



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Luiz Henrique Castro Costa Britto

**Ferramenta de Análise do Desempenho de uma Solução de Monitoramento de
Processo Agrícola Sucoalcooleiro**

Florianópolis
2020

Luiz Henrique Castro Costa Britto

**Ferramenta de Análise do Desempenho de uma Solução de Monitoramento de
Processo Agrícola Sucroalcooleiro**

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.

Orientador: Prof. Ricardo José Rabelo, Dr.
Supervisor: Hugo Pereira Fagundes, Eng.

Florianópolis
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Britto, Luiz Henrique Castro Costa
Ferramenta de Análise do Desempenho de uma Solução de
Monitoramento de Processo Agrícola Sucroalcooleiro / Luiz
Henrique Castro Costa Britto ; orientador, Ricardo José
Rabelo, coorientador, Hugo Pereira Fagundes, 2020.
93 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia de Controle e Automação,
Florianópolis, 2020.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Monitoramento
de desempenho. 3. Suporte. 4. Agricultura. 5.
Monitoramento de máquinas. I. Rabelo, Ricardo José. II.
Fagundes, Hugo Pereira. III. Universidade Federal de Santa
Catarina. Graduação em Engenharia de Controle e Automação.
IV. Título.

Luiz Henrique Castro Costa Britto

**Ferramenta de Análise do Desempenho de uma Solução de Monitoramento de
Processo Agrícola Sucoalcooleiro**

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de
Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de
Controle e Automação

Florianópolis, 20 de Outubro de 2020.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Ricardo José Rabelo, Dr.
Orientador
UFSC/CTC/DAS

Hugo Pereira Fagundes, Eng.
Supervisor
Hexagon

Marcelo Menezes Morato, Eng.
Avaliador
UFSC/CTC/DAS

Prof. Fabio Baldissera, Dr.
Presidente da Banca
UFSC/CTC/DAS

RESUMO

A Hexagon é uma empresa global, líder em sensoriamento, software e soluções autônomas. A sua divisão de agricultura é uma das grandes desenvolvedoras globais de tecnologia para a Agricultura 4.0, oferecendo ferramentas para a otimização de todas as etapas agrícolas, desde o planejamento do cultivo até o rastreamento do material colhido. Com o intuito de atender as mais novas demandas do agronegócio, no ano de 2019, a Hexagon finalizou o desenvolvimento de sua mais recente solução de monitoramento, Rastreabilidade de Matéria Prima (RMT), um sistema que executa o rastreamento automático da matéria prima desde a colheita no campo, até a chegada na indústria. Após a sua implantação para um primeiro cliente, detectou-se uma dificuldade no suporte e manutenção da solução, devido à dispersão e à dificuldade de acesso aos dados utilizados na identificação de problemas em campo e de desempenho da solução. Visando sanar essa dificuldade, o foco deste projeto foi o desenvolvimento de uma ferramenta cujo objetivo é auxiliar no diagnóstico de problemas em campo e acompanhamento do desempenho da solução de monitoramento RMT. A ferramenta proposta é dividida em duas grandes partes, um *Data Warehouse* que é populado de forma automática com dados de múltiplas fontes, juntamente com um conjunto de *Dashboards*, para a visualização e análise dos dados; e um código de análise correlacional de dados, que visa realizar um estudo exploratório dos dados. Dentre os conceitos e tecnologias utilizadas no desenvolvimento do projeto, destaca-se a linguagem de programação Python, Microsoft Power BI, serviços da plataforma de computação em nuvem Amazon Web Services - Amazon RDS, AWS Lambda, AWS IoT Core - técnicas de modelagem dimensional de banco de dado e conceitos estatísticos de correlação de dados. Por fim, o documento apresenta os resultados obtidos com a utilização da ferramenta desenvolvida, relatando detalhadamente os passos da execução de seu projeto.

Palavras-chave: *Data Warehouse*. *Dashboards*. Monitoramento. Agricultura 4.0. Suporte. Dados.

ABSTRACT

Hexagon is a global company, leader in sensing, software and autonomous solutions. Its agriculture division is one of the major global developers of technology for Agriculture 4.0, offering tools for the optimization of all agricultural stages, from crop planning to tracking harvested material. In order to meet the newest demands of agribusiness, in 2019, Hexagon completed the development of its most recent monitoring solution, Raw Material Traceability (RMT), a system that performs the automatic tracking of raw material since its harvest, until it reaches the industry. After its implantation for a first customer, a difficulty in the support and maintenance of the solution was detected, due to the dispersion and the difficulty of accessing the data used to identify problems in the field and solution performance. In order to remedy this difficulty, the focus of this project was the development of a tool whose objective is to assist in the diagnosis of problems in the field and monitoring the performance of the RMT monitoring solution. The proposed tool is divided into two large parts, a textit Data Warehouse that is automatically populated with data from multiple sources, together with a set of textit Dashboards, for the visualization and analysis of the data; and a correlational data analysis code, which aims to conduct an exploratory study of the data. Among the concepts and technologies used in the development of the project, we highlight the Python programming language, Microsoft Power BI, services of the cloud-based platform Amazon Web Services - Amazon RDS, AWS Lambda, AWS IoT Core - dimensional modeling techniques of database and statistical concepts of data correlation. Finally, the document presents the results obtained with the use of the developed tool, reporting in detail the steps of the execution of the project.

Keywords: Data Warehouse. Dashboards. Monitoring. Agriculture. Support. Data.

LISTA DE FIGURAS

Figura 1 – Esquema Estrela.	25
Figura 2 – Computador de Bordo Hexagon - Ti7.	29
Figura 3 – Caminhão realizando o descarregamento da <i>Bin</i> na usina de processamento.	30
Figura 4 – Fluxo da Solução Hexagon - Rastreabilidade de Matéria-Prima (RMT).	32
Figura 5 – Operação de colheita de cana de açúcar.	33
Figura 6 – Caminhão realizando a pesagem na balança.	34
Figura 7 – Caminhão chegando na usina transportando um <i>bin</i>	34
Figura 8 – Diagrama UML de depuração representando o sistema.	37
Figura 9 – Diagrama UML de atividade do primeiro ETL.	38
Figura 10 – Diagrama UML de atividade do segundo ETL.	39
Figura 11 – Diagrama UML de atividade do código de análise de dados.	40
Figura 12 – Esquemático da arquitetura do fluxo de dados do <i>data warehouse</i>	45
Figura 13 – Fluxo de dados do primeiro ETL.	45
Figura 14 – Estrutura de dados <i>Shipment JSON</i>	46
Figura 15 – Estrutura de dados <i>History JSON</i>	47
Figura 16 – Diagrama relacional do banco de dados.	48
Figura 17 – Função <i>lambda_handler</i>	49
Figura 18 – Construtor da classe <i>lotJsonParser</i> e o método <i>db_connect</i>	49
Figura 19 – Método <i>define</i>	50
Figura 20 – Método <i>parse_json</i>	51
Figura 21 – Arquivo JSON modelo.	51
Figura 22 – Método <i>create_shipment</i>	52
Figura 23 – Método <i>verify_day_history_json</i>	53
Figura 24 – Método <i>update_history_json</i>	53
Figura 25 – Método <i>get_vehicle_sk</i>	54
Figura 26 – Fluxo de dados do segundo ETL.	54
Figura 27 – Diagrama relacional do banco de dados completo.	57
Figura 28 – Arquivo JSON com as consultas SQL referentes à dimensão alarme.	58
Figura 29 – Função <i>lambda_handler</i>	58
Figura 30 – Método construtor da classe <i>DimensionUpdater</i>	59
Figura 31 – Método <i>update_dim</i>	60
Figura 32 – Método <i>get_dim_last_update</i>	61
Figura 33 – Método <i>update_onboard_computer</i>	61
Figura 34 – Método <i>fact_update</i>	62
Figura 35 – Método <i>insert_records</i>	63
Figura 36 – Dashboard dos alarmes das colhedoras.	64

Figura 37 – Dashboard dos problemas indicados pela balança.	65
Figura 38 – Dashboard das informações de interesse.	67
Figura 39 – Dashboard informações das tags dos bins.	67
Figura 40 – Relatório enviado diariamente ao cliente.	69
Figura 41 – Método <i>execute_analytics</i>	72
Figura 42 – Gráfico de desempenho RMT pela quantidade de carregamentos com código de <i>bin</i> inválido.	73
Figura 43 – Tabela <i>shipments - tags</i> corrompidas.	73
Figura 44 – Gráfico de desempenho RMT pela quantidade de carregamentos ditos fantasmas.	74
Figura 45 – Tabela <i>shipments - carregamentos fantasmas bin 117</i>	75
Figura 46 – Tabela <i>shipments - carregamentos fantasmas bin 139</i>	75
Figura 47 – Gráfico de desempenho da solução RMT pela contagem de alarmes de problemas de paremaneto Wi-Fi.	76
Figura 48 – Colhedoras com maior número de alarmes de problemas de pareamento Wi-Fi.	76
Figura 49 – Método <i>count_correlation</i>	85
Figura 50 – Gráfico do valor dos diferentes coeficientes de correlação entre cada contagem de alarme e os dados da porcentagem de carregamentos com coordenadas de origem.	86
Figura 51 – Gráfico do valor dos diferentes coeficientes de correlação entre cada contagem de alarme e os dados da porcentagem de carregamentos com ID de origem.	87
Figura 52 – Tabela dos valores de coeficiente de Pearson entre todos os dados referente a contagem dos alarmes.	88
Figura 53 – Tabela dos valores P entre todos os dados referente a contagem dos alarmes.	88
Figura 54 – Gráfico dos três alarmes nativos que tiveram o maior coeficiente de correlação de Pearson em relação a porcentagem de carregamentos com coordenadas de origem.	89
Figura 55 – Gráfico no tempo dos três alarmes configuráveis que tiveram o maior coeficiente de correlação de Pearson em relação a porcentagem de carregamentos com coordenadas de origem.	90
Figura 56 – Gráfico do valor dos diferentes coeficientes de correlação entre os dados de problemas relacionados a balança e os dados da porcentagem de carregamentos com coordenadas de origem.	91
Figura 57 – Gráfico do valor dos diferentes coeficientes de correlação entre os dados de problemas relacionados à balança e os dados da porcentagem de carregamentos com ID de origem.	92

Figura 58 – Tabela dos valores de coeficiente de Pearson entre todos os dados referentes aos problemas relacionados à balança.	93
Figura 59 – Tabela dos valores P entre todos os dados referentes aos problemas relacionados à balança.	93
Figura 60 – Gráfico no tempo dos dados referentes aos problemas relacionados à balança.	94
Figura 61 – Lista dos dez dados com maiores correlações com a porcentagem de carregamentos com coordenadas de origem e menores valor P. .	95

LISTA DE TABELAS

Tabela 1 – Possíveis origens de dados.	42
Tabela 2 – Informações de interesse.	43
Tabela 3 – Dados utilizados pelo código de análise correlacional.	71

SUMÁRIO

1	INTRODUÇÃO	13
1.1	O PROBLEMA	13
1.2	OBJETIVOS	14
1.2.1	Objetivo Geral	14
1.2.2	Objetivos Específicos	15
1.3	METODOLOGIA	15
1.4	ESTRUTURA DO DOCUMENTO	16
2	EMPRESA	17
2.1	HEXAGON AGRICULTURE	17
2.2	TIME DE IMPLANTAÇÃO E SUPORTE	18
3	FUNDAMENTAÇÃO TEÓRICA	20
3.1	CONCEITOS ESTATÍSTICOS	20
3.1.1	Coeficiente de correlação de Pearson	20
3.1.2	Coeficiente de correlação de Kendall	20
3.1.3	Coeficiente de correlação de Spearman	21
3.1.4	Valor-P	21
3.2	CONCEITOS DE MODELAGEM DE SISTEMAS	22
3.2.1	Unified Modeling Language (UML)	22
3.3	CONCEITOS E FERRAMENTAS DE BUSINESS INTELLIGENCE	22
3.3.1	Business Intelligence	22
3.3.2	Data Warehouse	23
3.3.3	Modelagem dimensional de banco de dados	24
3.3.4	Microsoft Power BI	25
3.4	TECNOLOGIAS DE IMPLEMENTAÇÃO	26
3.4.1	PostgreSQL	26
3.4.2	Python	26
3.4.2.1	Pandas	26
3.4.2.2	Matplotlib	27
3.4.2.3	Psycopg2	27
3.4.3	Amazon Web Services (AWS)	27
3.4.3.1	Amazon Relational Database Service (Amazon RDS)	27
3.4.3.2	AWS Lambda	28
3.4.3.3	Amazon EventBridge	28
3.4.3.4	AWS IOT Core	28
3.5	TECNOLOGÍAS E CONCEITOS HEXAGON	28
3.5.1	Computador de bordo	28
3.5.2	Bin	29

3.5.3	Tag RFID	30
3.5.4	RMT - Raw Material Tracking	30
4	ESPECIFICAÇÃO DO PROJETO	35
4.1	REQUISITOS	35
4.1.1	Data Warehouse	35
4.1.2	Código de Análise de Dados	36
4.2	MODELAGEM	36
4.2.1	Modelo Sistema	36
4.2.2	Modelo Primeiro ETL - Dados da Balança	37
4.2.3	Modelo Segundo ETL - Alarmes	37
4.2.4	Código de Análise Correlacional dos Dados	37
5	DESENVOLVIMENTO	41
5.1	MAPEAMENTO DAS FONTES DE DADOS	41
5.2	FORMULAÇÃO DA LISTA DAS INFORMAÇÕES DE INTERESSE	42
5.3	DESENVOLVIMENTO DOS ETLs	44
5.3.1	Desenvolvimento do primeiro ETL - Dados do serviço da balança da usina	45
5.3.1.1	Dados enviados do serviço da balança para o AWS IoT Core:	46
5.3.1.2	Modelagem do Banco de Dados:	47
5.3.1.3	Código ETL - Lambda	48
5.3.2	Desenvolvimento do segundo ETL - Registros dos alarmes	54
5.3.2.1	Os dados coletados	55
5.3.2.2	Modelagem do Banco de Dados	55
5.3.2.3	Código ETL - Lambda	56
5.4	VISUALIZAÇÃO	63
5.4.1	Dashboard dos alarmes das colhedoras	64
5.4.2	Dashboard dos problemas indicados pela balança	65
5.4.3	Dashboard das informações de interesse	66
5.4.4	Dashboard informações das tags dos bins	66
5.5	CÓDIGO DE ANÁLISE DE DADOS	70
5.6	VALIDAÇÃO	72
5.6.1	Caso 1 - Códigos inválidos de bins	73
5.6.2	Caso 2 - Tag RFID do caminhão corrompida	74
5.6.3	Caso 3 - Queda de performance por problemas de pareamento	76
6	CONCLUSÃO	77
6.1	O QUE FOI FEITO	77
6.2	COMPARAÇÃO DOS RESULTADOS E OBJETIVOS	77
6.3	ANÁLISE	78
6.4	LIMITAÇÕES	79

6.5	FUTUROS DESENVOLVIMENTOS	79
	REFERÊNCIAS	81
	APÊNDICE A – CÓDIGO DO MÉTODO COUNT_CORRELATION .	85
	APÊNDICE B – GRÁFICOS RESULTANTES DO CÓDIGO DE ANÁLISE CORRELACIONAL DE DADOS	86

1 INTRODUÇÃO

Com a crescente demanda de alimentos e energias renováveis, devido ao crescimento populacional acelerado, atrelado ao desenvolvimento de novas tecnologias e o barateamento delas, um novo conceito de agricultura tem sido empregado no campo nos últimos anos, a agricultura 4.0.

A agricultura 4.0 engloba as mais modernas tecnologias, tais como métodos computacionais de alto desempenho, rede de sensores, comunicação máquina para máquina (M2M), conectividade entre dispositivos móveis, computação em nuvem, métodos e soluções analíticas para processar grandes volumes de dados. Essas tecnologias são empregadas no campo com o intuito de contribuir para elevar os índices de produtividade, a eficiência do uso de insumos, a redução de custos com mão de obra, melhorar a qualidade do trabalho e a segurança dos trabalhadores e diminuir os impactos ao meio ambiente (MASSRUHÁ S. M. F. S.; LEITE, 2017).

A divisão de Agricultura da Hexagon é um dos líderes globais desse nicho de mercado. Ela desenvolve e fornece soluções que convertem as operações em campo em dados, em outras palavras, digitalizam as operações do campo. Assim, permitindo o planejamento inteligente, execução eficiente no campo, controle preciso de máquinas e fluxos de trabalho automatizados para otimizar as operações. Gerando assim, grandes ganhos de eficiência, produtividade e sustentabilidade para seus clientes (HEXAGON, 2020e).

Seu crescente portfólio de produtos e soluções abrangem toda a cadeia produtiva da indústria agrícola, indo desde o planejamento e administração da plantação até a automação de máquinas para colheita e monitoramento do transporte do produto colhido.

1.1 O PROBLEMA

Com o seu espírito inovador, sempre tentando desenvolver produtos que atendam às novas necessidades do agronegócio, em 2019 a divisão de agricultura da Hexagon finalizou o desenvolvimento de sua mais recente solução de monitoramento, o Rastreabilidade de Matéria Prima (RMT), um sistema que executa o rastreamento automático da matéria prima desde a sua colheita em campo, até a sua entrega à indústria.

No ano passado, foi realizada a primeira implantação do RMT em um cliente, e com isso foi necessário o desprendimento de um grande esforço do time de suporte na etapa de pós-implantação, devido ao recente desenvolvimento e por ser sua primeira implantação, a solução apresentou uma grande instabilidade de desempenho.

Com a alta frequência de atendimentos de suporte, foi identificada uma oportunidade de melhoria no processo de coleta de dados para diagnóstico da solução e

apoio ao cliente, pois eram realizadas de forma manual tornando as coletas extremamente trabalhosas, demoradas e propensas a erros. As principais formas de aquisição de dados para o apoio ao suporte eram os acessos aos servidores remotos para a obtenção de extensos *logs* de sistemas de processamento de dados ou a realização de complexas consultas em bancos de dados.

Dessa forma, o suporte da solução se dava de maneira demorada e reativa, visto que as análises eram realizadas conforme a detecção da queda de desempenho por parte do cliente, que realiza a compilação das informações uma vez por semana fazendo com que muitas vezes levasse dias para a detecção.

A intenção de alinhar a política de qualidade da empresa e foco no cliente à suas ações motivou o desenvolvimento deste projeto, cujo intuito é o desenvolvimento de uma ferramenta capaz de realizar o monitoramento da solução RMT e apoio ao suporte da mesma.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral o desenvolvimento de uma ferramenta que auxilie no diagnóstico de problemas e no acompanhamento do desempenho da solução de monitoramento RMT.

Com o uso dessa ferramenta, o time de suporte deverá ser capaz de realizar o monitoramento do desempenho da solução de forma diária; identificar possíveis problemas de software, hardware ou operacionais em campo; quantificar a relevância desses problemas; e apontar ou no mínimo restringir as possíveis causas dos problemas.

O trabalho foi dividido em duas etapas principais: a primeira consiste em implementar um fluxo de dados automático, que coleta dados de uma série de fontes e os armazena de forma centralizada em um *data warehouse* e a criação de um *dashboard* para a visualização dos mesmos; dados estes, referentes ao desempenho da solução ou que tenham a capacidade de trazer informações úteis para o diagnóstico de problemas.

A centralização das informações de forma automática tem como intuito a substituição dos procedimentos manuais antigos utilizados pela empresa, reduzindo o tempo de resposta do time de suporte.

A segunda parte consiste em um código para análise correlacional dos dados que indicam problemas conhecidos. A ideia é quantificar o impacto de cada um dos problemas de forma que a tomada de decisão referente a priorização de melhorias, seja tomada baseando-se em dados quantitativos.

1.2.2 Objetivos Específicos

- Centralizar os dados provenientes de fontes de dados diversas;
- Modelar um Banco de Dados;
- Automatizar o processo de coleta de dados, desenvolvendo o processo de ETL (Extração, Transformação e Carga);
- Criar um *Dashboard* para a visualização de dados;
- Desenvolver um código para análise correlacional de dados;

1.3 METODOLOGIA

A metodologia adotada para o desenvolvimento do projeto foi a Metodologia Ágil, já utilizada pela empresa. Essa metodologia utiliza uma abordagem de planejamento incremental e interativa. O desenvolvimento ocorre de forma cíclica, onde ao fim de cada ciclo são definidos as etapas seguintes a serem desenvolvidas. Dessa forma, há um acompanhamento mais próximo do desenvolvimento e caso haja necessidade de modificação dos requisitos, eles podem ser alterados de forma mais dinâmica e ágil, diminuindo o impacto no projeto e minimizando o retrabalho (TOMÁS, 2009).

O projeto utilizou de um ciclo quinzenal para o planejamento e execução das tarefas no início do projeto, o qual foi alterado para um período mensal quando o projeto teve sua proposta de escopo definida e o seu desenvolvimento iniciado. Ao fim de cada ciclo era apresentado aos líderes do time de suporte o que havia sido desenvolvido e os resultados, com isso as próximas etapas de desenvolvimento eram definidas.

As macro etapas de execução do projeto foram:

- O estudo para o entendimento da problemática;
- O estudo das tecnologias Hexagon;
- Pesquisa de soluções já existentes para o embasamento da criação de propostas de solução;
- Elaboração das propostas de solução do problema;
- Escolha da melhor proposta, levando em consideração as necessidades da empresa e os requisitos da disciplina de Projeto de Fim de Curso;
- Execução das etapas de desenvolvimento do projeto - serão apresentadas com detalhes no Capítulo 5;
- Validação - será descrita com detalhes no Capítulo 5;

1.4 ESTRUTURA DO DOCUMENTO

O presente documento está dividido em sete partes, sendo as seis primeiras partes os capítulos dos elementos textuais e a última parte os elementos pós textuais. O conteúdo de cada elemento textual é apresentado a seguir:

- **Capítulo 1: Introdução** - Informa sobre a temática do trabalho e descreve os objetivos traçados;
- **Capítulo 2: Empresa** - Neste capítulo, é apresentada a empresa em que o trabalho foi desenvolvido;
- **Capítulo 3: Fundamentação teórica** - Neste capítulo, são apresentadas as tecnologias e conceitos utilizados no projeto. Além disso são descritas as tecnologias Hexagon em que o projeto se baseou;
- **Capítulo 4: Especificação do Projeto** - Neste capítulo, são apresentados as especificações levantadas para o desenvolvimento do projeto, assim como a descrição da modelagem do sistema e suas partes;
- **Capítulo 5: Desenvolvimento** - Neste capítulo, são apresentadas todas as etapas de desenvolvimento da ferramenta, assim como a validação da mesma;
- **Capítulo 6: Conclusão** - Neste capítulo, são apresentadas os resultados do trabalho, limitações e perspectivas de trabalhos futuros;

2 EMPRESA

A Hexagon foi fundada no ano de 1992 na Suécia, tem a sua sede localizada em Estocolmo (COMMERCE FOR THE UK, 2020) e atualmente possui aproximadamente 20.000 colaboradores em mais de 50 países (HEXAGON, 2020e).

As soluções da empresa podem ser divididas em três principais categorias: soluções de software, soluções de sensoriamento e soluções autônomas. As soluções de sensoriamento digitalizam o mundo físico através da utilização dos mais diversos sensores, indo desde scanners a laser, câmaras aéreas e RPAS (*Remotely Piloted Aircraft System*) até monitoramento de máquinas, tecnologias de cartografia móvel e posicionamento de alta precisão (HEXAGON, 2020b).

As soluções de software trazem a inteligência de localização que permitem a utilização de dados reais em campos de forma geo referenciadas, com alta definição e em tempo real em mapas dinâmicos que dão apoio aos sistemas de tomada de decisão das empresas. Além disso as soluções de software envolvem design e simulação para a indústria englobando os segmentos de softwares de CAD (*Computer Aided Design*), CAM (*Computer Aided Manufacturing*) e CAE (*Computer Aided Engineering*) (HEXAGON, 2020b).

As soluções autônomas buscam entregar a automatização de tarefas ou processos em um *workflow* para uma operação ou indústria como um todo - em que carros, RPAS (*Remotely Piloted Aircraft System*), veículos industriais e outros, podem operar de forma segura, confiável e eficiente (HEXAGON, 2020b).

O grupo possui oito principais divisões sendo elas: Tecnologia Geoespacial, Segurança e Infraestrutura, Gerenciamento De Portfólio de Projetos (PPM, do inglês *Project Portfolio Management*), Geosistemas, Mineração, Posicionamento, Manufatura e Agricultura (HEXAGON, 2020c). Todas dividindo a mesma missão de empregar dados para permitir ecossistemas autônomos e conectados que tragam aumento de eficiência, produtividade e qualidade para os clientes (HEXAGON, 2020f).

2.1 HEXAGON AGRICULTURE

A divisão de agricultura da Hexagon nasceu em 2014 a partir de uma fusão de três empresas com profundo conhecimento e experiência no mercado agrícola: Arvus Tecnologia (Florianópolis, Brasil), ILab sistemas (Ribeirão Preto, Brasil) e a vertical de agricultura da Hexagon Geosystems (Madrid, Espanha).

Unindo as tecnologias desenvolvidas por essas empresas, foi alcançado um potencial de automatizar operações agrícolas e aumentar a produtividade no campo ao redor do mundo (HEXAGON, 2020a).

A empresa possui um crescente portfólio de produtos e soluções que proporcionam ao cliente uma visão completa de suas operações, indo desde o planejamento

e administração da plantação até a automação de máquinas para colheita e monitoramento do transporte do material colhido. A otimização dos processos é realizada de forma contínua, maximizando a produtividade e aumentando os lucros. As soluções Hexagon realizam a coleta de dados no campo e os transformam em informações úteis para a tomada de decisão do cliente, a partir das atividades de planejamento, monitoramento, automatização e sistematização de fluxos de trabalho (HEXAGON, 2020d).

A divisão de agricultura da Hexagon tem hoje três segmentos principais de geração de receita: cana de açúcar, florestal e *aftermarket* (vendas de produtos para parceiros e distribuidores). Dentre eles, a origem do maior faturamento no mercado nacional é o da cana de açúcar, onde é parceiro de grandes clientes corporativos como por exemplo o grupo São Martinho, um dos maiores grupos sucroenergéticos do mundo. Nesse mesmo segmento a Hexagon está presente em 46% dos processos de produção mundial de cana-de-açúcar (HEXAGON, 2019).

2.2 TIME DE IMPLANTAÇÃO E SUPORTE

Em 2018, o time foi fundado a partir do esforço de Hugo Pereira Fagundes, orientador da empresa deste projeto, que detectou a necessidade de criar um time específico para as atribuições de implantação e suporte aos clientes, visto a alta e crescente demanda de clientes corporativos de configurações personalizadas das soluções. Até aquele momento não haviam responsáveis definidos por essas atividades, impactando de forma negativa o desempenho dos times de desenvolvimento de produto, que dividiam seu tempo com essas atividades.

A atividade de implantação, uma das atribuições do time, é realizada em conjunto com os técnicos em campo do setor de Assistência técnica da Hexagon. O time de Implantação e Suporte realiza a preparação das configurações dos equipamentos e do ambiente da empresa, e a instalação física dos equipamentos em campo fica por parte dos técnicos.

O suporte ao cliente se dá em três níveis, sendo os dois primeiros executados pelo corpo técnico em campo. Quando o suporte não é concluído ou não há corpo técnico disponível para o atendimento devido a localização do cliente, o time de implantação e suporte é acionado. O que faz com que o time tenha que ter a capacidade de atender uma ampla gama de problemas de forma remota, indo desde o mais simples como problemas com cabeamento de antenas, até problemas complexos de software embarcado.

Com o crescente número de clientes e novas soluções sendo implantadas, a demanda de suporte a problemas em campo também cresce, e foi justamente o que motivou o desenvolvimento deste projeto. Foi com o intuito de capacitar o time de suporte e implantação da Hexagon para atender essa crescente demanda com o

padrão de qualidade exigido por seus clientes, que o projeto foi idealizado.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as tecnologias empregadas no desenvolvimento do projeto, assim como as bases teóricas necessárias para o entendimento do projeto. Além disso, serão apresentadas também as soluções e tecnologias Hexagon, que foram os motivadores da concepção da ferramenta.

3.1 CONCEITOS ESTATÍSTICOS

3.1.1 Coeficiente de correlação de Pearson

O coeficiente de correlação, ρ , foi desenvolvido por Karl Pearson e publicado em torno do ano de 1900. É um valor numérico que mede a força e direção de uma associação linear entre uma variável independente x e uma variável dependente y (ILLOWSKY; DEAN, 2013). O coeficiente de correlação de Pearson é calculado por,

$$\rho = \frac{n \sum(xy) - (\sum x)(\sum y)}{\sqrt{[n \sum x^2 - (\sum x)^2][n \sum y^2 - (\sum y)^2]}}, \quad (1)$$

que pode ser reescrito como,

$$\rho = \frac{cov(x, y)}{\sqrt{\sigma(x)\sigma(y)}}, \quad (2)$$

onde X e Y são variáveis; cov é a covariância; $\sigma(X)$ e $\sigma(Y)$ são os desvios padrões de X e Y respectivamente.

Seus valores podem ir de -1 a 1 e refletem a intensidade da relação linear entre os conjuntos de dados. A interpretação dada aos valores de correlação são os seguintes:

- $\rho = 1$: Significa que os conjuntos de dados tem uma correlação positiva perfeita, ou seja, se uma variável cresce, a outra também cresce.
- $\rho = -1$: Significa que os conjuntos de dados tem uma correlação negativa perfeita, ou seja, se uma variável cresce, a outra diminui.
- $\rho = 0$: Significa que os conjuntos de dados não apresentam uma dependência linear entre si, entretanto não elimina a possibilidade de existir uma dependência não linear.

3.1.2 Coeficiente de correlação de Kendall

O coeficiente de correlação de Kendall, também conhecido como τ , é uma medida de relação monotônica entre variáveis.

Considere $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ um conjunto de observações das variáveis aleatórias conjuntas X e Y respectivamente. Qualquer par de observações (x_j, y_j) e (x_k, y_k) , em que $j \neq k$, é dito que (x_j, y_j) e (x_k, y_k) são concordantes se $x_j < x_k$ e $y_j < y_k$ ou se $x_j > x_k$ e $y_j > y_k$; e discordante se $x_j < x_k$ e $y_j > y_k$ ou se $x_j > x_k$ e $y_j < y_k$ (MATHEMATICS, 2020).

O cálculo do coeficiente se dá através da fórmula,

$$\tau = \frac{c - d}{n(n-1)/2}, \quad (3)$$

onde C é o número de pares concordantes; D é o número de pares discordantes e n o número de elementos do conjunto. Os valores de tau podem ir de -1 a 1 e tem a mesma interpretação apresentada no coeficiente de correlação de Pearson.

3.1.3 Coeficiente de correlação de Spearman

Coeficiente de correlação de Spearman, similar ao coeficiente de Kendall, é uma medida de relação monotônica entre variáveis contínuas ou ordinárias. Considere uma amostra aleatória $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$. Denota-se $rg(X_i)$ o Rank de X_i entre os valores de X da amostra; e $rg(Y_i)$ o Rank de Y_i entre os valores de Y da amostra (WIKIPEDIA, 2020a). O cálculo do coeficiente é dado pela fórmula

$$r_s = 1 - \frac{6 \sum (d_i)^2}{n(n^2 - 1)}, \quad (4)$$

onde n é o número de observações da amostra; $d_i = rg(X_i) - rg(Y_i)$, é a diferença entre os dois postos de cada observação. O valor de r_s vai de -1 a 1 e tem a mesma interpretação dos outros dois coeficientes apresentados anteriormente.

3.1.4 Valor-P

O valor-P, também chamado de probabilidade de significância, é a probabilidade de se observar um valor de teste estatístico qualquer, maior ou igual ao encontrado em uma amostra assumindo a hipótese nula (H_0) como verdadeira (DODGE, 2008).

Em um teste de hipótese nula, são definidas duas hipóteses, a nula (H_0) e a alternativa (H_A). Na maioria dos casos a hipótese alternativa é definida como a hipótese formulada pelo pesquisador, enquanto a hipótese nula é o seu oposto. A princípio, a hipótese nula é considerada como verdadeira. Ao confrontar a hipótese nula com os achados de um teste probabilístico em uma amostra, é realizada a rejeição ou não da hipótese. Se H_0 não for rejeitada, ela permanece como verdadeira; caso contrário, H_A é tomada como verdadeira (WIKIPEDIA, 2020b).

A utilização de rejeição da hipótese nula com base em uma amostra pode fazer com que dois tipos de erros sejam cometidos, sendo eles:

- Erro tipo 1: Quando a hipótese nula é rejeitada quando não deveria. A probabilidade de ocorrência desse erro é chamado de nível de significância, denotado pela letra grega α . O nível de significância é determinado pelo pesquisador, mas em muitas aplicações é geralmente fixado em 0,01; 0,05 ou 0,10;
- Erro tipo 2: Quando a hipótese nula não é rejeitada quando deveria ser. A probabilidade de ocorrência desse erro é denotado pela letra grega β ;

Se o valor P for menor do que o nível de significância escolhido, então a hipótese nula é rejeitada, ou seja, a amostra fornece evidências razoáveis para apoiar a hipótese alternativa. Para análise correlacional o valor p representa a porcentagem da chance de se encontrar uma correlação maior ou igual ao calculado com base no conjunto de dados. A hipótese nula é de que o conjunto de dados não têm correlação entre si, e a hipótese alternativa é de que o conjunto de dados tem correlação entre si. Para que o valor da correlação seja estatisticamente significativo a porcentagem da chance de se obter um valor maior ou igual ao calculado deve ser menor que o nível de significância (valor $p < \alpha$).

3.2 CONCEITOS DE MODELAGEM DE SISTEMAS

3.2.1 Unified Modeling Language (UML)

A Linguagem Unificada de Modelagem (do inglês *Unified Modeling Language* - UML) é uma linguagem de modelagem padronizada que utiliza um conjunto de elementos visuais, desenvolvida para auxiliar desenvolvedores de sistema de software a especificar, visualizar, construir e documentar os artefatos e funcionalidades do sistema, além disso pode ser utilizada para a modelagem de negócios e outros sistemas não relacionados a software (INC., 2020).

A UML consiste em diferentes tipos de diagramas, cada um com uma forma própria de descrição do sistema. No conjunto, os diagramas UML descrevem o limite, a estrutura e o comportamento esperado do sistema e dos objetos dentro dele.

Neste trabalho a linguagem UML foi utilizada para a elaboração de diagramas que descrevem a funcionalidade de cada um dos códigos ETL, do código de análise de dados e para a apresentação do sistema como um todo.

3.3 CONCEITOS E FERRAMENTAS DE BUSINESS INTELLIGENCE

3.3.1 Business Intelligence

Segundo (NEGASH S., 2008), *business intelligence* é definido como um sistema que combina a coleta de dados, o armazenamento de dados e o gerenciamento do conhecimento; com análises para avaliar as complexas informações corporativas e

referentes ao mercado, para apresentar aos planejadores e tomadores de decisão, com o objetivo de melhorar a conformidade e a qualidade do insumo do processo de decisão. Em outras palavras, consiste em um conjunto de técnicas e ferramentas que atuam na transformação de dados brutos em informações relevantes que ajudam as empresas a tomarem decisões de forma inteligente e orientadas a dados.

Business intelligence é considerado um verdadeiro trunfo no mundo dos negócios. Com a sua utilização faz com que as empresas consigam ter maiores margens de lucros, diminuir despesas, reagir mais rapidamente e antecipar situações. Além disso, pode automatizar o gerenciamento de dados, facilitar a colaboração e ajudar a identificar indicadores de desempenho chave para o negócio (PLAINFIELD, 2012).

Dentro dos processos mais atuais que o *business intelligence* engloba, podemos citar os seguintes destaques (TABLEAU, 2020):

- Mineração de dados: uso de técnicas estatísticas e de aprendizado de máquina para o estudo de tendências em grandes e complexos conjuntos de dados.
- Geração de relatórios: o compartilhamento das análises e gráficos gerados com as partes interessadas.
- *Benchmarking* e métricas de desempenho: a comparação de dados de desempenho atuais e históricos para acompanhar o desempenho em relação à metas.
- Análise descritiva: o uso da análise de dados passados para entender o que aconteceu.
- Consultas: extração de informações dos dados por meio de consultas.
- Análise estatística: a aplicação dos resultados da análise descritiva para realizar um estudo exploratório dos dados usando conceitos estatísticos.
- Visualização de dados: transformar a análise de dados em representações visuais, como gráficos, diagramas e histogramas.
- Preparação de dados: realizar a compilação de várias fontes de dados, limpar os dados e pré processá-los se necessário.

3.3.2 Data Warehouse

Segundo Kimball & Caserta (2004), o *data warehouse* é um sistema que extrai, limpa, transforma, entrega os dados a um banco de dados dimensional, e depois implementa e dá suporte a consultas e análises com o propósito de apoio a tomada de decisões.

Ainda, segundo Kimball & Ross (2002) o *data warehouse* tem quatro elementos fundamentais, sendo eles:

- **Fontes operacionais de dados:** A função desses sistemas é registrar as transações ou dados de operação da empresa. As prioridades do sistema é desempenho e disponibilidade. Esse elemento é a origem de dados do *data warehouse*.
- **Área de preparação dos dados:** A área de preparação de dados pode ser comparada a uma cozinha em um restaurante, onde é coletado o material bruto é transformado para um cliente final, sendo o cliente final a área de apresentação dos dados. A transformação se dá por um processo comumente denominado de ETL: *Extract-Transform-Load*. Traduzindo do inglês - Extrair, transformar e carregar, é o processo de coleta dos dados, transformação dos mesmos quando necessário, filtragem para eliminar dados inválidos ou indesejados, e carregar os dados no banco de dados.
- **Área de apresentação dos dados:** A área de apresentação dos dados é onde os dados são organizados, armazenados e se tornam disponíveis para consulta dos usuários do *data warehouse*.
- **Ferramentas de acesso aos dados:** Esse termo é de uso amplo para se referir as diversas ferramentas ou formas que pode ser aplicado para a consulta e utilização dos dados na área de apresentação de dados. Podendo eles ser: um *dashboard*, sistema de elaboração de relatórios, consultas diretas ao banco de dados, entre outros.

3.3.3 Modelagem dimensional de banco de dados

Para Kimball & Ross (2002) modelagem dimensional é a metodologia aplicada para modelar os dados de forma lógica para melhorar o desempenho de consultas e prover facilidade de utilização do *data warehouse*, a partir de um conjunto base de medições.

Há dois tipos principais de tabelas na modelagem dimensional - a tabela fato e a tabela dimensão:

- Tabela fato: tabela principal do modelo onde as informações provenientes das medições são armazenadas;
- Tabela dimensão: acompanham a tabela fato pois é onde as informações que descrevem a medida da tabela fato são armazenadas;

Por exemplo, em um caso onde deve-se modelar o banco de dados de uma concessionária a tabela fato armazenaria as informações das compras e as tabelas dimensões armazenavam as informações que parametrizam cada registro de compra como por exemplo: veículo, cliente, data, etc.

Há diversos tipos de esquemática em que essas tabelas podem ser usadas, as mais comuns são o esquema estrela e esquema floco de neve.

O esquema estrela tem essa denominação devido a sua semelhança com o formato de uma estrela, onde uma tabela fato fica centralizada e as tabelas dimensão ao seu redor. É o esquema mais simples da modelagem dimensional. Um diagrama esquemático desse modelo de esquema é apresentado pela Figura 1.

3.3.4 Microsoft Power BI

Power BI é um apanhado de ferramentas e serviços que facilitam a aquisição e modelagem de dados, visualização e criação de relatórios, como também a distribuição de soluções analíticas (COATE; WEBB, 2020).

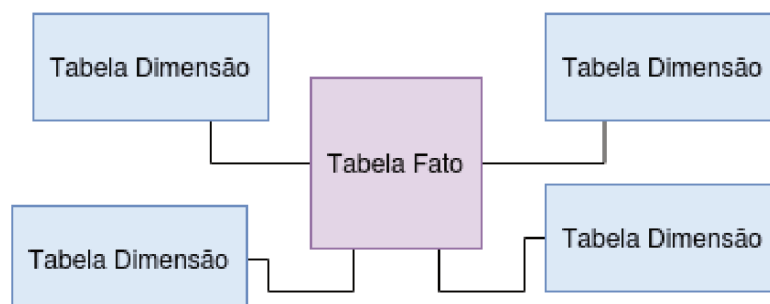
É possível dividir os seus serviços em três grandes blocos, sendo eles:

O Power BI Desktop, que é um software de licença gratuito de utilização de forma local onde é possível utilizar da grande e crescente biblioteca de conectores de dados, para trabalhar com uma ampla gama de tipos de origem de dados. Com ele é possível desenvolver visualizações e *dashboards*, utilizando de uma interface visual para a configuração dos mesmos. Os *dashboards* do Power BI Desktop podem ser exportados para o serviço do Power Bi Service, para que se tornem disponíveis para consulta de forma online.

O Power BI Service é o serviço online do Power BI. Através dele é possível compartilhar relatórios e *dashboards* com outros membros autorizados da organização além de criar relatórios em seu ambiente de edição online. É também possível configurar alarmes para indicadores e agendar a atualização dos *dashboards* para que seja realizada de forma automática.

O Power BI Mobile que é a versão para dispositivos móveis. Através deste serviço é possível realizar a visualização dos *dashboards* (NUNES, 2019).

Figura 1 – Esquema Estrela.



Fonte – Elaborado pelo autor.

3.4 TECNOLOGIAS DE IMPLEMENTAÇÃO

3.4.1 PostgreSQL

PostgreSQL é um poderoso, sistema *open-source* de banco de dados relacional que usa e estende a linguagem SQL combinado com vários outros recursos. As origens do PostgreSQL datam do ano 1986 como parte do projeto POSTGRES da universidade da Califórnia na cidade de Berkeley e tem mais de 30 anos de desenvolvimento ativo (POSTGRESQL, 2020a).

Nos últimos anos esse sistema tem ganhado grande popularidade alcançando a posição de quarto mais popular dentre os demais sistemas. Ficando atrás apenas dos gigantes Oracle, MySQL e Microsoft SQL Server (DB-ENGINES, 2020).

Foi escolhido como sistema de banco de dados para esse projeto devido a sua escalabilidade, custo zero, desempenho de consulta e inserção de dados, segurança e a sua compatibilidade com diversas ferramentas.

3.4.2 Python

Python é uma linguagem de programação interpretada, interativa e orientada a objetos. Ele incorpora módulos, exceções, tipagem dinâmica, tipos de dados dinâmicos de nível muito alto e classes. Oferece suporte a vários paradigmas de programação além da programação orientada a objetos, como a programação procedural e funcional. Tem grande portabilidade rodando em muitas variantes do Unix, incluindo Linux e macOS, e no Windows (FOUNDATION, 2020).

O primeiro artigo sobre essa linguagem foi escrito em 1991 por Gudi van Rossum e Jelke de Boer, mas foi nos últimos 15 anos que observou-se um crescimento da popularidade da linguagem (FOUNDATION, 2020).

A linguagem Python é empregada nas mais diversas frentes, atendendo demandas indo como: software embarcado de baixo nível; desenvolvimento de softwares; aplicações de sistemas Web, tanto *front* quanto *back end*; análise de dados; *machine learning*; entre outros.

A sua ampla e crescente utilização nos mais diversos meios, faz com que exista uma grande comunidade global interagindo ativamente, com muitos fóruns, documentações e um crescente número de bibliotecas para os mais diversos fins. A existência desse ecossistema faz com que a busca por informações seja facilitado, diminuindo e a curva de aprendizado.

3.4.2.1 Pandas

Pandas é uma biblioteca Python para manipulação e análise de dados. Apresenta uma ampla gama de funções matemáticas e estatísticas. É construído utilizando

a biblioteca Numpy e a sua principal estrutura é o DataFrame.

DataFrames são estruturas de dados bidimensional com dados alinhados de forma tabular em linhas e colunas, mutável em tamanho e potencialmente heterogênea onde os nomes das colunas e linhas são chamados de índices de coluna e linha respectivamente.

3.4.2.2 Matplotlib

Matplotlib é uma biblioteca de geração de gráficos e visualização de dados para Python. Desenvolvida originalmente pelo biólogo e neurocientista americano John D. Hunter (HUNTER, 2007).

3.4.2.3 Psycopg2

Psycopg é um adaptador PostgreSQL para a linguagem de programação Python (VARRAZZO, 2020). Ele realiza um envelopamento da biblioteca *libpq*, escrita em C, expondo uma API (Interface de Programação de Aplicativos) compatível com programas em Python, em outras palavras, ele implementa objetos do Python na linguagem C, para que seja possível a invocação das funções da biblioteca *libpq*. Onde *libpq* é a biblioteca que realiza a intermediação com o *back-end* dos sistemas PostgreSQL (POSTGRESQL, 2020b).

3.4.3 Amazon Web Services (AWS)

A Amazon Web Services (AWS) é uma plataforma de computação em nuvem amplamente adotada em todo o mundo, oferecendo mais de 175 serviços de infraestrutura de TI (AMAZON WEB SERVICES, Inc., 2020f). Os serviços que são utilizados na arquitetura do projeto serão apresentados a seguir.

3.4.3.1 Amazon Relational Database Service (Amazon RDS)

O Amazon Relational Database Service (Amazon RDS) é um serviço de banco de dados SQL gerenciado. Ele facilita a configuração, a operação e o dimensionamento de um banco de dados relacional na Nuvem AWS. Amazon RDS oferece suporte aos mecanismos de banco de dados MySQL, MariaDB, PostgreSQL, Oracle e Microsoft SQL Server (AMAZON WEB SERVICES, Inc, 2020b).

Ele possui valores acessíveis para armazenamento de dados, ferramentas para facilitar o redimensionamento quando necessário e gerencia as tarefas comuns de administração de bancos de dados.

3.4.3.2 AWS Lambda

O AWS Lambda é um serviço que permite a execução de códigos sem a necessidade de provisionar ou gerenciar servidores. O AWS Lambda executa o código somente quando necessário e dimensiona automaticamente, desde algumas solicitações por dia a milhares por segundo (AMAZON WEB SERVICES, Inc, 2020d).

Diversas linguagens de programação são suportadas nativamente pelo AWS Lambda como por exemplo Java, Go, PowerShell, Node.js, C#, Python e Ruby (AMAZON WEB SERVICES, Inc, 2020e).

O custo para utilização desse serviço depende do tempo de computação utilizado para execução do código, não havendo cobrança quando o código não está em execução. Pode ser configurado para que seja acionado automaticamente por meio de outros serviços da AWS, como por exemplo o Amazon Event Bridge ou o AWS IoT Core, ambos utilizados no presente trabalho.

3.4.3.3 Amazon EventBridge

O Amazon EventBridge é um serviço de barramento de eventos que facilita a conexão de aplicativos usando dados dos próprios aplicativos ou de serviços da AWS. O EventBridge fornece um fluxo de dados em tempo real de seus próprios aplicativos, aplicativos de software como serviço (SaaS) e serviços da AWS e roteia esses dados para destinos como o AWS Lambda (AMAZON WEB SERVICES, Inc, 2020c).

3.4.3.4 AWS IOT Core

AWS IOT Core O AWS IoT Core é um serviço de computação em nuvem de internet das coisas. Ele tem capacidade de comportar bilhões de dispositivos e trilhões de mensagens, assim como processar e rotear essas mensagens para *endpoints*¹ de serviços da AWS e para outros dispositivos de forma confiável e segura (AMAZON WEB SERVICES, Inc, 2020a).

3.5 TECNOLOGÍAS E CONCEITOS HEXAGON

3.5.1 Computador de bordo

Hardware desenvolvido pela Hexagon, necessário para a utilização das soluções da empresa. O computador, apresentado pela Figura 2, dispõe de uma versão de 5 (Ti5) e outra de 7 (Ti7) polegadas. Dispõe de opções de comunicação por rede de dados móvel, 3G e 4G, e Wi-Fi. Possui interfaces de comunicação CAN, USB, RS-232 e saída para antenas RFID. (AGRICULTURE, 2020) .

¹ Um *endpoint* é o ponto de entrada para um serviço.

Figura 2 – Computador de Bordo Hexagon - Ti7.



Fonte – Hexagon

3.5.2 Bin

Estrutura metálica de armazenamento temporário da cana de açúcar utilizada para o transporte do campo para a usina de processamento. O transporte se dá por meio de caminhões, mas a estrutura não é fixa aos mesmos. Normalmente é deixado em uma área específica do campo que seja de fácil acesso a caminhões. Tratores levam a cana recém colhida até a área específica e despejam dentro do *bin*. Quando preenchido, o *bin* é coletado por um caminhão da frota e levado até a usina. O momento da coleta do *bin* pelo caminhão, é apresentado na Figura 3.

Figura 3 – Caminhão realizando o descarregamento da *Bin* na usina de processamento.



Fonte – (WRITER, 2016).

3.5.3 Tag RFID

Dispositivo de armazenamento de dados de tecnologia RFID (do inglês *Radio Frequency Identification*) que tem como função o armazenamento e transporte de dados. São empregadas em caminhões e *bins* na solução RMT, servem como meio de identificação. Já para os *bins*, servem também para identificação, mas sua principal função é o armazenamento dos dados gerados em campo pela colhedora e trator. As *tags* no caso do RMT são do tipo passiva, ou seja, sua energização se dá através do próprio sistema de leitura, por meio de indução magnética ou campo eletromagnético da antena.

3.5.4 RMT - Raw Material Tracking

No setor da cana de açúcar há diversos fatores que influenciam na eficiência de uma usina de álcool ou açúcar, sendo a principal delas a qualidade da cana. Segundo a EMBRAPA (VIAN, 2020) os principais indicadores de qualidade são: POL - teor de sacarose aparente na cana, ATR - indicador que representa a quantidade total de açúcar na cana (sacarose, glicose e frutose), porcentagem de fibra na cana e açúcar redutores - quantidade de glicose e frutose presente na cana.

As usinas realizam testes na cana que chega para o processamento, com o intuito de determinar a sua qualidade segundo os indicadores já citados acima. A

motivação pode ser desde mapeamento das melhores culturas e fazendas até a determinação do valor que será pago ao produtor terceirizado pelo produto, o que depende da sua taxa de sacarose. As duas motivações citadas anteriormente, tem um requisito em comum, o local de origem da cana.

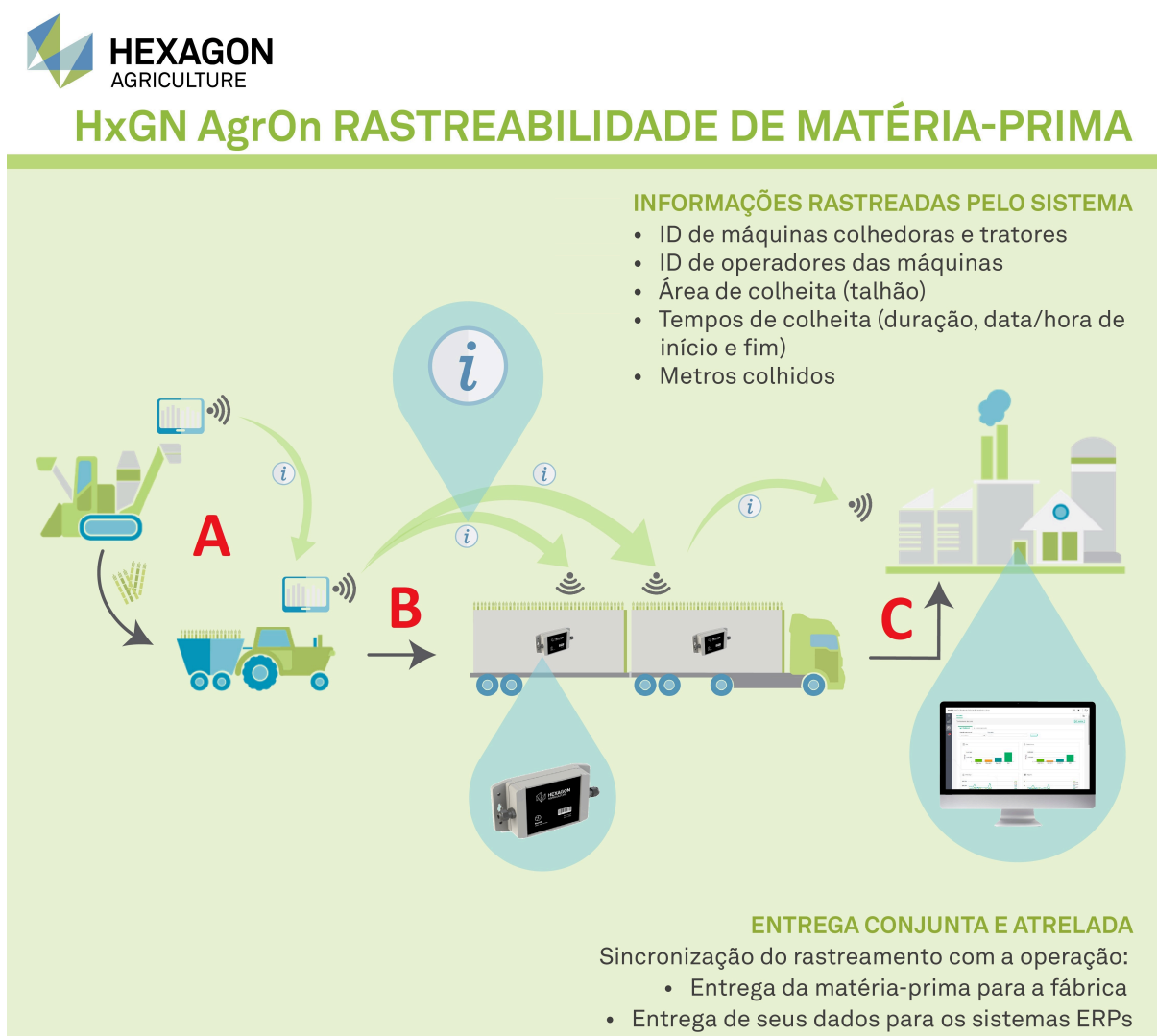
Em muitas usinas o processo de identificação da origem da cana é realizado através de apontamento manual de um supervisor em campo ou utilizando etiquetas emitidas em campo com informações de local de origem e hora da colheita, que acompanha a carga até a usina. Em um processo industrial sucroalcooleiro, onde há caminhões transportando material colhido 24 horas por dia por um período de até 7 meses, esse processo manual pode se tornar um problema, pois depende do fator humano que é um fator crítico para a ocorrência de erros como extravio de etiquetas, preenchimento com informações equivocadas, entre outros. Além disso quando executado em grande volume esse processo de forma manual consome tempo e recursos valiosos.

Foi com a motivação de automatizar o processo de identificação de origem do material colhido que a Hexagon desenvolveu a solução RMT. Que pode ser descrita resumidamente como um sistema que executa o rastreamento automatizado da matéria prima desde a origem, até a entrega na indústria. Para um maior entendimento da solução, ela foi dividida em três etapas identificadas pelas letras A, B e C na Figura 4.

- A - Na operação de colheita, representado pela Figura 5, a colhedora é acompanhada por um trator que realiza o transporte do material colhido, ambos equipados com um computador de bordo Hexagon. Os computadores de bordo durante a operação se comunicam em rede através de tecnologia *Wi-Fi* e a colhedora envia um arquivo² com as informações de colheita para o trator.
- B - Quando o trator chega a sua capacidade máxima de transporte, ele se encaminha a um *bin* próximo que esteja disponível para a transferência do material. Durante a operação de descarregamento do material, o trator, que é equipado com uma antena RFID, realiza a gravação das informações enviadas pela colhedora em uma das *tags* que estão fixadas ao redor do *bin*.
- C - Com o *bin* cheio, um caminhão da frota de transporte realiza a coleta do *bin* e se encaminha até a usina. O caminhão ao chegar na usina, realiza o processo de pesagem do material na balança rodoviária, operação ilustrada pela Figura 6, que é equipada com antenas RFID para leitura das *tags*. A balança realiza a leitura das *tags* do *bin*, e das *tags* do caminhão e envia as informações para um serviço local de processamento de dados. Assim finalizando o fluxo de dados da solução. A chegada do caminhão na usina com o *bin* é representado pela Figura 7.

² As informações presentes nesse arquivo foram omitidas deste documento por motivo de sigilo empresarial.

Figura 4 – Fluxo da Solução Hexagon - Rastreabilidade de Matéria-Prima (RMT).



Fonte – Hexagon (Editado).

A utilização do RMT faz com que o processo ganhe agilidade nas operações de transporte, redução da intervenção humana, aumento de precisão dos dados de produção e melhoria da qualidade das operações.

A solução RMT faz parte da família de soluções de monitoramento de colheita da Hexagon, onde a sua utilização depende que a solução de monitoramento de máquinas seja implantado em conjunto.

O monitoramento de máquinas é baseado no envio de dados de sensores dos veículos em operação e na descrição das operações através de um sistema ordenado de regras e atividades, chamado de *workflow*. As atividades descrevem as ações que o veículo executa em um dado momento, por exemplo: colhendo, transportando cana, etc. As atividades são determinadas por regras, que utilizam de valores de sensores

para validarem ou não as atividades. A cada minuto ou a cada transição de atividade do *workflow*, uma mensagem é enviada para o banco de dados de monitoramento da empresa com as informações³ gerais do veículo.

Figura 5 – Operação de colheita de cana de açúcar.

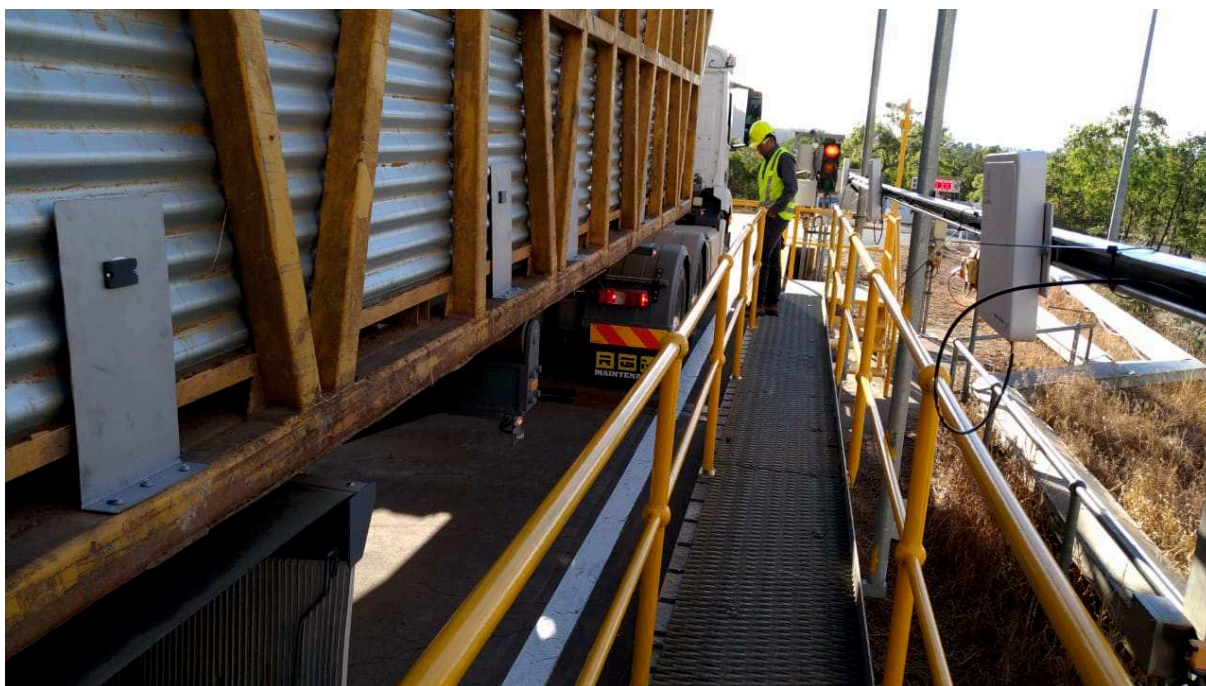


Fonte – Adaptado de (HEXAGON, 2020d).

O monitoramento de máquinas possibilita a configuração de alarmes com regras personalizadas, que quando se tornam verdadeiras enviam uma mensagem ao banco de dados de monitoramento da empresa, fazendo com que se torne possível monitorar eventos específicos de interesse. Além disso, o software embarcado já tem um conjunto de alarmes naturais com diversas funcionalidades indo desde temperatura do processador do computador de bordo até problemas com *workflow*.

³ A descrição das informações foram omitidas deste documento por motivo de sigilo empresarial.

Figura 6 – Caminhão realizando a pesagem na balança.



Fonte – Arquivo pessoal de Hugo Fagundes.

Figura 7 – Caminhão chegando na usina transportando um *bin*.



Fonte – Arquivo pessoal de Hugo Fagundes.

4 ESPECIFICAÇÃO DO PROJETO

Como anteriormente elaborado, o objetivo deste trabalho é o desenvolvimento de uma ferramenta que auxilie na coleta de dados para o suporte e acompanhamento do desempenho da solução RMT. A ferramenta proposta é dividida em duas partes, um *data warehouse* e um código de análise correlacional de dados, que são complementares em suas funções. São apresentados neste capítulo os requisitos funcionais e não funcionais de cada uma das partes da ferramenta, em seguida são apresentados os diagramas UML de modelagem do sistema e dos códigos desenvolvidos.

4.1 REQUISITOS

4.1.1 Data Warehouse

Segundo Kimball & Ross (2002), os principais requisitos que um *data warehouse* deve atender são:

- A sua fácil acessibilidade: Às informações contidas no *data warehouse* devem ser de fácil entendimento. Deve ser clara e intuitiva até para uma pessoa não familiarizada com o assunto.
- A credibilidade dos dados armazenados: Os dados devem ser cuidadosamente selecionados, limpos e processados de forma que garanta a qualidade dos dados.
- Fácil adaptação: A necessidade do cliente muda constantemente e o *data warehouse* deve mudar em conjunto, para sempre atender o usuário da melhor forma possível.
- Deve ser um local seguro para armazenamento de dados: Possivelmente o *data warehouse* pode conter informações confidenciais da empresa.
- Deve providenciar informações para apoio a tomada de decisão: As informações devem ter algum impacto na tomada de decisão para a empresa.

Com base nos requisitos apresentados acima e na problemática em que o *data warehouse* é empregado, os seguintes requisitos funcionais foram levantados:

- Atualização dos dados de forma automática frequência mínima de 24 horas.
- Hospedar o *data warehouse* na plataforma AWS.
- Os dados de interesse devem ser apresentados em forma de *dashboards*.
- Nos *dashboards* deve ser possível realizar a filtragem dos dados por veículo, frente de trabalho, empresa, data, computador de bordo e versão do *software* embarcado.

Requisitos não funcionais

- Deve unificar os dados e apresentá-los de uma forma que seja de fácil interpretação.
- Apresentar dados que tenham capacidade de apontar problemas operacionais, de *software* ou de *hardware*.
- Apresentar dados que tenham capacidade de auxiliar os gestores e o time de suporte a tomada de decisões.
- Ser escalável.
- Ter um código de fácil interpretação e manutenção.

4.1.2 Código de Análise de Dados

Durante o desenvolvimento do projeto foi percebido a necessidade da análise de dados além da apresentação de dados em *dashboards* ao usuário. Assim foi concebida a ideia de uma análise correlacional dos dados que indicam problemas conhecidos com o indicador de desempenho da solução, com o intuito de quantificar a influência dos problemas no desempenho. Os requisitos funcionais dessa etapa são:

- Possibilitar uma análise entre diferentes períodos de tempo.
- Deve indicar ao usuário os 10 conjuntos de dados com maior influência no indicador de desempenho da solução RMT.
- Apresentar todas as informações de forma gráfica em um arquivo formato PDF, para facilitar o compartilhamento dos resultados.

Os requisitos não funcionais levantados foram:

- Ser escalável.
- Ter um código de fácil interpretação e manutenção.

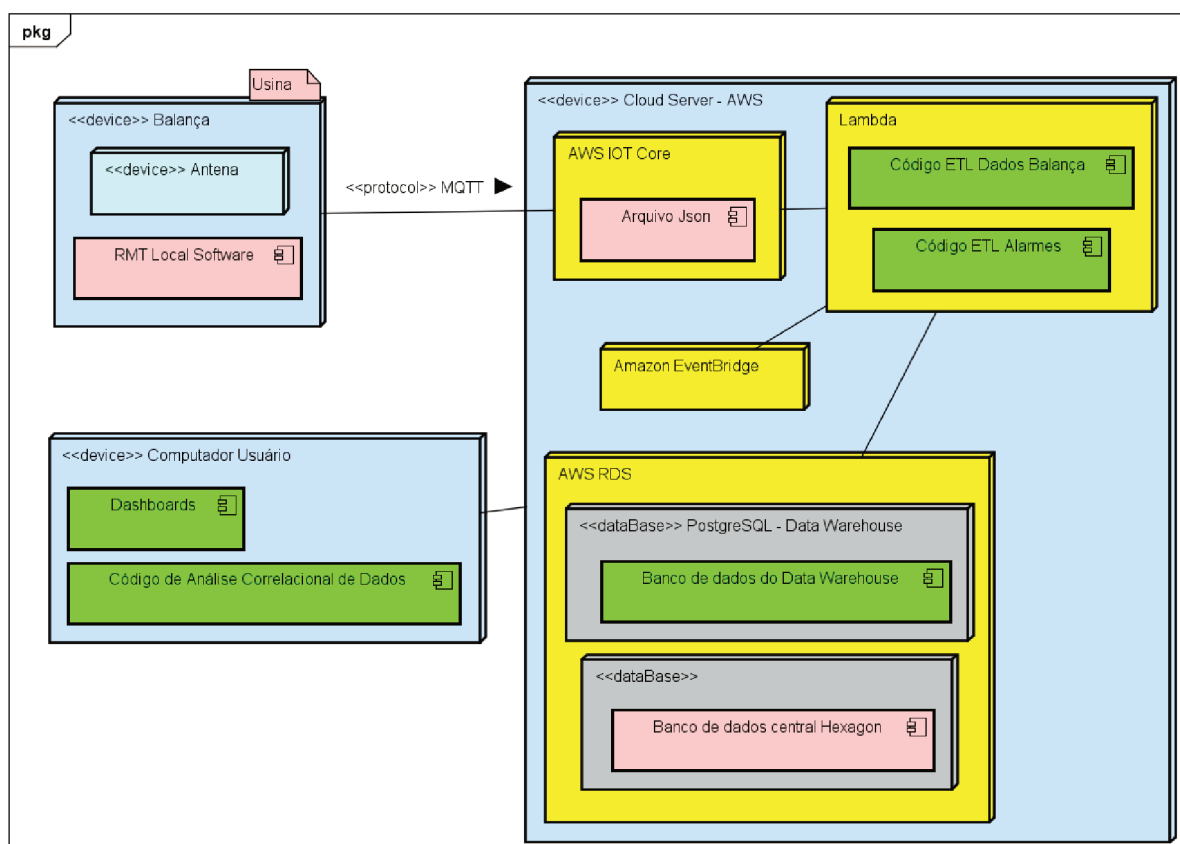
4.2 MODELAGEM

Nessa seção serão apresentados os modelos das partes desenvolvidas do sistema e o modelo do sistema como um todo.

4.2.1 Modelo Sistema

O sistema geral, que abrange a solução e as ferramentas utilizadas é apresentado em forma de um diagrama de depuração pela Figura 8, onde os componentes que foram de desenvolvimento do autor estão identificados pela cor verde.

Figura 8 – Diagrama UML de depuração representando o sistema.



Fonte – Elaborado pelo autor.

4.2.2 Modelo Primeiro ETL - Dados da Balança

Um diagrama de UML de atividade é apresentado pela Figura 9, demonstrando a funcionalidade do código do ETL referente aos dados da balança. No Capítulo 5 será apresentado em detalhes o seu funcionamento e sua motivação.

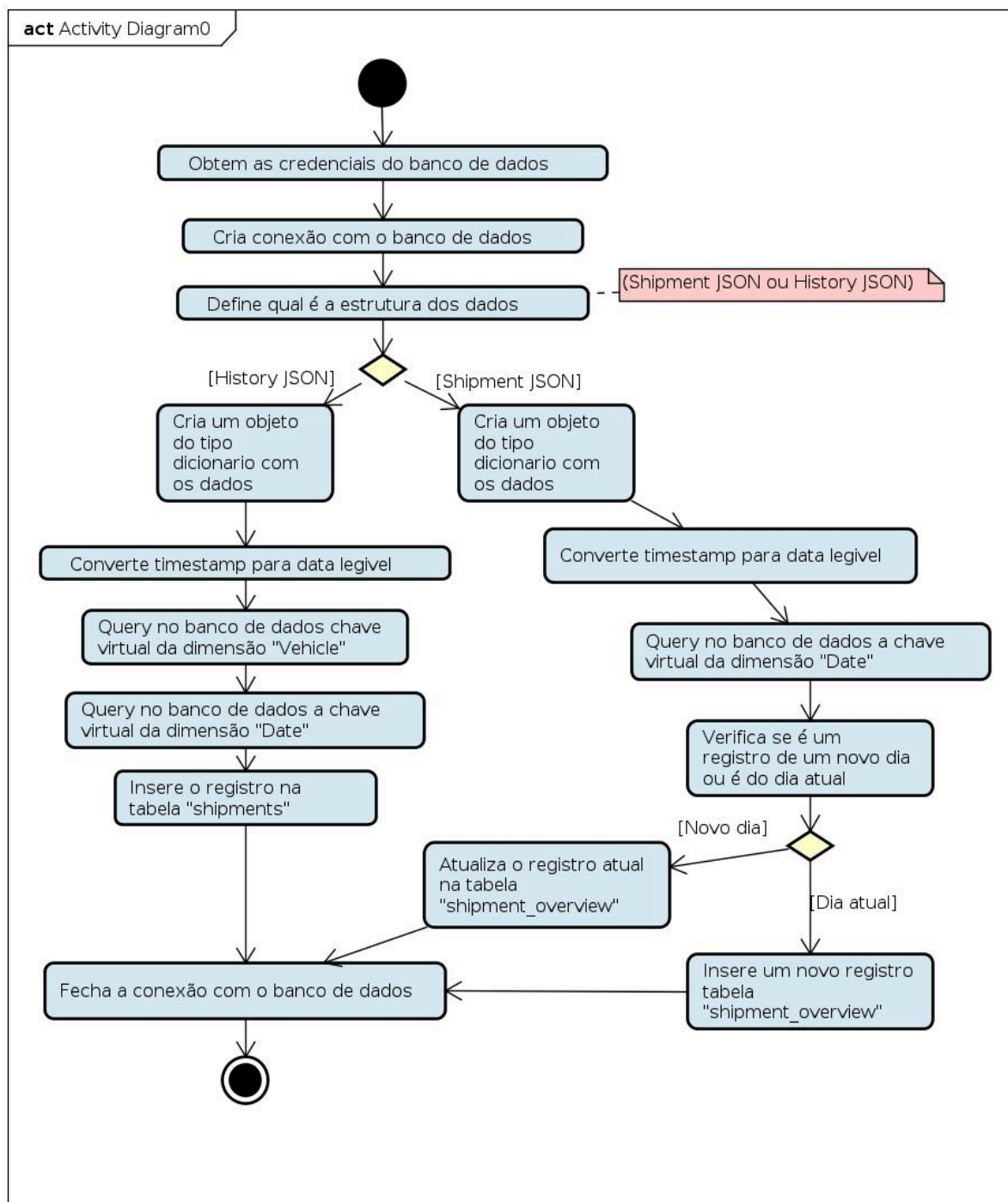
4.2.3 Modelo Segundo ETL - Alarmes

Um diagrama de UML de atividade é apresentado pela Figura 9, demonstrando a funcionalidade do código. No Capítulo 5 serão fornecidas descrições mais detalhadas e a apresentação de cada trecho do código.

4.2.4 Código de Análise Correlacional dos Dados

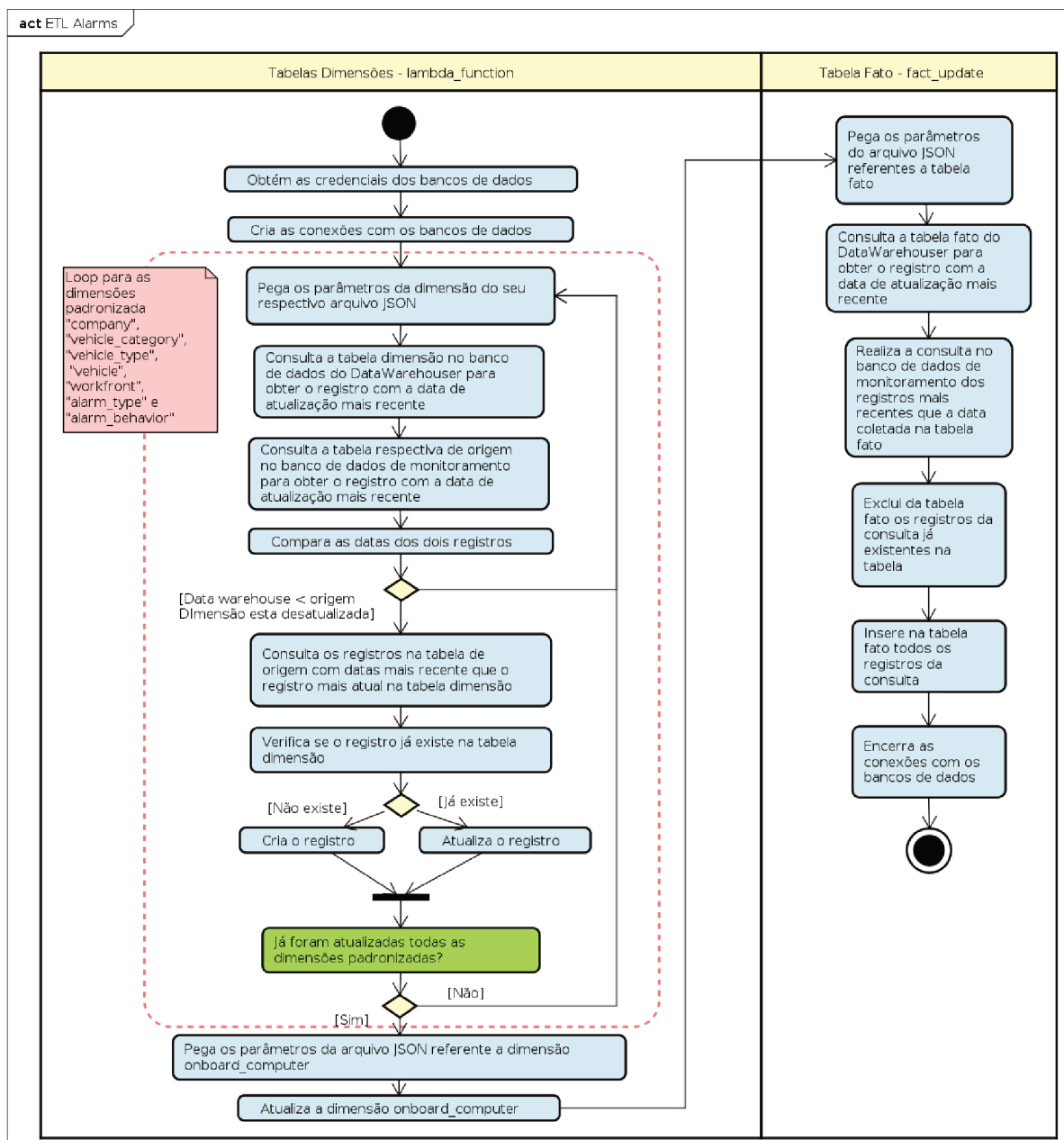
O funcionamento do código é representado por um diagrama UML de atividade apresentado pela Figura 11.

Figura 9 – Diagrama UML de atividade do primeiro ETL.



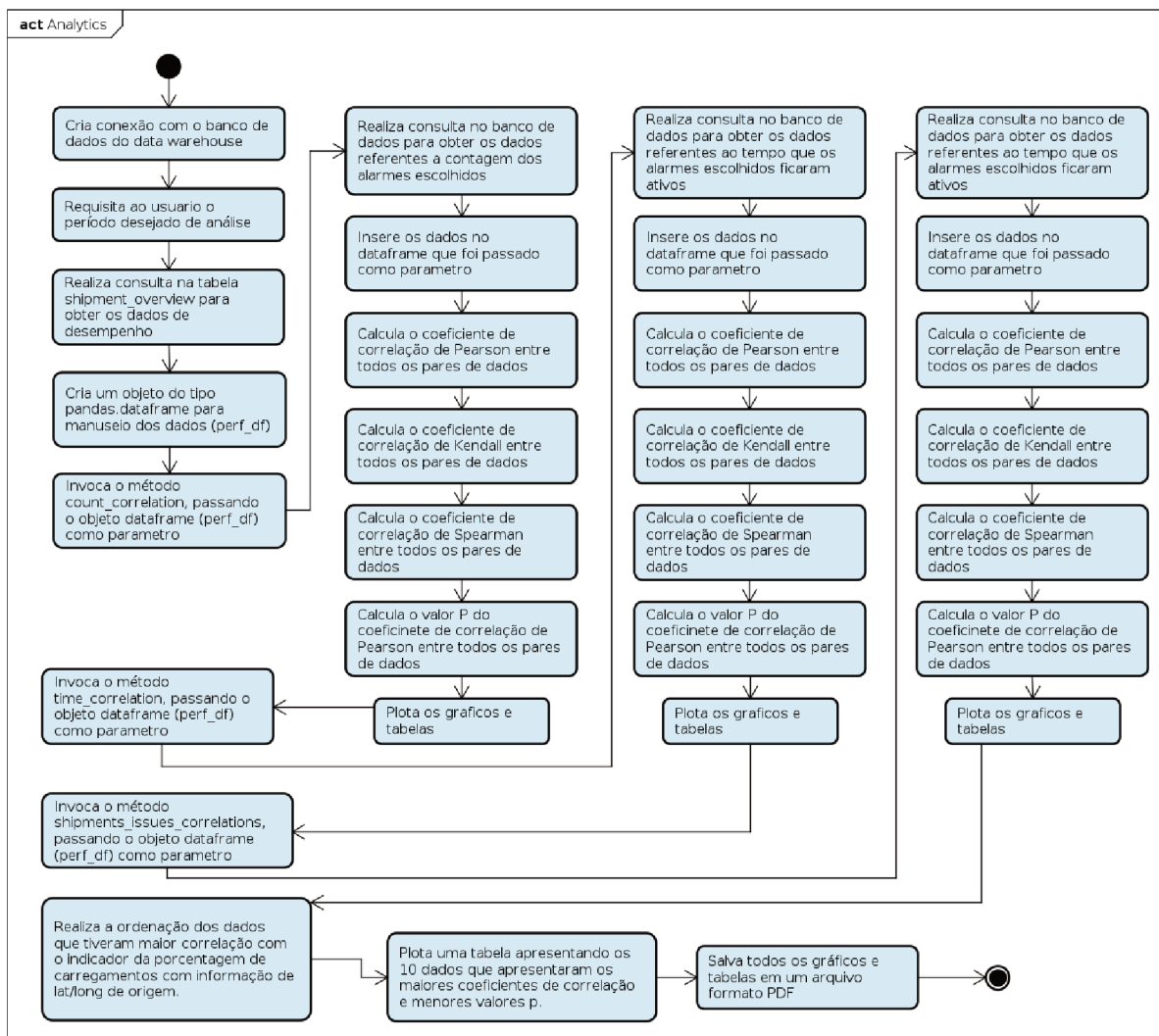
Fonte – Elaborado pelo autor.

Figura 10 – Diagrama UML de atividade do segundo ETL.



Fonte – Elaborado pelo autor.

Figura 11 – Diagrama UML de atividade do código de análise de dados.



Fonte – Elaborado pelo autor.

5 DESENVOLVIMENTO

Neste capítulo é apresentado o desenvolvimento da ferramenta, descrevendo em detalhes todas as etapas, desde os primeiros mapeamentos até a validação da ferramenta. Todas as etapas de desenvolvimento apresentadas foram realizadas exclusivamente pelo autor, exceto quando explicitamente apontado o contrário.

5.1 MAPEAMENTO DAS FONTES DE DADOS

A primeira etapa realizada no projeto foi o estudo da solução RMT e um entendimento do seu fluxo de dados, que foi descrito no capítulo 3. Com o fluxo de dados da solução mapeado, foi definida uma lista das possíveis origens de dados que pudessem dar apoio ao time de suporte na identificação de problemas em campo. Duas características principais foram estudadas em cada uma das fontes, sendo elas:

- **Nível de dificuldade de acesso:** Um banco de dados é uma fonte de fácil acesso, pois há a possibilidade de acesso diário e irrestrito. Um arquivo de *log* de sistema do computador de bordo é um arquivo de difícil acesso, visto que o software embarcado não dispõe de um recurso de envio de arquivos *log*.
- **Potencial informacional dos dados:** Os dados presentes no banco de dados de monitoramento da empresa trazem informações dos sensores e apontamento de atividades do veículo, ou seja, informações de utilização. Os dados contidos em um arquivo *log* do sistema do computador de bordo trazem informações referentes ao *software* em si.

A lista das possíveis origens de dados mapeadas é apresentada pela Tabela 1.

Tabela 1 – Possíveis origens de dados.

Origem	Descrição	Dados - Potencial Informacional	Dificuldade
Logs do sistema.	Logs operacionais do <i>software</i> embarcado.	Informações do <i>software</i> embarcado.	Alta, não há um recurso para o envio desses dados.
Diagnósticos de falhas do sistema.	Logs dos diagnósticos extraídos quando há falhas críticas do sistema.	Informações do <i>software</i> embarcado.	Alta, dependeria do apoio de clientes e técnicos para coleta dos mesmos.
Servidor RMT local do cliente.	Banco de dados do servidor local do cliente.	Dados dos carregamentos.	Alta, dependeria de um acesso remoto ao servidor do cliente, se tornando algo complexo e não seguro.
Banco de dados RMT na nuvem.	Dados não processados do serviço de leitura da balança, referente aos carregamentos.	Dados dos carregamentos.	Média, o acesso é fácil mas os dados dependem de um nível de processamento elevado. Estes são os dados de <i>backup</i> não processados do servidor RMT local.
Banco de dados de monitoramento Hexagon.	Banco de dados central da plataforma de monitoramento de máquinas Hexagon.	Dados de monitoramento. (Sensores, atividades, dados de configuração e alarmes).	Baixa, o banco de dados pertence a Hexagon e é hospedado no serviço de <i>cloud</i> AWS, fazendo com que seja de fácil acesso.
Dados enviados pela balança para o AWS IoT Core.	Mensagens em formato de arquivo JSON com os dados de carregamento de forma resumida que são enviados pela balança para o serviço AWS IoT Core.	Dados dos carregamentos.	Baixa, os dados são enviados para um serviço de <i>cloud</i> AWS IoT Core, fazendo-se necessário apenas direcionar esses dados para uma outra aplicação de processamento.

Fonte – Elaborado pelo autor.

5.2 FORMULAÇÃO DA LISTA DAS INFORMAÇÕES DE INTERESSE

Para a execução desta etapa, foi necessário o apoio dos líderes do time de suporte, que através de reuniões puderam dividir a sua experiência com a safra do ano anterior e indicar as informações que foram úteis para a identificação dos motivos de queda de desempenho da solução RMT naquele ano. A lista das informações de interesse é apresentada pela Tabela 2.

Tabela 2 – Informações de interesse.

Informação	Descrição
Número de carregamentos sem informação de origem do material	Número de carregamentos que quando processados pela balança, não contêm os dados de origem do material colhido nas <i>tags</i> . Como a função principal da solução é a entrega da informação da localização da origem da cana, este é considerado o indicador de desempenho da solução.
Tempo registrado sem <i>login</i> do operador	Tempo em que o computador de bordo fica sem operar por falta de <i>login</i> do operador do veículo.
Número de problemas de pareamento Wi-Fi entre veículos.	Número de vezes que o pareamento colhedora-trator apresenta algum erro devido a problemas com Wi-Fi.
Número de problemas de pareamento RFID entre veículos e <i>tags</i> .	Número de vezes que o pareamento trator- <i>tags</i> apresenta algum erro devido a problemas com RFID.
Número de <i>tags</i> lidas na balança em cada carregamento.	Número de <i>tags</i> lidas pela balança em cada carregamento.
Número de erros de leitura na balança	Número de erros de leitura na balança.

Fonte – Elaborado pelo autor.

Com os mapeamentos citados, foi possível realizar um estudo para a definição da mais adequada fonte para cada uma das informações de interesse, sendo elas:

Os dados enviados pela balança para o AWS IoT Core: é uma fonte de dados de fácil acesso, pois é possível ter acesso ao recebimento dos dados por uma aplicação na plataforma *cloud* AWS, e a sua utilização não tem impacto na solução RMT ou nas atividades da Hexagon. As informações que são enviadas pela balança são referentes a cada um dos carregamentos que chegam na usina, onde o seu conteúdo inclui o número de *tags* lidas, se há ou não informação de origem do material registrado nas *tags*, horário de chegada do caminhão na balança e o número de identificação da *bin* e do caminhão. Atendendo assim a necessidade das seguintes informações de interesse:

- Número de carregamentos sem origem
- Número de *tags* lidas por carregamento
- Número de erros de leitura na balança

Banco de dados de monitoramento da Hexagon: foi definido como a origem das seguintes informações de interesse:

- Tempo registrado sem *login* do operador
- Número de problemas de pareamento Wi-Fi entre veículos
- Número de problemas de pareamento RFID entre veículos e *tags*.

Há um grande volume de dados neste banco de dados, indo desde os valores lidos dos sensores a cada minuto até a indicação da atualização de *software* de

cada um dos computadores de bordo. Uma tabela em especial é a fonte das informações citadas: a tabela de alarmes, onde são registrados os alarmes oriundos dos computadores de bordo.

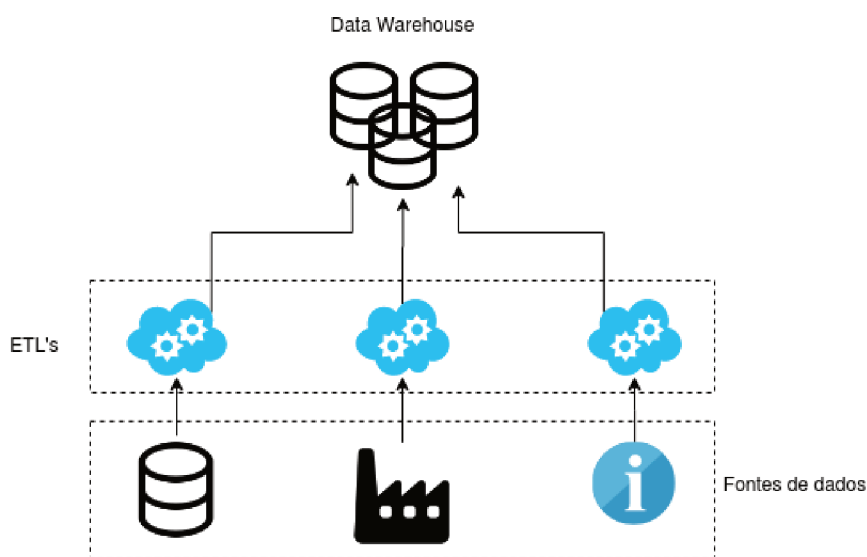
Além dos alarmes naturais do *software*, como por exemplo a falta de conexão com o GPS ou a falta de memória de armazenamento do computador de bordo, há a possibilidade da configuração de alarmes baseados em regras que são validadas ou não segundo valores dos sensores e atividades. No ano anterior, para o auxílio na depuração de problemas em campo, uma série de alarmes foram configurados para o alerta de cenários que poderiam indicar algum problema, como por exemplo o alarme 22 de *index* 187, que é acionado quando uma colhedora está na atividade “Colhendo” mas não está detectando o pareamento via Wi-Fi com um trator. A colhedora só entra na atividade colhendo quando o sensor que indica que o cortador de cana e o elevador (esteira que joga a cana para o trator) estão acionados. Pela lógica a atividade “Colhendo” só deveria ser verdadeira se o trator estiver realmente colhendo, e por sua vez para a colhedora realizar a operação de corte e colheita de cana é obrigatório que um trator esteja do lado dela, caso contrário a colhedora iria cortar a cana e jogar no chão. Para o RMT é necessário que o trator esteja ao lado da colhedora para que ele receba o arquivo de colheita que contém a origem do material através do pareamento Wi-Fi. Quando a colhedora não identifica o pareamento e está na atividade colhendo é um indício de que há algum problema no pareamento Wi-Fi entre colhedora e trator.

Como as três informações citadas tinham alarmes que poderiam indicar o eventos que são relacionados às informações de interesse, a tabela de alarmes como um todo foi adicionada ao escopo do projeto como fonte de dados do *data warehouse*.

5.3 DESENVOLVIMENTO DOS ETLs

Para o desenvolvimento da parte dos códigos de extração, limpeza e inserção (ETL) do *data warehouse*, foi adotada uma arquitetura horizontalizada, conforme representado na Figura 12. Serão distribuídas as coletas e processamentos das diferentes origens de dados entre códigos ETLs distintos, com execuções assíncronas, em outras palavras serão entidades separadas. Essa abordagem foi adotada devido a clara separação das fontes de dados e para facilitar a futura inserção de novas fontes, atendendo o requisito de escalabilidade do *data warehouse*.

Figura 12 – Esquemático da arquitetura do fluxo de dados do *data warehouse*.

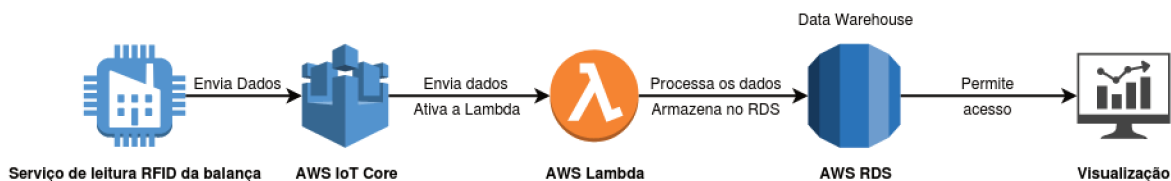


Fonte – Elaborado pelo autor.

5.3.1 Desenvolvimento do primeiro ETL - Dados do serviço da balança da usina

A arquitetura do fluxo de dados do primeiro ETL inicia na balança rodoviária da usina, onde há uma aplicação que realiza a leitura das *tags* da *bin* e do caminhão e realiza o processamento dos dados coletados. Em seguida as informações são enviadas para o servidor do cliente e uma versão resumida é enviada para o serviço de Internet Das Coisas na plataforma AWS, chamada de AWS IoT Core. Na chegada dos dados no AWS IoT core, o mesmo encaminha os dados e realiza a invocação do código de ETL no serviço AWS Lambda. Que realiza o processamento das informações e armazena no banco de dados hospedado no serviço AWS RDS. A arquitetura é apresentada de forma gráfica pela Figura 13.

Figura 13 – Fluxo de dados do primeiro ETL.



Fonte – Elaborado pelo autor.

5.3.1.1 Dados enviados do serviço da balança para o AWS IoT Core:

Os dados oriundos da balança vem em formato JSON e apresentam duas estruturas distintas, que são enviadas de forma sequencial para o serviço IoT Core. A primeira é enviada com as informações referentes ao carregamento do caminhão que consta naquele momento na balança, contendo as informações do código do cliente, número identificador do *bin* e do caminhão, status com que o carregamento foi fechado, número de *tags* lidas e a quantidade de arquivos de colheita gravadas nas *tags* RFID do *Bin* (*frames_count*). Para fins de facilitar o entendimento a primeira estrutura de dados foi nomeada de *Shipment JSON* (do inglês, JSON do Carregamento), a mesma é apresentada pela Figura 14.

Figura 14 – Estrutura de dados *Shipment JSON*.

```
{
  "format": "json",
  "payload": {
    "customer": "msf",
    "payload": {
      "bin_code": 118,
      "frames_count": 3,
      "shipment_state": "Shipment is closed. Got enough tags.",
      "shipment_state_code": 3,
      "tags_count": 12,
      "truck_code": 1002
    },
    "service": "data_collector"
  },
  "qos": 0,
  "timestamp": 1591203238739,
  "topic": "rmt/weighbridge/shipment/118"
}
```

Fonte – Elaborado pelo autor.

A segunda estrutura enviada logo na sequência, é o resultado de uma compilação de todos os carregamentos do dia (fuso horário UTC), trazendo as informações de quantidade total de carregamentos, número de carregamentos com informação de longitude e latitude de origem (*shipments_with_rmt_data*), número de carregamentos com informação do código de identificação do talhão de origem (*shipments_with_location_id*), número de carregamentos pelo número de *tags* lidas, número de carregamentos pela quantidade de arquivos de colheitas gravadas nas *tags* dos *bins* (*tipping_count*) e código do cliente (*customer*). Para fins de facilitar o entendimento a segunda estrutura foi nomeada de *History JSON* (do inglês, JSON do Histórico), a mesma é apresentada pela Figura 15.

Figura 15 – Estrutura de dados *History JSON*.

```

{
  "format": "json",
  "payload": {
    "customer": "msf",
    "payload": {
      "shipments_count": 81,
      "shipments_with_location_id": 70,
      "shipments_with_rmt_data": 78,
      "tags_count_history": [
        {
          "shipments_count": 80,
          "tags_count": 12
        },
        {
          "shipments_count": 1,
          "tags_count": 13
        }
      ],
      "tippings_count_history": [
        {
          "shipments_count": 40,
          "tipping_count": 13
        },
        {
          "shipments_count": 40,
          "tipping_count": 9
        },
        {
          "shipments_count": 1,
          "tipping_count": 7
        }
      ]
    }
  },
  "service": "data_collector"
},
"qos": 0,
"timestamp": 1591192207367,
"topic": "rmt/weighbridge/shipment_statistics/"
}

```

Fonte – Elaborado pelo autor.

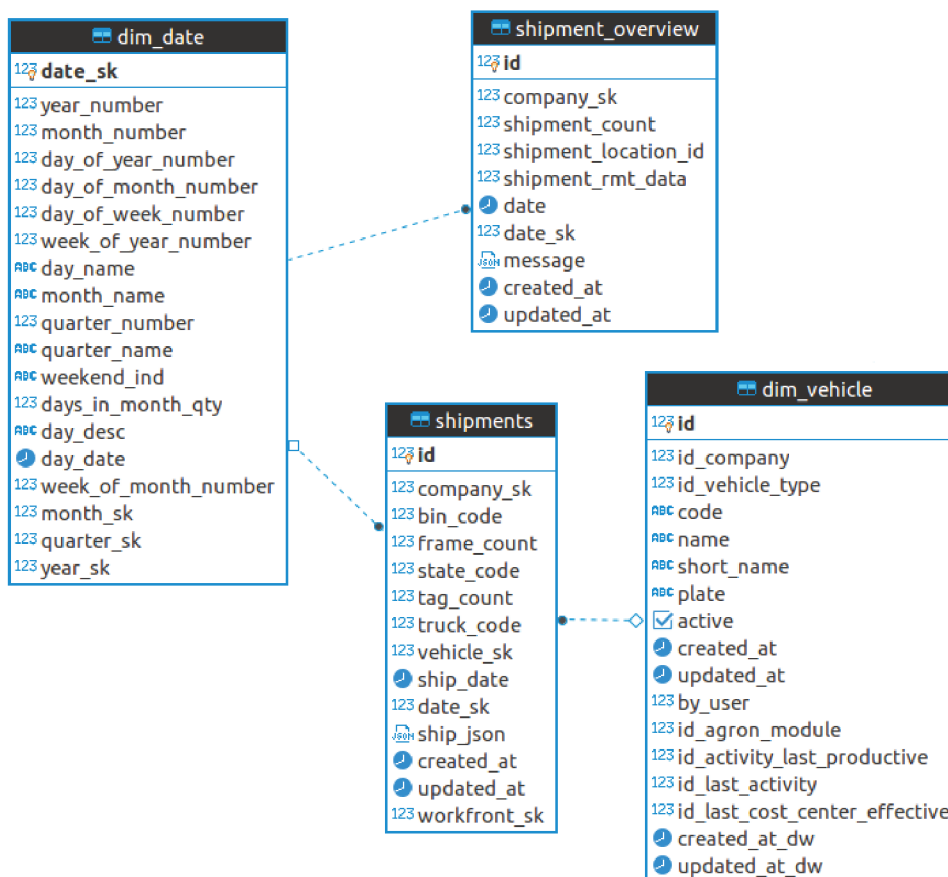
5.3.1.2 Modelagem do Banco de Dados:

As mensagens serão armazenadas em duas tabelas distintas, uma para cada tipo de estrutura de dados. Os dados com a estrutura do tipo *Shipment JSON*, são registradas na tabela *shipments*, que tem como função de tabela fato na arquitetura do banco de dados e tem como dimensões a tabela *dim_date* e *dim_vehicle* que serão apresentados posteriormente no desenvolvimento do segundo ETL.

Os dados referente a estrutura de dados *History JSON* são armazenados na tabela *shipment_overview*. As informações dessa tabela são utilizadas para a geração dos principais indicadores de desempenho da solução, a taxa de carregamentos com informações de origem latitude/longitude e a taxa de carregamentos com código do talhão de origem. Além disso, é também utilizada como parâmetro de comparação

para verificação da contagem de carregamentos registrados na tabela *shipments*. A arquitetura do banco de dados dessa etapa do projeto é representada pela Figura 16.

Figura 16 – Diagrama relacional do banco de dados.



Fonte – Elaborado pelo autor.

5.3.1.3 Código ETL - Lambda

O código do ETL foi desenvolvido na linguagem Python. Sua estrutura conta com uma função principal e uma classe, a qual tem como métodos todas as outras funções do código. Essa estrutura foi escolhida com o intuito de deixar o código mais organizado e legível, facilitando assim a sua manutenção.

A função principal do código mencionada anteriormente é a primeira função invocada quando o código é executado pela Lambda, fazendo o papel de uma função "main". Ela foi nomeada de *lambda_handler* e quando invocada cria um objeto do tipo *lotJsonParser*, que é a classe principal do código. A função é apresentada pela Figura 17, a qual o atributo "event" se refere aos dados recebidos do IoT Core - *Shipment JSON* ou *History JSON*.

Figura 17 – Função `lambda_handler`.

```
def lambda_handler(event, context):  
  
    print("## EVENT")  
    print(event)  
    IotJsonParsers = IotJsonParser()  
    IotJsonParsers.define(event)
```

Fonte – Elaborado pelo autor.

Os métodos da classe `IotJsonParser` serão apresentados a seguir, seguindo a ordem sequencial de execução:

db_connect: É invocado pelo construtor da classe e realiza a conexão com o banco de dados do *data warehouse* utilizando a biblioteca `psycopg2`. As credenciais de acesso são obtidas através do acesso às variáveis de ambiente da Lambda utilizando a biblioteca “os” (*Miscellaneous operating system interfaces*). Foi utilizado desse método por motivos de segurança, para que as credenciais não fiquem registradas diretamente no código. A parte do código referente ao construtor da classe e o método `db_connect` é apresentado abaixo pela Figura 18.

Figura 18 – Construtor da classe `IotJsonParser` e o método `db_connect`.

```
class IotJsonParser:  
  
    def __init__(self):  
        self.conn = self.db_connect(user,password,host,port,db_name)  
        self.cur = self.conn.cursor()  
  
    def db_connect(self,user,password,host,port,db_name):  
        try:  
            conn = psycopg2.connect(host=host, port=port,  
                                   user=user, password=password,  
                                   database=db_name, connect_timeout=10)  
        except psycopg2.Error as e:  
            print("ERROR: Unexpected error: Could not connect to Postgresql instance.")  
            print(e)  
            sys.exit()  
        else:  
            return conn
```

Fonte – Elaborado pelo autor.

define: Realiza a identificação de qual o tipo estrutural dos dados que serão processados (*History JSON* ou *Shipment JSON*). A identificação se dá por meio da verificação da existência da chave *shipments_count*, existente somente na estrutura *History JSON*. Após a identificação é realizada a leitura do arquivo JSON modelo do mesmo tipo da estrutura identificada e invocado o método *parse_json*, passando um objeto do tipo dicionário criado a partir das informações contidas no arquivo modelo. Em seguida o método *verify_day_history* é invocado no caso do *History JSON* ou o método *create_shipment* no caso do *Shipment JSON*. O trecho do código é apresentado pela Figura 19.

Figura 19 – Método *define*.

```
def define(self,event):
    data = event
    print("Event Loaded")
    if ("shipments_count" in data["payload"]):
        print ("+++++++ HistoricoJson ++++++")
        historyJson = open('ship_hist_model.json','r')
        model = json.load(historyJson)
        info = self.parse_json(data,model)
        print(info)
        print("Json was parsed with success!")
        self.verify_day_history_json(info)
    else:
        print ("+++++++ ShipmentJson ++++++")
        historyJson = open('shipment_model.json','r')
        model = json.load(historyJson)
        # print(self.parse_json(data,model))
        info = self.parse_json(data,model)
        print(info)
        print("Json was parsed with success!")
        self.create_shipment(info)
```

Fonte – Elaborado pelo autor.

parse_json: Realiza a comparação da mensagem que está sendo processada com a estrutura do JSON modelo para determinar se não há nenhuma chave faltante e cria um objeto dicionário com as informações contidas na mensagem. O modelo da estrutura *shipment JSON* e o trecho do código referente ao método *parse_json* são apresentados a seguir pela Figura 20 e Figura 21, respectivamente.

Figura 20 – Método *parse_json*.

```

def parse_json(self,data,model):
    shipDict = {}
    for key, value in model.items():
        if isinstance(value, dict):
            print("++++ Dict Parser +++++")
            for subkey, subvalue in self.parse_json(data[key], model=model[key]).items():
                shipDict[subkey] = subvalue
        else:
            try:
                print("{} ok".format(key))
                if key == "customer":
                    if data[key] == 'msf':
                        shipDict[key] = 50067
                    else:
                        shipDict[key] = data[key]
            except KeyError:
                shipDict[key] = None
                print("{} info is missing".format(key))

    shipDict['dataJson'] = json.dumps(data)
    return (shipDict)

```

Fonte – Elaborado pelo autor.

Figura 21 – Arquivo JSON modelo.

```

{
  "customer": "",
  "payload": {
    "shipments_count": 0,
    "shipments_with_location_id": 0,
    "shipments_with_rmt_data": 0,
    "tags_count_history": [
      {
        "shipments_count": 0,
        "tags_count": 0
      }
    ],
    "tippings_count_history": [
      {
        "shipments_count": 0,
        "tipping_count": 0
      }
    ]
  },
  "timestamp": 0,
  "service": "data_collector"
}

```

Fonte – Elaborado pelo autor.

create_shipment: Esse método tem como função a de criação do registro do carregamento no banco de dados. Ele recebe o objeto do tipo dicionário criado no método anterior chamado *info* com os dados do carregamento, transforma o *timestamp* de origem para uma data do tipo legível e invoca os métodos que buscam as chaves virtuais das dimensões veículo e data passando o objeto *info*. Com as chaves ele realiza uma inserção no banco de dados e finaliza a conexão com o mesmo. O trecho do código é apresentado pela Figura 22.

Figura 22 – Método *create_shipment*.

```
def create_shipment(self,info):
    registerDate = datetime.fromtimestamp(int(info['timestamp']/1000),
                                         timezone.utc).strftime('%Y-%m-%d %H:%M:%S')

    print(registerDate)
    info ['registerDate'] = registerDate
    info["workfront_sk"] = 0
    try:
        info["vehicle_sk"] = self.get_vehicle_sk(info)
    except:
        print('Something went wrong with getvehiclessk')
        info["vehicle_sk"] = 0
    try:
        info["date_sk"] = self.get_dateSk(info)
    except Exception as e:
        print('Something went wrong with get_dateSk')
        print(e)
        info["date_sk"] = 0
    try:
        self.cur.execute("INSERT INTO shipments(company_sk,bin_code,frame_count,\
                    state_code,tag_count,truck_code,vehicle_sk,ship_date,date_sk,ship_json,\
                    created_at,updated_at,workfront_sk) VALUES (%(customer)s,\
                    %(bin_code)s,%(frames_count)s,%(shipment_state_code)s,%(tags_count)s,\
                    %(truck_code)s,%(vehicle_sk)s,%(registerDate)s,%(date_sk)s,%(dataJson)s,\
                    now()::timestamp,now()::timestamp,%(workfront_sk)s);",info)

        self.conn.commit()
        self.conn.close()
        print("The shipment record was created with succes!")
    except:
        print("Error during shipment record inserton!")
```

Fonte – Elaborado pelo autor.

verify_day_history_json: Realiza a conversão do *timestamp* para formato de data em seguida realiza uma consulta na tabela *shipment_overview* para verificar se há algum registro com a mesma data. Caso exista, o método *update_history_json* é invocado para a atualização desse registro, caso contrário o método *create_history_json* é invocado para a criação de um novo registro. O trecho do código é apresentado na Figura 23.

Figura 23 – Método *verify_day_history_json*.

```
def verify_day_history_json(self, info):
    registerDate = datetime.fromtimestamp(int(info['timestamp']/1000),
                                         timezone.utc).strftime('%Y-%m-%d %H:%M:%S')
    info['registerDate'] = registerDate
    info["date_sk"] = self.get_dateSk(info)
    self.cur.execute("SELECT id FROM shipment_overview WHERE \
updated_at::date = %(registerDate)s::date;", info)
    query_result = self.cur.fetchall()
    if len(query_result) > 0 :
        print('Updating Overview Record!')
        self.update_history_json(info)
    else:
        print('Creating new record!')
        self.create_history_json(info)
```

Fonte – Elaborado pelo autor.

Update_history_json e create_history_json: Os dois métodos tem a função de atualização de registro e inserção de registro, respectivamente. A estrutura é a mesma para os dois, mudando apenas a instrução SQL. A Figura 24 apresenta o trecho do código do método *update_history_json*.

Figura 24 – Método *update_history_json*.

```
def verify_day_history_json(self, info):
    registerDate = datetime.fromtimestamp(int(info['timestamp']/1000),
                                         timezone.utc).strftime('%Y-%m-%d %H:%M:%S')
    info['registerDate'] = registerDate
    info["date_sk"] = self.get_dateSk(info)
    self.cur.execute("SELECT id FROM shipment_overview WHERE \
updated_at::date = %(registerDate)s::date;", info)
    query_result = self.cur.fetchall()
    if len(query_result) > 0 :
        print('Updating Overview Record!')
        self.update_history_json(info)
    else:
        print('Creating new record!')
        self.create_history_json(info)
```

Fonte – Elaborado pelo autor.

Get_vehicle_sk e get_dateSk: Realizam a consulta nas dimensões *dim_vehicle* e *dim_date* para a obtenção da chave primária dos registros do veículo que está reali-

zando a descarga na usina e do dia da operação respectivamente. O trecho do código referente ao método `get_dateSk` é apresentado pela Figura 25.

Figura 25 – Método `get_vehicle_sk`.

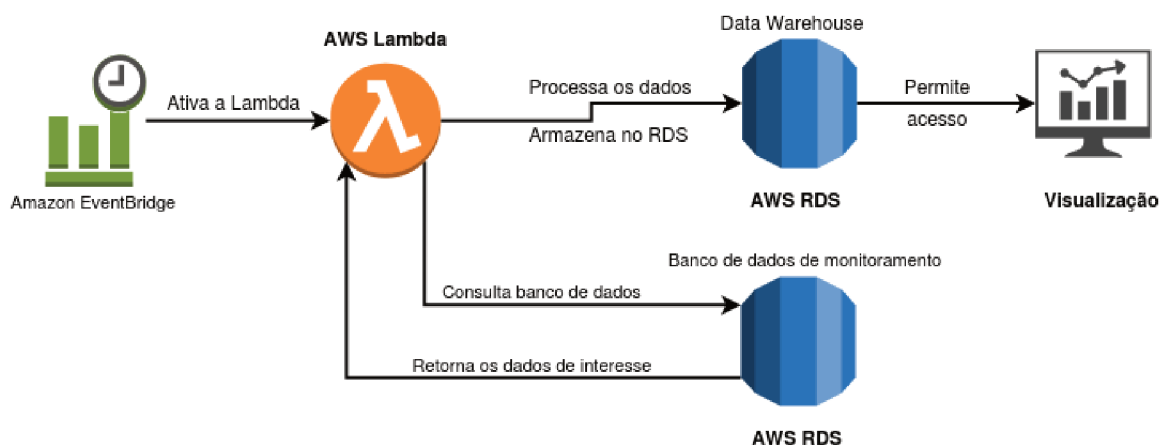
```
def get_dateSk(self,info):
    print("Veryfing date "+info["registerDate"]+" SK!")
    self.cur.execute("SELECT date_sk FROM dim_date WHERE \
        %s::timestamp::date = day_date::timestamp::date",[info["registerDate"]])
    query_result = self.cur.fetchall()
    if len(query_result) == 0:
        print ("There is something wrong with the date, could not find a matching"+
            "date on the date table dimesion!")
        return None
    print("DateSk is: "+ str(int(query_result[0][0])))
    return int(query_result[0][0])
```

Fonte – Elaborado pelo autor.

5.3.2 Desenvolvimento do segundo ETL - Registros dos alarmes

A arquitetura do fluxo de dados do segundo ETL é iniciado quando o serviço Amazon EventBridge aciona a execução do código ETL na Lambda, isso acontece diariamente em um horário programado. O código quando executado coleta informações do banco de dados de monitoramento da empresa e realiza a inserção destes no banco de dados do *data warehouse* hospedado no serviço AWS RDS. A arquitetura é apresentada de forma gráfica pela Figura 26.

Figura 26 – Fluxo de dados do segundo ETL.



Fonte – Elaborado pelo autor.

5.3.2.1 Os dados coletados

Os dados coletados por essa ETL tem sua origem no banco de dados de monitoramento da empresa. Onde lá contém todas as mensagens de monitoramento e alarmes enviados pelos computadores de bordo. Além disso tem todos os dados de cadastro de veículos, clientes, frentes de trabalho, entre outros. Em resumo, tem todas os dados utilizados para configuração das soluções de monitoramento da Hexagon, que tentam modelar o processo do cliente em seu ambiente digital.

5.3.2.2 Modelagem do Banco de Dados

Essa etapa de modelagem do banco de dados do Data Warehouse levou em consideração todos os requisitos de filtros de interesse para utilização nos *dashboards* levantados no Capítulo 4, além das informações de interesse levantados na Seção 5.2. Assim, adotando a modelagem dimensional do banco de dados, a tabela fato é constituída pelos registros de alarme e foi nomeada de `fact_alarm_history`. As dimensões são as informações que contém as características em que os registros de alarme ocorreram. As características selecionadas são citadas abaixo com os nomes dados as dimensões criadas e uma breve explicação:

- **Empresa - `dim_company`:** Tabela com as lista de todos os clientes que têm soluções de monitoramento e suas características básicas.
- **Veículo - `dim_vehicle`:** Tabela com a lista de todos os veículos cadastrados.
- **Tipo de veículo - `dim_vehicle_type`:** Tabela com a lista de todos os tipos de veículos cadastrados, onde tipo de veículo é uma classificação definida pelo cliente para agrupamento de máquinas. Normalmente utilizam-se as marcas e modelos em comum dos veículos.
- **Categoria de veículo - `dim_vehicle_category`:** Tabela com a lista das categorias dos veículos cadastrados, onde categoria de veículo é uma classificação definida pela Hexagon que agrupa os tipos de veículos em tratores e colhedoras, com o intuito de facilitar a análise dos dados de monitoramento visto que nem sempre os nomes dos veículos ou tipos de veículos são de fácil entendimento.
- **Frente de trabalho - `dim_workfront`:** Tabela com a lista de frentes de trabalho registradas pelo cliente. Onde frente de trabalho é a subdivisão da fazenda do cliente.
- **Data - `dim_date`:** Tabela com uma lista de datas e suas características como se é um dia da semana, qual dia do ano, entre outros. Indo de 01/01/2015 a 30/12/2025.

- **Computador de bordo - dim_onboard_computer:** Tabela com a lista de todos os computadores de bordo em operação e suas características básicas.
- **Tipo de alarme - dim_alarm_type:** Tabela com a lista de todos os tipos de alarmes existentes.
- **Tipo de alarme criado pelo cliente - dim_alarm_behavior:** Tabela com a lista de todos os alarmes do tipo 22. Que são os alarmes configurados pelo cliente.

Assim com a tabela fato e suas dimensões definidas foi possível criar a versão final do banco de dados do projeto, que é apresentado a seguir pela Figura 27. A tabela fato dos alarmes é identificada com um retângulo vermelho. As tabelas da etapa de desenvolvimento anterior são identificadas por um retângulo verde.

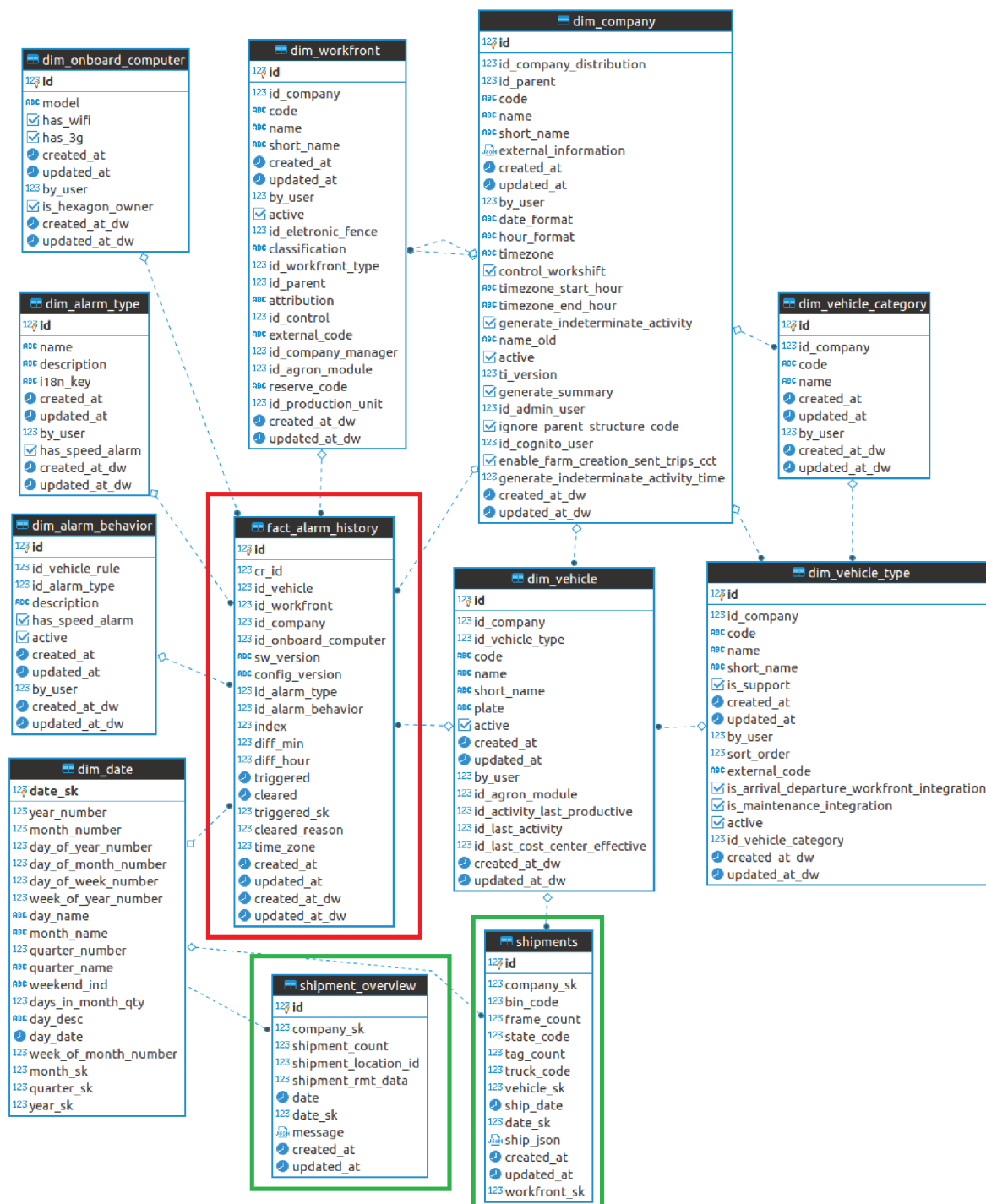
5.3.2.3 Código ETL - Lambda

O código do ETL foi desenvolvido na linguagem Python. Sua estrutura é similar ao ETL anterior, onde há uma função de administradora que é invocada pela Lambda e dois métodos que contém o restante das funcionalidades do código. Além disso, conta com nove arquivos JSON, um para cada uma das dimensões (com exceção da dimensão data) e uma para a tabela fato.

Os arquivos JSON contém os nomes e *schemas* das tabelas de origem e de destino dos dados, assim como as consultas em SQL necessárias para a execução da atualização de cada dimensão. As informações no arquivo JSON referente à dimensão dos tipos de alarme é apresentada a seguir pela Figura 28.

A primeira função a ser invocada pela Lambda é a função *lambda_handler*, apresentada pela Figura 29, que invoca a função *db_connect*, o qual realiza a conexão com bancos de dados. A mesma é invocada duas vezes em sequência, uma para criar a conexão com banco de dados de monitoramento da empresa e em seguida para abrir conexão com o banco de dados do *data warehouse*. Após a criação das conexões, é criado um objeto do tipo da classe *DimensionUpdater* e depois um objeto do tipo da classe *FactUpdate*, que realizam as operações necessárias para atualização das dimensões e tabela fato dos alarmes, respectivamente. A função que cria as conexões do banco de dados é idêntica à já apresentada no desenvolvimento do ETL anterior e por isso será omitida.

Figura 27 – Diagrama relacional do banco de dados completo.



Fonte – Elaborado pelo autor.

Figura 28 – Arquivo JSON com as consultas SQL referentes à dimensão alarme.

```
{
  "cr_schema": "alarm",
  "cr_table_name": "alarm_type",
  "dw_schema": "public",
  "dw_dim_table_name": "dim_alarm_type",

  "select_query": "select * from alarm.alarm_type where updated_at > %s;",

  "update_query": "UPDATE public.dim_alarm_type SET name=%(name)s, description=%
(description)s, i18n_key=%(i18n_key)s, created_at=%(created_at)s, updated_at=%
(updated_at)s, by_user=%(by_user)s, has_speed_alarm=%(has_speed_alarm)s, updated_at_dw=NOW
() WHERE id=%(id)s;",

  "create_query": "INSERT INTO public.dim_alarm_type
(id,name,description,i18n_key,created_at,updated_at,by_user,
has_speed_alarm,created_at_dw,updated_at_dw) VALUES (%(id)s, %(name)s, %(description)s, %
(i18n_key)s, %(created_at)s, %(updated_at)s, %(by_user)s, %(has_speed_alarm)s,NOW(),NOW
());"
}
```

Fonte – Elaborado pelo autor.

Figura 29 – Função *lambda_handler*.

```
def lambda_handler(event, context):

    conn_cr = db_connect(user_cr, password_cr, host_cr, port_cr, db_name_cr)
    cursor_cr = conn_cr.cursor(cursor_factory=psycopg2.extras.RealDictCursor)
    print("+++Connection to Control Room was Succsesfull!+++")

    conn_dw = db_connect(user_dw, password_dw, host_dw, port_dw, db_name_dw)
    cursor_dw = conn_dw.cursor(cursor_factory=psycopg2.extras.RealDictCursor)
    print("+++Connection to the Data Warehouse was Succsesfull!+++")

    update_dimensions = DimensionUpdater(conn_cr, cursor_cr, conn_dw, cursor_dw)
    update_fact = FactUpdate(conn_cr, cursor_cr, conn_dw, cursor_dw)
```

Fonte – Elaborado pelo autor.

No método construtor do objeto *update_dimensions*, Figura 30, é declarada uma lista dos nomes dos arquivos JSON de cada uma das dimensões (excluindo a dimensão data e computador de bordo). Para cada um dos nomes da lista a partir de um laço FOR, é realizada a leitura do arquivo JSON e invocado o método *update_dim*, passando as informações do arquivo JSON lido, utilizando um objeto do tipo dicionário nomeado de *info*. Ao fim do laço FOR, o método *update_onboard_computer_dim* é invocado para atualizar a dimensão *dim_onboard_computer*. A motivação da separação dessa dimensão das outras será apresentada ao decorrer da descrição do código.

Figura 30 – Método construtor da classe *DimensionUpdater*.

```
class DimensionUpdater:
    """Class responsible for executing all the dimensions tables update."""
    def __init__(self, conn_cr, cursor_cr, conn_dw, cursor_dw):

        self.conn_cr = conn_cr
        self.cursor_cr = cursor_cr
        self.conn_dw = conn_dw
        self.cursor_dw = cursor_dw

        dim_list = ["company", "vehicle_category", "vehicle_type", "vehicle", "workfront",
                    "alarm_type", "alarm_behaviour_config_new"]

        for dim in dim_list:
            with open('json/{}.json'.format(dim), 'r+') as json_file:
                data = json.load(json_file)
                print("\n-----Starting to update the {} dimension!-----"
                      .format(data["dw_dim_table_name"]))
                self.update_dim(data)
                print("-----Finished the {} update.-----".format(data["dw_dim_table_name"]))
        self.update_onboard_computer_dim()
        self.conn_dw.commit()
        print("\n+++Finished the Dimensions update.+++")
```

Fonte – Elaborado pelo autor.

O método *update_dim*, apresentado pela Figura 31, tem como função a atualização das dimensões padronizadas, sendo elas as que foram declaradas na lista de dimensões no construtor do objeto. Na sua execução é utilizado as informações do objeto *info* com as informações do arquivo JSON da dimensão que está sendo atualizada. O método inicia invocando os outros dois métodos *get_cr_last_update* e *get_dw_last_update*, apresentado pela Figura 32, que realizam consultas do registro com a data de atualização mais recente na tabela de origem no banco de dados de monitoramento e na tabela dimensão do Data Warehouse, respectivamente. Com essas informações é realizado uma comparação entre as datas. Caso a data da tabela dimensão seja maior ou igual ao da tabela de origem, a dimensão está atualizada, então a execução do código retorna para o *loop for* do método construtor. Caso contrário, é realizado uma consulta na tabela de origem para obtenção dos registros com data de atualização mais recente do que a data obtida na tabela dimensão.

Em seguida é processado cada registro individualmente para verificar se ele já existe na tabela dimensão ou se é um novo registro. Para os casos que já existem, é invocado o método *update_record* que realiza a atualização do registro, e nos casos de registros novos é invocado o método *create_record*, que realiza a inserção do registro na tabela dimensão. Esses dois métodos requerem dois parâmetros sendo eles: os dados do registro a ser processado, e do objeto dicionário que contém as informações do arquivo JSON da dimensão em questão.

Após a atualização das dimensões padronizadas, é realizada a atualização da dimensão referente aos computadores de bordo (*dim_onboard_computer*). A sua separação dos demais, se deve ao fato de que na tabela de origem no banco de dados de monitoramento o campo que armazena a data de atualização do registro é atualizado sempre que o display é ligado. Fazendo assim com que seja inviável utilizar a mesma arquitetura de atualização, pois quando realizado a consulta dos registros com a data de atualização mais recente que a data mais recente da tabela dimensão, quase todos os registros de computador de bordo são retornados.

Uma abordagem diferente foi adotada para a atualização dessa dimensão, o trecho de código é apresentado pela Figura 33. Uma consulta é realizada na tabela dimensão para obter todos os *id's*, que são o número de série dos computadores de bordo. Esse campo também é utilizado na tabela de origem que são valores únicos. Com a lista desses *id's*, é realizada uma consulta na tabela de origem para identificar quais *id's* não estão presentes nessa lista, retornando assim os registros de computador de bordo que não constam na tabela dimensão. Em seguida é realizada uma inserção desses registros na tabela dimensão do banco de dados do *data warehouse*.

Figura 31 – Método *update_dim*.

```
def update_dim(self, data):

    dim_last_update = self.get_dim_last_update(data)
    cr_last_update = self.get_cr_last_update(data)

    if dim_last_update >= cr_last_update:
        print("Dimension {} is already up to date.".format(data["dw_dim_table_name"]))
    else:
        print("Updating {}".format(data["dw_dim_table_name"]))
        cr_data = self.get_cr_data(data, dim_last_update)
        print("It has {} record(s) to update/insert!".format(str(len(cr_data))))
        row = 0
        for line in cr_data:
            print("\nProcessing {} result with ID number: {}".format(data["cr_table_name"], str(line["id"])))
            self.cursor_dw.execute(sql.SQL("SELECT * FROM {} where id = %(id)s;").format(
                sql.Identifier(data["dw_schema"], data["dw_dim_table_name"]), (line)))
            query_result = self.cursor_dw.fetchall()
            if len(query_result) == 0:
                self.create_record(line, data)
                print("Record {} was created!".format(str(row+1)))
            else:
                self.update_record(line, data)
                print("Record {} was updated!".format(str(row+1)))
            row += 1
```

Fonte – Elaborado pelo autor.

Figura 32 – Método `get_dim_last_update`.

```
def get_dim_last_update(self, data):

    print("Getting last DW update from " + data["dw_dim_table_name"])
    self.cursor_dw.execute(sql.SQL("SELECT max(updated_at)::timestamp without time zone \
        FROM {};" ).format(sql.Identifier(data['dw_schema'],\
        data['dw_dim_table_name'])))
    query_result = self.cursor_dw.fetchall()
    print("Dimension {} last update: {}".format(data["dw_dim_table_name"],\
        str(query_result[0]["max"])))
    return query_result[0]["max"]
```

Fonte – Elaborado pelo autor.

Figura 33 – Método `update_onboard_computer`.

```
def update_onboard_computer_dim(self):

    try:
        json_file = open('json/onboard_computer.json', 'r+')
        data = json.load(json_file)
        print("\n-----Starting to update the {} dimension!-----"\
            .format(data["dw_dim_table_name"]))
        print("Checking if {} needs to be updated.".format(data["dw_dim_table_name"]))
        self.cursor_dw.execute("SELECT id from public.dim_onboard_computer;")
        dw_data = self.cursor_dw.fetchall()
        # print(len(dw_data))
        array_serial = []
        for row in dw_data:
            array_serial.append(row["id"])
        self.cursor_cr.execute(data["select_query"], (tuple(array_serial),))
        query_result = self.cursor_cr.fetchall()
        if len(query_result) > 0:
            print("It has "+ str(len(query_result))+ " records to insert.")
            for row in query_result:
                row['model'] = row['model'].upper()
                print("Inserting Serial: {}".format(str(row['id'])))
                self.cursor_dw.execute(data["create_query"], row)
        else:
            print("dim_onboard_computer is up to date.")
        print("-----Finished the {} update.-----"\
            .format(data["dw_dim_table_name"]))
    except Exception as e:
        print("Onboard_computer_dim update failed - Exception: {}".format(str(e)))
```

Fonte – Elaborado pelo autor.

Ao fim da execução do método `update_onboard_computer_dim`, o construtor do objeto do tipo `DimensionUpdater` é finalizado. Em seguida é criado o objeto `update_fact`

do tipo *FactUpdate*, que contém todos os métodos que realizam a atualização da tabela fato.

O construtor do objeto *update_fact* invoca o método *fact_update*, apresentado pela Figura 34. Onde inicialmente realiza a importação dos dados do arquivo JSON referente à tabela fato e cria um objeto do tipo dicionário com as informações contidas nele. Em seguida é invocado o método *get_fact_last_update*, que realiza a consulta na tabela fato para obter a data de atualização mais recente entre todos os registros. Utilizando essa data, é realizado uma consulta ao banco de dados de monitoramento para obter os registros mais recentes que a data coletada.

Com base nestes registros, o método *delete_to_update* realiza uma consulta na tabela fato para identificação de quais registros já existem na tabela, mas que estão desatualizados. Essa desatualização se deve pelo fato de que há campos para o registro da data e hora do acionamento do alarme e outro para o desligamento do mesmo e algumas vezes esses alarmes demoram dias para serem desligados. Assim, o registro tem sua data de atualização modificada quando é registrado a hora de desligamento do alarme. Com os registros identificados eles são excluídos da tabela fato, para que então possam ser inseridos juntamente com os novos registros pelo método *insert_records*, apresentado pela Figura 35.

Figura 34 – Método *fact_update*.

```
def fact_update(self):
    """Read the json to get info needed and execute the sequence of
    functions needed to update fact table. At the end close all connections."""

    print("\n+++Starting to update fact table!+++")
    json_file = open('json/fact_table.json')
    info = json.load(json_file)
    fact_last_update = self.get_fact_last_update()
    print("Fact table last update date is {}".format(fact_last_update))
    data = self.get_data(info, fact_last_update)
    print("It has {} record(s) to update/insert!".format(str(len(data))))
    self.delete_to_update(data)
    self.insert_records(data)
    self.close_connections()
    print('Done!')
```

Fonte – Elaborado pelo autor.

Figura 35 – Método `insert_records`.

```

def insert_records(self, data):
    for row in data:
        if row['id_workfront'] is None:
            row['id_workfront'] = 0

    values = [self.cursor_dw.mogrify("(%(cr_id)s, %(id_vehicle)s, %(id_workfront)s,\
%(id_company)s, %(id_onboard_computer)s, %(sw_version)s, %(config_version)s,\
%(id_alarm_type)s, %(id_alarm_behavior)s, %(index)s, %(diff_min)s, %(diff_hour)s,\
%(triggered)s, %(cleared)s, %(triggered_sk)s, %(cleared_reason)s, %(time_zone)s,\
%(created_at)s, %(updated_at)s, Now(), Now())", row).decode('utf8') for row in data]
    chunks = (values[i:i + 20000] for i in range(0, len(values), 20000))
    print("Inserting records.")
    chunk_count = 1
    for chunk in chunks:
        print("Inserting {} chunk of data ({} records)".format(str(chunk_count),str(len(chunk))))
        chunk_count += 1
        query = "INSERT INTO public.fact_alarm_history (cr_id,id_vehicle,id_workfront,\
id_company, id_onboard_computer,sw_version,config_version,id_alarm_type,\
id_alarm_behavior,index,diff_min, diff_hour,triggered,cleared,triggered_sk,\
cleared_reason,time_zone,created_at,updated_at,created_at_dw,updated_at_dw)\
VALUES " + ", ".join(chunk)

        try:
            self.cursor_dw.execute(query)
        except Exception as e:
            print("Insert failed - Exception: {}".format(str(e)))
            self.conn_dw.rollback()
            self.close_connections()
            sys.exit()

    self.conn_dw.commit()
    print("Records were created succsesfully!\n")

```

Fonte – Elaborado pelo autor.

5.4 VISUALIZAÇÃO

A etapa de desenvolvimento das visualizações foi realizada utilizando o *software* Microsoft Power BI Desktop. Os motivos que levaram a escolha de utilização desse *software* foram principalmente: a sua licença gratuita, diminuindo assim o custo do projeto; e a sua ampla utilização no mercado, que implica na existência de uma grande comunidade ativa de usuários que desenvolvem e publicam materiais referentes ao *software*, o que facilita o aprendizado de sua utilização.

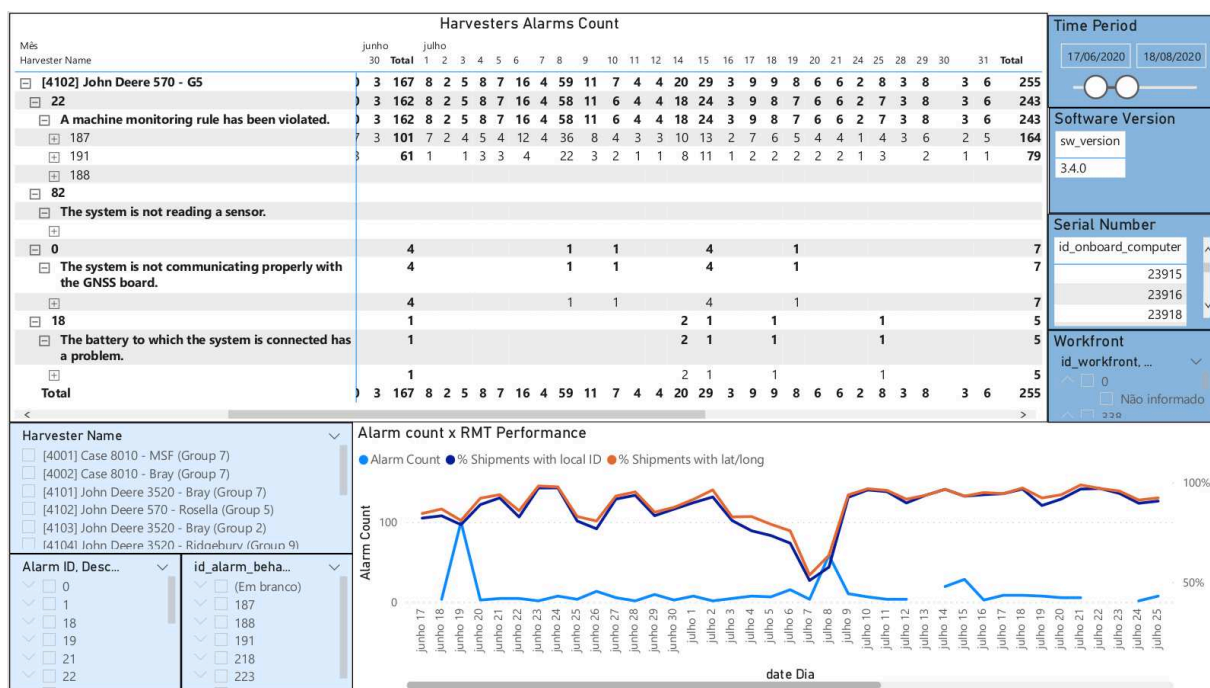
Para atender os requisitos levantados no Capítulo 4, foram desenvolvidos uma série de *dashboards* que apresentam as informações de interesse e que tem como objetivo auxiliar o time de implantação e suporte a analisar problemas em campo e acompanhar o desempenho da solução RMT. É importante salientar que para fins de análise exploratória de dados, foram desenvolvidos diversos *dashboards* com as mais diversas apresentações dos dados, mas para fins de documentação do projeto serão apresentados apenas os que preencheram os requisitos mínimos desejados.

5.4.1 Dashboard dos alarmes das colhedoras

O dashboard dos alarmes das colhedoras, apresentado pela Figura 36, tem como objetivo permitir ao usuário a visualização de qualquer tipo de alarme filtrando por período, versão de software embarcado, frente de trabalho e veículo, mas somente os veículos do tipo colhedora. A possibilidade de filtragem por esses parâmetros visa atender o pré-requisito funcional dos *dashboards* levantado no Capítulo 4.

A visualização se dá por forma de tabela, onde o usuário consegue visualizar a contagem dos alarmes; e de forma gráfica, onde a soma das contagens é plotada juntamente com os indicadores de desempenho utilizados pelo RMT, sendo eles a porcentagem de carregamentos com código identificador do local de origem e porcentagem de carregamentos com coordenadas do local de origem. O mesmo *dashboard* foi desenvolvido para os veículos do tipo trator, essa divisão dos *dashboards* por tipo de veículo foi julgada pertinente visto o impacto superior que as colhedoras tem no fluxo de dados da solução.

Figura 36 – Dashboard dos alarmes das colhedoras.



Fonte – Elaborado pelo autor.

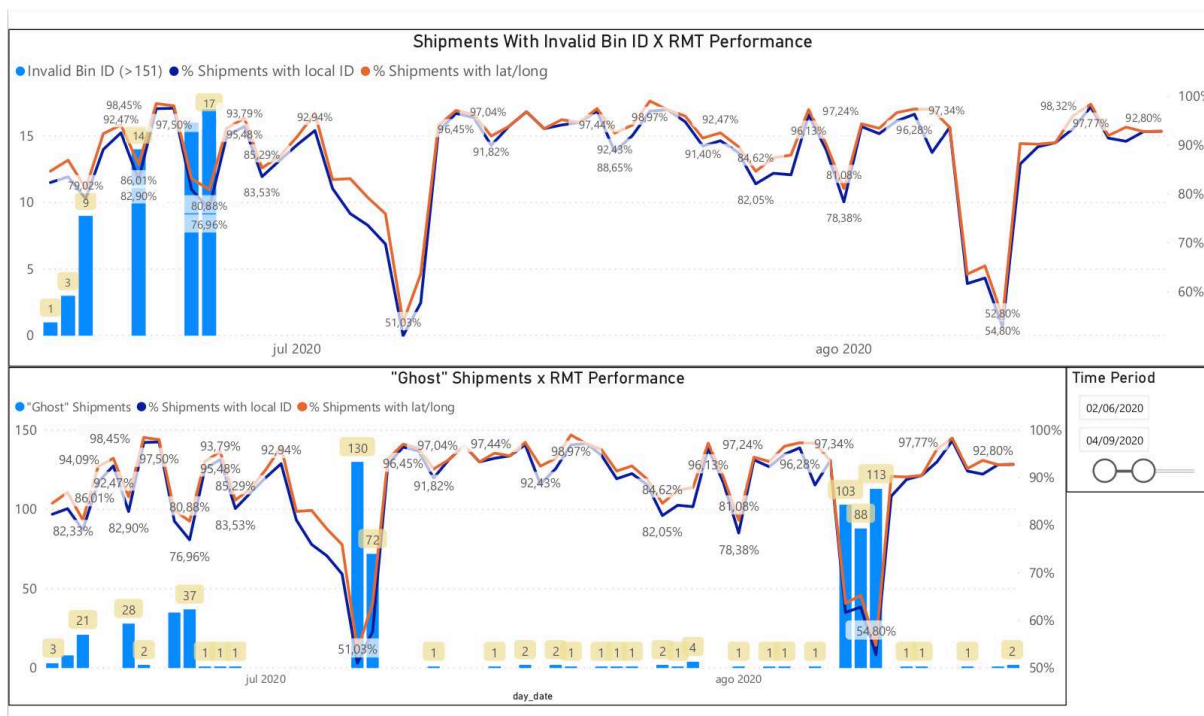
Com esse *dashboard* é possível realizar uma análise entre a queda de performance e a contagem de alarmes das colhedoras da frota. Essa é base para o suporte da solução, tentar identificar através dos alarmes dos veículos de forma remota o que acontece em campo. Com a ampla gama de alarmes existentes juntamente com a experiência do usuário, é possível realizar a identificação de problemas de *hardware*,

software e operacionais. Essa visualização condensa informações que demorariam horas para serem extraídas em apenas um *dashboard* de fácil e rápida atualização.

5.4.2 Dashboard dos problemas indicados pela balança

Esse *dashboard*, que é apresentado pela Figura 37, tem como intuito apresentar os dados de desempenho da solução acompanhados do dados referentes a dois problemas identificados durante o desenvolvimento do do projeto, que causaram um grande impacto no desempenho da solução. O intuito é de entregar dados de fácil interpretação para que seja possível um rápido diagnostico.

Figura 37 – Dashboard dos problemas indicados pela balança.



Fonte – Elaborado pelo autor.

- **O número de carregamentos que apresentam um código identificador inválido de Bin:** Os *bins* do cliente tem uma numeração de 1 a 151, caso algum *bin* seja lido pela balança com algum valor diferente, é assumido que a *Tag* RFID teve seus dados corrompidos e precisa ser reescrita. Essa informação é apresentada em forma de gráfico de barras pelo primeiro gráfico *dashboard*.
- **O número de carregamentos “fantasmas”:** Número de carregamentos que teve um intervalo de tempo inferior a sete minutos do seu antecessor, que foram chamados de carregamentos fantasmas. É sabido que o intervalo de tempo

médio entre a entrada do caminhão na balança e de sua saída é em torno de sete minutos, que é o tempo necessário para o caminhão realizar o descarregamento do material colhido na usina. Os carregamentos que apresentam o tempo de entrada na balança inferior a sete minutos após o seu antecessor, indicam um problema com chaveamento de leitura de *tags* corrompidas. Esse problema foi observado durante o desenvolvimento e será apresentado com maiores detalhes na Seção 5.6. A contagem de carregamentos fantasmas são apresentados em forma de gráfico de barras pelo segundo gráfico do *dashboard*.

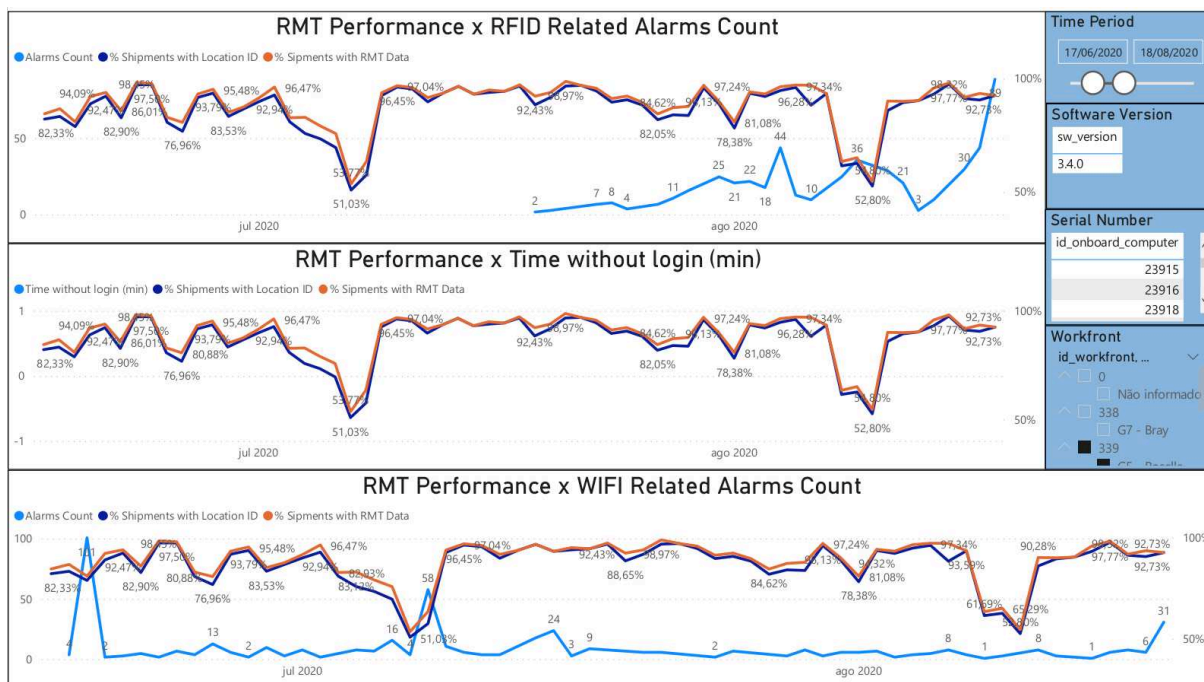
5.4.3 Dashboard das informações de interesse

Esse *dashboard* foi montado com o intuito de atender a demanda levantada pela etapa de levantamento das informações de interesse. São apresentados três gráficos, o primeiro plota a contagem do número de alarmes acionados que tem ligação com a tecnologia RFID da solução, o segundo plota a soma do tempo que o alarme de operador não logado ficou acionado e o terceiro plota a contagem do número de alarmes acionados que tem ligação com a tecnologia Wi-Fi da solução. Além disso todos os gráficos plotam os indicadores de desempenho do RMT junto com as informações, para que seja possível identificar a relação da contagem dos alarmes, que indicam a falta de *login* do operador ou problemas de conectividade de Wi-Fi ou RFID, com uma possível queda de desempenho. No canto direito do *dashboard* foi colocado um menu onde o usuário pode realizar o filtro por data, versão de software, número de serial do computador de bordo e frente de trabalho. O *dashboard* é apresentado pela Figura 38.

5.4.4 Dashboard informações das tags dos bins

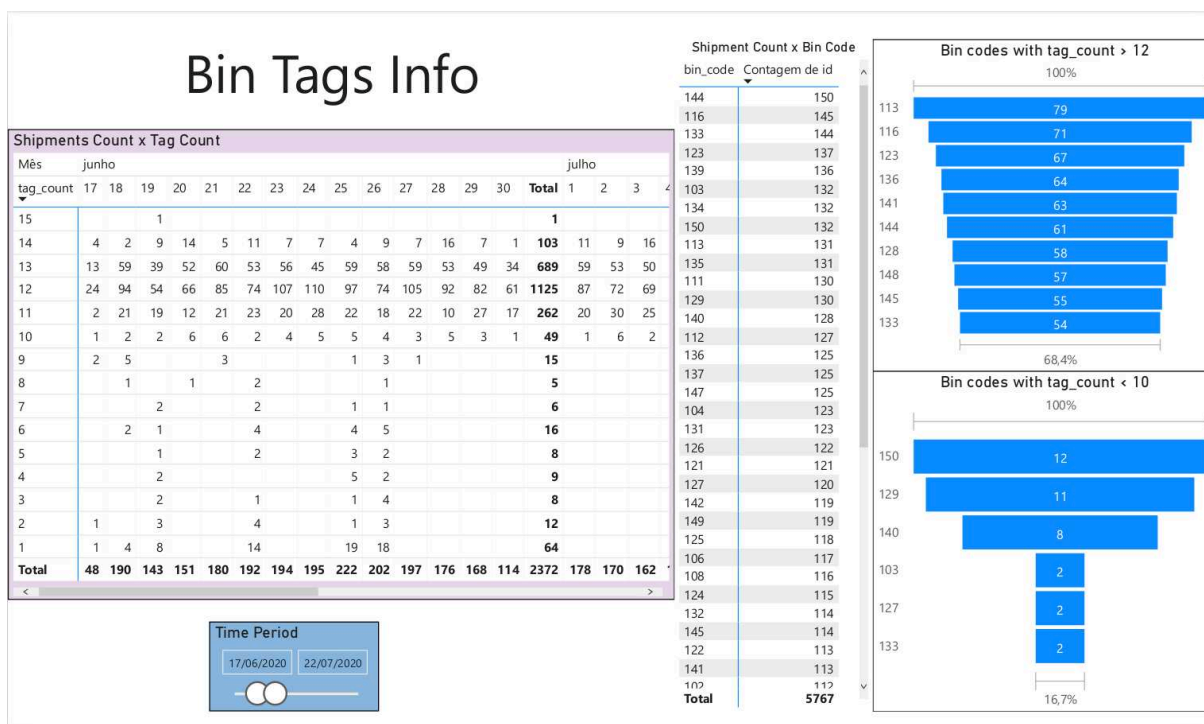
Esse dashboard apresenta quatro elementos gráficos, o primeiro é uma tabela que lista a contagem de carregamentos por quantidade de *tags* de *bin* lidas. O segundo é uma lista do total de carregamento que cada *bin* realizou no período selecionado, possibilitando uma análise de sub ou superutilização de uma determinada *bin*. A terceira aponta as dez *bins* que tiveram o maior número de carregamentos que apresentaram mais de doze tag lidas pela balança, indicando um bom funcionamento dessas *tags*. O quarto aponta os seis *bins* que apresentaram a maior contagem de carregamentos com menos de dez *tags* lidas, o que pode indicar algum problema com as *tags* dessas *bins* em específico. Nenhum dos elementos gráficos desse dashboard leva em consideração os carregamentos ditos fantasmas apresentados anteriormente, pois os mesmos não são carregamentos válidos. O dashboard é apresentado a seguir pela Figura 39.

Figura 38 – Dashboard das informações de interesse.



Fonte – Elaborado pelo autor.

Figura 39 – Dashboard informações das tags dos bins.



Fonte – Elaborado pelo autor.

Foi determinado durante logo no início do desenvolvimento do projeto que seria de grande importância o compartilhamento das informações coletadas com o cliente, mas como a etapa de desenvolvimento das visualizações só foi concluída ao final do projeto, foi desenvolvido uma planilha eletrônica para que um relatório fosse enviado ao cliente de forma diária. O relatório, apresentado pela Figura 40, que serviu como base para o desenvolvimento dos *dashboards* já apresentados, possui quatro elementos gráficos principais. O primeiro é a apresentação da quantidade de carregamentos total por dia e o desempenho da solução RMT - porcentagem de carregamentos com código identificador do local de origem e porcentagem de carregamentos com coordenadas do local de origem. O segundo é a apresentação da contagem de carregamentos pela quantidade de *tag* lidas na balança. O terceiro e quarto elemento são a contagem dos alarmes naturais do software embarcado e os alarmes configurados, respectivamente.

Como o relatório apresentou resultados positivos com a sua utilização e a o cliente já o havia validado, decidiu-se manter o envio do relatório para o cliente, mesmo com a conclusão do desenvolvimento dos *dashboards*, que no caso são utilizados pelo time de suporte quando necessário.

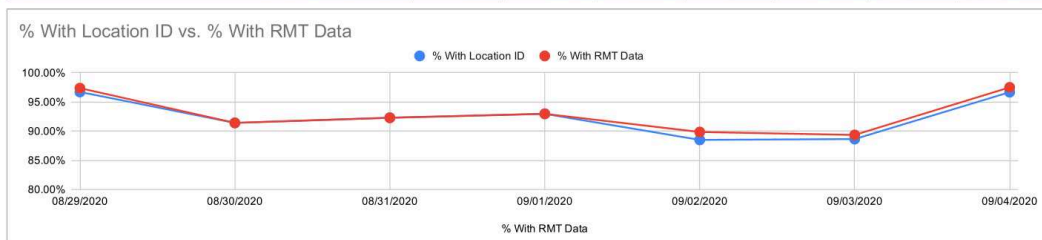
Figura 40 – Relatório enviado diariamente ao cliente.



HEXAGON

MSF - RMT - Status Report

Description	08/29/2020	08/30/2020	08/31/2020	09/01/2020	09/02/2020	09/03/2020	09/04/2020
Shipments Count	153	35	156	171	148	141	121
Shipments with location ID	148	32	144	159	131	125	117
Shipments with RMT data	149	32	144	159	133	126	118
% With Location ID	96.73%	91.43%	92.31%	92.98%	88.51%	88.65%	96.69%
% With RMT Data	97.39%	91.43%	92.31%	92.98%	89.86%	89.36%	97.52%



Shipment Tag Count	08/29/2020	08/30/2020	08/31/2020	09/01/2020	09/02/2020	09/03/2020	09/04/2020
13 Tags	1			1	2	4	2
12 Tags	137	33	142	142	121	121	108
11 Tags	9	2	12	19	21	11	10
10 Tags	3		1	6	3	5	1
9 Tags	2			1			
8 Tags			1	2	1		
7 Tags							
6 Tags							
5 Tags							
4 Tags							
3 Tags							
2 Tags	1						
1 Tags							
0 Tags							

Alarm ID	Description	08/29/2020	08/30/2020	08/31/2020	09/01/2020	09/02/2020	09/03/2020	09/04/2020
0	No GNSS communication	6	6	2	3	3	1	3
1	Synchronizing GNSS	3	1	1				1
16	A monitoring controller configured for this implement is not connected.							
18	Low voltage	2	3	2	9	12	8	10
19	High voltage			1			1	1
21	Machine monitor - incorrect task	48	30	22	58	72	52	68
22	Rule violation	317	187	103	179	237	259	248
29	Speed Limit	3554	165	2128	2871	2470		
30	Virtual fence	5		7	6	16		
82	Sensor	2959	1743	2169	3237	3130	2777	2749
67	Login					1		1
100	Invalid route							
108	Error evaluating rule							
110	Internal wifi error (password incorrect)							
114	Rfid reader (There is not enough free memory to save the data)	1		1	1			1
115	Unable to send data to the cloud due to poor connection.							
116	Rfid reader (failure writing the data to the tag.)	4	3	1	8	5	8	3
117	The software responsible for reading and writing the tags is not running properly.							
118	Rfid reader (searching for a tag, but theres no information to be stored)	2	1	1		1	3	
119	Rfid reader (searching for a tag, but theres no available tag.)	33	22	13	21	31	27	21

Alarm 22 Rule	Description	08/29/2020	08/30/2020	08/31/2020	09/01/2020	09/02/2020	09/03/2020	09/04/2020
187	Not paired to the haul-out tractor while harvesting.	87	67	49	89	61	50	58
188	Paired to the haul-out tractor but not harvesting.	1		4		1	2	1
191	Not paired to the haul-out tractor while harvesting.	57	37	25	50	40	17	19
218	Beside bin for over 18s but not unloading.Issue with unloading detection.	49	30	21	26	45	40	63
223	Potential issue with haulout, unloading not detected!	61	21	2	7	44	71	55
246	Potential issue with RFID feature, not reading any RFID tags when expected.	62	32	2	7	46	79	52

Note: The dates are in the UTC +00:00 time zone.

Fonte – Elaborado pelo autor.

5.5 CÓDIGO DE ANÁLISE DE DADOS

Com o intuito de capacitar o usuário a utilizar os dados com maior eficiência e clareza, foi desenvolvido um código que realiza a análise correlacional dos dados, ajudando o usuário a quantificar a influência de um determinado problema em campo, indicado por um alarme ou outro dado, no indicador de desempenho da solução RMT.

Com o auxílio dos líderes do time de suporte foi elaborada uma lista dos alarmes e dados mais relevantes e a partir deles foi desenvolvido o código. A lista dos alarmes e dados é apresentada pela Tabela 3 juntamente com a descrição de cada um dos dados selecionados. No caso dos alarmes duas análises são realizadas, uma em relação a correlação da contagem dos alarmes e outra a correlação do tempo em que os alarmes ficaram acionados.

O código foi desenvolvido em Python e teve sua estrutura baseada nos códigos de ETL desenvolvidos anteriormente, onde uma classe principal detém todas as funções necessárias para a execução do código e um objeto do tipo dessa classe é criado em uma função *main*.

A função *main* do código realiza a invocação de uma função que cria a conexão com o banco de dados do *data warehouse*. Em seguida cria um objeto do tipo da classe *Analytics* passando como parâmetros a conexão e o cursor criados. O construtor do objeto realiza a invocação do método *execute_analytics*, o qual é apresentado pela Figura 41. Esse método tem como função realizar a consulta dos dados de desempenho, criação de um objeto *dataframe* com esses dados e de realizar a invocação ordenada dos outros métodos. Sendo o primeiro deles o método *count_correlation*.

O método *count_correlation* apresentado no Apêndice A, realiza a consulta no banco de dados do *data warehouse* para obter a contagem dos alarmes, dos tipos listados anteriormente, acionados no período de interesse. Em seguida concatena esses dados em colunas no *dataframe* passado como parâmetro e calcula os coeficientes de correlação de forma matricial, todas as colunas por todas as colunas. São calculados três tipos de coeficientes de correlação: Pearson, Spearman e Kendall, todos utilizando o método “corr” da biblioteca Pandas.

Após o cálculo dos coeficientes é realizado o cálculo do valor P para o coeficiente de Pearson utilizando a biblioteca *scipy* através da invocação do método *p_value_perm_test* pelo método *corr* já mencionado anteriormente.

Por fim os gráficos e tabelas são plotados utilizando múltiplos métodos, onde todos utilizam da biblioteca *matplotlib* para a criação das imagens; e é retornado um objeto *dataframe* com os valores dos coeficientes de correlações de Pearson entre os alarmes e o indicador da porcentagem de carregamentos com coordenadas de origem e seus respectivos valores P.

Tabela 3 – Dados utilizados pelo código de análise correlacional.

Nome	Alarme ID	Alarme 22-Index	Descrição
percent_rmt_data			Porcentagem de carregamentos com informações de coordenadas da origem do material.
percent_id_local			Porcentagem de carregamentos com o identificador do talhão de origem do material.
alarm_67	67		Alarme que indica que o operador não realizou <i>login</i> no computador de bordo.
alarm_102	102		Alarme que indica problemas no <i>Workflow</i> . (Regras e atividades que determinam o estado de cada atividade do veículo)
alarm_114	114		Alarme que indica que falta memória na <i>tag</i> RFID.
alarm_116	116		Alarme que indica que ocorreu um erro ao escrever informações na <i>tag</i> RFID.
alarm_117	117		Alarme que indica que o software que escreve e lê as <i>tags</i> RFID não está respondendo.
alarm_118	118		Alarme que indica que indica que o sistema tentou encontrar uma <i>tag</i> para escrever mas não há informação para ser escrita.
alarm_119	119		Alarme que indica que indica que o sistema tentou encontrar uma <i>tag</i> para escrever mas não consegue encontrar nenhuma.
index_187	22	187	Alarme que indica que a colhedora está colhendo mas não está pareada com o trator por mais de 30 segundos, possível problema de Wi-Fi.
index_188	22	188	Alarme que indica que a colhedora está pareada com o trator mas não está colhendo, possível problema no sensor do elevador de cana.
index_191	22	191	Alarme que indica que a colhedora está colhendo mas não está pareada com o trator por mais de 60 segundos, possível problema de Wi-Fi.
index_218	22	218	Alarme que indica que o trator está do lado de uma <i>bin</i> mas não está realizando o descarregamento da cana.
index_223	22	223	Alarme que indica que o trator não apresentou a atividade de descarga da cana quando deveria.
index_246	22	246	Alarme que indica algum possível problema com o sistema de RFID.
fast_shipments			Número de carregamentos que foram registrados com um tempo inferior a 5 minutos depois do último carregamento, indicando que é um carregamento "fantasma" e que há algum problema.
invalid_bin_code			Carregamentos com código inválido do <i>bin</i> .
tag_count			Numero de <i>tags</i> RFID do <i>bin</i> que as antenas da balança conseguiram realizar a leitura.

Fonte – Elaborado pelo autor.

Figura 41 – Método `execute_analytics`.

```

def execute_analytics(self):

    self.get_dates()
    self.cursor_dw.execute("SELECT shipment_count, shipment_location_id, shipment_rmt_data,\
                            date::date, date_sk from public.shipment_overview where\
                            company_sk = 50067 and date > %s and date < %s;", (self.initial_date,\
                            self.final_date))

    query_result = self.cursor_dw.fetchall()
    perf_df = pd.DataFrame([i.copy() for i in query_result])
    perf_df = perf_df.sort_values(by = 'date_sk')
    perf_df.reset_index(drop=True, inplace=True)

    #Calculate the performace (%)
    perf_df['percent_rmt_data'] = perf_df['shipment_rmt_data']/perf_df['shipment_count']
    perf_df['percent_id_local'] = perf_df['shipment_location_id']/perf_df['shipment_count']
    perf_df['date'] = perf_df['date'].astype('datetime64')
    candidates_count = self.count_correlation(perf_df)
    candidates_time = self.time_correlation(perf_df)
    candidates_wheightbridge = self.shipments_issues_correlations(perf_df)
    self.best_fit_candidates(candidates_count,candidates_time,candidates_wheightbridge)
    pdf_output.close()
    # plt.show()
    print("Done.")

```

Fonte – Elaborado pelo autor.

Essa mesma estrutura é repetida pelo método `shipments_issues_correlations` e `time_correlation`, onde são analisados os dados referentes a soma do período de tempo em que ficaram ativos e os dados referentes a problemas relacionados a balança, respectivamente.

Em seguida é realizada a concatenação dos `dataframes` retornados por cada um dos três métodos mencionados e realizado uma ordenação dos valores do coeficiente de Pearson e dos valores P. Assim, possibilitando a listagem dos dados que tiveram a maior correlação com o indicador de desempenho da solução, podendo-se interpretar como influência no indicador. Ao fim, todos os gráficos e tabelas são salvos em um arquivo de formato PDF. Os gráficos das análises de contagem de alarmes e dos problemas relacionados a balança são apresentados estão apresentadas Apêndice B.

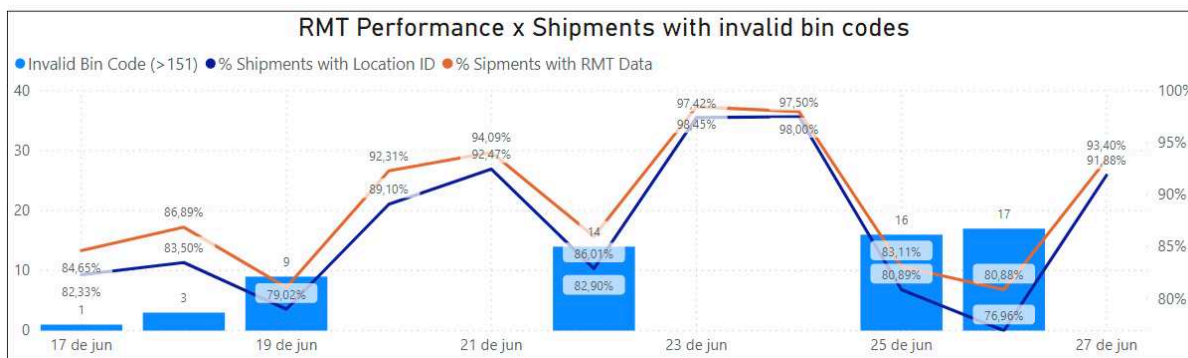
5.6 VALIDAÇÃO

A validação da ferramenta se deu através da utilização da mesma para acompanhar o desempenho da solução diariamente e identificação de problemas em campo de forma remota a partir dos dados do Data Warehouse. Três casos em que a ferramenta apresentou resultados práticos serão apresentados a seguir:

5.6.1 Caso 1 - Códigos inválidos de bins

No início da safra durante os acompanhamentos dos primeiros gráficos gerados, foi identificado quedas pontuais de desempenho da solução RMT, conforme indicado pela Figura 42.

Figura 42 – Gráfico de desempenho RMT pela quantidade de carregamentos com código de bin inválido.



Fonte – Elaborado pelo autor.

Inicialmente não era sabido a motivação das quedas, a suspeita de problemas de RFID ou de Wi-Fi foi descartado visto que a contagem dos alarmes não apresentavam números expressivos. Assim, foram analisados os registros de carregamentos da tabela *shipments* dos dias que apresentaram as quedas. Com essa análise foi possível perceber que haviam carregamentos com códigos de identificação do bin fora do padrão (entre 1 - 151), conforme é possível ver na Figura 43 na coluna bin_code. Além disso, foi possível perceber que os carregamentos com problema sempre apareciam seguidos de carregamentos com o bin 115, de forma que eles se revezam com tempos inferiores a sete minutos, que é o tempo mínimo estimado entre carregamentos.

Figura 43 – Tabela *shipments* - tags corrompidas.

	id	company_sk	bin_code	frame_count	state_code	tag_count	truck_code	vehicle_sk	ship_date
340	340	50,067	115	0	2	3	1,009	0	2020-06-19 17:18:06
341	341	50,067	1,098,908,929	0	2	1	1,009	0	2020-06-19 17:18:08
342	342	50,067	115	3	2	7	1,014	0	2020-06-19 17:22:15
343	343	50,067	1,098,908,929	0	1	1	1,014	0	2020-06-19 17:23:09
344	344	50,067	115	0	2	2	1,014	0	2020-06-19 17:24:28
345	345	50,067	1,098,908,929	0	2	1	1,014	0	2020-06-19 17:24:57
346	346	50,067	115	0	2	4	1,014	0	2020-06-19 17:25:17
347	347	50,067	1,098,908,929	0	2	1	1,014	0	2020-06-19 17:25:22
348	348	50,067	115	0	2	6	1,014	0	2020-06-19 17:26:59

Fonte – Elaborado pelo autor.

Com essas informações foi possível determinar que uma das tags do bin 115 estava corrompida. O que foi confirmado pelo técnico do cliente, que realizou a veri-

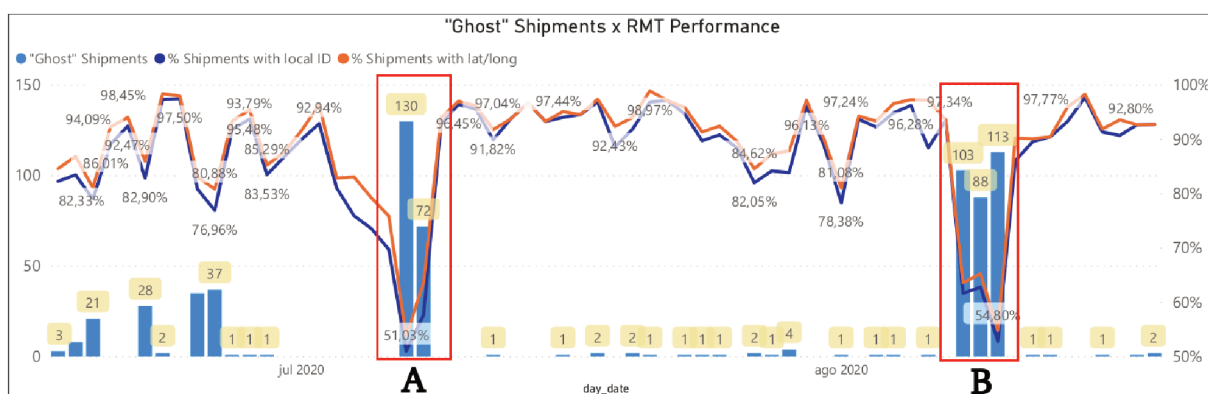
ficação em campo e a regravação do código identificador correto. Com isso as *tags* inválidas começaram a ser monitoradas para que a identificação da origem pudesse ocorrer com o menor tempo possível e não afetar a operação do cliente.

5.6.2 Caso 2 - Tag RFID do caminhão corrompida

Foi notado uma queda de desempenho indicada pela letra A na Figura 44, com isso foi realizado o procedimento padrão até o momento: verificar os alarmes e os códigos de *bin* inválidos. O qual nenhum deles apresentou nenhuma anormalidade, e com isso foi necessário recorrer novamente aos registros dos carregamentos.

Com a análise dos registros foi possível observar que havia um caminhão que estava apresentando uma série de carregamentos com tempo menores que um minuto entre si. Inicialmente a suspeita era de que fosse algum problema com as *tags* do *bin*, pois devido ao histórico do início da safra, era o mais provável. O que foi parcialmente confirmado pelos códigos de identificação dos bins (*bin_code*) que eram registrados nesses carregamentos, conforme é possível observar na Figura 45 e Figura 46, o *bin* 150 era recorrente nos carregamentos. Entretanto, o mesmo era sempre acompanhado de um outro *bin* em que todas as vezes era diferente. A questão foi amplamente discutida pelos desenvolvedores para interpretar esse comportamento e identificar o problema. Muitas teorias foram levantadas, mas uma foi eleita como a mais provável.

Figura 44 – Gráfico de desempenho RMT pela quantidade de carregamentos ditos fantasmas.



Fonte – Elaborado pelo autor.

A teoria era de que uma das *tags* de identificação do caminhão, por algum motivo, havia sido reescrita como uma *tag* de *bin*, mais especificamente *bin* 150, fazendo com que quando o caminhão entrasse na balança o sistema iria ler uma *tag* de caminhão e *tags* de dois *bins* diferentes, sendo um deles a *tag* reescrita do caminhão. Assim o sistema realizava o chaveamento entre os *bins* identificados, criando múltiplos carregamentos fantasmas.

Figura 45 – Tabela *shipments* - carregamentos fantasmas *bin* 117.

	123 id	123 company_sk	123 bin_code	123 frame_count	123 state_code	123 tag_count	123 truck_code	123 vehicle_sk	ship_date
3807	3,807	50,067	150	0	2	1	1,005	0	2020-07-08 18:25:03
3808	3,808	50,067	117	0	2	2	1,005	0	2020-07-08 18:25:37
3809	3,809	50,067	150	0	2	1	1,005	0	2020-07-08 18:29:51
3810	3,810	50,067	117	0	2	1	1,005	0	2020-07-08 18:30:00
3811	3,811	50,067	150	0	2	1	1,005	0	2020-07-08 18:30:18
3812	3,812	50,067	117	0	2	2	1,005	0	2020-07-08 18:30:27
3813	3,813	50,067	150	0	2	1	1,005	0	2020-07-08 18:30:33
3814	3,814	50,067	117	0	2	2	1,005	0	2020-07-08 18:31:28
3815	3,815	50,067	150	0	2	1	1,005	0	2020-07-08 18:31:31
3816	3,816	50,067	117	0	2	10	1,005	0	2020-07-08 18:32:06
3817	3,817	50,067	150	0	1	1	1,005	0	2020-07-08 18:33:54

Fonte – Elaborado pelo autor.

A teoria foi confirmada quando o técnico da empresa fez a verificação das *tags* do caminhão 1005 e uma delas estava gravada como uma *tag* de *bin* com código identificador 150. Assim a *tag* foi reescrita e o desempenho voltou ao normal.

Para esse caso a ferramenta proporcionou um ganho de tempo considerável, visto que sem ela seriam necessárias várias análises manuais dos *logs* do serviço da balança no servidor local do cliente, tomando em torno de quinze dias, conforme estimativa de um dos desenvolvedores *senior* da empresa, e com a ferramenta o problema foi identificado e resolvido em dois dias.

Figura 46 – Tabela *shipments* - carregamentos fantasmas *bin* 139.

	123 id	123 company_sk	123 bin_code	123 frame_count	123 state_code	123 tag_count	123 truck_code	123 vehicle_sk	ship_date
3589	3,589	50,067	150	0	2	1	1,005	0	2020-07-07 21:52:46
3590	3,590	50,067	139	0	2	2	1,005	0	2020-07-07 21:53:21
3591	3,591	50,067	150	0	2	1	1,005	0	2020-07-07 21:53:26
3592	3,592	50,067	139	0	2	1	1,005	0	2020-07-07 21:53:51
3593	3,593	50,067	150	0	2	1	1,005	0	2020-07-07 21:54:29
3594	3,594	50,067	139	0	2	3	1,005	0	2020-07-07 21:55:05
3595	3,595	50,067	150	0	2	1	1,005	0	2020-07-07 21:56:16
3596	3,596	50,067	139	0	2	1	1,005	0	2020-07-07 21:56:20
3597	3,597	50,067	150	0	2	1	1,005	0	2020-07-07 21:56:30
3598	3,598	50,067	139	0	2	3	1,005	0	2020-07-07 21:56:59
3599	3,599	50,067	150	0	2	1	1,005	0	2020-07-07 21:57:06
3600	3,600	50,067	139	0	2	2	1,005	0	2020-07-07 21:57:19
3601	3,601	50,067	150	0	2	1	1,005	0	2020-07-07 21:57:35
3602	3,602	50,067	139	0	2	1	1,005	0	2020-07-07 21:57:40
3603	3,603	50,067	150	0	2	1	1,005	0	2020-07-07 21:57:53

Fonte – Elaborado pelo autor.

Os carregamentos fantasmas, os quais são apresentados nos *dashboards* e usados no código de análise correlacional de dados, foi descoberto nesse momento, antes não se tinha registro desse tipo de falha.

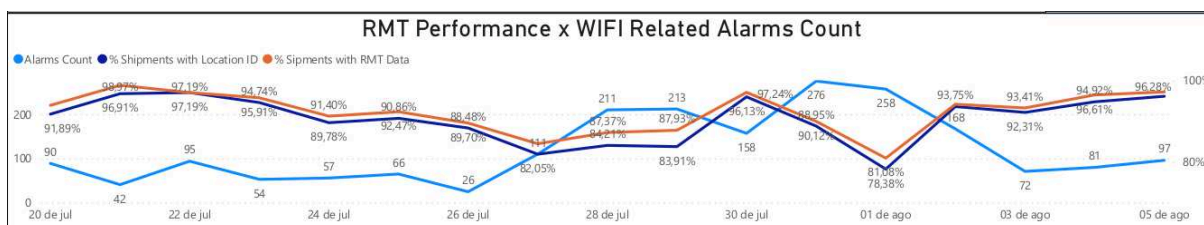
O problema se repetiu aproximadamente um mês depois em um caminhão diferente, indicado pela letra B na Figura 44. Como o problema já era conhecido e já havia um gráfico acompanhando os carregamentos fantasmas, a identificação do problema foi mais fácil. Infelizmente o tempo de identificação e resolução nesse caso

foi de 3 dias, pois os carregamentos fantasmas só começaram a aparecer em uma sexta-feira e acabou sendo identificado somente na segunda-feira da semana seguinte.

5.6.3 Caso 3 - Queda de performance por problemas de pareamento

Foi identificado uma queda no desempenho da solução RMT entre os dias 27 e 29, juntamente com um aumento da contagem de alarmes relacionados a problemas com pareamento Wi-Fi. A queda e a contagem dos alarmes são apresentados pela Figura 47.

Figura 47 – Gráfico de desempenho da solução RMT pela contagem de alarmes de problemas de pareamento Wi-Fi.



Fonte – Elaborado pelo autor.

Através da verificação da exploração dos dados no *dashboard* dos alarmes das colhedoras, foi possível realizar a identificação das colhedoras que apresentaram uma maior quantidade de alarmes relacionados a problemas com pareamento Wi-Fi acionados no período, apresentadas pela figura Figura 48. Com essa informação uma manutenção foi realizada somente nas duas colhedoras, diminuindo assim o impacto na operação e o tempo de suporte.

Figura 48 – Colhedoras com maior número de alarmes de problemas de pareamento Wi-Fi.

Mês	Harvester Name	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Tc
juho																																
	[4101] John Deere 3520 - G7	85	207	191	106	165	111	6	10	44	20	16	7	10	2	10	14	8	16	9	41	10	17	16	13	10	7	79	164	16	15	17
	187	51	111	101	57	88	62	4	9	24	15	11	6	7	2	8	10	8	10	8	24	8	13	10	7	7	5	43	88	10	9	12
	Not paired to the haul-out tractor while harvesting, there might be an issue with the Wi-Fi antenna	51	111	101	57	88	62	4	9	24	15	11	6	7	2	8	10	8	10	8	24	8	13	10	7	7	5	43	88	10	9	12
	191	34	96	90	49	77	49	2	1	20	5	5	1	3	2	4	6	1	17	2	4	6	6	3	2	36	76	6	6	5		
	Not paired to the haul-out tractor while harvesting, there might be an issue with the Wi-Fi antenna	34	96	90	49	77	49	2	1	20	5	5	1	3	2	4	6	1	17	2	4	6	6	3	2	36	76	6	6	5		
	[4202] John Deere 570 - G2	7	10	9	8	7	8	4	3	4	2	6	6	37	12	7	51	10	12	12	12	4	6	17	3	5	1	6	159	111	132	
	187	5	7	6	7	6	7	3	3	3	2	5	4	19	8	4	27	7	8	7	8	3	4	10	3	3	1	4	83	60	69	
	Not paired to the haul-out tractor while harvesting, there might be an issue with the Wi-Fi antenna	5	7	6	7	6	7	3	3	3	2	5	4	19	8	4	27	7	8	7	8	3	4	10	3	3	1	4	83	60	69	
	191	2	3	3	1	1	1	1	1	1	1	2	18	4	3	24	3	4	5	4	1	2	7	2	2	2	2	76	51	63		
	Not paired to the haul-out tractor while harvesting, there might be an issue with the Wi-Fi antenna	2	3	3	1	1	1	1	1	1	1	2	18	4	3	24	3	4	5	4	1	2	7	2	2	2	2	76	51	63		
	Total	92	217	200	114	172	119	10	13	48	22	22	13	47	2	22	21	59	26	21	53	22	21	22	30	13	12	80	170	175	126	149

Fonte – Elaborado pelo autor.

6 CONCLUSÃO

6.1 O QUE FOI FEITO

Resumindo os pontos do trabalho:

- No capítulo 2 foi apresentada a empresa em que o projeto foi desenvolvido.
- No capítulo 3 foi realizada uma revisão bibliográfica dos conceitos teóricos utilizados, apresentação das ferramentas e tecnologias empregadas no desenvolvimento do projeto e uma descrição das tecnologias Hexagon em que o projeto se baseou.
- No capítulo 4 foi apresentado os requisitos e especificações levantadas para cada uma das partes do projeto, além da modelagem do funcionamento dos códigos e da relação das partes do sistema, dando uma visão do funcionamento da ferramenta como um todo.
- No capítulo 5 foi apresentado as etapas de desenvolvimento do projeto e a validação do mesmo.

6.2 COMPARAÇÃO DOS RESULTADOS E OBJETIVOS

Comparando cada um dos objetivos traçados no capítulo 1 com os resultados obtidos temos:

- **Capacidade de acompanhar o desempenho da solução de forma diária:** Esse objetivo foi alcançado com o desenvolvimento do data warehouse, que realiza a coleta de forma automática das informações fazendo com que o tempo de consulta e análise torne factível a consulta diária das informações de desempenho.
- **Identificar possíveis problemas de software, hardware ou operacionais em campo:** Como apresentado na seção 5.6, vários casos de problemas em campo foram detectados com a utilização dos dashboards e bancos de dados desenvolvidos no projeto. Com isso podemos afirmar que o objetivo foi alcançado.
- **Quantificar a relevância dos problemas encontrados:** Com o código de análise correlacional de dados é possível realizar um cálculo do coeficiente de correlação da frequência de acionamento de alarmes com o indicador de desempenho da solução, com esses coeficientes é possível gerar uma lista dos alarmes que tem um maior impacto no desempenho da solução, assumindo assim uma relevância maior dentre os demais.

- **Apontar ou ao menos restringir as possíveis causas dos problemas:** Como apresentado na seção 5.6, diversos casos em que a fonte ou a causa da queda do desempenho eram desconhecidas, foram solucionados com o apoio dos dashboards desenvolvidos, os quais não indicaram com precisão o que estava acontecendo, mas conseguiram dar ao usuário uma direção de onde e o que investigar em campo.

6.3 ANÁLISE

A partir do trabalho desenvolvido, foi possível arquitetar uma ferramenta que atendesse aos objetivos gerais do projeto de forma satisfatória. Os objetivos específicos foram alcançados conforme o previsto, tendo seu desenvolvimento focado nos requisitos levantados. Uma ressalva deve ser feita referente ao código de análise de dados, que devido ao pouco tempo dedicado para o seu desenvolvimento, não foi possível atingir de forma satisfatória o requisito não funcional de escalabilidade do mesmo, devido a sua estrutura com modularidade inferior à desejada. Além disso, não houve uma etapa de validação dos resultados do código, fazendo com que os resultados fiquem inutilizáveis na prática; e as técnicas e conceitos aplicados podem não ser as mais adequadas. O código deverá ser melhorado e seus resultados analisados por uma etapa de validação em uma próxima versão.

Com a utilização da ferramenta, foi evidenciada uma diminuição do período de tempo entre a detecção e a resolução de problemas em campo. Permitindo uma considerável redução do custo de suporte. Também permitiu maior capacitação da área de suporte em relação às informações, pois eliminou a dependência de relatos do cliente para detecção de falhas. Como a ferramenta tornou possível a realização de análises diárias a equipe de suporte consegue detectar os problemas antes do cliente, proporcionando um suporte *just in time*. Fazendo assim, com que as solicitações de caráter emergenciais fossem reduzidas consideravelmente.

Adicionalmente, há um aspecto comercial que não pode ser relevado, pois houve um crescimento na aparente satisfação do cliente. Isso se deu devido a presença de um suporte mais ativo e apresentando respostas mais rápidas, que mostra um compromisso com a excelência e possibilita ao cliente ter uma maior confiança no pós venda da empresa. Considerado que a qualidade do produto é um fator decisivo para o cliente, ainda mais no mercado de tecnologia da agricultura 4.0, essa ferramenta pode ser fundamental para que a Hexagon garanta o patamar de qualidade esperada de seu produto.

Sob o aspecto de desenvolvimento, a ferramenta traz um código inicial para um estudo exploratório dos dados, para uma classificação de forma quantitativa dos problemas em campo com o intuito de facilitar o processo de priorização de possíveis melhorias e futuros desenvolvimentos.

6.4 LIMITAÇÕES

As tecnologias empregadas no projeto tinham o intuito de atender os objetivos com o menor custo possível. Com isso, para os *dashboards*, foi empregado o *software* Power BI Desktop que atendeu os requisitos e apresentou uma facilidade de utilização. Porém por ser um *software* de licença gratuita ele não apresenta o recurso de publicação dos *dashboards*, em outras palavras não há como realizar a divulgação do mesmo via internet e só é disponível o usuário que possua o arquivo das visualizações desenvolvidas. Além disso, o Power BI tem uma limitação de sistema operacional que na Hexagon é um ponto crítico, visto que os computadores da empresa em sua grande maioria utilizam um sistema operacional não compatível com o *software*, fazendo a sua utilização ainda mais restrita. Assim, o compartilhamento das informações depende de um usuário, que utilize um sistema operacional específico e que tenha o arquivo, para realizar a atualização dos gráficos e realizar o envio dos mesmos para as partes interessadas, fazendo com que a exploração dos dados seja restrita a apenas esse usuário.

Outro ponto a ser levantado é a identificação de problemas de *software*. Devido a não utilização de uma fonte de dados que traga informações diretamente relacionadas à execução do *software* embarcado, a identificação de falhas é muito limitada. Esse tipo de identificação acaba dependendo da interpretação subjetiva de problemas operacionais em campo para a identificação de possíveis indícios de problemas no *software* embarcado.

6.5 FUTUROS DESENVOLVIMENTOS

Para a superação das limitações apresentadas, os desenvolvimentos futuros a curto prazo propostos são:

- Utilização das fontes de dados de *logs* do sistema embarcado, com o intuito de capacitar a ferramenta para apontar problemas de *software* de forma mais precisa.
- Utilização de uma nova tecnologia para a criação dos *dashboards*, com o intuito de permitir o acesso de todas as partes interessadas. Fazendo com que a exploração de dados seja realizada por todos e o compartilhamento das informações não seja realizado por meio de um relatório estático dependendo de um usuário para a elaboração e envio do mesmo.
- Melhoria da modularidade e validação de resultados do código de análise de dados.

A ferramenta, por mais que tenha atendido todos os objetivos e apresentado resultados positivos com sua utilização, ainda está em uma etapa inicial quando comparado com o seu potencial de crescimento. Alguns dos possíveis desenvolvimentos a longo prazo para a ferramenta são:

- A implantação de metodologias de *machine learning* para análise dos dados, com o intuito de identificação de padrões e tendências de problemas em campo de forma preditiva.
- Utilização de dados de novas fontes, para a incorporação de novos dados apresentando um relatório e análise mais completa.
- A aplicação dessa ferramenta para outros produtos Hexagon. Para que as vantagens trazidas pela ferramenta possam ser expandidas para outras soluções da empresa.

REFERÊNCIAS

AGRICULTURE, Hexagon. **HxGN AgrOn Ti5 Ti7**. [S./], 2020. Disponível em:

<https://hexagonagriculture.com/pt-br/solutions/oem/embedded-electronics/ti5--ti7>. Acesso em: 21 ago. 2020.

AMAZON WEB SERVICES, Inc. **AWS IoT Core**. [S./], 2020a. Disponível em:

<https://aws.amazon.com/pt/iot-core/>. Acesso em: 10 set. 2020.

AMAZON WEB SERVICES, Inc. **O que é Amazon Relational Database Service (Amazon RDS)?** [S./], 2020b. Disponível em:

https://docs.aws.amazon.com/pt_br/AmazonRDS/latest/UserGuide/Welcome.html. Acesso em: 10 set. 2020.

AMAZON WEB SERVICES, Inc. **O que é o Amazon EventBridge?** [S./], 2020c. Disponível em:

https://docs.aws.amazon.com/pt_br/eventbridge/latest/userguide/what-is-amazon-eventbridge.html. Acesso em: 10 set. 2020.

AMAZON WEB SERVICES, Inc. **O que é o AWS Lambda?** [S./], 2020d. Disponível em:

https://docs.aws.amazon.com/pt_br/lambda/latest/dg/welcome.html. Acesso em: 10 set. 2020.

AMAZON WEB SERVICES, Inc. **Perguntas frequentes sobre o AWS Lambda**. [S./], 2020e. Disponível em:

<https://aws.amazon.com/pt/lambda/faqs/>. Acesso em: 10 set. 2020.

AMAZON WEB SERVICES, Inc. **Computação em nuvem com a AWS**. [S./], 2020f. Disponível em:

<https://aws.amazon.com/pt/what-is-aws/>. Acesso em: 10 set. 2020.

COATE, Melissa; WEBB, Chris. **Planning a Power BI Enterprise Deployment Page 1 of 25****V3.1as of: May2020****Planning a Power BI Enterprise Deployment**. [S./], 2020. Disponível em:

<https://download.microsoft.com/download/4/8/c/48c693c9-6729-43bb-8cd2-3e6bfe4f8f52/Planning%5C%20a%5C%20Power%5C%20BI%5C%20Enterprise%5C%20Deployment%5C%20-%5C%20whitepaper.pdf>. Acesso em: 17 ago. 2020.

COMMERCE FOR THE UK, Swedish Chamber of. **Hexagon AB**. [S./]. Disponível em: <https://www.scc.org.uk/about/patrons/hexagon-ab/>. Acesso em: 30 ago. 2020.

DODGE, Yadolah. **The Concise Encyclopedia of Statistics**. 1. ed. Switzerland: Springer, 2008.

DB-ENGINES. **DB-Engines Ranking - Trend Popularity**. [S./], 2020. Disponível em: https://db-engines.com/en/ranking_trend. Acesso em: 20 ago. 2020.

FOUNDATION, Python Software. **General Python FAQ**. [S./], 2020. Disponível em: <https://docs.python.org/3/faq/general.html#what-is-python>. Acesso em: 20 ago. 2020.

HEXAGON. **Agricultura**. [S./], 2020a. Disponível em: <https://hexagon.com.br/pt-br/geospatial-solutions/hexagon-agriculture>. Acesso em: 18 ago. 2020.

HEXAGON. **Core Capabilities**. [S./], 2020b. Disponível em: <https://hexagon.com/solutions/core-capabilities>. Acesso em: 31 ago. 2020.

HEXAGON. **Divisions**. [S./], 2020c. Disponível em: <https://hexagon.com/about/divisions>. Acesso em: 17 ago. 2020.

HEXAGON. **Hexagon traz produtividade para o setor sucroenergético**. [S./], 2019. Disponível em: <https://hexagonagriculture.com/pt-br/news/press-release/hexagon-brings-intelligence-efficiency-and-productivity-to-the-sugar-energy-sector>. Acesso em: 17 ago. 2020.

HEXAGON. **História de Sucesso: São Martinho Aumenta a Produtividade de suas Operações por Meio da Transformação Digital**. [S./], 2020d. Disponível em: https://bynder.hexagon.com/m/2ad2be962364b1db/original/Hexagon_AG_Sao-Martinho-Customer-Story_PT_2019.pdf. Acesso em: 30 ago. 2020.

HEXAGON. **Sobre**: Sobre nós. [S./], 2020e. Disponível em: <https://hexagonagriculture.com/pt-br/about>. Acesso em: 13 ago. 2020.

HEXAGON. **Vision, Mission & Core Values**. [S./], 2020f. Disponível em: <https://hexagon.com/about/vision-mission>. Acesso em: 18 ago. 2020.

HUNTER, J. D. Matplotlib: A 2D Graphics Environment. **Computing in Science Engineering**, v. 9, n. 3, p. 90–95, 2007.

ILLOWSKY, Barbara; DEAN, Susan. **Introductory Statistics**. 1. ed. [S.l.]: OpenStax, 2013.

INC., Lucid Software. **What is Unified Modeling Language**. [S.l.], 2020. Disponível em:

<https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language>. Acesso em: 17 ago. 2020.

KIMBALL, Ralph; CASERTA, Joe. **The Data Warehouse ETL Toolkit - Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data**. 1. ed. Indianapolis, IN: Wiley, 2004.

KIMBALL, Ralph; ROSS, Margy. **The Data Warehouse ETL Toolkit - The Complete Guide to Dimensional Modeling**. 2. ed. New York, NY: Wiley, 2002.

MASSRUHÁ S. M. F. S.; LEITE, M. A. de A. **AGRO 4.0 – RUMO À AGRICULTURA DIGITAL**. [S.l.], 2017. Disponível em: <https://www.embrapa.br/busca-de-publicacoes/-/publicacao/1073150/agro-40---rumo-a-agricultura-digital>. Acesso em: 28 ago. 2020.

MATHEMATICS, Encyclopedia of. **Kendall tau metric**. [S.l.], 2020. Disponível em: http://encyclopediaofmath.org/index.php?title=Kendall_tau_metric&oldid=50721. Acesso em: 28 ago. 2020.

NEGASH S., Gray P. **Handbook on Decision Support Systems**. 2. ed. Berlin Heidelberg: Springer, 2008. Disponível em: https://doi.org/10.1007/978-3-540-48716-6_9.

NUNES, Raphael. **Power BI Free vs Power BI Pro: Quais as Diferenças?** [S.l.], 2019. Disponível em: <https://www.voitto.com.br/blog/artigo/diferenca-power-bi-free-pro>. Acesso em: 17 ago. 2020.

PLAINFIELD, IL. **Business Intelligence in Plain Language: A practical guide to Data Mining and Business Analytics**. 2. ed. [S.l.]: Applied Data Labs, 2012.

POSTGRESQL. **About**. [S./], 2020a. Disponível em:

<https://www.postgresql.org/about/>. Acesso em: 19 ago. 2020.

POSTGRESQL. **Chapter 33. libpq - C Library**. [S./], 2020b. Disponível em:

<https://www.postgresql.org/docs/current/libpq.html>. Acesso em: 20 ago. 2020.

TABLEAU. **O que é business intelligence? Seu guia sobre o BI e por que ele é importante**. [S./], 2020. Disponível em:

<https://www.tableau.com/pt-br/learn/articles/business-intelligence>. Acesso em: 10 set. 2020.

TOMÁS, Mário Rui Sampaio. **Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação**. [S./], 2009. Disponível em:

https://run.unl.pt/bitstream/10362/2003/1/WPSeries_09_2009Tomas.pdf. Acesso em: 5 set. 2020.

VARRAZZO, Daniele. **Psycopg**. [S./], 2020. Disponível em:

<https://www.psycopg.org/>. Acesso em: 20 ago. 2020.

VIAN, Carlos Eduardo Freitas. **Qualidade de matéria-prima**. [S./]. Disponível em:

https://www.agencia.cnptia.embrapa.br/gestor/cana-de-acucar/arvore/CONTAG01_138_22122006154842.html. Acesso em: 20 ago. 2020.

WIKIPEDIA. **Coeficiente de correlação de postos de Spearman**. [S./], 2020a.

Disponível em: https://pt.wikipedia.org/wiki/Coeficiente_de_correla%C3%5C%A7%5C%5C%5C%A3o_de_postos_de_Spearman. Acesso em: 28 ago. 2020.

WIKIPEDIA. **Valor-p**. [S./], 2020b. Disponível em:

<https://pt.wikipedia.org/wiki/Valor-p>. Acesso em: 18 ago. 2020.

WRITER, Staff. **Hamelex White delivers new Jumbo Loaders**. [S./], 2016.

Disponível em:

<https://www.trailermag.com.au/hamelex-white-delivers-new-jumbo-loaders/>. Acesso em: 30 ago. 2020.

APÊNDICE A – CÓDIGO DO MÉTODO COUNT_CORRELATION

Figura 49 – Método *count_correlation*.

```

def count_correlation(self,perf_df):
    analysis_name = 'Alarm Count'
    alarms_22_df = self.query_data(alarm_22_count_query,(self.create_param_dict(alarm_22_indexes,50067)))
    alarms_df = self.query_data(alarms_count_query,(self.create_param_dict(alarms_of_interest,50067)))
    #Unite all columns from alarms_df with perf_df
    for alarm_type in sorted(alarms_df['id_alarm_type'].unique()):
        perf_df = pd.merge(perf_df,alarms_df[alarms_df['id_alarm_type']==alarm_type]
                           [['count','date']],on='date',how='left')

        name_column = 'alarm_'+str(alarm_type)
        perf_df.rename(columns={'count':name_column}, inplace=True)
    perf_df.fillna("0", inplace = True)

    #Unite all columns from alarms_22_df with perf_df
    for alarm_behavior in sorted(alarms_22_df['id_alarm_behavior'].unique()):
        perf_df = pd.merge(perf_df,alarms_22_df[alarms_22_df['id_alarm_behavior']==alarm_behavior]
                           [['count','date']],on='date',how='left')

        name_column = 'index_'+str(alarm_behavior)
        perf_df.rename(columns={'count':name_column}, inplace=True)
    perf_df.fillna("0", inplace = True)
    columns_not_intest = ['shipment_location_id','date_sk','shipment_rmt_data','shipment_count','date']
    #Convert all columns of insterest to int64 type.
    for col in perf_df.columns:
        if 'alarm' in col or 'index' in col:
            perf_df[col] = perf_df[col].astype('int64')

    pearson_df = perf_df.loc[:,[i for i in list(perf_df.columns) if i not in columns_not_intest]].
                  corr(method='pearson')
    kendal_df = perf_df.loc[:,[i for i in list(perf_df.columns) if i not in columns_not_intest]].
                corr(method='kendall')
    spearman_df = perf_df.loc[:,[i for i in list(perf_df.columns) if i not in columns_not_intest]].
                  corr(method='spearman')
    pearson_p_value_df = perf_df.loc[:,[i for i in list(perf_df.columns) if i not in columns_not_intest]].
                          corr(method='p_value_perm_test')

    self.plot_bar_correlations(pearson_df,kendal_df,spearman_df,analysis_name)
    self.plot_tables(pearson_df,'Pearson Correlations',analysis_name,header_size=4,values_size=5)
    self.plot_tables(pearson_p_value_df,'Pearson Correlations p Values',analysis_name,header_size=4,
                    values_size=5,color_map_conditioned=True)

    self.plot_alarms_corr(perf_df,pearson_df,analysis_name)

    pearson_df=pearson_df.merge(pearson_p_value_df['percent_rmt_data'].rename('percent_rmt_p'),
                               how='left',left_index=True,right_index=True)
    pearson_df = pearson_df.rename({'{}_count'.format}
    return pearson_df[['percent_rmt_data','percent_rmt_p']].loc[set(pearson_df.index) -
                    set(['percent_id_local_count','percent_rmt_data_count'])]

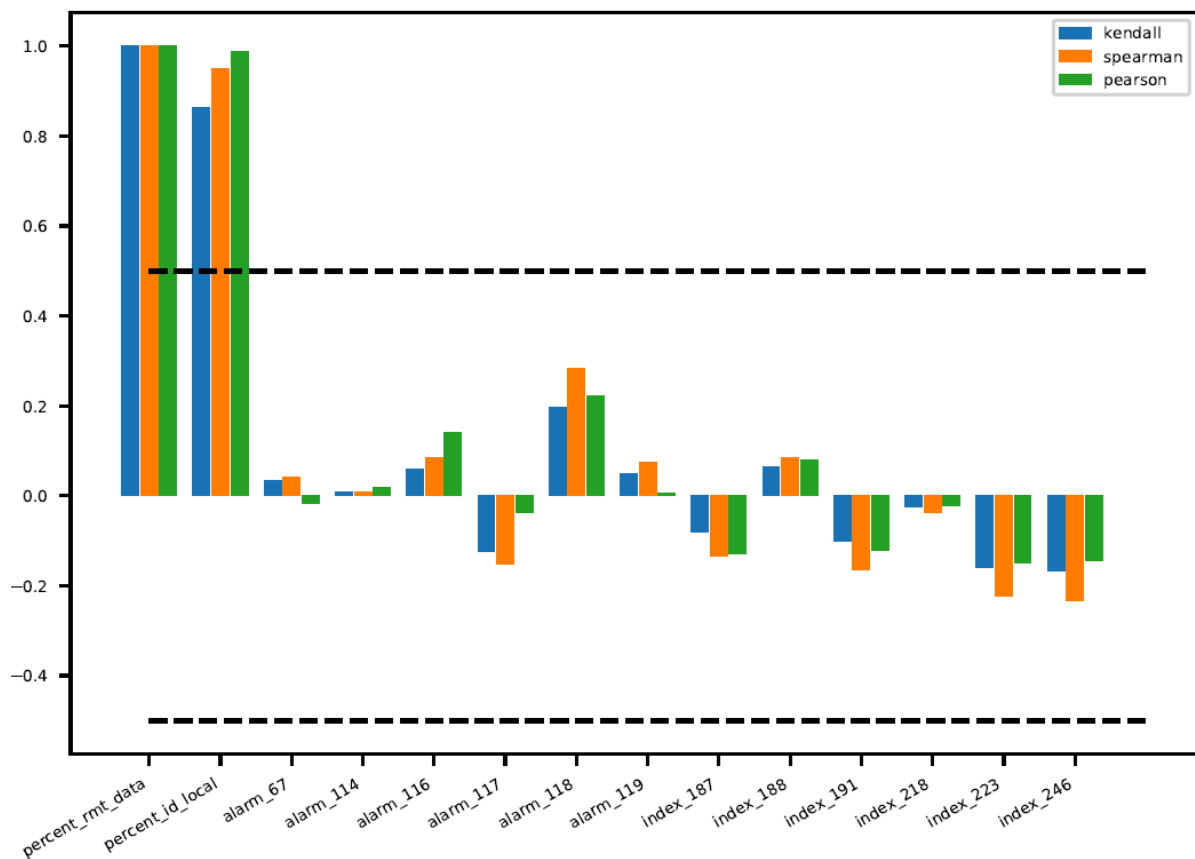
```

Fonte – Elaborado pelo autor.

APÊNDICE B – GRÁFICOS RESULTANTES DO CÓDIGO DE ANÁLISE CORRELACIONAL DE DADOS

Figura 50 – Gráfico do valor dos diferentes coeficientes de correlação entre cada contagem de alarme e os dados da porcentagem de carregamentos com coordenadas de origem.

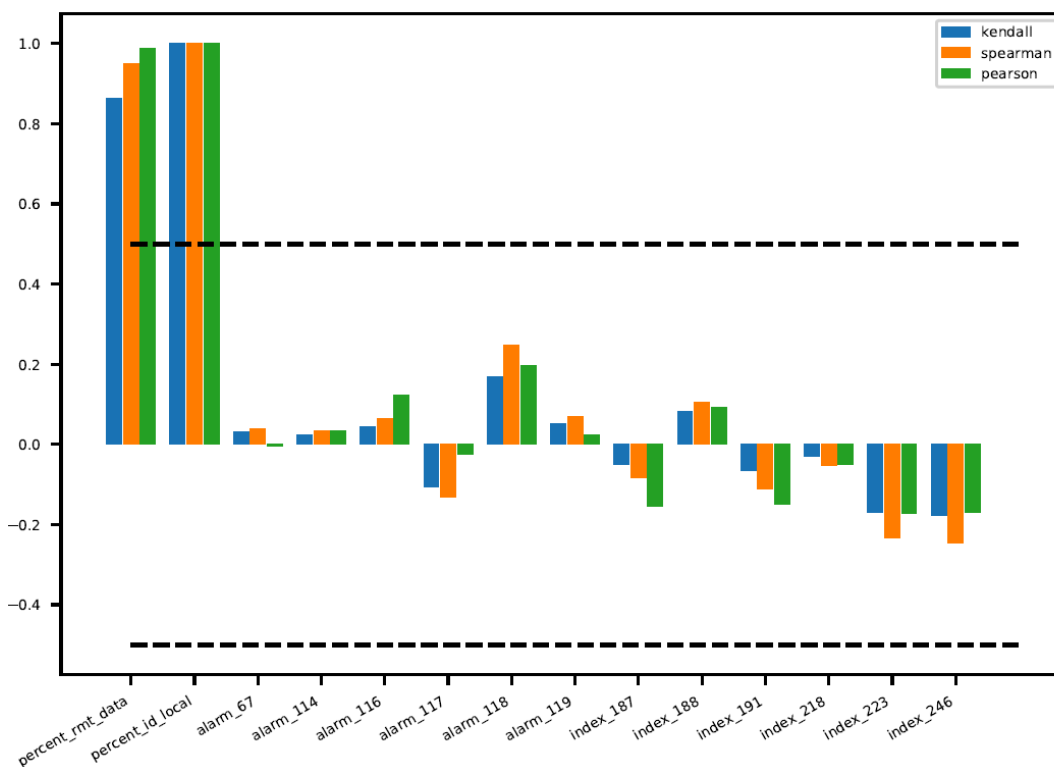
Correlation with RMT Data - Alarm Count



Fonte – Elaborado pelo autor.

Figura 51 – Gráfico do valor dos diferentes coeficientes de correlação entre cada contagem de alarme e os dados da porcentagem de carregamentos com ID de origem.

Correlation with Location ID - Alarm Count



Fonte – Elaborado pelo autor.

Figura 52 – Tabela dos valores de coeficiente de Pearson entre todos os dados referente a contagem dos alarmes.

Pearson Correlations - Alarm Count

	rmt_data	local_id	alarm_67	alarm_114	alarm_116	alarm_117	alarm_118	alarm_119	index_187	index_188	index_191	index_218	index_223	index_246
rmt_data	1.0	0.988	-0.016	0.019	0.141	-0.038	0.223	0.005	-0.129	0.081	-0.122	-0.021	-0.149	-0.143
local_id	0.988	1.0	-0.005	0.034	0.123	-0.024	0.197	0.023	-0.154	0.092	-0.149	-0.05	-0.172	-0.17
alarm_67	-0.016	-0.005	1.0	0.208	0.024	-0.043	0.025	0.328	-0.098	-0.133	-0.124	0.157	0.094	0.09
alarm_114	0.019	0.034	0.208	1.0	0.068	-0.04	0.062	0.214	0.231	-0.092	0.175	0.126	0.066	0.069
alarm_116	0.141	0.123	0.024	0.068	1.0	0.318	0.78	0.013	-0.074	-0.051	-0.047	-0.006	-0.184	-0.174
alarm_117	-0.038	-0.024	-0.043	-0.04	0.318	1.0	0.193	-0.089	-0.034	-0.034	-0.007	-0.042	-0.037	0.043
alarm_118	0.223	0.197	0.025	0.062	0.78	0.193	1.0	-0.077	-0.011	-0.024	0.002	-0.049	-0.205	-0.2
alarm_119	0.005	0.023	0.328	0.214	0.013	-0.089	-0.077	1.0	0.091	-0.192	0.05	0.169	0.199	0.182
index_187	-0.129	-0.154	-0.098	0.231	-0.074	-0.034	-0.011	0.091	1.0	-0.14	0.982	0.121	0.034	0.007
index_188	0.081	0.092	-0.133	-0.092	-0.051	-0.034	-0.024	-0.192	-0.14	1.0	-0.154	-0.05	0.016	0.05
index_191	-0.122	-0.149	-0.124	0.175	-0.047	-0.007	0.002	0.05	0.982	-0.154	1.0	0.088	0.005	-0.018
index_218	-0.021	-0.05	0.157	0.126	-0.006	-0.042	-0.049	0.169	0.121	-0.05	0.088	1.0	0.491	0.483
index_223	-0.149	-0.172	0.094	0.066	-0.184	-0.037	-0.205	0.199	0.034	0.016	0.005	0.491	1.0	0.974
index_246	-0.143	-0.17	0.09	0.069	-0.174	0.043	-0.2	0.182	0.007	0.05	-0.018	0.483	0.974	1.0

Fonte – Elaborado pelo autor.

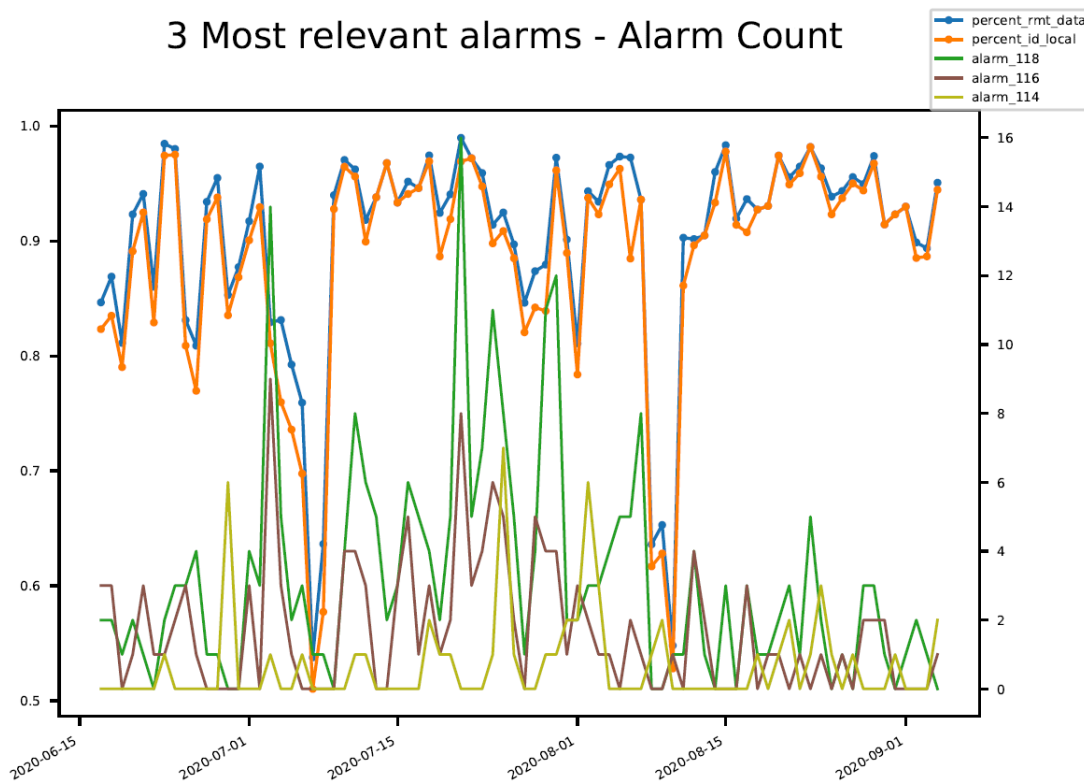
Figura 53 – Tabela dos valores P entre todos os dados referente a contagem dos alarmes.

Pearson Correlations p Values - Alarm Count

	rmt_data	local_id	alarm_67	alarm_114	alarm_116	alarm_117	alarm_118	alarm_119	index_187	index_188	index_191	index_218	index_223	index_246
rmt_data	1.0	0.0	0.887	0.866	0.212	0.735	0.047	0.966	0.253	0.477	0.281	0.853	0.188	0.205
local_id	0.0	1.0	0.963	0.762	0.279	0.832	0.081	0.842	0.172	0.415	0.188	0.659	0.126	0.132
alarm_67	0.887	0.963	1.0	0.064	0.835	0.704	0.823	0.003	0.385	0.241	0.274	0.164	0.409	0.427
alarm_114	0.866	0.762	0.064	1.0	0.548	0.724	0.587	0.057	0.039	0.415	0.121	0.267	0.558	0.543
alarm_116	0.212	0.279	0.835	0.548	1.0	0.004	0.0	0.909	0.514	0.655	0.676	0.961	0.102	0.123
alarm_117	0.735	0.832	0.704	0.724	0.004	1.0	0.086	0.43	0.765	0.765	0.948	0.71	0.747	0.703
alarm_118	0.047	0.081	0.823	0.587	0.0	0.086	1.0	0.499	0.926	0.832	0.985	0.665	0.068	0.075
alarm_119	0.966	0.842	0.003	0.057	0.909	0.43	0.499	1.0	0.421	0.089	0.657	0.135	0.077	0.107
index_187	0.253	0.172	0.385	0.039	0.514	0.765	0.926	0.421	1.0	0.216	0.0	0.286	0.764	0.95
index_188	0.477	0.415	0.241	0.415	0.655	0.765	0.832	0.089	0.216	1.0	0.172	0.66	0.889	0.657
index_191	0.281	0.188	0.274	0.121	0.676	0.948	0.985	0.657	0.0	0.172	1.0	0.439	0.963	0.874
index_218	0.853	0.659	0.164	0.267	0.961	0.71	0.665	0.135	0.286	0.66	0.439	1.0	0.0	0.0
index_223	0.188	0.126	0.409	0.558	0.102	0.747	0.068	0.077	0.764	0.889	0.963	0.0	1.0	0.0
index_246	0.205	0.132	0.427	0.543	0.123	0.703	0.075	0.107	0.95	0.657	0.874	0.0	0.0	1.0

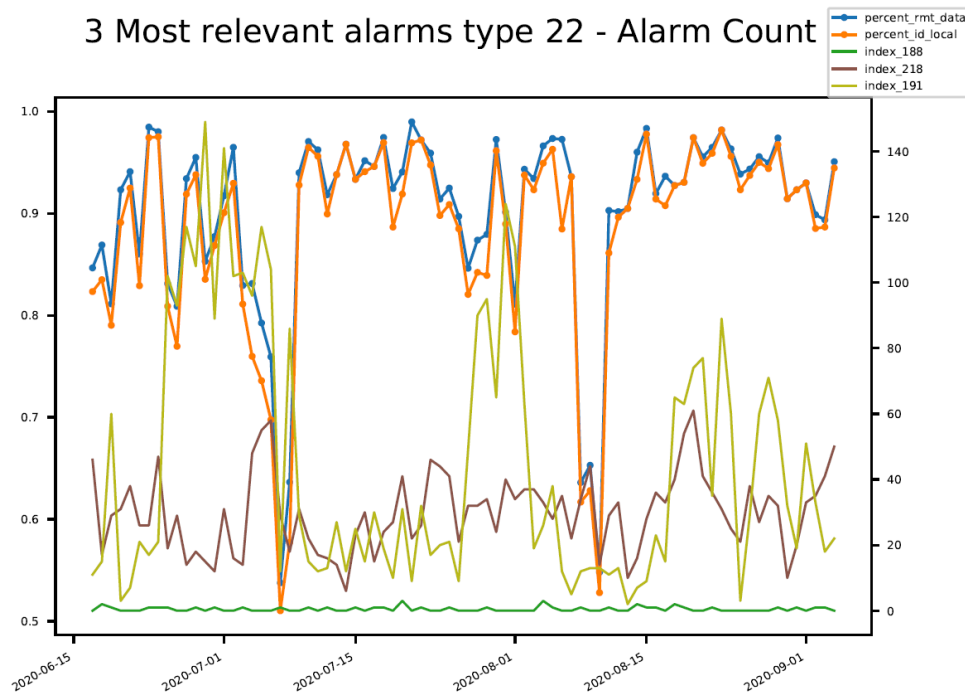
Fonte – Elaborado pelo autor.

Figura 54 – Gráfico dos três alarmes nativos que tiveram o maior coeficiente de correlação de Pearson em relação a porcentagem de carregamentos com coordenadas de origem.



Fonte – Elaborado pelo autor.

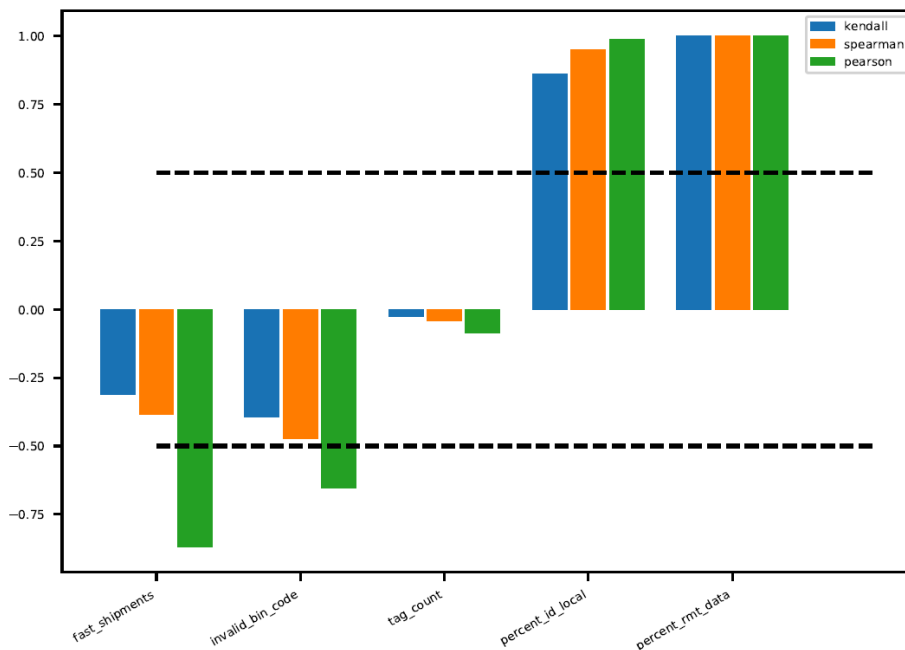
Figura 55 – Gráfico no tempo dos três alarmes configuráveis que tiveram o maior coeficiente de correlação de Pearson em relação a porcentagem de carregamentos com coordenadas de origem.



Fonte – Elaborado pelo autor.

Figura 56 – Gráfico do valor dos diferentes coeficientes de correlação entre os dados de problemas relacionados a balança e os dados da porcentagem de carregamentos com coordenadas de origem.

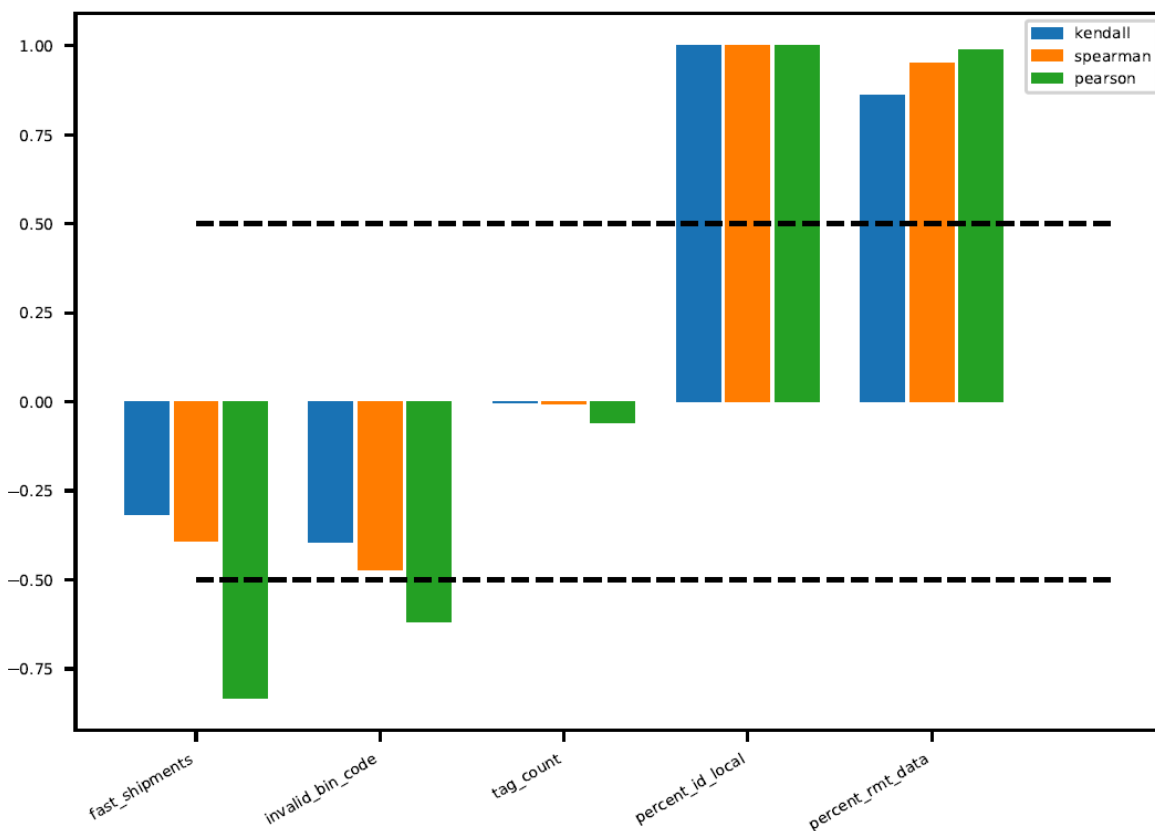
Correlation with RMT Data - Weighbridge Issues



Fonte – Elaborado pelo autor.

Figura 57 – Gráfico do valor dos diferentes coeficientes de correlação entre os dados de problemas relacionados à balança e os dados da porcentagem de carregamentos com ID de origem.

Correlation with Location ID - Weighbridge Issues



Fonte – Elaborado pelo autor.

Figura 58 – Tabela dos valores de coeficiente de Pearson entre todos os dados referentes aos problemas relacionados à balança.

Pearson Correlations - Weighbridge Issues

	fast_shipments	invalid_bin_code	tag_count	local_id	rmt_data
fast_shipments	1.0	0.765	0.121	-0.832	-0.869
invalid_bin_code	0.765	1.0	0.144	-0.617	-0.653
tag_count	0.121	0.144	1.0	-0.057	-0.086
local_id	-0.832	-0.617	-0.057	1.0	0.988
rmt_data	-0.869	-0.653	-0.086	0.988	1.0

Fonte – Elaborado pelo autor.

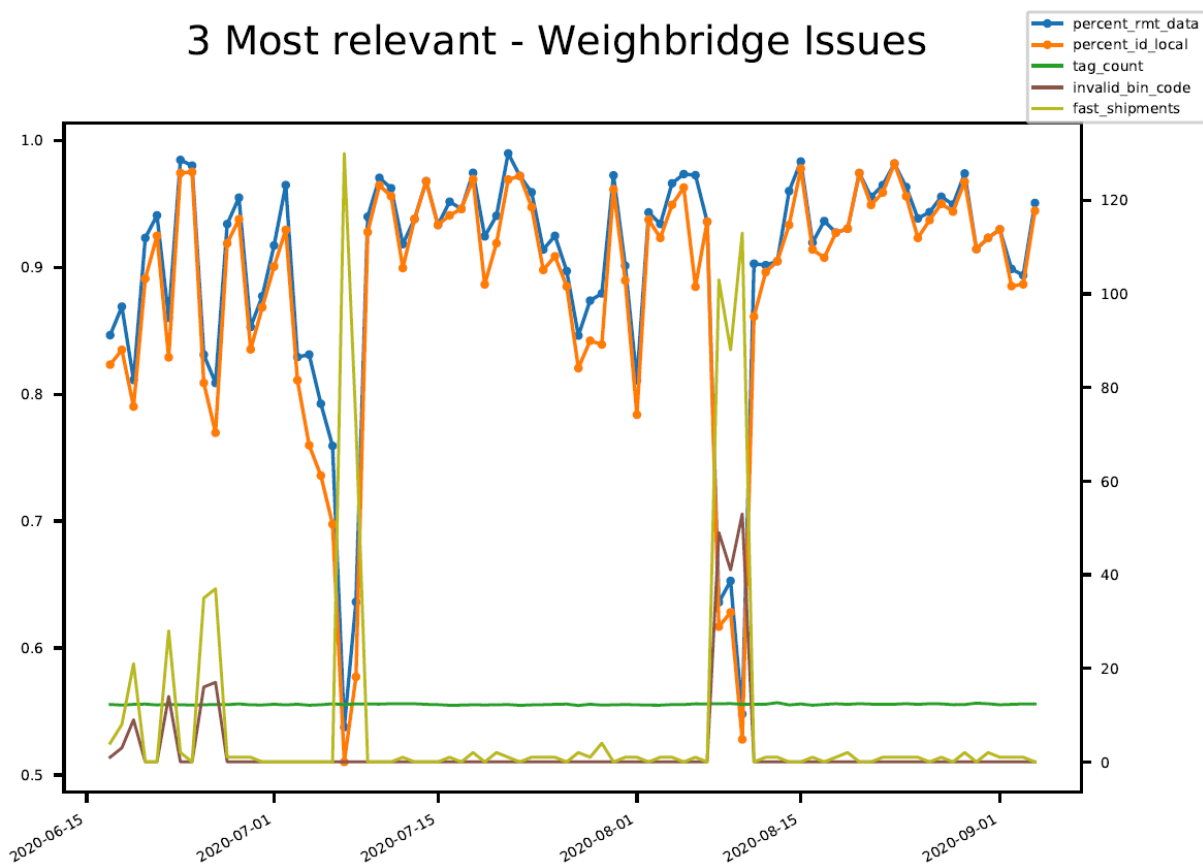
Figura 59 – Tabela dos valores P entre todos os dados referentes aos problemas relacionados à balança.

Pearson Correlations p Values - Weighbridge Issues

	fast_shipments	invalid_bin_code	tag_count	local_id	rmt_data
fast_shipments	1.0	0.0	0.283	0.0	0.0
invalid_bin_code	0.0	1.0	0.204	0.0	0.0
tag_count	0.283	0.204	1.0	0.613	0.451
local_id	0.0	0.0	0.613	1.0	0.0
rmt_data	0.0	0.0	0.451	0.0	1.0

Fonte – Elaborado pelo autor.

Figura 60 – Gráfico no tempo dos dados referentes aos problemas relacionados à balança.



Fonte – Elaborado pelo autor.

Figura 61 – Lista dos dez dados com maiores correlações com a porcentagem de carregamentos com coordenadas de origem e menores valor P.

Most relevant issues

	percent_rmt_data_correlation	percent_rmt_p
fast_shipments	-0.869	0.0
invalid_bin_code	-0.653	0.0
alarm_118_count	0.223	0.047
index_223_time	-0.174	0.124
index_223_count	-0.149	0.188
index_246_count	-0.143	0.205
alarm_116_count	0.141	0.212
index_187_count	-0.129	0.253
alarm_118_time	0.123	0.277
index_191_count	-0.122	0.281

Fonte – Elaborado pelo autor.