



Kissipo Learning for Deep Learning

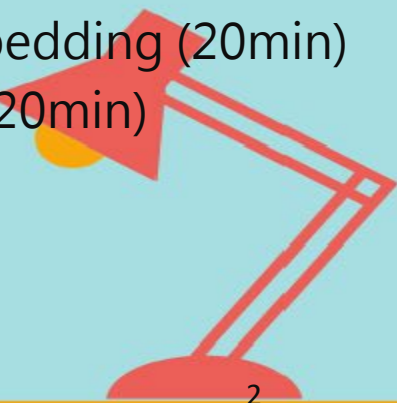
Topic 2: Kissipo Learning for Deep Learning

Hsueh-Ting Chu

KLDL-W1-02

Topics

- Topic 01: Introduction to Deep Learning (20min)
- Topic 02: Kissipo Learning for Deep Learning (20min)
- Topic 03: Python quick tutorial (20min)
- Topic 04: Numpy quick tutorial (15min)
- Topic 05: Pandas quick tutorial (15min)
- Topic 06: Scikit-learn quick tutorial (15min)
- Topic 07: OpenCV quick tutorial (15min)
- Topic 08: Image Processing basics (20min)
- Topic 09: Machine Learning basics (20min)
- Topic 10: Deep Learning basics (20min)
- Topic 11: TensorFlow overview (20min)
- Topic 12: CNN with TensorFlow (20min)
- Topic 13: RNN with TensorFlow (20min)
- Topic 14: PyTorch overview (20min)
- Topic 15: CNN with PyTorch (20min)
- Topic 16: RNN with Pytorch (20min)
- Topic 17: Introduction to AOI (20min)
- Topic 18: AOI simple Pipeline (A) (20min)
- Topic 19: AOI simple Pipeline (B) (20min)
- Topic 20: Introduction to Object detection (20min)
- Topic 21: YoloV5 Quick Tutorial (20min)
- Topic 22: Using YoloV5 for RSD (20min)
- Topic 23: Introduction to NLP (20min)
- Topic 24: Introduction to Word Embedding (20min)
- Topic 25: Name prediction project (20min)



Content

- Topic 2: Kissipo Learning for Deep Learning
 - Kissipo Learning
 - Jupyter notebook and Colab
 - Input/output of deep learning models



Kissipo

Kissipo = KISS principle + IPO model

KISS principle

https://en.wikipedia.org/wiki/KISS_principle

IPO model

https://en.wikipedia.org/wiki/IPO_model



KISS principle

KISS, an acronym for **keep it simple, stupid**, is a design principle noted by the U.S. Navy in 1960.

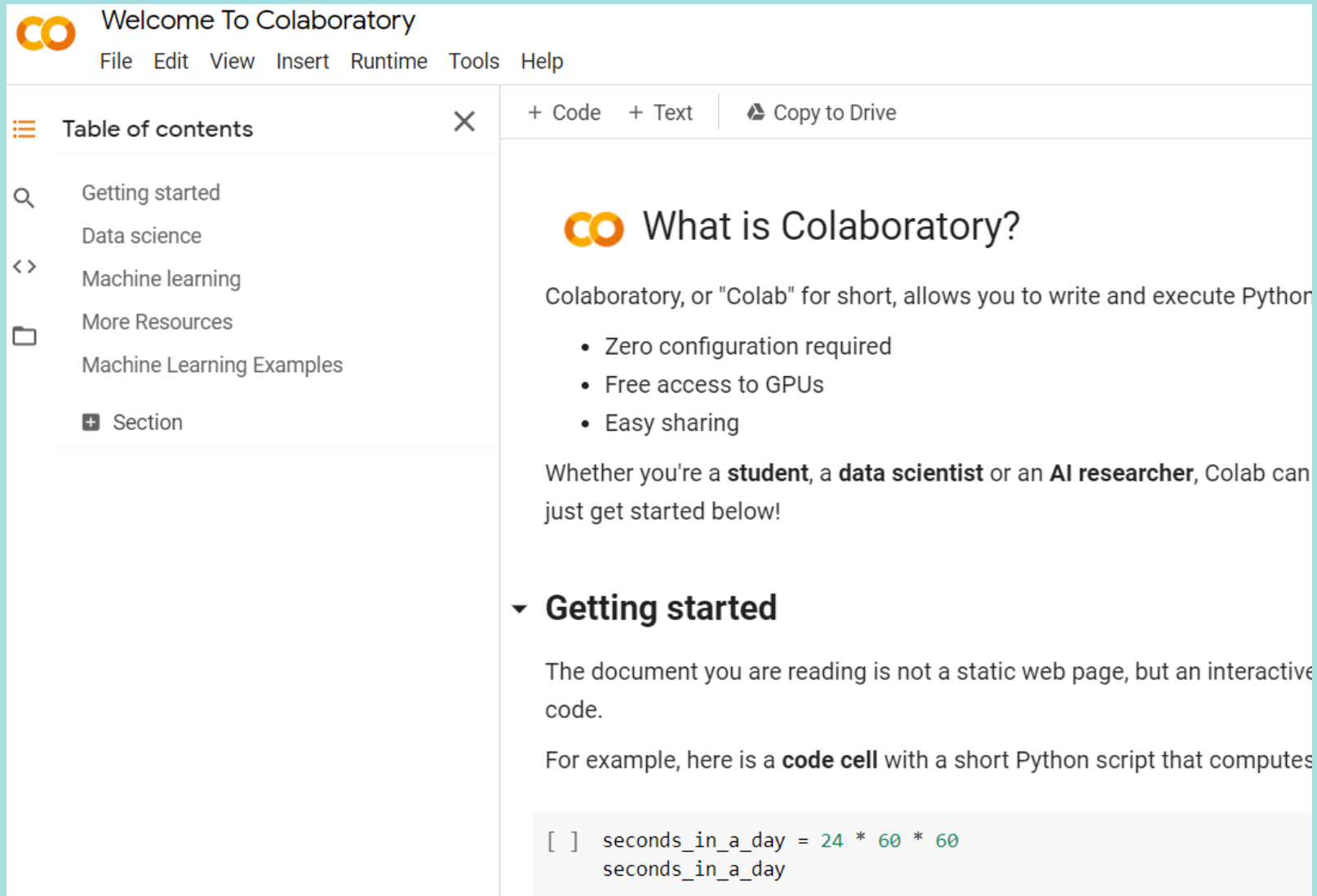
The KISS principle states that **most systems work best** if they are kept simple rather than made complicated;

therefore, **simplicity should be a key goal** in design, and unnecessary complexity should be avoided.

https://en.wikipedia.org/wiki/KISS_principle



Python programming with Colab



Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

Table of contents

- Getting started
- Data science
- Machine learning
- More Resources
- Machine Learning Examples
- + Section

+ Code + Text Copy to Drive

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can just get started below!

Getting started

The document you are reading is not a static web page, but an interactive code.

For example, here is a **code cell** with a short Python script that computes

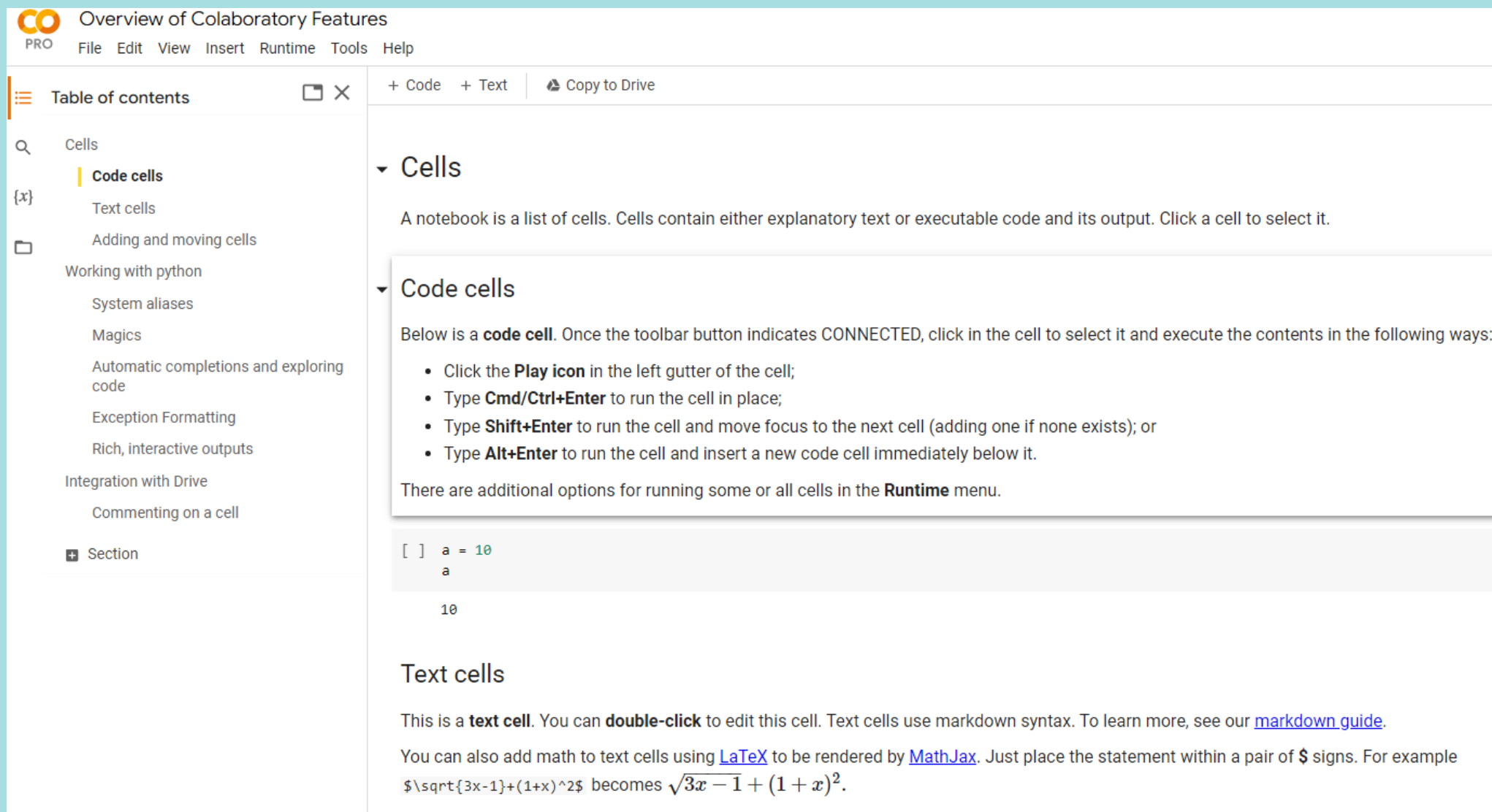
```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```



Jupyter Notebook



Overview of Colab Features



The screenshot shows the Google Colaboratory interface. At the top, the title bar reads "Overview of Colaboratory Features" with a "PRO" badge and a menu bar containing "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". On the left, a "Table of contents" sidebar lists sections like "Cells", "Text cells", "Adding and moving cells", "Working with python", "System aliases", "Magics", "Automatic completions and exploring code", "Exception Formatting", "Rich, interactive outputs", "Integration with Drive", "Commenting on a cell", and "Section". The main content area is divided into sections: "Cells" (describing notebooks as lists of cells), "Code cells" (providing instructions on how to execute code cells using toolbar icons or keyboard shortcuts like **Cmd/Ctrl+Enter**, **Shift+Enter**, and **Alt+Enter**), and "Text cells" (explaining markdown syntax and LaTeX rendering). A code cell is shown with the input `a = 10` and the output `10`. The interface also includes a toolbar with buttons for adding code or text cells and copying to Drive.

Overview of Colaboratory Features

File Edit View Insert Runtime Tools Help

Table of contents

- Cells
 - Code cells
 - Text cells
- Adding and moving cells
- Working with python
 - System aliases
 - Magics
 - Automatic completions and exploring code
 - Exception Formatting
 - Rich, interactive outputs
- Integration with Drive
 - Commenting on a cell
- Section

Cells

A notebook is a list of cells. Cells contain either explanatory text or executable code and its output. Click a cell to select it.

Code cells

Below is a **code cell**. Once the toolbar button indicates CONNECTED, click in the cell to select it and execute the contents in the following ways:

- Click the **Play icon** in the left gutter of the cell;
- Type **Cmd/Ctrl+Enter** to run the cell in place;
- Type **Shift+Enter** to run the cell and move focus to the next cell (adding one if none exists); or
- Type **Alt+Enter** to run the cell and insert a new code cell immediately below it.

There are additional options for running some or all cells in the **Runtime** menu.


```
[ ] a = 10
a
10
```

Text cells


This is a **text cell**. You can **double-click** to edit this cell. Text cells use markdown syntax. To learn more, see our [markdown guide](#).





You can also add math to text cells using [LaTeX](#) to be rendered by [MathJax](#). Just place the statement within a pair of **\$** signs. For example `$\sqrt{3x-1} + (1+x)^2$` becomes $\sqrt{3x-1} + (1+x)^2$.

Colab Markdown Guide

 **Markdown Guide**

File Edit View Insert Runtime Tools Help

+ Code + Text  Copy to Drive







What is Markdown?

Colab has two types of cells: text and code. Text cells are formatted using a simple markup language called Markdown.

To see the Markdown source, double-click a text cell, showing both the Markdown source and the rendered version. Above the Markdown source there is a toolbar to assist editing.

Reference

Markdown	Preview
<code>**bold text**</code>	bold text
<code>*italicized text*</code> or <code>_italicized text_</code>	<i>italicized text</i>
<code>`Monospace`</code>	Monospace
<code>~~strikethrough~~</code>	strikethrough
<code>[A link](https://www.google.com)</code>	A link
<code>![An image](https://www.google.com/images/rss.png)</code>	

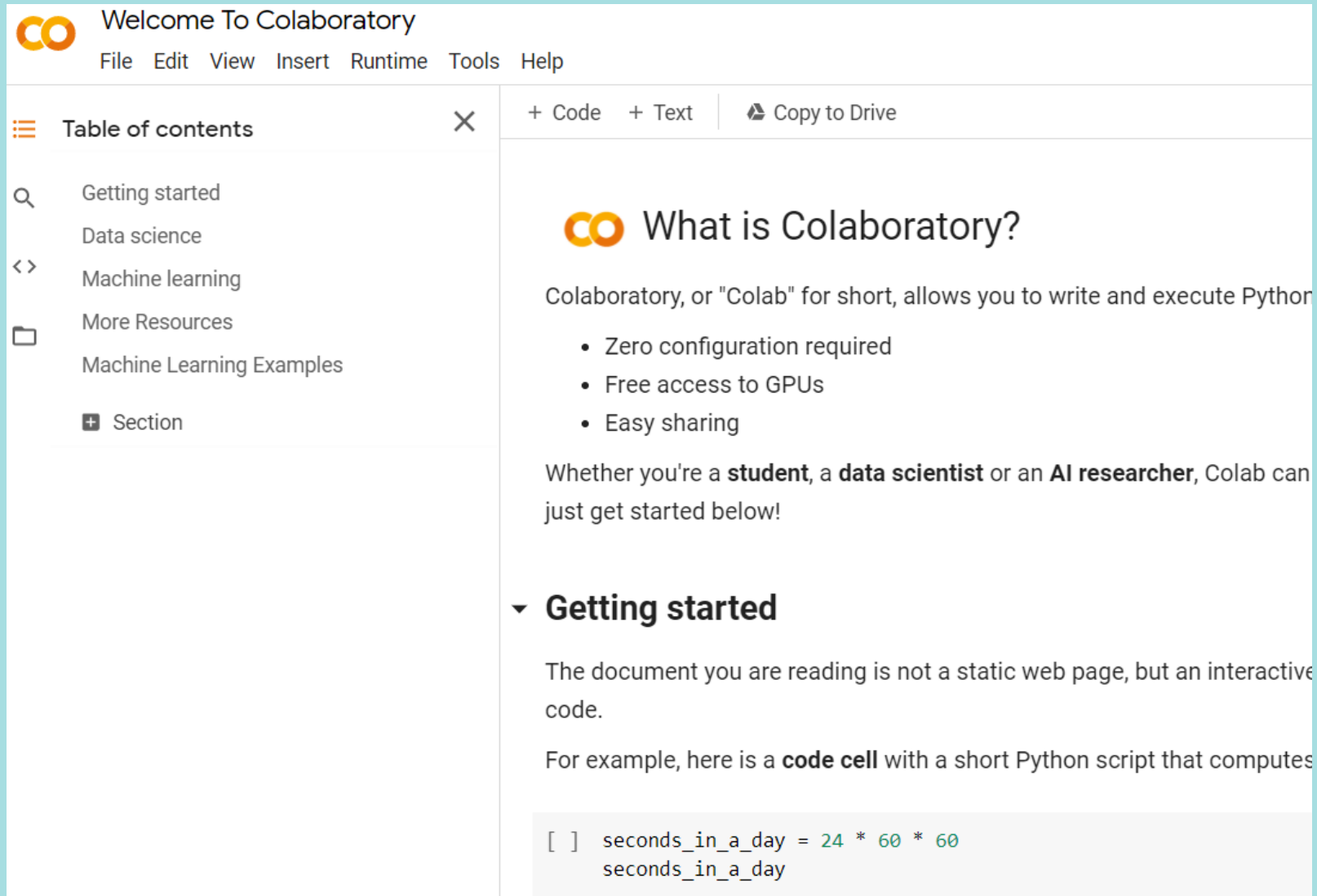
Headings are rendered as titles.

```
# Section 1
# Section 2
## Sub-section under Section 2
### Sub-section under the sub-section under Section 2
# Section 3
```

Section 1



Charts in Colab



Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

Table of contents

- Getting started
- Data science
- Machine learning
- More Resources
- Machine Learning Examples
- + Section

+ Code + Text Copy to Drive

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can just get started below!

Getting started

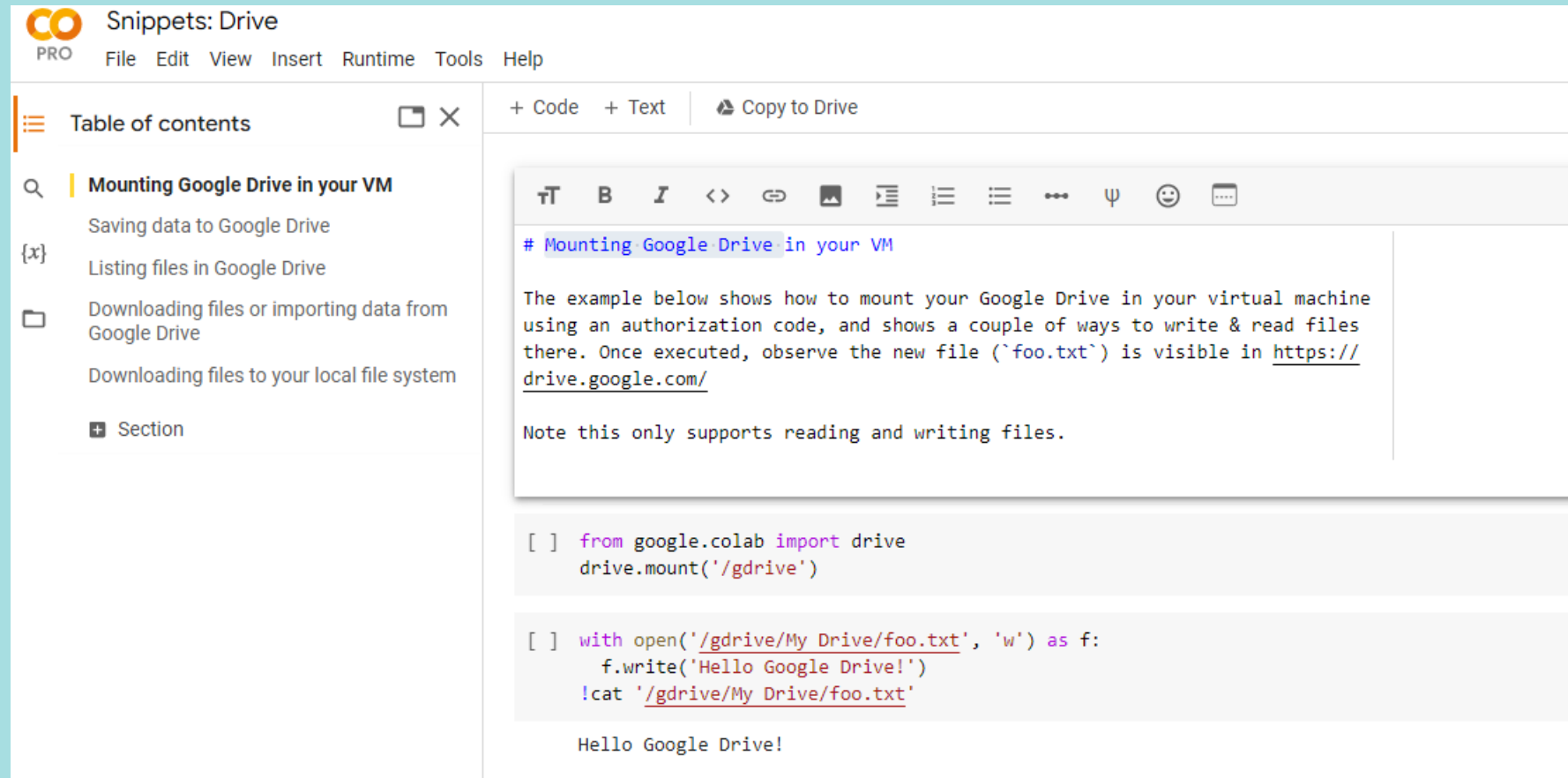
The document you are reading is not a static web page, but an interactive code.

For example, here is a **code cell** with a short Python script that computes

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



Mounting Google Drive



The screenshot shows the Google Colab 'Snippets: Drive' interface. On the left is a 'Table of contents' sidebar with a search icon and a list of topics: 'Mounting Google Drive in your VM', 'Saving data to Google Drive', 'Listing files in Google Drive', 'Downloading files or importing data from Google Drive', 'Downloading files to your local file system', and a '+ Section' button. The main area displays a snippet titled '# Mounting Google Drive in your VM'. The snippet text explains how to mount a Google Drive in a VM using an authorization code and shows two ways to write and read files. It includes a note that this only supports reading and writing files. Below the text are two code blocks. The first block shows the import and mount command. The second block shows a file write and cat command. The output of the second block is 'Hello Google Drive!'.

CO PRO Snippets: Drive
File Edit View Insert Runtime Tools Help

Table of contents

- Mounting Google Drive in your VM
- Saving data to Google Drive
- Listing files in Google Drive
- Downloading files or importing data from Google Drive
- Downloading files to your local file system
- + Section

+ Code + Text Copy to Drive

Mounting Google Drive in your VM

The example below shows how to mount your Google Drive in your virtual machine using an authorization code, and shows a couple of ways to write & read files there. Once executed, observe the new file (`'foo.txt'`) is visible in <https://drive.google.com/>

Note this only supports reading and writing files.

```
[ ] from google.colab import drive
    drive.mount('/gdrive')
```

```
[ ] with open('/gdrive/My Drive/foo.txt', 'w') as f:
    f.write('Hello Google Drive!')
    !cat '/gdrive/My Drive/foo.txt'
```

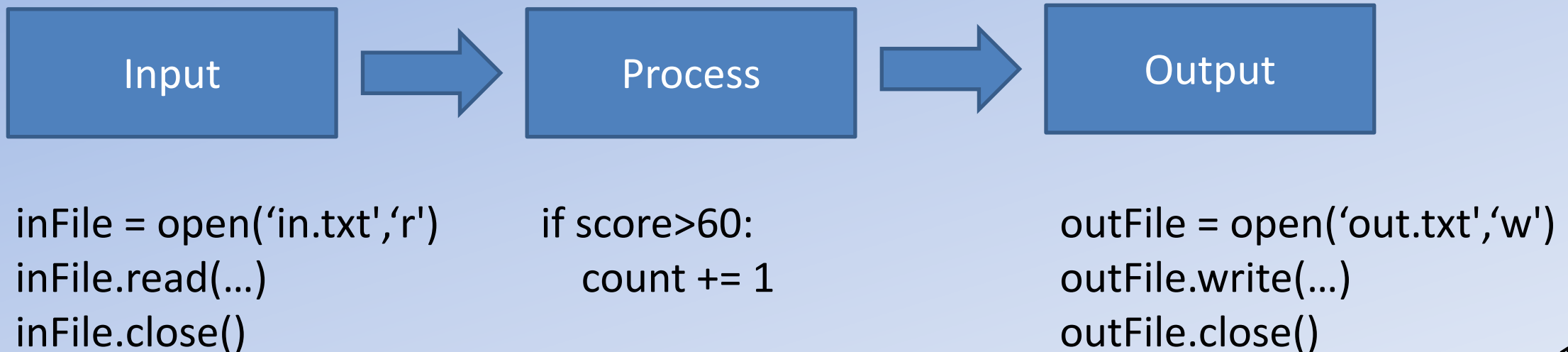
Hello Google Drive!



Input-Process-Output Model

The input–process–output (IPO) model, or input-process-output pattern, is a widely used approach in systems analysis and software engineering for describing the structure of an information processing program or other process. Many introductory programming and systems analysis texts introduce this as the most basic structure for describing a process.

https://en.wikipedia.org/wiki/IPO_model



Jupyter Notebook Cheat Sheet

• edureka! •
Learn PYTHON from experts at <https://www.edureka.co>

JUPYTER NOTEBOOK CHEAT SHEET

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It is used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Saving/Loading Notebook

File Edit View

- Create new Notebook
- Open...
- Make a Copy of the current Notebook
- Save as...
- Rename current Notebook
- Save and Checkpoint
- Revert Notebook to a previous Checkpoint
- Download Notebook as Python Notebook, HTML, Markdown, PDF
- Trusted Notebook
- Close Notebook & stop running scripts

Edit Cells

Edit View Insert

- Cut Cells
- Copy Cells
- Paste Cells Above
- Paste Cells Below
- Paste Cells & Replace
- Delete Cells
- Undo Delete Cells
- Split Cell
- Merge Cell Above
- Merge Cell Below
- Move Cell Up
- Move Cell Down
- Edit Notebook Metadata
- Find and Replace
- Copy Attachments of current cell
- Paste Attachments of current cell
- Insert Image

View Cells

View Insert Cell

- Toggle Header
- Toggle Line Numbers
- Cell Toolbar

Insert Cells

Insert Cell Above

Insert Cell Below

Keyboard Shortcuts

Command	Description
enter	enter edit mode
Command + a; Command + c; Command + v	select all; copy; paste
Command + z; Command + y	undo; redo
Command + s	save and checkpoint
Command + b; Command + a	insert cell below; insert cell above
Shift + Enter	run cell, select below
Shift + m	merge cells
Command +] Command + [indent; dedent
Ctrl + Enter	run cell
Option + Return	run cell, insert cell below
Escape	enter command mode
Escape + d + d	delete selected cell
Escape + y	change cell to code
Escape + m	change cell to markdown
Escape + r	change cell to raw
Escape + 1	change cell to heading 1
Escape + n	create cell below
Escape + b	create cell below
Escape + a	insert cell above

Magic Commands

Statement	Explanation	Example
%magic	Comprehensively lists and explains magic functions	%magic
%automagic	When active, enables you to call magic functions without the '%'	%automagic
%quickref	Launch IPython quick reference	%quickref
%pastebin	Paste bins lines from your current session.	%pastebin 3 18-20 ~1/1-5
%debug	Enters the interactive debugger	%debug
%hist	Print command input and output history	%hist
%pdb	Automatically enter python debugger after any exception	%pdb
%cpaste	Opens up a special prompt for manually pasting Python code for execution	%cpaste
%reset	Delete all variables and names defined in the current namespace	%reset
%run	Run a python script inside a notebook	%run script.py
%who, %who_ls, %whos	Display variables defined in the interactive namespace, with varying levels of verbosity	%who, %who_ls, %whos
%xdel	Delete a variable in the local namespace. Clear any references to that variable	%xdel variable
%time	Times a single statement	In [561]: %time method = [a for a in data if b.startswith('http')]

Execute Cells

Cell Kernel Widgets

- Run Current Cell down & create one below
- Run All Cells
- Run All Cells above the current one
- Toggle & clear current outputs
- Run Selected Cells
- Run Cells and Select Below
- Run Cells and Insert Below
- Run All
- Run All Above
- Run All Below
- Cell Type
- Current Outputs
- All Output

Kernel Cells

Kernel Widgets Help

- Restart Kernel
- Restart Kernel & Run all cells
- Shutdown all cells
- Run other installed kernels
- Interrupt kernel
- Interrupt kernel & Clear all output
- Reconnect to a remote Notebook
- Change kernel

Widgets

Widgets Help

- Save Notebook Widget State
- Clear Notebook Widget State
- Download Widget State
- Embed Widgets

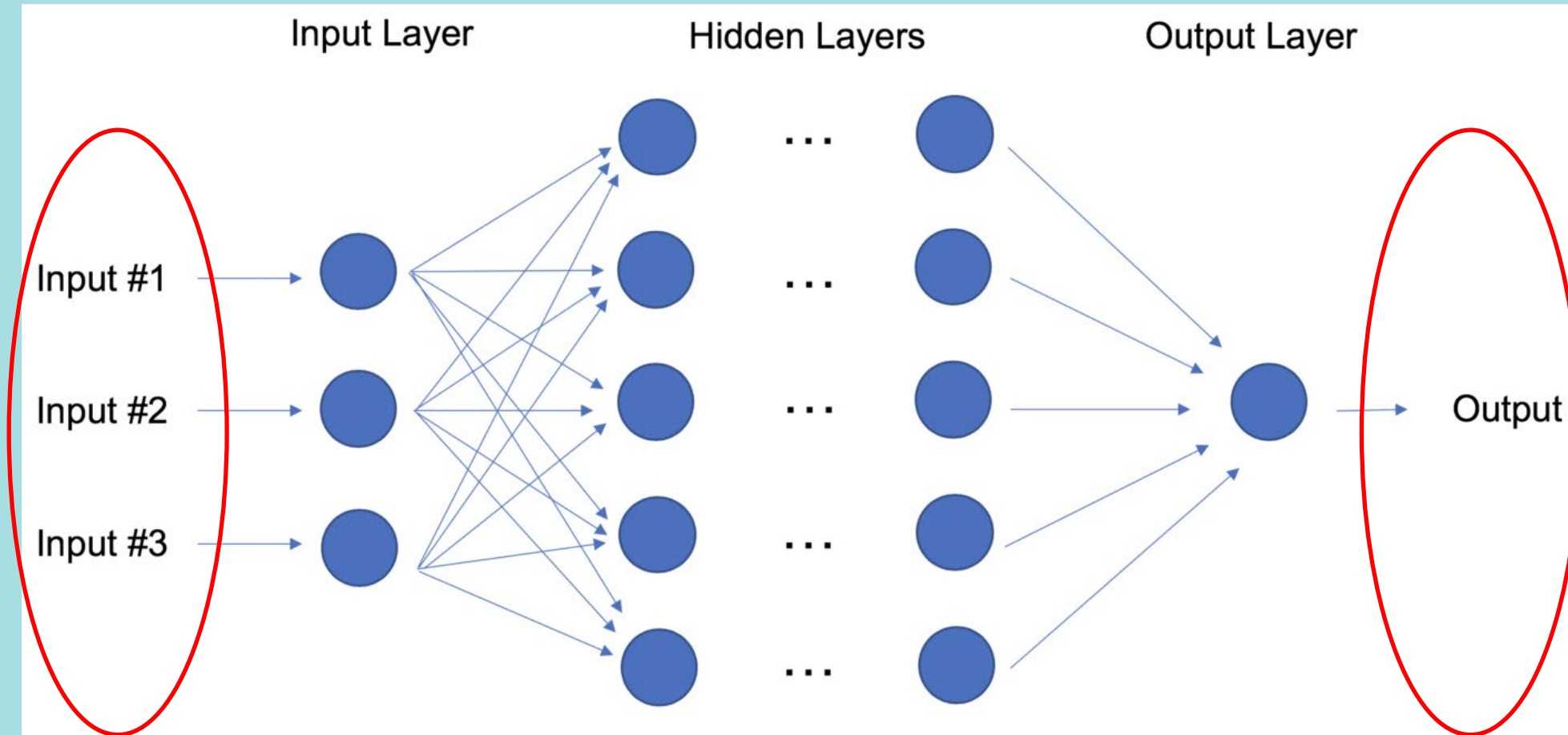
Help

Help

- User Interface Tour
- Keyboard Shortcuts
- Edit Keyboard Shortcuts
- Notebook Help
- Markdown
- Python help topics
- NumPy help topics
- Matplotlib help topics
- Pandas help topics
- Walk through a UI Tour
- Edit the default keyboard shortcuts
- Markdown available in Notebook
- IPython help topics
- SciPy help topics
- SymPy help topics
- About Jupyter Notebook

Command	Description
enter	enter edit mode
Command + a; Command + c; Command + v	select all; copy; paste
Command + z; Command + y	undo; redo
Command + s	save and checkpoint
Command + b; Command + a	insert cell below; insert cell above
Shift + Enter	run cell, select below
Shift + m	merge cells
Command +] Command + [indent; dedent
Ctrl + Enter	run cell
Option + Return	run cell, insert cell below
Escape	enter command mode
Escape + d + d	delete selected cell
Escape + y	change cell to code
Escape + m	change cell to markdown
Escape + r	change cell to raw
Escape + 1	change cell to heading 1
Escape + n	change cell to heading n
Escape + b	create cell below
Escape + a	Insert cell above

Input/output of deep learning models



Taxi Trajectory Prediction

ECML/PKDD 15: Taxi Trajectory Prediction (I)

Predict the destination of taxi trips based on initial partial trajectories

\$250

Prize Money

381 teams · 7 years ago

[Overview](#) [Data](#) [Code](#) [Discussion](#) [Leaderboard](#) [Rules](#)

Late Submission

...

Overview

Description

Evaluation

Prizes

Timeline

The taxi industry is evolving rapidly. New competitors and technologies are changing the way traditional taxi services do business. While this evolution has created new efficiencies, it has also created new problems.

One major shift is the widespread adoption of electronic dispatch systems that have replaced the VHF-radio dispatch systems of times past. These mobile data terminals are installed in each vehicle and typically provide information on GPS localization and taximeter state. Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride.



Another recent change is the switch from broadcast-based (one to many) radio messages for service dispatching to unicast-based (one to one) messages. With unicast-messages, the dispatcher needs to correctly identify which taxi they should dispatch to a pick up location. Since taxis using electronic dispatch systems do not usually enter their drop off location, it is extremely difficult for dispatchers to know which taxi to contact.



Taxi Trajectory Dataset

1. **TRIP_ID:** (String) It contains an unique identifier for each trip;
2. **CALL_TYPE:** (char) It identifies the way used to demand this service. It may contain one of three possible values:
 - a. 'A' if this trip was dispatched from the central;
 - b. 'B' if this trip was demanded directly to a taxi driver on a specific stand;
 - c. 'C' otherwise (i.e. a trip demanded on a random street).
3. **ORIGIN_CALL:** (integer) It contains an unique identifier for each phone number which was used to demand, at least, one service. It identifies the trip's customer if CALL_TYPE='A'. Otherwise, it assumes a NULL value;
4. **ORIGIN_STAND:** (integer): It contains an unique identifier for the taxi stand. It identifies the starting point of the trip if CALL_TYPE='B'. Otherwise, it assumes a NULL value;
5. **TAXI_ID:** (integer): It contains an unique identifier for the taxi driver that performed each trip;
6. **TIMESTAMP:** (integer) Unix Timestamp (in seconds). It identifies the trip's start;
7. **DAYTYPE:** (char) It identifies the daytype of the trip's start. It assumes one of three possible values:
 - a. 'B' if this trip started on a holiday or any other special day (i.e. extending holidays, floating holidays, etc.);
 - b. 'C' if the trip started on a day before a type-B day;
 - c. 'A' otherwise (i.e. a normal day, workday or weekend).
8. **MISSING_DATA:** (Boolean) It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing
9. **POLYLINE:** (String): It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start;



Thanks!

Q&A

