welcome-to-colaboratory

January 31, 2024

Welcome to Colab!

(New) Try the Gemini API

Generate a Gemini API key

Talk to Gemini with the Speech-to-Text API

Compare Gemini with ChatGPT

More notebooks

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.

[9]:
```

```

What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with - Zero configuration required - Access to GPUs free of charge - Easy sharing

Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier. Watch Introduction to Colab to find out more, or just get started below!

0.1 Getting started

The document that you are reading is not a static web page, but an interactive environment called a Colab notebook that lets you write and execute code.

For example, here is a code cell with a short Python script that computes a value, stores it in a variable and prints the result:

[10]:
```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

[10]:
```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut 'Command/Ctrl+Enter'. To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

[11]:
```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

[11]:
```
604800
```

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To find out more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: Create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To find out more about the Jupyter project, see jupyter.org.

0.2 Data science

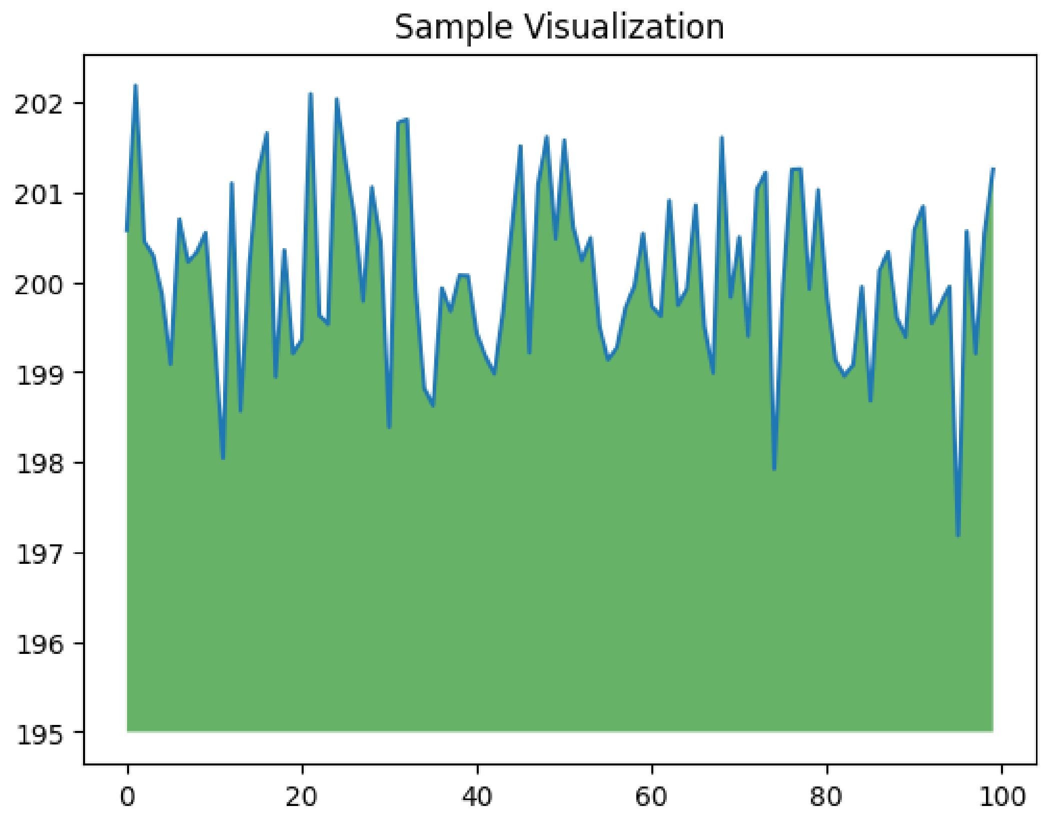
With Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses numpy to generate some random data, and uses matplotlib to visualise it. To edit the code, just click the cell and start editing.

[12]:
```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from GitHub and many other sources. To find out more about importing data, and how Colab can be used for data science, see the links below under Working with data.

0.3 Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including: - Getting started with TensorFlow - Developing and training neural networks - Experimenting with TPUs - Disseminating AI research - Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

0.4 More resources

0.4.1 Working with notebooks in Colab

- Overview of Colaboratory
- Guide to markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

### Working with data

- Loading data: Drive, Sheets and Google Cloud Storage
- Charts: visualising data
- Getting started with BigQuery

0.4.2 Machine learning crash course

These are a few of the notebooks from Google's online machine learning course. See the full course website for more. - Intro to Pandas DataFrame - Linear regression with tf.keras using synthetic data

### Using accelerated hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

0.4.3 Featured examples

- NeMo voice swap: Use Nvidia NeMo conversational AI toolkit to swap a voice in an audio fragment with a computer-generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB film reviews as either positive or negative.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine-learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

[12]:

```
```

[13]:

```
from collections import deque

# Define the graph as an adjacency list
graph = {
    0: [1, 2],
    1: [0, 3, 4],
    2: [0, 5],
    3: [1],
    4: [1],
    5: [2]
}

# Define the BFS function
def bfs(graph, start_vertex):
    # Initialize the visited set to keep track of visited vertices
    visited = set()

    # Initialize the queue with the starting vertex
    queue = deque([start_vertex])

    # Loop until the queue is empty
    while queue:
        # Dequeue the next vertex from the queue
        current_vertex = queue.popleft()

        # If the current vertex has not been visited yet, print it and mark it␣
 ↪as visited
        if current_vertex not in visited:
            print(current_vertex)
            visited.add(current_vertex)

            # Enqueue the neighbors of the current vertex that have not been␣
 ↪visited yet
            for neighbor in graph[current_vertex]:
                if neighbor not in visited:
                    queue.append(neighbor)

# Call the BFS function with the graph and a starting vertex
bfs(graph, 0)
```

```
0
1
2
3
4
5
```

[14]:
```
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            print(vertex)
            stack.extend(neighbor for neighbor in graph[vertex] if neighbor not␣
↪in visited)

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

dfs(graph, 'A')
```

```
A
C
F
E
B
D
```

[15]:

```
import heapq

class PuzzleNode:
    def init(self, state, g_value, heuristic):
        self.state = state
        self.g_value = g_value
        self.heuristic = heuristic

    def lt(self, other):
        return (self.g_value + self.heuristic) < (other.g_value + other.
↪heuristic)

class EightPuzzleSolver:
    def init(self, initial_state, goal_state):
        self.initial_state = initial_state
        self.goal_state = goal_state
        self.moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]  # Possible moves:␣
↪right, left, down, up

    def calculate_heuristic(self, state):
        # Your heuristic function h(x) implementation for the 8-puzzle problem
        # You can use various heuristics such as Manhattan distance, misplaced␣
↪tiles, etc.
```

```
        # For simplicity, let's assume the heuristic is the count of misplaced␣
↪tiles.
        misplaced_tiles = sum([1 for i, j in zip(state, self.goal_state) if i !
↪= j])
        return misplaced_tiles

    def is_valid_move(self, x, y):
        return 0 <= x < 3 and 0 <= y < 3

    def generate_next_states(self, current_state):
        zero_index = current_state.index(0)
        zero_x, zero_y = zero_index % 3, zero_index // 3
        next_states = []

        for dx, dy in self.moves:
            new_x, new_y = zero_x + dx, zero_y + dy

            if self.is_valid_move(new_x, new_y):
                new_state = current_state[:]
                new_index = new_y * 3 + new_x
                new_state[zero_index], new_state[new_index] =␣
↪new_state[new_index], new_state[zero_index]

                next_states.append(new_state)

        return next_states

    def solve_puzzle(self):
        initial_node = PuzzleNode(self.initial_state, 0, self.
↪calculate_heuristic(self.initial_state))
        priority_queue = [initial_node]
        visited_states = set()

        while priority_queue:
            current_node = heapq.heappop(priority_queue)

            if current_node.state == self.goal_state:
                return current_node.g_value  # Return the cost to reach the goal

            visited_states.add(tuple(current_node.state))

            for next_state in self.generate_next_states(current_node.state):
                if tuple(next_state) not in visited_states:
                    next_g_value = current_node.g_value + 1
                    next_heuristic = self.calculate_heuristic(next_state)
                    next_node = PuzzleNode(next_state, next_g_value,␣
↪next_heuristic)
```