Python para dados: do zero à análise completa

Asimov Academy

ASIMOV

Conteúdo

01. Bem-Vindo

O Cenário Atual de Informações

No mundo contemporâneo, estamos constantemente bombardeados por informações. Dados surgem de diversas fontes:

- Internet
- Computadores
- Dispositivos móveis
- Redes sociais
- Comunicações corporativas

Três Possíveis Abordagens

Diante deste fluxo constante de informações, existem três estratégias possíveis:

1. **Ficar à Margem**: Ignorar o potencial dos dados, desperdiçando seu valor
2. **Mergulhar sem Preparo**: Navegar de forma aleatória e sem direcionamento
3. **Navegar com Precisão**: Utilizar análise de dados como ferramenta de compreensão

A Importância da Visão para Dados

Análise de dados não é apenas manipular números, mas transformar informações brutas em conhecimento significativo. Como aprender a ler transforma símbolos em histórias, a análise de dados transforma números em insights.

Características de um Analista de Dados

Um analista de dados:

- Extrai valor onde outros veem apenas números
- Desenvolve capacidade de identificar padrões
- Transforma dados complexos em narrativas compreensíveis

Por que Python?

Vantagens para Análise de Dados

Python destaca-se como linguagem de análise de dados por:

- Ser simples e intuitiva
- Possuir bibliotecas especializadas em análise
- Permitir versatilidade além da análise de dados

Aplicações Abrangentes

Com Python, é possível:

- Analisar dados
- Criar automações
- Desenvolver dashboards
- Integrar com APIs
- Trabalhar com inteligência artificial

Projeto Prático: Análise de Dados Reais

O curso utilizará dados do Portal da Transparência do Brasil, oferecendo:

- Experiência com conjunto de dados reais
- Resolução de problemas complexos
- Aprendizado contextualizado

Compromisso do Curso

- Ensinar análise de dados de forma prática
- Desenvolver habilidades aplicáveis
- Transformar informação em conhecimento
- Mostrar o potencial da programação

Motivação

A programação não é apenas uma habilidade técnica, mas uma ferramenta de:

- Desenvolvimento profissional
- Crescimento pessoal
- Aumento de eficiência
- Resolução criativa de problemas

02. Como vamos aprender

Metodologia de Ensino

A metodologia de ensino deste curso difere significativamente de abordagens tradicionais de programação. Enquanto muitos cursos fragmentam o aprendizado, apresentando conceitos isolados e projetos desinteressantes, esta formação propõe uma estratégia diferente: aprendizado baseado em projetos práticos.

O Problema das Metodologias Tradicionais

Cursos convencionais de programação frequentemente:

- Apresentam conceitos de forma isolada e desconectada
- Utilizam projetos pedagógicos pouco estimulantes
- Dificultam a compreensão de como diferentes elementos se interconectam
- Provocam desmotivação nos estudantes

Dados que Revelam a Ineficácia Traditional

Um exemplo ilustrativo vem de um curso popular de programação no YouTube:

- 8 milhões de alunos iniciaram o curso
- Apenas 180 mil (aproximadamente 3%) concluíram
- Tempo total desperdiçado: estimado em 5.132 anos

Nossa Abordagem: Aprendizado Prático e Significativo

Este curso foi desenvolvido com princípios fundamentais:

- Foco em projetos práticos e aplicáveis
- Introdução de conceitos conforme necessidade do projeto
- Objetivo de tornar o aprendizado eficiente e motivador

Objetivo do Curso

O curso visa demonstrar como Python pode ser:

- Uma ferramenta para aumentar eficiência

- Uma linguagem versátil para resolução de problemas
- Uma habilidade para automatizar tarefas repetitivas

Escopo de Aprendizado

Ao final do curso, o estudante será capaz de:

- Desenvolver projetos práticos de análise de dados
- Compreender conceitos de programação de forma contextualizada
- Aplicar Python em situações reais do cotidiano

Compromisso

O instrutor assume o compromisso de:

- Explicar conceitos de forma clara e aplicável
- Mostrar a programação como uma ferramenta interessante
- Respeitar o tempo e o esforço do estudante

O estudante é convidado a assumir o compromisso de:

- Manter-se engajado no processo de aprendizagem
- Praticar os conceitos apresentados
- Desenvolver projetos propostos

03. Para quem é este curso

Objetivo do Curso

Este curso foi concebido para pessoas que desejam:

- Transformar informações em conhecimento
- Aprender programação do zero
- Utilizar análise de dados como ferramenta profissional

Pré-requisitos

Para participar, você precisa apenas de:

- Um computador
- Vontade de aprender
- Nenhum conhecimento prévio de programação

Por que Aprender Programação?

Desenvolvimento Pessoal

Aprender a programar desenvolve habilidades fundamentais:

1. **Resolução de Problemas**
    - Capacidade de analisar questões de forma lógica e estruturada
    - Pensamento sistêmico e algorítmico
2. **Aprendizado Contínuo**
    - Adaptação constante a novas tecnologias
    - Mentalidade de atualização permanente

Limitações das Ferramentas Tradicionais

Restrições de Ferramentas Low-Code e No-Code

Ferramentas como Excel e Power BI apresentam limitações:

- Funcionalidades predefinidas

- Impossibilidade de customização avançada
- Restrições em integrações específicas

Vantagens de Python

Python oferece:

- Customização ilimitada
- Integração com diversas plataformas
- Gratuidade
- Flexibilidade para criar soluções personalizadas

Por que Escolher Python?

Características Principais

1. **Facilidade de Aprendizado**
    - Sintaxe simples e intuitiva
    - Código próximo à linguagem natural
    - Curva de aprendizado suave
2. **Versatilidade**
    - Automações
    - Dashboards
    - Aplicações web
    - Análise de dados
    - Inteligência artificial
    - Machine learning
3. **Aplicações Profissionais**
    - Linguagem principal em análise de dados
    - Ideal para profissionais que buscam usar programação como ferramenta

Público-Alvo Ideal

O curso é recomendado para:

- Analistas de dados

- Profissionais que trabalham com planilhas
- Estudantes
- Pesquisadores
- Pessoas interessadas em transformar dados em insights

Considerações Finais

Python não é a melhor opção para:

- Desenvolvimento web complexo
- Criação de sites grandes

Para desenvolvimento web, recomenda-se:

- HTML
- CSS
- JavaScript
- Frameworks modernos (React, Angular)

Convite à Jornada

Este curso representa o primeiro passo em uma jornada de aprendizado em programação e análise de dados. Python será sua ferramenta para transformar informações em conhecimento valioso.

Prepare-se para uma experiência de aprendizado transformadora!

04. Por que “aprender a aprender”

O Desafio do Aprendizado em Programação

Aprender programação envolve múltiplos desafios:

- Compreensão de lógica de programação
- Resolução de problemas complexos
- Interpretação de mensagens de erro
- Acompanhamento de mudanças tecnológicas constantes

Importância da Mentalidade Correta

A abordagem mental é crucial para:

- Manter a motivação
- Superar obstáculos
- Evitar desistência
- Potencializar crescimento pessoal e profissional

Deficiências no Sistema Educacional Tradicional

Problemas Identificados

A educação tradicional frequentemente:

- Foca no “o que” aprender
- Negligencia o “como” aprender
- Não desenvolve estratégias de aprendizagem efetivas
- Desperdiça potencial humano

Analogia Educacional

É como ensinar futebol sem explicar as regras:

- Possível aprender por tentativa e erro
- Processo ineficiente e frustrante
- Alto risco de desistência

Benefícios de Aprender a Aprender

Desenvolvimento Interno

1. **Confiança**
    - Segurança para enfrentar novos desafios
    - Convicção na capacidade de aprendizado
2. **Mentalidade de Crescimento**
    - Adaptabilidade
    - Resiliência
    - Superação de limitações iniciais

Desenvolvimento Externo

1. **Velocidade de Aprendizado**
    - Domínio mais rápido de novas técnicas
    - Destacar-se em ambientes profissionais
2. **Características Pessoais Desenvolvidas**
    - Maior concentração
    - Autoconfiança
    - Preparação para mudanças futuras

Objetivos do Módulo

Tópicos a Serem Abordados

- Estímulos necessários para fixação de conhecimento
- Técnicas eficazes de aprendizagem
- Estratégias de estudo
- Mentalidade de aprendizagem contínua

Conclusão

Aprender a aprender transcende a programação. É uma habilidade essencial para:

- Crescimento pessoal
- Desenvolvimento profissional
- Adaptação constante
- Superação de desafios

05. O que, como e por quê

Mito do Programador Nato

Muitas pessoas questionam sua capacidade de aprender programação. A dúvida persistente é: “Serei capaz de aprender a programar?”. A resposta é definitiva: sim, desde que haja motivações sólidas.

Quebrando Paradigmas

Contrariando crenças populares, não existem pessoas divididas entre “deuses olímpicos” capazes de programar e “meros mortais” incapazes. A realidade é muito mais complexa:

- Pessoas inteligentes e com recursos não conseguem aprender programação
- Indivíduos com poucos recursos e aparentes limitações são capazes de dominar a programação

O Fator Decisivo: Motivação

O elemento crucial para aprender programação não está em:

- Recursos disponíveis
- Suposta inteligência
- Afinidade prévia com tecnologia

O verdadeiro diferencial é a **motivação**.

O Círculo Dourado de Simon Sinek

Estrutura de Decisão

A maioria das pessoas toma decisões seguindo uma sequência externa:

1. **O Quê**: Objetivo material e concreto
2. **Como**: Método para alcançar
3. **Por Quê**: Motivação profunda

Exemplo Comum

- **O Quê**: Formar-se em engenharia
- **Como**: Fazer vestibular e estudar
- **Por Quê**: Razões superficiais ou impostas externamente

A Importância do "Por Quê"

O "por quê" reside no campo:

- Emocional
- Sentimentos
- Aspirações pessoais

Encontrando Sua Motivação Verdadeira

Três Princípios Fundamentais

1. **Não Confunda Motivação com Resultado**
    - Ganhar dinheiro é um resultado, não uma motivação
    - Questione repetidamente "por quê" até encontrar a motivação essencial
2. **Valide Sua Própria Motivação**
    - Não adote motivações de outras pessoas
    - Reflita internamente sobre suas crenças
    - Seja crítico com influências externas
3. **Busque Motivações Altruístas**
    - Motivações que envolvem outras pessoas são mais poderosas
    - Compromissos que ultrapassam interesses pessoais geram energia adicional

Transformando Motivação em Ação

Características de Pessoas Inspiradoras

Pessoas verdadeiramente inspiradoras:

- Têm clareza sobre seu propósito
- Comunicam seu "por quê" com facilidade
- Demonstram comprometimento profundo

Reflexão Pessoal

Perguntas para descobrir sua motivação:

- O que te faz levantar todos os dias?
- Quais são seus sonhos mais profundos?
- Que impacto deseja causar no mundo?

Conclusão

Aprender programação transcende habilidades técnicas. O verdadeiro combustível para o aprendizado é uma motivação autêntica, nascida de dentro para fora, capaz de superar qualquer obstáculo.

Sua jornada de aprendizado começa com a compreensão profunda do seu “por quê”.

06. Análise de dados para revisão

Neurônios e Conexões Cerebrais

Baseado no livro *Aprendendo a Aprender* (Daniel Dall'Acqua) e no curso homônimo do Coursera. O cérebro possui bilhões de neurônios que se comunicam via sinapses (impulsos elétricos). O aprendizado ocorre pela criação de novas conexões neurais. Inicialmente essas conexões são fracas e se fortalecem com prática e repetição.

Um indicador de consolidação de aprendizado é conseguir explicar um conceito sem necessidade de consulta - como explicar uma função em Python, por exemplo.

Modos de Operação do Cérebro

Modo Foco

O modo foco é ativado durante momentos de concentração intensa, como:

- Estudo
- Leitura
- Resolução de problemas

Caracteriza-se por exigir esforço ativo e raciocínio lógico.

Modo Difuso

O modo difuso ocorre em momentos de relaxamento, como:

- Divagação
- Períodos criativos

Sua principal função é consolidar memórias de longo prazo e fixar aprendizados. Uma analogia útil é comparar o modo foco com treino na academia e o modo difuso com o período de descanso necessário para evitar "lesões" mentais.

Tipos de Memória

Memória de Trabalho (Curto Prazo)

Responsável por armazenar informações temporárias, como um número ditado momentos antes de ser anotado. Possui capacidade limitada, sendo facilmente impactada por distrações como notifi-

cações e múltiplas abas abertas.

Memória de Longo Prazo

Armazena conhecimentos consolidados, permitindo o acesso a informações após dias ou anos. É reforçada por:

- Revisão constante
- Prática frequente
- Retomada ativa do conteúdo

Características do Aprendizado

O processo de aprendizado não é linear. Barreiras e frustrações são etapas normais, especialmente em campos complexos como programação.

Estratégias para lidar com essas características:

- Fazer intervalos para ativar o modo difuso e “digerir” o conteúdo
- Revisar conceitos com a mente descansada
- Compreender os ciclos de aprendizado, sem esperar progresso constante

Recapitulação

Pontos fundamentais para um aprendizado eficiente:

- Alternar entre modo foco (concentração) e modo difuso (consolidação)
- Criar um ambiente livre de distrações para memória de trabalho
- Desenvolver técnicas de repetição espaçada para memória de longo prazo
- Encarar obstáculos como parte natural do processo de aprendizagem

07. Análise de dados para revisão

Introdução aos Hábitos Humanos

O ser humano é naturalmente um ser de hábitos. Diariamente, as pessoas tendem a seguir rotinas estabelecidas, realizando ações de forma quase automática, como chegar em casa e guardar objetos no mesmo lugar ou sentar-se no mesmo local de trabalho.

Os hábitos representam momentos de piloto automático, nos quais as ações são executadas sem necessidade de reflexão consciente. Esse mecanismo é uma estratégia cerebral para conservação de energia, permitindo que o cérebro economize recursos cognitivos em tarefas repetitivas.

Estrutura de Formação dos Hábitos

Um hábito é composto por quatro etapas fundamentais:

1. Estímulo ou Gatilho

O gatilho é o elemento inicial que desencadeia o hábito. Pode ser uma notificação no celular, uma vibração ou qualquer outro elemento que induza uma ação específica. Por exemplo, a notificação de uma rede social pode levar alguém a acessá-la instantaneamente.

2. Rotina

A rotina representa a ação efetivamente realizada durante o hábito. Pode ser uma ação física como olhar Instagram, assistir YouTube ou uma ação mental, como mudar de humor ao ver determinada pessoa.

3. Recompensa

Toda vez que um hábito é executado, o cérebro recebe um estímulo prazeroso. Mesmo que momentâneo, esse estímulo reforça a probabilidade de repetição do hábito, criando um ciclo de consolidação.

4. Crença

A formação de um hábito depende diretamente da crença individual sobre a própria capacidade. Se uma pessoa não acredita que pode realizar determinada ação, dificilmente se exporá a situações que permitam desenvolver esse hábito.

Procrastinação: O Oposto do Hábito

A procrastinação surge quando se depara com uma tarefa complexa, incerta ou desconhecida. Nesse momento, o cérebro tende a resistir, pois sabe que precisará investir energia para compreender e executar a atividade.

Como mecanismo de defesa, o cérebro tenta desviar a atenção para atividades mais conhecidas e que ofereçam recompensa com mínimo esforço. Se não houver consciência desse processo, a pessoa facilmente abandona a tarefa importante em favor de atividades habituais e menos desafiadoras.

Estratégias para Combater a Procrastinação

Eliminação de Distrações

Para trabalhar ou estudar profundamente, é fundamental:

- Manter o celular distante
- Fechar abas desnecessárias
- Desativar notificações de e-mail e mensagens
- Concentrar toda atenção na tarefa atual

Desenvolvimento de Bons Hábitos

Algumas recomendações para criar hábitos positivos:

1. **Exercício Físico**: Fundamental para um cérebro eficiente, permite momentos de criatividade e consolidação do aprendizado.
2. **Rotina de Sono**: Essencial para manter um cérebro saudável e produtivo.
3. **Vinculação de Recompensas**: Associar novos hábitos a atividades já prazerosas, como:
    - Convidar amigos para praticar exercícios
    - Ouvir podcasts durante atividades físicas
    - Praticar exercícios em ambientes agradáveis

Conclusão

A compreensão de como os hábitos funcionam oferece uma poderosa ferramenta para combater a procrastinação. Ao reconhecer os mecanismos cerebrais envolvidos, é possível desenvolver estratégias conscientes para criar hábitos positivos e manter-se focado nos objetivos de aprendizado.

08. As principais técnicas de aprendizado

Introdução

Aprender programação requer mais do que simplesmente assistir a vídeos ou ler documentação. É necessário adotar estratégias de estudo que maximizem a retenção e compreensão do conteúdo.

1. Técnica Pomodoro

Conceito

A técnica Pomodoro é um método de gerenciamento de tempo que divide o estudo em intervalos estruturados:

- 25 minutos de foco intenso
- 5 minutos de descanso

Princípios de Aplicação

- Elimine distrações durante o período de foco
- Desligue notificações
- Mantenha o celular distante
- Use os 5 minutos de intervalo para:
  - Alongamento
  - Pequena caminhada
  - Atividades relaxantes leves

Benefícios Psicológicos

- Reduz a procrastinação
- Torna as tarefas menos intimidantes
- Cria um ciclo de motivação e progresso

2. Técnica de Recall

Metodologia

Recall é o processo de recuperação ativa do conhecimento:

- Anote os pontos principais após cada aula
- Tente se lembrar do conteúdo sem consultar materiais
- Transforme o aprendizado em um processo mental ativo

Estratégias

- Use papel e caneta
- Faça anotações em tópicos
- Reconstrua mentalmente os conceitos aprendidos

3. Prática Deliberada

Princípio

Crie seus próprios desafios e testes de conhecimento:

- Formule perguntas sobre o conteúdo estudado
- Responda criticamente
- Estabeleça conexões entre diferentes conceitos

Exemplo de Exercício

Crie um questionário para si mesmo com perguntas como:

- Quais são os conceitos principais?
- Como esses conceitos se inter-relacionam?
- Quais são as aplicações práticas?

4. Interleaving (Estudos Intercalados)

Conceito

Integrar diferentes conceitos e disciplinas em um projeto único:

- Combinar múltiplos conhecimentos
- Criar conexões entre diferentes áreas
- Desenvolver uma compreensão holística

Aplicação em Programação

- Não estudar conceitos isoladamente
- Criar projetos que integrem diversos conhecimentos
- Entender como diferentes elementos se comunicam

Dica Final: Consistência

Recomendação de Estudo

- Estude 1 hora por dia
- Mantenha a regularidade
- Foco na continuidade, não na intensidade

Conclusão

Aprender programação é uma jornada de desenvolvimento contínuo. Ao aplicar essas técnicas, você transformará seu processo de aprendizado, tornando-o mais eficiente e significativo.

09. Como aprender a programar

Introdução ao Aprendizado Autodidata

A área de programação é caracterizada por mudanças rápidas e constantes, o que torna o aprendizado autodidata não apenas uma vantagem, mas uma necessidade fundamental para profissionais de tecnologia.

Lidando com Erros de Programação

Erros são parte integrante do processo de aprendizagem em programação. Alguns pontos importantes:

- Erros de programa não causam danos físicos ao computador
- Cerca de 70% do trabalho de um programador envolve identificar e corrigir erros
- A capacidade de resolver problemas é mais importante que evitá-los completamente

Estratégias para Resolução de Problemas

1. Análise de Mensagens de Erro

- Leia com calma todas as mensagens de erro
- Não desista imediatamente ao encontrar um problema
- Compreender a mensagem é o primeiro passo para a solução

2. Fontes de Pesquisa e Aprendizado

Google

- Ferramenta essencial para buscar soluções
- Dicas:
  - Pesquise a mensagem de erro ou dúvida
  - Utilize termos em inglês para resultados mais abrangentes

Documentação Oficial

- Fonte mais precisa e atualizada
- Documentação do Python disponível em português

- Recomendada para dúvidas conceituais

Stack Overflow

- Fórum público de perguntas e respostas
- Solução para praticamente qualquer dúvida de programação
- Disponível em português

3. Inteligência Artificial como Ferramenta de Aprendizado

- ChatGPT e outras IAs podem auxiliar na explicação de códigos
- Recomendações de uso:
  - Use para aprender, não apenas copiar
  - Entenda o motivo das correções sugeridas
  - Desenvolva senso crítico ao utilizar IAs

Comunidades e Recursos de Aprendizado

Comunidades Online

- Plataformas de troca de conhecimento
- Grupos de discussão
- Suporte de professores e desenvolvedores

Código Aberto

- Projetos open-source como fonte de aprendizado
- Áreas: dashboards, análise de dados, IA, automações
- Estude códigos de projetos da comunidade

A Melhor Estratégia: Projetos Práticos

Princípios Fundamentais

- Crie projetos que resolvam problemas reais
- Inspire-se em projetos existentes

- Modifique e adapte soluções
- Desenvolva soluções próprias para seu dia a dia

Conclusão

O aprendizado em programação é uma jornada de descobertas, erros, correções e crescimento contínuo. A chave está na persistência, curiosidade e disposição para aprender constantemente.

10. Os dados do Portal da Transparência

Introdução aos Dados Governamentais

O Portal da Transparência do Brasil é uma plataforma fundamental para acesso a informações públicas, especialmente relacionadas à gestão de recursos governamentais. Esta plataforma permite que qualquer cidadão consulte dados detalhados sobre gastos públicos, incluindo:

- Notas fiscais
- Informações sobre pessoas físicas e jurídicas
- Emendas parlamentares
- Licitações e contratos
- Registros de servidores públicos

Seleção dos Dados para Análise

Para este curso, será utilizada a base de dados de viagens a serviço de servidores públicos. Esta escolha demonstra como o Python pode ser uma ferramenta poderosa para análise de grandes volumes de dados.

Características dos Dados

No momento da coleta (ano de 2023), os dados revelam:

- Total de viagens nacionais: 85% dos gastos
- Valor em viagens nacionais: R$ 948.000
- Valor em viagens internacionais: R$ 166.000

Volume de Dados

A tabela de viagens a serviço contém mais de 800 mil registros, o que justifica a necessidade de uma ferramenta como Python para processamento e análise eficiente.

Obtendo os Dados

Passos para Download

1. Acesse o Portal da Transparência

2. Navegue até a seção “Viagens a Serviço”
3. Selecione a opção “Dados Abertos”
4. Escolha o ano desejado (neste caso, 2023)
5. Realize o download do arquivo de dados

Considerações Importantes

- Os dados baixados representam um ano completo
- A análise pode ser facilmente adaptada para outros anos
- O Python permite manipular e transformar esses dados com grande flexibilidade

Preparação para Análise

Na próxima etapa, serão explorados os detalhes dos dados baixados, demonstrando como o Python auxiliará na análise e transformação dessas informações governamentais.

11. Como Python ajuda na análise de dados

Introdução à Análise de Dados com Python

A análise de dados exige ferramentas eficientes e flexíveis. Python se destaca como uma linguagem de programação que oferece soluções robustas para processamento e manipulação de grandes volumes de informações.

Arquivos CSV (Comma-Separated Values)

Arquivos CSV são uma forma comum de armazenar dados tabulares. No exemplo trabalhado, utilizamos um arquivo com dados separados por ponto e vírgula (;).

Estrutura de um Arquivo CSV

```
id_processo;numero_proposta;situacao;urgente;orgao_superior;servidor
1;123456;Realizada;Não;Ministério X;João Silva
2;789012;Pendente;Sim;Ministério Y;Maria Souza
```

Vantagens do Python na Importação de Dados

1. **Velocidade de Carregamento**: Mais rápido que planilhas tradicionais
2. **Flexibilidade**: Suporta múltiplos formatos de arquivo
3. **Controle Fino**: Ajustes precisos na importação de dados

Leitura de Arquivos com Pandas

```
import pandas as pd

# Leitura de arquivo CSV
df = pd.read_csv('viagens.csv', sep=';', encoding='utf-8')

# Verificando informações básicas
print(df.info())
print(df.shape)
```

Output Esperado

```
<class 'pandas.core.frame.DataFrame'>
```

```
Número de linhas: 82.000
Número de colunas: 10

Colunas:
- id_processo
- numero_proposta
- situacao
- urgente
- orgao_superior
- servidor
...
```

Processamento de Dados com Python

Comparativo: Python vs Planilhas Tradicionais

Tarefa	Excel/Planilhas	Python
Carregamento de Dados	Lento	Rápido
Manipulação	Baseada em cliques	Baseada em código
Flexibilidade	Limitada	Altamente configurável
Reprodutibilidade	Baixa	Alta

Bibliotecas e Comunidade

Python possui bibliotecas para diversos tipos de processamento:

- Pandas: Análise de dados tabulares
- NumPy: Operações numéricas
- Matplotlib: Visualização de dados
- Scikit-learn: Machine Learning

Desafio para o Aluno

1. Baixe um conjunto de dados público
2. Importe utilizando Pandas

3. Explore as primeiras linhas com `.head()`
4. Verifique informações estatísticas básicas com `.describe()`

Conclusão

Python não é apenas uma linguagem de programação, mas uma ferramenta completa para análise de dados, oferecendo velocidade, flexibilidade e poder de processamento.

12. Conhecendo os Notebook do Google Colab

Introdução ao Ambiente de Análise de Dados

O Google Colab é uma ferramenta do Google que permite executar código Python na nuvem, eliminando a necessidade de instalação local do Python. Suas principais vantagens incluem:

- Ambiente pré-configurado
- Serviço gratuito
- Integração direta com Google Drive

Configuração Inicial

Acesso ao Google Drive

Para começar, siga estes passos:

1. Faça login em sua conta Google (crie uma se necessário)
2. Crie uma pasta no Drive dedicada ao projeto de análise de dados (sugestão: pasta “AD”)

Upload de Dados

- Carregue os arquivos CSV ou ZIP na pasta criada
- Atenção: arquivos grandes podem demandar mais tempo de upload

Criando um Notebook no Google Colab

Processo de Criação

1. Na pasta do Drive, clique com botão direito
2. Selecione **More** → **Google Collaboratory**
3. Um novo arquivo .ipynb será aberto
4. Renomeie o notebook (exemplo: “teste.ipynb”)

Configurações Recomendadas

Para otimizar seu ambiente de trabalho:

- Acesse **Ferramentas** → **Configurações** → **Assistência de IA**

- Marque a opção “Ocultar recursos de IA generativa”
- Objetivo: minimizar distrações durante a codificação

Integração com Google Drive

O notebook do Colab possui integração automática com a pasta do Drive, permitindo:

- Acesso direto aos arquivos carregados
- Execução de análises diretamente nos dados

Recapitulação

Pontos importantes sobre o Google Colab:

- Dispensa instalação local de Python
- Notebooks (.ipynb) combinam código, textos e resultados
- Dados do Drive são acessíveis diretamente no ambiente

13. Nosso primeiro código no Google Colab

Estrutura de Notebooks

Células de Texto (Markdown)

As células de texto permitem formatação básica como:

- Negrito
- Itálico
- Títulos com #

Atalho para renderizar: `Shift + Enter`

Células de Código

Características das células de código:

- Executam código Python diretamente
- Atalho de execução: `Shift + Enter`
- Resultados são exibidos abaixo da célula

Conceitos Básicos em Python

Variáveis

Variáveis armazenam diferentes tipos de valores:

```
mensagem = "Olá, mundo!"  # Variável do tipo string (texto)
nome = "Juliano"          # Texto
idade = 32                # Número inteiro
altura = 1.75             # Número decimal
```

Tipos comuns de variáveis:

- `str`: texto
- `int`: número inteiro
- `float`: número decimal

Operações Matemáticas

Exemplo de soma numérica:

```
x = 1
y = 2.5
soma = x + y
print(soma)  # Resultado: 3.5
```

Observação: números decimais usam ponto decimal (ex.: `2.5`)

Formatação de Texto com F-Strings

```
nome = "Juliano"
idade = 32
texto = f"Meu nome é {nome} e tenho {idade} anos."
print(texto)  # Exibe: "Meu nome é Juliano e tenho 32 anos."
```

Características das f-strings:

- Prefixo `f` antes das aspas
- Variáveis inseridas entre `{}`

Recursos do Google Colab

Painel de Variáveis

- Mostra variáveis criadas
- Exibe tipos e valores
- Acessível por ícone lateral

Exibição Automática

- Último valor de uma célula é exibido automaticamente
- Dispensa uso de `print()` para análises rápidas

Recapitulação

Pontos importantes:

- Notebooks combinam texto e código em blocos interativos
- Variáveis organizam e reutilizam dados

- F-strings facilitam criação de textos dinâmicos
- Google Colab oferece ferramentas integradas para agilizar o trabalho

14. Lendo dados do Google Drive

Preparação para Leitura de Arquivos

Antes de carregar os dados, é necessário garantir que o Google Drive esteja acessível no Google Colab, caso os arquivos estejam armazenados lá.

Montagem do Google Drive

Siga os passos abaixo para acessar arquivos no Google Drive:

1) No Google Colab, clique no ícone de **pastinha** no painel lateral esquerdo.
2) Selecione a opção **Mount Drive** (Montar Drive).
3) Autorize o acesso ao Google Drive quando solicitado.

Após a montagem, os arquivos ficam acessíveis no diretório `/content/drive/MyDrive/`.

Verificando Arquivos no Drive

Para garantir que o arquivo está no local correto, podemos listar os arquivos dentro de uma pasta:

```
import os

# Listar arquivos dentro de uma pasta específica
ios.listdir("/content/drive/MyDrive/AD/")
```

Saída esperada (exemplo):

```
['2023_viagem.csv', 'outro_arquivo.xlsx']
```

Se o arquivo desejado estiver listado, podemos prosseguir com a leitura.

Processo de Carregamento de Dados

Agora que o Google Drive está montado e sabemos que o arquivo existe, podemos carregá-lo usando a biblioteca **pandas**.

Definição do Caminho do Arquivo

Antes de carregar os dados, precisamos definir o caminho do arquivo:

```
caminho_dados = "/content/drive/MyDrive/AD/2023_viagem.csv"
```

Importação do Pandas

O Pandas é uma biblioteca poderosa para manipulação de dados em Python. Primeiro, precisamos importá-la:

```
import pandas as pd
```

Leitura do Arquivo CSV

Para carregar o arquivo CSV corretamente, usamos o método `pd.read_csv()` com alguns parâmetros importantes:

```
df_viagens = pd.read_csv(
    caminho_dados,    # Caminho do arquivo
    encoding='latin-1',  # Codificação apropriada para caracteres especiais
    sep=';'              # Separador utilizado no arquivo CSV
)
```

O que cada parâmetro faz?

- `caminho_dados`: caminho para o arquivo CSV.
- `encoding='latin-1'`: define a codificação para garantir que caracteres acentuados sejam lidos corretamente.
- `sep=';'`: define o separador de colunas (muitos arquivos CSV usam `;` ao invés de `,`).

Caso o arquivo seja carregado corretamente, podemos visualizar os dados no próximo passo.

Tratamento de Erros Comuns

Alguns erros podem ocorrer ao tentar carregar um arquivo CSV. Aqui estão os mais comuns e como resolvê-los:

Erro 1: UnicodeDecodeError

Se ao executar `pd.read_csv()`, aparecer um erro como:

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe7 in position 10: invalid con
```

Isso significa que o arquivo não está em `utf-8`. Solução:

- Teste com diferentes codificações, como `latin-1` ou `ISO-8859-1`:

  ```
  df_viagens = pd.read_csv(caminho_dados, encoding='latin-1', sep=';')
  ```

Erro 2: ParserError

Se o erro for:

```
pandas.errors.ParserError: Error tokenizing data. C error: Expected X fields in line
```

O problema geralmente é um separador incorreto. Solução:

- Confirme o separador do arquivo e ajuste o parâmetro `sep`. Se não souber qual é, tente `sep=","` ou `sep=";"`.

```
df_viagens = pd.read_csv(caminho_dados, encoding='latin-1', sep=',')
```

Visualização dos Dados

Após carregar os dados, é importante verificar se foram lidos corretamente.

Exibição do DataFrame

Para visualizar as primeiras linhas da tabela:

```
df_viagens.head()
```

Saída esperada (exemplo):

Nome do órgão	Data	Valor diárias	Valor passagens
Ministério X	01/01/2023	500.00	350.00
Ministério Y	02/01/2023	620.50	400.75

Podemos também visualizar as dimensões da tabela:

```
df_viagens.shape
```

Saída esperada:

```
(1000, 4)  # 1000 linhas e 4 colunas
```

Se precisar exibir todas as colunas, use:

```
pd.set_option('display.max_columns', None)
df_viagens.head()
```

Conceitos Importantes

DataFrame (`df`)

O **DataFrame** é a estrutura principal do Pandas para manipulação de dados.

- Ele funciona como uma tabela do Excel, onde cada linha representa um registro e cada coluna representa um atributo.

Parâmetros Importantes do `pd.read_csv()`

Alguns parâmetros essenciais:

- `encoding`: Define a codificação de caracteres para evitar erros de leitura.
- `sep`: Especifica o separador de colunas.
- `decimal`: Indica qual caractere é usado para decimais (`.` ou `,`).

Exemplo de uso de `decimal` para converter valores:

```
df_viagens = pd.read_csv(caminho_dados, encoding='latin-1', sep=';', decimal=',')
```

Características de Desempenho

- O Pandas é extremamente rápido para processar grandes volumes de dados.
- Um arquivo CSV com **800.000+ linhas** pode ser carregado em aproximadamente **19 segundos** no Google Colab.
- Ferramentas como Excel ou LibreOffice teriam dificuldades com esse volume.

Recapitulação

Pontos fundamentais abordados:

- **Montar o Google Drive** no Colab antes de carregar arquivos.
- **Utilizar Pandas** para ler arquivos CSV de maneira eficiente.
- **Atentar-se aos parâmetros** `encoding`, `sep` e `decimal` para evitar erros.
- **Visualizar os dados** corretamente para verificar a importação.

15. Acessando e manipulando colunas

Ajustes de Exibição

Ao trabalhar com DataFrames grandes, é comum que algumas colunas fiquem ocultas na visualização padrão. Para remover esse limite e exibir todas as colunas, utilize o seguinte comando:

Configuração de Visualização

```
import pandas as pd

# Remove limite de colunas exibidas
pd.set_option('display.max_columns', None)

# Exibe o DataFrame
print(df_viagens)
```

Output esperado (exemplo simplificado):

```
    Nome do órgão superior    Valor diárias    Data da viagem
0    Ministério da Saúde       500.00          2023-05-12
1    Ministério da Educação    750.00          2023-06-15
2    Ministério da Justiça     620.50          2023-07-20
```

> **Atenção:** Evite exibir todas as linhas de um dataset grande para não sobrecarregar a interface. Para visualizar apenas as primeiras linhas, use `df.head()`.

Seleção de Colunas

Seleção de Uma Única Coluna

Para acessar uma coluna específica de um DataFrame, utilize a sintaxe abaixo:

```
print(df_viagens['Nome do órgão superior'])
```

Output esperado:

```
0    Ministério da Saúde
1    Ministério da Educação
2    Ministério da Justiça
Name: Nome do órgão superior, dtype: object
```

Seleção de Múltiplas Colunas

Para selecionar mais de uma coluna, passe uma lista com os nomes desejados:

```
colunas = ['Nome do órgão superior', 'Valor diárias']
print(df_viagens[colunas])
```

Output esperado:

```
    Nome do órgão superior    Valor diárias
0    Ministério da Saúde       500.00
1    Ministério da Educação    750.00
2    Ministério da Justiça     620.50
```

> **Dica:** Use sempre os nomes exatos das colunas, respeitando maiúsculas, espaços e acentos.

Manipulação de Texto

Conversão para Maiúsculas

```
print(df_viagens['Nome do órgão superior'].str.upper())
```

Output esperado:

```
0    MINISTÉRIO DA SAÚDE
1    MINISTÉRIO DA EDUCAÇÃO
2    MINISTÉRIO DA JUSTIÇA
Name: Nome do órgão superior, dtype: object
```

Cálculo de Comprimento de Texto

```
print(df_viagens['Nome do órgão superior'].str.len())
```

Output esperado:

```
0    21
1    23
2    22
Name: Nome do órgão superior, dtype: int64
```

Substituição de Texto

```
print(df_viagens['Nome do órgão superior'].str.replace('MINISTÉRIO', 'MIN.'))
```

Output esperado:

```
0    MIN. DA SAÚDE
1    MIN. DA EDUCAÇÃO
2    MIN. DA JUSTIÇA
Name: Nome do órgão superior, dtype: object
```

> **Observação:** O método `.str.replace()` permite modificar partes do texto, facilitando a padronização dos dados.

Encadeamento de Métodos

Ao aplicar várias transformações em uma mesma coluna, é possível encadear métodos para tornar o código mais limpo e eficiente.

Transformações em Sequência

```
print(
    df_viagens['Nome do órgão superior']
    .str.upper()                            # Converte para maiúsculas
    .str.replace('MINISTÉRIO', 'MIN.')      # Substitui texto
    .str.len()                              # Calcula comprimento
)
```

Output esperado:

```
0    14
1    16
2    15
Name: Nome do órgão superior, dtype: int64
```

> **Vantagem:** Permite realizar múltiplas operações sem a necessidade de criar variáveis intermediárias.

Princípios Importantes:

- O encadeamento de métodos facilita transformações complexas
- Consultar a documentação do Pandas pode trazer soluções eficientes
- Praticar é mais importante que decorar comandos individuais

Com essas técnicas, você pode manipular colunas de forma eficiente, tornando os dados mais organizados e prontos para análise!

16. Operações entre colunas e tipos de dado

Identificando Tipos de Dados

Verificação de Tipos

Problema comum: Colunas numéricas podem ser lidas como texto (tipo `object`) devido a formatações específicas.

```
import pandas as pd

# Criando um DataFrame de exemplo
dados = {
    "Nome": ["João", "Maria", "Carlos"],
    "Valor diárias": ["1.200,50", "800,00", "950,75"],
    "Valor passagens": ["500,00", "650,50", "720,25"]
}

df_viagens = pd.DataFrame(dados)

# Verificando os tipos das colunas
df_viagens.info()
```

Saída esperada:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Nome              3 non-null      object
 1   Valor diárias     3 non-null      object
 2   Valor passagens   3 non-null      object
```

Exemplos de tipos:

- `int64`: número inteiro
- `object`: texto
- `float64`: número decimal

Caso típico: Colunas como “Valor diárias” aparecem como `object` devido ao uso de vírgula decimal.

Preparação de Colunas Numéricas

Substituição de Separador Decimal

```
df_viagens['Valor diárias'] = df_viagens['Valor diárias'].str.replace(',', '.')
df_viagens['Valor passagens'] = df_viagens['Valor passagens'].str.replace(',', '.')
```

Saída esperada (DataFrame atualizado):

```
        Nome Valor diárias Valor passagens
0      João      1200.50        500.00
1     Maria       800.00        650.50
2    Carlos       950.75        720.25
```

Conversão para Tipo Numérico

```
df_viagens['Valor diárias'] = df_viagens['Valor diárias'].astype(float)
df_viagens['Valor passagens'] = df_viagens['Valor passagens'].astype(float)
```

Saída esperada:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Nome              3 non-null      object
 1   Valor diárias     3 non-null      float64
 2   Valor passagens   3 non-null      float64
```

Agora as colunas são do tipo `float64`, permitindo operações matemáticas.

Operações Básicas

Soma de Colunas

```
df_viagens['Despesas totais'] = df_viagens['Valor diárias'] + df_viagens['Valor passagens']
print(df_viagens[['Nome', 'Despesas totais']])
```

Saída esperada:

```
     Nome  Despesas totais
0   João          1700.50
1  Maria          1450.50
2  Carlos         1671.00
```

Multiplicação e Divisão

```
df_viagens['Diárias x 3'] = df_viagens['Valor diárias'] * 3
df_viagens['Passagens / 2'] = df_viagens['Valor passagens'] / 2
print(df_viagens[['Nome', 'Diárias x 3', 'Passagens / 2']])
```

Saída esperada:

```
     Nome  Diárias x 3  Passagens / 2
0   João       3601.50         250.00
1  Maria       2400.00         325.25
2  Carlos      2852.25         360.12
```

Operações Combinadas

```
df_viagens['Cálculo complexo'] = (df_viagens['Valor diárias'] * 2) + (df_viagens['Valor
↪  passagens'] / 3)
print(df_viagens[['Nome', 'Cálculo complexo']])
```

Saída esperada:

```
     Nome  Cálculo complexo
0   João         2500.50
1  Maria         1766.83
2  Carlos        2204.08
```

Boas Práticas

Fluxo de Trabalho

1) **Verificar tipos de dados** com `df.info()`
2) **Substituir vírgulas e converter para `float`**
3) **Realizar operações matemáticas** entre colunas corretamente formatadas

Pontos de Atenção

- **Valores não numéricos (`object`) causam erros em operações matemáticas**
- Utilize `.str.replace()` para ajustar formatação antes da conversão
- Use `.astype(float)` para garantir que os dados estejam no formato correto

Recapitulação

Princípios fundamentais:

- Sempre verifique os tipos de dados antes de operações
- Converta colunas para o tipo numérico adequado
- Realize operações matemáticas corretamente
- Mantenha a consistência dos tipos de dados

Esse processo garante que seus dados estejam prontos para manipulações avançadas no Pandas!

17. Contando frequências

Carregamento Correto de Dados

Leitura com Separador Decimal

```
df_viagens = pd.read_csv(caminho_dados, decimal=',')  # Define vírgula como separador decimal
```

Benefício: Colunas como “Valor diárias” são lidas automaticamente como `float64` (números decimais).

Criação de Colunas Derivadas

Cálculo de Despesas Totais

```
df_viagens['Despesas totais'] = (
    df_viagens['Valor diárias'] +
    df_viagens['Valor passagens'] +
    df_viagens['Valor outros gastos']
)
```

Contagem de Frequências com `value_counts()`

Contagem Básica por Cargo

```
contagem_cargos = df_viagens['Cargo'].value_counts()
```

Exemplo de saída: | Cargo | Frequência | |—–|———| | Sigilo | 118000 | | Professor Magistério Superior | 54000 | | Professor Ensino Básico/Técnico | 37000 |

Proporção de Viagens

```
proporcao_cargos = df_viagens['Cargo'].value_counts(normalize=True) * 100
```

Exemplo de saída: | Cargo | Proporção (%) | |—–|———| | Sigilo | 23.0% | | Professor Magistério Superior | 10.0% |

Transformações Avançadas

```
resultado = (
    df_viagens['Cargo']
    .value_counts(normalize=True)
```

```
    .rename('Proporção de viagens')
    .reset_index()
)
```

Métodos e Parâmetros

Função *value_counts*

Principais características:

- Conta frequência de valores únicos em uma coluna
- Parâmetro `normalize=True` retorna proporções (0 a 1)

Transformações Comuns

- Multiplicar por 100 para converter proporção em porcentagem
- `.rename()` para alterar nome de colunas
- `.reset_index()` para converter índices em colunas

Recapitulação

Benefícios da Análise com Pandas:

- Contagem rápida de frequências
- Transformações simples de dados
- Análise exploratória ágil
- Dispensando interfaces gráficas como Excel

18. Criando novas tabelas e agrupando dados

Nesta seção, vamos explorar formas de agrupar os dados além do `value_counts`, permitindo análises mais avançadas com o Pandas.

Código Inicial de Análise de Dados

Antes de realizar os agrupamentos, é necessário preparar os dados:

1) Configurar a exibição de todas as colunas.
2) Ler os dados, especificando o separador decimal.
3) Criar uma nova coluna `despesas`, somando as três colunas de gastos das viagens.

Configuração de Exibição

```
import pandas as pd
pd.set_option("display.max_columns", None)  # Exibe todas as colunas do DataFrame
```

Carregamento dos Dados

```
df_viagens = pd.read_csv("/content/drive/MyDrive/AD/2023_viagem.csv", sep=';',
↪  encoding='latin-1')
```

Criando a Coluna de Despesas Totais

```
df_viagens["despesas"] = (
    df_viagens["Valor diárias"].astype(float) +
    df_viagens["Valor passagens"].astype(float) +
    df_viagens["Valor outros gastos"].astype(float)
)
```

Exibição das Primeiras Linhas

```
df_viagens.head()
```

Saída esperada:

```
   cargo          Valor diárias  Valor passagens  Valor outros gastos  despesas
0  Analista         120.50        450.00          50.00               620.50
1  Técnico          80.00         200.00          30.00               310.00
2  Gerente          300.00        700.00          100.00              1100.00
...
```

Utilizando `groupby` para Agrupar Dados

O `value_counts()` permite contar a frequência de elementos em uma coluna, mas para somar valores agrupados por um critério específico, utilizamos `groupby`.

Por exemplo, para obter o total de despesas por cargo:

```
# Agrupando por cargo
agrupado = df_viagens.groupby("cargo")
print(agrupado)
```

Saída esperada:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f8b1c2d9d60>
```

Isso indica que os dados estão agrupados, mas ainda precisam ser processados.

Aplicando Funções de Agregação

Para somar os valores da coluna `despesas` dentro de cada grupo:

```
# Somando despesas por cargo
despesas_por_cargo = df_viagens.groupby("cargo")["despesas"].sum()
print(despesas_por_cargo)
```

Saída esperada:

```
cargo
Analista    15200.50
Gerente     34250.00
Técnico     10300.00
Name: despesas, dtype: float64
```

Isso gera uma tabela com os cargos e a soma total de despesas.

Ajustes na Tabela

Resetando o Índice

Para transformar a coluna `cargo` em uma coluna comum:

```
despesas_por_cargo = despesas_por_cargo.reset_index()
```

Ordenando Valores

Para exibir os cargos do maior para o menor gasto:

```
despesas_por_cargo = despesas_por_cargo.sort_values(by="despesas", ascending=False)
```

Formatando os Números

Para exibir valores com duas casas decimais:

```
pd.set_option("display.float_format", "{:.2f}".format)
```

Outras Funções de Agregação

Além da soma, podemos utilizar outras funções de agregação:

Média das Despesas por Cargo

```
despesas_media = df_viagens.groupby("cargo")["despesas"].mean()
print(despesas_media)
```

Saída esperada:

```
cargo
Analista    950.03
Gerente     2200.25
Técnico     725.00
Name: despesas, dtype: float64
```

Maior e Menor Despesa Registrada por Cargo

```
despesas_max = df_viagens.groupby("cargo")["despesas"].max()
despesas_min = df_viagens.groupby("cargo")["despesas"].min()
```

Saída esperada:

```
Maior Despesa:
cargo
Analista    1800.00
Gerente     3500.00
```

```
Técnico     1200.00

Menor Despesa:
cargo
Analista    600.00
Gerente     1500.00
Técnico     400.00
```

Primeira e Última Ocorrência Registrada de Cada Cargo

```
despesas_first = df_viagens.groupby("cargo")["despesas"].first()
despesas_last = df_viagens.groupby("cargo")["despesas"].last()
```

Saída esperada:

```
Primeira ocorrência:
cargo
Analista    900.00
Gerente     2500.00
Técnico     700.00

Última ocorrência:
cargo
Analista    1300.00
Gerente     3100.00
Técnico     900.00
```

Conclusão

Utilizar `groupby` no Pandas permite explorar diferentes formas de análise, facilitando a obtenção de insights sobre os dados.

Combinando agrupamentos, funções de agregação e ordenação, é possível extrair informações valiosas de maneira eficiente.

Resumo dos Passos:

1) **Ler e preparar os dados**
2) **Criar colunas de interesse**

3) **Agrupar os dados por categorias**
4) **Aplicar funções de agregação**
5) **Otimizar a apresentação dos resultados**

Essa abordagem é essencial para análise exploratória de dados e construção de relatórios dinâmicos.

19. Séries boobleanas e filtros

Recapitulando o Script Final

A única adição feita até o momento foi a configuração de opções do pandas para formatar os dados numéricos com duas casas decimais. Agora, voltamos à nossa tabela de testes.

Armazenando a Tabela em uma Variável

```
viagens_por_cargo = tabela
```

Dessa forma, sempre que acessarmos a variável `viagens_por_cargo`, teremos acesso à tabela resultante. Essa tabela contém os cargos e a proporção de viagens feitas por cada um deles.

Comparações em Python

Antes de trabalharmos com séries booleanas, precisamos entender as comparações em Python.

```
print(10 > 5)  # True
print(10 < 5)  # False
print(10 >= 5) # True
print(10 <= 5) # False
print(10 == 5) # False
print(10 != 5) # True
```

Saída esperada:

```
True
False
True
False
False
True
```

Atenção especial à comparação de igualdade (==), pois um único = é utilizado para atribuição de valores.

Criando uma Série Booleana

Podemos usar comparações em uma coluna da nossa tabela para criar uma série booleana:

```
filtro_mais_de_1_porcento = viagens_por_cargo["Proporção Viagens"] > 1
print(filtro_mais_de_1_porcento)
```

Exemplo de saída esperada:

```
0     True
1     False
2     True
3     False
Name: Proporção Viagens, dtype: bool
```

Isso gera uma série de valores `True` e `False` que indicam quais linhas atendem ao critério.

Aplicando um Filtro

Podemos usar essa série booleana para filtrar os dados:

```
viagens_filtradas = viagens_por_cargo[filtro_mais_de_1_porcento]
print(viagens_filtradas)
```

Exemplo de saída esperada:

```
       Cargo  Proporção Viagens
0  Diretor                2.3
2  Analista               1.5
```

Assim, obtemos apenas os cargos cuja proporção de viagens é maior que 1%.

Filtrando por Texto

Para filtrar cargos que começam com “Técnico”:

```
filtro_tecnico = viagens_por_cargo["Cargo"].str.startswith("Técnico")
viagens_tecnicos = viagens_por_cargo[filtro_tecnico]
print(viagens_tecnicos)
```

Exemplo de saída esperada:

```
       Cargo  Proporção Viagens
1  Técnico              0.8
```

Isso retorna apenas os cargos que iniciam com “Técnico”.

Combinando Filtros

Podemos combinar filtros utilizando o operador & (AND):

```
filtro_combinacao = filtro_mais_de_1_porcento & filtro_tecnico
viagens_tecnicos_filtradas = viagens_por_cargo[filtro_combinacao]
print(viagens_tecnicos_filtradas)
```

Exemplo de saída esperada:

```
Empty DataFrame
Columns: [Cargo, Proporção Viagens]
Index: []
```

Nesse caso, se não houver cargos que atendam aos dois critérios simultaneamente, o DataFrame retornará vazio.

20. Lidando com dados nulos

Introdução

Na análise de dados, é comum encontrar valores nulos (missing data) que precisam ser tratados adequadamente. Nesta seção, vamos explorar técnicas para identificar, tratar e trabalhar com dados nulos utilizando Pandas.

Identificando Dados Nulos

Dados nulos, representados como None ou NaN (Not a Number), podem surgir por diversos motivos: cadastros incompletos, falhas na coleta de dados ou informações não disponíveis.

Verificando Valores Nulos com Value Counts

```
# Contagem de cargos com valores nulos
contagem_cargos = df['cargo'].value_counts(dropna=False)
```

Output Esperado:

```
cargo
None                        322000
Informações protegidas      118000
Outros cargos                65000
```

Métodos de Tratamento de Dados Nulos

1. Removendo Valores Nulos (dropna)

```
# Removendo linhas com valores nulos na coluna 'cargo'
df_limpo = df.dropna(subset=['cargo'])
```

Output Esperado:

```
# Número de linhas reduzido de 826.000 para 505.000
```

2. Preenchendo Valores Nulos (fillna)

```
# Substituindo valores nulos por texto descritivo
df['cargo'] = df['cargo'].fillna('Não identificado')
```

Output Esperado:

```
cargo
Não identificado           322000
Informações protegidas     118000
Outros cargos               65000
```

Considerações Importantes

- Nem sempre remover dados nulos é a melhor estratégia
- A decisão depende do contexto da análise
- No exemplo, mais de um terço dos dados tinham cargo não identificado

Boas Práticas

1. Sempre verifique a proporção de dados nulos
2. Entenda o motivo dos dados faltantes
3. Escolha a estratégia mais adequada:
    - Remoção
    - Preenchimento com valor padrão
    - Preenchimento com valor estatístico (média, mediana)

Exemplo Completo de Tratamento

```
import pandas as pd

# Configurações iniciais
pd.set_option('display.max_columns', None)

# Leitura dos dados
df_viagens = pd.read_csv('caminho_do_arquivo.csv')

# Análise de valores nulos
print(df_viagens['cargo'].value_counts(dropna=False))

# Tratamento de dados nulos
df_viagens['cargo'] = df_viagens['cargo'].fillna('Não identificado')
```

Desafio para o Aluno

1. Calcule a porcentagem de valores nulos em cada coluna
2. Compare as estatísticas descritivas antes e depois do tratamento de nulos
3. Implemente uma estratégia de preenchimento baseada na média ou mediana

Conclusão

Tratar dados nulos de forma consciente é fundamental para garantir a qualidade e confiabilidade de análises de dados.

21. Trabalhando com datas

Introdução

No ambiente de análise de dados com Python, saber manipular datas é uma habilidade fundamental. Nesta seção, vamos explorar como trabalhar com colunas de datas utilizando a biblioteca Pandas, transformando dados textuais em objetos de data e realizando operações avançadas.

Convertendo Colunas de Texto para Datetime

Frequentemente, os dados importados vêm com datas no formato de texto. No Pandas, é possível converter essas colunas para o tipo datetime, o que permite realizar diversas operações.

Conversão Básica

Para converter uma coluna de texto para datetime, utiliza-se o método `pd.to_datetime()`:

```
import pandas as pd

# Exemplo de conversão de coluna de data
df['data_inicio'] = pd.to_datetime(df['data_inicio'], format='%d/%m/%Y')
```

Explicação do Código:

- `pd.to_datetime()`: Método para conversão de dados
- `format='%d/%m/%Y'`: Especifica o formato da data (dia/mês/ano)
  - `%d`: Dia do mês
  - `%m`: Mês
  - `%Y`: Ano com 4 dígitos

Output Esperado:

```
# Antes da conversão:
dtype: object (texto)

# Depois da conversão:
dtype: datetime64[ns]
```

Extraindo Informações de Datas

Após a conversão para datetime, é possível extrair diversas informações das colunas.

Obtendo Nome do Mês

```
# Extraindo nome do mês de uma coluna de datas
df['mes_viagem'] = df['data_inicio'].dt.month_name()
```

Output Esperado:

```
# Exemplo de resultado
['January', 'February', 'January', 'December']
```

Calculando Duração de Eventos

Para calcular a duração de eventos ou viagens, basta subtrair as datas:

```
# Calculando dias de duração
df['dias_viagem'] = (df['data_fim'] - df['data_inicio']).dt.days
```

Output Esperado:

```
# Exemplo de resultado
[3, 5, 137, 19]  # Número de dias de cada viagem
```

Considerações Importantes

- Sempre converta datas para o formato datetime antes de realizar operações
- O Pandas facilita cálculos e extração de informações de datas
- Preste atenção ao formato da data durante a conversão

Exemplos de Uso Prático

```
# Script completo de manipulação de datas
import pandas as pd

# Leitura dos dados
df_viagens = pd.read_csv('caminho_do_arquivo.csv')

# Conversão de colunas de data
df_viagens['data_inicio'] = pd.to_datetime(df_viagens['data_inicio'], format='%d/%m/%Y')
```

```
df_viagens['data_fim'] = pd.to_datetime(df_viagens['data_fim'], format='%d/%m/%Y')

# Criando novas colunas
df_viagens['mes_viagem'] = df_viagens['data_inicio'].dt.month_name()
df_viagens['dias_viagem'] = (df_viagens['data_fim'] - df_viagens['data_inicio']).dt.days

# Exibição dos resultados
print(df_viagens)
```

Desafio para o Aluno

Tente adaptar o código para:

- Calcular a média de dias de viagem
- Filtrar viagens com duração superior a 30 dias
- Agrupar viagens por mês de início

Conclusão

Manipular datas no Pandas é simples e poderoso. Com poucos comandos, é possível transformar e extrair informações valiosas de colunas de data.

22. Combinando agregrações e filtros

Introdução

Neste capítulo, abordaremos técnicas avançadas de manipulação de dados utilizando Pandas, focando em agregações complexas e filtros sofisticados. Os conceitos apresentados permitirão análises mais profundas e detalhadas de conjuntos de dados.

Método .agg() - Agregações Múltiplas

O método `.agg()` permite realizar diferentes operações de agregação em múltiplas colunas simultaneamente.

Exemplo Prático de Agregação

```
import pandas as pd

# Realizando agregações múltiplas
df_agregado = (df
    .groupby('cargo')
    .agg({
        'despesa_media': ('despesas', 'min'),
        'duracao_media': ('dias_viagem', 'mean'),
        'despesas_totais': ('despesas', 'sum'),
        'destino_frequente': ('destinos', lambda x: pd.Series.mode(x)[0]),
        'total_viagens': ('nome', 'count')
    })
    .reset_index()
)
```

Output Esperado:

```
        cargo  despesa_media  duracao_media  despesas_totais  destino_frequente  tot
0    Vigilante         693.0           1.79         739000.0         Três Rios       1067
...
```

Explicação do Código

- `groupby('cargo')`: Agrupa os dados por cargo
- `.agg()`: Realiza múltiplas agregações
  - `'despesa_media'`: Valor mínimo de despesas

- `'duracao_media'`: Média de dias de viagem
- `'despesas_totais'`: Soma total de despesas
- `'destino_frequente'`: Destino mais recorrente
- `'total_viagens'`: Contagem total de viagens

Filtragem Avançada com .loc[]

Seleção de Cargos Relevantes

```
# Identificando cargos com mais de 1% das viagens
df_cargos = (df['cargo']
    .value_counts(normalize=True)
    .reset_index()
)
df_cargos.columns = ['cargo', 'proporcao']

# Filtrando cargos acima de 1%
cargos_relevantes = df_cargos.loc[df_cargos['proporcao'] > 0.01, 'cargo']
```

Aplicando Filtro na Tabela Consolidada

```
# Filtrando tabela agregada
df_final = df_viagens_consolidado[
    df_viagens_consolidado['cargo'].isin(cargos_relevantes)
]
```

Boas Práticas

1. Use `.agg()` para agregações complexas
2. Utilize `.loc[]` para filtros precisos
3. Encadeie operações para código mais legível
4. Sempre resete o índice após agregações

Considerações Finais

As técnicas apresentadas demonstram o poder do Pandas para análise de dados, permitindo operações sofisticadas que seriam complexas em ferramentas como Excel.

23. Criando e customizando um gráfico

Introdução à Visualização de Dados

A visualização de dados é uma etapa crucial na análise de informações, permitindo transformar dados brutos em representações visuais que facilitam a compreensão e interpretação. Neste capítulo, serão apresentadas técnicas para criar gráficos utilizando as bibliotecas Pandas e Matplotlib.

Bibliotecas Utilizadas

Para este capítulo, serão necessárias duas bibliotecas principais:

- Pandas: para manipulação e preparação dos dados
- Matplotlib: para criação e customização de gráficos

Preparação do Ambiente

Antes de iniciar, certifique-se de ter as bibliotecas instaladas:

```
pip install pandas matplotlib
```

Criando Gráficos com Pandas

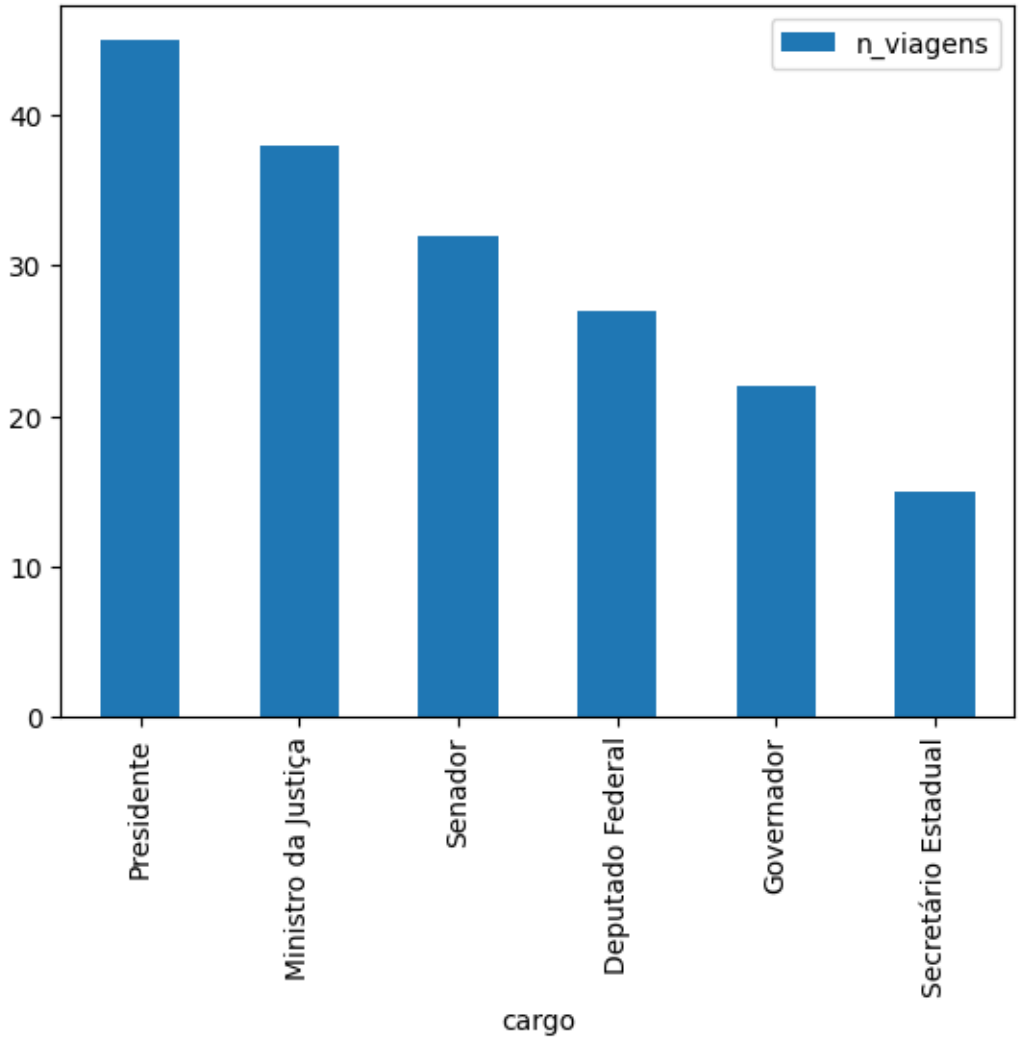
Método Básico de Plotagem

O Pandas oferece métodos próprios para geração rápida de gráficos:

```
import pandas as pd

# Criando DataFrame com dados de viagens por cargo público
dffinal = pd.DataFrame({
    'cargo': [
        'Presidente',
        'Ministro da Justiça',
        'Senador',
        'Deputado Federal',
        'Governador',
        'Secretário Estadual'
    ],
    'n_viagens': [45, 38, 32, 27, 22, 15]
})

# Exemplo de criação de gráfico com Pandas
dffinal.plot(x='cargo', y='n_viagens', kind='bar')
```



**Output Esperado

Ordenando Dados para Visualização

```
# Ordenando dados antes da plotagem
dffinal_ordenado = dffinal.sort_values(by='n_viagens', ascending=False)
dffinal_ordenado.plot(x='cargo', y='n_viagens', kind='bar')
```

Criação de Gráficos com Matplotlib

Configurações Básicas

```
import matplotlib.pyplot as plt

# Criando figura e eixos
fig, ax = plt.subplots(figsize=(16, 6))
```

```
# Plotando gráfico de barras
ax.bar(dffinal['cargo'], dffinal['n_viagens'])

# Adicionando título
plt.title('Viagens por Cargo Público 2023')

plt.show()
```

Customizações Avançadas

Gráfico Horizontal

```
# Transformando gráfico em barras horizontais
ax.barh(dffinal['cargo'], dffinal['n_viagens'])
```

Adicionando Elementos

```
# Adicionando rótulo no eixo X
plt.xlabel('Número de Viagens')

# Invertendo eixo Y
ax.invert_yaxis()

# Adicionando grade
plt.grid(color='grey', linestyle='--', linewidth=0.5)
```

Customização de Cores e Estilo

```
# Alterando cor das barras
ax.bar(dffinal['cargo'], dffinal['n_viagens'], color='#4CAF50')

# Definindo cor de fundo
ax.set_facecolor('#E0E0E0')
```

Adicionando Metadados

```
# Adicionando fonte dos dados
plt.figtext(0.65, 0.89, 'Fonte: Portal da Transparência', fontsize=8)
```

Boas Práticas

- Sempre ordene seus dados antes de plotar
- Utilize cores que facilitem a compreensão

- Adicione sempre a fonte dos dados
- Mantenha o gráfico limpo e objetivo

Próximos Passos

Na próxima seção, abordaremos técnicas de análise exploratória de dados, aprofundando ainda mais as possibilidades de interpretação de conjuntos de dados.

24. Exploração de dados e união de tabelas

A análise exploratória de dados é um processo essencial ao trabalhar com conjuntos de dados novos ou ao tentar extrair insights desconhecidos. Esse processo permite identificar padrões e relações entre variáveis através de visualização de dados.

Importando bibliotecas

```
import pandas as pd
import matplotlib.pyplot as plt
```

Carregando um conjunto de dados

```
dados = pd.read_csv('arquivo.csv')
print(dados.head())
```

Saída esperada:

```
   Coluna1  Coluna2  Coluna3
0       10       20       30
1       15       25       35
2       20       30       40
3       25       35       45
4       30       40       50
```

Estatísticas básicas

```
print(dados.describe())
```

Saída esperada:

```
         Coluna1    Coluna2    Coluna3
count   100.000   100.000   100.000
mean     50.500    60.500    70.500
std      29.011    29.011    29.011
min      10.000    20.000    30.000
max      90.000   100.000   110.000
```

Visualizando o DataFrame Inicial

Antes de iniciar a exploração, é útil visualizar o DataFrame original:

```
# Carregar os dados
caminho_arquivo = "dados.csv"
df_viagens = pd.read_csv(caminho_arquivo)

# Exibir as primeiras linhas
df_viagens.head()
```

Este comando exibe as primeiras linhas do DataFrame, permitindo uma visão inicial dos dados.

Analisando a Relação entre Dias de Viagem e Despesas

Uma das perguntas que podem surgir é: "Existe uma relação entre o tempo de viagem e o valor gasto?" Para responder a essa questão, podemos usar um gráfico de dispersão.

```
# Criar uma figura para o gráfico
fig, ax = plt.subplots(figsize=(16, 6))

# Criar um scatter plot
ax.scatter(df_viagens['dias_viagem'], df_viagens['despesas'], alpha=0.2)

# Adicionar rótulos
ax.set_xlabel("Dias de Viagem")
ax.set_ylabel("Despesas")
ax.set_title("Relação entre Dias de Viagem e Despesas")

plt.show()
```

Explicação do Código

1) Criamos uma figura de 16x6 polegadas para melhor visualização.
2) Utilizamos `scatter()` para gerar o gráfico de dispersão.
3) O argumento `alpha=0.2` adiciona transparência aos pontos, destacando regiões com maior concentração de dados.

Saída esperada:

O gráfico mostra que, em geral, viagens mais longas tendem a gerar mais despesas. Além disso, observa-se que a maioria das viagens dura menos de 100 dias e gera menos de R$25.000 em despesas.

Aplicando Zoom no Gráfico

Se quisermos observar melhor a distribuição dentro de um intervalo específico, podemos definir limites para os eixos:

```
# Ajustando os limites dos eixos
ax.set_xlim(0, 100)
ax.set_ylim(0, 25000)
plt.show()
```

Isso permite focar na região de maior concentração de viagens.

Filtrando Viagens com Altas Despesas

Identificar viagens com custos elevados pode ser útil para investigação. Podemos encontrar viagens com despesas superiores a R$175.000:

```
# Criar um filtro para despesas acima de R$175.000
filtro = df_viagens['despesas'] > 175000
viagens_caras = df_viagens[filtro]

# Exibir resultados
print(viagens_caras)
```

Explicação do Código:

1) Criamos um filtro `df_viagens['despesas'] > 175000`.
2) Aplicamos esse filtro ao DataFrame para obter apenas as viagens que atendem ao critério.
3) Exibimos os resultados.

Saída esperada:

A saída do código exibe uma tabela com as viagens de maior custo, permitindo investigação mais detalhada.

Gráficos de distribuição

```
dados['Coluna1'].hist()
plt.show()
```

(O gráfico será exibido na tela.)

Correlação entre variáveis

```
print(dados.corr())
```

Saída esperada:

```
         Coluna1    Coluna2    Coluna3
Coluna1     1.000      0.985      0.970
Coluna2     0.985      1.000      0.990
Coluna3     0.970      0.990      1.000
```

Essas técnicas ajudam a entender melhor os dados antes de realizar análises mais complexas.

25. Salvando os Resultados e Criando a análise Final

Salvando Arquivos Gerados

Para salvar os arquivos gerados na análise, é necessário definir um caminho de saída. No script final, o caminho de saída pode ser configurado da seguinte forma:

```
caminho_saida = "D:/output/tabela.xlsx"
```

Caso a pasta `output` não exista, ela deve ser criada manualmente ou via código. Depois disso, o arquivo pode ser salvo em formato Excel:

```
df_final.to_excel(caminho_saida, index=False)
```

Para salvar em outros formatos, como CSV, pode-se utilizar:

```
df_final.to_csv(caminho_saida.replace(".xlsx", ".csv"), index=False)
```

O Pandas também permite salvar em bancos SQL e outros formatos.

Salvando Gráficos

Para salvar figuras geradas pelo `matplotlib`, o seguinte procedimento deve ser seguido:

1) Definir o caminho de saída:

```
caminho_figura = "D:/output/figura.png"
```

2) Gerar e salvar a figura:

```
import matplotlib.pyplot as plt

plt.savefig(caminho_figura, bbox_inches='tight')
```

Caso a imagem fique cortada, adicionar `bbox_inches='tight'` ajuda a ajustar automaticamente o espaço.

Automatização da Análise

Para tornar a análise flexível para diferentes anos, pode-se utilizar variáveis dinâmicas com `f-strings`:

```
ano = 2023
caminho_saida_tabela = f"D:/output/tabela_{ano}.xlsx"
caminho_saida_grafico = f"D:/output/grafico_{ano}.png"
```

Essa abordagem permite alterar um único valor (`ano`) para atualizar toda a análise.

Execução em Blocos

No Jupyter Notebook, é possível rodar todas as células seguidas pressionando `Shift + Enter`. Caso ocorra um erro, é necessário verificar as mensagens de erro para corrigir problemas no código.

Expansão da Automalização

O mesmo processo pode ser automatizado para:

- Processar todos os arquivos de uma pasta automaticamente.
- Gerar vários gráficos e tabelas sem intervenção manual.
- Automatizar downloads de dados via Python.

Essas funcionalidades permitem que análises de dados sejam realizadas de forma eficiente e escalável.

26. Não pare por aqui!

Resumo da Jornada

Durante este curso, você percorreu um caminho completo de análise de dados utilizando Python, abordando:

- Importação de dados
- Consolidação de tabelas
- Criação de filtros
- Tratamento de dados nulos
- Transformações de dados
- Geração de gráficos
- Salvamento de resultados

Potencial de Aplicação

Democratização do Aprendizado

Python não é exclusivo para:

- Programadores experientes
- Graduados em Ciência de Dados
- Especialistas em Computação

É uma ferramenta acessível para profissionais de diversas áreas.

Desafio Pessoal

Reflexão e Aplicação

Proponha-se a responder:

- Como posso aplicar Python no meu trabalho atual?
- Que processos podem ser automatizados?
- Quais análises posso realizar com os dados disponíveis?

Próximos Passos

Continuidade de Aprendizado

Áreas para aprofundamento:

- Python básico
- Análise de dados
- Machine Learning
- Banco de dados
- Visualização de dados
- Projetos práticos

Mensagem Final

A programação é uma habilidade desenvolvida com:

- Prática constante
- Curiosidade
- Abertura para aprendizado
- Persistência

Sua jornada em Python está apenas começando!