NÃO PODE FALTAR

VISUALIZAÇÃO DE DADOS EM PYTHON

Vanessa Cadan Scheffer

BIBLIOTECAS E FUNÇÕES PARA CRIAÇÃO DE GRÁFICOS

Para a criação de gráficos em Python são utilizadas as bibliotecas matplotib e outras baseadas na matplotlib, e também funções que permitem criar e personalizar os gráficos.

Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

INTRODUÇÃO A VISUALIZAÇÃO DE DADOS EM PYTHON

Visualização de dados ou DataViz é um dos pilares dos profissionais que trabalham com dados. Existem profissionais que se especializam e inclusive tiram certificações para atuar nessa área específica. Após uma análise de dados e extração de informações é interessante que a entrega de resultados para a área de negócios seja feita de maneira visual, ou seja, por meio de gráficos. Um gráfico bem elaborado "fala" por si só e ajuda aos que assistem a entenderem os resultados.

A linguagem Python, conta com uma série de bibliotecas que permitem a criação de gráficos, os quais podem ser estáticos (sem iteração) ou dinâmicos, que apresentam iteração, como por exemplo, responder a eventos de clique do mouse. Nessa aula, nossa objetivo é apresentar um pouco desse universo de possibilidades.

MATPLOTLIB

Ao se falar em criação de gráficos em Python, o profissional precisa conhecer a biblioteca **matplotlib**, pois diversas outras são construídas a partir desta. A criação e grande parte da evolução dessa biblioteca se deve a John Hunter, que a desenvolveu como uma opção ao uso dos softwares gnuplot e MATLAB (MCGREGGOR, 2015). Com a utilização da linguagem Python na área científica para trabalhar com dados, após a extração dos resultados, o cientista precisava criar seus gráficos nos outros softwares mencionados, o que se tornava incoveniente, motivando a criação da biblioteca em Python.

A instalação da biblioteca pode ser feita via pip install: `pip install matplotlib`, lembrando que em ambientes como o projeto Anaconta e o Colab esse recurso já está disponível. O módulo **pyplot** possui uma coleção de funções que permitem criar e personalizar os gráficos
(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html). Existem duas sintaxes que são amplamente adotadas para importar essa biblioteca para o projeto:

- `import matplotlib.pyplot as plt`
- `from matplotlib import pyplot as plt`

Em ambas formas de importação utilizamos o apelido "plt" que é uma convenção adotada para facilitar o uso das funções. Ao trabalharmos no jupyter notebook com o kernel do IPython (o kernel do IPython é o backend de execução do Python para o Jupyter), podemos habilitar uma opção de fazer a impressão do gráfico "inline", ou seja, no próprio notebook. Essa opção faz parte do "Built-in magic commands", cuja documentação pode ser acessada no endereço https://ipython.readthedocs.io/en/stable/interactive/magics.html. Para habilitar utiliza-se a sintaxe: `%matplotlib inline`. Portanto, projetos no jupyter notebook, que utilizem o matplotlib sempre terão no começo, os comandos a seguir.

In [1]:

```
from matplotlib import pyplot as plt


%matplotlib inline
```

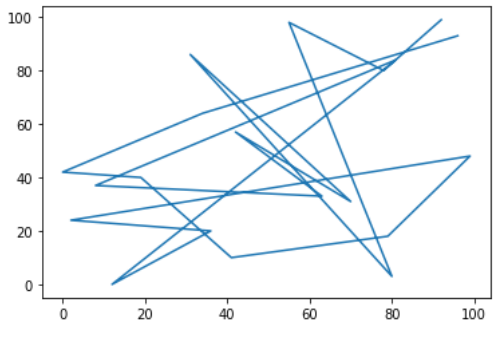
Os gráficos são uma forma visual de "contar" a história dos dados. Em breve contaremos várias histórias, mas como nosso primeiro gráfico, que tal um pouco de arte abstrata? Vamos criar duas listas aleatórias de valores inteiros com o módulo random e então plotar um gráfico de linhas, com a função `plt.plot()` do módulo pyplot. Veja a seguir, após criar duas listas com valores aleatórios, a função plot() as recebe como parâmetros, utilizando-as para os valores dos eixos horizontal (x) e vertical (y) e já cria o gráfico. Mágico não? Mas como "ele" sabia que era para criar um gráfico de linhas? Por que colocou a cor azul? Por que fez desse tamanho? Todos esses parâmetros e muitos outros, podem ser configurados!

In [2]:

```
import random


dados1 = random.sample(range(100), k=20)

dados2 = random.sample(range(100), k=20)


plt.plot(dados1, dados2) # pyplot gerencia a figura e o eixo
```

Out[2]:

```
[<matplotlib.lines.Line2D at 0x1cf32995940>]
```



Existem essencialmente duas maneiras de usar o Matplotlib:

1. Deixar para o pyplot criar e gerenciar automaticamente figuras e eixos, e usar as funções do pyplot para plotagem.
2. Criar explicitamente figuras e eixos e chamar métodos sobre eles (o "estilo orientado a objetos (OO)").

No gráfico que criamos, nós utilizamos a opção 1, ou seja, foi o próprio módulo que criou o ambiente da figura e do eixo. Ao utilizar a segunda opção, podemos criar uma figura com ou sem eixos, com a função `plt.subplots()`, que quando invocada sem parâmetros, cria um layout de figura com 1 linha e 1 coluna.

FIGURA COM EIXO COMO VARIÁVEL

Vamos explorar o estilo orientado a objetos, começando pela criação de eixos de forma explícita, ou seja com atribuição a uma variável. Vamos criar uma figura com 1 linha 2 duas colunas, ou seja, teremos dois eixos. Pense nos eixos como uma matriz, na qual cada eixo é uma posição que pode ter uma figura alocada. Vale ressaltar que sobre um eixo (sobre uma figura), podem ser plotados diversos gráficos. Para criar essa estrutura usamos a sintaxe: `fig, ax = plt.subplots(1, 2)`, onde fig e ax são os nomes das variáveis escolhidas. A variável ax, é do tipo array numpy, ou seja, os eixos nada mais são, que uma matriz de contêiners para se criar os plots. Como a figura possui dois eixos, temos que especificar em qual vamos plotar, para isso informamos qual contêiner vamos usar: ax[0] ou ax[1].

Veja o código a seguir. Na linha 6 criamos a figura, com 1 linha 2 colunas e o eixo que vamos utilizar e ainda definimos o tamanho das figuras por meio do parâmetro figsize. Nas linhas 8, 9 e 10, imprimimos algumas informações para entendermos esse mecanismo do pyplot. ax é do tipo 'numpy.ndarray', como já havíamos mencionado. Ao imprimir o conteúdo de ax[0] e ax[1], podemos ver as coordenadas alocadas para realizar a impressão da figuras. Das linhas 12 a 18, imprimimos 3 gráficos sobre o primeiro eixo (que será posicionado uma figura do

lado esquerdo), definimos os rótulos dos eixos, o título do gráfico e pedimos para exibir a legenda, que é construída a partir do parâmetro "label" na função plot(). Das linhas 20 a 26, criamos novos 3 gráficos, mas agora sobre o segundo eixo (que será posicionado uma figura do lado direito). Nesse novo conjunto de gráficos, configuramos a aparência das linhas, com os parâmetros 'r--' (red tracejado), 'b--' (blue tracejado) e 'g--' (green tracejado). Observe o resultado dos gráficos. Ao criar uma estrutura com 1 linha e 2 colunas, os gráficos ficam posicionados lado a lado, e se tivéssemos criado com 2 linhas e 1 coluna?

In [3]:

```
import numpy as np


x = range(5)
x = np.array(x) # temos que converter para um array numpy, senão o plot não
consegue fazer operações.


fig, ax = plt.subplots(1, 2, figsize=(12, 5)) # Cria figura com subplots: 1
linha, 2 colunas e eixos

print("Tipo de ax = ", type(ax))
print("Conteúdo de ax[0] = ", ax[0])
print("Conteúdo de ax[1] = ", ax[1])

ax[0].plot(x, x, label='eq_1') # cria gráfico sobre eixo 0
ax[0].plot(x, x**2, label='eq_2') # cria gráfico sobre eixo 0
ax[0].plot(x, x**3, label='eq_3') # cria gráfico sobre eixo 0
ax[0].set_xlabel('Eixo x')
ax[0].set_ylabel('Eixo y')
ax[0].set_title("Gráfico 1")
ax[0].legend()

ax[1].plot(x, x, 'r--', label='eq_1') # cria gráfico sobre eixo 1
ax[1].plot(x**2, x, 'b--', label='eq_2') # cria gráfico sobre eixo 1
ax[1].plot(x**3, x, 'g--', label='eq_3') # cria gráfico sobre eixo 1
ax[1].set_xlabel('Novo Eixo x')
ax[1].set_ylabel('Novo Eixo y')
ax[1].set_title("Gráfico 2")
ax[1].legend()
```

```
Tipo de ax =  <class 'numpy.ndarray'>
Conteúdo de ax[0] =  AxesSubplot(0.125,0.125;0.352273x0.755)
Conteúdo de ax[1] =  AxesSubplot(0.547727,0.125;0.352273x0.755)
```

Out[3]:

```
<matplotlib.legend.Legend at 0x1cf32a6cb00>
```

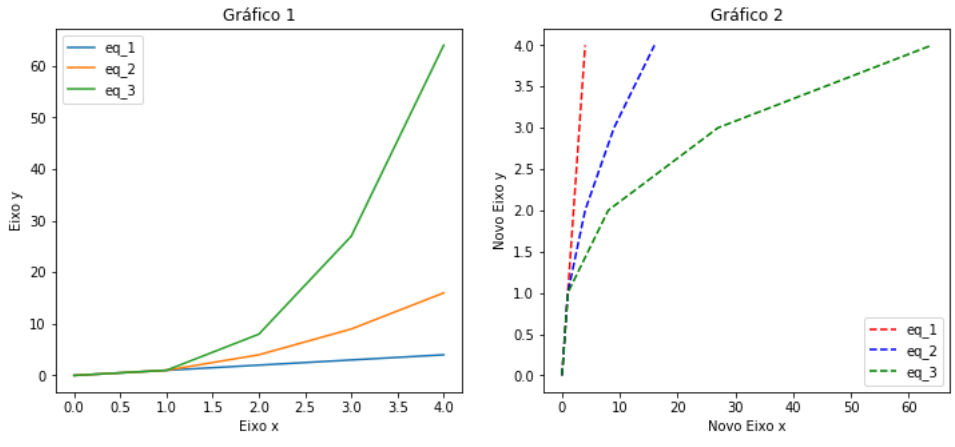


FIGURA SEM EIXO COMO VARIÁVEL

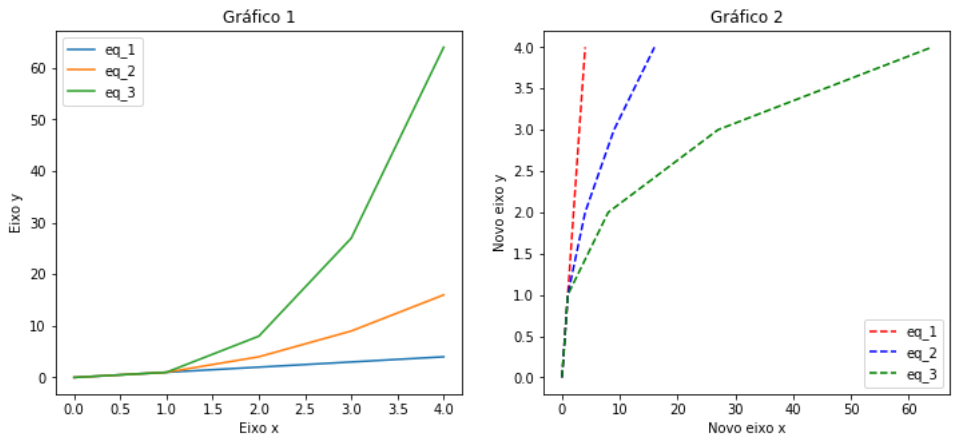
Também podemos criar uma figura, sem atribuir o eixo a uma variável. Nesse caso, temos que usar a função `plt.subplot(n_rows, n_cols2, plot_number)`, para definir onde será plotado o gráfico. Veja no código a seguir. Na linha 4 criamos uma figura, mas agora sem eixo e sem especificar o grid. Na linha 5, estamos adicionando um subplot com 1 linha, 2 colunas e o primeiro gráfico (121). O primeiro parâmetro do método subplot() é a quantidade de linhas; o segundo parâmetro é a quantidade de colunas; o terceiro é número do plot dentro da figura, deve começar em 1 e ir até a quantidade de plots que se tem. Como o eixo não está atribuído a nenhuma variável, agora usamos o próprio para acessar a função plot(). Veja que na linha 14 estamos adicionando um subplot de 1 linha, 2 colunas a mesma figura, mas agora especificando que plotaremos a segunda figura (122).

In [4]:

```
x = range(5)
x = np.array(x) # temos que converter para um array numpy, senão o plot não
consegue fazer operações.

fig = plt.subplots(figsize=(12, 5)) # Cria figura sem eixo
plt.subplot(121) # Adiciona um grid de subplots a figura: 1 linha, 2 colunas -
Figura 1
plt.plot(x, x, label='eq_1')
plt.plot(x, x**2, label='eq_2')
plt.plot(x, x**3, label='eq_3')
plt.title("Gráfico 1")
plt.xlabel('Eixo x')
plt.ylabel('Eixo y')
plt.legend()

plt.subplot(122) # Adiciona um grid de subplots a figura: 1 linha, 2 colunas -
Figura 2
plt.plot(x, x, 'r--', label='eq_1')
plt.plot(x**2, x, 'b--', label='eq_2')
plt.plot(x**3, x, 'g--', label='eq_3')
plt.title("Gráfico 2")
plt.xlabel('Novo eixo x')
plt.ylabel('Novo eixo y')
plt.legend()
```

Out[4]:

```
<matplotlib.legend.Legend at 0x1cf32b44c50>
```



Naturalmente obtivemos o mesmo resultado anterior, pois criamos a mesma estrutura com uma sintaxe diferente. Ao optar em utilizar eixos como variáveis ou não, o desenvolvedor deve ficar atento somente as regras de sintaxe e as funções disponíveis para cada opção. Podemos então resumir que:

- `plt.subplots()` é usado para criar um layout de figura e subplots. (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.html#matplotlib.pyplot.subplots).
- `plt.subplot()` é usado para adiconar um subplot em um figura existente. (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot).

Que tal treinar e explorar utilizando o simulador a seguir. Veja os exemplos que a própria ferramenta utiliza em sua página principal.

BIBLIOTECA PANDAS

As principais estruturas de dados da biblioteca pandas (Series e DataFrame) possuem o método `plot()`, construído com base no matplotlib e que permite criar gráficos a partir dos dados nas estruturas. Vamos começar criando um DataFrame a partir de um dicionário, com a quantidade de alunos em três turmas distintas.

In [5]:

```
import pandas as pd

dados = {
    'turma':['A', 'B', 'C'],
    'qtde_alunos':[33, 50, 45]
    }
df = pd.DataFrame(dados)

df
```

Out[5]:

	turma	qtde_alunos
0	A	33
1	B	50
2	C	45

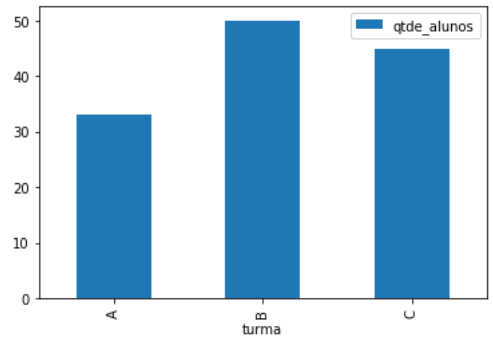
A partir de um DataFrame, podemos invocar o método: `df.plot(*args, **kwargs)` para criar os gráficos. Os argumentos dessa função, podem variar, mas existem três que são triviais: os nomes das colunas com os dados para os eixos x e y, bem como o tipo de gráfico (kind). Veja o código a seguir, os valores da coluna 'turma' serão usados no eixo x, da coluna 'qtde_alunos' no eixo y e o tipo de gráfico será o de barras (bar). Nas entradas 7 e 8, repetimos a mesma construção, entretanto mudando o tipo de gráfico para barra na horizontal (barh) e linha (line).

In [6]:

```
df.plot(x='turma', y='qtde_alunos', kind='bar')
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf339fe7b8>
```

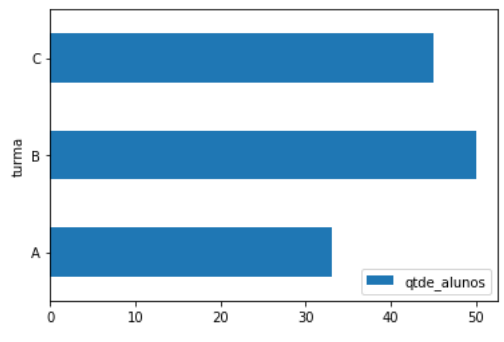


In [7]:

```
df.plot(x='turma', y='qtde_alunos', kind='barh')
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf32bd2470>
```

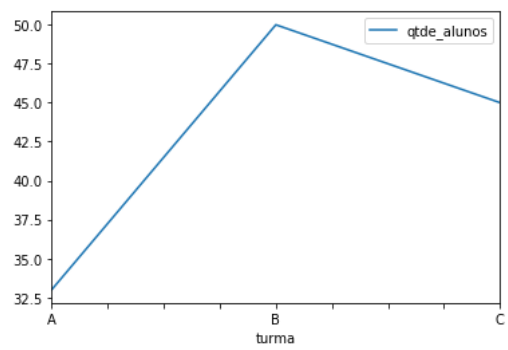


In [8]:

```
df.plot(x='turma', y='qtde_alunos', kind='line')
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf33aeb668>
```



No endereço https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html você pode encontrar a lista com todos os tipos de gráficos possíveis de serem construídos com o método plot() da biblioteca. Para construir um gráfico do tipo pizza (pie), precisamos definir como indíce os dados que serão usados como legenda. Veja a seguir, fazemos a transformação no DF seguido do plot com o tipo pie. Esse tipo de sintaxe é

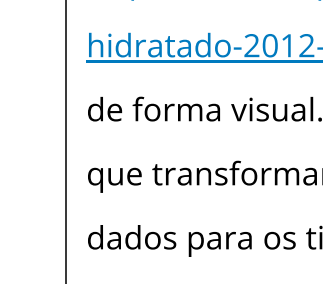
chamado de encadeamento, pois ao invés de fazermos a transformação, salvar em um novo objeto e depois plotar, fazemos tudo em uma única linha, sem precisar criar o novo objeto.

In [9]:

```
df.set_index('turma').plot(y='qtde_alunos', kind='pie')
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf33b7fda0>
```



Vale ressaltar que para todos os gráficos criados, a biblioteca oferece uma segunda opção de sintaxe, que é invocar o tipo de gráfico como método, por exemplo:

- `df.plot.bar(x='turma', y='qtde_alunos')`
- `df.plot.line(x='turma', y='qtde_alunos')`
- `df.set_index('turma').plot.pie(y='qtde_alunos')`

> **EXEMPLIFICANDO**
>
> Vamos utilizar os dados sobre a exportação de etanol hidratado (barris equivalentes de petróleo) 2012-2020, disponível no endereço: https://www.anp.gov.br/arquivos/dadosabertos/iee/exportacao-etanol-hidratado-2012-2020-bep.csv, para analisarmos e extrairmos informações de forma visual. Para conseguir utilizar o método plot nessa base, teremos que transformar a vírgula em ponto (padrão numérico) e converter os dados para os tipos float e int. Veja o código a seguir em que preparamos os dados
>
> In [10]:

```
df_etanol = pd.read_csv('exportacao-etanol-hidratado-2012-2020-bep.csv',
sep=';', encoding="ISO-8859-1")

# Apaga colunas que não usaremos
df_etanol.drop(columns=['PRODUTO', 'MOVIMENTO COMERCIAL', 'UNIDADE'],
inplace=True)

# Substitui a vírgula por ponto em cada coluna
for mes in 'JAN FEV MAR ABR MAI JUN JUL AGO SET OUT NOV DEZ
TOTAL'.split():
    df_etanol[mes] = df_etanol[mes].str.replace(',', '.')

# Converte os valores para float
df_etanol = df_etanol.astype(float)

# Converte o ano para inteiro
df_etanol['ANO'] = df_etanol['ANO'].astype(int)

df_etanol.head(2)
```

Out[10]:

	ANO	JAN	FEV	MAR	ABR	MAI	JUN	JUL	
0	2012	87231.41132	141513.5186	122157.33850	98004.42926	153286.6078	144373.6894	384743.6142	24
1	2013	673419.97670	387331.6487	96929.59201	54390.05046	115092.4820	387498.3792	339162.2100	35

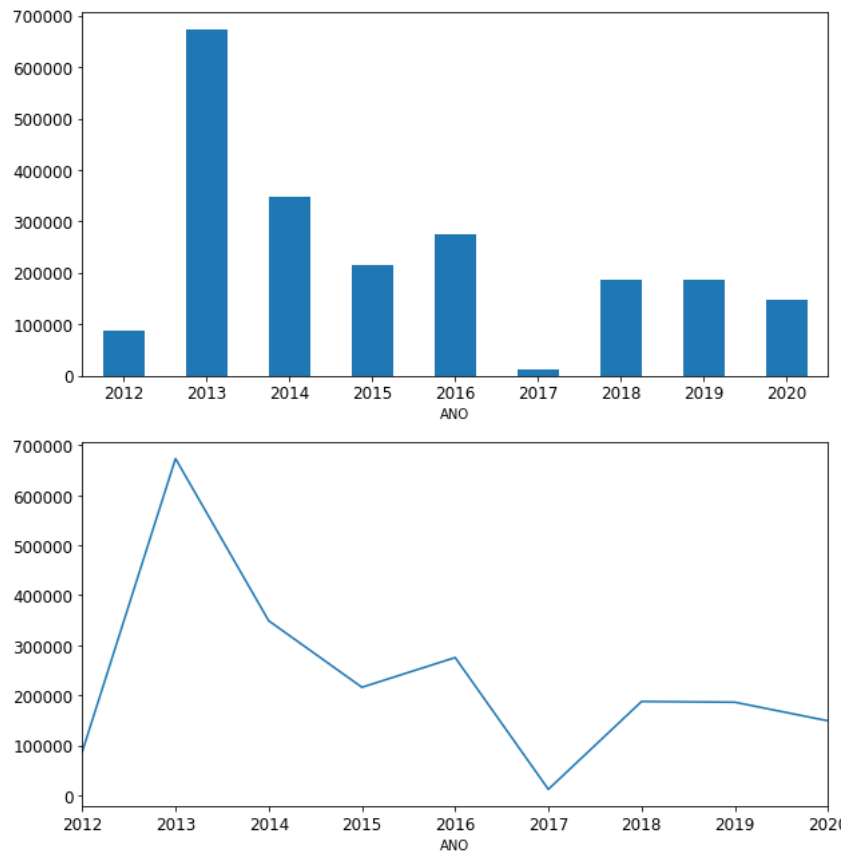
Agora com os dados corretos podemos usar o método de plotagem para que, de forma visual, possamos identificar o ano que teve a menor e maior arrecadação para o mês de janeiro. Veja o código a seguir. No eixo x, vamos usar a informação de ano, e no y todos os valores da coluna 'JAN'; (kind) o tipo do gráfico será de barras; (figsize) a figura terá um tamanho de 10 polegadas na horizontal por 5 na vertical; (rot) a legenda no eixo x não deve ser rotacionada; (fontsize) o tamanho das fontes na figura deve ser 12 pt; (legend) não queremos legenda nesse gráfico. Pelo gráfico, podemos identificar com facilidade que o ano que teve menor arrecadação nesse mês, foi 2017 e o maior 2013. Também plotamos o mesmo gráfico, mas em linhas, veja como é possível explicar os dados agora sobre outra perspectiva: o comportamento da variação da arrecadação ao longo do tempo.

In [11]:

```
df_etanol.plot(x='ANO',
               y='JAN',
               kind='bar',
               figsize=(10, 5),
               rot=0,
               fontsize=12,
               legend=False)

df_etanol.plot(x='ANO',
               y='JAN',
               kind='line',
               figsize=(10, 5),
               rot=0,
               fontsize=12,
               legend=False)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf34c166a0>
```



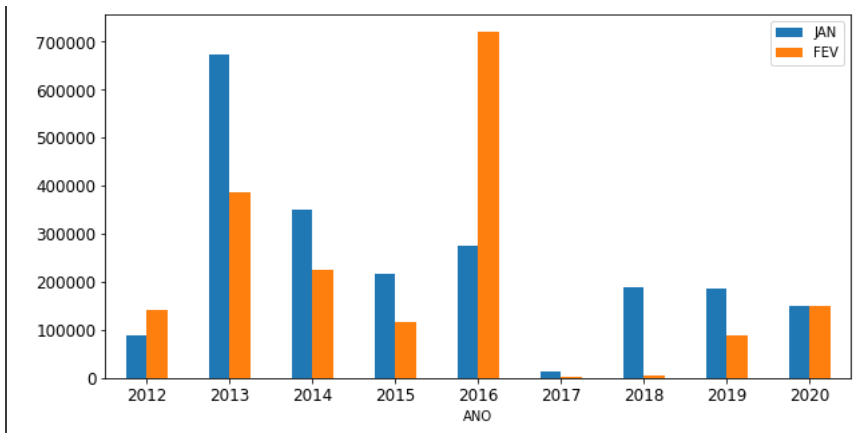
Queremos criar um gráfico, que nos possibilite comparar a arrecadação entre os meses de janeiro e fevereiro. Veja no código a seguir. Estamos selecionando três colunas do nosso DF e encadeando o método plot(), agora passando como parâmetro somente o valor x, o tipo, o tamanho, a rotação da legenda e o tamanho da fonte. Os valores para o eixo y, serão usados das colunas. Com essa construção se torna possível avaliar, visualmente, o desempenho nos meses. Veja que em determinados anos, a discrepância entre eles é consideravel.

In [12]:

```
df_etanol[['ANO', 'JAN', 'FEV']].plot(x='ANO', kind='bar', figsize=(10,
5), rot=0, fontsize=12)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf34c708d0>
```



BIBLIOTECA SEABORN

Seaborn é outra biblioteca Python, também baseada na matplotlib, que foi desenvolvida especificamente para criação de gráficos. Seaborn pode ser instalado via pip install: `pip install seaborn`, e para utilizar no projeto existe uma convenção para sintaxe: `import seaborn as sns`. A biblioteca conta com um repositório de datasets que podem ser usados para explorar as funcionalidades e estão disponíveis no endereço: https://github.com/mwaskom/seaborn-data. Vamos carregar os dados sobre gorjetas (tips) para nosso estudo. Veja no código a seguir, utilizamos a função load_dataset(), cujo retorno é um DataFrame pandas, para carregar a base de dados. A seguir imprimimos as informações básicas que foram carregadas. Temos 244 linhas e 7 colunas, cujos dados são do tipo ponto flutuante, categóricos e um inteiro.

In [13]:

```
import seaborn as sns


# Configurando o visual do gráfico. Leia mais em
https://seaborn.pydata.org/generated/seaborn.set.html#seaborn.set
sns.set(style="whitegrid") # opções: darkgrid, whitegrid, dark, white, ticks


df_tips = sns.load_dataset('tips')


print(df_tips.info())
df_tips.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           244 non-null float64
sex           244 non-null category
smoker        244 non-null category
day           244 non-null category
time          244 non-null category
size          244 non-null int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.2 KB
None
```

Out[13]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

O tipo de dados que uma coluna possui é muito importante para a biblioteca seborn, uma vez que as funções usadas para construir os gráficos são dividas em grupos: relacional, categóricos, distribuição, regressão, matriz e grids (https://seaborn.pydata.org/api.html).

FUNÇÃO BARPLOT()

Dentro do grupo de funções para gráficos de variáveis categóricas, temos o **barplot()**, que permite criar gráficos de barras, mas por que usaríamos essa função e não a da biblioteca pandas? A resposta está nas opções de parâmetros que cada biblioteca suporta. Veja o construtor da função barplot:

`seaborn.barplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, estimator=function mean, ci=95, n_boot=1000, units=None, seed=None, orient=None, color=None, palette=None, saturation=0.75, errcolor='.26', errwidth=None, capsize=None, dodge=True, ax=None, **kwargs)`. Esse construtor possui uma série de parâmetros estatísticos, que dão muita flexibilidade e poder aos cientista de dados, vamos falar sobre o parâmetro "estimator", que por default é a função média. Isso significa que cada barra do gráfico, exibirá a média dos valores de uma determinada coluna, o que pode não fazer sentido, uma vez que queremos exibir a quantidade dos valores (len) ou a soma (sum).

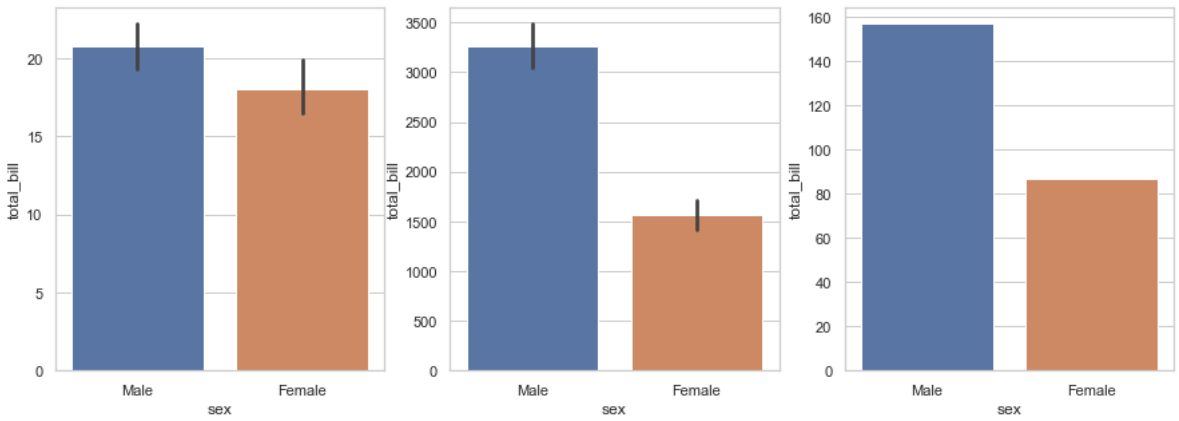
Para entender como esse parâmetro pode "afetar" a construção do gráfico, veja o código a seguir. Usamos o matplotlib para construir uma figura e um eixo com três posições. Nosso primeiro gráfico de barras (linha 3), utiliza o estimator padrão, que sabemos que é a média. O segundo (linha 4), utiliza a função de soma como estimator e o terceiro (linha 5), a função len, para contar. Veja no resultado como os gráficos são diferentes. No primeiro, nos conta que a o valor médio da conta entre homens e mulheres é próximo, embora os homens gastem um pouco mais. Já o segundo gráfico, nos mostra que os homens gastam "muito mais", será que é verdade? A soma da conta dos homens, de fato é superior a das mulheres, mas será que não existem muito mais homens do que mulheres na base? O terceiro gráfico nos conta isso, quantos homens e mulheres possuem na base.

In [14]:

```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))


sns.barplot(data=df_tips, x='sex', y='total_bill', ax=ax[0])
sns.barplot(data=df_tips, x='sex', y='total_bill', ax=ax[1], estimator=sum)
sns.barplot(data=df_tips, x='sex', y='total_bill', ax=ax[2], estimator=len)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf35b313c8>
```

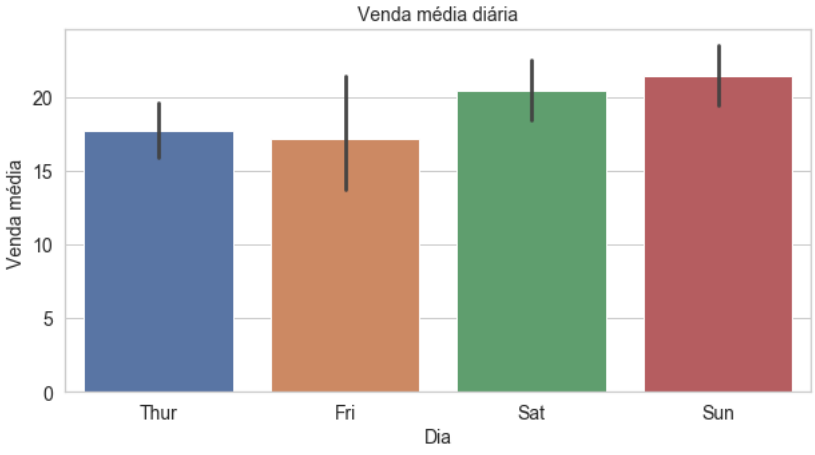


Construir gráficos não é somente plotar imagens bonitas, existem muitos conceitos estatísticos envolvidos e a biblioteca seaborn fornece mecanismos para que essas informações estejam presentes nos resultados visuais.

A biblioteca seaborn integra totalmente com a matplotlib. Vamos criar um gráfico que nos mostre o valor médio diário de venda. Veja o código a seguir, a criação do gráfico está na linha 3, mas configuramos seu tamanho na linha 1, e das linhas 5 a 8, configuramos os rótulos e os tamanhos.

In [15]:

```
plt.figure(figsize=(10, 5))

ax = sns.barplot(x="day", y="total_bill", data=df_tips)

ax.axes.set_title("Venda média diária", fontsize=14)
ax.set_xlabel("Dia", fontsize=14)
ax.set_ylabel("Venda média ", fontsize=14)
ax.tick_params(labelsize=14)
```



FUNÇÃO COUNTPLOT()

Conseguimos plotar a contagem de uma variável categórica, com a função barplot e o estimator len, entretanto, a biblioteca seaborn possui uma função específica para esse tipo de gráfico: `seaborn.countplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, dodge=True, ax=None, **kwargs)`.

Esse método não aceita que sejam passados valores de x e y ao mesmo tempo, pois a contagem será feita sobre uma variável categórica, portanto devemos especificar x ou y, a diferença será na orientação do gráfico. Se informamos x, teremos uma gráfico na vertical, se y, na horizontal.

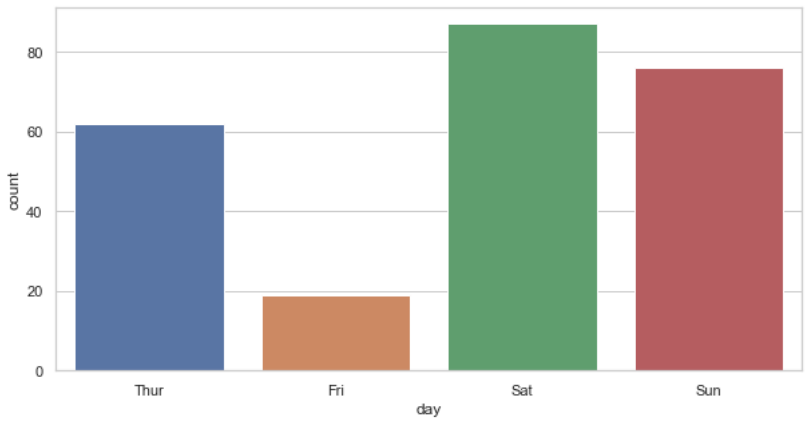
In [16]:

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df_tips, x="day")
```

Ver anotações 0

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf35ad2e80>
```



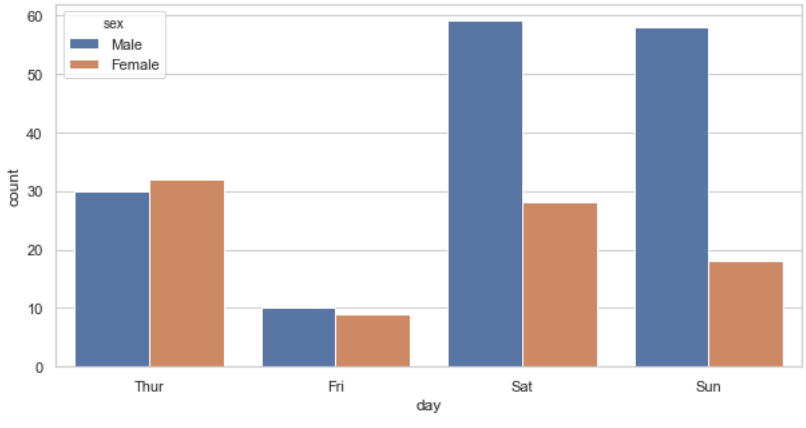
Vamos plotar mais um gráfico de contagem para mostrar o poder de um único parâmetro. O parâmetro **hue** é usado como entrada de dados, pois irá discriminar no gráfico a variável atribuída ao parâmetro. Para entendermos, vamos plotar a quantidade de pessoas por dia, mas discriminado por genêro, quantos homens e mulheres estiveram presentes em cada dia? Veja no código a seguir, a única diferença é o parâmetro.

In [17]:

```
plt.figure(figsize=(10, 5))

sns.countplot(data=df_tips, x="day", hue="sex")
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf35ba9240>
```



FUNÇÃO SCARTTERPLOT()

Os gráficos do grupo relacional, permitem avaliar, de forma visual a relação entre duas variáveis: x, y. A função possui a seguinte sintaxe: `seaborn.scatterplot(x=None, y=None, hue=None, style=None, size=None, data=None, palette=None, hue_order=None, hue_norm=None, sizes=None, size_order=None, size_norm=None, markers=True, style_order=None, x_bins=None, y_bins=None, units=None, estimator=None, ci=95, n_boot=1000, alpha='auto', x_jitter=None, y_jitter=None, legend='brief', ax=None, **kwargs).`

Vamos construir um gráfico que permita avaliar se existe uma relação entre o valor da conta e da gorjeta. Será que quem gastou mais também deu mais gorjeta? Veja o código a seguir, invocamos a função passando o valor da conta como parâmetro para x e a gorjeta para y. Agora vamos avaliar o resultado. Cada "bolinha" representa uma conta paga e uma gorjeta, por exemplo, a bolinha mais a direita, podemos interpretar que para uma conta de aproximadamente 50 e poucos doláres foi dada uma gorjeta de 10. Olhando para o gráfico, parece quanto maior o

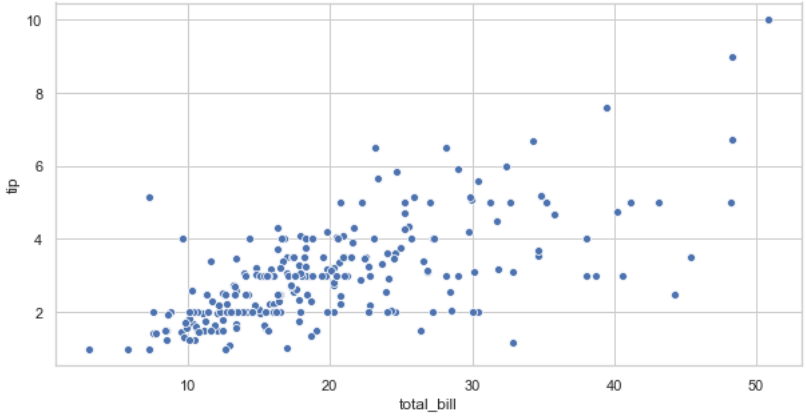
valor da conta, maior foi o valor da gorjeta. Esse comportamento é chamado de relação linear, pois conseguimos traçar uma reta entre os pontos, descrevendo seu comportamento através de uma função linear.

In [18]:

```
plt.figure(figsize=(10, 5))
sns.scatterplot(data=df_tips, x="total_bill", y="tip")
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf35c7e908>
```



O gráfico scatterplot é muito utilizado por cientistas de dados que estão buscando por padrões nos dados. O padrão observado no gráfico, que mostra a relação entre o valor da conta e da gorjeta, pode ser um forte indício que, caso o cientista precise escolher um algoritmo de aprendizado de máquina para prever a quantidade de gorjeta que um cliente dará, ele poderá uma regressão linear.

O universo dos dados é cada vez mais requisitado nas empresas, se você gostou do que aprendemos, não deixe de investir mais tempo em estudo e treinamento, pois há muito o que se aprender!

REFERÊNCIAS E LINKS ÚTEIS

IPython Development Team. Built-in magic commands. Disponível em: https://ipython.readthedocs.io/en/stable/interactive/magics.html. Acesso em: 17 jun. 2020.

Matplotlib development team. matplotlib.pyplot. Disponível em: (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html) Acesso em: 27 jun. 2020.

Michael Waskom. seaborn. Disponível em: (https://seaborn.pydata.org/api.html) Acesso em: 27 jun. 2020.

Pandas Team. pandas.DataFrame.plot. Disponível em: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html. Acesso em: 27 jun. 2020.

Secretaria de Tecnologia da Informação, Ministério do Planejamento, Desenvolvimento e Gestão. Conjuntos de dados. Disponível em: https://www.dados.gov.br/dataset. Acesso em: 27 jun. 2020.