

Bruno Marra (3029)
Daniel Freitas (2304)
Gustavo Viegas (3026)
Vitor Luís (3045)

Trabalho Prático 2

Documentação de Trabalho Prático - TP2

Universidade Federal de Viçosa - Campus Florestal
Gestão, Recuperação e Análise de Informações
Ciência da Computação

Florestal
4 de Novembro de 2019

Lista de ilustrações

Figura 1 – Comparação dos valores de SGBDs para um Datawarehouse (DUNN, 2016)	8
Figura 2 – Os passos para modelagem de um Datawarehouse	12
Figura 3 – Modelo estrela do Datawarehouse	14
Figura 4 – Modelo estrela do Datawarehouse	15
Figura 5 – Dados do cubo de dados criado pelo comando Listing 5.2	17
Figura 6 – Visualização do cubo de dados <i>cubeavgveiculos</i> pelo Metabase.	18
Figura 7 – Visualização do cubo de dados <i>cubetotalvenda</i> pelo Metabase.	19
Figura 8 – Dados do cubo de dados criado pelo comando Listing 5.3	19
Figura 9 – Dados do cubo de dados criado pelo comando Listing 5.4	21
Figura 10 – Visualização do cubo de dados <i>cubesinistroveiculo</i> pelo Metabase.	21
Figura 11 – Dados do cubo de dados criado pelo comando Listing 5.5	23
Figura 12 – Visualização do cubo de dados <i>cubesinistrocliente</i> pelo Metabase.	23
Figura 13 – Maior média de valor de compra Listing 5.6	24
Figura 14 – Distribuição de sinistros por estado Listing 5.7	25
Figura 15 – Distribuição de sinistros por modelo Listing 5.8	26
Figura 16 – Relação entre o ano de nascimento e a quantidade de sinistros associados - Listing 5.9	27
Figura 17 – Cluster - Relação entre o ano de nascimento e a quantidade de sinistros associados - Listing 5.9	28
Figura 18 – Venda de veículos por período	28
Figura 19 – Dashboard de vendas por período	29
Figura 20 – Dashboard de vendas por ano	30
Figura 21 – Dashboard de vendas por mês	30
Figura 22 – Dashboard de vendas por dia	31
Figura 23 – Dashboard de preço médio de compra de veículos	32
Figura 24 – Quantidade de sinistros e valor gasto pela seguradora	32
Figura 25 – Quantidade de sinistros por modelo	33
Figura 26 – Total gasto em sinistros por modelo	33
Figura 27 – Distribuição de sinistros por região	34
Figura 28 – Distribuição de sinistros por sexo	34
Figura 29 – Modelo gerado com NaiveBayes.	36
Figura 30 – Modelo gerado com JF8.	37
Figura 31 – Resultado do NaiveBayes.	37
Figura 32 – Resultado do JF8.	38

Sumário

1	Introdução	6
2	Ambiente	7
2.1	Subindo a aplicação	7
3	Tecnologias Utilizadas	8
3.1	Escolha do SGBD	8
3.2	Demais softwares utilizados	9
4	Configuração do BD Transacional	10
5	Atividades Realizadas	12
5.1	Atividade 1	12
5.1.1	Modelagem Multidimensional	12
5.1.1.1	O processo do negócio	12
5.1.1.2	Granularidade	13
5.1.1.3	Dimensões	13
5.1.1.4	Fatos	13
5.1.1.5	Modelo resultante	14
5.1.2	Criação do Banco de Dados do Datawarehouse	14
5.2	Atividade 2	16
5.2.1	Cubo de Dados - <i>cubeavgveiculos</i>	16
5.2.2	Cubo de Dados - <i>cubetotalvenda</i>	18
5.2.3	Cubo de Dados - <i>cubesinistroveiculo</i>	20
5.2.4	Cubo de Dados - <i>cubesinistrocliente</i>	21
5.3	Atividade 3	23
5.3.1	Média do valor de compra mais alto	24
5.3.2	Distribuição de Sinistros	24
5.3.3	Custo dos sinistros por modelo	25
5.3.4	Relação entre ano de nascimento e quantidade de sinistros	26
5.4	Atividade 4	28
5.4.1	O cubo de dados utilizado	28
5.4.2	Drill-down sobre os dados	29
5.5	Atividade 5	31
5.5.1	Dashboard Veículos	31
5.5.2	Dashboard Sinistros	32

5.6	Atividade 6	34
5.7	Atividade 7	38
5.7.1	Conexão com o Banco	38
5.7.2	Obtendo os Dados	38
5.7.3	Análise Feita	38
5.8	Atividade 8	39
6	Conclusão	40
	Referências	41

1 Introdução

Neste trabalho são apresentadas e discutidas as tarefas realizadas com base no modelo relacional e nos dados propostos. Para cada tarefa proposta, existe uma seção correspondente no documento.

O contexto proposto é de uma seguradora de veículos, na qual tivemos que pensar na modelagem do modelo do banco de dados warehouse do mesmo. A modelagem foi realizada de forma a facilitar nossas análises.

Para realização das atividades, utilizamos softwares de BI, de Machine Learning e um SGBD capaz de suportar os bancos transacionais e de warehouse, de forma a facilitar nosso trabalho, tanto para geração dos cubos de dados pertinentes, quanto para geração de relatórios úteis para responder às perguntas dos executivos.

Concomitantemente, o Machine Learning foi utilizado para fazer inferências sobre os dados armazenados, bem como fazer previsões úteis para os donos e executivos da Seguradora.

Por fim, desenvolvemos ainda um outro front-end para comparar as inferências e acertos da ferramenta que utilizamos com outras técnicas e comparamos os resultados obtidos por ambas as ferramentas.

2 Ambiente

Para o desenvolvimento do trabalho foram utilizados N containers principais que utilizam das imagens a seguir:

- **postgres** - Tag *12.0* da imagem oficial (do *DockerHub*) do banco de dados relacional PostgreSQL.
- **dpage/pgadmin4** - Tag *4.4.14* da imagem oficial (do *DockerHub*) da ferramenta web de administração do PostgreSQL.

Os containers foram orquestrados com o *Docker Compose*, e suas configurações de ambiente, portas, network e etc. podem ser consultadas no arquivo **docker-compose.yml** na raiz do projeto.

2.1 Subindo a aplicação

Para subir a aplicação, em qualquer máquina com o *Docker* e *Docker Compose* devidamente instalados, basta executar no terminal, na raiz do projeto:

```
1 docker-compose up
```

O *PGAdmin* estará disponível em <http://localhost>. Note que o banco de dados precisa dos arquivos .sql de inserção em sua pasta, conforme o arquivo *README.md*, na raiz do projeto, explica, para os dados serem inseridos.

3 Tecnologias Utilizadas

3.1 Escolha do SGBD

Após um estudo sobre o atual mercado e os valores de licenças dos mesmos, como mostra a [Figura 1](#), vimos que o SQL Server, como sugerido de usar na descrição do trabalho, possui um custo altíssimo. Em uma situação real, onde estaríamos aconselhando ou construindo um Datawarehouse para um cliente, poderíamos economizar milhares de dólares ao escolher um SGBD *open-source*.

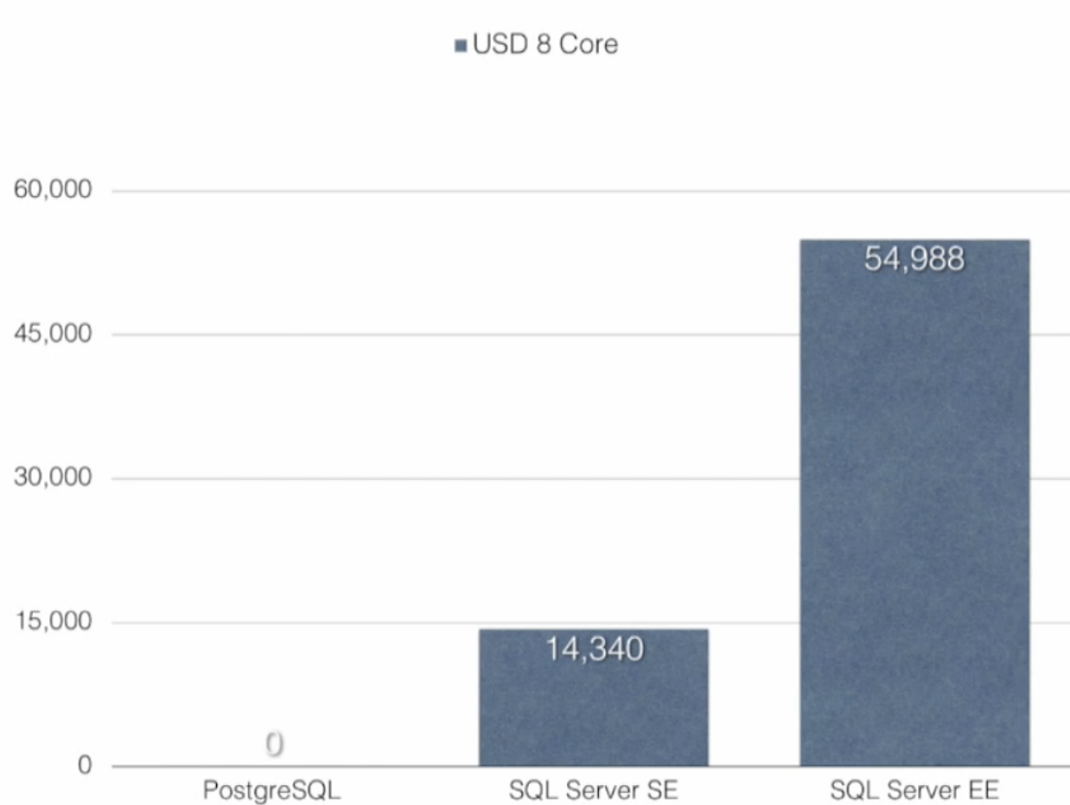


Figura 1 – Comparação dos valores de SGBDs para um Datawarehouse ([DUNN, 2016](#))

Por esse motivo, é interessante o uso do **PostgreSQL** (também conhecido simplesmente por **Postgres**), um SGBD *open-source* e maduro, com diversas características avançadas para sistema OLAP. Curiosamente, uma das ferramentas que mais crescem neste meio, principalmente por ser um serviço *cloud*, é o **Amazon Redshift**, é baseado no Postgres ([AMAZON WEB SERVICES, 2019](#)).

Uma outra vantagem considerável de usarmos o Postgres tanto no banco de dados

transacional a ser consumido como no banco de dados a ser modelado para o Datawarehouse, é o uso do poderoso comando **COPY** ([THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2019b](#)), que permite a fácil exportação dos dados no processo *E* da operação *ETL* (Extract, Transform, Load).

Dessa forma, o SGBD escolhido para este trabalho é o Postgres, visto que a sua utilização é bem fundada no ambiente de *datawarehousing* e que adquiriríamos conhecimento prático para sua utilização em caso eventual de um projeto no futuro de nossas vidas profissionais.

3.2 Demais softwares utilizados

Além da Escolha do SGBD, utilizamos mais alguns softwares para nos auxiliar na realização das atividades, tais como:

- **Metabase** - Sistema de BI para geração de relatórios sobre os dados do warehouse. Link: <https://metabase.com/>
- **Weka** - Sistema de Machine Learning para predição e análise sobre os dados do warehouse. Link: <https://www.cs.waikato.ac.nz/ml/weka/>

4 Configuração do BD Transacional

Como estamos usando docker e docker-compose, para criarmos o banco de dados e popular ele, basta-se adicionar os arquivos `.sql` em um diretório específico, o `/docker-entrypoint-initdb.d/` que ele será executado sequencialmente em ordem alfabética. No arquivo `docker-compose.yml`, é configurado um link do diretório do projeto com este diretório, onde o container ao ser criado já executa automaticamente estes arquivos `.sql`.

Dessa forma, o primeiro arquivo inserido nesse diretório foi o **01-create.sql**. O número no prefixo é apenas para garantir que a execução na sequência correta. Este arquivo é o responsável por criar as tabelas de acordo com o diagrama incluído na especificação do trabalho.

Um detalhe é que nas criações das tabelas, não foi especificado as *constraints* de chaves estrangeiras, já que temos um volume enorme de dados e é recomendado pelo Postgres a inserção das *constraints* após a inserção para melhorar a performance das inserções ([THE POSTGRES GLOBAL DEVELOPMENT GROUP, 2019a](#)).

Após a criação das tabelas, cada arquivo `.txt` enviado para os autores do trabalho para a inserção dos dados foi processado em uma pequena aplicação, feita em **Python**, para converter os inserts em arquivos `.csv`, e dessa forma ser muito rapidamente inseridos no banco de dados através do poderoso comando **COPY** do Postgres, já mencionado anteriormente. Para executar o conversor dos arquivos `.txt` de entrada para os `.csv` de saída, basta inserir os arquivos no diretório `"copy-maker/input"` e executar na raiz do projeto o comando a seguir:

```
1 docker-compose run copy python3 main.py
```

Dessa forma, pra cada tabela, existe também um arquivo `.sql` que segue o padrão dos números no prefixos para garantir a ordem de inserção. O diretório de inicialização do banco de dados inclui também os arquivos:

- **02-insert-montadoras.sql**
- **03-insert-modelos.sql**
- **04-insert-clientes.sql**
- **05-insert-carros.sql**
- **06-insert-seguros.sql**
- **07-insert-sinistros.sql**

onde cada um respectivamente usa o comando COPY para inserir os dados no banco de dados como no exemplo:

```
1 COPY sinistro(id_sinistro, data_sinistro, valor,  
    carro_id_carro) FROM '/data/csv/sinistro.csv' WITH  
    (FORMAT csv);
```

E por último, um arquivo **08-create-constraints.sql** que adiciona as *constraints*.

5 Atividades Realizadas

5.1 Atividade 1

5.1.1 Modelagem Multidimensional

Iniciamos a modelagem de nosso datawarehouse em um esquema de **Estrela**, por diversos motivos: é um modelo mais simples e fácil de modelar; é eficiente com queries rápidas; pode ser transformado mais facilmente a partir dele para outros esquemas sem a necessidade de transformar completamente o esquema. Se durante a modelagem aparecesse a eventual necessidade de usar outro esquema, o faríamos, mas inicialmente foi definido o modelo em um esquema de estrela.

Conforme mostra a [Figura 2](#), precisamos identificar o **PORQUÊ** (Processos de Negócio), o **QUANTO** (granularidade), as **DIMENSÕES**, etc.

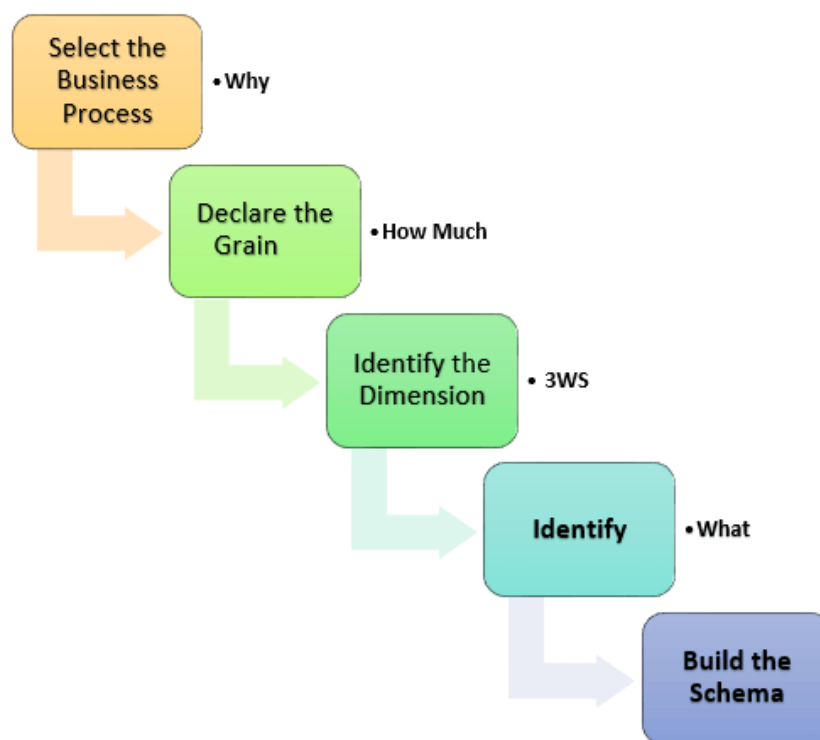


Figura 2 – Os passos para modelagem de um Datawarehouse

5.1.1.1 O processo do negócio

O primeiro passo para a modelagem do datawarehouse é identificar o objetivo do negócio. No caso da seguradora, algumas informações de interesse seriam de identificar:

- Quantos sinistros ocorreram com veículos de uma montadora em um ano de vigência.
- Quais modelos de veículos de uma montadora é mais provável de se ter um sinistro.
- Qual a *idade* dos carros que possuíram maior número de sinistros.
- Qual o perfil de clientes mais provável de se ter um sinistro.

5.1.1.2 Granularidade

A granularidade é o nível mais baixo de informação salva na tabela. As informações de interesse listadas anteriormente, mudam muito pouco diariamente, tampouco semanalmente ou mensalmente, fazendo mais sentido a análise a cada ano de vigência, visto que a maioria dos contratos de seguro são renovadas anualmente, onde essas informações são valiosas para poder avaliar o prêmio do veículo de um cliente, e que novos modelos de carros surgem no mercado.

Portanto, a granularidade da informação no nosso contexto é **anual**.

5.1.1.3 Dimensões

Dimensões provê o contexto em torno de um evento do negócio. Em termos simples ele descreve *quem*, *o quê* e *o onde* de um Fato.

Pegando as informações da [subseção 5.1.1.1](#), podemos identificar as dimensões de cada item como:

- Veículos, Montadoras, Ano
- Modelos de Veículos, Ano
- Carros, Ano
- Clientes, Ano

Fica claro que o objeto "**Ano**" aparece frequentemente em todas as dimensões devido a granularidade definida anteriormente. Além disso, alguns desses objetos podem ser agregados como um só objeto, como **Veículos**, **Modelos de Veículos** e **Carros** podem ser interpretados como um único objeto de dimensão, com todas essas características.

5.1.1.4 Fatos

Fatos são as métricas ou medidas de um negócio. No caso das informações modeladas anteriormente, o fato principal que envolve todo o datawarehouse é o **Sinistro**. É a

partir destes fatos que a seguradora poderá tirar informações relevantes de como avaliar um seguro, quais perfis de clientes deve captar, entre outros.

Outro possível fato em potencial seria o seguro, mas inicialmente, o modelo possui apenas o fato sinistro, e se houver a necessidade nas análises posteriores, o modelo poderia evoluir para atender as necessidades.

5.1.1.5 Modelo resultante

O modelo final pode ser interpretado conforme mostra a [Figura 3](#):

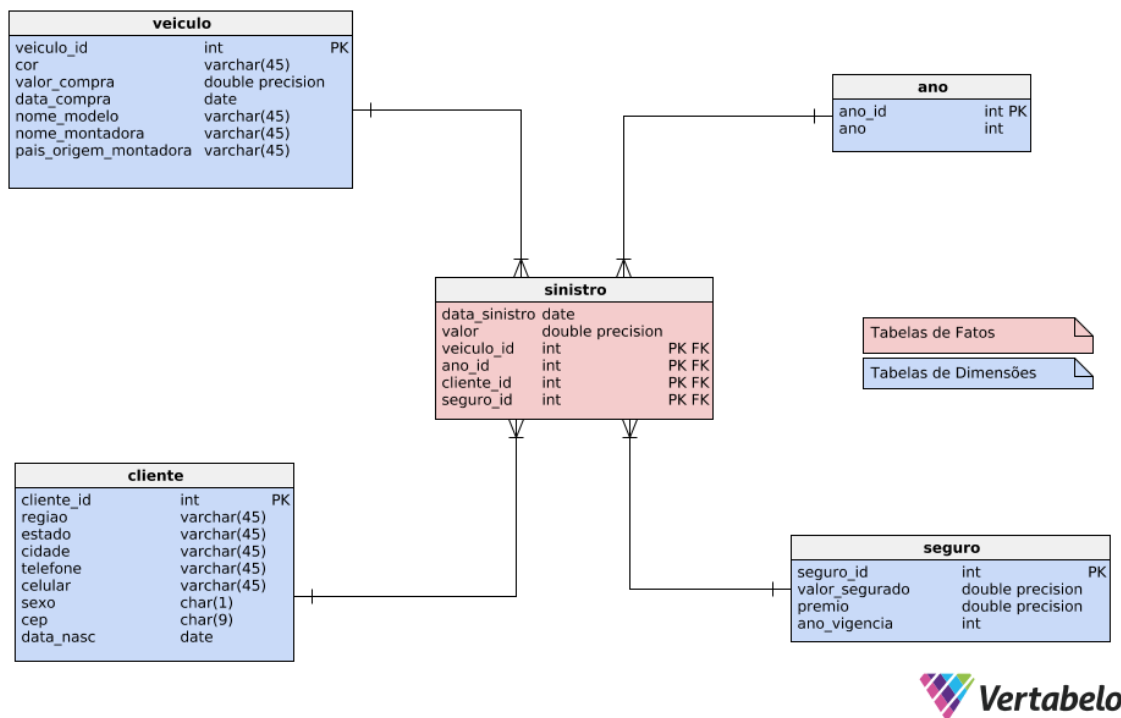


Figura 3 – Modelo estrela do Datawarehouse

O modelo foi construído utilizando a ferramenta [Vertabelo](#), onde foi criada uma conta estudantil gratuita. A ferramenta também permite a exportação do SQL para criação das tabelas e suas *constraints* para o Postgres. Foi utilizado esse SQL gerado com algumas modificações, para separar a criação das tabelas e constraints para otimizar a performance conforme já explicado no [Capítulo 4](#).

5.1.2 Criação do Banco de Dados do Datawarehouse

Para realizar o processo de **ETL** (*Extract, Transform & Load*), foi adotada uma biblioteca, em Python, de criação de pipelines, o [Luigi](#).

O processo foi desenvolvido com o uso de containers *Docker*, onde a conexão com os bancos de dados foi feita através de uma rede interna. Como não é o ponto focar do trabalho, os detalhes dessa conexão foram omitidos.

Uma vez conectado com ambos os bancos de dados, o fluxo se dá conforme a [Figura 4](#). Cada entidade tem um nó no fluxo de pipeline onde ele extrai os dados com um *.sql* correspondente, já formatando os dados da maneira modelada com o uso puramente de SQL.

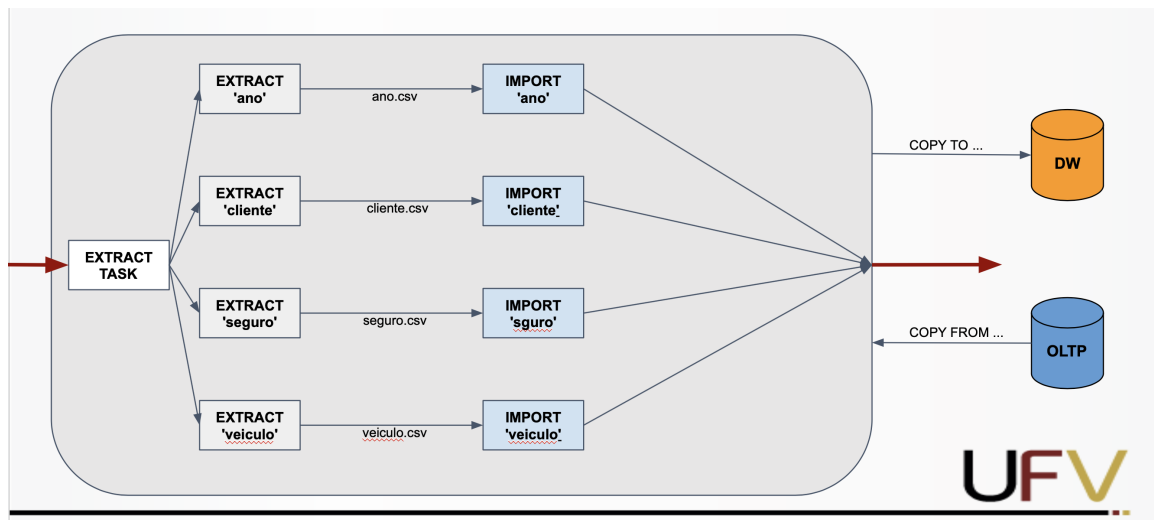


Figura 4 – Modelo estrela do Datawarehouse

O resultado das queries é armazenado em um *.csv* com o uso do comando **COPY** do Postgres, já citado neste trabalho. Este arquivo é lido e importado no banco de dados do datawarehouse também com o comando **COPY**. O uso de arquivos *.csv* só foi necessário pois decidimos isolar completamente os bancos de dados, e não sendo possível um único comando **COPY** entre dois bancos isolados.

Um exemplo de SQL usado para a extração pode ser visto no [Listing 5.1](#).

Listing 5.1 – SQL de extração de Sinistros

```
SELECT
    an.ano_id ,
    cr.id_carro AS veiculo_id ,
    sg.id_seguro AS seguro_id ,
    cl.id_cliente AS cliente_id ,
    sn.data_sinistro ,
    sn.valor
FROM sinistro sn
INNER JOIN carro cr ON sn.carro_id_carro = cr.id_carro
```

```
INNER JOIN seguro sg ON sg.carro_id_carro = cr.id_carro
AND sg.ano_vigencia = EXTRACT(YEAR FROM sn.data_sinistro)
INNER JOIN (
    SELECT
        ROW_NUMBER() OVER () AS ano_id,
        EXTRACT(YEAR FROM data_sinistro) AS ano
    FROM sinistro
    GROUP BY ano
    ORDER BY ano
) an ON EXTRACT(YEAR FROM sn.data_sinistro) = an.ano
INNER JOIN cliente cl ON cr.cliente_id_cliente = cl.id_cliente;
```

5.2 Atividade 2

O modelo dimensional pode ser visualizado na [Figura 3](#). Abaixo serão mostrados os diferentes cubos de dados criados.

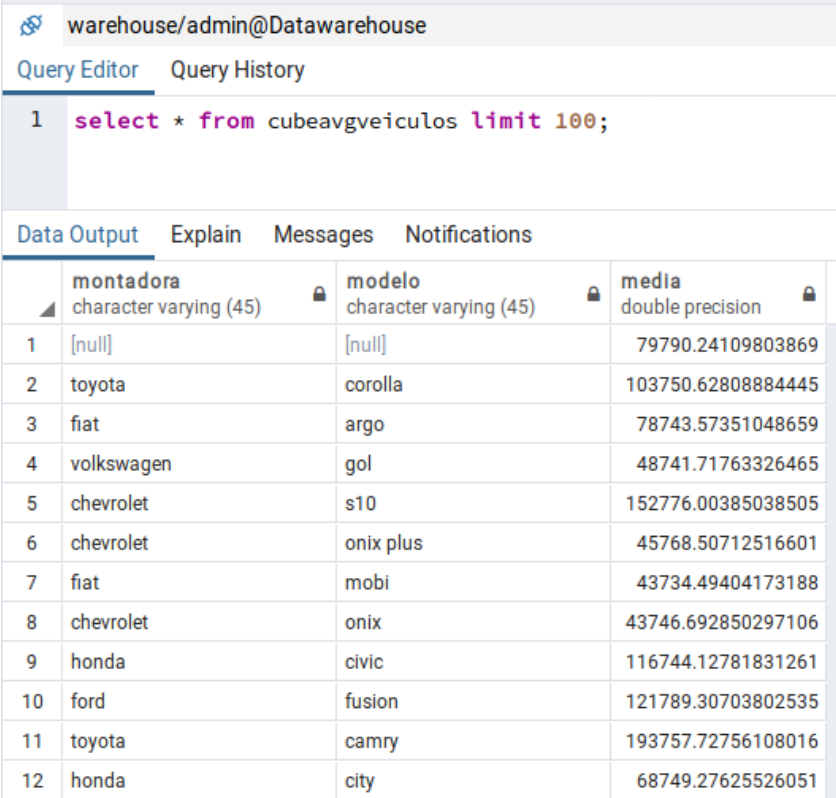
5.2.1 Cubo de Dados - *cubeavgveiculos*

O PostgreSQL permite uma facilidade quanto à criação dos cubos de dados. Por exemplo, para se criar uma visão para um cubo de dados formado pelos campos *nome_montadora* e *nome_modelo*, o comando abaixo pode ser executado:

Listing 5.2 – Cubo de Dados - Valor médio de compra referente a cada montadora e modelo

```
CREATE VIEW cubeavgveiculos AS
SELECT nome_montadora AS montadora, nome_modelo AS modelo,
    AVG(valor_compra) AS media
FROM veiculo
GROUP BY CUBE (nome_montadora, nome_modelo);
```

Alguns dos possíveis dados de serem consultados são mostrados abaixo na [Figura 5](#).



The screenshot shows a PostgreSQL Query Editor interface. At the top, the user is logged in as 'warehouse/admin@Datawarehouse'. Below the login bar, there are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, displaying a SQL query: `1 select * from cubeavgveiculos limit 100;`. Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with three columns: 'montadora' (character varying (45)), 'modelo' (character varying (45)), and 'media' (double precision). The table contains 12 rows of data, starting with a row where 'montadora' is null and 'media' is 79790.24109803869, followed by rows for various car models and manufacturers like Toyota Corolla, Fiat Argo, Volkswagen Gol, Chevrolet S10, Chevrolet Onix Plus, Chevrolet Onix, Fiat Mobi, Chevrolet Onix, Honda Civic, Ford Fusion, Toyota Camry, and Honda City.

	montadora character varying (45)	modelo character varying (45)	media double precision
1	[null]	[null]	79790.24109803869
2	toyota	corolla	103750.62808884445
3	fiat	argo	78743.57351048659
4	volkswagen	gol	48741.71763326465
5	chevrolet	s10	152776.00385038505
6	chevrolet	onix plus	45768.50712516601
7	fiat	mobi	43734.49404173188
8	chevrolet	onix	43746.692850297106
9	honda	civic	116744.12781831261
10	ford	fusion	121789.30703802535
11	toyota	camry	193757.72756108016
12	honda	city	68749.27625526051

Figura 5 – Dados do cubo de dados criado pelo comando Listing 5.2

De acordo com (POSTGRESQL, 1996) e (POSTGRESQL TUTORIAL, 2019), a cláusula *CUBE* é uma subcláusula da cláusula *GROUP BY*, e permite gerar múltiplos conjuntos de agrupamento. Assim, a consulta que a utiliza gera todos os possíveis conjuntos de agrupamentos baseados nas colunas de dimensão atribuídas ao cubo de dados. Neste sentido, a visão criada pelo uso do comando Listing 5.2 possui, na verdade, todas as seguintes combinações: {(nome_montadora, nome_modelo), (nome_montadora), (nome_modelo), ()}. Este é o motivo da primeira linha da Figura 5 incluir o conjunto vazio para as dimensões. A média associada ao conjunto vazio inclui todos os registros da visão, ou seja, ela representa o valor médio de compra de veículos considerando-se todos os modelos de todas as montadoras.

Com este cubo de dados, é possível retirar informações como "Valor médio de compra dos veículos da montadora Fiat", "Valor médio de compra dos veículos do modelo Gol", "Valor médio de compra de todos os veículos", etc.

A ferramenta de análise de dados Metabase (METABASE, 2019) foi utilizada. Ela permite a visualização dos cubos de dados criados. Veja abaixo na Figura 6 a visualização do cubo de dados pelo Metabase. Essa visualização e as demais dessa subseção foram geradas automaticamente pelo Metabot, uma IA presente no Metabase.

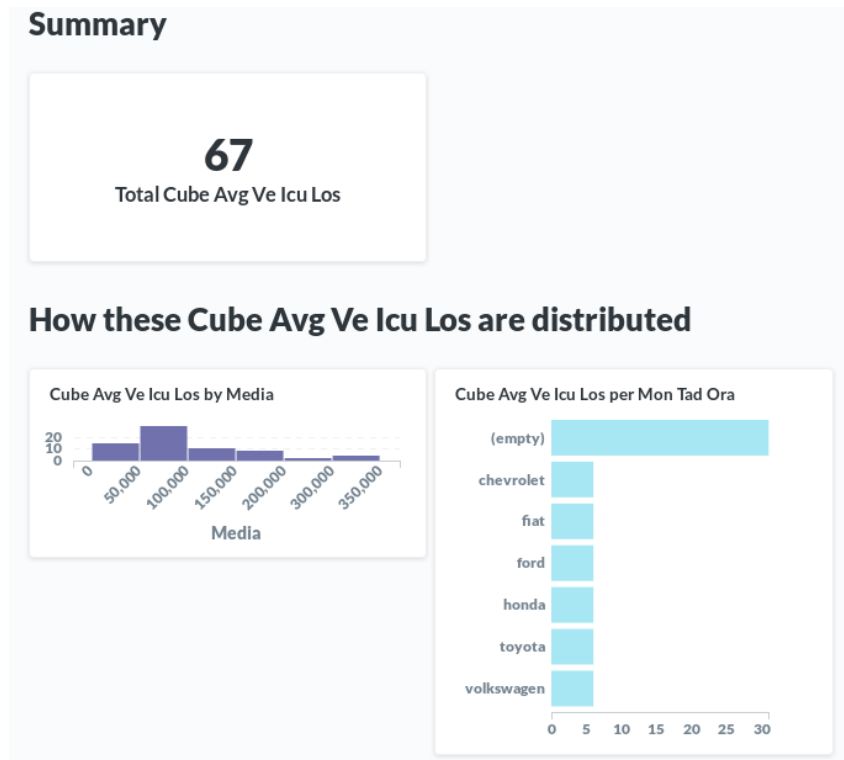


Figura 6 – Visualização do cubo de dados *cubeavgveiculos* pelo Metabase.

5.2.2 Cubo de Dados - *cubetotalvenda*

Veja abaixo na [Figura 7](#) a visualização do cubo de dados pelo Metabase. A descrição do cubo de dados correspondente pode ser visualizado em [Listing 5.3](#). Note a separação entre Mês, Dia e Ano, que são as dimensões do cubo considerado. Note que a função de agregação atua sobre todos os subconjuntos e, desta forma, com este cubo de dados é possível saber informações como "Valor total das vendas no mês de Outubro", "Valor total das vendas no ano de 2015", "Valor total considerando-se todas as vendas", etc.

Listing 5.3 – Cubo de Dados - Soma dos valores de compra dos veículos por uma data

```
CREATE VIEW cubetotalvenda AS
SELECT
  EXTRACT (YEAR FROM data_compra) ano ,
  EXTRACT (MONTH FROM data_compra) mes ,
  EXTRACT (DAY FROM data_compra) dia ,
  SUM (valor_compra) AS total
FROM veiculo
GROUP BY
  CUBE (
```

```
EXTRACT (YEAR FROM data_compra) ,  
EXTRACT (MONTH FROM data_compra) ,  
EXTRACT (DAY FROM data_compra)  
);
```

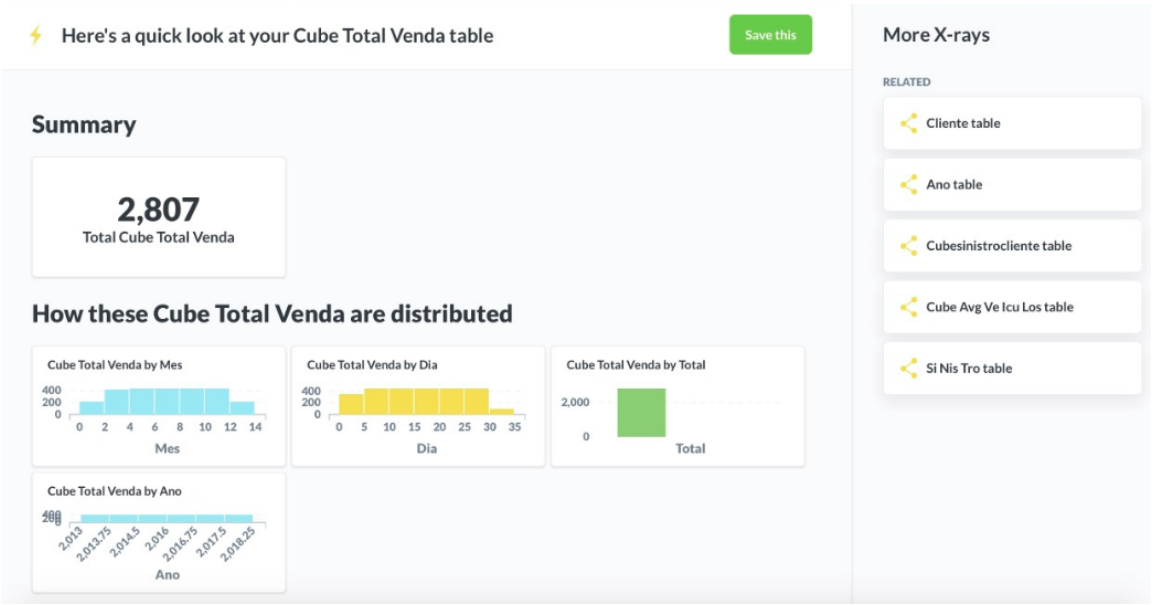


Figura 7 – Visualização do cubo de dados *cubetotalvenda* pelo Metabase.

Para uma visão mais ampla dos dados do cubo, veja a [Figura 8](#) abaixo.

warehouse/admin@Datawarehouse
Query Editor
1 SELECT *
2 FROM cubetotalvenda
3 LIMIT 100;
Data Output
ano double precision
mes double precision
dia double precision
total double precision
1 [null]
2 2013
3 2015
4 2013
5 2013
6 2018
7 2014
8 2016
9 2018
10 2016
11 2014
12 2013
13 2017

Figura 8 – Dados do cubo de dados criado pelo comando [Listing 5.3](#)

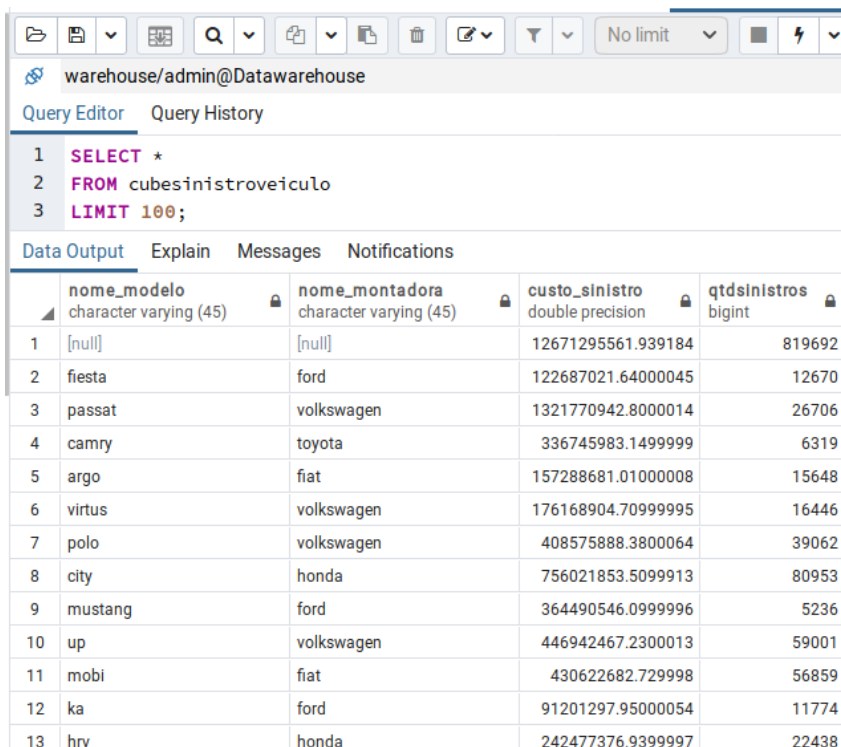
5.2.3 Cubo de Dados - *cubesinistroveiculo*

O cubo de dados a seguir ([Listing 5.4](#)) permite que certas informações sejam obtidas, como "Quantos sinistros ocorreram e quais seus custos, considerando-se a montadora Fiat?", "Quantos sinistros foram causados por veículos do modelo Gol?", "Quantos sinistros e qual o custo total, considerando-se todos os modelos e montadoras existentes?", etc.

Uma visualização de parte dos dados presentes neste cubo pode ser encontrada na [Figura 9](#). A [Figura 10](#) mostra a visualização do cubo de dados no Metabase.

Listing 5.4 – Cubo de Dados - Custo e quantidade de sinistros por veículo

```
CREATE VIEW cubesinistroveiculo AS
SELECT
    nome_modelo ,
    nome_montadora ,
    SUM(valor) AS custo_sinistro ,
    COUNT(*) AS qtdSinistros
FROM sinistro
NATURAL JOIN veiculo
GROUP BY
    CUBE(
        nome_modelo ,
        nome_montadora
    );
```



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with various icons. Below it, the user is logged in as 'warehouse/admin@Datawarehouse'. The 'Query Editor' tab is active, showing a SQL query:

```

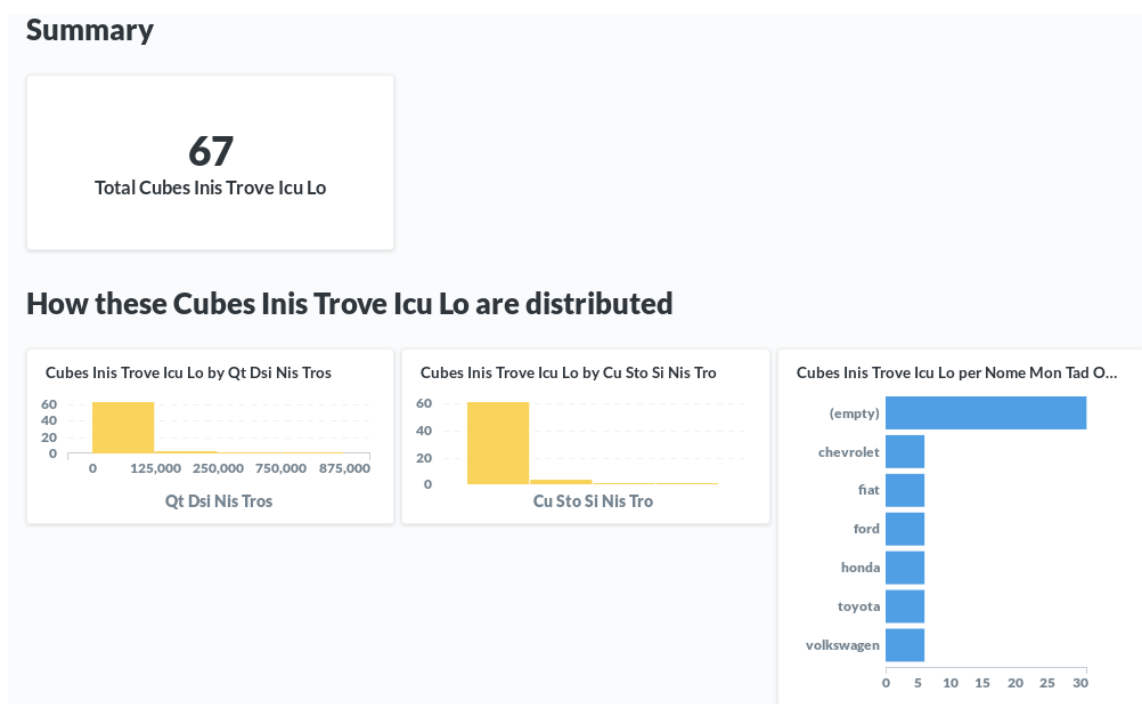
1 SELECT *
2 FROM cubesinistroveiculo
3 LIMIT 100;

```

Below the query, the 'Data Output' tab is active, displaying a table with 4 columns: 'nome_modelo', 'nome_montadora', 'custo_sinistro', and 'qtdsinistros'. The table contains 13 rows of data.

	nome_modelo character varying (45)	nome_montadora character varying (45)	custo_sinistro double precision	qtdsinistros bigint
1	[null]	[null]	12671295561.939184	819692
2	fiesta	ford	122687021.64000045	12670
3	passat	volkswagen	1321770942.8000014	26706
4	camry	toyota	336745983.14999999	6319
5	argo	fiat	157288681.01000008	15648
6	virtus	volkswagen	176168904.70999995	16446
7	polo	volkswagen	408575888.3800064	39062
8	city	honda	756021853.5099913	80953
9	mustang	ford	364490546.0999996	5236
10	up	volkswagen	446942467.2300013	59001
11	mobi	fiat	430622682.729998	56859
12	ka	ford	91201297.95000054	11774
13	hrv	honda	242477376.9399997	22438

Figura 9 – Dados do cubo de dados criado pelo comando Listing 5.4

Figura 10 – Visualização do cubo de dados *cubesinistroveiculo* pelo Metabase.

5.2.4 Cubo de Dados - *cubesinistrocliente*

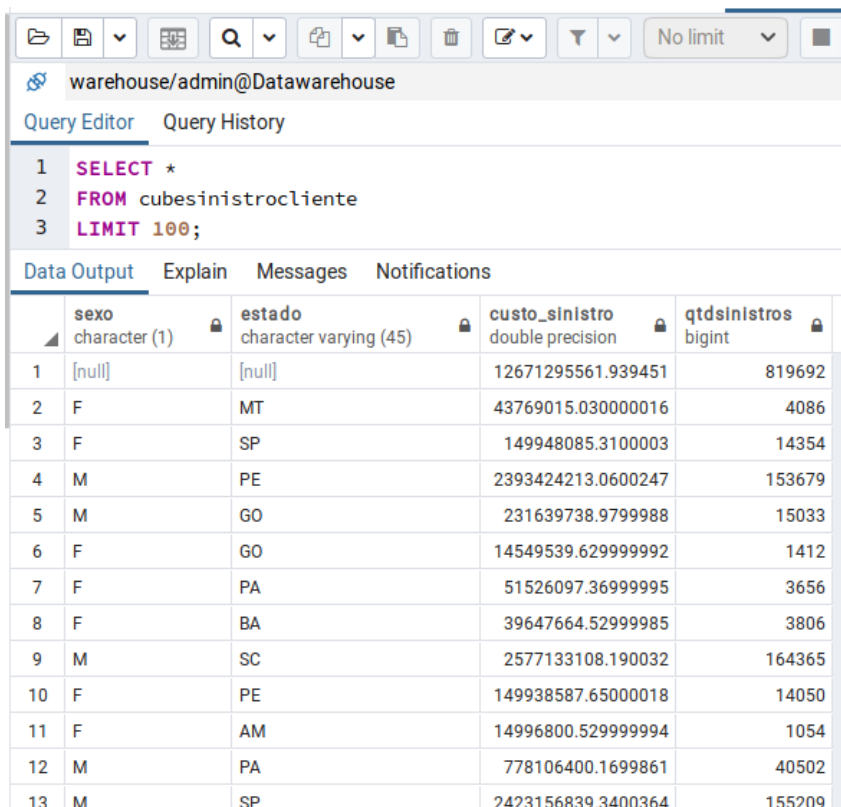
O cubo de dados a seguir (Listing 5.5) permite que certas informações sejam obtidas, como "Quantos sinistros ocorreram e quais seus custos, considerando-se o estado

de Minas Gerais?", "Quantos sinistros foram causados por pessoas do sexo Masculino no estado de Minas Gerais?", "Quantos sinistros e qual o custo total, considerando-se todos os sinistros já ocorridos?", etc.

Uma visualização de parte dos dados presentes neste cubo pode ser encontrada na [Figura 11](#). A visualização pelo Metabase se encontra na [Figura 12](#).

Listing 5.5 – Cubo de Dados - Custo e quantidade de sinistros por cliente

```
CREATE VIEW cubesinistrocliente AS  
SELECT  
    sexo ,  
    estado ,  
    SUM(valor) AS custo_sinistro ,  
    COUNT(*) AS qtdSinistros  
FROM sinistro  
NATURAL JOIN cliente  
GROUP BY  
    CUBE (  
        sexo ,  
        estado  
    );
```



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with icons for file operations, search, and execution. Below the toolbar, the user is logged in as 'warehouse/admin@Datawarehouse'. The 'Query Editor' tab is active, showing a SQL query:

```

1 SELECT *
2 FROM cubesinistrocliente
3 LIMIT 100;

```

Below the query, the 'Data Output' tab is active, displaying a table with 5 columns: 'sexo', 'estado', 'custo_sinistro', and 'qtdsinistros'. The table contains 13 rows of data.

	sexo character (1)	estado character varying (45)	custo_sinistro double precision	qtdsinistros bigint
1	[null]	[null]	12671295561.939451	819692
2	F	MT	43769015.030000016	4086
3	F	SP	149948085.3100003	14354
4	M	PE	2393424213.0600247	153679
5	M	GO	231639738.9799988	15033
6	F	GO	14549539.629999992	1412
7	F	PA	51526097.36999995	3656
8	F	BA	39647664.52999985	3806
9	M	SC	2577133108.190032	164365
10	F	PE	149938587.65000018	14050
11	F	AM	14996800.529999994	1054
12	M	PA	778106400.1699861	40502
13	M	SP	2423156839.3400364	155209

Figura 11 – Dados do cubo de dados criado pelo comando [Listing 5.5](#)

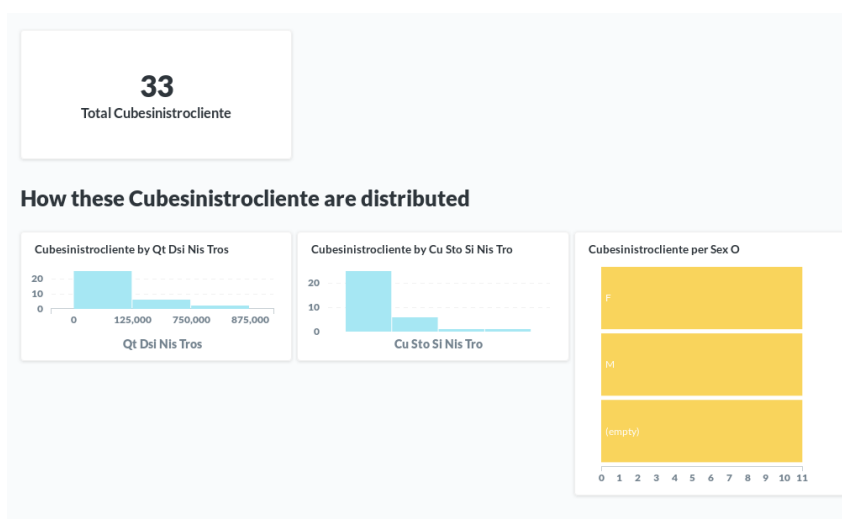


Figura 12 – Visualização do cubo de dados *cubesinistrocliente* pelo Metabase.

5.3 Atividade 3

Para execução dessa atividade, foram utilizados os próprios cubos de dados criados pela Atividade 2, de forma a fazermos consultas sobre os cubos de dados e obtermos informações que respondam perguntas interessantes para os executivos. Foi proposto uma pergunta por cubo de dados criado, porém outras perguntas podem ser respondidas pelos

mesmos variando somente a consulta. Ainda, a aplicação Weka ([WEKA, 2019](#)) também foi utilizada para responder uma outra pergunta sem a utilização dos cubos de dados.

5.3.1 Média do valor de compra mais alto

A primeira pergunta que fizemos é a seguinte: "Qual a média do valor de compra mais alto dentre os veículos da seguradora?". A consulta em [Listing 5.6](#) visa responder isso.

Listing 5.6 – Média de preço de compra mais alto de veículos

```
SELECT montadora , modelo , media
FROM public.cubeavgveiculos
ORDER BY media DESC
LIMIT 1
```

O resultado obtido por essa consulta responde a pergunta anterior, conforme apresentado na [Figura 13](#).




	Data Output	Explain	Messages	Notifications
	 montadora character varying (45)		modelo character varying (45)	 media double precision
1	chevrolet		camaro	315739.48860240757

Figura 13 – Maior média de valor de compra [Listing 5.6](#)

Dessa forma, o executivo pode tomar uma decisão acerca do valor que ele deve considerar sobre o preço desse tipo de veículo. A mesma consulta pode ser feita para o menor, bem como a média geral dos veículos, com poucas modificações na consulta.

5.3.2 Distribuição de Sinistros

A segunda pergunta proposta foi a seguinte: "Como se dá a distribuição de sinistros por estado?".

Listing 5.7 – Distribuição de sinistros por estado

```
SELECT estado , ROUND(CAST(custo_sinistro AS NUMERIC) , 2)
AS custo_sinistro , qtddsinistros
FROM public.cubesinistrocliente
WHERE estado IS NOT NULL AND sexo IS NULL
ORDER BY custo_sinistro DESC
```


O resultado obtido por essa consulta responde a pergunta anterior, conforme apresentado na Figura 14.

	Data Output	Explain	Messages	Notifications
	estado character varying (45)		custo_sinistro numeric	qtdsinistros bigint
1	SC		2735947152.37	179245
2	SP		2573104924.65	169563
3	PE		2543362800.71	167729
4	MG		1053061321.32	69206
5	RS		1018191534.39	66352
6	PA		829632497.54	44158
7	MT		744199036.42	49097
8	BA		680840876.63	44773
9	AM		246766139.30	13124
10	GO		246189278.61	16445

Figura 14 – Distribuição de sinistros por estado [Listing 5.7](#)

Dessa forma, o executivo pode tomar decisões acerca do preço determinado para o seguro de acordo com a quantidade de sinistros ocorridos em uma dada região, bem como o custo desses sinistros para a seguradora, visando maximizar seu lucro.

5.3.3 Custo dos sinistros por modelo

A terceira pergunta proposta, foi: "Quais modelos tem causado maior prejuízo para a seguradora, com relação ao custo de seus sinistros?".

Listing 5.8 – Distribuição de sinistros por modelo

```

SELECT nome_modelo , nome_montadora ,
       ROUND(CAST( custo_sinistro AS NUMERIC) , 2)
       AS custo_sinistro , qtdsinistros
FROM public.cubesinistroveiculo
WHERE nome_modelo IS NOT NULL
      AND nome_montadora IS NOT NULL
ORDER BY custo_sinistro DESC

```

O resultado obtido por essa consulta responde a pergunta anterior, conforme apresentado na Figura 15.

	Data Output	Explain	Messages	Notifications
	nome_modelo character varying (45)	nome_montadora character varying (45)	custo_sinistro numeric	qtdsinistros bigint
1	civic	honda	2224359651.25	53367
2	passat	volkswagen	1321770942.80	26706
3	fit	honda	895606548.22	86290
4	city	honda	756021853.51	80953
5	toro	fiat	635293601.97	25653
6	uno	fiat	506888074.33	61245
7	gol	volkswagen	501609493.42	63298
8	hilux	toyota	469187292.35	9342
9	camaro	chevrolet	451777553.75	6314
10	up	volkswagen	446942467.23	59001
11	mobi	fiat	430622682.73	56859
12	nova	volkswagen	408575888.38	39062

Figura 15 – Distribuição de sinistros por modelo [Listing 5.8](#)

Dessa forma, o executivo pode tomar decisões acerca do preço determinado para o seguro de acordo com a quantidade de sinistros ocorridos para determinados modelos, bem como o custo desses sinistros para a seguradora, visando maximizar seu lucro.

Existe ainda o cubo de vendas de veículos, que será melhor detalhado na [seção 5.4](#), para exemplificação de drill-down e roll-up no mesmo.

5.3.4 Relação entre ano de nascimento e quantidade de sinistros

A pergunta seguinte foi respondida utilizando-se o software Weka: "Qual o perfil das pessoas que possuem mais sinistros vinculados: os mais jovens ou os mais velhos?"

Listing 5.9 – Relação entre ano de nascimento (ou idade) e quantidade de sinistros

```
SELECT COUNT(*) AS qtd_sinistros ,
       EXTRACT(YEAR FROM data_nasc) AS ano_nasc
FROM cliente
NATURAL JOIN sinistro
GROUP BY ano_nasc;
```

Note na [Figura 16](#) que as pessoas mais jovens possuem uma quantidade de sinistros maior vinculadas a eles.

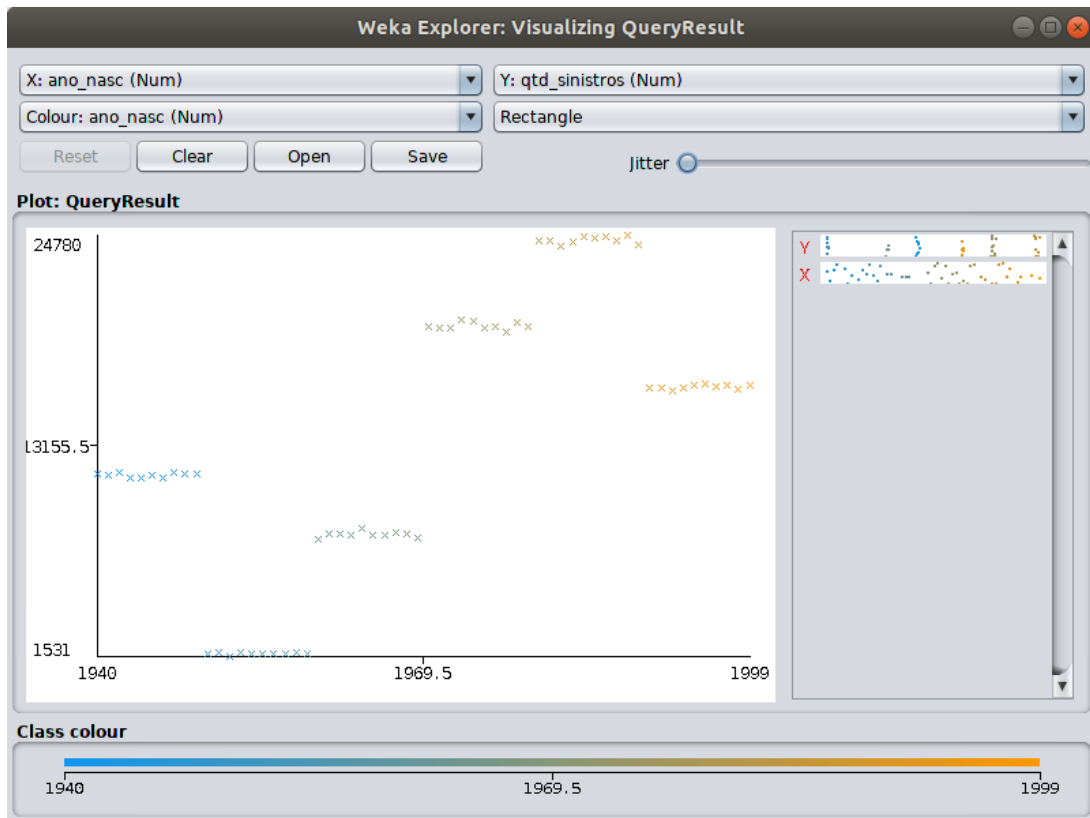


Figura 16 – Relação entre o ano de nascimento e a quantidade de sinistros associados - [Listing 5.9](#)

Ainda, utilizando o recurso de clusterização do Weka, pode-se inferir, por meio do XMeans, a partir de qual idade uma pessoa é considerada "jovem". De acordo com a [Figura 17](#), dois clusters foram identificados, onde é indicado que as pessoas que nasceram a partir de 1984 possuem mais sinistros vinculados do que pessoas que nasceram antes dessa data:

- Cluster 0: $\text{ano_nasc} \geq 1984.5$: aproximadamente 20197 sinistros
- Cluster 1: $\text{ano_nasc} < 1984.5$: aproximadamente 7126 sinistros

Dessa forma, o executivo pode tomar decisões acerca do preço determinado para o seguro de acordo com a idade das pessoas, atribuindo um preço maior para pessoas que nasceram depois de 1984 (pessoas com 35 anos de idade ou menos), por exemplo.

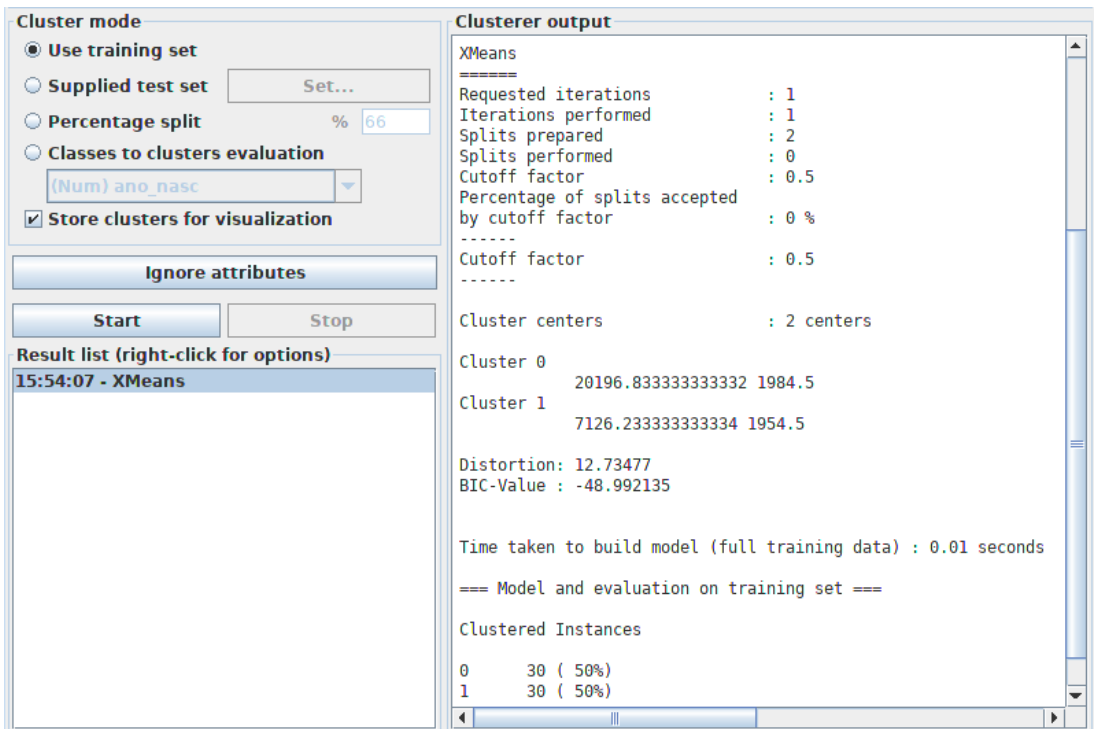


Figura 17 – Cluster - Relação entre o ano de nascimento e a quantidade de sinistros associados - Listing 5.9

5.4 Atividade 4

Nessa seção, iremos demonstrar inicialmente a análise do cubo de dados que foi utilizado para drill-down/roll-up na própria ferramenta SGBD, e posteriormente demonstrar alguns relatórios advindos dessas informações.

5.4.1 O cubo de dados utilizado

Para realizarmos essa análise, montamos o seguinte cubo tridimensional conforme mostrado na Atividade 2. A Figura 18 mostra um exemplo dos dados desse cubo.

	ano double precision	mes double precision	dia double precision	total double precision
1	[null]	[null]	[null]	119564878383
2	2013	11	4	8294699
3	2015	3	26	11013170
4	2013	9	16	210036244
5	2013	8	29	23004466
6	2018	1	10	37759759
7	2014	6	10	23372919
8	2016	6	21	35847615
9	2018	12	20	52002477

Figura 18 – Venda de veículos por período

Como podemos observar, essas informações podem ser bastante relevantes, caso a companhia de seguro deseje por exemplo direcionar o seu marketing para pontos de maior compra de veículos, já que naturalmente as pessoas procuram um seguro a partir do momento em que adquirem um veículo.

5.4.2 Drill-down sobre os dados

Dessa forma, a partir da ferramenta Metabase, montamos gráficos que exemplificam melhor e podem mostrar com mais detalhes o drill-down realizado. A Figura 19 mostra um dashboard com o resumo das informações que podemos extrair por períodos específicos. Vamos falar de cada uma adiante.

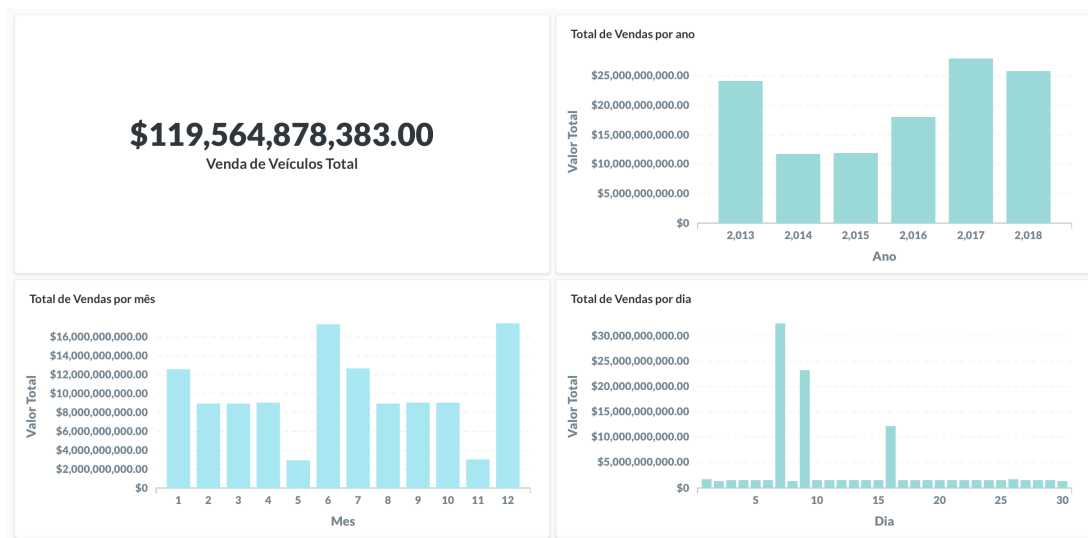


Figura 19 – Dashboard de vendas por período

Como podemos observar, ao fazer uma análise primária sobre o cubo, podemos fazer uma análise por ano, e traçarmos alguma tendência sobre a quantidade de veículos que supostamente serão comprados nos próximos anos. A Figura 20 mostra esses dados, e podemos observar que a compra de veículos novos foi aumentando ao longo dos anos de 2014 pra cá.

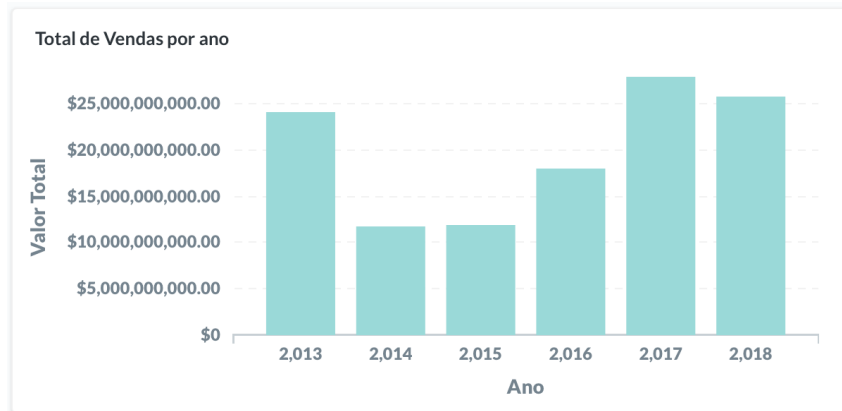


Figura 20 – Dashboard de vendas por ano

Apesar de ser muito interessante, essa informação por si só não é capaz de responder a equipe de marketing por exemplo, logo, podemos fazer um drill-down nos dados e fazer então uma análise mensal sobre a compra de veículos. A Figura 21 mostra a distribuição dos dados agora por meses.

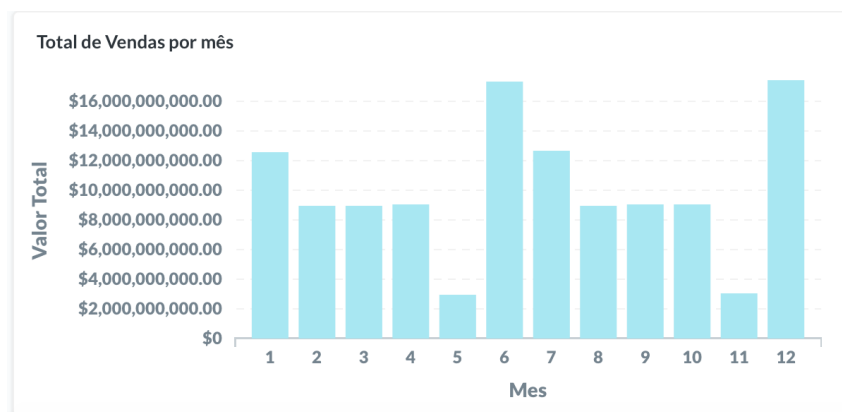


Figura 21 – Dashboard de vendas por mês

Conforme podemos observar, essa já é uma informação um pouco mais útil, já que se pode visualizar que normalmente, as compras se intensificam no início e final do ano, o que é justificável. Logo ela pode aumentar o seu gasto com marketing nos meses 6 e 12 por exemplo.

Ainda não sendo o suficiente, mais um drill-down pode ser aplicado, para analisar a configuração agora por dia do mês. A Figura 22 exemplifica melhor essa divisão e nos dá informações mais precisas.

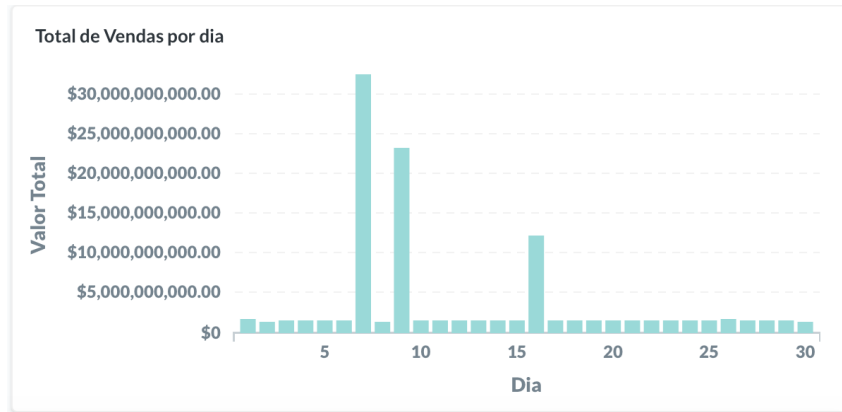


Figura 22 – Dashboard de vendas por dia

Com essa informação, conseguimos inclusive notar os dias nos quais a equipe de marketing deve investir mais pesado em suas campanhas. Notamos que a compra de veículos é muito mais intensa nos dias 7, 9 e 16 do mês.

5.5 Atividade 5

Para realização dessa atividade, utilizamos o software Metabase para geração de relatórios úteis para a tomada de decisão da empresa, inclusive para visualização gráfica das respostas obtidas pelas perguntas da Atividade 3.

Fizemos a divisão em três grandes grupos (Dashboards). Um para análise dos veículos da seguradora, outro para análise de sinistros ocorridos, e o já exemplificado na Atividade 4, para análise de compras de veículos.

Portanto nessa seção, iremos dar mais detalhes sobre os dois outros conjuntos de relatórios (Dashboards) restantes.

5.5.1 Dashboard Veículos

O intuito desse dashboard, é fornecer uma análise completa sobre o preço médio de compra dos veículos na qual a seguradora já trabalhou.

A Figura 23 exemplifica as informações que constam nesse dashboard.

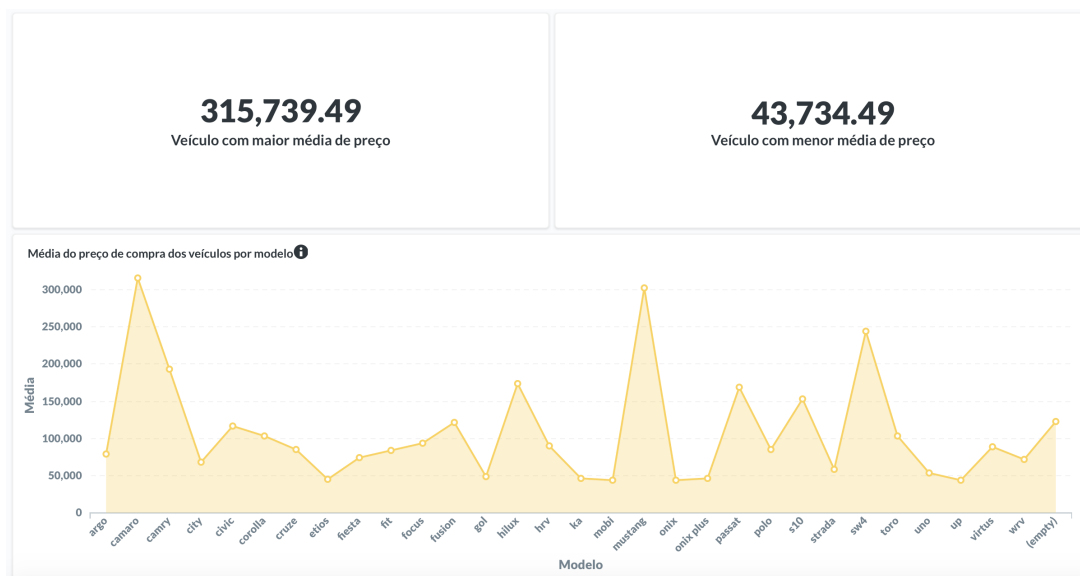


Figura 23 – Dashboard de preço médio de compra de veículos

Podemos observar então o maior e menor valor médio por modelo, bem como um gráfico contendo o valor médio de compra de cada modelo que já passou pela seguradora. A coluna (empty) mostra a média de todos os veículos, independente do modelo.

Essa informação pode ser muito interessante inclusive para definir o preço do seguro que será fornecido para cada veículo, visto que esse valor varia de acordo com o modelo do mesmo.

5.5.2 Dashboard Sinistros

Já o dashboard de sinistros, possui o intuito de dar uma visão bastante completa e interessante sobre a distribuição de sinistros, tanto por região, quanto por gênero, quanto por modelo de veículos, dando mais poder a seguradora de decidir o valor de seus seguros.

A Figura 24 mostra a quantidade de sinistros ocorridos bem como o custo total com sinistros que a seguradora teve durante todo seu tempo de existência.



Figura 24 – Quantidade de sinistros e valor gasto pela seguradora

Já a Figura 25 mostra a quantidade de sinistros ocorridos de acordo com o modelo do veículo. Note que existe um coluna (empty) que possui a soma de todos os sinistros, que bate exatamente com a quantidade exibida anteriormente pela Figura 24.

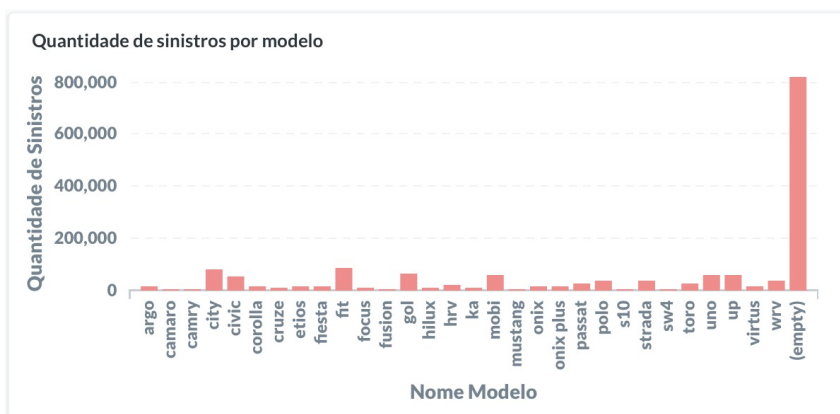


Figura 25 – Quantidade de sinistros por modelo

Essa informação pode ser muito útil na definição do custo do carro, em conjunto com o gráfico da Figura 26, que complementa a informação de custo para a seguradora dos sinistros referentes a um dado modelo. Da mesma forma, a coluna (empty) possui o total gasto em sinistros por todos os modelos.

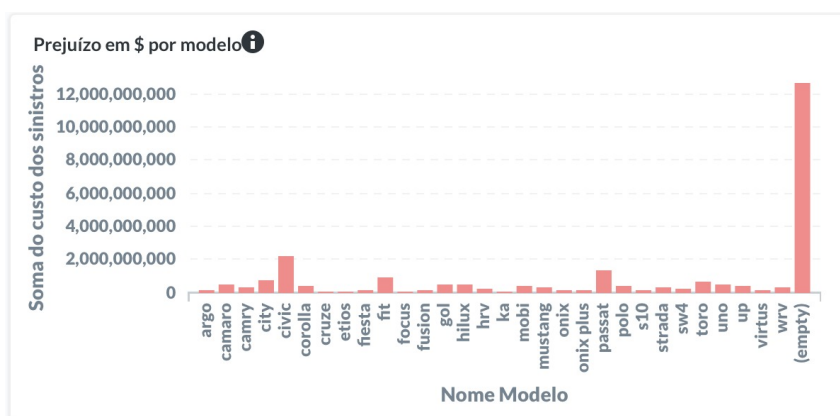


Figura 26 – Total gasto em sinistros por modelo

Além dessas informações por modelo, oferecemos ainda um gráfico de distribuição de sinistros por região do país. A Figura 27 mostra como fica essa distribuição, mostrando quantos sinistros ocorreram em cada estado. Isso pode influenciar também para decisão do valor de um dado seguro, inclusive pode ser útil para campanhas de conscientização, dentre outros.

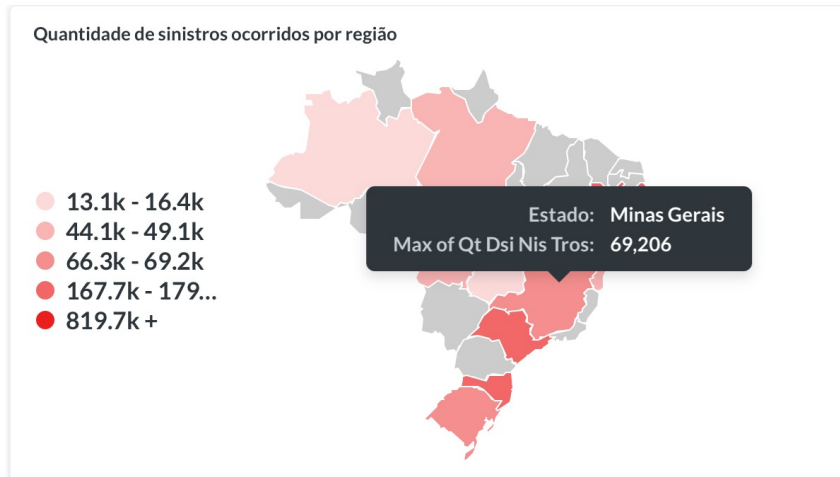


Figura 27 – Distribuição de sinistros por região

Por fim, temos um gráfico de distribuição de sinistros por sexo do cliente, e essa é uma informação bastante interessante, que mostra um paralelo enorme entre a quantidade de sinistros com clientes do sexo masculino ser bem maior que do sexo feminino. A Figura 28 mostra isso com mais detalhes.

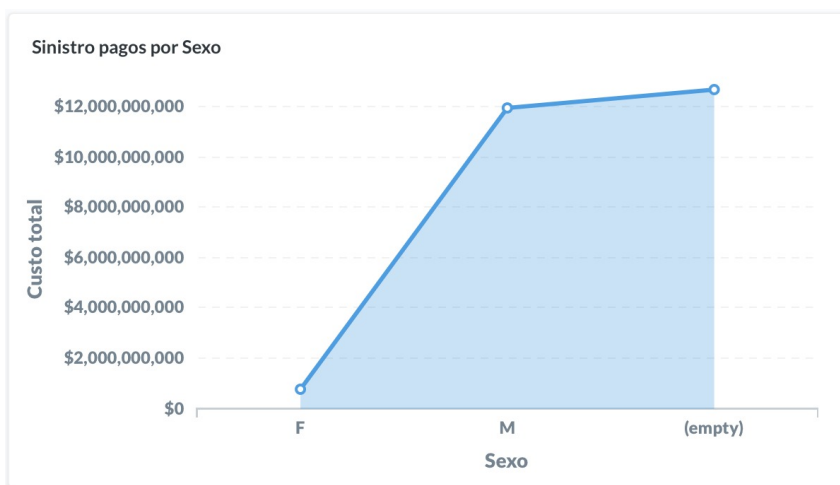


Figura 28 – Distribuição de sinistros por sexo

Todas essas informações podem ser de bastante interesse da seguradora, para definição de quanto será o valor de um seguro para um cliente, para realizar campanhas de prevenção, maximizando o lucro da seguradora, dentre outras decisões que podem ser tomadas a partir da análise desses dados.

5.6 Atividade 6

Nesta atividade, exigiu-se que pelo menos um dos conceitos de *Data Mining* fosse aplicado no conjunto de dados. Assim, o conceito abordado foi o de previsão. Para esta

abordagem, a ferramenta Weka foi utilizada para auxiliar na criação e teste do modelo preditivo.

Basicamente, a previsão funciona da seguinte forma: dado algumas características, qual é a probabilidade de determinado fator acontecer. No contexto do trabalho, as características relevantes escolhidas foram: *estado*, *cidade*, *idade*, *nome_modelo* e *ano_carro*. Tais atributos são retirados de duas tabelas do banco, *cliente* e *veiculo*. É importante ressaltar que *idade* e *ano_carro* são derivados de outros campos das tabelas, *data_nasc* e *data_compra*, respectivamente. Com essas informações, a previsão feita é se um dado cliente irá ou não sofrer sinistro. Além disso, para agilizar e facilitar o manuseio dos dados, todas as informações foram retiradas de clientes que sofreram (ou não) sinistros no ano de 2018.

Para ajudar na recuperação dos dados, uma visão que contém o *cliente_id* dos clientes que sofreram sinistro em 2018. Em [Listing 5.10](#), é possível observar a consulta criada.

Listing 5.10 – Criação da tabela com sinistros em 2018

```
CREATE VIEW visao AS  
SELECT cliente_id  
FROM cliente NATURAL JOIN sinistro NATURAL JOIN veiculo  
WHERE ano_id = 6;
```

Com a visão criada, uma nova consulta recuperando os dados com e sem sinistro foi feita. Neste ponto, é necessário realizar uma observação: como é um modelo de previsão, um atributo dos dados precisa ser presumido. Então, criou-se uma nova coluna nessa consulta, a *temSinistro*. Assim, se o cliente sofreu um sinistro em 2018, o valor será *true* e, caso contrário, *false*. A consulta se encontra em [Listing 5.11](#).

Listing 5.11 – Recuperação dos dados dos clientes e seus veículos com e sem sinistros em 2018

```
SELECT cl.cliente_id, estado, cidade, sexo,  
EXTRACT(YEAR FROM AGE(data_nasc)) AS idade, nome_modelo,  
EXTRACT(YEAR FROM data_compra) AS ano_carro,  
(CASE WHEN cl.cliente_id = visao.cliente_id  
THEN 'true' ELSE 'false' END) AS temSinistro  
FROM (cliente cl LEFT JOIN sinistro si  
ON cl.cliente_id = si.cliente_id) LEFT JOIN veiculo ve  
ON ve.veiculo_id = si.veiculo_id  
LEFT JOIN visao ON cl.cliente_id = visao.cliente_id;
```

Com os dados recuperados, dois modelos foram treinados. O primeiro utilizou classificador de NaiveBayes, enquanto o segundo foi classificado pelo J48. No primeiro, a classificação é feita da seguinte forma: um agente supervisor (no caso, a consulta SQL em [Listing 5.11](#)) classifica os dados previamente, no caso, em clientes que sofreram ou não sinistro em 2018. Dessa forma, a classificação de novos dados ocorre com base na probabilidade dele sofrer ou não um sinistro. Tal probabilidade é calculada com base em cada atributo do modelo, sendo que cada um deles possui um peso. A [Figura 29](#) ilustra o modelo gerado. Como é possível observar, sua acurácia é de aproximadamente 60%.

```

=== Summary ===

Correctly Classified Instances      524849          60.3339 %
Incorrectly Classified Instances    345058          39.6661 %
Kappa statistic                    0.2159
Mean absolute error                 0.45
Root mean squared error             0.479
Relative absolute error             90.2392 %
Root relative squared error         95.9294 %
Total Number of Instances          869907

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,488    0,269    0,667      0,488    0,564      0,225    0,645    0,709    true
                0,731    0,512    0,563      0,731    0,636      0,225    0,645    0,573    false
Weighted Avg.   0,603    0,384    0,618      0,603    0,598      0,225    0,645    0,644

=== Confusion Matrix ===

      a      b  <-- classified as
223237 233812 |      a = true
111246 301612 |      b = false

```

Figura 29 – Modelo gerado com NaiveBayes.

O segundo classificador cria uma árvore de decisão. No contexto deste trabalho, os nós folhas contém a resposta para a previsão desejada (se tem ou não sinistro em 2018). Os nós internos contém as perguntas, como por exemplo "qual o modelo do carro?", e os ramos são as respostas, "onix". Assim, o algoritmo realiza o caminho por essa árvore até que obtenha-se uma resposta. A [Figura 30](#) mostra o modelo gerado. É possível observar que ele possui um resultado ligeiramente melhor do que o primeiro, com 61.8% de acurácia.

```

=== Summary ===

Correctly Classified Instances      134480          61.8364 %
Incorrectly Classified Instances    82997           38.1636 %
Kappa statistic                    0.2388
Mean absolute error                 0.4312
Root mean squared error             0.4784
Relative absolute error             86.4555 %
Root relative squared error         95.8012 %
Total Number of Instances          217477

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,586   0,346   0,652     0,586   0,617     0,240   0,667   0,703   true
                0,654   0,414   0,589     0,654   0,620     0,240   0,667   0,612   false
Weighted Avg.   0,618   0,378   0,622     0,618   0,618     0,240   0,667   0,659

=== Confusion Matrix ===

      a      b  <-- classified as
66839 47250 |      a = true
35747 67641 |      b = false

```

Figura 30 – Modelo gerado com JF8.

Com o modelo treinado e validado, basta prever se um determinado cliente sofrerá sinistro ou não. Dessa forma, três clientes foram criados (ordem dos atributos é *cliente_id*, *estado*, *cidade*, *idade*, *nome_modelo*, *ano_carro* e *temSinistro* respectivamente):

- 1, MG, 'belo horizonte', 21, ka, 2017, ?
- 2, MG, 'belo horizonte', 21, gol, 2014, ?
- 3, SP, campinas, 38, hrv, 2011, ?

O resultado obtido pelo NaiveBayers pode ser visto na [Figura 31](#). Observe que, segundo o algoritmo, apenas o primeiro cliente sofrerá sinistro.

```

=== Predictions on user test set ===

```

inst#	actual	predicted	error	prediction
1	1:?	1:true	0.769	
2	1:?	2:false	0.581	
3	1:?	2:false	0.534	

Figura 31 – Resultado do NaiveBayers.

O resultado obtido pelo J48 pode ser visto na [Figura 32](#). Observe que, segundo o algoritmo, apenas o terceiro cliente não sofrerá sinistro.

```
=== Predictions on user test set ===
```

inst#	actual	predicted	error	prediction
1	1:?	1:true		0.667
2	1:?	1:true		0.694
3	1:?	2:false		0.535

Figura 32 – Resultado do JF8.

5.7 Atividade 7

Como segundo frontend, o grupo desenvolveu uma aplicação utilizando **Jupyter Notebooks**, em Python 3, utilizando-se principalmente das bibliotecas: **Pandas**, **Scikit-Learn**, **Matplotlib** e **Seaborn**. Essa pequena aplicação tinha como objetivo fazer uma análise exploratória simples dos dados, mas focada em correlação.

Além disso, esse frontend também tinha como objetivo a previsão, utilizando os conceitos estudados de data mining, se um novo cliente terá ou não um sinistro no ano seguinte, pois essa informação é de vital importância para a avaliação do prêmio, e dos valores do seguro, assim como na captação e conversão de clientes.

5.7.1 Conexão com o Banco

A conexão com o banco de dados foi feita diretamente pelo código, com o auxílio de uma biblioteca específica pra comunicação com o Postgres (*psycopg2*).

5.7.2 Obtendo os Dados

Uma vez que estamos utilizando o Pandas, que é excelente para manipulação de grandes volumes de dados, executamos somente uma query no banco de dados onde obtemos **TODOS** os dados que precisamos. Estes dados são armazenados em um *dataframe* que posteriormente é agrupado, clonado, filtrado, e manipulado de diversas maneiras para explorar os dados.

5.7.3 Análise Feita

Toda a análise feita é auto-documentada e contida no próprio notebook. Ele está disponível anexado junto com este documento.

5.8 Atividade 8

Fazendo a comparação dos resultados de predibilidade dos dois frontends (Weka e autoral), o grupo conseguiu extrair algumas conclusões:

O sistema preditivo autoral possui maior liberdade e flexibilidade, principalmente em relação a quais features utilizar, as transformações que podem ser feitas e as integrações com qualquer software que desejar, algo que não é visto em quase nenhum software de mercado.

Outro ponto positivo do sistema autoral são os resultados dos algoritmos preditivos que se mostraram em média 10% mais precisos (Figura 33) que os do Weka, além da facilidade e capacidade de analisar todas as colunas presentes no datawarehouse, tornando possível por exemplo, a predição na hora de fazer a cotação de um cliente pro seguro.

[[86129 4891]					
[19603 15351]]					
		precision	recall	f1-score	support
0	0.81	0.95	0.88	91020	
1	0.76	0.44	0.56	34954	
accuracy			0.81	125974	
macro avg	0.79	0.69	0.72	125974	
weighted avg	0.80	0.81	0.79	125974	

Figura 33 – Matriz de confusão e estatísticas de rede neural extraída do notebook

O ponto negativo do sistema autoral é a necessidade de conhecimento de data mining, data science e suas diversas áreas, saber como e onde aplicar as técnicas e modelos estatísticos, que não é algo trivial. As empresas muito provavelmente não possui um cientista da computação com conhecimento nesta área para poder montar um sistema tão completo e cognitivo como o Weka, que basta ir clicando em botões para montar um modelo preditivo.

6 Conclusão

Neste trabalho foi possível aprender como se dá o planejamento para a modelagem de um banco de dados warehouse e o que deve ser pensado e levado em consideração para o mesmo, bem como realizar a modelagem de cubos de dados interessantes para retirada de informações pertinentes ao contexto de uma empresa.

Além disso, este trabalho ofereceu a oportunidade de se trabalhar com uma ferramenta de BI, na qual foi possível gerar Dashboards e gráficos bastante interessantes para auxiliar os executivos em suas tomadas de decisão.

Ainda, técnicas de Machine Learning foram aplicadas, o que permitiu fazer inferências sobre os dados armazenados no warehouse da empresa, de duas formas diferentes, usando o software Weka e um outro desenvolvido pelo grupo.

Esse trabalho então mostra como essas técnicas podem auxiliar empresários e gestores nas tomadas de decisão, independentemente do contexto em que a empresa está inserida, facilitando assim a vida do executivo, que pode então tomar decisões cada vez mais assertivas sobre seus clientes, funcionários, marketing, dentre outros contextos de sua empresa.

Referências

AMAZON WEB SERVICES. *Amazon Redshift and PostgreSQL*. 2019. Disponível em: <https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-and-postgres-sql.html>. Acesso em: 07/11/2019. Citado na página 8.

DUNN, R. *Data Warehousing on PostgreSQL - FOSSASIA Summit*. 2016. Disponível em: <<https://www.youtube.com/watch?v=AuuLqKPldxs>>. Acesso em: 07/11/2019. Citado 2 vezes nas páginas 2 e 8.

METABASE. *Metabase*. 2019. Disponível em: <<https://www.metabase.com/>>. Acesso em: 23/11/2019. Citado na página 17.

POSTGRESQL. *Cube*. 1996. Disponível em: <<https://www.postgresql.org/docs/9.5/cube.html>>. Acesso em: 23/11/2019. Citado na página 17.

POSTGRESQL TUTORIAL. *PostgreSQL Cube*. 2019. Disponível em: <<http://www-postgresqltutorial.com/postgresql-cube/>>. Acesso em: 23/11/2019. Citado na página 17.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: Documentation: 9.6: Populating a Database*. 2019. Disponível em: <<https://www.postgresql.org/docs/9.6/populate.html>>. Acesso em: 07/11/2019. Citado na página 10.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: Documentation: COPY*. 2019. Disponível em: <<https://www.postgresql.org/docs/current/sql-copy.html>>. Acesso em: 07/11/2019. Citado na página 9.

WEKA. *Machine Learning at Waikato University*. 2019. Disponível em: <<https://www.cs.waikato.ac.nz/ml/index.html>>. Acesso em: 24/11/2019. Citado na página 24.