Aula 10 – Python para Visualização de Dados: Configurando o Ambiente

Seja bem-vindo(a) a esta jornada crucial no universo da visualização de dados com Python! Em um mundo onde somos bombardeados por informações a todo instante, a capacidade de transformar números brutos em histórias claras e impactantes é mais do que uma habilidade técnica; é uma forma de comunicação essencial. Imagine ter o poder de desvendar padrões ocultos, prever tendências e tomar decisões mais inteligentes, tudo isso a partir de dados que, à primeira vista, parecem apenas uma confusão de linhas e colunas.

Este é o ponto de partida para você se tornar um verdadeiro "contador de histórias com dados". Nesta aula, não apenas vamos configurar o seu ambiente de trabalho, mas também lançaremos as bases para que você possa manipular e preparar dados de forma eficiente, um passo indispensável antes de qualquer visualização. Afinal, um gráfico, por mais bonito que seja, só será útil se os dados por trás dele forem confiáveis e bem estruturados.

Ao final desta aula, você será capaz de configurar seu ambiente de desenvolvimento Python para análise de dados, entenderá os conceitos fundamentais da biblioteca Pandas, e estará apto a carregar, inspecionar e realizar as primeiras etapas de limpeza e preparação de DataFrames. Prepare-se para construir um alicerce sólido que sustentará todas as suas futuras explorações no fascinante campo da visualização de dados.

O Ponto de Partida: Anaconda e Jupyter Notebook

Anaconda

Plataforma completa que reúne Python, bibliotecas essenciais e ferramentas para ciência de dados em um único pacote

Jupyter Notebook

Laboratório interativo onde você escreve código, executa e visualiza resultados imediatamente

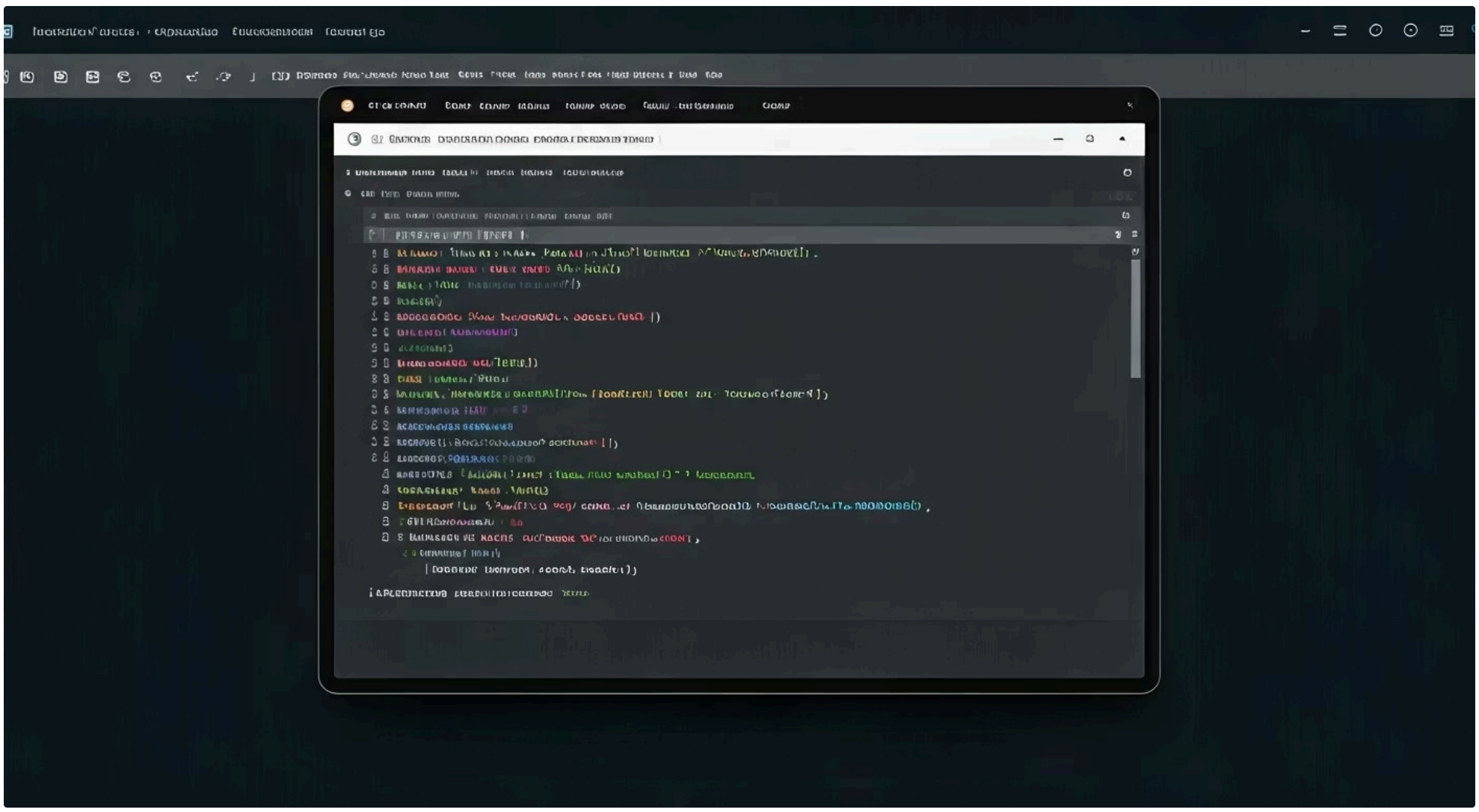
Iniciar uma jornada no mundo da programação e análise de dados pode parecer, à primeira vista, como entrar em uma loja de ferramentas gigantesca sem saber por onde começar. Há tantas opções, tantos softwares e configurações, que a tarefa de apenas "começar" já se torna um desafio. No entanto, para a visualização de dados com Python, existe um kit de ferramentas que simplifica enormemente esse processo, tornando a entrada muito mais suave e produtiva.

Esse kit é o Anaconda, uma plataforma de distribuição que reúne tudo o que você precisa para ciência de dados e aprendizado de máquina em um único pacote. Pense no Anaconda como uma "caixa de ferramentas completa" para o cientista de dados: ele já vem com Python, centenas de bibliotecas essenciais (como Pandas, NumPy, Matplotlib, que veremos em aulas futuras) e, o mais importante para nós agora, o Jupyter Notebook. O Jupyter Notebook, por sua vez, é o seu "laboratório interativo", um ambiente onde você pode escrever código, executar, ver os resultados imediatamente e até mesmo adicionar textos explicativos, tudo em um só lugar. É como ter um caderno digital que executa seu código e exibe gráficos na mesma página.

> **Vantagem Principal:** O Anaconda resolve o problema da compatibilidade e instalação de pacotes, que muitas vezes pode ser um calo para iniciantes. Com ele, você instala uma única vez e tem acesso a um ecossistema robusto e pronto para uso.

O Jupyter Notebook, por sua vez, permite uma abordagem exploratória e iterativa, ideal para a análise de dados, onde você frequentemente testa diferentes abordagens e visualiza resultados parciais antes de chegar à solução final.

Primeiros Passos com Jupyter Notebook



Agora que entendemos a importância do Anaconda como nosso ambiente e do Jupyter Notebook como nosso laboratório, é hora de colocar a mão na massa e dar os primeiros passos práticos. Abrir o Jupyter Notebook pela primeira vez é como entrar em um novo espaço de trabalho: você precisa se familiarizar com as ferramentas e a disposição do ambiente para tirar o máximo proveito dele.

01

Iniciar o Jupyter

Após instalar o Anaconda, abra o "Anaconda Navigator" e clique em "Launch" sob o ícone do Jupyter Notebook, ou digite `jupyter notebook` no terminal

02

Interface no Navegador

Uma nova aba abrirá no seu navegador web, mostrando os arquivos e pastas do seu diretório atual

03

Criar Novo Notebook

Clique em "New" no canto superior direito e selecione "Python 3" para criar um novo caderno

04

Trabalhar com Células

Use células de **Code** para código Python e **Markdown** para texto formatado

Um notebook Jupyter é composto por "células". Existem dois tipos principais de células: **Code** (onde você escreve e executa seu código Python) e **Markdown** (onde você escreve texto formatado, como este que você está lendo, usando uma linguagem simples de marcação). Para executar uma célula de código, basta clicar nela e pressionar **Shift + Enter**. Experimente digitar `print("Olá, Data Viz!")` em uma célula de código e execute-a. Você verá o resultado imediatamente abaixo da célula. Essa interatividade é o que torna o Jupyter tão poderoso para a exploração de dados, permitindo que você experimente, veja os resultados e ajuste seu código em tempo real, construindo sua análise passo a passo.

Pandas: O Alicerce da Manipulação de Dados

Imagine que você está prestes a construir uma casa. Você não começaria a pintar as paredes antes de ter uma fundação sólida, certo? Da mesma forma, no mundo da visualização de dados, antes de criar gráficos deslumbrantes, precisamos de uma fundação robusta para nossos dados.

Raramente os dados vêm prontos para serem visualizados; eles geralmente estão desorganizados, incompletos ou em formatos inadequados. É aqui que entra o Pandas, a biblioteca que se tornou o padrão ouro para manipulação e análise de dados em Python.

O Pandas é uma biblioteca de código aberto que oferece estruturas de dados de alto desempenho e fáceis de usar, projetadas para trabalhar com dados tabulares e séries temporais. Pense nele como uma versão superpoderosa de uma planilha eletrônica, mas com a flexibilidade e a capacidade de automação da programação. Com o Pandas, você pode carregar dados de diversas fontes (CSV, Excel, bancos de dados), limpá-los, transformá-los e prepará-los para a análise e visualização, tudo com poucas linhas de código.

Estruturas Principais do Pandas:

- **Series:** Uma única coluna de dados com índice associado
- **DataFrame:** Uma tabela bidimensional completa, como uma planilha

A essência do Pandas reside em suas duas estruturas de dados principais: a **Series** e o **DataFrame**. Compreender como elas funcionam é o primeiro passo para dominar a manipulação de dados. Uma Series é como uma única coluna de uma planilha ou uma lista de valores, mas com um índice associado a cada valor. Já o DataFrame é a estrela do show, representando uma tabela bidimensional, como uma planilha completa, onde cada coluna é uma Series e todas compartilham o mesmo índice. Essas estruturas são a base para quase todas as operações de dados que você fará, desde a simples seleção de colunas até transformações complexas.

Desvendando as Estruturas: Series e DataFrame

Para realmente aproveitar o poder do Pandas, é fundamental entender suas estruturas de dados básicas: a Series e o DataFrame. Elas são os blocos de construção com os quais você irá interagir e manipular seus dados. Pense na Series como uma lista etiquetada, e no DataFrame como uma coleção dessas listas, todas alinhadas para formar uma tabela.

Series

Array unidimensional capaz de armazenar qualquer tipo de dado (inteiros, strings, floats, objetos Python, etc.)

Cada elemento possui um rótulo (índice) para acesso intuitivo

DataFrame

Tabela bidimensional, mutável em tamanho, com colunas e linhas rotuladas

Coleção de objetos Series que compartilham o mesmo índice de linha

Uma **Series** no Pandas é um array unidimensional capaz de armazenar qualquer tipo de dado (inteiros, strings, floats, objetos Python, etc.). Ela é muito parecida com uma coluna em uma planilha ou um vetor em linguagens como R, mas com um diferencial importante: cada elemento possui um rótulo (índice). Por exemplo, se você tem uma lista de temperaturas diárias, cada temperatura pode ser um elemento da Series, e o dia correspondente pode ser seu índice. Isso permite um acesso e manipulação mais intuitivos.

Já o **DataFrame** é a estrutura de dados mais utilizada no Pandas. Ele é uma tabela bidimensional, mutável em tamanho, com colunas e linhas rotuladas. Você pode imaginá-lo como uma coleção de objetos Series, onde cada Series representa uma coluna do DataFrame, e todas as Series compartilham o mesmo índice de linha. Se a Series era uma coluna da sua planilha, o DataFrame é a planilha inteira. Com ele, você pode representar conjuntos de dados complexos, como informações de clientes (nome, idade, cidade), dados de vendas (produto, quantidade, preço) ou resultados de pesquisas.

Vamos ver um exemplo prático para solidificar esses conceitos:

```
import pandas as pd

# Criando uma Series
temperaturas = pd.Series([22, 25, 19, 28], index=['Seg', 'Ter', 'Qua', 'Qui'])
print("Exemplo de Series:\n", temperaturas)

# Criando um DataFrame
dados_vendas = {
    'Produto': ['Camisa', 'Calça', 'Meia', 'Sapato'],
    'Quantidade': [100, 50, 200, 30],
    'Preco_Unitario': [50.00, 120.00, 15.00, 250.00]
}
df_vendas = pd.DataFrame(dados_vendas)
print("\nExemplo de DataFrame:\n", df_vendas)
```

Neste exemplo, `temperaturas` é uma Series, e `df_vendas` é um DataFrame. Perceba como o DataFrame organiza os dados de forma tabular, facilitando a visualização e manipulação de conjuntos de dados mais complexos.

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
Series	Dados unidimensionais, como uma coluna de dados	Array NumPy + Índice	Lista de temperaturas diárias com rótulos de dias
DataFrame	Dados bidimensionais, como uma tabela ou planilha	Coleção de Series com índice comum	Tabela de vendas com colunas para Produto, Quantidade, Preço

Carregando Dados: Trazendo o Mundo para o Python

A jornada da visualização de dados quase sempre começa com a importação de informações de alguma fonte externa. Afinal, seus dados raramente nascem dentro do seu script Python. Eles podem estar em arquivos CSV, planilhas Excel, bancos de dados, APIs web, entre outros. A capacidade de carregar esses dados de forma eficiente é o primeiro passo prático para qualquer análise.

pd.read_csv()

Para arquivos de texto delimitados por vírgulas ou outros separadores

pd.read_excel()

Para planilhas do Microsoft Excel

Funções de Inspeção

.head(), .info(), .describe() para entender estrutura e conteúdo

O Pandas brilha intensamente nesse aspecto, oferecendo funções intuitivas para ler dados de uma vasta gama de formatos. As mais comuns são `pd.read_csv()` para arquivos de texto delimitados por vírgulas (ou outros separadores) e `pd.read_excel()` para planilhas do Microsoft Excel. Essas funções são incrivelmente versáteis, permitindo que você especifique o separador, o cabeçalho, os tipos de dados, e até mesmo pule linhas ou selecione colunas específicas durante a importação. É como ter um assistente que organiza automaticamente seus documentos ao invés de você ter que digitá-los manualmente.

Depois de carregar os dados para um DataFrame, é crucial fazer uma inspeção inicial para entender sua estrutura e conteúdo. Funções como `.head()` (para ver as primeiras linhas), `.info()` (para um resumo dos tipos de dados e valores não nulos) e `.describe()` (para estatísticas descritivas básicas) são suas melhores amigas nesse estágio. Elas fornecem um panorama rápido que ajuda a identificar problemas potenciais, como valores ausentes ou tipos de dados incorretos, antes mesmo de iniciar a limpeza.

```
import pandas as pd

# Exemplo: Carregando um arquivo CSV (assumindo que 'dados_exemplo.csv' existe)
# Se não tiver um arquivo, crie um com algumas colunas e linhas para testar.
# Ex: nome,idade,cidade
# Ana,25,São Paulo
# Bruno,30,Rio de Janeiro

try:
 df = pd.read_csv('dados_exemplo.csv')
 print("DataFrame carregado com sucesso:")
 print(df.head()) # Exibe as 5 primeiras linhas
 print("\nInformações do DataFrame:")
 df.info() # Exibe um resumo das colunas e tipos de dados
 print("\nEstatísticas descritivas:")
 print(df.describe()) # Exibe estatísticas para colunas numéricas
except FileNotFoundError:
 print("Arquivo 'dados_exemplo.csv' não encontrado. Por favor, crie um para testar.")
```

Limpeza e Preparação de Dados: O Segredo da Boa Visualização

> **"Lixo que entra, lixo que sai"**
>
> Por mais sofisticada que seja sua técnica de visualização, se os dados subjacentes estiverem sujos, seus gráficos serão enganosos.

Você já ouviu a expressão "lixo que entra, lixo que sai"? No mundo dos dados, isso é uma verdade absoluta. Por mais sofisticada que seja sua técnica de visualização, se os dados subjacentes estiverem sujos, incompletos ou inconsistentes, seus gráficos não apenas serão enganosos, mas também podem levar a conclusões erradas. A limpeza e preparação de dados é, sem dúvida, a etapa mais demorada e crucial de qualquer projeto de análise de dados. É como preparar os ingredientes para um prato gourmet: você precisa lavar, cortar, descascar e temperar tudo antes de cozinhar.

Valores Ausentes (NaN)

- Identificar com `.isnull().sum()`
- Remover com `.dropna()`
- Preencher com `.fillna()`

Valores Duplicados

- Detectar com `.duplicated()`
- Remover com `.drop_duplicates()`

Tipos de Dados

- Converter com `.astype()`
- Garantir tipos corretos para cálculos

O Pandas oferece um arsenal de ferramentas para lidar com os desafios comuns da limpeza de dados. Um dos problemas mais frequentes são os **valores ausentes** (NaN - Not a Number), que podem surgir por diversas razões (dados não coletados, erros de entrada, etc.). Você pode identificá-los com `.isnull().sum()`, removê-los com `.dropna()` (se tiver poucos) ou preenchê-los com valores lógicos usando `.fillna()` (média, mediana, valor constante). Outro desafio são os **valores duplicados**, que podem distorcer suas análises; `.duplicated()` e `.drop_duplicates()` são suas ferramentas para isso.

Além disso, muitas vezes os dados são carregados com o tipo errado (por exemplo, números como texto), o que impede cálculos ou visualizações adequadas. A função `.astype()` permite converter colunas para o tipo de dado correto (numérico, categórico, data). A preparação de dados também envolve a criação de novas colunas a partir de existentes, renomear colunas para maior clareza ou reformatar dados para um padrão consistente. Dominar essas técnicas não é apenas sobre corrigir erros, mas sobre moldar seus dados para que eles possam contar a história que você deseja, de forma precisa e convincente.

```
import pandas as pd
import numpy as np

# Criando um DataFrame de exemplo com dados sujos
dados_sujos = {
 'ID': [1, 2, 3, 4, 5, 6],
 'Nome': ['Alice', 'Bob', 'Charlie', 'Alice', 'David', 'Eve'],
 'Idade': [25, 30, np.nan, 25, 40, 35],
 'Cidade': ['São Paulo', 'Rio', 'Belo Horizonte', 'São Paulo', np.nan, 'Curitiba'],
 'Renda': ['5000', '7500', '6000', '5000', '8000', '4500']
}
df_sujo = pd.DataFrame(dados_sujos)
print("DataFrame Original:\n", df_sujo)

# 1. Verificando valores ausentes
print("\nValores ausentes por coluna:\n", df_sujo.isnull().sum())

# 2. Preenchendo valores ausentes na coluna 'Idade' com a média
df_sujo['Idade'].fillna(df_sujo['Idade'].mean(), inplace=True)

# 3. Preenchendo valores ausentes na coluna 'Cidade' com 'Desconhecida'
df_sujo['Cidade'].fillna('Desconhecida', inplace=True)

# 4. Verificando e removendo duplicatas
print("\nRegistros duplicados:\n", df_sujo.duplicated().sum())
df_limpo = df_sujo.drop_duplicates()

# 5. Convertendo tipo de dado da coluna 'Renda' para numérico
df_limpo['Renda'] = pd.to_numeric(df_limpo['Renda'])

print("\nDataFrame Após Limpeza e Preparação:\n", df_limpo)
print("\nTipos de dados após conversão:\n", df_limpo.info())
```

Manipulando DataFrames: Seleção e Filtragem

Com seus dados carregados e as primeiras etapas de limpeza realizadas, o próximo passo é começar a explorar e focar nas informações que realmente importam para sua análise. Um DataFrame, especialmente quando grande, pode conter uma vasta quantidade de dados, e nem tudo é relevante para cada pergunta que você quer responder. É aqui que as operações de seleção e filtragem se tornam ferramentas poderosas, permitindo que você extraia subconjuntos específicos de dados, como um escultor que remove o excesso de material para revelar a forma desejada.

Seleção de Colunas

Escolha atributos específicos para simplificar a análise

Filtragem de Linhas

Selecione linhas com base em condições específicas

Foco na História

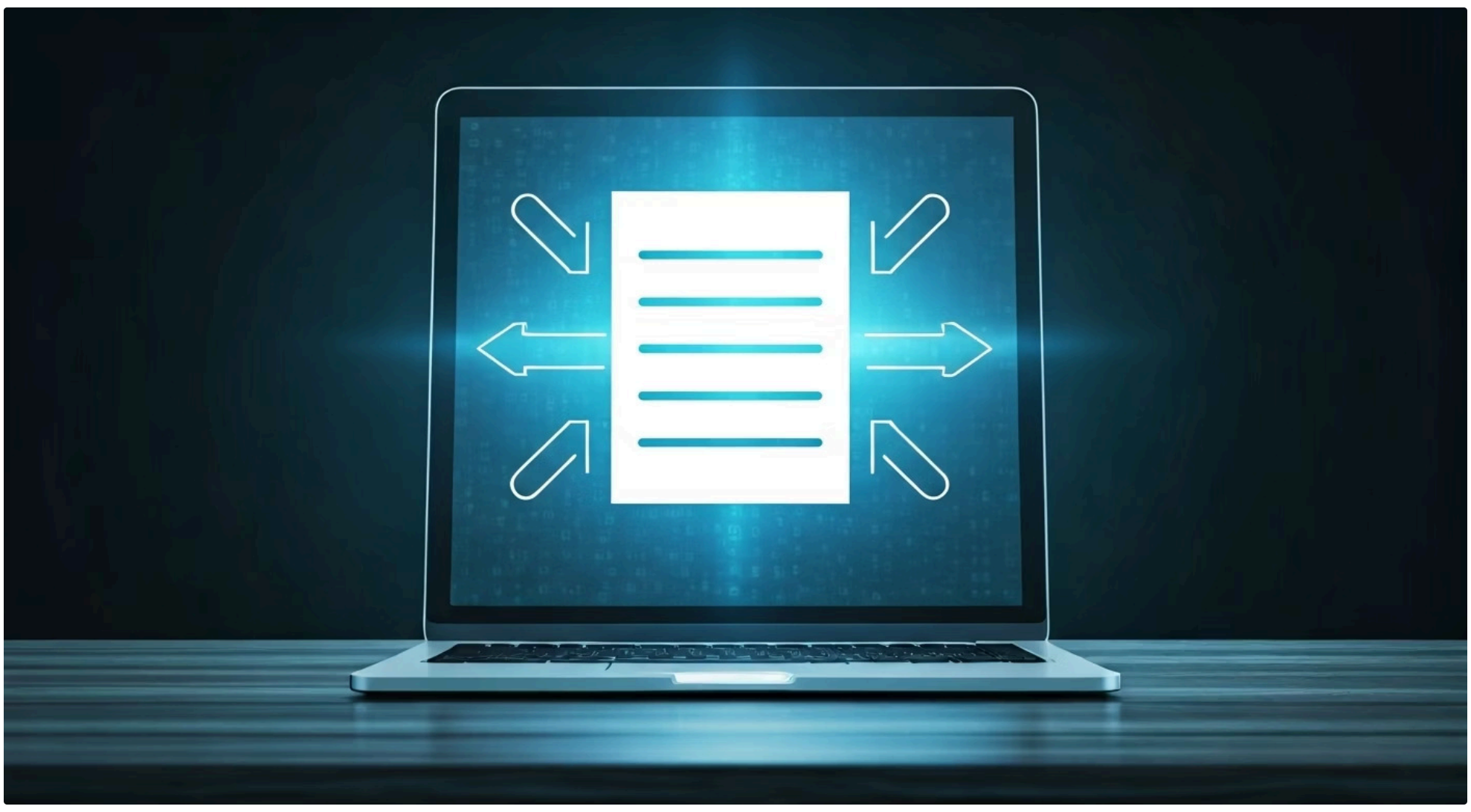
Isole segmentos para investigações detalhadas

A **seleção de colunas** é uma das operações mais básicas e frequentes. Você pode selecionar uma única coluna, que resultará em uma Series, ou múltiplas colunas, que retornarão um novo DataFrame. Pense nisso como escolher quais informações você quer ver em uma planilha: apenas a coluna de nomes, ou as colunas de nomes e idades. Essa capacidade de focar em atributos específicos é crucial para simplificar a análise e a visualização, evitando a sobrecarga de informações.

A **filtragem de linhas**, por sua vez, permite que você selecione linhas com base em condições específicas. Quer ver apenas os dados de vendas de um determinado produto? Ou talvez apenas os clientes com mais de 30 anos? A filtragem usa expressões booleanas (verdadeiro/falso) para decidir quais linhas manter. Essa técnica é fundamental para aprofundar sua análise, permitindo que você isole segmentos de dados para investigações mais detalhadas ou para criar visualizações que contem uma história muito específica, alinhada com os princípios do Data Storytelling.

```
import pandas as pd

# Criando um DataFrame de exemplo
dados_clientes = {
 'Nome': ['Ana', 'Bruno', 'Carla', 'Daniel', 'Eva', 'Fábio'],
 'Idade': [28, 35, 22, 40, 30, 29],
 'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte', 'São Paulo', 'Curitiba', 'Rio de Janeiro'],
 'Renda_Mensal': [4500, 7000, 3000, 9000, 5500, 6000]
}
df_clientes = pd.DataFrame(dados_clientes)
print("DataFrame Original de Clientes:\n", df_clientes)

# 1. Selecionando uma única coluna (resulta em uma Series)
nomes = df_clientes['Nome']
print("\nColuna 'Nome':\n", nomes)

# 2. Selecionando múltiplas colunas (resulta em um DataFrame)
nomes_idades = df_clientes[['Nome', 'Idade']]
print("\nColunas 'Nome' e 'Idade':\n", nomes_idades)

# 3. Filtrando clientes com idade superior a 30 anos
clientes_maiores_30 = df_clientes[df_clientes['Idade'] > 30]
print("\nClientes com mais de 30 anos:\n", clientes_maiores_30)

# 4. Filtrando clientes da cidade de 'São Paulo'
clientes_sp = df_clientes[df_clientes['Cidade'] == 'São Paulo']
print("\nClientes de São Paulo:\n", clientes_sp)

# 5. Filtrando clientes com idade > 30 E de São Paulo (condições múltiplas)
clientes_sp_maiores_30 = df_clientes[(df_clientes['Idade'] > 30) & (df_clientes['Cidade'] == 'São Paulo')]
print("\nClientes de São Paulo com mais de 30 anos:\n", clientes_sp_maiores_30)
```

Exportando Dados Tratados: Compartilhando Seus Insights



Após todo o trabalho de carregar, limpar, preparar e manipular seus dados, a etapa final antes de seguir para a visualização avançada é garantir que seus esforços sejam salvos e possam ser compartilhados. Imagine passar horas refinando um conjunto de dados complexo, apenas para perder todo o progresso porque você não o salvou. Ou, ainda, precisar compartilhar essa versão "limpa" com um colega que usará outra ferramenta de análise ou visualização. A capacidade de exportar seus DataFrames tratados é tão importante quanto a de importá-los.

df.to_csv()

Exporta para arquivos CSV

`df.to_csv('arquivo.csv', index=False)`

df.to_excel()

Exporta para planilhas Excel

`df.to_excel('arquivo.xlsx', index=False)`

Dica Importante: Use sempre `index=False` ao exportar para evitar que o índice do DataFrame seja salvo como uma coluna adicional no arquivo.

O Pandas oferece funções simples e eficazes para exportar DataFrames para diversos formatos, sendo os mais comuns `df.to_csv()` para arquivos CSV e `df.to_excel()` para planilhas Excel. Essas funções são o inverso das funções `read_csv()` e `read_excel()`, permitindo que você salve seu DataFrame em um arquivo no seu sistema. Um parâmetro crucial a ser lembrado é `index=False` ao salvar em CSV ou Excel. Por padrão, o Pandas inclui o índice do DataFrame como uma coluna no arquivo exportado, o que geralmente não é desejado e pode causar problemas na reimportação ou em outras ferramentas.

Exportar dados tratados é um passo fundamental para a colaboração, para a criação de backups de suas análises e para a integração com outras ferramentas de Business Intelligence, como Tableau ou Power BI, que podem ser usadas para criar dashboards dinâmicos. Ao salvar a versão limpa e preparada dos seus dados, você garante que a base para suas visualizações e narrativas de dados seja consistente e de alta qualidade, permitindo que você e sua equipe construam insights confiáveis a partir de uma fonte comum.

```
import pandas as pd
import numpy as np

# Criando um DataFrame de exemplo (simulando dados já limpos)
df_final = pd.DataFrame({
 'ID_Cliente': [101, 102, 103, 104],
 'Nome': ['João', 'Maria', 'Pedro', 'Ana'],
 'Idade': [30, 25, 40, 33],
 'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte', 'São Paulo'],
 'Renda_Anual': [55000, 48000, 72000, 60000]
})

print("DataFrame final a ser exportado:\n", df_final)

# Exportando para CSV
nome_arquivo_csv = 'clientes_tratados.csv'
df_final.to_csv(nome_arquivo_csv, index=False)
print(f"\nDataFrame exportado para '{nome_arquivo_csv}' (CSV) com sucesso!")

# Exportando para Excel
nome_arquivo_excel = 'clientes_tratados.xlsx'
df_final.to_excel(nome_arquivo_excel, index=False)
print(f"DataFrame exportado para '{nome_arquivo_excel}' (Excel) com sucesso!")

# Para verificar, você pode tentar carregar o arquivo novamente:
df_verificacao_csv = pd.read_csv(nome_arquivo_csv)
print(f"\nVerificação: Conteúdo de '{nome_arquivo_csv}':\n", df_verificacao_csv.head())
```

Consolidação e Próximos Passos

Chegamos ao final desta aula introdutória, mas fundamental, sobre a configuração do ambiente e os primeiros passos na manipulação de dados com Python e Pandas. Percorremos desde a instalação do Anaconda e a exploração do Jupyter Notebook, nosso laboratório interativo, até a compreensão das estruturas de dados Series e DataFrame. Aprendemos a carregar dados de diversas fontes, a realizar as primeiras e cruciais etapas de limpeza e preparação, e a selecionar e filtrar informações para focar em nossos objetivos de análise.

Em prática, você agora tem as ferramentas para:

- [ ] **Montar seu próprio ambiente de trabalho para ciência de dados**
- [ ] **Interagir com o Jupyter Notebook para escrever e executar código Python**
- [ ] **Entender como o Pandas organiza os dados em Series e DataFrames**
- [ ] **Importar dados para o Python e realizar uma inspeção inicial**
- [ ] **Lidar com valores ausentes e duplicados, e corrigir tipos de dados**
- [ ] **Extrair subconjuntos de dados relevantes através de seleção e filtragem**
- [ ] **Exportar seus dados tratados para uso futuro ou compartilhamento**

> **Este é o alicerce sobre o qual construiremos narrativas visuais impactantes.** A qualidade da sua visualização começa com a qualidade da sua preparação de dados.

Na próxima aula, daremos um salto para o mundo da visualização propriamente dita, explorando o Matplotlib, a biblioteca base para a criação de gráficos em Python. Prepare-se para transformar esses dados limpos e organizados em representações visuais que contam histórias.

Próxima Aula

Aula 11 – Matplotlib: A Base da Visualização em Python

Recursos Adicionais

- **Documentação Oficial do Pandas:** Para aprofundar em todas as funções e parâmetros
- **Guia de Instalação do Anaconda:** Para referência caso precise reinstalar ou configurar em outro ambiente
- **Tutoriais de Jupyter Notebook:** Para explorar funcionalidades avançadas do ambiente interativo

Autoavaliação

1

Qual das seguintes ferramentas é considerada um ambiente de desenvolvimento interativo, ideal para escrever código, executar e visualizar resultados imediatamente, incluindo gráficos?

1. Microsoft Word
2. Anaconda Navigator
3. Jupyter Notebook
4. Bloco de Notas

2

No Pandas, qual estrutura de dados é mais adequada para representar uma tabela bidimensional, com linhas e colunas rotuladas, similar a uma planilha?

1. Series
2. List
3. Dictionary
4. DataFrame

3

Qual função do Pandas é comumente utilizada para carregar dados de um arquivo de texto delimitado por vírgulas (CSV) para um DataFrame?

1. pd.load_csv()
2. pd.read_excel()
3. pd.import_data()
4. pd.read_csv()

4

Ao exportar um DataFrame para um arquivo CSV usando df.to_csv(), qual parâmetro é frequentemente usado para evitar que o índice do DataFrame seja salvo como uma coluna no arquivo?

1. header=False
2. index=False
3. columns=False
4. data=False

5

Explique a importância da etapa de limpeza e preparação de dados antes de iniciar a visualização.

(Questão dissertativa - espaço para resposta)

Gabarito:

1

Resposta

c) Jupyter Notebook

2

Resposta

d) DataFrame

3

Resposta

d) pd.read_csv()

4

Resposta

b) index=False