ChatGPT

Guia de Estudo – Data Science (ONE G8)

Fundamentos do Python

Abordado nos notebooks: *Projeto_Final-Python_Data_Science.ipynb*, *Python para Data Science – funções, estruturas de dados e exceções.ipynb*

- **Variáveis e Tipos:** Python utiliza atribuição direta para criar variáveis. Cada variável tem um tipo (inteiro, float, string, bool etc.) de acordo com o valor atribuído [1]. Por exemplo: `idade = 15` cria uma variável inteira. Funções built-in como `type()` ajudam a identificar o tipo. Strings suportam métodos úteis (como `upper()`, `lower()`, `replace()`) para manipulação de texto [2] [3].
- **Operadores e Estruturas Condicionais:** Operadores aritméticos (`+ - * /`), comparativos (`== != > < >= <=`) e lógicos (`and`, `or`, `not`, `in`) permitem construir expressões. Estruturas condicionais usam `if/elif/else` para controlar o fluxo com base em condições booleanas [4] [5]. Exemplo:

```
if saldo > 0:
    situacao = "Positivo"
elif saldo == 0:
    situacao = "Zerado"
else:
    situacao = "Negativo"
```

Esse bloco verifica o valor de `saldo` e define a situação financeira de acordo. Também é possível usar `if` em compreensão de listas ou com o operador ternário para simplificar expressões.

- **Laços de Repetição:** Utiliza-se `while` para repetir enquanto uma condição é verdadeira [6], e `for` para iterar sobre sequências (listas, strings, range numérico etc.) [7]. Por exemplo, para imprimir números de 1 a 5:

```
for num in range(1, 6):
    print(num)
```

O laço `for` acima usa `range(1,6)` para gerar os números de 1 a 5 [8]. O `while` pode ser usado quando não sabemos previamente o número de iterações, por exemplo:

```
contador = 1
while contador <= 5:
    print(contador)
    contador += 1
```

- **Estruturas de Dados:** Python possui estruturas compostas como **listas**, **tuplas**, **dicionários** e **conjuntos**. Listas (`[]`) armazenam uma sequência ordenada e mutável de itens de qualquer

tipo [9]. Podemos acessá-los por índice (iniciando em 0) e usar métodos como `append`, `pop` etc. Tuplas (`()`) são semelhantes a listas porém imutáveis. Dicionários (`{}`) armazenam pares chave-valor, permitindo acesso aos valores via chave. Conjuntos (`set()`) guardam elementos únicos (sem ordem definida). Os notebooks mostram, por exemplo, uma lista mista: `lista = ['Fabricio Daniel', 9.5, 9.0, 8.0, True]` [10]. Também é apresentado *list comprehension*, uma forma concisa de gerar listas a partir de iteráveis, e *dict comprehension* para gerar dicionários de forma compacta [11].

- **Funções:** São blocos reutilizáveis que executam uma tarefa. Definidas com `def nome_func(param1, ...):` seguido de um bloco indentado. Podemos retornar valores com `return`. No curso, vimos exemplos de funções para calcular média de notas [12] e para verificar aprovação de um estudante. Exemplo simplificado:

```
def media(notas):
    soma = sum(notas)
    return soma/len(notas)
```

Essa função aceita uma lista de `notas` e retorna a média. Também aprendemos sobre **funções anônimas (lambda)**, úteis para funções curtas e simples. Ex.: `resultado = list(map(lambda x: x*2, [1,2,3]))` aplicará uma função que duplica cada número [13].

- **Tratamento de Exceções:** Erros em tempo de execução (exceções) podem ser tratados para evitar que o programa quebre. Utiliza-se `try/except` para capturar exceções específicas e reagir adequadamente [14]. Por exemplo:

```
try:
    resultado = notas[nome_estudante]
except KeyError:
    print("Estudante não matriculado na turma")
else:
    print(resultado)
finally:
    print("Consulta encerrada")
```

No trecho acima, tentamos acessar um dicionário `notas` pela chave `nome_estudante`. Se a chave não existir (`KeyError`), exibimos uma mensagem e evitamos o *crash*. O bloco `else` executa caso nenhuma exceção ocorra, e `finally` executa sempre, garantindo que a mensagem final seja impressa [15] [16]. Tratar exceções é uma boa prática para tornar o código mais robusto e confiável.

NumPy

Abordado nos notebooks: *Regressão Linear – Técnicas avançadas de modelagem.ipynb* (uso de `numpy.log`), *Estatística com Python – Parte 2.ipynb* (distribuições)

NumPy é a biblioteca fundamental para computação numérica em Python. Ela introduz o objeto **array N-dimensional**, que é mais eficiente e permite operar vetorizadamente (ou seja, executar operações elemento a elemento de forma otimizada em código C). Conceitos-chave e recursos do NumPy incluem:

- **Arrays vs. listas Python:** Um *array* NumPy se assemelha a uma lista, mas ocupa menos memória e pode ser manipulado muito mais rapidamente em operações matemáticas massivas. Diferente das listas, os arrays NumPy são *homogêneos* (todos elementos de um array têm o mesmo tipo, geralmente numérico). Exemplo de criação de array:

  ```
  import numpy as np
  arr = np.array([10, 20, 30, 40])
  print(arr * 2)  # saída: [20 40 60 80]
  ```

  No exemplo acima, a multiplicação `arr * 2` é aplicada a *todos* os elementos simultaneamente (operação vetorizada), retornando um novo array com cada valor dobrado, de forma muito mais eficiente do que iterar item a item em Python puro.
- **Principais funções:** NumPy oferece diversas funções matemáticas e estatísticas. Por exemplo, `np.mean(arr)` calcula a média, `np.median(arr)` a mediana e `np.std(arr)` o desvio padrão de um array. Também há funções para criar sequências e arrays especiais: `np.arange(0, 10, 2)` gera um array [0, 2, 4, 6, 8]; `np.zeros((3,3))` cria uma matriz 3x3 de zeros; `np.ones(5)` um array de cinco elementos iguais a 1.
- **Manipulação de forma (shape) e operações matriciais:** Arrays NumPy possuem o atributo de forma (`shape`), indicando suas dimensões. Você pode redimensionar um array com `reshape`. A biblioteca suporta operações de álgebra linear (multiplicação de matrizes, transposição, inversão etc.) e computação de alto desempenho usando álgebra vetorial.
- **Uso em Data Science:** Nos notebooks, NumPy é utilizado, por exemplo, para aplicar transformações matemáticas nos dados. No projeto de regressão linear, foi usado `np.log` para aplicar a transformação logarítmica a uma variável com distribuição skew (assim aproximando-a de uma distribuição normal) [17]. NumPy também é base para outras bibliotecas; muitas funções do pandas e do scikit-learn utilizam arrays NumPy internamente. Sempre que for necessária performance em cálculos numéricos ou manipulações de grandes coleções de números, prefira usar NumPy ao invés de loops Python nativos.

Pandas

Abordado nos notebooks: *Projeto_imobiliaria_final – Conhecendo Pandas.ipynb*, *Pandas Transformação e Manipulação de dados.ipynb* (e usado em praticamente todos os projetos)

Pandas é a biblioteca essencial para **manipulação e análise de dados tabulares** (estruturados) em Python. A estrutura principal é o **DataFrame**, que pode ser imaginado como uma tabela (semelhante a uma planilha ou tabela SQL) com linhas e colunas nomeadas. Principais conceitos e funcionalidades:

- **Leitura de dados:** Pandas facilita importar dados de fontes diversas, como CSV, Excel, SQL, JSON, etc. Por exemplo, para ler um CSV usa-se `pd.read_csv('arquivo.csv')` [18]. Nos notebooks, vemos a importação de conjuntos de dados reais, como informações de imóveis [19], dados de aluguel, dados de clientes e vendas, entre outros. Após leitura, métodos como `df.head()` exibem as primeiras linhas do DataFrame para inspeção rápida [20], `df.shape` mostra o tamanho (nº de linhas, nº de colunas) [21], e `df.info()` resume tipos de cada coluna e contagem de valores não nulos.

- **Seleção e Indexação:** Podemos selecionar colunas como `df['Coluna']` ou linhas por rótulo/posição com `df.loc[]` e `df.iloc[]`. Também é possível filtrar linhas por condições lógicas, criando máscaras booleanas. Exemplo: `df[df['Valor'] > 1000]` retornaria um novo DataFrame apenas com imóveis de Valor > 1000. Os notebooks trazem exemplos práticos, como selecionar somente imóveis do tipo *Apartamento*:

```
filtrados = df.query('Tipo == "Apartamento"')
```

  O método `query` acima filtra `df` onde a coluna **Tipo** é "Apartamento" [22]. Alternativamente, poderia-se usar `df[df['Tipo'] == "Apartamento"]`. Pandas permite combinar múltiplas condições com operadores bitwise: por exemplo, `df[(df['Quartos'] >= 2) & (df['Valor'] < 3000)]` filtraria apartamentos com 2 ou mais quartos e valor menor que 3000.
- **Manipulação de colunas:** Renomear colunas (`df.rename`), criar novas colunas e alterar valores são tarefas comuns. Para criar coluna nova, basta atribuir: `df['NovaCol'] = ...`. Podemos derivar colunas de outras, como visto no projeto imobiliário em que foi criada uma coluna *Descrição* concatenando tipo e bairro do imóvel [23]. Pandas alinha operações por índice, então é fácil realizar operações como `df['Valor_mensal'] = df['Valor'] / 12` para obter, por exemplo, um valor mensal estimado a partir de um valor anual.
- **Estatísticas descritivas e resumo:** O DataFrame possui métodos para cálculo direto de estatísticas: `df['Coluna'].mean()`, `df['Coluna'].median()`, `df.describe()` (retorna contagem, média, desvio padrão, mínimos, quartis e máximo de colunas numéricas) etc. No projeto de estatística, por exemplo, `dados.Renda.mean()` foi usado para calcular a renda média da população pesquisada [24]. Também podemos contar frequências de categorias com `value_counts()`. Ex.: `dados['Sexo'].value_counts(normalize=True)` deu proporção de indivíduos de cada sexo na amostra (cerca de 69% masculino e 31% feminino) [25].
- **Agrupamento e agregação:** Com **Group By** é possível segmentar o DataFrame por categorias e calcular estatísticas em cada grupo. Por exemplo, para saber o valor médio de aluguel por tipo de imóvel:

```
df.groupby('Tipo')['Valor'].mean()
```

  Nos notebooks, isso foi demonstrado para obter médias de aluguel por categoria [26], e até para ordenar esses resultados [27]. Podemos aplicar diversas funções de agregação (`.mean(), .sum(), .min(), .max(), .count(), .agg()` etc.) e até múltiplas agregações simultaneamente.
- **Tratamento de dados faltantes:** Pandas representa valores ausentes como `NaN`. Métodos como `df.isnull()` identificam nulos, e podemos descartá-los (`df.dropna()`) ou substituí-los (`df.fillna(valor)`). Uma abordagem comum é preencher valores faltantes com média/mediana (para numéricos) ou uma categoria como "Desconhecido" (para strings), dependendo do contexto. No notebook de Pandas, há seção específica para *dados nulos* [28], mostrando como identificá-los e remover registros inválidos (por exemplo, imóveis com `Valor == 0` foram filtrados como possivelmente inconsistentes e removidos) [29].
- **Transformações e combinações de dados:** O Pandas possibilita mesclar DataFrames (joins) via `pd.merge`, concatenar tabelas (`pd.concat`), pivotar dados (`pivot_table`), reorganizar e transformar facilmente. No projeto *IA aplicada a Data Science*, dois DataFrames (clientes e vendas) foram combinados com base em um identificador de compra para análise integrada. No notebook de transformação, foi usado `pd.json_normalize` para **flatten** dados aninhados de um JSON em colunas do DataFrame [30]. Além disso, conversões de tipo (e.g. transformar strings

numéricas em float) e manipulações de datas (usando `pd.to_datetime`) foram aplicadas, ilustrando as poderosas funcionalidades de transformação.

- **Exportação de dados:** Após manipular ou filtrar, podemos salvar o resultado, por exemplo, `df.to_csv('saida.csv', index=False)` para guardar sem o índice. No notebook de Pandas, após filtrar apenas apartamentos, foi utilizado `df.to_csv('dados_apartamentos.csv')` para salvar esse subconjunto [31]. Pandas suporta também exportação para Excel (`to_excel`), JSON (`to_json`), SQL (`to_sql`), entre outros formatos.

(Dica: A documentação oficial do Pandas [32] é um ótimo recurso para aprofundar cada funcionalidade mencionada.)

Visualização de Dados

Abordado nos notebooks: *Python para Data Science* (exemplos Matplotlib básico), *Conhecendo Pandas* (gráficos de barras), *Regressão Linear II* (boxplot, scatterplot, pairplot), *Classificação* (matriz de confusão, curvas ROC/PR), *Projeto Final IA Data Science* (storytelling com gráficos)

A visualização de dados é crucial em Data Science para **explorar os dados** e **comunicar insights**. No curso, foram utilizadas principalmente as bibliotecas **Matplotlib** (base de visualização em Python) e **Seaborn** (biblioteca baseada no Matplotlib, com estilo aprimorado), além de métodos de plotagem do próprio Pandas. Alguns tipos de gráficos e casos de uso destacados:

- **Gráficos de Barra e Coluna:** Úteis para comparar valores entre categorias. No módulo introdutório, vimos um exemplo simples usando Matplotlib:

```
import matplotlib.pyplot as plt
estudantes = ["João", "Maria", "José"]
notas = [8.5, 9.0, 6.5]
plt.bar(x=estudantes, height=notas)  # gráfico de barras vertical
plt.show()
```

  O resultado exibe barras para as notas de cada estudante [33]. Já no projeto de Pandas, foi gerado um gráfico de barras horizontal comparando o valor médio de aluguel por tipo de imóvel. Nesse caso, aproveitaram a integração Pandas–Matplotlib: após calcular um DataFrame com médias por tipo via groupby, aplicaram `.plot(kind='barh', figsize=(14,10))` diretamente, produzindo um gráfico de barras horizontal ordenado [34].
- **Histogramas:** Mostram a distribuição de frequências de uma variável numérica, agrupando valores em faixas (bins). Foram utilizados para visualizar distribuições de renda, preços de imóveis e outras variáveis contínuas. Por exemplo, no projeto de regressão linear, traçou-se o histograma do preço dos imóveis para verificar sua assimetria (skew) e motivar a transformação logarítmica. Com Pandas, podemos fazer `df['Valor'].hist(bins=30)`, e com Matplotlib/Seaborn, `plt.hist(dados, bins=30)` ou `sns.histplot(dados, kde=True)` (que também plota a densidade).
- **Boxplot (Gráfico de Caixa):** Resume distribuição mostrando quartis (Q1, mediana, Q3) e possíveis outliers. No notebook de regressão, plotou-se um boxplot do preço dos imóveis (variável dependente) para identificar outliers e ver a dispersão [35]. Em Seaborn, é simples: `sns.boxplot(y=df['Valor'])` renderiza a caixa da variável Valor. Esse gráfico evidenciou uma cauda longa de preços, indicando alta dispersão e alguns valores bem acima da mediana.

- **Gráficos de Dispersão (Scatterplot):** Utilizados para visualizar relação entre duas variáveis contínuas. Por exemplo, para inspeção inicial de regressão, foram gerados scatterplots de `Valor` vs `Area`, `Valor` vs `Dist_Praia` etc., observando tendências (como era de se esperar, maior área tendia a correlacionar com maior preço do imóvel). Foi usado `sns.scatterplot(x='Area', y='Valor', data=df)` para cada par, e também o Seaborn *pairplot*, que automaticamente plotou matrizes de dispersão para todas combinações de variáveis selecionadas [36]. Esse recurso (`sns.pairplot(df)`) gera uma grade de gráficos de dispersão (e histogramas na diagonal), sendo muito útil na análise exploratória para ver correlações e distribuições de várias variáveis de uma só vez.
- **Mapa de Calor de Correlação:** Outra visualização vista no projeto de regressão foi a **matriz de correlação** entre variáveis explicativas e o target. Computou-se `df.corr()` para obter os coeficientes de correlação de Pearson e então utilizou-se Seaborn para plotar um heatmap colorido dessa matriz (com `sns.heatmap(corr, annot=True)`), facilitando a identificação de variáveis altamente correlacionadas positiva ou negativamente. Isso ajuda a evitar multicolinearidade e escolher quais variáveis incluir no modelo.
- **Gráficos de Série Temporal:** Embora não profundamente abordado, vale citar que pandas integra-se com Matplotlib para plotar séries temporais (usando index de datas) automaticamente no eixo x. No projeto de voos, por exemplo, poderíamos visualizar atrasos médios por ano ou mês com uma linha temporal, após agrupar dados por período.
- **Visualizações para Modelos:** Durante a avaliação de modelos de classificação, foram gerados gráficos especiais: a **Matriz de Confusão**, que é um tipo de tabela/visual gráfico mostrando acertos e erros de classificação (true/false positives/negatives). Foi utilizada a classe `ConfusionMatrixDisplay` do scikit-learn para plotar a matriz de confusão de um modelo que previa inadimplentes [37], facilitando ver onde ocorrem erros (e.g., quantos inadimplentes foram classificados erroneamente como adimplentes, e vice-versa). Além disso, foram traçadas a **Curva ROC** (Receiver Operating Characteristic) e a **Curva Precisão-Revocação** para avaliar o desempenho do modelo em diferentes limiares de decisão [38] [39]. Essas curvas ajudam a escolher o melhor limiar e a visualizar o trade-off entre *sensibilidade* e *especificidade*. A área sob a curva ROC (AUC) também foi calculada como métrica sumária [40].
- **Personalização:** Tanto Matplotlib quanto Seaborn permitem customizar cores, legendas, títulos, escalas etc. Nos projetos, vimos exemplos de ajuste de paleta de cores (`sns.set_palette`), estilos de grade de fundo (`sns.set_style`), dimensionamento de figuras (`plt.figure(figsize=(w,h))`), rótulos nos eixos (`plt.xlabel`, `plt.ylabel`), títulos (`plt.title`) e inclusão de legendas. Uma boa prática exibida é formatar gráficos para ficarem claros em relatórios (por exemplo, aumentar o tamanho de fonte ou rotular barras com valores quando necessário).

Em resumo, dominar visualizações permite verificar hipóteses rapidamente durante a análise exploratória e apresentar resultados de forma intuitiva. As bibliotecas utilizadas fornecem uma variedade enorme de gráficos prontos para usar, cobrindo desde gráficos básicos até plots estatísticos complexos.

Estatística Descritiva

Abordado nos notebooks: *Projeto_Final_Estatística_com_Python_resumindo_e_analisando_dados_Parte_2.ipynb*, *Regressão Linear II* (análise exploratória inicial), *Conhecendo Pandas* (métodos descritivos básicos)

Estatística descritiva reúne técnicas para **resumir e interpretar um conjunto de dados** de forma quantitativa, antes de partir para inferências ou modelos preditivos. No curso, trabalhamos com um dataset socioeconômico (PNAD 2015) e outros, aplicando medidas e conceitos chave:

- **Medidas de Tendência Central:** Indicadores que representam um valor "típico" do conjunto. A **média aritmética** é a mais comum – soma dos valores dividida pela contagem. Foi usada, por exemplo, para calcular a renda média da população na pesquisa PNAD (≈ 2000) [41]. A **mediana** é o valor central quando os dados são ordenados (50% acima, 50% abaixo) – útil pois não sofre tanta influência de outliers; nos dados de renda, a mediana costuma ser menor que a média em distribuições assimétricas positivas (muitas pessoas ganham pouco e poucas ganham muito). A **moda** é o valor mais frequente (relevante para variáveis categóricas ou distribuições discretas). Essas medidas foram discutidas no projeto de estatística (Parte 1 do curso, não fornecida aqui, cobriu cálculo de média, mediana, moda em Python/pandas).
- **Medidas de Dispersão:** Avaliam a variabilidade dos dados. O **desvio padrão** (σ) e **variância** (σ^2) medem o espalhamento em torno da média – vimos exemplos calculando o desvio padrão da altura e renda. Quartis e **intervalo interquartil (IQR)** são usados para entender distribuição: o IQR = Q3 - Q1 indica a amplitude central de 50% dos dados. No boxplot do preço dos imóveis, os quartis e possíveis outliers (pontos além de 1.5*IQR) ficaram evidentes, mostrando grande dispersão nos preços* [35]. *O range** (amplitude total = valor máx - mín) também foi observado (por exemplo, a renda máxima muito acima da mínima). Essas métricas ajudam a saber se os dados estão muito espalhados ou concentrados.
- **Distribuições de Frequência:** Para variáveis categóricas ou discretas, contamos frequências. No projeto estatístico, foi calculada a distribuição de frequências de níveis de escolaridade, cor/raça, etc., apresentando em tabelas ou gráficos de barras. A função `value_counts()` do pandas com `normalize=True` forneceu proporções, por exemplo ~30% da amostra era do sexo feminino [25]. Já para variáveis contínuas, construímos **histogramas** para ver a forma da distribuição (unimodal ou multimodal, simétrica ou assimétrica). Identificamos, por exemplo, que a distribuição de renda é assimétrica à direita (poucos ganham muito, muitos ganham pouco), o que fez a média ser bem maior que a mediana – um insight típico de dados de renda.
- **Formato da Distribuição:** Conceitos de **simetria** e **curtose** foram brevemente abordados. Observamos distribuições aproximadamente **normais** (forma de sino, simétricas, média ≈ mediana ≈ moda) em algumas variáveis, enquanto outras eram enviesadas. Reconhecer isso orienta decisões como aplicar transformações (log, por exemplo, para dados de preços/renda) para atender pressupostos de modelos. Também aprendemos que a distribuição normal tem propriedades úteis: ~68% dos dados dentro de 1 desvio padrão da média, ~95% dentro de 2σ, etc.
- **Outliers:** Dados estatísticos descritivos auxiliam a detectar outliers – valores muito distantes dos demais. Vimos isso visualmente em boxplots e nos próprios valores máximos em `describe()`. Por exemplo, um indivíduo com renda extremamente alta se destacaria. Decidir lidar com outliers (remover ou tratar) depende do contexto – às vezes indicam erro de dados, em outros casos são pontos legítimos e importantes (p.ex., grandes fortunas numa distribuição de renda). No projeto de estatística, registros de renda inválida (codificada como 999999999999) foram eliminados previamente [42].
- **Exemplo prático (PNAD 2015):** Foram calculadas medidas como média de renda, desvio padrão de altura, proporção de pessoas por nível de educação, etc., para traçar um perfil da amostra. Isso forneceu subsídios antes de partir para técnicas inferenciais e construção de intervalos de confiança (cobertos na Parte 2 do curso). Também simulamos amostragens: extraindo *amostras aleatórias* do dataset e comparando estatísticas da amostra com as da população original. No código, `dados.sample(n=1000, random_state=101)` extraiu 1000 registros aleatórios [43]; em seguida vimos que a média da amostra ficou próxima da média da população (ex.: renda

média ~1998 na amostra vs ~2000 na população) [24] [44], ilustrando o conceito de estimação amostral.
- **Distribuições de Probabilidade:** Embora além da *descritiva* estrito senso, no notebook Estatística Parte 2 entramos em **distribuições teóricas**: Binomial, Poisson, Normal. Entendemos parâmetros como média e variância dessas distribuições e calculamos probabilidades de eventos usando fórmulas e funções do SciPy. Por exemplo, calculamos a probabilidade de exatamente *k* sucessos em *n* tentativas (distribuição binomial) e a probabilidade de um certo número de eventos numa janela de tempo dada uma taxa (Poisson) [45] [46]. Esses conceitos conectam-se à estatística inferencial, mas reforçam a compreensão das características das distribuições – por exemplo, que na distribuição normal média = mediana = moda [47].

Em resumo, a estatística descritiva nos fornece **resumos numéricos e gráficos** para compreender os dados brutos. Antes de modelar ou concluir qualquer coisa, é fundamental aplicar essas técnicas: olhar para médias, medianas, dispersões, distribuições e outliers para ter certeza de que entendemos o que os dados estão nos dizendo.

Regressão Linear

Abordado nos notebooks: *Regressão Linear – Técnicas avançadas de modelagem.ipynb*, *Projeto_imobiliaria_final – Conhecendo Pandas.ipynb* (análise inicial do mesmo dataset de imóveis)

A regressão linear é uma técnica de **modelagem preditiva para variáveis numéricas contínuas**. O objetivo é encontrar uma relação linear (uma equação da forma $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$) que estime um alvo y a partir de um ou mais atributos x_i. Principais pontos estudados:

- **Contexto do Problema:** No projeto, buscamos avaliar imóveis no Rio de Janeiro. A variável alvo foi o **Valor** do imóvel (preço de oferta em R$) e tínhamos atributos como **Área** (m^2), **Distância à praia**, **Distância à farmácia**, etc. O primeiro passo foi entender o *dataset* – conferindo estatísticas descritivas de cada variável e a plausibilidade dos dados (por exemplo, verificar se havia valores faltantes ou inválidos). Em seguida, partiu-se para análise de relações entre variáveis para verificar se uma regressão linear faria sentido.
- **Análise Exploratória e Correlações:** Calculamos a **matriz de correlação** entre as variáveis. Observou-se, por exemplo, que *Área* tinha correlação positiva significativa com *Valor* (imóveis maiores tendem a custar mais), enquanto *Distância à praia* possuía correlação negativa (imóveis mais próximos do mar tendem a ser mais caros). Foi plotada uma matriz de dispersão (*pairplot*) para visualizar essas relações [48], confirmando tendências aproximadamente lineares em algumas combinações. Também fizemos boxplots e histogramas: identificamos que *Valor* era muito assimétrico (alguns imóveis de altíssimo valor). Essa descoberta levou à decisão de aplicar uma **transformação logarítmica** em *Valor* para aproximar a distribuição de um formato normal [49] [50] – uma prática comum para melhorar a qualidade do ajuste linear quando a variável alvo é skewed. Após a transformação, reavaliamos o histograma de log(Valor) e vimos a distribuição ficar mais simétrica.
- **Preparação dos Dados:** Selecionamos as variáveis explicativas relevantes (no caso, foram escolhidas Área, Dist_Praia, Dist_Farmacia – todas numéricas contínuas) e a variável alvo (Valor transformado). **Padronização** ou normalização dos atributos não foi necessária nesse caso específico para a regressão linear simples, mas é uma etapa a considerar se unidades forem muito diferentes ou se for usar regularização. *Dummies* para variáveis categóricas também não foram necessárias porque todos os atributos selecionados eram numéricos; contudo, fica o lembrete: em problemas gerais de regressão com variáveis categóricas, precisamos convertê-las em numéricas (p. ex. usando `pd.get_dummies` ou OneHotEncoder).

- **Divisão em Treino e Teste:** Sempre importante separar dados para **treino** do modelo e **teste** (validação) do modelo. Utilizamos `train_test_split` do scikit-learn para dividir o conjunto (por exemplo, 70% treino, 30% teste, aleatoriamente) [51]. Essa prática garante avaliar o modelo em dados não vistos durante o treino, medindo sua capacidade de generalização.
- **Treinamento do Modelo:** Empregamos duas abordagens: a biblioteca **statsmodels** para obter uma análise estatística completa, e o **scikit-learn** para treinamentos e previsões rápidas. Com statsmodels, construímos um modelo de Mínimos Quadrados Ordinários (OLS): adicionamos uma constante (β_0) e ajustamos o modelo. O resultado forneceu um resumo detalhado – coeficientes β_i estimados, seus erros padrões, valores *t* e *p* (testes de significância individual), R^2 do modelo, estatística F (teste de significância global), etc. [52] [53]. Observamos, por exemplo, que após a transformação logarítmica, o modelo linear teve melhora no R^2 e coeficientes mais interpretáveis. Com scikit-learn, o processo foi:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
# X_train: atributos selecionados, y_train: alvo
```

  Após o `fit`, podíamos inspecionar `model.coef_` (array de coeficientes $\beta_1 ... \beta_n$ estimados) e `model.intercept_` (termo β_0). Por exemplo, obtivemos algo como $\beta_{Área} \approx 5600$ (cada metro quadrado a mais acrescentando ~R$5.600 ao valor, mantendo demais fatores constantes) – claro, esse é um exemplo ilustrativo; no caso real foi feito na escala log, interpretação muda (coeficientes representavam percentuais aproximados de variação).
- **Avaliação do Modelo:** Para medir desempenho, usamos principalmente o **R^2 (coeficiente de determinação)**, que indica a proporção da variação da variável resposta explicada pelo modelo. No primeiro modelo com todos atributos, obtivemos um R^2 de, digamos, ~0.65 (65% da variação de preço explicada por área e distâncias). Também analisamos a significância dos coeficientes: no resumo do statsmodels, p-valores baixos (<0.05) indicaram que *Área* e *Dist_Praia* eram estatisticamente significativos, mas *Dist_Farmacia* não contribuía muito – o que nos levou a testar um segundo modelo removendo essa variável. No segundo modelo ajustado sem *Dist_Farmacia*, o R^2 caiu muito pouco e simplificamos a modelagem. Além de R^2, poderíamos calcular **MSE** (erro quadrático médio) ou **RMSE** (raiz do MSE) no conjunto de teste para ter a medida do erro médio de previsão em unidades de preço – por exemplo, um RMSE de 300 mil indicaria que, em média, o erro padrão das predições de preço é 300 mil R$. Esses cálculos podem ser feitos via `sklearn.metrics.mean_squared_error`.
- **Interpretação e Uso do Modelo:** Com o modelo final treinado (digamos, log(Valor) vs Área e Dist_Praia), podemos usar `model.predict(X_test)` para obter predições no conjunto de teste e avaliar quão próximos estão dos valores reais. Observamos graficamente que as predições seguiam a tendência dos reais, mas ainda havia dispersão (especialmente em preços muito altos, que são mais difíceis de prever). No projeto, enfatizou-se a importância de validar supostos do modelo linear: normalidade dos resíduos (com gráficos Q-Q), homocedasticidade (avaliar se resíduos têm variância constante ao longo do eixo, p.ex. plotando resíduos vs valores ajustados) e independência. Alguns desses diagnósticos foram possíveis via output do statsmodels e análises gráficas não exibidas no texto, mas que são boas práticas em regressão.
- **Melhorias e Técnicas Avançadas:** O nome do notebook fala em técnicas avançadas, e de fato aplicamos algumas: *transformação de variável* (log), *remoção de outliers* (poderia-se remover imóveis extremamente caros para ver efeito, embora não tenha sido necessário), e mencionou-se regularização (não aprofundado, mas citou-se que existe Ridge/Lasso quando há muitas variáveis). Também aprendemos sobre **multicolinearidade** – se dois atributos fossem altamente

correlacionados entre si, isso poderia distorcer coeficientes; observando a matriz de correlação, não havia colinearidade severa nesse conjunto reduzido, então não foi preciso tratar, mas em outros cenários poderíamos eliminar ou combinar variáveis correlacionadas.

Em resumo, construímos um modelo de regressão linear multivariada para prever preços de imóveis. Vimos passo a passo desde entender os dados, adequar distribuições, selecionar variáveis, treinar o modelo e avaliar sua performance. A regressão linear mostrou ser um método relativamente simples e interpretável: podemos explicar o efeito de cada variável no preço do imóvel e obter estimativas rapidamente. Contudo, notamos que ela tem limitações (não capturou 100% da variação, possivelmente pelas variáveis importantes que ficaram de fora, relações não-lineares, etc.), o que abre espaço para modelos mais complexos ou uso de features adicionais em projetos futuros.

Classificação

Abordado nos notebooks: *Classificação – aprendendo a classificar dados com ML.ipynb*, *Classificação – validação de modelos e métricas.ipynb*

A tarefa de classificação consiste em **predizer categorias** (classes) para os dados. No curso, trabalhamos com casos de classificação binária: identificar se um cliente aderiu ou não a um investimento, prever se um cliente será inadimplente ou não, etc. Passos e conceitos principais abordados:

- **Entendimento dos Dados e Preparação:** Iniciamos carregando os dados, como no caso de **marketing de investimento** (onde cada linha era um cliente de banco com atributos demográficos e comportamentais, e o rótulo indicava se fez ou não uma aplicação financeira) [54], e no caso de **empréstimo de automóveis** (atributos financeiros, scores de crédito, e rótulo inadimplente=0/1) [55] [56]. Após a leitura, fizemos análises exploratórias: frequência das classes (quantos “sim” vs “não”, ou adimplentes vs inadimplentes), média e distribuição dos atributos por grupo, etc., para entender padrões iniciais. Notamos, por exemplo, que no dataset de inadimplentes havia desbalanceamento (poucos inadimplentes em comparação aos adimplentes) – uma observação importante para a etapa de avaliação. Também identificamos possíveis *outliers* ou valores estranhos nos atributos (ex.: um cliente com saldo negativo incomum, o que levantou atenção).
- **Pré-processamento:** Incluiu tratar dados faltantes (remover ou imputar – nos datasets fornecidos, felizmente, não havia muitos nulos) e converter variáveis categóricas em numéricas. No conjunto de marketing, por exemplo, *estado_civil* tinha valores como "solteiro (a)", "casado (a)" – aplicamos *label encoding* ou *dummies* conforme necessário. Para simplificar, muitos algoritmos lidam internamente com variáveis binárias (ex.: "inadimplente" já estava 0/1), mas para categorias como escolaridade (p. ex. "médio", "superior") criamos indicadores ou uma ordem se fizesse sentido. Também verificamos escalas: alguns modelos (como KNN ou regressão logística) podem se beneficiar de normalização de atributos contínuos; já árvores de decisão e random forest não requerem escalonamento.
- **Divisão Treino/Teste:** Novamente vital, separamos parte dos dados para testar o modelo depois. No notebook de classificação de inadimplentes, além do treino-teste simples, enfatizamos **estratificação**: usamos `train_test_split(..., stratify=y)` para garantir que a proporção de classes no conjunto de treino reflete a do conjunto original [57]. Isso é importante especialmente em casos de classes desbalanceadas, para que o modelo não acabe treinado em uma amostra com proporção diferente da realidade.
- **Modelos de Classificação Utilizados:** Começamos com um modelo simples, frequentemente **Árvore de Decisão** (Decision Tree). No notebook, importamos e instanciamos um `DecisionTreeClassifier` do scikit-learn [58]. Sem muitas configurações iniciais, ajustamos a

árvore aos dados de treino (`arvore.fit(X_treino, y_treino)`) [58] e fizemos previsões em teste (`y_pred = arvore.predict(X_teste)`). A árvore de decisão é intuitiva (faz perguntas sequenciais sobre os atributos para classificar) e não exige normalização ou dummies explícitas para variáveis categóricas (ela lida internamente via critérios de impureza). Também experimentamos limitar a profundidade da árvore (`max_depth=3`) para evitar **overfitting** [59] – árvores muito profundas podem memorizar os dados de treino. Outro modelo possivelmente testado foi a **Regressão Logística** (um classificador linear que estima probabilidades sigmoid), embora o foco maior tenha sido nas árvores. Além disso, para o dataset de marketing, chegou-se a comparar múltiplos modelos (árvore, k-vizinhos, floresta aleatória, etc.) usando validação cruzada, para escolher o melhor.

- **Avaliação de Desempenho:** Usamos várias **métricas de classificação**: a **Acurácia** (proporção de acertos) foi a primeira métrica observada. No exemplo de modelo inicial, uma árvore não podada chegou a acurácia 1.0 nos dados de treino (overfitting claro) e ~0.90 nos de validação [60]. Porém, acurácia sozinha pode enganar quando as classes são desbalanceadas – por isso introduzimos **Precisão** e **Recall**. *Precisão* (Precision) responde: dentre as predições de *positivo* feitas, quantas realmente eram positivas? *Recall* (Sensibilidade ou Revocação) responde: dentre os casos positivos no conjunto, quantos o modelo conseguiu identificar? No problema de inadimplentes, por exemplo, temos interesse alto em *recall* da classe 1 (inadimplente) – não queremos deixar de identificar maus pagadores. No notebook, calculamos precisão e recall do modelo usando `precision_score(y_true, y_pred)` e `recall_score(y_true, y_pred)` [61]. Imprimimos esses valores para o conjunto de validação [62]. Também extraímos a **Matriz de Confusão**, onde pudemos contar quantos FP (falso positivo), FN (falso negativo) etc. ocorreram [63]. Por exemplo, visualizamos que nosso modelo podia estar acertando 90% no geral, mas talvez ainda errasse muitos inadimplentes (mostrando a importância de ajustar ou considerar técnicas de balanceamento).
- **Curvas e Reporte:** Plotamos a **Curva ROC** do modelo e calculamos a AUC (Área sob a curva) [38]. A ROC relaciona taxa de verdadeiros positivos vs. falsos positivos para vários limiares de decisão do modelo (no caso da árvore, limiar é trivial 0.5 pois ela dá classificação direta; mas para modelos probabilísticos, podemos variar o ponto de corte). Uma AUC próxima de 1 indica excelente separação, enquanto 0.5 seria aleatório. Também plotamos a **Curva Precisão-Revocação** [39], muito útil em casos desbalanceados: mostra a precisão obtida para cada nível de recall. E geramos o **classification report** (`classification_report(y_true, y_pred)`) que resume acurácia, precisão, recall e F1-score (média harmônica de precisão e recall) para cada classe [64] – esse relatório nos deu uma visão mais completa: por exemplo, vimos que para a classe minoritária a precisão ou o recall podiam estar bem mais baixos que para a classe majoritária, chamando atenção para possível necessidade de melhorar o modelo ou ajustar limiar.
- **Validação Cruzada e Ajuste de Modelos:** No segundo notebook de classificação, dedicamos bastante atenção à **validação cruzada**. Implementamos um *K-Fold* manualmente e com `cross_val_score` do sklearn para avaliar o modelo em múltiplas divisões dos dados [65]. Isso produz uma distribuição de métricas (ex.: várias acurácias ou recalls obtidos em cada fold). Calculamos a média e intervalo de confiança dessas métricas, tornando a avaliação mais robusta e menos dependente de uma única divisão treino/teste [66]. Por exemplo, obtivemos uma média de recall ~0.92 com desvio ~0.03 para inadimplentes, indicando consistência razoável [67]. Além disso, testamos diferentes hiperparâmetros – p.ex., variar `max_depth` da árvore, ou comparar com outro algoritmo. Vimos que limitar a profundidade reduziu um pouco a acurácia, mas evitou overfitting (a acurácia de validação se manteve próxima da de treino, sugerindo modelo mais generalizável) [59]. Em algumas iterações, também aplicamos técnicas como **undersampling/oversampling** para lidar com o desbalanceamento (não detalhado nos notebooks fornecidos, mas mencionado como possibilidade).

- **Algoritmos Alternativos:** Foi citado e em parte testado outros algoritmos de classificação: **k-Nearest Neighbors (KNN)** – um método baseado em instâncias, **Random Forest** – um conjunto de múltiplas árvores de decisão que tende a dar melhor performance e reduzir overfitting, e **Regressão Logística**. Cada um tem suas vantagens: RF lida bem com muitos atributos e interações, KNN é simples mas pode ser ineficiente com muitos dados, e regressão logística fornece probabilidades interpretáveis e é leve. Comparações entre eles foram feitas usando validação cruzada para ver qual tinha melhor recall de inadimplentes, por exemplo. No final, a Random Forest se mostrou promissora (devido à capacidade de generalização), mas a escolha do "melhor" dependeu também da simplicidade e interpretabilidade desejada.

Em suma, aprendemos a **construir modelos de classificação**, começando pela preparação cuidadosa dos dados, passando pelo treinamento de algoritmos (especialmente árvores de decisão) e terminando numa avaliação criteriosa com múltiplas métricas – essencial para problemas onde um simples percentual de acertos não conta toda a história. A combinação de técnicas como curva ROC, análise de erros via matriz de confusão, e validação cruzada fornece um panorama completo do desempenho e confiabilidade do classificador antes de implantá-lo.

Avaliação de Modelos

Abordado nos notebooks: *Classificação – validação de modelos e métricas.ipynb*, *Regressão Linear II* (avaliação do ajuste estatístico), *IA aplicada a DS* (importância de storytelling de resultados)

Avaliar um modelo de forma correta é tão importante quanto construí-lo. Nesse tópico, consolidamos boas práticas e métricas para avaliar modelos de aprendizado de máquina, cobrindo tanto modelos de regressão quanto de classificação:

- **Treino vs. Teste – Overfitting:** Uma lição fundamental reforçada foi **não avaliar um modelo nos mesmos dados usados para treiná-lo**. Isso inflaria indevidamente a medida de desempenho, pois o modelo "já conhece" aqueles exemplos. Em vez disso, separamos dados de **treino** (para ajustar os parâmetros do modelo) e dados de **teste/validação** (mantidos "escondidos" até a avaliação) [68] . Vimos casos práticos: uma árvore de decisão obteve acurácia 100% no conjunto de treino (memoriza perfeitamente), mas ~90% nos dados de teste [60] , evidenciando overfitting. Para mitigar esse sobreajuste, além da separação treino/teste, utilizamos técnicas como limitar a complexidade do modelo (por ex., profundidade da árvore) e **validação cruzada** para garantir que o desempenho é consistente em diferentes subconjuntos.
- **Validação Cruzada (Cross-Validation):** Consiste em dividir repetidamente os dados em partes de treino e teste e medir o desempenho em múltiplas rodadas. Implementamos especialmente o **K-Fold CV**, onde o dataset é dividido em *K* partes e o modelo treinado/testado *K* vezes, cada vez usando uma parte diferente como teste. No notebook de classificação, usamos `KFold(n_splits=5)` e a função `cross_val_score` para obter 5 valores de métrica (usamos *recall* dos inadimplentes como métrica principal nesse caso) [57] . Extraímos a média e também construímos um intervalo de confiança para essa métrica, dando uma noção da variabilidade. Por exemplo, obtivemos recall médio ~0.92 com ±0.03 – significando que esperamos que em geral o modelo consiga pegar ~92% dos inadimplentes, podendo variar alguns pontos percentuais dependendo da amostra [67] . A validação cruzada nos protege de conclusões tiradas de uma única sorte de divisão treino/teste, que pode não ser representativa.
- **Métricas para Regressão:** Para modelos de regressão, as métricas comuns incluem **Erro Quadrático Médio (MSE)** e sua raiz (RMSE), **Erro Absoluto Médio (MAE)** e o **R^2**. No projeto de regressão linear, o R^2 do modelo final foi uma métrica-chave (e.g., ~0.65 indicando capacidade moderada de explicação) e interpretamos também o MAE (em unidades monetárias) para ver o erro médio absoluto em reais, o que ajuda a traduzir o que aquele erro significa no domínio do

problema. Ao comparar modelos, menores valores de MSE/RMSE ou MAE indicam melhor ajuste (desde que comparando no mesmo conjunto de teste). Vale notar a importância de olhar gráficos de resíduos para ver se o modelo viola suposições (por ex., padrão nos resíduos pode indicar tendência não capturada).

- **Métricas para Classificação:** Já listadas anteriormente, mas recapitulando de forma sistemática:
- **Acurácia**: % de acertos totais. É intuitiva mas pode ser enganosa se uma classe for rara (acertar sempre a classe majoritária pode dar alta acurácia e zero utilidade).
- **Precisão (Precision)**: dos positivos previstos, quantos são realmente positivos. Importante quando o custo de um falso positivo é alto (ex.: modelo acusa fraude quando não há – incomoda clientes, por exemplo).
- **Recall (Sensibilidade)**: dos positivos reais, quantos o modelo pega. Crucial quando o custo de um falso negativo é alto (ex.: deixar de detectar um inadimplente pode gerar prejuízo).
- **F1-Score**: média harmônica de precisão e recall, dá um equilíbrio das duas – útil para comparar modelos quando há trade-off.
- **Matriz de Confusão**: tabela 2x2 (no caso binário) detalhando VP, FP, FN, VN. Nós a usamos tanto numericamente quanto visualmente (com `ConfusionMatrixDisplay`) para identificar onde o modelo estava errando [63] [69]. Por exemplo, vimos casos onde havia ainda um número considerável de falsos negativos – sinal para tentar aumentar recall possivelmente às custas de precisão, dependendo do objetivo.
- **ROC e AUC**: a curva ROC mostra a relação entre TPR (taxa de verdadeiros positivos = recall) e FPR (taxa de falsos positivos) para diversos thresholds. O AUC resume essa curva em um número de 0 a 1 – quanto mais próximo de 1, melhor separador de classes é o modelo. Traçamos a ROC e calculamos AUC para nosso modelo de classificação, obtendo um AUC em torno de 0.95, por exemplo, o que indica ótima separação [38].
- **Curva Precisão-Recall**: principalmente avaliada quando a classe positiva é rara. Plotamos precisão vs. recall e calculamos a *Average Precision* (área sob a curva P-R). Isso complementa a AUC, pois às vezes o ROC pode parecer bom mesmo quando o modelo não tem alta precisão para a classe minoritária. No nosso caso, analisamos essa curva para verificar em que ponto de recall a precisão caía muito e decidimos um limiar de decisão apropriado (por exemplo, aceitar um recall de ~90% com precisão ~50%, dependendo dos custos).
- **Relatório de Classificação**: gerado pelo sklearn, consolidando todas as métricas por classe e a média. Ele facilitou a comparação entre modelos e ajustes – por exemplo, após ajustar hiperparâmetros ou balancear a classe, poderíamos ver o F1 da classe "1" subir, indicando melhoria.
- **Consideração de Negócio na Métrica:** Ressaltou-se que a escolha da métrica depende do objetivo do projeto. No caso de inadimplência, supõe-se que pegar o máximo de inadimplentes (alto recall) era prioritário, mesmo que viessem alguns falsos alarmes (precisão um pouco menor). Então avaliamos modelos principalmente pelo recall de classe 1. Em outro contexto (detecção de fraude, por exemplo), poderíamos priorizar precisão para não acusar indevidamente clientes legítimos. Portanto, **sempre relacione a métrica ao problema real**.
- **Validação Estratificada e de Conjunto Externo:** Além de estratificar na validação cruzada (para manter proporções de classes), também discutimos que o ideal é, se possível, **reservar um conjunto de teste final** apenas para a avaliação derradeira, usando validação cruzada apenas no treino. Isso garante que nenhuma etapa de seleção de modelo/hiperparâmetro "vaze" informação do teste. Em nosso caso didático, usamos só treino/teste por simplicidade, mas enfatizamos que em projetos profissionais um *hold-out* final ou validação cruzada nested pode ser utilizada.
- **Interpretação de Resultados:** Avaliar não é só calcular número – é interpretar o que significa. Por exemplo, quando vimos uma acurácia de ~90%, tivemos o cuidado de verificar a taxa base (se 85% dos clientes já eram adimplentes, um modelo trivial obteria 85% de acerto prevendo

sempre "não inadimplente"; então 90% não é tão impressionante assim). Por isso olhamos as outras métricas para ver ganho real. Do lado da regressão, interpretar um R^2 de 65% no contexto imobiliário nos diz que ainda há 35% de variação de preço não explicada – possivelmente atributos não considerados (localização exata, número de quartos, estado do imóvel), ou fatores aleatórios – logo, o modelo tem limitações previsíveis. Essas interpretações guiariam recomendações e próximos passos (adquirir mais dados, usar modelo não-linear, etc.).

Em conclusão, avaliar modelos requer um conjunto de **ferramentas quantitativas (métricas, validação cruzada)** e **bom senso analítico**. O curso enfatizou evitar armadilhas como overfitting e avaliação enviesada, e escolher métricas que estejam alinhadas com os objetivos do negócio/projeto. Com essas práticas, garantimos que quando um modelo for implantado, teremos confiança de que ele performará de maneira consistente e atenderá às expectativas do problema real.

Boas Práticas em Projetos de Ciência de Dados

Abordado nos notebooks: *Todos os projetos finais (imobiliário, classificação, IA aplicada a DS etc.)* – as etapas e práticas a seguir estiveram presentes de forma transversal.

Nesta seção final, reunimos recomendações gerais e **boas práticas** demonstradas ao longo do curso para conduzir projetos de ciência de dados de forma eficaz, organizada e confiável:

- **Entendimento do Problema e dos Dados:** Antes de qualquer código, é fundamental esclarecer a pergunta de negócio ou objetivo do projeto. Nos notebooks, cada projeto começa com uma contextualização: seja a necessidade de uma empresa aumentar adesão em investimentos, identificar inadimplentes, ou visualizar faturamento de um e-commerce. Esse contexto orienta todas as decisões seguintes (como quais variáveis podem ser relevantes, que métrica otimizar, etc.). Sempre investigue a origem dos dados, entenda o que cada coluna significa (foi feito dicionário de variáveis no projeto estatístico [70] [71]) e avalie se os dados disponíveis são suficientes para responder à questão.
- **Análise Exploratória de Dados (EDA):** Dedicar tempo à exploração inicial é uma ótima prática. Isso inclui gerar estatísticas descritivas (média, quartis, distribuições) e visualizações simples. No curso, cada projeto teve uma seção de EDA bem definida antes de modelar. Por exemplo, no caso dos imóveis, analisamos distribuição de preços e relações com área antes de ajustar a regressão; no caso de marketing, vimos proporção de clientes que aderiram e comportamentos por faixa etária antes de treinar um classificador. A EDA frequentemente revela problemas de qualidade (valores faltantes, outliers absurdos, inconsistências) que precisam de tratamento – melhor descobri-los no início do que depois de modelar.
- **Limpeza e Preparação de Dados:** Dados do mundo real raramente vêm prontos. As boas práticas incluem:
- *Tratar valores ausentes:* Decidir se deve excluir registros ou imputar valores (média/mediana para numéricos, modos para categóricos, ou modelos de imputação). No projeto estatístico, entradas com renda inválida foram removidas [42] para não distorcer a análise.
- *Tratar outliers:* Verificar se são erros de digitação/coleta (ex: idade = 300 anos claramente inválido) – nesses casos, corrigir ou remover. Se forem valores reais porém extremos, considerar técnicas como transformação (log, winsorização) ou modelos robustos.
- *Feature engineering:* Criar variáveis auxiliares que façam sentido. Vimos um exemplo simples: criar a coluna "Descrição" unindo tipo + bairro do imóvel [23]; em outros cenários, poderia ser extrair mês/ano de uma data, ou agrupar categorias raras em "Outros". Boas novas features podem melhorar muito os modelos.
- *Encoding de variáveis categóricas:* Converter categorias em representação numérica sem perder informação. Usamos *one-hot encoding* (dummies) quando necessário, ou mapeamentos simples

(sim/não em 1/0). Sempre conferir se não introduzimos colinearidade (no caso de dummies, remover uma coluna de referência).

- *Escalonamento:* Padronizar ou normalizar atributos contínuos para faixa comparável, principalmente se algoritmos baseados em distância ou gradiente (SVM, redes neurais, regressão logística regularizada, etc.) forem usados. No curso, isso foi mencionado; por exemplo, *score_1, score_2* em um dataset variavam 0-1 enquanto *receita_cliente* estava na ordem de milhares – normalizar poderia ajudar alguns modelos, embora a árvore não precise.
- **Divisão Treino/Validação/Teste:** Já detalhada anteriormente, mas reforçando: use um conjunto de **treino** para ajustar o modelo e *não olhe* os dados de **teste** até a etapa final de avaliação. Se possível, tenha um conjunto de **validação** separado para escolher hiperparâmetros (ou use validação cruzada) e somente após definir o modelo final, avalie no teste para obter a performance real esperada. Essa abordagem triplamente particionada evita *overfitting* indireto do teste durante a fase de tuning. No curso, muitas vezes usamos apenas treino/teste por simplicidade didática, mas sempre com o cuidado de não usar o teste na modelagem.
- **Escolha de Métricas Adequadas:** Direcione sua avaliação pelas métricas alinhadas com o objetivo do negócio. Essa é uma prática essencial – já discutimos que nem sempre acurácia ou R^2 contam a história completa. Em projetos, deixe claro para as partes interessadas o que cada métrica significa (ex.: "Nosso modelo identifica 90% dos inadimplentes com 10% de falso alarme" é mais informativo do que dizer "94% de acurácia"). No notebook de classificação, essa comunicação apareceu quando imprimimos precisão e recall e explicamos o que representavam em termos de clientes identificados [61] .
- **Simplicidade e Interpretabilidade:** A menos que o problema exija, comece com modelos mais simples e interpretabis (regr. linear, árvore de decisão rasa, etc.). Eles fornecem *baseline* e insights iniciais. Por exemplo, a árvore de decisão inicial mostrou quais variáveis e divisores ela usava – isso ajuda a entender o fenômeno. Se modelos complexos (Random Forest, XGBoost, Redes Neurais) forem usados, tente extrair alguma interpretabilidade – como features importances, ou usar técnicas como SHAP values – para validar se o modelo faz sentido ou se está se apoiando em fatores espúrios. No curso, vimos a importância de verificar importâncias numa Random Forest de exemplo (não detalhado no texto, mas geralmente se faz via `model.feature_importances_`). Uma boa prática é não tratar o modelo como uma caixa-preta total, sempre buscar entender *por que* ele está prevendo X ou Y.
- **Iteração e Experimentação Controlada:** Ciência de dados é iterativa. Teste diferentes abordagens, mas **mantenha registro das tentativas**. Nos notebooks, cada grande mudança (ex: aplicar log, remover variável, ajustar profundidade da árvore) foi feita em uma nova célula, comparando resultados antes/depois. Isso é importante para aprendizado e para justificar decisões. Recomenda-se usar ferramentas de versionamento (Git) para código e mesmo parâmetros de modelo, especialmente em projetos longos. No ambiente de notebook, adotar uma boa organização de seções (como foi feito, com títulos claros por etapa) ajuda a manter o controle do que já foi feito.
- **Tratamento de *Bias* e *Variance*:** Embora não formalizado nesses termos no material, praticamos abordar o trade-off viés/variância. Por exemplo, quando limitamos a complexidade da árvore (reduzindo variância do modelo, aumentando ligeiramente viés), fizemos isso para melhorar generalização. Boas práticas incluem diagnosticar overfitting (modelo va bem em treino e mal em teste) vs underfitting (vai mal em ambos), e então aplicar a correção adequada (complexificar mais ou simplificar/regularizar).
- **Uso de IA Aumentada e Ferramentas:** No projeto *IA aplicada a Data Science*, foi demonstrado uso do ChatGPT para auxiliar na geração de código de visualização e análise. Essa é uma prática emergente interessante: ferramentas de IA podem agilizar tarefas repetitivas ou ajudar com sintaxe de biblioteca. Entretanto, reforça-se que o especialista de dados deve validar e entender o output. No caso, o ChatGPT foi usado para esboçar gráficos solicitados e responder perguntas

exploratórias rapidamente, mas sempre sob supervisão. Isso pode aumentar a produtividade mantendo a qualidade (desde que haja revisão humana).

- **Comunicação e Visualização dos Resultados:** Boas práticas finais incluem **apresentar os insights** de forma clara para stakeholders não técnicos. Todos os gráficos que produzimos servem a esse propósito – no projeto final da Zoop (e-commerce), por exemplo, geramos visualizações sobre perfil de clientes e faturamento para contar uma história de negócio. É recomendado enfatizar as descobertas-chave (ex.: "Clientes do tipo X contribuem com Y% do faturamento"), usando gráficos limpos e textos explicativos. Além disso, documente o processo e as limitações: nos notebooks, há textos markdown explicando cada etapa e inferência (por exemplo, apontando a necessidade de transformação logarítmica, ou interpretando a matriz de confusão). Em um relatório final, isso se traduziria em conclusões e possivelmente sugestões (ex.: "Recomenda-se implementar o modelo de classificação no processo de análise de crédito para automatizar a detecção de alto risco, com expectativa de recall de ~90%").

Em suma, um projeto de ciência de dados de qualidade envolve **mais do que aplicar algoritmos** – requer planejamento, limpeza diligente, escolhas racionais de técnica, validação rigorosa e comunicação efetiva. Ao seguir essas boas práticas apresentadas no curso, aumentamos as chances de obter modelos que não apenas tenham boa performance nos dados, mas que sejam confiáveis, interpretáveis e realmente úteis na tomada de decisão no mundo real.

---

1 2 3 4 5 6 7 8 9 10 Projeto_Final-Python_Data_Science.ipynb
file://file-Ef115H3z4AGUnqsTFXnLH3

11 12 13 14 15 16 33 Python para Data Science trabalhando com funções, estruturas de dados e exceções.ipynb
file://file-3oGpckapFwERWDCA6V8sgd

17 35 36 48 49 50 51 52 53 Regressão Linear Tecnicas avançadas de modelagem.ipynb
file://file-1TbYdHWjPVFP9j9Ddco5Ne

18 19 20 21 22 23 26 27 28 29 31 32 34 Projeto_imobiliaria_final - Conhecendo Pandas.ipynb
file://file-9yVtHkcXKZGp3seHopRxWc

24 25 41 42 43 44 45 46 47 70 71
Projeto_Final_Estatística_com_Python_resumindo_e_analisando_dados_Parte_2.ipynb
file://file-HM96LSKVHnBp456uZhBZ4Y

30 Pandas Transformação e Manipulação de dados.ipynb
file://file-TbvM7vS9vKf4pNXoT2AYT7

37 38 39 40 55 56 57 60 61 62 63 64 65 66 67 68 69 Classificação validação de modelos e métricas de avaliação.ipynb
file://file-5bysfJRUxLZsubd51L4QEL

54 58 59 Classificação aprendendo a classificar dados com Machine Learning.ipynb
file://file-LfCXonTYD3BUnpQFBF8BTn