

End-to-end Python Practice on Industrial Data Engineering

葉信和 / Hsin-Ho Yeh
Software Engineer / CEO @ 信誠金融科技
hsinho.yeh@footprint-ai.com



2022/06/01

Download Slides

<https://reurl.cc/p1pg8Q>



About me

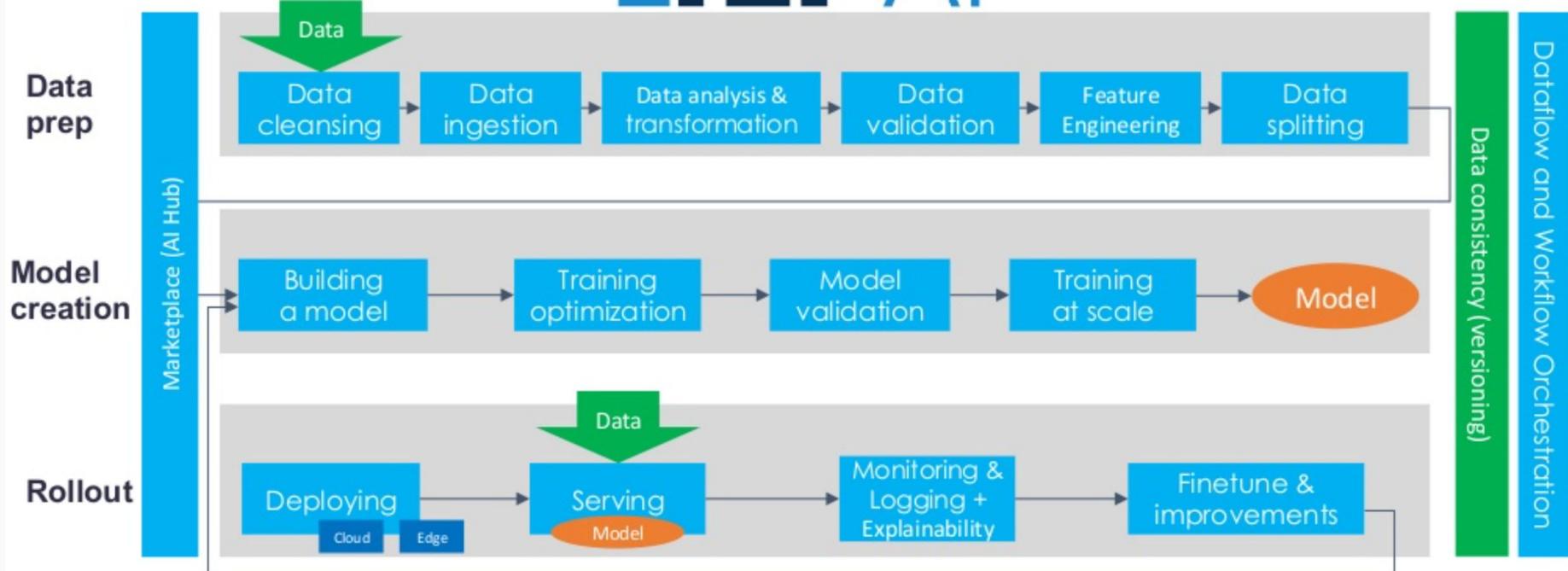
- 2020 - Present at 信誠金融科技
 - Shrimping: A data-sharing platform
 - <https://get-shrimping.footprint-ai.com>
 - Tintin: a machine learning platform for everyone
 - <https://get-tintin.footprint-ai.com>
- 2016 - 2020 at Igloolnsure (16M+ in series A+ 2020)
 - Provide digital insurance for e-economic world
 - Funded in KUL, Headquartered in Singapore
 - First employee/ Engineering Lead / Regional Head/ Chief Engineer
- 2013 - 2016 at Studio Engineering @ hTC
 - Principal Engineer on Cloud Infrastructure Team
- 2009 - 2012 at IIS @ Academia Sinica
 - Computer vision, pattern recognition, and data mining
- CS@CCU, CS@NCKU alumni



Agenda

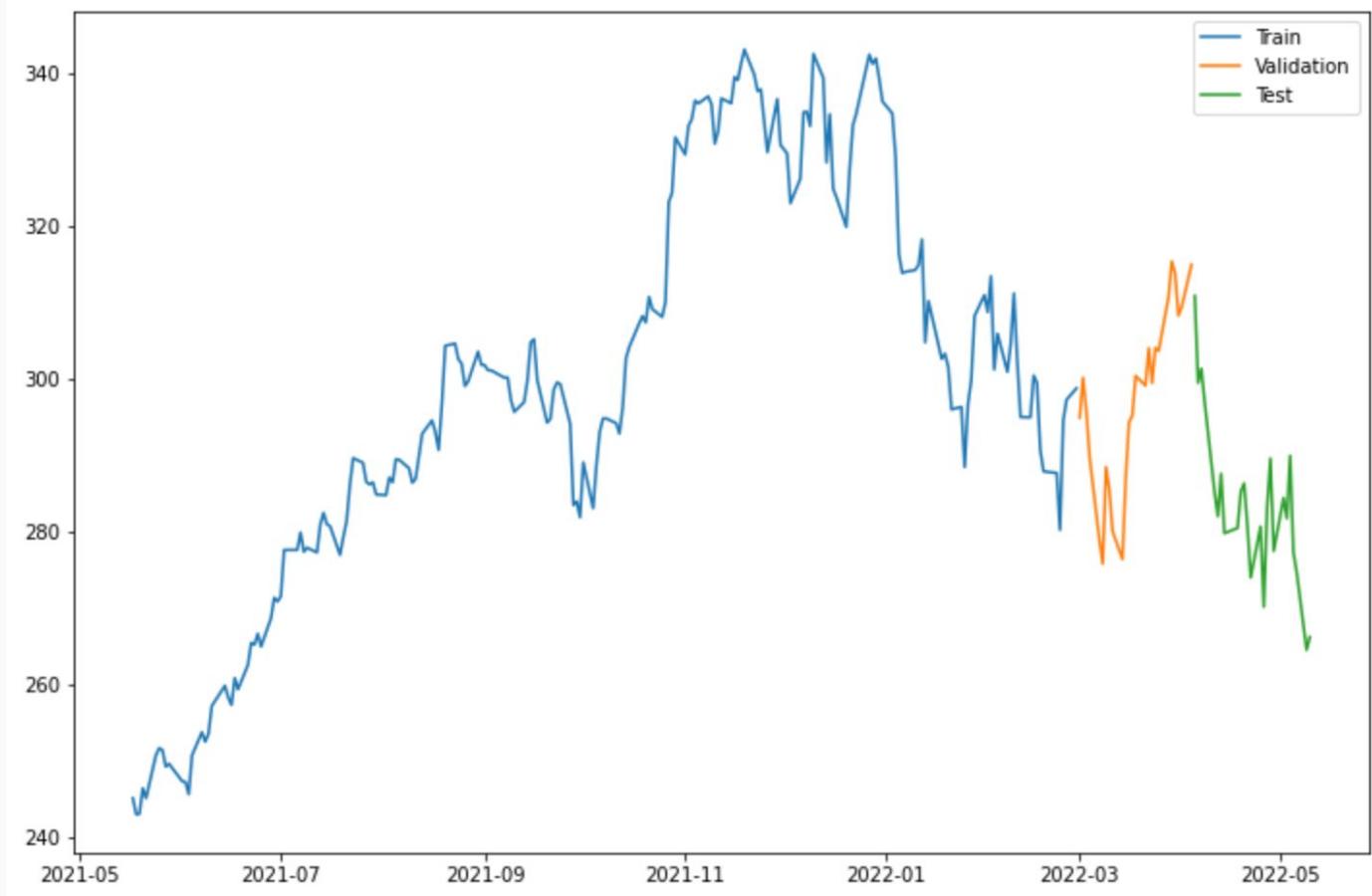
- End-to-end flow on industrial data analysis practice
- How and where do we collect data?
- Web scraping with selenium
- Data Analysis with Pandas
- Build a financial model with Kubeflow
- Q&A

Real-world Machine Learning Application - End-to-End ML LifeCycle



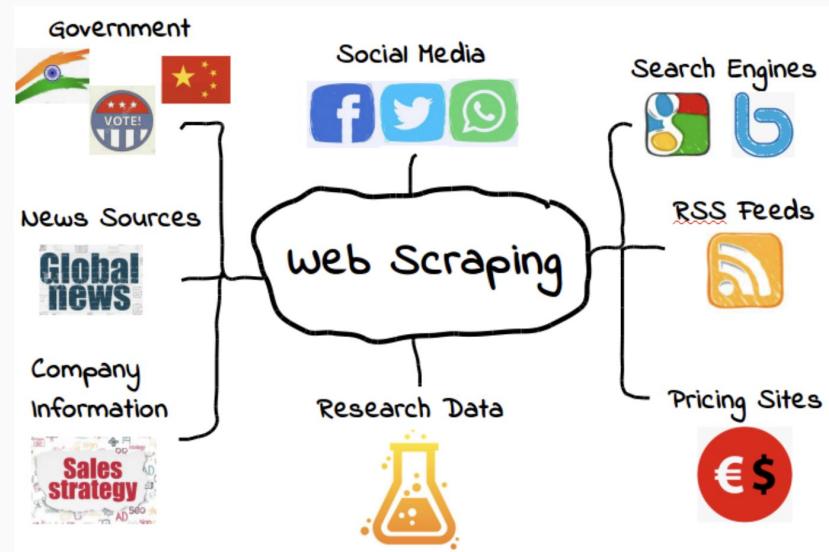
Source: <https://www.slideshare.net/AnimeshSingh/advanced-model-inferencing-leveraging-kubeflow-serving-knative-and-istio-196096385>

How to predict stock price for the next day?



What is Web Scraping?

- Web Scraping is an **automation** process of collecting **structure** data from public websites.
- Common Use Cases including
 - Competitor price monitoring on E-commerce
 - News monitoring from social network



Different ways of collecting data from websites (1/3)

- Manually (slow & slow & slow)



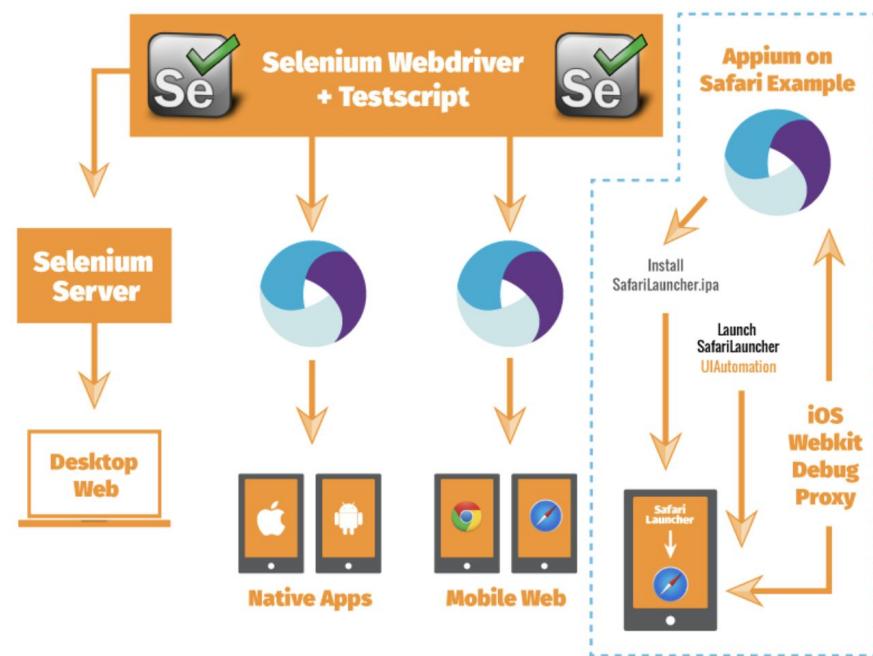
Different ways of collecting data from websites (2/3)

- API Requests with Python(Stable & Fast but not always works...)

```
>>> import requests  
  
>>> r = requests.get('https://www.google.com.tw/')  
  
>>> print(r.status_code)  
200  
  
>>>print(r.text)  
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"  
lang="zh-TW"><head><meta content="text/html; charset=UTF-8"  
http-equiv="Content-Type"><meta  
content="/images/branding/googleg/1x/googleg_standa....
```

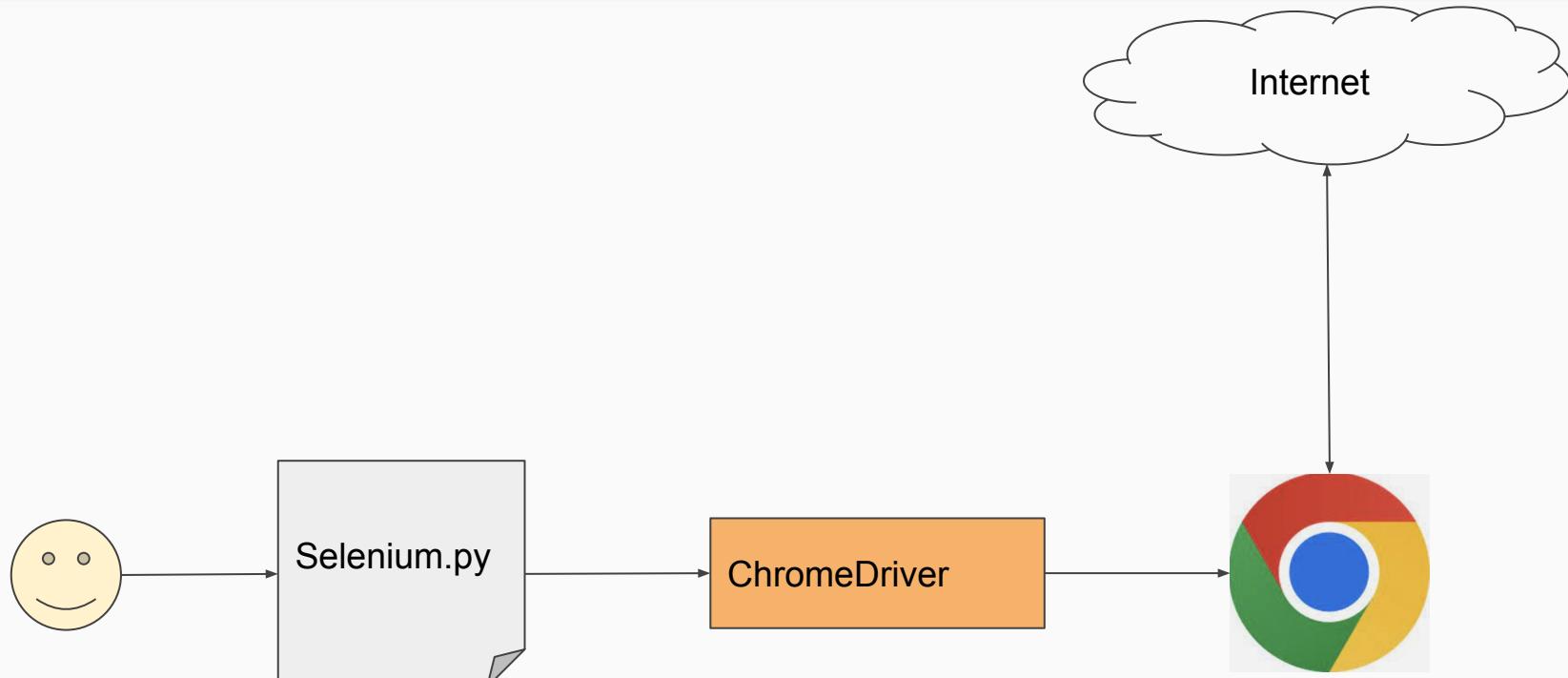
Different ways of collecting data from websites (3/3)

- UI Automation with Selenium
 - Selenium is an web automation tool which allows you to automate UI testing for different browsers via its webdriver.
 - Webdriver is used to control each Browser behaviors making it more like real-human interaction.
- Because of its nature of automation and mimic human behavior, making it a good human-like web-scraping



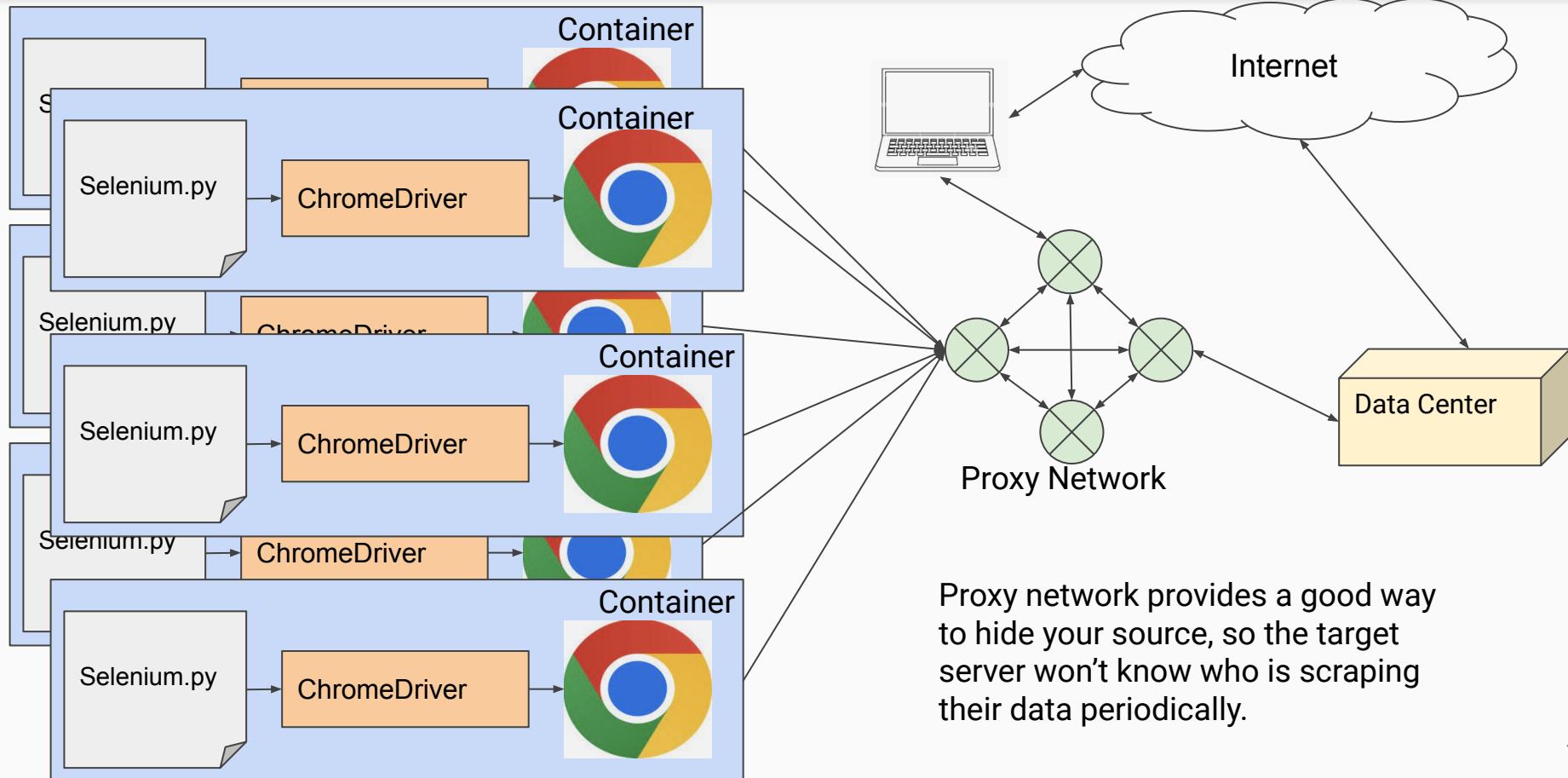
Ref: <https://smartbear.com/blog/selenium-cross-browser-testing-on-mobile-devices/>

How Selenium WebDriver Works?



ChromeDriver relies on the pre-installed chrome package to send the requests and render the HTML tags

How Selenium WebDriver Works In Scale?



Selenium Examples

<https://github.com/FootprintAI/selenium-example>

The screenshot shows a GitHub repository page for 'FootprintAI / selenium-example'. The repository is public and has 1 branch and 0 tags. The main file list includes 'hsinhyoyeh remove', 'drivers', 'examples', 'Dockerfile', 'LICENSE', 'README.md', and 'chromedriver'. On the right side, there is a 'Code' dropdown menu with options for 'Clone' via HTTPS, SSH, or GitHub CLI, and links for 'Open with GitHub Desktop' and 'Download ZIP'.

File	Description
hsinhyoyeh remove	
drivers	add headless as options
examples	remove
Dockerfile	add Dockerfile
LICENSE	Initial commit
README.md	Initial commit
chromedriver	add ig/goodinfo example

Selenium: Basic Example

```
1 from drivers.webdriver import WebDriver
2
3 from selenium.webdriver.common.by import By
4 from selenium.webdriver.common.keys import Keys
5
6 def do(driver: WebDriver, q: str):
7     driver.get("http://www.python.org")
8     assert "Python" in driver.title
9     elem = driver.find_element(By.NAME, "q")
10    elem.clear()
11    elem.send_keys(q)
12    elem.send_keys(Keys.RETURN)
13    assert "No results found." not in driver.page_source
```

By is for element selection
Key is for mapping keyboard

Selenium: Webdriver Inheritance & encapsulation

```
1 import os
2 import platform
3 from selenium.webdriver import Chrome, ChromeOptions
4
5 class WebDriver(Chrome):
6     def __init__(self, headless:bool=False):
7         exec_path = None
8         if platform.system() == "Darwin":
9             exec_path = self.__wrap_abspath('chromedriver.darwin')
10            elif platform.system() == "Windows":
11                exec_path = self.__wrap_abspath('chromedriver.exe')
12                elif platform.system() == "Linux":
13                    exec_path = self.__wrap_abspath('chromedriver.linux')
14
15        chrome_options = ChromeOptions()
16        # comment this to enable guest browser to show up (for debugging)
17        if headless:
18            chrome_options.add_argument('--headless')
19            chrome_options.add_argument('--no-sandbox')
20
21        Chrome.__init__(self,chrome_options=chrome_options, executable_path=exec_path)
22
23    def __wrap_abspath(self, binary)-> str:
24        return os.path.join(os.path.dirname(os.path.abspath(__file__)), binary)
```

inheritance

encapsulation

Selenium: Cookies & Navigation

```
1  from drivers.webdriver import WebDriver
2
3  from time import sleep
4
5  def navigation(driver: WebDriver):
6      driver.get("https://www.google.com")
7      sleep(1)
8      driver.get("https://www.python.org")
9      sleep(1)
10     driver.back()
11     sleep(1)
12     driver.forward()
13
14 def get_cookies(driver: WebDriver):
15     driver.get("https://www.google.com")
16     print(driver.get_cookies())
```

Selenium: Exceptions

```
1 from drivers.webdriver import WebDriver
2
3 def exception_handling(driver: WebDriver):
4     from selenium.common.exceptions import TimeoutException, NoSuchElementException
5     try:
6         driver.get("https://www.google.com")
7     except TimeoutException:
8         print("time out")
9     try:
10         driver.find_element_by_id("you-should-not-see-me")
11     except NoSuchElementException:
12         print("no element found")
```

Selenium: query methods

Tag

```
<input type="text" name="passwd" id="passwd-id" />
```

Attribute

```
// we have four different ways to locate an individual UI element
```

```
element = driver.find_element(By.ID, "passwd-id")
element = driver.find_element(By.NAME, "passwd")
element = driver.find_element(By.XPATH, "//input[@id='passwd-id']")
element = driver.find_element(By.CSS_SELECTOR, "input#passwd-id")
```

DOM(Document Object Model) Tree

An example of the DOM

Let's start with the following simple document:

```
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4   <title>About elk</title>
5 </head>
6 <body>
7   The truth about elk.
8 </body>
9 </html>
```

The DOM represents HTML as a tree structure of tags. Here's how it looks:



Ref: <https://javascript.info/dom-nodes>

Example: Instagram Login

```
0
7     driver.get("https://www.instagram.com")
8     sleep(5) # wait for the page loading is finished
9     # find username fields
10    username_field=driver.find_element_by_css_selector("input[name='username']")
11    # find password fields
12    password_field=driver.find_element_by_css_selector("input[name='password']")
13    username_field.clear()
14    password_field.clear()
15    # send email in username field
16    username_field.send_keys(email)
17    # send password in password field
18    password_field.send_keys(password)
19    sleep(5) # sleep 5 seconds
20
21    # click login button
22    driver.find_element_by_css_selector("button[type='submit']").click()
23    __post_login(driver)
24
25 def __post_login(driver: WebDriver):
26
```

Example: Find Stock Price

```
1  from drivers.webdriver import WebDriver
2
3  from selenium.webdriver.common.by import By
4  from selenium.webdriver.support.ui import WebDriverWait
5  from selenium.webdriver.support import expected_conditions as EC
6
7  def do_find(driver: WebDriver, symbol: str):
8      driver.get("https://goodinfo.tw/StockInfo/index.asp")
9      ele = driver.find_element_by_xpath(u"//*[@id='txtStockCode']")
10     ele.clear()
11     ele.send_keys(symbol)
12     driver.find_element_by_xpath(u"//input[@value='股票查詢']").click()
13
14     # the following code will wait until the element is visible
15     try:
16         wait = WebDriverWait(driver, 30)
17         wait.until(EC.visibility_of_element_located((By.XPATH, u"//*[@id='imgKC']")))
18     except:
19         raise Exception("target element is not show up")
```

You have a scraping, we have lots of anti-Scraping tools.

- How to bypass Anti-Scraping Tool
 - Keep Rotate Your IP Address
 - By a well-paid proxy service provider
 - By restricting a certain area of your location
 - Use Real User Agent and valid Referrer
 - UA(User-Agent) are http header which is used identify what browser type you used to visit this website.
 - Keep a list of valid UI and use them randomly.
 - Avoid Periodically Requests
 - Keep random intervals between requests, random delay are helpful.
 - Update your code frequently
 - Anti-scraping tool would change DOM structure frequently, making your script failed to find target elements.
- However, Implementing these guidelines could costly if your application is still below a certain scale.

Shrimping: A data-sharing platform

Shrimping provides a unified way for clients to get human-centric information in a simple, easy, and low cost fashion.

<https://get-shrimping.footprint-ai.com/>

Web scraping scenario for Shrimping.

- UI Validation
 - When working with business partners, it is extremely important that your partner has interpreted your product correctly.
- Collecting sell history on an ecommerce platform
 - Track the sell volume of all products on an eCommerce platform.
 - As the number of products could be big (approximately 100M active products), how to get each product's sell records on daily basis is extremely challenging.
- Scraping and analyzing KOL's feeds on social network platforms
 - Find out a KOL and his/her fans preference for retargeting, reselling, or other marketing strategies.
 - Social network platform always implemented anti-bot mechanism, making it hard to collect in a large scale fashion.

Data Analysis with Panda

Think like table, work like table...

- Pandas is a Python library and the de-facto standard for working with **structured** tabular data on Python
- Rich Format Supported including CSV, JSON, Parquet, MS EXCEL, ...

asset	max_training	max_validation	mean_training	mean_validation	med_training	med_validation	min_training	min_validation	std_training	std_validation
AAN	2.685913	5.132591	0.046193	0.029751	0.120043	0.237710	-4.324665	-7.955100	1.356045	2.851172
ACC	1.546164	3.473945	0.024481	-0.191317	0.067734	0.000000	-1.493781	-8.192803	0.662752	2.181609
ACGL	1.225804	3.676009	0.035951	-0.350968	0.072673	0.000000	-1.788876	-6.347304	0.607838	1.949640
ADC	1.343051	1.567481	0.118847	-0.090484	0.152312	0.001203	-1.302874	-4.310633	0.603213	1.131439
AEL	3.002969	5.464704	0.026192	-0.094789	0.076017	0.000000	-3.330113	-7.611560	1.290956	2.659290
AFG	1.095037	3.558017	0.005339	-0.387737	0.072613	-0.060772	-2.050200	-7.330084	0.614609	2.013369
AFL	1.159893	3.050265	0.051159	-0.295960	0.072861	-0.038611	-1.414419	-5.840748	0.514807	1.709449
AGM	2.189914	2.572398	0.119186	-0.179405	0.130402	-0.031308	-2.258110	-5.378916	0.944030	1.627170
AGNC	0.934225	2.371371	-0.016914	-0.196734	0.026154	0.159754	-1.373769	-6.580712	0.481505	1.653974

Example of a simple Pandas table with “assets” as index and various numerical columns

Pandas Operations

```
Persons = pd.read_csv('person.csv') # load data

# Projection
Persons[['age', 'height']]

# Filtering
Persons[Persons['age'] > 21]

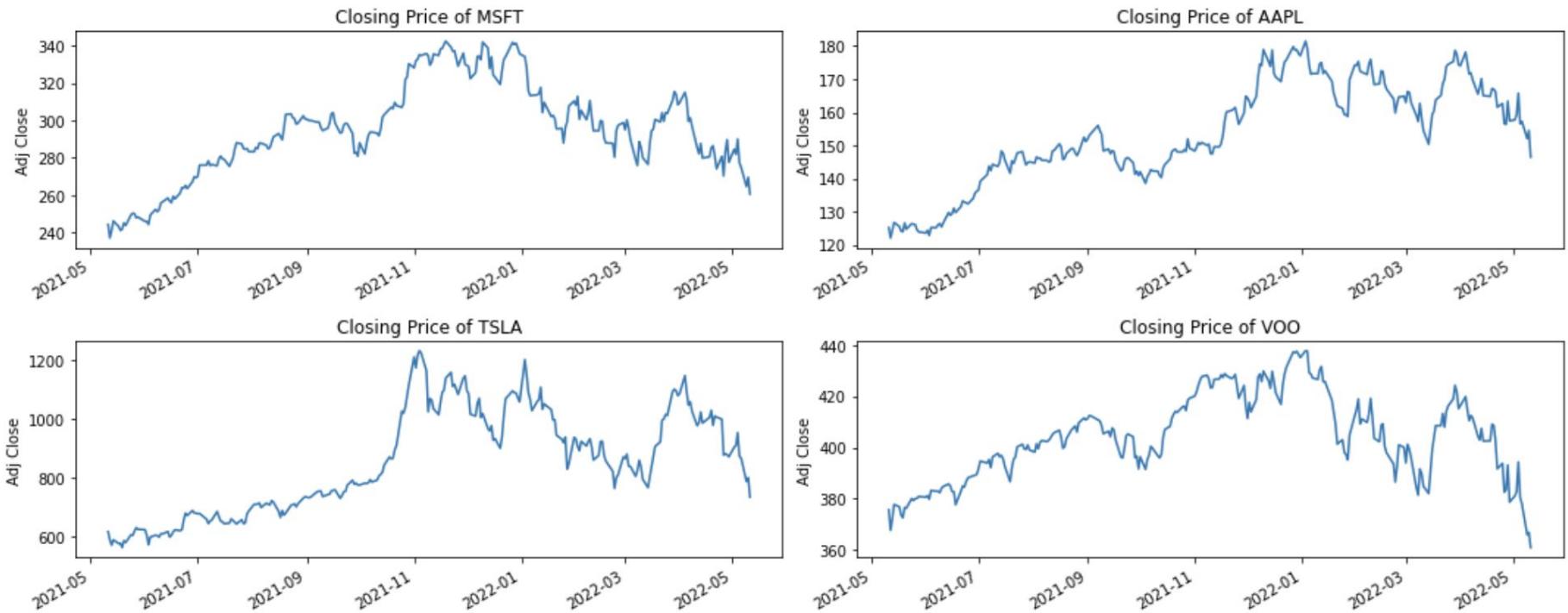
# Join
Persons.join(addresses, on='name')

# Concatenation
Persons2 = pd.read_csv('person2.csv')
pd.concat([Persons, Persons2])
```

city	
name	
Alice	Hamburg
Bob	Frankfurt
Henry	Berlin

	age	height	name	sex
0	23	156	Alice	female
1	21	181	Bob	male
2	27	176	Charlie	male
3	24	167	Eve	female
4	19	172	Frances	female
5	31	191	George	female

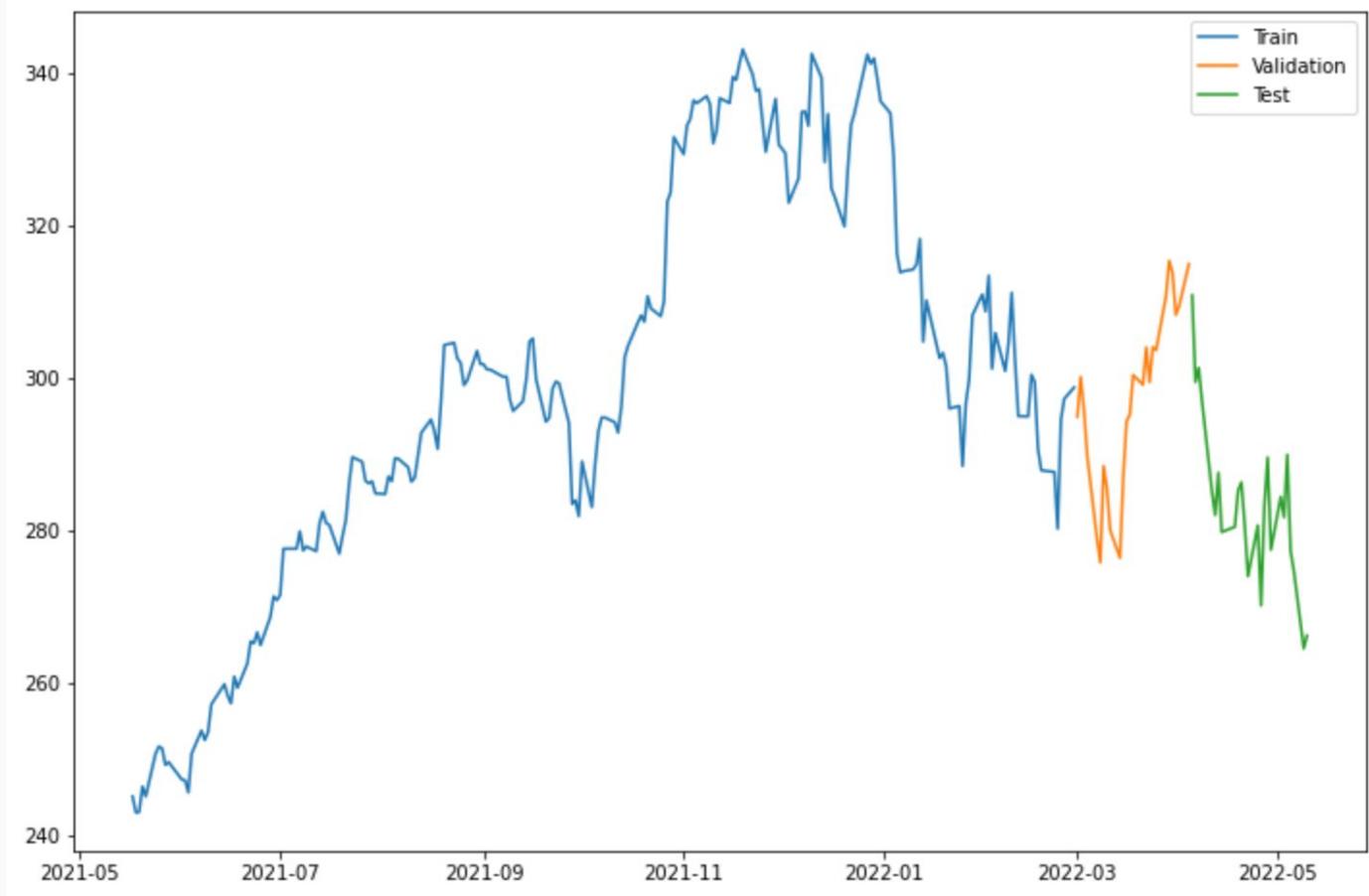
Data Visualization



Financial Model Analysis with Kubeflow

Can LSTM model be used to predict
the stock price?

How to predict stock price for the next day?



What sequences means in a series?

Input: [300, 320, 310, 350, 390]

t0	300
t1	320
t2	310
t3	350
t4	390
t5	?

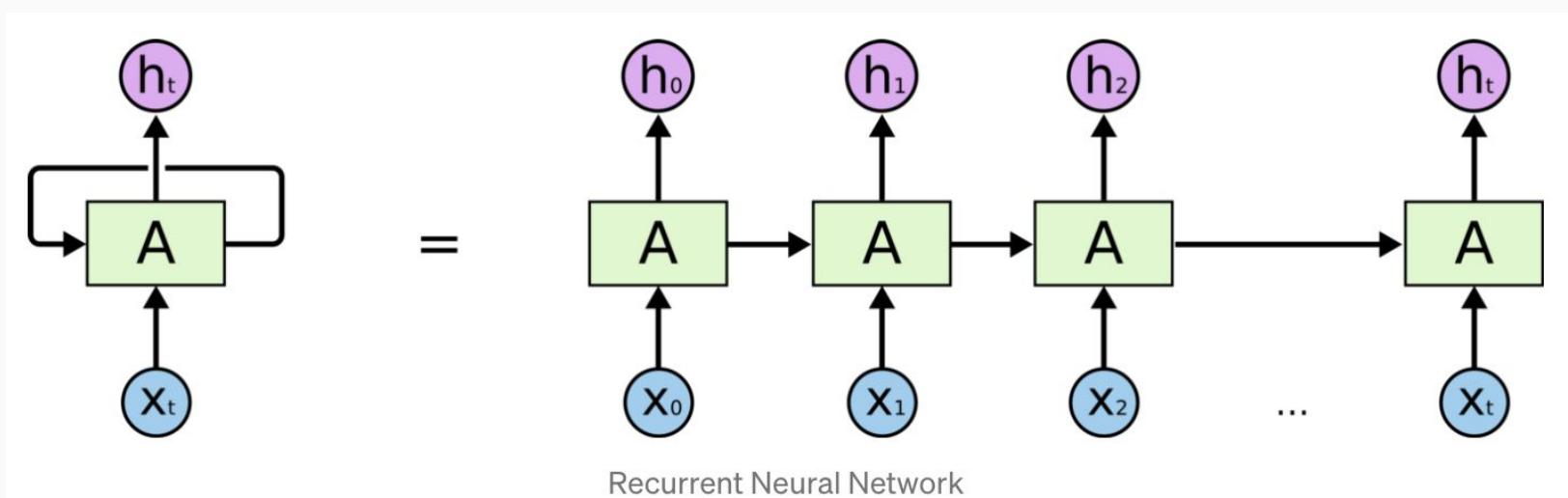
What sequences means in a sentence?

Input: I am a good guy

t0	I
t1	am
t2	a
t3	good
t4	guy
t5	?

What Is RNN (Recurrent Neural Network)?

Recurrent Neural Network (RNN) takes decisions on CURRENT (X_t) and PREVIOUS (X_{t-1}) inputs. Especially useful in topics including machine translation, speech recognition.

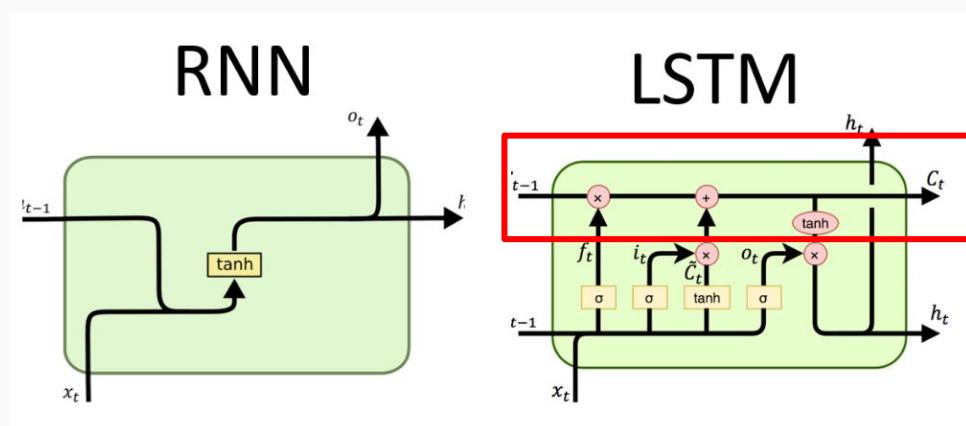


What Is LSTM (Long-short term memory)?

RNN only remember the latest things from X and it didn't remember(no memory) anything before at the beginning.

LSTM provides an information highway to let the neuron to selectively choose

1. forget from its memory (focus on the current inputs)
2. Listens to what information it added into memory (though information highway)



<https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3>

<https://www.quora.com/How-is-LSTM-different-from-RNN-In-a-layman-explanation>

How to formulate a sequence into a trainable dataset?

t0	t1	t2	t3	t4
300	320	310	350	390

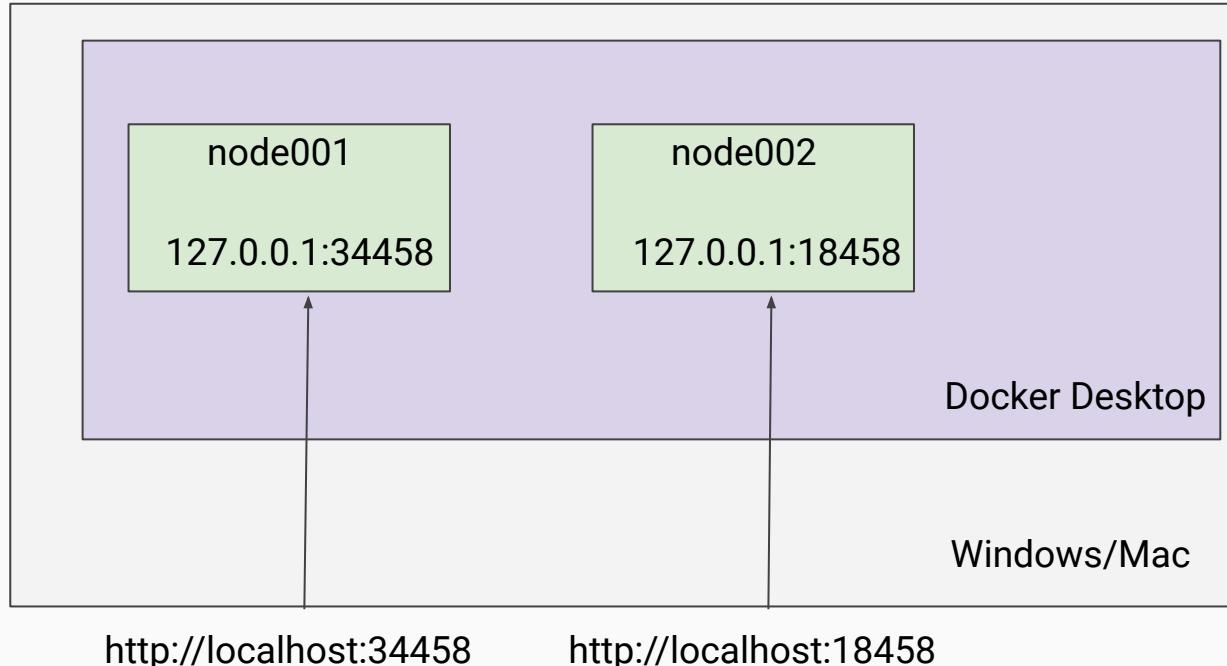
Windows size 3, we looks last three days data

Feature 1	Feature 2	Feature 3	Target
300	320	310	350
320	310	350	390

Environment Setup

Deployment Overview

Provisioning with multikf <https://github.com/FootprintAI/multikf/tree/main/docs>



Deployment Overview: few steps setup

```
// install dockerd (windows)  
https://github.com/FootprintAI/kubeflow-workshop/blob/main/install/windows/dockerd.bat.md
```

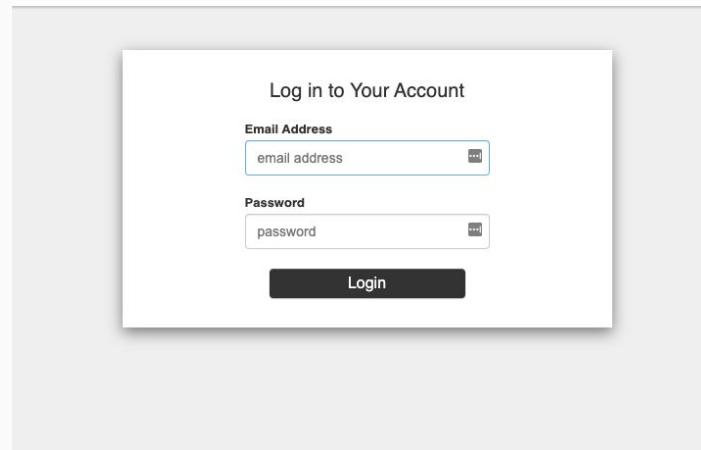
```
// install multikf (windows)  
wget https://github.com/FootprintAI/multikf/raw/main/build/multikf.windows.exe  
chmod +x multikf.windows.exe
```

```
// add an instances with port 80/443 exported  
.multikf.windows.exe add node002 --export_ports 80:80,443:443
```

```
// connect kubeflow  
.multikf.windows.exe connect kubeflow node002
```

Wait! 所以我說那個帳號密碼呢?

Account: user@example.com
Password: 12341234



Step1: Use Notebook as Online IDE (1/3)

The screenshot shows the Kubeflow web interface. On the left, a sidebar menu is visible with the following items:

- Home
- Notebooks** (highlighted with a red box)
- Tensorboards
- Volumes
- Models
- Experiments (AutoML)
- Experiments (KFP)
- Pipelines
- Runs
- Recurring Runs
- Artifacts
- Executions

At the bottom of the sidebar, there is a link to "Privacy • Usage Reporting" and a note about the build version: "build version dev_local".

The main content area is titled "Notebooks" and displays a table with columns: Status, Name, Type, Age, Image, GPUs, CPUs, Memory, and Volumes. There is no data in the table.

A large red arrow points upwards from the "+ NEW NOTEBOOK" button, which is located in the top right corner of the main content area. The "+ NEW NOTEBOOK" button is also highlighted with a red box.

Step1: Use Notebook as Online IDE (2/3)

The screenshot shows the Kubeflow interface for creating a new Notebook Server. On the left is a sidebar with various options: Home, Notebooks (highlighted), Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. Below these are Manage Contributors and Privacy & Help links.

The main area is titled "Specify the name of the Notebook Server and the Namespace it will belong to." It contains two input fields: "Name" (containing "demo") and "Namespace" (containing "kubeflow-user-example-com").

Below this is a section for the "Image". It includes a "Custom Image" checkbox (unchecked), a dropdown menu showing "jupyterlab" selected (with options 1 and 2), and a detailed view of the "jupyterlab" entry. This view shows the image URL "j1r0q0g6/notebooks/notebook-servers/jupyter-tensorflow-full:v1.4".

Under "Advanced Options", there is a "CPU / RAM" section. It asks to "Specify the total amount of CPU and RAM reserved by your Notebook Server. For CPU-intensive workloads, you can choose more than 1 CPU (e.g. 1.5)." Two input fields are shown: "Requested CPU" (containing "0.5") and "Requested Memory (in GiB)" (containing "1").

A large red box highlights the "Name" field. Two red arrows point from the text "Specify a name and resources like CPU and Memory" to the "Requested CPU" and "Requested Memory" fields respectively.

Specify a name and resources like CPU and Memory

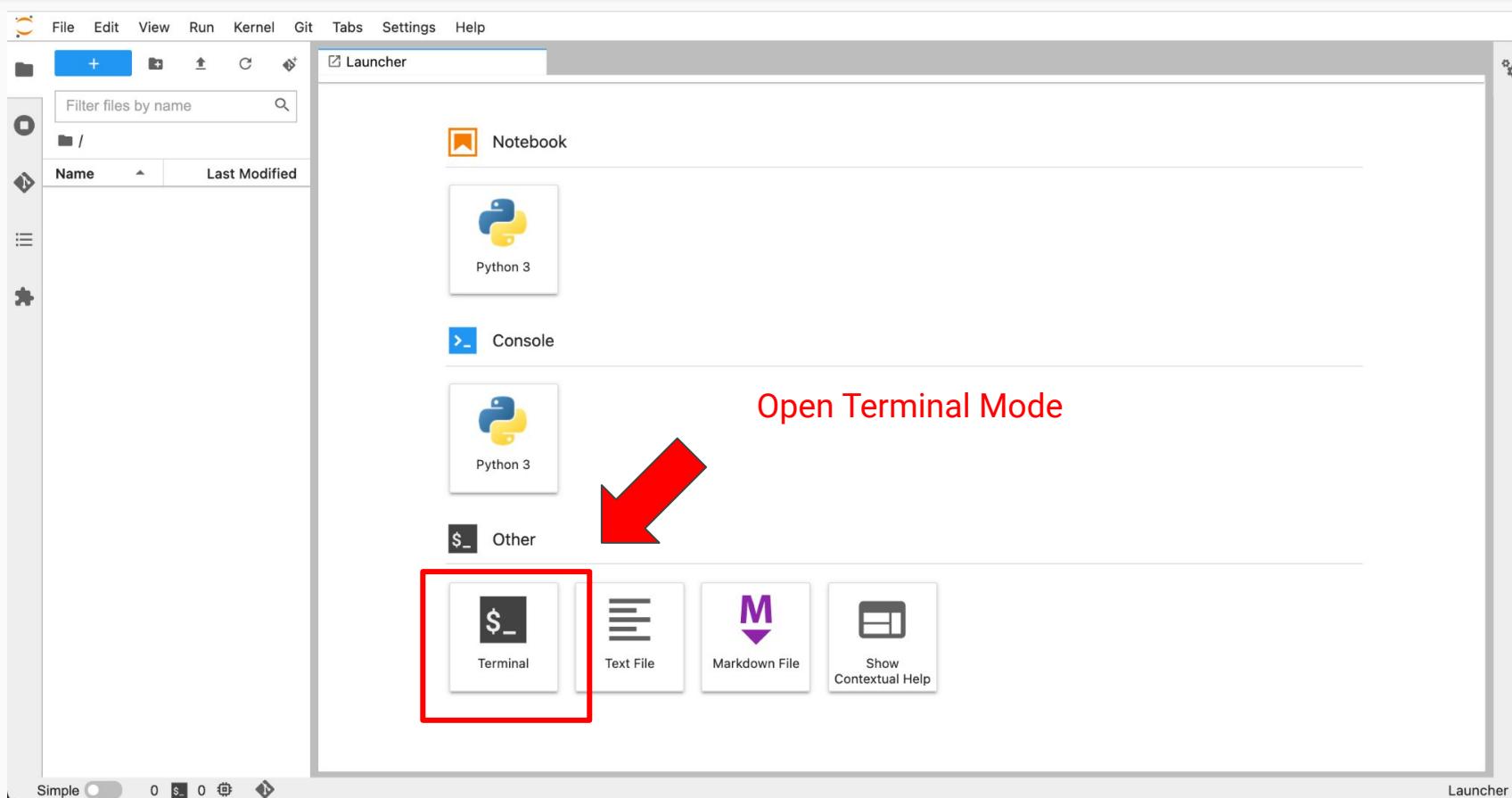
Step1: Use Notebook as Online IDE (3/3)

The screenshot shows the Kubeflow interface with the 'Notebooks' section selected. Two notebooks are listed:

Status	Name	Type	Age	Image	GPUs	CPUs	Memory	Volumes	⋮	CONNECT	⋮	⋮
✓	demo1	jupyter	20 hours ago	jupyter-scipy:v1.4	0	0.5	1Gi		⋮	CONNECT	⋮	⋮
✓	demo2	jupyter	2 hours ago	jupyter-tensorflow-full:v1.4	0	0.5	1Gi		⋮	CONNECT	⋮	⋮

A large red arrow points upwards from the bottom right towards the 'CONNECT' button for the second notebook, demo2.

Step2: Use terminal to download the materials (1/3)



Step2: Use terminal to download the materials (2/3)

The screenshot shows the Jupyter Notebook interface. On the left is a file browser sidebar. In the center is a terminal window titled "Terminal 1" containing a command-line session. A red box highlights the command "git clone https://github.com/footprintai/kubeflow-workshop". A large red arrow points upwards from a code block below towards this highlighted command. The code block contains the same command: "git clone https://github.com/footprintai/kubeflow-workshop".

```
groups: cannot find name for group ID 1222
(base) jovyan@dem01-0:~$ git clone https://github.com/footprintai/kubeflow-workshop
Cloning into 'kubeflow-workshop'...
remote: Enumerating objects: 164, done.
remote: Counting objects: 100% (164/164), done.
remote: Compressing objects: 100% (98/98), done.
remote: Total 164 (delta 89), reused 133 (delta 62), pack-reused 0
Receiving objects: 100% (164/164), 1.71 MiB | 8.29 MiB/s, done.
Resolving deltas: 100% (89/89), done.
(base) jovyan@dem01-0:~$
```

```
git clone https://github.com/footprintai/kubeflow-workshop
```

Step2: Use terminal to download the materials (3/3)

The screenshot shows a Jupyter Notebook interface. On the left is a file browser pane with a sidebar containing icons for file operations like new file, copy, move, and delete. A search bar at the top of the browser allows filtering by file name. Below the search bar is a list of files under the path `/kubeflow-workshop / pipelines /`. The list includes:

- img (directory)
- 0.helloworld.ipynb
- 1.conditional-flow.ipynb
- 2.persistvolume.ipynb
- 3.calc_metrics.ipynb
- 4.mnist.ipynb
- 5.auto-mnist-with-katib.ipynb
- 6.kfserving.ipynb
- 7.kfserving-canary-rollout.ipynb
- kfp.ipynb
- testdata.jpg
- testdata2.jpg
- testdata3.jpg

A red box highlights the first seven items in the list. To the right of the file browser is a terminal window titled "Terminal 1". It displays the output of a `git clone` command:

```
groups: cannot find name for group ID 1337
(base) jovyan@demo1-0:~$ git clone https://github.com/footprintai/kubeflow-workshop
Cloning into 'kubeflow-workshop'...
remote: Enumerating objects: 164, done.
remote: Counting objects: 100% (164/164), done.
remote: Compressing objects: 100% (98/98), done.
remote: Total 164 (delta 89), reused 133 (delta 62), pack-reused 0
Receiving objects: 100% (164/164), 1.71 MiB | 8.29 MiB/s, done.
Resolving deltas: 100% (89/89), done.
(base) jovyan@demo1-0:~$
```

The bottom of the terminal window shows the prompt "(base) jovyan@demo1-0:~\$". At the very bottom of the interface, there are some status indicators: "Simple", "1", "\$", "0", and a gear icon.

Financial Model Builder

Step3: Open financial model ipynb under tutorials/stockprice

The screenshot shows a Jupyter Notebook interface. On the left, there is a file browser with a red box highlighting the list of files in the 'tutorials/stockprice/' directory. The list includes '1.visualize-and-build-stock.ipynb' which was modified 'seconds ago'. The main area is a code editor titled '1.visualize-and-build-stock'. It displays Python code for stock price prediction using LSTM. The code includes comments explaining the purpose of the notebook, how to use Yahoo Finance via the yfinance package, and how to partition the data for training and prediction. It also shows how to install required packages and import libraries like pandas, numpy, and matplotlib.pyplot. The code editor has tabs for 'Code' and 'git' at the top right, and a 'Python 3' kernel indicator.

```
[ ]: # This jupyter notebook demonstrates how to predict stock with LSTM(long-short term memory) to
# predict stock price of the next day

# In this tutorial, we are going to use yahoo finance (called yfinance https://github.com/raroussi/yfinance)
# packages to download market data in daily basis.

# For model training and prediction, we partition the whole stock price series into several time-windowed (or
# sequence for input and use tensorflow to build the model.

# reference:
# [1] https://colab.research.google.com/drive/1Bk4zPQwAfzoSHZokKUefKL1s6lqmam6S?usp=sharing#scrollTo=8b-JsTv
# [2] https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm/notebook

[ ]: # Install the required package with script mode
!pip install pandas_datareader yfinance

[4]: import pandas as pd
import numpy as np

# For plot
import matplotlib.pyplot as plt

# For reading stock data from yahoo
from pandas_datareader.data import DataReader
import yfinance as yf
```

Kubeflow Terms

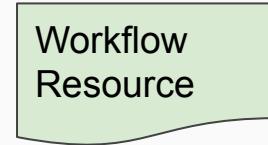
Kubeflow Terms



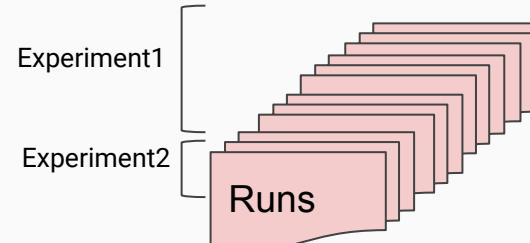
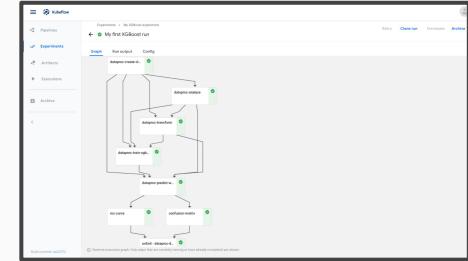
```
1.1. with open('requirements.txt', 'w') as f:  
1.2.     f.write("tensorflow<1.8.9\n")  
1.3.     # pip install --upgrade --user  
1.4.  
1.5. import kfp  
1.6. from kfp import dsl  
1.7.  
1.8. @dsl.pipeline()  
1.9. def pipeline():  
1.10.     return dsl.ContainerOp(  
1.11.         name='Adonis-0.4.23',  
1.12.         image='library/adonis:0.4.23',  
1.13.         command=['adonis', 'run',  
1.14.             '--args', 'text1', 'text2'])  
1.15.  
1.16.     @dsl.step()  
1.17.     def step1(text1):  
1.18.         return dsl.ContainerOp(  
1.19.             name='Adonis-0.4.23',  
1.20.             image='library/adonis:0.4.23',  
1.21.             command=['adonis', 'run',  
1.22.                 '--args', 'text1', 'text2'])  
1.23.  
1.24.     @dsl.step()  
1.25.     def step2(text2):  
1.26.         return dsl.ContainerOp(  
1.27.             name='Adonis-0.4.23',  
1.28.             image='library/adonis:0.4.23',  
1.29.             command=['adonis', 'run',  
1.30.                 '--args', 'text2'])  
1.31.  
1.32.     del pipeline  
1.33.  
1.34.     # run this pipeline to demonstrate execution order management.  
1.35.     if __name__ == '__main__':  
1.36.         pipeline().run()  
1.37.  
1.38.         # This will demonstrate how to directly return execution order  
1.39.         # by returning the step object from the pipeline function.  
1.40.         step1_task = adonis_op(step1)  
1.41.         step2_task = adonis_op(step2)  
1.42.         step2_task.after(step1_task)  
1.43.  
1.44.         # generate workflow artifacts in .zip format  
1.45.         MyCompiler.compile(pipeline(), compiler_output_pipeline, 'yellowworld.zip')
```

Pipeline Code

→ Compiled



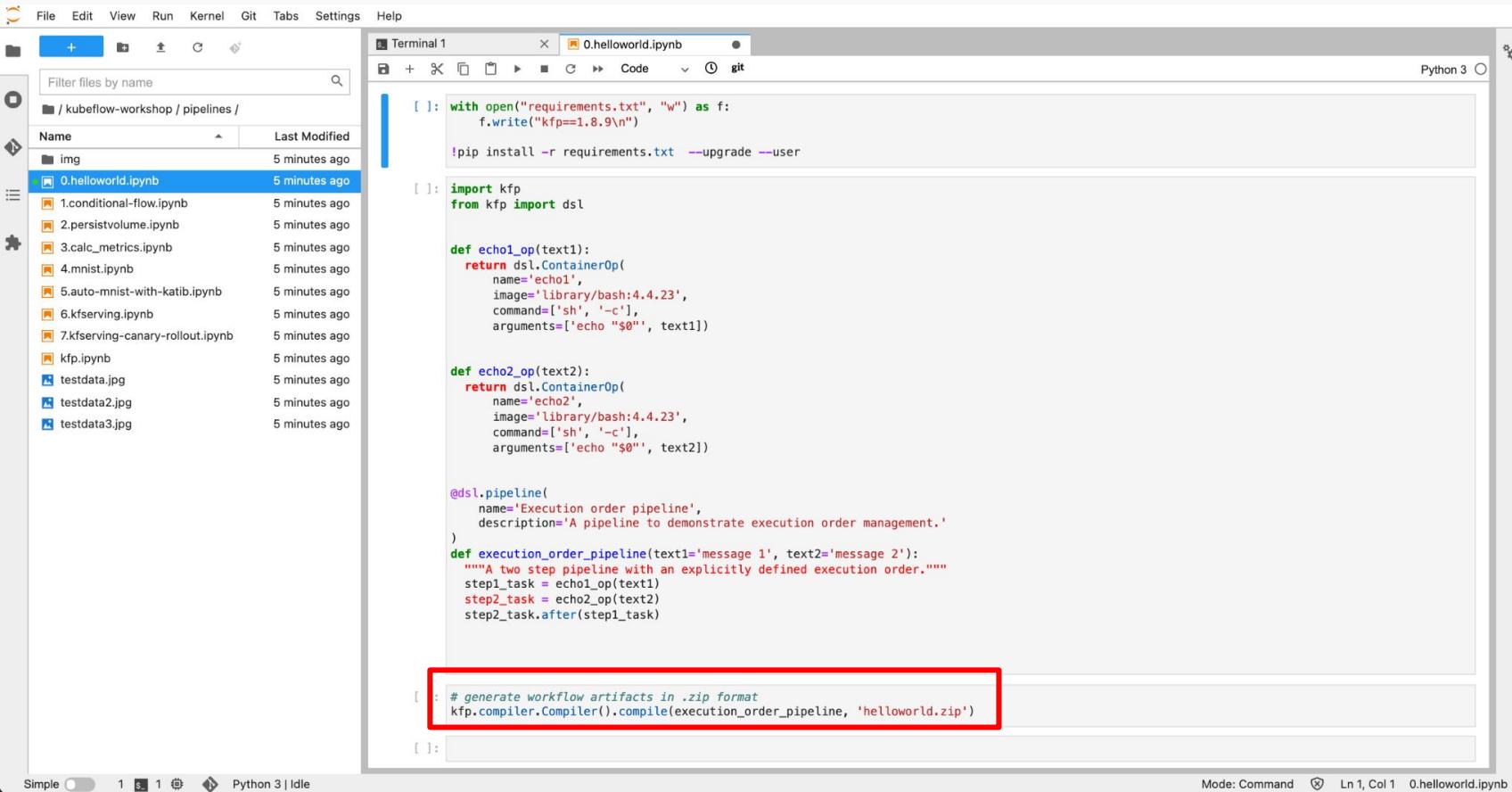
→ Create a Pipeline



← Create Run

Hello World Example

Step4: Compile helloworld.ipynb (1/2)



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help.
- Left Sidebar:** Shows a file tree under the path `/kubeflow-workshop / pipelines /`. The current file selected is `0.helloworld.ipynb`.
- Terminal 1:** A terminal window titled `0.helloworld.ipynb` running Python 3. It displays the following code:

```
[ ]: with open("requirements.txt", "w") as f:  
    f.write("kfp==1.8.9\n")  
  
!pip install -r requirements.txt --upgrade --user  
  
[ ]: import kfp  
from kfp import dsl  
  
def echo1_op(text1):  
    return dsl.ContainerOp(  
        name='echo1',  
        image='library/bash:4.4.23',  
        command=['sh', '-c'],  
        arguments=['echo "$0"', text1])  
  
def echo2_op(text2):  
    return dsl.ContainerOp(  
        name='echo2',  
        image='library/bash:4.4.23',  
        command=['sh', '-c'],  
        arguments=['echo "$0"', text2])  
  
@dsl.pipeline(  
    name='Execution order pipeline',  
    description='A pipeline to demonstrate execution order management.')  
def execution_order_pipeline(text1='message 1', text2='message 2'): """A two step pipeline with an explicitly defined execution order.""""  
    step1_task = echo1_op(text1)  
    step2_task = echo2_op(text2)  
    step2_task.after(step1_task)  
  
[ ]: # generate workflow artifacts in .zip format  
kfp.compiler.Compiler().compile(execution_order_pipeline, 'helloworld.zip')
```

A red box highlights the line `kfp.compiler.Compiler().compile(execution_order_pipeline, 'helloworld.zip')`.

Bottom Status Bar: Simple, Python 3 | Idle, Mode: Command, Ln 1, Col 1, 0.helloworld.ipynb.

Step4: Compile helloworld.ipynb (2/2)



File Edit View Run Kernel Git Tabs Settings Help

Terminal 1 × 0.helloworld.ipynb Python 3

Filter files by name

/ kubeflow-workshop / pipelines /

Name	Last Modified
img	7 minutes ago
0.helloworld.ipynb	seconds ago
1.conditional-flow.ipynb	7 minutes ago
2.persistvolume.ipynb	7 minutes ago
3.calc_metrics.ipynb	7 minutes ago
4.mnist.ipynb	7 minutes ago
5.auto-mnist-with-katib.ipynb	7 minutes ago
6.kfserving.ipynb	7 minutes ago
7.kfsserving_end2end_order.ipynb	7 minutes ago
helloworld.zip	seconds ago
kfp.ipynb	7 minutes ago
requirements.txt	seconds ago
testdata.jpg	7 minutes ago
testdata2.jpg	7 minutes ago
testdata3.jpg	7 minutes ago

```
name='echo1',
image='library/bash:4.4.23',
command=['sh', '-c'],
arguments=['echo "$0"', text1]

def echo2_op(text2):
    return dsl.ContainerOp(
        name='echo2',
        image='library/bash:4.4.23',
        command=['sh', '-c'],
        arguments=['echo "$0"', text2])

@dsl.pipeline(
    name='Execution order pipeline',
    description='A pipeline to demonstrate execution order management.')
def execution_order_pipeline(text1='message 1', text2='message 2'):
    """A two step pipeline with an explicitly defined execution order."""
    step1_task = echo1_op(text1)
    step2_task = echo2_op(text2)
    step2_task.after(step1_task)

[3]: # generate workflow artifacts in .zip format
kfp.compiler.Compiler().compile(execution_order_pipeline, 'helloworld.zip')

/opt/conda/lib/python3.8/site-packages/kfp/dsl/_container_op.py:1150: FutureWarning: Please create reusable components instead of constructing ContainerOp instances directly. Reusable components are shareable, portable and have compatibility and support guarantees. Please see the documentation on: https://www.kubeflow.org/docs/pipelines/sdk/component-development/#writing-your-component-definition-file The components can be created manually (or, in case of python, using kfp.components.create_component_from_func or func_to_container_op) and then loaded using kfp.components.load_component_from_file, load_component_from_uri or load_component_from_text: https://kubeflow-pipelines.readthedocs.io/en/stable/source/kfp.components.html#kfp.components.load\_component\_from\_file
warnings.warn(
```

generated helloworld.zip

Step4: Create a Pipeline (1/7)

The screenshot shows the Kubeflow Pipelines interface. On the left, a sidebar menu is visible with various options: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines (selected and highlighted with a red box), Runs, Recurring Runs, Artifacts, and Executions. At the bottom of the sidebar is a 'Manage Contributors' link. The main content area is titled 'Pipelines' and shows a list of existing pipelines. A red box highlights the '+ Upload pipeline' button in the top right corner, and a large red arrow points upwards towards it. The table lists six pipelines, each with a checkbox, a name, a description, and a timestamp. The descriptions include links to source code and GCP permission requirements. The timestamp for all pipelines is 11/30/2021, 1:02:25 PM.

	Pipeline name	Description	Upload
<input type="checkbox"/>	[Tutorial] V2 lightweight Python com...	source code Shows different component input and output options for KFP v2 components.	11/30/2021, 1:02:25 PM
<input type="checkbox"/>	[Tutorial] DSL - Control structures	source code Shows how to use conditional execution and exit handlers. This pipeline will randomly fail to demonstr...	11/30/2021, 1:02:24 PM
<input type="checkbox"/>	[Tutorial] Data passing in python co...	source code Shows how to pass data between python components.	11/30/2021, 1:02:23 PM
<input type="checkbox"/>	[Demo] TFX - Taxi tip prediction mod...	source code GCP Permission requirements . Example pipeline that does classification with model analysis based on...	11/30/2021, 1:02:22 PM
<input type="checkbox"/>	[Demo] XGBoost - Iterative model tra...	source code This sample demonstrates iterative training using a train-eval-check recursive loop. The main pipeline ...	11/30/2021, 1:02:21 PM

Step4: Create a Pipeline (2/7)

The screenshot shows the Kubeflow interface for creating a new pipeline. The left sidebar lists various options like Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. The main area is titled "Upload Pipeline or Pipeline Version" and has two radio button options: "Create a new pipeline" (selected) and "Create a new pipeline version under an existing pipeline". A red box highlights the "Pipeline Name" field, which contains "0.helloworld". A large red arrow points from this field to the text "1.Pipeline Name". Below it, another red box highlights the "File*" input field containing "helloworld (19).zip", with a "Choose file" button next to it. A large red arrow points from this field to the text "2.specify zip file location". At the bottom, a red box highlights the "Create" button. A large red arrow points from this button to the text "3.Create".

Kubeflow

kubeflow-user-example-c... ▾

Pipeline Versions

← Upload Pipeline or Pipeline Version

Create a new pipeline Create a new pipeline version under an existing pipeline

Upload pipeline with the specified package.

Pipeline Name*
0.helloworld

Pipeline Description*
0.helloworld

Choose a pipeline package file from your computer, and give the pipeline a unique name.
You can also drag and drop the file here.
For expected file format, refer to [Compile Pipeline Documentation](#).

Upload a file helloworld (19).zip Choose file

Import by url Package Url

Code Source (optional)

Create Cancel

1.Pipeline Name

2.specify zip file location

3.Create

Step4: Create a Pipeline (3/7)

The screenshot shows the Kubeflow Pipelines interface. On the left, a sidebar menu includes Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. Below this is a section for Manage Contributors.

The main area displays a pipeline named "0.helloworld (0.helloworld)". It shows a graph with two nodes: "echo1" at the top and "echo2" below it. An arrow points from "echo1" to "echo2". Above the graph, there are tabs for "Graph" (selected) and "YAML". To the right of the graph are buttons for "+ Create run", "+ Upload version", and "+ Create experiment". A large red box highlights the "+ Create experiment" button, and a red arrow points upwards towards it from the bottom right. A red callout box contains the text "Create an experiment".

Pipelines

← 0.helloworld (0.helloworld)

+ Create run + Upload version **+ Create experiment**

Graph **YAML**

Simplify Graph

```
graph TD; echo1[echo1] --> echo2[echo2]
```

Summary Hide

ID: 6f25028f-01e3-4acd-9389-7ec2031fb04b
Version: 0.helloworld

Version source

Step4: Create a Pipeline (4/7)

The screenshot shows the Kubeflow interface for creating a new experiment. On the left is a dark sidebar with navigation links: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP) (which is expanded to show Pipelines, Runs, Recurring Runs, Artifacts, and Executions), Manage Contributors, and GitHub integration.

The main area is titled "Experiments" and "← New experiment". It has a sub-section titled "Experiment details" with a descriptive text: "Think of an Experiment as a space that contains the history of all pipelines and their associated runs".

A form is displayed for entering experiment details:

- A text input field containing "0.helloworld.exp" is highlighted with a red border.
- An optional "Description (optional)" text input field is empty.
- At the bottom are two buttons: "Next" (highlighted with a red border) and "Cancel".

A large red arrow points from the text instructions below the form towards the "Next" button.

Enter experiment name and press Next button.

Step4: Create a Pipeline (5/7)

The screenshot shows the Kubeflow UI for creating a pipeline run. The left sidebar contains navigation links for Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP) (selected), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. Below this is a 'Manage Contributors' section, followed by GitHub and Documentation links, and finally Privacy and Usage Reporting information.

The main area is titled 'Run details' and includes the following fields:

- Pipeline***: A dropdown menu showing '0.helloworld' with a 'Choose' button.
- Pipeline Version***: A dropdown menu showing '0.helloworld' with a 'Choose' button.
- Run name***: An input field containing 'Run of 0.helloworld (1e261)'.
- Description (optional)**: A text input field.
- This run will be associated with the following experiment**: A dropdown menu showing '0.helloworld.exp' with a 'Choose' button.
- Service Account (Optional)**: An input field.
- Run Type**: Radio buttons for 'One-off' (selected) and 'Recurring'.
- Run parameters**: A section for specifying pipeline parameters:
 - text1**: Input field containing 'message 1'.
 - text2**: Input field containing 'message 2'.
- Start**: A large blue button at the bottom left, highlighted with a red border.
- Skip this step**: A link next to the Start button.

Two red arrows point from the right towards the 'Pipeline Version' and 'Experiment' fields, indicating the steps described in the text.

1. Run a pipeline and specify its version

2. Add its experiment name

Step4: Create a Pipeline (6/7)

The screenshot shows the Kubeflow interface for managing experiments. On the left, a sidebar navigation bar includes links for Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs (which is selected and highlighted with a red box), Recurring Runs, Artifacts, and Executions. Below these are sections for Manage Contributors and GitHub.

The main content area is titled 'Experiments' and shows the details for '0.helloworld.exp'. It includes sections for 'Recurring run configs' (0 active) and 'Experiment description'. There are buttons for '+ Create run' and '+ Create recurring run'. The 'Runs' section has tabs for 'Active' (selected) and 'Archived'. A table lists runs, with the first row ('Run of 0.helloworld (1e261)') highlighted by a red box. The table columns are: Run name, Status, Duration, Pipeline Version, Recurring Run, and Start time. The start time for the highlighted run is 12/2/2021, 4:33:10 PM. A red arrow points upwards from the bottom of the page towards the highlighted run row.

Run name	Status	Duration	Pipeline Version	Recurring Run	Start time
Run of 0.helloworld (1e261)	?	-	0.helloworld	-	12/2/2021, 4:33:10 PM

Run List

Step4: Create a Pipeline (7/7)

The screenshot shows the Kubeflow interface for creating a pipeline. On the left, the sidebar includes options like Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. The main area displays a pipeline graph titled "Run of 0.helloworld (1e261)". The graph consists of two nodes: "echo1" and "echo2", connected by a downward arrow. A red box highlights this graph. To the right, a modal window provides execution details for "execution-order-pipeline-fxxfn-4223123588". The "Input/Output" tab is selected, showing:

- Input parameters:** text1 (message 1)
- Input artifacts:** None
- Output parameters:** None
- Output artifacts:** main-logs (minio://mipeline/artifacts/execution-order-pipeline-qvlt5/2021/11/30/execution-order-pipeline-qvlt5-137025724/main.log, message 1)

A large red arrow points upwards from the "Output artifacts" section towards the pipeline graph, with the text "Running outputs" written below it.

Hyperparameter Example

Step5: Hyperparameter tuning with katib (1/4)

The screenshot shows a Jupyter Notebook interface. On the left, there is a file browser pane with a red border around it, displaying a list of files in the directory `/.../tutorials/stockprice-with-lstm/`. The files listed are:

- 1.visualize-... 32 minutes ago
- 2.optimize... 6 minutes ago
- helloworld... 7 minutes ago
- requireme... 8 hours ago

The main pane displays Python code for hyperparameter tuning using Katib. The code consists of two code cells:

```
[16]: ## import required pkgs
import kfp
import kfp.dsl as dsl
from kfp import components

from kubeflow.katib import ApiClient
from kubeflow.katib import V1beta1ExperimentSpec
from kubeflow.katib import V1beta1AlgorithmSpec
from kubeflow.katib import V1beta1EarlyStoppingSpec
from kubeflow.katib import V1beta1EarlyStoppingSetting
from kubeflow.katib import V1beta1ObjectiveSpec
from kubeflow.katib import V1beta1ParameterSpec
from kubeflow.katib import V1beta1FeasibleSpace
from kubeflow.katib import V1beta1TrialTemplate
from kubeflow.katib import V1beta1TrialParameterSpec

[17]: ## define katib objective, stopping criteria, and parameter spaces
experiment_name = "median-stop-lstm"
experiment_namespace = "kubeflow-user-example-com"

# Trial count specification.
max_trial_count = 4
max_failed_trial_count = 2
parallel_trial_count = 1

# Objective specification.
objective=V1beta1ObjectiveSpec(
    type="minimize",
    goal= 5
```

At the bottom of the interface, there are status indicators: "Simple" (button), "1 \$ 0" (button), "No Kernel | Idle" (text), "Saving completed" (text), "Mode: Command" (text), "Ln 7, Col 20" (text), and the file name "2.optimize-model-wit".

Step5: Hyperparameter tuning with katib (2/4)

The screenshot shows the Kubeflow interface for managing experiments and runs. On the left, a sidebar lists various options: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, and Artifacts. The 'Runs' option is selected.

The main area displays the 'Experiments' section for 'hello1'. It includes a 'Recurring run configs' section with a 'Manage' link, an 'Experiment description' input field, and a 'Create run' button. Below this is the 'Runs' section, which has tabs for 'Active' (selected) and 'Archived'. It features a 'Create recurring run' button and links for 'Compare runs', 'Clone run', and 'Archive'. A search bar labeled 'Filter runs' is also present.

A table lists three runs, each with a checkbox and columns for Run name, Status, Duration, Pipeline Version, Recurri..., Start time, quotient, and remainder. The first run's row is highlighted with a red border.

	Run name	Status	Duration	Pipeline Version	Recurri...	Start time	quotient	remainder
<input type="checkbox"/>	Run of hello-world_versio...	?	-	hello-world_vers...	-	5/17/2022, 9:42:...		
<input type="checkbox"/>	Run of hello-world_versio...	✓	0:06:19	hello-world_vers...	-	5/17/2022, 9:30:...		
<input type="checkbox"/>	Run of hello-world_versio...	✓	0:01:24	hello-world_vers...	-	5/17/2022, 9:23:...	0.000	6.000

Step5: Hyperparameter tuning with katib (3/4)

The screenshot shows the Kubeflow interface with the sidebar navigation bar on the left and the main content area on the right.

Left Sidebar (Kubeflow Navigation):

- Home
- Notebooks
- Tensorboards
- Volumes
- Models
- Experiments (AutoML)** (This item is highlighted with a red border.)
- Experiments (KFP)
- Pipelines
- Runs
- Recurring Runs

Main Content Area (Experiments Page):

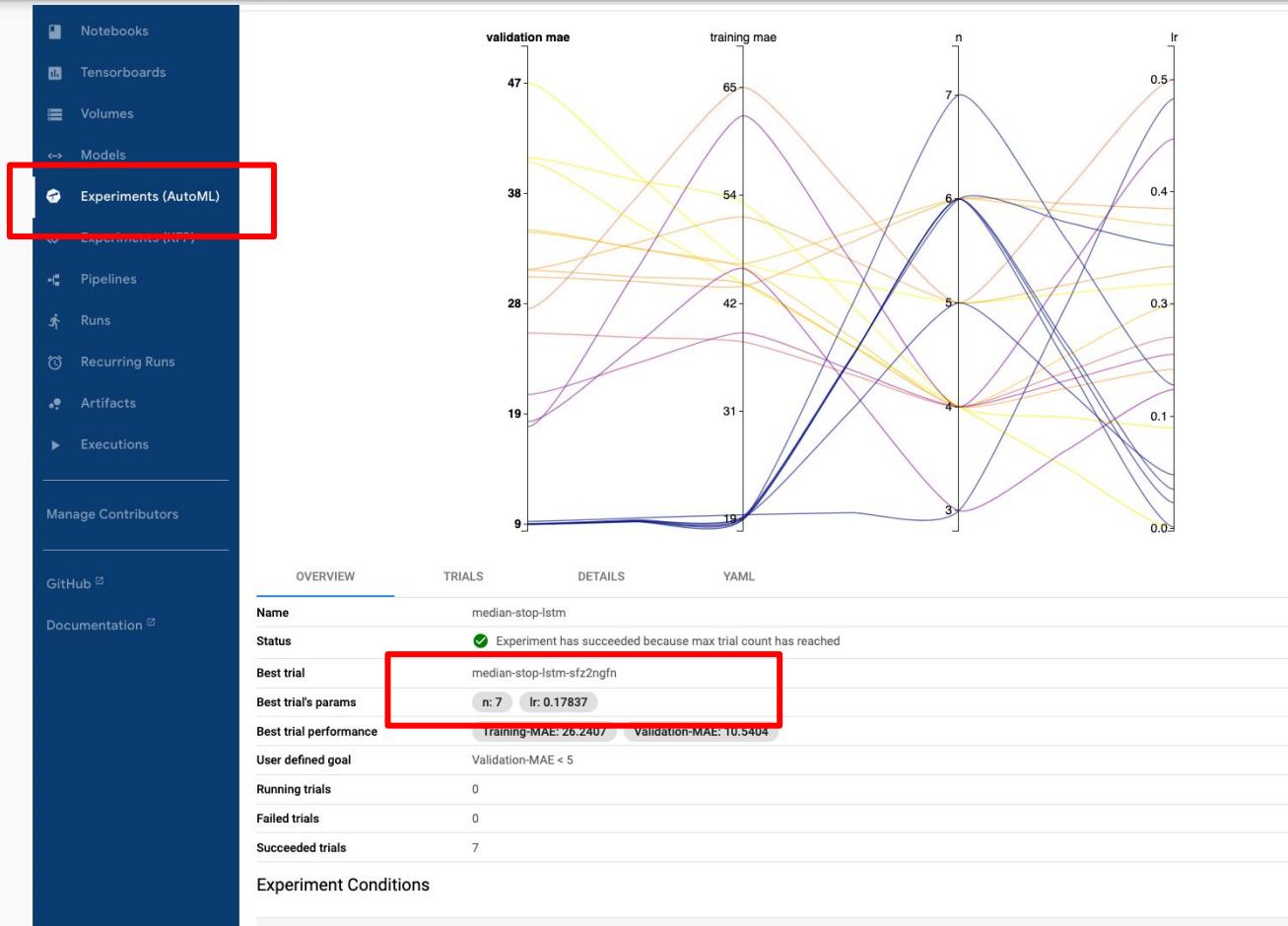
Top navigation bar: **kubeflow-user-example-c...** ▾ and a blue **↗** icon.

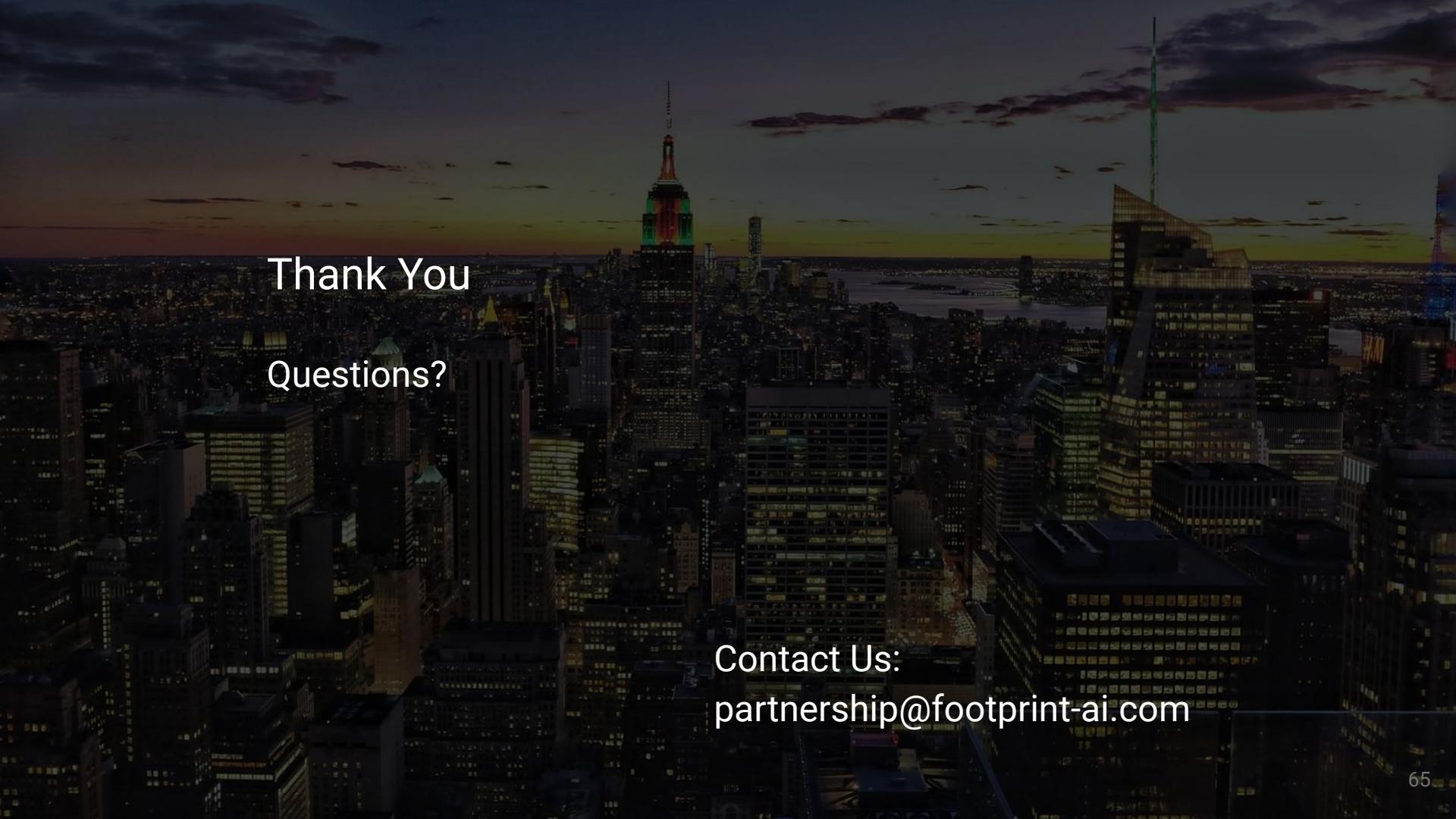
Experiments

+ NEW EXPERIMENT

Status	Name	Age	Successful trials	Running trials	Failed trials	Optimal trial
✓	median-stop	10 hours ago	4	0	0	Validation accuracy: 0.92028
✓	median-stop-lstm	7 minutes ago	7	0	0	Training mae: 26.2407

Step5: Hyperparameter tuning with kтив (4/4)



The background of the slide is a photograph of a city skyline at night, likely New York City, with the Empire State Building prominently visible. The sky is a mix of dark blues and purples with some warm orange and yellow hues from the setting sun.

Thank You

Questions?

Contact Us:
partnership@footprint-ai.com



***“The Best Engineers
Are Lazy”***

-Ancient Engineering Proverb

Materials

- Slides:
 - <https://github.com/FootprintAI/talks/tree/main/slides>
- Multikf
 - <https://github.com/FootprintAI/multikf>
- Kubeflow Workshop
 - <https://github.com/footprintai/kubeflow-workshop>