◇ 1. Verificar Dados (Análise Inicial)

✔ Teoria:

Antes de qualquer limpeza, é essencial entender como estão os dados. Verificar as primeiras linhas, os tipos das variáveis, a existência de valores ausentes e as estatísticas básicas ajuda a planejar os próximos passos.

✔ Funções e Parâmetros:

head(n) → Mostra as primeiras n linhas (padrão 5).

info() → Exibe tipos de dados, quantidade de não-nulos e memória.

describe() → Estatísticas descritivas (média, desvio, mín., máx., etc.) de colunas numéricas.

✔ Exemplo:

```
import pandas as pd

# Ler o arquivo CSV
dados = pd.read_csv('dados.csv')

# Ver primeiras 5 linhas
print(dados.head())

# Informações gerais sobre os dados
print(dados.info())

# Estatísticas básicas
print(dados.describe())
```

◇ 2. Verificação e Conversão de Tipos

✔ Teoria:

Dados importados podem vir com tipos incorretos. Por exemplo, números como texto, datas como strings ou dados categóricos como objetos. Isso pode afetar filtros, cálculos e análises.

✔ Função:

astype(tipo) → Converte o tipo da coluna.

Parâmetro:

tipo: o tipo desejado (int, float, str, datetime, etc.).

✔ Exemplo:

```
# Conferir tipos
print(dados.dtypes)

# Converter idade para inteiro
dados['idade'] = dados['idade'].astype(int)

# Converter uma coluna para string
dados['cpf'] = dados['cpf'].astype(str)
```

◇ 3. Identificação de Dados Faltantes

✔ Teoria:

Dados faltantes (null, NaN) são comuns em bases reais e precisam ser tratados, pois muitos modelos e análises não aceitam valores nulos.

✔ Funções:

isnull() → Retorna um dataframe booleano indicando onde há nulos.

isnull().sum() → Soma de nulos por coluna.

✔ Exemplo:

```
# Verificar onde há valores nulos
print(dados.isnull())

# Contagem de valores nulos por coluna
print(dados.isnull().sum())
```

◇ 4. Tratamento de Dados Faltantes

✔ Teoria:

Existem basicamente três formas de tratar dados faltantes:

Remover linhas ou colunas.

Preencher com valores (média, mediana, zero, texto, etc.).

Preencher com métodos estatísticos ou machine learning (mais avançado).

✔ Funções:

◇ dropna()

Remove linhas ou colunas com valores nulos.

Parâmetros principais:

axis=0 → remove linhas (padrão).

axis=1 → remove colunas.

how='any' → remove se tiver qualquer nulo (padrão).

how='all' → remove se tiver todos os valores nulos.

inplace=True → altera direto no dataframe.

◇ fillna(valor)
Preenche valores nulos com um valor específico.
Parâmetros principais:

value: valor que irá substituir os nulos (número, string, dicionário, etc.).

method: método de preenchimento (ffill → propaga para frente, bfill → para trás).

inplace=True: aplica direto no dataframe.

✔ Exemplos:

```
# Remover linhas com qualquer valor nulo
dados.dropna(inplace=True)

# Preencher nulos com zero
dados.fillna(0, inplace=True)

# Preencher nulos com a média da coluna 'salario'
dados['salario'].fillna(dados['salario'].mean(), inplace=True)

# Preencher usando o valor anterior (forward fill)
dados.fillna(method='ffill', inplace=True)
```

◇ 5. Filtragem de Dados
✔ Teoria:
Filtrar dados é selecionar subconjuntos que atendem a condições específicas, essencial para análises direcionadas.

✔ Sintaxe de Filtro:

dataframe[condição]

✔ Operadores Lógicos:

& → E (AND)

| → OU (OR)

~ → NÃO (NOT)

✔ Exemplos de Condições:

```
# Filtrar pessoas com idade maior que 30
filtro = dados[dados['idade'] > 30]

# Idade maior que 30 e salário acima de 5000
filtro = dados[(dados['idade'] > 30) & (dados['salario'] > 5000)]

# Selecionar onde a cidade NÃO é 'São Paulo'
filtro = dados[dados['cidade'] != 'São Paulo']
```

◇ Usando isin() para múltiplos valores:

```
# Selecionar quem é de SP ou RJ
dados[dados['cidade'].isin(['São Paulo', 'Rio de Janeiro'])]
```

◇ Filtrar valores nulos ou não nulos:

```
# Mostrar onde o salário é nulo
dados[dados['salario'].isnull()]

# Mostrar onde o salário NÃO é nulo
dados[dados['salario'].notnull()]
```

◇ Filtrar intervalos (between):

```
# Idade entre 20 e 40
dados[dados['idade'].between(20, 40)]
```

◇ 4. Detecção de Outliers

✔ Teoria:

Outliers são valores que estão muito fora do padrão dos dados e podem distorcer análises. Eles podem ser:

Erros de digitação (ex.: salário de 5000000)

Casos raros que precisam ser avaliados (ex.: clientes muito fora do perfil)

✔ Métodos comuns para identificar outliers:
◇ Usando Estatística – Desvio Padrão:
Valores fora de 3 desvios padrão costumam ser considerados outliers.

```
media = dados['salario'].mean()
desvio = dados['salario'].std()

# Selecionar salários fora de 3 desvios padrão
outliers = dados[(dados['salario'] > media + 3*desvio) |
                 (dados['salario'] < media - 3*desvio)]

print(outliers)
```

◇ Usando o Método do IQR (Interquartil):
Calcula o intervalo entre Q1 (25%) e Q3 (75%).

Outliers estão abaixo de Q1 - 1.5*IQR ou acima de Q3 + 1.5*IQR.

```
Q1 = dados['salario'].quantile(0.25)
Q3 = dados['salario'].quantile(0.75)
IQR = Q3 - Q1

# Filtro de outliers
outliers = dados[(dados['salario'] < Q1 - 1.5 * IQR) |
                 (dados['salario'] > Q3 + 1.5 * IQR)]

print(outliers)
```

◇ 5. Tratando Outliers
✔ O que fazer com eles:
Remover do dataset:

```
dados = dados[~((dados['salario'] < Q1 - 1.5 * IQR) |
                 (dados['salario'] > Q3 + 1.5 * IQR))]
```

Substituir por um valor mais adequado (média, mediana, limite do IQR).

```
# Substituir salários acima do limite por Q3 + 1.5*IQR
limite_superior = Q3 + 1.5 * IQR
dados.loc[dados['salario'] > limite_superior, 'salario'] = limite_superior
```

Ou analisar caso a caso (nem sempre outlier deve ser removido).

◇ 6. Seleção de Colunas e Linhas

✔ Teoria:

Selecionar colunas e linhas específicas permite focar em partes relevantes dos dados.

✔ Funções:

◇ Seleção de Colunas

```
# Uma coluna
dados['nome']

# Várias colunas
dados[['nome', 'idade']]
```

◇ Seleção por Índice (loc e iloc)

loc[]: seleção por rótulo (nome).

iloc[]: seleção por índice (posição numérica).

✔ Exemplos:

```
# Selecionar linhas com índice de 0 a 5 e coluna 'nome'
dados.loc[0:5, 'nome']

# Selecionar linhas de posição 0 a 5 e colunas nas posições 0 e 2
dados.iloc[0:5, [0, 2]]
```

◇ 7. Ordenação dos Dados

✔ Teoria:

Ordenar dados facilita análises, organização e leitura.

✔ Função:

sort_values(by, ascending=True)

Parâmetros:

by: coluna usada para ordenar.

ascending: True (crescente) ou False (decrescente).

inplace: se True, aplica diretamente no dataframe.

✔ Exemplos:

```
# Ordenar por salário crescente
dados.sort_values(by='salario', inplace=True)

# Ordenar por idade decrescente
dados.sort_values(by='idade', ascending=False, inplace=True)
```

◇ 8. Remoção de Dados Duplicados

✔ Teoria:

Registros duplicados podem distorcer análises.

✔ Funções:

duplicated() → retorna um booleano indicando linhas duplicadas.

drop_duplicates() → remove duplicatas.

✔ Parâmetros:

subset: colunas a considerar na verificação.

keep: 'first' (mantém o primeiro), 'last' (último) ou False (remove todos).

inplace: se True, altera diretamente.

✔ Exemplos:

```
# Verificar duplicados
print(dados.duplicated().sum())

# Remover duplicados
dados.drop_duplicates(inplace=True)

# Remover considerando apenas coluna 'cpf'
dados.drop_duplicates(subset='cpf', inplace=True)
```

◇ 9. Resetando o Índice

✔ Teoria:

Após remover linhas, os índices podem ficar desalinhados. Resetar o índice reorganiza para uma sequência contínua.

✔ Função:

reset_index()

Parâmetros:

drop=True → remove o índice antigo.

```
inplace=True → aplica no dataframe diretamente.

✔ Exemplo:

# Resetar índice após remoção de linhas
dados.reset_index(drop=True, inplace=True)
```