Kissipo Learning for Deep Learning

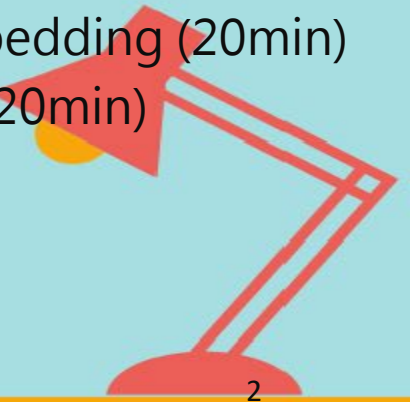
Topic 2: Kissipo Learning for Deep Learning

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.

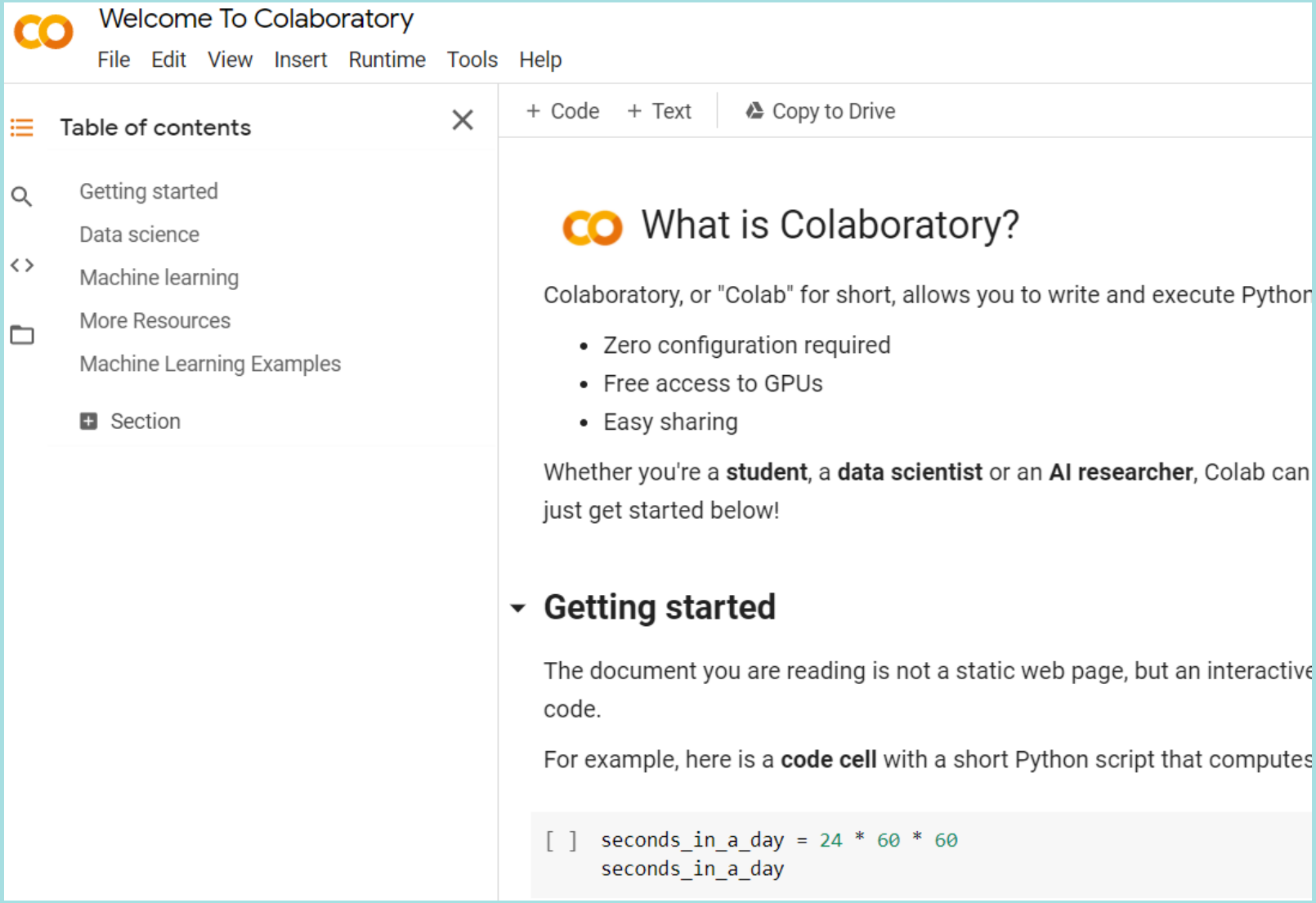
The KISS principle states that most systems work best if they are kept simple rather than made complicated;

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

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

5

Python programing with Colab



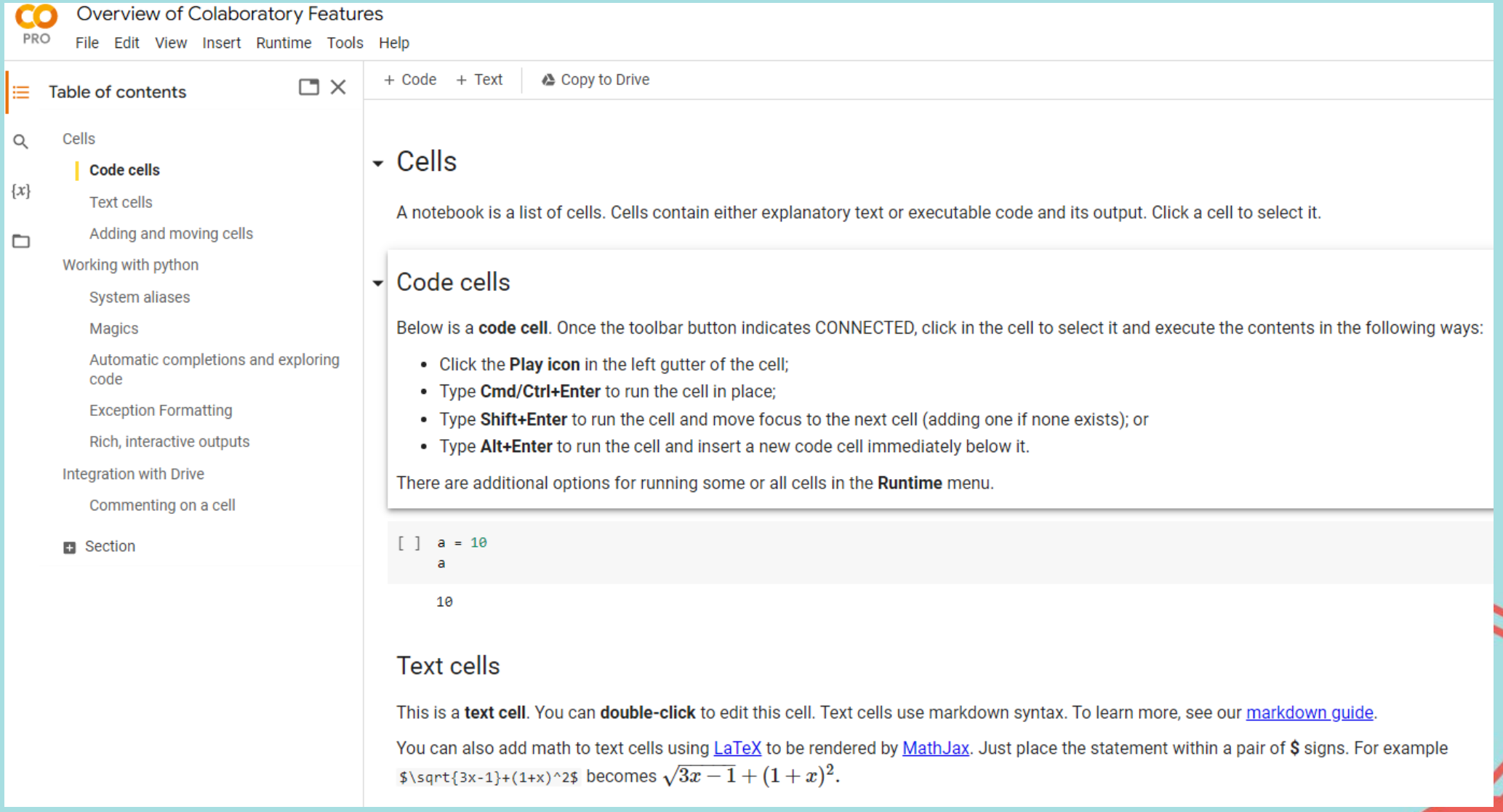
https://colab.research.google.com/

Jupyter Notebook

https://jupyter.org/

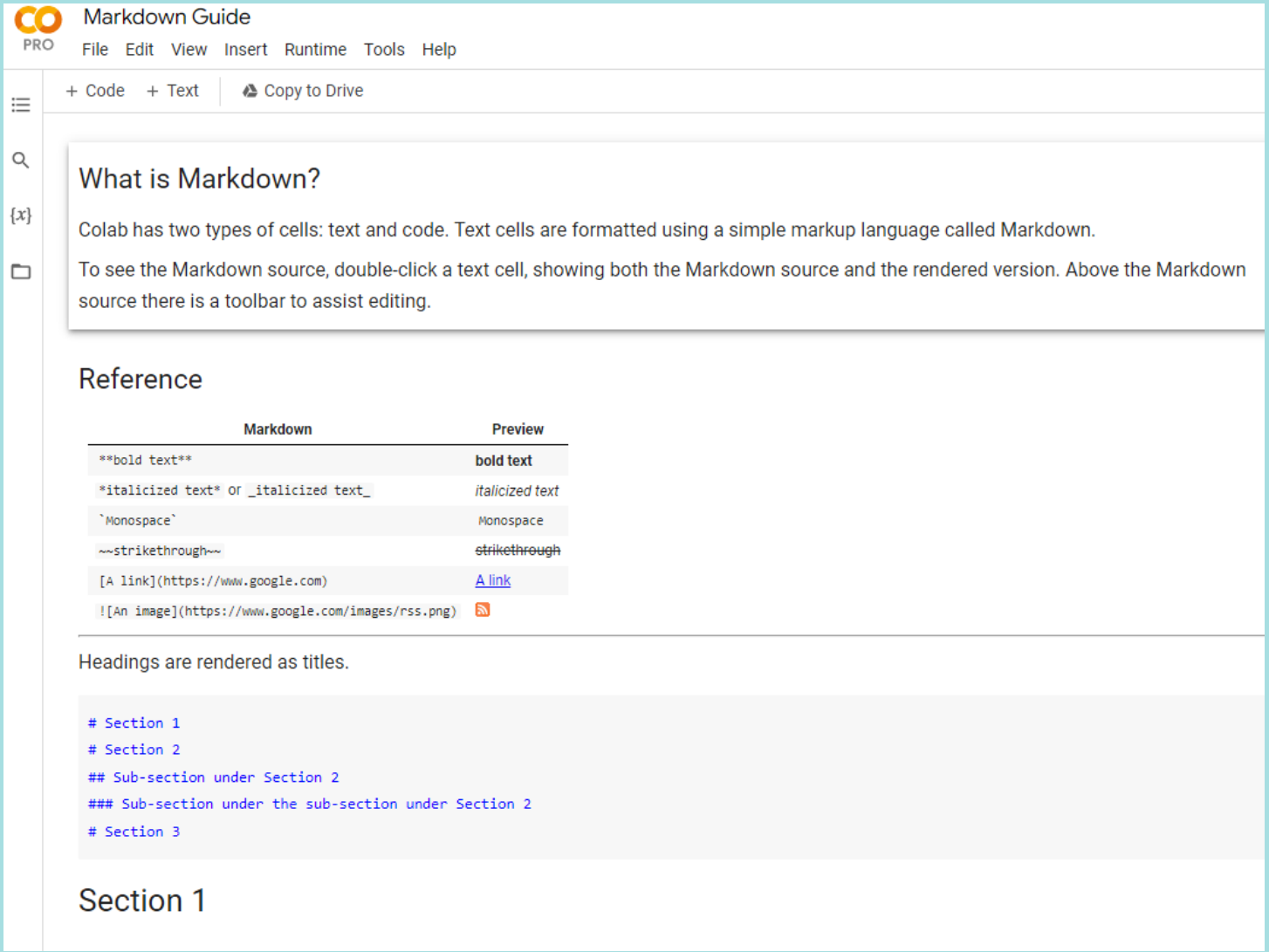
https://www.anaconda.com/products/distribution

Overview of Colab Features



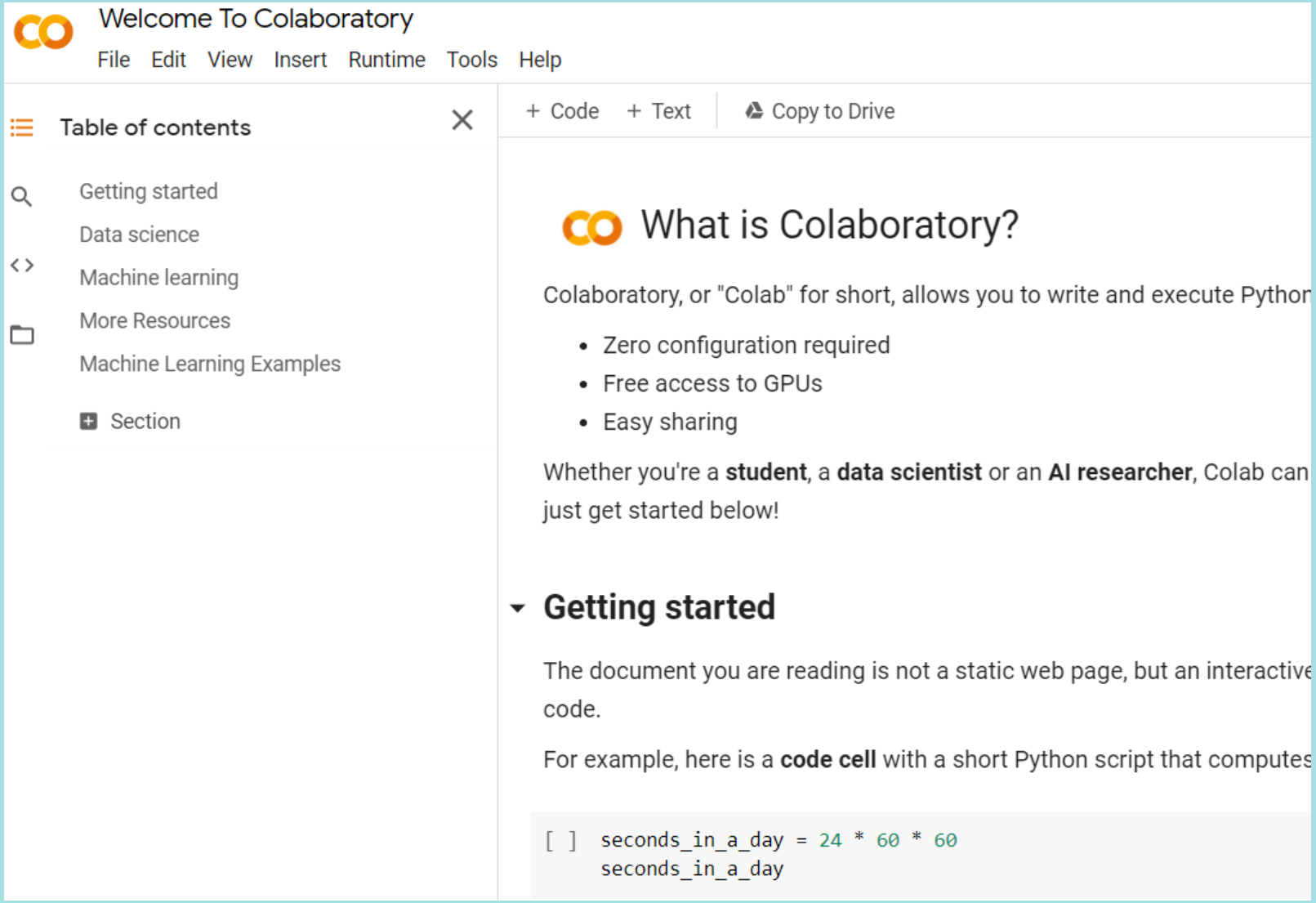
https://colab.research.google.com/notebooks/basic_features_overview.ipynb

Colab Markdown Guide



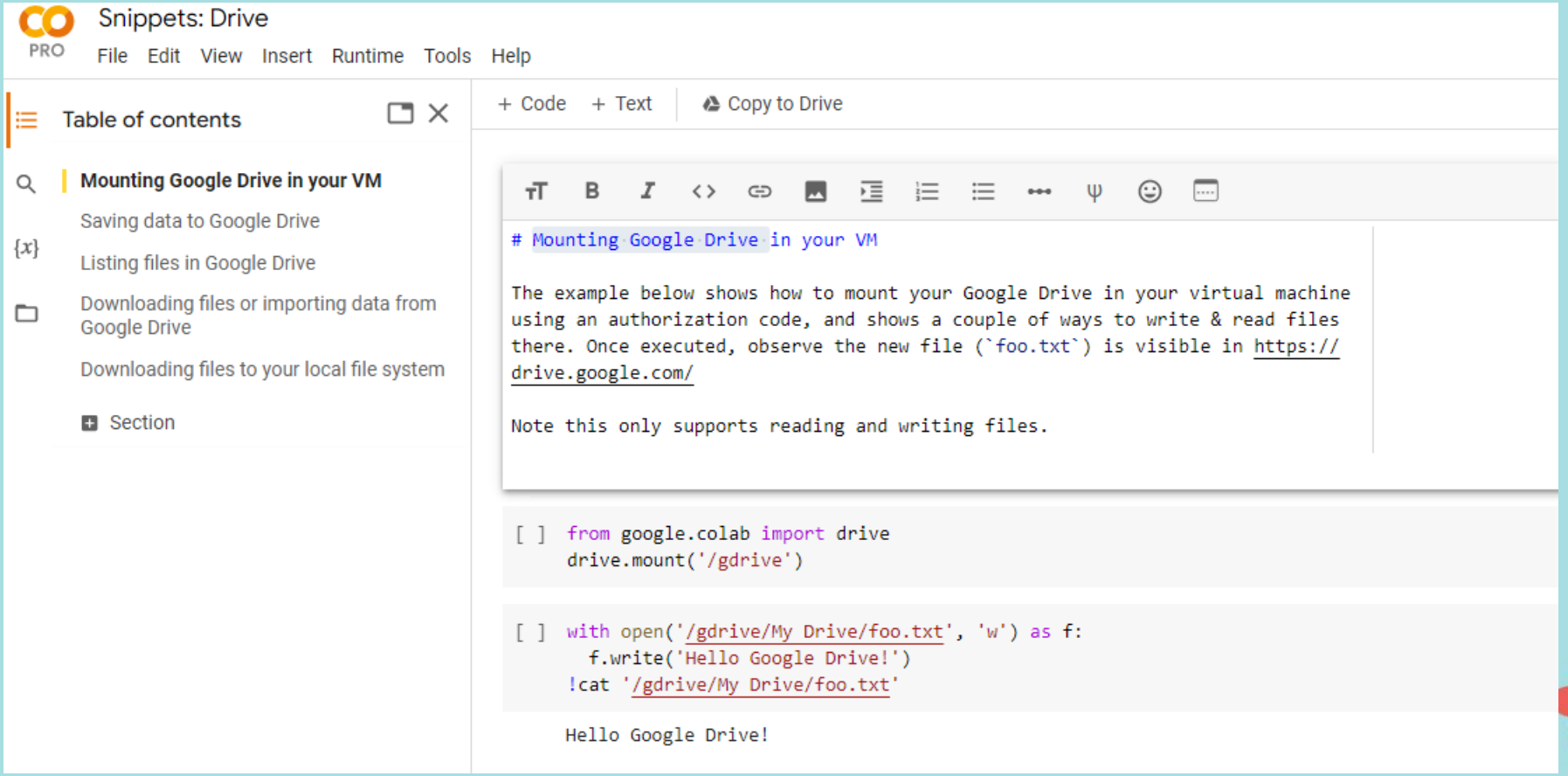
https://colab.research.google.com/notebooks/markdown_guide.ipynb

Charts in Colab



https://colab.research.google.com/notebooks/charts.ipynb

Mounting Google Drive

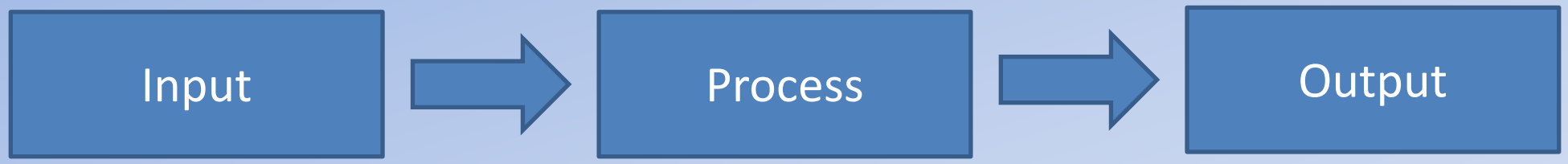


https://colab.research.google.com/notebooks/snippets/drive.ipynb?hl=en

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

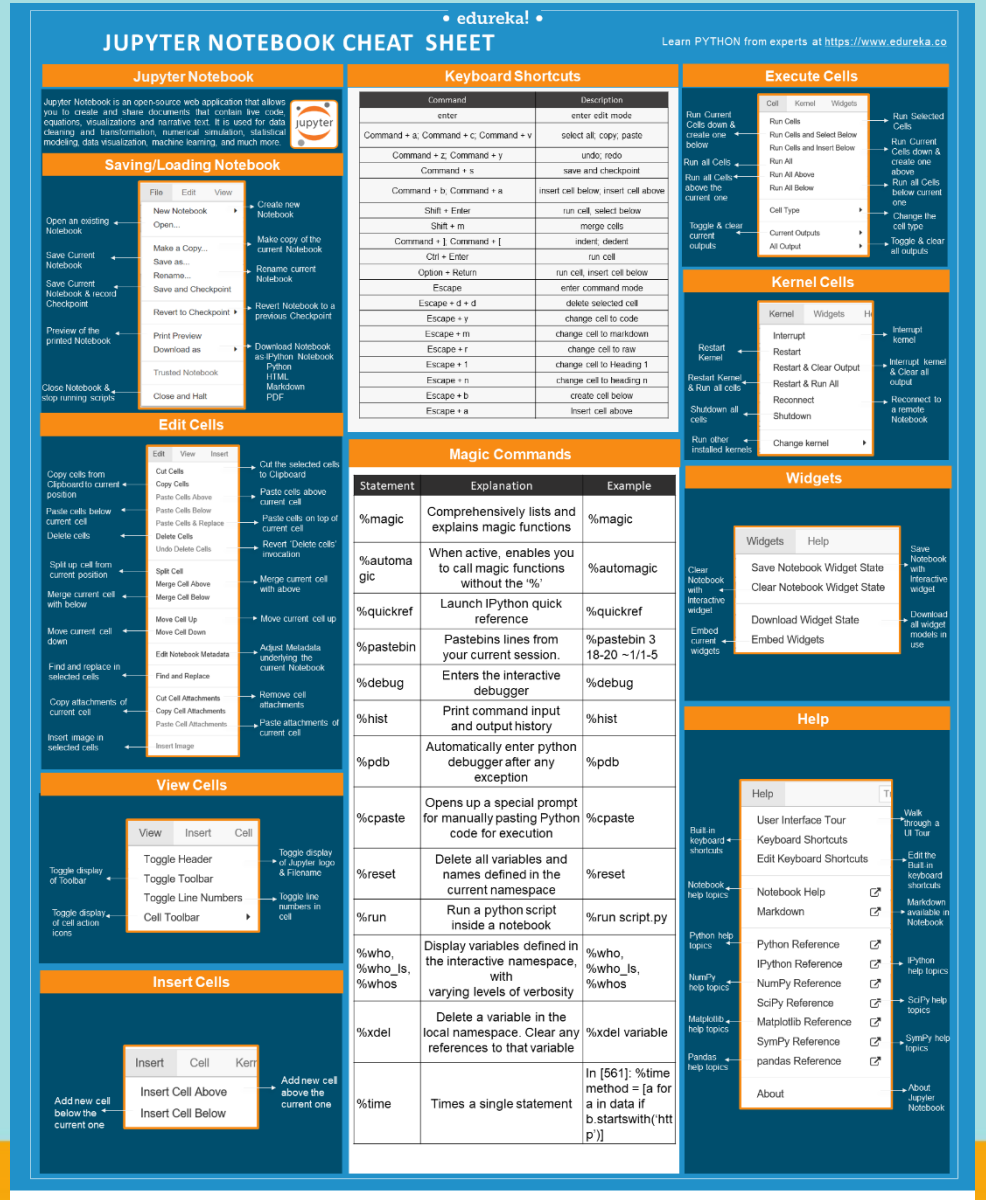



```
inFile = open('in.txt','r')
inFile.read(...)
inFile.close()
```

```
if score>60:
    count += 1
```

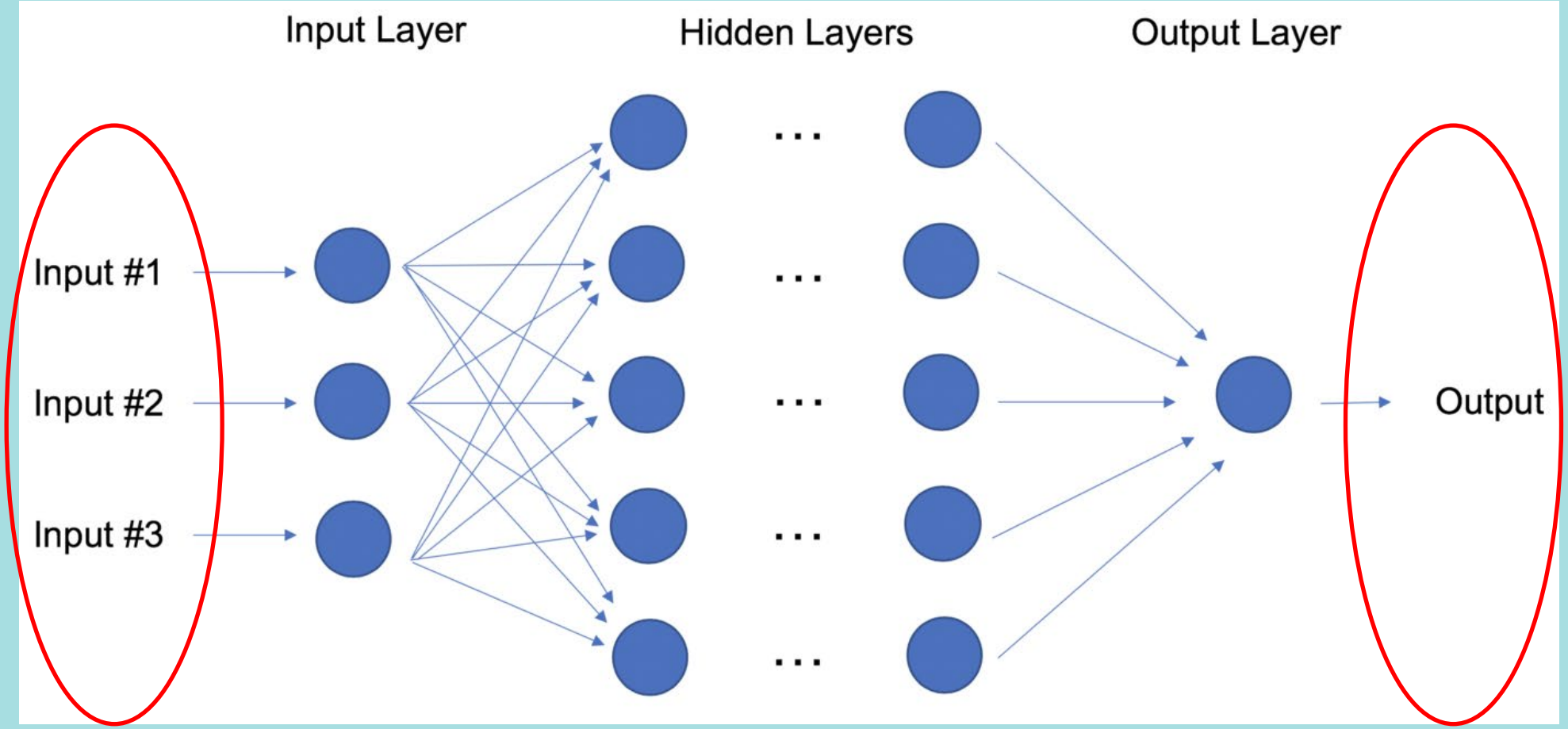
```
outFile = open('out.txt','w')
outFile.write(...)
outFile.close()
```

Jupyter Notebook Cheat Sheet



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.

https://www.kaggle.com/competitions/pkdd-15-predict-taxi-service-trajectory-i/

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