

DON'T EVER LET SOMEBODY TELL YOU,
YOU CAN'T DO SOMETHING
NOT EVEN ME, ALRIGHT?
YOU GOT A DREAM, YOU GOTTA PROTECT IT
PEOPLE CAN'T DO SOMETHING THEMSELVES
THEY WANNA TELL YOU THAT YOU CAN'T DO IT
YOU WANT SOMETHING, GO GET IT. PERIOD.



Find the slides, [here](#)

Project

has a

Structure

Don't know the structure

Not always defined

Not always consistent

Don't know the technical terms

Project structuring with cookiecutter

A command-line utility that creates projects from boilerplate project templates

[Cookiecutter Official Documentation](#)

[Cookiecutter Data Science, by drivendata](#)

[Data Science Cookiecutter Template](#) by me

Installation

```
python3 -m pip install --user cookiecutter
```

Run

```
cookiecutter <github-link-to-boilerplate -template>
```


What happens if you don't manage dependency?

1. Reproducibility Issues

[shared the same project with a friend but he is getting different results?]

2. Version Conflicts

[Have you ever tried running a famous old projects (like Mask R-CNN) from github and faced an error “**version conflict error**”]

3. Security Vulnerabilities

[unexpected errors and behaviours]

4. Reduced Productivity

[errors can be a time-consuming and frustrating process]

5. Deployment/Scalability Issues

[larger project,more packages, more dependencies to manage]

```
[tool.poetry.dependencies]
python = "^3.8"
notebook = "^6.5.3"
torch = "^1.13.1"
torchvision = "^0.14.1"
pandas = "^1.5.3"
pre-commit = "^3.1.1"
gdown = "^4.6.4"
tensorboard = "^2.12.0"
scikit-learn = "^1.2.2"
ipykernel = "^6.21.3"
Flask = "^2.2.3"
Flask-Cors = "^3.0.10"
opencv-python = "^4.7.0"
watermark = "^2.3.1"
python-dotenv = "^1.0.0"
loguru = "^0.6.0"

[tool.poetry.dev-dependencies]
pytest = "^5.2"
```

Installation

```
python3 pip install poetry
```

```
poetry --version
```

Create a local virtual environment

```
poetry config virtualenvs.in-project true
```

```
poetry config --list
```

Initialization

```
poetry init
```

Install Dependencies

```
poetry install
```

Activate the virtual environment

```
poetry shell
```

```
.venv
```

```
source .venv/bin/activate
```

```
which python >> "/home/kamal/Downloads/acm-session/.venv/bin/python"
```

```
[tool.poetry.dependencies]
python = "^3.8"
notebook = "^6.5.3"
torch = "^1.13.1"
torchvision = "^0.14.1"
pandas = "^1.5.3"
pre-commit = "^3.1.1"
gdown = "^4.6.4"
tensorboard = "^2.12.0"
scikit-learn = "^1.2.2"
ipykernel = "^6.21.3"
Flask = "^2.2.3"
Flask-Cors = "^3.0.10"
opencv-python = "^4.7.0"
watermark = "^2.3.1"
python-dotenv = "^1.0.0"
loguru = "^0.6.0"

[tool.poetry.dev-dependencies]
pytest = "^5.2"
```

Installation with pip

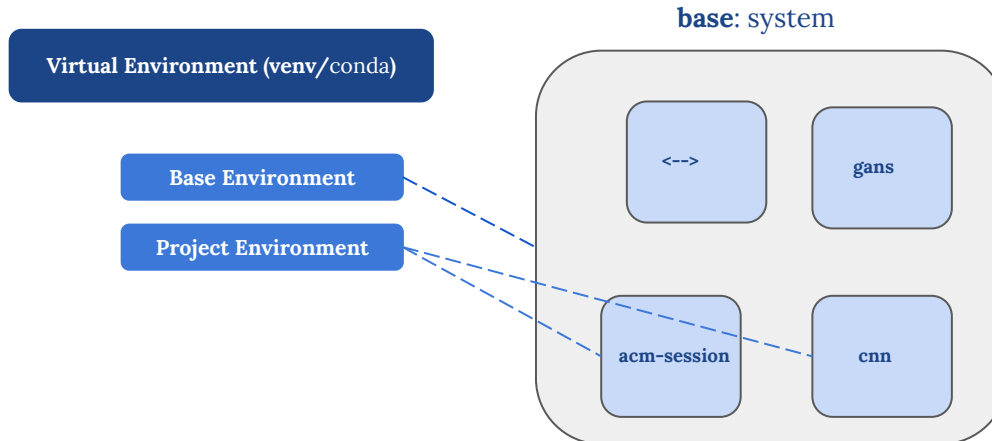
```
pip3 install numpy
```

```
poetry -version
```

Installation with poetry

```
poetry add numpy
```

```
cat pyproject.toml
```





Do you use tabs or spaces for indentation?

Do you do 4 space indentation or 2 space indentation?

Your Choice = Your Style

Food for thoughts:

1. Each organization defines their own style
[style of coding, commit messages, formatters, linters]
2. You are working on a team, remember your code is/should be read by a lot of people
3. Imagine coming to the same code after one year, would you be able to understand your code then?

PEP8 = Python Style Guide

Google Style Guide

Airbnb Style Guide (javascript)



Pre-commit Hooks

<https://pre-commit.com/>



```
from typing import List, Tuple
def find_duplicates(l):
    a = set()
    b = set()
    for i in l:
        if i in seen:
            b.add(i)
        else:
            a.add(i)
    return [(i, i) for i in b]
```

Unformatted

```
from typing import List, Tuple

def find_duplicates(lst: List[int]) -> List[Tuple[int, int]]:
    """
    Given a list of integers, returns a list of tuples representing pairs of integers that are duplicates.

    Args:
        lst (List[int]): The list of integers.

    Returns:
        List[Tuple[int, int]]: A list of tuples representing pairs of integers that are duplicates.
    """
    seen = set()
    duplicates = set()
    for num in lst:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return [(num, num) for num in duplicates]
```

Well Formatted

(naming conventions, docstrings,
indentations, line lengths, type hints)



Pre-commit Hooks

<https://pre-commit.com/>



ఐఐఐటీ హైదరాబాద్
आई आई टी हैदराबाद
IIT Hyderabad

Installation

```
poetry add pre-commit
```

```
pre-commit --version
```

Install Pre-commit Hooks

```
pre-commit install
```

Create pre-commit-config file

```
gedit .pre-commit-config.yaml >> Paste sample hooks
```

Gets activated when committing code

```
git add .
git commit -m "modify readme"
<---pre-commit gets activated now--->
git push origin master
```

```
# See https://pre-commit.com for more information
# See https://pre-commit.com/hooks.html for more hooks
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v3.2.0
  hooks:
  - id: trailing-whitespace
  - id: end-of-file-fixer
  - id: check-yaml
  - id: check-added-large-files
    args: ["--maxkb=10000"]
  - id: check-json
- repo: https://github.com/psf/black
  rev: d9b8a6407e2f46304a8d36b18e4a73d8e0613519
  hooks:
  - id: black
- repo: https://github.com/PyCQA/isort
  rev: 3a72e069635a865a92b8a0273aa829f630cbcd6f
  hooks:
  - id: isort
```

Pre-commit are some checks (format, filesize, imports ...) that you run before pushing your code into Github

<https://pre-commit.com/hooks.html>

Sample:

<https://gist.github.com/shresthakamal/34ff61afaf742828d83f58485f0cf76d>



Pre-commit Hooks

<https://pre-commit.com/>



```
~/Downloads/acm-session master +2 !2 > git add .                                acm-session 12:21:36
~/Downloads/acm-session master +2 > git commit -m "function change"           acm-session 12:21:37
Trim Trailing Whitespace.....Passed
Fix End of Files.....Passed
Check Yaml.....Passed
Check for added large files.....Passed
Check JSON.....(no files to check)Skipped
black.....Passed
isort.....Passed
[master 5bd5579] function change
 2 files changed, 31 insertions(+)
 create mode 100644 .pre-commit-config.yaml
~/Downloads/acm-session master >                                           acm-session 12:21:40
```

Exploratory Data Analysis (EDA) flourishes with jupyter notebook

1. Cleaning Data
2. Visualizing Data
3. Extracting meaningful insights

Interactivity (widgets, user inputs, grades)

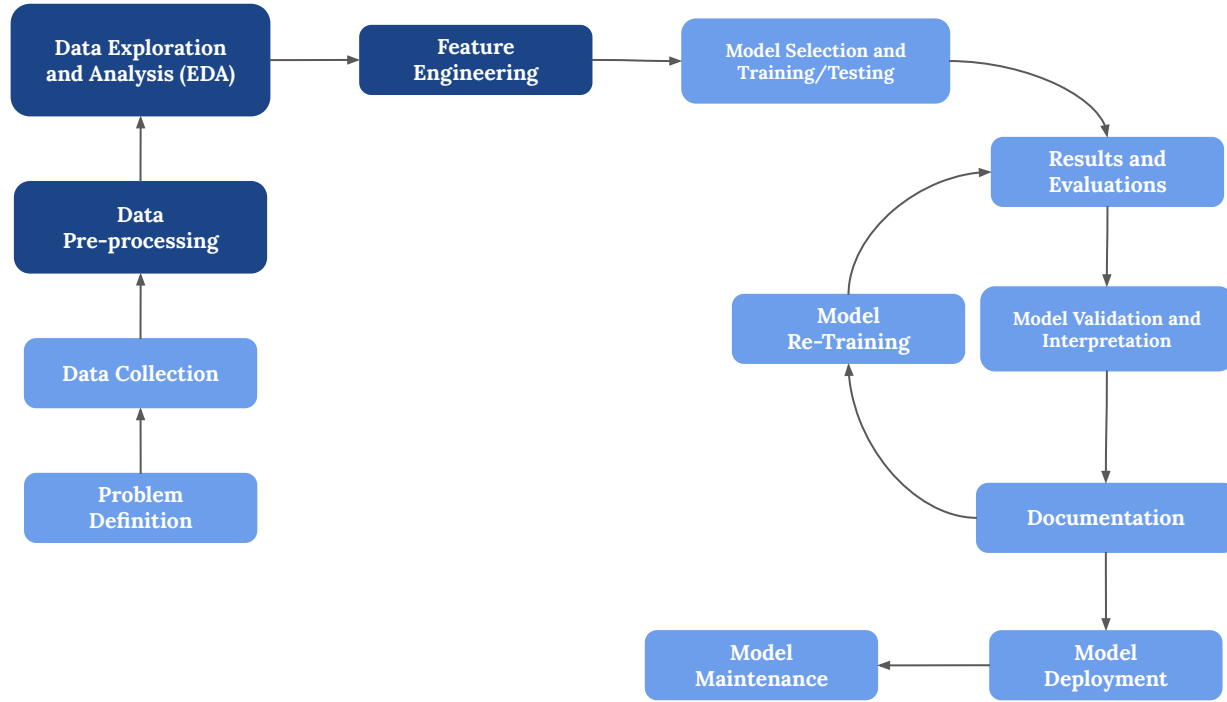
Documentation (markdown with code)

Easy Data Manipulation

Quick Prototyping

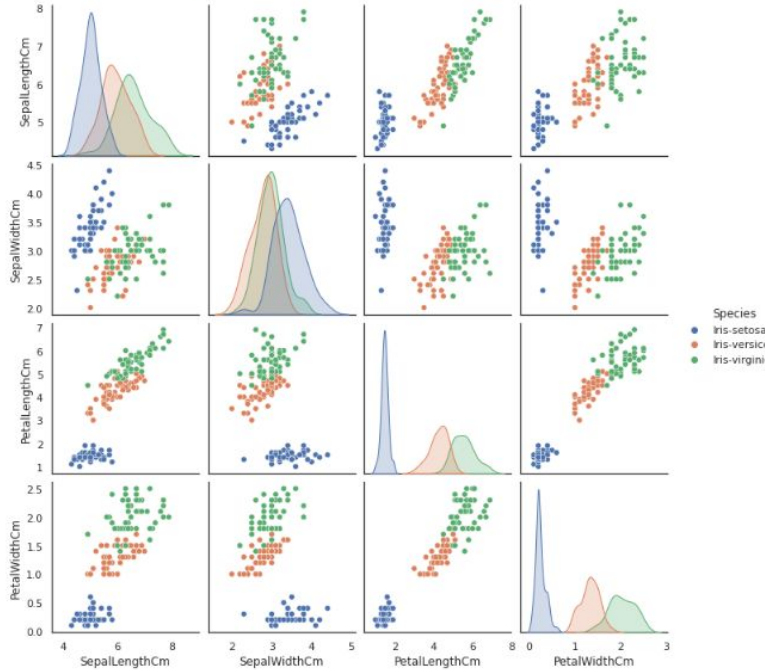
Online Courses (Automatic Code graders)

[nbgrader]



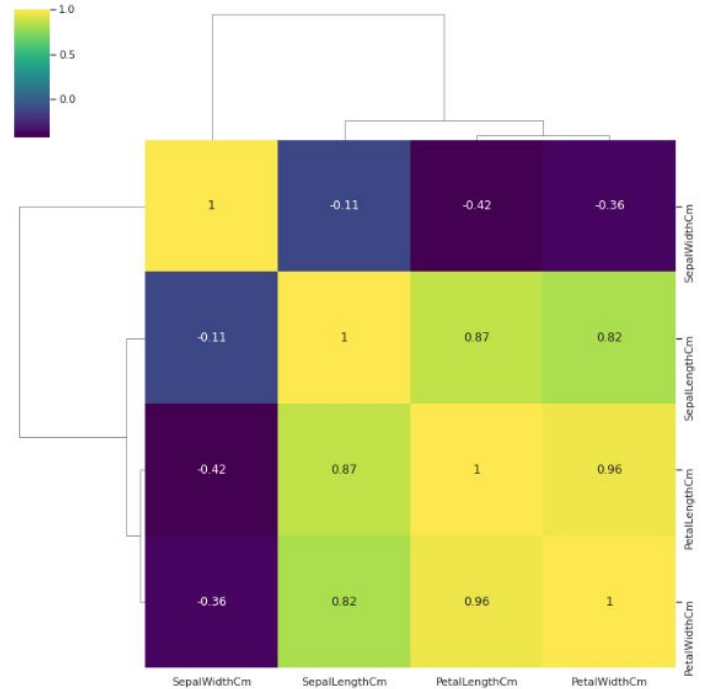

```
In [159]: sns.pairplot(iris, hue ="Species")
```

```
Out[159]: <seaborn.axisgrid.PairGrid at 0x7f0fbd542240>
```



```
In [162]: sns.clustermap(corr, annot=True,cmap='viridis')
```

```
Out[162]: <seaborn.matrix.ClusterGrid at 0x7f0fbd540208>
```



Initialization

1. Create a filename called "Makefile"
2. Add `keywords:commands pair`

WHY LOGGING?

- Logging multiple experiment Hyperparameters**
 [Did you ever run multiple experiments for an hour but forgot what parameters are used and had to re-run all the experiment?]
- Unexpected Stoppage**
 [What happens while you are training your code, but suddenly power cuts off or the server gets disconnected?,
 What are you currently doing now?]
- Debugging and troubleshooting**
 [Helps you track down errors, from files, line numbers, on each steps]
- Compliance**
 [Logging can also be required for compliance reasons, such as to meet regulatory or legal requirements]

```

logs > 2023-03-29_16-12-45_628472.log
1 2023-03-29 16:12:45 | INFO | __main__:train:28) - Using: cuda
2 2023-03-29 16:12:46 | INFO | __main__:train:52) - Model:
3 nnGAN(
4 (generator): Sequential(
5 (0): Linear(in_features=75, out_features=128, bias=True)
6 (1): LeakyReLU(negative_slope=0.01, inplace=True)
7 (2): Dropout(p=0.5, inplace=False)
8 (3): Linear(in_features=128, out_features=784, bias=True)
9 (4): Tanh()
10 )
11 (discriminator): Sequential(
12 (0): Linear(in_features=784, out_features=128, bias=True)
13 (1): LeakyReLU(negative_slope=0.01, inplace=True)
14 (2): Dropout(p=0.5, inplace=False)
15 (3): Linear(in_features=128, out_features=1, bias=True)
16 (4): Sigmoid()
17 )
18 )
19 2023-03-29 16:12:47 | INFO | __main__:train:111) - Epoch: 001/010 | Batch 000/469
20 2023-03-29 16:12:48 | INFO | __main__:train:111) - Epoch: 001/010 | Batch 100/469
21 2023-03-29 16:12:50 | INFO | __main__:train:111) - Epoch: 001/010 | Batch 200/469
22 2023-03-29 16:12:52 | INFO | __main__:train:111) - Epoch: 001/010 | Batch 300/469
23 2023-03-29 16:12:53 | INFO | __main__:train:111) - Epoch: 001/010 | Batch 400/469
24 2023-03-29 16:12:55 | INFO | __main__:train:123) - Time elapsed: 0.16 min
25 2023-03-29 16:12:55 | INFO | __main__:train:111) - Epoch: 002/010 | Batch 000/469
26 2023-03-29 16:12:56 | INFO | __main__:train:111) - Epoch: 002/010 | Batch 100/469
27 2023-03-29 16:12:58 | INFO | __main__:train:111) - Epoch: 002/010 | Batch 200/469
28 2023-03-29 16:13:00 | INFO | __main__:train:111) - Epoch: 002/010 | Batch 300/469
29 2023-03-29 16:13:01 | INFO | __main__:train:111) - Epoch: 002/010 | Batch 400/469
30 2023-03-29 16:13:03 | INFO | __main__:train:123) - Time elapsed: 0.29 min
31 2023-03-29 16:13:03 | INFO | __main__:train:111) - Epoch: 003/010 | Batch 000/469
32 2023-03-29 16:13:04 | INFO | __main__:train:111) - Epoch: 003/010 | Batch 100/469
33 2023-03-29 16:13:06 | INFO | __main__:train:111) - Epoch: 003/010 | Batch 200/469
34 2023-03-29 16:13:08 | INFO | __main__:train:111) - Epoch: 003/010 | Batch 300/469
35 2023-03-29 16:13:10 | INFO | __main__:train:111) - Epoch: 003/010 | Batch 400/469
36 2023-03-29 16:13:11 | INFO | __main__:train:123) - Time elapsed: 0.43 min
37 2023-03-29 16:13:11 | INFO | __main__:train:111) - Epoch: 004/010 | Batch 000/469
38 2023-03-29 16:13:13 | INFO | __main__:train:111) - Epoch: 004/010 | Batch 100/469
39 2023-03-29 16:13:15 | INFO | __main__:train:111) - Epoch: 004/010 | Batch 200/469
40 2023-03-29 16:13:17 | INFO | __main__:train:111) - Epoch: 004/010 | Batch 300/469
  
```

Installation

```
poetry add loguru
```

```
loguru --version
```

Initialization

```
from logger import logger

log_format = <define_your_log_format>

# print logs in terminal
logger.add("logs.log", format = log_format)

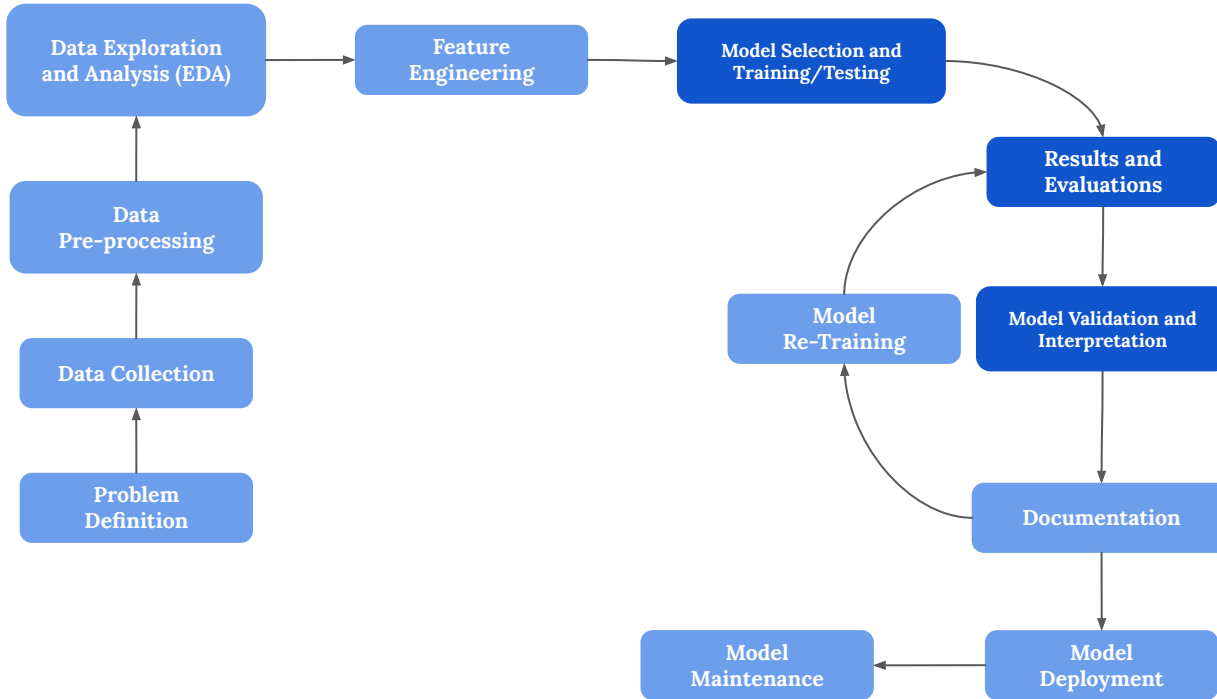
# print logs in a file
logger.add("sys.stderr", format=log_format)
```

Usage: **simply remove print with logger.info()**

```
# Before
print("Time elapsed: %.2f min" % ((time.time() - start) / 60))

# After
logger.info("Time elapsed: %.2f min" % ((time.time() - start) / 60))
```

Web based real-time visualization tool for tracking, plotting, visualizing scalars, graphs, models and more..

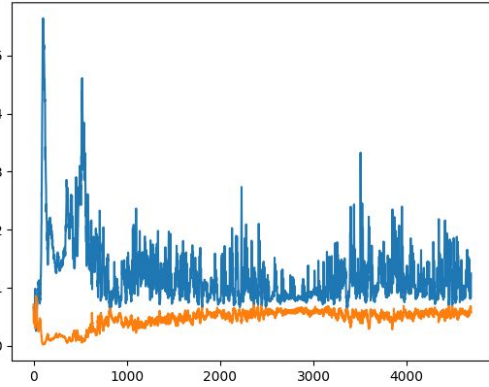


Model Training and Evaluations

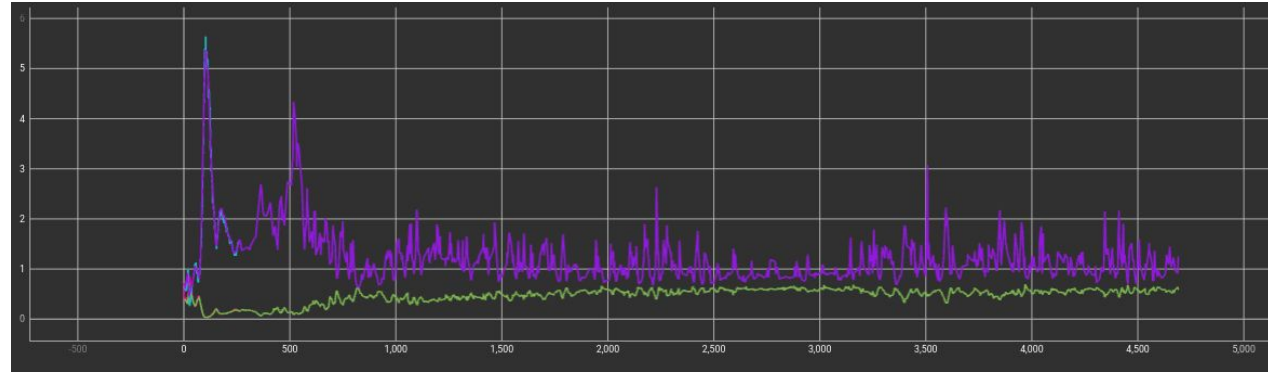
[Plotting the following metrics using matplotlib]

1. Accuracy
2. Training and Testing Losses
3. F1 Scores
4. ...

Web based real-time visualization tool for tracking, plotting, visualizing scalars, graphs, models and more..



matplotlib



tensorboard

Installation

```
poetry add tensorboard
```

```
tensorboard --version
```

Initialization

```
from torch.utils.tensorboard import SummaryWriter
```

```
# Before
```

```
losses = []  
losses.append(loss)
```

```
# After
```

```
writer = SummaryWriter("runs/")  
writer.add_scalar(y=loss, x=epochs)
```

Launch tensorboard

```
tensorboard -logdir=runs/
```


tmux is a **terminal multiplexer**.

It lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal.

Problem:

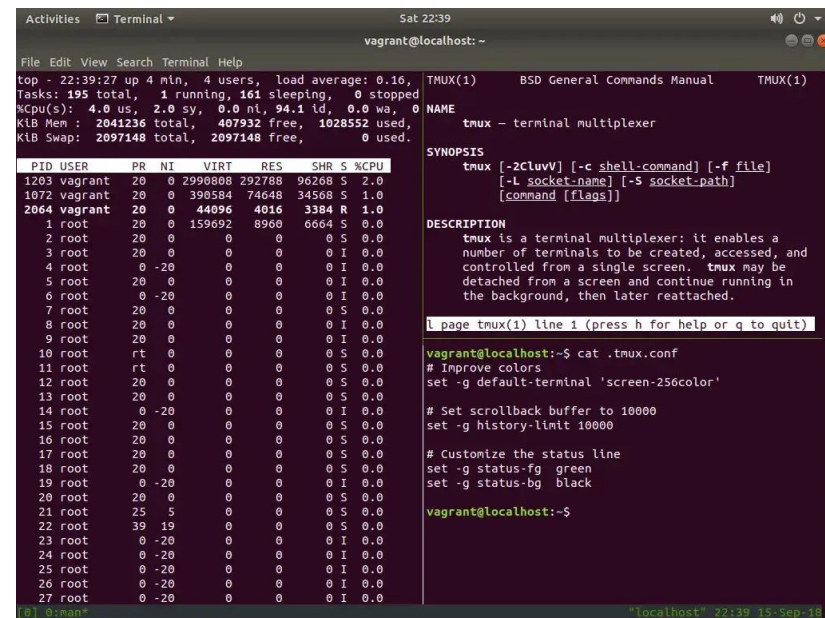
Have you ever waited hours for a model training to complete so that you can get the results?

Have you waited for one experiment to complete so that you can run the next one?

Solution:

What if I say, you can leave training your model on the server, close your laptop and come back later wards?

tmux with SSH



```

Activities Terminal Sat 22:39 vagrant@localhost: ~
File Edit View Search Terminal Help
top - 22:39:27 up 4 min, 4 users, load average: 0.16,
Tasks: 195 total, 1 running, 161 sleeping, 0 stopped
%Cpu(s): 4.0 us, 2.0 sy, 0.0 ni, 94.1 id, 0.0 wa, 0
KlB Mem : 2041236 total, 407932 free, 1028552 used,
KlB Swap: 2097148 total, 2097148 free, 0 used.
PID USER PR NI VIRT RES SHR S %CPU
1203 vagrant 20 0 2990808 292788 96268 S 2.0
1072 vagrant 20 0 390584 74648 34508 S 1.0
2064 vagrant 20 0 44096 4016 3384 R 1.0
1 root 20 0 159692 8960 6664 S 0.0
2 root 20 0 0 0 0 S 0.0
3 root 20 0 0 0 0 I 0.0
4 root 0 -20 0 0 0 I 0.0
5 root 20 0 0 0 0 I 0.0
6 root 0 -20 0 0 0 I 0.0
7 root 20 0 0 0 0 S 0.0
8 root 20 0 0 0 0 I 0.0
9 root 20 0 0 0 0 I 0.0
10 root rt 0 0 0 0 S 0.0
11 root rt 0 0 0 0 S 0.0
12 root 20 0 0 0 0 S 0.0
13 root 20 0 0 0 0 S 0.0
14 root 0 -20 0 0 0 I 0.0
15 root 20 0 0 0 0 S 0.0
16 root 20 0 0 0 0 S 0.0
17 root 20 0 0 0 0 S 0.0
18 root 20 0 0 0 0 S 0.0
19 root 0 -20 0 0 0 I 0.0
20 root 20 0 0 0 0 S 0.0
21 root 25 5 0 0 0 S 0.0
22 root 39 19 0 0 0 S 0.0
23 root 0 -20 0 0 0 I 0.0
24 root 0 -20 0 0 0 I 0.0
25 root 0 -20 0 0 0 I 0.0
26 root 0 -20 0 0 0 I 0.0
27 root 0 -20 0 0 0 I 0.0
tmux(1) BSD General Commands Manual tmux(1)
NAME
tmux - terminal multiplexer
SYNOPSIS
tmux [-2Cluv] [-c shell-command] [-f file]
[-L socket-name] [-S socket-path]
[command [flags]]
DESCRIPTION
tmux is a terminal multiplexer: it enables a
number of terminals to be created, accessed, and
controlled from a single screen. tmux may be
detached from a screen and continue running in
the background, then later reattached.
1 page tmux(1) line 1 (press h for help or q to quit)
vagrant@localhost:~$ cat .tmux.conf
# Improve colors
set -g default-terminal 'screen-256color'
# Set scrollbar buffer to 10000
set -g history-limit 10000
# Customize the status line
set -g status-fg green
set -g status-bg black
vagrant@localhost:~$
    
```




Thanks!

Any **questions** ?

You can find me at:

- Kamal Shrestha

cs21mtech16001@iith.ac.in

shresthakamal.com.np/home



Feedbacks