                             if 'http' not in word
5                             and not word.startswith('@')
6                             and word != 'RT']
```

```

7   ])
8
9 tweet_coloring = imread(path.join("dilma.png"))
10
11 punctuation = list(string.punctuation)
12 stop = nltk.corpus.stopwords.words('portuguese') + punctuation
13 wordcloud = WordCloud(background_color="white", max_words=500, mask=←
14   tweet_coloring,
15   stopwords=stop, width=1800, height=1400).generate(←
16   no_urls_no_tags)
17
18 plt.figure(figsize=(10,10))
19 image_colors = ImageColorGenerator(tweet_coloring)
20 plt.imshow(wordcloud)
21 plt.axis("off")
22 plt.figure(figsize=(10,10))
23 plt.imshow(tweet_coloring, cmap=plt.cm.gray)
24 plt.axis("off")
25 plt.figure(figsize=(10,10))
26 plt.imshow(wordcloud.recolor(color_func=image_colors))
27 plt.axis("off")
28 plt.show()

```

CÓDIGO 14: Implementação da *Word Cloud*

É possível também utilizar a entidade de localização, apresentada pela API do *Twitter* como *coordinates*, para conseguir as coordenadas de alguns *tweets* e posicioná-las em ambientes de geolocalização possibilitando a criação de um mapa que mostra de qual localidade os *tweets* foram postado, situação esta que permite ao pesquisador verificar qual o estado, região ou país esteve mais ativo durante a coleta de dados.

O Código 15 apresenta o desenvolvimento para este tipo de funcionalidade utilizando a biblioteca *Folium*. No entanto, a forma como o *Twitter* disponibiliza os dados de posicionamento não fornecia a correta localização da postagem. Para tanto, foi necessário inverter a ordem de latitude e longitude disponibilizada pelo *Twitter* para aplicar a correta localização no mapa, conforme apresentado na linha 3.

```

1 coordinate = []
2 for col in tweets2['coordinates'][~tweets2['coordinates'].isnull()]:
3     coord = col['coordinates'][::-1]
4     #     coord = col['coordinates']
5     coordinate.append(coord)
6
7 print coordinate[10]
8
9 m = folium.Map([-14,-53.25], zoom_start=4)
10

```

```

11 for x, text in enumerate(coord_text):
12     folium.Marker(coordinate[x], popup=str(coordinate[x])).add_to(m)

```

#### CÓDIGO 15: Implementação de coordenadas dos tweets

A data de publicação dos *tweets* é identificado pelo atributo "created\_at", onde, através do Código 6 é possível contabilizar o número de *tweets* de um determinado horário e apresentá-los através de um gráfico.

Por padrão, a API do *Twitter* define o horário de criação dos *tweets* em relação ao Tempo Universal Coordenado (UTC) (TWITTER, 2016b). Logo, foi necessário realizar a conversão para contemplar o fuso horário brasileiro. As linhas 4 e 5 especificam as informações do fuso horário, sendo preciso definir, primeiramente, o horário de Greenwich (GMT) e, então, subtrair três horas para considerar o horário de Brasília.

A linha 7 é especificado a contabilização dentro de um intervalo de uma hora e as linhas do Código 6 que se seguem são as funcionalidades para a criação do gráfico.

```

1 tweets[ 'created_at' ] = pd.to_datetime(pd.Series(tweets[ 'created_at' ]))
2
3 tweets.set_index( 'created_at' , drop=False , inplace=True)
4 tweets.index = tweets.index.tz_localize( 'GMT' )
5 tweets.index = tweets.index - DateOffset(hours = 3)
6
7 tweets30s = tweets[ 'created_at' ].resample( '1h' , how= 'count' )
8 tweets30s.head()
9
10 avg = tweets30s.mean()
11
12 vincent.core.initialize_notebook()
13 area = vincent.Area(tweets30s)
14 area.colors(brew= 'Spectral' )
15 area.display()

```

#### CÓDIGO 16: Data da publicação dos tweets

Todos os códigos implementados neste trabalho, exceto o Código 2 e os *scripts* para a limpeza de *dirty data*, utilizaram a funcionalidade *notebook* do *IPython* para serem implementados, onde foi possível apresentar os códigos ao interpretador do Python utilizando um navegador *web*.

Os resultados apresentados durante a implementação são apresentados no capítulo posterior a este, onde serão analisado os resultados encontrados.

## 6 ANÁLISE DOS RESULTADOS

Este capítulo tem como finalidade apresentar os resultados obtidos através das implementações demonstrados no Capítulo 5.

### 6.1 APRESENTAÇÃO DOS RESULTADOS

Após o mapeamento das informações do *DataFrame*, apresentado pelo Código 5 e Código 6, foi construído um gráfico em barras indicando as quantidades de *tweets* em relação aos quatro idiomas mais significativos no conjunto de dados coletados, Gráfico 1.

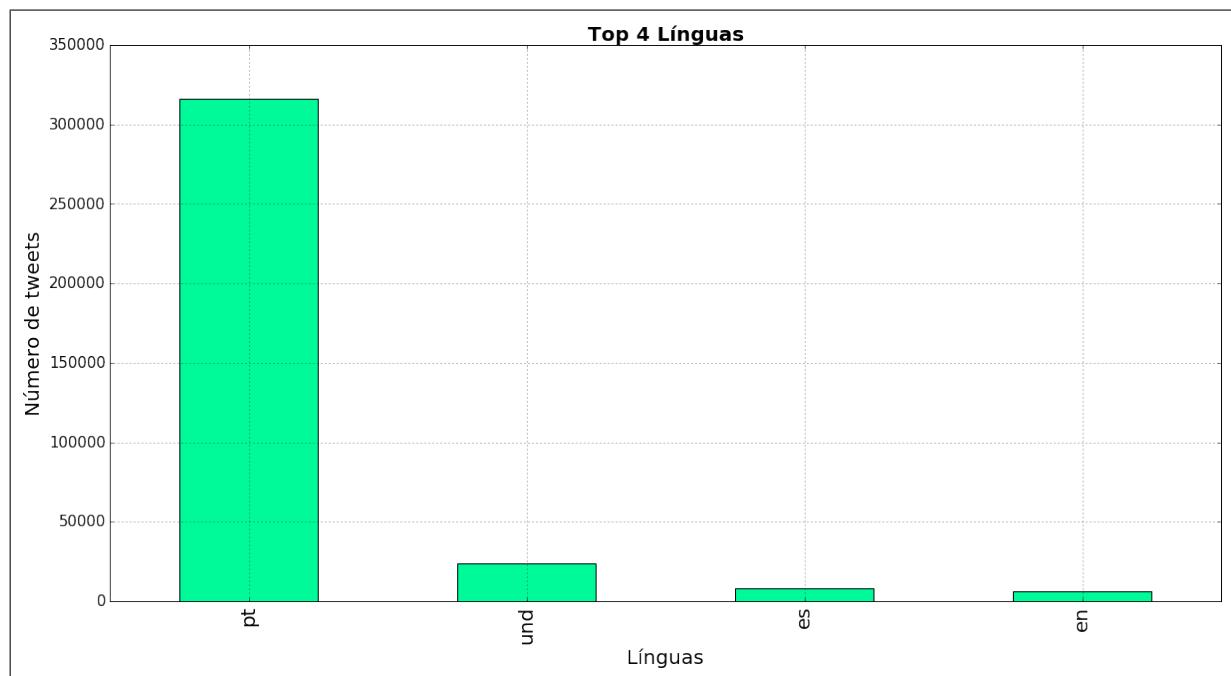


GRÁFICO 1: Idiomas que mais realizaram *tweets*

FONTE: Elaborado pelo autor

É possível verificar nesta figura que existe uma barra com o nome de *und* para especificar o segundo idioma que mais realizou *tweets*. Essa é uma condição em que não foi identificado o idioma de origem e, então, a API do *Twitter* classifica como *undefined*, ou indefinido no português. Essa linguagem indefinida ocorre devido ao *software* em que o usuário está realizando o *tweet*, por exemplo; um navegador *web*, um aplicativo *mobile* do *Twitter* ou algum aplicativo de terceiro que permite realizar ações no *Twitter*. Caso a linguagem não esteja definida nestes ambientes, a API a classifica como indefinida (TWITTER, 2016b).

O Gráfico 2 demonstra, claramente, que a maior número de *tweets* foram publicados do Brasil e apenas uma pequena quantia deles foram realizados nos Estados

Unidos, Argentina, Portugal e Venezuela. É importante notar que nem todos os *tweets* gerados possuem um país de origem, o que se assemelha a situação anterior, onde a API do Twitter os classifica como *undefined*, porém não sendo apresentado no Gráfico 2.

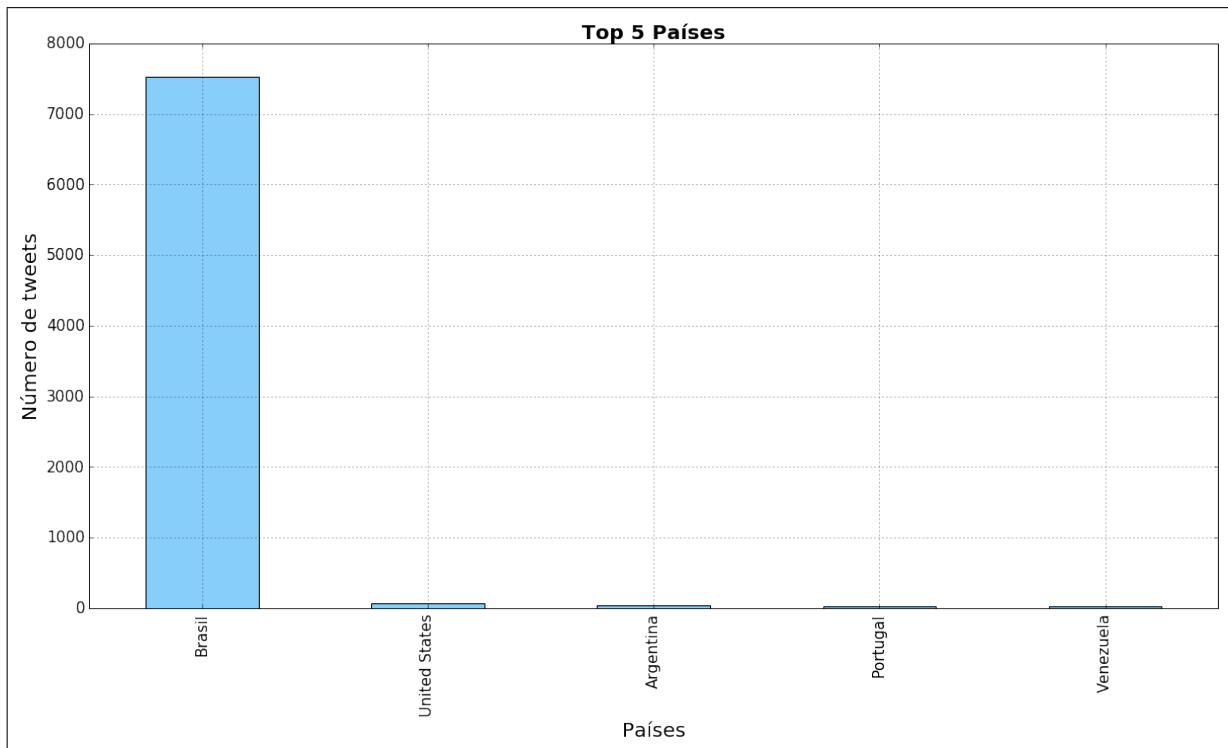


GRÁFICO 2: Países que mais realizaram *tweets*

FONTE: Elaborado pelo autor

Durante a votação no Congresso para a continuação do processo de Impeachment da Presidente Dilma Rousseff, os deputados declaravam o seu posicionamento dizendo se o seu voto era "SIM" para o processo ou "NÃO" caso o contrário.

O Gráfico 3, apresenta a contabilização de palavras "SIM" e "NÃO" e as hashtags #ForaDilma e #NaoVaiTerGolpe em *tweets*. Dentro da hashtag #ForaDilma foi verificado a quantidades de palavras "SIM" isoladas e para a hashtag #NaoVaiTerGolpe verificou-se a disponibilidade da palavra "NÃO". É válido informar que o Código 9 possui um filtro para não contabilizar as palavra "NÃO" que fazem parte da palavra-chave da hashtag #NaoVaiTerGolpe, podendo coletar apenas as palavras isoladas que fazem menções a votação.

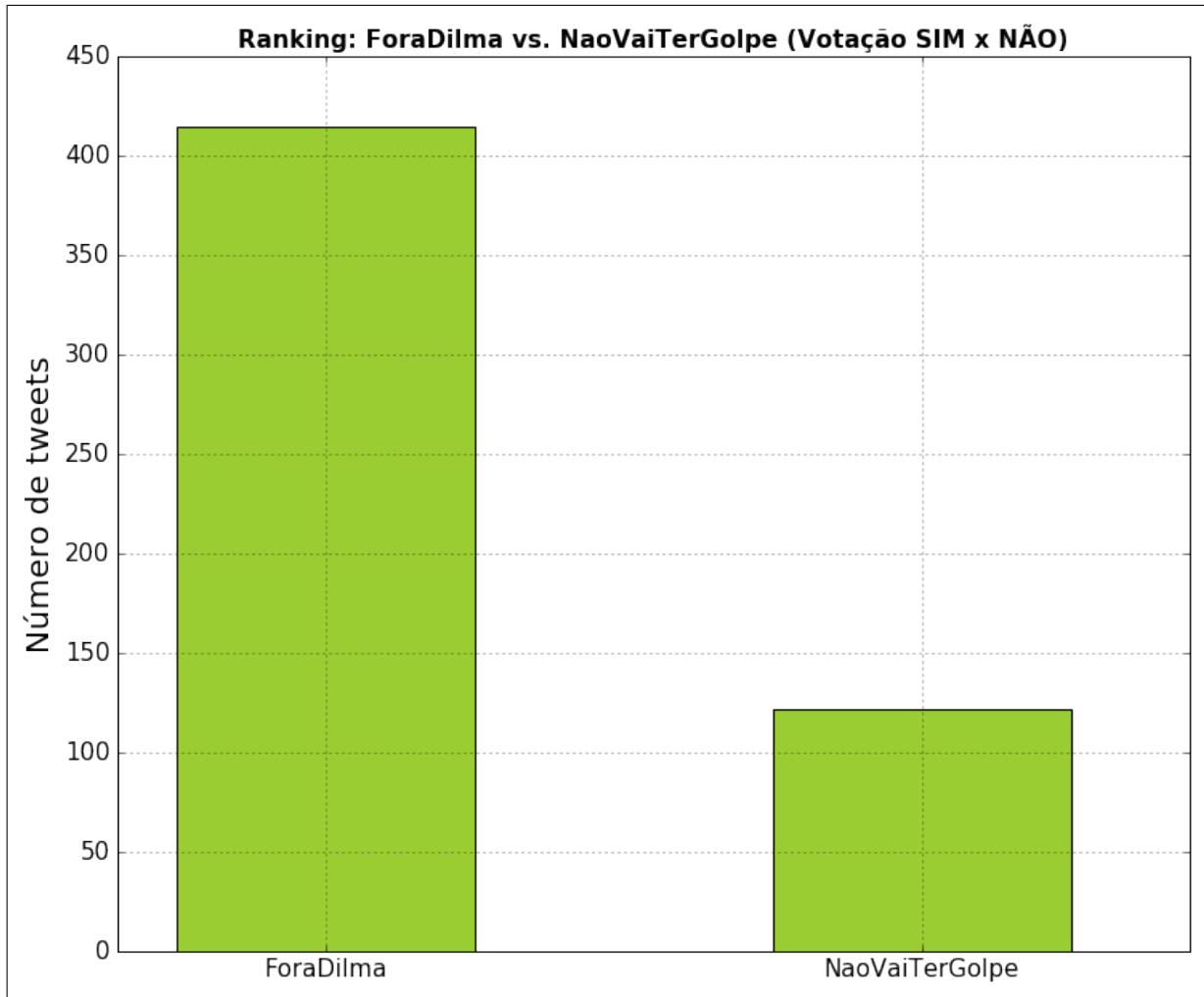


GRÁFICO 3: Número de menções de SIM e NÃO em *tweets*

FONTE: Elaborado pelo autor

Semelhante a solução anterior, é possível contabilizar as *hashtags* e apresentá-las através do gráfico de setores, onde é possível verificar a porcentagem referente aos *tweets* realizados com a *hashtag* #ImpeachmentDay, Gráfico 4.

Nota-se no Gráfico 4, que as *hashtags* #ForaDilma e #TchauQuerida somaram um total de 60,7%, indicando que a maioria dos *tweets* eram de apoio ao processo de Impeachment da Presidente da República, diferentemente dos 34,3% dos *tweets* que é o somatório de #BrasilContraOGolpe e #NaoVaiTerGolpe.

É possível também verificar na legenda o número de *tweets* para cada uma das *hashtags* apresentadas pelo gráfico. Nota-se que a *hashtag* #ImpeachmentDay não foi apresentada pois ela foi utilizada como critério de busca para a coleta destes dados, fazendo com que todos os *tweets* possuam menções a essa *hashtag*.

O Código 11 explicado no capítulo anterior, constrói um gráfico em setores, onde é possível verificar a porcentagem de *tweets* para nomes específicos, Gráfico 5, porém não é possível, ainda, predizer se os argumentos a respeito destas pessoas

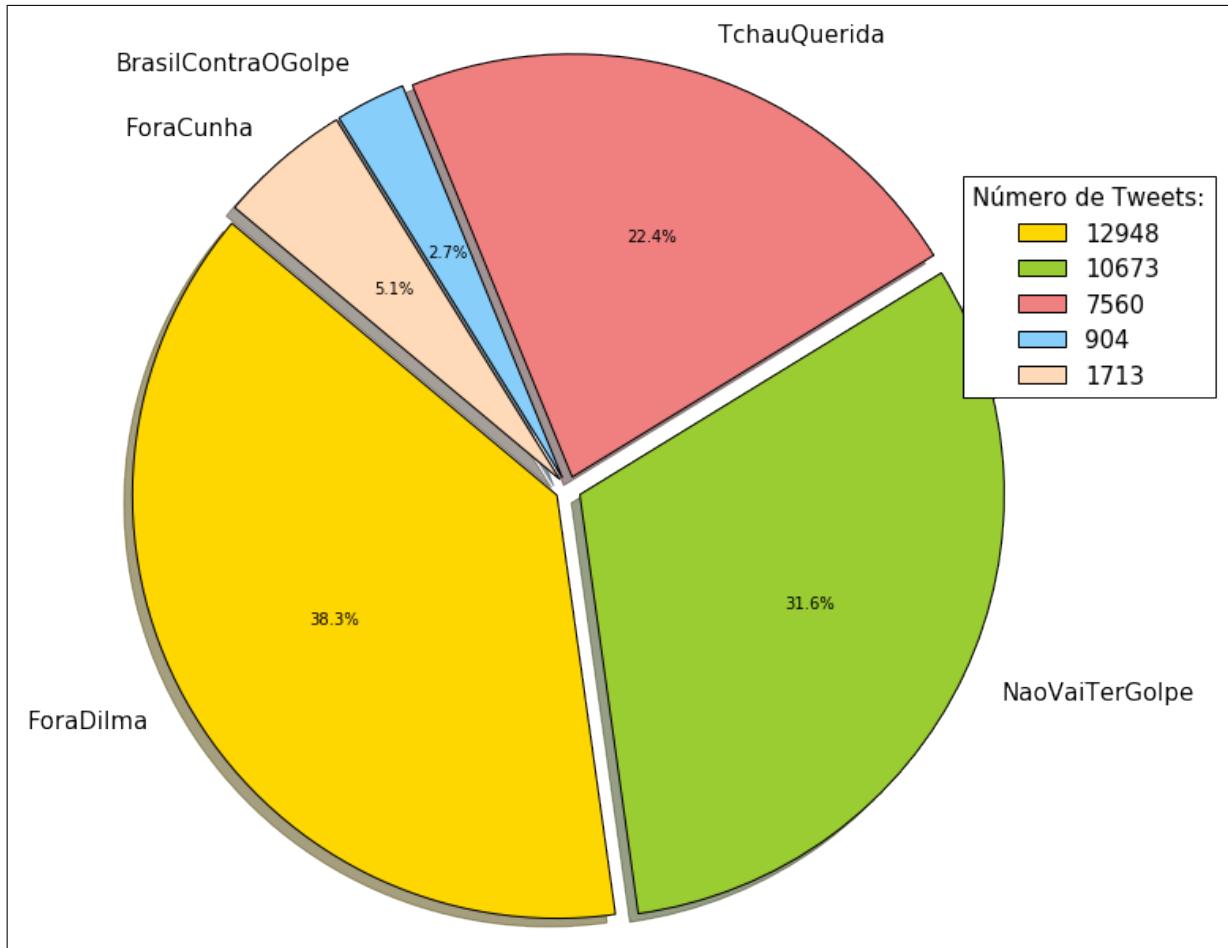


GRÁFICO 4: *Hashtags* com o maior número de *tweets*

FONTE: Elaborado pelo autor

são positivos ou negativos.

O Gráfico 5, demonstra que o nome mais mencionado foi de Eduardo Cunha e com apenas 2618 *tweets* o nome de Michel Temer foi publicado. Até mesmo as menções ao ex-presidente Luiz Inácio "Lula" da Silva recebeu apenas 18,8% do total dos nomes filtrados.

Semelhante ao Gráfico 4, é possível verificar o número de *tweets* em que foi mencionado cada figura política. Porém não é contabilizado o nome de Dilma Rousseff, pois quase todos os *tweets* fazem alguma menção a ela, resultando então em uma porcentagem mínima para os nomes apresentados no Gráfico 5.

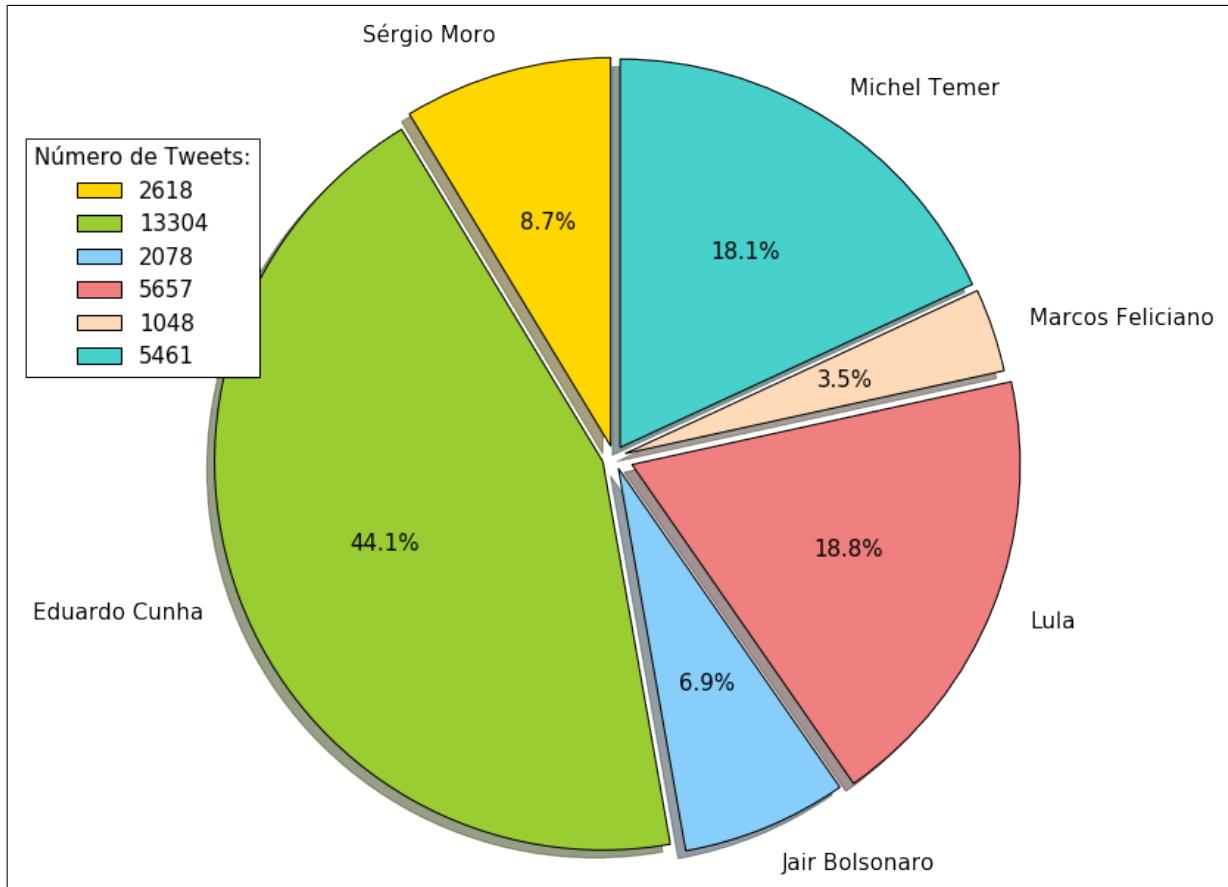


GRÁFICO 5: Gráfico em setores para figuras importantes

FONTE: Elaborado pelo autor

Uma outra maneira de apresentar a contabilização de palavras mais mencionadas é através da *Word Cloud*. Conforme implementado e apresentado no Código 14 é possível visualizar algumas palavras-chaves, onde as mais citadas apresentam um tamanho de fonte maior, assim como as menos citadas são representadas através das palavras menores.

Através da Figura 13, é possível verificar claramente palavras como "ImpeachmentDay", "Dilma", "Cunha", "Brasil", "ForaDilma", "impeachment" entre outras.

As cores disponibilizadas na Figura 13 são da biblioteca padrão do *Word Cloud* não representando nenhum outro indício a não ser uma funcionalidade aleatória para melhor visualizar as palavras.

A biblioteca *Word Cloud* na linguagem Python, permite que o desenvolvedor utilize alguma imagem como modelo. Onde as cores e o posicionamento das palavras são disponibilizados na tentativa de representar a imagem modelo.

A Figura 14 ilustra este tipo de funcionalidade apresentando a imagem modelo uma foto da Presidente suspensa Dilma Rousseff. Ao lado direito é demonstrado o *Word Cloud* gerado, que representa cores com tonalidades semelhantes a imagem



FIGURA 13: *Word Cloud* das 500 palavras mais mencionadas

FONTE: Elaborado pelo autor

modelo.

Da mesma maneira que a Figura 13 apresenta as palavras com mais menções, o *Word Cloud* da Figura 14 também disponibiliza as 500 palavras e *hashtags* mais publicadas coletadas durante o dia da votação no Congresso brasileiro.

Exercendo o uso dos atributos de coordenadas, implementados no Código 15 utilizando a biblioteca *Folium*, é possível verificar a distribuição dos *tweets* geograficamente. As informações de latitude e longitude permitem que cada *tweet* seja representado por um marcador conforme ilustrado pela Figura 15.

A coleta de dados resultou em um arquivo JSON com 358.293 *tweets*, porém apenas 470 destas publicações possuem informações de coordenadas. Este total de *tweets* com atributos de posições geográficas representa apenas míseros 0,13% da totalidade dos dados, não sendo possível, então, afirmar que é um padrão ou uma tendência.



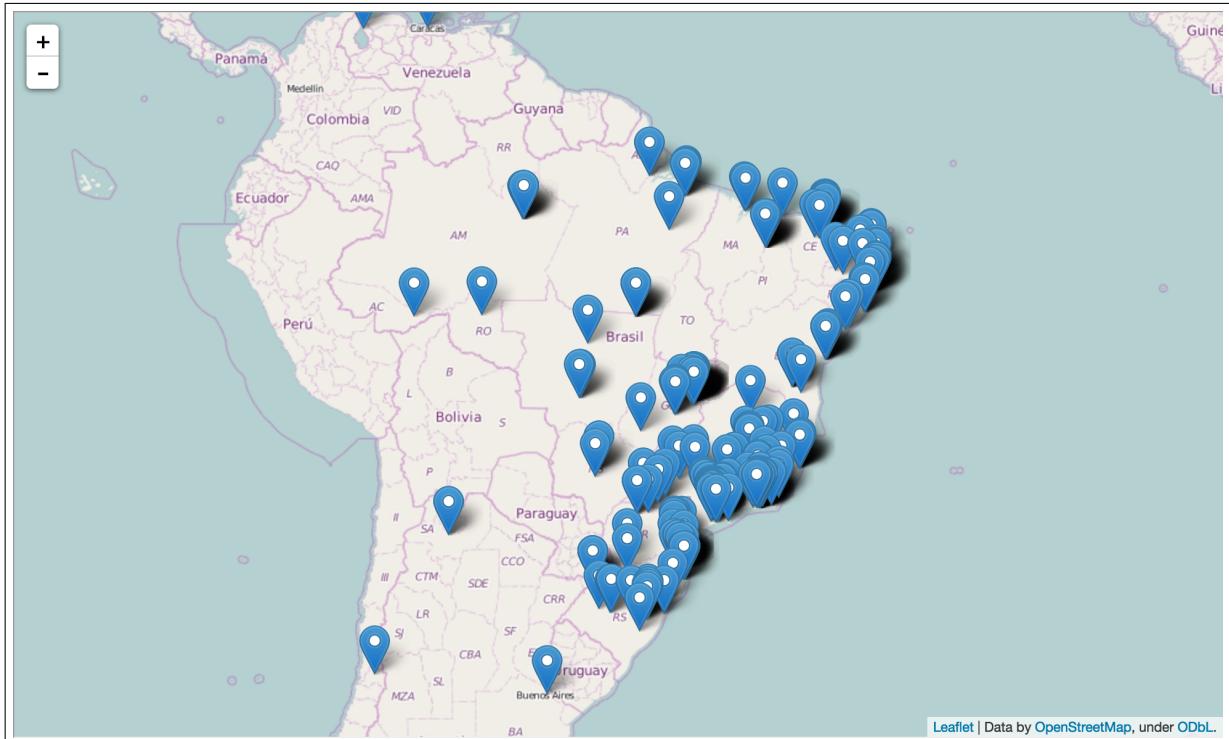
FIGURA 14: *Word Cloud* utilizando imagem como modelo

FONTE: Elaborado pelo autor

A utilização dos dados geográficos, representados pela Figura 15, demonstra mais um das funcionalidades que as bibliotecas da linguagem Python disponibiliza para a apresentação dos dados. Através do *IPython* é possível navegar através do mapa, ter interações com os marcadores e se distanciar ou aproximar para verificar detalhes sobre as localidades da publicação.

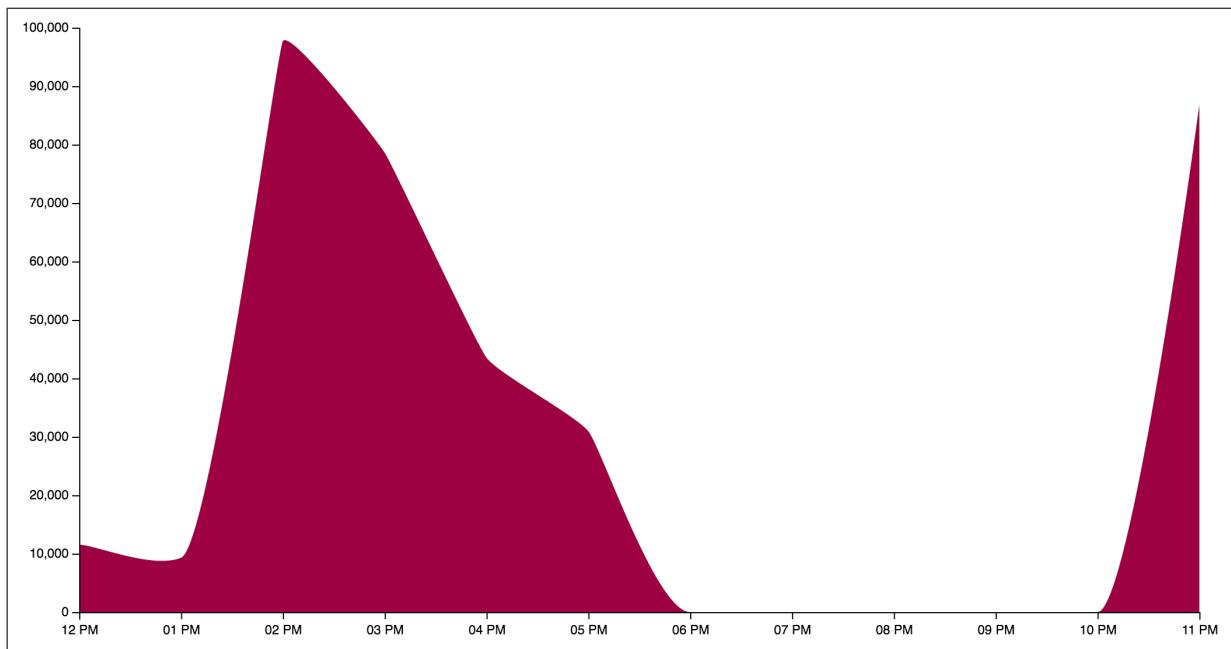
O Gráfico 6 apresenta a quantidade de publicações de *tweets* por hora dentro das 12 horas de coletas.

É notado que no período das 14 horas (02 PM) ocorreu o maior número de *tweets*, chegando a quase 100 mil publicações. O horário da votação no congresso começou às 16 horas, horário de Brasília, momento em que teve as maiores baixas apresentadas pelo Gráfico 6. A ascensão dos *tweest* volta a acontecer após às 22 horas, próximo ao horário de conclusão e resultado da votação.



**FIGURA 15:** Distribuição geográfica de *tweets*

FONTE: Elaborado pelo autor



**GRÁFICO 6:** Publicação de *tweets* por hora

FONTE: Elaborado pelo autor

As bibliotecas que Python dispõe, permitiu a apresentação de gráficos em colunas, setores e áreas. Também a visualização de palavras mais mencionadas através do *Word Cloud* e a distribuições dos *tweets* através de coordenadas para ser apresentadas em mapa.

## 7 CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

A proposta dessa dissertação foi de utilizar técnicas e algoritmos de *data mining* para analisar e minerar os dados provenientes da rede social *Twitter*, utilizando os recursos e bibliotecas da linguagem de programação Python. Para tanto, foi necessário o estudo de conceitos de KDD e *data mining*, implementar técnicas utilizando bibliotecas da linguagem em estudo e apresentar informações úteis para possíveis interpretações.

Durante o estudo de *data mining*, foi determinado que a sua definição consiste em um processo de descoberta de padrões interessantes e conhecimentos de um vasto conjunto de dados. Sendo necessário, então, ferramentas que auxiliem na coleta, limpeza, seleção e apresentação dos dados.

Para a coleta e limpeza dos dados a linguagem Python se tornou extremamente útil, ao permitir o desenvolvimento de um código que se autenticou com o *Twitter* e a utilização da API de *streaming* para coletar os dados que continham a *hashtag* #ImpeachmentDay.

## REFERÊNCIAS

- BIRD, S. Nltk: the natural language toolkit. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the COLING/ACL on Interactive presentation sessions.** [S.I.], 2006. p. 69–72.
- BRACHMAN, R. J. et al. The process of knowledge discovery in databases. 1996. Acesso em 23 de outubro de 2015. Disponível em: <<https://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-001.pdf>>.
- BRAIN, S. **Twitter Statistics.** 2016. Acesso em 20 de abril de 2016. Disponível em: <<http://www.statisticbrain.com/twitter-statistics/>>.
- DEMO, P. **Avaliação qualitativa.** [S.I.]: Autores Associados, 1991. v. 25.
- DIENER, M. **Python Geospatial Analysis Cookbook.** [S.I.]: Packt Publishing Ltd, 2015.
- FAYYAD, U. et al. From data mining to knowledge discovery in databases. 1996–a. Acesso em 23 de outubro de 2015. Disponível em: <<http://www.csd.uwo.ca/faculty/ling/cs435/fayyad.pdf>>.
- FAYYAD, U. et al. Advances in knowledge discovery in data mining. 1996–b.
- FIELDING, R. T. Architectural styles and the design of network-based software architectures. 2000.
- GIL, A. C. Como elaborar projetos de pesquisa. **São Paulo**, v. 5, p. 61, 2002.
- GOLDENBERG, M. **A arte de pesquisar.** [S.I.]: Editora Record, 2002.
- HAN, J. et al. **Data Mining: Concepts and Techniques.** [S.I.]: Elsevier, 2012.
- HUBERMAN, B. A.; ROMERO, D. M.; WU, F. Social networks that matter: Twitter under the microscope. **Available at SSRN 1313405**, 2008.
- KALDERO, N. **Why is Python a language of choice for data scientists?** 2015. Acesso em 27 de outubro de 2015. Disponível em: <<http://qr.ae/RkleiB>>.
- KWAK, H. et al. What is twitter, a social network or a news media? In: ACM. **Proceedings of the 19th international conference on World wide web.** [S.I.], 2010. p. 591–600.
- LEMOS, E. P. **Análise de crédito bancário com o uso de data mining: redes neurais e árvores de decisão.** Tese (Doutorado) — Universidade Federal do Paraná, 2003.
- MCKINNEY, W. **Python for Data Analysis.** [S.I.]: O'Reilly, 2013.
- NAVEGA, S. Princípios essenciais do data mining. 2002.
- PIPINO, L.; KOPCSO, D. Data mining, dirty data, and costs. In: **IQ.** [S.I.: s.n.], 2004. p. 164–169.

PYTHON. **Python 3.4.3 Documentation**. 2015. Acesso em 21 de novembro de 2015. Disponível em: <<https://docs.python.org/3.4/>>.

RUP, I. Rational unified process. **Engenharia de Software**, p. 52, 2003.

RUSSELL, M. A. **Mining the Social Web**. [S.I.]: O'Reilly Media, 2013. ISBN 978-1-449-36761-9.

SFERRA, H. H.; CORREA, A. M. C. J. Conceitos e aplicações de data mining. 2003.

SILVA, E. L. da; MENEZES, E. M. Metodologia da pesquisa e elaboração de dissertação. **UFSC, Florianópolis, 3a. edição**, 2001.

SILVA, M. P. da; BOSCAROLI, C.; PERES, S. M. Análise de logs da web por meio de técnicas de data mining. 2003.

STEINER, M. T. A. et al. Data-mining como suporte à tomada de decisões-uma aplicação no diagnóstico médico. **XXXVI SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL,"O IMPACTO DA PESQUISA OPERACIONAL NAS NOVAS TENDÊNCIAS MULTIDISCIPLINARES**, v. 23, p. 96–107, 2004.

TWEETPY, D. **Biblioteca Tweepy - 3.5.0**. 2009. Acesso em 03 de abril de 2016. Disponível em: <<http://tweepy.readthedocs.org/en/v3.5.0/index.html>>.

TWITTER. **Tweet Example**. 2016. Acesso em 24 de abril de 2016. Disponível em: <<https://twitter.com/unixstickers/status/724338974043574272>>.

TWITTER. **Twitter Documentation**. 2016. Acesso em 21 de abril de 2016. Disponível em: <<https://dev.twitter.com/overview/documentation>>.

WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. **REST in Practice - Hypermedia and Systems Architecture**. [S.I.]: O'Reilly, 2010.

WIENER, E. Matplotlib tools. 2014. Acesso em 27 de novembro de 2015. Disponível em: <<https://wiki.ucar.edu/display/ral/Matplotlib+Tools>>.

YANG, S.; KAVANAUGH, A. L. Collecting, analyzing and visualizing tweets using open source tools. In: ACM. **Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times**. [S.I.], 2011. p. 374–375.