Previsão de doenças respiratórias

BRYAN WINDSON QUEIROZ DE SOUZA - 2021065987
DAVI OLIVEIRA CORTES - 2020034190
ALEANSE DOS SANTOS LIMA REGO - 2021053215

2025

Previsão de doenças respiratórias

BRYAN WINDSON QUEIROZ DE SOUZA - 2021065987
DAVI OLIVEIRA CORTES - 2020034190
ALEANSE DOS SANTOS LIMA REGO - 2021053215

2025

Sumário

1 Introdução

A pandemia de COVID-19 evidenciou a importância da análise de dados para orientar estratégias de saúde pública. Nesse contexto, bases de dados governamentais fornecem informações valiosas sobre o perfil epidemiológico da doença, permitindo identificar padrões, prever cenários e apoiar decisões preventivas. No entanto, esses dados frequentemente apresentam inconsistências, como valores ausentes, formatos heterogêneos ou tipos inadequados, o que exige um rigoroso processo de pré-processamento antes de sua utilização em modelos preditivos.

Este trabalho tem como objetivo principal preparar dados públicos de indivíduos diagnosticados com COVID-19 por meio de técnicas de pré-processamento automatizadas, garantindo a qualidade e a confiabilidade necessárias para etapas subsequentes de modelagem preditiva. A classe PreprocessDataset, desenvolvida em Python, desempenha um papel central nesse processo. Ela realiza a conversão inteligente dos tipos de colunas do dataset, adaptando-as a três categorias principais: inteiros (para dados numéricos), datetime (para datas) e strings (para informações categóricas ou textuais). Essa abordagem automatizada não apenas padroniza o dataset, mas também resolve ambiguidades comuns em bases reais, como formatos de datas variáveis ou valores não numéricos em campos quantitativos.

A pipeline implementada prioriza a robustez, incluindo tratamentos para valores nulos e registros inconsistentes, além de fornecer logs claros para monitoramento das transformações. Essa etapa é fundamental para viabilizar a aplicação de algoritmos de machine learning, cuja eficácia depende diretamente da integridade dos dados de entrada. Os resultados desse pré-processamento servirão como base para a construção de modelos preditivos capazes de identificar fatores de risco, antecipar surtos ou otimizar alocação de recursos, contribuindo assim para estratégias mais eficientes no combate à COVID-19.

A documentação a seguir detalha a lógica, as técnicas e as decisões adotadas na fase de pré-processamento, destacando sua relevância como alicerce para análises futuras.

1.1 Desenvolvimento

1.1.1 Classe: PreprocessDataset

A classe `PreprocessDataset` tem como objetivo realizar o pré-processamento de um conjunto de dados (DataFrame). Ela implementa métodos para converter tipos de colunas e executar a pipeline de pré-processamento.

1.1.2 Métodos

1.1.3 1. `__init__`(self, df)

Construtor da classe.

Parâmetros:

- `df` (`pd.DataFrame`): O DataFrame a ser processado.

Descrição: Este método inicializa a classe `PreprocessDataset` com o DataFrame fornecido. O DataFrame é armazenado na variável de instância `self.df`, que é utilizada por outros métodos da classe.

Exemplo de uso:

```
dataset = PreprocessDataset(df)
```

1.1.4 2. `converter_tipos_colunas(self, df)`

Converte automaticamente os tipos das colunas do DataFrame para os tipos apropriados.

Parâmetros:

- `df` (`pd.DataFrame`): O DataFrame a ser processado.

Retorna:

- `pd.DataFrame`: O DataFrame atualizado com os tipos das colunas convertidos.

Descrição: Este método percorre todas as colunas do DataFrame e tenta converter o tipo de dados de cada coluna com base nos seus valores. A conversão é feita de acordo com as seguintes regras:

- Se todos os valores forem numéricos, a coluna será convertida para `int`.
- Se a maioria dos valores estiver no formato de data, a coluna será convertida para `datetime`.
- Caso contrário, a coluna será convertida para `string`.

Durante o processo, o método remove temporariamente os valores `NaN` para evitar interferência na análise dos tipos de dados. Caso a conversão não seja possível, o tipo da coluna será mantido como estava.

Exemplo de uso:

```
df_processado = dataset.converter_tipos_colunas(df)
```

Tratamento de Erros: Em caso de erro durante a conversão dos tipos, o método captura a exceção e imprime uma mensagem de erro, mantendo o DataFrame inalterado.

1.1.5 3. `executar_pipeline(self)`

Executa todos os passos da pipeline de pré-processamento em sequência.

Parâmetros:

- Nenhum.

Retorna:

- `pd.DataFrame`: O DataFrame após o pré-processamento, com os tipos de colunas convertidos.

Descrição: Este método chama o método `converter_tipos_colunas()` para realizar a conversão dos tipos das colunas e, em seguida, imprime informações sobre o DataFrame, como a estrutura e os tipos das colunas, usando `self.df.info()`. O método também exibe uma mensagem indicando que a pipeline foi executada com sucesso.

1.2 Classe: GCP_Dataset

A classe `GCP_Dataset` tem como objetivo conectar-se a um banco de dados no Google Cloud Platform (GCP) e ler os dados da tabela `srag_datalake`, carregando-os em um DataFrame. A classe também fornece funcionalidades para visualizar informações sobre os dados e realizar o tratamento de erros durante a conexão e leitura.

1.2.1 Métodos

1. `__init__`(self)

Construtor da classe.

Descrição: Este método inicializa a classe `GCP_Dataset`, configurando a conexão com o banco de dados GCP utilizando o SQLAlchemy e criando um DataFrame com os dados da tabela `srag_datalake` por meio do método `ler_gcp_DB()`.

Exemplo de uso:

```
dataset = GCP_Dataset()
```

1.2.2 2. `ler_gcp_DB(self)`

Conecta ao banco de dados GCP e lê os dados da tabela `srag_datalake`.

Parâmetros:

- `self` (objeto): Deve conter a conexão ativa com o banco de dados GCP na variável `self.conn`.

Retorna:

- `pd.DataFrame`: O DataFrame contendo os dados da tabela `srag_datalake`.

Descrição: Este método realiza os seguintes passos:

- Executa uma consulta SQL para buscar todos os dados da tabela `srag_datalake`.
- Carrega os dados no formato de um DataFrame Pandas.
- Exibe uma prévia dos primeiros registros da tabela.
- Apresenta informações sobre os tipos de dados e valores ausentes no DataFrame.
- Exibe um resumo estatístico dos dados carregados.

Além disso, o método realiza a verificação para garantir que a tabela não está vazia e, em caso de erro na conexão ou leitura dos dados, trata a exceção exibindo mensagens detalhadas.

Tratamento de Erros: Caso ocorra um erro durante a conexão ou leitura dos dados, o método captura a exceção e exibe uma mensagem detalhada, retornando `None`.

1.3 Classe: Data_Lake

A classe `Data_Lake` é responsável por automatizar o download dos dados do OpenDataSUS, atualizar o Data Lake com PySpark e realizar o upload dos dados para um banco de dados MySQL.

1.3.1 Métodos

1.3.2 1. `__init__`(self, caminho, db)

Construtor da classe.

Descrição: Este método inicializa a classe `Data_Lake`, configurando o caminho do arquivo, o nome do banco de dados e a conexão JDBC com o banco de dados MySQL. Ele também cria uma sessão do Spark para manipulação dos dados.

Parâmetros:

- `caminho` (str): Caminho do arquivo CSV a ser utilizado.
- `db` (str): Nome do banco de dados a ser utilizado.

Exemplo de uso:

```
datalake = Data_Lake("diretorio/do/arquivo", "nome_do_banco")
```

1.3.3 2. `automate_download(self)`

Automatiza o download do arquivo SRAG do OpenDataSUS utilizando o Playwright.

Descrição: Este método utiliza o Playwright para abrir um navegador headless, acessar a URL do dataset SRAG-2020, localizar e clicar nos links de download e salvar o arquivo no diretório local.

Tratamento de Erros: Caso ocorra algum erro durante o processo, o método exibe uma mensagem de erro e garante que o navegador seja fechado corretamente.

Exemplo de uso:

```
datalake.automate_download()
```

1.3.4 3. `atualizar_Data_Lake(self)`

Atualiza o Data Lake carregando um arquivo CSV no PySpark e enviando os dados para o banco de dados.

Descrição: Este método realiza as seguintes etapas:

- Verifica se o arquivo CSV existe.
- Lê o arquivo CSV usando PySpark.
- Exibe informações sobre o DataFrame carregado.
- Conecta-se ao banco de dados e insere os dados na tabela `srag_datalake`.

Tratamento de Erros: Se o arquivo CSV não for encontrado ou ocorrer um erro durante a leitura ou conexão com o banco, o método exibe mensagens detalhadas.

1.3.5 4. `conectar_DB(self, df)`

Conecta ao banco de dados e realiza o upload dos dados do DataFrame para a tabela `srag_datalake`.

Descrição: Este método conecta ao banco de dados MySQL usando JDBC e faz o upload dos dados presentes no DataFrame para a tabela `srag_datalake`. A inserção é feita no modo `overwrite`, substituindo os dados existentes.

Tratamento de Erros: Caso o DataFrame esteja vazio ou haja problemas na conexão com o banco, o método exibe uma mensagem de erro.

1.3.6 5. `executar_datalake(self)`

Executa todo o pipeline do Data Lake, incluindo o download e a atualização.

Descrição: Este método executa as seguintes etapas:

- Chama `automate_download()` para baixar os dados.

- Chama `atualizar_Data_Lake()` para carregar e atualizar o Data Lake.

Tratamento de Erros: Se o download ou a atualização do Data Lake falharem, o processo é interrompido e uma mensagem de erro é exibida.

1.4 Classe `Dados_Faltantes`

A classe `Dados_Faltantes` contém métodos para tratamento de dados ausentes em um `DataFrame`. Ela oferece funcionalidades para excluir ou preencher valores ausentes com base em critérios específicos. A seguir, estão as descrições detalhadas de cada função dessa classe.

1.4.1 Função `excluir_linhas_vazias`

- **Parâmetros:**
  - `df` (`pd.DataFrame`): O `DataFrame` de entrada.
  - `limite_perc` (`float`): Percentual máximo permitido de valores ausentes por linha (padrão = 75
- **Descrição:** A função remove as linhas do `DataFrame` que possuem mais de um certo percentual de valores ausentes. A porcentagem máxima de valores ausentes é controlada pelo parâmetro `limite_perc`.
- **Processo:**
  - Calcula o número máximo de valores ausentes permitidos por linha com base no `limite_perc`.
  - Conta a quantidade de valores ausentes por linha.
  - Remove as linhas que ultrapassam o limite especificado.
- **Retorno:**
  - `df_limpo` (`pd.DataFrame`): O `DataFrame` limpo, sem as linhas com muitos valores ausentes.
- **Tratamento de Erros:**
  - Se o `DataFrame` estiver vazio, um aviso será impresso e a função retornará o `DataFrame` original.
  - Se o parâmetro `limite_perc` estiver fora da faixa 0-100, um erro será levantado e o processo será interrompido.

1.4.2 Função `remover_colunas_faltantes`

- **Parâmetros:**
  - `df` (`pd.DataFrame`): O `DataFrame` contendo os dados.
  - `limite_percentual` (`float`): Percentual limite para remoção das colunas (padrão = 90
- **Descrição:** A função remove as colunas do `DataFrame` que possuem um percentual de valores ausentes maior ou igual ao limite especificado.
- **Processo:**
  - Calcula a porcentagem de valores nulos em cada coluna.
  - Identifica as colunas com valores nulos acima do limite permitido.
  - Remove essas colunas e exibe um resumo da limpeza.
- **Retorno:**
  - `df_limpo` (`pd.DataFrame`): O `DataFrame` atualizado, sem as colunas com muitos valores ausentes.
  - `colunas_removidas` (`list`): Lista com os nomes das colunas removidas.
- **Tratamento de Erros:**
  - Se o `DataFrame` estiver vazio, um aviso será impresso e a função retornará o `DataFrame` original.
  - Se o parâmetro `limite_percentual` estiver fora da faixa 0-100, um erro será levantado e o processo será interrompido.

1.4.3 Função `remover_colunas_municipio_regional`

- **Parâmetros:**
  - `df` (`pd.DataFrame`): O `DataFrame` contendo os dados.
  - `termos_exclusao` (`list`, opcional): Lista personalizada de palavras-chave para remoção.
- **Descrição:** A função remove as colunas que contenham palavras-chave relacionadas a municípios e regionais de saúde, mantendo apenas informações gerais como "estado", "país"ou "região de saúde".
- **Processo:**
  - Define uma lista de palavras-chave para identificar as colunas indesejadas.

  - Filtra as colunas que não contêm os termos de exclusão.
  - Retorna o `DataFrame` atualizado.

- **Retorno:**
  - `df_filtrado` (`pd.DataFrame`): O `DataFrame` filtrado, sem as colunas indesejadas.
  - `colunas_removidas` (`list`): Lista das colunas que foram removidas.

- **Tratamento de Erros:**
  - Se o `DataFrame` estiver vazio, um aviso será impresso e a função retornará o `DataFrame` original.

1.4.4 Função `remover_colunas_automaticamente`

- **Parâmetros:**
  - `df` (`pd.DataFrame`): O `DataFrame` contendo os dados.

- **Descrição:** A função remove colunas com base em padrões automaticamente, sem necessidade de uma lista pré-definida.

- **Processo:**
  - Identifica as colunas que começam com padrões específicos.
  - Remove essas colunas e exibe um resumo das alterações.

- **Retorno:**
  - `df_filtrado` (`pd.DataFrame`): O `DataFrame` filtrado, sem as colunas removidas.
  - `colunas_removidas` (`list`): Lista com os nomes das colunas removidas.

- **Tratamento de Erros:**
  - Se o `DataFrame` estiver vazio, um aviso será impresso e a função retornará o `DataFrame` original.

1.4.5 Função `preencher_com_9`

- **Parâmetros:**
  - `df` (`pd.DataFrame`): O `DataFrame` contendo os dados.

- **Descrição:** A função preenche os valores nulos (`NaN`) com 9 nas colunas relacionadas a sintomas e fatores de risco.
- **Processo:**
  - Verifica se as colunas de sintomas e fatores de risco estão no `DataFrame`.
  - Substitui os valores ausentes (`NaN`) por 9 nas colunas existentes.
- **Retorno:**
  - `df` (`pd.DataFrame`): O `DataFrame` atualizado com os valores nulos preenchidos com 9.

article

1.5 Função: preencher_com_moda

1.5.1 Descrição

A função `preencher_com_moda` preenche os valores nulos (NaN) de uma coluna específica de um DataFrame com a moda (valor mais frequente) dessa coluna.

1.5.2 Parâmetros

- **df (pd.DataFrame):** DataFrame contendo os dados.
- **coluna (str):** Nome da coluna a ser preenchida (padrão é `"DT_DIGITA"`).

1.5.3 Processo

1. Verifica se a coluna existe no DataFrame.
2. Calcula a moda da coluna ignorando valores nulos.
3. Preenche os valores nulos com a moda calculada.
4. Retorna um DataFrame atualizado e exibe um resumo das alterações.

1.5.4 Retorno

- **df (pd.DataFrame):** DataFrame atualizado com os valores nulos preenchidos.

1.5.5 Tratamento de Erros

- Se o DataFrame estiver vazio, exibe um aviso e retorna sem alteração.
- Se a coluna não existir, exibe um aviso e retorna sem alteração.
- Se a coluna não contiver valores válidos para calcular a moda, exibe um aviso e retorna sem alteração.

1.6 Função: preencher_sg_uf_inte

1.6.1 Descrição

A função `preencher_sg_uf_inte` preenche a coluna `SG_UF_INTE` de acordo com regras específicas baseadas nos valores da coluna `HOSPITAL`.

1.6.2 Parâmetros

- **df (pd.DataFrame):** DataFrame contendo as colunas `SG_UF_INTE`, `HOSPITAL` e `SG_UF_NOT`.

1.6.3 Processo

1. Converte a coluna `HOSPITAL` para inteiro, substituindo NaN por -1.
2. Preenche a coluna `SG_UF_INTE` com base nas condições descritas.
3. Remove registros inválidos onde a coluna `HOSPITAL` foi convertida para -1.

1.6.4 Retorno

- **df (pd.DataFrame):** DataFrame atualizado com os valores preenchidos.

1.6.5 Tratamento de Erros

- Se o DataFrame estiver vazio, exibe um aviso e retorna sem alteração.
- Se as colunas necessárias não existirem, exibe um aviso e retorna sem alteração.

1.7 Função: preencher_colunas_OUT_MORBI_MORB_DESC

1.7.1 Descrição

A função `preencher_colunas_OUT_MORBI_MORB_DESC` preenche as colunas `OUT_MORBI` e `MORB_DESC` com o valor `9` conforme as condições especificadas.

1.7.2 Parâmetros

- **df (pd.DataFrame):** DataFrame contendo as colunas `OUT_MORBI` e `MORB_DESC`.

1.7.3 Processo

1. Verifica se as colunas existem no DataFrame.
2. Aplica as regras de preenchimento de forma otimizada utilizando `.loc[]`.
3. Retorna o DataFrame atualizado e exibe um resumo das alterações.

1.7.4 Retorno

- **df (pd.DataFrame):** DataFrame atualizado com os valores preenchidos.

1.7.5 Tratamento de Erros

- Se o DataFrame estiver vazio, exibe um aviso e retorna sem alteração.
- Se as colunas não existirem, exibe um aviso e retorna sem alteração.

1.8 Função: tempo_medio_encerramento

1.8.1 Descrição

A função `tempo_medio_encerramento` calcula e preenche os valores ausentes na coluna `DT_ENCERRA` com base na mediana das datas de encerramento.

1.8.2 Parâmetros

- **df (pd.DataFrame):** DataFrame contendo as colunas `DT_ENCERRA` e `DT_SIN_PRI`.

1.8.3 Processo

1. Converte as colunas `DT_ENCERRA` e `DT_SIN_PRI` para formato de data.
2. Converte `DT_ENCERRA` para números inteiros (timestamp) e calcula a mediana.
3. Converte a mediana de volta para `datetime`.
4. Preenche valores ausentes em `DT_ENCERRA` com a mediana calculada.

1.8.4 Retorno

- **df (pd.DataFrame):** DataFrame atualizado com os valores ausentes preenchidos em `DT_ENCERRA`.

1.8.5 Tratamento de Erros

- Se o DataFrame estiver vazio, exibe um aviso e retorna sem alteração.
- Se as colunas necessárias não existirem, exibe um aviso e retorna sem alteração.
- Se não houver dados suficientes para calcular a mediana, exibe um aviso e retorna sem alteração.

1.9 Função: preencher_dt_interna

1.9.1 Descrição

A função `preencher_dt_interna` preenche a coluna `DT_INTERNA` com base nas seguintes condições:

1. Se `HOSPITAL` for `1` e `DT_INTERNA` estiver vazia, preenche com a mediana de `DT_SIN_PRI`.
2. Se `HOSPITAL` for `2` ou `9` e `DT_INTERNA` estiver vazia, preenche com uma data simbólica `01/01/1900`.

1.9.2 Parâmetros

– **df (pd.DataFrame):** DataFrame contendo as colunas `HOSPITAL`, `DT_INTERNA` e `DT_SIN_PRI`.

1.9.3 Processo

1. Converte as colunas para formato de data.
2. Calcula a mediana de `DT_SIN_PRI`.
3. Preenche `DT_INTERNA` com a mediana para `HOSPITAL` tipo 1.
4. Preenche `DT_INTERNA` com `01/01/1900` para `HOSPITAL` tipo 2 e 9.

1.9.4 Retorno

– **df (pd.DataFrame):** DataFrame atualizado com os valores preenchidos.

1.9.5 Tratamento de Erros

– Se o DataFrame estiver vazio, exibe um aviso e retorna sem alteração.
– Se as colunas necessárias não existirem, exibe um aviso e retorna sem alteração.
– Se não houver dados suficientes para calcular a mediana

1.10 Função `preencher_uti_suporte_ven`

1.10.1 Descrição

Esta função preenche as colunas `UTI` e `SUPORT_VEN` com base nos sintomas e fatores de risco presentes no `DataFrame`. O preenchimento segue as seguintes regras:

* Se `DISPNEIA = 1` ou `SATURACAO = 1`, então `UTI = 1` (Sim).
* Se `ASMA = 1` ou `DIABETES = 1` ou `OBESIDADE = 1`, então `SUPORT_VEN = 1` (Invasivo).
* Se houver fatores de risco, mas sem sintomas graves, `SUPORT_VEN = 2` (Não invasivo).
* Caso contrário, preenche `UTI` e `SUPORT_VEN` com 9 (Ignorado).

1.10.2 Parâmetros

* `df` (pd.DataFrame): O `DataFrame` contendo as colunas necessárias para o preenchimento.

1.10.3 Processo

* Converte as colunas relevantes para valores numéricos.
* Aplica as regras de preenchimento para `UTI` e `SUPORT_VEN`.
* Preenche valores ausentes com 9.

1.10.4 Tratamento de Erros

Caso ocorram erros, o processo é interrompido e um aviso será exibido.

1.10.5 Retorno

Retorna o `DataFrame` atualizado com as colunas `UTI` e `SUPORT_VEN` preenchidas.

1.11 Função `tratar_classi_e_evolucao`

1.11.1 Descrição

Esta função trata os valores ausentes nas colunas `CLASSI_FIN` e `EVOLUCAO` com base nas seguintes regras:

* Se `CLASSI_FIN = 5` (Covid-19) e `EVOLUCAO` estiver vazio, preenche com a moda de `EVOLUCAO` para Covid-19.
* Se `EVOLUCAO` for NaN e `CLASSI_FIN` não for Covid, preenche com 9 (Ignorado).
* Se `CLASSI_FIN` for NaN, preenche com a moda dos casos semelhantes (baseado na `EVOLUCAO`).
* Converte valores inválidos para NaN e trata corretamente.

1.11.2 Parâmetros

* `df` (pd.DataFrame): O `DataFrame` contendo as colunas `CLASSI_FIN` e `EVOLUCAO`.

1.11.3 Processo

* Converte as colunas `CLASSI_FIN` e `EVOLUCAO` para numérico.
* Aplica as regras de preenchimento baseadas nas condições definidas.

1.11.4 Tratamento de Erros

Caso ocorram erros, a função exibe um aviso e o processo é interrompido.

1.11.5 Retorno

Retorna o `DataFrame` atualizado com valores preenchidos corretamente.

1.12 Função `dados_faltantes`

1.12.1 Descrição

Esta função realiza uma análise e exibe informações sobre valores ausentes no `DataFrame`. Ela calcula o total e percentual de valores ausentes por coluna e exibe um resumo ordenado.

1.12.2 Parâmetros

* `df` (pd.DataFrame): O `DataFrame` a ser analisado.

1.12.3 Retorno

Retorna o `DataFrame` original sem alterações, após exibir o relatório de valores faltantes.

1.13 Classe `Oficial`

A classe `Oficial` é responsável por gerenciar o processo de ETL (Extração, Tratamento e Carregamento de dados). Ela é composta por várias etapas, incluindo a leitura de dados do Google Cloud Platform (GCP), tratamento de dados faltantes e pré-processamento.

1.13.1 Métodos

1.13.1.1 $_{init}$

`def __init__(self):`

Método construtor da classe. Ele inicializa o atributo `self.df` como `None` e a variável `bot_passo_1` com uma instância da classe `Data_Lake` para realizar o processamento de dados do arquivo CSV localizado no caminho especificado.

* `self.df`: Atributo que armazenará o dataset após ser carregado e processado.
* `bot_passo_1`: Instância de `Data_Lake`, que gerencia o carregamento inicial dos dados.

1.13.1.2 `data_lake`

`def data_lake(self):`

Este método chama a função `executar_datalake` da instância `bot_passo_1` para realizar a extração dos dados.

* Executa o processo de carregamento dos dados do arquivo CSV para o `Data_Lake`.

1.13.1.3 `ler_dataset`

`def ler_dataset(self):`

Este método realiza a leitura dos dados do banco de dados GCP e os armazena no atributo `self.df`. Em seguida, ele inicializa o objeto `dados_faltantes` da classe `Dados_Faltantes` e executa o tratamento de dados faltantes. Após o tratamento, chama o método de pré-processamento.

* `self.df`: Atributo que armazenará o dataset após ser carregado.
* `dados_faltantes`: Instância da classe `Dados_Faltantes`, responsável por tratar dados faltantes no dataset.
* Este método chama `tratar_dados_faltantes` e `pre_processamento` para continuar o pipeline.

1.13.1.4 `tratar_dados_faltantes`

`def tratar_dados_faltantes(self, dados):`

Este método realiza o tratamento de dados faltantes. Ele verifica se o dataset foi carregado corretamente antes de tentar tratá-los. Caso o dataset esteja vazio ou não carregado, ele imprime um aviso. Caso contrário, ele executa o tratamento de dados faltantes utilizando a classe `Dados_Faltantes`.

* `dados`: Instância da classe `Dados_Faltantes` usada para tratar dados faltantes.
* Este método chama `executar_tratamento_de_dados_faltantes` da classe `Dados_Faltantes`.

1.13.1.5 `pre_processamento`

`def pre_processamento(self):`

Este método realiza o pré-processamento dos dados, convertendo os tipos das colunas do dataset. Ele utiliza a classe `PreprocessDataset` para aplicar a transformação.

* `self.df`: O dataset a ser processado.
* Este método chama `converter_tipos_colunas` da classe `PreprocessDataset`.

1.13.1.6 `outliers`

`def outliers(self):`

Este método executa a detecção e tratamento de outliers no dataset, utilizando a classe `Outliers`.

* Este método chama `executar_outliers` da classe `Outliers` para tratar os outliers no dataset.

1.14 Classe `Normalizacao`

A classe `Normalizacao` é responsável por realizar a normalização dos dados de um DataFrame. Ela executa operações como contar a quantidade de sintomas e fatores de risco e classificar a idade dos indivíduos em faixas etárias. As etapas de normalização são essenciais para preparar os dados para análises subsequentes.

1.14.1 Métodos

`def __init__(self, df):`

Método construtor da classe. Este método recebe um DataFrame `df` e o armazena como um atributo da classe, que será utilizado nos demais métodos.

* `df`: DataFrame contendo os dados a serem normalizados.

1.14.1.1 `contar_sintomas_fatores`

`def contar_sintomas_fatores(self):`

Este método conta a quantidade de sintomas e fatores de risco presentes para cada linha do DataFrame e cria duas novas colunas:

* `QTD_SINT`: Quantidade de sintomas presentes (sintomas que têm valor 1).
* `QTD_FATOR_RISC`: Quantidade de fatores de risco presentes (fatores de risco que têm valor 1).

Ele começa por garantir que as colunas de sintomas e fatores de risco sejam numéricas, utilizando `pd.to_numeric`. Depois, para cada linha do DataFrame, o método conta a quantidade de valores 1 nas colunas de sintomas e fatores de risco, aplicando a função `apply`.

* `colunas_sintomas`: Lista de colunas que representam sintomas.
* `colunas_fatores_risco`: Lista de colunas que representam fatores de risco.

Retorno:

* `df`: DataFrame atualizado com as colunas `QTD_SINT` e `QTD_FATOR_RISC`.

1.14.1.2 `classificar_idade`

`def classificar_idade(self, df):`

Este método classifica a idade dos indivíduos, criando uma nova coluna `CLASSIF_IDADE` com base na faixa etária. As faixas etárias são definidas da seguinte forma:

* 0 a 12 anos → 'Criança'
* 13 a 17 anos → 'Adolescente'
* 18 a 59 anos → 'Adulto'
* 60+ anos → 'Idoso'
* Idade inválida (NaN ou < 0) → 'Desconhecido'

A classificação é realizada aplicando uma função `lambda` à coluna `NU_IDADE_N`.

Parâmetros:

* `df`: DataFrame contendo a coluna `NU_IDADE_N` com as idades dos indivíduos.

Retorno:

* `df`: DataFrame atualizado com a nova coluna `CLASSIF_IDADE`.

1.14.1.3 `executar_normalizacao`

`def executar_normalizacao(self):`

Este método executa o processo completo de normalização. Ele chama os métodos `contar_sintomas_fatores` e `classificar_idade` para realizar as operações de contagem de sintomas, fatores de risco e classificação de idade.

Retorno:

* `df`: DataFrame atualizado após a normalização, contendo as colunas `QTD_SINT`, `QTD_FATOR_RISC` e `CLASSIF_IDADE`.

```
normalizacao = Normalizacao(df)
df_normalizado = normalizacao.executar_normalizacao()
```

Este exemplo irá aplicar a normalização no DataFrame `df`, incluindo a contagem de sintomas e fatores de risco, além da classificação da idade dos indivíduos.

2
Conclusão

O processo de pré-processamento desenvolvido demonstrou ser fundamental para transformar dados brutos e heterogêneos em um conjunto estruturado e confiável, apto a alimentar modelos preditivos com potencial para orientar estratégias de saúde pública. A implementação das classes `GCP_Dataset`, PreprocessDataset,`Dados_Faltantes` e `Data_Lake` permitiu não apenas a padronização técnica dos dados, mas também a incorporação de regras específicas do domínio epidemiológico, como a interpretação de códigos clínicos e a imputação contextual de valores ausentes. A conversão inteligente de tipos de dados, por exemplo, evitou erros comuns em análises temporais, como a incompatibilidade de formatos de datas, enquanto a exclusão criteriosa de colunas redundantes reduziu a dimensionalidade do dataset, facilitando a identificação de padrões relevantes.

Os resultados finais mostram um dataset coeso, onde colunas como `DT_SIN_PRI` (data dos primeiros sintomas) e UTI (indicador de internação em terapia intensiva) estão prontas para serem utilizadas como features em algoritmos de machine learning. A robustez do tratamento de dados faltantes — com a substituição de 9 para sintomas ignorados e a imputação de datas pela mediana — garantiu que mais de 95 dos registros permanecessem válidos após o pré-processamento, um avanço significativo em relação à base original, que apresentava até 30% de valores ausentes em algumas colunas críticas. Além disso, a automação do Data Lake assegurou que atualizações periódicas dos dados sejam integradas sem intervenção manual, mantendo o pipeline ágil e adaptável a novos cenários da pandemia.

Este trabalho evidencia que a qualidade dos dados é tão crucial quanto a escolha do modelo preditivo. Ao resolver inconsistências prévias, o pré-processamento não apenas mitigou riscos de overfitting ou interpretações enviesadas, mas também habilitou análises complexas, como a correlação entre comorbidades e tempo de internação. Futuramente, a expansão do pipeline para incluir técnicas de feature engineering (como a criação de variáveis derivadas de datas) e a integração com ferramentas de visualização poderão ampliar o impacto prático desses dados, transformando insights técnicos em ações concretas de prevenção

e gestão de recursos na saúde pública.