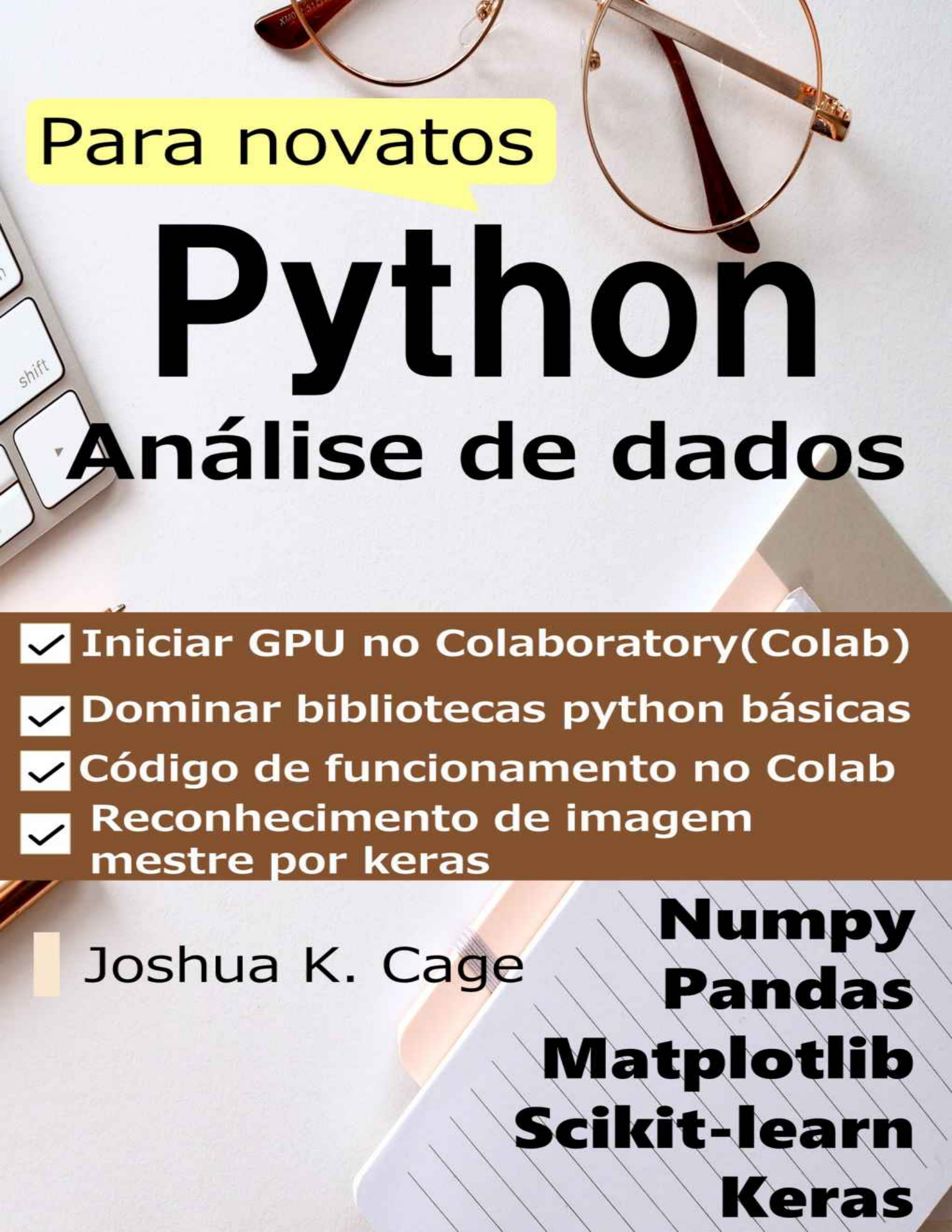

Para novatos
Python
Análise de dados
Iniciar GPU no Colaboratory(Colab)
Dominar bibliotecas python básicas
Código de funcionamento no Colab
Reconhecimento de imagem mestre por keras
Joshua K. Cage
Numpy
Pandas
Matplotlib
Scikit-learn
Keras

Análise de dados Python para novatos

Joshua K. Cage

1. Introdução

Thank você por ter pegado este livro. Este livro é uma introdução para iniciantes na análise de dados usando a programação Python. Este livro é escrito para os seguintes leitores.

1) Interessado na aprendizagem de máquinas e aprendizagem profunda
2) Interessado em programar com Python.
3) Interessados na análise dos dados.
4) Interessado em usar Numpy/Pandas/Matplotlib/ScikitLearn.
5) Não está interessado em construir ambientes de aprendizagem de máquinas.
6) Não estou interessado em gastar muito dinheiro para aprender.
7) Vagamente preocupada com a nova epidemia da coroa e o futuro.

Muitos dos meus amigos e conhecidos começaram a análise de dados com uma vingança, apenas para ficarem satisfeitos com o processo de criação de um ambiente ao longo do dia, e então, depois de fazer MNIST (conjuntos de dados de imagens numéricas escritas à mão) e tutoriais de classificação da íris, eles se ocupam com seus trabalhos diários e abandonam por um tempo.

Este livro utiliza o ambiente de execução Python gratuito fornecido pelo Google para executar o código fonte testado no livro, permitindo que você aprenda programando com tempo zero para configurar o seu próprio ambiente.

This livro concentra-se no mínimo de conhecimento necessário para que um principiante se possa dedicar à análise séria de dados em Python. Nosso objetivo é que até o final do livro, os leitores terão alcançado os cinco objetivos seguintes.

1) Construir e treinar modelos de aprendizagem profunda e modelos de aprendizagem de máquinas a partir de dados arbitrários a serem

treinados e previstos usando bibliotecas de aprendizagem profunda (keras) e bibliotecas de aprendizagem de máquinas (scikit-learn).

2) Usar Pandas em vez de Excel para processamento de dados em larga escala.

3) Para manipular arrays multidimensionais usando Numpy.

4) Para desenhar gráficos livremente usando Matplotlib.

5) Realizar simples análise de dados sobre a propagação de novos coronavírus.

Com o novo coronavírus espalhando-se pelo mundo e os vários relatórios nestes tempos de incerteza sobre o destino, muitos de vocês podem não saber no que acreditar e como lidar com a situação.

One é certo, haverá uma diferença notável nos conjuntos de habilidades dos indivíduos, dependendo de como eles fazem uso do novo tempo livre criado pelo teletrabalho, e estamos entrando numa era de claros vencedores e perdedores que fará uma enorme diferença em seu valor na empresa e no mercado de trabalho.

I acredita que é vital que não continuemos agarrados a medos vagos num estado de ansiedade, mas sim que transformemos cada ansiedade num problema resolúvel através da análise de dados, um a um, para que cada pessoa possa escolher uma linha de acção.

[Importante] Anúncio

A série de palestras do autor Udemy "Pytorch Deep Learning From Zero to Hero" foi lançada em Janeiro de 2021. Neste curso prático de mais de 5 horas de palestra utilizando o Google Colab, poderá primeiro compreender como as redes neuronais aprendem e como implementá-lo. Este curso abrange também a FCNN, CNN, e RNNN. Em última análise, poderá implementar a aprendizagem de transferência BERT usando a biblioteca de transformadores huggingface.

Estamos a dar cupões a 10 leitores deste livro para fazerem o curso gratuitamente durante um período de tempo limitado. Este é o primeiro a chegar, primeiro a ser servido, por isso é favor inscrever-se o mais cedo possível, e por favor rever após a aula.

https://www.udemy.com/course/pytorch-deep-learning-hero/?couponCode=CC776B516CA6BD4F1771

* A palestra é explicada em inglês

2. Isenção de responsabilidade

As informações contidas neste documento são apenas para fins informativos. Portanto, o uso deste livro é sempre por conta e risco do próprio leitor. A utilização do Colaborador Google descrita neste livro é feita por conta e risco do próprio leitor após a revisão dos Termos de Serviço e da Política de Privacidade do Google.

Em nenhum caso o autor será responsável por quaisquer lucros consequentes, acidentais ou perdidos ou outros danos indirectos, previstos ou previsíveis, decorrentes ou relacionados com a utilização do código fonte que acompanha este livro ou o serviço do Colaborador Google.

You deve aceitar as precauções acima antes de usar este livro. O autor não poderá responder às perguntas sem estas precauções. Tenha em conta que o autor não poderá responder às suas perguntas se não ler estas notas.

3. Marcas comerciais e marcas registadas

All nomes de produtos que aparecem neste manual são geralmente marcas registradas ou marcas comerciais das respectivas empresas. ™, ® e outras marcas podem ser omitidas do texto.

4. Feedback

While o maior cuidado foi tomado na escrita deste livro, você pode notar erros, imprecisões, linguagem enganosa ou confusa, ou simples erros e erros tipográficos. Nesses casos, agradecemos o seu feedback para o seguinte endereço, para que possamos melhorar as edições futuras. Sugestões para futuras revisões também são bem-vindas. As informações para contato estão abaixo.

Joshua K. Cage

joshua.k.cage@gmail.com

5. Jupyter Caderno

The Jupyter Notebook, que lhe permite executar o código descrito neste livro, está agora disponível no Google Colaboratory. Pode aceder-lhe a partir do seguinte link, por isso consulte-o quando ler este livro (recomenda-se o Chrome*).

https://drive.google.com/file/d/1G7_YFCGMqV2bfTmR82pSwqLkSxMfhTDh/view?usp=sharing

6. Ambiente GPU -Google Colaboratory

Years atrás, programar um programa Python para análise de dados requeria a criação de um ambiente UNIX e a compilação de bibliotecas individuais, o que consomeva muito tempo. Atualmente, no entanto, a Continuum Analytics oferece o Anaconda, um ambiente virtual Python para computação científica que pode ser facilmente instalado usando um instalador e um conjunto de bibliotecas. Se você quiser configurar um ambiente Python em um PC local, como um Windows ou Mac, você pode facilmente criar um ambiente Python autônomo usando o Anaconda.

However, a Anaconda tem os seus problemas. Você precisa de ter o seu próprio "PC local". Poucos usuários novatos têm um PC com especificações suficientes para rodar o aprendizado da máquina no mundo real ou simulações de aprendizado profundo em casa. Isso desperdiça muito tempo. Este livro recomenda o uso de GPUs no ambiente do Google Colaboratory (Colab).

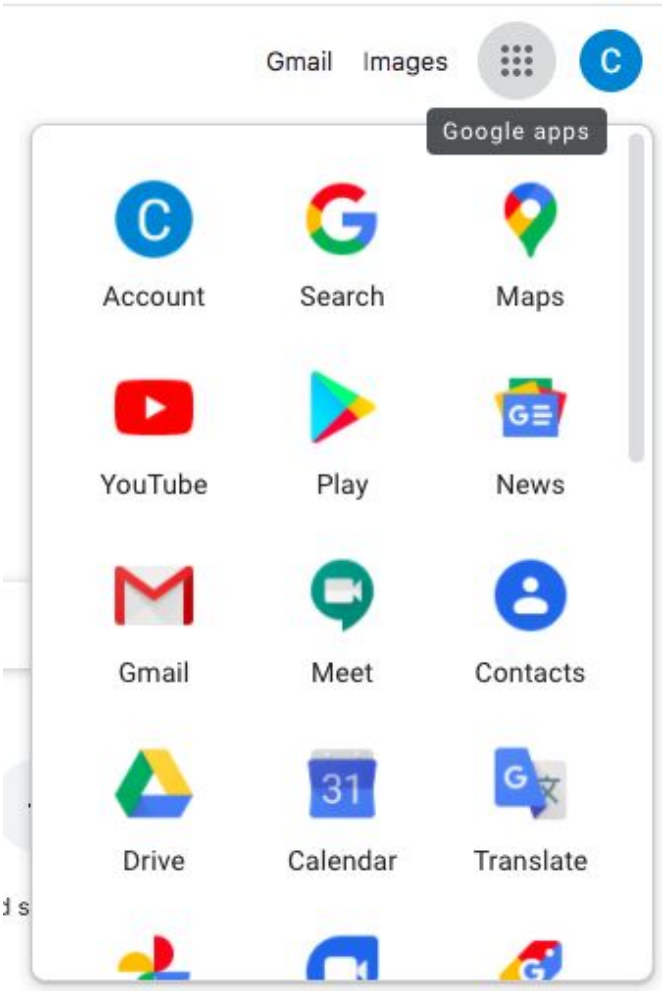
Colab usa uma ferramenta chamada Jupyter Notebook, que também está incluída no Anaconda, para executar Python a partir de um navegador web na nuvem (e é grátis!). O Colab é padrão com Pandas/Numpy/Matplotlib/Keras, que é usado neste livro. Este é um ótimo serviço que lhe permite trabalhar nos seus projetos de aprendizagem da máquina a qualquer hora, em qualquer lugar, desde que você tenha uma conexão de internet, mesmo em um PC ou tablet sem energia. Com risco zero para começar e custo zero para configurar, agora que você pegou uma cópia deste livro você pode rodar seus programas Python no Colab e você vai se surpreender com a facilidade de escrever e a rapidez com que você pode rodar programas Python de aprendizado profundo em GPUs.

If você não tem uma conta GMAIL, você precisará criar uma clicando no link aqui. A explicação seguinte continua assumindo que

você já tem uma conta no GMAIL.

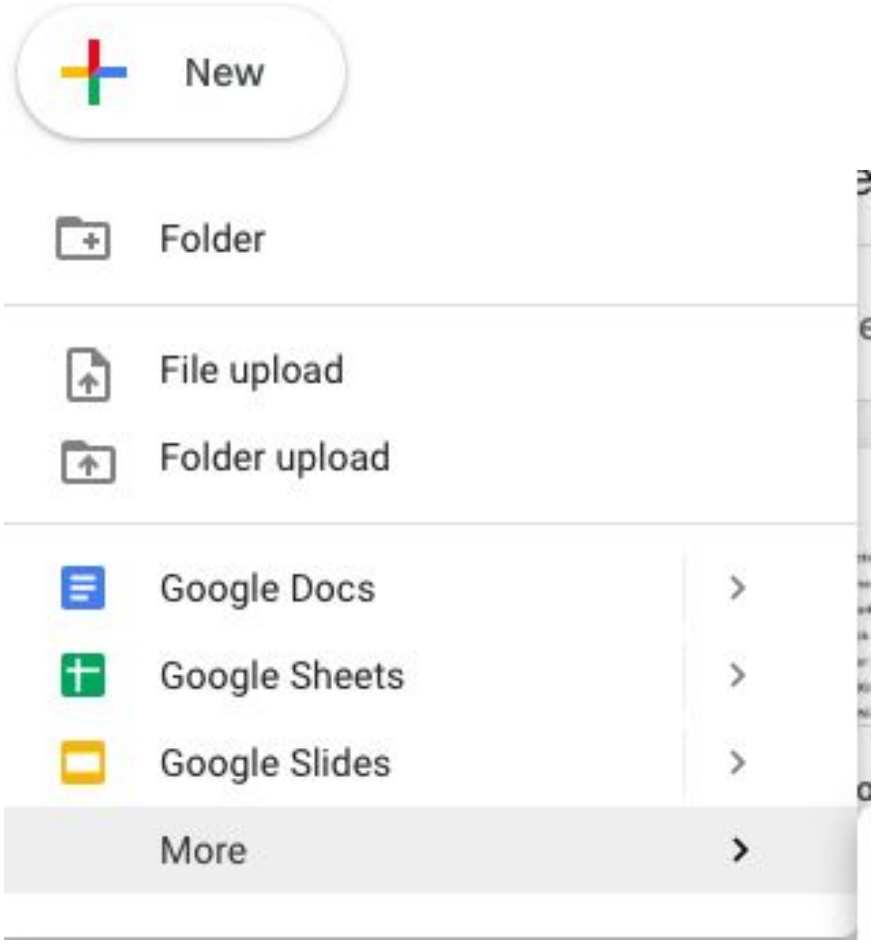
<u>Como configurar o Colab</u>

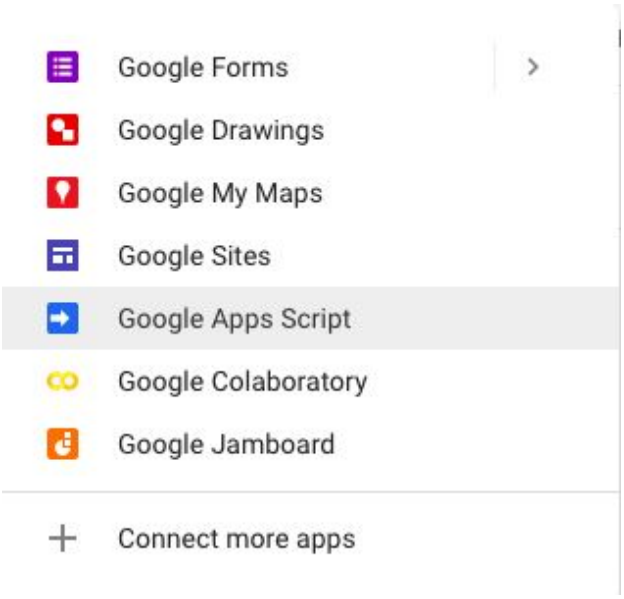
(1) Ao acessar o GMAIL, no canto superior direito da tela você verá um Menu Bento com nove quadrados, clique sobre isso e depois clique no ícone "Drive".



(2) Pressione o botão "+ Novo" na parte inferior da unidade e selecione "Mais >" no menu, depois clique em "Google

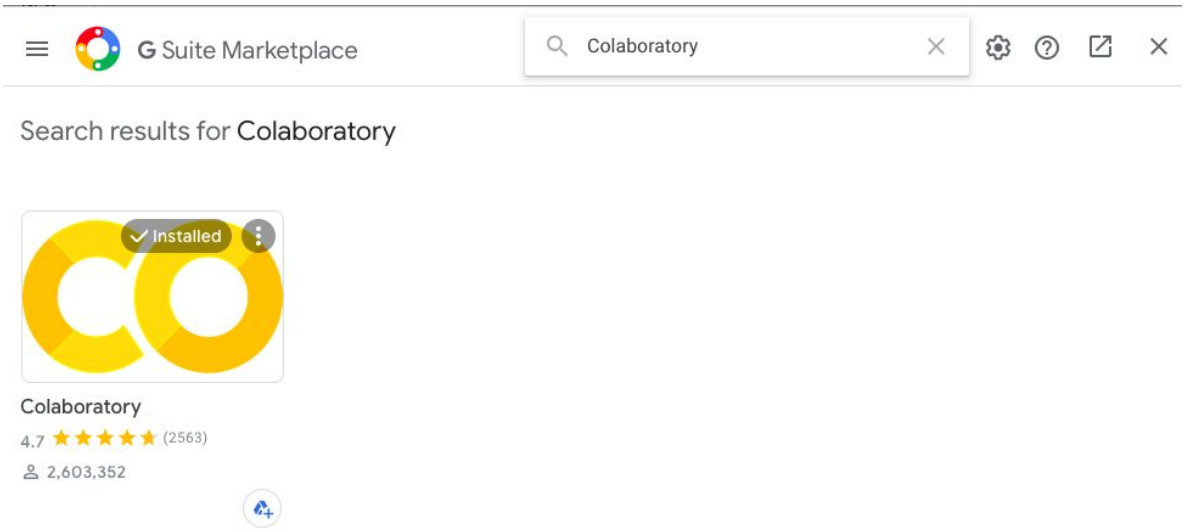
Colaboratory" se existir, caso contrário escolha "Conectar mais aplicativos".

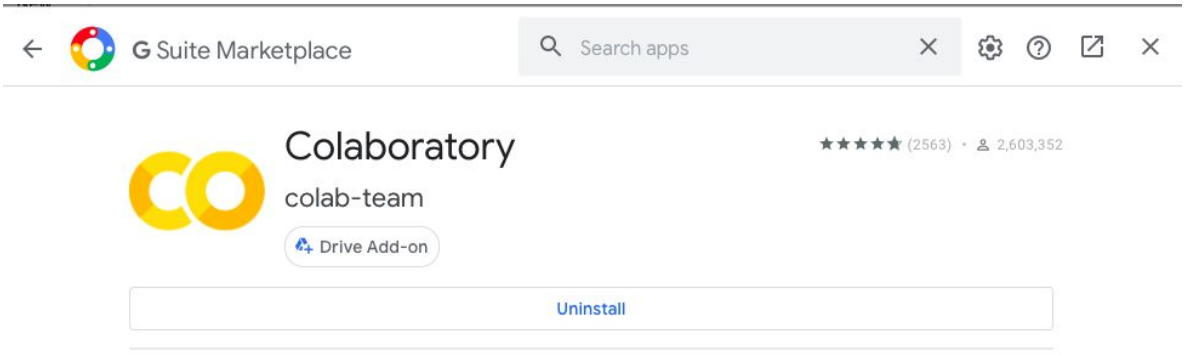




(3) Quando "G Suite Marketplace" for exibido, clique na marca da lupa, e na caixa de texto para pesquisar no aplicativo, digite "Colaborador". Por favor clique no botão "+" na parte inferior direita do logotipo, e depois clique no botão "Instalar" na tela que aparece.

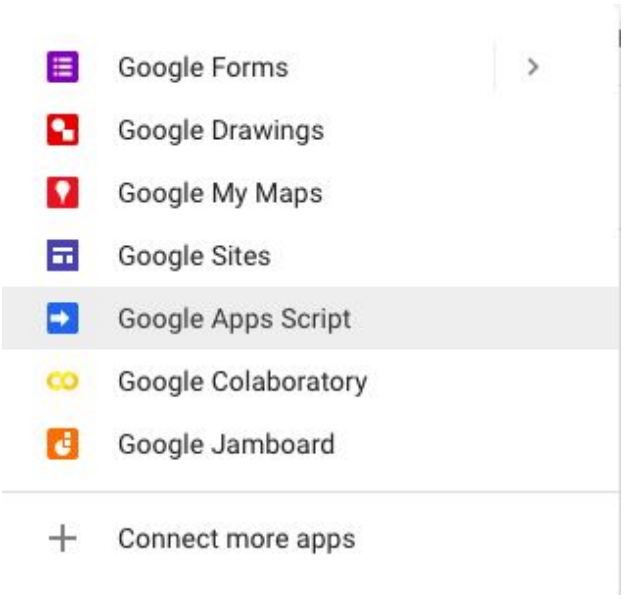
Por favor, clique no botão "Instalar" na tela exibida na parte inferior direita do logotipo.





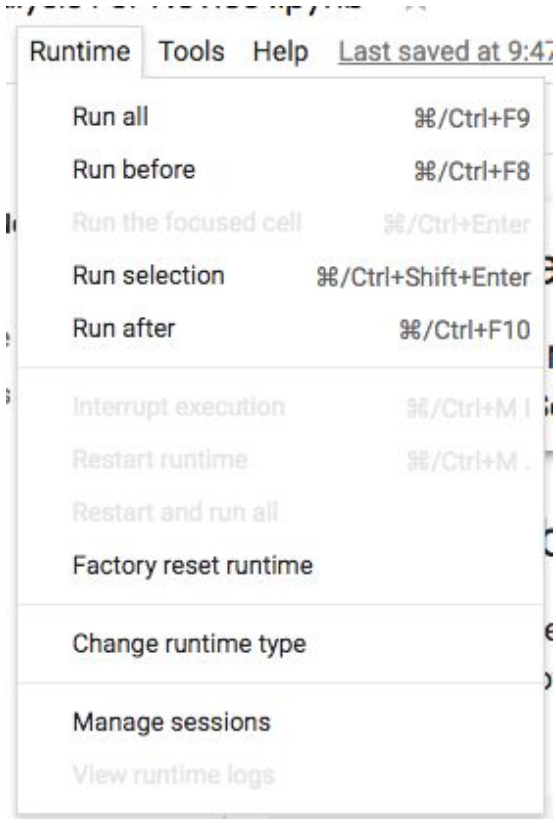
(4) Você pode ser solicitado a fazer o login novamente, por favor continue. Quando a tela do "Google Colaboratory está agora conectada ao Google Drive. Quando a tela do "Google Colaboratory foi conectada ao Google Drive" aparecer, marque a caixa de "Fazer do Google Colaboratory a aplicação padrão" e clique no botão "OK". Uma janela modal que diz "O Google Colaboratory foi instalado. Quando você vir a janela modal "Você instalou o Colaboratory", você pode usar o Colab. Agora, quando você fizer upload de um arquivo com a extensão Colab (arquivo .ipynb) para o Google Drive, ele deverá abrir no Colab por padrão.

(5) Feche a janela modal e mais uma vez, clique no botão "Novo +" e selecione o aplicativo "Outro >". Agora você pode selecionar "Google Colaboratory".



(6) Quando você seleciona o Google Colaboratory, a seguinte tela se abrirá, mas, por padrão, o Colab está em modo de uso de CPU, o que significa que levará mais tempo para se ter uma aprendizagem profunda. Então, vá para o menu "Tempo de execução", clique em "Alterar tipo de tempo de execução" e selecione "GPU" na seção "Acelerador de hardware" e clique no botão Salvar.

Também é possível usar TPU aqui, mas é um pouco difícil tirar a performance dele, e para a maioria das aplicações não há muita diferença na velocidade de execução entre GPU e GPU, então vamos usar "GPU" neste manual.



Notebook settings

Hardware accelerator

GPU

To get the most out of Colab, avoid using a GPU unless you need one. Learn more

Omit code cell output when saving this notebook

CANCEL SAVE

(7) Para ter certeza de que a GPU está disponível, copie o seguinte código para uma célula e execute-a. Você pode executá-lo pressionando o botão play no lado esquerdo da célula, ou você pode usar o atalho "Shift + Enter". Se você vir "device_type": "GPU" no resultado da execução, isso significa que a GPU é reconhecida.

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

Output:

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 8604083664829407890, name: "/device:XLA_CPU:0"
 device_type: "XLA_CPU"
 memory_limit: 17179869184
 locality {
 }
 incarnation: 18180926124650645506
 physical_device_desc: "device: XLA_CPU device", name:
"/device:XLA_GPU:0"
 device_type: "XLA_GPU"
 memory_limit: 17179869184
 locality {
 }
 incarnation: 18355618728471253196
 physical_device_desc: "device: XLA_GPU device", name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 11146783616
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 18112086373768308297
 physical_device_desc: "device: 0, name: Tesla K80, pci bus id:
0000:00:04.0, compute capability: 3.7"]
```

7. Aprendizagem mínima MNIST mais profunda

So vamos em frente e usar a GPU para realizar um aprendizado profundo para o reconhecimento de caracteres de imagens numéricas para ter uma noção da velocidade da GPU. Insira o seguinte código para treinar um classificador de 60.000 números de 0 a 9, e construa um modelo que olhe para a imagem e preveja quais números 0-9 estão escritos nela.

i. primeiro, execute o código abaixo para importar a biblioteca de aprendizagem profunda (tensorflow)

```
# Import the required library
from __future__ import absolute_import, division, print_function,
unicode_literals

# Import TensorFlow
import tensorflow as tf
```

ii. Carregar os dados da imagem numérica MNIST.

```
# Load MNIST data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

iii. Vamos verificar que tipo de dados temos, usando numpy e matplotlib.

Vamos verificar o tipo de dados do MNIST.

```
print(type(x_train))
```

Output:

```
<class 'numpy.ndarray'>
```

The O conjunto de dados MNIST é do tipo numpy ndarray, que tem o atributo "forma", assim, imprimindo a forma, podemos ver quantos registros desta estrutura de dados são armazenados.

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Output:

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

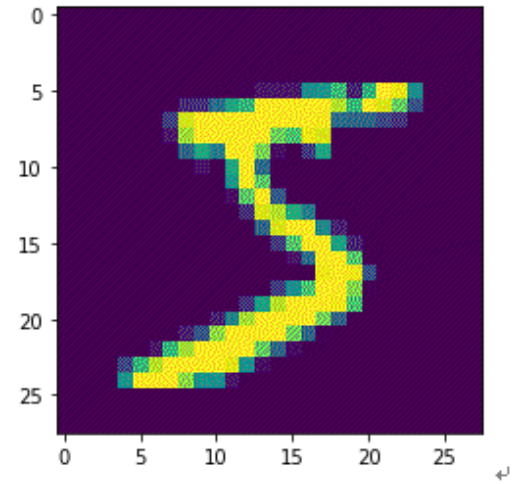
Como você pode ver acima, x_train.shape e x_test.shape contêm 60.000 conjuntos de dados de treinamento de 28 x 28 pixels e 10.000 conjuntos de dados de teste de 28 x 28 pixels, respectivamente. Também podemos ver a partir de y_train.shape e y_test.shape que existem 60.000 etiquetas de resposta correcta para os dados de treino e 10.000 para os testes, respectivamente.

Você também pode usar a biblioteca de desenhos matplotlib para desenhar a partir de um numpy.ndarray (objeto array

multidimensional). Como um teste, vamos mostrar uma imagem de 28 x 28 pixel de um conjunto de dados de treinamento.

```
import matplotlib.pyplot as plt
plt.imshow(x_train[0])
```

Output:



Esta é uma imagem, mas acho que a pode reconhecer como a número 5. Vou também mostrar-lhe a etiqueta de resposta correcta que se junta a esta imagem.

```
print(y_train[0])
```

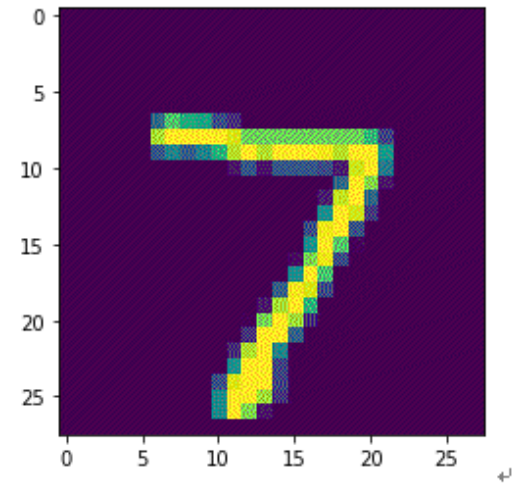
Output:

```
5
```

Pudemos ver 5, que reconhecemos pela imagem. Então veremos o conjunto de dados do teste da mesma forma.

```
plt.imshow(x_test[0])
```

Output:




```
y_test[0]
```

Output:

```
7
```

We irá treinar um modelo de aprendizagem profunda usando o conjunto de dados de 60.000 treinamentos e o par de etiquetas de resposta correta x_train e y_train que acabamos de usar para ver se podemos prever o número 7 a partir da imagem 7 neste conjunto de dados de teste. Começamos com o seguinte código para construir uma rede neural totalmente acoplada com três camadas intermediárias.

```
#  Building a Deep Learning Model
model = tf.keras.models.Sequential([
 tf.keras.layers.InputLayer(input_shape=(784, )),
 tf.keras.layers.Dense(256, activation='relu'),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(128, activation='relu'),
 tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(256, activation='relu'),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation='softmax')
                                    ])
# Compilation and overview of the model
model.compile(
   optimizer='adam',
   loss='categorical_crossentropy',
   metrics=['accuracy']
)

print(model.summary())
```

Uma visão geral do modelo construído pode ser visualizada no seguinte código

```
print(model.summary())
```

Output:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 256)               200960
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               33024
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                2570
=================================================================
Total params: 269,450
Trainable params: 269,450
Non-trainable params: 0
_________________________________________________________________
```

Você pode ver que fomos capazes de construir um modelo com 269.450 parâmetros. Este é um modelo de um tamanho que levaria um tempo razoável para treinar na CPU.

Nós pré-processamos os dados antes de os treinarmos. Como os dados 2D de (28, 28) não podem ser introduzidos na rede neural totalmente acoplada como está, temos de remodelar os dados em dados unidimensionais de (784,). Além disso, a informação de cor (RGB) é normalizada para estar entre 0 e 1.

```
# Preprocessing of MNIST data (flatten 28 x 28 pixel to 784 for input)
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# Casting to a float32 type because of a true divide error when dividing by itself
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalizes data to a float32 type in the range of 0 to 1
x_train /= 255
x_test /= 255

# Convert to categorical data (1 => [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Vamos treinar o modelo com o seguinte código, épocas é o número de vezes que temos de treinar um dado de treino, mas por agora vamos treiná-lo 20 vezes.

```
model.fit(x_train, y_train, epochs=20)
```

Output:

```
Epoch 1/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.1362 - accuracy: 0.9586
Epoch 2/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.1105 - accuracy: 0.9673
Epoch 3/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.0916 - accuracy: 0.9719
Epoch 4/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.0784 - accuracy: 0.9767
Epoch 5/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.0702 - accuracy: 0.9788
Epoch 6/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.0647 - accuracy: 0.9805
```

-Omitir 7-19 épocas para a brevidade

```
Epoch 20/20
1875/1875 [==============================] - 7s 4ms/step - loss:
0.0354 - accuracy: 0.9895
```

It pode depender do tempo de acesso e da velocidade da internet, mas leva cerca de 3 minutos para aprender. Graças à GPU do Google, pudemos aprender em um tempo super curto, enquanto que teríamos levado um dia inteiro para aprender se tivéssemos usado um PC local que não fosse capaz de fazer isso.

The precisão é uma percentagem de quantas vezes acertámos, numa escala de 0,0 a 1,0. 1,0 significa que acertamos 100 vezes em 100, enquanto 0,9895 significa que acertamos cerca de 99 vezes

em 100 nos dados de treinamento. Você pode ver que a porcentagem de respostas corretas aumenta a cada época, de cerca de 95% no final da primeira época para quase 99% na 20ª sessão de treinamento. A propósito, a perda é o erro entre a saída da rede neural e a resposta correta dada pelos dados de treinamento como uma avaliação do processo de aprendizagem pela rede neural. Neste capítulo, usaremos uma porcentagem intuitiva e fácil de entender das respostas corretas para explicar. A Matriz de Confusão e os valores F são às vezes usados para avaliar a precisão em situações em que são necessárias medições de precisão severas, como em pesquisa e desenvolvimento. No capítulo sobre a mais curta aprendizagem profunda do MNIST, vamos introduzir apenas o nome do método.

In o treinamento que acabamos de descrever, você está treinando em 60.000 conjuntos de dados de treinamento e fazendo previsões nesses 60.000 conjuntos de dados de treinamento. Este é o primeiro ponto de tropeço para iniciantes na aprendizagem de máquinas e aprendizagem profunda, mas mesmo que você faça previsões em relação aos dados usados para treinamento, você não sabe qual será a porcentagem de respostas corretas (ou seja, desempenho de generalização) em relação a outros dados que você ainda não tenha visto.

If olhamos para o seguinte código para avaliar o modelo em relação aos dados do teste, vemos que a percentagem de respostas correctas é de 0,9816. Podemos ver que as previsões nos dados de treinamento foram próximas a 99%, mas quando fazemos previsões nos dados do teste, a taxa de previsão cai cerca de 1%, e nos dados do MNIST, as características das imagens nos conjuntos de dados de treinamento e teste não foram tão diferentes, portanto a correção das previsões no conjunto de dados de treinamento e no teste. Embora a percentagem de respostas corretas no conjunto de dados não mostre uma grande discrepância, no mundo da ciência dos dados, onde o aprendizado e a previsão são realizados com

dados reais, não é raro que os dados de treinamento e os dados de teste tenham resultados de desempenho completamente diferentes.

```
model.evaluate(x_test, y_test, verbose=2)
```

Output:

```
313/313 - 1s - loss: 0.0954 - accuracy: 0.9816
[0.09539060294628143, 0.9815999865531921]
```

So dividimos o conjunto de 60.000 dados em 42.000 dados de treinamento e 18.000 dados de validação, treinamos usando apenas 42.000 dados, e depois vemos se ele pode ser validado contra dados que são desconhecidos para os 18.000 modelos. Chamamos a isto um "holdout". Na implementação, isto pode ser feito simplesmente passando o argumento validação_split: 0,3, 30% do conjunto de dados de treinamento, ou seja, 18.000, é usado para validação e 80%, ou seja, 42.000, é usado para treinamento.

```
model.fit(x_train, y_train, epochs=20, validation_split=0.3)
```

Output:

```
Epoch 1/20
1313/1313 [==============================] - 7s 6ms/step - loss:
0.0249 - accuracy: 0.9931 - val_loss: 0.0107 - val_accuracy: 0.9970
Epoch 2/20
1313/1313 [==============================] - 7s 6ms/step - loss:
0.0256 - accuracy: 0.9937 - val_loss: 0.0102 - val_accuracy: 0.9973
Epoch 3/20
1313/1313 [==============================] - 7s 6ms/step - loss:
0.0251 - accuracy: 0.9935 - val_loss: 0.0083 - val_accuracy: 0.9979
Epoch 4/20
1313/1313 [==============================] - 7s 6ms/step - loss:
0.0274 - accuracy: 0.9929 - val_loss: 0.0145 - val_accuracy: 0.9953
Epoch 5/20
1313/1313 [==============================] - 7s 5ms/step - loss:
0.0227 - accuracy: 0.9933 - val_loss: 0.0099 - val_accuracy: 0.9969
Epoch 6/20
```

-Omitir 7-19 épocas para a brevidade

```
Epoch 20/20
1313/1313 [==============================] - 7s 6ms/step - loss:
0.0235 - accuracy: 0.9946 - val_loss: 0.0397 - val_accuracy: 0.9906
<tensorflow.python.keras.callbacks.History at 0x7f92e5fa1780>
```

Adicionamos mais duas métricas à saída, val_loss e val_accuracy, que são os valores que avaliamos nos 18.000 dados de treinamento, e vimos que após 20 sessões de treinamento, a val_accuracy também está acima de 99%.

Agora, vamos ver até que ponto conseguimos acertar as previsões quando fazemos previsões sobre outros dados além do conjunto de dados de treinamento (dados de teste) em um modelo que foi treinado com 42.000 dados de treinamento.

```
model.evaluate(x_test, y_test, verbose=2)
```

Output:

```
313/313 - 1s - loss: 0.1341 - accuracy: 0.9811
```

Quando dividimos os 60.000 dados de treinamento em 42.000 dados de treinamento e 18.000 dados de verificação, e depois os treinamos com os dados de treinamento e os avaliamos em relação aos outros 10.000 dados de teste, a porcentagem de respostas corretas é de 0,9811, ou seja, prevemos o número manuscrito 100 vezes e o corrigimos 98 vezes Você pode ver que construímos um modelo que nos permite fazer isso.

Podemos ver que a percentagem de respostas correctas não é muito diferente (na verdade, é ligeiramente inferior) do que quando o tivemos treinado e previmos com o conjunto de dados de treino que acabámos de descrever.

Fizemos as seguintes hipóteses sobre a causa disso

1) Falta de um conjunto de dados de treinamento.
→ Num conjunto de dados de treinamento de 42.000 dados de treinamento, não é possível capturar as características do conjunto de 10.000 dados de treinamento para testes (ou seja, há uma falta de variação nos dados de treinamento)?

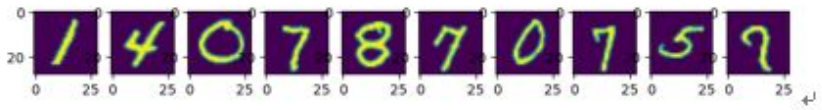
2) Ausência de épocas
→validation_split reduziu o número de dados para treinar o modelo de 60.000 para 42.000, portanto o treinamento ainda pode ser completado antes do pico da taxa de correção?

Vou exibir 10 dados aleatórios de cada treinamento e teste para verificar se (1) é verdade. Anteriormente, converti os dados da lista para planos no pré-processamento, então estou carregando-os novamente.

```
import matplotlib.pyplot as plt
(xo_train, yo_train), (xo_test, yo_test) =
tf.keras.datasets.mnist.load_data()

def plot_random10(mnist_data, title):
 """ Get 10 randomly from mnist data and draw
 """
 fig = plt.figure(figsize=(28,28), dpi=100)
 fig.suptitle(title, fontsize=18, x=0, y=0.92)
 for index, x in enumerate(mnist_data[np.random.choice(np.arange(0,
len(mnist_data)), 10, replace=False)], 1):
   ax = fig.add_subplot(28, 28, index)
   plt.imshow(x)

plot_random10(xo_train, 'Train Set')
plot_random10(xo_test, 'Test Set')
```

Output:

Train Set



Test Set

Ao verificar, descobri que os dados do teste (Test Set) contêm mais caracteres que parecem um pouco mais difíceis de decifrar do que os dados do treinamento (Train Set). Por exemplo, o terceiro dado da direita do Conjunto de Testes está tão desfocado que não tenho a certeza se é um 2 ou um 3.

Então, para testar a hipótese de 2), vou usar os resultados das épocas de treinamento 100 vezes para fazer uma previsão. Para poupar espaço, mostraremos apenas os resultados da 100ª sessão de treino.

```
model.fit(x_train, y_train, epochs=100, validation_split=0.3)
```

Output:

```
1313/1313 [==============================] - 4s 3ms/step - loss:
0.0205 - accuracy: 0.9951 - val_loss: 0.1809 - val_accuracy: 0.9784
```

Após o 100º treinamento, o conjunto de dados de treinamento tem uma taxa de correção de 0,9951, enquanto que o conjunto de dados de validação tem uma precisão de validação de 0,9784. Parece que o conjunto de dados se ajustava em excesso aos dados de treinamento (nota: ajuste em excesso aos dados de treinamento e perda de desempenho de generalização). Estes resultados sugerem que a hipótese (1) parece fazer mais sentido.

Agora, nós queremos resolver a próxima tarefa de prever usando o modelo treinado.
Com o seguinte código, podemos usar os dados do teste para realizar o reconhecimento da imagem.

```
prediction = model.predict(x_test)
```

Se verificarmos as previsões no código seguinte, o resultado é um array de 10 elementos, com números reais variando de 0 a 1. O índice do array com o maior número é o número previsto. (O índice do array começa em 0, mas você não precisa adicionar 1 porque os números MNIST também são preditos de 0 a 9).

```
print(prediction[0])
```

```
Output:
[1.4635492e-20 4.3042563e-11 7.6560958e-10 1.0590934e-11 5.7983397e-16
 5.9631535e-21 2.1780077e-25 1.0000000e+00 3.7560503e-15 3.0590461e-
11]
```

Esta forma é um pouco confusa, por isso podemos usar a função argmax do Numpy para retornar diretamente o subscrito do maior conjunto de valores.

```
import numpy as np

print(np.argmax(prediction[0]))
```

```
Output:
7
```

Você pode dizer se essa previsão está correta, emitindo a etiqueta de resposta correta.

```
print(np.argmax(y_test[0]))
```

```
Output:
7
```

Até agora, construímos um ambiente de aprendizagem profunda da GPU e até usámos o fluxo tensor da biblioteca de aprendizagem profunda para realizar o reconhecimento de imagens. Acho que provavelmente tivemos o aprendizado profundo mais rápido com o ambiente de GPU em um livro similar. No entanto, alguns de vocês podem ser capazes de executar o código, mas não têm idéia do que estão fazendo, mas não se preocupem, a Amazon, que tem sido uma grande ajuda para nós na publicação do Kindle e nas compras

on-line na nova era Corona, tem um lema da empresa chamado "Trabalhando ao contrário" e eles fazem os comunicados de imprensa que entregam aos seus clientes primeiro. Com isso em mente, usamos a política de "acostumar-se" neste livro para agilizar apenas alguns exemplos do que você será capaz de fazer quando aprender este livro. No próximo capítulo, vou começar a explicar o básico de Python e construir sobre o básico para chegar ao coração da análise de dados Python.

8. Python

In no capítulo anterior, começamos com um aprendizado profundo, que pode ter deixado alguns de vocês um pouco confusos, mas não se preocupem, vamos começar de novo neste capítulo, começando "do zero". Primeiro, vamos discutir a gramática básica de Python.

Olá Mundo Pitonico!

A programação Python foi concebida para simplificar e é muito fácil de ler e compreender, mesmo para principiantes. Se você é um leitor que já trabalhou com outras linguagens de programação, você ficará surpreso com o quão conciso e pequeno é o mesmo código quando escrito em Python.
Primeiro, eu gostaria de tocar nesta filosofia Python. Execute o seguinte código.

```
import this
```

Output:

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Esta é uma descrição da filosofia do design Python.

Antes de mais nada, tente o seguinte código para quebrar os seus ombros: "Olá Mundo Pitonico!

```
print("Hello Pythonic World!")
```

Output:

```
Hello Pythonic World!
```

Depois execute o seguinte programa: use a lista Python (lista) para exibir a tradução inglês-japonesa do par filosófico Python. Eu escolhi a tradução japonesa porque o primeiro elemento da lista pythonic é sobre a filosofia "Zen". "Zen" vem do Japão, "Zen" é basicamente a busca da verdade de quem você é. Zen é uma abreviação para Zen Budismo, um ramo do Budismo Mahayana, ou uma transcrição fonética da palavra sânscrita ध्यान (dhyāna, dhyana), Zenna.

```
pythonic_english_list = [
 "The Zen of Python, by Tim Peters",
 "Beautiful is better than ugly.",
 "Explicit is better than implicit.",
 "Simple is better than complex.",
 "Complex is better than complicated.",
 "Flat is better than nested.",
 "Sparse is better than dense.",
 "Readability counts.",
 "Special cases aren't special enough to break the rules.",
 "Although practicality beats purity.",
 "Errors should never pass silently.",
 "Unless explicitly silenced.",
 "In the face of ambiguity, refuse the temptation to guess.",
 "There should be one-- and preferably only one --obvious way to do it.",
 "Although that way may not be obvious at first unless you're Dutch.",
 "Now is better than never.",
 "Although never is often better than *right* now.",
 "If the implementation is hard to explain, it's a bad idea.",
 "If the implementation is easy to explain, it may be a good idea.",
 "Namespaces are one honking great idea -- let's do more of those!"
]

pythonic_japanese_list = [
 "Python哲学　ティム・ピーターズ",
 "「綺麗さ」は「汚さ」よりも良い。",
 "「明示的」は「暗示的」よりも良い。",
 "「シンプル」は「複雑」よりも良い。",
 "「込み入っている」よりは「複雑」な方がマシ。",
 "「ネスト構造」より「フラットな構造」の方が良い。",
 "「密」より「疎」の方が良い。",
 "「読みやすさ」が重要。",
 "特殊なケースはルールを破るほど特別ではない。",
 "実用性は純粋さに勝る。",
 "エラーは静かに通過してはいけない。",
 "明示的に静かにしていない限りは。",
 "曖昧なコードがあった時、推測して突き進む誘惑に負けないで。",
 "１つだけ、できれば１つだけ明らかに良い方法があるはずだ。",
 "その方法はオランダ人でない限り最初はわからないかもしれない。",
 "今が一番良い。",
 "しかし「今」は、「たった今」には負ける。",
 "その実装を説明するのが難しければ、それは悪いアイデアだ。",
```

```
 "その実装を説明するのが簡単であれば、それは良いアイデアかもしれない。",
 "ネームスペース（名前空間）はとても素晴らしいアイデアなので、もっと使うべきだ。"
]

for english, japanese in zip(pythonic_english_list, pythonic_japanese_list):
 print(f'英語：{english}／日本語：{japanese}')
```

Output:

```
英語：The Zen of Python, by Tim Peters／日本語：Python 哲学　ティム・ピーターズ
英語：Beautiful is better than ugly.／日本語：「綺麗さ」は「汚さ」よりも良い。
英語：Explicit is better than implicit.／日本語：「明示的」は「暗示的」よりも良い。
英語：Simple is better than complex.／日本語：「シンプル」は「複雑」よりも良い。
英語：Complex is better than complicated.／日本語：「込み入っている」よりは「複雑」な方がマシ。
英語：Flat is better than nested.／日本語：「ネスト構造」より「フラットな構造」の方が良い。
英語：Sparse is better than dense.／日本語：「密」より「疎」の方が良い。
英語：Readability counts.／日本語：「読みやすさ」が重要。
英語：Special cases aren't special enough to break the rules.／日本語：特殊なケースはルールを破るほど特別ではない。
英語：Although practicality beats purity.／日本語：実用性は純粋さに勝る。
英語：Errors should never pass silently.／日本語：エラーは静かに通過してはいけない。
英語：Unless explicitly silenced.／日本語：明示的に静かにしていない限りは。

英語：In the face of ambiguity, refuse the temptation to guess.／日本語：曖昧なコードがあった時、推測して突き進む誘惑に負けないで。
英語：There should be one-- and preferably only one --obvious way to do it.／日本語：1つだけ、できれば1つだけ明らかに良い方法があるはずだ。
英語：Although that way may not be obvious at first unless you're Dutch.／日本語：その方法はオランダ人でない限り最初はわからないかもしれない。
英語：Now is better than never.／日本語：今が一番良い。
英語：Although never is often better than *right* now.／日本語：しかし「今」は、「たった今」には負ける。
英語：If the implementation is hard to explain, it's a bad idea.／日本語：その実装を説明するのが難しければ、それは悪いアイデアだ。
英語：If the implementation is easy to explain, it may be a good idea.／日本語：その実装を説明するのが簡単であれば、それは良いアイデアかもしれない。
英語：Namespaces are one honking great idea -- let's do more of those!／日本語：ネームスペース（名前空間）はとても素晴らしいアイデアなので、もっと使うべきだ。
```

A lista Python é representada em Python, encerrando-os entre parênteses rectos[]. É comum que todos os elementos da lista sejam do mesmo tipo (todo o código acima mencionado é do tipo string). (Todo o código acima é do tipo string).

The para declaração é uma declaração de repetição. python usa uma sintaxe "para variáveis em lista:" que permite armazenar elementos em uma lista de variáveis e depois recuperá-los um de cada vez, começando com a primeira. Isto é chamado de "for loop" (para loop). Também, em Python, um indentação é um bloco de código. O comando print acima é recuado com dois espaços sob o comando for, para que possamos recuperar os elementos armazenados nas variáveis do bloco for.

The sintax for print(f'English:{english}/Japanese:{japanese}') está disponível na versão Python 3.6 e superior, e a citação simples após a f pode ser citada duas vezes. ("" ou ""), você pode escrever o nome da variável diretamente nas chaves para interpretar e exibir como uma string. f"{variable }". Existem outros métodos como o método de formato do tipo String e %, mas o autor recomenda usar a notação f"{}" porque é mais rápida e mais visível.

The "List:" parte do comando for pode ser qualquer tipo de iterador (um objeto com o método __iter__), então pode ser um tipo de dicionário, um tipo tuple, ou mesmo um método zip como o acima que pode expandir múltiplas listas em ordem desde o início . Vamos verificar se o zip tem um método __iter__, só para ter certeza.

```
print(zip.__iter__)
```

Output:

```
<slot wrapper '__iter__' of 'zip' objects>
```

Also, eu mencionei anteriormente que é comum armazenar elementos do mesmo tipo em todas as listas, mas como Python é uma linguagem que aloca memória dinamicamente e é solta em

tipos, também é possível colocar elementos de diferentes tipos - inteiro, string, ponto flutuante, dicionário, tuple, etc. - em um, como mostrado abaixo.

```
diversity_list = [1, 'One', 1.0, {'1':'One'}, {'One'}]
for element in diversity_list:
 print(element)
```

Output:

```
1
One
1.0
{'1': 'One'}
{'One'}
```

A maioria das linguagens de programação, exceto Python, não consegue alcançar uma lista tão flexível, certo? Vamos exibir os tipos no seguinte código também.

```
diversity_list = [1, 'One', 1.0, {'1':'One'}, {'One'}]
for element in diversity_list:
 print(type(element))
```

Output:

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'dict'>
<class 'set'>
```

A seguir, vamos escrever e executar uma função que retorna uma string da filosofia Python enquanto você digita e se está em japonês ou inglês.

```
def get_pythonic_language(texts):↓
  """ Return which language a pythonic sentence is written in.↓
↓
     @Args:↓
       texts(str):input text↓
      @Returns:↓
       language(str):language of the input text↓
  """↓
  pythonic_english_list = [↓
  "The Zen of Python, by Tim Peters",↓
  "Beautiful is better than ugly.",↓
  "Explicit is better than implicit.",↓
  "Simple is better than complex.",↓
  "Complex is better than complicated.",↓
  "Flat is better than nested.",↓
  "Sparse is better than dense.",↓
  "Readability counts.",↓
  "Special cases aren't special enough to break the rules.",↓
  "Although practicality beats purity.",↓
  "Errors should never pass silently.",↓
  "Unless explicitly silenced.",↓
  "In the face of ambiguity, refuse the temptation to guess.",↓
  "There should be one-- and preferably only one --obvious way to do it.",↓
  "Although that way may not be obvious at first unless you're Dutch.",↓
  "Now is better than never.",↓
  "Although never is often better than *right* now.",↓
  "If the implementation is hard to explain, it's a bad idea.",↓
  "If the implementation is easy to explain, it may be a good idea.",↓
  "Namespaces are one honking great idea -- let's do more of those!"↓
  ]↓
  pythonic_japanese_list = [↓
  "Python哲学　ティム・ピーターズ",↓
  "「綺麗さ」は「汚さ」よりも良い。",↓
  "「明示的」は「暗示的」よりも良い。",↓
  "「シンプル」は「複雑」よりも良い。",↓
  "「込み入っている」よりは「複雑」な方がマシ。",↓
  "「ネスト構造」より「フラットな構造」の方が良い。",↓
  "「密」より「疎」の方が良い。",↓
  "「読みやすさ」が重要。",↓
  "特殊なケースはルールを破るほど特別ではない。",↓
  "実用性は純粋さに勝る。",↓
  "エラーは静かに通過してはいけない。",↓
  "明示的に静かにしていない限りは。",↓
```

```
  "曖昧なコードがあった時、推測して突き進む誘惑に負けないで。",
  "１つだけ、できれば１つだけ明らかに良い方法があるはずだ。",
  "その方法はオランダ人でない限り最初はわからないかもしれない。",
  "今が一番良い。",
  "しかし「今」は、「たった今」には負ける。",
  "その実装を説明するのが難しければ、それは悪いアイデアだ。",
  "その実装を説明するのが簡単であれば、それは良いアイデアかもしれない。",
  "ネームスペース（名前空間）はとても素晴らしいアイデアなので、もっと使うべきだ。"
  ]
  language = ""
  if texts in pythonic_english_list:
    language = "english"
  elif texts in pythonic_japanese_list:
    language = "japanese"
  else:
    language = "not pythonic"
  return language

print(get_pythonic_language("Now is better than never."))
```

Output:

```
'english'
```

Here, nós usamos a sintaxe de uma instrução def de função e uma instrução if. A instrução def também é muito simples, escrita como "def nome da função (argumento):" e então no próximo bloco de código recuado, nós escrevemos "return return value" para pegar algum argumento de entrada e retornar o resultado do processo por Se você não incluir a instrução return, a função retornará o valor especial "None" como resultado da execução da função.

A declaração se é seguida por uma expressão condicional, e se a expressão condicional é cumprida, o bloco de código diretamente abaixo do código recuado é executado. Se a expressão condicional não corresponder à expressão condicional da instrução if, mas a expressão condicional da instrução elif corresponder, o bloco de código indentado diretamente abaixo da outra instrução é executado.

"(aspas duplas) A parte incluída nas três (aspas duplas) deve ser escrita para um comentário de várias linhas chamado docstring, e o # (hashtag) deve ser escrito para um comentário de uma linha. Ela é ignorada em tempo de execução do fonte, mas deve ser preenchida para melhorar a legibilidade do fonte.PEP8, as convenções de codificação Python, também especificam como escrever um bom docstring no PEP257.Para o docstring, você pode usar o estilo Google Style e Numpy é o estilo mais comum, mas este livro será descrito usando o subjetivo do autor e fácil de escrever no estilo Google.

So longe, revisamos a sintaxe básica e mínima da definição de funções e execução de declarações de controle, tocando na filosofia Python. No próximo capítulo, vamos começar a falar sobre Pandas, uma biblioteca que pode ser muito útil para analisar dados reais como entrada.

9. Pandas

O significado de Pandas

Pandas é uma ferramenta muito poderosa para análise de dados usando dados reais, e embora seja similar ao EXCEL, planilhas e RDB, os benefícios de usar Pandas ainda são ótimos!

What faz com que a biblioteca de pandas se destaque do EXCEL é a sua capacidade de processar grandes quantidades de dados a uma velocidade ultra-alta, especialmente quando se trata de manusear 100MB de dados de texto, e congela e trava. Ela também pode ser executada com rapidez em arquivos e suporta a leitura de dados no formato EXCEL.
As declarações "para" do Python são conhecidas por serem lentas, mas Pandas usa Numpy internamente, e como o Numpy é escrito em C, ele corre tão rápido quanto a linguagem C.

Tipo de DataFrame

Before enviando seus dados para pandas, primeiro precisamos preparar os dados para tipos de dicionários e compreensões de listas. Um tipo de dicionário é um tipo de dados onde a chave e o valor são emparelhados. No exemplo a seguir, definimos as Chaves 'Ano' e 'Índice' e seus correspondentes tipos de lista Valores. No capítulo anterior, definimos os tipos de lista um por um usando caracteres literais, mas desta vez usamos notação de "compreensão de lista". A sintaxe das compreensões de lista é [Variables for Variables in Iterator Type]. Agora você pode retornar uma lista de inteiros gerados pelo tipo de iterador, armazenados em ordem.

You pode usar a função de intervalo para gerar uma sequência de números inteiros num intervalo especificado. Passar um único argumento produz uma lista de números inteiros que são incrementados em 1 de 0 (o tamanho da lista é gerado para o número especificado de argumentos). Se você passar dois argumentos, você pode especificar o início e o fim da lista mais um. Por exemplo, você pode escrever [i para i no intervalo(1, 3)] para gerar [1, 2]. É uma boa idéia lembrar disso, assim como escrever uma lista[start:end+1] quando você obtém o intervalo especificado de uma lista.

Em 'Ano' poderíamos definir uma lista que contém elementos que aumentam em 1 de 2010 a 2030, e em 'Índice' poderíamos definir uma variável tipo dicionário year_index que permite listas de 0 a 20.

```
import pandas as pd
year_index = {'Year':[y for y in range(2010, 2031)],
              'Index':[i for i in range(21)],
             }
print(year_index)
```

Output:

```
{'Year': [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019,
2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030],
'Index': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20]}
```

O tipo mais comum usado em pandas é o tipo DataFrame, seguido pelo tipo Série e, dependendo do trabalho, o tipo Painel, que não é usado tanto. Vou explicar apenas os tipos DataFrame neste livro.

Primeiro, vamos converter o ano_index do tipo léxico que definimos anteriormente para um tipo de quadro de dados para Pandas e armazená-lo em uma variável.

```
df = pd.DataFrame(year_index)
```

E vou usar um método prático de tipo frame de dados para ver o que está nos dados tão facilmente como está no EXCEL.

Usando DataFrame.head(), podemos ver os cinco primeiros registros dos dados.

```
print(df.head())
```

Output:

```
   Year  Index
0  2010      0
1  2011      1
2  2012      2
3  2013      3
4  2014      4
```

Consegui exibir facilmente os primeiros cinco registos em forma de tabela como esta.
Como já deve ter reparado com os tipos em execução no Google Colaboratory e Jupyter Notebook, a função de impressão pode apresentar variáveis sem ter de as escrever. A razão pela qual as temos escrito explicitamente no passado é para que elas possam ser executadas em um ambiente de execução diferente (por exemplo, REPL). Se você der um argumento à cabeça, você pode exibir desde o início até o argumento especificado.

```
df.head(10)
```

Output:

	Year	Index
0	2010	0
1	2011	1
2	2012	2
3	2013	3
4	2014	4
5	2015	5
6	2016	6
7	2017	7
8	2018	8
9	2019	9

Você pode usar o método DataFrame.sample para buscar e exibir os registros aleatoriamente com um número especificado de argumentos. Isso é útil quando você tem dados que têm uma

tendência completamente diferente do início ao meio ou ao fim dos dados.

```
df.sample(5)
```

Output:

	Year	Index
7	2017	7
2	2012	2
9	2019	9
16	2026	16
8	2018	8

A seguir, vamos experimentar os recursos de visualização de Pandas, mas visualizar o Índice não é divertido, então vamos usar um módulo chamado random para gerar um número aleatório para cada elemento do Ano e armazená-lo em uma variável do tipo DataFrame, df1.

```
import random
years_random1 = {'Year':[y for y in range(2010, 2031)],
                 'Random':[random.random() for y in range(21)],
                }
df1 = pd.DataFrame(years_random1)
```

Visualização apenas com Pandas

Pandas também faz uso de matplotlib internamente. Portanto, o tipo DataFrame tem um método chamado "plot" que pode facilmente visualizar os dados armazenados nele. Por exemplo, se você quiser exibir um gráfico de dispersão de df1 com Ano como eixo x e Aleatório como eixo y, o código a seguir servirá perfeitamente para gráficos simples, como os xticks para especificar as etiquetas de tick do eixo x.

```
df1.plot(x=df1.columns[0], y=df1.columns[1], kind='scatter', xticks=[year
for year in range(2010, 2031, 5)])
```

Output:

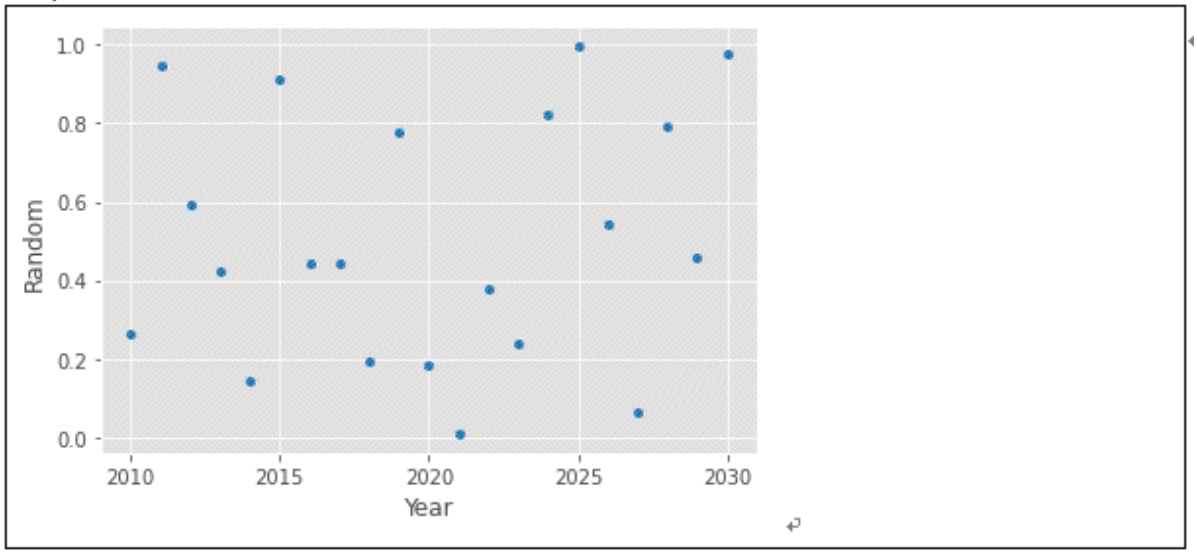


Tabela junta-se com uma chave específica

Next, vamos olhar para as mesas de junção. VLOOOKUP e XLOOOKUP do EXCEL também são úteis, mas quando a quantidade de dados se torna grande, pode congelar instantaneamente. Usando Pandas, você pode juntar rapidamente tabelas umas às outras usando colunas específicas como chaves.

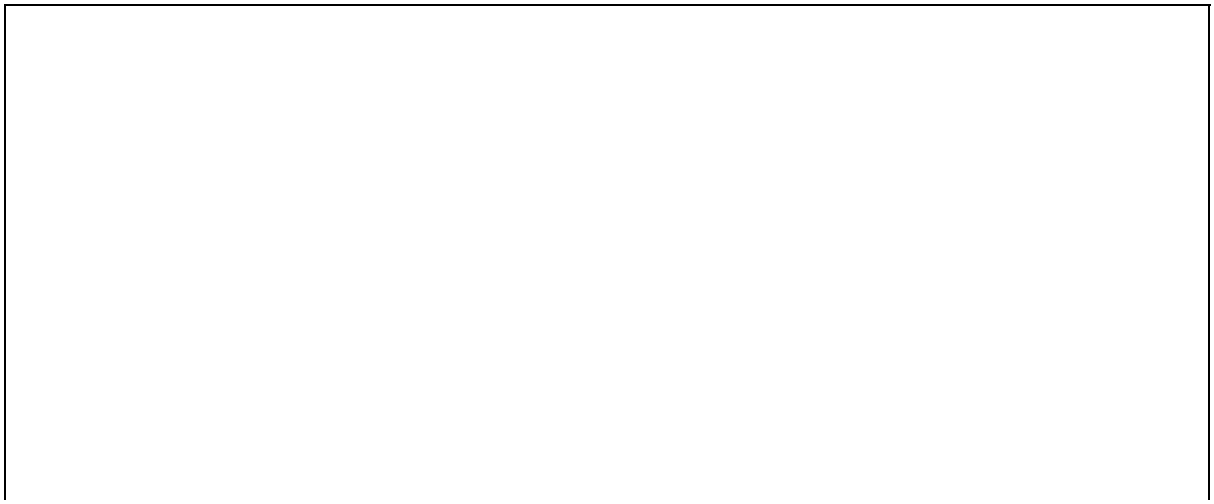
Primeiro, vamos ter diferentes DataFrames para combinar.

```
years_random2 = {'Year':[y for y in range(2011, 2032)],
                 'Random':[random.random() for y in range(21)],
                }
df2 = pd.DataFrame(years_random2)
```

Primeiro, acho que a maneira mais comum de usar isto é usar uma coluna como chave e apenas puxar registros que existem em ambas as tabelas, mas você pode fazer isto da seguinte forma. A seguir, estamos usando o Ano como chave e puxando apenas os registros que existem em ambas as tabelas. Originalmente, tanto df1 como df2 tinham 21 registros, mas após a fusão, o número de registros agora é de 20.

```
pd.merge(df1, df2, on='Year', how='inner')
```

Output:



	Year	Random_x	Random_y
0	2011	0.250203	0.288115
1	2012	0.601560	0.610345
2	2013	0.644055	0.044870
3	2014	0.856640	0.960191
4	2015	0.465893	0.221651
5	2016	0.132779	0.537764
6	2017	0.849658	0.312864
7	2018	0.273191	0.864590
8	2019	0.970955	0.615659
9	2020	0.732940	0.193200
10	2021	0.959330	0.973050
11	2022	0.324461	0.607832
12	2023	0.956938	0.871722
13	2024	0.448081	0.197782
14	2025	0.851742	0.327532
15	2026	0.652935	0.061544
16	2027	0.685593	0.163434
17	2028	0.611289	0.649251
18	2029	0.951157	0.862454
19	2030	0.569948	0.665695

Uma união externa pode então ser feita da seguinte forma para extrair todos os registros que existem em uma das duas tabelas, usando uma coluna como chave. A seguir, usamos o Ano como chave para extrair todos os registros que existem em qualquer uma das tabelas. Note que os registros que existem em apenas uma tabela são NaNs na outra, e existem 22 registros na tabela.

```
pd.merge(df1, df2, on='Year', how='outer')
```

Output:

	Year	Random_x	Random_y
0	2010	0.077238	NaN
1	2011	0.250203	0.288115
2	2012	0.601560	0.610345
3	2013	0.644055	0.044870
4	2014	0.856640	0.960191
5	2015	0.465893	0.221651
6	2016	0.132779	0.537764
7	2017	0.849658	0.312864
8	2018	0.273191	0.864590
9	2019	0.970955	0.615659
10	2020	0.732940	0.193200
11	2021	0.959330	0.973050
12	2022	0.324461	0.607832
13	2023	0.956938	0.871722
14	2024	0.448081	0.197782
15	2025	0.851742	0.327532
16	2026	0.652935	0.061544
17	2027	0.685593	0.163434
18	2028	0.611289	0.649251
19	2029	0.951157	0.862454
20	2030	0.569948	0.665695
21	2031	NaN	0.239212

A seguir, juntamos df1 e df2 usando as juntas exteriores esquerdas. Note que na união externa esquerda, deixamos sempre intacto o ano existente em df1.

```
pd.merge(df1, df2, on='Year', how='left')
```

Output:

	Year	Random_x	Random_y
0	2010	0.077238	NaN
1	2011	0.250203	0.288115
2	2012	0.601560	0.610345
3	2013	0.644055	0.044870
4	2014	0.856640	0.960191
5	2015	0.465893	0.221651
6	2016	0.132779	0.537764
7	2017	0.849658	0.312864
8	2018	0.273191	0.864590
9	2019	0.970955	0.615659
10	2020	0.732940	0.193200
11	2021	0.959330	0.973050
12	2022	0.324461	0.607832
13	2023	0.956938	0.871722
14	2024	0.448081	0.197782
15	2025	0.851742	0.327532
16	2026	0.652935	0.061544
17	2027	0.685593	0.163434
18	2028	0.611289	0.649251
19	2029	0.951157	0.862454
20	2030	0.569948	0.665695

Agora vamos olhar para as juntas exteriores certas, que sempre deixam registos que existem em df2.

```
pd.merge(df1, df2, on='Year', how='right')
```

Output:

	Year	Random_x	Random_y
0	2011	0.250203	0.288115
1	2012	0.601560	0.610345
2	2013	0.644055	0.044870
3	2014	0.856640	0.960191
4	2015	0.465893	0.221651
5	2016	0.132779	0.537764
6	2017	0.849658	0.312864
7	2018	0.273191	0.864590
8	2019	0.970955	0.615659
9	2020	0.732940	0.193200
10	2021	0.959330	0.973050
11	2022	0.324461	0.607832
12	2023	0.956938	0.871722
13	2024	0.448081	0.197782
14	2025	0.851742	0.327532
15	2026	0.652935	0.061544
16	2027	0.685593	0.163434
17	2028	0.611289	0.649251
18	2029	0.951157	0.862454
19	2030	0.569948	0.665695
20	2031	NaN	0.239212

Entrada/saída de arquivo CSV (versão compatível com Colab)

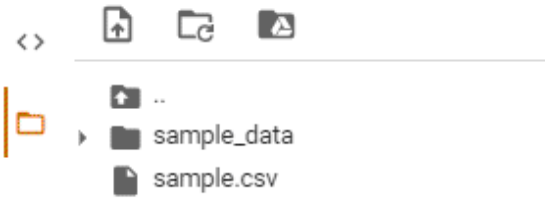
Ao analisar dados reais, muitas vezes você precisa ler e escrever arquivos CSV, e pandas faz com que seja muito fácil fazê-lo. Tanto que as pessoas frequentemente usam Pandas apenas com o propósito de importar arquivos CSV, porque é tão intuitivo.

Let's pegue o df1 que acabamos de definir e exporte-o para um arquivo CSV usando o método Pandas.to_csv. O Google Colab também tem uma funcionalidade útil para montar o Google Drive. Neste caso, vamos dar uma olhada em uma maneira mais conveniente de produzir o CSV para uma área que é descartada por sessão.

O seguinte código irá emitir um arquivo chamado sample.csv para a área temporária.

```
df1.to_csv('sample.csv', index=None)
```

Clique no ícone da pasta no lado esquerdo da tela do Colab e se a saída foi bem sucedida, você verá uma amostra.csv.



Você também pode baixar o arquivo de saída usando o menu de clique com o botão direito do mouse.

Em seguida, o arquivo CSV é lido com o método Pandas.read_csv como mostrado abaixo, e especificando a codificação, podemos também ler o arquivo de codificação CP932 do Windows. Como acabamos de fazer o to_csv no ambiente Python do Colab, a saída está em UTF-8 e não precisamos especificá-lo, mas se quisermos ler o arquivo CSV do Windows, precisamos definir encoding='cp932' como Por favor note que df_r.head() também mostra as 5 primeiras linhas para ter certeza de que elas estão carregadas corretamente.

```
df1_r = pd.read_csv('sample.csv', encoding='utf8')
df1_r.head()
```

Output:

	Year	Random
0	2010	0.991556
1	2011	0.435058
2	2012	0.224170
3	2013	0.753400
4	2014	0.442977

Obter qualquer célula ou intervalo de células

Vamos aprender um pouco mais sobre as operações básicas dos pandas, tal como o EXCEL.
Se você quiser obter uma célula específica em Pandas, use DataFrame.at ou DataFrame.iat. (O número de linhas começa em zero.) A diferença entre elas é que o segundo argumento pode ser ou um nome de coluna ou um índice da coluna. A diferença entre os dois é se o segundo argumento especifica o nome da coluna ou o índice da coluna.

```
df1_r.at[3, 'Year']
```

Output:

```
2013
```

```
df1_r.iat[3, 0]
```

Output:

```
2013
```

Então, se você quiser obter os valores de um intervalo de células, como no EXCEL, você pode usar DataFrame.loc ou iloc. Assim como a diferença entre at e iat, há uma diferença entre especificar nomes de colunas e índices de colunas. Além disso, você deve observar que o índice de linha do loc é diferente do fatiamento de uma lista Python, pois é um intervalo que inclui o endpoint, como você pode ver no código abaixo.

```
df1_r.loc[0:2, 'Year':'Random']
```

Output:

	Year	Random
0	2010	0.991556
1	2011	0.435058
2	2012	0.224170

```
df1_r.iloc[0:3, 0:2]
```

Output:

	Year	Random
0	2010	0.991556
1	2011	0.435058
2	2012	0.224170

Recuperar apenas os valores das células que preenchem condições específicas.

Outra maneira conveniente de escrever tipos DataFrame é usar a simples df[expressão condicional] para recuperar apenas os registros que correspondem às condições do tipo DataFrame pandas. Por exemplo, em df1_r, para recuperar apenas os registros a partir do ano 2020, você pode usar a seguinte sintaxe Existem duas maneiras de escrevê-lo, mas isso significa exatamente a mesma coisa. As duas maneiras de codificar são mostradas abaixo.

```
df1_r[df1_r.Year>=2020]
↓
df1_r[df1_r['Year']>=2020]
```

Output:

	Year	Random
10	2020	0.580729
11	2021	0.606586
12	2022	0.847076
13	2023	0.872025
14	2024	0.535352
15	2025	0.854549
16	2026	0.657712
17	2027	0.477294
18	2028	0.637123
19	2029	0.520275
20	2030	0.942131

Impacto da nova coroa nos preços das ações como visto em Pandas

No final de Pandas, gostaríamos de mostrar um exemplo prático usando a biblioteca pandas-datareader, que você pode instalar com o seguinte comando. A propósito, por favor instale Jupyter
No Notebook, se você começar com "!" você pode executar diretamente comandos no ambiente onde o Jupyter Notebook está rodando (Command Prompt for Windows ou Terminal for UNIX). É chamado de "comando mágico" e é muito útil.

```
!pip install pandas-datareader
```

O pandas-datareader facilita o acesso a uma variedade de fontes na web e a recuperação de dados do tipo pandas.DataFrame. As seguintes fontes de dados são suportadas

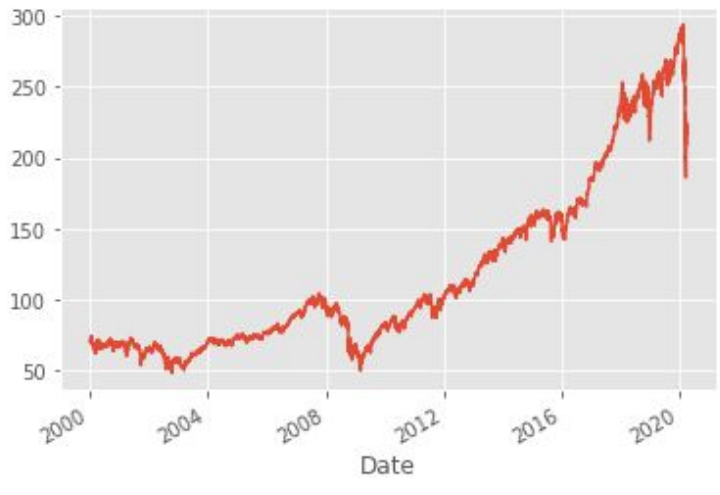
- Tiingo
- IEX
- Alpha Vantage
- Enigma
- Quandl
- St.Louis FED (FRED)
- Biblioteca de dados do Kenneth French
- Banco Mundial
- OCDE
- Eurostat
- Plano de Poupança com Poupança
- Definições do símbolo Nasdaq Trader Trader
- Stooq
- MOEX

O YAHOO Finance API está disponível a partir de abril de 2020, mas consulte a documentação oficial, pois ela pode ficar indisponível devido a alterações no API. Abaixo está uma visualização do preço médio de fechamento industrial Dow Jones de 1/1/2000 a 4/1/2020. Também mostra os últimos 100 registros de dados.

```
import datetime
import pandas_datareader.data as web
import matplotlib.pyplot as plt
from matplotlib import style

style.use('ggplot')
start = datetime.datetime(2000,1,1)
end = datetime.datetime(2020,4,1)

df = web.DataReader("dia", "yahoo", start, end).dropna()
print(df.tail(100))

df['Adj Close'].plot()
plt.show()
```

Output:

```
                 High         Low  ...      Volume   Adj Close
Date                               ...
2019-11-07  278.049988  276.559998  ...   3023800.0  273.987885
2019-11-08  277.239990  276.049988  ...   1512000.0  274.007690
2019-11-11  277.410004  275.390015  ...   1729000.0  274.096680
2019-11-12  277.989990  276.640015  ...   1694000.0  274.225220
2019-11-13  278.399994  276.170013  ...   2595600.0  274.976837
...                ...         ...  ...         ...         ...
2020-03-26  225.869995  214.009995  ...  16071900.0  225.070007
2020-03-27  223.259995  214.570007  ...  10545600.0  216.350006
2020-03-30  223.720001  215.100006  ...   7691800.0  223.100006
2020-03-31  224.750000  218.440002  ...   8426600.0  219.229996
2020-04-01  214.779999  207.770004  ...   8570300.0  209.380005

[100 rows x 6 columns]
```



Só de olhar aqui, podemos ver que o impacto recente das novas coroas na Média Industrial Dow Jones tem sido notável, com uma queda muito mais violenta e directa nos preços das acções do que a forma como caíram durante o Choque Lehman de 2008.

10. Numpy

O significado do Numpy

Numpy é uma biblioteca muito importante na qual se baseiam outras bibliotecas de análise de dados (por exemplo, Pandas/Matplotlib/SciPy), e é uma das principais razões pelas quais Python se tornou uma linguagem tão importante. Numpy permite operações de dados super-rápidas em arrays multidimensionais.

Numpy usa um formato de retenção de dados chamado ndarray, que, ao contrário das listas Python acima mencionadas, só pode armazenar dados do mesmo tipo em ndarray, mas em vez disso aloca os dados continuamente na memória (RAM). Ao armazenar dados em uma área contínua da memória, os dados podem ser lidos eficientemente em registros de CPU.

Another A maior vantagem é que o Numpy está ligado a bibliotecas lineares como BLAS e LAPACK, que realizam operações de vectorização na CPU em tempo de compilação, para que possa usufruir dos benefícios das operações paralelas de alta velocidade sem ter conhecimento delas.

In para usar o numpy, você precisa importá-lo com o numpy de importação como np. A partir de abril de 2020, o Colab parece ter sido ligado ao openblas, que verificamos abaixo para ver quais bibliotecas de vetorização de CPU estão ligadas e compiladas no Numpy.

```
import numpy as np
np.__config__.show()
```

Output:

```
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
```

Now vamos dar uma olhada nos tipos de lista Python e como o ndarray do Numpy é armazenado na memória usando o método id, que é semelhante ao uso do método id para obter o endereço de uma área de memória alocada. O Python do Google Colaboratory é escrito na linguagem C (CPython).

Portanto, o valor obtido pelo método id é um valor inteiro que é lançado como um ponteiro longo não assinado para o PyObject de uma programação em C, que é um número com implicações do tipo endereço.

In o seguinte código, usamos a notação de compreensão de lista que aprendemos anteriormente para gerar uma lista de lista1 com 10 números aleatórios como elementos, converter a lista para um tipo ndarray e armazená-la em np_arr.

```
import random
list1 = [random.random() for i in range(10)]
np_arr = np.array(list1)
```

O seguinte código recupera então os valores do índice, ID e elementos de np_arr, na ordem dos índices, e os exibe

```
for i, elm in enumerate(np_arr):
 print(i, id(np_arr[i]), elm)
```

Output:

```
0 140481734505192 0.4730826510610239
1 140481734505144 0.4876796906696862
2 140481734505192 0.9450564589749871
3 140481734505144 0.5117794392437419
4 140481734505192 0.6818941064583345
5 140481734505144 0.5795670075426068
6 140481734505192 0.5898959343675029
7 140481734505144 0.36767021804934696
8 140481734505192 0.613532458907983
9 140481734505144 0.48151774232816447
```

Vejo que a memória de np_arr do tipo ndarray tem o mesmo valor para 0 e pares e o mesmo valor para números ímpares. Isto é porque eles são armazenados em uma atribuição contínua de espaço.

```
for i, elm in enumerate(list1):
 print(i, id(list1[i]), elm)
```

Output:

```
0 140481734504784 0.4730826510610239
1 140481734504904 0.4876796906696862
2 140481734504832 0.9450564589749871
3 140481734505216 0.5117794392437419
4 140481734504808 0.6818941064583345
5 140481734505168 0.5795670075426068
6 140481734505240 0.5898959343675029
7 140481734505264 0.36767021804934696
8 140481734505288 0.613532458907983
9 140481734505312 0.48151774232816447
```

Por outro lado, a memória do tipo lista1 é toda diferente, e podemos ver que ela é armazenada de forma descontínua. Como os tipos de lista python são tipados dinamicamente, é intuitivo que armazená-los assim em endereços distantes leva tempo para serem recuperados.

Função de Geração Numpy Array

Numpy fornece uma variedade de características de geração de matriz multidimensional. Tenha em mente que isto é útil para a geração de matrizes multidimensionais de uma forma específica com valores iniciais para aprendizagem da máquina e processamento de imagem.
Numpy.zeros pode gerar um array com todos os elementos zero de uma determinada forma. Por exemplo, o seguinte irá gerar um array de 3 filas e 3 colunas com todos os 0's

```
import numpy as np

z = np.zeros((3,3))
print(z)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Vamos mostrar o resumo do array gerado pelo Numpy.zeros, e você pode ver que ele é um array ndarray, 3 x 3. Além disso, cada elemento é um item de 8 bytes com um tipo float64. Nós podemos obter o número de elementos em tamanho e podemos obter o número de dimensões em ndim. O número de dimensões é o número de eixos. Um erro comum é assumir que, como existem três colunas em cada linha, o número de dimensões é tridimensional. A interpretação correta é que a matriz está aninhada (isto é, encerrada em []) e o ninho mais externo é o eixo 0, depois o ninho externo é o eixo 1, e assim por diante, que é bidimensional.

```
def print_labeled_val(label, val):
 print(f"{label}:{val}")
 def print_ndr_basic_info(z):
 print_labeled_val('type', type(z))
 print_labeled_val('shape', z.shape)
 print_labeled_val('itemsize', z.itemsize)
 print_labeled_val('size', z.size)
 print_labeled_val('dtype', z.dtype)
 print_labeled_val('ndim', z.ndim)

print_ndr_basic_info(z)
```

Output:

```
type:<class 'numpy.ndarray'>
shape:(3, 3)
itemsize:8
size:9
dtype:float64
ndim:2
```

Como é importante explicar o número de dimensões, vamos usar outro exemplo para ilustrá-lo um pouco mais: O Numpy.arange permite criar um array unidimensional com um elemento inicial e um elemento final de +1, e o Numpy.reshape permite transformar a forma, portanto o seguinte código define um array tridimensional de 2 x 3 x 4 pode ser feito. Neste caso, os colchetes mais externos implicam o eixo 0, de modo que podemos ver que há dois elementos no eixo 0. Há também três elementos dentro do segundo colchete externo, e o terceiro colchete externo (ou seja, o colchete interno) tem quatro elementos dentro dele. Como existem três pares de colchetes do exterior, seria mais fácil lembrar que ndim=3, representando os eixos 0, 1, e 2 do exterior em ordem.

```
z = np.arange(0,24).reshape(2,3,4)
z
```

Output:

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

Os Numpy.zeros que mencionei anteriormente geraram um tipo float64 por padrão. Se você não usar cálculos científicos de alta precisão, então um tipo float32 pode ser suficiente em muitos casos. O exemplo a seguir mostra como gerar Numpy.zeros com um tipo float32. O exemplo a seguir mostra como gerar Numpy.zeros, que requer metade do tamanho da memória.

```
z = np.zeros(10, dtype='float32')
print(z.dtype)
print(z.itemsize)
```

Output:

```
float32
4
```

Em seguida, gerar uma matriz com os elementos todos inicializados em 1 como se segue

```
z = np.ones(10, dtype='float32')
z
```

Output:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], dtype=float32)
```

Para acelerar a geração de matrizes, um método para gerar matrizes multidimensionais da forma especificada sem inicializar em 0 ou 1 também é mostrado abaixo.

```
z = np.empty(10, dtype='float32')
z
```

Output:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

O seguinte é um exemplo de utilização do Numpy.linspace, que permite gerar um ndarray a partir de um elemento inicial especificado para um número especificado de elementos finais, divididos igualmente em um número especificado de elementos. No exemplo a seguir, geramos um ndarray contendo 5 elementos, divididos por 2 a 10 em incrementos de 2.

```
z = np.linspace(2, 10, 5)
z
```

Output:

```
array([ 2.,  4.,  6.,  8., 10.])
```

Dicas para o caderno de notas Jupyter

Aqui está um pequeno truque no Jupyter Notebook, você pode adicionar uma marca "?" após o objeto.
Quando você executar a célula, você verá um docstring sobre como usá-la. Se você esquecer como usá-lo, ele é mais rápido do que ir à página de ajuda oficial, para que você possa passar pelo processo de verificação de forma mais eficiente. Por exemplo, vamos exibir a ajuda para o np.linspace

```
np.linspace?
```

Output:

```
Signature: np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None,
axis=0)
Docstring:
Return evenly spaced numbers over a specified interval.

Returns `num` evenly spaced samples, calculated over the
interval [`start`, `stop`].

The endpoint of the interval can optionally be excluded.

.. versionchanged:: 1.16.0
   Non-scalar `start` and `stop` are now supported.

Parameters
----------
start : array_like
   The starting value of the sequence.
stop : array_like
   The end value of the sequence, unless `endpoint` is set to False.
   In that case, the sequence consists of all but the last of ``num + 1``
   evenly spaced samples, so that `stop` is excluded.  Note that the step
   size changes when `endpoint` is False.
num : int, optional
   Number of samples to generate. Default is 50. Must be non-negative.
endpoint : bool, optional
   If True, `stop` is the last sample. Otherwise, it is not included.
   Default is True.
retstep : bool, optional
   If True, return (`samples`, `step`), where `step` is the spacing
   between samples.
dtype : dtype, optional
   The type of the output array.  If `dtype` is not given, infer the data
   type from the other input arguments.

   .. versionadded:: 1.9.0

```

```
axis : int, optional
    The axis in the result to store the samples.  Relevant only if start
    or stop are array-like.  By default (0), the samples will be along a
    new axis inserted at the beginning. Use -1 to get an axis at the end.

    .. versionadded:: 1.16.0

Returns
-------
samples : ndarray
    There are `num` equally spaced samples in the closed interval
    ``[start, stop]`` or the half-open interval ``[start, stop)``
    (depending on whether `endpoint` is True or False).
step : float, optional
    Only returned if `retstep` is True

    Size of spacing between samples.


See Also
--------
arange : Similar to `linspace`, but uses a step size (instead of the
         number of samples).
geomspace : Similar to `linspace`, but with numbers spaced evenly on a log
            scale (a geometric progression).
logspace : Similar to `geomspace`, but with the end points specified as
           logarithms.

Examples
--------
>>> np.linspace(2.0, 3.0, num=5)
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
>>> np.linspace(2.0, 3.0, num=5, endpoint=False)
array([2. ,  2.2,  2.4,  2.6,  2.8])
>>> np.linspace(2.0, 3.0, num=5, retstep=True)
(array([2.  ,  2.25,  2.5 ,  2.75,  3.  ]), 0.25)

Graphical illustration:

>>> import matplotlib.pyplot as plt
>>> N = 8
>>> y = np.zeros(N)
>>> x1 = np.linspace(0, 10, N, endpoint=True)
>>> x2 = np.linspace(0, 10, N, endpoint=False)
>>> plt.plot(x1, y, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.plot(x2, y + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.ylim([-0.5, 1])
(-0.5, 1)
```

```
>>> plt.show()
File:      /usr/local/lib/python3.6/dist-packages/numpy/core/function_base.py
Type:      function
```

O ndarray gerado pelo método Numpy.linspace também é um tipo float64, e ao contrário dos zeros e uns, não se pode especificar o tipo inicial de ndarray. Portanto, é importante lembrar como mudar para um tipo float32 mais tarde.

```
print(z.dtype)
z = z.astype('float32')
print(z.dtype)
```

Output:

```
float64
float32
```

Muitas vezes é o caso de você querer usar números aleatórios para gerar um número especificado de números inteiros em um intervalo especificado. Se você quiser reproduzir o resultado (ou seja, você quer gerar o mesmo valor no tempo de repetição = reprodutibilidade), você pode fixar a semente da seguinte forma: O argumento da semente pode ser qualquer tipo de número inteiro, mas note que você precisa dar o mesmo número cada vez para garantir a reprodutibilidade.

Neste exemplo, geramos 6 arrays para números de 0-9.

```
np.random.seed(0)
z1 = np.random.randint(10, size=6)
z1
```

```
array([5, 0, 3, 3, 7, 9])
```

Aqui está um exemplo de conversão de um tipo de lista Python para ndarray, que pode ser feito com bastante facilidade com o método Numpy.array.

```
b_list = [[9,8,7],[1,2,3],[4,5,6]]
z = np.array(b_list)
z
```

Output:

```
array([[9, 8, 7],
       [1, 2, 3],
       [4, 5, 6]])
```

O ndarray gerado foi encontrado como um array 3 x 3 2D com cada elemento sendo do tipo int64.

```
print_ndr_basic_info(z)
```

Output:

```
type:<class 'numpy.ndarray'>
shape:(3, 3)
itemsize:8
size:9
dtype:int64
ndim:2
```

Numpy.ndarray's Indexing/Slicing

Recuperar os elementos do ndarray é semelhante ao tipo de lista Python, por isso vamos mantê-lo simples.
Primeiro, considere o código para obter a primeira linha de código dos elementos z do tipo ndarray que vimos anteriormente, que foram dispostos da seguinte forma.

```
array([[9, 8, 7],
       [1, 2, 3],
       [4, 5, 6]])
```

O índice ndarray também começa em 0, por isso temos o seguinte.

```
z[0]
```

Output:

```
array([9, 8, 7])
```

A seguir, aqui está o código para obter os elementos da primeira coluna da primeira linha: note que as listas bidimensionais de Python usam l[0][0], mas Numpy.ndarray usa uma lista de linhas e colunas separadas por vírgula.

```
z[0,0]
```

Output:

```
9
```

O corte pode ser usado da mesma forma que as listas Python, exceto que as arrays 2D requerem um intervalo de linhas e colunas separado por vírgula.

```
z[0:2, 0:2]
```

Output:

```
array([[9, 8],
       [1, 2]])
```

Para obter a última linha e coluna, escreva o seguinte.

```
z[-1, -1]
```

Output:

```
6
```

Processamento de imagem CIFAR10 em Numpy

At no final do capítulo Numpy, vamos lidar com imagens no Numpy como um exemplo prático, eu gostaria de usar uma imagem colorida RGB chamada CIFAR10.

1.ILSVRC (Concurso Internacional de Reconhecimento de Imagem) 2012 vencedor AlexNet's
2. mantido por Alex Krizhevsky
Imagem colorida 3.RGB
4. 10 classes de etiquetas: avião, automóvel, pássaro, gato, gato, veado, cão, rã, cavalo, navio e camião
5.60.000 imagens (50.000 imagens de treinamento e 10.000 imagens de teste)
6. O tamanho da imagem é 32 pixels x 32 pixels
7. Carregável a partir de câmeras (Biblioteca de invólucros do Tensorflow)

First, vamos usar keras para carregar os dados cifar10. Os conjuntos de dados carregados são todos do tipo ndarray, e podemos ver que existem 50.000 dados de treinamento e 10.000 dados de teste para 32x32 pixels de dados RGB. Podemos ver que a etiqueta correta tem um valor de tipo int64.

```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

print_ndr_basic_info(x_train)
print_ndr_basic_info(x_test)
print_ndr_basic_info(y_train)
print_ndr_basic_info(y_test)
```

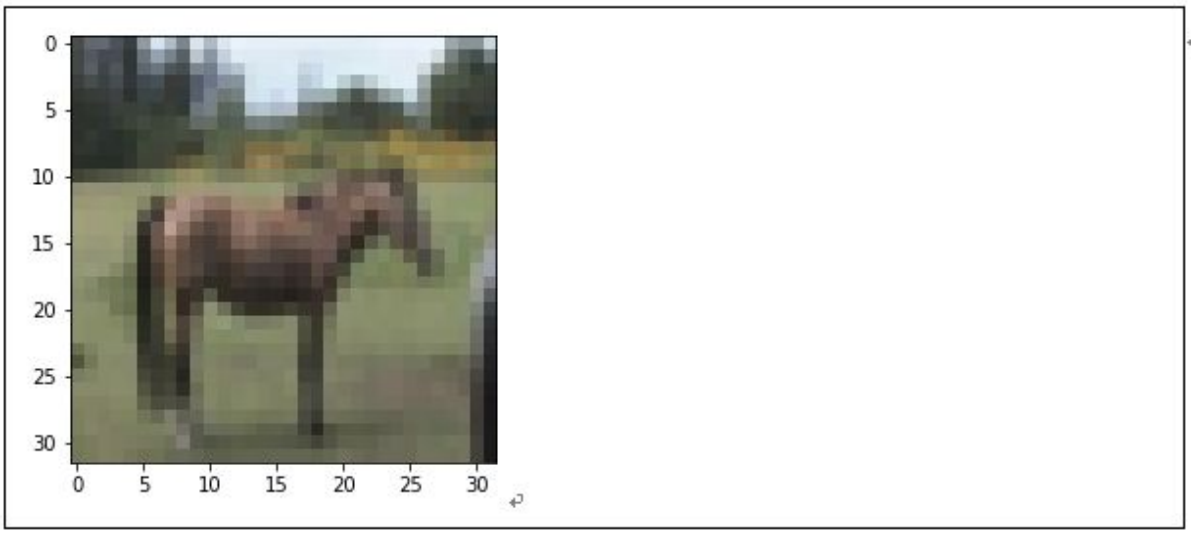
Output:

```
<class 'numpy.ndarray'>
(50000, 32, 32, 3)
1
153600000
uint8
4
<class 'numpy.ndarray'>
(10000, 32, 32, 3)
1
30720000
uint8
4
<class 'numpy.ndarray'>
(50000, 1)
1
50000
uint8
2
<class 'numpy.ndarray'>
(10000, 1)
8
10000
int64
2
```

Como os dados são dados de imagem, eles podem ser facilmente exibidos com o seguinte código usando matplotlib, a biblioteca que discutiremos na próxima seção. Tente exibir o 10.000º registro da imagem nos dados de teste. Cavalo" é agora exibido.

```
from matplotlib import pyplot as plt
plt.imshow(x_test[9999])
```

Output:



O corte em Numpy.ndarray pode ser feito como uma lista. Assim, a saída do eixo 0 (ou seja, as linhas) em ordem inversa é feita como em z[::-1]. Por exemplo, fazer isso para um array bidimensional 3 x 3 resultaria no seguinte

```
z = np.array([[1,2,3],[4,5,6],[7,8,9]])
z[::-1]
```

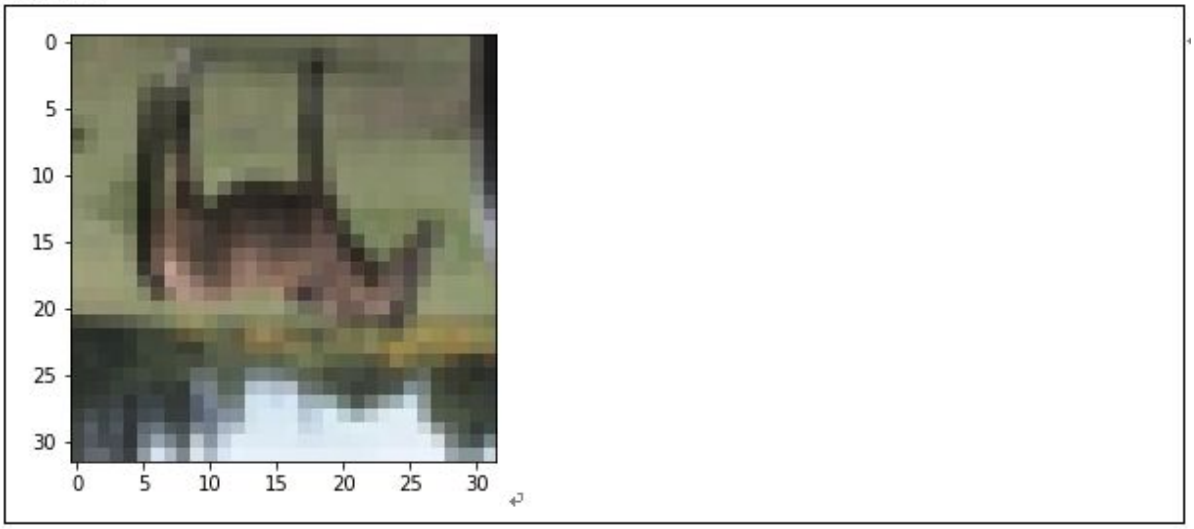
Output:

```
array([[7, 8, 9],
       [4, 5, 6],
       [1, 2, 3]])
```

Se você vir o resultado de processar isto contra a imagem do cavalo que acabei de lhe mostrar, você verá um cavalo de cabeça para baixo na imagem. É intuitivo e fácil de entender.

```
plt.imshow(x_test[9999][::-1])
```

Output:



Em seguida, o eixo de saída 1 (isto é, a coluna) em ordem inversa é feito da seguinte forma:

```
z = np.array([[1,2,3],[4,5,6],[7,8,9]])
z[:, ::-1]
```

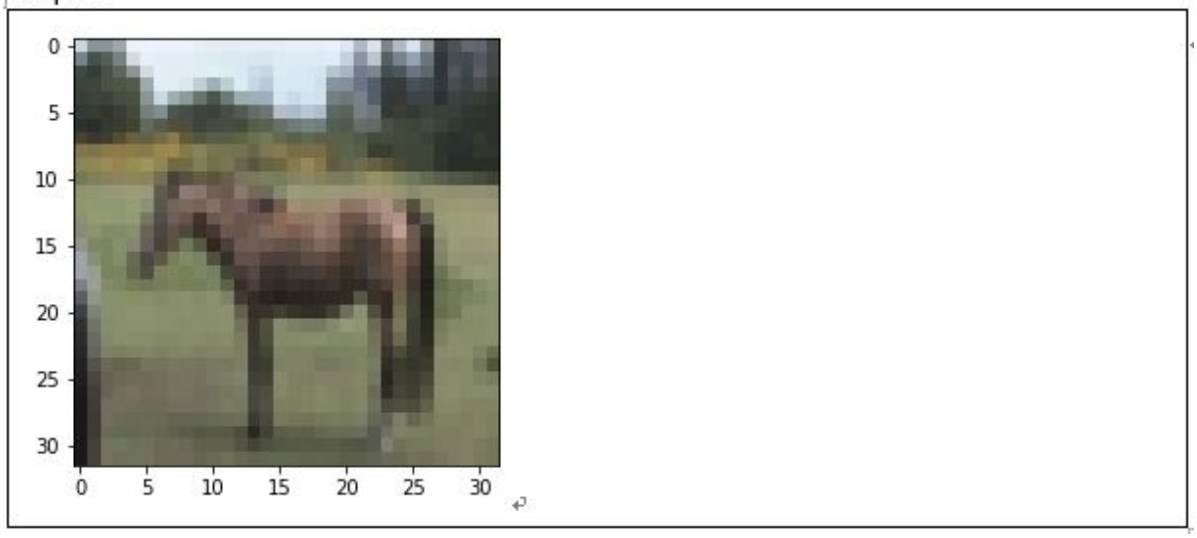
Output:

```
array([[3, 2, 1],
       [6, 5, 4],
       [9, 8, 7]])
```

Se você processar e exibir isto contra a imagem do cavalo de antes, você verá um cavalo cuja frente e costas estão invertidas desta vez. Mais uma vez, o resultado é muito fácil de entender.

```
plt.imshow(x_test[9999][:, ::-1])
```

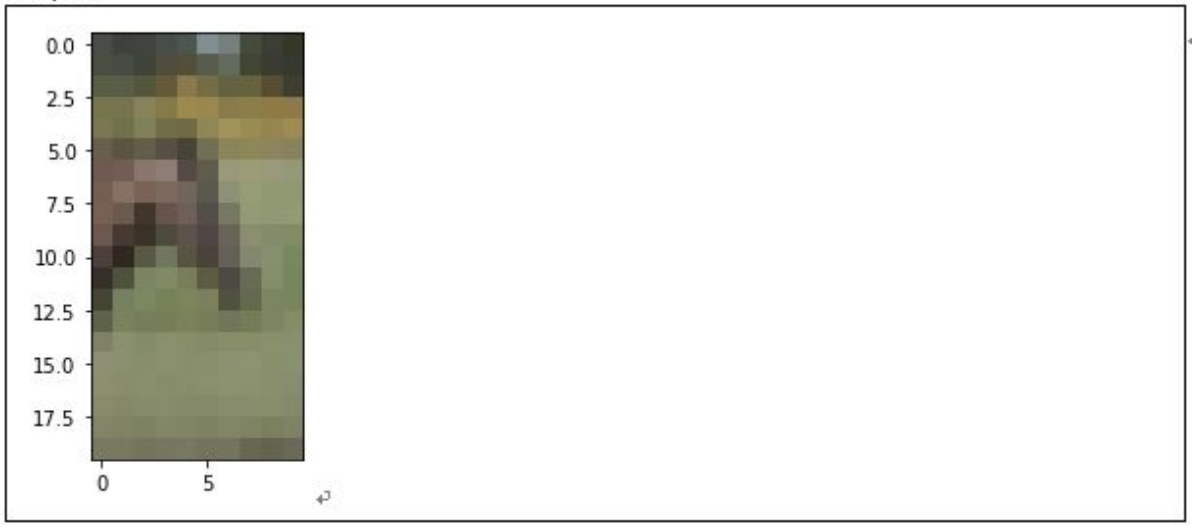
Output:



O código para cortar a cabeça do cavalo é mostrado abaixo. Cortando tanto linhas como colunas (mais precisamente, tanto o eixo 0 como o eixo 1), podemos facilmente conseguir o processo de cortar uma parte da área da imagem.

```
plt.imshow(x_test[9999][5:25, 20:30])
```

Output:



Em seguida, eu gostaria de olhar para a compressão: assim como o Slicing para tipos de listas Python, o Numpy permite que você gere um array especificando um intervalo (quantos a cada outro). Por exemplo, o exemplo seguinte cria um novo array pulando as linhas e colunas do array original uma a uma. A forma é reduzida de 5 x 5 para 3 x 3.

```
z =
np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20],[21,22,
23,24,25]])
print(z)
print(z.shape)
print(z[::2, ::2])
print(z[::2, ::2].shape)
```

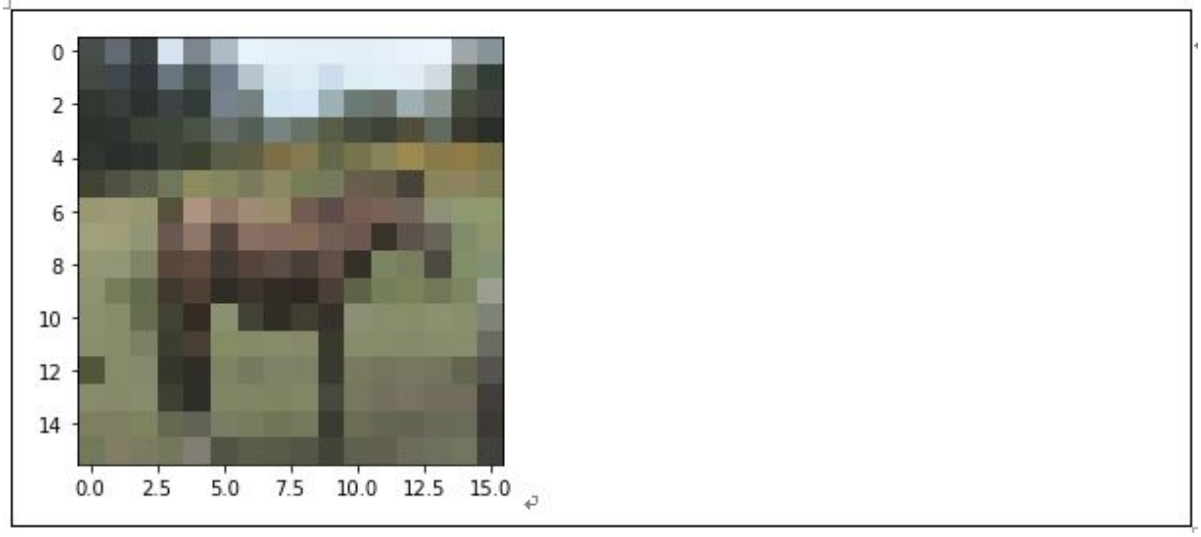
Output

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
(5, 5)
[[ 1  3  5]
 [11 13 15]
 [21 23 25]]
(3, 3)
```

A aplicação deste corte de um salto a uma imagem permite-lhe comprimir a imagem (e metade da quantidade de informação). (É também metade da quantidade de informação - se você verificar a escala no eixo x e no eixo y, é definitivamente 16 x 16 pixels.

```
plt.imshow(x_test[9999][::2, ::2])
```

Output:



Aqui está a saída da imagem do cavalo ndarray, usando todas as funções básicas de agregação do numpy.

```
photo = x_test[9999]
def print_labeled_val(label, val):
 print(f"{label}:{val}")

print_labeled_val("Sum", np.sum(photo))
print_labeled_val("product of each element of arrays",np.prod(photo))
print_labeled_val("mean", np.mean(photo))
print_labeled_val("standard deviation", np.std(photo))
print_labeled_val("variance", np.var(photo))
print_labeled_val("minimum value", np.min(photo))
print_labeled_val("maximum value",np.max(photo))
print_labeled_val("the minimum element's index", np.argmin(photo))
print_labeled_val("the maximum element's index", np.argmax(photo))
```

Output:

```
Sum:331662
product of each element of arrays:0
mean:107.962890625
standard deviation:46.36682798593862
variance:2149.8827374776206
minimum value:25
maximum value:252
the minimum element's index:1745
the maximum element's index:38
```

Ao escrever ndarray[expressão condicional], é possível aplicar a expressão a todos os elementos do array e criar um novo array contendo o número de elementos booleanos do array original.

```
z = np.array([1,2,3,4,5])
z <= 3
```

Output:

```
array([ True,  True, False, False, False])
```

Usando este mecanismo, é possível extrair apenas os elementos que satisfaçam os critérios a seguir

```
z[z<=3]
```

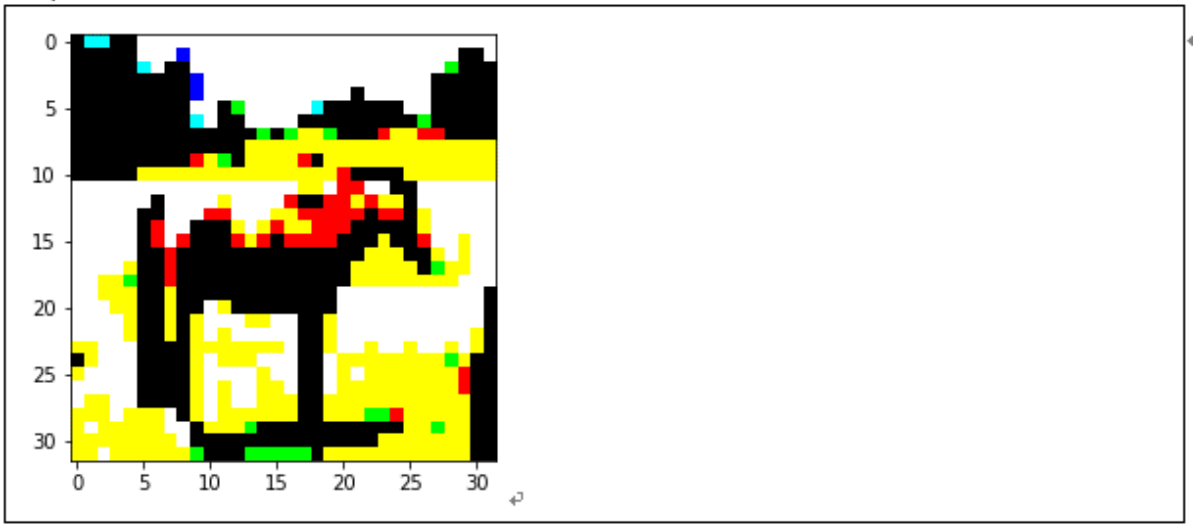
Output:

```
array([1, 2, 3])
```

Usando o mecanismo de expressão condicional ndarray no processamento de imagens, é possível filtrar as cores RGB para uma aparência ilustrativa, como mostrado no exemplo seguinte.

```
# Convert to 255 if each RGB value is greater than 100, or to 0 if it is
less than or equal to 100.
photo = x_test[9999]
photo_masked = np.where(photo > 100, 255, 0)
plt.imshow(photo_masked)
```

Output:



Transmissão do Numpy.ndarray

Now vamos dar uma olhada em uma das características mais importantes do Numpy: as transmissões. Broadcast é uma funcionalidade que expande automaticamente as linhas e colunas quando não há elementos suficientes. Um exemplo óbvio é a quadratura ndarray e escalar. É possível realizar cálculos sobre todos os elementos da ndarray, como mostrado abaixo.

```
a = np.array([1,2,3,4,5])
print(a + 2)
print(a - 2)
print(a * 2)
print(a / 2)
```

Output:

```
[3 4 5 6 7]
[-1  0  1  2  3]
[ 2  4  6  8 10]
[0.5 1.  1.5 2.  2.5]
```

A transmissão também pode ser usada para calcular matrizes multidimensionais de diferentes formas, como mostrado abaixo. Na seguinte operação quadrática de a_array e b_array, você pode ver que b_array é automaticamente expandido e computado como np.array([[10,10,10,10,10],[20,20,20]]).

```
a_array = np.array([[1,3,5],[7,9,11]])
b_array = np.array([[10],[20]])
print(a_array.shape)
print(b_array.shape)
```

Output:

```
(2, 3)
(2, 1)
```

```
print(a_array + b_array)
print(a_array - b_array)
print(a_array / b_array)
print(a_array * b_array)
```

Output:

```
[[11 13 15]
 [27 29 31]]
[[ -9  -7  -5]
 [-13 -11  -9]]
[[0.1  0.3  0.5 ]
 [0.35 0.45 0.55]]
[[ 10  30  50]
 [140 180 220]]
```

Cálculo do produto interno

One dos cálculos que são feitos frequentemente usando o Numpy é o cálculo do produto interno. O produto interno é representado pela seguinte equação, que, quando expandido, simplesmente multiplica os elementos do vetor w e do vetor x por sua vez e os soma, portanto não é tão difícil de calcular, mas é muito útil para calcular distâncias ambíguas, como para determinar a similaridade entre imagens e palavras.

$$w \cdot x = \sum_{i=0}^{n} w_i x_i$$

Existem três maneiras de escrever o código fonte interno do produto Numpy, como mostrado abaixo, e os resultados são os mesmos não importa qual método você use, mas se você estiver rodando em Python 3.6 ou superior, eu recomendo usar @ porque é o mais legível e fácil de escrever.(A partir de 9/9/2020, o Python do Colab estava na versão 3.6.9).

```
w = np.array([1,2,3])
x = np.array([4,5,6]).T
print(np.dot(w,x))
print(w.dot(x))
print(w@x)
```

Output:

```
32
32
32
```

A propósito, você pode obter a versão do ambiente Jupyter Notebook Python que você está executando usando o comando mágico.

```
!python --version
```

Output:

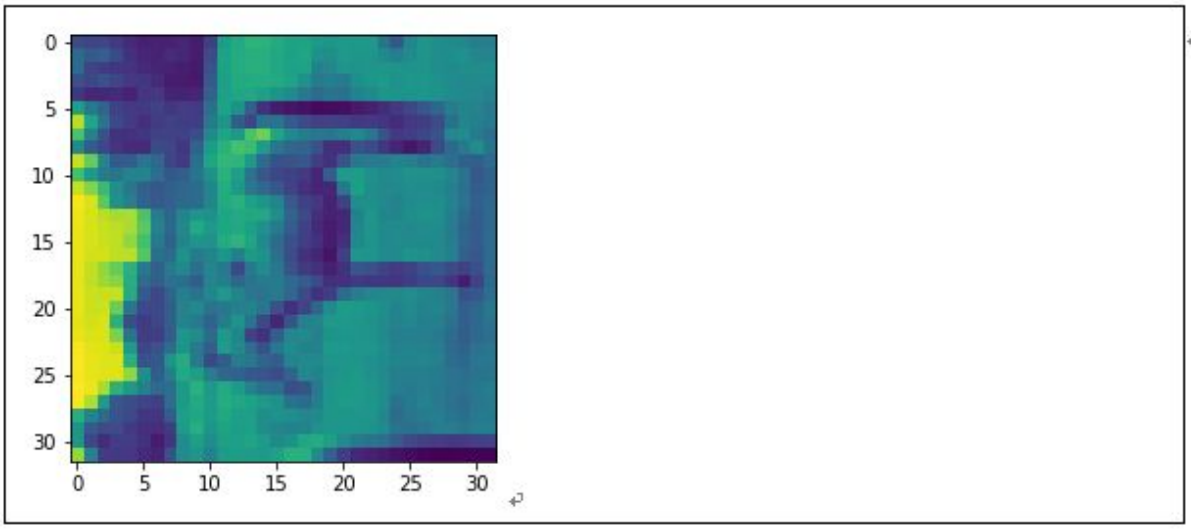
```
Python 3.6.9
```

Método T no cálculo anterior do produto interno, onde T significa transpor, o que significa substituir as linhas e colunas. No cálculo do produto interno, a forma do objeto é muito importante: podemos calcular o produto interno das matrizes L x N e N x M, mas não das matrizes N x L e N x M. No exemplo anterior, o produto é calculado automaticamente como o produto de uma forma 1 x 3 w e 3 x 1 x, sem utilizar o método T, pois o produto foi calculado entre duas ndarrays unidimensionais.

Se você aplicar esta matriz de transposição à imagem do cavalo mencionada acima, as linhas e colunas são trocadas na saída como mostrado abaixo, o que deve ajudá-lo a ter uma idéia do que esperar.

```
plt.imshow(photo[:,:,0].T)
```

Output:



O Numpy tem muitas características que ainda não abordamos, mas recomendamos os exercícios Numpy no Github, que o ajudarão a melhorar suas habilidades Numpy trabalhando com boas perguntas.

11. Matplotlib

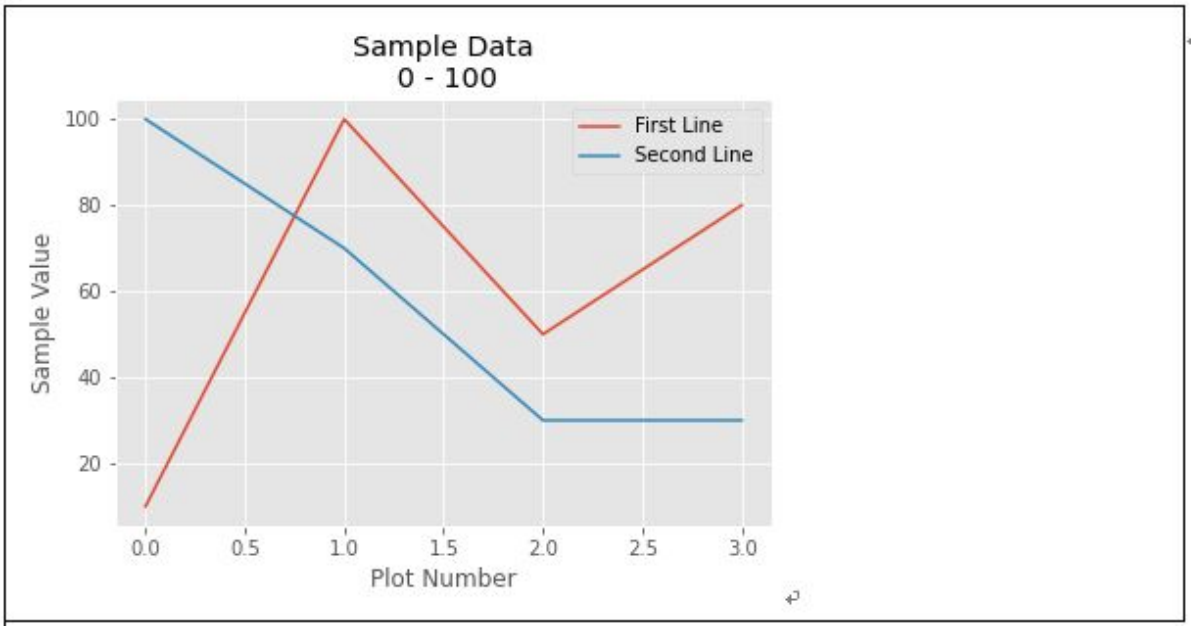
Matplotlib é uma das ferramentas de visualização de dados mais populares para Python, e enquanto Pandas também faz uso do Matplotlib internamente para facilitar a grafia e exibição, esta é apenas uma pequena parte da funcionalidade do Matplotlib. O Matplotlib inclui muitos módulos, mas este livro não os pode cobrir todos, por isso se precisar deles, por favor verifique o site oficial (https://matplotlib.org/py-modindex.html).

Legenda, título, etiquetas do eixo X e do eixo Y

First, vamos dar uma olhada no módulo pyplot mais usado, plot (comumente importado como matplotlib.pyplot como plt e usado na forma abreviada plt). Execute o seguinte código para plotar dois dados plt.plot cujo eixo x é o índice (um número inteiro entre 0 e 3) e cujo eixo y é [10, 100, 50, 80], [100, 70, 30, 30], respectivamente. Você pode chamar o método de legenda para exibir a legenda, passando o argumento da palavra-chave label. Pode passar a etiqueta como argumento de palavra-chave e chamar o método de legenda para exibir a legenda com a etiqueta especificada. A etiqueta do eixo x e a etiqueta do eixo y podem ser especificadas nos métodos plt.xlabel e plt.ylabel, respectivamente. Todo o título do gráfico pode lidar com códigos de alimentação de linha, então você não precisa usar plt.subplot para exibir um título de duas linhas.

```
import matplotlib.pyplot as plt
x = [0,1,2,3]
y = [10, 100, 50, 80]
x2 = [0,1,2,3]
y2 = [100, 70, 30, 30]

plt.plot(x, y, label='First Line')
plt.plot(x2,y2, label='Second Line')

plt.xlabel('Plot Number')
plt.ylabel('Sample Value')
plt.title('Sample Data\n 0 - 100')
plt.legend()
plt.show()
```

Output:

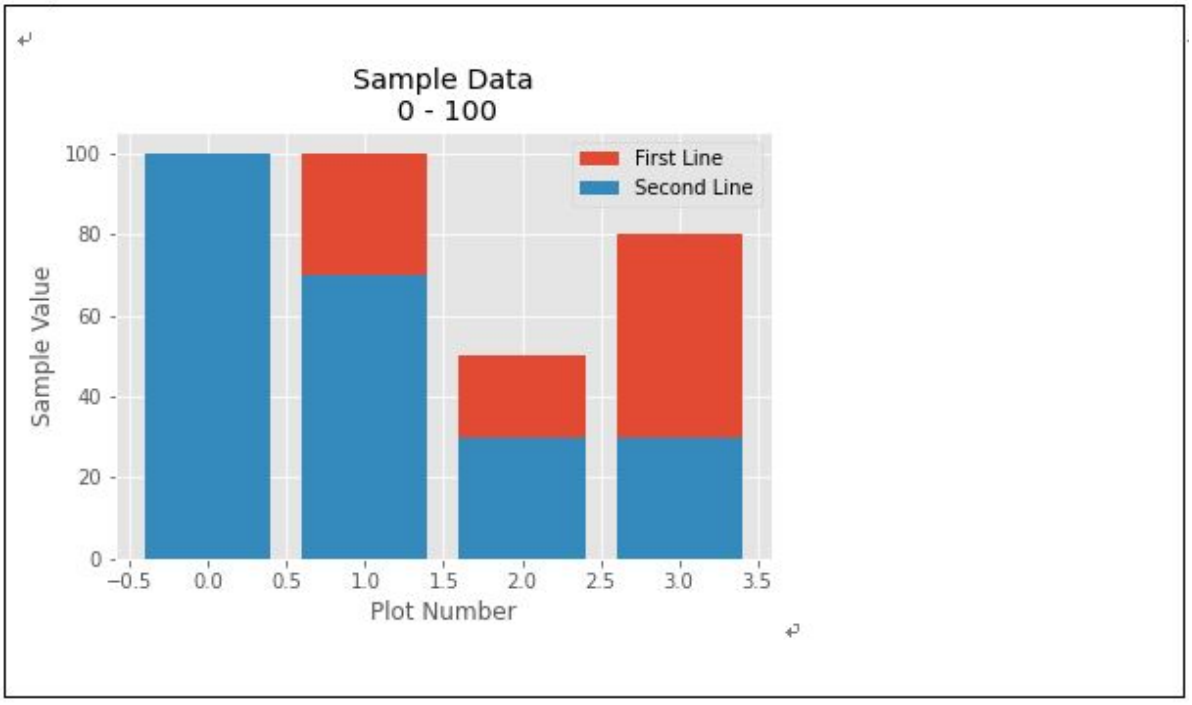


Gráficos de Barras

Agora vamos ver os dados como um gráfico de barras. O gráfico de barras pode ser exibido usando o método pyplot.bar.

```
import matplotlib.pyplot as plt
x = [0,1,2,3]
y = [10, 100, 50, 80]
x2 = [0,1,2,3]
y2 = [100, 70, 30, 30]

plt.bar(x, y, label='First Line')
plt.bar(x2,y2, label='Second Line')

plt.xlabel('Plot Number')
plt.ylabel('Sample Value')
plt.title('Sample Data\n 0 - 100')
plt.legend()
plt.show()
```

Output:

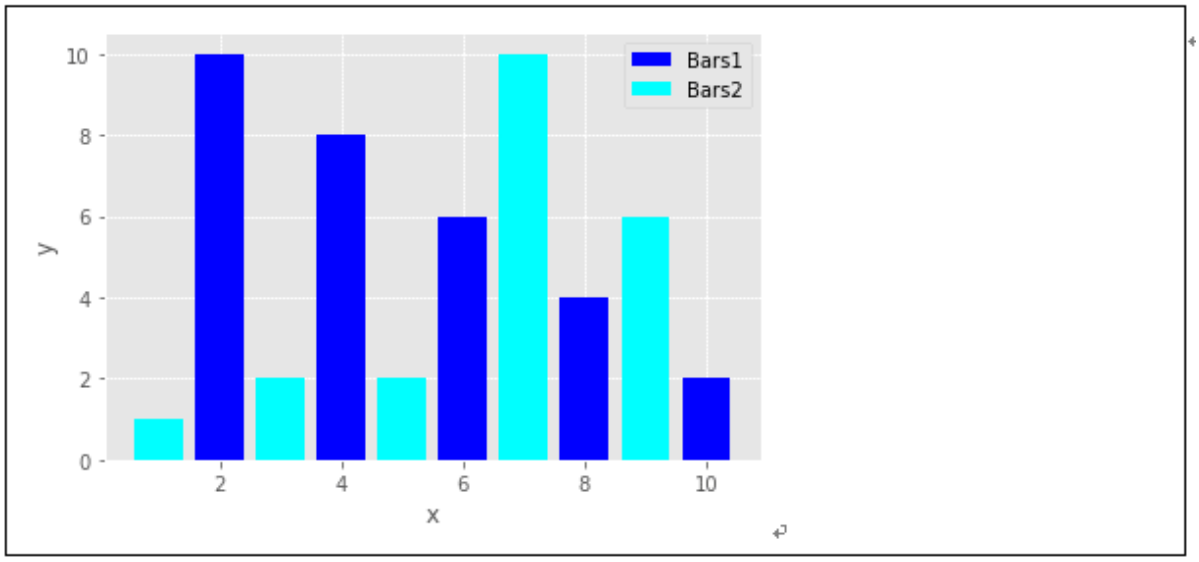


Especificação de cores da série

Você pode usar a cor como argumento para o método de barra para exibi-la em qualquer cor que você quiser, ou você pode usar hexadecimal para especificar o RGB.

```
import matplotlib.pyplot as plt
import random
x = [i for i in range(2, 11, 2)]
y = [i for i in range(10, 1, -2)]

x2 = [i for i in range(1, 10, 2)]
y2 = [random.randint(0, 10) for i in range(11, 1, -2)]

plt.bar(x, y, label='Bars1', color='blue')
plt.bar(x2, y2, label='Bars2', color='cyan')

plt.xlabel('x')
plt.ylabel('y')
plt.title('')
plt.legend()
plt.show()
```

Output:

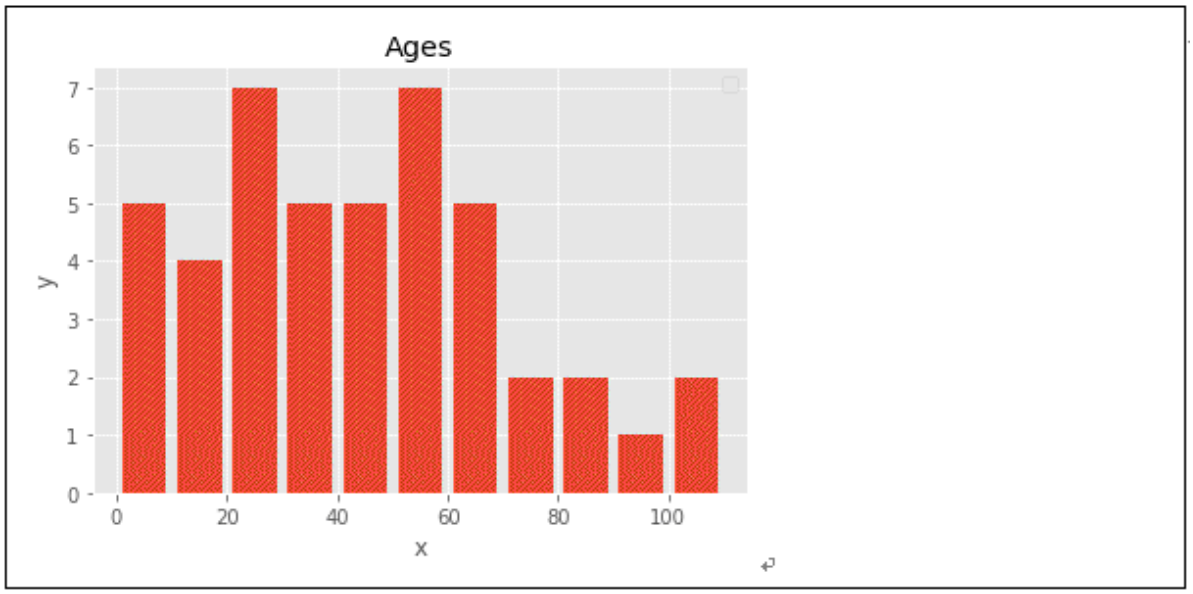


Histograma

Em seguida, vamos dar uma olhada em um gráfico de histograma (distribuição de freqüência), que pode ser facilmente desenhado usando o método plt.hist. O primeiro argumento são os dados e o segundo é o intervalo (o intervalo de frequências a ser agregado). No exemplo a seguir, o intervalo é especificado dividindo por 10 de 0 a 110 usando a compreensão de lista. Rwidth especifica a largura do intervalo de cada barra, mas eu a defino como 0,8 para facilitar a visualização (para evitar que as barras se colem umas às outras).

```
import matplotlib.pyplot as plt

population_ages = [2, 7, 75, 55, 55, 46, 60, 65, 68, 59, 45, 48, 58, 55, 8,
10, 100, 95, 40, 39, 34, 28, 2, 29, 34, 3, 46, 50, 20, 30, 80, 60, 68, 58,
23,12, 102,79,80, 39, 20, 21, 20, 18, 19]
bins = [i for i in range(0, 120, 10)]
plt.hist(population_ages, bins, histtype='bar', rwidth=0.8)

plt.xlabel('x')
plt.ylabel('y')
plt.title('Ages')
plt.legend()
plt.show()
```

Output:

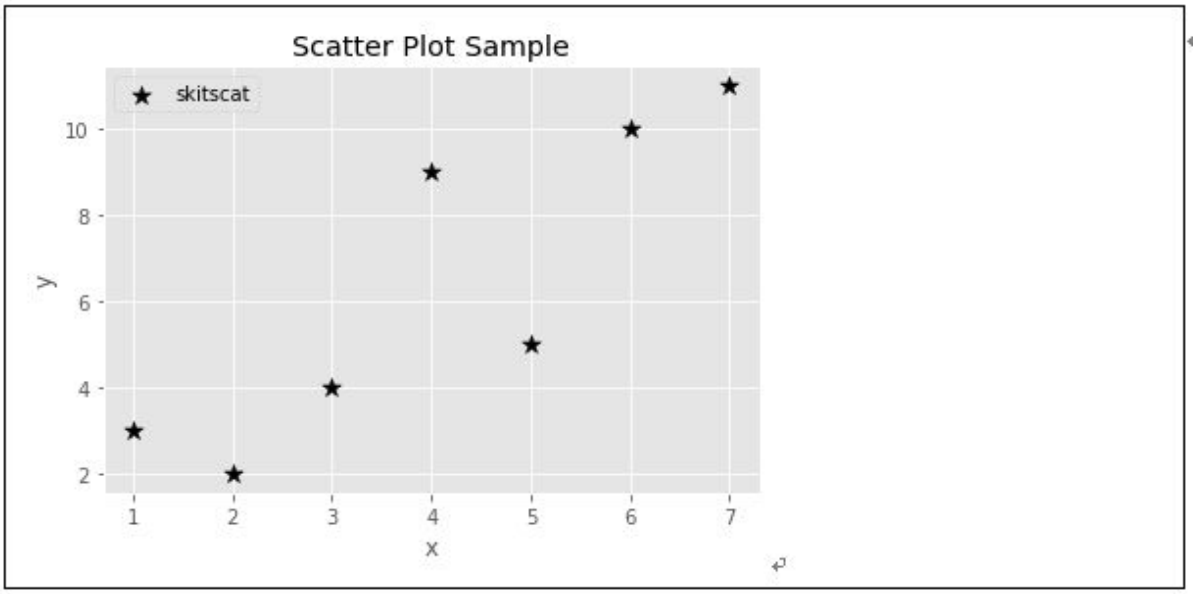


Lote de Dispersão

Scatter plots são úteis quando você quer ver a correlação entre duas ou mais variáveis. plt.scatter pode ser usado para desenhar um gráfico de dispersão. Existem muitos outros marcadores que lhe permitem especificar uma estrela, uma trave e um círculo com "*", "x" e "o" respectivamente. Por favor, consulte o site oficial (https://matplotlib.org/3.2.1/api/markers_api.html). Se você quiser aumentar o tamanho do marcador, especifique-o com um s. A amostra seguinte mostra um gráfico de dispersão com grandes estrelas pretas. A amostra seguinte mostra um diagrama de dispersão com grandes estrelas pretas.

```
x = [1,2,3,4,5,6,7]
y = [3,2,4,9,5,10,11]
plt.scatter(x, y, label='skitscat', color='black', marker='*', s=100)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Scatter Plot Sample')
plt.legend()
plt.show()
```

Output:

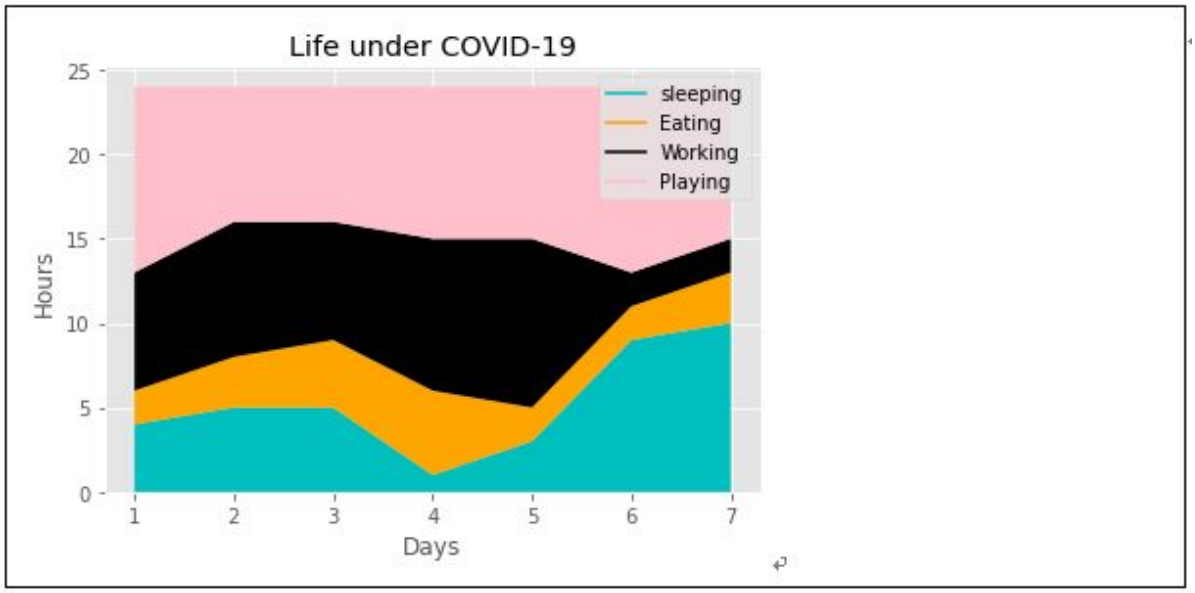


Pilha de Plot

Stack são usados para mostrar a percentagem de dados para cada barra em um gráfico de barras. A exibição da legenda do Stack Plot requer alguma técnica, por isso, consulte o seguinte código. Na era da nova pneumonia, quando nos tornamos uma população nacionalmente reclusa, é muito importante regular o ritmo de nossas vidas, então vamos usar plt.stackplot para visualizar a porcentagem de tempo que dormimos, comemos, trabalhamos e brincamos em uma semana.

```
import matplotlib.pyplot as plt
days = [1,2,3,4,5,6,7]

sleeping = [4,5,5,1,3,9,10]
eating = [2,3,4,5,2,2,3]
working = [7,8,7,9,10,2,2]
playing = [11,8,8,9,9,11,9]
plt.plot([],[],color='c', label='sleeping')
plt.plot([],[],color='orange',label='Eating')
plt.plot([],[],color='k',label='Working')
plt.plot([],[],color='pink',label='Playing')

plt.stackplot(days, sleeping, eating, working, playing, colors=['c',
'orange', 'k', 'pink'])
plt.xlabel('Days')
plt.ylabel('Hours')
plt.title('Life under COVID-19')
plt.legend()
plt.show()
```

Output:



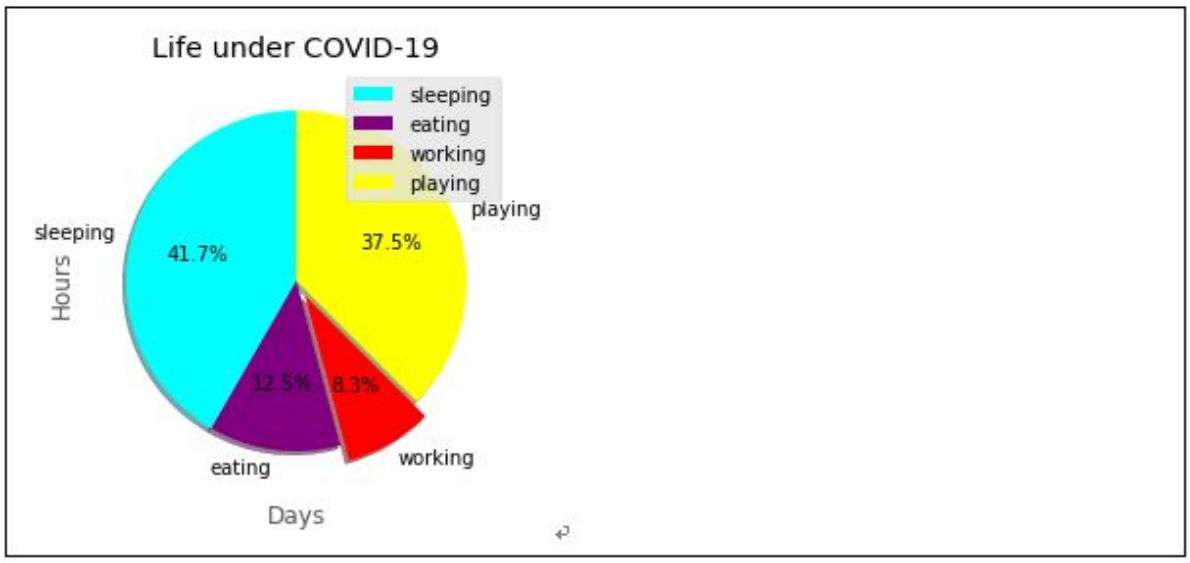
Gráficos de Tartes

I pensa que uma forma útil de mostrar a percentagem é através de gráficos de tartes (pie charts), e eu expressei a percentagem relacionada com o tempo de actividade que acabei de mencionar nos gráficos de tartes como se segue.

O startangle é usado para inclinar o gráfico da torta no ângulo em que foi iniciada, e a sombra é usada para criar uma sombra no gráfico. Destaquei também o terceiro elemento (=playing) em explodir para que ele se sobreponha um pouco; em autopct, estou mostrando-o como uma porcentagem.

```
import matplotlib.pyplot as plt
days = [1,2,3,4,5,6,7]

sleeping = [4,5,5,1,3,9,10]
eating = [2,3,4,5,2,2,3]
working = [7,8,7,9,10,2,2]
playing = [11,8,8,9,9,11,9]
slices = [10, 3, 2, 9]
activities = ['sleeping','eating','working','playing']
colors = ['cyan', 'purple', 'red', 'yellow']
plt.pie(slices, labels=activities, colors=colors, startangle=90,
shadow=True, explode=(0, 0, 0.1, 0), autopct="%1.1f%%")

plt.xlabel('Days')
plt.ylabel('Hours')
plt.title('Life under COVID-19')
plt.legend()
plt.show()
```

Output:



3 D Gráfico

There são muitos fenômenos no mundo que não podem ser julgados em duas dimensões. Por exemplo, não seria possível julgar o risco de infecção no Japão com base apenas no número de pessoas infectadas com um novo tipo de coronavírus no país. Outras variáveis como o número de testes PCR e as taxas de mortalidade, assim como avaliações qualitativas, também devem ser incluídas na avaliação do risco de infecção.

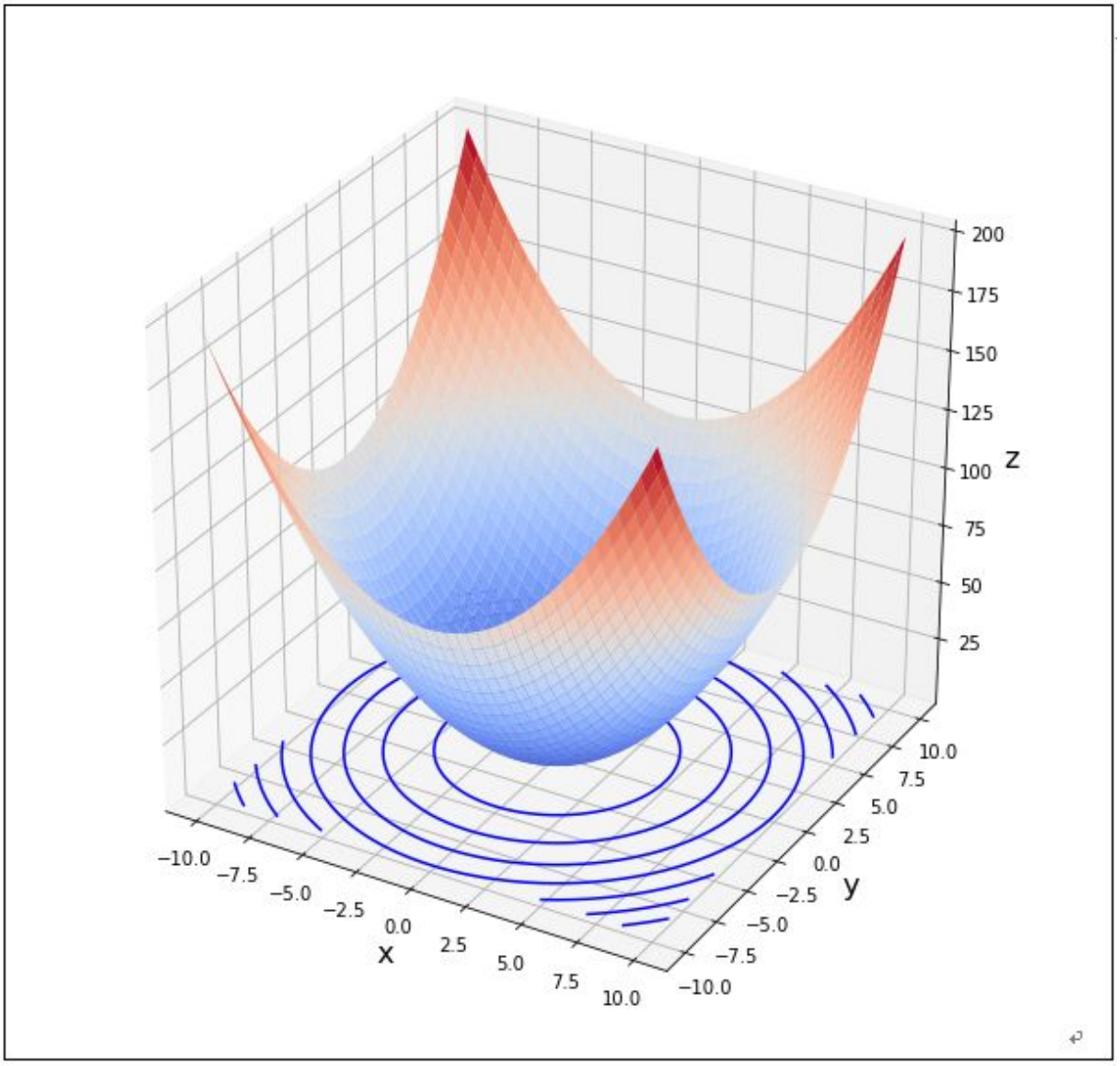
Also, em aprendizagem profunda supervisionada, o objetivo é encontrar pesos que minimizem a perda (o erro entre a resposta correta e a saída do modelo), mas o que pode ser um valor mínimo em uma dimensão pode convergir (armadilha) para um ponto de sela que é um valor máximo em outra dimensão.

Visualizing subir as dimensões é muito importante para capturar a essência das coisas, por isso vou explicar como visualizá-lo em três dimensões usando Axes3D do mpl_toolkits.mplot3D.

Tente executar o seguinte código o mais rápido possível.

```
# 3D Plot
import numpy as np
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot_3d_func(X, Y, Z):
 """ Three-dimensional plot of a function Z with inputs of two variables X
and Y

   Args:
     X: ndarray
     Y: ndarray
     Z: ndarray
 """
 fig = plt.figure(figsize = (10, 10))
 ax = fig.add_subplot(1, 1, 1, projection="3d")

 ax.set_xlabel("x", size = 16)
 ax.set_ylabel("y", size = 16)
 ax.set_zlabel("z", size = 16)
 ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
#  ax.plot_surface(X, Y, Z, cmap=cm.summer)

 ax.contour(X, Y, Z, colors = "blue", offset = -1)
#  ax.scatter(X, Y, Z)

 plt.show()

def func_z1(X, Y):
 return X**2 + Y**2

# Generates 256 elements equally spaced from -10 to 10
x = np.linspace(-10, 10, 256)
y = np.linspace(-10, 10, 256)

# Grid Generation
X, Y = np.meshgrid(x, y)

# Calling func_z1
Z = func_z1(X, Y)

# 3D Plot
plot_3d_func(X, Y, Z)
```

Ouput:



Agora você pode ver um gráfico de contorno com uma linha de contorno em forma de tigela na parte inferior, como mostrado na figura acima. Este é um gráfico que visualiza o seguinte

$$Z = X^2 + Y^2$$

Você pode desenhar uma superfície tridimensional por Axes3D.plot_surface e Axes3D.contour.
Você pode especificar o colormap com cmap, que é o argumento para plot_surface. O cm.summer comentado também tem cores

bonitas, portanto, por favor, experimente. Além disso, se você usar Axes3D.scatter em vez de contorno, você pode desenhar um diagrama de dispersão 3D. Neste exemplo, nós desenhamos uma função, mas os diagramas de dispersão 3D também são frequentemente usados para traçar dados reais.

Numpy.meshgrid é um método para a geração de redes. Você pode pensar em uma grade de malha como uma grade de todas as combinações de elementos no domínio dos eixos x e y. É difícil explicar em palavras, então vamos executar o seguinte código fonte.

```
x_sample = np.array([1,2,3])
y_sample = np.array([1,2,3])
X_sample, Y_sample = np.meshgrid(x_sample, y_sample)
print(X_sample)
print(Y_sample)
```

Output:

```
[[1 2 3]
 [1 2 3]
 [1 2 3]]
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

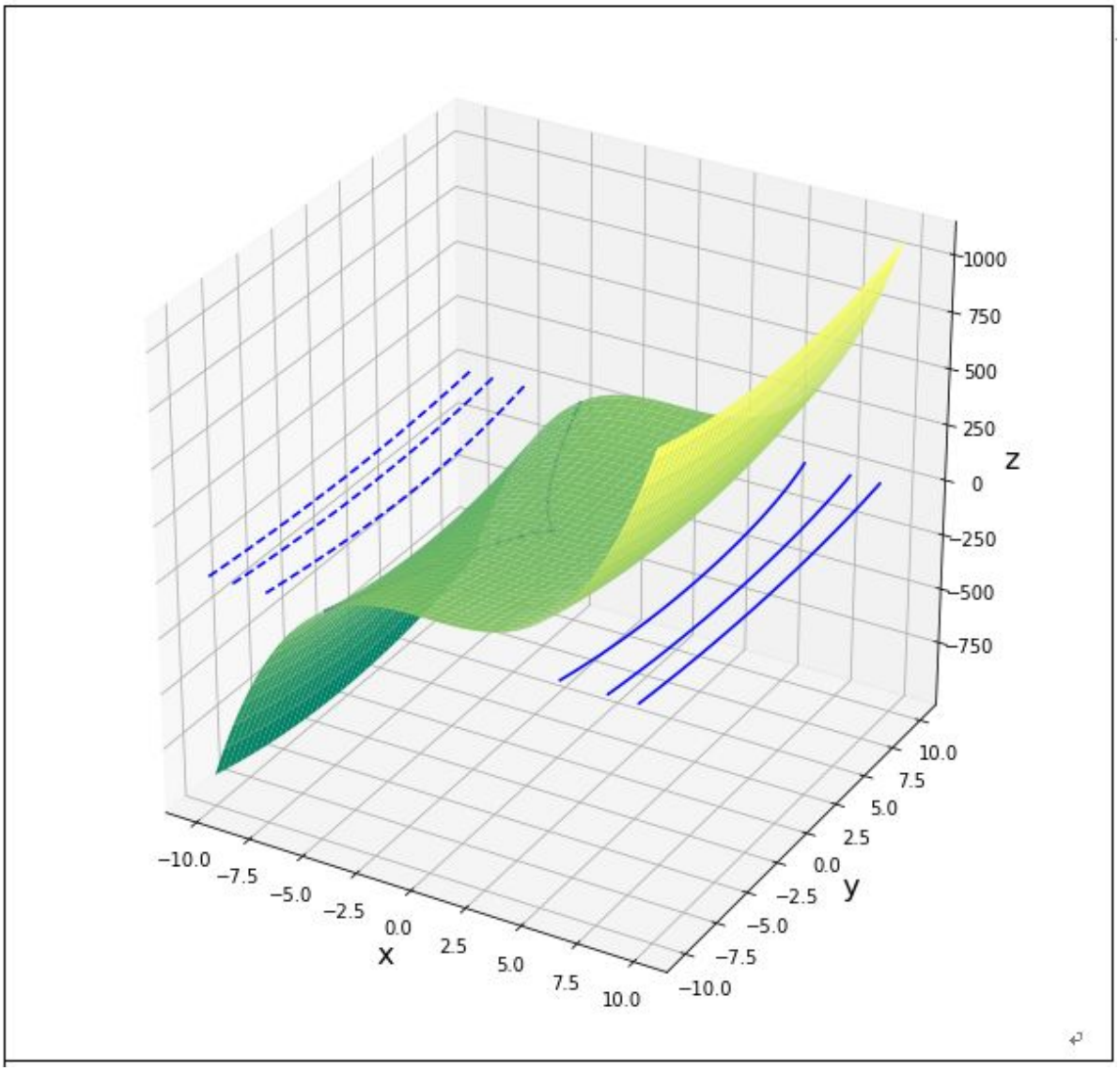
Acima, poderíamos facilmente obter os pontos de grade para todas as combinações de inteiros com x de 1 a 3 e y de 1 a 3. Isto é muito útil para visualizar a saída de uma função que toma x e y como seus argumentos. No gráfico 3D anterior, nós geramos 256 elementos nos eixos x e y, igualmente espaçados de -10 a 10, respectivamente, e geramos todas as combinações como uma sequência de grid também.

Então, vamos mostrar outra função, $Z = X^3 + Y^2$.

Se você comentar a linha cmap='summer' no plot_3d_func e executar o código, a saída é a seguinte.

```
def func_z2(X, Y):
 return X**3 + Y**2
Z = func_z2(X, Y)
plot_3d_func(X, Y, Z)
```

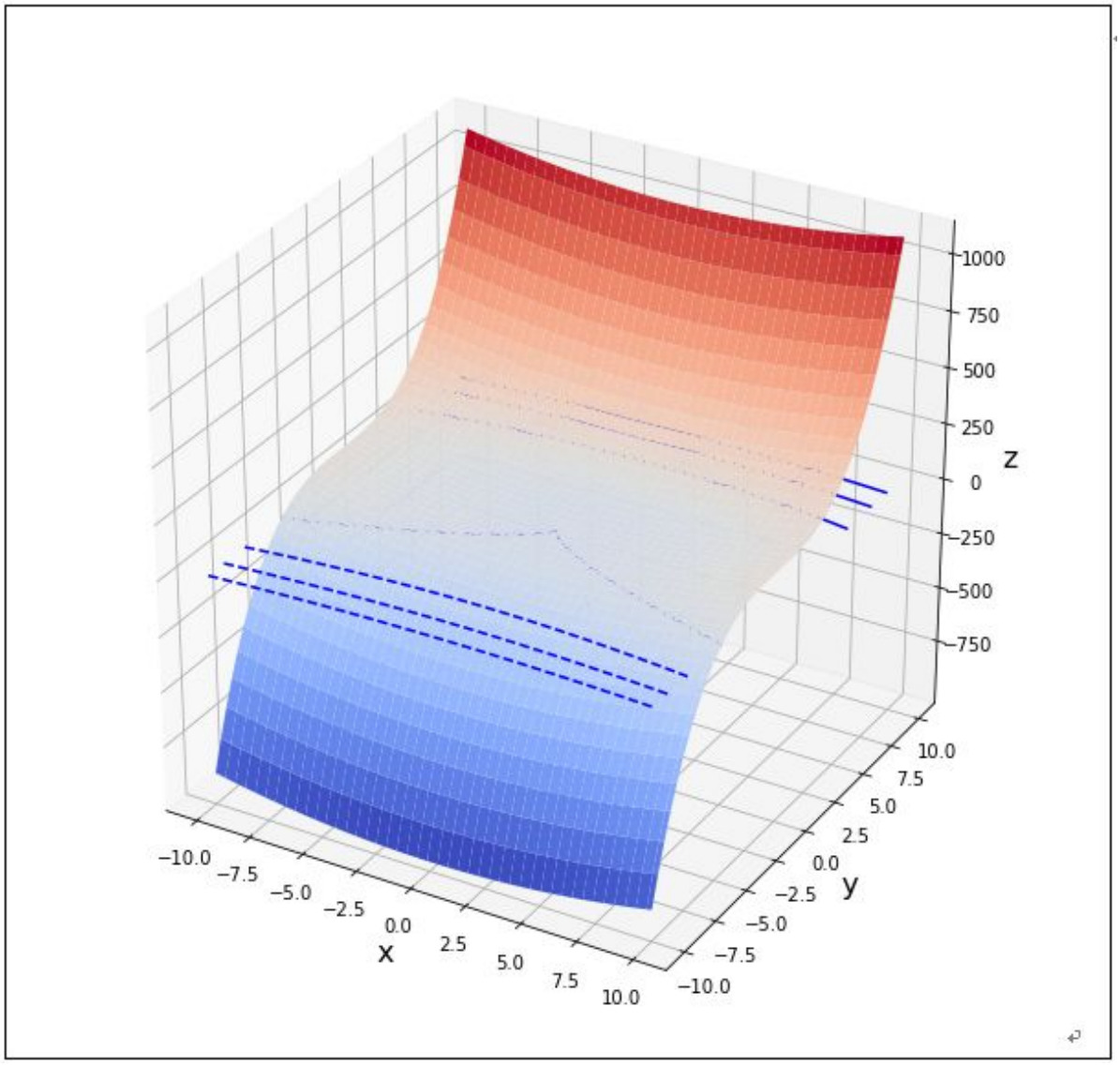
Output:



Em seguida, vamos exibir o gráfico de função, $Z = X^3 + Y^2$. Mais uma vez, eu defini o cmap para arrefecer.

```
def func_z3(X, Y):
 return X**2 + Y**3

Z = func_z3(X, Y)
plot_3d_func(X, Y, Z)
```

Output:



Please consulte o tutorial oficial do mplot3d - documentação Matplotlib 2.0.2 para aprender mais sobre técnicas de visualização em três dimensões, pois você pode aprender métodos mais diversos no tutorial oficial do mplot3d.

l espera que no final deste capítulo você tenha uma idéia de quão poderosos são os recursos de visualização do Matplotlib, e que você possa desenhar qualquer tipo de gráfico. Há algumas funcionalidades que não são abordadas neste livro, por isso verifique o tutorial oficial (https://matplotlib.org/tutorials/introductory/pyplot.html) se precisar dele.

12. Scikit-Learn

Scikit-Learn fornece uma variedade de algoritmos de aprendizagem de máquinas que você pode facilmente experimentar. Se você é um iniciante, há três pontos principais com os quais você pode ficar preso ao usar o Scikit-Learn.

1. pré-processamento de dados reais
2. seleção de algoritmos
3. selecção de hiperparâmetros

Como você foi capaz de usar Numpy/Pandas/Matplotlib no capítulo anterior para transformar e visualizar livremente a forma e dimensões dos seus dados, eu espero que você seja capaz de resolver a 1) tarefa de pré-processamento. Ao executar as amostras neste capítulo e experimentá-las, espero que você veja que pode facilmente construir um modelo de aprendizagem da máquina e resolver a tarefa de previsão com uma pequena quantidade de código.

2) Para a seleção do algoritmo, o site oficial da Scikit-Learn tem um diagrama muito útil chamado Escolhendo o estimador certo - scikit-learn 0.22.2 documentação, que pode ser encontrada aqui , consulte por favor. O ponto principal é que ao responder as perguntas sequencialmente, como a quantidade de dados disponíveis e se é um problema de classificação que aplica uma etiqueta não contínua ou um problema de regressão que prevê um número contínuo, é possível decidir qual algoritmo usar.

3) Em relação à selecção de hiperparâmetros (parâmetros passados como argumentos de execução em cada algoritmo, número de épocas de treino, etc.), é difícil mesmo para os profissionais da aprendizagem de máquinas para viver, mas existem métodos amplamente conhecidos, tais como pesquisa aleatória, optimização Bayesiana, pesquisa em grelha, etc., por isso, se estiver interessado nestes métodos, por favor contacte por favor procure por estas

palavras para chegar ao ponto. Desde que o algoritmo usado não esteja muito errado, é possível alcançar um certo grau de precisão prática executando simulações enquanto se olha para indicadores de avaliação como perda e precisão, por isso não entraremos em detalhes neste livro.

Neste capítulo, você aprenderá primeiro como classificar seus dados de amostra usando o software SVM/RandomForest/XGBoost, empacotado com o Scikit-learn. Finalmente, você irá treinar o modelo de aprendizagem da sua máquina em dados reais para realizar classificação (etiquetagem) e regressão (adivinhação).

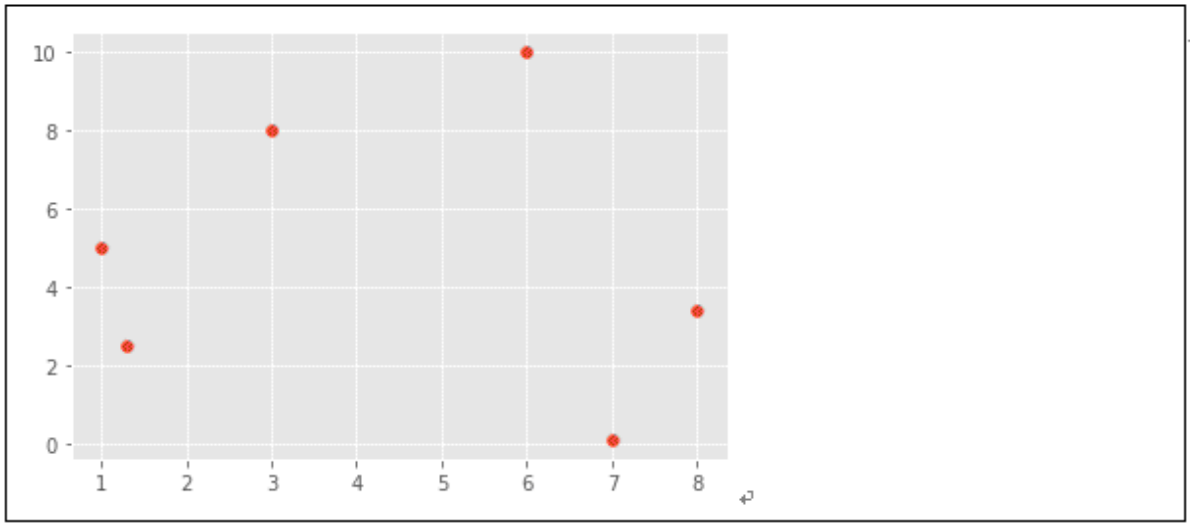
SVM:Máquina Vetorial de Suporte

SVM é um método de classificação apoiado pela teoria matemática avançada, e até o ILSVRC em 2012, o reconhecimento de imagens era frequentemente referido como SVM, e a ideia básica por detrás da SVM é muito clara: encontrando o plano limite (ou linha limite no caso de dados bidimensionais) que maximiza a distância entre os dados e a classificação. Para fazer isso, SVM toma a abordagem de mapear dados de alta dimensão e classificá-los linearmente no espaço de alta dimensão, e esse mapeamento é chamado de função kernel, e o mapeamento é chamado de truque kernel eu vou chamá-lo.

Não é fácil de entender apenas pela explicação, então vamos olhar para os dados da amostra e o código de trabalho.
Primeiro, vamos traçar os dados que queremos classificar em um gráfico de dispersão usando matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm
x = [1, 3, 8, 1.3, 7, 6]
y = [5, 8, 3.4, 2.5, 0.1, 10]
plt.scatter(x, y)
plt.show()
```

Output:



Como não podemos entrar no formato de lista no modelo Scikit-Learn, vamos convertê-lo para o formato Numpy.ndarray, que é um array bidimensional. Além disso, como a SVM é uma classificação supervisionada, precisamos preparar algumas etiquetas para professores. As etiquetas para professores correspondem a etiquetas de classificação, onde 0 é etiqueta 1 e 1 é etiqueta 2.

```
X = np.array([
  [1, 5],
  [3, 8],
  [8, 3.4],
  [1.3, 2.5],
  [7, 0.1],
  [6, 10]]
)

t = np.array([0, 0, 1, 0, 1, 1])
```

O seguinte código define então o classificador SVM. A função do kernel usa um mapa linear.

```
clf = svm.SVC(kernel='linear')
```

O seguinte código irá realmente fazer a aprendizagem da máquina (treinamento) para você.

```
clf.fit(X, t)
```

Output:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
   max_iter=-1, probability=False, random_state=None, shrinking=True,
   tol=0.001, verbose=False)
```

Agora que já treinamos o modelo, vamos tentar resolver a tarefa de previsão imediatamente.
Dado um novo ponto (0,38, 0,8), queremos tentar prever que etiqueta será esta. Do ponto de vista humano, parece que lhe será dado o rótulo 0 (ou seja, o mesmo rótulo de (1, 5)), mas consegui classificá-lo como esperado quando o previ com a SVM.

```
print(clf.predict(np.array([[0.38, 0.8]])))
```

Output:

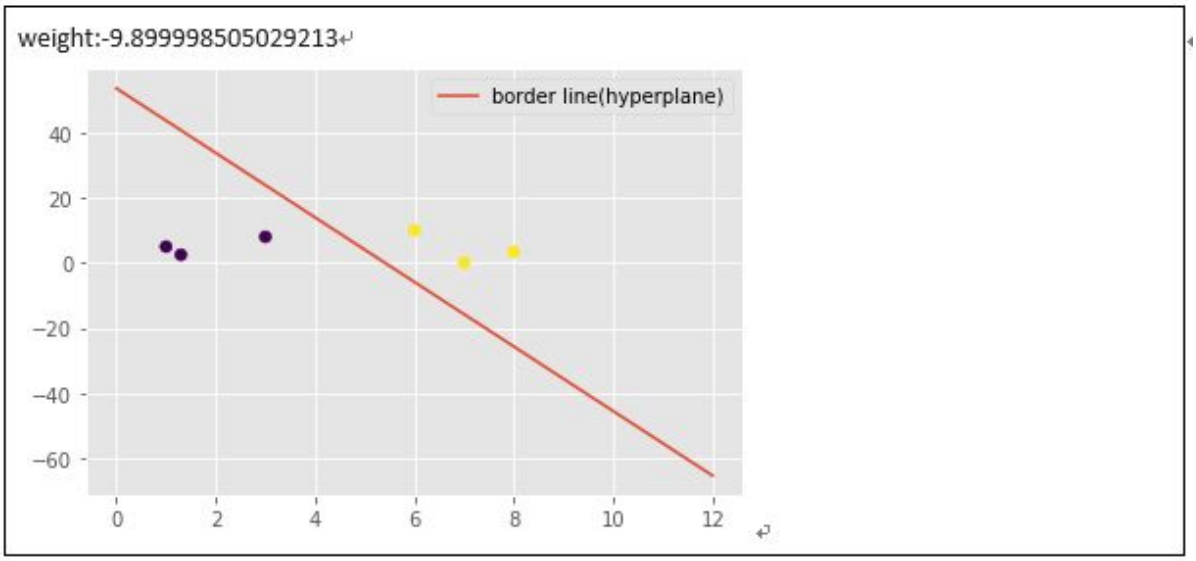
```
[0]
```

Vamos dar uma olhada no seguinte código para visualizar os limites que a SVM treinada usa para tomar decisões.

```
w = clf.coef_[0]

a = -w[0] / w[1]

print(f"weight:{a}")

xx = np.linspace(0,12)
yy = a * xx - clf.intercept_[0] / w[1]

h0 = plt.plot(xx, yy, '-', label="border line(hyperplane)")

plt.scatter(X[:, 0], X[:, 1], c = y)
plt.legend()
plt.show()
```

Output:

```
weight:-9.899998505029213
```



Vamos dar uma olhada em outras quatro incógnitas, executar a tarefa de previsão e calcular a porcentagem de respostas corretas (Precisão). A exatidão é calculada como a porcentagem de todos os eventos para os quais a previsão concorda com o resultado real.

```
X_test = np.array([[0.38, 0.8], [6, 8], [7,7], [8, 4], [2, 2]])
y_test = np.array([0, 1, 1, 1, 0])
prediction = clf.predict(X_test)
print(f"Accuracy:{100 * len(y_test[y_test == prediction])/len(X_test)}%")
```

Output:

```
Accuracy:100.0%
```

As previsões feitas no modelo SVM que criámos neste estudo foram 100% correctas para a não afinação. Acho que isto confirma que a função do kernel é susceptível de ser suficientemente precisa para dados de amostra separáveis linearmente (linearmente separáveis).

RandomForest

Next, vamos criar um modelo usando uma técnica de classificação chamada RandomForest, que aplica uma árvore de decisão (DecisionTree), e depois treiná-la e testá-la usando os dados de treinamento que acabamos de usar.

```
#   Training by RandomForest
from sklearn.ensemble import RandomForestClassifier as classifier
random_forest = classifier(random_state=0)
random_forest.fit(X, t)


#   Inference by RandomForest
X_test = np.array([[0.38, 0.8], [6, 8], [7,7], [8, 4], [2, 2]])
y_test = np.array([0, 1, 1, 1, 0])
prediction = random_forest.predict(X_test)
print(f"Accuracy:{100 * len(y_test[y_test == prediction])/len(y_test)}%")
```

Output:

```
Accuracy:100.0%
```

Even no modelo RandomForest, a percentagem de respostas correctas para os dados da amostra foi de 100% para não afinar. O método de aprendizagem supervisionado pode, em última análise, ser abstraído para a tarefa de produzir um padrão contínuo ou descontínuo a partir dos dados de entrada. Assim, a idéia básica das árvores de decisão é que se verificarmos quais características nos dados de entrada tomam quais valores por sua vez, e então criarmos condições de ramificação (caminhos de ramificação) como se as declarações na programação, podemos eventualmente restringir o refinamento condicional a um único output, que pode ser usado para uma tarefa de previsão. O RandomForest seleciona aleatoriamente essas características para criar árvores de decisão múltiplas, e então decide o output final do modelo por maioria de votos com base nos resultados de previsão dessas árvores de decisão.

Learning com vários modelos desta forma é chamado de aprendizado em conjunto. Também, como o RandomForest, o

método de treinamento em paralelo usando múltiplos modelos com alguns dados do todo é chamado de ensacamento.

XGBOOST(Classificação)

On por outro lado, o método de extrair iterativamente alguns dados e treiná-los sequencialmente é chamado de boosting. No método de boosting, um modelo é criado primeiro e treinado, depois os parâmetros são ajustados para priorizar as respostas corretas aos dados errados, depois os dados errados são ponderados contra os dados errados no modelo anterior e o treinamento prossegue, e finalmente o output é gerado como um modelo único. Este reforço é aplicado a um método chamado XgBoost, que é frequentemente visto no topo do concurso internacional de análise de dados Kaggle. Como o boosting é sequencial, o tempo de aprendizagem tende a ser mais preciso do que o ensacamento, que pode ser aprendido em paralelo, como o RandomForest, ao invés de tomar mais tempo computacional. No entanto, o XgBoost não é conhecido por ser muito preciso para problemas típicos de separação linear, com uma taxa correcta de 40% para os seguintes.

```
#    Training with the XgBoost Model
import xgboost as xgb
xgb_model = xgb.XGBClassifier(random_state=0)
xgb_model.fit(X, t)

#    Inference by XgBoost Model
X_test = np.array([[0.38, 0.8], [6, 8], [7,7], [8, 4], [2, 2]])
y_test = np.array([0, 1, 1, 1, 0])
prediction = xgb_model.predict(X_test)
print(f"Accuracy:{100 * len(y_test[y_test == prediction])/len(y_test)}%")
```

Output

```
Accuracy:40.0%
```

XGBOOST(Regressão)

So de longe, realizamos uma tarefa de classificação (previsão de etiquetas descontínuas) usando um modelo de aprendizagem de máquina que fizemos com SVM/RandomForest/XgBoost usando dados de amostra. A seguir, vamos executar um problema de regressão (previsão de valores contínuos) na aprendizagem de máquinas usando o XgBoostRegressor.

The são preparadas amostras de dados de treinamento para 10 pessoas com os seguintes dados de altura e peso. A etiqueta de resposta correcta é IMC. Se conseguirmos derivar a função IMC apenas dos dados, teremos treinado um modelo para prever o IMC a partir da altura e do peso usando a aprendizagem da máquina.

```
X = np.array([
 [160, 45],
 [180, 70],
 [190, 65],
 [155, 48],
 [168, 65],
 [140, 25],
 [95, 15],
 [70, 8],
 [200, 100],
 [178, 66]
]
)

def get_bmi(hw):
 """ Calculate BMI
  @Args
   hw(np.array([[height, width],...])): weight and height ndarray(ndim=2)
  @Returns
   bmi_arr(np.ndarray): BMI ndarray(ndim=1)
 """
 return hw[:, 1]/((hw[:, 0]/100)*(hw[:, 0]/100))

t = get_bmi(X)
```

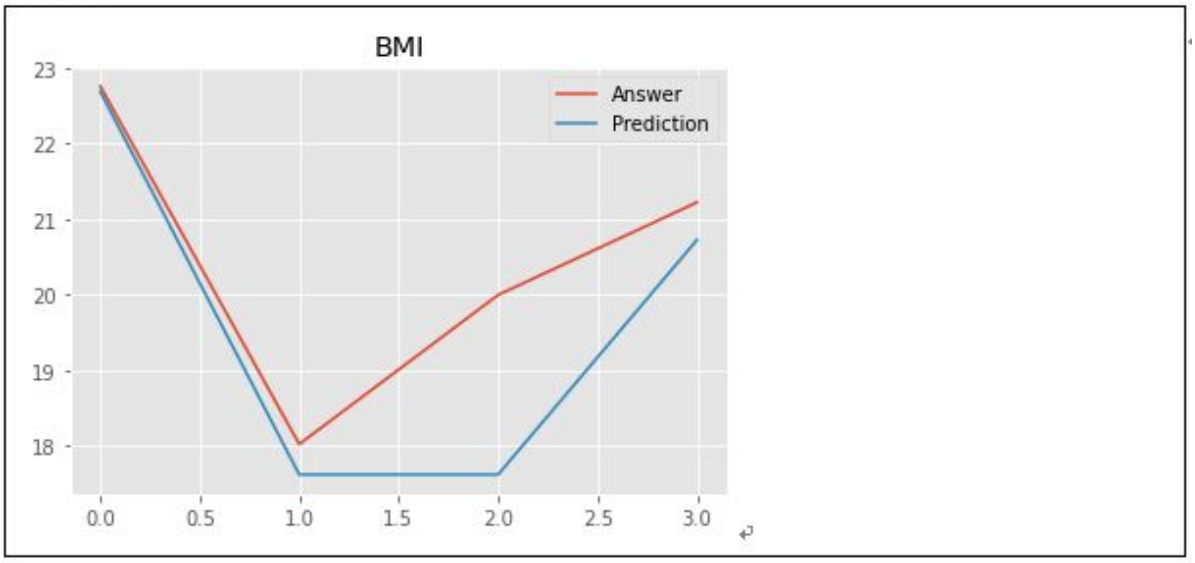
O seguinte código treina o XgBoostRegressor (o modelo XgBoost para a tarefa de regressão). Também. Nós preparamos dados de teste de altura e peso de quatro pessoas e realizamos uma tarefa de previsão. É possível prever o IMC dessas quatro pessoas?

```
xgbr_model = xgb.XGBRegressor(random_state=0)
xgbr_model.fit(X, t)

X_test = np.array([
 [169, 65],
 [158, 45],
 [150, 45],
 [175, 65],
]
)
y_test = get_bmi(X_test)
prediction = xgbr_model.predict(X_test)
```

Vamos visualizar os resultados correctos e previstos usando o matplotlib. Embora houvesse apenas 10 dados de treinamento, pode-se ver que a resposta correta e os resultados da previsão estão bem próximos.

```
import matplotlib.pyplot as plt

plt.title('BMI')
plt.plot(y_test, label="Answer")
plt.plot(prediction, label="Prediction")
plt.legend()


plt.show()
```

Output:



Na avaliação da tarefa de regressão, é utilizado o erro (a diferença entre os dados previstos e os dados correctos), quanto menor for o erro, mais precisa é a previsão. Existem vários tipos de erros, mas geralmente é utilizado o erro quadrático médio (MSE) ou o erro quadrático médio (MSER).

$$\mathrm{MSE}(c) = \frac{1}{n}\sum_{i=1}^{n}(x_i - c)^2$$

Scikit-Learn também tem um método prático para calcular esses valores, então vamos usá-lo.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, prediction)
print(f"Root Mean Square:{np.sqrt(mse)}")
```

Output

```
Root Mean Square:1.2304201718698495
```

Apesar de termos dado apenas 10 dados e, além disso, não termos feito nenhuma afinação dos hiperparâmetros, podemos ver que os erros são relativamente pequenos.

13. Keras

Keras é uma biblioteca de alto nível de redes neurais desenvolvida por François Chollet do Google, que a desenvolveu como um invólucro para o TensorFlow. É uma forma intuitiva de construir redes neuronais, mesmo para iniciantes no aprendizado profundo.

LeNet

AlexNet, que foi capaz de dominar o Concurso de Reconhecimento de Imagem 2012 (ILSVRC2012), é um tipo de rede neural convolucional, e esta rede neural convolucional é baseada na LeNet, que foi desenvolvida pelo Prof. Yann Lecun. Keras É relativamente fácil implementar a LeNet na LeNet, e esperamos que você possa experimentar a precisão que não conseguimos alcançar com a simples rede neural que usamos no método de aprendizagem profunda MNIST no início deste artigo.

Primeiro, importamos as bibliotecas necessárias.

```
import os
import keras
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten, Dropout
from keras.layers.core import Dense
from keras.datasets import mnist
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
```

Definir as funções que definem o modelo LeNet.

```
def lenet(input_shape, num_classes):
 model =Sequential()
 # Convolution using 20 5 x 5 filters and propagation of values to the next layer by the activation function Relu
 # When padding="same", the size of the feature map is 28 x 28, the same as the input, so the output is 28 x 28 x 20.
 model.add(Conv2D(20, kernel_size = 5, padding="same", input_shape=input_shape, activation="relu"))
 # MaxPooling is performed to compress the feature map to 14 x 14 x 20 after processing to find the maximum value every 2 x 2 regions.
 model.add(MaxPooling2D(pool_size=(2,2)))
 # A general technique that can be placed on CNN is to increase the number of filters in the convolution the deeper it is.
 model.add(Conv2D(50, kernel_size=5, padding="same", activation="relu"))
 # Output: 7 x 7 x 50
 model.add(MaxPooling2D(pool_size=(2,2)))
 # The output is converted to a vector.
 model.add(Flatten())
 model.add(Dense(500, activation="relu"))
 # It is used to classify num_classes.
 model.add(Dense(num_classes))
 model.add(Activation("softmax"))
 return model
```

Carregue os dados do MNIST. O pré-processamento também é útil se for methodatizado em um só lugar, por isso classificamos da seguinte forma.

```
class MNISTDataset():
 def __init__(self):
  self.image_shape = (28, 28, 1)
  self.num_classes = 10

 def get_batch(self):
  (x_train, y_train), (x_test, y_test) = mnist.load_data()

  x_train, x_test = [self.preprocess(d) for d in [x_train, x_test]]
  y_train, y_test = [self.preprocess(d, label_data=True) for d in [y_train, y_test]]
  return x_train, y_train, x_test, y_test

 def preprocess(self, data, label_data=False):
  if label_data:
   data = keras.utils.to_categorical(data, self.num_classes)
  else:
   data = data.astype("float32")
   data /= 255
   shape = (data.shape[0],) + self.image_shape
   data = data.reshape(shape)
  return data
```

Não é fácil especificar parâmetros para funções de avaliação e assim por diante, por isso definimos uma classe a ser treinada, dando um modelo.

```
class Trainer():
 def __init__(self, model, loss, optimizer):
  self._target = model
  self._target.compile(loss=loss, optimizer=optimizer, metrics=["accuracy"])

 def train(self, x_train, y_train, batch_size, epochs, validation_split):
   self._target.fit(
     x_train, y_train,
     batch_size=batch_size,
     epochs = epochs,
     validation_split = validation_split
   )
```

Os dados do MNIST estão carregados. Embora a normalização e transformação de forma sejam feitas no pré-processamento, é importante notar que a entrada para a rede neural convolucional é

uma imagem de 28 x 28 pixels que é entrada como está sem achatamento, então a forma não é o mesmo que a entrada para a rede neural simples do método de aprendizagem profunda MNIST mais curto no início.

```
dataset = MNISTDataset()
x_train, y_train, x_test, y_test = dataset.get_batch()
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Output:

```
(60000, 28, 28, 1)
(60000, 10)
(10000, 28, 28, 1)
(10000, 10)
```

O modelo é criado instanciando-o da seguinte forma: summary() exibe o resumo do modelo. Como o número de parâmetros é relativamente grande (1.256.080), é recomendado o uso do modo GPU do Colab porque levará muito tempo para treinar o modelo na CPU.

```
model = lenet(dataset.image_shape, dataset.num_classes)
model.summary()
```

Output:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 28, 28, 20)        520
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 50)        25050
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 50)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2450)              0
_________________________________________________________________
dense_9 (Dense)              (None, 500)               1225500
_________________________________________________________________
dense_10 (Dense)             (None, 10)                5010
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 1,256,080
Trainable params: 1,256,080
Non-trainable params: 0
_________________________________________________________________
```

O treinamento é realizado pela classe Trainer. A taxa de aprendizagem é ajustada usando Adam().

```
trainer = Trainer(model, loss="categorical_crossentropy", optimizer=Adam())
trainer.train(x_train, y_train, batch_size=128, epochs=20, validation_split=0.2)
```

Output:

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 6s 118us/step - loss: 0.0087 - accuracy: 0.9969 -
val_loss: 0.0415 - val_accuracy: 0.9904
Epoch 2/20
48000/48000 [==============================] - 5s 111us/step - loss: 0.0067 - accuracy: 0.9978 -
val_loss: 0.0339 - val_accuracy: 0.9923
Epoch 3/20
48000/48000 [==============================] - 5s 110us/step - loss: 0.0052 - accuracy: 0.9984 -
val_loss: 0.0400 - val_accuracy: 0.9908
Epoch 4/20
48000/48000 [==============================] - 5s 109us/step - loss: 0.0044 - accuracy: 0.9984 -
val_loss: 0.0538 - val_accuracy: 0.9889
Epoch 5/20
```

Época de 6 a 19 é omitida por brevidade.

```
Epoch 20/20
48000/48000 [==============================] - 5s 109us/step - loss: 0.0022 - accuracy: 0.9994 -
val_loss: 0.0607 - val_accuracy: 0.9904
```

Realizaremos uma tarefa de previsão para reconhecimento de números no modelo que criamos e analisaremos as taxas de perda e correção. No caso da rede neural com entrada plana introduzida no capítulo sobre o MNIST mais curto no início deste artigo, a porcentagem de respostas corretas nos dados do teste foi de cerca de 97-98%, enquanto LeNet foi capaz de aumentar a porcentagem de respostas corretas nos dados do teste para 99,1%. Eles foram capazes de melhorar a precisão ao ponto de conseguirem errar ou errar um dos 100 cartões, apesar de terem sido incluídos. Pensa-se que este feito está relacionado com o facto de a estrutura da rede convolucional ter sido inspirada pelo córtex visual humano.

```
score = model.evaluate(x_test, y_test, verbose=0)
y_pred = model.predict(x_test)
print(f"Loss:{score[0]}, Accuracy:{score[1]}")
```

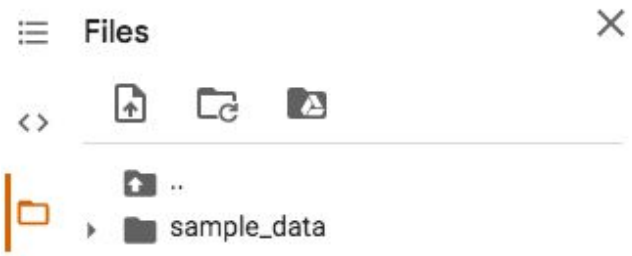
Output:

```
Loss:0.04733146464541562, Accuracy:0.9912999868392944
```

VGG

A modelo pré-treinado de um conjunto de dados de mais de um milhão de imagens chamado ImageNet foi publicado usando o VGG de Oxford, que ficou em segundo lugar no ILSVRC2014. Neste capítulo, a fim de desenvolver uma aplicação de inteligência artificial mais prática, vamos usar este modelo para reconhecer uma imagem arbitrária e executar o código fonte para prever o que é retratado na imagem. Primeiro, nós carregamos a imagem que queremos reconhecer na área temporária do Colab.

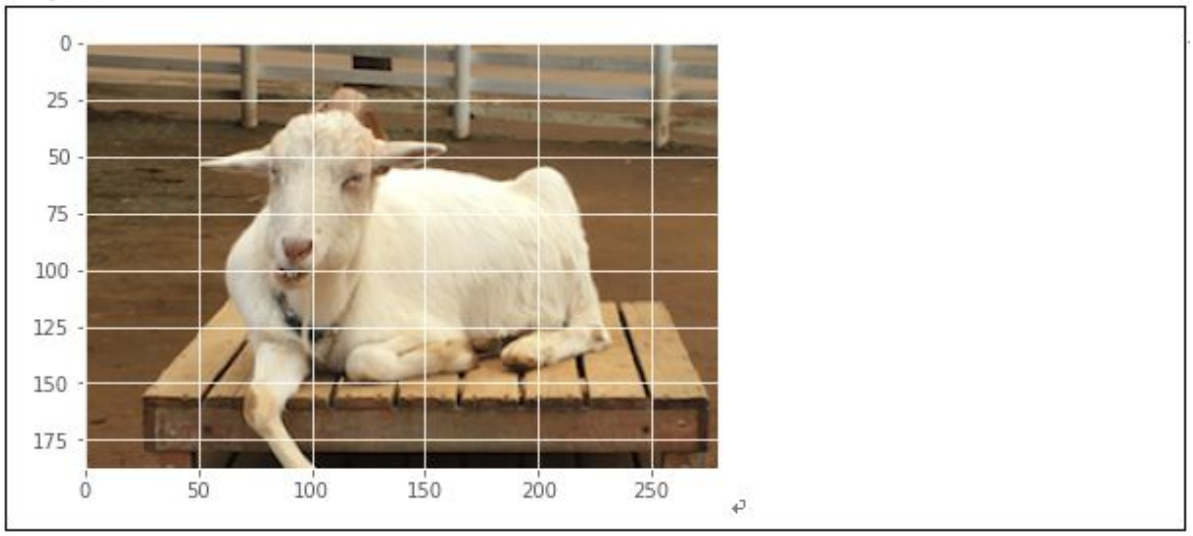
Clique no ícone da pasta no menu superior esquerdo do Colab, depois clique no botão "up-load" e carregue qualquer ficheiro de imagem que queira. Se os colocar na pasta mais antiga, como mostrado abaixo, você pode consultá-los, especificando o caminho relativo do diretório atual. O autor fez o upload de uma imagem de cabra tirada num determinado jardim zoológico no Japão.



Primeiro, vamos garantir que a imagem é carregada e tratada pela Colab usando matplotlib.

```
import matplotlib.pyplot as plt
# Please upload any file from the left folder mark and then specify it.
image_path = "goat.png"
image = Image.load_img(image_path, target_size=(188,280))
plt.imshow(image)
plt.show()
```

Output:



Depois importamos as bibliotecas necessárias para carregar o modelo de pré-treinamento VGG.

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input, decode_predictions
import keras.preprocessing.image as Image
import numpy as np
```

O seguinte código simples pode ser usado para carregar um modelo VGG16 que tenha sido pré-treinado na ImageNet.

```
model = VGG16(weights="imagenet", include_top=True)
```

A seguir, vamos tentar o reconhecimento de imagem. A ImageNet é um conjunto de dados com mais de um milhão de imagens de 224 x 224 pixels, por isso redimensionamos as imagens originais, treinamo-las e executamos a tarefa de previsão. Então, usamos o decode_predictions para convertê-lo para uma etiqueta.

```
image_path = "goat.png"
image = Image.load_img(image_path, target_size=(224, 224))

x = Image.img_to_array(image)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

result = model.predict(x)
result = decode_predictions(result, top=10)[0]
for res in result:
 print(f"Label:{res[1]}, Probability:{round(100*res[2],1)}%")
```

Output:

```
Label:ram, Probability:36.3%
Label:ox, Probability:17.8%
Label:borzoi, Probability:17.5%
Label:oxcart, Probability:7.1%
Label:Ibizan_hound, Probability:4.8%
Label:worm_fence, Probability:2.9%
Label:wallaby, Probability:2.8%
Label:llama, Probability:1.6%
Label:hog, Probability:1.0%
Label:kelpie, Probability:0.6%
```

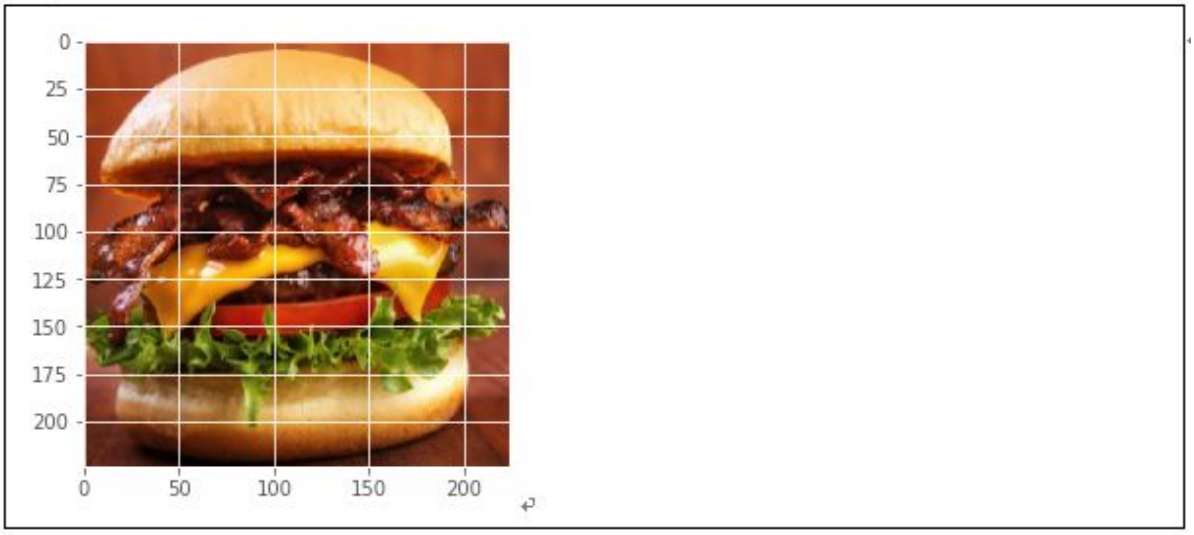
O 1 de cima era um carneiro. Pode ser verdade, mas não havia cabras (cabras) no top 10 porque não havia muitas cabras japonesas nos dados de treino da ImageNet. É bem sabido que os dados japoneses e os dados da ImageNet são diferentes, por isso o udon e o macarrão duque são julgados como esparguete. Para lidar com os dados de imagem japoneses, é necessário treinar novamente uma parte do modelo que está próxima da camada de saída do modelo pré-treinado, chamada fine tuning. Eu não vou cobrir o ajuste fino neste livro, mas se você estiver interessado, há um artigo na Qiita que procura criar uma IA de reconhecimento facial conectando todas as camadas acopladas ao VGG16 e usando a GPU para ajustar o VGG16.

Como a ImageNet é um conjunto de dados centrado nos EUA, para começar, achei que se parecesse americana, deveria ser

reconhecida, então consegui uma imagem de um hambúrguer e deixei que ela a reconhecesse.

```
image_path = "hamburger2.png"
# Please upload any file from the left folder mark and then specify it.
image = Image.load_img(image_path, target_size=(224,224))
plt.imshow(image)
plt.show()
```

Output:



```
x = Image.img_to_array(image)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

result = model.predict(x)
result = decode_predictions(result, top=10)[0]

for res in result:
 print(f"Label:{res[1]}, Probability:{round(100*res[2],1)}%")
```

Output:

```
Label:cheeseburger, Probability:100.0%
Label:bagel, Probability:0.0%
Label:guacamole, Probability:0.0%
Label:hotdog, Probability:0.0%
Label:meat_loaf, Probability:0.0%
Label:potpie, Probability:0.0%
Label:burrito, Probability:0.0%
Label:plate, Probability:0.0%
Label:broccoli, Probability:0.0%
Label:mushroom, Probability:0.0%
```

Então, para minha surpresa, eles responderam com 100% de confiança que era 100% cheeseburgers, incluindo o tipo de hambúrguer, e eles estavam 100% corretos. Afinal, parece que devíamos tratar a ImageNet como um olho americano.

Ligação de dados para a nova Corona (covid-19)

A estão disponíveis ao público dados variados, incluindo dados sobre o número de testes e séries de casos dos novos coronavírus por região e o número de pessoas infectadas pelos novos coronavírus, e várias competições estão sendo realizadas, por isso incluí links para cada uma das competições nacionais e internacionais mais famosas, uma para cada uma.

If você é um cientista de dados, engenheiro de aprendizagem de máquinas ou outro profissional, usando suas habilidades de análise de dados para compartilhar os resultados de sua análise pode resultar em salvar vidas. E se você é um iniciante, se você leu este livro cuidadosamente enquanto executava o código fonte, você será habilitado o suficiente para criar modelos preditivos e enviar os resultados em um concurso. Kaggle também poderá usar Numpy/Pandas/Matplotlib para criar EDA (Muitos notebooks com os resultados da análise exploratória de dados) já estão disponíveis, então você não precisa entrar no concurso para se referir a eles, o que é muito útil. Esperamos que você dê uma chance!

Kaggle(Portuguese)

https://www.kaggle.com/covid19

SIGNATO(Inglês/Japonês)

https://signate.jp/competitions/260