UFCD 5417
Lab 7 (`pip3`, `virtualenv`)
v1.3

Nelson Santos
`nelson.santos.0001376@edu.atec.pt`

ATEC — 27 de novembro de 2024

Conteúdo

1 Introdução

Este laboratório foi desenvolvido para ajudar os alunos a configurarem o seu ambiente `Python` para o projeto de Análise de Dados, bem como para introduzir as principais bibliotecas que serão utilizadas. Iremos também abordar como instalar e gerir bibliotecas com o `pip`, além de como configurar e utilizar ambientes virtuais em `Python` utilizando o `virtualenv`.

2 Gestor de bibliotecas `pip3`

O `pip3` é o gestor de bibliotecas padrão do `Python` 3, utilizado para instalar bibliotecas e dependências adicionais. Este permite que os utilizadores adicionem rapidamente ferramentas poderosas ao seu ambiente de desenvolvimento. Seguem alguns comandos básicos para a sua utilização:

2.1 Verificar se `pip3` está instalado

Antes de começar a utilizar o `pip3`, é importante verificar se está corretamente instalado no seu sistema operativo. Para isso, execute o seguinte comando no terminal (ou prompt de comando no Windows):

```
pip3 --version
```

Se o `pip3` estiver instalado, deverá aparecer a versão instalada e a referência ao `Python` 3. Se este comando retornar um erro, será necessário instalá-lo.

2.1.1 Instalação do `pip3` no Windows

Caso o `pip3` não esteja instalado no Windows, siga os seguintes passos:

1. Abra o tprompt de comando ou PowerShell
2. Execute o seguinte comando para garantir que o `pip3` seja instalado:

   ```
   python -m ensurepip --upgrade
   ```

3. Se o comando acima não funcionar, pode reinstalar o `Python` a partir do link: `https://www.python.org/downloads/`. Durante a instalação, assegure-se de selecionar a opção `Add Python to PATH` para que o `pip3` funcione corretamente no prompt de comando

2.2 Instalação de bibliotecas Globalmente e Localmente

O `pip3` permite instalar bibliotecas de duas formas: **globalmente** ou **localmente**.

- **Instalação global**: Por omissão, quando se executa o comando `pip3 install` sem qualquer configuração adicional, as bibliotecas são instaladas na pasta de bibliotecas do `Python` disponível para todo o sistema operativo, ou seja, **globalmente**. Estas bibliotecas ficam acessíveis para qualquer utilizador ou projeto
- **Instalação local (em ambientes virtuais)**: Para instalar bibliotecas localmente e de forma isolada num projeto específico, recomenda-se o uso de **ambientes virtuais**. Um ambiente virtual cria uma pasta (ou ambiente) separada para guardar as bibliotecas apenas para aquele projeto, evitando conflitos outras bibliotecas instaladas globalmente. Esta é a abordagem recomendada para projetos mais complexos ou quando se trabalha em equipa, pois garante que todos os membros do projeto estão a usar as mesmas versões de bibliotecas

  Para mais detalhes sobre a criação e utilização de ambientes virtuais, veja a secção 3.

2.3 Instalação de bibliotecas

Para instalar uma biblioteca, utiliza-se o comando `pip3 install`. Por exemplo, para instalar a biblioteca `pandas`, usada para manipulação de dados, pode-se executar:

```
pip3 install pandas
```

2.4 Listar bibliotecas instaladas

Se desejar ver uma lista de todas as bibliotecas instalados no seu ambiente, utilize o seguinte comando:

```
pip3 list
```

2.5 Atualização de bibliotecas

Para atualizar uma biblioteca instalada, por exemplo `matplotlib`, executar o comando:

```
pip3 install --upgrade pandas
```

2.6 Remoção de bibliotecas

Para remover bibliotecas que não são mais necessárias, usa-se o comando `pip3 uninstall`. Exemplo:

```
pip3 uninstall pandas
```

3 Ambientes Virtuais com `virtualenv`

Um ambiente virtual em `Python` permite criar um espaço isolado para as bibliotecas de um projeto, evitando conflitos entre dependências de diferentes projetos. A ferramenta mais comum para esta tarefa é o `virtualenv`.

3.1 Verificar se `virtualenv` está instalado

Antes de utilizar o `virtualenv`, é necessário verificar se está corretamente instalado no sistema. Para isso, deve-se executar o seguinte comando no terminal:

```
virtualenv --version
```

Se o `virtualenv` estiver instalado, será exibida a sua versão. Caso contrário, será necessário instalá-lo (veja a próxima subsecção).

3.2 Instalar o `virtualenv` com `pip3`

Caso o `virtualenv` não esteja instalado, pode ser instalado utilizando o `pip3`, o gestor de bibliotecas do `Python`. Para tal, execute o seguinte comando:

```
pip3 install virtualenv
```

3.3 Criar um Ambiente Virtual

Depois de o `virtualenv` estar instalado, pode-se criar um ambiente virtual com o seguinte comando:

```
virtualenv nome_do_ambiente
```

Substitua `nome_do_ambiente` por um nome descritivo para o projeto. Este comando cria uma pasta contendo o ambiente virtual, onde as bibliotecas instaladas estarão isolados do resto do sistema.

Poderá entrar dentro da pasta do projetos e verificar que foram adicionadas algumas pastas dentro deste: `lib, bin e pyenv.cfg`.

Estas pastas contêm as bibliotecas instaladas, os executáveis e a configuração do ambiente virtual, respetivamente. As bibliotecas instaladas neste ambiente virtual não afetarão as bibliotecas globais do sistema.

3.4 Ativar o Ambiente Virtual

Para começar a utilizar o ambiente virtual criado, é necessário ativá-lo. O comando para ativar o ambiente virtual depende do sistema operativo:

- **Windows**:

```
nome_do_ambiente\Scripts\activate
```

- **Linux/macOS**:

```
source nome_do_ambiente/bin/activate
```

Após ativar o ambiente, o nome do ambiente aparecerá no terminal, indicando que se está a trabalhar nesse ambiente virtual.

3.5 Desativar o Ambiente Virtual

Para desativar o ambiente virtual e voltar ao ambiente `Python` global, utiliza-se o seguinte comando:

```
deactivate
```

3.6 Instalar bibliotecas num ambiente virtual

Depois de ativar o ambiente virtual, as bibliotecas podem ser instalados como habitualmente com o `pip3`, mas estarão isolados do ambiente global do sistema. Por exemplo:

```
pip3 install pandas matplotlib seaborn
```

Poderá confirmar a instalação das bibliotecas anteriormente instaladas através do comando:

```
pip3 list
```

Todos as bibliotecas instaladas neste ambiente serão específicos para este projeto, sem afetar outros projetos ou bibliotecas instaladas globalmente. Isto é especialmente importante quando se trabalha em equipa ou em vários projetos simultaneamente, pois garante que as versões das bibliotecas utilizadas não entram em conflito. Cada developer pode ter o seu próprio ambiente de desenvolvimento isolado, evitando problemas de compatibilidade e facilitando a colaboração, já que todos terão um conjunto de bibliotecas consistente para o projeto em questão.

3.7 `requirements.txt` e `virtualenv`

Num projeto `Python`, a gestão de dependências é essencial para garantir que as bibliotecas utilizadas sejam compatíveis entre si e que o ambiente de desenvolvimento seja reproduzível em diferentes sistemas, como já explicado anteriormente. O ficheiro `requirements.txt`, em conjunto com a utilização de um ambiente virtual criado com `virtualenv`, desempenha um papel fundamental para atingir estes objetivos.

O `requirements.txt` permite documentar as bibliotecas e respetivas versões necessárias para o projeto, enquanto o `virtualenv` assegura que essas dependências são instaladas num ambiente isolado, evitando conflitos com outras aplicações ou configurações globais do sistema.

3.7.1 Requirements.txt

O ficheiro `requirements.txt` contém uma lista de todas as dependências necessárias para o projeto, frequentemente acompanhadas das versões específicas ou intervalos compatíveis. Um exemplo típico deste ficheiro seria:

```
flask==2.2.3
pandas>=1.4.0
numpy~=1.23.1
```

Este ficheiro é utilizado para instalar rapidamente todas as dependências necessárias de forma automática.

3.7.2 Workflow entre `requirements.txt` e `virtualenv`

- **Instalação de dependências:** Após criar e ativar um ambiente virtual com o `virtualenv`, o ficheiro `requirements.txt` é utilizado para instalar todas as bibliotecas necessárias através do comando:

```
pip install -r requirements.txt
```

- **Exportação de dependências:** Se o ambiente virtual já estiver configurado, as dependências instaladas podem ser exportadas para um ficheiro `requirements.txt` com o comando:

```
pip freeze > requirements.txt
```

- **Isolamento:** O `virtualenv` assegura que as bibliotecas instaladas a partir do `requirements.txt` ficam isoladas, evitando conflitos com outras versões de pacotes instalados no sistema global ou noutros projetos

Desta forma, o `requirements.txt` e o `virtualenv` trabalham em conjunto para garantir um ambiente de desenvolvimento eficiente e controlado.

4 Bibliotecas Importantes para Análise de Dados

Esta secção será dedicada a apresentar algumas das bibliotecas mais importantes para a análise de dados em `Python`. Estas bibliotecas são amplamente utilizadas em projetos de *data science*, aprendizagem automática, inteligência artificial, visualização de dados, etc.

4.1 `pandas`

A biblioteca `pandas` é amplamente utilizada para manipulação e análise de dados em `Python`. Permite carregar dados de vários formatos (CSV, Excel, JSON, etc.), realizar operações de limpeza, transformação e análise de dados, e exportar os resultados para ficheiros.

4.1.1 Exemplo de Utilização

Crie uma ambiente virtual, adicione o `requirements.txt`, e sigua o exemplo abaixo para carregar um ficheiro CSV e análise de dados. Instale as dependências necessárias.

```
import pandas as pd

# Carregar o ficheiro CSV
    df = pd.read_csv('vendas_loja.csv')

    # Verificar a existencia de valores nulos em todas as colunas
    print("Valores nulos por coluna antes do preenchimento:")
    print(df.isna().sum())

    # Preencher valores nulos na coluna "Quantidade" com a media dessa coluna
    df['Quantidade'].fillna(df['Quantidade'].mean(), inplace=True)

    # Preencher valores nulos na coluna "Preco_Unitario" com a media dessa coluna
    df['Preco_Unitario'].fillna(df['Preco_Unitario'].mean(), inplace=True)

    # Adicionar uma nova coluna "coluna_numerica" como exemplo (calculando Total a partir
     de Quantidade * Preco_Unitario)
    df['coluna_numerica'] = df['Quantidade'] * df['Preco_Unitario']

    # Preencher valores nulos na nova coluna "coluna_numerica" (se houver)
    df['coluna_numerica'].fillna(df['coluna_numerica'].mean(), inplace=True)

    # Verificar novamente se ha valores nulos apos o preenchimento
    print("\nValores nulos por coluna apos o preenchimento:")
    print(df.isna().sum())

    # Exibir as primeiras linhas do DataFrame atualizado
    print("\nPrimeiras linhas do DataFrame atualizado:")
    print(df.head())
```

```
    # Guardar o DataFrame atualizado num novo ficheiro CSV
    df.to_csv('vendas_loja_atualizado.csv', index=False)
```

Este exemplo mostra como trabalhar com dados reais, lidando com valores ausentes e exportando o resultado para outro ficheiro CSV.

4.2 `matplotlib` e `seaborn`

As bibliotecas `matplotlib` e `seaborn` são essenciais para a visualização de dados em `Python`. O `matplotlib` fornece uma interface de baixo nível para criar gráficos e visualizações personalizadas, enquanto o `seaborn` constrói sobre o `matplotlib` e facilita a criação de visualizações estatísticas atraentes e informativas.

4.2.1 Exemplo de Utilização

No exemplo a seguir, utilizamos as bibliotecas `pandas`, `matplotlib` e `seaborn` para carregar um conjunto de dados de vendas e gerar diferentes tipos de visualizações.

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

# Carregar o ficheiro CSV
df = pd.read_csv('vendas_loja.csv')

# Configurar o estilo do seaborn
sns.set(style='whitegrid')

# Criar um grafico de barras para visualizar as vendas por produto
plt.figure(figsize=(10, 6))
sns.barplot(x='Produto', y='Total_Venda', data=df, estimator=sum)
plt.title('Total de Vendas por Produto')
plt.xlabel('Produto')
plt.ylabel('Total de Vendas')
plt.xticks(rotation=45)
plt.tight_layout()

# Exibir o grafico
plt.show()

# Criar um grafico de dispersao para visualizar a relacao entre Preco Unitario e
    Quantidade Vendida
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Preco_Unitario', y='Quantidade', data=df, hue='Produto', palette='deep
    ')
plt.title('Relacao entre Preco Unitario e Quantidade Vendida')
plt.xlabel('Preco Unitario')
plt.ylabel('Quantidade Vendida')
plt.legend(title='Produto')
plt.tight_layout()

# Exibir o grafico
plt.show()

# Criar um grafico de linhas para visualizar a tendencia de vendas ao longo do tempo
df['Data'] = pd.to_datetime(df['Data'])  # Converter a coluna 'Data' para o formato de
    data
plt.figure(figsize=(10, 6))
sns.lineplot(x='Data', y='Total_Venda', data=df, marker='o')
plt.title('Tendencia de Vendas ao Longo do Tempo')
plt.xlabel('Data')
plt.ylabel('Total de Vendas')
plt.xticks(rotation=45)
plt.tight_layout()

# Exibir o grafico
plt.show()
```

Neste exemplo, utilizamos `seaborn` para criar um gráfico de barras que mostra o total de vendas por produto, um gráfico de dispersão que explora a relação entre o preço unitário e a quantidade vendida,

e um gráfico de linhas que revela a tendência de vendas ao longo do tempo. Essas visualizações ajudam a compreender melhor os dados e a identificar padrões significativos. A personalização e a estética aprimorada do `seaborn` tornam o processo de visualização mais intuitivo e eficaz.

4.3 Desafios

4.3.1 Desafio 1: Gráfico Circular

Deverão construir um gráfico circular para visualizar a proporção das vendas totais de cada produto.

Bom trabalho!