UNIVERSIDADE FEDERAL FLUMINENSE

THIAGO DA CUNHA BORGES

ZEUS OLENCHUK GANIMI

EXTRAÇÃO DE DADOS COM WEB SCRAPING PARA ANÁLISE DA VARIAÇÃO DE PREÇO DE VEÍCULOS AUTOMOTORES

Niterói

2018

THIAGO DA CUNHA BORGES
ZEUS OLENCHUK GANIMI

EXTRAÇÃO DE DADOS COM WEB SCRAPING PARA ANÁLISE DA VARIAÇÃO DE PREÇO DE VEÍCULOS AUTOMOTORES

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Orientador:
Jean de Oliveira Zahn

NITERÓI
2018

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

B732 Borges, Thiago da Cunha
Extração de dados com *Web Scraping* para análise da variação de preço de veículos automotores / Thiago da Cunha Borges, Zeus Olenchuck Ganimi. – Niterói, RJ : [s.n.], 2018.
53 f.

Projeto Final (Tecnólogo em Sistemas de Computação) – Universidade Federal Fluminense, 2018.
Orientador: Jean de Oliveira Zahn.

1. Engenharia de software. 2. Python (Linguagem de programação de computador). I. Ganimi, Zeus Olenchuk. II. Título.

CDD 005.1

THIAGO DA CUNHA BORGES
ZEUS OLENCHUK GANIMI

EXTRAÇÃO DE DADOS COM WEB SCRAPING PARA ANÁLISE DA VARIAÇÃO DE PREÇO DE VEÍCULOS AUTOMOTORES

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Niterói, 22 de novembro de 2018.

Banca Examinadora:

_____

Prof. Jean de Oliveira Zahn, M.Sc. – Orientador
UFF – Universidade Federal Fluminense

_____

Prof. Leonardo Pio Vaconcelos, M.Sc. – Avaliador
UFF – Universidade Federal Fluminense

Dedico este trabalho ao meu Deus, à minha esposa e à minha avó. (Zeus)

Dedico este trabalho à minha esposa Denise, aos meus pais Vera e Pedro, aos meus sogros Lucia e Jorge por todo incentivo que recebi para chegar até aqui, à minha filha Maria Fernanda, para que todo o esforço empregado seja exemplo para sua vida e, sobretudo, a Deus que me deu forças para chegar até aqui. (Thiago)

AGRADECIMENTOS

A Deus, por ter me sustentado e por ter me dado mais uma vitória (a Ele seja toda a glória). A todos os que me ajudaram nesta caminhada, em especial minha esposa e minha avó. (Zeus)

A Deus, por ter nos dado saúde e forças para superar as dificuldades. Aos meus entes queridos, por terem me motivado constantemente a buscar meu sonho e apoiaram a minha decisão de buscar algo novo para minha vida. Em especial minha esposa Denise que, com seu esforço pessoal, me amparou em todos os momentos de dificuldades.

Ao projeto CEDERJ, e a todos os envolvidos, graças a eles pude realizar o sonho da minha vida de obter uma graduação.

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados, aos quais, sem nominar, terão os meus eternos agradecimentos. (Thiago)

“Para se ter sucesso, é necessário amar de verdade o que se faz. Caso contrário, levando em conta apenas o lado racional, você simplesmente desiste. É o que acontece com a maioria das pessoas”.

Steve Jobs

RESUMO

A previsão de acontecimentos é uma necessidade do homem, o qual, para orientar melhor suas decisões, buscou formas de conhecer os fatos futuros através de diversos métodos ao longo da história. Estimar eventos futuros com base na observação de dados de fácil interpretação é algo que todos nós fazemos, como saber que há grande chance de ocorrer chuvas quando o céu está negro pela densidade das nuvens e trovões. Porém, atualmente, as necessidades de prever eventos futuros demandam a análise de dados mais complexos e, portanto, métodos mais elaborados, os quais podem fornecer as respostas de que necessitamos, bem como oferecer maior probabilidade de acerto, do que por métodos empíricos. Diante de tal necessidade, será desenvolvida uma ferramenta, que implementa os conceitos de *Web Scraping* e de persistência de dados, a qual permite a aquisição automatizada de dados semiestruturados de *sites* da *Internet*, bem como o seu armazenamento estruturado, provendo uma base de dados de interesse para consultas e análises, as quais podem ser úteis para apoiar diversos processos, bem como a otimização de recursos. Por fim, apresentaremos uma amostra de dados de veículos automotivos, obtida do *site* da FIPE, através da ferramenta (fipe-*scraper*) objeto deste Trabalho.

Palavras-chaves: *Web Scraping, Python*, *PostgreSQL* e Engenharia de Software.

ABSTRACT

The prediction of events is a necessity of man, who, in order to better guide his decisions, sought ways to know the future facts through various methods throughout history. Estimating future events based on observation of easily interpreted data is something we all do, such as knowing that there is a high chance of rainfall when the sky is black by the density of clouds and thunder. However, nowadays, the need to predict future events demandates more complex data analysis and therefore more elaborate methods, which can provide the answers we need, as well as offer a greater probability of success, than by empirical methods. Facing this need, a tool will be developed, which implements the concepts of Web Scraping and data persistence, which allows the automated acquisition of semi structured data from Internet sites, as well as its structured storage, providing a database of interest for queries and analyzes, which can be useful to support several processes as well as resource optimization. Finally, we will present a sample of automotive vehicle data, obtained from the FIPE website, through the tool (fipe-scraper) object of this Work.

Key words: Web Scraping, Python, PostgreSQL and Software Engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX – *Asynchronous Javascript and XML*

API – *Application Programing Interface*

BD – Banco de Dados

BI – *Business Intelligence*

FIPE - Fundação Instituto de Pesquisas Econômicas

HTML – *HyperText Markup Language*

IDE – *Integrated Development Enviroment*

JSON – *JavaScript Object Notation*

ORM – *Object Relational Mapper*

SGBD – Sistema Gerenciador de Banco de Dados

WEB – *World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

Boa parte das pessoas, se não a maioria delas, deve ter tido a dúvida sobre qual o melhor automóvel adquirir, segundo diversos critérios, externos ou internos a cada pessoa. Fatores diversos como inflação, desvalorização, valor médio do seguro contra roubo, ou serviços de manutenção e reposição de partes podem ser fatores decisivos na escolha de um auto que atenda às necessidades de um dado consumidor [21].

Atualmente, as formas pelas quais um comprador de carro orienta sua escolha por um determinado veículo, se dá por meio de informações divulgadas em anúncios televisivos, *reviews* de especialistas, e outros meios de divulgação da informação [20]. No entanto, fazer uma análise comparativa, entre diversos modelos, apenas por estes meios, se torna algo custoso e propenso a erros, dadas as limitações humanas.

Considerando o alto investimento que um veículo automotor representa à renda de uma pessoa e necessidade de análise por parte do usuário, fica claro que uma escolha "ótima" (que oferece a melhor relação custo-benefício) é necessária para que se faça bom uso do valor investido.

Portanto, este trabalho propõe o desenvolvimento de uma ferramenta de obtenção de dados de *Websites* – procedimento conhecido como *Web Scraping* – com os quais, consultas e análises podem ser feitas a fim de apoiar o processo de decisão. Como caso de uso, uma amostra de dados foi obtida do site da FIPE[1], da qual alguns gráficos serão apresentados.

O presente trabalho subdivide-se desta forma: o Capítulo 2 apresenta os trabalhos relacionados ao tema, o Capítulo 3 o embasamento teórico dos conceitos utilizados, o Capítulo 4 a análise e o projeto da ferramenta, o Capítulo 5 a execução da ferramenta e o Capítulo 6 as conclusões e os trabalhos futuros.

[1] http://www.veiculos.fipe.org.br

2 TRABALHOS RELACIONADOS

Este Capítulo apresentará alguns trabalhos cujo conceito principal se assemelha ao proposto neste trabalho. Apenas um deles fala de automotores, o que é muito bom pois demonstra que as técnicas de *Web Scraping* e *BI* podem dar suporte a diversas áreas do conhecimento, como saúde, arte e qualquer outra que necessite da análise de dados obtidos na *Internet* de forma automatizada.

Os trabalhos citados nesse Capítulo mostram, quase na sua totalidade, os processos de aquisição automatizada, persistência local e análise dos dados obtidos para apoiar uma área ou necessidade específica, o que é, basicamente, a proposta deste trabalho de conclusão de curso.

2.1 UMA ANÁLISE EMPÍRICA DO MERCADO DE REVENDA DE CARROS USANDO RASPAGEM DE DADOS DA INTERNET (UNB)

Esta tese [22], de Patrick Franco Alves, apresenta uma análise sobre o mercado secundário de automóveis (carros usados), com o intuito de contribuir para a pesquisa econômica, ao analisar a depreciação dos autos. Uma das informações obtidas pelo estudo, por exemplo, é que o comprador do carro novo absorve maior perda de preço no momento da revenda do que aquele cujo auto tem idade superior ao ponto de inflexão (“o momento do tempo onde termina a fase de desvalorização mais intensa e começa a fase de estabilização do processo de depreciação”) no gráfico de evolução temporal da depreciação.

As análises foram feitas sobre os dados baixados de anúncios de venda de carros na *Internet,* pelo processo automatizado de *Web Scraping*, utilizando a linguagem *Python* e o *framework Scrapy*.

2.2 EXTRAÇÃO E ANÁLISE DE INFORMAÇÕES JURÍDICAS PÚBLICAS (IME – USP)

Este trabalho [23], de Alessandro Calò, apresenta algumas análises feitas com acórdãos (documentos) do Supremo Tribunal Federal, os quais foram extraídos pela técnica de *Web Scraping,* armazenados em um SGBD orientado a documentos e analisados por meio de *queries*.

Para incrementar o número de análises que podiam ser feitas com os documentos, o algoritmo *PageRank* foi usado para elencar quais documentos tinham a maior importância, e isso foi feito ao analisar se um acórdão tinha muitas citações ou se ele era citado por acórdãos importantes (com *PageRank* alto).

A raspagem[2] de dados dos acórdãos permitiu, no contexto deste trabalho correlato, contornar algumas dificuldades, como as limitações quanto às análises que podiam ser feitas, impostas pela plataforma de consulta aos documentos do STJ, e a morosidade da entrega desses dados, que, apesar de terem sido solicitados formalmente, não foram entregues até à data da publicação do TCC.

2.3 BUSCA LIVRO: UM SISTEMA DE BUSCAS EFETIVAS EM ACERVOS BIBLIOGRÁFICOS (UEMS)

Este trabalho [24], de Rodolpho P. Sabino e Rogers P. de Pelle, propõe a criação de um sistema que melhore a eficiência das consultas aos sistemas da UEMS e UFGD, pela unificação dos acervos. Para tal, um banco de dados foi populado com os dados raspados desses sistemas.

O sistema proposto é constituído, de um mecanismo de coleta de dados (implementando os conceitos de *Web Scraping*), um servidor de buscas (formado pelo

[2] Raspagem faz referência à técnica conhecida como *Web Scraping* (raspagem da *Web*), que é aquisição automatizada de dados de interesse oriundos de *sites* da *Web [6]*.

SGBD PostgreSQL e por um indexador) e duas *interfaces* para realizar as consultas, uma *Web* e outra *mobile*, projetadas para este fim.

2.4 EXTRAÇÃO DE DADOS WEB COMO SUPORTE NA ELABORAÇÃO DE INDICADORES DO TURISMO DE MINAS GERAIS: UMA INICIATIVA EM BIG DATA (UFMG)

Esta dissertação [25], de Rafael Almeida de Oliveira, propõe uma análise de *BI* sobre os dados obtidos do *site* de viagens TripAdvisor, por meio da raspagem da *Web* para um banco de dados estruturado, sobre o qual pode fazer as análises, com o fito de levantar indicadores a respeito do turismo em Minas Gerais.

Ressaltando a importância de como este processo pode ser útil, fez um "acompanhamento dos dados referentes ao Conjunto Arquitetônico da Pampulha com o objetivo de avaliar um possível impacto do seu reconhecimento como patrimônio mundial da humanidade na percepção dos visitantes". Ressaltou, também, que os benefícios da análise de dados obtidos pela raspagem da *Web* podem, também, auxiliar as esferas públicas no melhor uso dos seus recursos.

2.5 MEUHORÁRIO 2: UMA APLICAÇÃO WEB PARA SIMULAÇÃO DE MATRÍCULA (UFBA)

Este trabalho [26], de Gabriel Assis Erbetta, propõe uma versão atualizada da aplicação *Web MeuHorário*, responsável por simular as matrículas dos estudantes da UFBA antes de serem efetivadas no sistema da referida universidade.

Para tal, foram usadas as técnicas de *Web Scraping* para fazer a raspagem dos dados da UFBA, os quais foram armazenados, posteriormente, em um banco de dados do SGBD *PostgreSQL*, com o propósito de serem utilizados no novo sistema *MeuHorário 2.*

2.6 ALUMNI TOOL: RECUPERAÇÃO DE DADOS PESSOAIS NA WEB EM REDES SOCIAIS AUTENTICADAS (PUC)

Esta dissertação [27], de Luis Gustavo Almeida, apresenta uma análise sobre os dados coletados via *Web Scraping*. Segundo o autor, “a ideia central desta dissertação consiste em recuperar dados da Internet que traduzam a trajetória profissional de ex-alunos da PUC-Rio para os mais variados fins, inclusive respondendo à pergunta de quanto tempo em média leva um ex-aluno de graduação a atingir um cargo de relevância. O caso de uso principal inclui o *LinkedIn*, com a possibilidade de se anexar outras fontes de dados”.

Como esse *site* requer autenticação, fato que dificulta a extração de dados por meio de robôs, foram usados *scripts* em conjunto com o *framework Selenium*, a fim de manter a sessão ativa, enquanto a raspagem dos dados era realizada.

2.7 VANTAGENS E DESVANTAGENS DE CADA ABORDAGEM

Observa-se que a motivação para uso do *scraping* é das mais diversas, e que as linguagens de programação utilizadas, também. De maneira geral, as motivações para o emprego da raspagem de dados nesses trabalhos, foi a aquisição automatizada de dados (ou por motivo de conveniência ou por algum outro impedimento), uso dos dados em um novo sistema, proposta de melhoria do anterior e análises para diversos fins.

Percebe-se, também, que os trabalhos acima podem ser divididos em dois grupos: aqueles fizeram uso de programação e que analisaram a estrutura do site antes de executar um scraper, e aqueles que não precisaram programar, mas utilizaram um serviço de raspagem. A vantagem da primeira abordagem reside no fato de que uma aplicação personalizada pode ser adaptada a qualquer *Web site* que se deseja raspar, porém isso demanda tempo de desenvolvimento e consequentemente, maior tempo para extrair os dados. Já na segunda, vemos que não foi necessário

conhecimento de programação para extrair dados de um site alvo, porém, apesar de não ter sido possível testar o *Import.io* por conta da escassez de tempo, parece razoável afirmar que todo *software* pronto para uso possui limitações, e como essa aplicação é oferecida na forma de um serviço da *Web*, mesmo que o usuário quisesse, não poderia adapta-lo na tentativa de otimizar a raspagem ou de contornar possíveis problemas que normalmente acontecem durante a extração, como interrupção abrupta da raspagem pela ação de mecanismos que restringem a atuação de robôs, para prevenir a sobrecarga do site.

A Tabela a seguir resume a implementação do *Web scraping* em cada trabalho:

Trabalho Relacionado	**Motivação para a raspagem de dados**	**Pré-análise do *site***	**Linguagens de programação**	**Ferramentas de extração**	**Bancos de dados**	***Software* BI, visualizador de dados e outros**
2.1	Contribuir com a pesquisa econômica ao analisar a depreciação dos autos	Sim	*Python* e *R*	*Scrapy* e SAS	Arquivo CSV	Não mencionado
2.2	Análise de documentos jurídicos com auxílio do algoritmo *PageRank*	Sim	*Java* e *Python*	*Scrapy*	*MongoDB*	Não mencionado, apresenta estatísticas dos dados por meio de gráficos
2.3	Criação de um sistema de biblioteca que unifique dois acervos e melhorar os resultados das consultas	Sim	*Python*	*Requests* e *lxml*	*Solr* e *PostgreSQL*	Sistema com *interfaces Web* e *mobile*
2.4	Auxiliar na elaboração de indicadores referente ao turismo em Minas Gerais, por meio da análise dos dados extraídos do site *TripAdvisor*	Não	—	Aplicação *Web Import.io* (www.import.io)	Arquivo CSV	Microsoft Excel
2.5	Reformulação de um sistema *Web* de simulação de matrículas	Sim	*Ruby*	*Nokogiri* e *Mechanize*	*PostgreSQL*	Aplicação *Web* objeto do trabalho
2.6	Estudar as trajetórias profissionais dos egressos de uma universidade por meio da raspagem de dados de sites de emprego	Sim	PHP	*Selenium*	*MySQL*	Módulo "web" da ferramenta objeto da dissertação
Fipe-*scraper*	Extrair dados de veículos automotores para realização de análises comparativas	Sim	*Python*	*Selenium*	*PostgreSQL*	*Tableau Desktop*

Tabela 1: Características principais dos trabalhos relacionados.

Nota-se, então, que estes trabalhos mostram duas formas de se fazer a raspagem, uma que exige bastante conhecimento técnico (saber programar), outra que não exige tanto. Podemos, ainda, ressaltar os trabalhos que se beneficiaram da biblioteca *Selenium*, pois o fato de ter sido usado um robô de automação, torna a

construção do *scraper* mais fácil, dentre outras vantagens, por isso pode-se resumir os prós e contras de cada abordagem, pelo comparativo a seguir:

Método	Extrator	Tempo necessário	Vantagens	Desvantagens
1	*Import.io*	Breve	• Não requer habilidades em programação; • Interface amigável; • Tempo reduzido de extração, pois a ferramenta é está pronta para o uso	Pode não conseguir raspar todo tipo de *site*, por não permitir a adaptação da ferramenta a cada caso
2	*Selenium*	Intermediário	• Visualizar a atuação do robô facilita a programação; • Contornar dificuldades encontradas pelo método 3	Tempo muito maior para extrair os dados, pois o robô simula a interação humana com o *site*
3	Outros	Intermediário a demorado	• Extração otimizada, podendo ser muito rápida; • O *scraper* pode ser adaptado a cada caso, permitindo a extração de uma quantidade de *sites* muito maior que no método 1	Maior tempo para realizar a extração, pelo tempo gasto com desenvolvimento

Tabela 2: Vantagens e desvantagens de cada abordagem.

Quanto à persistência dos dados, os métodos adotados foram, resumidamente: armazenamento em arquivos do sistema operacional e armazenamento por meio de um Sistema Gerente de Banco de Dados. Vemos que, no trabalho 2.4, o autor manipulou os dados com a ferramenta Microsoft Excel, o que poderia ocasionar a corrupção dos dados devido a falhas humanas, o que é bem difícil de acontecer quando se usa a abordagem SGBD, pois, para realizar alterações nos dados, é preciso explicitar a ação por meio de comandos em uma linguagem de manipulação de dados, como a SQL. Um Sistema de Banco de Dados também permite que o banco seja restaurado a um estado anterior e também é tolerante a falhas, como interrupção no fornecimento de energia elétrica. Há, também, outras vantagens em se adotar a abordagem SGBD como: acesso eficiente aos dados, controle de concorrência e permissões avançadas de acesso aos dados que não estão disponíveis na abordagem "armazenamento em arquivos do sistema operacional." Tendo em vista o grande valor que os dados raspados podem representar aos seus interessados, levando em conta, também, que o processo de raspagem pode ter custo elevado, mesmo consumindo uma parcela de recursos do sistema operacional, conclui-se a adoção de um SGBD é a melhor escolha para armazenar os dados coletados na raspagem.

Outro ponto a ser enfatizado é que, até o momento, não foi encontrado outro trabalho que tenha realizado *Web scraping* da base de veículos da FIPE e disponibilizado gráficos interativos dos mesmos por meio do serviço online do *Tableau*, além do fipe-*scraper*, a ferramenta de extração apresentada neste Trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

Nesse Capítulo, serão apresentados, brevemente, os conceitos tecnológicos que apoiaram o desenvolvimento deste trabalho, os quais, estão listados nesta ordem: *hardware* e *software*, linguagem de programação, ambiente de desenvolvimento integrado, *web crawler*, *web scraping*, sistema de gerenciamento de banco de dados e linguagem SQL.

3.1 *HARDWARE* E *SOFTWARE*

Constituem, respectivamente, as partes física e lógica de um computador [4, p. 20]. Entende-se como parte física, os diversos componentes eletrônicos que constituem um computador, como o processador, as memórias de acesso aleatório, a placa mãe e outros; e como parte lógica, os programas desenvolvidos para serem executados sobre o *hardware* do computador.

Os programas, ou aplicações, são um conjunto de instruções que podem ser compreendidas por um computador, e é por meio deles que os homens encontram soluções para diversos problemas do mundo real.

3.2 LINGUAGEM DE PROGRAMAÇÃO

Os computadores são máquinas que processam dados, mas para tal, precisam ser instruídos. A única linguagem que um computador eletrônico compreende, são sequências de “zeros” e “uns” (a linguagem binária). Como este nível de comunicação não é familiar aos homens e como o computador não consegue lidar com a linguagem humana, a qual é complexa e propensa a ambiguidades, as linguagens de programação foram criadas [16, p. 1].

Uma linguagem de programação é uma abstração de alto nível para se criar programas de computador. São formadas por comandos inteligíveis (normalmente em língua inglesa), possuindo uma sintaxe particular a cada linguagem [5, p. 63-64].

Antes que um programa, escrito em linguagem de alto nível, possa ser utilizado, é necessário “traduzi-lo” à linguagem de baixo nível do computador, o que é feito com o uso de compiladores (para linguagens compiladas) ou de interpretadores [5, p. 66] (para linguagens interpretadas, como é o caso do *Python* [13]).

3.3 AMBIENTE DE DESENVOLVIMENTO INTEGRADO

Os recursos mais básicos para se criar programas, em uma dada linguagem de programação, são um editor de texto convencional e o compilador ou interpretador da linguagem, porém, criar softwares desta forma é algo mais custoso do que desenvolver com o auxílio de um Ambiente de Desenvolvimento Integrado [7].

IDE, *Integrated Development Enviroment*, é um *software* que oferece diversas ferramentas ao programador para auxiliá-lo no desenvolvimento de outros *softwares*, provendo, além dos recursos mencionados, ferramentas como depuradores (que buscam falhas no código) e ferramentas de teste [7], a fim de facilitar e acelerar o processo de construção de um *software*.

Para apoiar o desenvolvimento da solução proposta por este trabalho, o IDE *PyCharm* [7] foi adotado, por ser um IDE projetada para o desenvolvimento com a linguagem *Python*.

3.4 *WEB CRAWLER*

Uma forma eficiente de automatizar a pesquisa em páginas da *Internet* é usando *Web crawlers,* também conhecidos como aranhas ou robôs, são programas

que baixam páginas *Web* de maneira automatizada” [9, p. 311], com o intuito de indexar páginas de *Internet* (o que fazem os buscadores, como o Google) ou de obter informações de interesse.

Basicamente, são programas que percorrem a *Web,* visitando um conjunto inicial de *links*, em busca de outros *links* nestas páginas. Ao encontrá-los, visita-os de maneira recursiva, até que uma condição de parada seja satisfeita.

Como nem sempre é interessante percorrer toda a *Web*, os crawlers podem ser configurados para atuar em um domínio específico, ou seja, para navegar apenas em *sites* de interesse do usuário. Os *crawlers* que são configurados para navegar desta forma são chamados de *focused crawler* [9, p. 327], ou *crawlers* focados.

3.5 *WEB SCRAPING*

Web Scraping, ou raspagem da *Web*, é o processo de aquisição automatizada de dados não estruturados de *sites* da *Internet*, seguido do armazenamento estruturado dos mesmos, para posterior análise. A motivação para esta prática é que muitos dados disponíveis na *Internet* “não foram projetados para ser consumidos programaticamente pela *Web”* [2, p. 143-144].

Um cenário típico de utilização da técnica de *scraping* seria quando se deseja realizar análises mais profundas sobre dados que estão disponíveis para consulta na Internet, mas que só podem ser acessados por meio de sistemas *Web* limitados quanto ao número e tipos de análises que podem ser feitas com esses dados.

A ideia do *Web Scraping* é transpor esse tipo de limitação, já que as possibilidades de análise multiplicam grandemente quando os dados estão estruturados em uma mídia local.

3.6 SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS

Um Sistema de Gerenciamento de Banco de Dados (SGBD) é um conjunto de *softwares* com o propósito comum de gerenciar bancos de dados (coleções de dados inter-relacionados que modelam um cenário do mundo real), o qual provê diversas funcionalidades importantes para gestão dos dados [3, p. 4], como controle de concorrência (prevenir que a base se torne inconsistente pela manipulação concomitante de um mesmo dado do banco), restrição de acesso a usuários e acesso otimizado aos dados [3].

Uma das grandes vantagens do uso de um SGBD é que o projeto de aplicações se torna mais fácil, e consequentemente mais barato, já que detalhes como armazenamento e acesso aos dados são tratados pelo SGBD e não pela aplicação [3, p. 15].

3.7 LINGUAGEM SQL

Structured Query Language – Linguagem Estruturada de Consulta – é a linguagem padrão de mercado para manipulação de bancos de dados relacionais [3, p. 148], ou seja, é uma linguagem que permite definir, modificar e consultar dados em bancos de dados baseados em Tabelas [3, p. 32], dentre outras funcionalidades, como a criação de visões (Tabelas virtuais) e especificação de restrições.

É uma linguagem de fácil aprendizado, poderosa e de alto nível, a qual permite ao usuário especificar, na declaração SQL, como será o resultado da consulta, deixando a cargo do SGBD a tarefa de escolher qual a melhor forma de executar a query do usuário [3, p. 148].

4 ANÁLISE E PROJETO DA FERRAMENTA

Nesse Capítulo, será abordada a construção da Ferramenta objeto deste trabalho. Os subcapítulos estão listados nesta ordem: introdução, análise *HTML*, caixas de seleção, requisitos, desenvolvimento da ferramenta, banco de dados, manipulação do banco de dados, automação do navegador e controle e manipulação da ferramenta.

A construção de um *scraper* demanda o conhecimento da estrutura da página de onde os dados serão extraídos [6, p. 36]. Para tal, podem ser usadas ferramentas de inspeção, presentes em navegadores populares como *Google Chrome* ou *Mozilla Firefox.*

Para o projeto de raspagem de dados do *site* da FIPE, foi observado que não existe navegação em diversos endereços na *Internet*, o que requer a criação de um *scraper* [6, p. 17] e não de um *crawler*. O endereço acessado exibe os resultados das consultas na própria página, através de requisições AJAX feita à sua API.

Essa API possui um mecanismo de proteção contra requisições que não sejam feitas através de navegadores, o qual retorna o HTML de seu endereço principal quando elas não são feitas pelo *JavaScript* da página.

Esses fatores levaram à escolha da biblioteca *Selenium* [15] para o desenvolvimento do *scraper*, pois através dele é possível enviar os comandos requeridos pela página da Fipe, através de um *headless* browser (navegador *Web* executado sem interface gráfica), manipulado por um robô que simula a interação humana com periféricos de entrada, navegando pelas caixas de seleção da página, de forma sistemática e automatizada.

4.1 ANÁLISE HTML

Para se obter os dados a serem extraídos, o primeiro elemento a ser observado foi o *link* com o texto “Consulta de Carros e Utilitários Pequenos”. Ao ser efetuado o clique sobre o *link*, um pequeno formulário é exibido com apenas quatro

caixas de seleção e um botão para pesquisa. Essas caixas de seleção são o alvo principal do *scraper* e serão manipuladas pela ferramenta desenvolvida.

A primeira caixa de seleção possui o período de referência ao qual será feita a consulta. Por exemplo, podemos consultar quanto custava um Fiat Palio City 1.0 4p 2000 Gasolina no mês de março de 2012, ou quanto custava o mesmo veículo em março de 2018 alterando apenas esse campo. O sítio da Fipe possui períodos de referência a serem consultados desde janeiro de 2001 até a presente data.

Uma vez selecionado o período de referência, conforme instruções presentes no próprio site, devemos selecionar a marca do veículo a ser consultado. Essa informação está presente em uma segunda caixa de seleção. Este campo pode ser modificado de acordo com o período de referência selecionado.

O passo seguinte é o preenchimento das duas últimas caixas de seleção. O preenchimento de uma delas, remove elementos da outra que não possuam correspondência. A primeira dessas últimas caixas possui os modelos de automóveis da marca selecionada anteriormente. Sendo a última caixa responsável pelo ano de fabricação do veículo.

Apesar de não importar a ordem de seleção das últimas duas caixas, foi escolhido fazer a seleção de cada uma delas na ordem em que aparecem na página (primeiro o modelo e depois o ano modelo, de cima para baixo).

Após todos os campos serem selecionados, o botão “pesquisar” deve ser clicado para que os dados do veículo sejam exibidos. Depois deste passo, os dados são mostrados em uma Tabela com os seguintes atributos: Mês de referência (escolhido na primeira caixa de seleção), Código Fipe, Marca (escolhida na segunda caixa de seleção), Modelo (escolhido na terceira caixa de seleção), Ano Modelo (escolhido na quarta e última caixa de seleção), Autenticação, Data da consulta e Preço Médio.

Abaixo da Tabela, é exibido um botão de “limpar pesquisa”, o qual suprime a referida Tabela e atribui às caixas de seleção os seus valores *default*, como na primeira visita feita à página. Essa limpeza é opcional, mas foi escolhido realiza-la para que uma consulta não contaminasse a seguinte.

4.2 CAIXAS DE SELEÇÃO

Analisando o código fonte das caixas de seleção desta seção da página, é possível observar que, na verdade, elas são apenas uma *tag* “a” que disparam um evento *JavaScript*, o qual preencherá uma lista inicialmente vazia.

Ao realizar um *click* de mouse nessa *tag* “a”, será preenchida uma lista contendo todos os nomes de carros e uma caixa de inserção de texto será exibida. A seleção desejada deve ter seu texto inserido para que a próxima caixa de seleção tenha seus valores validados.

O texto a ser inserido nessas caixas de inserção de texto podem ser encontrados em uma caixa de seleção real que é ocultada dos usuários como uso da propriedade *display*, configurada com o valor *“none”*, a qual pode ser encontrada no código HTML.

A Figura 1 destaca, por meio de um balão de diálogo na cor preta, o *input* onde deve ser inserido o nome das marcas dos automóveis, e na linha destacada em azul, no HTML da página, o código correspondente a esse elemento:

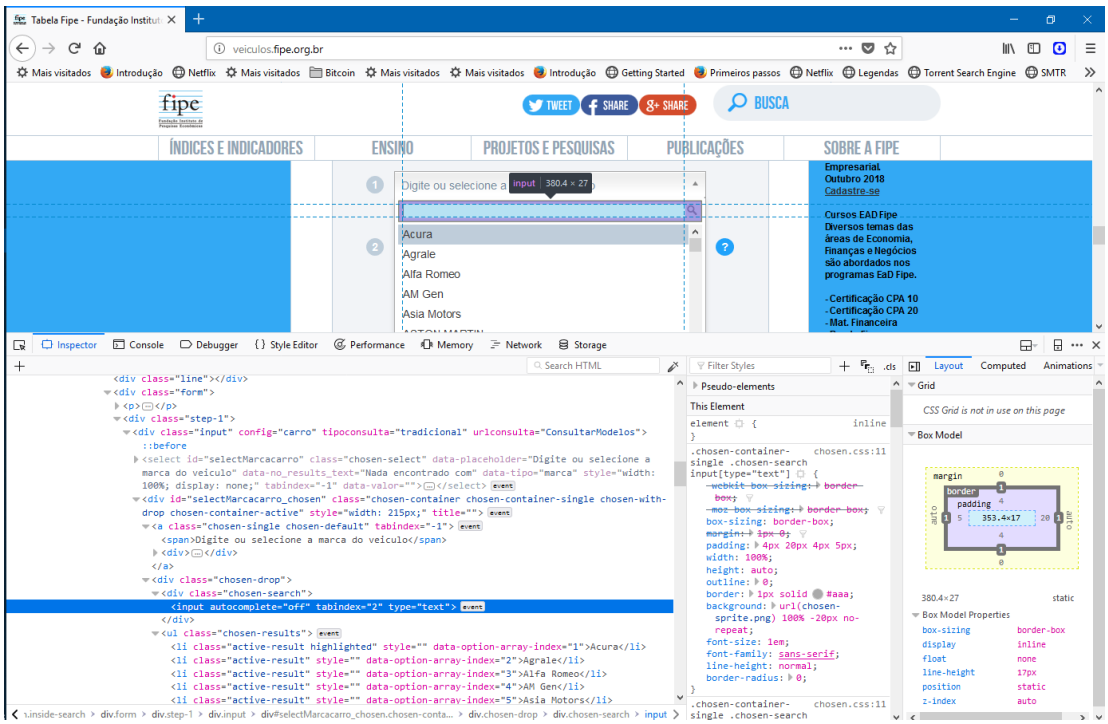

Figura 1: inspeção do HTML do *site* da Fipe (Fonte: do Autor).

4.3 REQUISITOS

A ferramenta desenvolvida simula a interação de um usuário que manipula o *site* da FIPE. Dessa forma, é preciso registrar os passos necessários para realizar uma consulta pelo valor de mercado de um veículo automotivo. Após o carregamento da página, é necessário clicar em um *menu dropdown* com texto “CONSULTA DE CARROS E UTILITÁRIOS PEQUENOS”, permitindo assim o acesso ao formulário que deve ser preenchido para cada consulta.

O formulário possui quatro campos selecionáveis, cada um composto por um *select* que possui os valores disponíveis a serem inseridos no seu referido *input*. Os valores desses ***select's*** são preenchidos com requisições AJAX a cada interação feita com o *select* anterior.

O primeiro *select* a ser preenchido deve ser o período de referência, ao se fazer a seleção deste, o *select* com as marcas de veículos será modificado, para corresponder à data selecionada no primeiro campo. Em seguida, deve ser feita a seleção da marca do veículo, que fará com que as caixas de seleção de modelo e ano modelo do veículo, que se encontravam vazias até então sejam preenchidas. A última etapa é o preenchimento das últimas duas caixas de seleção que, ao serem preenchidas, levarão à exclusão de valores não correspondentes da outra caixa de seleção, não importando a ordem em que forem selecionadas.

Para a exibição dos dados, é necessário clicar no botão pesquisar do formulário. Sendo assim, serão exibidos os dados referentes às seleções, contendo o mês de referência da consulta, o código Fipe do automóvel, marca, modelo, ano de fabricação, autenticação da consulta, data da consulta e preço médio do veículo no mercado nacional.

Como o *scraper* manipulará o *site* de forma automatizada, foi definido um algoritmo que descreve os passos necessários para extrair dos dados:

1. Capturar todos os valores do *select* com *id “*selectTabelaReferenciacarro*”;*
2. Inserir o primeiro elemento ainda não extraído no *input* do período de referência;

3. Capturar todos os valores do *select* com *id* “selectMarcacarro”;
4. Inserir o primeiro elemento não extraído no *input* da marca do veículo;
5. Capturar todos os valores do *select* com *id* “selectAnoModelocarro”;
6. Inserir o primeiro elemento não extraído no *input* do modelo do veículo;
7. Capturar os valores do *select* com *id* “selectAnocarro”;
8. Inserir o primeiro elemento não extraído do ano modelo do veículo;
9. Clicar no *link* com *id* “buttonPesquisarcarro” para fazer a pesquisa;
10. Salvar o resultado da pesquisa no banco de dados;
11. Clicar no botão limpar com *id* “buttonLimparPesquisarcarro”;
12. Retornar ao passo 8 até que todos os valores do *select* “selectAnocarro” tenham sido capturados;
13. Retornar ao passo 6 até que todos os valores do *select* “selectAnoModelocarro” tenham sido capturados;
14. Retornar ao passo 4 até que todos os valores do *select* “selectMarcacarro” tenham sido capturados;
15. Retornar ao passo 2 até que todos os valores do *select* “*selectTabelaReferenciacarro”* tenham sido capturados.

Na Figura 2 pode ser visto o Diagrama de Sequência que ilustra as iterações da aplicação com o navegador e o banco de dados.

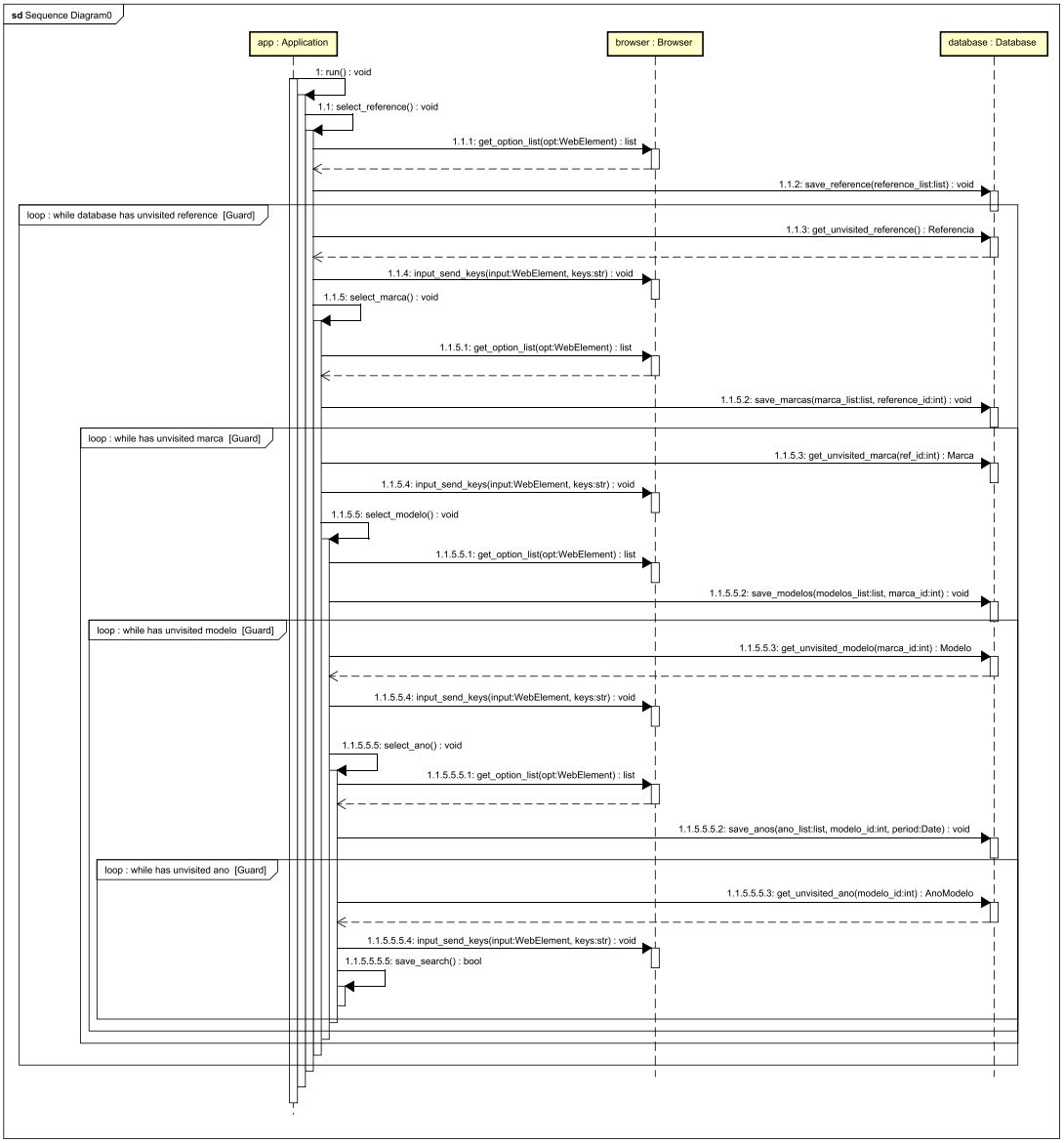

Figura 2: Diagrama de Sequência (Fonte: do Autor).

Na Figura 3 pode ser observado no Diagrama de Classes a estrutura da ferramenta desenvolvida.

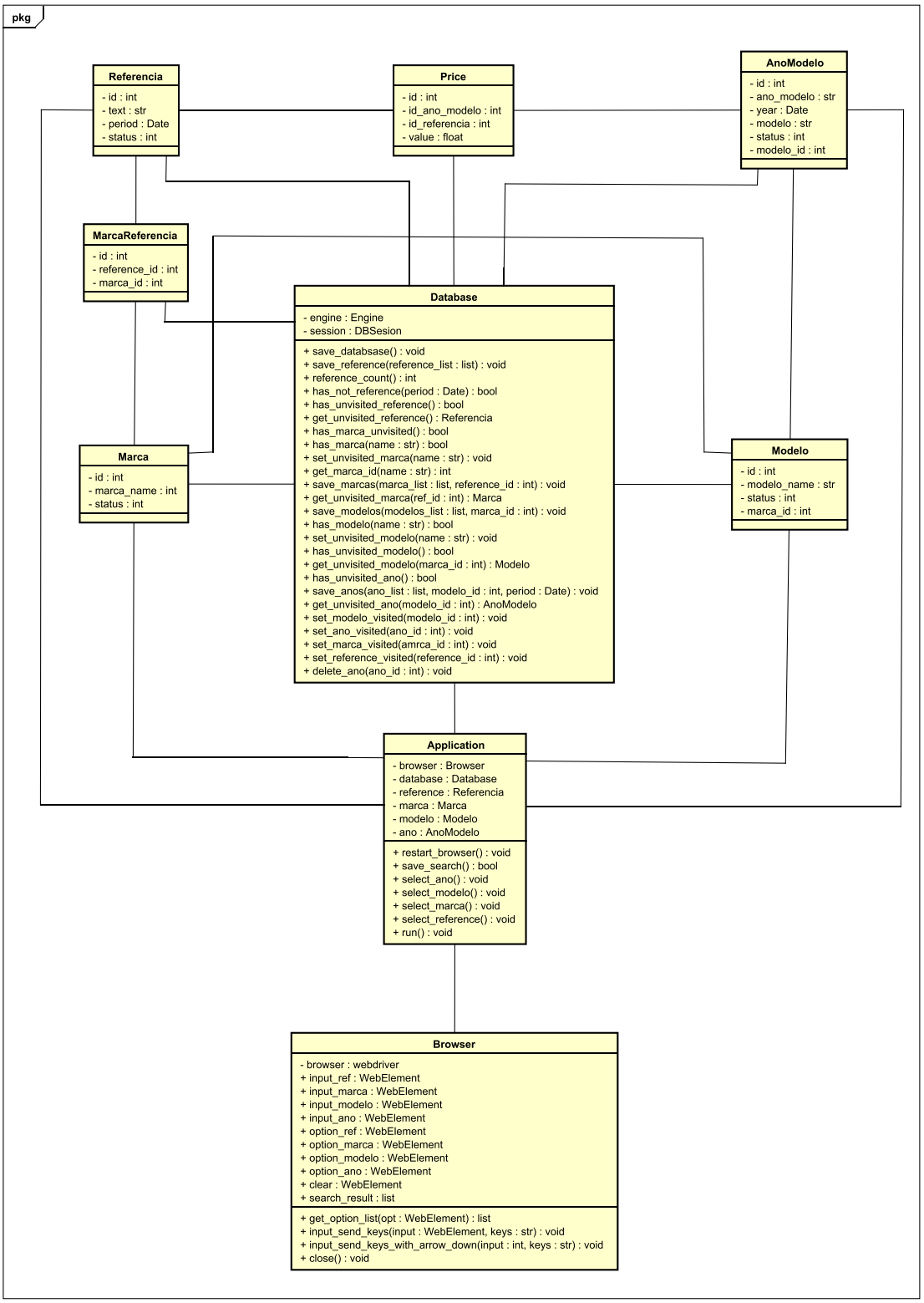

Figura 3: Diagrama de Classes (Fonte: do Autor).

Para o desenvolvimento da ferramenta de extração, foi utilizada a linguagem *Python*, o IDE *PyCharm*, a biblioteca *Selenium* e o *ORM SQLAlchemy* para interação com o banco de dados *PostgreSQL*.

4.4 DESENVOLVIMENTO DA FERRAMENTA

O desenvolvimento do banco de dados do feito usando o modelo Entidade-Relacionamento a fim otimizar o espaço ocupado na memória. Para desenhar o modelo do banco foi usado o MySQL Workbench 6.3 CE [12], ferramenta que permite a modelagem do banco de dados por meio de uma interface gráfica. No entanto, a estrutura do banco foi criada através do módulo db_declarative.py do “fipe-scraper”.

Toda a escrita do código foi feita utilizando o PyCharm Professional 2018.2.1. O PyCharm é um IDE com suporte ao Python, com ótimos recursos como autopreenchimento e detecção de erros de código. Com ele é possível usar uma pesquisa inteligente saltando entre classes, arquivos ou símbolos. Também possui um ótimo desempenho ao prover refatorações de código rápidas e seguras [7].

Para a execução do fipe-scraper foi utilizado o Python 3.7.0. Python é uma linguagem de programação de alto nível, interpretada, orientada a objetos, funcional e de tipagem dinâmica e forte, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional, prioriza legibilidade do código sobre a velocidade e expressividade, e combina uma sintaxe concisa e clara com recursos poderosos de suas bibliotecas, módulos e frameworks [14].

Além do Python, foi usado o ORM SQLAlchemy, uma biblioteca poderosa que fornece um conjunto completo e bem conhecido de modelos de persistência de dados. Foi projetada para a alta performance de acesso a banco de dados, adaptado a uma linguagem de domínio simples. Para se conectar a um banco de dados, o SQLAlchemy requer outra biblioteca, para isso foi utilizada a Psycopg2 [18].

O scraper foi construído utilizando a biblioteca Selenium, que permite a manipulação de navegadores de Internet em modo headless, simulando o comportamento de um usuário. O Selenium aceita comandos fornecidos pelos métodos de seu webdriver, os quais são enviados, posteriormente, a um navegador.

O browser utilizado para as extrações foi o Firefox Quantum 62.0.2 [10], que é um navegador livre e multiplataforma desenvolvido pela Mozilla Foundation com ajuda de vários colaboradores.

4.5 BANCO DE DADOS

Cada caixa seletora do formulário de consulta foi mapeada como uma entidade do banco de dados, sendo essas: *referencia*, *marca, modelo* e *ano_modelo*. Além destas, também foram criadas as entidades *marca_referencia* para fazer a ligação entre os dados das Tabelas *referencia, marca* e *price* que guardará o valor de cada veículo.

Entre essas Tabelas foram feitas as seguintes conexões, com o uso de chaves estrangeiras: *referencia*, que possui conexão com *marca_referencia* e *price*, sendo adicionado uma chave estrangeira nestas; *marca* é ligada a *marca_referencia* e a *modelo*, que receberam a chave estrangeira de *marca*; em *ano_modelo* foi adicionado uma chave estrangeira de *modelo*; e *price* foi conectada, também, a *ano_modelo*, recebendo assim mais uma chave estrangeira.

A entidade *referencia* possui como atributos um *id* para identificação do tipo *int4* auto incrementável, *text* para guardar o valor da caixa seletora do tipo *varchar*, *period* do tipo *date* para guardar a data do valor referente do automóvel e *status* do tipo *int2* para que a ferramenta construída identifique quais datas já tiveram as extrações concluídas.

Na Tabela *marca* além de seu *id* com tipo *int4* auto incrementável, foi adicionado um atributo para guardar o nome da marca e o valor do campo, *marca_name* do tipo varchar e um atributo *status*, *int2*, para sinalizar ao *scraper* se a marca já foi visitada ou não.

Em *modelo*, além de seu *id*, *int4* auto incrementável e *status*, *int2*, foram adicionados dois atributos do tipo *varchar*, *modelo_name,* que guarda o nome do modelo e o valor da caixa de seleção referente e *fipe_code,* que salva o código da Tabela FIPE para o modelo do veículo selecionado.

A Tabela *ano_modelo* possui, além de seu *id* (*int4* auto incrementável) e *status* (int2), os atributos *year* (*date)* que guarda o ano de fabricação do automóvel, *modelo* (*varchar)* que guarda o combustível do mesmo, e *ano_modelo* que salva o valor da caixa seletora a ser selecionada.

Por último, *price* recebeu além de seu *id* do tipo *int4* auto incrementável, um atributo *value* do tipo float8 para guardar o preço médio do carros, estimado pela FIPE.

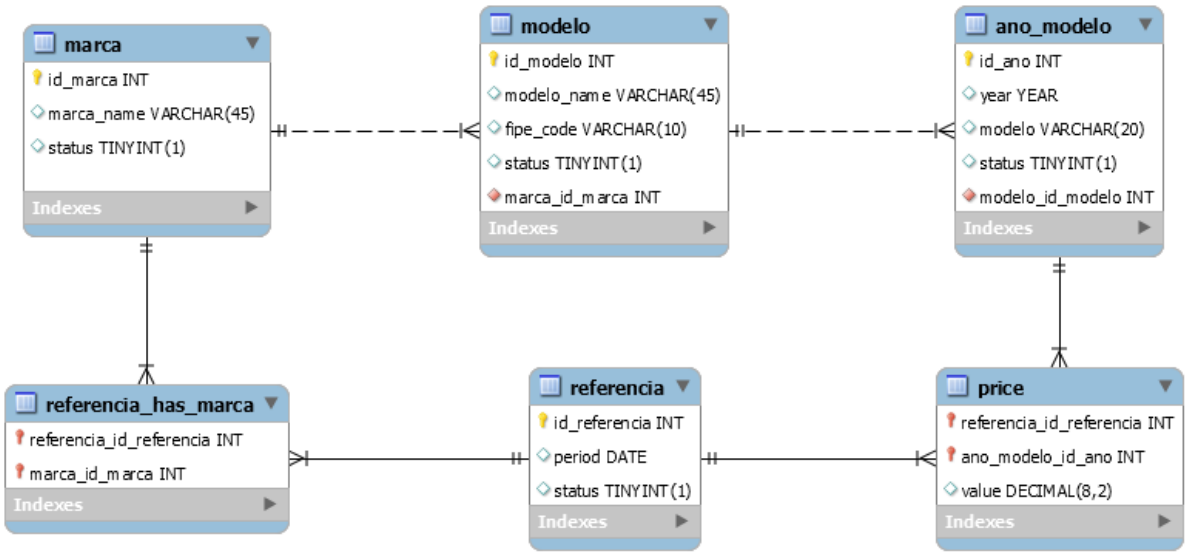

Figura 4: esquema do banco de dados (Fonte: do Autor).

A Figura 4 demonstra o Diagrama Entidade Relacionamento do banco de dados do *fipe-scraper*.

4.6 MANIPULAÇÃO DO BANCO DE DADOS

O banco de dados da ferramenta foi construído usando o *SQLAlchemy* [11], uma biblioteca que facilita a comunicação entre aplicações *Python* e bancos de dados, a qual provê uma *interface* que permite ao desenvolvedor comunicar-se com diversos bancos de dados através da linguagem *Python*, sem se preocupar com a implementação da linguagem SQL daquele banco [8].

Essa biblioteca foi usada como uma ferramenta ORM, convertendo classes em Tabelas de bancos de dados relacionais de maneira automatizada.

Para que a conexão com o banco de dados seja feita, o *SQLAlchemy* requer que seja informada uma biblioteca capaz de gerenciar a conexão do *Python* com um banco de dados, a DBAPI[3]. Como o banco de dados utilizado foi o *PostgreSQL*, foi usada a biblioteca *psycopg2-binary*, pois é a mais popular biblioteca de integração entre o Python e o *PostgreSQL*.

Foi construído um módulo à ferramenta, com o único propósito de criar e manipular o banco de dados. O módulo *db_declarative* possui uma classe para cada entidade do banco de dados e uma outra classe, chamada *Database*, responsável por fazer a manipulação das demais classes do banco de dados. Como pode ser visto no Apêndice, a classe *Database* possui métodos dedicados à manipulação dos dados que serão salvos no banco.

- *save_reference*: recebe como parâmetro uma lista de *strings*, capturadas da caixa de seleção do período de referência, trata todos os valores presentes nessa caixa e salva na Tabela *referencia* do banco de dados apenas se ainda não houver valor semelhante;
- *reference_count*: verifica a quantidade de registros da Tabela referencia;
- *has_not_reference*: verifica se há algum registro na Tabela referencia;

[3] DBAPI é uma biblioteca para a comunicação entre o banco de dados e a aplicação em execução [19].

- *has_unvisited_reference*: verifica se existe algum registro não visitado na Tabela referencia;
- *get_unvisited_reference*: busca a primeira referência não visitada no banco de dados;
- *set_reference_visited:* recebe como parâmetro um inteiro com o *id* da Tabela referencia a ser marcado no banco de dados como visitado;
- *save_marcas*: recebe como parâmetro uma lista de marcas e um *id* de referência. Para cada marca na lista é verificado se já existe um registro no banco de dados com a marca informada, caso exista, o status dessa marca é marcado como não visitado através do método *set_unvisited_marca*, mas, caso ainda não exista no banco de dados, é criado um registro. Por fim é criado um registro na Tabela marca_referencia com *reference_id* e marca_id;
- *has_unvisited_marca*: consulta no banco de dados se existe alguma marca não visitada;
- *has_marca*: consulta no banco se existe algum registro na Tabela marca;
- *set_unvisited_marca*: recebe como parâmetro uma *string* com o nome da marca e altera o valor do atributo *status* para não visitado;
- *get_marca_id*: recebe como parâmetro uma *string* com o nome da marca e retorna um número inteiro com o *id* da marca;
- *get_unvisited_marca*: recebe como parâmetro um inteiro com o *id* da referência e retorna o primeiro registro do banco de dados com uma marca não visitada;
- *set_marca_visited*: recebe como parâmetro um inteiro com o *id* da marca a ser atribuída como visitada no banco;
- *save_modelos*: São passados como parâmetros uma lista de modelos e o *id* da marca a qual esses modelos se referem. Para cada modelo na lista é verificado se o mesmo existe no banco de dados, caso exista, então é marcado como não visitado pelo método

set_unvisted_modelo, mas caso ainda não exista no banco de dados, é criado um registro;

- *has_modelo*: consulta no banco de dados se existe algum registro na Tabela modelo;
- *set_unvisited_modelo*: recebe como parâmetro uma *string* com o nome do modelo que deve ser marcado no banco como não visitado;
- *has_unvisited_modelo*: consulta na Tabela modelo do banco de dados se existe algum registro não visitado;
- *get_unvisited_modelo*: recebe como parâmetro um inteiro com o *id* do modelo a ser consultado e retorna o primeiro registro não visitado na Tabela modelo do banco de dados;
- *set_modelo_visited*: recebe como parâmetro um inteiro com o *id* do modelo a ser marcado no banco de dados como visitado;
- *save_anos*: São passados como parâmetros uma lista de anos de cada modelo, o *id* do modelo e o período referente. Cada ano do modelo referente é convertido para o tipo *date* e salvo no banco de dados. Caso seja um veículo zero quilômetro, o período de referência será o ano do veículo;
- *has_unvisited_ano*: consulta na Tabela ano_modelo do banco de dados se há algum registro não visitado;
- *get_unvisted_ano*: recebe como parâmetro um inteiro com o *id* do modelo a ser consultado na Tabela ano_modelo e retorna o primeiro registro não visitado desta Tabela;
- *set_ano_visited*: recebe como parâmetro um inteiro com o *id* da Tabela ano_modelo a ser marcado no banco como visitado;
- *delete_ano*: este método foi criado para prevenção de erros na ferramenta caso o AJAX com os anos dos modelos selecionados demore mais de dois segundos para carregar. Ele recebe o *id* da Tabela ano_modelo que foi inserida no banco de dados indevidamente e a remove do banco.

Por fim foi utilizado o método create_all, da classe MetaData [1], presente no ORM SQLAlchemy, para a criação das Tabelas necessárias ao banco de dados, quando o módulo db_declarative.py do projeto for executado. Dessa forma, não foi necessário criar as Tabelas no banco usando instruções SQL.

4.7 AUTOMAÇÃO DO NAVEGADOR

Como foi falado no início deste Capítulo, a manipulação do navegador foi feita pela biblioteca *Selenium* [15], a qual permite o acesso às informações do *site* de forma sistemática e automatizada.

Essa biblioteca requer um driver fornecido pelo desenvolvedor do navegador a ser utilizado. Como o *browser* usado neste projeto foi o *Mozilla Firefox,* o driver utilizado foi o *geckodriver*, que se encontra na pasta *firefox* do sistema de arquivos do projeto.

Através do *Selenium*, é possível localizar elementos HTML pelo seu *id*, ou pela sua classe, ou *tag*, ou nome, ou texto do *link*, ou CSS ou ainda pelo *XPath*. Para o projeto desenvolvido, foram utilizadas as buscas por elementos pelo *id*, texto do *link* e *XPath*.

A *Selenium* disponibiliza uma série de métodos para interação com os elementos da página *Web*. Os métodos usados nesse projeto foram o *click* e o *send_keys*. Enquanto o primeiro realiza um *click*, o segundo envia uma cadeia de caracteres para o elemento manipulado. Existe, ainda, uma classe que auxilia o método *send_keys*, disponibilizando caracteres do teclado como “Enter” (entrar) e “arrow_down” (seta para baixo), ambos utilizados nesse projeto.

Conforme o padrão de projeto *Indirection*, foi criada uma classe a fim de intermediar o uso da biblioteca *Selenium* e controlar as ações para manipulação do navegador. A classe *Browser* deste projeto assume essa responsabilidade.

Ao ser inicializada, essa classe abre uma instância navegador *Firefox* em modo *headless*, o qual navega até a página http://veiculos.fipe.org.br/, de onde buscará o *link* com o texto “Consulta de Carros e Utilitários Pequenos”, e clicará no mesmo.

Com essa implementação da inicialização da classe, foi possível automatizar o processo de tratar erros ocorridos durante a extração de dados. Esses erros serão relatados no Capítulo 5.1.

Como descrito no Capítulo 4.3, para acessar cada item das caixas de seleção, foi criado um método chamado *get_option_list*, que recebe todos os elementos com a *tag option* presente dentro dos *selects* de cada uma das caixas de seleção.

Também foi criado um método chamado *input_send_keys*, responsável por passar uma cadeia de caracteres como argumento para os *inputs* desses campos de seleção, fazendo, assim, a seleção do período de referência, marca, modelo ou ano de cada veículo a ser consultado. Por fim, o método *close* é responsável por fechar o navegador.

4.8 CONTROLE E MANIPULAÇÃO DA FERRAMENTA DE EXTRAÇÃO

Ao executar a aplicação, o navegador, em modo *headless*, visita o site da Fipe de onde extrairá as Tabelas relativas a cada consulta e conecta ao banco de dados.

Após isso, são capturados todos os meses do período de referência, salvando-os na Tabela *referência* do banco como não visitados. É iniciado um *loop*, que verificará, a cada iteração, se existe algum mês não visitado.

Enquanto houver período de referência não visitado, será selecionado, no banco de dados, o primeiro valor não visitado para a extração referente às marcas de veículos disponíveis naquele período. Tendo sido capturadas as marcas, elas serão salvas no banco de dados na Tabela *marca* como não visitado.

Então, o procedimento se repete para a escolha da marca. Em seguida a caixa seletora de *modelo* passará pelos mesmos procedimentos das caixas seletoras de *referência* e marca. Por último a caixa seletora de *ano modelo* também terá seus valores salvos no banco e selecionados um a um para realização da consulta.

Após a passagem de valores, por meio das caixas seletoras, será feita uma pesquisa ao banco da Fipe, e será retornada, na mesma página, uma Tabela

resultado, da qual os dados serão extraídos os dados do veículo com as características selecionadas.

Finalmente, o último passo de cada iteração realizado nas caixas seletoras é marcar o elemento que teve suas consultas terminadas como visitado, para que a extração avance passar para o item seguinte. A condição de parada do laço ocorre quando não há mais valores, de suas caixas seletoras, a serem visitados.

O código fonte da aplicação fipe-*scraper* pode ser acessado em seu repositório online através do endereço eletrônico https://github.com/thcborges/fipe-scraper.

5 EXTRAÇÃO DE DADOS DO SITE DA FIPE

Esse Capítulo relata a execução do processo de raspagem de dados do site da Fipe. Os subCapítulos estão listados desta forma: cenário e conFiguração, dificuldades encontradas e visualização dos dados extraídos.

5.1 CENÁRIO E CONFIGURAÇÃO

A ferramenta foi executada em um notebook Dell Inspiron 5559 com processador Intel Core i5-6200 de 2.3GHz, 12GB de memória RAM, 128GB de SSD, Windows 10 Pro e uma conexão a cabo, da TIM, de 35Mb/s. Como todos os dados foram persistidos no armazenamento local, não houve problema de alto consumo de memória, nem de elevada demanda por processamento. O único limitador da execução foi a conexão com a Internet.

O início da extração ocorreu no dia trinta de julho de dois mil e dezoito e terminou no dia nove de outubro de dois mil e dezoito. A extração dos dados de todos os carros dos períodos de referência mais recentes levou em média 24 horas, enquanto os períodos de referência mais antigo levou aproximadamente 11 horas. De tal forma foi estimado um prazo de seis meses para que todos os preços fossem extraídos.

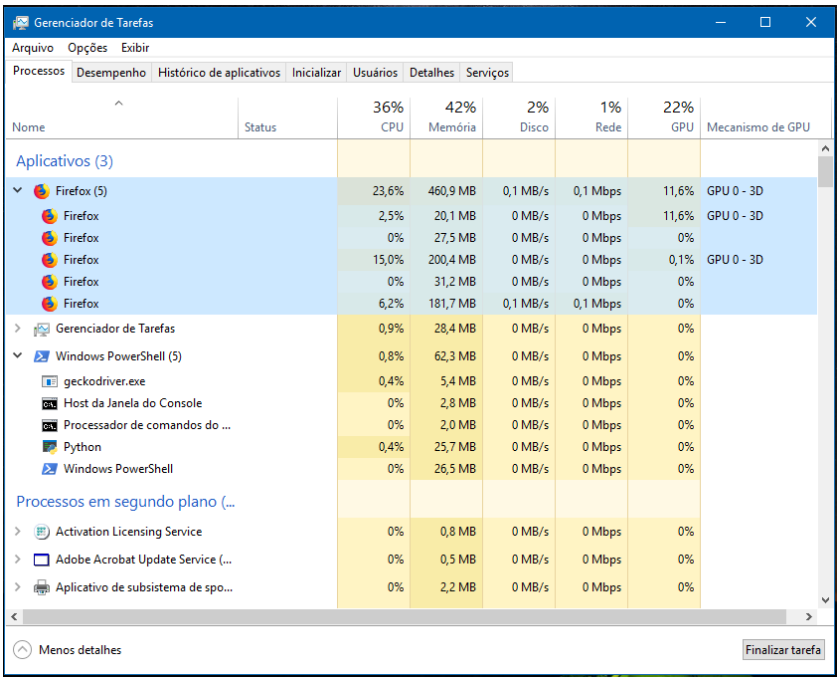

Figura 5: processos relacionados ao *scraper* (Fonte: do Autor).

Para nossa análise optamos por extrair todos os preços dos automóveis até agosto de dois mil e dezessete, de agosto de dois mil e doze a agosto de dois mil e dezessete, as extrações foram feitas a cada três meses. E, para o período de janeiro de dois mil e um a agosto de dois mil e doze, os dados foram extraídos a cada seis meses. Os dados extraídos estão disponíveis para download no endereço: https://goo.gl/LdmB2V.

5.2 DIFICULDADES ENCONTRADAS

Durante a extração houve alguns contratempos que não puderam ser previstos na análise do projeto e, portanto, precisaram ser contornados no período da raspagem dos dados.

O primeiro ponto positivo foi o acerto das pausas para aguardar o carregamento das solicitações AJAX que a página faz ao servidor. A extração dos dados foi iniciada sem que nenhuma pausa fosse estipulada. Porém algumas falhas de exe-

cução ocorreram, por isso foram testados tempos de pausa de um, dois e três segundos a cada iteração com as caixas seletoras, e ao final desses testes, o melhor tempo encontrado foi o de dois segundos.

Também, no início da extração, foi detectado que o fipe-scraper, ao tentar fazer a seleção da marca “Rover”, selecionava a marca “Land Rover”. Tal fato ocorria devido ao nome de uma marca ser substring da segunda, sendo a segunda anterior à primeira na lista de seleção. Para contornar o problema, a cada iteração com a caixa de seleção da marca, foi verificado se o id da marca era o mesmo id da marca “Rover”, neste caso era chamado um método de seleção específico que executa os seguintes comandos do teclado: escreve o nome da marca no input, simula o pressionamento da seta para baixo e por fim pressiona a tecla “enter”.

Outra dificuldade encontrada, foi a interrupção da busca por algum agente externo, a qual acontecia todos os dias, exatamente às três horas da manhã. Pressupondo que algum mecanismo de bloqueio estava impedindo a atuação contínua de robôs na página, um trecho de código foi criado para tratar essa exceção, cujo algoritmo executa estes passos: durante a iteração de cada caixa seletora, encerra o processo do navegador, aguarda um minuto e reinicia o browser, retornando de onde parou.

Para realizar a seleção da caixa Ano Modelo, por vezes foi detectado que, com uma pausa razoavelmente grande (três segundos), eram adicionados valores ao banco que não pertenciam ao modelo selecionado. Para não ter de aumentar novamente o tempo de pausa, o que afetaria o tempo de execução da ferramenta, foi criada uma exceção que trata esse evento, removendo os valores que foram inseridos erroneamente da Tabela.

5.3 VISUALIZAÇÃO DOS DADOS EXTRAÍDOS

Para visualização dos dados extraídos, foi utilizado o *software* Tableau Desktop 2018.1.3 [16]. Com ele, é possível construir, com facilidade, gráficos interativos que auxiliam na análise dos dados. Esse *software* permite, também, a conexão com diversas e volumosas fontes de dados, de diferentes formatos.

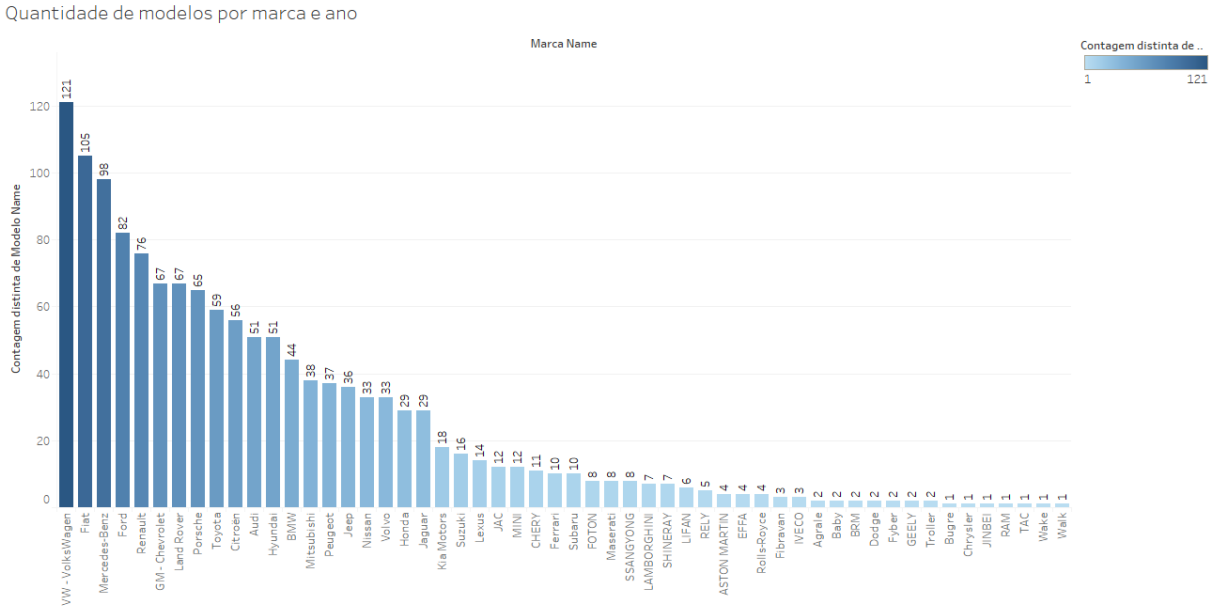

Figura 6: quantidade de modelos por marca e ano (Fonte: do Autor).

Como podemos ver no gráfico da Figura 6, em 2018 a Volkswagen foi a marca com a maior quantidade de modelos de veículos zero quilômetro comercializados (121 modelos). Enquanto sete marcas possuem apenas um modelo disponível no mercado: Walk, Wake, TAC, RAM, JINBEI, Chrysler e Bugre. É possível interagir com esse gráfico através do *link*: https://goo.gl/JAHctt.

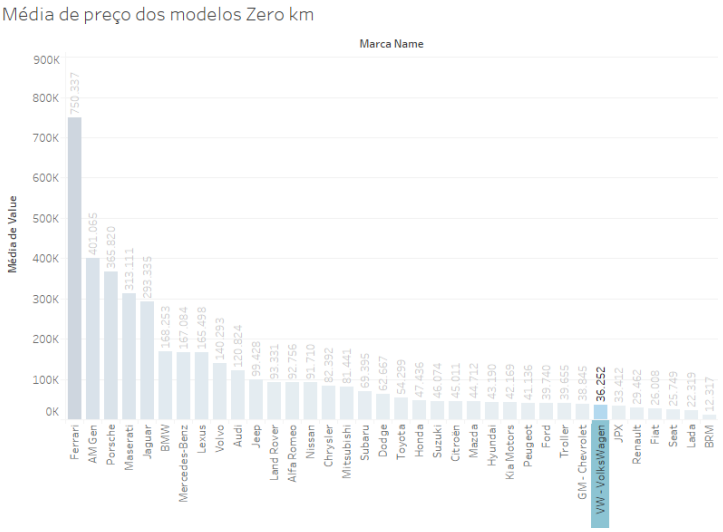

Figura 7: média de preço dos modelos zero km - parte 1 (Fonte: do Autor).

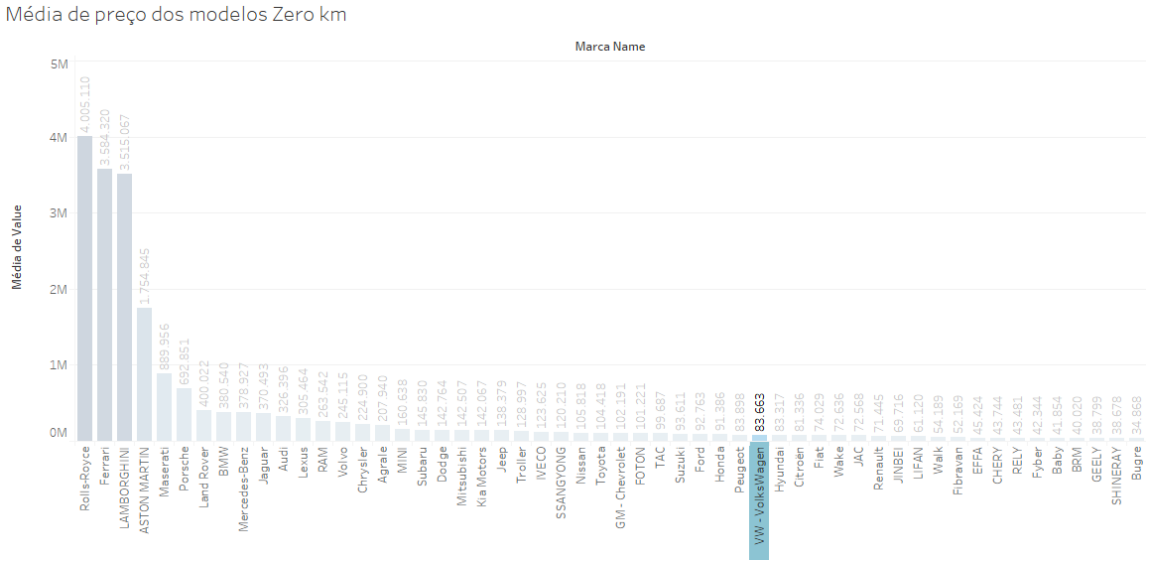

Figura 8: média de preço dos modelos zero km - parte 2 (Fonte: do Autor).

Outro fator relevante que pode ser constatado, é a média de preços dos veículos zero quilômetro comercializados pelas marcas de veículos. Consultando os dois últimos gráficos (Figura 7 e Figura 8), percebe-se que, em 2001, o valor médio de um veículo da Volkswagen custava R$ 36.252,00 enquanto em 2018 o preço médio passou para R$ 83.663,00. Esse gráfico está disponível para interação do usuário através do *link*: https://goo.gl/6DdqsF.

Veículos sofrem desvalorização com o passar do tempo. A Figura 9 mostra o comportamento de veículos de marcas populares no mercado de automóveis. Foram escolhidos veículos em pouca quantidade e pertencentes a faixas de preços distintas para melhor visualização do gráfico.

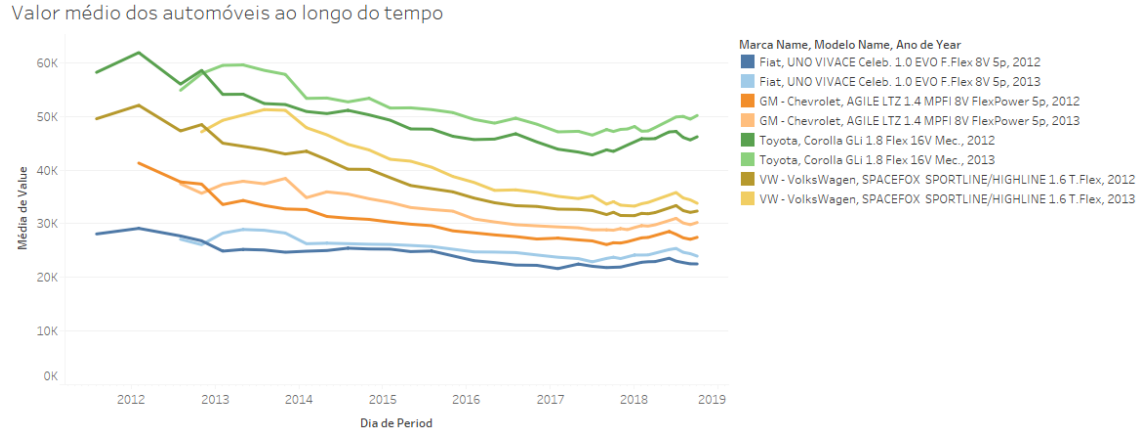

Figura 9: valor médio dos automóveis ao longo do tempo (Fonte: do Autor).

Com esse gráfico podemos ver que o modelo UNO VIVACE Celeb, da Fiat, sofre pouca desvalorização, pois a progressão de sua linha varia pouco com relação ao eixo vertical, mostrando que seu preço, ao longo dos anos, é mais estável do que outros modelos, como o SPACEFOX SPORTLINE da Volkswagen, cuja evolução da curva varia mais com relação ao eixo referente ao preço do auto. O gráfico da Figura 9 está disponível para interação através do link: https://goo.gl/7R9Gim.

A Figura 10 mostra o desvio padrão das mesmas marcas selecionadas acima. Quanto maior o desvio padrão, mais o preço do carro varia ao longo do tempo: A mesma pode ser visualizada em: https://goo.gl/UpQzdJ.

Desvio padrão dos preços dos automóveis

		Year	
Marca Name	Modelo Name	2012	2013
Fiat	UNO VIVACE Celeb. 1.0 EV..	2.435	2.067
GM - Chevrolet	AGILE LTZ 1.4 MPFI 8V Fle..	4.886	3.608
Toyota	Corolla GLi 1.8 Flex 16V M..	6.026	4.627
VW - VolksWagen	SPACEFOX SPORTLINE/HI..	7.298	6.498

Figura 10: desvio padrão dos preços dos automóveis (Fonte: do Autor).

6 CONCLUSÕES E TRABALHOS FUTUROS

Esse Capítulo trata das conclusões a que foram alcançadas com o trabalho realizado, melhorias que poderiam ser aplicadas ao trabalho e propostas que deem continuidade ao mesmo. Os subcapítulos estão listados nestas ordem: desenvolvimento e extrações, armazenamento dos dados, a escolha de um veículo.

6.1 DESENVOLVIMENTO E EXTRAÇÕES

A simplicidade da biblioteca *Selenium*, disponível para *Python* e diversas outras linguagens, facilita a construção de *Web crawlers* e *Web scrapers*. Com ela, é possível extrair dados da *Web* bastando, apenas, conhecimentos de *HTML* e da linguagem utilizada para manipular o navegador.

No entanto, a escolha dessa biblioteca tornou as extrações demasiadamente lentas, pois é necessário aguardar o processamento de todas as requisições *HTTP* por parte do navegador, para que a página seja construída. Informações que poderiam ser descartadas, como o *download* de imagens, que custam novas requisições ao *browser,* atrasam os próximos passos responsáveis pela extração dos dados de interesse.

Outro ponto negativo do uso da referida biblioteca, é a necessidade de aguardar as animações feitas em *JavaScript*. O efeito de “rolamento suave”, aplicado à página, deve ter sido um dos principais fatores que contribuíram para o prolongamento de toda extração, pois, para cada valor extraído, foi necessário aguardar o término da animação, a qual “deslizava” a página para o local onde estava a Tabela que era o alvo da raspagem de dados.

Concluímos, então, que um *Web scraper* pode realizar a extração dos dados em tempo reduzido se a biblioteca *Selenium* for substituída por um mecanismo

de *scraping* orientado à requisições *HTTP*, como é o caso da biblioteca *Requests*[4] e do *framework Scrapy*[5].

6.2 ARMAZENAMENTO DOS DADOS

Os dados obtidos pelo processo de raspagem foram armazenados em um banco de dados e estruturados de forma a permitir a compatibilidade com softwares de análise, como o Tableau [16], a fim de que seus usuários possam realizar análises e interpretação dos dados.

Com o decorrer das extrações, foi percebido que a Tabela ano_modelo poderia ter sido otimizada. Ao final do processo de raspagem, havia, nessa Tabela, muitos dados duplicados que poderiam ter sido evitados estabelecendo um relacionamento muitos-para-muitos entre as Tabelas ano_modelo e modelo, em vez do relacionamento um-para-muitos, determinado na modelagem do banco de dados desta ferramenta. Nesta nova modelagem, teríamos uma Tabela intermediária entre as mencionadas, o que economizaria espaço de armazenamento.

6.3 A ESCOLHA DE UM VEÍCULO

Com os gráficos disponibilizados no Tableau Public [16], é possível comparar o comportamento do preço dos automóveis ao longo do tempo. Os gráficos disponibilizados podem ser úteis para estimar a depreciação que um veículo sofrerá ao longo do tempo.

Na interação[6] com o gráfico da Figura 9 é possível comparar o comportamento do preço dos automóveis ao longo do tempo. Os gráficos disponibilizados podem ser úteis para estimar a depreciação que um veículo sofrerá ao longo do tempo.

---

[4] http://docs.python-requests.org/en/master/
[5] https://www.scrapy.org
[6] https://goo.gl/7R9Gim

Observando os gráficos, percebe-se que qualquer automóvel perde mais valor de mercado logo no seu primeiro ano. No entanto, é possível notar que existem fatores econômicos que impactam diretamente no mercado automobilístico.

Analisando o gráfico Valor médio dos automóveis ao longo do tempo, é possível perceber que a partir de agosto de 2008 houve uma queda abrupta no preço médio dos veículos. Vemos, também, que a partir de julho de 2017, alguns veículos usados foram valorizados, de forma que seus preços médios de agosto de 2018 e de fevereiro de 2014 são equivalentes.

Dessa forma, propomos como um trabalho futuro, a análise desses dados a fim de detectar as possíveis causas que impactam o mercado automobilístico e os efeitos econômicos decorrentes dessas causas.

REFERÊNCIAS BIBLIOGRÁFICAS

1. BAYER, Michael. **SQLAlchemy 1.2 Documentation**. <https://docs.sqlalchemy.org/> Acesso em 07/11/2018.
2. DALE, Kyre. **Data Visualization with Python and JavaScript: Scrape, Clean, Explore & Transform your Data**, O'Reilly Media Inc., California, 2016.
3. ELMASRI, Ramez; NAVATHE, Shamkant. **Sistemas de banco de dados**, Pearson Education do Brasil, 4ª edição, São Paulo, 2006.
4. EQUIPE DIGERATI. **Hardware para profissionais**, Editora Universo dos Livros, São Paulo, 2009.
5. FILHO, Gilberto; ALEXANDRE, Eduardo. **Introdução a Computação**, Editora da UFPB, João Pessoa, 2014.
6. JARMUL, K; LAWSON, R. **Python Web Scraping**, 2ª edição, John Wiley & Sons, 2011. Packt Publishing, second edition, Birmingham, 2017.
7. JETBRAINS. **PyCharm Python IDE for Professional Developers** <https://www.jetbrains.com/help/pycharm/> Acesso em 02 out. 2018.
8. KREBS, Bruno. **SQLAlchemy ORM Tutorial for Python Developers.** <https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/> Acesso em 01 out. 2018.
9. LIU, Bing. **Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric System and Applications)**, 2nd ed. Heidelberg, New York, Springer, 2011.
10. MOZILLA FOUNDATION. **Firefox Quantum**. <https://www.mozilla.org/pt-BR/firefox/> Acesso em 26 out. 2018.
11. MYERS, Jason; COPELAND, Rick. **Essential SQLAlchemy: Mapping Python to Databases**. O'Reilly Media Inc., California, 2016.
12. ORACLE CORPORATION. **MySQL Workbench 6.3 CE**, <https://dev.mysql.com/downloads/workbench/> Acesso em 07/11/2018.
13. POSTGRESQL, **The PostgreSQL Global Development Group** <https://www.postgresql.org/> Acesso em 02 out. 2018.
14. PSF, **The Python Software Foundation** <https://www.python.org/> Acesso em 01 out. 2018.

15. SELENIUM PROJECT. **Selenium HQ Browser Automation** <https://www.seleniumhq.org/docs/03_webdriver.jsp#setting-up-webdriver-project/> Acesso em 26 out. 2018.
16. TABLEAU DESKTOP. **Tableau Software** <https://www.tableau.com/pt-br/products/desktop/> Acesso em 26 out. 2018.
17. TUCKER, Allen; NOONAN, Robert. **Linguagens de Programação - 2.ed.: Princípios e Paradigmas**, McGraw-Hill Interamericana do Brasil, AMGH Editora, São Paulo, 2010.
18. VARRAZZO, Daniele. **PostgreSQL database adapter for Python** <http://initd.org/psycopg/> Acesso em 26 out. 2018.
19. The Python Wiki. **The Python Software Foundation** <https://wiki.python.org/moin/DatabaseProgramming/> Acesso em 08/12/2018.
20. YAZBEK, Priscila**. Grupo Abril** <https://exame.abril.com.br/seu-dinheiro/8-dicas-para-escolher-o-carro-certo-para-o-seu-perfil/3/> Acesso em 08/12/2018.
21. RODRIGUEZ, Henrique. **O Globo** <https://oglobo.globo.com/economia/carros/metade-dos-brasileiros-pesquisa-na-internet-antes-de-comprar-carro-0km-13487301/> Acesso Em 08/12/2018.
22. ALVES, Patrick. **Uma análise empírica do mercado de revenda de carros usando raspagem de dados da *Internet***, Tese de Doutorado - Curso de Departamento de Economia. Área de Economia Aplicada - Universidade de Brasília, 2016. <https://sucupira.capes.gov.br/sucupira/public/consultas/coleta/trabalhoConclusao/viewTrabalhoConclusao.jsf?popup=true&id_trabalho=4501405/> Acesso em 08/12/2018.
23. CALÒ, Alessandro. **Extração e análise de informações jurídicas públicas**, Trabalho de Conclusão de Curso - Bacharelado em Ciência da Computação no Instituto de Matemática e Estatística da Universidade de São Paulo, 2014. <https://bcc.ime.usp.br/tccs/2014/sandro/Monografia.pdf> Acesso em 08/12/2018.
24. SABINO, Rodolpho; PELLE, Rogers. **Busca Livro: Um Sistema de Buscas Efetivas em Acervos Bibliográficos**, Trabalho de Conclusão de Curso - Curso de Ciência da Computação - Universidade Estadual de Mato Grosso do Sul, 2015. <http://www.comp.uems.br/~PFC/PFC%20165.pdf> Acesso em 08/12/2018.

25. OLIVEIRA, RAFAEL. **Extração de dados web como suporte na elaboração de indicadores do turismo de Minas Gerais: uma iniciativa em *Big Data***, Pós-Graduação em Gestão e Organização do Conhecimento da Escola de Ciência da Informação da Universidade Federal de Minas Gerais, 2017. <https://sucupira.capes.gov.br/sucupira/public/consultas/coleta/trabalhoConclusao/viewTrabalhoConclusao.jsf?popup=true&id_trabalho=5445275> Acesso em 08/12/2018.
26. ERBETTA, Gabriel. **MEUHORÁRIO 2: uma aplicação *Web* para simulação de matrícula**, Trabalho de Conclusão de Curso - Bacharelado em Ciência da Computação - Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal da Bahia, 2017. <https://repositorio.ufba.br/ri/bitstream/ri/22939/1/monografia-gabriel-final.pdf> Acesso em 08/12/2018.
27. ALMEIDA, Luis. **ALUMNI Tool: recuperação de dados pessoais na Web em redes sociais autenticadas**, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2018. <https://www.maxwell.vrac.puc-rio.br/34643/34643.PDF> Acesso em 08/12/2018.