

Ficha de Avaliação Final	
Curso:	UFCD 10793
UFCD/Módulo/Temática:	UFCD 10793 - Fundamentos de Python
Ação:	10793_2/AT & 10793_5/N
Formador/a:	Sandra Liliana Meira de Oliveira
Data:	Março de 2025
Nome do Formando/a:	

O presente documento é composto por uma parte teórica e uma parte prática. A resolução da parte prática corresponde à realização da Ficha de Avaliação Final.

Tratamento e Análise de Dados com PANDAS	2
Visualização de Dados.....	8
Atividade 1 – Biblioteca Seaborn (5 valores).....	10
Atividade 2 (5 valores)	14
Atividade 3 (5 valores)	15
Atividade 4 (5 valores)	15

Teoria

Tratamento e Análise de Dados com PANDAS

Um estudo completo de limpeza de dados usando pandas inclui várias etapas:

1. Exploração dos Dados: A primeira etapa é explorar os dados para entender sua estrutura, tipos de variáveis e a presença de valores ausentes ou duplicados. Isso pode ser feito usando funções do pandas como `head()`, `info()`, `describe()` e `value_counts()`.
2. Tratamento de Valores Ausentes: Após entender os dados, a próxima etapa é tratar os valores ausentes. Isso pode ser feito excluindo os valores ausentes, substituindo-os por um valor específico ou imputando os valores ausentes com um método estatístico como média, mediana ou moda. É importante considerar o efeito de excluir ou imputar valores ausentes na análise ou aplicação para a qual os dados serão utilizados.
3. Tratamento de Valores Duplicados: Após tratar os valores ausentes, a próxima etapa é tratar os valores duplicados. Isso pode ser feito excluindo os valores duplicados ou mantendo somente uma cópia de cada valor duplicado.
4. Preparação das Variáveis: Depois de tratar os valores ausentes e duplicados, a próxima etapa é preparar as variáveis para a análise ou aplicação. Isso pode incluir trabalhar com variáveis categóricas, normalizando variáveis numéricas ou criando novas variáveis a partir das existentes.
5. Exportar os Dados Limpos: Finalmente, é importante exportar os dados limpos para um formato apropriado para a análise ou aplicação, como CSV ou Excel.

Um procedimento para realizar uma limpeza de dados usando pandas inclui os seguintes passos:

1. Importar os dados: Use a função `pd.read_csv()` ou `pd.read_excel()` para importar os dados de um arquivo CSV ou Excel, respectivamente.
2. Analisar as informações gerais dos dados: Utilize as funções `head()`, `info()`, `describe()` e `value_counts()` para obter uma visão geral dos dados e identificar problemas como valores ausentes, tipos de variáveis incorretos e outliers.

3. Tratar valores ausentes: Utilize as funções `dropna()` ou `fillna()` para remover ou preencher valores ausentes, respectivamente.
4. Tratar valores duplicados: Utilize a função `drop_duplicates()` para remover valores duplicados.
5. Renomear colunas: Utilize a função `rename()` para renomear colunas para nomes mais claros e significativos.
6. Converter tipos de variáveis: Utilize as funções `astype()` ou `to_datetime()` para converter tipos de variáveis como strings para inteiros, flutuantes ou datas.
7. Filtrar linhas: Utilize a notação de colchetes para selecionar apenas as linhas que atendem a certas condições.
8. Agrupar e resumir dados: Utilize a função `groupby()` para agrupar dados por uma ou mais colunas e aplicar funções de agregação como `mean()`, `sum()` ou `count()`.
9. Salvar dados limpos: Utilize as funções `to_csv()` ou `to_excel()` para salvar os dados limpos

A função `astype()` e outras funções de conversão

A função `astype()` em pandas é usada para converter um tipo de dados de uma coluna ou série. Ela é muito útil quando os dados importados estão em um formato incorreto ou queremos fazer uma conversão de tipo antes de realizar uma análise.

```
import pandas as pd


# cria um exemplo de dataframe
df = pd.DataFrame({'A': [1, 2, 3], 'B': ['4', '5', '6']})

# mostra o tipo de dados antes da conversão
print(df.dtypes)

# converte a coluna 'B' para inteiro
df['B'] = df['B'].astype(int)

# mostra o tipo de dados depois da conversão
print(df.dtypes)
```

Saída:

 Copy code

```
A    int64
B    object
dtype: object
A    int64
B    int32
dtype: object
```

A função `astype()` pode ser aplicada a um `DataFrame` ou a uma série específica. Pode converter entre vários tipos, como `int`, `float`, `bool`, `datetime` e `categorical`.

```
df["A"] = df["A"].astype(float)
```

A biblioteca `pandas` possui várias funções para converter tipos de dados, algumas delas são:

- `astype()`: converte um tipo de dados de uma coluna ou série. Exemplo: ``df["A"] = df["A"].astype(float)``
- `to_datetime()`: converte uma coluna ou série de strings para tipo `datetime`. Exemplo: ``df["date"] = pd.to_datetime(df["date"])``
- `to_numeric()`: converte uma coluna ou série de strings para tipo numérico (`int` ou `float`). Exemplo: ``df["value"] = pd.to_numeric(df["value"])``
- `to_timedelta()`: converte uma coluna ou série de strings para tipo `timedelta`. Exemplo: ``df["duration"] = pd.to_timedelta(df["duration"])``
- `to_period()`: converte uma coluna ou série de strings para tipo `period`. Exemplo: ``df["quarter"] = pd.to_period(df["quarter"])``
- `to_categorical()`: converte uma coluna ou série para tipo `categorical`. Exemplo: ``df["status"] = pd.to_categorical(df["status"])``

Essas funções são muito úteis quando os dados importados estão em um formato incorreto ou queremos fazer uma conversão de tipo antes de realizar uma análise. É importante verificar se a conversão de tipo não cria problemas na análise ou aplicação para a qual os dados serão usados.

A função groupby()

A função `groupby()` do pandas é usada para agrupar um `DataFrame` por um ou mais níveis de uma coluna ou série específica e aplicar funções de agregação aos dados agrupados. É uma das funções mais poderosas e flexíveis dos pandas, permitindo realizar várias operações de análise de dados em um único passo.

Exemplo de utilização:

```
import pandas as pd

# importa o arquivo csv
df = pd.read_csv("sales_data.csv")

# agrupa os dados pela coluna "region" e aplica a função de soma à coluna "sales"
sales_by_region = df.groupby("region")["sales"].sum()

# imprime os dados agrupados
print(sales_by_region)
```

Na entrada do ficheiro csv:

```
date,region,sales
2020-01-01,East,100
2020-01-02,West,120
2020-01-03,East,90
2020-01-04,West,110
2020-01-05,East,80
2020-01-06,West,100
```

A função `groupby()` também pode ser usada para agrupar dados com base em várias colunas ao mesmo tempo, utilizando uma lista de colunas. Por exemplo:

```
# agrupa os dados pela coluna "region" e "product" e aplica a função de soma à
coluna "sales"
sales_by_region_product = df.groupby(["region","product"])["sales"].sum()
```

Além disso, é possível aplicar várias funções de agregação diferentes aos dados agrupados, como `mean()`, `min()`, `max()`, entre outras.

A função `groupby()` também pode ser combinada com outras funções do pandas, como `filter()`, `aggregate()` e `transform()` para realizar análises mais complexas.

Aqui está um exemplo de como os dados de "sales_by_region_product" poderiam ser apresentados após a utilização da função `groupby()`:

```
region product
East  Product A    270
      Product B    260
West  Product A    330
      Product B    310
Name: sales, dtype: int64
```

Neste exemplo, os dados foram agrupados primeiramente pela coluna "region" e depois pela coluna "product". A função `sum()` foi aplicada à coluna "sales" para calcular a soma total das vendas para cada combinação de "region" e "product". Os dados são apresentados como uma série pandas, onde o índice é composto pelas colunas agrupadas "region" e "product" e o valor é a soma das vendas.

A função `filter()`

A função `filter()` do pandas é usada para filtrar linhas de um DataFrame baseado em uma condição específica. Ela retorna um novo DataFrame com as linhas que satisfazem a condição dada.

Exemplo de utilização:

```
import pandas as pd

# importa o arquivo csv
df = pd.read_csv("sales_data.csv")

# filtra as linhas onde a coluna "sales" é maior que 100
sales_above_100 = df.filter(df["sales"] > 100)

# imprime os dados filtrados
print(sales_above_100)
```

Na entrada do ficheiro csv:

```
date,region,product,sales
2020-01-01,East,Product A,90
2020-01-02,West,Product B,120
2020-01-03,East,Product A,110
2020-01-04,West,Product B,100
2020-01-05,East,Product A,80
2020-01-06,West,Product B,90
```

A função `filter()` pode ser usada para filtrar dados com base em várias condições. Por exemplo:

```
# filtra as linhas onde a coluna "sales" é maior que 100 e a coluna "region" é
igual a "East"
sales_above_100_east = df.filter((df["sales"] > 100) & (df["region"] == "East"))
```

A função `filter()` também pode ser usada para filtrar dados com base em uma função personalizada. Por exemplo:

```
# filtra as linhas onde a coluna "date" está entre 2020-01-02 e 2020-01-05
date_range = df.filter(lambda x: (x["date"] >= "2020-01-02") & (x["date"] <=
"2020-01-05"))
```

A função `filter()` é uma ótima ferramenta para selecionar linhas específicas de um `DataFrame` baseado em condições complexas. Ela é particularmente útil quando se trabalha com grandes conjuntos de dados e é necessário selecionar apenas uma parte dos dados para análise.

A saída dos exemplos acima seria uma tabela com as linhas que satisfazem as condições específicas. No primeiro exemplo, a saída seria todas as linhas onde a coluna "sales" tem valores maiores que 100, no segundo exemplo as linhas onde "sales" é maior que 100 e "region" é igual a "East" e no terceiro exemplo as linhas onde a coluna "date" está entre as datas indicadas na condição.

Técnicas para tratar valores ausentes

Existem várias técnicas para tratar valores ausentes em dados, cada uma com suas próprias vantagens e desvantagens. Algumas técnicas comuns incluem:

1. Remover as linhas ou colunas que contêm valores ausentes: Essa técnica é útil se os valores ausentes representarem uma pequena percentagem dos dados e não fornecerem informações importantes.

2. Substituir os valores ausentes com a média, mediana ou moda: Essa técnica é útil quando os valores ausentes são distribuídos de forma aleatória e não têm uma distribuição assimétrica.
3. Utilizar algoritmos de machine learning para prever os valores ausentes: Essa técnica é útil quando existem relações complexas entre as variáveis e a quantidade de dados ausentes é grande.
4. Utilizar técnicas de imputação estatística como imputação múltipla ou imputação por regressão.

Em geral, a escolha da técnica de tratamento de valores ausentes dependerá do conjunto de dados e do objetivo do estudo. É importante realizar uma avaliação dos dados e do impacto dos valores ausentes no modelo e resultado final.

Visualização de Dados

O Python oferece várias bibliotecas para visualização de dados:

1. **Matplotlib:** oferece uma grande variedade de gráficos 2D e 3D. É amplamente utilizada para criar gráficos de linhas, gráficos de dispersão, gráficos de barras e histogramas.
2. **Seaborn:** Seaborn é uma biblioteca de visualização de dados baseada em Matplotlib. Oferece um interface mais apelativo para criar gráficos estatísticos mais atraentes e informativos.
3. **Plotly:** é uma biblioteca interativa de visualização de dados que permite aos utilizadores criar gráficos e painéis interativos. Pode ser utilizada para criar gráficos 2D e 3D, gráficos de dispersão, gráficos de linhas, mapas de calor e muito mais.
4. **Bokeh:** é uma biblioteca Python para criar visualizações interativas para browsers. Oferece uma forma flexível e poderosa de criar visualizações dinâmicas com fluxos de dados, grandes conjuntos de dados e atualizações em tempo real.
5. **ggplot:** é uma implementação Python do popular pacote *R ggplot2*, que fornece uma gramática de gráficos para criar visualizações de dados. É projetada para tornar fácil a criação de visualizações esteticamente atraentes e informativas.
6. **Altair:** é uma biblioteca de visualização estatística declarativa. Permite aos utilizadores criar, facilmente, visualizações interativas usando uma sintaxe concisa e intuitiva.

Matplotlib e Seaborn são duas das bibliotecas de visualização de dados mais comumente usadas em Python. Fornecem uma ampla gama de funcionalidades para criar uma variedade de gráficos 2D e 3D. Matplotlib é uma biblioteca de baixo nível que fornece muita flexibilidade e

controlo sobre os detalhes do gráfico, enquanto Seaborn é uma biblioteca de nível superior que torna mais fácil criar gráficos estatísticos atraentes e informativos.

Além de Matplotlib e Seaborn, Plotly e Bokeh também são bibliotecas populares para criar visualizações interativas. Plotly é particularmente adequado para criar gráficos e painéis interativos, enquanto Bokeh é projetado para criar visualizações interativas para navegadores da web modernos.

A escolha da biblioteca depende das necessidades específicas do projeto e dos tipos de visualizações necessárias. Algumas bibliotecas podem ser mais adequadas para determinados tipos de dados ou visualizações, e algumas podem ser mais fáceis de usar do que outras, dependendo do nível de experiência do utilizador com Python e visualização de dados.

Deixo aqui os links para a documentação das duas bibliotecas de visualização de dados a usar ao longo deste trabalho.

MatPlotLib: <https://matplotlib.org/>

Seaborn: <https://seaborn.pydata.org/>

Para além do material fornecido nas sessões anteriores recorre aos seguintes tutoriais/bibliotecas para aprofundares os conhecimentos das bibliotecas:

<https://www.geeksforgeeks.org/python-plotly-tutorial/>

<https://plotly.com/python/plotly-fundamentals/>

<https://seaborn.pydata.org/tutorial.html>

<https://www.geeksforgeeks.org/python-seaborn-tutorial/>

<https://matplotlib.org/stable/tutorials/introductory/pyplot.html>

<https://www.geeksforgeeks.org/matplotlib-tutorial/>

Explore também o link seguinte. Ajuda a perceber qual o gráfico pretendido dependendo da análise que se pretende efetuar.

<https://www.data-to-viz.com/>

Prática

As seguintes atividades têm como objetivo a configuração e a utilização de algumas bibliotecas de Python no processo de Tratamento, Análise e Visualização de Dados.

A concretização deste projeto implica a realização de todas as atividades práticas abaixo indicadas. Os ficheiros resultantes deverão ser anexados à tarefa de avaliação final.

Atividade 1 – Biblioteca Seaborn (5 valores)

Esta atividade é uma atividade orientada, que implica, apenas, a reprodução do código, de forma a entrar em contato com a biblioteca Seaborn.

1. Lê o tutorial da biblioteca em <https://seaborn.pydata.org/tutorial.html>
2. Instala a biblioteca seaborn.
3. Cria um jupyter notebook com o nome **seabornguide**. Ao longo desta atividade, serão utilizados os datasets pré-existent na biblioteca
4. No ficheiro anterior coloca o seguinte código que irá permitir importar as bibliotecas a utilizar

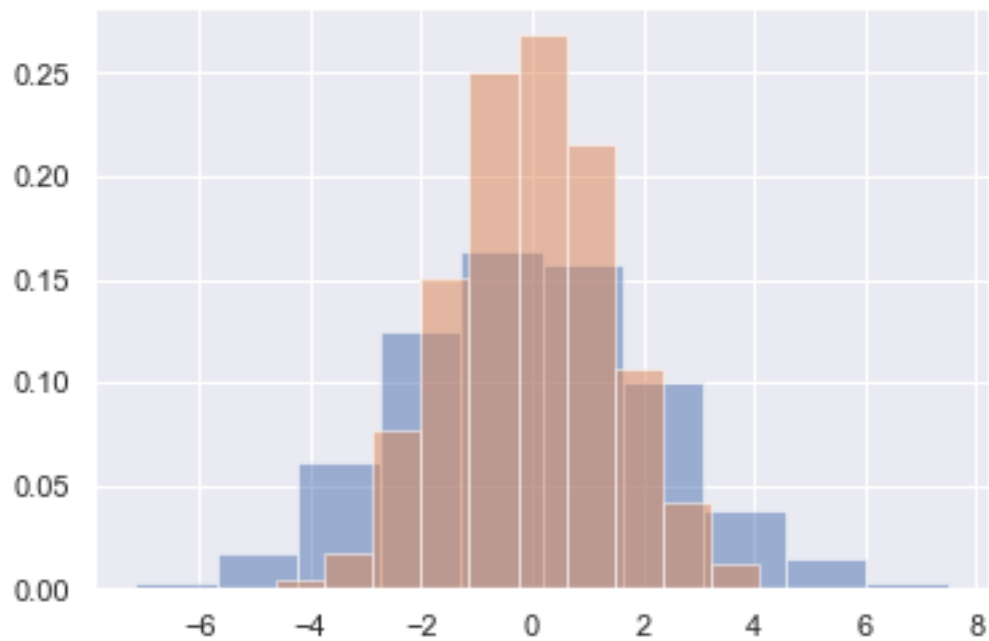
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

sns.set() # seaborn's method to set its chart style
```

5. Histograms, KDE, and Densities:

```
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

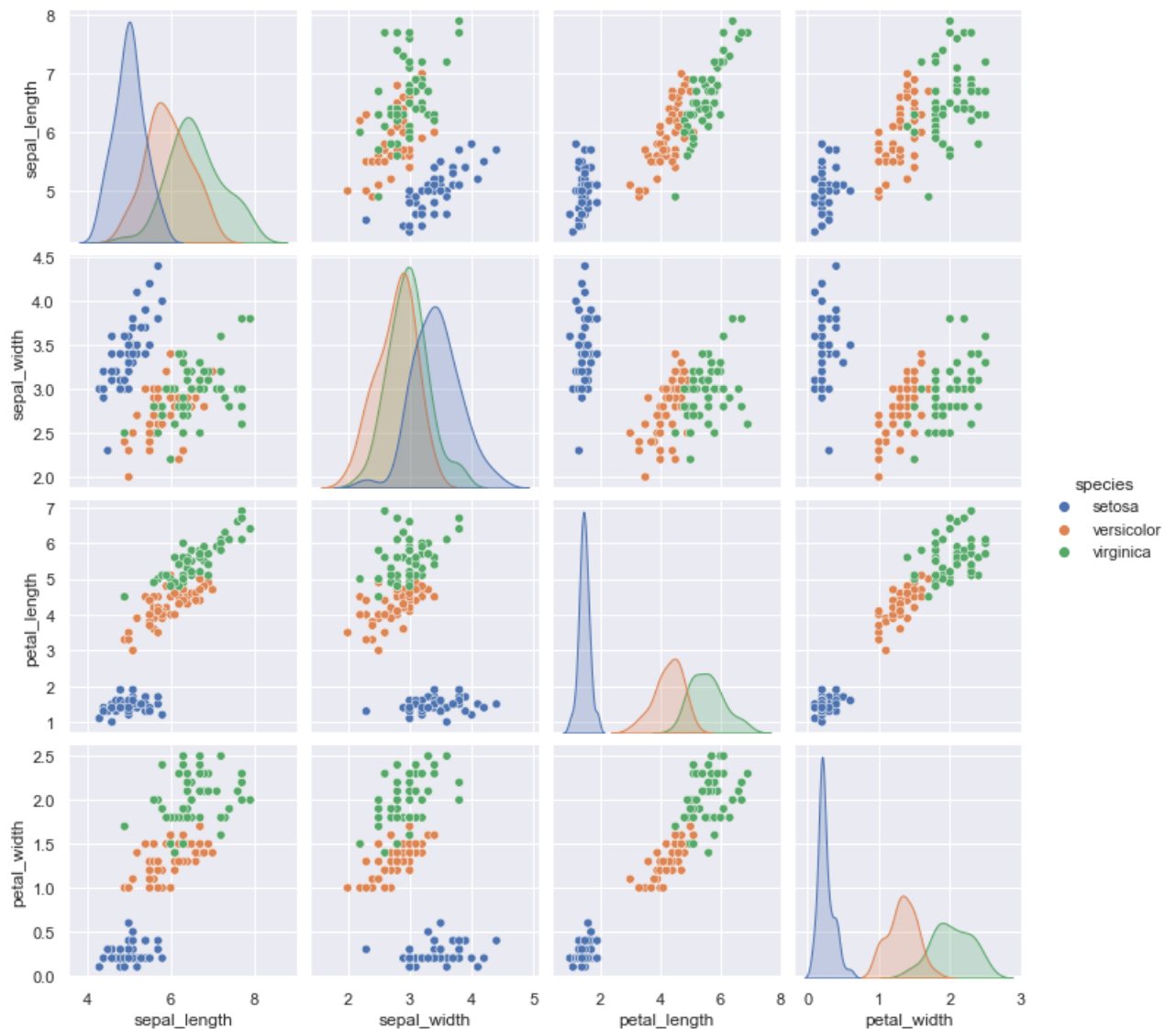
for col in 'xy':
    plt.hist(data[col], density=True, alpha=0.5)
```



6. Pair Plots

```
iris = sns.load_dataset("iris")
iris.head()

sns.pairplot(iris, hue='species', height=2.5);
```

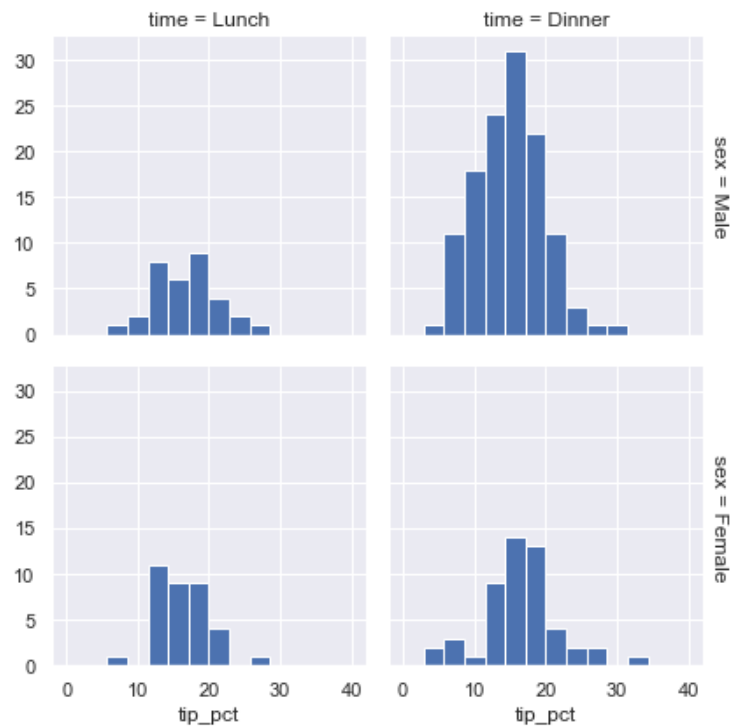


7. Histogramas

```
tips = sns.load_dataset('tips')
tips.head()

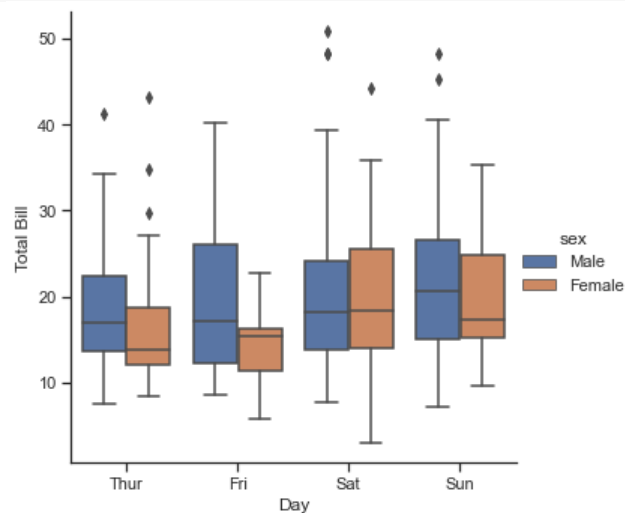
tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']

grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```



8. Categorical Plots

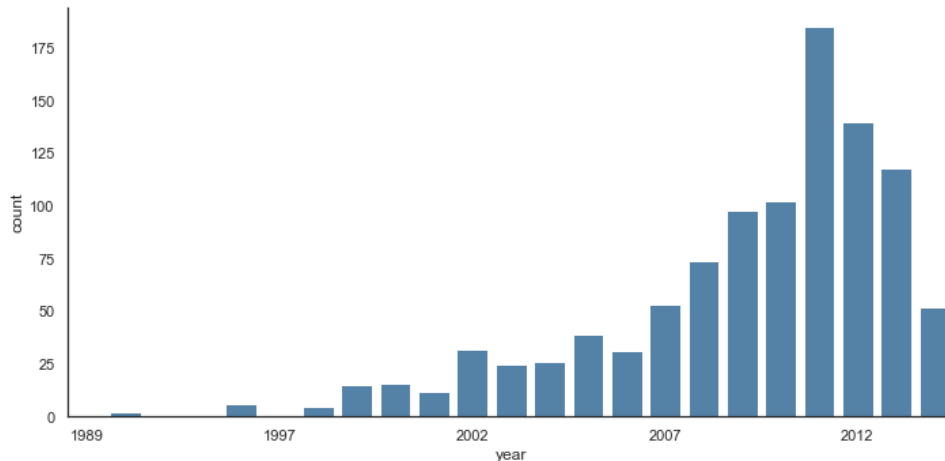
```
with sns.axes_style(style='ticks'):
    g = sns.catplot(x="day", y="total_bill", hue="sex", data=tips, kind="box")
    g.set_axis_labels("Day", "Total Bill");
```



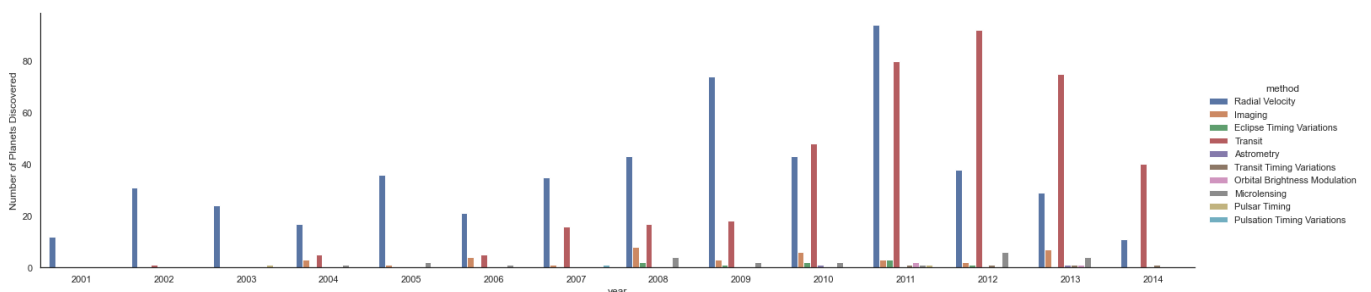
9. Bar Plots

```
planets = sns.load_dataset('planets')
planets.head()
```

```
with sns.axes_style('white'):
    g = sns.catplot(x="year", data=planets, aspect=2,
                    kind="count", color='steelblue')
    g.set_xticklabels(step=5)
```



```
with sns.axes_style('white'):
    g = sns.catplot(x="year", data=planets, aspect=4.0, kind='count',
                    hue='method', order=range(2001, 2015))
    g.set_ylabels('Number of Planets Discovered')
```



Atividade 2 (5 valores)

Tendo por base o dataset **Airplane_Crashes_and_Fatalities_Since_1908.csv** e **os ficheiros exemplos fornecidos**, elabora um notebook jupyter no qual efetues:

- Limpeza e tratamento de dados;
- Processamento de dados: groupby, filtering, criação de novas colunas,....;
- Visualização de dados;

Atividade 3 (5 valores)

Tendo por base o dataset **ecommerce_customers.csv** e os **ficheiros exemplo fornecidos**, elabora um notebook jupyter no qual efetues:

- Limpeza e tratamento de dados;
- Processamento de dados: groupby, filter, criação de novas colunas,...;
- Visualização de dados;

Atividade 4 (5 valores)

Tendo por base o dataset **netflix1.csv** e os **ficheiros exemplo fornecidos ou um dataset à tua escolha**, elabora um notebook jupyter no qual efetues:

- Limpeza e tratamento de dados;
- Processamento de dados: groupby, filter, criação de novas colunas,...;
- Visualização de dados;

Bom Trabalho