Universidade Federal
do Rio de Janeiro
Escola Politécnica

PACOTE DE CONSULTA A DADOS CENSITÁRIOS DO CENSO DEMOGRÁFICO DE 2010 DO IBGE UTILIZANDO LINGUAGEM PYTHON

Pedro Henrique Gueiros Samú

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Flávio Luis de Mello

Rio de Janeiro

Junho de 2018

PACOTE DE CONSULTA A DADOS CENSITÁRIOS DO CENSO DEMOGRÁFICO DE 2010 DO IBGE UTILIZANDO LINGUAGEM PYTHON

Pedro Henrique Gueiros Samú

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Pedro Henrique Gueiros Samú

Orientador:

Flávio Luis de Mello, D. Sc.

Examinador:

Heraldo Luis Silveira de Almeida, D. Sc.

Examinador:

Manoel Villas Boas Junior, M. Sc.

Rio de Janeiro – RJ, Brasil

Junho de 2018

Declaração de Autoria e de Direitos

Eu, Pedro Henrique Gueiros Samú CPF 145.592.127-08, autor da monografia *PACOTE DE CONSULTA A DADOS CENSITÁRIOS DO CENSO DEMOGRÁFICO DE 2010 DO IBGE UTILIZANDO LINGUAGEM PYTHON*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.

Pedro Henrique Gueiros Samú

Pedro Henrique Gueiros Samú

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

AGRADECIMENTO

Agradeço a meus pais Elza Maria e Flávio por tornarem possíveis todas as minhas conquistas, meu irmão João Paulo por seu companheirismo e apoio diário, meus avós Marlene e Jerônimo e minha bisavó Elza pelo apoio imensurável e incondicional dado a mim ao longo desses anos, meus avós Dileta e João, meu padrinho Antônio Augusto por ser um eterno exemplo em minha vida, minha namorada Catharine Rocha e a Amora por estarem sempre presentes nos momentos difíceis que enfrentei durante essa jornada trazendo felicidade e amor ao meu dia-a-dia, meus companheiros de curso Felipe Batista, Lucas Adorno e Gabriel Pimentel que junto a mim muito aprenderam como também muito ensinaram, meus amigos Augusto Senna, Vitor Hugo, José Felipe, João Renato, Maria Beatriz, Gabriel Loureiro, Duncan MacFarlane e André Waldherr e todos que de alguma forma estiveram presentes nesse período tão importante da minha vida.

RESUMO

O presente trabalho tem como objetivo desenvolver uma ferramenta que permita um usuário obter informações de dados censitários oriundos do Censo Demográfico de 2010 realizado pelo IBGE de forma simples e eficaz. Seu desenvolvimento foi pautado na criação de módulos de funções na linguagem de programação Python para executar a extração, transformação e consulta dos dados presentes em repositórios públicos disponibilizados nos servidores do IBGE, simplificando este processo através de métodos de automatização. Por não apresentar uma indexação que permita a relação direta entre dados cadastrais, como o endereço de um usuário, e dados de setores censitários foram elaboradas duas abordagens para sanar esse problema, uma delas entregando um resultado com maior exatidão, porém requerendo uma conexão com a Internet e outra menos exata, mas com a possibilidade de execução em ambientes offline. Por fim foi gerada uma documentação para auxiliar o uso da ferramenta.

Palavras-Chave: IBGE, setor censitário, censo demográfico, Python, ETL.

ABSTRACT

The present work aims to develop a tool that allows a user to obtain information from census data presented on the Demographic Census of 2010 conducted by IBGE in a simple and effective manner. The development was characterized by the creation of modules of functions using Python programming language to execute the extraction, transformation and query of the data available in IBGE's server public repositories, simplifying this process through automation methods. Due to the lack of a viable indexing that allows the direct relation between registration data, such as a user address, and census sector data two approaches were developed to deal with this problem, one of them delivering a result with greater accuracy, however requiring an Internet connection and the other less accurate, but achievable in offline environments. Lastly a documentation was generated to help the tool's usability.

Key-words: IBGE, census sector, demographic census, Python.

SIGLAS

API	– Application Programming Interface
CSV	– Comma-separated Values
ETL	– Extract, Transform and Load
FTP	– File Transfer Protocol
GIS	– Geographic Information System
GPS	– Global Positioning System
IBGE	– Instituto Brasileiro de Geografia e Estatística
IDH	– Índice de Desenvolvimento Humano
JSON	– JavaScript Object Notation
NAN	– Not a Number
RJ	– Rio de Janeiro
SP	– São Paulo
UF	– Unidade Federativa
UFRJ	– Universidade Federal do Rio de Janeiro
Unesco	– Organização das Nações Unidas para a Educação, a Ciência e a Cultura
XML	– Extensible Markup Language

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O tema do trabalho é a criação de um pacote de funções na linguagem Python que possibilite a consulta de dados censitários do Censo Demográfico do ano de 2010 realizado pelo IBGE. Neste sentido, o problema a ser resolvido é extrair dados disponíveis nos repositórios públicos disponíveis no servidor do IBGE e manipulá-los de forma a obter a implementação de uma ferramenta que realize a consulta a tais repositórios de maneira simples e eficaz.

1.2 – Delimitação

O trabalho contempla a criação de uma ferramenta de consulta que posteriormente poderá integrar demais sistemas, sendo assim, não possui em seu escopo uma etapa responsável por gerar informações a partir dos dados processados. O resultado do trabalho será oferecido de forma que a consulta seja feita independente da modelagem de dados do sistema usuário.

Neste sentido, será adotado como produto resultante do trabalho um pacote Python de funções com saída na forma de variáveis e arquivos de texto no formato CSV. Os dados utilizados serão obtidos de fontes públicas disponíveis online, especificamente relacionadas ao Censo 2010 realizado pelo IBGE.

1.3 – Justificativa

O surgimento de novas tecnologias de inteligência artificial e o desenvolvimento das já existentes, acompanhado da evolução na capacidade de processamento dos computadores modernos fez com que uma etapa envolvendo o uso de um agente inteligente se tornasse cada vez mais presente, senão essencial, em sistemas de computadores. Resultados mais eficientes e confiáveis garantem a aplicação de tais técnicas, complementando o trabalho humano e muitas vezes prevalecendo sobre o mesmo se levado em consideração o impacto no objetivo final do sistema.

O principal recurso utilizado por um agente inteligente para gerar informação é o conjunto de dados disponível para análise. Seja o agente uma aplicação de reconhecimento de fala ou facial, redes neurais ou sistemas especialistas, em todo e qualquer caso o resultado está diretamente relacionado ao conjunto de dados processado. A partir desta relação, pode-se concluir que a etapa referente à consulta de dados é um fator crítico para o funcionamento, validade e desempenho de um sistema.

Tradicionalmente sistemas procuram utilizar dados que atendam diretamente a necessidade do cliente ou usuário alvo, descartando qualquer tipo de redundância ou conteúdo em excesso que poderia vir a impactar negativamente no desempenho geral. Em uma segunda etapa pode-se enriquecer o conjunto de dados com agregação de informações provenientes de fontes diversas, tais como dados censitários.

Muitas vezes os dados a serem processados por um sistema não se encontram devidamente estruturados para a consulta. São comuns situações em que os dados sejam adquiridos de fontes distintas, ocasionando em padrões e formatos diferentes em um mesmo sistema. Torna-se assim vantajosa a inclusão de uma etapa anterior à análise dos dados, isto é, uma etapa responsável pela coleta, transformação e carga dos dados.

1.4 – Objetivos

O objetivo do trabalho é obter os dados censitários referentes ao Censo 2010 provenientes do repositório do IBGE e estruturá-los de forma a viabilizar a consulta de

forma simples aprimorando o desempenho de sistemas que busquem informações desta fonte de dados. Os objetivos específicos do trabalho são:

- Compreender o conteúdo do Censo IBGE 2010
- Compreender a estrutura do repositório de dados censitários do IBGE
- Definir quais fontes serão relevantes para o projeto
- Extrair os dados do repositório
- Fazer a validação dos dados obtidos
- Fazer a transformação dos dados
- Fazer a validação dos dados transformados
- Definir quais tipos de consulta serão feitos
- Criar as funções de consulta
- Carregar o objeto final do projeto
- Gerar a documentação do objeto final do projeto

1.5 – Metodologia

Para obter o êxito do projeto o mesmo foi dividido em fases distintas, uma fase responsável pelo estudo e compreensão do conteúdo disponível nos repositórios do IBGE, uma destinada à manipulação dos dados obtidos e finalmente uma fase final visando a forma como o resultado será disponibilizado.

A primeira parte do trabalho visa compreender o objeto de estudo, apontando quais fatores são relevantes para serem considerados nas etapas seguintes. Em seguida é realizada a organização dos dados obtidos utilizando bibliotecas Python para automatizar o processo, para então viabilizar a aplicação de técnicas ETL (Extract Transform and Load) através de bibliotecas de análise e manipulação de dados em Python auxiliadas do aplicativo Jupyter Notebook com o objetivo de validar, manipular e organizar os dados. Por final o resultado obtido será carregado em um objeto final.

1.6 – Descrição

No capítulo 2 é realizado um estudo do Censo Demográfico no Brasil. São apresentadas informações históricas e técnicas desse processo vital para o desenvolvimento do país. Neste capítulo é abordada de forma aprofundada o principal atributo que será utilizado ao longo do trabalho, o setor censitário. Também são abordados a fundo os repositórios disponíveis nos servidores do IBGE que possuem o conjunto de dados que será utilizado no trabalho.

Para a execução do projeto foram adotadas diversas ferramentas e técnicas de ciência de dados, por este motivo no capítulo 3 é realizada uma breve discussão sobre o assunto e as respectivas ferramentas utilizadas, principalmente bibliotecas de Python.

O capítulo 4 descreve a implementação do trabalho em si, discutindo o problema a ser resolvido, as abordagens encontradas para a sua solução e as etapas percorridas para o êxito do projeto. Também é tratada nesse capítulo a criação da documentação do projeto.

No capítulo de conclusão é feita uma análise do trabalho como todo e do resultado obtido.

Capítulo 2

Dados Censitários do IBGE

2.1 – Censo Demográfico no Brasil

Os primeiros registros oficiais de estimativas populacionais no Brasil datam de 1799 e 1765 nas cidades do Rio de Janeiro e São Paulo respectivamente, até então esses dados eram levantados por autoridades e integrantes da Igreja. No ano de 1846 é oficialmente estabelecido um censo demográfico responsável por estimar a população do país, sendo esse aprovado em 1850 e tendo seu início em 1852. Devido a questões de intolerância racial esse censo nunca veio a ser concretizado [1].

Em 1870 foi novamente estipulada a execução de um censo, definindo um intervalo de dez anos entre cada censo realizado. Finalmente em 1872 foi realizado o primeiro censo nacional no Brasil nomeado Recenseamento da População do Império do Brasil. Nas próximas décadas o intervalo de dez anos entre os censos viria a ser continuamente desrespeitado. [1]

Após a criação do Instituto Brasileiro de Geografia e Estatística em 1936 o processo de recenseamento toma uma nova cara. O intervalo de dez anos entre censos passa a ser respeitado, com o primeiro censo aplicado pelo IBGE acontecendo em 1940, abrangendo parâmetros demográficos socioeconômicos muito além da contagem da população. [1]

Com o passar dos anos cada vez mais os censos realizados pelo IBGE vem incorporando técnicas de informatização, facilitando a coleta, armazenamento, distribuição e estudo dos dados obtidos. Em 2011 o instituto é premiado pela Unesco motivado pelas inovações tecnológicas utilizadas para a realização do censo de 2010 [2].

Em 2010 o IBGE realizou o XII Censo Demográfico, destinando R$1,677 bilhão para a execução dessa operação. Mais de 190 mil recenseadores visitaram 67,7 milhões

de domicílios nos 5565 municípios brasileiros. Foram gastos R$82 milhões para a aquisição de 150 mil smartphones adaptados para o processo de recenseamento, além de outros 70 mil computadores de mãos contando com tecnologia GPS que haviam sido utilizados para a contagem da população em 2007 [3].

Em 2007 o IBGE iniciou o planejamento do Censo 2010, em 2009 realizou o Censo Experimental e entre 2009 e 2010 realizou a aquisição do equipamento, recrutamento e treinamento das equipes de coleta e supervisão. No dia 1º de agosto de 2010 deu início à coleta dos dados com duração estimada de 3 a 4 meses. Em dezembro de 2010 foram divulgados os primeiros resultados contendo informações de cerca de 314 mil setores censitários [4].

As dimensões exatas do Censo como divulgado pelo IBGE são [5]:

- Universo recenseado: todo o Território Nacional;
- Número de municípios: 5.565 municípios;
- Número de domicílios: 67.569.688 de domicílios;
- Número de setores censitários: 314.018 setores censitários;
- Pessoal contratado e treinado: cerca de 240 mil pessoas (coleta, supervisão, apoio e administrativo);
- Orçamento: aproximadamente R$ 1,4 bilhão;
- Tecnologia: centenas de computadores em rede nacional, rede de comunicação em banda larga e 220 mil computadores de mão equipados com receptores de GPS;
- Unidades executoras: 27 unidades estaduais, 7 mil postos de coleta informatizados e 1.283 Coordenações de Subárea.

2.2 – Setor Censitário

De acordo com o Manual do Recenseador [6] “Setor Censitário é a unidade de controle cadastral formada por área contínua, integralmente contida em área urbana ou rural, cuja dimensão, número de domicílios e de estabelecimentos permitem ao

Recenseador cumprir suas atividades em um prazo determinado, respeitando o cronograma de atividades.”.

Tendo em vista a grande dimensão territorial do país e, por conseguinte sua grande quantidade de subdivisões territoriais, também levando em consideração fatores de disparidade social e grande densidade demográfica em pequenas regiões torna-se vantajosa a criação de novas divisões setoriais, visando contemplar tais parâmetros sociais e demográficos que as demais unidades territoriais não contemplam.

Para o Censo 2010 foram consideradas as seguintes divisões políticas do território:

- Unidades da Federação (total de 27) – são os estados, criados por lei federal, e o Distrito Federal;
- Municípios (total de 5565) – dividem integralmente os estados em áreas menores, criados por legislação estadual;
- Distritos (total de 10138) – dividem integralmente os municípios em áreas menores, criados por legislação municipal. Todo município tem, pelo menos, um distrito;
- Subdistritos (total de 489) – dividem os distritos em unidades menores, criadas por legislação municipal. Geralmente, são estabelecidos apenas em algumas grandes cidades para subdividir distritos de grande população ou extensão.

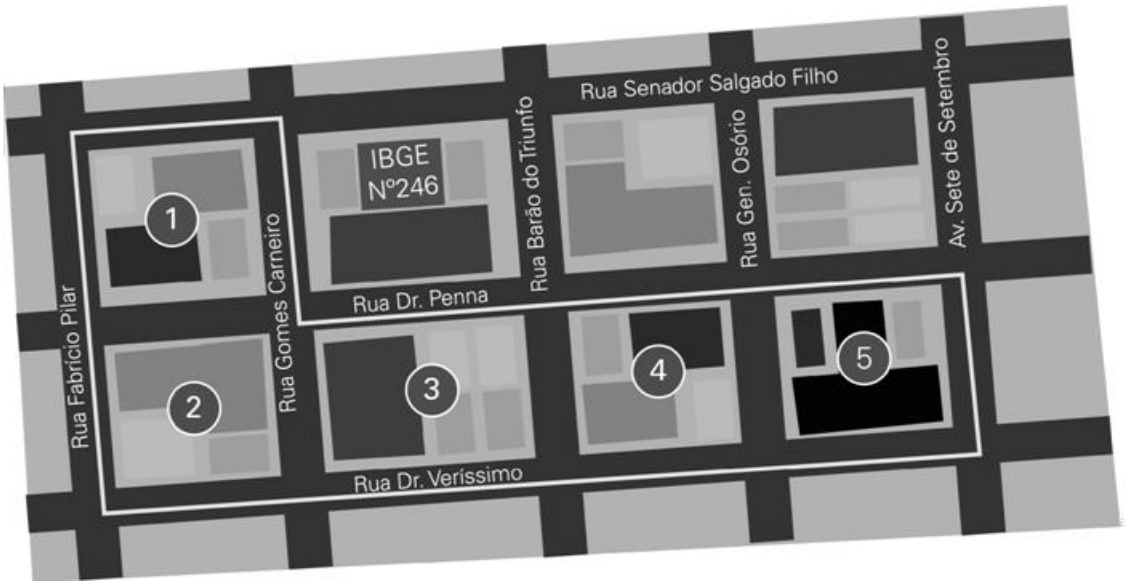
Além disso o território municipal é dividido em duas áreas distintas:

- Área urbana – área interna ao perímetro urbano de uma cidade ou vila. Para as cidades ou vilas onde não existe legislação que regulamente essas áreas, é estabelecido um perímetro urbano para fins da coleta censitária, cujos limites são aprovados pelo prefeito local; e
- Área rural – área externa ao perímetro urbano. Alguns poucos municípios não possuem área rural, sendo, portanto, integralmente urbanos.

O setor censitário entra como uma subdivisão inferior a todas as demais subdivisões respeitando seus devidos limites territoriais, ou seja, um setor censitário está sempre integralmente contido em um único município, um único distrito, um único subdistrito, e em uma única situação (urbana ou rural).

Torna-se notória a necessidade dessa subdivisão ao avaliarmos a realidade da cidade do Rio de Janeiro como exemplo. De acordo com o IBGE os valores de IDH dos bairros da Zona Sul do Rio de Janeiro figuram acima de 0,952, indicando alto índice de desenvolvimento humano, fato que não se reflete na realidade urbana da cidade aonde habitações de população de baixa renda contrastam com condomínios de alta renda no mesmo bairro, muitas vezes na mesma rua.

Na figura 2.1 é apresentado um exemplo gráfico de setor e o mapa disponibilizado pelo IBGE para guiar o recenseador durante a coleta de dados.



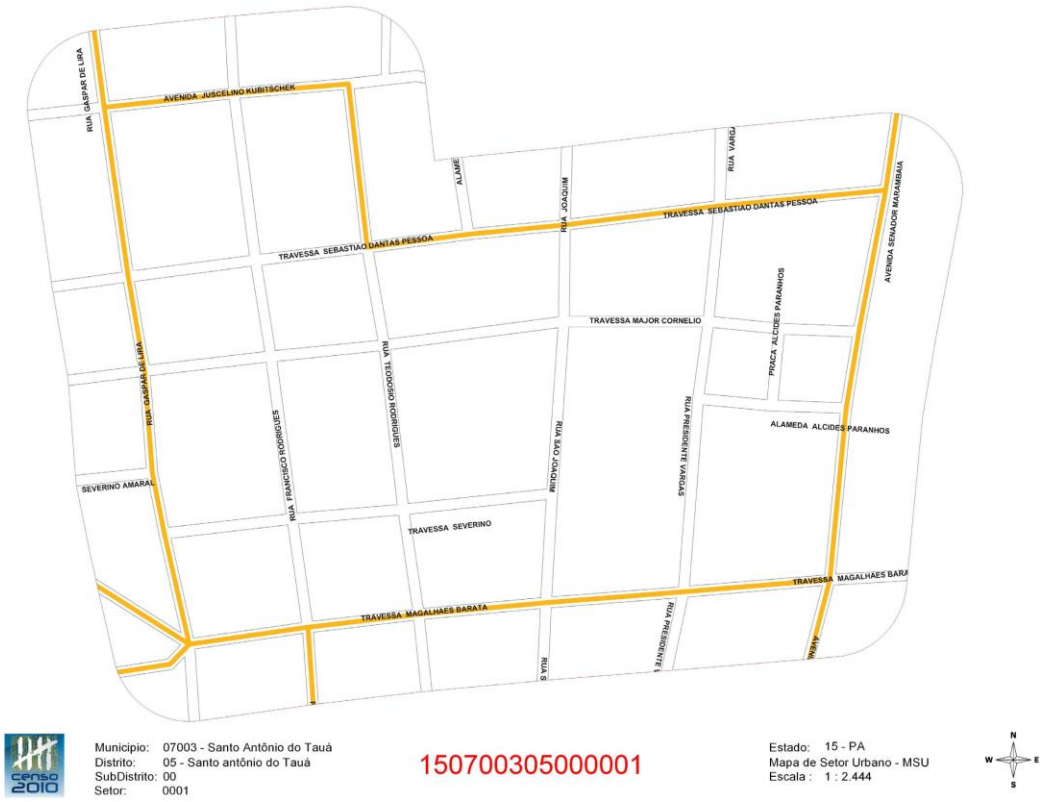


Figura 2.1: Exemplo de setor censitário e mapa do setor de código 150700305000001 utilizado pelo recenseador para auxiliá-lo na coleta de dados [6]

Cada setor censitário definido pelo IBGE recebe uma numeração de 15 dígitos contendo informações sobre o território o qual esse está inserido. Para o setor de número 150700305000001 tem-se, por exemplo, a codificação ilustrada na Figura 2.2.

UF	Município	Distrito	Subdistrito	Setor
15	07003	05	00	0001

Figura 2.2: Numeração do setor censitário [6]

O perímetro do setor é definido a partir das faces que o delimitam, onde faces são trechos de logradouros que compõem o setor. Comumente em áreas urbanas uma face pode ser compreendida como um dos quatro lados de um quarteirão.

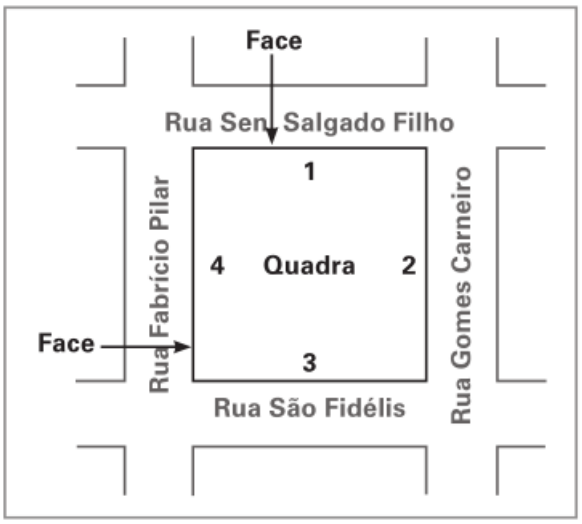


Figura 2.3: Uma quadra ou quarteirão e suas faces [6]

A figura 2.4 mostra a descrição do perímetro do setor em texto disponível no documento utilizado pelo recenseador.

CENSO 2010

Página :1 de 1
Data : 08-01-2010
Hora : 13:37:42 h

UF : Rio Grande do Sul	43
MUNICÍPIO : Uruguaiana	22400
DISTRITO : Uruguaiana	05
SUBDISTRITO :	00
SETOR: 0029	0029
SITUAÇÃO : 10-URBANA	
AGÊNCIA :432240000-URUGUAIANA	
BAIRRO :São João	005

Ponto Inicial e Ponto final:

RUA JULIO DE CASTILHOS COM A AV. GEN. FLORES DA CUNHA

Descrição do Perímetro:

DO PONTO INICIAL SEGUE PELA RUA JULIO DE CASTILHOS ATÉ A RUA BENTO GONCALVES SEGUE POR ESTA ATÉ A RUA GEN. HIPÓLITO SEGUE POR ESTA ATÉ A RUA DOS ANDRADAS SEGUE POR ESTA ATÉ A RUA PRADO LIMA SEGUE POR ESTA ATÉ A AV. GEN. FLORES DA CUNHA SEGUE POR ESTA ATÉ O PONTO INICIAL

Setores a serem excluídos:

NADA A REGISTRAR

Aglomerados Rurais, Subnormais, Assentamentos Rurais Somente Identificados:

NADA A REGISTRAR

Figura 2.4: Descrição do perímetro do setor [6]

A métrica adotada pelo IBGE para a definição de um setor censitário é pautada em dois critérios: o número de unidades construídas nele e sua extensão territorial. Sendo assim, um setor censitário pode estar restrito a umas poucas quadras, uma única quadra ou até mesmo uma única edificação.

O Censo 2010 atribuiu cerca de 3.000 variáveis a cada setor pesquisado, buscando obter o resultado mais fiel à realidade do país. Dentre as características levantadas encontram-se informações de domicílio, alfabetização, cor e raça, idade e gênero e renda [8].

Um setor censitário pode ainda ser caracterizado pela situação do setor e o tipo de setor. A situação do setor varia da seguinte forma:

- Situação urbana
    - Área urbanizada de cidade ou vila
    - Área não-urbanizada de cidade ou vila
    - Área urbana isolada
- Situação rural
    - Aglomerado rural de extensão urbana
    - Aglomerado rural isolado – povoado
    - Aglomerado rural isolado – núcleo
    - Aglomerado rural isolado – outros aglomerados
    - Zona rural, exclusive aglomerado rural

O tipo de setor pode ser classificado como:

- Setor comum ou não especial
- Setor especial de aglomerado subnormal
- Setor especial de quartéis, bases militares, etc.
- Setor especial de alojamento, acampamentos, etc.
- Setor especial de embarcações, barcos, navios, etc.
- Setor especial de aldeia indígena.
- Setor especial de penitenciárias, colônias penais, presídios, cadeias, etc.
- Setor especial de asilos, orfanatos, conventos, hospitais, etc.
- Setor especial de projetos de assentamentos rurais

Os arquivos referentes aos resultados do Censo Demográfico de 2010 estão disponíveis para livre acesso da população nos repositórios públicos do IBGE. Para o trabalho realizado serão estudados três principais repositórios descritos a seguir.

2.3 – Agregados por Setores Censitários

Obtido através do servidor de FTP ftp.ibge.gov.br no endereço /Censos/Censo_Demografico_2010/Resultados_do_Universo/Agregados_por_Setores_Censitarios o repositório Agregado por Setores Censitários contém a coleção completa de arquivos contendo as informações de cada variável dos setores censitários. Os arquivos estão divididos por Unidade de Federação, cada Unidade possui um diretório compactado no formato ZIP com os dados censitários referentes aos setores da Unidade em questão tanto em CSV como em XLS. A figura 2.5 mostra a estrutura do diretório compactado para a UF RJ.

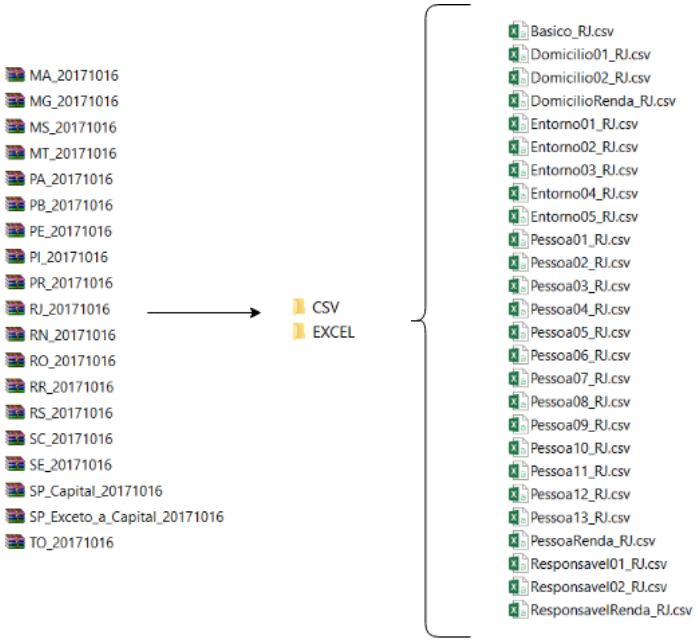


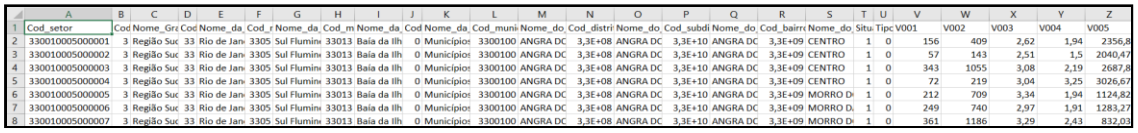
Figura 2.5: Agregados por Setores Censitários

De acordo com a documentação do repositório [8] o conteúdo de cada diretório de UF é composto das seguintes 18 planilhas:

- Básico - o arquivo Basico_UF.xls contém os códigos e nomes das subdivisões geográficas e a informação básica do cadastro de áreas (totais, médias e variâncias), onde UF é a sigla da Unidade da Federação;

- Domicílio - São duas planilhas (Domicilio01_UF.xls e Domicilio02_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informações sobre características dos domicílios, informações sobre os moradores por sexo, idade e características do domicílio;
- Responsável - São duas planilhas (Responsavel01_UF.xls e Responsavel02_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informações sobre os responsáveis por domicílios particulares permanentes [1] por sexo, idade, alfabetização;
- Alfabetização - São duas planilhas (Pessoa01_UF.xls e Pessoa02_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informação sobre a população residente por sexo, idade;
- Cor e Raça - São três planilhas (Pessoa03_UF.xls a Pessoa05_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informação sobre cor ou raça da população por sexo, idade;
- Parentesco - São quatro planilhas (Pessoa06_UF.xls a Pessoa09_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informação sobre a população;
- Registro Civil - O arquivo Pessoa10_UF.xls contém as informações sobre registro de nascimento da população, onde UF é a sigla da Unidade da Federação;
- Pessoa - São três planilhas (Pessoa11_UF.xls a Pessoa13_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informação sobre a população residente por sexo, idade;
- Entorno - São cinco planilhas (Entorno01_UF.xls a Entorno05_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informações sobre 10 variáveis a respeito do entorno das quadras/faces dos setores censitários;
- Renda - São três planilhas (DomicílioRenda_UF.xls, PessoaRenda_UF.xls e ResponsavelRenda_UF.xls, onde UF é a sigla da Unidade da Federação) que fornecem informação sobre os rendimentos dos domicílios, pessoas e responsáveis.

A figura 2.6 apresenta a planilha Básico_RJ.csv.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	Cod_setor	Cod	Nome_Gra	Cod	Nome_da_	Cod_r	Nome_da_	Cod_m	Nome_da_	Cod	Nome_da_	Cod_muni	Nome_do_	Cod_distri	Nome_do_	Cod_subdi	Nome_do_	Cod_bairr	Nome_do_	Situ	Tipc	V001	V002	V003	V004	V005
2	330010005000001	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	CENTRO	1	0	156	409	2,62	1,94	2356,8
3	330010005000002	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	CENTRO	1	0	57	143	2,51	1,5	2040,47
4	330010005000003	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	CENTRO	1	0	343	1055	3,08	2,19	2687,8
5	330010005000004	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	CENTRO	1	0	72	219	3,04	3,25	3026,67
6	330010005000005	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	MORRO D	1	0	212	709	3,34	1,94	1124,82
7	330010005000006	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	MORRO D	1	0	249	740	2,97	1,91	1283,27
8	330010005000007	3	Região Suc	33	Rio de Jan	3305	Sul Fluminе	33013	Baía da Ilh	0	Municípios	3300100	ANGRA DC	3,3E+08	ANGRA DC	3,3E+10	ANGRA DC	3,3E+09	MORRO D	1	0	361	1186	3,29	2,43	832,03

Figura 2.6: Planilha exibindo arquivo Básico_RJ.csv

Na Figura 2.6, a coluna Cod_setor se refere ao código do setor censitário e a coluna V001 se refere à variável 001 que por sua vez está documentada como "Domicílios particulares permanentes ou pessoas responsáveis por domicílios particulares permanentes" [8].

2.4 – Base de Faces de Logradouros

Obtido através do servidor de FTP geoftp.ibge.gov.br no endereço /recortes_para_fins_estatisticos/malha_de_setores_censitarios/censo_2010/base_de_faces_de_logradouros o repositório Base de Faces de Logradouros contém a coleção completa de arquivos possuindo a relação entre os setores censitários e as faces que os compõem. Os arquivos estão divididos por Unidade de Federação, cada Unidade está dividida em arquivos compactados em ZIP referentes a subdistritos dessa Unidade. Dentro do arquivo compactado encontram-se arquivos referentes a faces, setores e subdistritos nos formatos DBF, PRJ, SHP, XML, SHX e PNG. Além disso na raiz do repositório encontra-se um arquivo contendo uma planilha com a descrição detalhada de cada arquivo de subdistrito. A figura 2.7 apresenta a planilha "composição dos arquivos da Base de Faces de Logradouros do CD2010".

6248	UF	Município	Nome Município	Distrito	Nome Distrito	Subdistrito	Nome Subdistrito	Arquivo
6249	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	06	Portuária	33045570506
6250	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	07	Centro	33045570507
6251	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	08	Rio Comprido	33045570508
6252	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	09	Botafogo	33045570509
6253	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	10	Copacabana	33045570510
6254	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	11	Lagoa	33045570511
6255	RJ	3304557	Rio de Janeiro	05	Rio de Janeiro	12	São Cristovão	33045570512

Figura 2.7: Descrição dos Subdistritos

A figura 2.8 mostra a estrutura do repositório para a UF RJ, subdistrito 33045570510 (Copacabana).

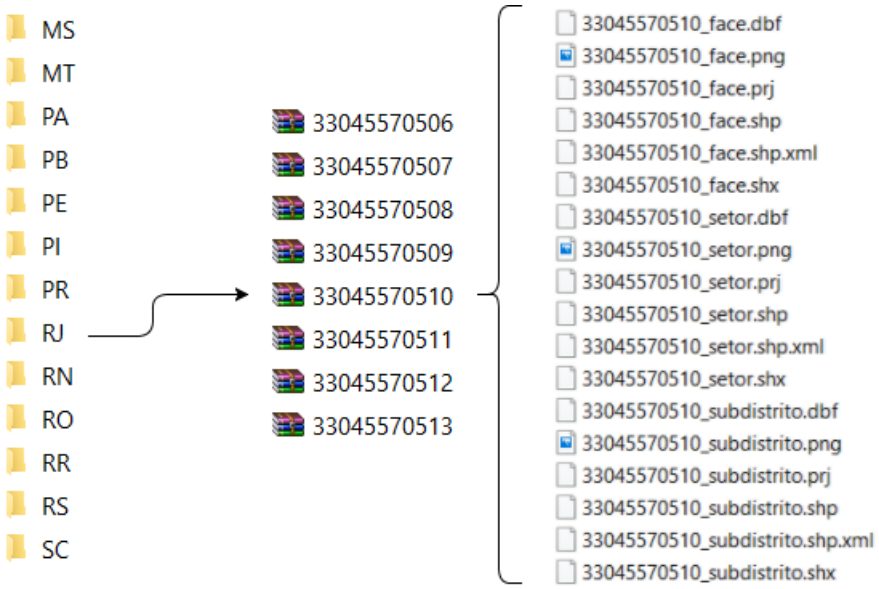


Figura 2.8: Estrutura do repositório de Base de Faces de Logradouros

Realizando a conversão do arquivo 33045570510_face.dbf para CSV usando uma ferramenta online de conversão (https://dbfconv.com/) obtém-se o arquivo com o conteúdo exibido na figura 2.9.

```
ID,CD_GEO,CD_SETOR,CD_QUADRA,CD_FACE,NM_TIPO_LO,NM_TITULO_,NM_NOME_LO,TOT_RES,TOT_GERAL
24757914,"330455705100001001002","330455705100001","001","002","PRACA","ALMIRANTE","JULIO DE NORONHA",45,46
24668993,"330455705100001001003","330455705100001","001","003","AVENIDA","","ATLANTICA",127,130
24792688,"330455705100002001001","330455705100002","001","001","RUA","","GUSTAVO SAMPAIO",242,247
24792687,"330455705100004001001","330455705100004","001","001","RUA","","GUSTAVO SAMPAIO",169,169
24667230,"330455705100005001002","330455705100005","001","002","RUA","","GUSTAVO SAMPAIO",57,57
24778580,"330455705100005001004","330455705100005","001","004","AVENIDA","","ATLANTICA",137,138
24792686,"330455705100006001001","330455705100006","001","001","RUA","","GUSTAVO SAMPAIO",152,152
24737114,"330455705100006001002","330455705100006","001","002","RUA","","AURELIANO LEAL",43,43
24791904,"330455705100007001001","330455705100007","001","001","LADEIRA","","ARY BARROSO",3,3
24791905,"330455705100007001002","330455705100007","001","002","LADEIRA","","ARY BARROSO",9,10
24799554,"330455705100007002001","330455705100007","002","001","LADEIRA","","ARY BARROSO",12,12
24791903,"330455705100007003002","330455705100007","003","002","LADEIRA","","ARY BARROSO",25,26
24799556,"330455705100008001003","330455705100008","001","003","LADEIRA","","ARY BARROSO",35,35
24737115,"330455705100008001001","330455705100008","001","001","RUA","GENERAL","RIBEIRO DA COSTA",76,76
24799555,"330455705100008001002","330455705100008","001","002","RUA","GENERAL","RIBEIRO DA COSTA",158,159
24801786,"330455705100009001002","330455705100009","001","002","RUA","GENERAL","RIBEIRO DA COSTA",163,163
```

Figura 2.9: Arquivo 33045570510_face.dbf após conversão para CSV

A figura 2.10 exibe a especificação adotada para a estrutura do arquivo de face de acordo com a documentação do repositório [9].

Arquivo	XXXXXXXXXXX_face		
Campo	**Tipo**	**Tamanho**	**Descrição**
ID	N	10	Código identificador
CD_GEO	C	21	Concatenação dos campos CD_SETOR+CD_QUADRA+CD_FACE
CD_SETOR	C	15	Identificação do Setor
CD_QUADRA	C	3	Número da Quadra
CD_FACE	C	3	Número da Face
NM_TIPO_LO	C	20	Tipo do segmento do Logradouro : RUA, AVENIDA, TRAVESSA, etc...
NM_TITULO_	C	30	Título do segmento do Logradouro (Almirante, Visconde...etc): ABADE ABADESSA ACADEMICO, etc.
NM_NOME_LO	C	60	Nome do segmento do Logradouro
TOT_RES	N	3	Total de espécie residencial
TOT_GERAL	N	3	Total de espécies

Figura 2.10: Estrutura do arquivo de face

Tem-se então uma fonte de dados que permite a correlação entre nomes de logradouros e códigos de setores censitários.

2.5 – Malha de Setores Censitários Divisões Intramunicipais

Obtido através do servidor de FTP geoftp.ibge.gov.br no endereço /organizacao_do_territorio/malhas_territoriais/malhas_de_setores_censitarios__divisoes_intramunicipais/censo_2010/setores_censitarios_shp/ o repositório Malha de Setores Censitários Divisões Intramunicipais contém a coleção completa de arquivos responsáveis por caracterizar a geometria dos setores censitários. Os arquivos estão divididos por Unidade de Federação, cada Unidade está dividida em arquivos compactados em ZIP referentes aos distritos, municípios, setores censitários e subdistritos da respectiva UF. Dentro dos arquivos compactados encontram-se arquivos nos formatos DBF, PRJ, SHP e SHX que permitem delinear os limites dos setores censitários, descrevendo assim o polígono que o caracteriza. A figura 2.11 mostra a estrutura do repositório para a UF RJ.

Figura 2.11: Estrutura do repositório Malha de Setores Censitários Divisões Intramunicipais para a UF RJ

Arquivos com a extensão SHP são denominados arquivos *shapefile*, carregados juntos aos arquivos DBF, SHX e PRJ em ferramentas de visualização de mapas são capazes de criar uma camada por cima do mapa base. A figura 2.12 mostra o arquivo rj_municipios.zip sendo carregado no visualizador de mapas online do ArcGIS [10], criando assim a camada dos municípios do Estado do Rio de Janeiro.

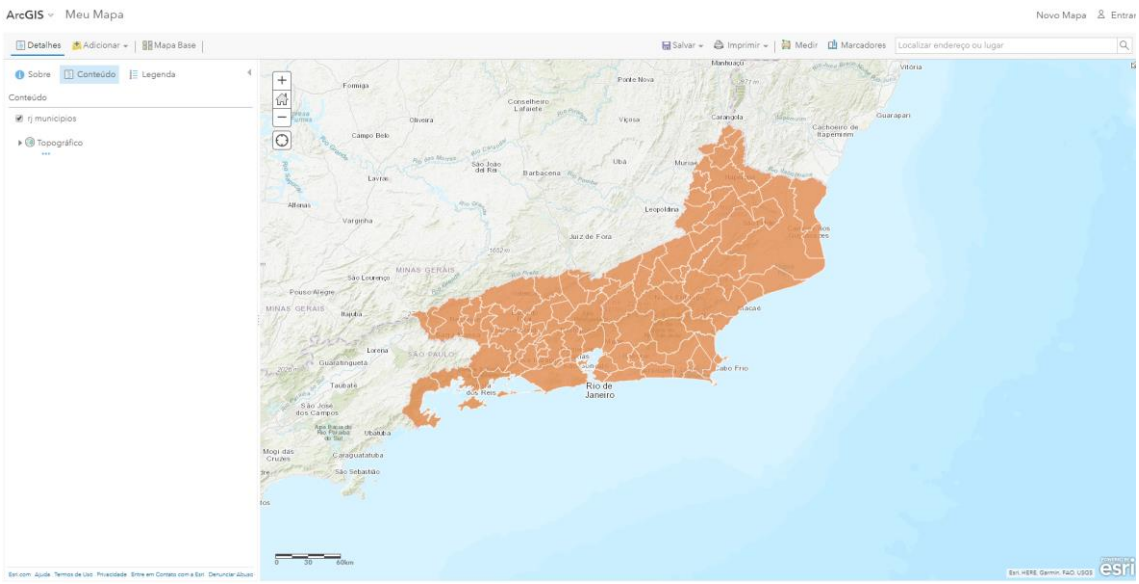


Figura 2.12: Arquivo rj_municipios.zip carregado no visualizador de mapas online do ArcGIS.

Utilizando a biblioteca Python *fiona* é possível realizar a leitura de arquivos SHP e dessa maneira extrair informações úteis que permitem criar o polígono que caracteriza um setor censitário.

2.6 – Resultado do Estudo

Levando em consideração o estudo realizado nas seções anteriores pode-se concluir que a extração de dados e informações dos repositórios do IBGE pode ser aprimorada através da criação de uma ferramenta que realize esse processo automaticamente, reduzindo a complexidade e a necessidade de um estudo aprofundado dos repositórios.

No formato atual, um sistema que visa fazer uma consulta aos dados obtidos pelo Censo 2010 necessita que o processo seja realizado manualmente extraindo e convertendo os arquivos através de ferramentas externas e dependendo do conjunto de informações de entrada o processo pode tornar-se mais custoso ainda pois há a necessidade de compreensão da indexação das variáveis de setores censitários. A princípio não há uma relação direta entre um endereço e a variável referente a renda média dos moradores do setor censitário em que esse endereço está inserido.

Para lidar com esse obstáculo será elaborada uma abordagem que munida dos três repositórios citados anteriormente consiga criar uma relação direta entre dados cadastrais de endereço e dados de setores censitários.

Capítulo 3

Ciência de Dados

Ciência de Dados como o nome sugere é uma ciência conduzida pelo estudo de dados e sua capacidade de gerar informação através da aplicação de métodos científicos tais como algoritmos computacionais, análises estatísticas e modelagens matemáticas.

A popularização do termo *Data Science* e o crescimento de ofertas de empregos buscando por cientistas de dados [11] fez com que surgisse o questionamento se a Ciência de Dados é uma aplicação da Estatística ou uma ciência por si só. Alguns consideram o termo somente uma nova roupagem para Estatística clássica [12]. Por outro lado, Ciência de Dados é considerada uma ciência que utiliza da estatística como uma técnica assim como usa programação e inteligência de negócios [13], sendo assim um cientista de dados é capaz de aplicar a informação obtida a partir de análises estatísticas enquanto o estatístico é capaz de formular tais modelos teóricos que permitem essas análises de serem feitas.

Compete à Ciência de Dados a aplicação de métodos de aprendizado de máquina, classificação, organização, clusterização, quantificação de incertezas, ciência computacional, mineração de dados, base de dados e visualização.

3.1 – Extração, Transformação e Carga de Dados

Para a execução do trabalho será adotada a solução de organização de dados conhecida como ETL, em português Extração, Transformação e Carga. Comumente aplicada em soluções de Data Warehouse, a ETL é adotada de forma a isolar a etapa responsável pela manipulação e limpeza dos dados que serão utilizados por um sistema, evitando assim que essas tarefas estejam presentes na execução do sistema impactando diretamente em seu desempenho.

A definição de ETL não se restringe ao uso de uma única ferramenta, podendo assim ser considerada uma coleção de ferramentas que executam as três funções descritas no nome da prática. Além disso sua usabilidade não está restrita somente a ambientes de Data Warehouse podendo ser aplicada em diversos outros ambientes, especialmente em ambientes com foco em extração de informações a partir de uma grande quantidade de dados.

A figura 3.1 exibe um diagrama de execução de um procedimento de ETL.

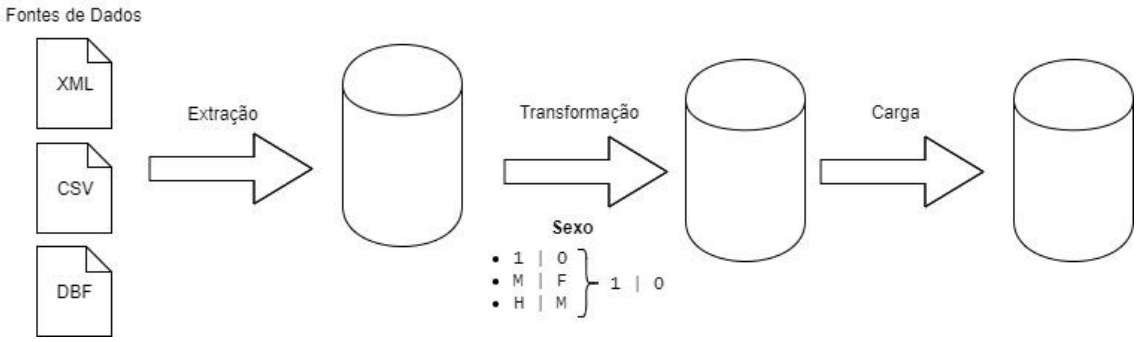


Figura 3.1: Diagrama do ETL

A etapa de extração é responsável por obter os dados que serão carregados no sistema. Devido ao fato das fontes possivelmente terem diferentes origens e formatos é comum e esperado que não haja uma padronização dos dados de entrada do ETL, delimitando a função dessa etapa somente ao processo de extração e consolidação dos dados em um formato que permita o trabalho da próxima etapa. Exemplos de fontes de dados são bases de dados relacionais, arquivos XML, planilhas de Excel, objetos JSON e arquivos de texto simples (também conhecidos como flat files). A figura 3.2 exibe diversas formas de fontes de dados.

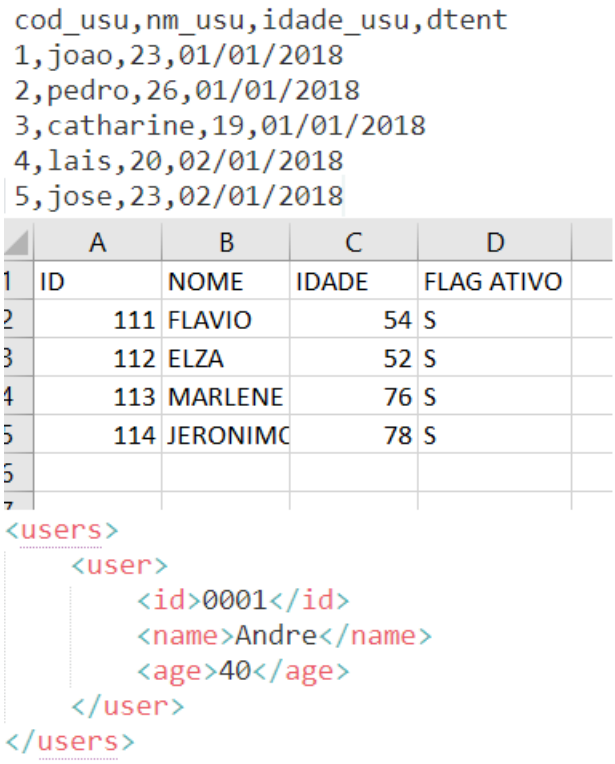


Figura 3.2: Fontes diversas de dados

Essa etapa é caracterizada por ser a mais custosa do processo uma vez que está diretamente ligada à quantidade de dados que serão carregados no sistema e os quão distintos são os padrões adotados pelas fontes consultadas.

É delegada a etapa de transformação as seguintes funções:

- Definição de padrões e regras a serem seguidos;
- Validação de dados;
- Consolidação de padrões;
- Limpeza dos dados;
- Exclusão de redundância;
- Ordenação de dados;
- Cálculos envolvendo os dados obtidos;
- União e concatenação de valores.

Um exemplo clássico de atuação nessa etapa é a necessidade de criação de uma regra para padronizar valores de sexo ou gênero de um sistema envolvendo cadastramento de usuário. Existem sistemas que utilizam valores numéricos binários (0 ou 1, 1 ou 2) para especificar o sexo do usuário, outros optam por pares de valores alfanuméricos (H para homens e M para mulheres, M para machos e F para fêmeas) e ainda existem outros sistemas que incluem mais opções como "prefiro não informar" ou "outro", o Facebook por exemplo permite você inserir seu próprio gênero ao cadastrar seu perfil na rede social [14].

Outro exemplo é a abreviação de nomes para tipos de logradouros, um sistema pode se referir ao tipo de logradouro como "avenida" ou em sua forma abreviada "av.".

Além disso alguns dados podem ser excluídos do sistema por falta de consistência, por exemplo, em um sistema que visa calcular a renda média dos seus usuários não faz sentido levar em conta usuários que não tenham informado seus respectivos valores de renda.

A última etapa do processo é responsável por carregar os dados previamente tratados de forma consolidada para uso no sistema final. A forma que os arquivos são dispostos pode variar de acordo com a necessidade do sistema, podendo estar disposto de forma mais genérica em um único arquivo simples de texto ou mais especificamente em um banco de dados relacional.

3.2 – Ferramentas de Ciência de Dados

3.2.1 – Python

Criada em 1991 pelo holandês Guido van Rossum a linguagem de programação Python em 2017 ocupava o segundo lugar na classificação de linguagens de programação mais populares globalmente, superando linguagens como PHP, C# e Javascript nos últimos 3 anos [15]. Sua popularidade é justificada principalmente pela legibilidade e simplicidade, características dos códigos em Python, além de se beneficiar de uma vasta comunidade ativa e diversas bibliotecas que possibilitam sua

aplicação nos mais diversos campos da computação, dentre eles o campo de Ciência de Dados.

A linguagem é distribuída gratuitamente nas versões 2.7.x e 3.x, sendo que a primeira por se tratar de uma versão mais antiga possui maior suporte a bibliotecas, enquanto a versão 3.x oferece uma implementação mais atualizada da linguagem, porém não conta com algumas bibliotecas disponíveis na versão 2.7.x.

A linguagem R se figura como o principal competidor frente ao uso de Python por se tratar de uma linguagem criada justamente para análise estatística e gráfica de dados. A grande vantagem do uso de Python está na sua facilidade de aprendizado e integração com outros sistemas [16], motivo o qual a versão 2.7.x será adotada para o trabalho.

3.2.2 – Jupyter Notebook

Jupyter Notebook é um ambiente executado em browser capaz de criar e editar documentos compostos por código, elementos de texto, figuras, gráficos e tabelas. Assim como o nome do aplicativo sugere cada documento pode ser tratado como um "caderno" onde se realiza a análise e visualização de dados, permitindo assim uma exibição do processo por completo, da execução do código ao resultado gráfico.

A figura 3.3 mostra o aplicativo sendo executado.

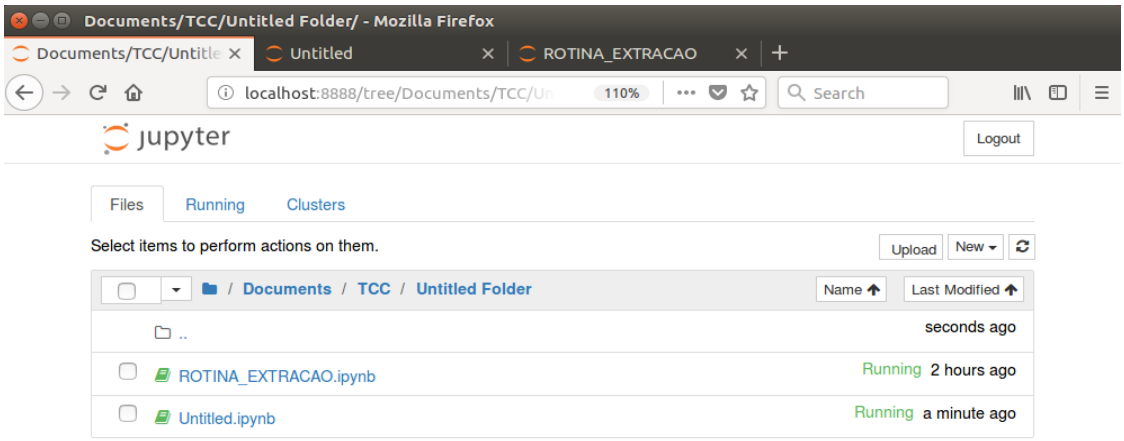


Figura 3.3: Jupyter Notebook sendo executado

A figura 3.4 mostra um documento sendo executado, no documento é exibido o código de criação de um *dataframe* e seu respectivo gráfico.

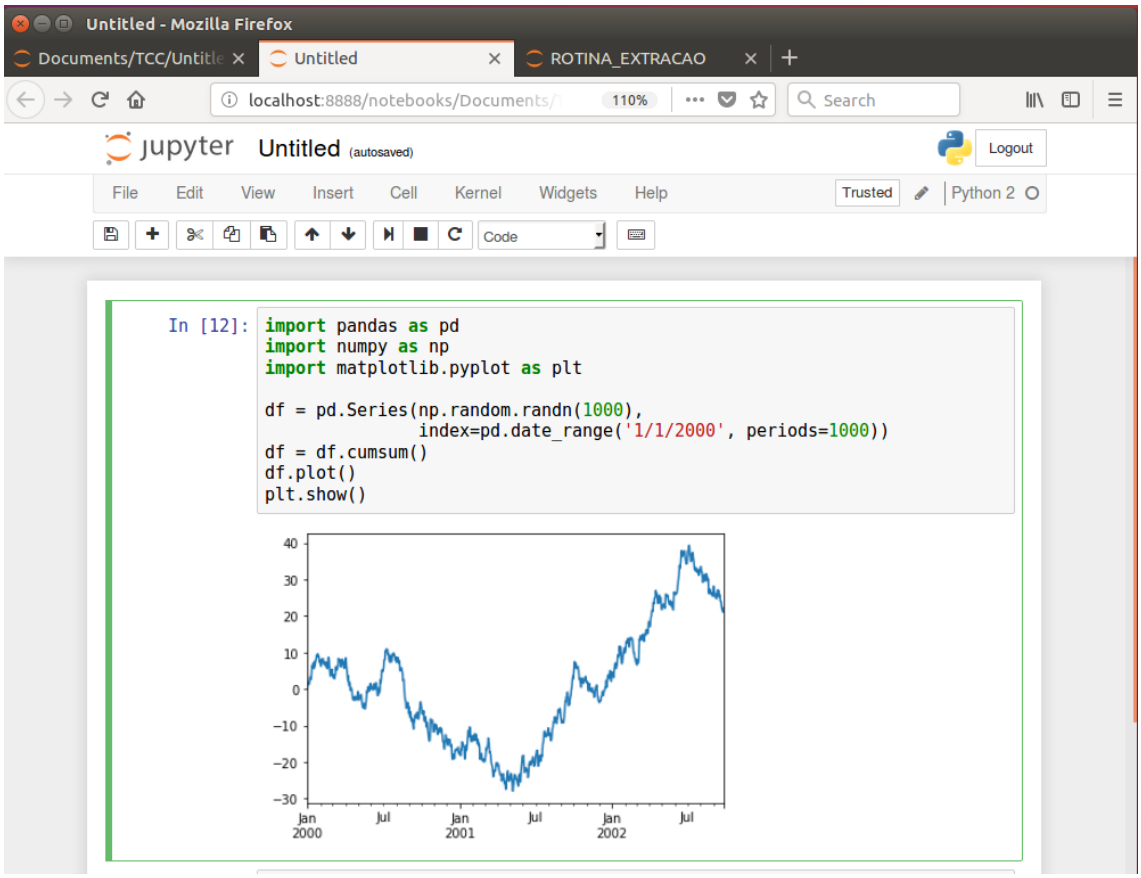


Figura 3.4: Execução de um documento no Jupyter Notebook

3.2.3 – Anaconda

Anaconda é um pacote composto por aplicativos e bibliotecas destinados para uso em Ciência de Dados em Python ou R. São parte do pacote as bibliotecas *numpy*, *pandas*, *matplotlib*, assim como o aplicativo *Jupyter Notebook*, além de um gerenciador de pacotes próprio chamado *conda*. Seu uso torna-se bastante prático pois dispensa a instalação manual de diversas bibliotecas, concentrando o necessário para o trabalho em uma única instalação.

A figura 3.5 mostra a execução do navegador do Anaconda.

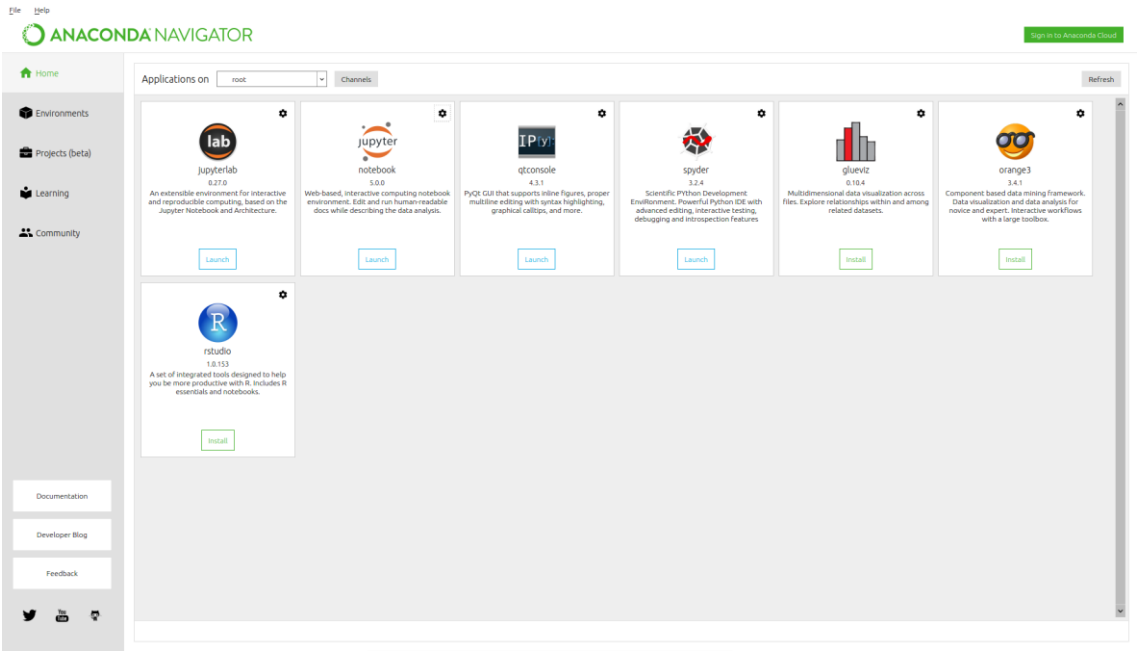


Figura 3.5: Anaconda Navigator sendo executado

3.2.4 – pandas

O manuseio e análise de dados no trabalho serão realizados principalmente através do uso da biblioteca Python *pandas*. Seu funcionamento se dá através da estruturação de dados em estruturas unidimensionais e bidimensionais nomeadas *Series* e *Data Frames*. A partir dessas estruturas são realizados comandos de transformação, conversão, agrupamento e indexação de forma a manipular os dados.

A figura 3.6 mostra um exemplo de uso da biblioteca.

```
In [26]: series = pd.Series([1,5,4,3,2,1],index=['A','B','C','D','E','F'])
         dataframe = pd.DataFrame(np.random.randint(low=0, high=5, size=(5, 6)),
                                  columns=['A','B','C','D','E','F'])
         dataframe = dataframe.append(series, ignore_index=True)
         dataframe[dataframe.sum(axis=1)==16]
```

Out[26]:

	A	B	C	D	E	F
4	4	0	2	2	4	4
5	1	5	4	3	2	1

Figura 3.6: Uso da biblioteca pandas

No código executado na figura 3.6 é criada uma estrutura *series* contendo uma série de inteiros fixos de 0 a 5 e uma estrutura *dataframe* contendo 5 linhas e 6 colunas, cada linha sendo uma série de inteiros randômicos de 0 a 5. Em seguida a *series* é

acrescentada ao *dataframe* e são exibidas as linhas do *dataframe* que possuem a soma de valores igual a 16.

3.2.5 – NumPy

NumPy é uma biblioteca com foco em operações matemáticas e uso de matrizes e vetores. Será utilizada no trabalho como ferramenta auxiliar para operações de conversão e classificação de tipos de dados.

A figura 3.7 mostra a biblioteca sendo utilizada para gerar uma função seno.

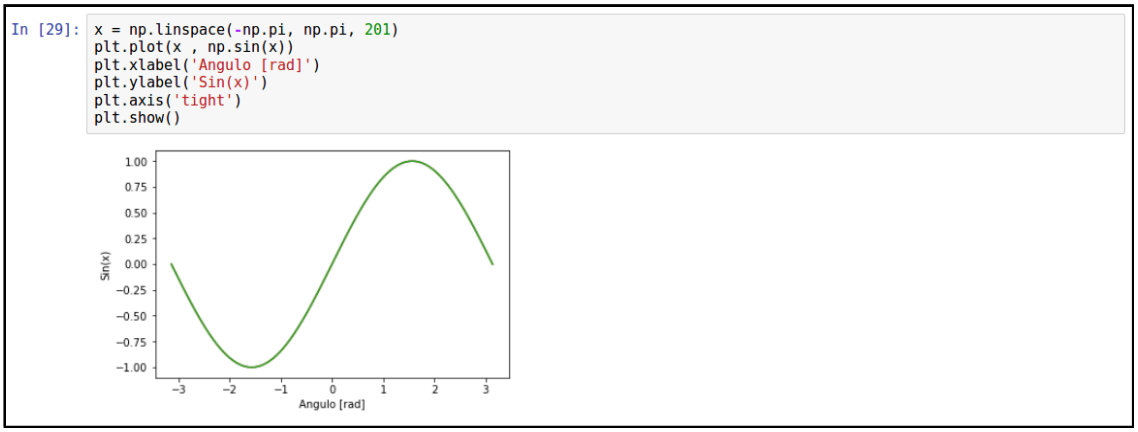


Figura 3.7: Função seno através da biblioteca NumPy

3.2.6 – ZipFile

A biblioteca *ZipFile* é utilizada para tratar arquivos compactados no formato ZIP. Seu repertório de funções permite a extração de um único arquivo, a descompactação total e a leitura e escrita do conteúdo byte a byte de arquivos de e para arquivos compactados.

3.2.7 – os, sys, shutil e glob

As bibliotecas *os*, *sys*, *shutil* e *glob* possuem funções que permitem acessar funcionalidades do sistema operacional, tais como navegar entre diretórios, mover, copiar e remover arquivos.

3.2.8 – csv e dbfpy

As funções presentes na biblioteca *csv* permitem manipular arquivos no formato CSV. É bastante útil em sistemas de análise de dados pois oferece uma forma simples e veloz de carregar e armazenar dado.

A biblioteca *dbfpy* permite a leitura de arquivos no formato DBF e sua conversão para o formato CSV.

3.2.9 – fiona e shapely

A biblioteca *fiona* é responsável por realizar a leitura de arquivos *shapefile*, integrando classes de sistemas de visualização de mapas com atributos e classes Python. Através da biblioteca *fiona* é possível definir a geometria de um setor censitário, funcionalidade crucial para a execução do trabalho.

A biblioteca *shapely* por sua vez permite definir formas geométricas utilizando coordenadas cartesianas. A partir dessa funcionalidade pode-se definir pontos em mapas utilizando informações de coordenadas geográficas.

3.2.10 – configparser

A biblioteca *configparser* permite a leitura de um arquivo de configurações com uma estrutura pré-definida, facilitando a execução de um sistema. A figura 3.8 mostra um exemplo de arquivo de configuração.

Figura 3.8: Estrutura dos arquivos de configurações

3.2.11 – Google Maps Geocoding API e requests

A API da Google é um serviço que oferece a geocodificação e geocodificação reversa de endereços. Sua utilização é feita a partir de requisições HTTP, no trabalho isso será feito através da biblioteca *requests*. A resposta é entregue no formato JSON, possuindo em seu corpo os dados de latitude e longitude de um endereço informado.

Capítulo 4

Implementação

Considerando o conteúdo dos repositórios estudados e as ferramentas disponíveis para análise e manipulação de dados torna-se não somente viável, mas também indispensável a incorporação de uma ferramenta que resolva o problema da consulta aos dados censitários do Censo Demográfico de 2010 do IBGE.

4.1 - Descrição do Problema

A indexação do resultado do Censo 2010 por código de setor censitário impede que informações de um indivíduo sejam obtidas diretamente utilizando seu endereço como ponto de referência para execução da consulta. Para isso é necessária a criação de uma fonte de dados consistente de forma a conferir credibilidade na relação entre o setor censitário e o endereço.

Como visto anteriormente no capítulo 2 um logradouro pode possuir diversos setores censitários, fato que sugere um impacto negativo na confiabilidade dos dados, além disso os dados presentes no repositório Base de Faces de Logradouros não estão dispostos de forma otimizada para consulta, tornando assim necessária uma manipulação que corrija tal aspecto.

Devido ao grande número de dados na composição dos repositórios utilizados será selecionada uma fatia parcial do conjunto total de dados para ser trabalhada. Por estar mais próximo à realidade da UFRJ para efeito de comparação e por se tratar do terceiro Estado em população e o Estado com maior densidade demográfica do país os dados do Rio de Janeiro serão utilizados na implementação do trabalho.

4.2 – Abordagens de Solução

Para resolver o problema descrito na seção anterior serão implementadas duas abordagens, uma utilizando somente bibliotecas Python para manipular os dados dos repositórios e outra indo além e utilizando a API de geocodificação do Google para auxiliar na exatidão do resultado.

A primeira abordagem levará em consideração somente a informação de logradouro, descartando o número do endereço, tendo como resultado assim uma média de valores em vez de um valor exato. Para isso serão utilizados os repositórios Agregados por Setores Censitários e Base de Faces de Logradouros. A consulta por meio desta abordagem poderá ser realizada sem necessidade de conexão com Internet.

A segunda abordagem por se utilizar da API de geocodificação do Google requer uma conexão com a Internet, porém seu resultado será uma relação direta entre entrada e saída. Para isso serão utilizados os repositórios Agregados por Setores Censitários e Malha de Setores Censitários Divisões Intramunicipais.

4.3 – Organização do Ambiente da Solução

Para que os processos de conversão e acesso a dados descritos nas seções a seguir sejam executados automaticamente é vital a organização dos arquivos do sistema em uma estrutura específica, além de estarem acompanhados do arquivo de configuração do projeto. A execução do sistema será realizada a partir do diretório contendo os módulos desenvolvidos. A estrutura de diretórios a ser seguida é a seguinte:

- Diretório raiz do projeto
    - Repositório Agregados por Setores Censitários
    - Repositório Base de Faces de Logradouros
    - Repositório Malhas de Setores Censitários Divisões Intramunicipais
    - Arquivo de Configuração
    - Scripts
        - Módulo de extração

- Módulo de transformação
- Módulo de consulta

O diretório Scrips é o ponto de partida para a execução das funções desenvolvidas.

4.4 – Extração e Conversão

A extração dos dados foi realizada a partir dos repositórios descritos no capítulo 2. A tabela 4.1 exibe informações referentes ao processo de extração para cada repositório considerando o uso de uma conexão de Internet com velocidade de download de 60Mb/s.

Tabela 4.1: Extração dos dados

	Agregados Por Setores Censitários	Base de Faces de Logradouros	Malhas de Setores Censitários Divisões Intramunicipais
Arquivos Compactados	30	10908	108
Tamanho (GB)	1.64	1.83	0.51
Tempo (minutos)	4	18	4
Total de Arquivos	1560	196344	432

A tabela 4.2 exibe as informações dos dados de cada UF no repositório Base de Face de Logradouros, assim como a média.

Tabela 4.2: Dados da Base de Face de Logradouros

UF	Arquivos	Tamanho (MB)
AC	23	8,12
AL	122	19,00
AM	94	21,30
AP	33	4,20
BA	869	122,00
CE	845	72,40

DF	19	30,90
ES	301	51,80
GO	379	64,90
MA	243	39,10
MG	1767	357,00
MS	175	26,00
MT	255	48,10
PA	250	38,00
PB	288	29,80
PE	404	51,30
PI	229	30,20
PR	821	106,00
RJ	381	126,00
RN	186	21,80
RO	104	13,80
RR	15	4,40
RS	1370	132,00
SC	461	80,00
SE	83	17,80
SP	1036	335,00
TO	153	21,10
	403,93	69,33

Para realizar a extração desses arquivos foi criado um módulo de funções Python que será utilizado para auxiliar o módulo de consulta. A seguir são apresentadas as funções que compõem o módulo de extração.

A figura 4.1 mostra a função *extrairEstado* responsável por extrair do repositório Base de Faces de Logradouros o conteúdo de uma UF específica e carregá-lo em um *dataframe*. A função recebe como parâmetro a sigla da UF de interesse e o caminho do arquivo de configuração no sistema do usuário que será abordado mais a frente.

```
def extrairEstado(siglaUF, pathConfig):
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if siglaUF.upper() not in listaUF:
        print('Sigla de UF invalida.')
        return None
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    baseFaceLogradouros = configuracoes['fonte']['baseFaceLogradouros'] + '/' + siglaUF + '/'
    extrairDbf(baseFaceLogradouros)
    converterDbfCsv(baseFaceLogradouros)
    frame = lerCsvs(baseFaceLogradouros)
    return frame
```

Figura 4.1: Função extrairEstado

A figura 4.2 mostra a função *extrairTodos* que por sua vez chama a função *extrairEstado* acrescentando o conteúdo de cada UF ao *dataframe* resultante.

```
def extrairTodos(pathConfig):
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    frame = pd.DataFrame()
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    for siglaUF in listaUF:
        try:
            frame = frame.append(extrairEstado(siglaUF, pathConfig), ignore_index=True)
        except:
            print('Não foi possível realizar a extração da UF '+siglaUF)
            return None
    return frame
```

Figura 4.2: Função extrairTodos

Estas funções são compostas de rotinas secundárias que realizam o trabalho de extração de fato. São elas *extrairDbf*, *converterDbfCsv*, *lerCsv* que serão exibidas a seguir.

A figura 4.3 mostra a função *extrairDbf*. Essa função recebe o caminho para o repositório Base de Faces de Logradouros configurado no arquivo de configuração concatenado com a sigla da UF de interesse. A partir deste caminho é realizada uma varredura em todos os arquivos ZIP contidos no diretório da UF e os arquivos de faces no formato DBF são descompactados no próprio diretório utilizando a biblioteca *zipfile*.

```
def extrairDbf(baseFaceLogradouros):
    for (dirpath, dirnames, filenames) in os.walk(baseFaceLogradouros):
        for filename in filenames:
            if filename.endswith('.zip'):
                arquivoZip = zipfile.ZipFile(baseFaceLogradouros+filename)
                try:
                    arquivoZip.extract(filename.split('.')[0]+'_face.dbf',baseFaceLogradouros)
                except:
                    pass
                arquivoZip.close()
```

Figura 4.3: Função extrairDbf

Em seguida é executada a função *converterDbfCsv*. A figura 4.4 mostra a execução dessa função. O código da função exibida na figura 4.4 quando executado realiza a leitura dos arquivos DBF extraídos anteriormente, carregando-os em arquivos no formato CSV. A execução dessa função depende das bibliotecas *dbfpy* e *csv*. Ao término da execução os arquivos com extensão DBF são removidos do diretório.

```
def converterDbfCsv(baseFaceLogradouros):
    for (dirpath, dirnames, filenames) in os.walk(baseFaceLogradouros):
        for filename in filenames:
            if filename.endswith('.dbf'):
                csv_fn = filename[:-4]+ ".csv"
                with open(dirpath+'/'+csv_fn,'wb') as csvfile:
                    in_db = dbf.Dbf(dirpath+'/'+filename)
                    out_csv = csv.writer(csvfile)
                    names = []
                    for field in in_db.header.fields:
                        names.append(field.name)
                    out_csv.writerow(names)
                    for rec in in_db:
                        out_csv.writerow(rec.fieldData)
                    in_db.close()
                os.remove(baseFaceLogradouros+filename)
```

Figura 4.4: Função converterDbfCsv

Após o processo de conversão dos arquivos é então realizada a carga do conteúdo dos arquivos CSV em um objeto *dataframe* através da função *lerCsvs* como mostra a figura 4.5.

```python
def lerCsvs(baseFaceLogradouros):
    allFiles = glob.glob(baseFaceLogradouros+'/*.csv')
    frame = pd.DataFrame
    list_ = []
    for file_ in allFiles:
        df = pd.read_csv(file_,index_col=None,header=0)
        list_.append(df)
        os.remove(file_)
    frame = pd.concat(list_)
    return frame
```

Figura 4.5: Função lerCsvs

Tem-se então como produto do módulo de extração um *dataframe* contendo os dados do repositório Base de Faces de Logradouros de uma UF ou de todas as UFs.

O resultado para as faces do Estado do Rio de Janeiro compõe um *dataframe* com 633.063 registros como mostra a figura 4.6.

```python
warnings.filterwarnings('ignore')
pathConfig = '/home/pedrohenrique/Documents/TCC/DATA/config.ini'
siglaUF = 'RJ'
frame = extrairEstado(siglaUF, pathConfig)
frame
```

65	9817945	33047061700000100100 3	3.304706e+14	1.0	3.0	RUA	NaN	SEM DENOMINACAO A	0	0
66	17606207	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0
67	17606209	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0
68	17606211	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0
69	17606212	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0
70	9816510	33047061700000100500 4	3.304706e+14	5.0	4.0	RUA	NaN	PROJETADA D	0	0
71	9816513	33047061700000100600 2	3.304706e+14	6.0	2.0	RUA	NaN	PROJETADA B	0	0
72	9817292	33047061700000100400 3	3.304706e+14	4.0	3.0	PRACA	NaN	ARMANDA CAMPELO DE BARROS	0	0

633063 rows × 10 columns

Figura 4.6: Dataframe com os dados extraídos do repositório Base de Faces de Logradouros

A próxima etapa é realizar a transformação dos dados presentes no dataframe, neste caso, dos 633.063 registros do Estado do Rio de Janeiro. Muitos desses registros possuem dados incoerentes, nulos ou outro tipo de dado infrutífero, sendo assim a quantidade total de registros será reduzida ao final do processo. Dados deste tipo ocorrem por se tratar de um processo de recenseamento em que o território é dividido de maneira a tornar a coleta de dados mais eficiente, o que não significa que para cada região levada em consideração existirão indivíduos residentes e consequentemente dados a se considerar.

Sendo assim o impacto dessa operação de descarte é a geração de maior confiabilidade nos dados encontrados, uma vez que dados irrelevantes passam a ser ignorados e desconsiderados.

4.5 – Transformação

O *dataframe* resultante dos processos anteriores é exibido na figura 4.7 através do comando *head* passando como parâmetro o número 10 para exibir os dez primeiros registros do *dataframe*.

```
In [113]: frame.head(10)
```

Out[113]:

	ID	CD_GEO	CD_SETOR	CD_QUADRA	CD_FACE	NM_TIPO_LO	NM_TITULO_	NM_NOME_LO	TOT_RES	TOT_GERAL
0	9803537	330370815000001002017	3.303708e+14	2.0	17.0	RUA	NaN	MARMORE	3	3
1	9803493	330370815000001001001	3.303708e+14	1.0	1.0	RUA	NaN	JOSE JOAQUIM GONCALVES	2	3
2	9803495	330370815000001001003	3.303708e+14	1.0	3.0	ESTRADA	NaN	QUEIMA SANGUE	4	4
3	9803554	330370815000001008002	3.303708e+14	8.0	2.0	ESTRADA	NaN	QUEIMA SANGUE	1	1
4	9803496	330370815000001001004	3.303708e+14	1.0	4.0	RUA	NaN	FERNANDO	22	24
5	9803548	330370815000001003002	3.303708e+14	3.0	2.0	RUA	NaN	FERNANDO	16	24
6	9803499	330370815000001001007	3.303708e+14	1.0	7.0	RUA	NaN	PAULO CHALTEIN BELO	35	36
7	9803502	330370815000001001010	3.303708e+14	1.0	10.0	RUA	NaN	PAULO CHALTEIN BELO	2	2
8	9803515	330370815000001001023	3.303708e+14	1.0	23.0	RUA	NaN	PAULO CHALTEIN BELO	10	12
9	9803545	330370815000001002025	3.303708e+14	2.0	25.0	RUA	NaN	PAULO CHALTEIN BELO	19	21

Figura 4.7: Dataframe resultante dos processos de extração e conversão

Nessa etapa serão realizadas transformações na estrutura *dataframe* e nos dados de forma a viabilizar uma consulta eficiente. Para isso foi definido um módulo com suas respectivas funções de transformação. A função principal é mostrada na figura 4.8.

```
def transformar(frame):
    frame = frame.dropna(subset=['CD_SETOR'])
    frame = criaColunaEnder(frame)
    frame = converteInteiros(frame)
    frame = limpaGenericos(frame)
    frame = limparColunas(frame)
    return frame
```

Figura 4.8: Função principal de transformação

Primeiramente a função *transformar* limpa registros com código de setor censitário *NaN*. Esse efeito é obtido através do comando *dropna* sendo executado na coluna referente ao código de setor censitário. O *dataframe* resultante passa a possuir 475.159 registros, uma redução de 157.904 registros como mostra a figura 4.9.

```
In [14]: frameAux = frame.dropna(subset=['CD_SETOR'])
         print('Nova quantidade de registros: '+ str(frameAux.shape[0]))
         frameAux.head(5)

         Nova quantidade de registros: 475159

Out[14]:
```

	ID	CD_GEO	CD_SETOR	CD_QUADRA	CD_FACE	NM_TIPO_LO	NM_TITULO_	NM_NOME_LO	TOT_RES	TOT_GERAL
0	9803537	330370815000001002017	3.303708e+14	2.0	17.0	RUA	NaN	MARMORE	3	3
1	9803493	330370815000001001001	3.303708e+14	1.0	1.0	RUA	NaN	JOSE JOAQUIM GONCALVES	2	3
2	9803495	330370815000001001003	3.303708e+14	1.0	3.0	ESTRADA	NaN	QUEIMA SANGUE	4	4
3	9803554	330370815000001008002	3.303708e+14	8.0	2.0	ESTRADA	NaN	QUEIMA SANGUE	1	1
4	9803496	330370815000001001004	3.303708e+14	1.0	4.0	RUA	NaN	FERNANDO	22	24

Figura 4.9: Execução do comando dropna

Apesar da redução representar uma grande parcela, aproximadamente 25% do total, tais registros são inúteis para o objetivo do trabalho por não possuírem o código do setor censitário, ademais em sua maioria registros com código de setor censitário vazio possuem também os demais campos referentes a informações de logradouros também vazios, como pode ser constatado na figura 4.10. Considerando os 633.063 registros iniciais, apenas 3.470, cerca de 0.55% representam uma perda de dados real, isto é, registros que possuíam a informação de nome de logradouro que possivelmente seria utilizada no módulo de consulta.

```
In [15]: semConteudo = frame[(frame.CD_SETOR.isnull() == True) & (frame.NM_NOME_LO.isnull() == True)].shape[0]
         comConteudo = frame[(frame.CD_SETOR.isnull() == True) & (frame.NM_NOME_LO.isnull() == False)].shape[0]

         print('Com conteúdo: ' + str(comConteudo)+'\nSem conteúdo: ' + str(semConteudo))

         Com conteúdo: 3470
         Sem conteúdo: 154434
```

Figura 4.10: Quantidade de registros de setores censitários vazios com e sem conteúdo

Realizando uma melhor inspeção nesses 3.470 dados perdidos tem-se 1.958 registros referentes a logradouros com nomes genéricos (SEM NOME e variantes) como mostra a figura 4.11.

```
In [16]: frameComConteudo = frame[(frame.CD_SETOR.isnull() == True) & (frame.NM_NOME_LO.isnull() == False)]
         frameComConteudo['NM_NOME_LO'].value_counts().head(5)

Out[16]: SEM NOME      1658
         B               27
         SEM NOME 1      27
         A               26
         EVEREST         25
         Name: NM_NOME_LO, dtype: int64

In [17]: qtd = frameComConteudo[frameComConteudo.NM_NOME_LO.str.contains('NOME')].shape[0]
         qtd = qtd + frameComConteudo[frameComConteudo.NM_NOME_LO.str.len() <= 2].shape[0]
         print(qtd)

         1958
```

Figura 4.11: Exemplo de logradouros sendo removidos por possuírem campos vazios

Em seguida são executadas as funções *criaColunaEnder*, *converteInteiros*, *limparGenericos* e *limparColunas*. A função *criaColunaEnder* é mostrada na figura 4.12. A função *criaColunaEnder* realiza uma manipulação da estrutura do *dataframe* de forma a criar uma coluna unindo o tipo, título e nome do logradouro em um único texto. Antes disso é necessário que as colunas correspondentes ao tipo, título e nome de logradouro não possuam como conteúdo *NaN*, sendo assim é realizada uma transformação nessas colunas substituindo *NaN* por texto vazio.

```
def criaColunaEnder(frame):
    if len(frame.columns)==2:
        return frame
    frame['NM_TITULO_'] = frame['NM_TITULO_'].replace(np.nan, '', regex=True)
    frame['NM_TIPO_LO'] = frame['NM_TIPO_LO'].replace(np.nan, '', regex=True)
    frame['NM_NOME_LO'] = frame['NM_NOME_LO'].replace(np.nan, '', regex=True)
    frame['ender'] = frame['NM_TIPO_LO'].map(str) + ' ' + frame['NM_TITULO_'].map(str) +
    ' ' + frame['NM_NOME_LO'].map(str)
    frame['ender'] = frame['ender'].replace('\s+', ' ', regex=True)
    frame = frame[~frame.ender.str.contains('SEMNOME|IDENTIFICACAO|SEM NOME|DENOMINACAO')]
    frame = frame[frame.ender != ' ']
    return frame
```

Figura 4.12: Função criaColunaEnder

Como visto na figura 4.11 alguns logradouros possuem nomes não descritivos como por exemplo RUA SEM DENOMINACAO D, sendo assim removidos. Outros casos similares são:

- Variados de SEM DENOMINACAO
- Variados de SEM NOME e SEMNOME
- Variados de SEM IDENTIFICACAO
- Nomes vazios

Após a remoção o saldo de registros no *dataframe* passa a ser de 444.664 registros, o que significa uma redução de 30.495 registros.

Em seguida é executada a função *converteInteiros* que converte o tipo de dado da coluna de código de setor censitário para o tipo *int64*. Como visto anteriormente na figura 4.7 a coluna responsável por exibir o código do setor censitário não está no

formato desejado, exibindo um valor numérico em notação científica. A princípio não há com o que se preocupar nesse caso pois mesmo sendo exibido em forma de notação científica ainda é possível fazer uma consulta a partir de uma comparação com um número inteiro.

Porém torna-se necessária a transformação do tipo de dado atribuído a coluna de código do setor censitário ao realizar uma exportação do *dataframe* para arquivo CSV uma vez que o valor referente ao código do setor censitário passa a ser exibido em notação científica como texto, perdendo assim a informação do código como número inteiro. A figura 4.13 mostra a exportação do *dataframe* para arquivo CSV.

```
In [123]: rootFolder = '/home/pedrohenrique/Documents/TCC/DATA/base_de_faces_de_logradouros'
          output = rootFolder + '/frame_nao_transformado.csv'

          frame.to_csv(output)
```

Figura 4.13: Exportação do dataframe para arquivo CSV

A figura 4.14 exibe o arquivo CSV originado da exportação em um editor de texto. O código do setor (CD_SETOR) está em um formato indesejado.

```
,ID,CD_GEO,CD_SETOR,CD_QUADRA,CD_FACE,NM_TIPO_LO,NM_TITULO_,NM_NOME_LO,TOT_RES,TOT_GERAL
0,9803537,330370815000001002017,3.30370815e+14,2.0,17.0,RUA,,MARMORE,3,3
1,9803493,330370815000001001001,3.30370815e+14,1.0,1.0,RUA,,JOSE JOAQUIM GONCALVES,2,3
2,9803495,330370815000001001003,3.30370815e+14,1.0,3.0,ESTRADA,,QUEIMA SANGUE,4,4
3,9803554,330370815000001008002,3.30370815e+14,8.0,2.0,ESTRADA,,QUEIMA SANGUE,1,1
4,9803496,330370815000001001004,3.30370815e+14,1.0,4.0,RUA,,FERNANDO,22,24
5,9803548,330370815000001003002,3.30370815e+14,3.0,2.0,RUA,,FERNANDO,16,24
6,9803499,330370815000001001007,3.30370815e+14,1.0,7.0,RUA,,PAULO CHALTEIN BELO,35,36
7,9803502,330370815000001001010,3.30370815e+14,1.0,10.0,RUA,,PAULO CHALTEIN BELO,2,2
8,9803515,330370815000001001023,3.30370815e+14,1.0,23.0,RUA,,PAULO CHALTEIN BELO,10,12
```

Figura 4.14: Exportação do dataframe para CSV sendo exibido em um editor de texto

A figura 4.15 exibe os tipos de dados relacionados às colunas do *dataframe*, onde a coluna CD_SETOR possui o tipo float64.

```
In [9]: frame.dtypes

Out[9]: ID              int64
        CD_GEO         object
        CD_SETOR      float64
        CD_QUADRA     float64
        CD_FACE       float64
        NM_TIPO_LO     object
        NM_TITULO_     object
        NM_NOME_LO     object
        TOT_RES         int64
        TOT_GERAL       int64
        ender          object
        dtype: object
```

Figura 4.15: Tipos de dados do dataframe

Apesar do código do setor censitário ser um número extenso não há necessidade de configurá-lo como *float64*, sendo que o tipo *int64* basta para representar os valores da coluna sem perder qualquer informação. De acordo com a documentação da biblioteca *numpy* [17] o tipo *int64* tem um alcance de -9223372036854775808 a 9223372036854775807, satisfazendo assim o inteiro com 15 dígitos, usado para representar o código de setor censitário segundo o critério do IBGE.

Sob esta ótica foi criada a função *converteInteiros* para alterar o tipo da coluna de código de setor censitário para *int64* como mostra a figura 4.16.

```
def converteInteiros(frame):
    frame['CD_SETOR'] = frame['CD_SETOR'].astype(np.int64)
    return frame
```

Figura 4.16: Função converteInteiros

Dessa forma torna-se possível exportar o *dataframe* para arquivo CSV sem que haja perda de informações dos dados, permitindo assim o armazenamento dos dados para futuramente serem consultados por sistemas externos caso haja necessidade.

Executando o comando que faz a contagem de incidência dos nomes únicos de logradouros fica clara a existência de nomes genéricos que se repetem demasiadamente em diversos casos de setores censitários como mostra a figura 4.17.

```
In [54]: frame['ender'].value_counts()
Out[54]: RUA A                          2036
         RUA B                          1665
         RUA C                          1503
         RUA PROJETADA                  1495
         RUA D                          1246
         RUA UM                         1051
         RUA E                           973
         RUA DOIS                        906
         AVENIDA BRASIL                  889
         RUA F                           879
         RUA 1                           760
         RUA 2                           737
         RUA TRES                        711
         RODOVIA AMARAL PEIXOTO          702
         RUA G                           696
         RUA QUATRO                      684
```

Figura 4.17: Incidência de registros com nomes de endereço únicos

De acordo com a figura 4.17 o logradouro RUA A está presente em 2.036 registros do *dataframe*, o que, no entanto, por questão de bom senso, não significa que existe uma única RUA A que se estende através de cerca de 2.036 setores censitários diferentes. A figura 4.18 exibe os dados da RUA A.

```
In [55]: print(len(frame['CD_SETOR'][frame.ender == 'RUA A'].unique()))
         print(frame['CD_SETOR'][frame.ender == 'RUA A'].unique())

         605
         [330370815000016 330040705140002 330130610000006 330130610000013
          330130610000015 330130610000016 330022505000003 330023305000004
          330023305000006 330023305000023 330023305000034 330023305000036
          330023305000052 330023305000059 330455705210065 330455705210094
          330455705210145 330455705210153 330455705210251 330455705210521
          330455705210803 330455705210582 330455705210556 330455705210199
          330455705210819 330100965000008 330100965000013 330100965000015
          330100965000016 330030810000005 330455705280010 330180105000001
          330030815000001 330030815000008 330240305060005 330250215000002
          330250215000005 330250215000016 330250215000035 330080310000002
          330080310000011 330093605010001 330093605010009 330455705270028
          330455705270116 330455705270151 330455705270180 330455705270231
          330190010000003 330190010000005 330190010000013 330190010000025
          330390605000165 330270010000018 330270010000019 330270010000026
          330270010000030 330270010000038 330270010000040 330270010000107
          330270010000108 330455705320136 330600820000001 330455705170549
          330412805000014 330412805000015 330412805000024 330412805000025
          330600805000033 330600805000111 330550515000038 330260120000023
```

Figura 4.18: Dados dos registros com nome de endereço RUA A

Conclui-se que a RUA A está presente em 605 setores censitários diferentes. Verificando os valores da lista de códigos de setores censitários únicos exibidos na figura 4.18 nota-se uma grande variedade entre eles, o que novamente não faz sentido se considerar um único, ou até mesmo poucos casos de logradouros com nome RUA A.

A presença desse padrão torna-se indesejada pois um único nome de logradouro presente em muitos setores censitários deveria ser exclusividade de logradouros extensos como avenidas ou estradas, caso contrário, quando o nome de uma rua se repete em muitos setores tem-se uma possível causa de ruído, tanto para a obtenção dos dados que trazem a informação daquele logradouro quanto na validação dos dados do próprio repositório (figura 4.18), uma situação que não condiz com a realidade.

Levando em consideração a composição do código do setor censitário estudado na seção 2 do capítulo 2 verifica-se que os 7 primeiros dígitos do código caracterizam a UF e o Município onde o setor está presente. A partir dessa informação é realizada uma validação dos dados referentes a RUA A e demais logradouros que sugerem um nome genérico. Para efeito de comparação a validação será feita também para os logradouros AVENIDA BRASIL e RODOVIA AMARAL PEIXOTO que também possuem grande quantidade de setores censitários. A figura 4.19 exibe o resultado da consulta.

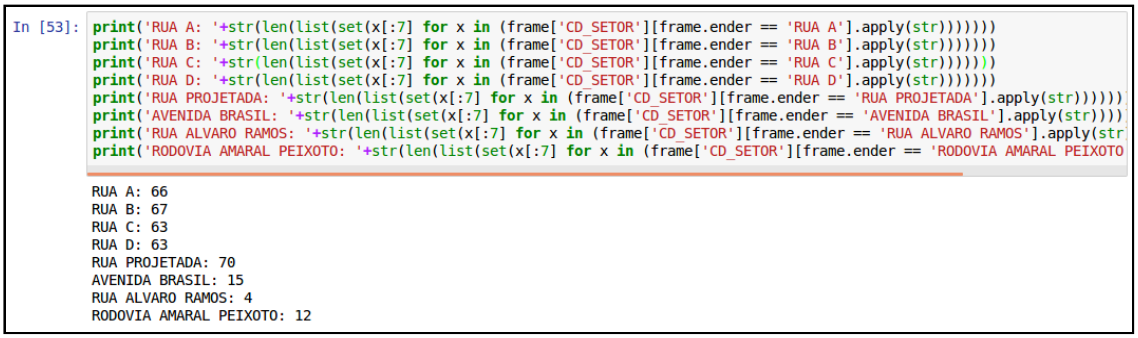


Figura 4.19: Validação de nomes de logradouros genéricos

De acordo com o resultado obtido pode-se concluir que a presença de nomes genéricos como RUA A e RUA PROJETADA na lista de registros com maiores incidências de setores censitários não condiz com os casos da presença dos logradouros AVENIDA BRASIL e RODOVIA AMARAL PEIXOTO. A Avenida Brasil é a via de maior extensão do país, possuindo 58 quilômetros de extensão, cortando 27 bairros do município do Rio de Janeiro [18], já a Rodovia Amaral Peixoto trata-se de uma rodovia de 200 quilômetros de extensão cruzando os municípios de Macaé, Maricá, Saquarema, Araruama, Iguaba Grande, São Pedro d'Aldeia, Cabo Frio, Casimiro de Abreu, Rio das Ostras, São Gonçalo e Niterói [19], fatos que justificam a grande quantidade de setores censitários atribuídos a esses logradouros.

Existem ainda casos isolados de logradouros com nomes específicos que aparecem em mais de um município como no caso de RUA ALVARO RAMOS, presente em Botafogo, Jacarepaguá, Duque de Caxias e Nova Iguaçu.

Para contornar essa situação serão removidos casos de nomes de logradouros genéricos através da função *limpaGenericos*. A figura 4.20 mostra esse processo.

```
def limpaGenericos(frame):
    if len(frame.columns)==2:
        return frame
    listGenericosX = ['RUA SETE','RUA SEIS','RUA QUATRO','RUA DOIS','RUA CINCO',
                      'RUA UM','RUA TRES','RUA NOVE','RUA OITO','RUA DEZ','RUA ONZE',
                      'RUA TREZE','RUA QUATORZE','RUA DOZE','RUA QUINZE']
    listNomeDuploX = []
    listNomeNumericoX = []
    listIgnoradosX = ['ESTRADA','BR','CONDOMINIO','ALAMEDA','BLOCO','SAIDA','QUILOMETRO',
                      'ENTRADA','LOTE','LOTEAMENTO','PRACA','VILA','VIA']
    for x in frame['ender'].unique():
        if (len(x.split(' '))==2) and (len(x.split(' ')[1])==1):
            listNomeDuploX.append(x)
        if (len(x.split(' '))==2) and (x.split(' ')[1].isdigit())
        and not (any(x.split(' ')[0] in s for s in listIgnoradosX)):
            listNomeNumericoX.append(x)
    frame = frame[~frame.ender.isin(listNomeDuploX)]
    frame = frame[~frame.ender.str.contains('PROJETADA')]
    frame = frame[~frame.ender.isin(listGenericosX)]
    frame = frame[~frame.ender.isin(listNomeNumericoX)]
    return frame
```

Figura 4.20: Função limpaGenericos

Esta execução remove nomes de logradouros genéricos que apareciam com uma grande quantidade de setores censitários associados a tais, nomes de logradouros compostos em que o segundo nome se trata de um único caractere, como no caso de RUA A e nomes de logradouros compostos em que o segundo nome é uma denominação numérica como RUA 1.

A nova quantidade de registros passa a ser 400.770, o que corresponde a uma redução de 43.894 registros da quantidade anterior de 444.664 registros.

Realizando a mesma consulta pela listagem de logradouros com a maior incidência de setores censitários nota-se que para os 20 primeiros não se tem casos de nomes genéricos, como mostra a figura 4.21.

```
In [50]: frame['ender'].value_counts()

Out[50]: AVENIDA BRASIL                          889
         RODOVIA AMARAL PEIXOTO                  702
         RODOVIA BR 101                          500
         RUA SAO JOSE                            499
         RODOVIA PRESIDENTE DUTRA                459
         AVENIDA DAS AMERICAS                    424
         RUA SANTO ANTONIO                       402
         RUA SAO PAULO                           400
         AVENIDA ATLANTICA                       373
         RUA BELA VISTA                          341
         AVENIDA PRESIDENTE KENNEDY              331
         RUA DAS ROSAS                           327
         RUA SAO JOAO                            318
         ESTRADA DOS BANDEIRANTES                302
         RUA SAO SEBASTIAO                       300
         AVENIDA AUTOMOVEL CLUBE                 299
         RUA BEIRA RIO                           297
         RUA RIO DE JANEIRO                      297
         RUA SAO JORGE                           294
```

Figura 4.21: Incidência de logradouros nos setores censitários do Município do Rio de Janeiro

Por fim é executada uma função que mantém as colunas de código de setor censitário e nome de logradouro e remove as demais colunas irrelevantes para o módulo de consulta. A figura 4.22 mostra a função *limparColunas*.

```
def limparColunas(frame):
    if len(frame.columns)==2:
        return frame
    nomeColunas = ['ID','CD_GEO','CD_QUADRA','CD_FACE','NM_TIPO_LO',
                   'NM_TITULO_','NM_NOME_LO','TOT_RES','TOT_GERAL']
    if len(frame.columns>2):
        frame = frame.drop(nomeColunas, axis=1)
        if 'ender' in frame.columns:
            frame = frame.rename(columns={'ender': 'logradouro'})
    return frame
```

Figura 4.22: Função limparColunas

Após as transformações realizadas nas etapas anteriores serão feitas consultas visando validar a consistência dos dados obtidos do repositório Base de Faces de Logradouros. Esta validação é descrita na próxima seção.

4.6 – Validação

O resultado das etapas anteriores são 400.770 linhas de registros de logradouros que compõem os setores censitários correspondentes ao Estado do Rio de Janeiro, cerca de 64.4% do total de 633.063 extraídos do repositório, onde 25% do total são registros desprezíveis como visto anteriormente. A figura 4.23 mostra alguns dados estatísticos referente ao *dataframe* resultante.

```
In [8]: frame['logradouro'].value_counts().describe()

Out[8]: count    66983.000000
        mean         5.983160
        std         13.488503
        min          1.000000
        25%          2.000000
        50%          3.000000
        75%          6.000000
        max        889.000000
        Name: logradouro, dtype: float64

In [10]: qtdSetorUnico = len(frame['CD_SETOR'].unique())
         moda = frame['logradouro'].value_counts().mode()

         print('Moda: ' + str(moda[0]) + '\n' + 'Quantidade de setores censitários únicos: ' + str(qtdSetorUnico))

         Moda: 2
         Quantidade de setores censitários únicos: 23516
```

Figura 4.23: Dados estatísticos do Data Frame resultante

De acordo com a função *describe* executada como mostra a figura 4.23 tem-se uma contagem de 66.983 nomes de logradouros únicos, com uma média 5,98 setores censitários por nome de logradouro e um desvio padrão de 13,46 setores censitários. Os resultados se mostram consistentes com a realidade, porém estes valores serão testados mais a frente.

Para chegar à informação da quantidade média de logradouros presentes em cada setor censitário realiza-se a consulta mostrada na figura 4.24.

```
In [11]: qtd = []
         for x in pd.DataFrame(frame.groupby('CD_SETOR')['logradouro'].unique())['logradouro'].values:
             qtd.append(len(x))
         float(float(sum(qtd))/float(len(qtd)))

Out[11]: 6.5641265521347165
```

Figura 4.24: Quantidade média de nomes de logradouros únicos por setor censitário

Para efeito de comparação com a realidade o valor encontrado na execução da figura 4.24 mostra-se coeso. É de se esperar que a descrição de um setor possua no mínimo 4 nomes de logradouros formando o polígono típico de uma quadra, não indo muito além disso. Desta forma, o valor de 6,56 não pode ser caracterizado como uma anormalidade.

Analisando agora exclusivamente a quantidade de nomes de logradouros únicos podemos ver a progressão na alteração da quantidade total a cada etapa de transformação na figura 4.25.

```
In [15]: print('---- QUANTIDADE DE LOGRADOUROS ÚNICOS ----')
         print('Quantidade inicial: ' + str(len(frame['NM_NOME_LO'].unique())))
         frameEtapa1 = frame.dropna(subset=['CD_SETOR'])
         print('Após remover registros NaN: ' + str(len(frameEtapa1['NM_NOME_LO'].unique())))
         frameEtapa2 = criaColunaEnder(frameEtapa1)
         print('Após criação da coluna de nome completo do logradouro: ' + str(len(frameEtapa2['ender'].unique()))))
         frameEtapa3 = limpaGenericos(frameEtapa2)
         print('Após limpar nomes genéricos: ' + str(len(frameEtapa3['ender'].unique())))

         ---- QUANTIDADE DE LOGRADOUROS ÚNICOS ----
         Quantidade inicial: 58331
         Após remover registros NaN: 58239
         Após criação da coluna de nome completo do logradouro: 67976
         Após limpar nomes genéricos: 66983
```

Figura 4.25: Alteração na quantidade total de nomes de logradouros únicos

Percebe-se que após a remoção de registros com código de setor censitário *NaN* não houve uma perda considerável de conteúdo já que, como visto anteriormente durante a etapa de transformação, apenas 0.55% dos registros removidos dessa maneira possuíam conteúdo relevante. Após a etapa de criação da coluna referente ao nome completo do logradouro concatenando o tipo, título e nome houve um acréscimo considerável no valor da quantidade total. Isto pode ser explicado pela diferenciação gerada pelo acréscimo do tipo e título do logradouro, os quais não estavam sendo considerados nas contagens anteriores aonde se levou apenas em consideração a coluna referente ao nome, ou seja, Rua Brasil e Avenida Brasil passam a ser contados como dois registros diferentes. Por final, após a remoção de registros com nomes genéricos a contagem termina com 66.938 nomes, pouco menos que os 67.976 da etapa anterior.

Outro parâmetro relevante para análise é a quantidade de setores censitários presentes no *dataframe* resultante comparado ao total de setores censitários no Estado do RJ. Para realizar essa verificação o arquivo Basico_RJ.csv presente no repositório Agregados por Setores Censitários, estudado na seção 2.3 do capítulo 2, foi carregado em um *dataframe* e a quantidade de códigos de setores presente nesse arquivo foi comparada com a quantidade de códigos de setores únicos do *dataframe* resultante das etapas anteriores. Esse procedimento é mostrado na figura 4.26.

```
In [20]: basicoRJ = pd.read_csv('/home/pedrohenrique/Documents/TCC/DATA/Basico_RJ.csv', sep=';', encoding='ISO-8859-1')

         print('---- QUANTIDADE DE CÓDIGOS DE SETORES ÚNICOS ----')
         print('Quantidade Basico_RJ.csv: ' + str(len(basicoRJ['Cod_setor'].unique())))
         print('Quantidade inicial: ' + str(len(frame['CD_SETOR'].unique())))
         frameEtapa1 = frame.dropna(subset=['CD_SETOR'])
         print('Após remover registros NaN: ' + str(len(frameEtapa1['CD_SETOR'].unique())))
         frameEtapa2 = criaColunaEnder(frameEtapa1)
         print('Após criação da coluna de nome completo do logradouro: ' + str(len(frameEtapa2['CD_SETOR'].unique())))
         frameEtapa3 = limpaGenericos(frameEtapa3)
         print('Após limpar nomes genéricos: ' + str(len(frameEtapa3['CD_SETOR'].unique())))

         ---- QUANTIDADE DE CÓDIGOS DE SETORES ÚNICOS ----
         Quantidade Basico_RJ.csv: 27769
         Quantidade inicial: 24651
         Após remover registros NaN: 24650
         Após criação da coluna de nome completo do logradouro: 23665
         Após limpar nomes genéricos: 23516
```

Figura 4.26: Comparação entre as quantidades de códigos de setores censitários

Inicialmente, como resultado da extração dos dados, tem-se um total de 24.651 códigos únicos, 3.118 a menos que a quantidade de códigos presentes no arquivo Basico_RJ.csv usado como referência. A quantidade decresce em um único valor após a remoção dos registros com código *NaN*, apesar da redução de 25% dos registros totais no *dataframe* nesta etapa, quando os códigos únicos são listados, somente um faz referência ao código com valor *NaN*. Na etapa de criação da coluna contendo o nome completo do logradouro, nomes sem identificação são removidos ocasionando assim um decréscimo de 985 códigos. Finalmente ao limpar os registros com nomes de logradouros genéricos termina-se com 23.516 códigos únicos de setores censitários, 95,39% do total extraído do repositório Base de Faces de Logradouros e 84,68% do total de setores censitários do Estado do Rio de Janeiro.

Para realizar o estudo da relação entre logradouros, faces e setores censitários resultantes será utilizado como parâmetro de exemplo o logradouro Rua Eutíquio Soledade, do bairro de Tauá na Ilha do Governador, região metropolitana do Estado do Rio de Janeiro, cuja escolha foi aleatória. Primeiramente são listados os setores censitários que estão relacionados ao logradouro como mostra a figura 4.27, encontrando assim seis setores diferentes.

```
In [22]: pd.Series(frame['CD_SETOR'][frame.logradouro.str.contains('EUTIQUIO SOLEDADE')].sort_values().unique())

Out[22]: 0    330455705250185
         1    330455705250186
         2    330455705250187
         3    330455705250197
         4    330455705250198
         5    330455705250321
         dtype: int64
```

Figura 4.27: Setores censitários da Rua Eutíquio Soledade

Analisando o mapa de setores censitários do site do Ministério Público do Rio de Janeiro [20] é possível confirmar a existência dos seis setores censitários obtidos através da consulta realizada na figura 4.27. A figura 4.28 mostra o mapa e os setores em questão.

Figura 4.28: Mapa de setores censitários do Ministério Público do Rio de Janeiro [19]

Estão listados com as letras A, B, C, D, E e F os setores de código 330455705250321, 330455705250197, 330455705250198, 330455705250187, 330455705250186 e 330455705250185 respectivamente. Apesar de que uma pequena fração do logradouro Rua Eutíquio Soledade não ser contemplada pela consulta, tem-se nesse exemplo uma representação fidedigna do dado de média de aproximadamente 6 setores censitários por nome de logradouro como apontado anteriormente.

Analisando individualmente o setor censitário de código 330455705250198 encontram-se 7 nomes de logradouros relacionados como mostra a figura 4.29.

```
In [8]: pd.Series(frame['ender'][frame.CD_SETOR==330455705250198].unique())
Out[8]: 0    RUA SOLDADO WANDEL SARMENTO
        1                   RUA CAPANEMA
        2          RUA EUTIQUIO SOLEDADE
        3            RUA DOMINGOS MONDIM
        4         RUA DEMETRIO DE TOLEDO
        5       RUA ESCULTOR LEAO VELOSO
        6              PRACA FREI PAULO
        dtype: object

In [ ]:
```

Figura 4.29: Logradouros relacionados ao setor censitário de código 330455705250198

A figura 4.30 mostra a visualização da região do setor de código 330455705250198 no aplicativo Google Maps.

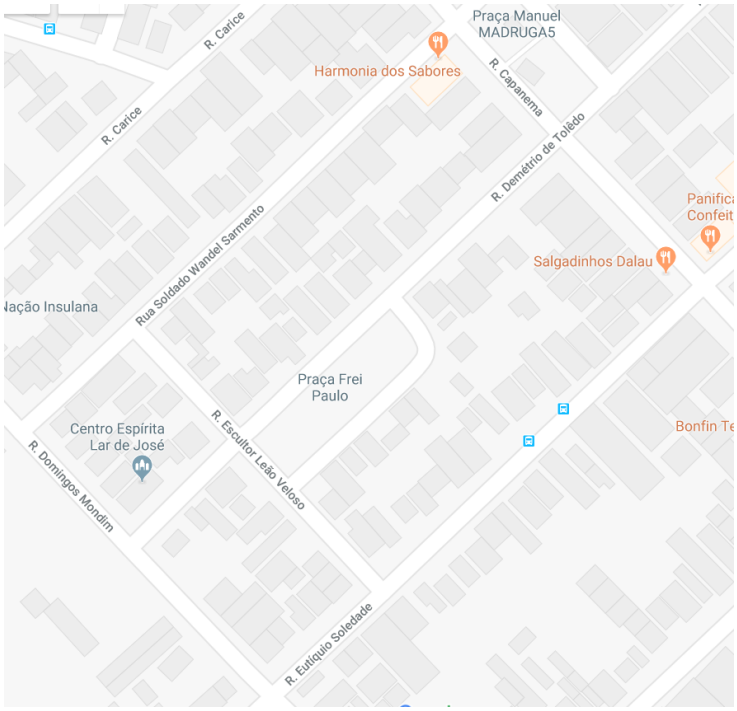


Figura 4.30: Região referente ao setor censitário de código 330455705250198 sendo exibida no Google Maps

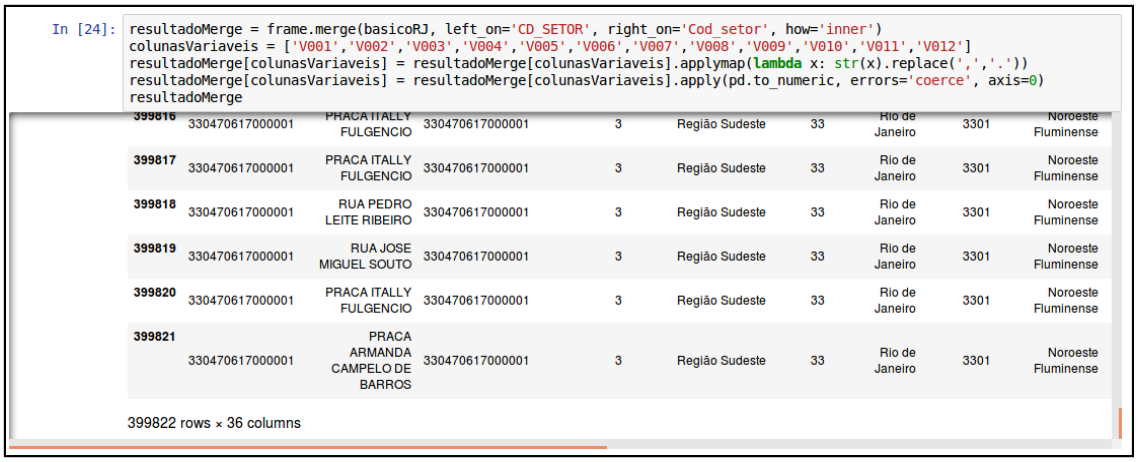
Comparando o resultado da consulta com a figura do mapa conclui-se que o *dataframe* possui a descrição exata do setor censitário de código 330455705250198, composto pelos logradouros Rua Soldado Wandel Sarmento, Rua Capanema, Rua Eutíquio Soledade, Rua Domingos Mondim, Rua Demétrio de Toledo, Rua Escultor Leão Veloso, Praça Frei Paulo.

Realizando a mesma pesquisa para o logradouro Travessa Alves, no bairro de Barro Vermelho no município de São Gonçalo, interior do Estado do Rio de Janeiro foram encontrados dois setores censitários (330240305030003 e 330490420000026) sendo apenas um deles de fato referente ao logradouro pesquisado. Também foi pesquisado o resultado para o logradouro Avenida Nossa Senhora de Copacabana, do bairro de Copacabana na região metropolitana do Rio de Janeiro, o resultado foram 61 setores censitários, sendo sua extensa maioria de fato localizado na região pesquisada. Para lidar com ambos os casos serão aplicadas medidas de filtragem, no caso da pesquisa por Travessa Alves e diferentes abordagens que buscam por resultados mais exatos, no caso da pesquisa por Avenida Nossa Senhora de Copacabana. Em seguida, foi realizada a validação das variáveis atribuídas aos setores censitários correlacionando-as aos nomes de logradouros. Para efeito de validação serão analisadas as variáveis referentes ao arquivo Basico_RJ.csv, são elas:

- V001 – Domicílios particulares permanentes ou pessoas responsáveis por domicílios particulares permanentes;
- V002 – Moradores em domicílios particulares permanentes ou população residente em domicílios particulares permanentes;
- V003 – Média do número de moradores em domicílios particulares permanentes (obtida pela divisão de Var2 por Var1);
- V004 – Variância do número de moradores em domicílios particulares permanentes;
- V005 – Valor do rendimento nominal médio mensal das pessoas responsáveis por domicílios particulares permanentes (com e sem rendimento);
- V006 – Variância do rendimento nominal mensal das pessoas responsáveis por domicílios particulares permanentes (com e sem rendimento);
- V007 – Valor do rendimento nominal médio mensal das pessoas responsáveis por domicílios particulares permanentes (com rendimento);
- V008 – Variância do rendimento nominal mensal das pessoas responsáveis por domicílios particulares permanentes (com rendimento);

- V009 – Valor ou rendimento nominal médio mensal das pessoas de 10 anos ou mais de idade (com e sem rendimento);
- V010 – Variância do rendimento nominal mensal das pessoas de 10 anos ou mais de idade (com e sem rendimento);
- V011 – Valor do rendimento nominal médio mensal das pessoas de 10 anos ou mais de idade (com rendimento)
- V012 – Variância do rendimento nominal mensal das pessoas de 10 anos ou mais de idade (com rendimento)

Para isso é executado um comando de união do *dataframe* resultante da transformação e o *dataframe* importado do arquivo BasicoRJ.csv mostrado na figura 4.31.



```
In [24]: resultadoMerge = frame.merge(basicoRJ, left_on='CD_SETOR', right_on='Cod_setor', how='inner')
colunasVariaveis = ['V001','V002','V003','V004','V005','V006','V007','V008','V009','V010','V011','V012']
resultadoMerge[colunasVariaveis] = resultadoMerge[colunasVariaveis].applymap(lambda x: str(x).replace(',','.'))
resultadoMerge[colunasVariaveis] = resultadoMerge[colunasVariaveis].apply(pd.to_numeric, errors='coerce', axis=0)
resultadoMerge
```

399816	330470617000001	PRACA ITALLY FULGENCIO	330470617000001	3	Região Sudeste	33	Rio de Janeiro	3301	Noroeste Fluminense
399817	330470617000001	PRACA ITALLY FULGENCIO	330470617000001	3	Região Sudeste	33	Rio de Janeiro	3301	Noroeste Fluminense
399818	330470617000001	RUA PEDRO LEITE RIBEIRO	330470617000001	3	Região Sudeste	33	Rio de Janeiro	3301	Noroeste Fluminense
399819	330470617000001	RUA JOSE MIGUEL SOUTO	330470617000001	3	Região Sudeste	33	Rio de Janeiro	3301	Noroeste Fluminense
399820	330470617000001	PRACA ITALLY FULGENCIO	330470617000001	3	Região Sudeste	33	Rio de Janeiro	3301	Noroeste Fluminense
399821	330470617000001	PRACA ARMANDA CAMPELO DE BARROS	330470617000001	3	Região Sudeste	33	Rio de Janeiro	3301	Noroeste Fluminense

399822 rows × 36 columns

Figura 4.31: Comando de união dos dataframes

Os códigos dos setores censitários de cada *dataframe* foram utilizados como referência para realizar a união, optando por excluir os casos aonde não houvesse correspondência. Após a união foi necessário realizar uma conversão do tipo de dado das colunas de variáveis pois estes são importados com o tipo *object* enquanto operações matemáticas que serão realizadas a seguir devem ser realizadas somente sobre tipos de dados numéricos.

Uma pesquisa pelas variáveis do logradouro de nome Rua Álvaro Ramos é apresentada na figura 4.32.

```
In [39]: resultadoMerge[['CD_SETOR']+colunasVariaveis][resultadoMerge.logradouro.str.contains('RUA ALVARO RAMOS')].head(10)
Out[39]:
```

	CD_SETOR	V001	V002	V003	V004	V005	V006	V007	V008	V009	V010	V011	V012
12943	330455705210497	218.0	727.0	3.33	2.26	1212.94	1578685.29	1335.45	1574509.11	676.97	928709.26	1093.70	1044921.40
12972	330455705210497	218.0	727.0	3.33	2.26	1212.94	1578685.29	1335.45	1574509.11	676.97	928709.26	1093.70	1044921.40
12973	330455705210497	218.0	727.0	3.33	2.26	1212.94	1578685.29	1335.45	1574509.11	676.97	928709.26	1093.70	1044921.40
12981	330455705210497	218.0	727.0	3.33	2.26	1212.94	1578685.29	1335.45	1574509.11	676.97	928709.26	1093.70	1044921.40
13012	330455705210500	168.0	552.0	3.29	3.44	801.34	448049.71	975.54	374978.56	480.42	381253.94	838.75	365017.80
13161	330455705210515	170.0	545.0	3.21	2.65	1242.88	1200549.63	1380.98	1142856.27	521.78	654361.40	1050.55	762282.56
13166	330455705210515	170.0	545.0	3.21	2.65	1242.88	1200549.63	1380.98	1142856.27	521.78	654361.40	1050.55	762282.56
13171	330455705210515	170.0	545.0	3.21	2.65	1242.88	1200549.63	1380.98	1142856.27	521.78	654361.40	1050.55	762282.56
14331	330455705210768	233.0	774.0	3.32	3.26	715.03	397187.14	833.01	364280.00	451.14	306160.11	765.88	278631.29
14333	330455705210768	233.0	774.0	3.32	3.26	715.03	397187.14	833.01	364280.00	451.14	306160.11	765.88	278631.29

Figura 4.32: Resultado da pesquisa pelas variáveis do logradouro Rua Álvaro Ramos

A partir do resultado da pesquisa pode-se notar a reincidência de códigos de setores censitários e suas respectivas variáveis. Para contornar esse fato, o resultado da pesquisa será agrupado pelo código de setor e avaliado sobre sua média, exibindo assim o resultado de variáveis de setores censitários distintos. A figura 4.33 mostra o novo resultado da pesquisa.

```
In [50]: resultadoMerge[['CD_SETOR']+colunasVariaveis][
             resultadoMerge.logradouro.str.contains('RUA ALVARO RAMOS')].groupby('CD_SETOR').mean()
Out[50]:
```

CD_SETOR	V001	V002	V003	V004	V005	V006	V007	V008	V009	V010	V011	V012
330030825000008	267.0	966.0	3.62	4.22	670.09	903549.42	748.59	950834.60	365.02	409635.29	616.53	537079.14
330170205000358	286.0	917.0	3.21	2.45	687.52	283243.39	877.82	194196.85	506.31	314264.00	872.91	221597.70
330170205000359	186.0	583.0	3.13	2.21	995.61	856976.75	1089.31	835433.53	571.63	605921.70	943.68	649340.31
330350005200041	229.0	677.0	2.96	2.56	719.78	355944.93	796.28	332748.49	408.25	284359.75	712.84	279374.65
330455705090028	229.0	585.0	2.55	1.56	4707.85	67832318.67	4856.30	69256694.89	3108.58	36214017.55	3867.48	42133859.13
330455705090029	228.0	659.0	2.89	2.95	2389.11	6376948.38	2487.30	6394886.98	1364.12	4076017.31	1983.34	4699531.82
330455705090030	262.0	642.0	2.45	1.14	5662.90	21285153.65	6156.34	20097114.50	4053.52	18318256.05	5274.03	17394817.99
330455705090061	165.0	419.0	2.54	2.26	4963.76	22631697.43	5118.88	22544363.49	2875.30	16223175.30	4079.14	18111467.92
330455705090062	142.0	354.0	2.49	2.05	4295.39	17117478.75	4551.83	16971049.45	2832.69	13354934.40	3788.57	14243753.03
330455705090064	265.0	671.0	2.53	1.79	3429.57	15134178.88	3786.81	15358766.26	2249.17	10172643.58	2949.65	11276974.10
330455705090463	166.0	420.0	2.53	2.42	5585.91	27624276.35	5906.12	27314654.31	3703.33	21958642.92	4692.58	23186396.09
330455705090466	229.0	532.0	2.32	1.32	5618.20	21522720.36	6012.00	20659773.12	4121.85	19766793.86	5111.95	19452770.39
330455705090470	163.0	387.0	2.37	1.59	3662.66	12335127.26	3731.33	12310005.43	2419.04	9609949.41	2984.85	10170740.06
330455705210496	330.0	1077.0	3.26	2.93	641.21	335576.67	829.80	277556.91	505.67	329245.91	797.81	286311.75
330455705210497	218.0	727.0	3.33	2.26	1212.94	1578685.29	1335.45	1574509.11	676.97	928709.26	1093.70	1044921.40
330455705210500	168.0	552.0	3.29	3.44	801.34	448049.71	975.54	374978.56	480.42	381253.94	838.75	365017.80
330455705210515	170.0	545.0	3.21	2.65	1242.88	1200549.63	1380.98	1142856.27	521.78	654361.40	1050.55	762282.56
330455705210768	233.0	774.0	3.32	3.26	715.03	397187.14	833.01	364280.00	451.14	306160.11	765.88	278631.29

Figura 4.33: Resultado da pesquisa pelas variáveis do logradouro Rua Álvaro Ramos agrupando por código de setor censitário

A figura 4.33 mostra o resultado dos 18 setores correlacionados ao logradouro Rua Álvaro Ramos. Analisando a diferença entre os códigos de setores censitários e tendo em mente que seu código é formado pela concatenação de códigos de municípios e bairros, pode-se concluir que do total encontrado somente alguns pertencem ao bairro de Botafogo. Esse resultado pode ser melhorado se além da filtragem pelo logradouro também for realizada a filtragem pelo bairro como a figura 4.34 mostra.

```
In [52]: resultadoMerge[['CD_SETOR']+colunasVariaveis][
             (resultadoMerge.logradouro.str.contains('RUA ALVARO RAMOS')) & (resultadoMerge.Nome_do_bairro.str.contains('Botafogo')
                                                                            )].groupby('CD_SETOR').mean()
```

Out[52]:

CD_SETOR	V001	V002	V003	V004	V005	V006	V007	V008	V009	V010	V011	V012
330455705090028	229.0	585.0	2.55	1.56	4707.85	67832318.67	4856.30	69256694.89	3108.58	36214017.55	3867.48	42133859.13
330455705090029	228.0	659.0	2.89	2.95	2389.11	6376948.38	2487.30	6394886.98	1364.12	4076017.31	1983.34	4699531.82
330455705090030	262.0	642.0	2.45	1.14	5662.90	21285153.65	6156.34	20097114.50	4053.52	18318256.05	5274.03	17394817.99
330455705090061	165.0	419.0	2.54	2.26	4963.76	22631697.43	5118.88	22544363.49	2875.30	16223175.30	4079.14	18111467.92
330455705090062	142.0	354.0	2.49	2.05	4295.39	17117478.75	4551.83	16971049.45	2832.69	13354934.40	3788.57	14243753.03
330455705090064	265.0	671.0	2.53	1.79	3429.57	15134178.88	3786.81	15358766.26	2249.17	10172643.58	2949.65	11276974.10
330455705090463	166.0	420.0	2.53	2.42	5585.91	27624276.35	5906.12	27314654.31	3703.33	21958642.92	4692.58	23186396.09
330455705090466	229.0	532.0	2.32	1.32	5618.20	21522720.36	6012.00	20659773.12	4121.85	19766793.86	5111.95	19452770.39
330455705090470	163.0	387.0	2.37	1.59	3662.66	12335127.26	3731.33	12310005.43	2419.04	9609949.41	2984.85	10170740.06

Figura 4.34: Resultado da pesquisa pelas variáveis do logradouro Rua Álvaro Ramos do bairro de Botafogo

O número de setores resultantes foi reduzido para 9, considerando somente aqueles que estão presentes no logradouro Rua Álvaro Ramos do bairro de Botafogo.

Para que a relação entre o nome de logradouro e valor das variáveis seja um para um, resultando em apenas um objeto de resultado e pesquisa, é feita uma média como mostra a figura 4.35. Tem-se então o resultado da média dos valores das variáveis dos 9 setores censitários presentes na Rua Álvaro Ramos do bairro de Botafogo.

```
In [71]: resultadoVariaveis = resultadoMerge[['CD_SETOR']+colunasVariaveis][
             (resultadoMerge.logradouro.str.contains('RUA ALVARO RAMOS')) & (resultadoMerge.Nome_do_bairro.str.contains('Botafogo')
                                                                             )].groupby('CD_SETOR').mean().sum()
         qtdSetores = len(resultadoMerge[['CD_SETOR']+colunasVariaveis][
             (resultadoMerge.logradouro.str.contains('RUA ALVARO RAMOS')) & (resultadoMerge.Nome_do_bairro.str.contains('Botafogo')
                                                                             )].groupby('CD_SETOR').mean())
         resultado = resultadoVariaveis.apply(lambda x: '{:f}'.format((x/qtdSetores)))
         resultado

Out[71]: V001         205.444444
         V002         518.777778
         V003           2.518889
         V004           1.897778
         V005        4479.483333
         V006    23539988.858889
         V007        4734.101111
         V008    23434145.381111
         V009        2969.733333
         V010    16632714.486667
         V011        3859.065556
         V012    17852256.725556
         dtype: object
```

Figura 4.35: Resultado da média das variáveis para o logradouro Rua Álvaro Ramos do bairro de Botafogo

O resultado obtido foi comparado com o resultado que se tem levando em consideração a abordagem que utiliza a API de Geocodificação do Google. Primeiramente é necessário obter as coordenadas geográficas de um endereço específico. A figura 4.36 mostra a obtenção das coordenadas do endereço Rua Álvaro Ramos 405, Botafogo presentes no corpo de um objeto JSON fruto de uma requisição GET utilizando uma chave para a API obtida no site Guia de Desenvolvedor do Google [21].

```
In [2]: chave = 'AIzaSyBdk94TtKhSCGPDDLGxgk6tGhl1UfiLdxE'
        endereco = 'Rua Álvaro Ramos 405, Botafogo'

        requestURL = 'https://maps.googleapis.com/maps/api/geocode/json?' \
                  'address='+endereco.replace(' ', '+') + \
                  '&key=' + chave

        result = requests.get(requestURL)
        data = result.json()
        latitude = str(data['results'][0]['geometry']['location']['lat'])
        longitude = str(data['results'][0]['geometry']['location']['lng'])
        coordDict = {'latitude': latitude, 'longitude':longitude}
        coordDict

Out[2]: {'latitude': '-22.9578552', 'longitude': '-43.1839092'}
```

Figura 4.36: Obtenção das coordenadas geográficas do endereço Rua Álvaro Ramos 405, Botafogo

Preenchendo o campo de busca da ferramenta online Google Maps com as coordenadas encontradas obtém-se o mapa mostrado na figura 4.37. O endereço exibido mostra-se congruente com o endereço utilizado no processo de geocodificação.

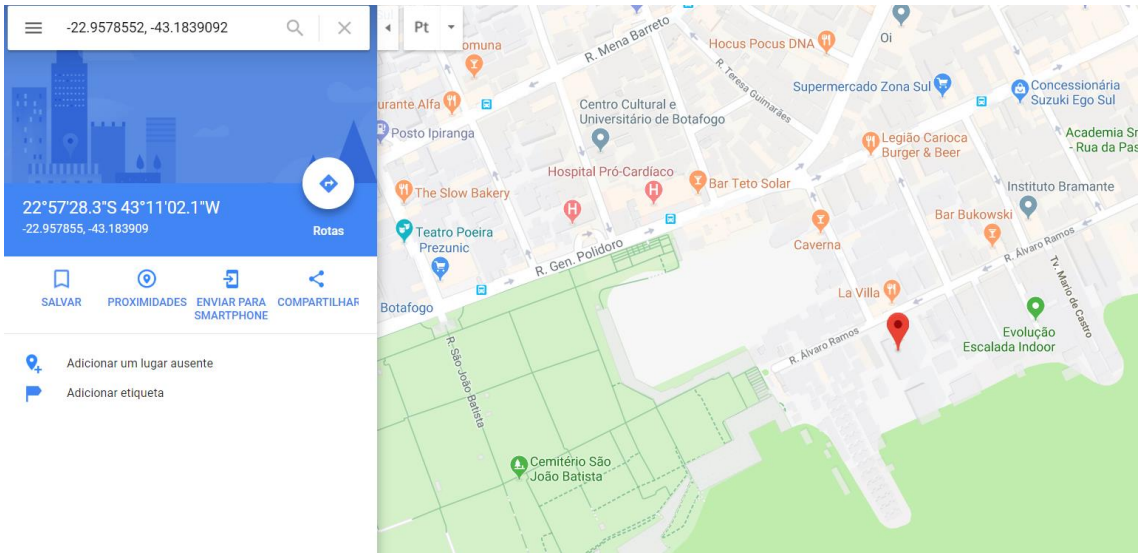


Figura 4.37: Resultado da pesquisa pelas coordenadas -22.9578552, -43.1839092 no Google Maps

O próximo passo para encontrar o setor censitário referente ao endereço em questão é relacionar as coordenadas geográficas com a área do polígono que descreve um setor. Para isto são utilizadas as bibliotecas *fiona* e *shapely:* a primeira interpreta o conteúdo de arquivos *shapefile* presentes no repositório Malha de Setores Censitários Divisões Intramunicipais e a segunda cria um ponto para as coordenadas encontradas assim como o polígono de cada setor interpretado através da biblioteca *fiona*. A figura 4.38 mostra a obtenção do código do setor censitário.

```
In [5]: arquivoSHP = '/home/pedrohenrique/Documents/TCC/DATA/rj_setores_censitarios/33SEE250GC_SIR.shp'

        with fiona.open(arquivoSHP) as fiona_collection:
            for fiona_item in fiona_collection:
                shape = shapely.geometry.asShape(fiona_item['geometry'])
                properties = fiona_item['properties']
                point = shapely.geometry.Point(float(coordDict['longitude']), float(coordDict['latitude']))
                if shape.contains(point):
                    codigoSetor = properties['CD_GEOCODI']
                    break
        codigoSetor
Out[5]: u'330455705090030'
```

Figura 4.38: Obtenção do código de setor censitário a partir do repositório Malha de Setores Censitários Divisões Intramunicipais

Em seguida são pesquisadas as variáveis do setor censitário de código 3304557050900030 no arquivo Basico_RJ.csv como mostra a figura 4.39.

```
In [10]: basicoRJ = pd.read_csv('/home/pedrohenrique/Documents/TCC/DATA/Basico_RJ.csv', sep=';', encoding='ISO-8859-1')
         colunasVariaveis = ['V001','V002','V003','V004','V005','V006','V007','V008','V009','V010','V011','V012']
         basicoRJ[colunasVariaveis][basicoRJ.Cod_setor == int(codigoSetor)]
```

Out[10]:

	V001	V002	V003	V004	V005	V006	V007	V008	V009	V010	V011	V012
12891	262,0	642,0	2,45	1,14	5662,9	21285153,65	6156,34	20097114,5	4053,52	18318256,05	5274,03	17394817,99

Figura 4.39: Variáveis do arquivo Basico_RJ.csv para o setor censitário de código 330455705090030

A tabela 4.3 compara os resultados das duas abordagens. Nota-se uma diferença nos valores dos resultados obtidos. É esperada uma maior confiabilidade aos dados obtidos recorrendo ao uso da API de geocodificação do Google por se tratar de um valor exato e não uma média. Apesar da clara vantagem em optar pela abordagem online, a abordagem offline encontra seu mérito em sistemas e situações em que uma conexão com a Internet não é possível de ser realizada.

Tabela 4.3: Comparação do resultado das variáveis para diferentes abordagens

	OFFLINE	ONLINE
V001	205.4	262.0
V002	518.7	642.0
V003	2.52	2.45
V004	1.90	1.14
V005	4479.48	5662.9
V006	23539988.86	21285153.65
V007	4734.10	6156.34
V008	23434145.38	20097114.5
V009	2969.73	4053.52
V010	16632714.48	18318256.05
V011	3859.06	5274.03
V012	17852256.72	17394817.99

Com os resultados da extração e transformação validados passa-se a próxima fase aonde serão criados métodos de consulta que se favorecem do produto obtido nas etapas anteriores.

4.7 – Consulta

Por meio do conteúdo levantado nas etapas anteriores do projeto será agora criado um módulo de consulta aos dados dos repositórios de dados censitários

Para auxiliar o módulo de consulta foi também definido um arquivo de configuração composto por uma listagem de chaves e valores visando facilitar a utilização da ferramenta a ser criada flexibilizando sua aplicação a partir de definições globais. A figura 4.40 mostra o conteúdo do arquivo config.ini.

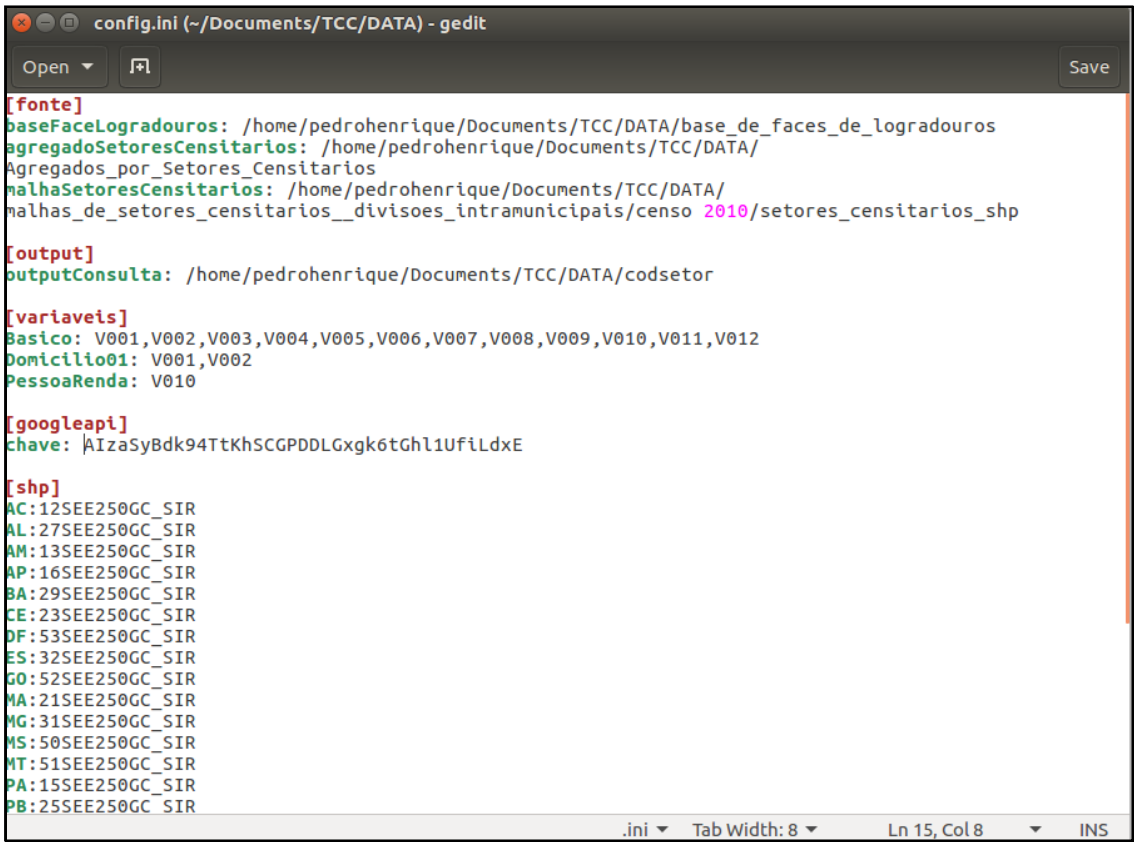


Figura 4.40: Arquivo de configuração config.ini

O arquivo de configuração é composto pelas seções fonte, output, variáveis, googleapi e shp. As descrições destas seções são apresentadas conforme se segue:

- Fonte: local do sistema do usuário em que os repositórios estão gravados;
- Output: local em que o output da consulta será gravado;
- Variáveis: variáveis de interesse para consulta;

- Google API: chave utilizada na API de geocodificação do Google;
- SHP: Arquivo shapefile referente a cada UF.

Sob esta ótica, passa-se a exibição das funções do módulo de consulta. A função *obtemInfoLogradouro* recebe como parâmetros o nome do logradouro, a sigla da UF do logradouro, o nome do bairro, o nome do arquivo de variáveis, a variável que se deseja obter e o caminho para o arquivo de configuração. A função é mostrada na figura 4.41.

```
def obtemInfoLogradouro(nomeLogradouro, siglaUF, nomeBairro, nomeMunicipio, nomeArqVariaveis, variavel, pathConfig):
    resultado = None
    fatorConfianca = None
    erro = None
    qtd = None
    listaSetores = None
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if len(nomeLogradouro)==0:
        erro = 'Nome de logradouro invalido.'
        resultadoDict = {'resultado': resultado,
                         'fatorConfianca': fatorConfianca,
                         'erro': erro, 'qtdSetores': qtd,
                         'listaSetores':listaSetores,
                         'municipiosDistintos': False,
                         'bairrosDistintos': False}
        return resultadoDict
    if not siglaUF.upper() in listaUF:
        erro = 'Sigla de UF invalida.'
        resultadoDict = {'resultado': resultado,
                         'fatorConfianca': fatorConfianca,
                         'erro': erro, 'qtdSetores': qtd,
                         'listaSetores':listaSetores,
                         'municipiosDistintos': False,
                         'bairrosDistintos': False}
        return resultadoDict
    if len(pathConfig)==0:
        erro = 'Arquivo de configuracao invalido.'
        resultadoDict = {'resultado': resultado,
                         'fatorConfianca': fatorConfianca,
                         'erro': erro,
                         'qtdSetores': qtd,
                         'listaSetores':listaSetores,
                         'municipiosDistintos': False,
                         'bairrosDistintos': False}
        return resultadoDict
    nomeLogradouro = formataTexto(nomeLogradouro.decode('UTF-8'))
    nomeLogradouro = nomeLogradouro.upper()
    if ('AV. ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('AV.','AVENIDA')
    elif ('AV ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('AV', 'AVENIDA')
    elif ('R. ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('R.', 'RUA')
    elif ('R ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('R ', 'RUA')
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    pathVariaveis = configuracoes['fonte']['agregadoSetoresCensitarios']
    frameETL = EXTRACAO.extrairEstado(siglaUF, pathConfig)
    frameETL = TRANSFORMACAO.transformar(frameETL)
    extrairZipUF(pathVariaveis, siglaUF, '')
    arqVariaveis = configuracoes['fonte']['agregadoSetoresCensitarios']
    pathBasico = arqVariaveis + '/' + siglaUF + '/' + 'Basico_'+siglaUF.upper()+'.csv'
    arqVariaveis = arqVariaveis + '/' + siglaUF + '/' + nomeArqVariaveis + '.csv'
    listaSetores = obtemListaSetores(frameETL, pathBasico, nomeLogradouro, nomeBairro, nomeMunicipio)
    if len(listaSetores)==0:
        erro = 'Nome de logradouro nao encontrado.'
        resultadoDict = {'resultado': resultado,
                         'fatorConfianca': fatorConfianca,
                         'erro': erro, 'qtdSetores': qtd,
                         'listaSetores':listaSetores,
                         'municipiosDistintos': False,
                         'bairrosDistintos': False}
        return resultadoDict
    elif len(listaSetores)>1:
        bMunicipio = checaMunicipiosDistintos(listaSetores)
        bBairro = checaBairrosDistintos(listaSetores)
    else:
        bMunicipio = False
        bBairro = False
    qtd = len(listaSetores)
    frameVariaveis = pd.read_csv(arqVariaveis, sep=';', encoding='ISO-8859-1')
    frameVariaveis = converteColunaVariaveis(frameVariaveis)
    frame = mergeVariaveis(frameETL, frameVariaveis)
    frameConsulta = frame[['CD_SETOR',variavel]][frame.CD_SETOR.isin(listaSetores)]
    soma = frameConsulta.groupby('CD_SETOR').mean().sum()
    resultado = float(soma)/float(qtd)
    fatorConfianca = float(100) / float(qtd)
    shutil.rmtree(pathVariaveis+'/'+siglaUF)
    resultadoDict = {'resultado': resultado,
                     'fatorConfianca': fatorConfianca,
                     'erro': erro,
                     'qtdSetores': qtd,
                     'listaSetores': listaSetores,
                     'municipiosDistintos': bMunicipio,
                     'bairrosDistintos': bBairro}
    return resultadoDict
```

Figura 4.41: Função obtemInfoLogradouro

A função *obtemInfoLogradouro* realiza o processo de extração e transformação por completo e exibe como retorno um dicionário contendo o resultado da variável desejada, o fator de confiança do resultado, quantos setores foram utilizados para realizar a média e a lista destes respectivos setores. A figura 4.42 mostra a execução da função.

Para o resultado da função é adotado um fator de confiança com a intenção de avaliar as condições em que o resultado foi gerado. Neste trabalho, o fator de confiança é dado por:

Fator de Confiança (%) = 100 / (número de setores censitários da consulta)

```
In [6]: nomeLogradouro = 'Rua Alvaro Ramos'
        siglaUF = 'RJ'
        nomeBairro = 'Botafogo'
        nomeArqVariaveis = 'Basico_RJ'
        variavel = 'V009'
        pathConfig = '/home/pedrohenrique/Documents/TCC/DATA/config.ini'

        resultado = obtemInfoLogradouro(nomeLogradouro, siglaUF, nomeBairro, nomeArqVariaveis, variavel, pathConfig)
        resultado

Out[6]: {'erro': None,
         'fatorConfianca': 11.11111111111111,
         'listaSetores': [330455705090028,
          330455705090029,
          330455705090030,
          330455705090061,
          330455705090062,
          330455705090064,
          330455705090463,
          330455705090466,
          330455705090470],
         'qtdSetores': 9,
         'resultado': 2969.733333333333}
```

Figura 4.42: Execução da funçõao obtemInfoLogradouro

Para auxiliar a função *obtemInfoLogradouro* foi criada uma função *formataTexto* com o intuito de formatar nomes de logradouros que possuam caracteres especiais como acentos ou cedilha. A figura 4.43 mostra a função *formataTexto*.

```
def formataTexto(texto):
    return unicodedata.normalize('NFKD', texto).encode('ASCII', 'ignore')
```

Figura 4.43: Função formataTexto

Além da *formataTexto* também foram criadas outras funções auxiliares. A função *mergeVariaveis* (ver figura 4.44) realiza a união dos *dataframes* que possuem os dados dos repositórios Agregados por Setores Censitários e Base de Faces de Logradouros.

```
def mergeVariaveis(frameETL, frameVariaveis):
    merge = frameETL.merge(frameVariaveis, left_on='CD_SETOR', right_on='Cod_setor', how='inner')
    return merge
```

Figura 4.44: Função mergeVariaveis

A função *converteColunaVariaveis* (ver figura 4.45) converte o tipo de dado das colunas de variáveis para dados numéricos a fim que sejam manipulados em seguida.

```
def converteColunaVariaveis(varFrame):
    colList = []
    for colNome in list(varFrame.columns.values):
        if len(colNome)==4 and colNome[0]=='V':
            colList.append(colNome)
    varFrame[colList] = varFrame[colList].applymap(lambda x: str(x).replace(',','.'))
    varFrame[colList] = varFrame[colList].apply(pd.to_numeric, errors='coerce', axis=0)
    return varFrame
```

Figura 4.45: Função converteColunaVariaveis

Para a função *obtemInfoLogradouro* foi adotada uma gradação do resultado de acordo com o valor de média de setores censitários obtido na seção de validação, sendo assim, um resultado superior à média de seis setores censitários por logradouro um resultado ruim.

A função *removeAcento* (ver figura 4.46) tem como objetivo remover todos os casos de acentuação e presença de caracteres com cedilha em um *DataFrame*. Já a função *checaMunicipiosDistintos* e *chechaBairrosDistintos* (ver figura 4.46) avalia se na lista de códigos de setores censitários obtidos estão presentes setores censitários de municípios ou bairros distintos.

```
def removeAcentos(frame):
    cols = frame.select_dtypes(include=[np.object]).columns
    frame[cols] = frame[cols].apply(lambda x: x.str.normalize('NFKD').
        str.encode('ascii', errors='ignore').
        str.decode('utf-8'))
    return frame

def checaMunicipiosDistintos(lista):
    munBase = str(lista[0])[:7]
    for mun in lista:
        if munBase != str(mun)[:7]:
            return True
    return False

def checaBairrosDistintos(lista):
    munBase = str(lista[0])[:10]
    for mun in lista:
        if munBase != str(mun)[:10]:
            return True
    return False
```

Figura 4.46: Funções removeAcentos, checaMunicipiosDistintos e checaBairrosDistintos

Por final, para fazer a pesquisa pelos setores censitários considerados para obtenção do resultado da função *obtemInfoLogradouro* a função *obtemListaSetores* (ver figura 4.47) é chamada, utilizando como filtro, não obrigatório, o nome do bairro do logradouro e o nome do município do logradouro.

```
def obtemListaSetores(frameETL, pathBasico, nomeLogradouro, nomeBairro, nomeMunicipio):
    frameBasico = pd.read_csv(pathBasico, sep=';', encoding='ISO-8859-1')
    frame = mergeVariaveis(frameETL,frameBasico)
    frame = removeAcentos(frame)
    nomeLogradouro = formataTexto(nomeLogradouro.decode('UTF-8'))
    nomeLogradouro = nomeLogradouro.upper()
    if len(nomeBairro)>0:
        frame['Nome_do_bairro'] = frame['Nome_do_bairro'].str.upper()
        nomeBairro = formataTexto(nomeBairro.decode('UTF-8'))
        nomeBairro = nomeBairro.upper()
    if len(nomeMunicipio)>0:
        frame['Nome_do_municipio'] = frame['Nome_do_municipio'].str.upper()
        nomeMunicipio = formataTexto(nomeMunicipio.decode('UTF-8'))
        nomeMunicipio = nomeMunicipio.upper()
    if len(nomeBairro)>0 and len(nomeMunicipio)>0:
        listaSetores = frame[['CD_SETOR']][(
            frame.logradouro.str.contains(nomeLogradouro)) & (
            frame.Nome_do_bairro.str.contains(nomeBairro)) & (
            frame.Nome_do_municipio.str.contains(nomeMunicipio))]
    elif len(nomeBairro)>0 and len(nomeMunicipio)==0:
        listaSetores = frame[['CD_SETOR']][(
            frame.logradouro.str.contains(nomeLogradouro)) & (
            frame.Nome_do_bairro.str.contains(nomeBairro))]
    elif len(nomeBairro)==0 and len(nomeMunicipio)>0:
        listaSetores = frame[['CD_SETOR']][(
            frame.logradouro.str.contains(nomeLogradouro)) & (
            frame.Nome_do_municipio.str.contains(nomeMunicipio))]
    else:
        listaSetores = frame[['CD_SETOR']][(
            frame.logradouro.str.contains(nomeLogradouro))]
    return listaSetores['CD_SETOR'].unique()
```

Figura 4.47: Função obtemListaSetores

A próxima função criada para o módulo é *obtemSetorCensitario*. A partir de dados de nome de logradouro, número de endereço, nome do bairro, sigla da UF, nome do município e caminho do arquivo de configuração é retornado um código de setor censitário correspondente. A função utiliza a API de geocodificação do Google para obter as coordenadas geográficas e realiza um cruzamento destas coordenadas com os polígonos de setores censitários presentes no repositório Malha de Setores Censitários Divisões Intramunicipais. Neste momento é utilizado o parâmetro referente aos arquivos *shapefile* configurados no config.ini. A figura 4.48 mostra a função *obtemSetorCensitario* e a função auxiliar *obtemGeocodificacao*.

```
def obtemGeocodificacao(endereco, chave):
    requestURL = 'https://maps.googleapis.com/maps/api/geocode/json?' \
              'address='+endereco.replace(' ', '+') + \
              '&key=' + chave
    result = requests.get(requestURL)
    data = result.json()
    latitude = str(data['results'][0]['geometry']['location']['lat'])
    longitude = str(data['results'][0]['geometry']['location']['lng'])
    coordDict = {'latitude': latitude, 'longitude':longitude}
    return coordDict


def obtemSetorCensitario(nomeLogradouro, numero, nomeBairro, siglaUF, pathConfig, nomeMunicipio):
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if len(nomeLogradouro)==0:
        print('Nome de logradouro invalido.')
        return None
    if not siglaUF.upper() in listaUF:
        print('Sigla de UF invalida.')
        return None
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    if len(numero)==0:
        print('Numero do endereco invalido.')
        return None
    siglaUF = siglaUF.lower()
    endereco = nomeLogradouro + ' ' +numero
    if len(nomeBairro)>0:
        endereco = endereco + ' ' + nomeBairro
    if len(nomeMunicipio)>0:
        endereco = endereco + ' ' + nomeMunicipio
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    chaveAPI = configuracoes['googleapi']['chave']
    if len(chaveAPI)==0:
        print('Chave da API nao configurada. Verifique o arquivo config.ini.')
        return None
    coord = obtemGeocodificacao(endereco, chaveAPI)
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    nomeArqSHP = configuracoes['shp'][siglaUF.upper()]
    pathSHP = configuracoes['fonte']['malhaSetoresCensitarios'] + '/' + siglaUF + '/'
    arquivoZip = zipfile.ZipFile(pathSHP+siglaUF+'_setores_censitarios.zip')
    arquivoZip.extractall(pathSHP)
    arquivoSHP = pathSHP + nomeArqSHP + '.shp'
    with fiona.open(arquivoSHP) as fiona_collection:
        for fiona_item in fiona_collection:
            shape = shapely.geometry.asShape(fiona_item['geometry'])
            properties = fiona_item['properties']
            point = shapely.geometry.Point(float(coord['longitude']), float(coord['latitude']))
            if shape.contains(point):
                arquivoZip.close()
                os.remove(pathSHP+nomeArqSHP+'.shp')
                os.remove(pathSHP+nomeArqSHP+'.shx')
                os.remove(pathSHP+nomeArqSHP+'.prj')
                os.remove(pathSHP+nomeArqSHP+'.dbf')
                return properties['CD_GEOCODI']
    arquivoZip.close()
    os.remove(pathSHP+nomeArqSHP+'.shp')
    os.remove(pathSHP+nomeArqSHP+'.shx')
    os.remove(pathSHP+nomeArqSHP+'.prj')
    os.remove(pathSHP+nomeArqSHP+'.dbf')
    return None
```

Figura 4.48: Funções obtemGeocodificacao e obtemSetorCensitario

A figura 4.49 mostra a execução da função *obtemSetorCensitario*.

```
In [14]: nomeLogradouro = 'Rua Alvaro Ramos'
         siglaUF = 'RJ'
         numero = '405'
         nomeBairro = 'Botafogo'
         nomeMunicipio = 'Rio de Janeiro'
         pathConfig = '/home/pedrohenrique/Documents/TCC/DATA/config.ini'

         resultado = obtemSetorCensitario(nomeLogradouro, numero, nomeBairro, siglaUF, pathConfig, nomeMunicipio)
         resultado

Out[14]: u'330455705090030'
```

Figura 4.49: Execução da função obtemSetorCensitario

Uma vez munido do código setor censitário abre-se um leque de opções de consultas. Primeiramente é realizada a consulta pela descrição do setor, ou seja, os logradouros que o compõem, através da função *obtemDescricaoSetorCensitario*. A figura 4.50 mostra a função.

```
def obtemDescricaoSetorCensitario(codSetorCensitario, siglaUF, pathConfig):
    if len(codSetorCensitario)!=15:
        print('Codigo de setor censitario invalido.')
        return None
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if siglaUF.upper() not in listaUF:
        print('Sigla de UF invalida.')
        return None
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    frameETL = extrairEstado(siglaUF, pathConfig)
    frameETL = transformar(frameETL, pathConfig)
    listaDescr = frameETL['logradouro'][frameETL.CD_SETOR==int(codSetorCensitario)].unique().tolist()
    return listaDescr
```

Figura 4.50: Função obtemDescricaoSetorCensitario

A figura 4.51 mostra a execução da função para um código de setor censitário da Cidade Universitária da UFRJ.

```
In [20]: codSetorCensitario = '330455705250278'
         siglaUF = 'RJ'
         pathConfig = '/home/pedrohenrique/Documents/TCC/DATA/config.ini'

         resultado = obtemDescricaoSetorCensitario(codSetorCensitario, siglaUF, pathConfig)
         resultado

Out[20]: ['AVENIDA BRIGADEIRO TROMPOWSKI',
          'AVENIDA HORACIO MACEDO',
          'AVENIDA PEDRO CALMON',
          'AVENIDA ATHOS DA SILVEIRA RAMOS',
          'RUA PROFESSOR RODOLPHO PAULO ROCCO',
          'RUA BRUNO LOBO',
          'RUA MANOEL FROTA MOREIRA',
          'RUA MARIA DOLORES LINS DE ANDRADE',
          'LARGO WANDA DE OLIVEIRA',
          'AVENIDA CARLOS CHAGAS FILHO',
          'RUA OCTAVIO CASTANHEDE',
          'RUA MONIZ DE ARAGAO',
          'RUA HELIO DE ALMEIDA',
          'RUA PAULO EMIDIO BARBOSA',
          'RUA PASCOAL LEMME',
          'RUA LUIZ RENATO CALDAS 1',
          'RUA CESAR PERNETTA',
          'PRACA JORGE MACHADO MOREIRA',
          'RUA MARIA PAULINA DE SOUZA',
          'RUA LOBO CARNEIRO',
          'RUA AFRANIO COUTINHO',
          'RUA MILTON SANTOS',
          'PRACA SAMIRA NAHID MESQUITA 2',
          'PONTE BRIGADEIRO TROMPOWSKI']
```

Figura 4.51: Execução da função obtemDescricaoSetorCensitario

Nota-se pelo resultado encontrado que a Cidade Universitária é basicamente composta por somente um único setor censitário de código 330455705250278.

Através do código de setor censitário pode-se também consultar diretamente o valor de uma variável presente nos arquivos de variáveis por meio da função *obtemInfoSetorCensitario*. A função obtém o mesmo resultado da função *obtemInfoLogradouro* com maior exatidão por estar contemplando necessariamente somente um único setor censitário. A figura 4.52 mostra a função e a figura 4.53 sua execução.

```python
def obtemInfoSetorCensitario(codSetorCensitario, siglaUF, nomeArqVariaveis, variavel, pathConfig):
    if len(codSetorCensitario)!=15:
        print('Codigo de setor censitario invalido.')
        return None
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    if len(variavel)==0:
        print('Arquivo de configuracao invalido.')
        return None
    if len(nomeArqVariaveis)==0:
        print('Arquivo de configuracao invalido.')
        return None
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if siglaUF.upper() not in listaUF:
        print('Sigla de UF invalida.')
        return None
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    pathVariaveis = configuracoes['fonte']['agregadoSetoresCensitarios']
    extrairZipUF(pathVariaveis, siglaUF, codSetorCensitario)
    arqVariaveis = configuracoes['fonte']['agregadoSetoresCensitarios']
    arqVariaveis = arqVariaveis + '/' + siglaUF + '/' + nomeArqVariaveis + '.csv'
    frameVariaveis = pd.read_csv(arqVariaveis, sep=';', encoding='ISO-8859-1')
    resultado = frameVariaveis[variavel][frameVariaveis.Cod_setor==int(codSetorCensitario)]
    shutil.rmtree(pathVariaveis+'/'+siglaUF)
    return resultado.values[0]
```

Figura 4.52: Função obtemInfoSetorCensitario

```
In [31]: siglaUF = 'RJ'
         nomeArqVariaveis = 'Basico_RJ'
         variavel = 'V009'
         pathConfig = '/home/pedrohenrique/Documents/TCC/DATA/config.ini'
         nomeLogradouro = 'AV. ATHOS DA SILVEIRA RAMOS'

         resultadoLogradouro = obtemInfoLogradouro(nomeLogradouro, siglaUF, nomeBairro, nomeArqVariaveis, variavel, pathConfig)
         resultadoSetorCensitario = obtemInfoSetorCensitario('330455705250278', siglaUF, nomeArqVariaveis, variavel, pathConfig)

         print('V009 - obtemInfoLogradouro: ' + str(resultadoLogradouro['resultado']) + ' - qtd setores: ' + str(resultadoLogradouro
         print('V009 - obtemInfoSetorCensitario: ' + str(resultadoSetorCensitario))

         V009 - obtemInfoLogradouro: 808.35 - qtd setores: 1
         V009 - obtemInfoSetorCensitario: 808,35
```

Figura 4.53: Execução da função obtemInfoSetorCensitario

A figura 4.53 também apresenta a comparação entre as duas abordagens de consulta levando em consideração o logradouro Avenida Athos da Silveira Ramos e o setor censitário que o mesmo se encontra. Os resultados são iguais uma vez que o logradouro consultado está presente em uma única descrição de setor censitário.

As funções *obtemInfoLogradouro* e *obtemInfoSetorCensitario* são auxiliadas pela função *extrairZipUF* que por sua vez é responsável por extrair o arquivo de variáveis correspondente a UF do setor ou logradouro informado. A figura 4.54 mostra o conteúdo da função.

```
def extrairZipUF(pathVariaveis, siglaUF, codSetorCensitario):
    for (dirpath, dirnames, filenames) in os.walk(pathVariaveis):
        for filename in filenames:
            if filename.endswith('.zip') and siglaUF in filename:
                if 'EXCETO' in filename.upper() and siglaUF=='SP' and codSetorCensitario.startswith('3550308'):
                    continue
                if not 'EXCETO' in filename.upper() and siglaUF=='SP' and not codSetorCensitario.startswith('3550308'):
                    continue
                source = zipfile.ZipFile(dirpath+'/'+filename, 'r')
                target = zipfile.ZipFile(dirpath+'/'+siglaUF+'.zip', 'w', zipfile.ZIP_DEFLATED)
                for file in source.filelist:
                    if '.csv' in file.filename:
                        if 'SP' in file.filename and 'Exceto' in file.filename:
                            target.writestr(file.filename[80:], source.read(file.filename))
                        elif 'SP' in file.filename and 'Exceto' not in file.filename:
                            target.writestr(file.filename[64:], source.read(file.filename))
                        else:
                            target.writestr(file.filename[48:], source.read(file.filename))
                target.close()
                source.close()
                zipUF = zipfile.ZipFile(dirpath+'/'+siglaUF+'.zip')
                zipUF.extractall(dirpath+'/'+siglaUF)
                os.remove(dirpath+'/'+siglaUF+'.zip')
```

Figura 4.54: Função extrairZipUF

A função é baseada no uso da biblioteca *zipfile*, devido a certas peculiaridades de como é estruturada a base do IBGE, foi necessário um ajuste em seu modo de execução.

Primeiramente há a necessidade de lidar com o fato da UF SP estar dividida em dois diretórios diferentes, o diretório dos setores censitários da capital paulista e os demais setores censitários do Estado de São Paulo.

Outro aspecto do repositório que tornou necessária uma execução personalizada foi o fato do nome do diretório contido no arquivo ZIP possuir a palavra “informações” com a letra C com cedilha, impossibilitando assim a execução plena dos métodos da biblioteca *zipfile*, precisando assim ser realizada uma leitura e escrita do conteúdo do arquivo ZIP por meio do método *writestr*.

Por último foi criada a função *obtemVariaveis* que retorna valores de um conjunto de variáveis de interesse configurado pelo usuário por meio do arquivo config.ini. A função também faz uso da *extrairZipUF* e recebe como parâmetro o código do setor censitário que se deseja buscar as variáveis, a UF do setor e o caminho para o arquivo de configuração. A seção do arquivo de configuração referente a essa função é mostrada na figura 4.55.

```
[output]
outputConsulta: /home/pedrohenrique/Documents/TCC/DATA/codsetor

[variaveis]
Basico: V001,V002,V003,V004,V005,V006,V007,V008,V009,V010,V011,V012
Domicilio01: V001,V002
PessoaRenda: V010
```

Figura 4.55: Configuração do arquivo config.ini necessária para a execução da função obtemVariaveis

De acordo com a configuração da figura 4.56 as variáveis V001 a V012 do arquivo Basico_UF.csv, V001 e V002 do arquivo Domicilio01_UF.csv e V010 do arquivo PessoaRenda_UF.csv serão gravadas em um diretório no caminho definido pelo parâmetro outputConsulta.

A figura 4.56 mostra o conteúdo da função *obtemVariaveis*.

```
def obtemVariaveis(codSetorCensitario, siglaUF, pathConfig):
    if len(codSetorCensitario)!=15:
        print('Codigo de setor censitario invalido.')
        return None
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if siglaUF.upper() not in listaUF:
        print('Sigla de UF invalida.')
        return None
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    outputConsulta = configuracoes['output']['outputConsulta']
    listaArquivosVariaveis = configuracoes['variaveis'].keys()
    pathVariaveis = configuracoes['fonte']['agregadoSetoresCensitarios']
    extrairZipUF(pathVariaveis, siglaUF, codSetorCensitario)
    data = str(datetime.datetime.now())[:-7]
    for arquivoVariaveis in listaArquivosVariaveis:
        listaVariaveis = configuracoes['variaveis'][arquivoVariaveis].split(',')
        for (dirpath, dirnames, filenames) in os.walk(pathVariaveis+'/'+siglaUF):
            for filename in filenames:
                if filename.endswith('.csv') and arquivoVariaveis.upper() in filename.upper():
                    frame = pd.read_csv(dirpath+'/'+filename, sep=';', encoding='ISO-8859-1')
                    frame = frame[frame.Cod_setor==int(codSetorCensitario)]
                    frame = frame[listaVariaveis]
                    if not os.path.exists(outputConsulta.replace('codsetor', codSetorCensitario)+'_'+data):
                        os.makedirs(outputConsulta.replace('codsetor', codSetorCensitario)+'_'+data)
                    frame.to_csv(outputConsulta.replace('codsetor', codSetorCensitario+'_'+data+'/'+filename),
                                 index=False)
    shutil.rmtree(pathVariaveis+'/'+siglaUF)
    return('Resultado em '+outputConsulta.replace('codsetor',codSetorCensitario)+'_'+data)
```

Figura 4.56: Função obtemVariaveis

A figura 4.57 mostra a execução da função *obtemVariaveis*.

```
In [4]: nomeLogradouro = 'Rua Constante Ramos'
        numero = '150'
        siglaUF = 'RJ'
        nomeBairro = 'Copacabana'
        nomeMunicipio = 'Rio de Janeiro'
        pathConfig = '/home/pedrohenrique/Documents/TCC/DATA/config.ini'

        codSetorCensitario = obtemSetorCensitario(nomeLogradouro, numero, nomeBairro, siglaUF, pathConfig, nomeMunicipio)
        obtemVariaveis(codSetorCensitario, siglaUF, pathConfig)

Out[4]: u'Resultado em /home/pedrohenrique/Documents/TCC/DATA/330455705100246_2018-05-09 20:00:02'
```

Figura 4.57: Execução da função obtemVariaveis

Para verificar o resultado da função *obtemVariaveis* é necessário consultar o diretório apontado no retorno como mostra a figura 4.58.

Figura 4.58: Caminho apontado pelo resultado da função obtemVariaveis

A figura 4.59 mostra o conteúdo do arquivo Dominicilio01_RJ.csv.

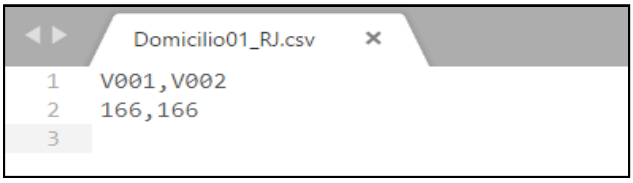


Figura 4.59: Conteúdo do arquivo resultante da execução da função obtemVariaveis

Uma função equivalente foi desenvolvida levando em consideração a abordagem offline, porém seu resultado, como visto anteriormente, é limitado a uma média de valores de dados censitários, sendo assim sensível à quantidade de setores censitários obtidos para o logradouro consultado.

A figura 4.60 mostra o corpo da função *obtemVariaveisLogradouro*.

```
def obtemVariaveisLogradouro(nomeLogradouro, siglaUF, nomeBairro, nomeMunicipio, pathConfig):
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    if siglaUF.upper() not in listaUF:
        print('Sigla de UF invalida.')
        return None
    nomeLogradouro = formataTexto(nomeLogradouro.decode('UTF-8'))
    nomeLogradouro = nomeLogradouro.upper()
    if ('AV. ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('AV.','AVENIDA')
    elif ('AV ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('AV', 'AVENIDA')
    elif ('R. ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('R.', 'RUA')
    elif ('R ' in nomeLogradouro):
        nomeLogradouro = nomeLogradouro.replace('R ', 'RUA')
    configuracoes = configparser.ConfigParser()
    configuracoes.read(pathConfig)
    outputConsulta = configuracoes['output']['outputConsulta']
    pathVariaveis = configuracoes['fonte']['agregadoSetoresCensitarios']
    listaArquivosVariaveis = configuracoes['variaveis'].keys()
    if len(listaArquivosVariaveis)==0:
        return None
    frameETL = EXTRACAO.extrairEstado(siglaUF, pathConfig)
    frameETL = TRANSFORMACAO.transformar(frameETL)
    if siglaUF=='SP':
        encontrou=False
        codSetorAux = 3550308
    if os.path.exists(pathVariaveis+'/SP'):
        shutil.rmtree(pathVariaveis+'/SP')
    if os.path.exists(pathVariaveis+'/SP'):
            os.remove(pathVariaveis+'/SP.zip')
        while codSetorAux<3550309:
            extrairZipUF(pathVariaveis,siglaUF,str(codSetorAux))
        files = os.listdir(pathVariaveis+'/SP/')
            for file in files:
                if 'Basico' in file:
                    nmArq = file
            frameSetores = mergeVariaveis(frameETL,pd.read_csv(pathVariaveis+'/SP/'+nmArq, sep=';', encoding='ISO-8859-1'))
            nomeBairro = formataTexto(nomeBairro.decode('UTF-8')).upper()
        testeSetores = frameSetores['CD_SETOR'][frameSetores.logradouro.str.contains(nomeLogradouro.upper())]
        listaSetores = frameSetores['CD_SETOR'][frameSetores.logradouro.str.contains(nomeLogradouro.upper())].unique().tolist()
            if len(listaSetores)>0:
                codSetorAux=codSetorAux+5
        else:
            codSetorAux = codSetorAux+1
    if len(listaSetores)==0:
        return None
    codSetorAux=codSetorAux-5
        extrairZipUF(pathVariaveis, siglaUF, str(codSetorAux))
    else:
        extrairZipUF(pathVariaveis, siglaUF, '')
    data = str(datetime.datetime.now())[:-7]
    for (dirpath, dirnames, filenames) in os.walk(pathVariaveis+'/'+siglaUF):
        listaAux = [x.split('_')[0].lower() for x in filenames]
        indexBasico = listaAux.index('basico')
        listaSetores = obtemListaSetores(frameETL, dirpath+'/'+filenames[indexBasico], nomeLogradouro, nomeBairro, nomeMunicipio)
        for arquivoVariaveis in listaArquivosVariaveis:
            if arquivoVariaveis in (x.split('_')[0].lower() for x in filenames):
                listaAux = [x.split('_')[0].lower() for x in filenames]
                index = listaAux.index(arquivoVariaveis)
                listaVariaveis = configuracoes['variaveis'][arquivoVariaveis].split(',')
        frameVariaveis = None
        frameVariaveis = pd.read_csv(dirpath+'/'+filenames[index], sep=';', encoding='ISO-8859-1')
        frameVariaveis = frameVariaveis[listaVariaveis][frameVariaveis.Cod_setor.isin(listaSetores)]
        frameVariaveis = converteColunaVariaveis(frameVariaveis)
        frameVariaveis = frameVariaveis.mean()
        frameVariaveis = pd.DataFrame(frameVariaveis).T
        if not os.path.exists(outputConsulta.replace('codsetor', nomeLogradouro)+'_'+data):
            os.makedirs(outputConsulta.replace('codsetor', nomeLogradouro)+'_'+data)
        frameVariaveis.to_csv(outputConsulta.replace('codsetor', nomeLogradouro+'_'+data+'/'+filenames[index]),
                              index=False)
    shutil.rmtree(pathVariaveis+'/'+siglaUF)
    return('Resultado em '+outputConsulta.replace('codsetor',nomeLogradouro)+'_'+data)
```

Figura 4.60: Corpo da função obtemVariaveisLogradouro

Para a execução da função foi necessária a criação de um tratamento especial para logradouros do Estado de São Paulo, sendo que a filtragem pelo nome do bairro teve de ser descartada uma vez que o arquivo de variáveis referente a UF SP não segue o modelo das demais UFs aonde os nomes dos bairros são listados no arquivo.

Tem-se então como objeto final do projeto os módulos de extração, transformação e consulta e suas respectivas funções.

4.8 – Documentação

Para auxiliar na utilização do pacote de funções foi criada uma documentação em HTML abordando os módulos e suas funções. Para isso foi utilizada a ferramenta de geração automática de documentação Sphinx.

Uma vez tendo instalado o pacote Sphinx [22] basta executar a função *sphinx-quickstart* no diretório que se deseja criar os arquivos de documentação e em seguida definir alguns parâmetros de configuração como nome do projeto, nome do autor e versão.

Para que a ferramenta gere automaticamente os arquivos HTML com a documentação dos módulos é necessário que as funções recebam comentários estruturados de uma forma especificada na documentação do Sphinx [22] como mostra a figura 4.61.

```
def extrairTodos(pathConfig):
    '''
    Realiza o processo de extração de dados do Repositório Base de Face de Logradouros de todos as UFs

    :type pathConfig: string
    :param pathConfig: Caminho do arquivo de configurações

    :return: DataFrame resultante da extração dos dados
    '''
    if len(pathConfig)==0:
        print('Arquivo de configuracao invalido.')
        return None
    frame = pd.DataFrame()
    listaUF = ['AC','AL','AM','AP','BA','CE','DF','ES','GO',
               'MA','MG','MS','MT','PA','PB','PE','PI','PR',
               'RJ','RN','RO','RR','RS','SC','SE','SP','TO']
    for siglaUF in listaUF:
        try:
            frame = frame.append(extrairEstado(siglaUF, pathConfig), ignore_index=True)
        except:
            print('Não foi possível realizar a extração da UF '+siglaUF)
            return None
    return frame
```

Figura 4.61: Exemplo de comentários contemplados na criação da documentação automática via Sphinx

Executando a função *sphinx-apidoc* os arquivos referentes aos módulos são criados. A figura 4.62 mostra a execução no terminal.

```
D:\Documentos D\TCC\PYTHON\projeto>sphinx-apidoc -o . ./scripts
Creating file .\CONSULTA.rst.
Creating file .\EXTRACAO.rst.
Creating file .\TRANSFORMACAO.rst.
Creating file .\modules.rst.

D:\Documentos D\TCC\PYTHON\projeto>
```

Figura 4.62: Execução da função sphinx-apidoc

Executando então a função *sphinx-build* temos a documentação criada em HTML. A figura 4.63 mostra o resultado da função no terminal.

```
D:\Documentos D\TCC\PYTHON\projeto>sphinx-build -b html . ./html
Running Sphinx v1.7.4
loading translations [pt_BR]... done
loading pickled environment... done
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 4 source files that are out of date
updating environment: 0 added, 4 changed, 0 removed
reading sources... [100%] modules
looking for now-outdated files... none found
pickling environment... done
checking consistency... D:\Documentos D\TCC\PYTHON\projeto\modules.rst: WARNING: document isn't included in any toctree
done
preparing documents... done
writing output... [100%] modules
generating indices... genindex py-modindex
writing additional pages... search
copying static files... done
copying extra files... done
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded, 1 warnings.

The HTML pages are in html.

D:\Documentos D\TCC\PYTHON\projeto>
```

Figura 4.63: Execução da função sphinx-build

O resultado da geração automática de documentação é disponibilizado em um diretório com o nome html criado no ponto de execução da função *sphinx*-build. Os arquivos gerados podem ser transferidos para um servidor de hospedagem de sites via FTP como também podem ser interpretados localmente em um browser de Internet apenas executando o arquivo *index.html*. As figuras a seguir mostram o resultado.

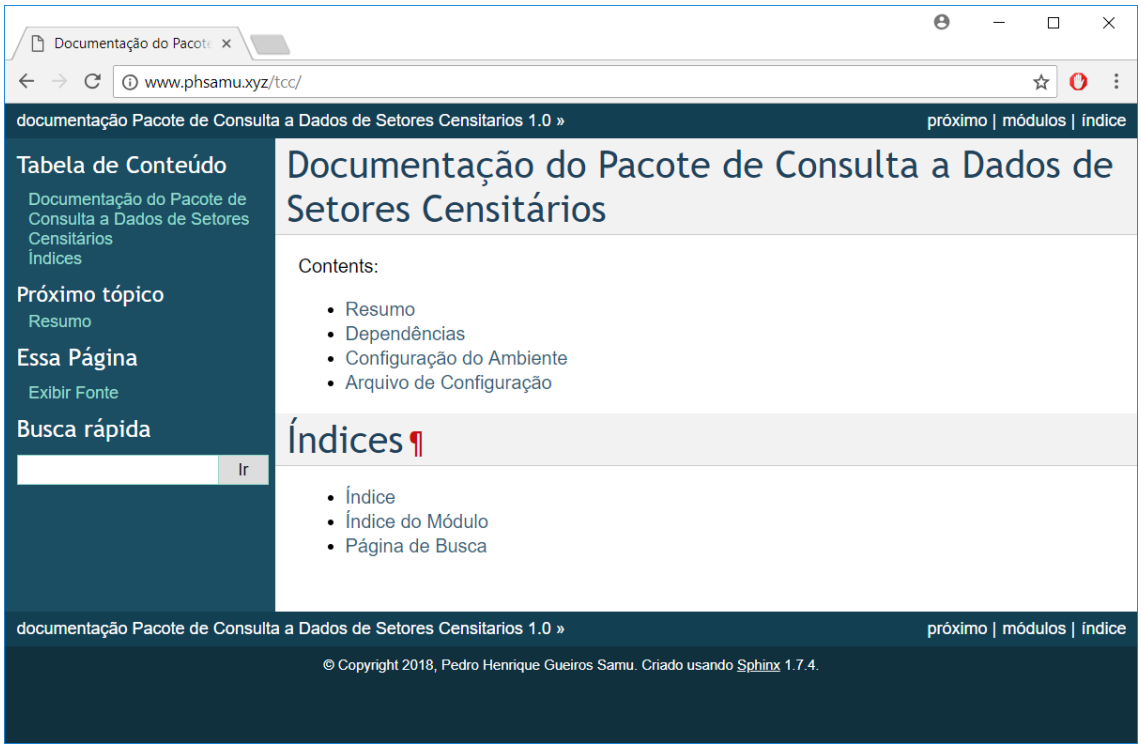


Figura 4.64: Página home do site de documentação

Figura 4.65: Página de módulos do site de documentação

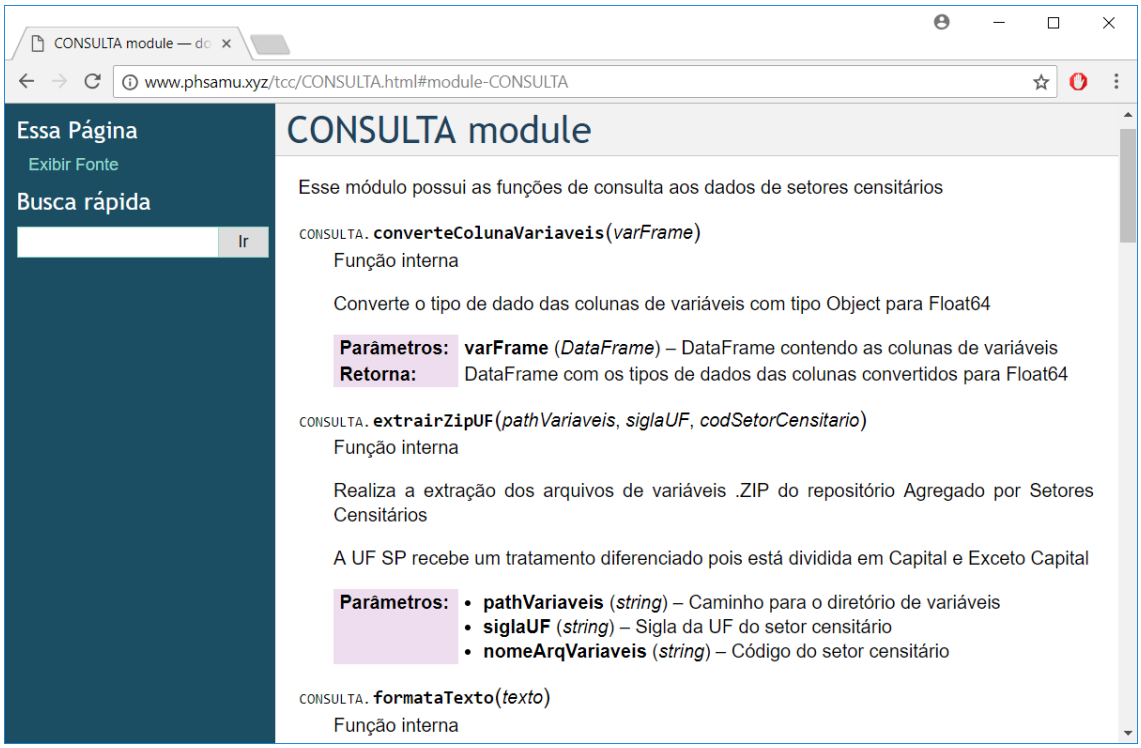


Figura 4.66: Página do módulo de consulta do site de documentação

Capítulo 5

Conclusão

5.1 – Conclusões

Tendo alcançado o objeto final do trabalho que fora estipulado nos capítulos iniciais resta agora uma seção para discutir os detalhes do processo como um todo. Primeiramente foram realizados estudos que permitiram a interpretação do ambiente que possibilitaria a implementação do projeto. Para isso foi analisado o conceito de setor censitário e sua relação com as demais formas de fragmentação do território no âmbito da execução do procedimento de recenseamento do Censo Demográfico de 2010 pelo IBGE.

Em seguida foi proposta uma discussão acerca das ferramentas e conceitos que permitiram com que o objetivo fosse alcançado. Em meio a diversas linguagens, modelos e frameworks disponíveis para soluções de ciência de dados foram escolhidos aqueles que viabilizaram uma implementação simples e eficaz do projeto. A escolha pela linguagem Python mostrou-se bastante útil pelo fato desta linguagem apresentar um alto grau de flexibilidade e conter em seu escopo diversas bibliotecas de manipulação de dados. Outra ferramenta crucial para o projeto foi a biblioteca *pandas* que possibilitou a análise e validação das etapas de desenvolvimento.

Considerando o conteúdo das seções citadas acima foi possível definir de forma clara o problema a ser resolvido assim como apontar as abordagens para a sua devida solução. A princípio o trabalho contemplava somente um método para obtenção da solução, menos complexo, porém com resultados não tão satisfatórios: interpretar o resultado da consulta por dados censitários de um logradouro como uma média de valores de dados de setores censitários presentes no mesmo. Durante a implementação dessa abordagem, realizando um estudo mais aprofundado das ferramentas disponíveis, foi estabelecido um novo caminho para alcançar o objetivo desejado, utilizando as bibliotecas *fiona* e *shapely* de manipulação de dados geográficos junto ao uso da API de

geocodificação do Google, possibilitando uma maior exatidão dos valores obtidos através de uma consulta.

No capítulo de implementação tal processo foi fragmentado na criação de módulos de extração e conversão, transformação e consulta, adotando como modelo base o conceito de ETL. Surgiram algumas adversidades ao longo da implementação, tais como a necessidade de automatizar o processo de extração do conteúdo de um arquivo ZIP que por sua vez possuía o caractere “ç” em seu caminho. Outro ponto que acrescentou mais um grau de complexidade ao projeto foi a divisão dos arquivos do Estado de São Paulo entre a capital e os demais municípios, comportamento não presentes nas demais Unidades Federativas, sendo assim havendo a necessidade de contornar tal adversidade.

Por fim, apesar de não ter sido contemplada inicialmente, houve a necessidade da criação de uma documentação para auxiliar aqueles que virão a utilizar o objeto resultante do trabalho. Uma forma simples de alcançar esse objetivo foi utilizando a ferramenta Sphinx para geração automática de documentação de códigos em Python.

O resultado obtido através das soluções desenvolvidas mostrou-se em conforme com os objetivos propostos inicialmente, levando em consideração as limitações encontradas. Assim, está entregue uma ferramenta capaz de relacionar dados cadastrais de endereço e dados de setores censitários de forma automática e simples.

5.2 – Trabalhos Futuros

Uma vez que se torna possível relacionar dados de endereços e dados de setores censitários, abre-se uma gama de possibilidades de incorporação da ferramenta capaz de realizar esta tarefa em sistemas que se beneficiam desses dados, principalmente sistemas que possuem um agente inteligente.

Outra forma de continuidade ao trabalho é desenvolver uma distribuição do pacote criado no formato de biblioteca, disponibilizando seu livre acesso através da hospedagem em um servidor ou repositório *git*.

Levando em consideração que em 2020 será realizado um novo Censo Demográfico é possível que um trabalho similar ou até mesmo uma adaptação do

presente trabalho, respeitando as possíveis mudanças na estrutura do processo realizado pelo IBGE, seja aplicada para o novo recenseamento. Por se tratar de um evento que se repete a cada década mostra-se válida a criação de um sistema com características genéricas pronto para atender as especificações do Censo Demográfico em questão.

Bibliografia

[1] IBGE, “IBGE | Memória | sínteses históricas | Históricos dos Censos | panorama introdutório”, https://memoria.ibge.gov.br/sinteses-historicas/historicos-dos-censos/panorama-introdutorio.html, 2013, (Acesso em 16 de janeiro 2017).

[2] _____, “Unesco premia IBGE por inovação tecnológica no Censo 2010 – Governo do Brasil”, http://www.brasil.gov.br/governo/2011/01/unesco-premia-ibge-por-tecnologia-no-censo-2010, 2011, (Acesso em 26 de dezembro 2017).

[3] _____, “Aparelhos usados no censo são destinados a programas sociais – Rede Brasil Atual”, http://www.redebrasilatual.com.br/cidadania/2010/08/aparelhos-usados-no-censo-serao-destinados-a-programas-sociais-1, 2010, (Acesso em 27 de dezembro 2017).

[4] IBGE, “IBGE | censo 2010 | materiais | guia do Censo | operação censitária”, https://censo2010.ibge.gov.br/materiais/guia-do-censo/operacao-censitaria.html, sem data, (Acesso em 27 de dezembro 2017).

[5] IBGE, “IBGE | censo 2010 | sobre | dimensões do Censo 2010”, https://censo2010.ibge.gov.br/sobre-censo/dimensoes-do-censo-2010.html, sem data, (Acesso em 27 de dezembro 2017).

[6] IBGE, *Manual do Recenseador - CD-1.09*. Rio de Janeiro, IBGE, 2010.

[7] Instituto Pereira Passos; IBGE, “Tabela 1172 – Índice de Desenvolvimento Humano Municipal (IDH), por ordem de IDH, segundo os bairros ou grupo de bairros – 2000 (XLS)”, 14 de janeiro de 2018, (Acesso em 16 de janeiro de 2018).

[8] IBGE, “Base de informações do Censo Demográfico 2010: Resultados do Universo por setor censitário, Documentação do Arquivo”, Rio de Janeiro, IBGE, 2011.

[9] IBGE, “CENSO DEMOGRÁFICO 2010; Divulgação da Base de Faces de Logradouros do CD 2010”, IBGE, sem data.

[10] ArcGIS, “ArcGIS – Meu Mapa”, https://www.arcgis.com/home/webmap/viewer.html, sem data, (Acesso em 4 de maio de 2018).

[11] Statistics Views, “Nate Silver: What I need from statisticians”, http://www.statisticsviews.com/details/feature/5133141/Nate-Silver-What-I-need-from-statisticians.html, 23 de agosto de 2013, (Acesso em 12 de março de 2018).

[12] Statistics Views, “Nate Silver: What I need from statisticians”, http://www.statisticsviews.com/details/feature/5133141/Nate-Silver-What-I-need-from-statisticians.html, 23 de agosto de 2013, (Acesso em 12 de março de 2018).

[13] Justin Megahan, “This is the difference between statistics and data science”, https://mixpanel.com/blog/2016/03/30/this-is-the-difference-between-statistics-and-data-science/, 30 de março de 2016, (Acesso em 12 de março de 2018).

[14] Will Oremus, “Here Are All the Different Genders You Can Be On Facebook”, http://www.slate.com/blogs/future_tense/2014/02/13/facebook_custom_gender_options_here_are_all_56_custom_options.html, 13 de fevereiro de 2014, (Acesso em 20 de janeiro de 2018).

[15] _____, “PYPL PopularitY of Programming Language”, http://pypl.github.io/PYPL.html, sem data, (Acesso em 13 de março de 2018).

[16] Karlijn Willems, “Choosing R or Python for Data Analysis? An Infographic”, https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis, 12 de maio de 2015, (Acesso em 13 de março de 2018).

[17] The Scipy community, “Data types – NumPy v1.13 Manual”, https://docs.scipy.org/doc/numpy-1.13.0/user/basics.types.html, sem data, (Acesso em 28 de fevereiro de 2018).

[18] Rank Brasil – Recordes Brasileiros, “Avenida mais extensa do país, Rank Brasil – Recordes Brasileiros”, http://www.rankbrasil.com.br/Recordes/Materias/0TAK/Avenida_Mais_Extensa_Do_Pais, 24 de janeiro de 2013, (Acesso em 1 de março de 2018).

[19] Departamento de Estradas de Rodagem, “Subsecretaria de Comunicação Social – Rodovia Amaral Peixoto (BR-106) receberá nova sinalização”, http://www.rj.gov.br/web/imprensa/exibeconteudo?article-id=1587960, 20 de maio de 2013, (Acesso em 1 de fevereiro de 2018).

[20] Ministério Público do Rio de Janeiro, “MP em Mapas – In Loco”, http://apps.mprj.mp.br/sistema/inloco/, sem data, (Acesso em 28 de março de 2018).

[21] Google, “Guia do desenvolvedor | Google Maps Geocoding API | Google Developers”, https://developers.google.com/maps/documentation/geocoding/intro?hl=pt-br, sem data, (Acesso em 6 de maio de 2018).

[22] Sphinx, “Sphinx documentation contentes – Sphinx 1.8.0+ documentation”, http://www.sphinx-doc.org/en/master/contents.html, sem data, (Acesso em 9 de maio de 2018).