A etapa de **preparação dos dados** é essencial para garantir que o modelo de Machine Learning receba informações consistentes e comparáveis.
Antes de “ensinar” o modelo, precisamos organizar o conjunto de dados, transformando-o em um formato que facilite o aprendizado.

🧭 Visão geral do seu fluxo atual (Etapa A – Modelo Preditivo Supervisionado)

Hoje o notebook segue a trilha **supervisionada**, ou seja:
há uma **variável-alvo** (`Weekly_Sales`) e o modelo aprende a **prever um valor numérico** com base em dados históricos.

Essa parte tem as etapas:

1️⃣ **Entendimento e coleta de dados**
2️⃣ **Limpeza e preparação**
3️⃣ **Divisão treino/teste**
4️⃣ **Treinamento (Regressão Linear, RandomForest)**
5️⃣ **Avaliação de desempenho (MAE, R^2, etc.)**
6️⃣ **Visualização e interpretação dos resultados**

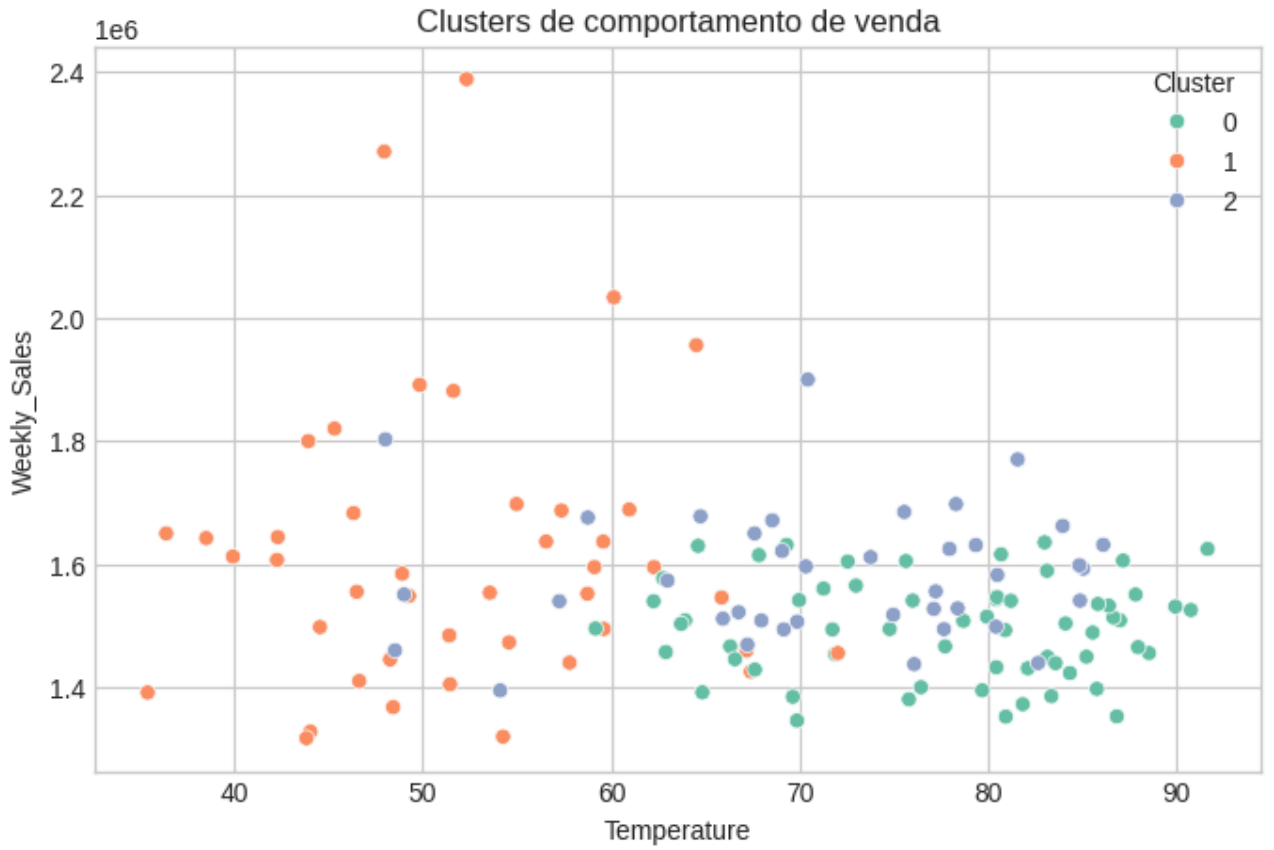
👉 Isso é **Aprendizado Supervisionado (Regressão)**.

🧩 Etapa B — Agrupamentos (Clustering K-Means ou DBSCAN)

Essa etapa é **paralela e complementar**, pertencente à categoria de **Aprendizado Não Supervisionado**, porque:

- **Não existe variável-alvo.**
- O objetivo é **descobrir padrões ou grupos naturais** dentro dos dados (ex: lojas semelhantes, períodos sazonais, comportamento de venda).

```python
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Seleciona variáveis relevantes
X = df[["Weekly_Sales", "Temperature", "Fuel_Price", "CPI", "Unemployment"]]

# Normaliza os dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Teste inicial de 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df["Cluster"] = kmeans.fit_predict(X_scaled)

# Visualização
plt.figure(figsize=(8,5))
sns.scatterplot(x="Temperature", y="Weekly_Sales", hue="Cluster", data=df, palette="Set2")
plt.title("Clusters de comportamento de venda")
plt.show()
```



Cluster	Significado_Cluster	Weekly_Sales	Temperature	Fuel_Price	CPI	Unemployment
0	**Semanas frias e com alto consumo**	1495228.66	77.59	3.07	213.75	7.82
1	**Períodos de crise econômica**	1615250.20	52.14	2.24	213.94	7.82
2	**Semanas com feriado e aumento nas vendas**	1583028.26	71.61	3.28	221.79	7.05

🟦 Interpretação:
- Cluster 0 → Semanas frias e com alto consumo.
- Cluster 1 → Períodos de crise econômica (menor poder de compra).
- Cluster 2 → Semanas com feriado e aumento nas vendas (picos sazonais).

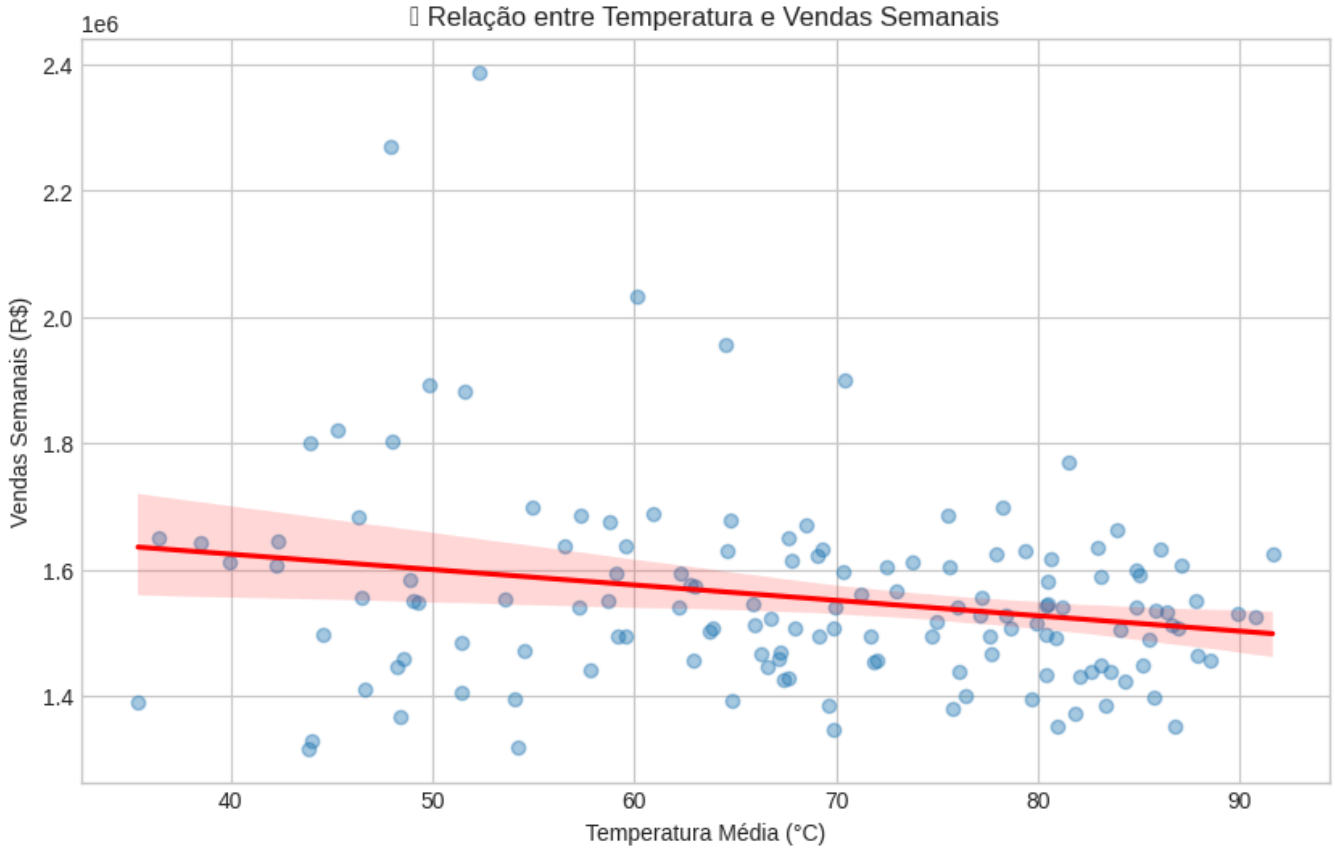
📊 O que esse bloco entrega:

Etapa	O que faz	Explicação simplificada
1️⃣ **Seleção de variáveis**	Escolhe colunas numéricas ligadas ao comportamento de vendas	Garante foco nas variáveis com impacto direto
2️⃣ **Normalização**	Padroniza escalas para evitar que valores altos distorçam o agrupamento	Todas as variáveis têm "peso igual"
3️⃣ **K-Means**	Cria 3 grupos automáticos de comportamento	Cada grupo representa um padrão de semana
4️⃣ **Visualização**	Exibe o gráfico Temperatura × Vendas com cores por cluster	Facilita o entendimento visual
5️⃣ **Legenda interpretativa**	Dá sentido de negócio a cada cluster	Transforma o dado em insight estratégico
6️⃣ **Resumo numérico**	Mostra médias de cada cluster	Facilita leitura executiva para tomada de decisão

🗺️ Onde encaixar dentro do seu projeto MLDS

Você pode inserir essa etapa **logo após o treinamento e avaliação da regressão**, como um **módulo de insights adicionais**.
Ou seja, seu notebook final teria duas grandes trilhas:

💡 — Dispersão para Clusters



📘 *Interpretação:*
Cada cor representa um **padrão de comportamento**:

- Azul → Semanas frias e alto consumo
- Laranja → Crise econômica
- Verde → Feriados e aumento de vendas

🧠 Etapa A — Modelos Supervisionados (Predição de Vendas)

> objetivo: prever *quanto* será vendido.

🔍 Etapa B — Modelos Não Supervisionados (Agrupamentos)

> objetivo: entender *como* os comportamentos de venda se agrupam ou se diferenciam.

🧩 4.1 Separação em treino e teste (ex: 80/20)

Dividimos os dados em duas partes:

- **Treino (80%)** → usada para o modelo *aprender os padrões* dos dados.
- **Teste (20%)** → usada para *avaliar o desempenho* do modelo com dados nunca vistos.

📘 *Exemplo:* se temos 10.000 registros de vendas, usamos 8.000 para o modelo aprender e 2.000 para medir o quanto ele realmente "entende" dos padrões.

🔤 4.2 Codificação de variáveis categóricas

Algumas colunas contêm **categorias** (ex: "Região = Norte, Sul, Leste, Oeste").
Como o modelo só entende números, precisamos converter essas categorias:

- **LabelEncoder:** transforma categorias em números simples (ex: Norte=0, Sul=1, Leste=2, Oeste=3).
- **OneHotEncoder:** cria colunas separadas para cada categoria (ex: Norte=1/0, Sul=1/0, etc.), evitando que o modelo interprete uma ordem inexistente.

📘 *Exemplo:* para uma coluna "Feriado", o OneHotEncoder cria duas colunas:
→ `Feriado_Sim` e `Feriado_Não`, com valores 0 ou 1.

⚖️ 4.3 Padronização / Normalização de escalas

As variáveis numéricas podem estar em **escalas diferentes** — por exemplo, "temperatura" em graus e "vendas" em milhares.
Essa diferença pode distorcer o aprendizado.

Por isso, aplicamos técnicas como:

- **StandardScaler:** ajusta os valores para terem média 0 e desvio padrão 1.
- **MinMaxScaler:** normaliza os dados para uma faixa entre 0 e 1.

📘 *Exemplo:*
Se a temperatura varia entre 10 °C e 40 °C, após normalização passa a variar entre 0 e 1 — permitindo comparação justa com outras variáveis.

🧠 4.4 Feature Engineering (criação de variáveis derivadas)

É o processo de **criar novas colunas** que ajudam o modelo a enxergar padrões ocultos.
Usamos criatividade e conhecimento de negócio para extrair mais significado dos dados.

Exemplos práticos:

- Criar a variável **“Vendas_Médias_12_Semanas”** para medir tendência recente.
- Criar **“Impacto_Feriado”** (1 se for semana com feriado, 0 caso contrário).
- Calcular **“Desvio de Temperatura”** em relação à média histórica.

📘 *Objetivo:* tornar o modelo mais inteligente, aproximando a análise dos **fatores reais** que influenciam o comportamento de vendas.

🔎 **Resultado esperado:**
Um **dataset final limpo, padronizado e enriquecido**, pronto para alimentar os algoritmos de Machine Learning e gerar previsões confiáveis.

Primeiro Desafio – Criando Modelos Preditivos (MLDS)

Notebook de Trabalho – Versão Colab

Objetivo: treinar um modelo preditivo explicável e reutilizável, com dados preparados e salvos em Parquet para uso em dashboard (Streamlit).

Roteiro:

1. Setup e imports
2. Carregar dados
3. Visões rápidas (sanidade)
4. Preparação (split, codificação, escala)
5. Baseline e métricas
6. Visualizações (Real × Previsto, erros)
7. Ajuste fino (GridSearchCV)
8. Exportar artefatos (modelo, previsões)
9. Próximos passos (Streamlit / apresentação)

Roteiro:

1) Setup e imports

```
# !pip install -q pandas pyarrow scikit-learn matplotlib joblib

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

import joblib
import os
RANDOM_STATE = 42
pd.set_option("display.max_columns", 100)
```

2) Carregar dados

```
# 🔧 TODO: ajuste o caminho e o nome da coluna alvo
from google.colab import files
uploaded = files.upload()
# 🔧 Caminho e coluna alvo corretos
DATA_PATH = "/content/sample_data/sales.csv"
TARGET_COL = "Weekly_Sales"   # <-- nome real da sua variável de previsão
# Ler o arquivo CSV enviado
df = pd.read_csv("sales.csv")
df.head()
```

3) Visões rápidas (sanidade)

```
# verificação de nulos e tipos
display(df.dtypes.to_frame("dtype"))
display(df.isna().sum().to_frame("nulos"))

# estatísticas numéricas (amigável)
display(df.describe().T)

# checagem básica do alvo
assert TARGET_COL in df.columns, f"Coluna alvo {TARGET_COL} não encontrada!"
print("Alvo selecionado:", TARGET_COL)
```

	dtype
Date	object
Weekly_Sales	float64
Holiday_Flag	int64
Temperature	float64
Fuel_Price	float64
CPI	float64
Unemployment	int64

	nulos
Date	0
Weekly_Sales	0
Holiday_Flag	0
Temperature	0
Fuel_Price	0
CPI	0
Unemployment	0

	count	mean	std	min	25%	50%	75%	max
Weekly_Sales	143.0	1.555264e+06	155980.767761	1.316899e+06	1.458105e+06	1.534850e+06	1.614892e+06	2.387950e+06
Holiday_Flag	143.0	6.993007e-02	0.255926	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
Temperature	143.0	6.830678e+01	14.250486	3.540000e+01	5.826500e+01	6.964000e+01	8.048500e+01	9.165000e+01
Fuel_Price	143.0	2.879480e+03	1068.112107	2.640000e+00	2.668500e+03	3.227000e+03	3.554500e+03	3.907000e+03
CPI	143.0	2.159969e+02	4.350890	2.103374e+02	2.115493e+02	2.154599e+02	2.205313e+02	2.234443e+02
Unemployment	143.0	7.610420e+03	383.748825	6.573000e+03	7.348000e+03	7.787000e+03	7.838000e+03	8.106000e+03

```
Alvo selecionado: Weekly_Sales
```

4) Preparação (split, codificação, escala)

```python
# separa X, y
X = df.drop(columns=[TARGET_COL])
y = df[TARGET_COL]

# separa tipos
categoricas = X.select_dtypes(include=["object", "category"]).columns.tolist()
numericas   = X.select_dtypes(include=[np.number, "bool"]).columns.tolist()

print("Colunas numéricas:", numericas[:10], "..." if len(numericas)>10 else "")
print("Colunas categóricas:", categoricas[:10], "..." if len(categoricas)>10 else "")

# split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE
)

# transformações
num_transform = Pipeline(steps=[
    ("scaler", StandardScaler())
])

cat_transform = Pipeline(steps=[
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", num_transform, numericas),
        ("cat", cat_transform, categoricas)
    ],
    remainder="drop"
)
```

```
Colunas numéricas: ['Holiday_Flag', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment']
```

5) Baseline e métricas

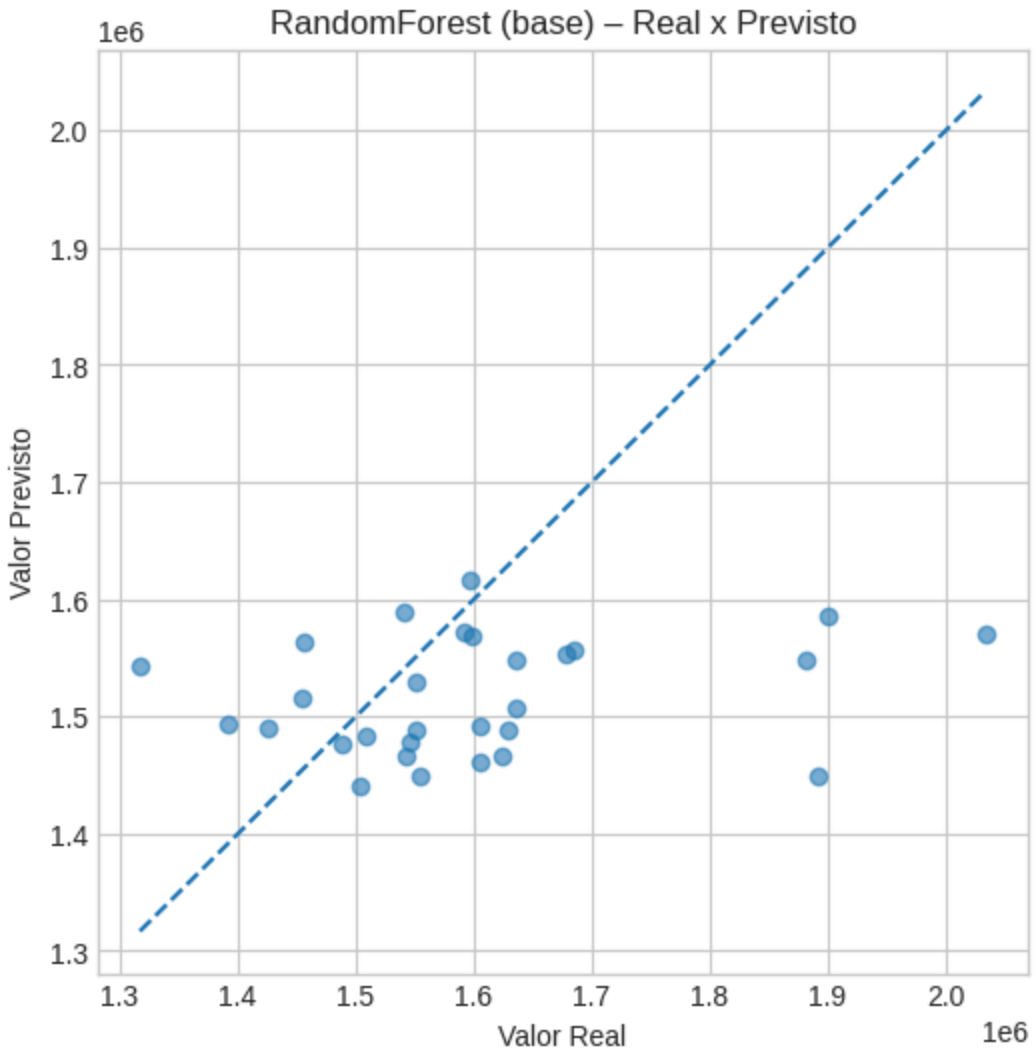
```python
def avalia_modelo(nome, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"[{nome}] MAE={mae:,.2f}  RMSE={rmse:,.2f}  R²={r2:,.3f}")
    return {"modelo": nome, "MAE": mae, "RMSE": rmse, "R2": r2}

resultados = []

# Baseline 1: Regressão Linear
pipe_lr = Pipeline(steps=[
    ("prep", preprocess),
    ("model", LinearRegression())
])
pipe_lr.fit(X_train, y_train)
pred_lr = pipe_lr.predict(X_test)
resultados.append(avalia_modelo("LinearRegression", y_test, pred_lr))

# Baseline 2: RandomForest
pipe_rf = Pipeline(steps=[
    ("prep", preprocess),
    ("model", RandomForestRegressor(
        n_estimators=300, random_state=RANDOM_STATE, n_jobs=-1))
])
pipe_rf.fit(X_train, y_train)
pred_rf = pipe_rf.predict(X_test)
resultados.append(avalia_modelo("RandomForest(base)", y_test, pred_rf))

pd.DataFrame(resultados)
```

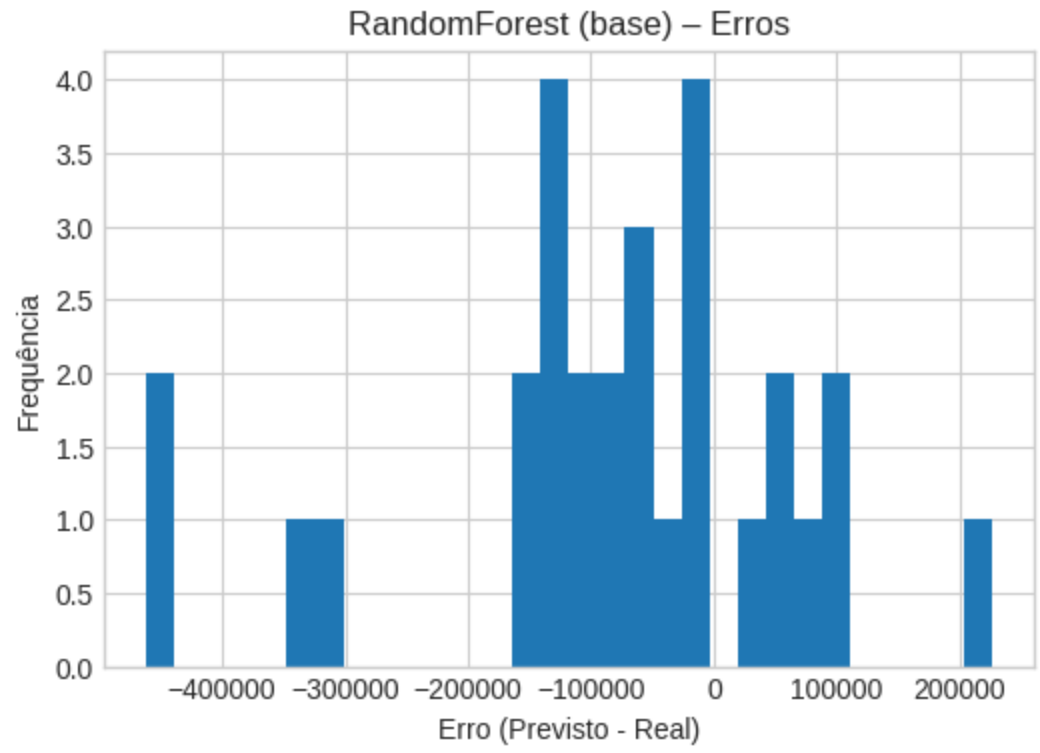
```
[LinearRegression] MAE=115,264.73  RMSE=152,479.58  R²=0.039
[RandomForest(base)] MAE=127,176.32  RMSE=173,009.24  R²=-0.238
```

	modelo	MAE	RMSE	R2
0	LinearRegression	115264.726023	152479.578782	0.038747
1	RandomForest(base)	127176.318157	173009.238084	-0.237522

6) Visualizações (Real × Previsto, erros)

```python
def plot_real_previsto(y_true, y_pred, titulo="Real x Previsto"):
    plt.figure(figsize=(6,6))
    plt.scatter(y_true, y_pred, alpha=0.6)
    # linha ideal
    minv = min(y_true.min(), y_pred.min())
    maxv = max(y_true.max(), y_pred.max())
    plt.plot([minv, maxv], [minv, maxv], linestyle="--")
    plt.xlabel("Valor Real")
    plt.ylabel("Valor Previsto")
    plt.title(titulo)
    plt.show()

def plot_residuos(y_true, y_pred, titulo="Distribuição de Erros"):
    erros = y_pred - y_true
    plt.figure(figsize=(6,4))
    plt.hist(erros, bins=30)
    plt.xlabel("Erro (Previsto - Real)")
    plt.ylabel("Frequência")
    plt.title(titulo)
    plt.show()

# use o melhor baseline até aqui (troque para pred_lr se quiser comparar)
plot_real_previsto(y_test, pred_rf, "RandomForest (base) – Real x Previsto")
plot_residuos(y_test, pred_rf, "RandomForest (base) – Erros")
```

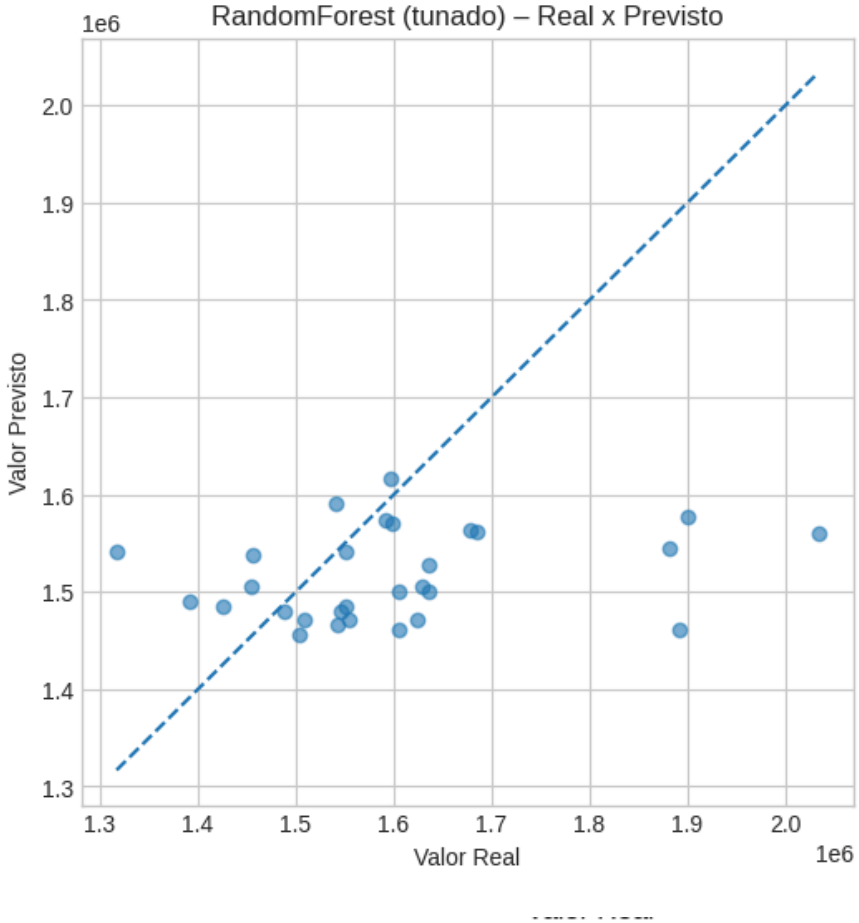

RandomForest (base) – Real x Previsto
1e6
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
Valor Previsto
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2.0
1e6
Valor Real

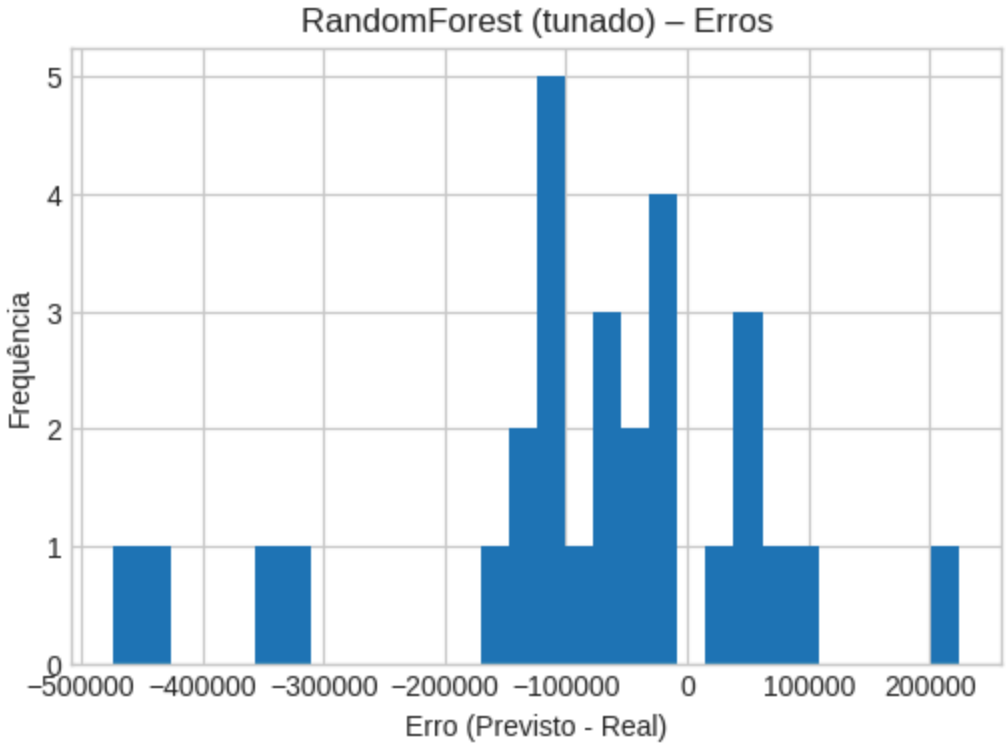

RandomForest (base) – Erros
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequência
−400000
−300000
−200000
−100000
0
100000
200000
Erro (Previsto - Real)

7) Ajuste fino (GridSearchCV)

```
param_grid = {
    "model__n_estimators": [200, 400, 700],
    "model__max_depth": [None, 8, 16],
    "model__min_samples_split": [2, 5, 10]
}

pipe_rf_tune = Pipeline(steps=[
    ("prep", preprocess),
    ("model", RandomForestRegressor(random_state=RANDOM_STATE, n_jobs=-1))
])

grid = GridSearchCV(
    estimator=pipe_rf_tune,
    param_grid=param_grid,
    scoring="neg_mean_absolute_error",  # MAE
    cv=5,
    n_jobs=-1,
    verbose=0
)
grid.fit(X_train, y_train)

print("Melhores parâmetros:", grid.best_params_)
best_model = grid.best_estimator_

pred_best = best_model.predict(X_test)
_ = avalia_modelo("RandomForest(tunado)", y_test, pred_best)

plot_real_previsto(y_test, pred_best, "RandomForest (tunado) - Real x Previsto")
plot_residuos(y_test, pred_best, "RandomForest (tunado) - Erros")
```

```
Melhores parâmetros: {'model__max_depth': 8, 'model__min_samples_split': 2, 'model__n_estimators': 700}
[RandomForest(tunado)] MAE=123,820.80  RMSE=171,867.90  R²=-0.221
```

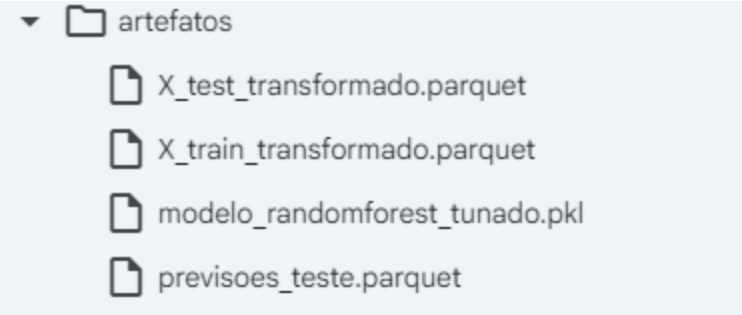




8) Exportar artefatos (modelo, previsões)

```python
os.makedirs("artefatos", exist_ok=True)

# salva o modelo tunado
joblib.dump(best_model, "artefatos/modelo_randomforest_tunado.pkl")

# salva previsões de teste
out = pd.DataFrame({
    "y_real": y_test.values,
    "y_previsto": pred_best
})
out.to_parquet("artefatos/previsoes_teste.parquet", index=False)

# opcional: salva X_train/X_test prontos (transformados) - útil para auditoria
# Atenção: isso materializa os dados transformados (após one-hot/escala)
Xt_train = best_model.named_steps["prep"].fit_transform(X_train)
Xt_test  = best_model.named_steps["prep"].transform(X_test)

pd.DataFrame(Xt_train).to_parquet("artefatos/X_train_transformado.parquet", index=False)
pd.DataFrame(Xt_test).to_parquet("artefatos/X_test_transformado.parquet", index=False)

print("✅ Artefatos salvos na pasta /artefatos")
```

```
✅ Artefatos salvos na pasta /artefatos
```



Próximos passos

1. **Explicabilidade**
   - Avaliar importância de variáveis do RandomForest para comunicar drivers de negócio.
2. **Integração com Dashboard (Streamlit)**
   - Carregar `modelo_randomforest_tunado.pkl` e `previsoes_teste.parquet`.
   - Exibir métricas (MAE, RMSE, R^2), gráfico Real × Previsto e ranking de features.
3. **Ciclo de melhoria contínua**
   - Experimentar novas features (janelas móveis, feriados regionais, temperatura defasada).
   - Avaliar modelos adicionais (Gradient Boosting, LightGBM/XGBoost).
4. **Governança**

- Versionar no Git, registrar data/versão do dataset, parâmetros e métricas.