Uma Proposta de Data Linter para Qualidade de Dados

Juliano Abi Thomas
UFMS - Universidade Federal de Mato Grosso do Sul
Campo Grande, Brasil
juliano.thomas@ufms.br

Weslley Felippe Picinin dos Santos
UFMS - Universidade Federal de Mato Grosso do Sul
Campo Grande, Brasil
weslley.picinin@ufms.br

Diogo de Lima Menezes
UFMS - Universidade Federal de Mato Grosso do Sul
Campo Grande, Brasil
diogo.menezes@ufms.br

Awdren de Lima Fontão
UFMS - Universidade Federal de Mato Grosso do Sul
Campo Grande, Brasil
awdren.fontao@ufms.br

Resumo

A qualidade dos dados é um fator importante em aplicações de Inteligência Artificial, aprendizado de máquina e engenharia de dados, uma vez que falhas de qualidade podem comprometer análises e reduzir a confiabilidade dos resultados obtidos. Embora existam ferramentas de validação de dados, a identificação precoce de não conformidades ainda representa um desafio em *pipelines* modernos de dados. Este trabalho apresenta uma ferramenta automatizada para verificação da qualidade de dados inspirada no conceito de *Data Linter*, baseada na execução de heurísticas para identificação preliminar de problemas em *datasets*. A avaliação foi realizada utilizando 27 conjuntos de dados públicos do benchmark *MLE-Bench*, além de testes complementares com arquivos no formato Parquet. Os resultados indicaram que a ferramenta foi capaz de identificar registros duplicados, colunas vazias, desequilíbrios de categorias, *miscoding* numérico, valores negativos e *outliers*, evidenciando seu potencial como mecanismo de apoio à verificação da qualidade dos dados em aplicações de análise de dados e aprendizado de máquina.

Palavras-chave

Qualidade de dados, Data linter, Validação de dados, Aprendizado de máquina, Engenharia de dados.

1 Introdução

A crescente adoção de aplicações baseadas em Inteligência Artificial (IA), aprendizado de máquina e análise de dados tem ampliado a importância da engenharia de dados no desenvolvimento de sistemas modernos. Embora avanços recentes tenham facilitado a construção e utilização de modelos de aprendizado de máquina, organizações ainda enfrentam desafios relacionados à preparação, gerenciamento e qualidade dos dados utilizados nesses sistemas [11]. Com isso, os dados deixam de ser apenas um insumo para os modelos e passam a representar um elemento central para a confiabilidade de todo o processo analítico.

Em ambientes de Engenharia de Dados, DataOps e MLOps, os dados percorrem diferentes etapas, incluindo coleta, armazenamento, transformação, validação e consumo. Problemas introduzidos em qualquer uma dessas etapas propagam-se ao longo do *pipeline*, impactando análises, modelos e decisões baseadas em dados [9, 11]. Dessa forma, a identificação precoce de irregularidades nos dados torna-se uma atividade importante para reduzir retrabalho e aumentar a confiabilidade dos artefatos produzidos.

Falhas como registros duplicados, valores ausentes, divergências de representação, desequilíbrios de categorias e valores discrepantes são frequentemente encontradas em *datasets* utilizados em aplicações de aprendizado de máquina. Essas ocorrências afetam a qualidade dos dados utilizados em processos analíticos, comprometendo a confiabilidade dos resultados e dificultando etapas posteriores do ciclo de vida dos dados [1, 5]. Além disso, estudos recentes apontam que uma parcela significativa dos desafios enfrentados por equipes de ciência de dados está diretamente relacionada ao gerenciamento e à qualidade dos dados utilizados nos processos de desenvolvimento e operação de sistemas inteligentes [10].

Diversas ferramentas têm sido propostas para apoiar a validação de dados, incluindo soluções baseadas em regras explícitas e mecanismos de monitoramento contínuo. Embora essas abordagens contribuam para a melhoria da qualidade dos dados, diversas delas dependem da configuração manual de verificações específicas para cada contexto de uso. Esse processo tende a aumentar o esforço de adoção e dificultar a identificação automática de problemas em diferentes conjuntos de dados.

Diante desse cenário, este trabalho apresenta uma ferramenta automatizada para verificação da qualidade de dados inspirada no conceito de *Data Linter* [8]. A proposta busca apoiar a detecção automatizada de problemas de qualidade dos dados por meio da execução automática de heurísticas integradas ao fluxo de desenvolvimento. A solução foi implementada utilizando Node.js, Python, PySpark e Husky, permitindo sua utilização em processos automatizados de validação de dados.

O presente trabalho propõe uma ferramenta automatizada para apoio à verificação da qualidade de dados, baseada em heurísticas inspiradas no conceito de *Data Linter* e integrada ao fluxo de desenvolvimento por meio de mecanismos de automação.

2 Fundamentação Teórica

2.1 Qualidade de Dados

Dados de qualidade são aqueles adequados ao propósito para o qual serão utilizados. Esse conceito está relacionado a características como confiabilidade, consistência, completude e correção das informações empregadas em processos analíticos e sistemas computacionais [5].

Problemas de qualidade surgem em diferentes etapas do ciclo de vida dos dados, incluindo coleta, integração, transformação e armazenamento. Em aplicações de aprendizado de máquina, esses problemas podem introduzir vieses, comprometer a capacidade de

generalização dos modelos e reduzir a confiabilidade dos resultados obtidos [1].

Dentre as ocorrências mais comuns encontram-se valores ausentes, registros duplicados, inconsistências de formatação, valores discrepantes e erros de representação dos dados. Segundo Foidl et al. [5], essas ocorrências podem indicar falhas latentes capazes de impactar sistemas baseados em Inteligência Artificial.

Com isso, mecanismos de verificação e monitoramento tornam-se importantes para auxiliar na identificação de problemas e contribuir para o uso mais confiável dos *datasets* em aplicações computacionais.

2.2 Data Linters

O conceito de *linter* surgiu na engenharia de software como uma ferramenta voltada à identificação de erros, inconsistências e más práticas em código-fonte. Com o crescimento das aplicações baseadas em dados, essa abordagem passou a ser aplicada também à análise de *datasets* [3].

Surgiram então os *Data Linters*, ferramentas destinadas à inspeção automática de conjuntos de dados. Essas soluções utilizam verificações e heurísticas capazes de identificar falhas, distribuições incomuns e outros padrões que possam comprometer análises e modelos de aprendizado de máquina [8].

O trabalho de Hynes et al. [8] consolidou esse conceito ao propor uma ferramenta voltada à análise automática de *datasets* utilizados em aplicações de aprendizado de máquina. A proposta busca auxiliar usuários na identificação de possíveis problemas antes da utilização dos dados em processos analíticos.

Portanto, os *Data Linters* atuam como um mecanismo complementar às técnicas tradicionais de verificação, contribuindo para a melhoria da qualidade dos dados e para o uso mais confiável dos *datasets*.

2.3 Big Data e Processamento Distribuído

A escala sem precedentes de dados produzidos por sistemas atuais trouxe desafios relacionados ao armazenamento e ao processamento eficiente das informações. O termo *Big Data* é utilizado para descrever cenários caracterizados por grandes volumes de dados, alta velocidade de geração e diversidade de formatos.

Para lidar com esses desafios, técnicas de processamento distribuído tornaram-se amplamente utilizadas. Nessa abordagem, as tarefas são divididas entre diferentes recursos computacionais, permitindo a execução paralela das operações e aumentando a eficiência do processamento.

Entre as principais tecnologias empregadas nesse caso destaca-se o Apache Spark, *framework* voltado ao processamento distribuído de dados em larga escala [13]. O Spark oferece recursos para processamento paralelo, otimização de consultas e suporte a diferentes formatos de armazenamento, sendo amplamente utilizado em ambientes de *Big Data*. Neste trabalho, o PySpark foi utilizado para viabilizar o processamento dos datasets e a compatibilidade com arquivos em formato Parquet, além de permitir a evolução da solução para cenários que demandem o processamento de maiores volumes de dados.

2.4 Automação e Integração Contínua

A automação de processos tornou-se uma prática comum no desenvolvimento de software, permitindo a execução automática de tarefas e reduzindo a necessidade de verificações manuais. Entre seus principais benefícios estão a padronização das atividades, a redução de erros humanos e a detecção mais rápida de problemas.

Entre os mecanismos utilizados para esse fim destacam-se os *hooks* de sistemas de controle de versão, capazes de executar comandos automaticamente em etapas específicas do fluxo de trabalho dos desenvolvedores.

Segundo Breck et al. [1], mecanismos contínuos de verificação constituem uma estratégia para identificação de problemas em *pipelines* de aprendizado de máquina, contribuindo para a melhoria da qualidade dos dados utilizados.

3 Trabalhos Relacionados

3.1 The Data Linter

O artigo *The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets* [8] apresenta uma ferramenta voltada à inspeção automática de *datasets* utilizados em aplicações de aprendizado de máquina.

A solução utiliza heurísticas para identificar irregularidades, distribuições incomuns e outros padrões capazes de comprometer a qualidade dos dados. Sua principal característica é a realização de verificações exploratórias sem a necessidade de regras específicas para cada conjunto de dados analisado [8].

Outro aspecto relevante é a arquitetura baseada em heurísticas independentes, permitindo a inclusão de novas verificações sem alterações significativas no núcleo da ferramenta.

O presente trabalho foi inspirado nos conceitos propostos por Hynes et al. [8], utilizando parte das heurísticas apresentadas pelos autores como base para a implementação da solução desenvolvida. O uso de hooks de pré-commit é especialmente adequado para conjuntos de dados empregados durante etapas de desenvolvimento, experimentação e preparação dos dados. Adicionalmente, o suporte ao PySpark e ao formato Parquet amplia o potencial de aplicação da ferramenta em ambientes que utilizam maiores volumes de dados, embora a integração por pré-commit seja mais frequentemente adotada para artefatos mantidos sob controle de versão local.

3.2 Ferramentas de Qualidade de Dados

Diversas ferramentas têm sido propostas para auxiliar na verificação da qualidade dos dados, incluindo soluções como *Great Expectations* [6], *dbt* [4] e o próprio *Data Linter* [8]. Essas abordagens diferem principalmente quanto ao uso de regras manuais, mecanismos automatizados de análise e integração com fluxos de desenvolvimento.

A ferramenta proposta neste trabalho combina conceitos presentes nessas abordagens, utilizando heurísticas automatizadas inspiradas no *Data Linter*, integração com *hooks* de pré-*commit* e suporte ao processamento distribuído por meio do PySpark.

Ferramentas como *Great Expectations* e *dbt* oferecem mecanismos robustos para validação e monitoramento de dados, permitindo a definição explícita de regras, testes e expectativas de qualidade.

Entretanto, sua utilização normalmente requer configuração manual das verificações de acordo com o contexto de cada conjunto de dados.

Em contrapartida, a solução proposta prioriza a execução automática de heurísticas previamente implementadas e integradas ao fluxo de desenvolvimento. Como principal vantagem, essa abordagem reduz o esforço inicial necessário para adoção da ferramenta, uma vez que as verificações podem ser executadas sem a necessidade de configuração detalhada para cada *dataset* analisado. Como contrapartida, apresenta menor flexibilidade de personalização quando comparada a soluções baseadas em regras definidas explicitamente pelos usuários.

3.3 Automação e Validação em Pipelines

Práticas de automação têm sido amplamente incorporadas a ambientes modernos de engenharia de software e engenharia de dados. Mecanismos contínuos de verificação contribuem para a identificação de problemas durante o fluxo de desenvolvimento, reduzindo a propagação de falhas para etapas posteriores dos *pipelines* [1].

3.4 Comparação entre os Trabalhos

Os trabalhos analisados apresentam diferentes abordagens para verificação da qualidade de dados. Enquanto o *Data Linter* [8] utiliza heurísticas automatizadas para identificação de irregularidades em *datasets*, ferramentas como *Great Expectations* [6] e *dbt* [4] concentram-se principalmente na definição explícita de regras de validação.

A proposta desenvolvida neste trabalho busca combinar características dessas abordagens, incorporando heurísticas automatizadas, integração ao fluxo de desenvolvimento por meio de *hooks* de pré-*commit* e suporte ao processamento distribuído utilizando PySpark. A Tabela 1 apresenta uma comparação entre as principais características das soluções analisadas e a ferramenta proposta.

Tabela 1: Comparação entre os trabalhos relacionados

Característica	Data Linter	Great Expectations / dbt	Este trabalho
Validação automatizada	Sim	Sim	Sim
Heurísticas	Sim	Não	Sim
Regras manuais	Não	Sim	Opcional
Pré-commit	Não	Não	Sim
Proc. distribuído	Não	Parcial	Sim
PySpark	Não	Parcial	Sim
CSV	Sim	Sim	Sim
Parquet	Não	Sim	Sim
Extensibilidade	Sim	Parcial	Sim

A principal diferença da solução proposta está na combinação entre a execução automática de heurísticas inspiradas no *Data Linter*, a integração ao fluxo de desenvolvimento por meio de *hooks* de pré-*commit* e uma arquitetura modular para inclusão de novas verificações. Essa abordagem reduz a necessidade de configuração manual de regras específicas para cada conjunto de dados analisado, favorecendo a automatização do processo de validação e a extensibilidade da ferramenta.

4 Metodologia

4.1 Seleção das Heurísticas

As heurísticas implementadas foram definidas com base nos conceitos apresentados por Hynes et al. [8] e adaptadas aos objetivos deste trabalho. As verificações contemplam problemas recorrentes de qualidade dos dados, incluindo colunas vazias, registros duplicados, variações de capitalização, *miscoding* numérico, valores negativos, valores extremos e desequilíbrio de categorias.

Embora as heurísticas implementadas tenham sido inspiradas no trabalho de Hynes et al. [8], os experimentos foram realizados sobre conjuntos de dados distintos daqueles utilizados pelos autores. Dessa forma, não é possível realizar uma comparação quantitativa direta entre os resultados obtidos neste trabalho e os apresentados no estudo original.

4.2 Estratégia de Avaliação

A avaliação da ferramenta foi realizada utilizando 27 *datasets* públicos do benchmark *MLE-Bench* [2], abrangendo diferentes domínios e estruturas de dados.

Também foram realizados testes com arquivos no formato Parquet utilizando um *dataset* do projeto *HackRep* [7], permitindo verificar a compatibilidade da solução com cenários de processamento distribuído utilizando PySpark.

4.3 Ambiente Experimental

O desenvolvimento da ferramenta foi realizado em ambiente Windows utilizando Visual Studio Code, Node.js, Python, PySpark e Husky. Para execução do ambiente Spark foi utilizada a plataforma Java 17 e um ambiente virtual Python (*venv*) para gerenciamento das dependências da aplicação.

Tabela 2: Ambiente de desenvolvimento

Componente	Versão	Finalidade
Python	3.11.9	Execução do linter e heurísticas
PySpark	4.1.1	Processamento distribuído de dados
OpenJDK	17.0.18	Execução do ambiente Apache Spark
Node.js	24.14.0	Configuração inicial da ferramenta
Husky	9.1.7	Automação do *hook* de pré-*commit*
Visual Studio Code	Atual	Ambiente de desenvolvimento

As versões utilizadas são apresentadas na Tabela 2, contribuindo para a reprodutibilidade do ambiente experimental.

4.4 Datasets Utilizados

A avaliação da ferramenta foi realizada utilizando *datasets* públicos nos formatos CSV e Parquet. Os experimentos principais foram conduzidos com conjuntos de dados do benchmark *MLE-Bench* [2], enquanto os testes com arquivos Parquet foram realizados utilizando um *dataset* do projeto *HackRep* [7].

A utilização de diferentes formatos permitiu avaliar tanto o comportamento das heurísticas quanto a compatibilidade da ferramenta com cenários distintos de processamento de dados.

4.5 Datasets em Formato CSV

Para validação experimental da ferramenta foram utilizados 27 *datasets* do benchmark *MLE-Bench* [2], abrangendo diferentes tarefas e domínios de aprendizado de máquina.

A diversidade dos conjuntos de dados permitiu avaliar o comportamento das heurísticas em cenários variados, contribuindo para a análise da capacidade da ferramenta em identificar anomalias em diferentes tipos de *datasets*.

4.6 Dataset em Formato Parquet

Para validar a utilização da ferramenta em cenários de processamento distribuído, foi utilizado um *dataset* em formato Parquet proveniente do projeto *HackRep* [7].

Diferentemente dos *datasets* do benchmark *MLE-Bench*, utilizados na avaliação quantitativa das heurísticas implementadas, o *dataset* HackRep foi empregado apenas para verificar o funcionamento da ferramenta com arquivos no formato Parquet e com o ambiente baseado em PySpark. Dessa forma, sua utilização teve caráter complementar, permitindo validar a execução da solução em um formato amplamente empregado em cenários de engenharia de dados e *Big Data*.

A Figura 1 apresenta um exemplo da execução da ferramenta sobre o *dataset* HackRep em formato Parquet. Esse experimento foi realizado para demonstrar a capacidade da solução em processar esse formato de armazenamento utilizando o ambiente baseado em PySpark.

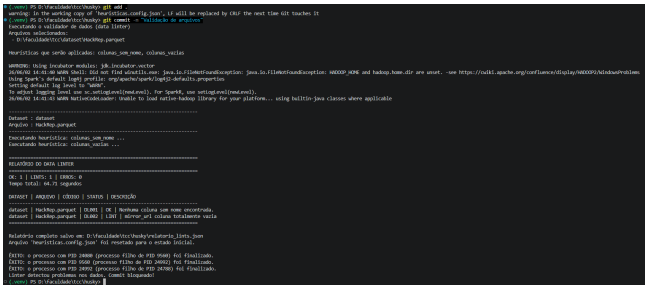

Figura 1: Execução da ferramenta sobre o dataset HackRep em formato Parquet

4.7 Procedimento de Execução

O fluxo de execução da ferramenta inicia-se com a seleção dos *datasets* e heurísticas que serão utilizados nas análises. Essas configurações são armazenadas e posteriormente utilizadas pelo módulo principal da aplicação.

Durante o processo de *commit*, o Husky executa automaticamente as validações configuradas. O sistema realiza a leitura dos dados, inicializa o ambiente de processamento e executa as heurísticas selecionadas sobre os *datasets* analisados.

Ao final da execução, os resultados são apresentados ao usuário de forma padronizada, contendo informações sobre os problemas identificados nos dados. Caso ocorrências sejam encontradas, o processo de *commit* pode ser interrompido automaticamente.

5 Desenvolvimento da Ferramenta

5.1 Visão geral da ferramenta

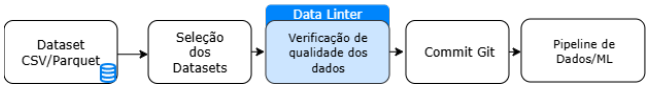

Figura 2: Posicionamento da ferramenta no ciclo de vida dos dados

A Figura 2 apresenta o posicionamento da ferramenta proposta no ciclo de vida dos dados. A solução atua como uma etapa de verificação da qualidade dos *datasets* antes das fases de processamento, análise e treinamento de modelos.

Dessa forma, a ferramenta contribui para a identificação de problemas de qualidade ainda nas etapas iniciais do desenvolvimento, reduzindo sua propagação para fases posteriores do fluxo de trabalho.

5.2 Estrutura do Projeto

O projeto foi organizado de forma modular, separando os componentes responsáveis pela configuração da ferramenta, execução das heurísticas, processamento dos dados e geração dos resultados.

Os *datasets* utilizados nos testes são armazenados em um diretório específico, enquanto as heurísticas permanecem organizadas em módulos independentes. Essa estrutura facilita a manutenção da aplicação e permite a adição de novas verificações sem alterações significativas nos demais componentes do sistema.

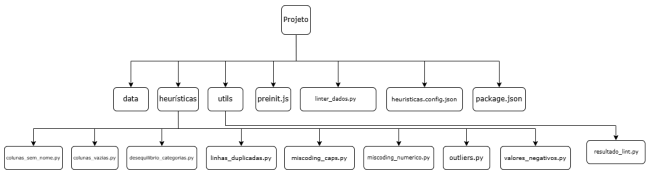

Figura 3: Estrutura do projeto

5.3 Arquitetura da Solução

A arquitetura da ferramenta foi desenvolvida de forma modular, utilizando Node.js para configuração das análises e Python com PySpark para processamento dos dados.

O fluxo inicia-se com a definição dos *datasets* e heurísticas a serem utilizados. Em seguida, as configurações são carregadas pelo módulo principal, responsável pela leitura dos dados, inicialização da sessão Spark e execução das verificações implementadas.

As heurísticas são carregadas dinamicamente durante a execução, permitindo a inclusão de novas validações sem modificações na arquitetura principal da solução.

5.4 Fluxo de execução

O fluxo de execução foi desenvolvido para automatizar a verificação dos *datasets* durante o processo de desenvolvimento. Inicialmente, o usuário seleciona os arquivos e heurísticas que serão utilizados na análise.

Durante o processo de *commit*, o Husky executa automaticamente as verificações configuradas. O sistema então realiza a leitura dos dados, executa as heurísticas selecionadas e apresenta os resultados ao usuário.

Caso divergências sejam identificadas, o processo de *commit* pode ser interrompido, evitando a integração de dados potencialmente problemáticos ao projeto.

Figura 4: Fluxo de execução da ferramenta

5.5 Carregamento Dinâmico de Heurísticas

As heurísticas implementadas na ferramenta são carregadas dinamicamente durante a execução, permitindo a inclusão de novas validações sem modificações na arquitetura principal da aplicação.

Essa abordagem favorece a modularidade e a extensibilidade da solução, facilitando sua evolução e adaptação a diferentes cenários de uso.

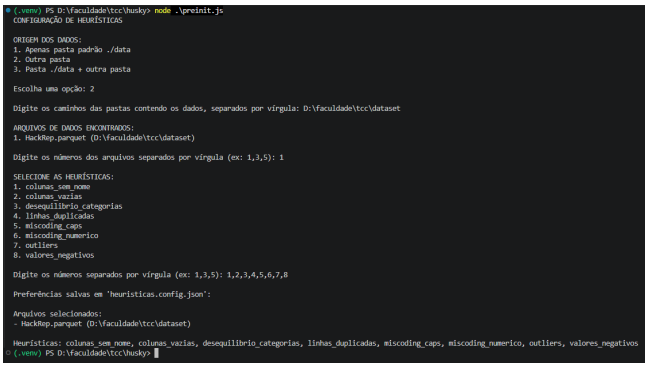

Figura 5: Carregamento dinâmico das heurísticas

5.6 Processamento com PySpark

Inicialmente, a ferramenta utilizava a biblioteca Pandas para processamento dos *datasets*. Com o objetivo de aumentar a capacidade de processamento e compatibilidade com cenários de maior volume de dados, a solução foi adaptada para utilização do PySpark [13].

Além do processamento distribuído, a ferramenta oferece suporte aos formatos CSV e Parquet, permitindo sua utilização em diferentes cenários de análise e engenharia de dados.

5.7 Heurísticas Implementadas

As heurísticas implementadas têm como objetivo identificar possíveis problemas de qualidade nos *datasets* analisados. As validações foram inspiradas nos conceitos apresentados por Hynes et al. [8] e adaptadas para o contexto da solução desenvolvida neste trabalho.

Cada heurística é executada de forma independente e retorna informações sobre os problemas identificados, utilizando um formato padronizado de saída. Essa abordagem facilita a manutenção da ferramenta e permite a inclusão de novas verificações sem alterações significativas em sua arquitetura. As subseções seguintes apresentam as heurísticas implementadas na solução proposta.

Colunas sem nome

A heurística de colunas sem nome identifica colunas geradas incorretamente durante processos de importação de dados, normalmente associadas ao padrão `Unnamed`. Esse tipo de ocorrência pode indicar problemas estruturais no *dataset*. Durante a execução, a validação verifica os nomes das colunas do *DataFrame* e gera alertas quando padrões suspeitos são encontrados.

Colunas vazias

A heurística de colunas vazias identifica atributos compostos totalmente por valores nulos. Esse tipo de problema pode indicar falhas de coleta, integração ou preparação dos dados. A validação compara a quantidade de valores nulos com o total de registros da coluna, gerando alertas quando ocorrências suspeitas são identificadas.

Linhas duplicadas

A heurística de linhas duplicadas detecta registros repetidos no *dataset*, problema que pode comprometer análises estatísticas e modelos de aprendizado de máquina. A validação compara a quantidade total de registros com a quantidade de linhas distintas presentes no *DataFrame*, retornando exemplos das duplicidades identificadas.

Desequilíbrio de categorias

A heurística de desequilíbrio de categorias identifica distribuições excessivamente dominantes em atributos categóricos, auxiliando na detecção preliminar de possíveis problemas de representatividade dos dados. Durante a execução, o sistema calcula a frequência relativa das categorias e gera alertas quando valores ultrapassam um limite previamente definido. Uma categoria é considerada potencialmente desequilibrada quando sua frequência relativa ultrapassa 90% dos registros válidos da coluna.

Miscoding numérico

A heurística de *miscoding* numérico identifica colunas textuais compostas majoritariamente por valores numéricos, mas que apresentam registros inconsistentes. A validação realiza tentativas de conversão para tipos numéricos e retorna exemplos de valores incompatíveis encontrados no *dataset*.

Miscoding de capitalização

A heurística de *miscoding* de capitalização identifica variações na escrita de valores categóricos relacionadas ao uso de letras maiúsculas e minúsculas. Durante a execução, os valores textuais são comparados em formato padronizado, permitindo detectar representações semanticamente equivalentes escritas de formas diferentes.

Outliers

A heurística de detecção de *outliers* identifica valores extremos em colunas numéricas do *dataset*. A implementação utiliza o método do Intervalo Interquartil (IQR) para definição dos limites inferiores e superiores da distribuição dos dados. Para melhorar o desempenho, a validação utiliza funcionalidades de cálculo aproximado de quantis disponíveis no PySpark. São considerados *outliers* os valores localizados abaixo de $Q1 - 1{,}5 \times IQR$ ou acima de $Q3 + 1{,}5 \times IQR$.

Valores negativos

A heurística de valores negativos identifica ocorrências de números menores que zero em colunas numéricas. Embora possam ser válidos em determinados contextos, valores negativos podem indicar problemas relacionados à coleta ou preparação dos dados. Quando ocorrências são encontradas, o sistema retorna exemplos e quantidade de registros associados ao problema identificado.

6 Resultados

6.1 Resultados Obtidos

Os resultados apresentados nesta seção referem-se à execução da ferramenta sobre 27 *datasets* em formato CSV provenientes do benchmark *MLE-Bench*, utilizados para a avaliação quantitativa das heurísticas implementadas. Os experimentos realizados com o *dataset* HackRep em formato Parquet tiveram caráter complementar, sendo utilizados apenas para verificar a compatibilidade da solução com esse formato de armazenamento e com o ambiente baseado em PySpark.

A Tabela 3 apresenta a consolidação dos resultados obtidos. Ao todo, foram realizadas 1477 validações, das quais 616 (41,7%) foram classificadas como conformes (*OK*) e 861 (58,3%) resultaram em alertas de qualidade (*LINT*). Não foram observados erros de execução durante o processamento dos arquivos avaliados.

Tabela 3: Ocorrências por heurística

Heurística	OK	LINT	ERRO	Total
Colunas sem nome	91	0	0	91
Colunas vazias	87	4	0	91
Desequilíbrio de categorias	78	80	0	158
Linhas duplicadas	71	20	0	91
Miscoding de capitalização	62	40	0	102
Miscoding numérico	85	144	0	229
Outliers	63	504	0	567
Valores negativos	79	69	0	148
Total	**616**	**861**	**0**	**1477**

Observa-se que a heurística de detecção de *outliers* apresentou a maior quantidade de ocorrências, totalizando 504 ocorrências classificadas como LINT, o que corresponde a aproximadamente (58,5%) de todos os alertas identificados.

Também se destacaram as verificações relacionadas a *miscoding* numérico, com 144 ocorrências (16,7%), e desequilíbrio de categorias, com 80 ocorrências (9,3%). A Figura 6 apresenta a distribuição percentual dos alertas identificados pelas heurísticas implementadas.

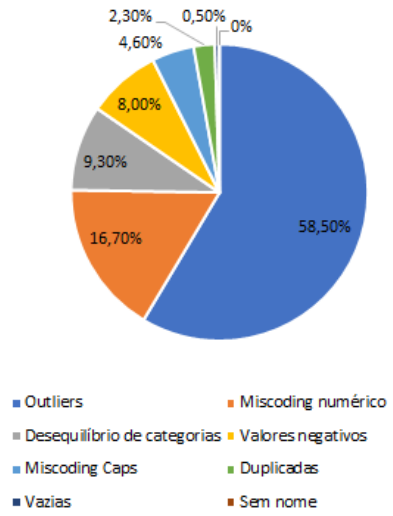

Figura 6: Percentual de ocorrências por heurística

Por outro lado, não foram identificadas ocorrências relacionadas a colunas sem nome, enquanto problemas como colunas vazias e registros duplicados apresentaram baixa incidência nos *datasets* avaliados.

Além da avaliação quantitativa realizada com os datasets do benchmark MLE-Bench, a ferramenta foi executada com sucesso sobre o dataset HackRep em formato Parquet. O experimento confirmou a compatibilidade da solução com esse formato de armazenamento e com o ambiente baseado em PySpark, sem ocorrência de erros durante o processamento.

7 Discussão

O objetivo deste trabalho foi propor uma ferramenta automatizada para apoio à verificação de *datasets* em etapas iniciais do ciclo de vida dos dados. Os resultados obtidos demonstraram que a solução foi capaz de identificar diferentes tipos de anomalias e potenciais falhas em conjuntos de dados amplamente utilizados em aplicações de aprendizado de máquina, indicando seu potencial como mecanismo de apoio à validação preliminar.

A identificação de 861 alertas em 27 *datasets* do benchmark *MLE-Bench* reforça a importância de mecanismos automatizados de inspeção de dados. Em ambientes de engenharia de dados, DataOps e MLOps, falhas relacionadas aos dados podem propagar-se ao longo do *pipeline*, impactando etapas de preparação, treinamento e avaliação de modelos [1, 10, 11]. A utilização de verificações automatizadas contribui para a identificação precoce de possíveis anomalias, reduzindo o esforço necessário para correções em fases posteriores do desenvolvimento.

Outro aspecto relevante está relacionado à flexibilidade da solução proposta. A possibilidade de selecionar heurísticas específicas e incorporar novas verificações de forma modular permite adaptar a ferramenta a diferentes cenários de uso e requisitos de validação. Essa característica é particularmente importante em contextos nos quais diferentes conjuntos de dados apresentam necessidades distintas de análise.

Além da avaliação realizada com os *datasets* do benchmark *MLE-Bench*, a execução bem-sucedida do *dataset* HackRep em formato Parquet demonstrou a compatibilidade da ferramenta com formatos amplamente utilizados em ambientes de engenharia de dados e com o ambiente baseado em PySpark. O uso dessa tecnologia amplia o potencial de aplicação da solução em cenários que demandam processamento distribuído e maiores volumes de dados, embora a presente avaliação não tenha incluído análises específicas de desempenho ou escalabilidade.

Os resultados obtidos pelas heurísticas indicaram a presença de diferentes tipos de ocorrências, incluindo valores extremos, inconsistências de representação e desequilíbrios de categorias. Entretanto, os alertas gerados não devem ser interpretados automaticamente como erros, uma vez que determinados padrões podem ser aceitáveis dependendo do contexto de aplicação. Dessa forma, a ferramenta deve ser entendida como um mecanismo de apoio à análise, complementando a avaliação realizada por especialistas.

Em conjunto, esses resultados sugerem que a ferramenta foi capaz de identificar automaticamente diferentes tipos de ocorrências de qualidade em *datasets* de naturezas distintas, atendendo ao objetivo proposto neste trabalho.

As heurísticas implementadas neste trabalho não foram desenvolvidas especificamente para cenários de Big Data. O uso do PySpark teve como objetivo ampliar a compatibilidade da solução com formatos e ambientes frequentemente empregados em contextos de processamento distribuído.

Diferentemente de abordagens que dependem da configuração explícita de verificações para cada conjunto de dados, como ocorre em diversas ferramentas de qualidade de dados, a solução proposta permite a execução automática de heurísticas previamente implementadas durante o fluxo de desenvolvimento. Essa característica reduz o esforço inicial de configuração e favorece sua utilização em cenários nos quais se deseja realizar verificações preliminares de forma rápida e integrada ao processo de desenvolvimento.

Uma limitação deste trabalho é a ausência de um conjunto de referência contendo anotações formais sobre problemas identificados nos *datasets* avaliados. Assim, não foi possível comparar diretamente os alertas gerados com um conjunto previamente validado, limitando uma avaliação quantitativa da precisão das heurísticas implementadas.

Além disso, embora as heurísticas tenham sido inspiradas no trabalho de Hynes et al. [8], os experimentos foram realizados sobre conjuntos de dados distintos daqueles utilizados pelos autores. Dessa forma, não é possível estabelecer uma comparação quantitativa direta entre os resultados obtidos neste trabalho e aqueles apresentados no estudo original.

8 Conclusão

Este trabalho apresentou o desenvolvimento de uma ferramenta automatizada para verificação da qualidade de dados inspirada nos conceitos do *Data Linter*. A proposta teve como objetivo apoiar a identificação de problemas de qualidade em *datasets* por meio da execução automática de heurísticas integradas ao fluxo de desenvolvimento.

A ferramenta desenvolvida adota uma abordagem modular para verificação automatizada da qualidade de dados, permitindo a execução de diferentes heurísticas durante etapas iniciais do fluxo de desenvolvimento. A integração ao processo de *pre-commit* possibilita que potenciais problemas sejam identificados antes que os dados sejam propagados para etapas posteriores do *pipeline*, contribuindo para processos de validação mais sistemáticos.

Os experimentos demonstraram a capacidade da ferramenta em executar verificações automatizadas sobre arquivos nos formatos CSV e Parquet, identificando diferentes tipos de problemas de qualidade em conjuntos de dados utilizados em aplicações de aprendizado de máquina. Além disso, sua arquitetura permite a inclusão de novas heurísticas sem alterações significativas no núcleo da aplicação, favorecendo sua adaptação a diferentes contextos de uso.

Essa característica tem potencial para aumentar a confiabilidade dos conjuntos de dados utilizados em aplicações orientadas a dados, incluindo cenários de aprendizado de máquina, engenharia de dados e sistemas de inteligência artificial. Ao possibilitar a identificação precoce de desvios e potenciais problemas de qualidade, a ferramenta contribui para reduzir o risco de utilização de dados inadequados em processos analíticos e de treinamento de modelos.

Uma limitação da ferramenta é que as verificações realizadas são genéricas e independentes do contexto específico de aplicação dos dados. Dessa forma, os alertas gerados não devem ser interpretados automaticamente como erros, mas sim como indícios que podem demandar análise adicional por parte do usuário, considerando o domínio e os objetivos associados ao conjunto de dados analisado.

Como trabalhos futuros, sugere-se ampliar o conjunto de heurísticas disponíveis, incorporar novas verificações de qualidade dos dados, gerar relatórios visuais automaticamente e integrar a ferramenta a plataformas de automação utilizadas em ambientes corporativos.

Conclui-se, portanto, que o objetivo proposto foi alcançado, resultando em uma ferramenta capaz de apoiar a verificação automatizada da qualidade de dados e contribuir para a identificação precoce de potenciais problemas em diferentes conjuntos de dados.

Disponibilidade do Código-Fonte

O código-fonte da ferramenta desenvolvida está disponível em [12], contendo os scripts, heurísticas e instruções necessárias para reprodução dos experimentos.

Referências

[1] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning. *Proceedings of Machine Learning and Systems* 1 (2019), 334–347.

[2] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. 2024. MLE-Bench: Evaluating Machine Learning Agents on Machine Learning Engineering. *arXiv preprint arXiv:2410.07095* (2024). arXiv:2410.07095 [cs.LG] https://arxiv.org/abs/2410.07095

[3] Qing Chen, Fuling Sun, Xinyue Xu, Zui Chen, Jiazhe Wang, and Nan Cao. 2021. VizLinter: A Linter and Fixer Framework for Data Visualization. *arXiv preprint arXiv:2108.10299* (2021). arXiv:2108.10299 https://arxiv.org/abs/2108.10299

[4] dbt Labs. 2025. dbt Documentation. https://docs.getdbt.com/ Accessed: 2026-06-16.

[5] Harald Foidl, Michael Felderer, and Rudolf Ramler. 2022. Data Smells: Categories, Causes and Consequences, and Detection of Suspicious Data in AI-based Systems. In *Proceedings of the 1st Conference on AI Engineering – Software Engineering for AI (CAIN '22)*. Association for Computing Machinery (ACM), New York, NY, USA, 229–239. doi:10.1145/3522664.3528590

[6] Great Expectations. 2025. Great Expectations Documentation. https://greatexpectations.io/ Accessed: 2026-06-16.

[7] Sjoerd Halmans, Lavinia Paganini, Alexander Serebrenik, and Alexander Nolte. 2026. HackRep: A Large-Scale Dataset of GitHub Hackathon Projects. *arXiv preprint arXiv:2603.29672* (2026). arXiv:2603.29672 [cs.SE] https://arxiv.org/abs/2603.29672

[8] Nick Hynes, D. Sculley, and Michael Terry. 2017. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. In *Proceedings of the NIPS 2017 Workshop on Machine Learning Systems (MLSys)*. Long Beach, California, USA. http://learningsys.org/nips17/assets/papers/paper_19.pdf

[9] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. 2022. MLOps: Overview, Definition, and Architecture. In *Proceedings of the International Conference on AI Engineering – Software Engineering for AI (CAIN '22)*. Association for Computing Machinery (ACM), New York, NY, USA, 39–45. doi:10.1145/3522664.3528595

[10] Sasu Mäkinen, Henrik Skogström, Eero Laaksonen, and Tommi Mikkonen. 2021. Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?. In *2021 IEEE/ACM 1st Workshop on AI Engineering – Software Engineering for AI (WAIN)*. IEEE, 109–112. doi:10.1109/WAIN52551.2021.00024

[11] Aiswarya Raj Munappy, David Issa Mattos, Jan Bosch, Helena Holmström Olsson, and Anas Dakkak. 2020. From Ad-Hoc Data Analytics to DataOps. In *Proceedings of the International Conference on Software and System Processes (ICSSP '20)*. Association for Computing Machinery, Seoul, Republic of Korea. doi:10.1145/3379177.3388909

[12] Juliano Abi Thomas. 2026. TCC Data Linter. https://github.com/julianothomas/tcc_data. Acesso em: 18 jun. 2026.

[13] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (2016), 56–65. doi:10.1145/2934664