PYTHON PARA ANÁLISE DE DADOS

Sobre o Curso:

Este curso oferece uma abordagem abrangente para os fundamentos da programação em Python, focando especificamente na sua aplicação em análise de dados. Destinado a pessoas que já possuem algum conhecimento prévio em Python, o programa é ideal para aqueles que desejam aprofundar suas habilidades de programação e aplicá-las na análise e manipulação de dados. Ao longo de uma série de módulos estruturados de forma progressiva, os participantes serão guiados desde os conceitos básicos até técnicas avançadas, culminando em uma compreensão sólida da linguagem Python no contexto da análise de dados.

APRESENTAÇÃO

O MATERIAL

Este conteúdo foi elaborado considerando o seu processo de aprendizado. Aqui, você terá acesso a informações cruciais que poderão servir como guia para criar projetos incríveis utilizando Python. Python é uma linguagem poderosa, contemporânea e amplamente empregada em aplicações e análise de dados. Portanto, adquirir conhecimento sobre seu funcionamento pode ser exatamente o que você está procurando.

O CURSO

Neste curso, você mergulhará no mundo da análise de dados utilizando Python, explorando a criação de algoritmos que processam conjuntos de dados do mundo real. Com ênfase na aplicação prática, você dominará os conceitos fundamentais da linguagem Python enquanto trabalha em projetos de análise de dados. Ao longo do curso, você utilizará recursos como manipulação de dados, visualização de dados, aplicação de bibliotecas especializadas e automação de tarefas para resolver problemas e extrair insights valiosos dos dados. Este curso é ideal para aqueles que desejam desenvolver habilidades sólidas em Python voltadas para a análise e interpretação de dados.

APRENDIZAGEM

O curso é completamente orientado para a aplicação prática. Ao longo dele, você estará envolvido na criação de algoritmos e sistemas que reforçarão sua compreensão dos conceitos apresentados durante as aulas.

OBJETIVO DO CURSO

O curso de Análise de Dados com Python tem como objetivo capacitar os participantes a dominarem as habilidades essenciais para explorar, analisar e visualizar dados de forma eficiente utilizando a linguagem de programação Python e suas principais bibliotecas especializadas, como Pandas, NumPy, Matplotlib, Seaborn, Dash e Plotly.

Durante o curso, os alunos serão introduzidos aos conceitos fundamentais da análise de dados, desde a formulação de perguntas até a interpretação e comunicação dos resultados. Eles aprenderão sobre o ciclo de uma análise de dados, que inclui aquisição, limpeza, exploração, modelagem e visualização de dados.

Uma parte significativa do curso será dedicada ao aprendizado prático do uso das bibliotecas Python mencionadas. Os participantes aprenderão a manipular e transformar conjuntos de dados complexos, realizar análises estatísticas, criar visualizações gráficas informativas e construir dashboards interativos para apresentar suas descobertas de forma clara e impactante.

Além disso, o curso abordará técnicas avançadas de análise de dados, como o tratamento de datas e horas, combinação de dados de diferentes fontes e a aplicação de modelos de machine learning para insights preditivos.

Ao final do curso, os alunos estarão aptos a aplicar seus conhecimentos em uma variedade de contextos, desde análises exploratórias simples até projetos complexos de análise de dados, capacitando-os a tomar decisões informadas e estratégicas com base em evidências quantitativas.

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

O INÍCIO

No contexto atual, onde a geração massiva de dados permeia todos os aspectos da vida moderna, a análise de dados emerge como uma prática crucial para extrair valor e insights significativos a partir desse mar de informações. Trata-se de um processo multifacetado que abarca desde a coleta até a interpretação dos dados, utilizando ferramentas estatísticas e computacionais para descobrir padrões, tendências e relações ocultas.

A análise de dados não se restringe apenas ao domínio empresarial, mas permeia todas as esferas da sociedade, desde a tomada de decisões estratégicas em corporações até a pesquisa científica, passando por aplicações em saúde, educação, finanças e muitos outros campos. Neste cenário, a habilidade de conduzir uma análise de dados eficaz tornou-se uma competência fundamental para profissionais em diversos setores, impulsionando a demanda por ferramentas e técnicas que facilitem esse processo e, neste contexto, o Python tem se destacado como uma linguagem de programação versátil e poderosa, oferecendo um ecossistema robusto de bibliotecas especializadas em análise, manipulação e visualização de dados. Assim, compreender os fundamentos da análise de dados e dominar as ferramentas disponíveis, como o Python, torna-se não apenas uma vantagem competitiva, mas uma necessidade premente em um mundo cada vez mais orientado por dados.

CAPÍTULO 2

O QUE É ANÁLISE DE DADOS?

VISÃO GERAL

Análise de dados é o processo de examinar, limpar, transformar e modelar conjuntos de dados com o objetivo de descobrir informações úteis, identificar padrões, tendências e relações, e tomar decisões embasadas em evidências. Essa prática envolve uma série de etapas, desde a coleta inicial dos dados até a interpretação dos resultados.

No cerne da análise de dados está a capacidade de extrair insights significativos que podem fornecer uma compreensão mais profunda de problemas, fenômenos ou oportunidades. Isso pode ser realizado por meio de técnicas estatísticas, algoritmos de aprendizado de máquina, visualização de dados e outras abordagens analíticas.

A análise de dados é aplicada em uma ampla variedade de campos, incluindo negócios, ciência, saúde, finanças, marketing, governo e muito mais. Seja para prever o comportamento do mercado, otimizar processos empresariais, entender padrões de doenças em uma população ou identificar fraudes, a análise de dados desempenha um papel fundamental na tomada de decisões informadas e na resolução de problemas complexos.

CICLO DE UMA ANÁLISE DE DADOS

O ciclo de um projeto de análise de dados é composto por várias etapas interligadas, que geralmente seguem uma sequência lógica. Embora possa haver variações dependendo do contexto e dos requisitos específicos do projeto, as principais etapas incluem:

FORMULAÇÃO DE PERGUNTAS:

- Identificação dos objetivos do projeto e das questões que a análise de dados deve abordar.
- Definição de métricas de sucesso para avaliar os resultados.

AQUISIÇÃO DE DADOS:

- Coleta dos dados relevantes para o projeto, seja de fontes internas ou externas.
- Isso pode envolver a integração de dados de bancos de dados, arquivos locais, APIs ou outras fontes.

LIMPEZA E PREPARAÇÃO DE DADOS:

- Verificação e limpeza dos dados para corrigir erros, remover valores ausentes ou inconsistentes e garantir a qualidade dos dados.
- Transformação dos dados em um formato adequado para análise, incluindo normalização, agregação e codificação de variáveis.

ANÁLISE EXPLORATÓRIA DE DADOS (EDA):

- Exploração inicial dos dados por meio de estatísticas descritivas, visualizações e técnicas de mineração de dados.
- Identificação de padrões, tendências e insights preliminares que possam guiar análises mais avançadas.

MODELAGEM DE DADOS:

- Desenvolvimento e aplicação de modelos estatísticos, algoritmos de aprendizado de máquina ou outras técnicas analíticas para extrair insights ou fazer previsões com base nos dados.
- Avaliação e validação dos modelos para garantir sua eficácia e generalização.

INTERPRETAÇÃO E COMUNICAÇÃO:

- Interpretação dos resultados da análise e sua relevância em relação aos objetivos do projeto.
- Comunicação dos insights de forma clara e eficaz para as partes interessadas, por meio de relatórios, apresentações ou visualizações interativas.

ITERAÇÃO E MELHORIA:

- Revisão do processo de análise com base no feedback recebido e nos novos requisitos identificados.
- Iteração nas etapas anteriores para refinar os modelos, aprimorar os resultados e abordar novas perguntas ou desafios.

Essas etapas geralmente formam um ciclo iterativo, onde a análise de dados é um processo contínuo e dinâmico, sujeito a refinamentos e ajustes à medida que novas informações são obtidas e novos insights são descobertos.

CAPÍTULO 3

APRESENTANDO O PYTHON PARA ANÁLISE DE DADOS

Python é uma linguagem de programação de alto nível, conhecida por sua versatilidade, sintaxe simples e extensa comunidade de desenvolvedores. Sua popularidade crescente no campo da análise de dados é atribuída a diversas razões. Primeiramente, Python oferece uma ampla gama de bibliotecas especializadas que simplificam e agilizam o processo de análise de dados. Entre essas bibliotecas, destacam-se o Pandas, que fornece estruturas de dados poderosas e ferramentas para manipulação e análise de conjuntos de dados tabulares, e o NumPy, que oferece suporte para operações matemáticas e numéricas eficientes, essenciais para análises estatísticas e modelagem de dados.

Além disso, a linguagem Python é conhecida por sua legibilidade e expressividade, o que a torna ideal para desenvolvimento rápido de protótipos e análises exploratórias. Sua sintaxe clara e concisa permite que os analistas de dados se concentrem mais nos problemas em questão e menos na complexidade da linguagem de programação.

Outra vantagem significativa do Python é sua capacidade de integração com outras tecnologias e ferramentas, incluindo bancos de dados, sistemas de big data e frameworks de machine learning. Isso permite que os profissionais de análise de dados construam pipelines de dados completos, desde a aquisição e limpeza dos dados até a implementação de modelos preditivos avançados.

Além das bibliotecas mencionadas, Python oferece uma variedade de opções para visualização de dados. O Matplotlib e o Seaborn são duas bibliotecas populares para criação de gráficos estáticos e estatísticos, enquanto o Plotly e o Dash são amplamente utilizados para construir dashboards interativos e aplicativos web de análise de dados, proporcionando uma maneira dinâmica de explorar e compartilhar insights.

Em resumo, o uso de Python na análise de dados é impulsionado por sua facilidade de uso, vasta gama de bibliotecas especializadas, capacidade de integração com outras tecnologias e ferramentas, e opções avançadas de visualização. Este curso visa capacitar os participantes a dominar essas ferramentas e técnicas, permitindo-lhes extrair informações valiosas a partir de dados complexos e tomar decisões fundamentadas com base em evidências quantitativas.

SINTAXE CLARA E CONCISA:

A linguagem de programação Python é amplamente elogiada por sua sintaxe simples e legível, tornando-a uma escolha popular para desenvolvedores de todos os níveis de experiência. Sua clareza e concisão não apenas facilitam a escrita de código, mas também simplificam a compreensão do código por outros desenvolvedores, facilitando a colaboração em projetos de software.

Essa simplicidade sintática faz com que Python seja uma excelente opção para iniciantes em programação, pois reduz a curva de aprendizado e permite que os novos programadores se concentrem mais nos conceitos fundamentais de programação do que na complexidade da linguagem em si. Além disso, a vasta quantidade de recursos educacionais disponíveis para aprender Python, como tutoriais online, documentação abrangente e comunidades de desenvolvedores ativas, contribui para sua acessibilidade e popularidade entre os iniciantes.

No entanto, a simplicidade do Python não significa uma falta de poder ou capacidade. Ao contrário, Python é uma linguagem extremamente versátil, capaz de lidar com uma ampla variedade de tarefas, desde scripts simples até aplicativos web complexos e análises de dados avançadas. Sua grande quantidade de bibliotecas especializadas, como Pandas para análise de dados, Django para desenvolvimento web e TensorFlow para aprendizado de máquina, expande significativamente suas capacidades e o torna uma escolha viável para uma variedade de aplicações.

Além disso, Python é uma linguagem altamente portátil, executando-se em praticamente qualquer sistema operacional, o que facilita sua adoção em uma variedade de ambientes de desenvolvimento. Sua compatibilidade com outras linguagens de programação, como C e C++, permite a integração de módulos de alto desempenho para otimização de desempenho quando necessário.

Em resumo, a combinação de simplicidade, poder e portabilidade faz de Python uma escolha ideal para uma ampla gama de projetos de desenvolvimento de software, tornando-a uma linguagem acessível e eficaz para desenvolvedores de todos os níveis de habilidade.

CAPÍTULO 4

ECOSSISTEMA DE BIBLIOTECAS:

Python possui um ecossistema rico de bibliotecas especializadas em análise de dados, incluindo Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, TensorFlow e muitas outras. Essas bibliotecas oferecem uma ampla gama de funcionalidades para manipulação, visualização e modelagem de dados.

INTEGRAÇÃO COM OUTRAS LINGUAGENS E FERRAMENTAS:

Python pode ser facilmente integrado com outras linguagens de programação, como C, C ++ e Java, bem como com ferramentas populares de análise de dados, como SQL, Spark e Hadoop.

COMUNIDADE ATIVA:

Python possui uma comunidade globalmente ativa de desenvolvedores, cientistas de dados e entusiastas que contribuem com bibliotecas, tutoriais e recursos educacionais, facilitando o aprendizado e a resolução de problemas.

PLATAFORMAS DE DESENVOLVIMENTO AMIGÁVEIS:

Python é suportado por uma variedade de ambientes de desenvolvimento integrado (IDEs) e ambientes de computação científica,

como Jupyter Notebook, Spyder e Google Colab, que oferecem ferramentas poderosas para análise interativa de dados e visualização.

FLEXIBILIDADE E ESCALABILIDADE:

Python é uma linguagem flexível que pode ser usada para uma ampla variedade de tarefas, desde análises simples até projetos de aprendizado de máquina e ciência de dados em grande escala.

Em resumo, Python se destaca como uma escolha popular para análise de dados devido à sua simplicidade, eficácia e ao vasto ecossistema de ferramentas e recursos disponíveis, tornando-o uma opção preferencial para profissionais e organizações que buscam extrair insights valiosos de seus dados.

PANDAS

Pandas, uma biblioteca de software em Python, é amplamente reconhecida por sua eficiência e versatilidade na manipulação e análise de dados. Ela oferece uma variedade de estruturas de dados poderosas, como o DataFrame, que permite aos usuários organizar e manipular conjuntos de dados tabulares de forma intuitiva e eficaz.

Além disso, o Pandas fornece uma ampla gama de ferramentas para lidar com tarefas comuns de análise de dados, incluindo limpeza, transformação, agregação e visualização de dados. Sua capacidade de realizar operações complexas em grandes conjuntos de dados de maneira rápida e eficiente torna-o uma escolha popular entre profissionais em campos como ciência de dados, engenharia de dados, análise financeira, pesquisa acadêmica e muitos outros.

Uma das principais vantagens do Pandas é sua integração perfeita com outras bibliotecas Python populares, como NumPy, Matplotlib e Scikit-learn, o que permite aos usuários criar pipelines de análise de dados completos, desde a preparação inicial dos dados até a modelagem estatística e a criação de visualizações informativas.

Além disso, o Pandas possui uma comunidade ativa de desenvolvedores e usuários que contribuem constantemente com novos recursos, melhorias de desempenho e soluções para problemas comuns enfrentados pelos analistas de dados. Isso garante que a biblioteca esteja sempre atualizada e pronta para atender às necessidades em constante evolução dos profissionais de análise de dados em todo o mundo.

Em resumo, o Pandas é uma ferramenta essencial para qualquer pessoa que trabalhe com dados em Python, fornecendo as ferramentas e recursos necessários para realizar análises de dados complexas e extrair insights valiosos de conjuntos de dados de qualquer tamanho ou complexidade.

PRINCIPAIS COMPONENTES

DataFrame: O principal objeto do Pandas é o DataFrame, uma estrutura de dados bidimensional semelhante a uma tabela, onde os dados são organizados em linhas e colunas. Cada coluna pode ter um tipo de dado diferente (como inteiro, flutuante, string, etc.).

Series: Uma Series é uma estrutura de dados unidimensional que pode conter qualquer tipo de dado, semelhante a um array ou lista em Python. Uma Series é uma única coluna de um DataFrame.

PRINCIPAIS FUNCIONALIDADES

Leitura e Escrita de Dados: O Pandas oferece funções para ler e escrever dados em vários formatos, incluindo CSV, Excel, SQL, JSON e HDF5.

Manipulação de Dados: O Pandas fornece uma ampla gama de funções para manipular dados, incluindo seleção, filtragem, ordenação, agrupamento, pivotamento, junção e combinação de dados.

Limpeza de Dados: O Pandas facilita a limpeza e a preparação de dados, permitindo lidar com valores ausentes, remover duplicatas, converter tipos de dados, renomear colunas e muito mais.

Visualização de Dados: Embora o Pandas não seja uma biblioteca de visualização de dados por si só, ele é frequentemente utilizado em conjunto com bibliotecas como Matplotlib e Seaborn para criar visualizações de dados poderosas e informativas.

Análise de Dados Estatísticos: O Pandas oferece funções para calcular estatísticas descritivas, como média, mediana, desvio padrão e correlação, além de suportar operações de agregação e resumo de dados.

Operações de Indexação Avançadas: O Pandas suporta indexação sofisticada, permitindo acessar e manipular dados com base em rótulos de linhas e colunas, números inteiros, expressões booleanas e até mesmo índices hierárquicos.

Tratamento de Dados Temporais: O Pandas possui funcionalidades específicas para lidar com dados temporais, como conversão de tipos de dados de data e hora, resampling, shifting e rolling calculations.

Concatenação e Mesclagem de DataFrames: O Pandas oferece métodos flexíveis para combinar múltiplos DataFrames, seja através de concatenação simples, mesclagem de conjuntos de dados com base em chaves comuns ou junção de conjuntos de dados em operações de banco de dados estilo SQL.

Manipulação de Cadeias de Caracteres: O Pandas inclui métodos poderosos para manipular strings dentro de colunas, como pesquisa, substituição, divisão e formatação de strings.

Leitura de Dados Remotos: Além de ler dados de arquivos locais, o Pandas pode carregar dados diretamente de fontes remotas, como URLs da web, bancos de dados SQL e APIs RESTful.

Suporte a Dados Categóricos: O Pandas oferece suporte para dados categóricos, permitindo a representação eficiente de variáveis categóricas em conjuntos de dados e facilitando operações como ordenação e filtragem.

Desempenho Otimizado: O Pandas foi projetado para oferecer desempenho otimizado para manipulação de grandes conjuntos de dados, com métodos internos implementados em C ou Cython para garantir velocidade e eficiência.

Extensibilidade: O Pandas é altamente extensível, permitindo a criação de novos tipos de dados, funções personalizadas e métodos de extensão para atender às necessidades específicas de análise de dados de um projeto.

Esses são apenas alguns dos recursos adicionais que o Pandas oferece, mostrando o quão poderosa e versátil essa biblioteca é para a análise de dados em Python.

NUMPY

NumPy, uma biblioteca fundamental em Python para computação numérica, desempenha um papel crucial em uma ampla gama de domínios, desde ciência de dados até processamento de sinais. Seu principal objeto, o ndarray, é uma estrutura de dados multidimensional que oferece suporte a arrays multidimensionais e permite armazenar elementos de tipos de dados homogêneos. Esses arrays são altamente eficientes em termos de memória e possibilitam a execução de operações vetorizadas, o que significa que operações matemáticas podem ser aplicadas a todos os elementos do array de uma só vez, resultando em um desempenho significativamente mais rápido em comparação com loops tradicionais.

Adicionalmente, NumPy oferece uma vasta coleção de funções para realizar operações matemáticas eficientes em grandes conjuntos de dados. Essas funções abrangem uma variedade de áreas, desde operações básicas

de álgebra linear, como multiplicação de matriz e decomposição de valores singulares, até funções mais avançadas, como transformada de Fourier e manipulação de dados estatísticos.

Graças à sua eficiência e flexibilidade, NumPy é amplamente utilizado em uma variedade de campos, incluindo ciência de dados, computação científica, aprendizado de máquina, processamento de sinais e muito mais. Sua capacidade de lidar com grandes volumes de dados e realizar cálculos complexos de forma rápida e eficiente o torna uma escolha indispensável para qualquer pessoa envolvida em análises numéricas e computacionais.

PRINCIPAIS FUNCIONALIDADES

Operações Matemáticas: NumPy fornece uma ampla gama de funções matemáticas e operadores para realizar operações em arrays, incluindo adição, subtração, multiplicação, divisão, exponenciação, trigonometria e álgebra linear.

Indexação e Fatiamento: NumPy suporta indexação sofisticada e fatiamento de arrays, permitindo acessar e manipular elementos individuais ou subconjuntos de dados dentro de arrays.

Broadcasting: NumPy permite operações entre arrays de diferentes formas e tamanhos, por meio de um mecanismo chamado "broadcasting", que estende automaticamente os arrays para que possam ser compatíveis para operações elementares.

Funções de Redução e Estatísticas: NumPy oferece funções para calcular estatísticas descritivas, como média, mediana, desvio padrão, mínimo, máximo, soma e produtos cumulativos, ao longo de eixos específicos dos arrays.

Operações de Álgebra Linear: NumPy inclui funções para realizar operações de álgebra linear, como multiplicação de matrizes, inversão

de matrizes, decomposição de valores singulares, decomposição de valores próprios e resolução de sistemas de equações lineares.

Geração de Números Aleatórios: NumPy possui um submódulo **random** que fornece funções para gerar números aleatórios com várias distribuições, bem como funções para permutação e amostragem aleatória de dados.

Essa é apenas uma introdução ao NumPy. A biblioteca oferece muitos recursos adicionais, incluindo suporte para álgebra linear, transformada de Fourier, interpolação, processamento de imagens e muito mais. NumPy é amplamente considerado como uma pedra angular para computação científica em Python, e seu desempenho eficiente e ampla gama de funcionalidades o tornam uma escolha popular entre pesquisadores, cientistas de dados e engenheiros.

PANDAS E NUMPY

OBTER DADOS DE UMA PASTA DE TRABALHO DO EXCEL

Para obter dados de uma pasta de trabalho do Excel em Python, você pode usar a biblioteca Pandas, que oferece suporte à leitura de arquivos Excel. Aqui está um exemplo básico de como fazer isso.

```
import pandas as pd

caminho_arquivo_excel = 'caminhoDoArquivo/nomeDoArquivo.xlsx'
dados = pd.read_excel(caminho_arquivo_excel)

print(dados)
```

OBTER DADOS DE UM ARQUIVO TEXTO/CSV

Para obter dados de um arquivo texto ou CSV em Python, você pode usar a biblioteca Pandas, que oferece suporte à leitura desses tipos de arquivos. Aqui está um exemplo básico de como fazer isso:

```
import pandas as pd

dados_csv = pd.read_csv('caminhodoarquivo/nomedoarquivo.csv')
print(dados_csv.head())
```

Note que, sendo o delimitador (“,”) podemos simplesmente solicitar a leitura, mas caso seja outro, é necessário informá-lo.

```
dados_csv = pd.read_csv(
    'caminhodoarquivo/nomedoarquivo.csv',
    delimiter='\t'
)
```

OBTER DADOS DE UM PDF

Para obter dados de um arquivo PDF em Python, você pode usar várias bibliotecas, como PyPDF2, pdfplumber ou PyMuPDF, para extrair o texto do PDF. Aqui está um exemplo básico usando a biblioteca pdfplumber:

```
import pdfplumber

with pdfplumber.open('caminhoarquivo/arquivo.pdf') as pdf:
    paginas = [page.extract_text() for page in pdf.pages]

[
    print(f"Texto da página {i+1}:\n{texto}\n")
    for i, texto in enumerate(texto_paginas)
]
```

Veja como ficaria utilizando a lib PyPDF2

```
import PyPDF2

with open('caminhodoarquivo/arquivo.pdf', 'rb') as pdf:
    leitor = PyPDF2.PdfFileReader(pdf)
    paginas = [
        leitor.getPage(num_pagina)
        .extractText()
        for num_pagina in range(leitor.numPages)
    ]

[
    print(f"Dados ↓ {i+1}:\n{texto}\n")
    for i, texto in enumerate(paginas)
]
```

CAPÍTULO 5

DATAFRAME

DataFrame em Python é uma estrutura de dados bidimensional tabular, semelhante a uma planilha ou tabela de banco de dados. Ele é parte integrante da biblioteca Pandas, que é amplamente utilizada para manipulação e análise de dados em Python. Aqui estão algumas características importantes de um DataFrame:

O DataFrame permite operações flexíveis de manipulação e análise de dados, incluindo filtragem, seleção, agregação e visualização de dados. Essas operações podem ser realizadas de forma eficiente e intuitiva, facilitando a exploração e compreensão dos dados.

BIDIMENSIONAL

Um DataFrame é bidimensional, o que significa que ele tem linhas e colunas. Cada linha representa uma observação ou registro, enquanto cada coluna representa uma variável ou atributo.

TABULAR

Um DataFrame é organizado em uma forma tabular, onde os dados são armazenados em células dispostas em linhas e colunas. Isso facilita a visualização e manipulação dos dados, especialmente quando se trabalha com conjuntos de dados grandes.

HETEROGENEIDADE

Um DataFrame pode conter diferentes tipos de dados em diferentes colunas. Por exemplo, uma coluna pode conter valores inteiros, enquanto outra coluna pode conter valores de ponto flutuante ou strings.

ÍNDICES

Um DataFrame possui um índice para identificar exclusivamente cada linha. Por padrão, o índice é uma sequência numérica começando do zero, mas também pode ser personalizado para ser uma coluna existente ou uma combinação de colunas.

OPERAÇÕES DE MANIPULAÇÃO

Pandas fornece uma ampla gama de funcionalidades para manipular DataFrames, incluindo seleção de dados, filtragem, agrupamento, ordenação, junção e muito mais. Isso torna a análise de dados e preparação de dados mais eficientes e convenientes.

INTEGRAÇÃO COM OUTRAS BIBLIOTECAS

DataFrames Pandas são frequentemente usados em conjunto com outras bibliotecas Python, como NumPy, Matplotlib, Scikit-learn e muitas outras, para realizar análises de dados avançadas, visualizações e modelagem de machine learning.

Para criar um DataFrame em Python, você pode usar a função pd.DataFrame() da biblioteca Pandas, passando um dicionário ou lista de dicionários como argumento, onde as chaves do dicionário representam os nomes das colunas e os valores do dicionário representam os dados de cada coluna.

```python
import pandas as pd

vendedores = {
    'id': [1, 2, 3, 4, 5],
    'nome': ['Léo', 'Maria', 'Camila', 'Ebony', 'Luiza'],
    'data de nascimento': [
        '1990-05-15', '1987-09-22', '1985-02-10',
        '1993-11-30', '1988-07-05'
    ],
    'cidade': [
        'São Paulo', 'Rio de Janeiro', 'Belo Horizonte',
        'Salvador', 'Brasília'
    ],
    'estado': ['SP', 'RJ', 'MG', 'BA', 'DF'],
    'país': ['Brasil', 'Brasil', 'Brasil', 'Brasil', 'Brasil']
}

df_vendedores = pd.DataFrame(vendedores)
print(df_vendedores)
```

Veja um exemplo abaixo:

Saída:

```
PS C:\Users\Leonardo Alves\Documents\GitHub\default_cursos\Clarify_Python2\01_Pandas> python .\lib_pandas.py
   id    nome data de nascimento         cidade estado    país
0   1     Léo        1990-05-15      São Paulo     SP  Brasil
1   2   Maria        1987-09-22  Rio de Janeiro     RJ  Brasil
2   3  Camila        1985-02-10  Belo Horizonte     MG  Brasil
3   4   Ebony        1993-11-30       Salvador     BA  Brasil
4   5   Luiza        1988-07-05       Brasília     DF  Brasil
```

INSERÇÃO DE DADOS MANUAL

```
import pandas as pd

novo_vendedor = dict(
    id = 6,
    nome = 'Miguel',
    cidade = 'Porto Alegre',
    país = 'Brasil'
)

df_vendedores = df_vendedores._append(
    pd.DataFrame(novo_vendedor), ignore_index=True)

print(df_vendedores)
```

Note que algumas colunas propositalmente não foram informadas ao inserir um novo registro, isto para demonstrar que não seria de fato um problema em relação ao funcionamento da operação, mas observe na saída como ficaram as colunas ‘data de nascimento’ e ‘estado’ na última linha.

```
PS C:\Users\Leonardo Alves\Documents\GitHub\default_cursos\Clarify_Python2\01_Pandas> python .\lib_pandas.py
   id    nome data de nascimento          cidade estado    país
0   1     Léo        1990-05-15       São Paulo     SP  Brasil
1   2   Maria        1987-09-22  Rio de Janeiro     RJ  Brasil
2   3  Camila        1985-02-10  Belo Horizonte     MG  Brasil
3   4   Ebony        1993-11-30        Salvador     BA  Brasil
4   5   Luiza        1988-07-05        Brasília     DF  Brasil
5   6  Miguel               NaN    Porto Alegre    NaN  Brasil
```

CAPÍTULO 6

TRATAMENTO E MANIPULAÇÃO DE DADOS

O tratamento e a manipulação de dados são partes essenciais do processo de análise de dados, envolvendo uma série de técnicas para preparar, limpar, transformar e analisar conjuntos de dados. Aqui estão algumas das principais etapas e técnicas envolvidas:

COLETA DE DADOS

O processo começa com a coleta de dados relevantes de diversas fontes, que podem incluir bancos de dados, arquivos de texto, feeds de mídia social, dispositivos IoT (Internet das Coisas) e muitos outros.

LIMPEZA DE DADOS

Muitas vezes, os dados coletados estão incompletos, inconsistentes ou incorretos. A limpeza de dados envolve a identificação e correção de erros, remoção de valores ausentes ou duplicados e padronização de formatos.

TRANSFORMAÇÃO DE DADOS

Esta etapa envolve a modificação dos dados para torná-los mais adequados para análise. Isso pode incluir a agregação de dados, a criação de novas variáveis ou a normalização de valores para uma escala comum.

ANÁLISE EXPLORATÓRIA DE DADOS

Antes de aplicar técnicas mais avançadas de análise, é útil realizar uma análise exploratória para entender a distribuição dos dados, identificar padrões e tendências e formular hipóteses iniciais.

MODELAGEM DE DADOS

Dependendo dos objetivos da análise, podem ser aplicadas técnicas estatísticas ou algoritmos de aprendizado de máquina para extrair insights dos dados. Isso pode incluir desde modelos simples de regressão até redes neurais profundas, dependendo da complexidade do problema.

VISUALIZAÇÃO DE DADOS

A visualização de dados é uma ferramenta poderosa para comunicar resultados e insights de forma eficaz. Gráficos, mapas e dashboards interativos são comumente utilizados para apresentar informações de maneira clara e intuitiva.

VALIDAÇÃO DE DADOS

É importante verificar a qualidade e a integridade dos dados ao longo de todo o processo de tratamento e análise. Isso pode envolver a aplicação de técnicas estatísticas para detectar outliers ou a realização de testes de sensibilidade para avaliar a robustez dos resultados.

GERENCIAMENTO DE DADOS

Uma vez concluída a análise, os resultados devem ser documentados e armazenados de forma adequada. Isso pode incluir a criação de relatórios ou apresentações para compartilhar os insights com outras partes interessadas, bem como o arquivamento dos dados originais e do código utilizado para análise futura.

Agora, vamos manipular uma base que está no formato CSV delimitado por vírgula.

```
import pandas as pd

dados = pd.read_csv('dados.csv')
```

Vamos remover linhas duplicadas com nas colunas ‘nome’ e ‘data de nascimento’, em seguida vamos corrigir inconsistências nas datas de nascimento.

```
import pandas as pd

dados = pd.read_csv('dados.csv')

dados = dados.drop_duplicates(
    subset=[
        'nome', 'data_nascimento'
    ]
)

dados['data_nascimento'] = pd.to_datetime(
    dados['data_nascimento'], errors='coerce'
)
```

Vamos adicionar uma coluna para manter a Idade calculada e em seguida normalizar os nomes das colunas ‘cidade’ e ‘estado’.

```
dados['idade'] = pd.to_datetime('now').year - \
    pd.to_datetime(dados['data_nascimento']).dt.year

dados['cidade'] = dados['cidade'].str.upper()
dados['estado'] = dados['estado'].str.upper()
```

Faremos agora, uma análise exploratória de dados calculando a média e mediana de idade, em seguida usaremos uma técnica de clustering para identificar grupos demográficos semelhantes.

```
media_idade = dados['idade'].mean()
mediana_idade = dados['idade'].median()

X = dados[['idade', 'cidade', 'estado']].copy()
X['cidade_estado'] = X['cidade'] + ', ' + X['estado']
X_encoded = pd.get_dummies(X[['idade', 'cidade_estado']])
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_encoded)
dados['cluster'] = kmeans.labels_
```

Por fim, vamos salvar os dados em um novo arquivo.

```
dados.to_csv('dados_processados.csv', index=False)
```

COMBINAR DADOS DE DIFERENTES FONTES

Na análise de dados, o tratamento e a manipulação dos dados desempenham um papel crucial na preparação dos dados para análise. Essas etapas envolvem uma série de técnicas que vão desde a limpeza e transformação dos dados até a agregação e formatação para análise posterior. Entre as técnicas comuns estão a remoção de valores ausentes, a normalização de dados, a detecção e tratamento de outliers, a codificação de variáveis categóricas e a agregação de dados.

A escolha da técnica adequada depende da estrutura e da natureza dos dados, bem como dos objetivos específicos da análise. Por exemplo, em um conjunto de dados com muitos valores ausentes, pode ser necessário realizar imputação de dados para preencher lacunas, enquanto em um conjunto de dados com variáveis categóricas, pode ser necessário realizar codificação one-hot ou codificação ordinal para permitir sua inclusão em modelos de análise.

Além disso, a eficácia das técnicas de tratamento e manipulação de dados pode impactar diretamente a qualidade e a confiabilidade dos resultados da análise. Portanto, é importante considerar cuidadosamente cada etapa do processo e aplicar as técnicas mais apropriadas para garantir que os dados estejam prontos para análise precisa e significativa.

CONCATENAÇÃO

Quando os dados de diferentes fontes possuem estruturas de colunas semelhantes, é possível mesclá-los simplesmente concatenando-os ao longo das linhas (verticalmente) ou das colunas (horizontalmente). Essa abordagem é especialmente útil quando se deseja combinar conjuntos de dados que contenham informações semelhantes, permitindo uma análise mais abrangente e integrada dos dados disponíveis. A concatenação de dados ao longo das linhas é frequentemente usada para combinar conjuntos de dados que representam observações semelhantes, enquanto a concatenação ao longo das colunas é útil para adicionar novas variáveis ou informações aos dados existentes. Essa flexibilidade na manipulação e combinação de conjuntos de dados é fundamental para a análise de dados eficaz, proporcionando aos analistas uma visão mais completa e integrada dos dados disponíveis.

```
df_combined = pd.concat([df1, df2], axis=0)
```

MERGE

Quando trabalhamos com conjuntos de dados provenientes de diversas fontes, muitas vezes encontramos informações complementares que podem ser combinadas para enriquecer nossa análise. Para realizar essa integração de dados, podemos utilizar a função merge() disponível na biblioteca Pandas do Python.

A função merge() permite combinar dois ou mais DataFrames com base em uma ou mais colunas-chave, que servem como referência para

identificar as relações entre os dados. Esse processo é análogo à operação de junção (join) em bancos de dados relacionais, onde as linhas dos DataFrames são combinadas de acordo com os valores das colunas-chave especificadas.

Ao mesclar DataFrames, podemos agregar informações relacionadas de diferentes fontes em um único conjunto de dados, facilitando a análise e a tomada de decisões. Por exemplo, podemos mesclar dados de vendas com dados de clientes com base em um identificador único de cliente, combinando assim informações de transações com detalhes do cliente, como nome, endereço e histórico de compras.

Além disso, a função merge() oferece flexibilidade para especificar o tipo de junção a ser realizada, como junção interna, junção externa, junção à esquerda ou junção à direita, permitindo controlar como os dados serão combinados e tratados em casos de valores ausentes.

Em resumo, a função merge() é uma ferramenta poderosa para integrar e consolidar dados de diferentes fontes, possibilitando uma análise mais abrangente e insights mais ricos ao trabalhar com conjuntos de dados complexos e inter-relacionados.

```
df_merged = pd.merge(df1, df2, on='coluna_chave')
```

JOIN

Quando precisamos combinar dados de várias fontes com base em índices ou colunas de índices, a operação de junção é a ferramenta adequada. Similar à mesclagem, a junção permite essa integração, porém com foco específico em utilizar os índices como base para a combinação dos dados. No contexto do Pandas em Python, podemos empregar o método join() para realizar essa operação.

O método join() possibilita a combinação de DataFrames utilizando os índices como referência, garantindo que as linhas correspondentes em ambos os DataFrames sejam unidas corretamente. Isso é particularmente útil

quando os DataFrames têm índices comuns ou relacionados, e desejamos agregar os dados de acordo com esses índices compartilhados.

Por exemplo, podemos ter um DataFrame contendo informações sobre vendas, onde os índices representam datas, e outro DataFrame contendo informações sobre o clima, onde os índices também representam datas. Ao usar o método join() com base nos índices de datas, podemos combinar esses dois conjuntos de dados de forma a incluir as informações sobre o clima junto com as informações de vendas para cada data correspondente.

Além disso, o método join() oferece flexibilidade para especificar o tipo de junção a ser realizada, assim como na função merge(), permitindo controlar como os dados serão combinados e tratados em casos de valores ausentes.

Em suma, o método join() é uma ferramenta valiosa para integrar dados de várias fontes com base em índices comuns, facilitando a análise e permitindo uma compreensão mais abrangente dos dados ao trabalhar com conjuntos de dados relacionados.

```
df_joined = df1.join(df2, how='inner')
```

CAPÍTULO 7

TRATAMENTO DE DATA E HORA PARA ANÁLISES TEMPORAIS.

O tratamento de datas e horas desempenha um papel crucial na análise de dados temporais em diversas áreas, desde finanças e saúde até ciência climática e logística. Em Python, uma linguagem de programação amplamente utilizada em ciência de dados e análise, existem várias técnicas e bibliotecas disponíveis para lidar com esse aspecto essencial da análise de dados. Este trabalho tem como objetivo explorar algumas das técnicas comuns para o tratamento de datas e horas em Python, destacando suas aplicações e importância na análise de dados temporais.

IMPORTÂNCIA DO TRATAMENTO DE DATAS E HORAS:

Antes de adentrar nas técnicas específicas, é fundamental compreender a importância do tratamento adequado de datas e horas na análise de dados. Datas e horas são elementos fundamentais em muitos conjuntos de dados, fornecendo contexto temporal para eventos e transações. A precisão na manipulação desses dados é essencial para garantir a validade e confiabilidade das análises realizadas. Além disso, uma compreensão sólida das técnicas de tratamento de datas e horas permite uma análise mais detalhada e precisa, levando a insights mais significativos e decisões mais informadas.

CONVERTER STRINGS EM OBJETOS DE DATA E HORA E EXTRAIR COMPONENTES.

```
df['data'] = pd.to_datetime(df['data'])
df['ano'] = df['data'].dt.year
df['mes'] = df['data'].dt.month
df['dia'] = df['data'].dt.day
```

GROUPBY

A técnica de GroupBy é uma ferramenta poderosa em Python, utilizada para agrupar dados com base em valores de uma ou mais colunas. Uma vez que os dados são agrupados, é possível aplicar funções de agregação aos grupos resultantes. Por exemplo, podemos agrupar dados por uma coluna-chave e calcular a média, soma, mediana, máximo, mínimo ou outras estatísticas para cada grupo.

Essa capacidade de agrupar e resumir dados de maneira flexível é essencial para análises exploratórias e sumarizações de dados em Python. O processo de GroupBy permite uma visão mais detalhada dos padrões e tendências nos dados, facilitando a tomada de decisões informadas com base em insights derivados da análise.

Além disso, o GroupBy pode ser combinado com outras operações, como filtragem e transformação, para realizar análises mais complexas e sofisticadas. Por exemplo, podemos filtrar grupos com base em critérios específicos ou aplicar transformações personalizadas aos dados dentro de cada grupo.

Em resumo, a técnica de GroupBy é uma ferramenta essencial em Python para análise de dados, permitindo a segmentação e sumarização eficiente de conjuntos de dados complexos, facilitando a compreensão e interpretação dos dados e fornecendo insights valiosos para orientar a tomada de decisões.

```
data = {'Grupo': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Valor': [10, 20, 30, 40, 50, 60]}
df = pd.DataFrame(data)

media_por_grupo = df.groupby('Grupo').mean()
print(media_por_grupo)
```

MODA

Na análise estatística, a moda é uma medida de tendência central que representa o valor mais frequente em um conjunto de dados. Ela nos fornece insights sobre a ocorrência mais comum dentro de um conjunto de observações. A identificação da moda é especialmente útil para compreender a distribuição dos dados e destacar padrões predominantes.

Ao examinar conjuntos de dados, a moda ajuda a identificar picos ou agrupamentos significativos, oferecendo uma visão sobre as preferências, comportamentos ou características predominantes na amostra analisada. Além disso, a presença de modas múltiplas ou a ausência de uma moda clara podem indicar diferentes aspectos da distribuição dos dados, como uniformidade, bimodalidade ou heterogeneidade.

Embora a moda forneça informações valiosas sobre a frequência de ocorrência de valores específicos, é importante considerá-la juntamente com outras medidas de tendência central, como a média e a mediana, para obter uma compreensão completa da distribuição dos dados. Essa abordagem holística ajuda a interpretar de forma mais precisa a natureza dos dados e a tomar decisões informadas com base em sua análise estatística.

```
moda = df['Valor'].mode()
print("Moda:", moda[0])
```

MÉDIA E MEDIANA

A média, uma medida de tendência central, é calculada como a média aritmética de um conjunto de valores, representando o valor típico ou central dos dados. Por outro lado, a mediana é o valor que ocupa a posição central quando os dados estão ordenados em ordem crescente ou decrescente. Enquanto a média é sensível a valores extremos, a mediana é uma medida mais robusta, menos influenciada por valores atípicos.

Ao calcular a média e a mediana para uma coluna específica de um DataFrame, obtemos informações valiosas sobre a distribuição dos dados e sua tendência central. A média nos dá uma estimativa do valor típico dos dados, enquanto a mediana representa o ponto central quando os dados estão organizados em ordem. A comparação entre essas duas medidas ajuda a compreender melhor a forma da distribuição e a identificar possíveis assimetrias ou valores extremos que possam influenciar a interpretação dos dados.

Essas medidas são essenciais na análise estatística e são frequentemente utilizadas para resumir e entender conjuntos de dados, fornecendo insights sobre suas características centrais. No entanto, é importante interpretar essas medidas em conjunto com outras informações, como a moda e a variabilidade dos dados, para obter uma compreensão abrangente da distribuição e da variabilidade dos dados.

```
media = df['Valor'].mean()
mediana = df['Valor'].median()
print(f'Média:, {media} \nMediana: {mediana}')
```

INDEXAÇÃO

A indexação desempenha um papel crucial na manipulação e acesso de dados em um DataFrame ou em uma série no Pandas. Por meio da indexação, é possível selecionar e acessar linhas, colunas ou elementos específicos com base em seus índices correspondentes.

Ao utilizar a indexação em um DataFrame ou série, podemos especificar os índices desejados para recuperar dados específicos. Isso permite a flexibilidade de acessar dados de acordo com requisitos específicos de análise ou manipulação. Por exemplo, podemos acessar uma coluna específica de um DataFrame ou uma série de valores com base em seus índices, facilitando a análise de dados específicos.

Além disso, a indexação oferece suporte a diferentes tipos de índices, como índices inteiros, índices de rótulos ou índices de datas, proporcionando uma ampla gama de opções para acessar e manipular dados de maneira eficiente.

Ao dominar as técnicas de indexação, os usuários podem explorar e analisar dados de forma mais eficaz, permitindo uma compreensão mais profunda dos conjuntos de dados e facilitando a tomada de decisões informadas com base nas informações extraídas.

```
valor_na_linha_3 = df.loc[2, 'Valor']
print("Valor na linha 3:", valor_na_linha_3)
```

CLASSIFICAÇÃO

A classificação é uma operação fundamental na manipulação de dados em um DataFrame no Pandas. Ela permite ordenar os dados com base nos valores de uma ou mais colunas, organizando as linhas em uma ordem específica de acordo com os critérios de classificação definidos.

Ao realizar a classificação em um DataFrame, os valores de uma ou mais colunas são utilizados como referência para determinar a ordem das linhas. Isso é útil para identificar padrões, tendências ou outliers nos dados, facilitando a análise e interpretação dos resultados.

Por exemplo, podemos classificar um DataFrame de vendas com base no valor das vendas, do maior para o menor, para identificar os produtos mais lucrativos. Da mesma forma, podemos classificar um conjunto de dados de clientes com base na idade, do mais novo para o mais velho, para entender melhor a distribuição etária da base de clientes.

A capacidade de classificar dados de maneira flexível e eficiente é essencial para explorar e compreender conjuntos de dados complexos. Ao dominar as técnicas de classificação, os usuários podem organizar os dados de acordo com suas necessidades analíticas, facilitando a identificação de insights e a tomada de decisões informadas com base nas informações apresentadas.

```
df_classificado = df.sort_values(by='Valor', ascending=False)
print(df_classificado)
```

ASSIGN E DROP

Assign e Drop são operações essenciais na manipulação de DataFrames no Pandas.

O método Assign permite adicionar novas colunas a um DataFrame com base em cálculos ou valores existentes. Isso é particularmente útil para criar novas variáveis derivadas ou realizar transformações nos dados existentes. Por exemplo, podemos utilizar Assign para calcular uma nova coluna que represente a soma de duas outras colunas ou para aplicar uma função a uma coluna existente e atribuir o resultado a uma nova coluna.

Por outro lado, o método Drop é utilizado para remover colunas ou linhas de um DataFrame, o que pode ser necessário para simplificar os dados ou para focar em características específicas. Ao usar Drop, podemos descartar colunas ou linhas que não são relevantes para a análise ou que contenham valores ausentes ou inconsistentes. Isso ajuda a limpar e preparar os dados para análises posteriores, garantindo que apenas as informações relevantes sejam mantidas no DataFrame.

Essas operações oferecem flexibilidade e poder na manipulação de DataFrames, permitindo aos usuários ajustar e personalizar seus conjuntos de dados de acordo com as necessidades específicas de análise. Ao dominar o uso de Assign e Drop, os analistas de dados podem preparar e limpar eficientemente seus dados para análises mais avançadas e insights mais profundos.

```
df = df.assign(Novo_Valor=df['Valor'] * 2)
df_sem_grupo = df.drop(columns=['Grupo'])
print(df_sem_grupo)
```

DROPNA E FILLNA

Dropna e Fillna são ferramentas importantes na manipulação de DataFrames no Pandas.

O método Dropna é utilizado para remover linhas ou colunas que contenham valores ausentes (NaN) de um DataFrame. Isso é útil para limpar os dados e garantir que apenas observações completas sejam consideradas em análises subsequentes. Por exemplo, se tivermos um conjunto de dados onde algumas linhas possuem valores ausentes em uma ou mais colunas, podemos usar o Dropna para descartar essas linhas e garantir que nosso conjunto de dados esteja completo.

Por outro lado, o método Fillna é utilizado para preencher os valores ausentes com um valor específico. Isso é útil quando queremos manter as linhas ou colunas no DataFrame, mas substituir os valores ausentes por valores conhecidos ou estimados. Por exemplo, podemos usar Fillna para substituir todos os valores ausentes por zero, pela média dos valores existentes ou por qualquer outro valor relevante para o contexto da análise.

Essas operações são essenciais na limpeza e preparação de dados, garantindo que os conjuntos de dados estejam completos e prontos para análises subsequentes. Ao dominar o uso de Dropna e Fillna, os analistas de dados podem lidar eficientemente com valores ausentes em seus conjuntos de dados e realizar análises mais precisas e robustas.

```python
df.loc[0, 'Valor'] = None
df_sem_nulos = df.dropna()
valor_medio = df['Valor'].mean()
df_preenchido = df.fillna(valor_medio)

print(f"DataFrame sem valores nulos: {df_sem_nulos}")
print(f"DataFrame com valores preenchidos: {df_preenchido}")
```

MATPLOTLIB

Matplotlib é uma biblioteca fundamental em Python para visualização de dados, oferecendo uma ampla gama de opções para criar gráficos e visualizações de forma flexível e poderosa. Com ela, é possível gerar gráficos de linhas, gráficos de dispersão, histogramas, gráficos de barras, gráficos de pizza e muitos outros tipos de visualizações.

Uma das vantagens da Matplotlib é sua API simples e intuitiva, que permite aos usuários criar e personalizar gráficos de maneira eficiente. É possível ajustar todos os aspectos visuais dos gráficos, incluindo cores, estilos de linha, marcadores, rótulos, títulos e legendas, de acordo com as necessidades específicas de cada análise.

Além disso, a Matplotlib é altamente integrada com outras bibliotecas de análise de dados em Python, como NumPy e pandas. Isso significa que os usuários podem facilmente criar gráficos a partir de dados armazenados em estruturas de dados dessas bibliotecas, facilitando a análise exploratória de dados e a comunicação visual de resultados.

Graças à sua versatilidade e integração, a Matplotlib é uma ferramenta indispensável para cientistas de dados, analistas e pesquisadores que desejam explorar e comunicar insights a partir de seus dados de forma eficaz. Com ela, é possível criar visualizações impactantes que ajudam a entender melhor os dados e a extrair informações valiosas.

SEABORN

Seaborn é uma biblioteca de visualização de dados em Python construída sobre a Matplotlib, projetada para simplificar a criação de gráficos estatísticos informativos e visualmente atraentes. Ela oferece uma interface de alto nível que permite aos usuários criar gráficos complexos com poucas linhas de código, acelerando o processo de exploração e comunicação de dados.

Uma das características distintivas do Seaborn é sua capacidade de visualizar relacionamentos estatísticos em conjuntos de dados. Isso inclui a criação de gráficos de dispersão para explorar correlações, gráficos de categorias para comparar diferentes grupos e gráficos de distribuição para examinar a distribuição dos dados. Além disso, o Seaborn facilita a visualização de relacionamentos entre variáveis complexas, tornando mais fácil identificar padrões e tendências nos dados.

Outra vantagem do Seaborn são seus estilos predefinidos atraentes, que proporcionam uma aparência profissional aos gráficos sem a necessidade de ajustes adicionais. Além disso, a biblioteca oferece suporte para cores semânticas, paletas de cores personalizadas e uma variedade de opções de personalização para adaptar a aparência dos gráficos às necessidades específicas do usuário.

O Seaborn permite criar visualizações poderosas para explorar e comunicar padrões e insights nos dados de forma eficaz. Sua facilidade de uso, funcionalidades avançadas e estética atraente o tornam uma ferramenta valiosa para cientistas de dados, analistas e pesquisadores em busca de insights em seus conjuntos de dados.

MATPLOTLIB E SEABORN

Juntas, Matplotlib e Seaborn oferecem uma poderosa combinação de flexibilidade, funcionalidade e estética para a visualização de dados em Python. Seja explorando tendências, identificando padrões ou comunicando resultados, essas bibliotecas fornecem as ferramentas necessárias para criar visualizações impactantes que ajudam os usuários a extrair insights significativos de seus dados.

CAPÍTULO 8

TIPOS DE GRÁFICOS

A capacidade de visualizar dados é uma habilidade-chave para qualquer cientista de dados ou analista. Python oferece uma ampla gama de bibliotecas poderosas para criar gráficos e visualizações informativas. Aqui estão alguns tipos comuns de gráficos que podem ser facilmente criados usando Python:

Gráfico de Linhas com Matplotlib: Em Python, a biblioteca Matplotlib permite criar gráficos de linhas para visualizar tendências ao longo do tempo. Com apenas algumas linhas de código, você pode traçar o crescimento de vendas ao longo dos meses ou acompanhar a evolução de indicadores-chave de desempenho.

Gráfico de Barras com Seaborn: Com o Seaborn, uma biblioteca de visualização de dados em Python, é fácil criar gráficos de barras para comparar diferentes categorias. Esses gráficos são ideais para visualizar a receita por região, a performance de diferentes produtos ou o desempenho de equipes em diferentes departamentos.

Gráfico de Dispersão com Pandas: Utilizando a biblioteca Pandas, você pode criar gráficos de dispersão para visualizar a relação entre duas variáveis. Esses gráficos são valiosos para identificar padrões e correlações nos dados, como a relação entre a idade dos clientes e o valor gasto em uma loja online.

Histograma com NumPy: NumPy, uma biblioteca fundamental para computação numérica em Python, facilita a criação de histogramas para representar a distribuição de uma variável numérica. Com apenas algumas linhas de código, você pode entender a distribuição de idades em uma amostra populacional ou a distribuição de notas em um conjunto de dados educacionais.

Gráfico de Pizza com Plotly: Plotly é outra biblioteca de visualização popular em Python que permite criar gráficos interativos, incluindo gráficos de pizza. Esses gráficos são úteis para mostrar a proporção de diferentes

categorias em um conjunto de dados, como a distribuição de gastos em diferentes categorias de despesas.

Com Python e suas bibliotecas de análise de dados, criar visualizações impactantes e informativas torna-se uma tarefa acessível e eficiente. Essas ferramentas permitem explorar dados de forma mais profunda, comunicar insights importantes e tomar decisões mais informadas em uma variedade de contextos analíticos.

APLICAÇÃO E ESCOLHA ADEQUADA DE GRÁFICOS

A seleção do tipo de gráfico mais adequado é uma decisão crucial na visualização de dados, pois depende não apenas do tipo de dados que está sendo analisado, mas também da mensagem específica que se deseja transmitir. Por exemplo, ao comparar o desempenho de diferentes equipes ao longo do tempo, um gráfico de linhas é frequentemente escolhido devido à sua capacidade de destacar tendências e variações ao longo de uma série temporal. Esse tipo de gráfico permite uma análise clara das mudanças no desempenho ao longo do tempo, facilitando a identificação de padrões e tendências.

Por outro lado, se o objetivo é visualizar a distribuição de idades em uma população, um histograma é mais apropriado. Os histogramas são úteis para representar a distribuição de uma variável contínua, mostrando a frequência com que diferentes faixas de valores ocorrem. Isso oferece uma compreensão visual imediata da distribuição dos dados e destaca características como a concentração ou dispersão dos valores.

Além desses exemplos, existem uma variedade de outros tipos de gráficos disponíveis, como gráficos de dispersão, gráficos de barras, gráficos de pizza, entre outros, cada um com suas próprias vantagens e usos específicos. A escolha do gráfico mais adequado deve levar em consideração a natureza dos dados, o objetivo da análise e o público-alvo, garantindo que a mensagem seja transmitida de forma clara e eficaz.

CUSTOMIZAÇÃO DE GRÁFICOS

Tanto a Matplotlib quanto o Seaborn proporcionam uma ampla gama de opções para personalizar a aparência dos gráficos, permitindo aos usuários adaptar visualmente as visualizações de acordo com suas preferências e necessidades específicas de análise. Essas opções incluem a capacidade de modificar cores, estilos de linha, marcadores, rótulos, títulos e legendas, entre outros elementos visuais.

Por exemplo, é possível ajustar a cor das linhas em um gráfico de dispersão para destacar diferentes grupos de dados ou para corresponder a uma paleta de cores específica. Além disso, é possível adicionar uma grade de fundo ao gráfico para facilitar a leitura e interpretação dos dados, especialmente em gráficos mais complexos com muitas séries ou pontos de dados.

Outras personalizações possíveis incluem a modificação dos estilos de linha para realçar tendências ou padrões específicos nos dados, a adição de marcadores para enfatizar pontos de interesse e a inclusão de rótulos e legendas claros para fornecer contexto e explicação aos gráficos.

Essas opções de personalização são essenciais para criar visualizações de dados eficazes e informativas, que ajudam os usuários a extrair insights significativos e a comunicar resultados de forma clara e convincente. Ao dominar as técnicas de personalização disponíveis na Matplotlib e no Seaborn, os usuários podem criar visualizações visualmente atraentes que atendam às necessidades específicas de suas análises e projetos.

UTILIZAÇÃO DE ESTILOS PRÉ-CONFIGURADOS

Tanto a Matplotlib quanto o Seaborn fornecem estilos pré-configurados que facilitam a alteração rápida da aparência de todos os gráficos em um script ou notebook. Esses estilos pré-configurados oferecem uma maneira simples de ajustar a estética dos gráficos, fornecendo opções como 'seaborn', 'ggplot', 'dark_background' e outros.

Por exemplo, ao definir um estilo como 'seaborn', os gráficos adotarão automaticamente a estética visual característica do Seaborn, com cores suaves e linhas agradáveis. Da mesma forma, ao selecionar o estilo 'ggplot', os gráficos terão uma aparência semelhante àquela encontrada no sistema de gráficos do R, conhecido como ggplot2, com cores vibrantes e linhas distintas.

Essa capacidade de escolher entre diferentes estilos pré-configurados permite aos usuários adaptar rapidamente a aparência de seus gráficos para atender às suas preferências pessoais ou aos requisitos do projeto. Além disso, os estilos pré-configurados garantem uma consistência visual em todos os gráficos produzidos, tornando mais fácil a criação de visualizações coesas e profissionais em um conjunto de dados.

Com essa funcionalidade, os usuários podem criar visualizações de dados com uma aparência visualmente atraente e consistente, garantindo que suas análises sejam apresentadas de forma clara e profissional.

GRÁFICOS ESTÁTICOS

Gráficos estáticos são representações visuais de dados que não possuem interatividade e são amplamente utilizados para comunicar resultados de análises de forma eficaz. Esses gráficos são criados utilizando bibliotecas como Matplotlib e Seaborn em Python e são frequentemente utilizados em relatórios, apresentações e artigos científicos.

Ao contrário dos gráficos interativos, os gráficos estáticos são gerados em formato de imagem e não permitem que os usuários interajam diretamente com os dados. No entanto, eles oferecem uma representação visual clara e concisa dos resultados da análise, permitindo aos espectadores entender rapidamente os insights apresentados.

Uma das vantagens dos gráficos estáticos é a capacidade de serem salvos em diferentes formatos de arquivo, como PNG, JPEG ou PDF, facilitando a inclusão em documentos e apresentações. Além disso, eles podem ser personalizados de acordo com as necessidades específicas do usuário, incluindo ajustes de cores, estilos de linha, títulos e legendas.

Por meio do uso de gráficos estáticos, os analistas de dados podem comunicar de forma eficaz os resultados de suas análises, fornecendo uma representação visual clara e concisa dos insights extraídos dos dados. Isso ajuda a tornar a análise de dados mais acessível e compreensível para uma ampla audiência.

VISUALIZAÇÃO MÚLTIPLA DE VARIÁVEIS

Às vezes, é benéfico explorar as relações entre várias variáveis simultaneamente para obter uma compreensão mais completa do conjunto de dados. Uma maneira eficaz de fazer isso é por meio dos gráficos de matriz, também conhecidos como pairplots, disponíveis no Seaborn. Nesses gráficos, cada combinação de variáveis é representada em um gráfico separado, permitindo uma análise visual rápida das relações entre elas.

Esses pairplots são particularmente úteis para identificar padrões, tendências e correlações entre as variáveis. Ao visualizar múltiplas combinações de variáveis em uma matriz de gráficos, os usuários podem detectar facilmente associações e entender a estrutura dos dados de maneira abrangente.

Por exemplo, ao examinar um conjunto de dados que contém informações sobre características físicas de indivíduos, um pairplot pode revelar a relação entre altura e peso, bem como a distribuição conjunta de outras variáveis, como idade e índice de massa corporal.

Dessa forma, os gráficos de matriz no Seaborn oferecem uma ferramenta poderosa para explorar e entender as relações entre variáveis em um conjunto de dados, auxiliando na identificação de insights e padrões relevantes para a análise.Observe a criação básica de alguns tipos de gráficos.

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

# Exemplo de gráfico de linhas
sns.lineplot(x='tempo', y='vendas', data=df)
plt.title('Vendas ao longo do tempo')

# Exemplo de gráfico de barras
sns.barplot(x='equipe', y='pontuação', data=df)
plt.title('Pontuação por equipe')

# Exemplo de gráfico de dispersão
sns.scatterplot(x='idade', y='salário', data=df)
plt.title('Relação entre idade e salário')
```

```python
# Exemplo de histograma
sns.histplot(x='idade', data=df)
plt.title('Distribuição de idades')

# Exemplo de gráfico de pizza
plt.pie(x=df['categoria'].value_counts(),
        labels=df['categoria'].unique())
plt.title('Proporção de categorias')

# Exemplo de gráfico de matriz (pairplot)
sns.pairplot(data=df, vars=['idade', 'salário',
                            'tempo_emprego'])
plt.suptitle('Relações entre variáveis')
```

DASH E PLOTLY

Dash é uma estrutura de desenvolvimento de código aberto, criada pela equipe do Plotly, que facilita a criação de aplicativos da web interativos e personalizáveis para análise de dados e visualização. Enquanto isso, o Plotly é uma biblioteca de visualização de dados em Python que oferece uma ampla gama de gráficos interativos e altamente customizáveis.

Essa integração entre Dash e Plotly permite aos desenvolvedores criar aplicativos da web que incluem visualizações de dados dinâmicas e interativas, fornecendo uma experiência imersiva aos usuários. Com Dash, é possível criar interfaces de usuário complexas e intuitivas, combinando elementos como gráficos, tabelas, botões e controles deslizantes, tudo em um ambiente web.

O Plotly, por sua vez, oferece uma variedade de gráficos interativos, incluindo gráficos de dispersão, gráficos de linhas, gráficos de barras, mapas, entre outros. Esses gráficos podem ser altamente customizados em termos de estilo, cores, layouts e interatividade, permitindo aos usuários explorar e visualizar dados de maneira dinâmica.

Dessa forma, Dash e Plotly se complementam, proporcionando uma plataforma robusta e flexível para a criação de aplicativos da web interativos e visualizações de dados personalizadas em Python. Essa combinação poderosa é amplamente utilizada em diversos campos, incluindo ciência de dados, análise financeira, visualização de dados e muito mais.

INTRODUÇÃO AO DASH E PLOTLY

O Dash é uma estrutura de desenvolvimento construída sobre o Plotly, oferecendo aos desenvolvedores a capacidade de criar aplicativos da web interativos utilizando Python. O Plotly, por sua vez, é uma biblioteca de visualização de dados que proporciona uma extensa variedade de gráficos interativos, abrangendo desde gráficos de dispersão e gráficos de linhas até gráficos de barras e mapas, entre outros tipos de visualizações.

Essa integração entre o Dash e o Plotly permite que os desenvolvedores criem aplicativos da web que apresentem visualizações de dados dinâmicas e envolventes, proporcionando aos usuários uma experiência interativa ao explorar e analisar os dados. Com o Dash, é possível criar interfaces de usuário complexas e altamente personalizadas, combinando elementos como gráficos, tabelas e controles de entrada, enquanto o Plotly oferece a capacidade de criar visualizações de dados poderosas e personalizadas que podem ser incorporadas diretamente nos aplicativos da web. Essa

combinação de recursos torna o Dash uma ferramenta poderosa para a criação de aplicativos da web interativos e visualizações de dados dinâmicas em Python.

ESTRUTURA BÁSICA DE UM APP DASH

Um aplicativo Dash é construído com dois elementos essenciais: a estrutura de layout e os callbacks. A estrutura de layout é responsável por definir a aparência e a organização dos componentes visuais do aplicativo, como gráficos, tabelas e controles de entrada. Isso permite que os desenvolvedores personalizem a interface do usuário de acordo com as necessidades específicas do aplicativo, garantindo uma experiência de usuário agradável e intuitiva.

Por outro lado, os callbacks são responsáveis por definir a lógica por trás da interatividade do aplicativo. Eles determinam como os diferentes componentes do aplicativo devem responder às ações dos usuários, como cliques em botões, seleções em menus ou entrada de dados em campos de texto. Por meio dos callbacks, os desenvolvedores podem criar funcionalidades dinâmicas e interativas, permitindo que o aplicativo responda de forma inteligente às interações dos usuários.

Essa combinação de estrutura de layout e callbacks torna o Dash uma ferramenta poderosa para o desenvolvimento de aplicativos da web interativos e personalizáveis em Python. Ao aproveitar esses recursos, os desenvolvedores podem criar aplicativos da web sofisticados que oferecem uma experiência de usuário envolvente e eficaz.

COMPONENTES

O Dash disponibiliza uma ampla variedade de componentes para a construção de interfaces de usuário de aplicativos web. Esses componentes incluem gráficos Plotly interativos, controles de formulário, botões, caixas de seleção e muitos outros. Cada um desses componentes pode ser facilmente integrado na estrutura de layout do aplicativo, permitindo aos desenvolvedores criar interfaces ricas e funcionais.

Os gráficos Plotly fornecem uma forma poderosa de visualizar dados de maneira interativa, enquanto os controles de formulário permitem aos usuários interagir com os dados e ajustar parâmetros conforme necessário. Os botões são úteis para acionar ações específicas no aplicativo, enquanto as caixas de seleção oferecem opções de escolha para os usuários.

Com essa variedade de componentes à disposição, os desenvolvedores têm a flexibilidade necessária para criar interfaces de usuário que atendam às necessidades específicas do aplicativo, proporcionando uma experiência de usuário intuitiva e eficiente. Essa capacidade de personalização é fundamental para o sucesso de qualquer aplicativo Dash, permitindo que ele seja adaptado para uma ampla gama de casos de uso e requisitos.

CALL-BACKS

Os callbacks no Dash são funções Python que são acionadas em resposta a eventos específicos, como a alteração de um controle de formulário ou o clique em um botão. Essas funções são essenciais para criar interatividade entre os diferentes componentes do aplicativo e são responsáveis por atualizar dinamicamente a saída com base nas interações do usuário.

Por exemplo, um callback pode ser configurado para atualizar um gráfico Plotly sempre que um usuário selecionar uma nova opção em uma caixa de seleção. Da mesma forma, outro callback pode ser usado para atualizar uma tabela de dados quando um usuário enviar um novo valor através de um controle de entrada.

Essa capacidade de resposta dinâmica às interações do usuário é fundamental para a criação de aplicativos web interativos e envolventes. Os callbacks permitem que os desenvolvedores criem experiências de usuário personalizadas e adaptáveis, garantindo que os aplicativos Dash atendam às necessidades específicas dos usuários de forma eficaz e eficiente.

PERSONALIZAÇÃO

Tanto o Dash quanto o Plotly disponibilizam uma ampla variedade de opções de personalização para gráficos e layouts de aplicativos. Essas opções incluem ajustes de estilo, cores, legendas, títulos e muitos outros atributos visuais.

Com o Dash, os desenvolvedores têm a liberdade de personalizar a aparência e o comportamento dos componentes do aplicativo, como gráficos e controles de formulário, utilizando parâmetros específicos. Isso permite que os aplicativos Dash sejam adaptados para atender às necessidades visuais e funcionais de cada projeto.

Por outro lado, o Plotly oferece uma série de ferramentas e métodos para personalizar gráficos de forma detalhada. Isso inclui a capacidade de ajustar cores, estilos de linha, marcadores, rótulos, legendas e muito mais. Com essas opções de personalização, os desenvolvedores podem criar visualizações de dados altamente sofisticadas e esteticamente agradáveis que se alinham com as exigências de seus projetos específicos.

Em resumo, tanto o Dash quanto o Plotly oferecem recursos robustos de personalização que permitem aos desenvolvedores criar aplicativos e visualizações de dados que são únicos e adaptados às necessidades de cada projeto. Essa flexibilidade é fundamental para garantir que os aplicativos e visualizações produzidos atendam aos mais altos padrões de qualidade e estética.

INTEGRAÇÃO COM PANDAS

A integração entre Dash e Pandas facilita a análise de dados em aplicativos web. Os dados podem ser carregados diretamente em DataFrames Pandas e, em seguida, utilizados para gerar gráficos interativos e alimentar os diversos componentes do aplicativo Dash.

Essa integração simplifica significativamente o processo de criação de aplicativos de análise de dados, permitindo que os desenvolvedores explorem

e visualizem conjuntos de dados de forma eficiente e intuitiva. Ao utilizar Pandas em conjunto com Dash, os desenvolvedores podem tirar proveito das poderosas funcionalidades de manipulação e análise de dados oferecidas pelo Pandas, enquanto aproveitam a flexibilidade e a interatividade proporcionadas pelo Dash para criar aplicativos web dinâmicos e responsivos.

DEPLOY

Após a criação de um aplicativo Dash, ele pode ser implantado em diferentes plataformas de hospedagem, como Heroku, AWS, Google Cloud, entre outros, para que ele possa ser acessado e usado por outros usuários.

Em resumo, Dash e Plotly são ferramentas poderosas para criação de aplicativos da web interativos e visualizações de dados, permitindo que os usuários explorem e interajam com seus dados de forma dinâmica e eficaz.

Observe o exemplo simples de um aplicativo Dash utilizando Plotly e permitindo que o usuário selecione um conjunto de dados e visualize em um gráfico de dispersão.

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.graph_objs as go


dados_conceitos = {
    'java': {'variaveis': 8, 'condicionais': 7, 'loops': 5,
             'poo': 3, 'funcoes': 9},
    'python': {'variaveis': 7, 'condicionais': 9, 'loops': 6,
               'poo': 4, 'funcoes': 8},
    'sql': {'variaveis': 6, 'condicionais': 8, 'loops': 7,
             'poo': 5, 'funcoes': 7},
    'golang': {'variaveis': 5, 'condicionais': 7, 'loops': 8,
               'poo': 6, 'funcoes': 6},
    'javascript': {'variaveis': 4, 'condicionais': 6,
                   'loops': 7, 'poo': 8, 'funcoes': 5}
}
```

```
cores_map = {
    'java': 'red',
    'python': 'blue',
    'sql': 'green',
    'golang': 'orange',
    'javascript': 'purple'
}
```

```
app = dash.Dash(__name__)
```

```
app.layout = html.Div([
    html.H1("Leonardo Alves",
            style={'textAlign': 'center'}),
    html.Div([
        dcc.Dropdown(
            id='dropdown-linguagens',
            options=[
                {'label': 'Java', 'value': 'java'},
                {'label': 'Python', 'value': 'python'},
                {'label': 'SQL', 'value': 'sql'},
                {'label': 'GoLang', 'value': 'golang'},
                {'label': 'JavaScript', 'value': 'javascript'}
            ],
            value=['java'],
            multi=True,
            style={'width': '50%', 'margin': '0 auto'}
        ),
    ]),
    dcc.Graph(id='scatter-plot')
], style={'width': '80%', 'margin': '0 auto'})
```

```
@app.callback(
    Output('scatter-plot', 'figure'),
    [Input('dropdown-linguagens', 'value')]
)
def update_scatter_plot(selected_languages):
    scatter_trace = []
    for language in selected_languages:
        dados_linguagem = dados_conceitos[language]
        for conceito, conhecimento in dados_linguagem.items():
            scatter_trace.append(go.Scatter(
                x=[conceito],
                y=[conhecimento],
                mode='markers',
                name=language.capitalize(),
                marker=dict(size=15, color=cores_map[language]
                showlegend=False
            ))
```

```
    scatter_layout = go.Layout(
        title=f'Nível de Conhecimento',
        xaxis=dict(title='Conceito', showgrid=False),
        yaxis=dict(title='Nível de Conhecimento', showgrid=Fal
    )
    return {'data': scatter_trace, 'layout': scatter_layout}

if __name__ == '__main__':
    app.run_server(debug=True)
```

PROJETO:

ANÁLISE DE DADOS E CONSTRUÇÃO DE UM DASHBOARD.

INTRODUÇÃO

O projeto desenvolvido em sala junto com o Professor. Consiste em uma aplicação web construída com o framework Dash, que permite a criação de dashboards interativos em Python. O principal objetivo desta aplicação é fornecer uma interface intuitiva para visualização e análise de dados de vendas de uma loja fictícia, denominada "Ebony Store ©". Utilizando dados de vendas, a aplicação oferece insights sobre os produtos mais vendidos, a rentabilidade dos produtos, o desempenho de diferentes categorias de produtos, entre outras análises relevantes.

OBJETIVO

O objetivo principal deste projeto é fornecer uma plataforma interativa e visualmente atraente para análise de dados de vendas da Ebony Store. Através desta aplicação, os usuários podem obter insights valiosos sobre o desempenho de vendas, identificar padrões e tendências, e tomar decisões informadas para impulsionar o sucesso do negócio.

APLICAÇÃO

A aplicação consiste em um dashboard dividido em diferentes seções, cada uma focada em aspectos específicos dos dados de vendas. Os principais recursos incluem visualizações dos produtos mais vendidos, análise de rentabilidade, distribuição de vendas por categoria, entre outros. Os usuários podem interagir com a aplicação selecionando diferentes filtros, como cliente, mês e categoria, para personalizar as visualizações de acordo com suas necessidades.

PRINCIPAIS TÓPICOS:

Produtos Mais Vendidos

Esta seção exibe os cinco principais produtos em termos de vendas totais, apresentando gráficos de barras e linhas para visualizar as vendas individuais de cada produto.

Rentabilidade dos Produtos

Aqui, os usuários podem explorar a rentabilidade dos produtos com base no preço unitário e na localização da loja. Gráficos de dispersão fornecem uma visão clara da relação entre preço e rentabilidade.

Desempenho por Categoria

Nesta seção, os usuários podem analisar a distribuição de vendas por categoria de produto, utilizando um gráfico de pizza para visualizar a participação de cada categoria nas vendas totais.

Indicadores de Desempenho

São apresentados indicadores numéricos que destacam o melhor produto e a melhor categoria em relação à média de vendas, oferecendo insights rápidos sobre o desempenho.

Clientes Principais

Esta seção destaca os cinco principais clientes em termos de volume de vendas, permitindo uma análise rápida dos principais clientes da loja.

Seleção de Filtros

A aplicação oferece opções de filtragem flexíveis, permitindo aos usuários selecionarem um cliente específico, mês ou categoria para visualizar dados específicos.

CORREÇÃO DO PROJETO

```
import dash
import dash_bootstrap_components as dbc
from dash import html, dcc, Input, Output
from dash_bootstrap_templates import ThemeSwitchAIO
from dash.dependencies import Input, Output
import plotly.graph_objects as go
import plotly.express as px
import pandas as pd
import calendar
```

```
font_awesome = ['https://use.fontawesome.com/releases/v5.10.2/
app = dash.Dash(__name__, external_stylesheets=font_awesome)
app.scripts.config.serve_locally = True
server = app.server
```

```
card_h100 = {'height': '100%'}

main_config = {
    'hovermode': 'x unified',
    'legend': {
        'yanchor': 'top',
        'y': 0.9,
        'xanchor': 'left',
        'x': 0.1,
        'title': {
            'text': None
        },
        'font': {
            'color': 'grey'
        },
        'bgcolor': '#101010',
    },
    'margin': {
        'l': 10,
        'r': 10,
        't': 10,
        'b': 10
    }}
```

```
graph_config = {
    'displayModeBar': False,
    'showTips': False}

dark_theme = 'darkly'
flatly_theme = 'flatly'
```

```
url_dark_theme = dbc.themes.DARKLY
url_flatly_theme = dbc.themes.FLATLY
df = pd.read_csv('../src/data/dataset_comp.csv')
df['dt_Venda'] = pd.to_datetime(df['dt_Venda'])
df_meses = df['dt_Venda'].dt.month.unique()
df_meses_txt = [calendar.month_name[month] for month in df_mes
options_client = [{'label': cliente, 'value': cliente} for cli
options_client.append({'label': 'Todos', 'value':'todos'})
options_month = [{'label': 'Ano', 'value': 0}]

for num_mes in df['dt_Venda'].dt.month.unique():
    nome_mes = calendar.month_name[num_mes]
    options_month.append({'label': nome_mes, 'value': num_mes}

options_month = sorted(options_month, key=lambda x: x['value']
category_options = [{'label': 'Todas', 'value': 0}]
for c in df['Categorias'].unique():
    category_options.append({'label': c, 'value': c})
```

```
def month_filter(month):
    if month == 0: mask = df['dt_Venda'].dt.month.isin(df['dt_
    else: mask = df['dt_Venda'].dt.month.isin([month])
    return mask

def cliente_filter(cliente_selected):
    if cliente_selected is None:
        return pd.Series(True, index=df.index)
    else:
        return df['Cliente'] == cliente_selected

def category_filter(category):
    if category == 0: mask = df['Categorias'].isin(df['Categor
    else: mask = df['Categorias'].isin([category])
    return mask

def convert_to_text(month):
    meses = ['Ano Completo', 'Janeiro', 'Fevereiro', 'Março',
    'Abril', 'Maio', 'Junho', 'Julho', 'Agosto', 'Setembro',
    'Outubro','Novembro', 'Dezembro']
    return meses[month]
```

```
app.layout = dbc.Container(children=[
    dbc.Row([
        dbc.Col([
                dcc.Dropdown(
                    id='cliente-dropdown',
                    options=options_client,
                    value=None,
                    placeholder='Selecione um cliente',
                    style={'background-color': 'transparent',
                           'border': 'none',
                           'margin-top': '5px'
                           ,'color' : 'green'},
                           className = 'dropdown-custom'
                ),
            ],
            align='left',
        ),
```

```
        dbc.Col([
            dbc.Row([
                dbc.Col([
                    html.Legend("Ebony Store ©",
                                style={
                                    'font-size': '150%',
                                    'text-align': 'center'}),
                        html.Legend(···
                        ]),
                    ]),
                ], sm=8, align='center'
                ),
        dbc.Col([
            dbc.Row([
                dbc.Col([
                    ThemeSwitchAIO(aio_id="theme",
                                   themes=[
                                       url_dark_theme,
                                       url_flatly_theme
                                       ])
                        ])
                    ]),
                ], align='center'
```

```
                )
            ],
        className='g-1 my-auto',
        style={'text-align': 'center'}
    ),
    dbc.Row([
        dbc.Col([
            dbc.Card(
                dbc.CardBody([
                    dbc.Row([
                        dbc.Col([
                            html.H6(
                                "Top5 Produtos mais vendidos."
                                style={"text-align": "center",
                                       'background-color': 'tr
                            )]),
                        ]),
                    dbc.Row([
                        dbc.Col([
                            dcc.Graph(
```

```
                        dcc.Graph(
                            id="graph1",
                            className="dbc",
                            config={"displayModeBar": Fals
                        )
                    ], style=card_h100),
                    dbc.Col([
                        dcc.Graph(
                            id="graph2",
                            className="dbc",
                            config={"displayModeBar": Fals
                        )
                    ],style=card_h100),
                ]),
            ]),
            style={'background-color': 'transparent','bord
        )]
    ),
    dbc.Col([
        dbc.Card(
            dbc.CardBody([
                dbc.Row([
                    dbc.Col([
```

```
                dbc.Col([
                    dbc.RadioItems(
                        id="radio-month",
                        options=[
                            {"label": "Ano",
                             "value": 0}]
                        + sorted([
                            {"label":
                             calendar.month_name[month
                             "value": month,}
                        for month in range(1, 13)],
                        key=lambda x: x["value"],),
                        value=0,
                        inline=True,
                        labelCheckedClassName="text-su
                        inputCheckedClassName="border
                        style={"text-align": "center",
                               "margin-top": "10%",
                               "margin-bottom": "15px"
                    ),
```

```
                    html.Div(id="month-select",
                             style={
                                 "text-align": "center
                                  "margin-top": "30px
                                  'margin-bottom': '5
                             className="dbc",
                    ),
                ]),
            ]),
            dbc.Col([
                dbc.Card([
                        dbc.RadioItems(
                            id="radio_category",
                            options=category_options,
                            value=0,
                            inline=True,
                            labelCheckedClassName="tex
                            inputCheckedClassName="bor
                            style={"text-align": "cent
                        ),
                    html.Div(
```

```
                        html.Div(
                            id='category-select',
                            style={
                                'text-align': 'center',
                                'margin-top': '30px'},
                            className='dbc'
                            ),],
                            style={
                                'background-color':
                                'transparent','border': 'n
                        )],
                    style={"height": "100%"}
                )
                ]),
                style={'background-color':
                    'transparent','border': 'none'}
            )
        ]),
    ],
    style={"margin-top": "20px"},
    className="g-0 my-auto",
),
```

```
dbc.Row([
    dbc.Col([
        dbc.Row([
            dbc.Col([
                dbc.Card([
                    dcc.Graph(id='graph3',
                              className='dbc', config=grapl
                ], style={'background-color':
                          'transparent','border': 'none'})
            ])
        ]),
        dbc.Row([
            dbc.Col([
                dbc.Card([
                    dcc.Graph(id='graph4',
                              className='dbc', config=grapl
                ], style={'background-color':
                          'transparent','border':
                          'none', 'margin-top':'15px'})
            ])
        ], className='g-3 my-auto', style={'margin-top': '
    ], sm=12, lg=5),
```

```
        dbc.Col([
            dbc.Row([
                dbc.Col([
                    dbc.Card([
                        dcc.Graph(id='graph5',
                                  className='dbc',
                                  config=graph_config)
                    ], style={'padding': '10px','border': 'non
                ], sm=6),
                dbc.Col([
                    dbc.Card([
                    dcc.Graph(id='graph6',
                              className='dbc',
                              config=graph_config)
                    ], style={'padding': '10px',
                              'background-color':
                              'transparent','border': 'none'})
                ], sm=6)
            ], className='g-10',style={'margin-top':
                                       '7px','border': 'none'}
            dbc.Row([
                dbc.Col([
```

```
            dbc.Row([
                dbc.Col([
                    dbc.Card([
                        dcc.Graph(id='graph7',
                                  className='dbc',
                                  config=graph_config)
                    ], style={'background-color':
                              'transparent','border': 'none'})
                ])
            ], className='g-10 my-auto', style={
                'margin-top': '7px','border': 'none'})
        ], sm=12, lg=4),
        dbc.Col([
            dcc.Graph(id='graph8', className='dbc',
                      config=graph_config)
        ], sm=12, lg=3)
    ], className='g-6 my-auto', style={'margin-top':
                                       '7px','border':
                                       'none'}),],
    fluid=True,
    style={'height': '100vh'})
```

```
@app.callback(
    [Output('graph1', 'figure'),
    Output('graph2', 'figure'),
    Output('month-select', 'children')],
    [Input('radio-month', 'value'),
    Input('cliente-dropdown', 'value'),
    Input(ThemeSwitchAIO.ids.switch("theme"), "value")]
)
```

```
def graph1(month, client, toggle):
    template = dark_theme if toggle else flatly_theme
    mask_month = month_filter(month)
    mask_client = cliente_filter(client)
    mask = mask_client & mask_month
    df_1 = df.loc[mask]
    df_1_grouped = df_1.groupby(['Produto', 'Categorias'])['To
    df_1_top5 = df_1_grouped.sort_values(by='Total Vendas', as
    fig1 = px.bar(
    df_1_top5,
    x='Produto',
    y='Total Vendas',
    text='Total Vendas',
    color='Total Vendas',
    color_continuous_scale='greens',
    height=280,
    template=template,
    )
    fig1.update_traces(texttemplate='%{text:.2s}', textpositio
    fig1.update_layout(
```

```
        xaxis=dict(showgrid=False),
        yaxis=dict(showgrid=False, range=[df_1_top5['Total Ven
        Vendas'].max() * 1.1]),
        xaxis_title=None,
        yaxis_title=None,
        xaxis_tickangle=-15,
        font=dict(size=13),
    )
    fig2 = go.Figure(go.Scatter(
    x=df_1_top5['Produto'],
    y=df_1_top5['Total Vendas'],
    mode='lines+markers',
    line=dict(color='green')
    ))
    fig2.update_layout(
        main_config,
        title_text=None,
        height=60,
        xaxis_title=None,
        yaxis_title=None,
```

```
        yaxis=dict(showgrid=False, showticklabels=False),
        template=template
    )
    select = html.H2(convert_to_text(month))
    return fig1, fig2, select

@app.callback(
    Output('graph3', 'figure'),
    [Input('radio_category', 'value'),
    Input('cliente-dropdown', 'value'),
    Input('radio-month', 'value'),
    Input(ThemeSwitchAIO.ids.switch("theme"), "value")]
)
def graph3(category, client, month, toggle):
    template = dark_theme if toggle else flatly_theme
    mask_catetogy = category_filter(category)
    mask_client = cliente_filter(client)
    mask_month = month_filter(month)
    mask = mask_catetogy & mask_client & mask_month
    df_3 = df.loc[mask]
    df_3['dt_Venda'] = pd.to_datetime(df_3['dt_Venda'])
```

```
df_3_grouped = df_3.groupby(df_3['dt_Venda'].dt.month)
['Quantidade'].sum().reset_index()
fig3 = go.Figure(
    go.Scatter(
        x=df_3_grouped['dt_Venda'],
        y=df_3_grouped['Quantidade'],
        mode='lines',
        fill='tonexty',
        line=dict(color='#007d11'),
        marker=dict(
            color='#007d11',
            line=dict(color='#aafb8d')
        ,),
        fillcolor='rgba(0, 125, 17, 0.3)',
        hoverinfo='text+x+y',
        hovertemplate='<b>Mês</b>: \
        %{x}<br><b>Quantidade</b>: %{y}<br><b> \
        Total de Vendas</b>: %{text}',
```

```
            text=[(df_3[df_3['dt_Venda'].dt.month == mes]
                   ['Preço Unitário'] * df_3[df_3['dt_Venda']
                    .dt.month == mes]['Quantidade']).sum()
                    for mes in df_3_grouped['dt_Venda']]
            )
        )
    fig3.update_yaxes(range=[df_3_grouped['Quantidade']
        .min()-1000, df_3_grouped['Quantidade'].max()+1000])
    total_vendas = (df_3['Preço Unitário'] *
                    df_3['Quantidade']).sum()
    fig3.add_annotation(
        text = f'Vendas Ano: R$ {total_vendas}',
        xref='paper',
        yref='paper',
        font=dict(
            size=12,
            color='#8afba6'
        ),
        align= 'center',
        x=0.01, y=1,
        showarrow=False
```

```
    fig3.update_layout(main_config, height=180, template=templ
    return fig3

def calcular_rentabilidade(custo, localizacao):
    porcentagem_base = 0.03
    if 100 <= custo < 1000:porcentagem_base = 0.05
    elif 1000 <= custo < 10000:porcentagem_base = 0.08
    elif custo >= 10000:porcentagem_base = 0.12
    porcentagem_adicional = 0.0
    if localizacao == 'São Paulo':porcentagem_adicional = 0.05
    elif localizacao == 'Rio de Janeiro':porcentagem_adicional
    elif localizacao == 'Salvador':porcentagem_adicional = 0.1
    elif localizacao == 'Três Rios':porcentagem_adicional = 0.
    elif localizacao == 'Santos':porcentagem_adicional = 0.14
    elif localizacao == 'Salvador':porcentagem_adicional = 0.3
    custo_total = (custo * porcentagem_base) + (custo * porcen
    return custo - custo_total
```

```
@app.callback(
    Output('graph4', 'figure'),
    [Input('radio_category', 'value'),
    Input('cliente-dropdown', 'value'),
    Input('radio-month', 'value'),
    Input(ThemeSwitchAIO.ids.switch("theme"), "value")]
)
```

```
def graph4(category, client, month, toggle):
    template = dark_theme if toggle else flatly_theme
    mask_catetogy = category_filter(category)
    mask_client = cliente_filter(client)
    mask_month = month_filter(month)
    mask = mask_catetogy & mask_client & mask_month
    df_4 = df.loc[mask]
    df_4['Rentabilidade'] = df_4.apply(
        lambda row: calcular_rentabilidade(row['Preço Unitário
    )
    fig4 = px.scatter(
    df_4,
    x='Preço Unitário',
    y='Rentabilidade',
    color='Preço Unitário',
    labels={'Rentabilidade': 'Rentabilidade (%)', 'Preço Unitá
    color_continuous_scale='greens',
    )
    fig4.update_layout(main_config, height=180, template=templ
    return fig4
```

```
@app.callback(
    Output('graph5', 'figure'),
    Output('graph6', 'figure'),
    Input('radio-month', 'value'),
    Input(ThemeSwitchAIO.ids.switch("theme"), "value")
)
```

```python
def graph5(month,toggle):
    template = dark_theme if toggle else flatly_theme
    mask_month = month_filter(month)
    df_5 = df_6 = df.loc[mask_month]
    df_5 = df.groupby(['Produto', 'Categorias'])['Total Vendas
    df_5 = df_5.sort_values(ascending=False).reset_index()
    df_6 = df.groupby(['Categorias'])['Total Vendas'].sum()
    df_6 = df_6.sort_values(ascending=False).reset_index()
    fig5 = go.Figure()
    fig5.add_trace(go.Indicator(
        mode = 'number+delta', # nome e referência
        title ={
            'text':
                f"<span style='font-size:90%'>"
                f"Melhor Produto"
                f"</span><br>"
                f"<span style='font-size:90%'>"
                f"{df_5['Produto'].iloc[0]}"
                f"</span><br>"
                f"<span style='font-size:50%'>"
                f"...em relação a média."
                f"</span><br>"
        },
```

```
        value = df_5['Total Vendas'].iloc[0],
        number = {'prefix': 'R$'},
        delta = {
            'relative': True, 'valueformat': '.1%', 'reference
        }
    ))

    fig6 = go.Figure()
    fig6.add_trace(go.Indicator(
        mode = 'number+delta', # nome e referência
        title ={
            'text':
                f"<span style='font-size:100%'>"
                f"Melhor Categoria"
                f"</span><br>"
                f"<span style='font-size:90%'>"
                f"{df_6['Categorias'].iloc[0]}"
                f"</span><br>"
                f"<span style='font-size:70%'>"
                f"...em relação a {df_6['Categorias'].iloc[1]}
```

```
                f"</span><br>"
        },
        value = df_6['Total Vendas'].iloc[0],
        number = {'prefix': 'R$'},
        delta = {
            'relative': True, 'valueformat': '.1%', 'reference
        }
    ))

    fig5.update_layout(main_config, height=200, template=templ
    fig6.update_layout(main_config, height=200, template=templ
    fig5.update_layout({"margin": {"l":0, "r":0, "t":80, "b":0
    fig6.update_layout({"margin": {"l":0, "r":0, "t":80, "b":0
    fig5.update_layout(
    plot_bgcolor='rgba(0,0,0,0)',
    paper_bgcolor='rgba(0,0,0,0)')
    return fig5, fig6
```

```
@app.callback(
    Output('graph7', 'figure'),
    Input(ThemeSwitchAIO.ids.switch("theme"), "value")
)

def graph7(toggle):
    template = dark_theme if toggle else flatly_theme
    df_7 = df.groupby('Cliente')['Total Vendas'].sum().reset_i
    df_7 = df_7.sort_values(by=['Total Vendas', 'Cliente'], as
    df_7 = df_7.head(5)
    fig7 = go.Figure()
    fig7.add_trace(go.Bar(
        x=df_7['Total Vendas'],
        y=df_7['Cliente'],
        orientation='h',
        marker=dict(color='rgba(0, 125, 17, 0.3)'),
        text=df_7['Total Vendas'],
        textfont=dict(color='white'),
        textposition='auto'
    ))
```

```
    fig7.update_layout(
        main_config,
        yaxis={'title': 'Top5  Clientes'},
        height=110,
        template=template,
        showlegend=False
    )

    colors = ['rgba(0, 0, 255, 0.7)' if cliente == df_7['Clien
    ' for cliente in df_7['Cliente']]
    fig7.update_layout(main_config, xaxis={'title':None}, heig

@app.callback(
    Output('graph8', 'figure'),
    [Input('radio-month', 'value'),
    Input('cliente-dropdown', 'value'),
    Input(ThemeSwitchAIO.ids.switch("theme"), "value")]
)
```

```
def graph8(month, client, toggle):
    template = dark_theme if toggle else flatly_theme
    mask_month = month_filter(month)
    mask_client = cliente_filter(client)
    mask = mask_month & mask_client
    df_8 = df.loc[mask]
    df_8 = df_8.groupby('Categorias')['Quantidade'].sum().rese
    df_8 = df_8.sort_values(by='Quantidade', ascending=True).h
    fig8 = px.pie(
        df_8,
        values='Quantidade',
        names='Categorias',
        title='Distribuição de Vendas por Categoria',
        labels={'Categorias': 'Quantidade'},
        hole=0.7
    )
    fig8.update_traces(textposition='outside',
                       textinfo='percent+label',
                       pull=[0.05, 0.05, 0.05, 0.05, 0.05])

    fig8.update_layout(main_config, height=360, template=templ
    fig8.update_layout(showlegend=False)
    fig8.update_layout(
```

```
    title=dict(text='Distribuição de Vendas por Categoria',
               x=0.5, y=0.92, font=dict(size=20)),
    margin=dict(t=50, b=50),
    height=400,
    )
    return fig8

if __name__ == '__main__':
    app.css.append_css({'external_url': 'src/css/style.css'})
    app.run_server(debug=True, port=8051)
```

CONCLUSÃO

Uma das principais razões para a popularidade do Python na análise de dados é a presença de bibliotecas especializadas, como Pandas e NumPy. Pandas oferece estruturas de dados flexíveis e poderosas, como o DataFrame, que simplifica a manipulação e análise de conjuntos de dados tabulares. NumPy, por sua vez, fornece suporte para arrays multidimensionais e uma ampla gama de funções matemáticas, tornando-o essencial para cálculos numéricos e científicos.

Além disso, as bibliotecas de visualização, como Matplotlib e Seaborn, permitem a criação de gráficos esteticamente atraentes e informativos, que ajudam na compreensão dos dados e na comunicação de insights de forma eficaz. Essas ferramentas permitem a criação de uma variedade de gráficos, desde simples gráficos de linha e barras até visualizações mais complexas, como mapas de calor e gráficos de dispersão.

Para além da análise estática, Python também oferece recursos para criação de aplicativos da web interativos para análise de dados em tempo real. Dash e Plotly são exemplos de bibliotecas que possibilitam a criação de aplicativos da web interativos e personalizáveis, permitindo que os usuários explorem dados de forma dinâmica e intuitiva.

A versatilidade do Python na análise de dados é ainda mais amplificada pela sua integração com outras tecnologias e linguagens, tornando-o uma escolha preferida para cientistas de dados, analistas e pesquisadores em uma variedade de campos, incluindo ciência, negócios, finanças, saúde e muito mais.

Em resumo, Python oferece um conjunto abrangente de ferramentas e recursos que tornam a análise de dados acessível e eficaz para uma ampla gama de usuários. Sua simplicidade, flexibilidade e poder tornam-no uma escolha ideal para aqueles que desejam explorar, entender e comunicar insights valiosos a partir de conjuntos de dados complexos.