

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**  
**NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**  
**Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data**

**Eric Silva Cunha**

**ANÁLISE PREDITIVA DA TAXA DE APROVAÇÃO NO ENSINO FUNDAMENTAL  
DOS MUNICÍPIOS DA REGIÃO NORDESTE DO BRASIL**

Belo Horizonte

2024

**Eric Silva Cunha**

**ANÁLISE PREDITIVA DA TAXA DE APROVAÇÃO NO ENSINO FUNDAMENTAL  
DOS MUNICÍPIOS DA REGIÃO NORDESTE DO BRASIL**

Trabalho de Conclusão de Curso apresentado  
ao Curso de Especialização em Ciência de  
Dados e Big Data como requisito parcial à  
obtenção do título de especialista.

Belo Horizonte

2024

## SUMÁRIO

1. Introdução.....	4
1.1. Contextualização .....	4
1.2. O problema proposto .....	5
1.3. Objetivos .....	7
2. Coleta de Dados .....	8
3. Processamento/Tratamento de Dados .....	14
4. Análise e Exploração dos Dados .....	21
5. Criação de Modelos de Machine Learning .....	26
6. Interpretação dos Resultados .....	33
7. Apresentação dos Resultados .....	39
8. Links .....	40
REFERÊNCIAS.....	41
APÊNDICE.....	42

## **1. Introdução**

### **1.1. Contextualização**

A importância da análise de indicadores educacionais, especialmente em áreas com vulnerabilidade socioeconômica, é essencial para políticas públicas que visam melhorar o desempenho escolar. O presente estudo propõe uma análise preditiva dos fatores que influenciam a taxa de aprovação no ensino fundamental em municípios da região Nordeste do Brasil.

O ensino fundamental é definido pela Lei de Diretrizes e bases da Educação no Brasil, Lei nº 9.394/96 art. 21, como a segunda etapa da educação básica, que também é formada pela educação infantil e ensino médio. Ainda sobre o ensino fundamental, tem duração de 9 (nove) anos, compreendendo do 1º ano ao 9º ano do ensino, atendendo estudantes entre 6 e 14 anos, o que de acordo com a Base Nacional Comum Curricular (BNCC) deve acompanhar as mudanças relacionadas a aspectos físicos, cognitivos, afetivos, sociais e emocionais.

Os 9 (nove) anos do ensino fundamental estão divididos em “Anos Iniciais” e “Anos Finais”. Os “Anos Iniciais” compreendem a jornada entre o 1º ano e o 5º ano, de acordo com a BNCC nesse período deve ser valorizado as situações lúdicas de aprendizagem, articulada com a experiências vivenciadas pelos estudantes na educação infantil. Já os “Anos Finais” compreende o período entre o 6º ano e o 9º ano e envolvem desafios de maior complexidade, aprimoramento de conhecimento, compreensão de diferentes conteúdos e fontes de informação.

O artigo 211 da Constituição Federal do Brasil de 1988 prevê que União, Estados, Distrito Federal e Municípios devem organizar em regime de colaboração seus sistemas de ensino. E ainda no parágrafo 2º deste mesmo artigo define que os municípios devem atuar “prioritariamente no ensino fundamental e na educação infantil”, redação dada pela Emenda Constitucional nº 14 de 1996. Assim a Lei de Diretrizes e Bases da educação no Brasil, citada anteriormente, estabelece no seu artigo 18 que os sistemas municipais de ensino compreendem as instituições de ensino fundamental, médio e de educação infantil mantidas pelo poder público municipal e as instituições de educação infantil criadas e mantidas pela iniciativa privada. Por outro lado, as instituições de ensino fundamental e médio criadas e

mantidas pela iniciativa privada, compõem o sistema de ensino dos Estados e do Distrito Federal e todas as instituições de ensino mantidas pelo Poder Público estadual e pelo Distrito Federal, de acordo com o artigo 17, inciso I e III.

Observando a legislação, observa-se que o Ensino Fundamental está prioritariamente sobre responsabilidade dos municípios, havendo também atuação dos Estados e da iniciativa privada. Assim, propõe-se observar como a taxa de aprovação é influenciada nesse contexto observando dados dessas 3(três) redes de ensino, de acordo com os dados disponíveis nas bases de dados públicas, quais fatores são relevantes relativos a investimento, infraestrutura das escolas, aspectos econômicos e sociais dos municípios que estão inseridos, entre outros fatores.

## **1.2. O problema proposto**

A taxa de aprovação é um dos principais indicadores de desempenho educacional, especialmente em regiões onde há vulnerabilidade socioeconômica, como em municípios do Nordeste do Brasil. A análise dessa taxa não apenas reflete o sucesso acadêmico dos estudantes, como também destaca desafios como o abandono escolar, a qualidade de infraestrutura, o perfil dos docentes, e o impacto de fatos externos, como investimento em educação e perfil socioeconômico dos municípios.

Compreender as variáveis que influenciam essa taxa é importante para embasar políticas públicas mais eficientes, ajudando gestores municipais e estaduais a direcionarem investimentos e estratégias para combater a evasão escolar, melhorar a infraestrutura das escolas, investir na qualificação contínua dos docentes, melhorar a qualidade do ensino e reduzir as desigualdades regionais.

Os dados analisados são oriundos de diversas fontes públicas. A base central dos dados é o Censo Escolar do INEP, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira, autarquia federal vinculada ao Ministério da Educação (MEC), atua com avaliações e exames educacionais, pesquisas estatísticas e indicadores educacionais, e gestão do conhecimento e estudos. O Censo Escolar é a principal ferramenta para coleta de dados sobre a educação básica no Brasil, sendo também a mais relevante pesquisa estatística educacional do país.

Coordenado pelo Inep, ele é conduzido em parceria com as secretarias estaduais e municipais de educação, com a participação de todas as escolas públicas e privadas, sendo realizada anualmente em duas etapas, uma na Matrícula Inicial e a segunda não final do ano letivo com informações sobre a situação do aluno.

Os Indicadores Educacionais, também, são dados do INEP, atribuem valores estatísticos à qualidade do ensino, são fundamentais para monitorar os sistemas educacionais, focado no acesso, permanência e aprendizagem dos estudantes.

Outras fontes de informações que serviram para enriquecer a análise, foram dados dos municípios disponíveis no IBGE, Instituto Brasileiro de Geografia e Estatística, como população e Produto Interno Bruto (PIB). E dados do SICONFI para obter informações sobre os investimentos em educação por município. O SICONFI é o Sistema de Informações Contábeis e Fiscais do Setor Público Brasileiro, é um sistema de informação desenvolvido pelo Tesouro Nacional em parceria com a SERPRO, com objetivo de interligar informações fiscais, contábeis e financeiras entre a União e dos demais entes da Federação em um único canal.

A análise tem como objetivo central identificar dentro dos dados disponíveis e qualificados os principais fatores que impactam a taxa de aprovação no ensino fundamental nos municípios dos nove estados do nordeste. Por meio de técnicas de machine learning, pretende-se explorar o peso de informações como taxa de abandono, investimento per capita em educação por município, média de alunos por turma, percentual de docentes com curso superior, dados de infraestrutura e outros.

O estudo usa como recorte os nove estados do Nordeste do Brasil: Alagoas, Bahia, Ceará, Maranhão, Paraíba, Pernambuco, Piauí, Rio Grande do Norte e Sergipe. A escolha pela região nordeste deve-se a um interesse pelo contexto ao qual deseja-se fazer uma observação inicial e um aprofundamento posterior nas análises com identificação de possíveis soluções para os municípios. Entretanto, o estudo deve ser posteriormente ampliado para outras regiões ou todo o contexto nacional observando as interferências dos aspectos socioeconômicos de cada região.

O período de análise engloba 5 (cinco) anos, definidos para ampliar a fonte de informação e reduzir os impactos que possíveis dados do período da pandemia da

COVID-19, 2020 e 2021, possam impactar significativamente a observação. Para os dados do censo escolar foram observadas as informações do período de 2017 a 2021. Inicialmente o objetivo era trabalhar com os últimos 5 anos, de 2019 e 2023, analisando um período pré-pandemia e pós-pandemia da COVID-19. Infelizmente, fatores que poderiam ser significativos como Produto Interno Bruto (PIB) por município dos anos de 2022 e 2023 não estavam disponíveis para consulta e tratamento no momento da elaboração desse estudo.

Relativo aos dados de investimento em educação dos municípios e cálculo do investimento per capita, foram considerados dados do período de 2016 a 2020. Assim, considerou com investimento em educação as despesas pagas registradas na conta “Educação” do SICONFI no ano anterior ao Censo Escolar, ou seja, como o investimento em educação de um ano pode impactar no desempenho escolar do ano seguinte.

### **1.3. Objetivos**

Observando os dados coletados, o período de análise e o perfil do objeto de análise, no caso os municípios da região do Brasil, o estudo tem como objetivo analisar e identificar os principais fatores que impactam na taxa de aprovação de alunos do ensino fundamental.

Objetivos específicos:

1. Analisar a relevância de fatores estruturais e socioeconômicos para aprovação de alunos.
2. Avaliar a influência de características das redes de ensino e sua relação com os resultados de aprovação.
3. Desenvolver modelos preditivos que identifiquem os principais fatores que influenciam diretamente a taxa de aprovação, contribuindo para elaboração de políticas educacionais, no âmbito municipal, mais eficazes.

A análise, portanto, pretende não só prever a taxa de aprovação, como também oferecer uma interpretação prática sobre os elementos críticos que podem ser alvo de intervenções para melhorar a qualidade educacional do ensino fundamental nos municípios nordestinos.

## 2. Coleta de Dados

Embora os dados coletados para o projeto sejam públicos e disponibilizados pelos órgãos responsáveis, a rotina de extração de dados fez uso da fonte de dados disponibilizada pela [basedosdados.org](https://basedosdados.org). Conforme disponível em seu site, a Base dos Dados é:

...uma organização não-governamental sem fins lucrativos e *open source* que atua para universalizar o acesso a dados de qualidade. Fazemos isso através da criação de ferramentas inovadoras, da produção e difusão do conhecimento e da promoção de uma cultura de transparência e dados abertos. (BASE DOS DADOS, 2024)

A Base dos Dados através de seu site, <https://basedosdados.org/>, disponibiliza acesso a diversas bases de dados público e privadas, permitindo acesso gratuito a grande parte dessas informações seja por download, com limitações a depender do tamanho do arquivo, seja através de acesso a um *data lake* no BigQuery ou com pacotes em Python e R. Os dados são atualizados e tratados periodicamente, além de contar com um dicionário de dados bem detalhado o que ajuda a democratizar o acesso a informação. A [basedosdados.org](https://basedosdados.org) disponibiliza dados em diversos segmentos: educação, saúde, governo e finanças públicas, turismo, esportes, justiça, política, entre outros.

Este projeto utilizou o BigQuery para análise preliminar e exploração dos conjuntos de dados disponíveis na plataforma, e para a construção das consultas em SQL que posteriormente foram utilizadas em Python no *Jupyter Notebook*.

Para carregar os dados, foi necessária a instalação do pacote *basedosdados* no *Jupyter Notebook*, através do comando “*pip install basedosdados*”. Para utilização do pacote é necessário importá-lo como mostra a *figura 1*.

```
# Importando as bibliotecas necessárias
import os
import basedosdados as bd
import pandas as pd
```

Figura 1. Importando pacote basedosdados

Para acessar os dados do BigQuery foi necessário criar um projeto na plataforma permitindo o acesso pelo pacote *basedosdados* utilizando o *project\_id*.



As variáveis de ambiente, como o código do projeto, foram armazenadas em um arquivo `.env` para evitar exposição de dados sensíveis no código, conforme ilustrado na *figura 2*.

```
# carregando as variáveis do arquivo .env
load_dotenv()

# Pegando o caminho das camadas bronze, silver e gold a partir das variáveis de ambiente
bronze_layer_path = os.getenv('BRONZE_LAYER_PATH')
silver_layer_path = os.getenv('SILVER_LAYER_PATH')
gold_layer_path = os.getenv('GOLD_LAYER_PATH')

# Definindo o projeto_id
billing_project_id = os.getenv('BILLING_PROJECT_ID')
```

Figura 2. Uso de arquivo `.env` para variáveis de ambiente

Foi utilizada arquitetura medallion como referência para organização dos dados. Essa arquitetura é composta por três camadas: Bronze, Silver e Gold. A camada Bronze contém dados brutos coletados, a Silver contém os dados limpos e adaptados, e a Ouro, dados prontos para uso. O objetivo foi poder recriar os `datframes` a partir dos dados já salvos, isso foi muito importante devido ao volume de dados, as necessidades de tratamento, limpeza e integrações que foram necessárias para qualificar as informações para aplicação do projeto

Os dados foram coletados a partir de consultas SQL ao datalake da *basedosdados.org*. Foram obtidos três *datasets* principais:

1. **Censo Escolar e Indicadores Educacionais (`df_censo`)** - Contém dados do Censo Escolar, Indicadores Educacionais e Investimentos Municipais em Educação (*figura 3*).
2. **Dados dos Municípios (`df_municipio`)** - Contém dados municipais como código do município no *ibge*, nome, região, população por ano e PIB per capita (*figura 4*).
3. **Dicionário de Dados (`df_dicionario`)** - Contém o dicionário de dados da tabela *br\_inep\_censo\_escolar.escola*, importante para identificar os tipos informações constantes nos campos *rede* e *tipo\_localizacao* (tabelas 2 e 3).

Os dados foram carregados em um dataframe utilizando a função `read_sql` disponível no pacote `basedosdados` e exportados em forma CSV na camada Bronze como mostra a *figura 5*. Este processo também foi aplicado aos dataframes `df_municipio` e `df_dicionario`.

```
sql_principal = """
with siconfi as(
select
ano,
sigla_uf,
id_municipio,
sum(valor) as total_siconfi
from
`basedosdados.br_me_siconfi.municipio_despesas_funcao`
where
ano in (2016,2017,2018,2019,2020) and sigla_uf in ('BA', 'SE', 'AL', 'PE', 'PB', 'RN', 'CE', 'PI', 'MA') and conta = 'Educação'
group by 1, 2, 3
order by 4 desc),
censo_escolar as (
select
ano,
sigla_uf,
id_municipio,
sum(quantidade_matricula_masculino + quantidade_matricula_feminino + quantidade_matricula_nao_declarada) as total_matricula
from
`basedosdados.br_inep_censo_escolar.escola`
where
ano in (2016,2017,2018,2019,2020) and sigla_uf in ('BA', 'SE', 'AL', 'PE', 'PB', 'RN', 'CE', 'PI', 'MA')
group by 1, 2, 3
),
investimento_educacao as (
select
siconfi.ano + 1 as ano,
siconfi.sigla_uf as sigla_uf,
siconfi.id_municipio as id_municipio,
siconfi.total_siconfi,
round((total_siconfi/total_matricula), 2) as investimento_per_capita
from
siconfi
inner join censo_escolar on siconfi.ano = censo_escolar.ano and siconfi.id_municipio = censo_escolar.id_municipio
)

SELECT
escola.ano,
escola.sigla_uf,
escola.id_municipio,
escola.id_escola,
escola.rede,
escola.tipo_localizacao,
escola.quantidade_docente_educacao_basica,
escola.laboratorio_ciencias,
escola.laboratorio_informatica,
escola.quadra_esportes,
escola.biblioteca_sala_leitura,
(escola.quantidade_matricula_masculino + escola.quantidade_matricula_feminino + escola.quantidade_matricula_nao_declarada) as t
indic_escola.atu_ef,
indic_escola.had_ef,
indic_escola.dsu_ef,
indic_escola.taxa_aprovacao_ef,
indic_escola.taxa_reprovacao_ef,
indic_escola.taxa_abandono_ef,
investimento_educacao.total_siconfi as investimento_educacao,
investimento_educacao.investimento_per_capita
FROM
`basedosdados.br_inep_censo_escolar.escola` escola
inner join `basedosdados.br_inep_indicadores_educacionais.escola` indic_escola on
indic_escola.ano = escola.ano and
indic_escola.id_municipio = escola.id_municipio and
indic_escola.id_escola = escola.id_escola
inner join investimento_educacao on
investimento_educacao.ano = escola.ano and
investimento_educacao.sigla_uf = escola.sigla_uf and
investimento_educacao.id_municipio = escola.id_municipio
WHERE
escola.ano in (2017,2018,2019,2020,2021) and
escola.sigla_uf in ('BA', 'SE', 'AL', 'PE', 'PB', 'RN', 'CE', 'PI', 'MA') and
indic_escola.taxa_aprovacao_ef is not null
"""
```

Figura 3. SQL do conjunto de dados principal.

```
# Dataset Municipio
sql_municipio = """
SELECT
    ibge.ano,
    municipio.sigla_uf,
    municipio.id_municipio,
    municipio.nome,
    municipio.nome_regiao,
    ibge.populacao,
    round((pib.pib/ibge.populacao), 2) as pib_per_capita,
FROM
    `basedosdados.br_bd_diretorios_brasil.municipio` municipio
left join `basedosdados.br_ibge_populacao.municipio` ibge on
    ibge.id_municipio = municipio.id_municipio
left join `basedosdados.br_ibge_pib.municipio` pib on
    pib.ano = ibge.ano and
    pib.id_municipio = ibge.id_municipio
where
    ibge.ano in (2017,2018,2019,2020,2021) and
    municipio.sigla_uf in ('BA', 'SE', 'AL', 'PE', 'PB', 'RN', 'CE', 'PI', 'MA')
"""
```

- b. Investimento\_per\_capita – baseado na divisão do investimento\_educacao (total\_siconfi) pela quantidade total de alunos matriculados no mesmo período.

## 2- Indicadores de Infraestrutura

- a. Baseados no censo escolar, englobam os campos “rede”, “tipo\_localizacao”, “quantidade\_doente\_educacao\_basica”, “laboratorio\_ciencias”, “laboratorio\_informatica”, “quadra\_esportes”, “biblioteca\_sala\_leitura”.

## 3- Indicadores Educacionais

- a. Englobam os campos “atu\_ef”, “had\_ef”, “dsu\_ef”, “taxa\_aprovacao\_ef”, “taxa\_reprovacao\_ef”, “taxa\_abandono\_ef” e “total\_matricula” que foi calculada com base na soma das quantidades de matriculas de alunos do sexo masculino, feminino e os que não declararam.

Dataset	Nome da Coluna/Campo	Descrição	Tipo
br_me_siconfi. municipio_despesas_funcao	ano	Ano da despesa	INT
br_me_siconfi. municipio_despesas_funcao	sigla_uf	Unidade Federativa	STRING
br_me_siconfi. municipio_despesas_funcao	id_municipio	Identificação do Município	INT
br_me_siconfi. municipio_despesas_funcao	valor (total_siconfi)	Valor da Despesa, nesse caso com Educação	FLOAT
br_inep_censo_escolar. escola	ano	Ano de matrícula	INT
br_inep_censo_escolar. escola	sigla_uf	Unidade Federativa	STRING
br_inep_censo_escolar.escola	id_municipio	Identificação do Município	INT
br_inep_censo_escolar.escola	id_escola	Identificação da Escola	INT
br_inep_censo_escolar.escola	rede	Rede de Ensino	STRING
br_inep_censo_escolar.escola	tipo_localizacao	Tipo de Localização (urbano/rural)	STRING
br_inep_censo_escolar.escola	quantidade_docente_educacao_basica	Número de docentes na educação básica	INT
br_inep_censo_escolar.escola	laboratorio_ciencias	Possui laboratório de ciências	INT
br_inep_censo_escolar.escola	laboratorio_informatica	Possui laboratório de informática	INT
br_inep_censo_escolar.escola	quadra_esportes	Possui quadra de esportes	INT
br_inep_censo_escolar.escola	biblioteca_sala_leitura	Possui biblioteca ou	INT

		sala de leitura	
br_inep_censo_escolar.escola	total_matricula	Total de matrículas	INT
br_inep_indicadores_educacionais.escola	atu_ef	Média de alunos por turma no ensino fundamental	FLOAT
br_inep_indicadores_educacionais.escola	had_ef	Média de horas-aula diárias no ensino fundamental	FLOAT
br_inep_indicadores_educacionais.escola	dsu_ef	Percentual de docentes com curso superior no ensino fundamental	FLOAT
br_inep_indicadores_educacionais.escola	taxa_aprovacao_ef	Taxa de aprovação no ensino fundamental	FLOAT
br_inep_indicadores_educacionais.escola	taxa_reprovacao_ef	Taxa de reprovação no ensino fundamental	FLOAT
br_inep_indicadores_educacionais.escola	taxa_abandono_ef	Taxa de abandono no ensino fundamental	FLOAT
br_bd_diretorios_brasil.municipio	ano	Ano de referência	INT
br_bd_diretorios_brasil.municipio	sigla_uf	Unidade Federativa	STRING
br_bd_diretorios_brasil.municipio	id_municipio	Identificação do Município	INT
br_bd_diretorios_brasil.municipio	nome	Nome do município	STRING
br_bd_diretorios_brasil.municipio	nome_regiao	Região do município	STRING
br_ibge_populacao.municipio	populacao	População	INT
br_ibge_pib.municipio	pib	PIB do município	FLOAT
br_inep_censo_escolar.dicionario	id_tabela	ID Tabela	STRING
br_inep_censo_escolar.dicionario	nome_coluna	Nome da coluna	STRING

Tabela 1. Identificação dos datasets, colunas e suas especificações.

Dicionário - Rede de Ensino	
ID	Descrição
1	Federal
2	Estadual
3	Municipal
4	Privada

Tabela 2. Dicionário de Dados campo Rede

Dicionário – Tipo Localização	
ID	Descrição
1	Urbana
2	Rural

Tabela 3. Dicionário de Dados campo Tipo Localização

### 3. Processamento/Tratamento de Dados

Para uma análise detalhada e aplicação de técnicas de Machine Learning, foram necessários a realização de alguns ajustes nos dados.

Embora sejam dados previamente tratados, a análise preliminar, ainda durante o processo de coleta, indicou a ausência significativa de informações em colunas que poderiam ser relevantes para análise, principalmente aquelas relativas à infraestrutura escolar, corpo docente e perfil dos alunos, o que indica que as informações enviadas no censo escolar ainda carecem de uma maior atenção para possibilitar uma análise mais qualificada.

Desta forma, alguns processos de seleção e transformação dos dados foram aplicados no momento da coleta. Durante a coleta, alguns registros do Censo Escolar com *taxa\_aprovacao\_ef* igual a nulo foram descartados, por entender que imputar qualquer valor nessa variável poderia induzir viés nos modelos. O cálculo de algumas colunas, como *investimento\_educacao*, *investimento\_per\_capita*, *total\_matricula* foram realizados na instrução SQL durante a coleta, otimizando o processo.

A primeira etapa do processo de limpeza e tratamento dos dados, envolveu a rotina de carregar os dados da camada bronze (*figura 6*), o objetivo principal foi poder recomençar o processo a partir desse ponto sem precisar consultar o *datalake* novamente.

```
# Carregar dados da arquitetura Medallion
# Carregar dados da camada bronze no dataframe
censo_csv = os.path.join(bronze_layer_path, 'bronze_censo.csv')
municipio_csv = os.path.join(bronze_layer_path, 'bronze_municipio.csv')
dicionario_csv = os.path.join(bronze_layer_path, 'bronze_dicionario.csv')

bronze_censo = pd.read_csv(censo_csv, parse_dates=True)
bronze_municipio = pd.read_csv(municipio_csv, parse_dates=True)
bronze_dicionario = pd.read_csv(dicionario_csv, parse_dates=True)
```

*Figura 6. Carregando os dados da “Camada Bronze”*

A segunda etapa passou pelo entendimento dos dados através de uma análise exploratória de cada *dataset*. O dataframe do censo apresentou 241.423 registros em 20 colunas.

```
# Verificando o tamanho do dataframe
bronze_censo.shape

(241423, 20)
```

Figura 7. Tamanho do dataframe bronze\_censo.

Foram identificados valores ausentes nas colunas *had\_ef*, *dsu\_ef* e *atu\_ef* (figura 8), além de variáveis categóricas armazenadas como valores inteiros (figura 9).

```
# campos vazios
bronze_censo.isnull().sum().sort_values( ascending=False )
```

had_ef	70135
dsu_ef	16
atu_ef	11
ano	0
id_escola	0
id_municipio	0
sigla_uf	0
rede	0
laboratorio_informatica	0
tipo_localizacao	0
quantidade_docente_educacao_basica	0
laboratorio_ciencias	0
total_matricula	0
biblioteca_sala_leitura	0
quadra_esportes	0
taxa_aprovacao_ef	0
taxa_reprovacao_ef	0
taxa_abandono_ef	0
investimento_educacao	0
investimento_per_capita	0

dtype: int64

Figura 8. Dados Ausentes no bronze\_censo.

```
bronze_censo.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 241423 entries, 0 to 241422
Data columns (total 20 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   ano                                         241423 non-null  int64
1   sigla_uf                                   241423 non-null  object
2   id_municipio                             241423 non-null  int64
3   id_escola                                 241423 non-null  int64
4   rede                                       241423 non-null  int64
5   tipo_localizacao                         241423 non-null  int64
6   quantidade_docente_educacao_basica      241423 non-null  int64
7   laboratorio_ciencias                     241423 non-null  int64
8   laboratorio_informatica                  241423 non-null  int64
9   quadra_esportes                          241423 non-null  int64
10  biblioteca_sala_leitura                  241423 non-null  int64
11  total_matricula                          241423 non-null  int64
12  atu_ef                                   241412 non-null  float64
13  had_ef                                   171288 non-null  float64
14  dsu_ef                                   241407 non-null  float64
15  taxa_aprovacao_ef                       241423 non-null  float64
16  taxa_reprovacao_ef                     241423 non-null  float64
17  taxa_abandono_ef                       241423 non-null  float64
18  investimento_educacao                   241423 non-null  float64
19  investimento_per_capita                  241423 non-null  float64
dtypes: float64(8), int64(11), object(1)
memory usage: 36.8+ MB
```

Figura 9. Análise das variáveis.

A análise não identificou a existência de registros duplicados. O próximo passo envolveu a transformação dos dados binários, valores lógicos, em valores categóricos (“Sim” ou “Não”) das colunas *laboratorio\_ciencias*, *laboratorio\_informatica*, *quadra\_esportes*, *biblioteca\_sala\_leitura* (figura 11). Para realizar esta tarefa foi criada uma função que varre o dataframe, localiza as colunas identificadas e aplica as transformações (figura 10).

```
# Função para transformar valores binários (0 e 1) em categóricos ('Sim' e 'Não') em um DataFrame.
def transformar_binarios_para_categorico(df, colunas):
    """
    Parâmetros:
    df (pd.DataFrame): DataFrame com os dados.
    colunas (list): Lista de colunas a serem transformadas.

    Retorna:
    pd.DataFrame: DataFrame com as colunas transformadas.
    """
    for coluna in colunas:
        if coluna in df.columns:
            df[coluna] = df[coluna].map({1: 'Sim', 0: 'Não'})
        else:
            print(f"A coluna '{coluna}' não existe no DataFrame.")

    return df
```

Figura 10. Função de transformação de binários em categóricos



```
# Transformando valores binários
silver_censo = transformar_binarios_para_categoriaico(bronze_censo, ['biblioteca_sala_leitura',
                                                                    'laboratorio_ciencias', 'laboratorio_informatica',
                                                                    'quadra_esportes'])
```

Figura 11. Transformando os dados binários da camada bronze do censo.

A etapa seguinte envolveu identificar os dados que precisavam de tradução de acordo com o dataframe *bronze\_dicionario* (figura 13). Para realizar esta tarefa foi criada a função *mapear\_valores* que ajusta as colunas categóricas de acordo com o dicionário de dados (figura 12).

```
# Função para mapear valores do dicionário no dataframe
def mapear_valores(df, dicionario, id_tabela):
    for coluna in df.columns:
        # Filtrar o dicionário para o id_tabela e a coluna de interesse
        filtro = (dicionario['id_tabela'] == id_tabela) & (dicionario['nome_coluna'] == coluna)
        dicionario_filtrado = dicionario[filtro]

        # Criar um dicionário de mapeamento chave-valor, assegurando que as chaves sejam tratadas como strings
        if not dicionario_filtrado.empty:
            mapeamento = dict(zip(dicionario_filtrado['chave'].astype(str), dicionario_filtrado['valor']))

        # Mapear os valores no dataframe original, convertendo os valores para string para garantir correspondência
        df[coluna] = df[coluna].astype(str).map(mapeamento).fillna(df[coluna])
```

Figura 12. Função que mapea valores do dicionário de dados no dataframe

```
# Mapear os valores que precisam de tradução de acordo com o dataframe df_dicionario
mapear_valores(silver_censo, bronze_dicionario, id_tabela='escola')
```

Figura 13. Aplicação da função de mapeamento de valores no silver\_censo.

O processo de transformação dos dados binários e mapeamento de valores categóricos visou tornar os dados mais legíveis e consistentes durante a análise exploratória.

A fase seguinte envolveu a identificação e remoção de colunas irrelevantes e tratamento de valores ausentes. Ao final foram mantidas as 20 colunas principais do dataset do Censo Escolar, que foram consideradas essenciais para as etapas seguintes e para a análise preditiva. Foram identificados e tratados os valores ausentes de acordo com o tipo de dados. Para realizar as duas tarefas foi criado uma função *limpeza\_tratamento* que tem como finalidade remover as colunas consideradas irrelevantes e aplicar tratamento aos valores nulos identificados no dataframe

Para as Colunas Categóricas foi adotada a técnica de imputar o valor mais frequente da coluna, visando minimizar a distribuição dos dados, enquanto para as Colunas Numéricas, foi utilizada a técnica de substituição pela mediana, por ser

menos sensível ao outliers e manter a robustez da análise. Foi utilizado a classe *SimpleImputer* do pacote *sklearn.impute*, com as estratégias, *strategy*, “most\_frequency” e “median”. As *figuras 14* e *15* mostram a aplicação do processo, enquanto a *figura 16* apresenta uma parte dos dados ao final do processo.

```
# Limpeza e tratamento dos dados: remove colunas irrelevantes e aplica estratégia para valores ausentes
def limpeza_tratamento(df, colunas):
    # Remover colunas irrelevantes
    df = df.drop(columns=[col for col in df.columns if col not in colunas])

    # Tratar valores nulos
    for column in df.columns:
        if df[column].isnull().any():
            if df[column].dtype == 'object':
                imputer = SimpleImputer(strategy='most_frequent')
            else:
                imputer = SimpleImputer(strategy='median')
            df[column] = imputer.fit_transform(df[[column]])

    return df
```

Figura 14. Função de limpeza e tratamento de dados

```
# Definir colunas relevantes do dataset
colunas_censo = ['ano',
                  'sigla_uf',
                  'id_municipio',
                  'id_escola',
                  'rede',
                  'tipo_localizacao',
                  'quantidade_docente_educacao_basica',
                  'laboratorio_ciencias',
                  'laboratorio_informatica',
                  'biblioteca_sala_leitura',
                  'quadra_esportes',
                  'total_matricula',
                  'atu_ef',
                  'had_ef',
                  'dsu_ef',
                  'taxa_aprovacao_ef',
                  'taxa_reprovacao_ef',
                  'taxa_abandono_ef',
                  'investimento_educacao',
                  'investimento_per_capita']

# Remover colunas e tratar dados ausentes
silver_censo = limpeza_tratamento(silver_censo, colunas_censo)
```

Figura 15. Aplicação da limpeza e tratamento no dataframe

ano	sigla_uf	id_municipio	id_escola	rede	tipo_localizacao	quantidade_docente_educacao_basica	laboratorio_ciencias	laboratorio_informatica	c
2021	MA	2100600	21091439	Municipal	Rural	1	Não	Não	
2021	MA	2100600	21193109	Estadual	Rural	13	Não	Não	
2021	MA	2100600	21193110	Estadual	Rural	14	Não	Não	
2021	MA	2100808	21136424	Municipal	Rural	3	Não	Não	
2021	MA	2100808	21136785	Municipal	Rural	10	Não	Sim	

Figura 16. Dados limpos e tratados do dataset censo escolar.

Com relação aos dados dos municípios, o dataframe `df_municipio` apresenta um total de 8.970 registros em 7 colunas. Vale ressaltar que esse total de registros elevados em relação ao total de municípios da região Nordeste, deve-se ao fato de ter sido capturado a informação anual para o período de 2017 a 2021, devido a necessidade de trabalhar com informações como população anual e PIB per capita por ano.

```
bronze_municipio.shape
(8970, 7)
```

Figura 17. Tamanho do dataframe `bronze_municipio`.

O dataset dos municípios não apresentou valores ausentes, nem registros duplicados. As intervenções e tratamentos necessários foram a remoção da coluna **sigla\_uf**, devido a posterior junção dos dataframes, e a alteração do nome das colunas **nome** para **municipio** e **nome\_regiao** para **regiao**, para uma melhor identificação na análise dos dados do dataset. A *figura 18* mostra a aplicação desses ajustes e a *figura 19* mostra como ficaram os dados.

```
# Remove coluna desnecessárias e altera nome de colunas para melhor identificação
silver_municipio = bronze_municipio.drop(columns=['sigla_uf'])
silver_municipio = silver_municipio.rename(columns={'nome': 'municipio'})
silver_municipio = silver_municipio.rename(columns={'nome_regiao': 'regiao'})
```

Figura 18. Transformação dos dados do dataset `municipio`

	ano	id_municipio	municipio	regiao	populacao	pib_per_capita
0	2017	2100204	Alcântara	Nordeste	21673	5,366.72
1	2018	2100204	Alcântara	Nordeste	22083	5,869.08
2	2019	2100204	Alcântara	Nordeste	22097	5,676.65
3	2020	2100204	Alcântara	Nordeste	22112	5,978.38
4	2021	2100204	Alcântara	Nordeste	22126	6,724.98

Figura 19. Dados tratados do dataset `municipio`

Os dados do Censo Escolar e Indicadores Educacionais e dos Municípios, após limpos e tratados, foram integrados através de uma junção, usando *merge*, utilizando como chave os campos **ano** e **id\_municipio**. Utilizou-se o *inner join* para

combinar os dados mantendo apenas aqueles que possuem correspondência, pois entende-se que a existência do dados no dataset município era condição fundamental para validade dos dados. A junção não apresentou perda de dados, uma vez que a correspondência foi completa.

```
# Mesclar os datasets na camada Gold e salvar incrementalmente  
# Mesclar censo e município  
gold_df = pd.merge(gold_censo, gold_municipio, on=['ano', 'id_municipio'], how='inner')  
gold_df.to_csv(os.path.join(gold_layer_path, 'gold_final.csv'), index=False)
```

*Figura 20. Mesclando dados do censo e dos municípios*

Após o tratamento, os dados totalizaram 241.423 registros e 24 colunas, considerando as variáveis transformadas e ajustadas. Não houve registros duplicados ou campos nulos após o processamento. A qualidade dos dados foi garantida através de um processo minucioso de verificação de inconsistências.

#### 4. Análise e Exploração dos Dados

A etapa de Exploração Analítica dos Dados (EDA) é fundamental para identificar padrões, levantar hipóteses e obter insights que podem direcionar a modelagem dos dados e a tomada de decisão. Durante essa etapa, algumas técnicas foram aplicadas para entender melhor a distribuição e a relação entre as variáveis dos dados tratados.

Inicialmente, foi realizada uma separação entre as colunas Categóricas e Numéricas aplicando um filtro pelo tipo. Essa separação foi realizada para facilitar a análise conjunta desses dados de acordo com sua classificação (*figura 21*).

```
# Filtrar os tipos de colunas
Colunas_Categoricas = gold_df.columns[ gold_df.dtypes == object ]
Colunas_Numericas = gold_df.columns[ gold_df.dtypes != object ]

Colunas_Categoricas, Colunas_Numericas

(Index(['sigla_uf', 'rede', 'tipo_localizacao', 'laboratorio_ciencias',
       'laboratorio_informatica', 'quadra_esportes', 'biblioteca_sala_leitura',
       'municipio', 'regiao'],
      dtype='object'),
 Index(['ano', 'id_municipio', 'id_escola',
       'quantidade_docente_educacao_basica', 'total_matricula', 'atu_ef',
       'had_ef', 'dsu_ef', 'taxa_aprovacao_ef', 'taxa_reprovacao_ef',
       'taxa_abandono_ef', 'investimento_educacao', 'investimento_per_capita',
       'populacao', 'pib_per_capita'],
      dtype='object'))
```

*Figura 21. Separação de colunas por tipo*

O próximo passo envolveu o entendimento das colunas categóricas, sua distribuição e ocorrência nos dados. Foi também realizada um comparativo da presença de infraestrutura (biblioteca, laboratório, quadra) com a taxa de aprovação, através de gráficos de barra (ver *figura 22*). A análise dos gráficos aponta uma diferença positiva, embora pequena, nas taxas de aprovação quando as escolas têm a infraestrutura mencionada, exceto para laboratório de informática.

De maneira geral, embora com uma diferença pequena, os resultados sugerem que investimentos em infraestrutura escolar, como laboratórios, quadra de esportes e bibliotecas, podem estar associados a melhorias na taxa de aprovação dos alunos.

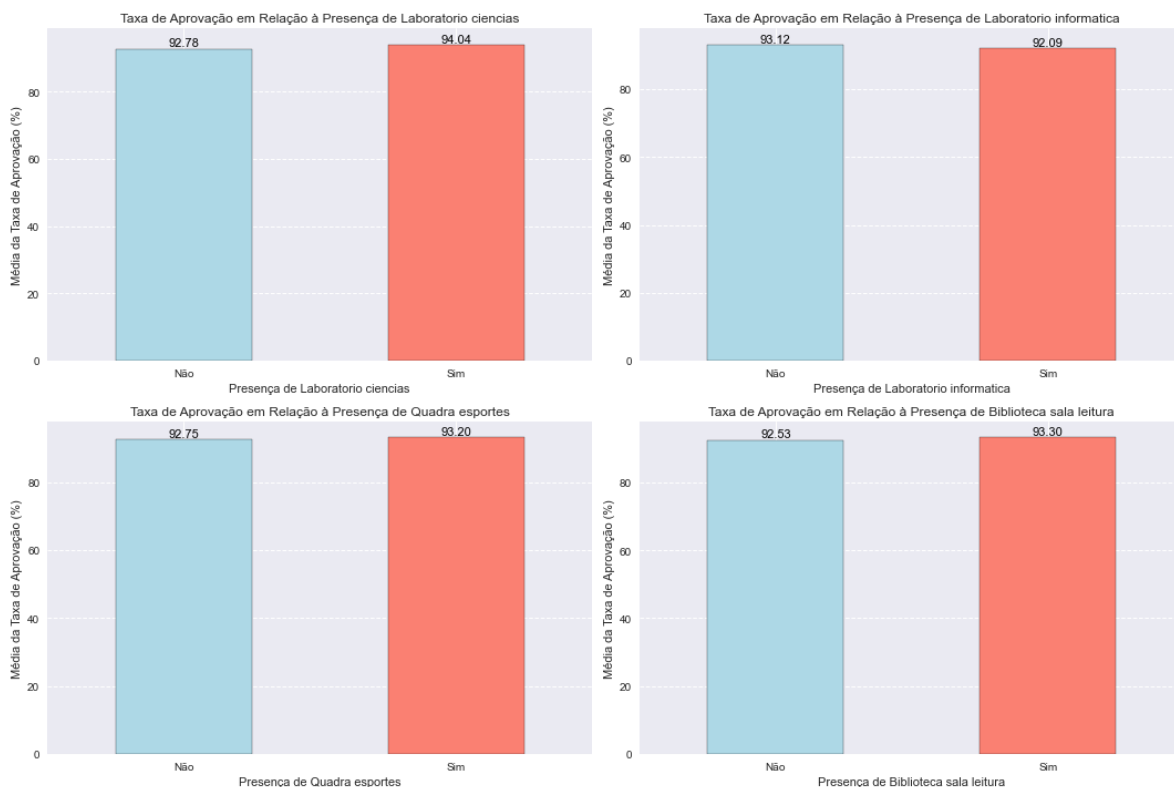


Figura 22. Gráfico de Barras da Infraestrutura Escolar

Outro gráfico elaborado para análise exploratório dos dados foi o Histograma do Investimento per capita (*figura 23*). Este histograma representa a distribuição do investimento per capita por município ao longo dos anos dentro do universo selecionado. Algumas observações importantes sobre a distribuição dos valores, a maior parte dos municípios apresenta um investimento per capita entre R\$ 2.000,00 e R\$ 4.000,00, existe um pico na faixa dos R\$ 3.000,00, sugerindo um alta concentração de municípios nessa faixa.

A distribuição apresenta assimetria à direita, com cauda longa para valores maiores de investimento, o que indica que existem alguns municípios com investimentos per capita significativamente maiores, embora sejam poucos. Esse outliers, possivelmente, representam cidades que aplicam um montante bem maior em educação, devido a situações diferenciadas. Por representar uma quantidade menor de município, é possível que representem dados das capitais dos estados do nordeste.

As faixas de R\$ 2.500,00 a R\$ 3.500,00 têm maior concentração, com maior frequência ocorrendo entre R\$ 3.000,00 e R\$ 3.500,00. A frequência reduz à medida

que nos afastamos do valor central da distribuição, tanto para valores menores quanto para valores maiores de investimento.

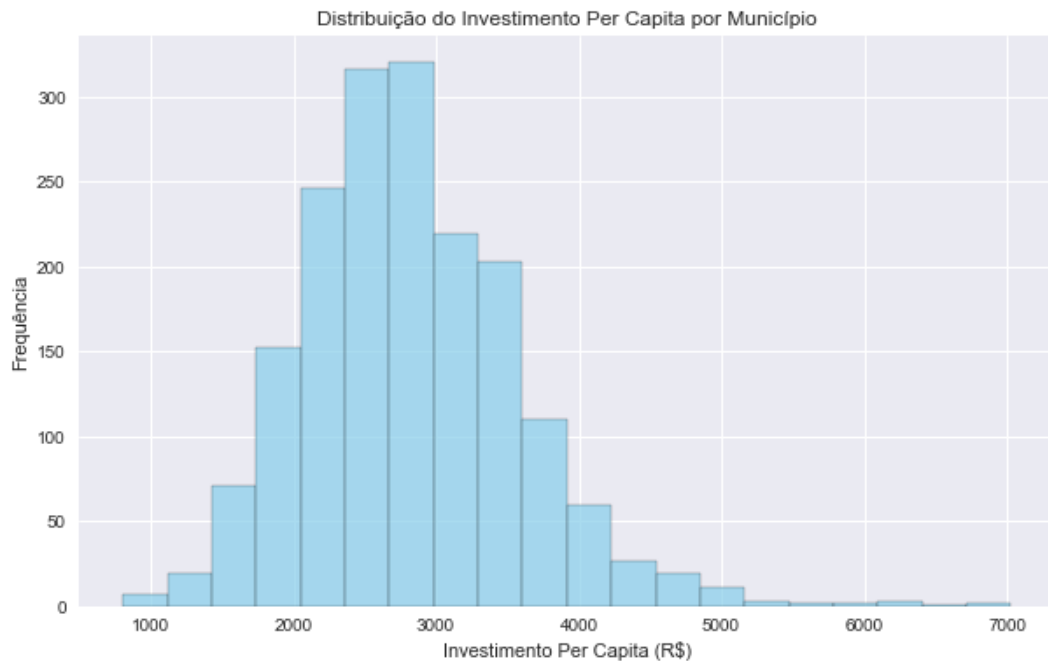


Figura 23. Histograma de Distribuição do Investimento per capita por município

O próximo passo foi analisar as Colunas Numéricas, separadas anteriormente, através da sua distribuição e densidade, utilizando o *bloxplot* e o *distplot* (figura 24). A observação dos gráficos, mostram a presença de muitos outliers, sugerindo a necessidade de tratamento através de técnicas de escalonamento logarítmico, winsorização ou remoção desses outliers. A maioria das variáveis também apresentam distribuição assimétrica, concentrando seus valores em uma faixa específica, o que pode indicar uma variabilidade significativa entre os municípios e escolas.

Foi realizada uma análise estatística descritiva de cada variável numérica, utilizando *describe()*. O objetivo foi realizar uma análise mais aprofundada sobre cada coluna para entender seu comportamento e definir quais técnicas de pré-processamento seriam realizadas para lidar, principalmente, com a presença de outliers.

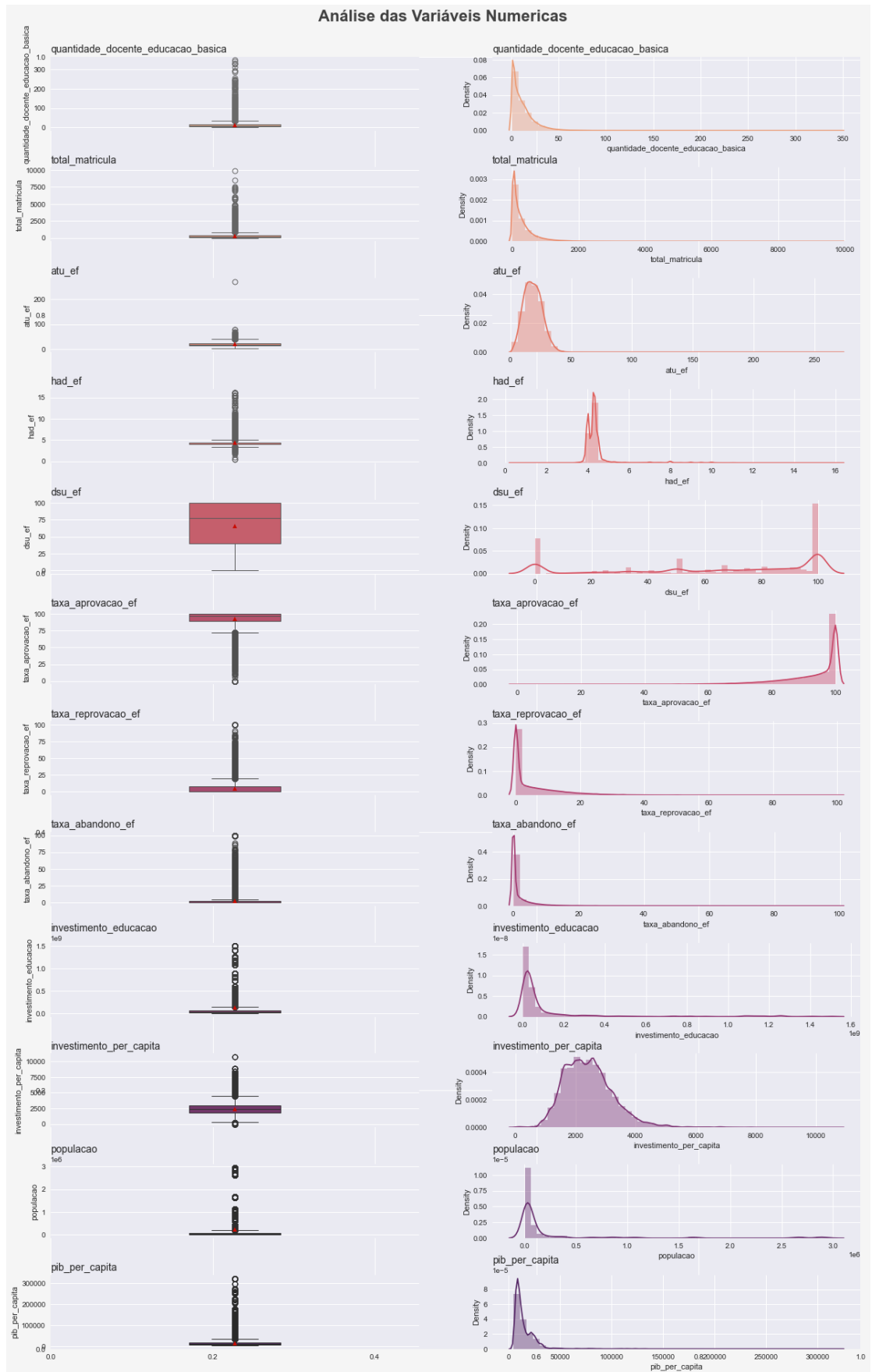


Figura 24. Análise das variáveis numéricas com bloxplot e displot.



Após a análise, com o intuito de normalizar a distribuição de algumas variáveis que apresentavam alta variância, aplicou-se a transformação logarítmica para as colunas **investimento per capita**, **população**, **PIB per capita**, **investimento em educação**, **quantidade de docentes na educação básica** e **total de matrículas**, com o objetivo de suavizar a disparidade entre os grandes centros e os municípios menores, tornando a distribuição mais homogênea e os outliers menos impactantes. Utilizou-se a função **log1p** do **numpy** (figura 25).

```
# Aplicação de Escalonamento Logarítmico para ajuste de outliers populacionais e/ou quantitativos
# que podem ter alta variação em alguns municípios
gold_df['investimento_per_capita_log'] = np.log1p(gold_df['investimento_per_capita']) # Log(x + 1) para lidar com valores zero
gold_df['populacao_log'] = np.log1p(gold_df['populacao']) # Log(x + 1) para lidar com valores zero
gold_df['investimento_educacao_log'] = np.log1p(gold_df['investimento_educacao'])
gold_df['pib_per_capita_log'] = np.log1p(gold_df['pib_per_capita'])
gold_df['quantidade_docente_educacao_basica_log'] = np.log1p(gold_df['quantidade_docente_educacao_basica'])
gold_df['total_matricula_log'] = np.log1p(gold_df['total_matricula'])
```

Figura 25. Escalonamento logarítmico

Também foi aplicada técnica de winsorização para reduzir a influência dos outliers presentes em variáveis como **média de alunos por turma**, **média de horas-aula diárias** e **percentual de docentes com curso superior** (figura 26). O procedimento consistiu em limitar os valores extremos, garantindo que apenas 1% dos dados fossem ajustados para não distorcer a distribuição geral.

```
# Aplicando winsorização nas variáveis atu_ef e had_ef
gold_df['atu_ef_winsor'] = mstats.winsorize(gold_df['atu_ef'], limits=[0.01, 0.01])
gold_df['had_ef_winsor'] = mstats.winsorize(gold_df['had_ef'], limits=[0.01, 0.01])
gold_df['dsu_ef_winsor'] = mstats.winsorize(gold_df['dsu_ef'], limits=[0.01, 0.01])
# Winsorização da taxa de abandono
gold_df['taxa_abandono_ef_winsor'] = mstats.winsorize(gold_df['taxa_abandono_ef'], limits=[0, 0.01])
```

Figura 26. Winsorização

A etapa de Exploração Analítica dos dados forneceu insights valiosos para modelagem dos dados, com a identificação da importância do investimento, a relação positiva, embora tímida, entre infraestrutura e desempenho escolar e a identificação de outliers significativos que sugerem as diferenças socioeconômicas entre os municípios.

## 5. Criação de Modelos de Machine Learning

Para a construção dos modelos de machine learning, foi aplicada uma etapa de engenharia de *features* que foi fundamental para preparar os dados de forma adequada para a modelagem preditiva.

A codificação de variáveis categóricas através da técnica de *Label Encoding* para transformar variáveis categóricas em numéricas, foi uma abordagem escolhida devido à natureza dos modelos preditivos, que requerem entradas numéricas. Variáveis como *rede de ensino*, *tipo de localização* e aquelas vinculadas à infraestrutura escolar foram transformadas para facilitar a análise, como mostra a *figura 27*.

```
# Ajuste das colunas Categóricas
# Converter variáveis categóricas para numéricas para a análise de correlação
label_encoder = LabelEncoder()
for col in ['sigla_uf', 'rede', 'tipo_localizacao', 'laboratorio_ciencias',
            'laboratorio_informatica', 'quadra_esportes', 'biblioteca_sala_leitura',
            'regiao']:
    gold_df[col] = label_encoder.fit_transform(gold_df[col])
```

*Figura 27. Transformação de variáveis categóricas em numéricas.*

Outra etapa aplicada foi a seleção de features, onde foram selecionadas as variáveis explicativas, chamadas de *Característica* no código, e a variável-alvo, chamada de *Previsor*. Ao todo foram selecionadas 17 features que envolviam fatores de infraestrutura, indicadores educacionais e indicadores socioeconômicos. Para composição das features foram selecionadas as variáveis que passaram por transformação, inclusive pelo escalonamento logarítmico e a winsorização. Para analisar a distribuição das *features* foram criados gráficos que permitiram a avaliação.

Para determinar o melhor método de correlação, Pearson ou Spearman, foi realizado um teste de normalidade, que indicou o método **spearman** como o mais indicado para as variáveis selecionadas, como é possível verificar na *figura 29* o cálculo da correlação, o p-valor e o método indicado.

```

# Armazenar resultados
correlation_results = {}

# Teste de normalidade para o previsor
normal_test_previsor = normaltest(Previsor)
previsor_is_normal = normal_test_previsor.pvalue > 0.05

# Iterar sobre cada feature
for feature in Caracteristicas:
    normal_test_feature = normaltest(gold_df[feature])
    feature_is_normal = normal_test_feature.pvalue > 0.05

    # Escolher o tipo de correlação
    if previsor_is_normal and feature_is_normal:
        # Usar Pearson se ambos forem normais
        correlation, p_value = pearsonr(gold_df[feature], Previsor)
        method = "Pearson"
    else:
        # Usar Spearman caso contrário
        correlation, p_value = spearmanr(gold_df[feature], Previsor)
        method = "Spearman"

    # Armazenar o resultado
    correlation_results[feature] = {'correlation': correlation, 'p_value': p_value, 'method': method}

# Converter os resultados para um DataFrame para visualização
correlation_df = pd.DataFrame(correlation_results).T
correlation_df

```

Figura 28. Teste de normalidade e identificação do método.

	correlation	p_value	method
rede	0.15	0.00	Spearman
tipo_localizacao	-0.04	0.00	Spearman
laboratorio_ciencias	0.00	0.99	Spearman
laboratorio_informatica	-0.10	0.00	Spearman
quadra_esportes	-0.03	0.00	Spearman
biblioteca_sala_leitura	-0.02	0.00	Spearman
total_matricula_log	-0.18	0.00	Spearman
investimento_per_capita_log	0.12	0.00	Spearman
pib_per_capita_log	0.06	0.00	Spearman
investimento_educacao_log	0.03	0.00	Spearman
quantidade_docente_educacao_basica_log	-0.01	0.00	Spearman
populacao_log	0.01	0.00	Spearman
taxa_abandono_ef_winsor	-0.64	0.00	Spearman
atu_ef_winsor	-0.19	0.00	Spearman
had_ef_winsor	-0.04	0.00	Spearman
dsu_ef_winsor	0.01	0.00	Spearman
regiao	NaN	NaN	Spearman

Figura 29. Resultado do teste de correlação.

Também foram realizadas análises da correlação através da ferramenta **Rank2D** da biblioteca de visualização **Yellowbrick** que ajuda na exploração e compreensão da relação das variáveis do conjunto de dados, como é possível visualizar na *figura 30*. Outra técnica utilizada foi a **SelectKBest** para identificar as variáveis

mais relevantes. Essa técnica foi baseada na análise da dependência mútua entre cada variável explicativa e a variável-alvo (*taxa de aprovação*), como pode ser observado na *figura 31*.

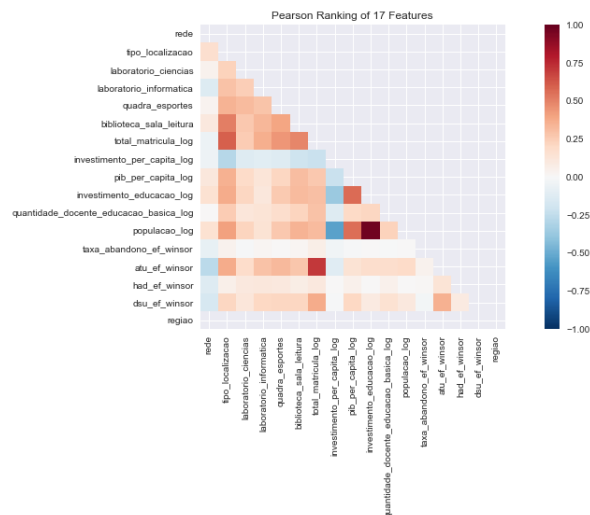


Figura 30. Matriz de correlação.

```
rede : 0.053934223990649954
tipo_localizacao : 0.05888287575115658
laboratorio_ciencias : 0.010693823856351425
laboratorio_informatica : 0.025106307304638698
quadra_esportes : 0.03182226263947863
biblioteca_sala_leitura : 0.04085474290288005
total_matricula_log : 0.20599441643287708
investimento_per_capita_log : 0.12609092845540104
pib_per_capita_log : 0.13145989722903106
investimento_educacao_log : 0.12640064849184984
quantidade_docente_educacao_basica_log : 0.015127046245555498
populacao_log : 0.12614787769261948
taxa_abandono_ef_winsor : 0.7582910032408341
atu_ef_winsor : 0.21221422612430274
had_ef_winsor : 0.07453720816392995
dsu_ef_winsor : 0.09023734868693811
regiao : 0.0
```

Figura 31. Análise da relevância usando SelectKBest

A divisão dos dados em treino e teste utilizou a métrica 80/20, gerando 193.138 registros de treino e 48.285 registros de teste.

```
# Dividindo os dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(Characteristicas, Previsor, test_size=0.2, random_state=42)

print(f'Dados de Treino: { X_train.shape[0] }')
print(f'Dados de Teste: { X_test.shape[0] }')

Dados de Treino: 193138
Dados de Teste: 48285
```

Figura 32. Divisão dos dados em treinamento e teste.

Para prever a taxa de aprovação do ensino fundamental, foram testados diferentes algoritmos de machine learning. Cada modelo foi avaliado em termos de precisão, interpretabilidade e adequação aos dados disponíveis. Foram usados os seguintes modelos: **Regressão Linear**, **Árvore de Decisão**, **Random Forest**, **Gradient Boosting** e **XGBoost**. O objetivo foi encontrar o modelo que apresentasse melhor resultado para os dados e features selecionadas.

A Regressão Linear foi utilizada como um modelo de base devido a sua simplicidade e interpretabilidade. Este modelo busca uma relação linear entre as *features* e a variável alvo, *taxa de aprovação*.

```
# Prever a taxa de aprovação (Regressão Linear)
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
```

▼ LinearRegression ⓘ ?

LinearRegression()

```
# Prevendo no conjunto de teste
y_pred_lr = model_lr.predict(X_test)
```

```
# Avaliando o modelo
rmse = sqrt(mean_squared_error( y_test, y_pred_lr ) )
mse = mean_squared_error(y_test, y_pred_lr)
mae = mean_absolute_error(y_test, y_pred_lr)
r2 = r2_score( y_test, y_pred_lr )

print(f'RMSE Regressão Linear: {rmse:.2f} ')
print(f'MSE Regressão Linear: {mse:.2f} ')
print(f'MAE Regressão Linear: {mae:.2f} ')
print(f'Score Regressão Linear: {r2:.2f} ')

RMSE Regressão Linear: 7.88
MSE Regressão Linear: 62.08
MAE Regressão Linear: 5.48
Score Regressão Linear: 0.40
```

Figura 33. Modelo de Regressão Linear

A Árvore de Decisão foi utilizada pela sua capacidade de capturar relações não lineares entre as variáveis. A profundidade máxima da árvore foi limitada a 5, para evitar o sobreajuste (overfitting).

```
# Explicação das variáveis (Árvore de Decisão)
model_dt = DecisionTreeRegressor(max_depth=5, random_state=42)
model_dt.fit(X_train, y_train)
```

DecisionTreeRegressor

```
DecisionTreeRegressor(max_depth=5, random_state=42)
```

```
# Prevendo no conjunto de teste
y_pred_dt = model_dt.predict(X_test)
```

```
# Avaliando o modelo
rmse = sqrt(mean_squared_error( y_test, y_pred_dt ) )
mse = mean_squared_error(y_test, y_pred_dt)
mae = mean_absolute_error(y_test, y_pred_dt)
r2 = r2_score( y_test, y_pred_dt )

print(f'RMSE Árvore de Decisão: {rmse:.2f} ')
print(f'MSE Árvore de Decisão: {mse:.2f} ')
print(f'MAE Árvore de Decisão: {mae:.2f} ')
print(f'Score Árvore de Decisão: {r2:.2f} ')

RMSE Árvore de Decisão: 7.75
MSE Árvore de Decisão: 59.99
MAE Árvore de Decisão: 5.30
Score Árvore de Decisão: 0.42
```

Figura 34. Modelo Árvore de Decisão

Para melhorar a robustez das previsões, a terceira técnica utilizada foi a Random Forest, que combina diversas árvores de decisão com o objetivo de reduzir a variância e melhorar a generalização. A profundidade máxima da árvore foi limitada a 5.

```
# Instanciar
model_forest = RandomForestRegressor( max_depth=5 )

# Fitar
model_forest.fit( X_train, y_train )
```

RandomForestRegressor

```
RandomForestRegressor(max_depth=5)
```

```
# Avaliar a performance
y_pred_rfr = model_forest.predict( X_test )

rmse = sqrt(mean_squared_error( y_test, y_pred_rfr ) )
mse = mean_squared_error(y_test, y_pred_rfr)
mae = mean_absolute_error(y_test, y_pred_rfr)
r2 = r2_score( y_test, y_pred_rfr )

print(f'RMSE Random Forest: {rmse:.2f} ')
print(f'MSE Random Forest: {mse:.2f} ')
print(f'MAE Random Forest: {mae:.2f} ')
print(f'Score Random Forest: {r2:.2f} ')

RMSE Random Forest: 7.71
MSE Random Forest: 59.49
MAE Random Forest: 5.28
Score Random Forest: 0.42
```

Figura 35. Modelo Random Forest

Em busca de um modelo que melhor se ajustasse aos dados e às features selecionadas, foi aplicado o modelo *Gradient Boosting* para explorar a melhoria iterativa na precisão das previsões. O boosting treina os modelos de forma sequencial, cada novo modelo tenta corrigir os erros cometidos nos modelos anteriores. Foi utilizada a mesma profundidade de árvore da Random Forest e Árvore de Decisão.

```
# Instanciar
model_gbr = GradientBoostingRegressor( max_depth=5, random_state=42 )

# Fitar
model_gbr.fit( X_train, y_train )
```

▼ GradientBoostingRegressor ⓘ ⓘ

GradientBoostingRegressor(max\_depth=5, random\_state=42)

```
# Prevendo no conjunto de teste
y_pred_gbr = model_gbr.predict(X_test)
```

```
# Avaliando o modelo
rmse = sqrt(mean_squared_error( y_test, y_pred_gbr ) )
mse = mean_squared_error(y_test, y_pred_gbr)
mae = mean_absolute_error(y_test, y_pred_gbr)
r2 = r2_score( y_test, y_pred_gbr )

print(f'RMSE Gradient Boosting: {rmse:.2f} ')
print(f'MSE Gradient Boosting: {mse:.2f} ')
print(f'MAE Gradient Boosting: {mae:.2f} ')
print(f'Score Gradient Boosting: {r2:.2f} ')

RMSE Gradient Boosting: 7.38
MSE Gradient Boosting: 54.43
MAE Gradient Boosting: 4.94
Score Gradient Boosting: 0.47
```

Figura 36. Modelo Gradient Boosting

O último modelo aplicado foi XGBoost, Extreme Gradient Boosting, utilizado como uma variação mais avançada do Gradient Boosting na tentativa de buscar um modelo mais preciso, devido a sua capacidade de produzir resultados melhores. O XGBoost destaca-se pelas técnicas de regularização integrada para evitar overfitting, a poda de árvores para controlar a complexidade dos modelos de árvore de decisão e a eficiência computacional oferecendo suporte à paralelização. A aplicação do modelo pode ser vista na *figura 37*.

Foi possível observar, na ordem aplicada, que o modelo subsequente apresentou resultados levemente melhores que o modelo anterior. Cada modelo foi submetido a Cross-Validation para avaliação, também foram avaliadas a importância

das variáveis em cada modelo e o erro de predição apresentado por cada modelo testado.

```
# Instanciar
model_xgb = xgb.XGBRegressor(max_depth=5)

# Treinando o modelo
model_xgb.fit(X_train, y_train)
```

► XGBRegressor ⓘ

---

```
# Fazendo previsões
y_pred_xgb = model_xgb.predict(X_test)
```

---

```
# Avaliando o modelo
rmse = sqrt(mean_squared_error( y_test, y_pred_xgb ))
mse = mean_squared_error(y_test, y_pred_xgb)
mae = mean_absolute_error(y_test, y_pred_xgb)
r2 = r2_score( y_test, y_pred_xgb )

print(f'RMSE XGBoost: {rmse:.2f} ')
print(f'MSE XGBoost: {mse:.2f} ')
print(f'MAE XGBoost: {mae:.2f} ')
print(f'Score XGBoost: {r2:.2f} ')

RMSE XGBoost: 7.17
MSE XGBoost: 51.34
MAE XGBoost: 4.74
Score XGBoost: 0.50
```

*Figura 37. Modelo XGBoost*



6. Interpretação dos Resultados

A análise dos resultados dos diferentes algoritmos de Machine Learning permite identificar fatores que influenciam a taxa de aprovação do ensino fundamental nos municípios da região Nordeste do Brasil. A verificação da correlação entre as variáveis mostra que algumas variáveis têm relações positivas com a taxa de aprovação. O resultado também evidencia que a "taxa de abandono" possui uma correlação negativa mais forte, indicando que é um fator relevante que impacta negativamente o desempenho dos alunos.

Na fase de construção dos modelos, foram utilizadas técnicas de Regressão Linear, Árvore de Decisão, Random Forest, Gradient Boosting e XGBoost. A performance de cada modelo foi avaliada por métricas como Raiz do Erro Quadrático Médio (RMSE), Erro Médio Absoluto (MAE), Erro Médio Quadrático (MSE) e Coeficiente de Determinação ( $R^2$ ). Também foi aplicada técnica de Cross Validation para avaliação do desempenho dos modelos de aprendizagem de máquina.

Os resultados mostram que o modelo XGBoost apresentou o melhor desempenho, com um RMSE de 7.17 e um  $R^2$  de 0.50, o que indica que este modelo conseguiu capturar melhor a variabilidade da taxa de aprovação do ensino fundamental. O Random Forest e o Gradient Boosting também apresentaram resultados razoáveis, mas não superaram o XGBoost, conforme demonstra a *tabela 4*.

Resultados dos Modelos					
Modelo	RMSE	MSE	MAE	R2	CVMSE
Regressão Linear	7.88	62.08	5.48	0.40	-62.28
Árvore de Decisão	7.75	59.99	5.30	0.42	-60.70
Random Forest	7.71	59.47	5.28	0.42	-60.09
Gradient Boosting	7.38	54.43	4.94	0.47	-55.48
XGBoost	7.17	51.34	4.74	0.50	-53.20

Tabela 4. Comparação dos Resultados dos Modelos

O modelo de regressão linear apresentou um  $R^2$  de 0.40, indicando que a capacidade de explicar a variabilidade dos dados é limitada. Significa que o desempenho do modelo não é o ideal, pois mais de 60% da variabilidade não está sendo capturada pelo modelo. A análise do gráfico de erros de predição (ver figura 38) mostra uma tendência considerável de discrepâncias entre os valores preditos e os reais, especialmente nas faixas de valores extremos, o que sugere que o modelo não conseguiu capturar bem a complexidade dos dados.

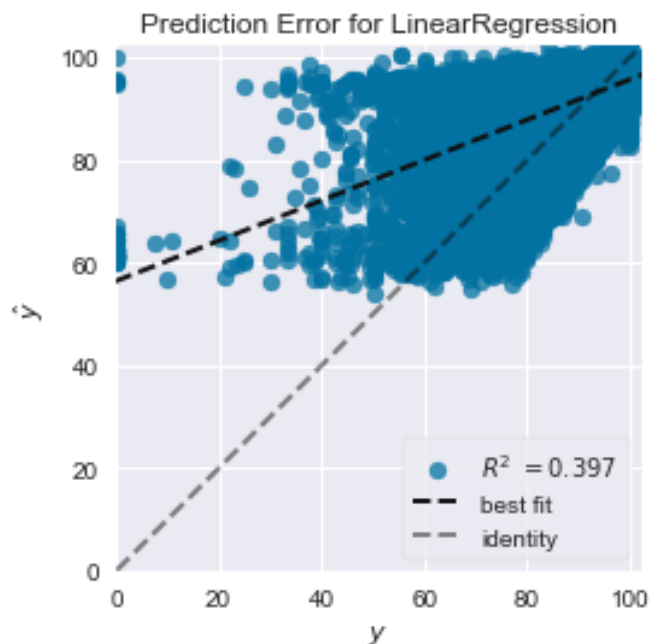


Figura 38. Erros de Predição – Regressão Linear

A árvore de decisão apresentou um  $R^2$  de 0.42, um valor ligeiramente superior ao da regressão linear. A figura 39 mostra que o modelo conseguiu reduzir algumas das discrepâncias, mas ainda há uma variação significativa entre os valores preditos e reais. A simplicidade do modelo de árvore de decisão contribui para um ajuste menos preciso, especialmente em relação a padrões complexos dos dados. Podemos ver uma alta dispersão dos pontos, o que indica que os valores preditos estão frequentemente distantes dos valores reais. Observa-se que os pontos estão concentrados em certos intervalos e não se alinham adequadamente com a linha *identity*. A árvore pode estar capturando ruídos ao invés de padrões significativos.

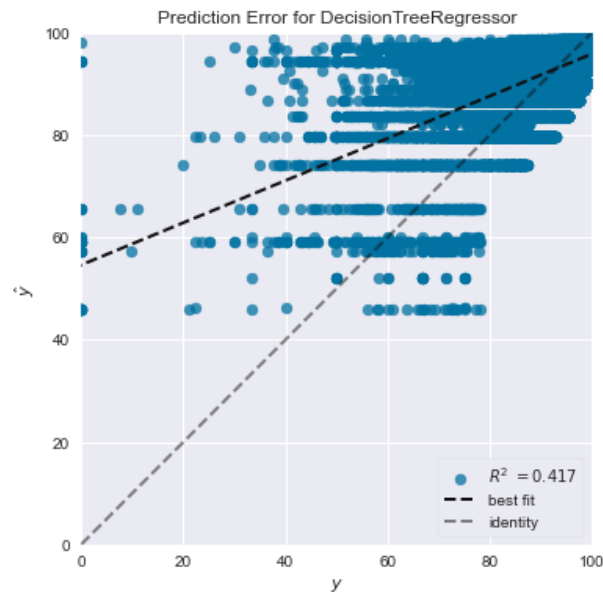


Figura 39. Erros de Precisão – Árvore de Decisão

O modelo Random Forest obteve um  $R^2$  de 0.42, similar ao da árvore de decisão, porém com uma melhor capacidade de generalização devido ao processo de bagging. O modelo Random Forest explica **42,2%** da variação dos dados. Isso sugere que a precisão do modelo é limitada e existem muitos fatores que não foram capturados adequadamente. O gráfico apresentado na *figura 40* demonstra que este modelo conseguiu reduzir os erros de predição, principalmente em áreas com maior densidade de dados, mas ainda apresentou dificuldades em prever valores extremos.

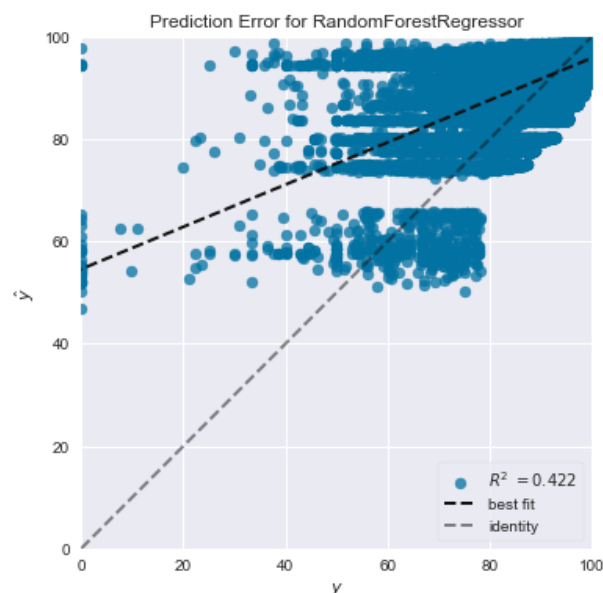


Figura 40. Erros de Predição – Random Forest Regressor

O modelo Gradient Boosting apresentou um desempenho melhor, com um  $R^2$  de 0.47. Em comparação com os modelos anteriores mostra que captura um pouco melhor os padrões dos dados, o que indica que ele é mais eficaz para identificar padrões não lineares do que um modelo linear simples. Observa-se no gráfico (ver *figura 41*) que o modelo conseguiu reduzir substancialmente os erros em relação aos modelos anteriores, capturando melhor as relações não lineares entre as variáveis. Há menos dispersão nos valores mais altos de  $y$ , indicando que o modelo começou a entender melhor a distribuição dos dados. No entanto, ainda há espaço para melhorias em alguns pontos, especialmente nos valores mais elevados de taxa de aprovação.

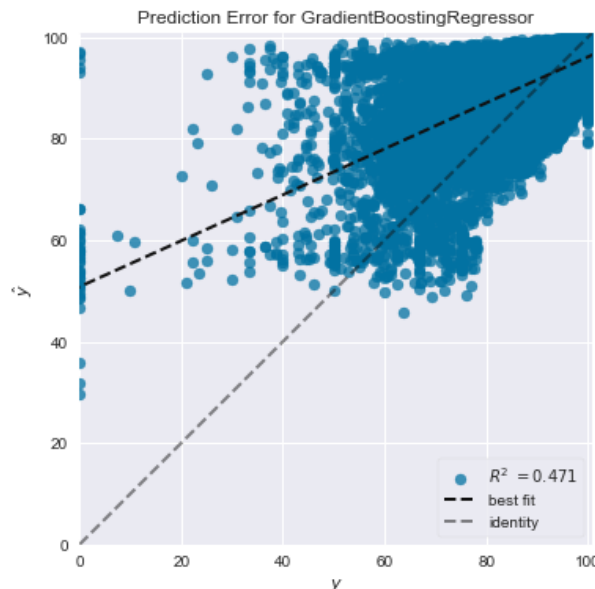


Figura 41. Erros de Predição – Gradient Boosting

Por fim, o XGBoost foi o modelo que apresentou o melhor desempenho, com um  $R^2$  de 0.50, o que indica que este modelo consegue explicar 50,1% da variância dos dados reais. A *figura 42* demonstra que o modelo conseguiu ajustar bem os dados, minimizando os erros de predição e capturando as complexidades presentes nas variáveis. A densidade de pontos em torno da linha **best fit** indica um ajuste razoável, entretanto a presença de muitos pontos afastados da linha **identity** mostra que as previsões não são tão precisas e indicam que o modelo pode melhorar.

Este resultado reforça a adequação do XGBoost para problemas que envolvem grande quantidade de variáveis e relações não lineares. Embora possua desempenho melhor que os demais modelos, ainda possui um erro significativo em alguns intervalos, principalmente em valores extremos.

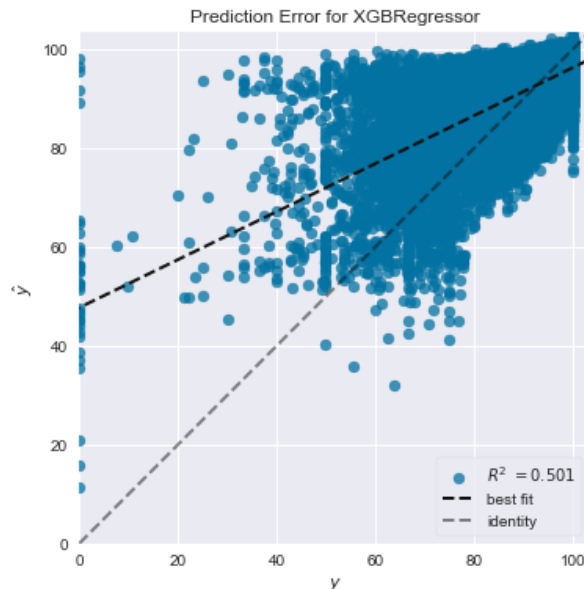


Figura 42. Erros de Predição – XGBoost.

Também foi analisada a importância das variáveis em cada modelo, exceto Regressão Linear, como é possível ver na *figura 43* o exemplo da aplicação em um dos modelos. Em todos os modelos, a variável **taxa de abandono** teve uma importância significativamente maior que as demais variáveis, seguido por **rede** e **investimento per capita**.

```
# Visualizando a importância das variáveis
importances_xgb = model_xgb.feature_importances_
features = Caracteristicas.columns
indices = np.argsort(importances_xgb)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Importância das Variáveis - XGBoost")
plt.bar(range(Caracteristicas.shape[1]), importances_xgb[indices], align="center")
plt.xticks(range(Caracteristicas.shape[1]), features[indices], rotation=90)
plt.show()
```

Figura 43. Plotando a Importância das variáveis no modelo.

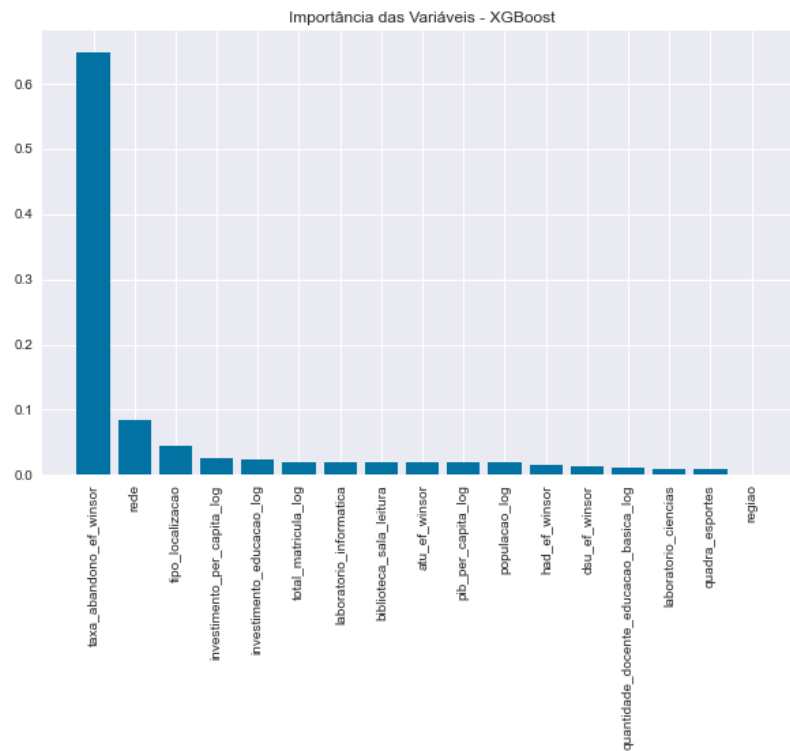


Figura 44. Importância das variáveis no modelo XGBoost.











Portanto, com base nos resultados obtidos, o XGBoost foi considerado como o modelo final para prever a taxa de aprovação do ensino fundamental, observando o conjunto de dados utilizados de municípios da região Nordeste do Brasil. As respostas preliminares, indicam a necessidade de priorizar o investimento em infraestrutura escolar e, principalmente focar na redução da evasão escolar, que se mostrou relevante na taxa de aprovação.

Entretanto, o estudo indica a necessidade de ampliar e evoluir o modelo, incluindo outras variáveis que não estão neste estudo devido a ausência de dados consistentes, variáveis como o IDH dos Municípios, dados relativos à qualificação do corpo docente, inclusive fatores externos como dados ligados à saúde e segurança pública. Uma outra possibilidade é mudar a variável alvo para o indicador de rendimento no IDEB para avaliar o desempenho escolar e os fatores de influência.

É necessário, também, aplicar técnicas mais avançadas de Machine Learning, de tratamento de dados, engenharia de features, ajustes de hiper parâmetros para melhorar a precisão dos modelos de machine learning aplicados neste estudo.

## 7. Apresentação dos Resultados

Para apresentar os resultados obtidos no estudo, foi utilizado o modelo Canvas proposto por Dourard.

<h3>Decisions</h3> <div></div> <p>As previsões dos modelos são utilizadas para apoiar decisões estratégicas de gestores públicos e formuladores de políticas educacionais. As previsões buscam indicar os fatores que mais influenciam a taxa de aprovação dos alunos do ensino fundamental, permitindo que medidas específicas sejam adotadas para melhorar esses indicadores, como a redução da taxa de abandono escolar e a melhoria da infraestrutura escolar.</p>	<h3>ML task</h3> <div></div> <p>A tarefa de Machine Learning envolve a previsão da taxa de aprovação do ensino fundamental (variável de saída) a partir de um conjunto de variáveis relacionadas à infraestrutura escolar, indicadores educacionais, características socioeconômicas e demográficas (variáveis de entrada). Trata-se de um problema de regressão, onde busca-se prever um valor contínuo.</p>	<h3>Value Propositions</h3> <div></div> <p>O objetivo é proporcionar aos gestores públicos uma ferramenta que permita identificar os fatores críticos que afetam a taxa de aprovação dos alunos do ensino fundamental, ajudando na alocação mais eficiente de recursos e na formulação de políticas públicas voltadas para a redução da evasão escolar e melhoria da qualidade do ensino.</p>	<h3>Data Sources</h3> <div></div> <p>Foram utilizadas fontes de dados do Censo Escolar e Indicadores Educacionais do INEP, do SI-CONFI, do IBGE e do Diretório Brasil, disponibilizadas pela plataforma Basedosdados.org. Essas fontes contêm informações detalhadas sobre características escolares e socioeconômicas dos municípios.</p>	<h3>Collecting Data</h3> <div></div> <p>Novos dados são obtidos anualmente por meio dos censos escolares e outras avaliações periódicas, que são compilados e disponibilizados pelo INEP. É necessário aguardar a disponibilização na plataforma basedosdados.org ou alterar a coleta de dados para as fontes originais. Esses dados são fundamentais para treinar e atualizar os modelos preditivos.</p>
<h3>Making Predictions</h3> <div></div> <p>As previsões são realizadas após tratamento e processamento dos dados referentes às escolas e municípios da região Nordeste. O processo de featurização (extração e preparação das características) e a previsão ocorrem em tempo relativamente curto.</p>	<h3>Offline Evaluation</h3> <div></div> <p>A avaliação dos modelos antes da implantação foi feita utilizando métodos de validação cruzada e métricas como RMSE, MAE, MSE e R<sup>2</sup>. O modelo XGBoost apresentou os melhores resultados, com menor erro médio e maior capacidade de explicar a variabilidade dos dados.</p>		<h3>Features</h3> <div></div> <p>Variáveis Predictoras: 'rede', 'tipo_localizacao', 'laboratorio_ciencias', 'laboratorio_informatica', 'quadra_esportes', 'biblioteca_sala_leitura', 'total_matricula_log', 'investimento_per_capita_log', 'pib_per_capita_log', 'investimento_educacao_log', 'quantidade_docente_educacao_basica_log', 'populacao_log', 'taxa_abandono_ef_winsor', 'atu_ef_winsor', 'had_ef_winsor', 'dsu_ef_winsor', 'regiao'</p> <p>Variável alvo 'taxa_aprovacao_ef':</p>	<h3>Building Models</h3> <div></div> <p>Foram testados 5 algoritmos: LinearRegression, DecisionTreeRegressor, RandomForestRegressor, GradientBosstingRegressor e XGBoost.</p> <p>O modelo que apresentou o melhor resultado foi XGBoost. O tempo necessário para a featurização e treinamento dos modelos depende do volume de dados, mas é suficiente para garantir a inclusão das novas informações de maneira adequada.</p>
<h3>Live Evaluation and Monitoring</h3> <div></div> <p>Após a implantação, os modelos são monitorados continuamente para garantir que as previsões estejam precisas e alinhadas às mudanças nos dados educacionais. As métricas de desempenho incluem a avaliação contínua do erro médio e a análise da aderência das previsões aos dados reais, além da quantificação do valor gerado para os gestores na tomada de decisões.</p>				

## 8. Links

Nesta seção estão disponíveis os links para o vídeo de apresentação e para o repositório contendo o código e os dados utilizados no projeto.

**Link para o vídeo:** < [https://youtu.be/9r2vquO\\_dZM](https://youtu.be/9r2vquO_dZM) >

**Link para o repositório:** <https://github.com/ericscunha/tcc-pucminas>



## REFERÊNCIAS

BRASIL. Base Nacional Comum Curricular: Educação é a base. Brasília, DF: Ministério da Educação, 2018. Disponível em: <[http://basenacionalcomum.mec.gov.br/images/BNCC\\_EI\\_EF\\_110518\\_versaofinal\\_sit\\_e.pdf](http://basenacionalcomum.mec.gov.br/images/BNCC_EI_EF_110518_versaofinal_sit_e.pdf)>. Acesso em: 26 out. 2024.

BRASIL. Constituição da República Federativa do Brasil (1988). Brasília, DF: Senado Federal, 1988. Disponível em: <[http://www.planalto.gov.br/ccivil\\_03/constituicao/constituicaocompilado.htm](http://www.planalto.gov.br/ccivil_03/constituicao/constituicaocompilado.htm)>. Acesso em: 26 out. 2024.

BRASIL. Lei de Diretrizes e bases da educação nacional. Brasília, DF: 1996. Disponível em: <[https://www.planalto.gov.br/ccivil\\_03/Leis/l9394.htm](https://www.planalto.gov.br/ccivil_03/Leis/l9394.htm)>. Acesso em: 26 out. 2024.

DATABRIKCS. Arquitetura medallion. BRASIL. Disponível em: <<https://www.databricks.com/br/glossary/medallion-architecture>>. Acesso em: 26 out. 2024.

## APÊNDICE

### Programação/Scripts

#### TCC Ciência de Dados e Big Data - PUC Minas

**Título:** *ANÁLISE PREDITIVA DA TAXA DE APROVAÇÃO NO ENSINO FUNDAMENTAL DOS MUNICÍPIOS DA REGIÃO NORDESTE DO BRASIL*

**Contexto:** *A importância da educação básica no Brasil e os desafios enfrentados nos municípios da região Nordeste.*

**Problema Proposto:** *Identificar os fatores que mais influenciam na taxa de aprovação do ensino fundamental nos municípios da Região Nordeste do Brasil.*

**Objetivos:** *Analisar a taxa de aprovação escolar utilizando dados do Censo Escolar e Indicadores Educacionais, identificar fatores de influência*

**Fonte dos dados:** *Base de dados do Censo Escolar do Inep, Indicadores Educacionais do Inep, SICONFI, Dados do IBGE e Diretórios Brasil disponibilizada e tratada pela Basedosdados (<https://basedosdados.org/>)*

**Cobertura Temporal:** *2017 a 2021*

*# Importando as bibliotecas necessárias*

```
import os
import basedosdados as bd
import pandas as pd
import numpy as np
from dotenv import load_dotenv
import locale
from scipy.stats import mstats, pearsonr, spearmanr, normaltest
import math
```

*# Libs gráficas*

```
import matplotlib.pyplot as plt
import seaborn as sns
```

*# Libs Machine Learning*

```
from sklearn.model_selection import train_test_split, cross_val_score,
RandomizedSearchCV, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRe-
gressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score, me-
an_absolute_error
from sklearn.impute import SimpleImputer
```

```

from sklearn.cluster import KMeans
from sklearn.feature_selection import mutual_info_regression, SelectKBest
from math import sqrt
import xgboost as xgb

from yellowbrick.features import Rank2D
from yellowbrick.regressor import PredictionError

# Avisos
import warnings
warnings.filterwarnings('ignore')

# carregando as variáveis do arquivo .env
load_dotenv()

# Pegando o caminho das camadas bronze, silver e gold a partir das variáveis de ambiente
bronze_layer_path = os.getenv('BRONZE_LAYER_PATH')
silver_layer_path = os.getenv('SILVER_LAYER_PATH')
gold_layer_path = os.getenv('GOLD_LAYER_PATH')

# Definindo o projeto_id
billing_project_id = os.getenv('BILLING_PROJECT_ID')

# Configuração do pandas
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 100)
pd.options.display.float_format = '{:,.2f}'.format

# Ajustar o separador decimal e de milhar
locale.setlocale(locale.LC_ALL, 'pt_BR.UTF-8')

# Configuração do Matplotlib
plt.rcParams['figure.figsize'] = [15, 6]
plt.style.use('seaborn-v0_8-darkgrid')

```

## Coleta dos dados

```

# Dataset: Principal (Censo e Indicadores Educacionais)
sql_principal = """
with siconfi as(
select
    ano,
    sigla_uf,
    id_municipio,
    sum(valor) as total_siconfi
from
    `basedosdados.br_me_siconfi.municipio_despesas_funcao`
where
    ano in (2016,2017,2018,2019,2020) and sigla_uf in ('BA', 'SE', 'AL',
'PE', 'PB', 'RN', 'CE', 'PI', 'MA') and conta = 'Educação' and estagio =
'Despesas Pagas'
group by 1, 2, 3
order by 4 desc),

```

```

censo_escolar as (
select
  ano,
  sigla_uf,
  id_municipio,
  sum(quantidade_matricula_masculino + quantidade_matricula_feminino +
quantidade_matricula_nao_declarada) as total_matricula
from
  `basedosdados.br_inep_censo_escolar.escola`
where
  ano in (2016,2017,2018,2019,2020) and sigla_uf in ('BA', 'SE', 'AL',
'PE', 'PB', 'RN', 'CE', 'PI', 'MA')
group by 1, 2, 3
),
investimento_educacao as (
select
  siconfi.ano + 1 as ano,
  siconfi.sigla_uf as sigla_uf,
  siconfi.id_municipio as id_municipio,
  siconfi.total_siconfi,
  round((total_siconfi/total_matricula), 2) as investimento_per_capita
from
  siconfi
  inner join censo_escolar on siconfi.ano = censo_escolar.ano and sicon-
fi.id_municipio = censo_escolar.id_municipio
)

SELECT
  escola.ano,
  escola.sigla_uf,
  escola.id_municipio,
  escola.id_escola,
  escola.rede,
  escola.tipo_localizacao,
  escola.quantidade_docente_educacao_basica,
  escola.laboratorio_ciencias,
  escola.laboratorio_informatica,
  escola.quadra_esportes,
  escola.biblioteca_sala_leitura,
  (escola.quantidade_matricula_masculino + esco-
la.quantidade_matricula_feminino + esco-
la.quantidade_matricula_nao_declarada) as total_matricula,
  indic_escola.atu_ef,
  indic_escola.had_ef,
  indic_escola.dsu_ef,
  indic_escola.taxa_aprovacao_ef,
  indic_escola.taxa_reprovacao_ef,
  indic_escola.taxa_abandono_ef,
  investimento_educacao.total_siconfi as investimento_educacao,
  investimento_educacao.investimento_per_capita
FROM
  `basedosdados.br_inep_censo_escolar.escola` escola
  inner join `basedosdados.br_inep_indicadores_educacionais.escola` in-
dic_escola on

```

```

        indic_escola.ano = escola.ano and
        indic_escola.id_municipio = escola.id_municipio and
        indic_escola.id_escola = escola.id_escola
    inner join investimento_educacao on
        investimento_educacao.ano = escola.ano and
        investimento_educacao.sigla_uf = escola.sigla_uf and
        investimento_educacao.id_municipio = escola.id_municipio
WHERE
    escola.ano in (2017,2018,2019,2020,2021) and
    escola.sigla_uf in ('BA', 'SE', 'AL', 'PE', 'PB', 'RN', 'CE', 'PI',
'MA') and
    indic_escola.taxa_aprovacao_ef is not null
"""

# Para carregar o dado direto no pandas
df_censo = bd.read_sql(query=sql_principal,
                        billing_project_id=billing_project_id)

# Caminho completo para salvar o arquivo
file_path = os.path.join(bronze_layer_path, 'bronze_censo.csv')
df_censo.to_csv(file_path, index=False)

# Dataset Municipio
sql_municipio = """
SELECT
    ibge.ano,
    municipio.sigla_uf,
    municipio.id_municipio,
    municipio.nome,
    municipio.nome_regiao,
    ibge.populacao,
    round((pib.pib/ibge.populacao), 2) as pib_per_capita,
FROM
    `basedosdados.br_bd_diretorios_brasil.municipio` municipio
    left join `basedosdados.br_ibge_populacao.municipio` ibge on
        ibge.id_municipio = municipio.id_municipio
    left join `basedosdados.br_ibge_pib.municipio` pib on
        pib.ano = ibge.ano and
        pib.id_municipio = ibge.id_municipio
where
    ibge.ano in (2017,2018,2019,2020,2021) and
    municipio.sigla_uf in ('BA', 'SE', 'AL', 'PE', 'PB', 'RN', 'CE', 'PI',
'MA')
"""

# Para carregar o dado direto no pandas
df_municipio = bd.read_sql(query=sql_municipio,
                            billing_project_id=billing_project_id)

# Caminho completo para salvar o arquivo
file_path = os.path.join(bronze_layer_path, 'bronze_municipio.csv')
df_municipio.to_csv(file_path, index=False)

# Dataset: DICIONARIO
sql_dicionario = """

```

```

SELECT *
FROM `basedosdados.br_inep_censo_escolar.dicionario`
WHERE id_tabela in ('escola')
"""

# Para carregar o dado direto no pandas
df_dicionario = bd.read_sql(query=sql_dicionario,
                             billing_project_id=billing_project_id)

# Caminho completo para salvar o arquivo
file_path = os.path.join(bronze_layer_path, 'bronze_dicionario.csv')
df_dicionario.to_csv(file_path, index=False)

```

## Análise Exploratória

- Carregar dados da camada bronze
- Limpeza e Tratamento dos dados
- Salvar na camada Silver

```

# Carregar dados da arquitetura Medallion
# Carregar dados da camada bronze no dataframe
censo_csv = os.path.join(bronze_layer_path, 'bronze_censo.csv')
municipio_csv = os.path.join(bronze_layer_path, 'bronze_municipio.csv')
dicionario_csv = os.path.join(bronze_layer_path, 'bronze_dicionario.csv')

bronze_censo = pd.read_csv(censo_csv, parse_dates=True)
bronze_municipio = pd.read_csv(municipio_csv, parse_dates=True)
bronze_dicionario = pd.read_csv(dicionario_csv, parse_dates=True)

# Limpeza e tratamento dos dados: remove colunas irrelevantes e aplica es-
# tratégia para valores ausentes
def limpeza_tratamento(df, colunas):
    # Remover colunas irrelevantes
    df = df.drop(columns=[col for col in df.columns if col not in colu-
nas])

    # Tratar valores nulos
    for column in df.columns:
        if df[column].isnull().any():
            if df[column].dtype == 'object':
                imputer = SimpleImputer(strategy='most_frequent')
            else:
                imputer = SimpleImputer(strategy='median')
            df[column] = imputer.fit_transform(df[[column]])

    return df

# Função para mapear valores do dicionário no dataframe
def mapear_valores(df, dicionario, id_tabela):
    for coluna in df.columns:
        # Filtrar o dicionário para o id_tabela e a coluna de interesse
        filtro = (dicionario['id_tabela'] == id_tabela) & (diciona-
rio['nome_coluna'] == coluna)
        dicionario_filtrado = dicionario[filtro]

```



```
# Mapear os valores que precisam de tradução de acordo com o dataframe
df_dicionario
```

```
mapear_valores(silver_censo, bronze_dicionario, id_tabela='escola')
```

```
# Definir colunas relevantes do dataset
```

```
colunas_censo = ['ano',
                  'sigla_uf',
                  'id_municipio',
                  'id_escola',
                  'rede',
                  'tipo_localizacao',
                  'quantidade_docente_educacao_basica',
                  'laboratorio_ciencias',
                  'laboratorio_informatica',
                  'biblioteca_sala_leitura',
                  'quadra_esportes',
                  'total_matricula',
                  'atu_ef',
                  'had_ef',
                  'dsu_ef',
                  'taxa_aprovacao_ef',
                  'taxa_reprovacao_ef',
                  'taxa_abandono_ef',
                  'investimento_educacao',
                  'investimento_per_capita']
```

```
# Remoer colunas e tratar dados ausentes
```

```
silver_censo = limpeza_tratamento(silver_censo, colunas_censo)
```

```
silver_censo.info()
```

```
# Verificando os dados após a transformação
```

```
silver_censo.head()
```

### **Análise Exploratória -> Municípios**

```
bronze_municipio.shape
```

```
bronze_municipio.head()
```

```
# Tipos de colunas
```

```
bronze_municipio.dtypes.value_counts()
```

```
# campos vazios
```

```
bronze_municipio.isnull().sum().sort_values( ascending=False )
```

```
bronze_municipio.info()
```

```
# Verificação de Duplicados
```

```
num_duplicados = bronze_municipio.duplicated().sum()
```

```
print(f"Total de registros duplicados: {num_duplicados}")
```

```
# Remove coluna desnecessárias e altera nome de colunas para melhor identificação
```

```
silver_municipio = bronze_municipio.drop(columns=['sigla_uf'])
```

```
silver_municipio = silver_municipio.rename(columns={'nome': 'municipio'})
```



```
silver_municipio = silver_municipio.rename(columns={'nome_regiao': 'regiao'})
```

```
# Verificando os dados após a transformação
```

```
silver_municipio.head()
```

```
# Salvar nos datasets da camada Silver apenas os datasets relevantes
```

```
silver_censo.to_csv(os.path.join(silver_layer_path, 'silver_censo.csv'),  
index=False)
```

```
silver_municipio.to_csv(os.path.join(silver_layer_path, 'silver_municipio.csv'), index=False)
```

## Integração dos dados para análise

```
# Carregando os dados da camada silver na camada gold
```

```
gold_censo = pd.read_csv(os.path.join(silver_layer_path, 'silver_censo.csv'))
```

```
gold_municipio = pd.read_csv(os.path.join(silver_layer_path, 'silver_municipio.csv'))
```

```
# Mesclar os datasets na camada Gold e salvar incrementalmente
```

```
# Mesclar censo e municipio
```

```
gold_df = pd.merge(gold_censo, gold_municipio, on=['ano', 'id_municipio'],  
how='inner')
```

```
gold_df.to_csv(os.path.join(gold_layer_path, 'gold_final.csv'), index=False)
```

```
gold_df.shape
```

```
gold_df.head()
```

```
gold_df.isnull().sum().sort_values(ascending=False)
```

```
gold_df.info()
```

```
# Verificação de Duplicados
```

```
num_duplicados = gold_df.duplicated().sum()
```

```
print(f"Total de registros duplicados: {num_duplicados}")
```

## Exploração Analítica (EDA)

```
# Filtrar os tipos de colunas
```

```
Colunas_Categoricas = gold_df.columns[ gold_df.dtypes == object ]
```

```
Colunas_Numericas = gold_df.columns[ gold_df.dtypes != object ]
```

```
Colunas_Categoricas, Colunas_Numericas
```

```
(Index(['sigla_uf', 'rede', 'tipo_localizacao', 'laboratorio_ciencias',  
       'laboratorio_informatica', 'quadra_esportes', 'biblioteca_sala_leitura',  
       'municipio', 'regiao'],  
      dtype='object'),
```

```
Index(['ano', 'id_municipio', 'id_escola',  
      'quantidade_docente_educacao_basica', 'total_matricula', 'atu_ef',  
      'had_ef', 'dsu_ef', 'taxa_aprovacao_ef', 'taxa_reprovacao_ef',  
      'taxa_abandono_ef', 'investimento_educacao', 'investimen-
```

```

to_per_capita',
    'populacao', 'pib_per_capita'],
    dtype='object'))

# Analise dos campos objetos
gold_df['municipio'].value_counts( normalize=True ) * 100

# Loop
for Coluna in Colunas_Categoricas:

    # Fazendo a analise
    Analise = gold_df[Coluna].value_counts( normalize=True ) * 100

    # Mostrando
    print( Coluna, '\n', Analise, '\n')

# Gráfico de Barras da Infraestrutura Escolar:
infra_cols = ['laboratorio_ciencias', 'laboratorio_informatica', 'qua-
dra_esportes', 'biblioteca_sala_leitura']

# Configura a figura com 2 Linhas e 2 colunas
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.flatten() # Converte para uma lista para facilitar o loop

# Loop através de cada variável de infraestrutura e plota o gráfico de
barras
for i, col in enumerate(infra_cols):
    grouped_data = gold_df.groupby(col)['taxa_aprovacao_ef'].mean()

    bars = grouped_data.plot(kind='bar', ax=axes[i], color=['lightblue',
'salmon'], edgecolor='black')
    axes[i].set_xlabel(f'Presença de {col.replace("_", "
").capitalize()}' )
    axes[i].set_ylabel('Média da Taxa de Aprovação (%)')
    axes[i].set_title(f'Taxa de Aprovação em Relação à Presença de
{col.replace("_", " ").capitalize()}' )
    axes[i].set_xticks([0, 1])
    axes[i].set_xticklabels(['Não', 'Sim'], rotation=0)
    axes[i].grid(axis='y', linestyle='--')

    # Adicionar o valor em cima de cada barra
    for bar in bars.patches:
        axes[i].annotate(format(bar.get_height(), '.2f'),
                        (bar.get_x() + bar.get_width() / 2,
bar.get_height()),
                        ha='center', va='bottom', fontsize=11, co-
lor='black')

# Ajusta o layout para evitar sobreposição
plt.tight_layout()
plt.show()

```

```

# Histograma de Investimento per capita
# Agrupar por município e calcular a média do investimento per capita
df_histograma =
gold_df.groupby('id_municipio')['investimento_per_capita'].mean().reset_in
dex()

# Plotar o histograma com o valor médio por município
plt.figure(figsize=(10, 6))
plt.hist(df_histograma['investimento_per_capita'], bins=20, co-
lor='skyblue', edgecolor='black', alpha=0.7)
plt.xlabel('Investimento Per Capita (R$)')
plt.ylabel('Frequência')
plt.title('Distribuição do Investimento Per Capita por Município')
plt.show()

# Grid - Gráficos

# Converter Colunas Numericas para uma lista, para remover campos que não
serão avaliados
lista_numerica = list(Colunas_Numericas)

# Valores que você quer remover
valores_para_remover = ['ano', 'id_municipio', 'id_escola']

# Converter de volta para um Index, se necessário
Colunas_Numericas_Filtradas = pd.Index([col for col in lista_numerica if
col not in valores_para_remover])

# Tamanho
Figura, Eixo = plt.subplots( figsize=(20, 30) )

# Cor de fundo
Cor_Fundo = '#f5f5f5'
Figura.set_facecolor( Cor_Fundo )

# Paleta de Cores
Paleta_Cores = sns.color_palette( 'flare',
len(Colunas_Numericas_Filtradas) * 2 )

# Titulo
plt.suptitle('Análise das Variáveis Numericas', fontsize=22, co-
lor='#404040', fontweight=600 )

# Estrutura
Linhas = len( Colunas_Numericas_Filtradas ) # (Todas as infos numericas)
Colunas = 2 #( Boxplot - Distplot)
Posicao = 1 # Posicao inicial do grid

# Loop para plotar os gráficos
for Coluna in Colunas_Numericas_Filtradas:

    # Plot no Grid -- Boxplot

```

```

plt.subplot( Linhas, Colunas, Posicao )

# Titulo
plt.title( f'{Coluna}', loc='left', fontsize=14, fontweight=200 )

# Plot
sns.boxplot( data=gold_df, y=Coluna, showmeans=True, saturation=0.75,
             linewidth=1, color=Paleta_Cores[Posicao], width=0.25 )

# Mudar
Posicao += 1

# Plot no Grid -- Distplot
plt.subplot( Linhas, Colunas, Posicao )

# Titulo
plt.title( f'{Coluna}', loc='left', fontsize=14, fontweight=200 )

# Plot
sns.distplot( gold_df[Coluna], color=Paleta_Cores[ Posicao - 1 ] )

# Mudar
Posicao += 1

# Ajute de Grid
plt.subplots_adjust( top=0.95, hspace=0.5 )

# Analisando as variáveis
gold_df['investimento_per_capita'].describe()
gold_df['populacao'].describe()
gold_df['investimento_educacao'].describe()
gold_df['pib_per_capita'].describe()
gold_df['quantidade_docente_educacao_basica'].describe()
gold_df['total_matricula'].describe()
gold_df['atu_ef'].describe()
gold_df['had_ef'].describe()
gold_df['dsu_ef'].describe()
gold_df['taxa_abandono_ef'].describe()
gold_df['taxa_aprovacao_ef'].describe()

# Aplicação de Escalonamento Logarítmico para ajuste de outliers populaci-
# onais e/ou quantitativos
# que podem ter alta variação em alguns municípios
gold_df['investimento_per_capita_log'] =
np.log1p(gold_df['investimento_per_capita']) # log(x + 1) para lidar com
valores zero

```

```

gold_df['populacao_log'] = np.log1p(gold_df['populacao']) #  $\log(x + 1)$ 
para lidar com valores zero
gold_df['investimento_educacao_log'] =
np.log1p(gold_df['investimento_educacao'])
gold_df['pib_per_capita_log'] = np.log1p(gold_df['pib_per_capita'])
gold_df['quantidade_docente_educacao_basica_log'] =
np.log1p(gold_df['quantidade_docente_educacao_basica'])
gold_df['total_matricula_log'] = np.log1p(gold_df['total_matricula'])

# Aplicando winsorização nas variáveis atu_ef e had_ef
gold_df['atu_ef_winsor'] = mstats.winsorize(gold_df['atu_ef'], li-
mits=[0.01, 0.01])
gold_df['had_ef_winsor'] = mstats.winsorize(gold_df['had_ef'], li-
mits=[0.01, 0.01])
gold_df['dsu_ef_winsor'] = mstats.winsorize(gold_df['dsu_ef'], li-
mits=[0.01, 0.01])
# Winsorização da taxa de abandono
gold_df['taxa_abandono_ef_winsor'] =
mstats.winsorize(gold_df['taxa_abandono_ef'], limits=[0, 0.01])

```

## Engenharia de Features

```

# Ajuste das colunas Categóricas
# Converter variáveis categóricas para numéricas para a análise de corre-
lação
label_encoder = LabelEncoder()
for col in ['sigla_uf', 'rede', 'tipo_localizacao', 'laborato-
rio_ciencias',
            'laboratorio_informatica', 'quadra_esportes', 'bibliote-
ca_sala_leitura',
            'regiao']:
    gold_df[col] = label_encoder.fit_transform(gold_df[col])

# remove a coluna nome do municipio por não influenciar no modelo
gold_df.drop( columns=['municipio'], inplace=True)

gold_df

# define as colunas que irão compor as features
features_selecionadas = ['rede', 'tipo_localizacao', 'laborato-
rio_ciencias', 'laboratorio_informatica',
                        'quadra_esportes', 'biblioteca_sala_leitura',
                        'total_matricula_log', 'investimento_per_capita_log',
                        'pib_per_capita_log', 'investimen-
to_educacao_log', 'quantidade_docente_educacao_basica_log', 'popula-
cao_log',
                        'taxa_abandono_ef_winsor', 'atu_ef_winsor',
                        'had_ef_winsor', 'dsu_ef_winsor',
                        'regiao']

# Separando as features (X) e a variável-alvo (y) para previsão de taxa de
aprovação
Caracteristicas = gold_df[features_selecionadas]
Previsor = gold_df['taxa_aprovacao_ef']

```

```

# Verificar
Caracteristicas.shape, Previsor.shape

# Dados das features
Caracteristicas.head()

# Dados do Previsor (variável-alvo)
Previsor.head()

# Definir o número de linhas e colunas para os gráficos
num_features = len(features_selecionadas)
num_cols = 2
num_rows = math.ceil(num_features / num_cols)

# Criar a grade de subplots com o número adequado de linhas e colunas
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 5))
axes = axes.flatten()

# Loop para gerar os gráficos
for i, feature in enumerate(features_selecionadas):
    sns.histplot(gold_df[feature], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribuição da Feature: {feature}')

# Remover quaisquer eixos não utilizados, se houver
for i in range(len(features_selecionadas), len(axes)):
    fig.delaxes(axes[i])

# Ajustar o layout para evitar sobreposição de elementos
plt.tight_layout()
plt.show()

# Armazenar resultados
correlation_results = {}

# Teste de normalidade para o previsor
normal_test_previsor = normaltest(Previsor)
previsor_is_normal = normal_test_previsor.pvalue > 0.05

# Iterar sobre cada feature
for feature in Caracteristicas:
    normal_test_feature = normaltest(gold_df[feature])
    feature_is_normal = normal_test_feature.pvalue > 0.05

    # Escolher o tipo de correlação
    if previsor_is_normal and feature_is_normal:
        # Usar Pearson se ambos forem normais
        correlation, p_value = pearsonr(gold_df[feature], Previsor)
        method = "Pearson"
    else:
        # Usar Spearman caso contrário
        correlation, p_value = spearmanr(gold_df[feature], Previsor)
        method = "Spearman"

    # Armazenar o resultado
    correlation_results[feature] = {'correlation': correlation, 'p_value':

```

```

p_value, 'method': method}

# Converter os resultados para um DataFrame para visualização
correlation_df = pd.DataFrame(correlation_results).T
correlation_df

# Yellowbrick
# Definir o metodo
Correlacao = Rank2D( algoritmo='spearman' )

# Fitar função
Correlacao.fit( Caracteristicas, Previsor )
Correlacao.transform( Caracteristicas )
Correlacao.show();

# Dividindo os dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(Caracteristicas, Previsor, test_size=0.2, random_state=42)

print(f'Dados de Treino: { X_train.shape[0] }')
print(f'Dados de Teste: { X_test.shape[0] }')

# Features mais relevantes
# Selecao de features
def Selecao_Features( X_train, y_train ):

    # Configurar para selecionar as features
    Selecao = SelectKBest( score_func=mutual_info_regression, k=5 )

    # Fitar o aprendizado
    Selecao.fit( X_train, y_train )

    return Selecao

# Aplicar essa função
Scores = Selecao_Features( X_train, y_train )

# Analisar
for Posicao, Score in enumerate( Scores.scores_ ):
    print( f' { X_train.columns[Posicao] } : {Score}' )

```

## Construção do Modelo

### Regressão Linear

```

resultados = {}

# Prever a taxa de aprovação (Regressão Linear)
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)

LinearRegression()

# Prevendo no conjunto de teste
y_pred_lr = model_lr.predict(X_test)

```

#### # Avaliando o modelo

```
rmse = sqrt(mean_squared_error( y_test, y_pred_lr ) )
mse = mean_squared_error(y_test, y_pred_lr)
mae = mean_absolute_error(y_test, y_pred_lr)
r2 = r2_score( y_test, y_pred_lr )
```

```
print(f'RMSE Regressão Linear: {rmse:.2f} ')
print(f'MSE Regressão Linear: {mse:.2f} ')
print(f'MAE Regressão Linear: {mae:.2f} ')
print(f'Score Regressão Linear: {r2:.2f} ')
```

#### # Aplicando Cross-Validation

```
cv_scores_lr = cross_val_score(model_lr, Caracteristicas, Previsor, cv=5,
scoring='neg_mean_squared_error')
print("CV MSE Regressão Linear: %0.2f (+/- %0.2f)" % (cv_scores_lr.mean(),
cv_scores_lr.std() * 2))
```

```
resultados['Regressão Linear'] = {'RMSE': rmse, 'MSE': mse, 'MAE': mae,
'R2': r2, 'CVMSE': cv_scores_lr.mean() }
```

#### # Avaliando Yellowbrick

##### # Instanciar

```
Modelo = LinearRegression()
Erro_Modelo = PredictionError( Modelo )
```

##### # Fitar

```
Erro_Modelo.fit( X_train, y_train )
Erro_Modelo.score( X_test, y_test )
Erro_Modelo.show();
```

## Árvore de Decisão

#### # Explicação das variáveis (Árvore de Decisão)

```
model_dt = DecisionTreeRegressor(max_depth=5, random_state=42)
model_dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=5, random_state=42)
```

#### # Prevendo no conjunto de teste

```
y_pred_dt = model_dt.predict(X_test)
```

#### # Avaliando o modelo

```
rmse = sqrt(mean_squared_error( y_test, y_pred_dt ) )
mse = mean_squared_error(y_test, y_pred_dt)
mae = mean_absolute_error(y_test, y_pred_dt)
r2 = r2_score( y_test, y_pred_dt )
```

```
print(f'RMSE Árvore de Decisão: {rmse:.2f} ')
print(f'MSE Árvore de Decisão: {mse:.2f} ')
print(f'MAE Árvore de Decisão: {mae:.2f} ')
print(f'Score Árvore de Decisão: {r2:.2f} ')
```

#### # Aplicando Cross-Validation

```
cv_scores_dt = cross_val_score(model_dt, Caracteristicas, Previsor, cv=5,
scoring='neg_mean_squared_error')
```



```

print("CV MSE Árvore de Decisão: %0.2f (+/- %0.2f)" %
      (cv_scores_dt.mean(), cv_scores_dt.std() * 2))

resultados['Árvore de Decisão'] = {'RMSE': rmse, 'MSE': mse, 'MAE': mae,
                                     'R2': r2, 'CVMSE': cv_scores_dt.mean()}

# Avaliando Yellowbrick
# Instanciar
Modelo = DecisionTreeRegressor(max_depth=5, random_state=42)
Erro_Modelo = PredictionError( Modelo )

# Fitar
Erro_Modelo.fit( X_train, y_train )
Erro_Modelo.score( X_test, y_test )
Erro_Modelo.show();

# Visualizando a importância das variáveis
importances_dt = model_dt.feature_importances_
features = Caracteristicas.columns
indices = np.argsort(importances_dt)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Importância das Variáveis - Árvore de Decisão")
plt.bar(range(Caracteristicas.shape[1]), importances_dt[indices],
        align="center")
plt.xticks(range(Caracteristicas.shape[1]), features[indices], rotation=90)
plt.show()

```

## Random Forest Regressor

```

# Instanciar
model_forest = RandomForestRegressor( max_depth=5 )

# Fitar
model_forest.fit( X_train, y_train )

RandomForestRegressor(max_depth=5)

# Avaliar a performance
y_pred_rfr = model_forest.predict( X_test )

rmse = sqrt(mean_squared_error( y_test, y_pred_rfr ) )
mse = mean_squared_error(y_test, y_pred_rfr)
mae = mean_absolute_error(y_test, y_pred_rfr)
r2 = r2_score( y_test, y_pred_rfr )

print(f'RMSE Random Forest: {rmse:.2f} ')
print(f'MSE Random Forest: {mse:.2f} ')
print(f'MAE Random Forest: {mae:.2f} ')
print(f'Score Random Forest: {r2:.2f} ')

```

### # Aplicando Cross-Validation

```
cv_scores_rfr = cross_val_score(model_forest, Caracteristicas, Previsor,
cv=5, scoring='neg_mean_squared_error')
print("CV MSE Random Forest : %0.2f (+/- %0.2f)" % (cv_scores_rfr.mean(),
cv_scores_rfr.std() * 2))
```

```
resultados['Random Forest'] = {'RMSE': rmse, 'MSE': mse, 'MAE': mae, 'R2':
r2, 'CVMSE': cv_scores_rfr.mean() }
```

### # Avaliando Yellowbrick

#### # Instanciar

```
Modelo = RandomForestRegressor( max_depth=5 )
Erro_Modelo = PredictionError( Modelo )
```

#### # Fitar

```
Erro_Modelo.fit( X_train, y_train )
Erro_Modelo.score( X_test, y_test )
Erro_Modelo.show();
```

### # Visualizando a importância das variáveis

```
importances_rfr = model_forest.feature_importances_
features = Caracteristicas.columns
indices = np.argsort(importances_rfr)[::-1]
```

```
plt.figure(figsize=(10, 6))
plt.title("Importância das Variáveis - Random Forest")
plt.bar(range(Caracteristicas.shape[1]), importances_rfr[indices],
align="center")
plt.xticks(range(Caracteristicas.shape[1]), features[indices], rota-
tion=90)
plt.show()
```

## Gradient Boosting

#### # Instanciar

```
model_gbr = GradientBoostingRegressor( max_depth=5, random_state=42 )
```

#### # Fitar

```
model_gbr.fit( X_train, y_train )
```

```
GradientBoostingRegressor(max_depth=5, random_state=42)
```

#### # Prevendo no conjunto de teste

```
y_pred_gbr = model_gbr.predict(X_test)
```

#### # Avaliando o modelo

```
rmse = sqrt(mean_squared_error( y_test, y_pred_gbr ) )
mse = mean_squared_error(y_test, y_pred_gbr)
mae = mean_absolute_error(y_test, y_pred_gbr)
r2 = r2_score( y_test, y_pred_gbr )
```

```
print(f'RMSE Gradient Boosting: {rmse:.2f} ')
```

```

print(f'MSE Gradient Boosting: {mse:.2f} ')
print(f'MAE Gradient Boosting: {mae:.2f} ')
print(f'Score Gradient Boosting: {r2:.2f} ')

# Aplicando Cross-Validation
cv_scores_gbr = cross_val_score(model_gbr, Caracteristicas, Previsor,
cv=5, scoring='neg_mean_squared_error')
print("CV MSE Gradient Boosting: %0.2f (+/- %0.2f)" %
(cv_scores_gbr.mean(), cv_scores_gbr.std() * 2))

resultados['Gradient Boosting'] = {'RMSE': rmse, 'MSE': mse, 'MAE': mae,
'R2': r2, 'CVMSE': cv_scores_gbr.mean()}

# Avaliando Yellowbrick
# Instanciar
Modelo = GradientBoostingRegressor( max_depth=5, random_state=42 )
Erro_Modelo = PredictionError( Modelo )

# Fitar
Erro_Modelo.fit( X_train, y_train )
Erro_Modelo.score( X_test, y_test )
Erro_Modelo.show();

# Visualizando a importância das variáveis
importances_gbr = model_gbr.feature_importances_
features = Caracteristicas.columns
indices = np.argsort(importances_gbr)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Importância das Variáveis - Gradient Boosting")
plt.bar(range(Caracteristicas.shape[1]), importances_gbr[indices],
align="center")
plt.xticks(range(Caracteristicas.shape[1]), features[indices], rota-
tion=90)
plt.show()

```

## XGBoost

```

# Instanciar
model_xgb = xgb.XGBRegressor(max_depth=5)

# Treinando o modelo
model_xgb.fit(X_train, y_train)

# Fazendo previsões
y_pred_xgb = model_xgb.predict(X_test)

# Avaliando o modelo
rmse = sqrt(mean_squared_error( y_test, y_pred_xgb ) )
mse = mean_squared_error(y_test, y_pred_xgb)
mae = mean_absolute_error(y_test, y_pred_xgb)
r2 = r2_score( y_test, y_pred_xgb )

print(f'RMSE XGBoost: {rmse:.2f} ')

```

```

print(f'MSE XGBoost: {mse:.2f} ')
print(f'MAE XGBoost: {mae:.2f} ')
print(f'Score XGBoost: {r2:.2f} ')

# Aplicando Cross-Validation
cv_scores_xgb = cross_val_score(model_xgb, Caracteristicas, Previsor,
cv=5, scoring='neg_mean_squared_error')
print("CV MSE XGBoost: %0.2f (+/- %0.2f)" % (cv_scores_xgb.mean(),
cv_scores_xgb.std() * 2))

resultados['XGBoost'] = {'RMSE': rmse, 'MSE': mse, 'MAE': mae, 'R2': r2,
'CVMSE': cv_scores_xgb.mean()}

# Avaliando Yellowbrick
# Instanciar
Modelo = xgb.XGBRegressor(max_depth=5)
Erro_Modelo = PredictionError( Modelo )

# Fitar
Erro_Modelo.fit( X_train, y_train )
Erro_Modelo.score( X_test, y_test )
Erro_Modelo.show();

# Visualizando a importância das variáveis
importances_xgb = model_xgb.feature_importances_
features = Caracteristicas.columns
indices = np.argsort(importances_xgb)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Importância das Variáveis - XGBoost")
plt.bar(range(Caracteristicas.shape[1]), importances_xgb[indices],
align="center")
plt.xticks(range(Caracteristicas.shape[1]), features[indices], rota-
tion=90)
plt.show()

# Comparar os resultados
tabela_resultados = pd.DataFrame(resultados).T
print(tabela_resultados)

# Visualizar comparação dos resultados
plt.figure(figsize=(10, 6))
sns.barplot(x=tabela_resultados.index, y='R2', data=tabela_resultados)
plt.title('Comparação dos Modelos - R2 Score')
plt.ylabel('R2 Score')
plt.xlabel('Modelo')
plt.show()

```