

SCALABILITY OF MODEL DEPLOYMENT – PART I & II

Session Agenda (Part 1)

1. Model Deployment Overview
 1. Machine Learning Pipelines
 2. ML Deployment Challenges
 3. Research Environment: Machine Learning Pipeline
2. System Architecture
3. Introduction to Object Oriented Programming
4. Production Code: Transforming Notebooks to Production Code

Session Agenda (Part 2)

1. Production Code: Transforming Notebooks to Production Code (contd.)
2. Serving Model via REST API
3. Packaging a Model
4. CI/CD
5. Running Apps in Containers (Docker)

Session Requirement

Assuming you have understanding of the following:

1. Machine learning models such as:

1. Linear Regression
2. Tree based algorithms

2. Model Evaluation Techniques

1. Mean Squared Error
2. R-square
3. ROC-AUC

3. Feature Engineering & Selection

4. Python Ecosystem:

1. Confidence with python programming
2. Familiarity with packages such as NumPy, Sklearn, Pandas, etc.

5. Basic GIT commands

6. Download datasets: <https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data>

ML Deployment Overview

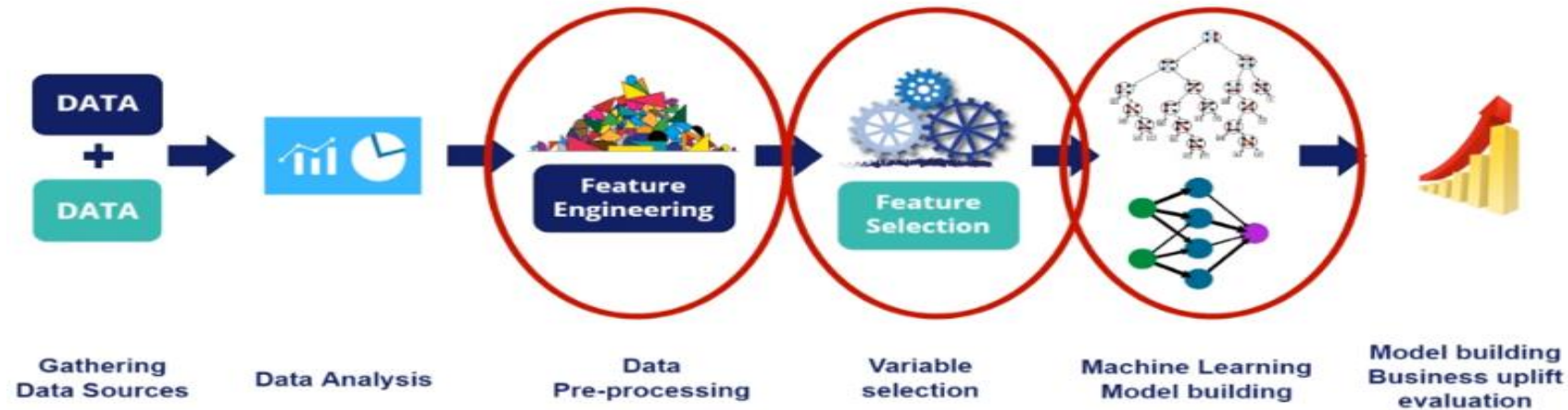
Model Deployment Overview



Machine Learning Pipeline

When we talk about machine learning model deployment, we actually refer to deployment of machine learning pipelines.

Machine Learning Pipeline: Production



What are the Benefits of Machine Learning Pipeline?

Constructing ML Pipelines provides many advantages. Some of them are:

1. **Flexibility** - Computation units are easy to replace. For better implementation, it is possible to rework that part without changing the rest of the system.
2. **Extensibility** - When the system is partitioned into pieces, it is easy to create new functionality.
3. **Scalability** - Each part of the computation is presented via a standard interface. If any part has an issue, it is possible to scale that component separately.

Model Deployment Challenges

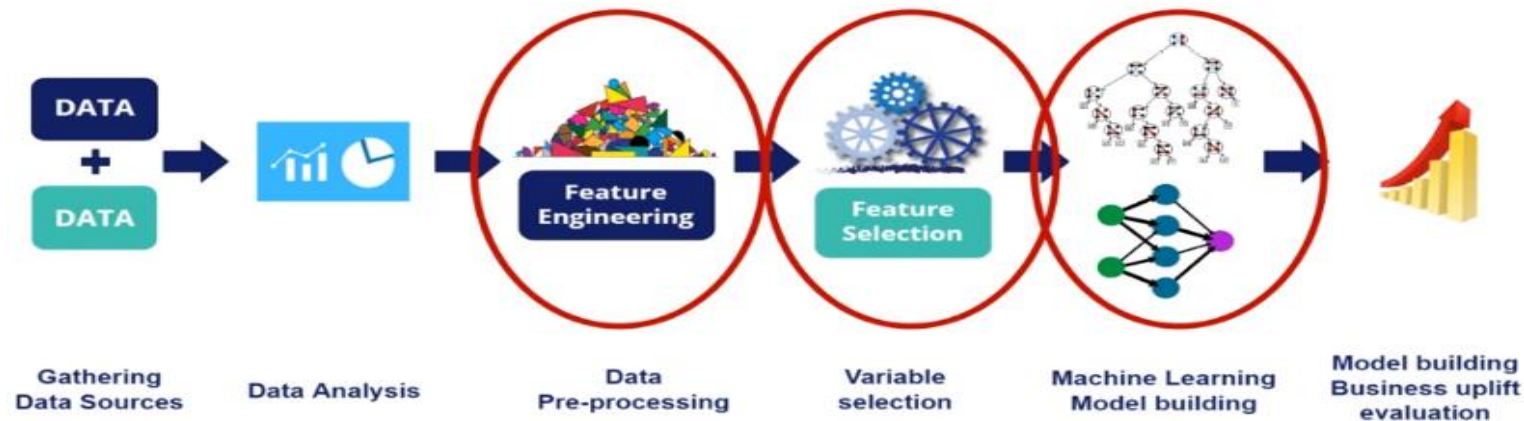
Why is modern deployment challenging?

- ✓ Modern deployment presents the challenges of traditional software like **reliability, reusability, maintainability and flexibility**, and it also presents additional challenges that are **specific to machine learning, like reproducibility**

So what do we need to do to ensure reproducibility?

- ✓ And more importantly, at which step of the pipeline do we need to ensure reproducibility? **Well, pretty much at every step of the pipeline.**

Machine Learning Pipeline: Production



- ✓ This means that all these steps should produce identical results, given the same data, both in a research environment and in our production and deployed models.

Research & Production Environment

- ✓ The **research environment** is a setting that contains the tools, programs and software that are suitable for data analysis, pre processing and the development of machine learning models.
- ✓ The **production environment** is a real time setting, with running programs and hardware setups. Daily operations in the production environment is where the machine learning models are actually available for business use.

ML System Architecture

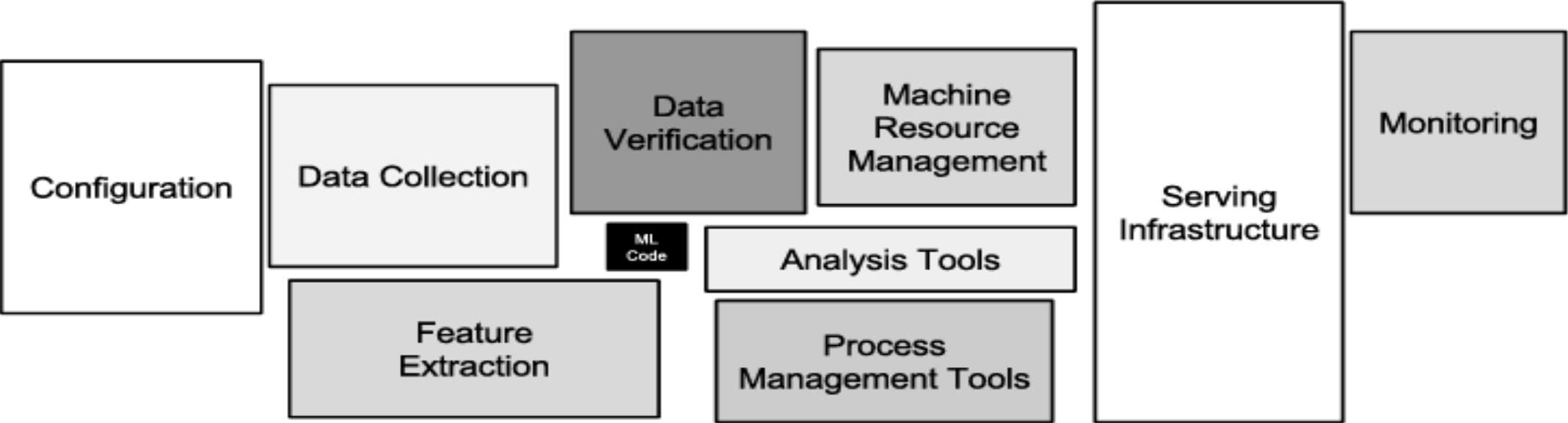
Machine Learning System Architecture

A system is comprised of multiple parts such as:

- ✓ **Infrastructure** encompasses everything from hardware to networking components to operating systems.
- ✓ **Applications** are programs for performing specific tasks, and we also have data, documentation and configuration.

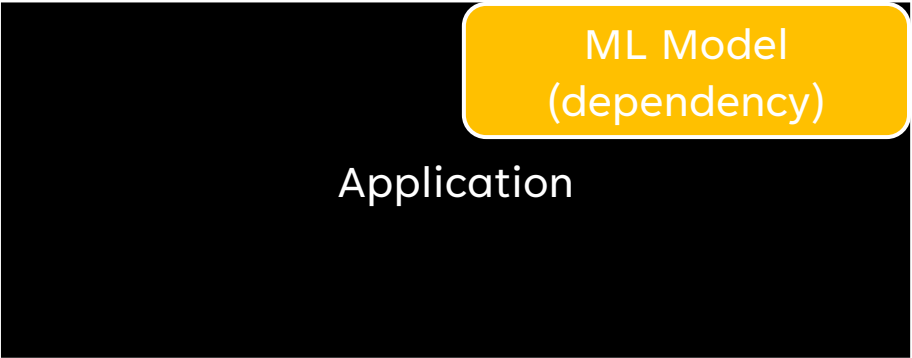
Architecture is how we describe a system.

Machine Learning System Architecture – Challenges & Best Practices

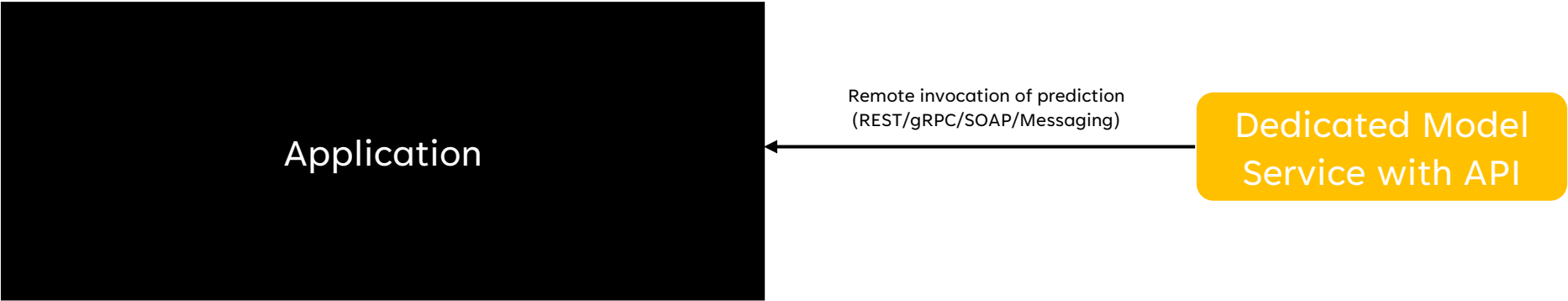


Machine Learning System Architecture Approach

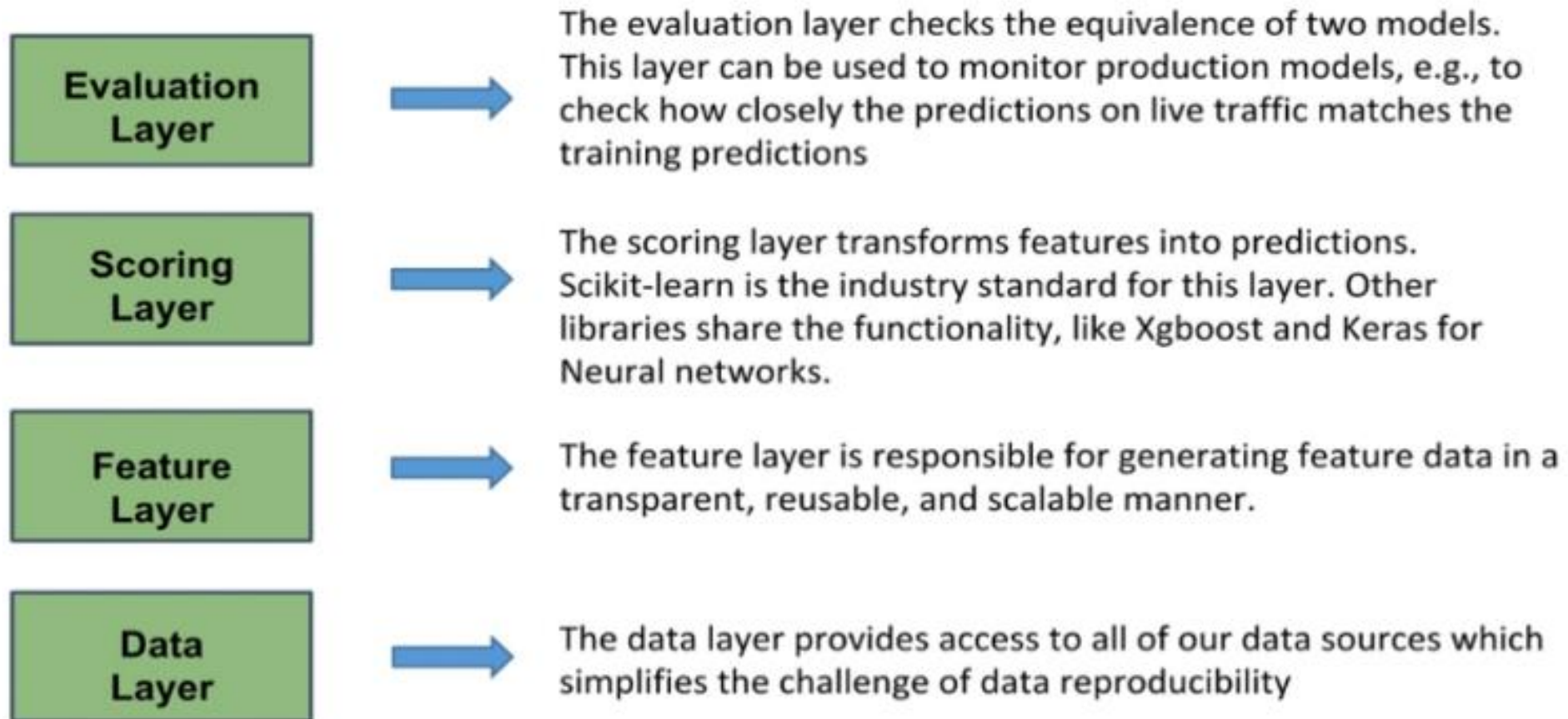
Embedded Architecture



Dedicated Model API Architecture



Machine Learning System Architecture Approach(contd.)

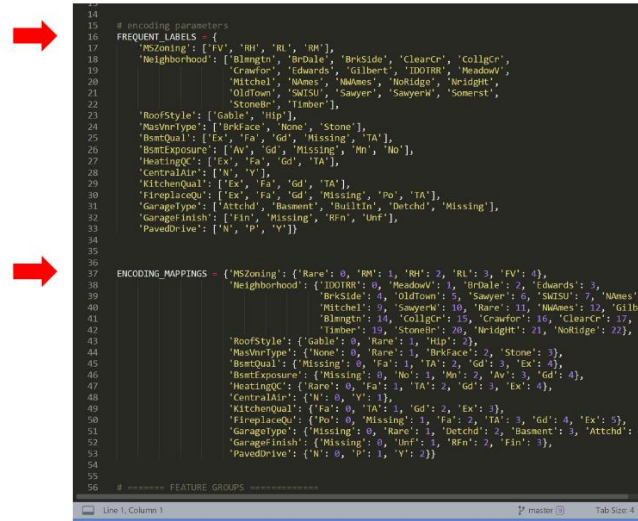


Source: Building a Reproducible Machine Learning Pipeline, Surgimura and Hart

Object Oriented Programming

Introduction: Object Oriented Programming

Procedural programming



```
14
15 # encoding parameters
16 FREQUENT_LABELS = {
17     'MSZoning': ['FV', 'RH', 'RM', 'RL', 'RM'],
18     'Neighborhood': ['Blmngtn', 'BrDale', 'BrkSide', 'ClearCr', 'CollCr',
19                     'Crawfor', 'Edwards', 'Gilbert', 'IDOTRR', 'Meadow',
20                     'Mitchel', 'Wams', 'Wawnes', 'NoRidge', 'NoRidge',
21                     'OldTown', 'SMISU', 'Sawyer', 'Sawyer', 'Somerst',
22                     'StoneB', 'Timber'],
23     'RoofStyle': ['Gable', 'Hip'],
24     'MasVnrType': ['BrkFace', 'None', 'Stone'],
25     'BsmtQual': ['Ex', 'Fa', 'Gd', 'Missing', 'TA'],
26     'BsmtExposure': ['Av', 'Gd', 'Missing', 'Mn', 'No'],
27     'HeatingQC': ['Ex', 'Fa', 'Gd', 'TA'],
28     'CentralAir': ['N', 'Y'],
29     'KitchenQual': ['Ex', 'Fa', 'Gd', 'TA'],
30     'FireplaceQu': ['Ex', 'Fa', 'Gd', 'Missing', 'Po', 'TA'],
31     'GarageType': ['Attchd', 'Basement', 'BuiltIn', 'Detchd', 'Missing'],
32     'GarageFinish': ['Fin', 'Missing', 'Rfn', 'Unf'],
33     'PavedDrive': ['N', 'P', 'Y']}
34
35
36
37 ENCODING_MAPPINGS = {'MSZoning': {'Rare': 0, 'RM': 1, 'RH': 2, 'RL': 3, 'FV': 4},
38                       'Neighborhood': {'IDOTRR': 0, 'Meadow': 1, 'BrDale': 2, 'Edwards': 3,
39                                         'BrkSide': 4, 'OldTown': 5, 'Sawyer': 6, 'SMISU': 7, 'Wams':
40                                         'Mitchel': 8, 'Sawyer': 10, 'Rare': 11, 'Wawnes': 12, 'Gilbert':
41                                         'Blmngtn': 14, 'CollCr': 15, 'Crawfor': 16, 'ClearCr': 17, 'S':
42                                         'Timber': 19, 'StoneB': 20, 'NoRidge': 21, 'NoRidge': 22},
43                       'RoofStyle': {'Gable': 0, 'Rare': 1, 'Hip': 2},
44                       'MasVnrType': {'None': 0, 'Rare': 1, 'BrkFace': 2, 'Stone': 3},
45                       'BsmtQual': {'Missing': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4},
46                       'BsmtExposure': {'Missing': 0, 'No': 1, 'Mn': 2, 'Av': 3, 'Gd': 4},
47                       'HeatingQC': {'Rare': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4},
48                       'CentralAir': {'N': 0, 'Y': 1},
49                       'KitchenQual': {'Fa': 0, 'TA': 1, 'Gd': 2, 'Ex': 3},
50                       'FireplaceQu': {'Po': 0, 'Missing': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
51                       'GarageType': {'Missing': 0, 'Rare': 1, 'Detchd': 2, 'Basement': 3, 'Attchd': 4},
52                       'GarageFinish': {'Missing': 0, 'Unf': 1, 'Rfn': 2, 'Fin': 3},
53                       'PavedDrive': {'N': 0, 'P': 1, 'Y': 2}}
54
55
56 # ===== FEATURE GROUPS =====
```

- Straightforward

- Hard-code parameters

- Save multiple objects or data structures

OUTPUT_SCALER_PATH = 'scaler.pkl'
OUTPUT_MODEL_PATH = 'lasso_regression.pkl'

Object oriented programming

- Data \Rightarrow attributes, properties
- Code or Instructions \Rightarrow methods (procedures)

Introduction: Object Oriented Programming(contd.)

Creating a Class in Python

The properties or parameters that the class takes whenever it is initialized, are indicated in the `__init__()` method.

The first parameters will always be a variable called `self`.

We can give any number of parameters to `__init__()`

```
class MeanImputer:  
    pass
```

```
class MeanImputer:  
  
    def __init__(self, variables):  
        self.variables = variables
```

```
>> my_imputer = MeanImputer(  
>>     variables = ['age', 'fare']  
>> )  
  
>> my_imputer.variables  
['age', 'fare']
```


Introduction: Object Oriented Programing(contd.)

Methods are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called self.

Our fit() methods learns parameters

```
class MeanImputer:  
    def __init__(self, variables):  
        self.variables = variables  
  
    def fit(self, X, y=None):  
        self.imputer_dict_ =  
            X[self.variables].mean().to_dict()  
  
        return self
```

Introduction: Object Oriented Programing(contd.)

Methods are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called self.

Our fit() methods learns parameters

Our transform() method transforms data

```
class MeanImputer:
    def __init__(self, variables):
        self.variables = variables

    def fit(self, X, y=None):
        self.imputer_dict_ =
            X[self.variables].mean().to_dict()
        return self

    def transform(self, X):
        for x in self.variables:
            X[x] = X[x].fillna(
                self.imputer_dict_[x])
        return X
```

Introduction: Object Oriented Programing(contd.)

Inheritance is the process by which one class takes on the attributes and methods of another.

```
class TransformerMixin:
```

```
    def fit_transform(self, X, y=None):  
        X = self.fit(X, y).transform(X)  
        return X
```

Parent Class

```
>> my_imputer = MeanImputer(  
>>     variables = ['age', 'fare']  
>> )  
  
>> data_t = my_imputer.fit_transform(my_data)  
>> data_t.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	0.083333	0.0	0.495064	0.0	0.0	0.0	0.666667
1	0.083333	0.0	0.499662	0.0	0.0	0.0	0.666667
2	0.083333	0.0	0.466207	0.0	0.0	0.0	0.666667
3	0.083333	0.0	0.485693	0.0	0.0	0.0	0.666667
4	0.083333	0.0	0.265271	0.0	0.0	0.0	0.666667

MeanImputer: Child Class

```
class MeanImputer(TransformerMixin):  
    def __init__(self, variables):  
        self.variables = variables  
  
    def fit(self, X, y=None):  
        self.imputer_dict_ =  
            X[self.variables].mean().to_dict()  
        return self  
  
    def transform(self, X):  
        for x in self.variables:  
            X[x] = X[x].fillna(  
                self.imputer_dict[x])  
        return X
```



ML Model Development and Production Code – Hands On

Production Code

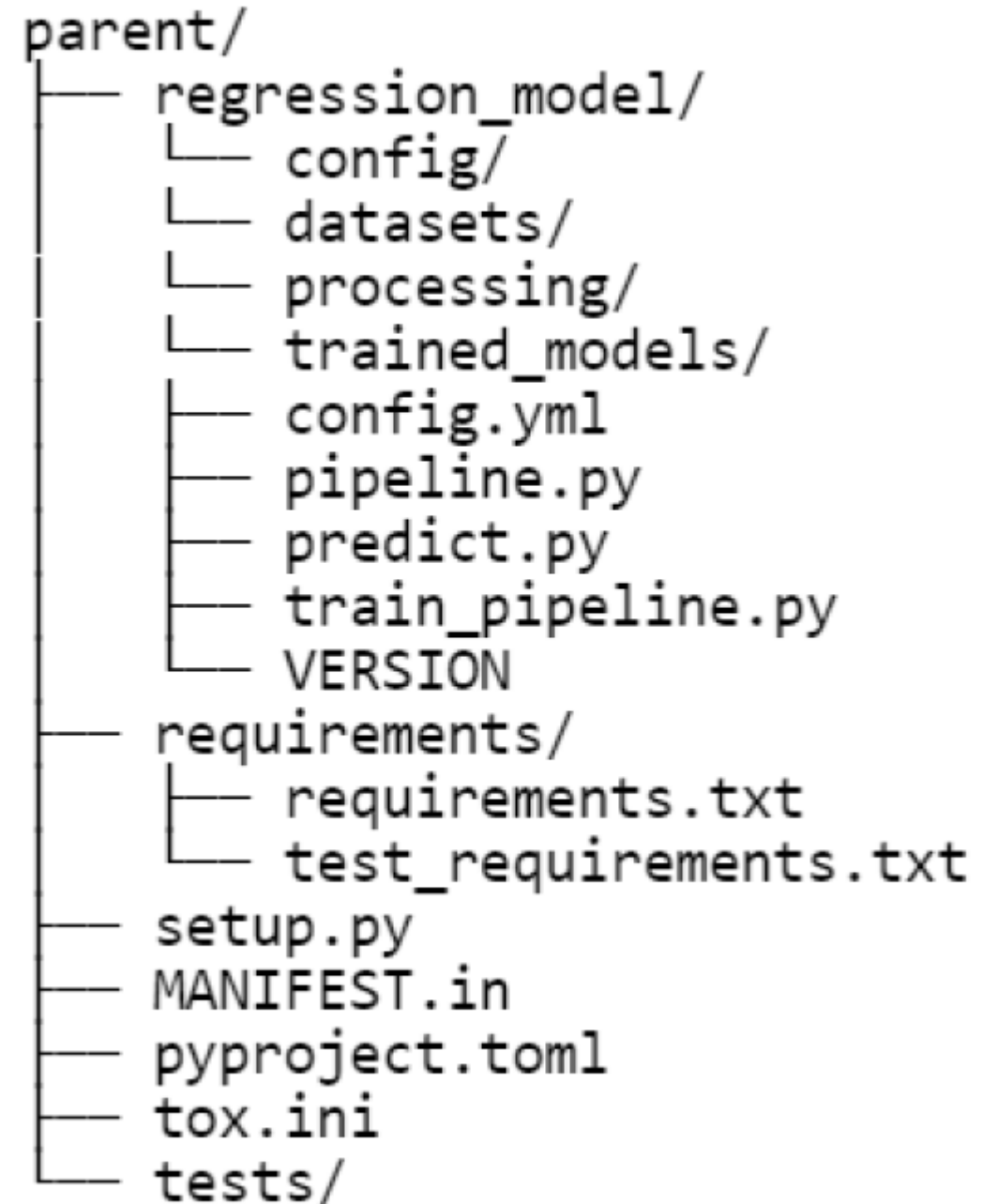
Production code is designed to be deployed to end users.

Production Code considerations:

1. Testability & Maintainability
2. Scalability & Performance
3. Reproducibility

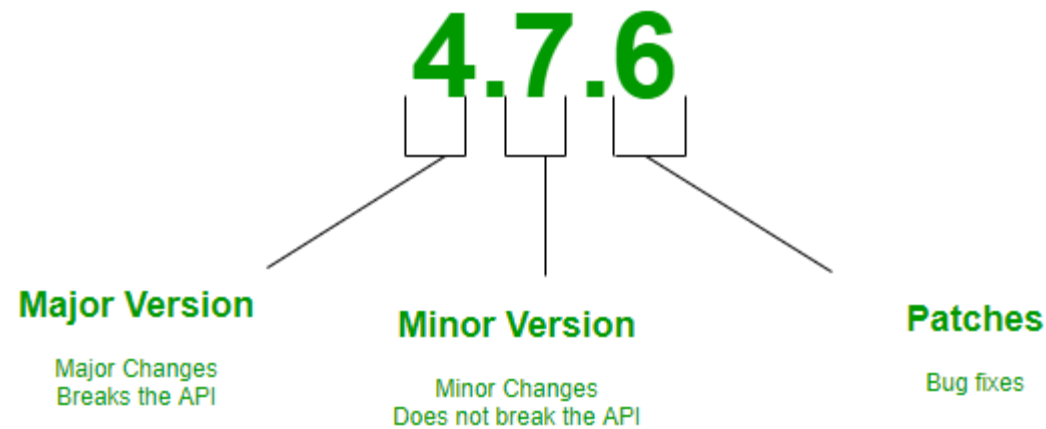
Python Packages

1. A **module** is a file which contains various Python functions and global variables. It is just a file with a *.py* extension which has python executable code.
2. A **package** is a collection of modules.



Production Code

Semantic Versioning



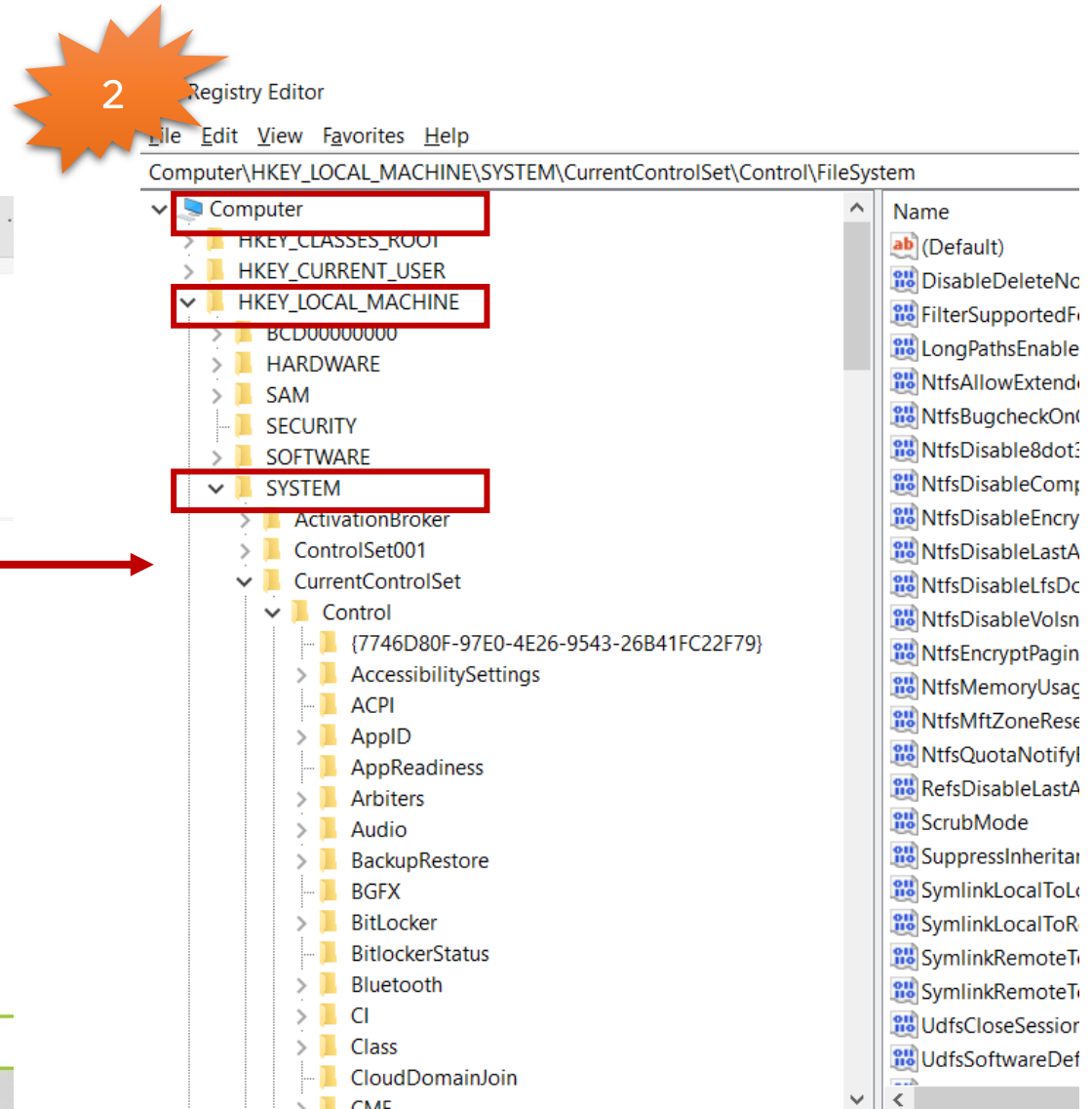
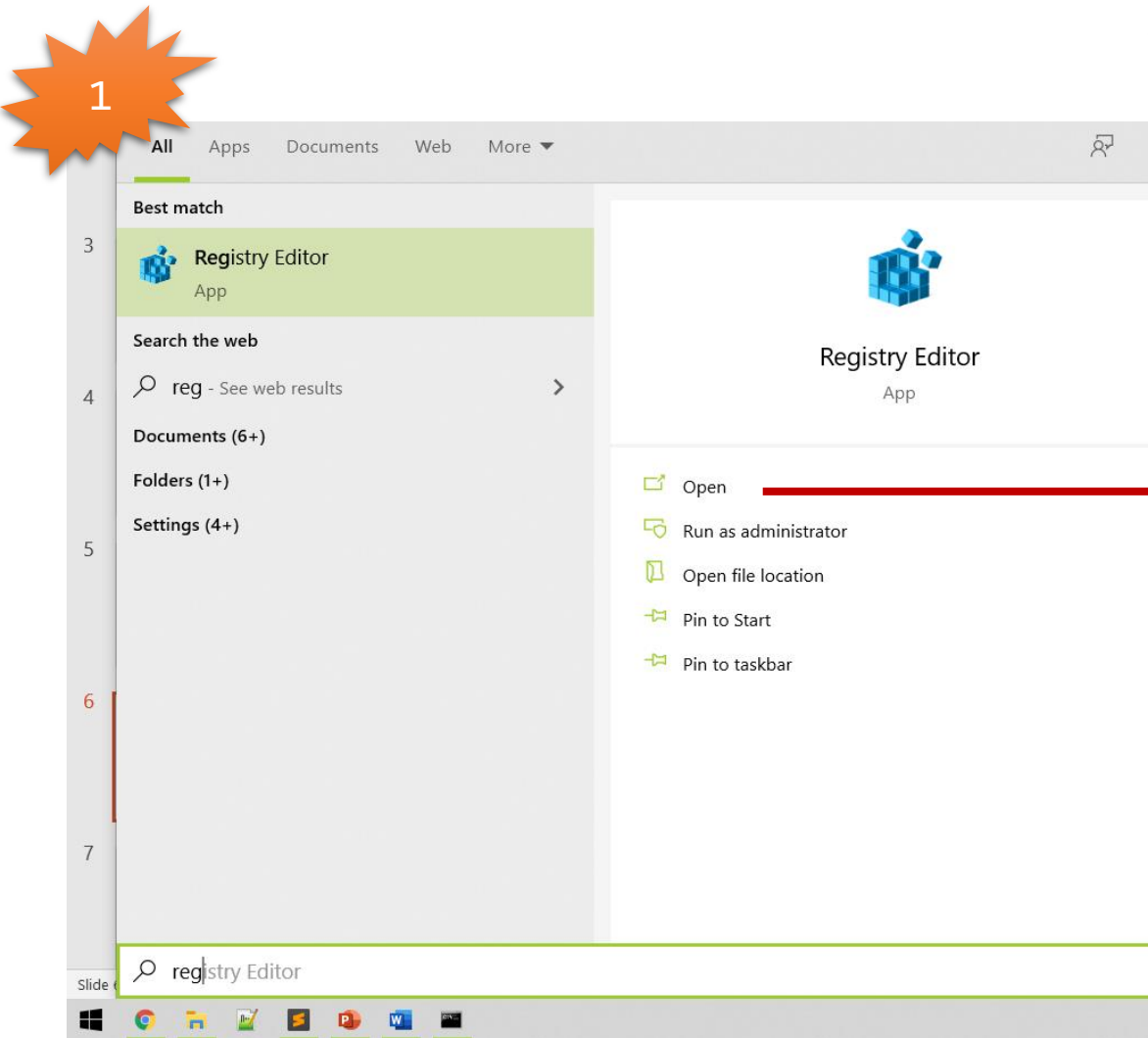
Semantic Versioning is a 3-component number in the format of X.Y.Z, where :

1. **X stands for a major version.** The leftmost number denotes a major version. **When you increase the major version number, you increase it by one but you reset both patch version and minor versions to zero. If the current version is 2.6.9 then the next upgrade for a major version will be 3.0.0.** Increase the value of X when breaking the existing API.
2. **Y stands for a minor version.** **It is used for the release of new functionality in the system. When you increase the minor version, you increase it by one but you must reset the patch version to zero.** If the current version is 2.6.9 then the next upgrade for a minor version will be 2.7.0. Increase the value of Y when implementing new features in a backwards-incompatible way.
3. **Z stands for a Patch Versions:** **Versions for patches are used for bug fixes. There are no functionality changes in the patch version upgrades. If the current version is 2.6.9 then the next version for a patch upgrade will be 2.6.10.** There is no limit to these numbers. Increase the value of Z when fixing bugs.

TOX

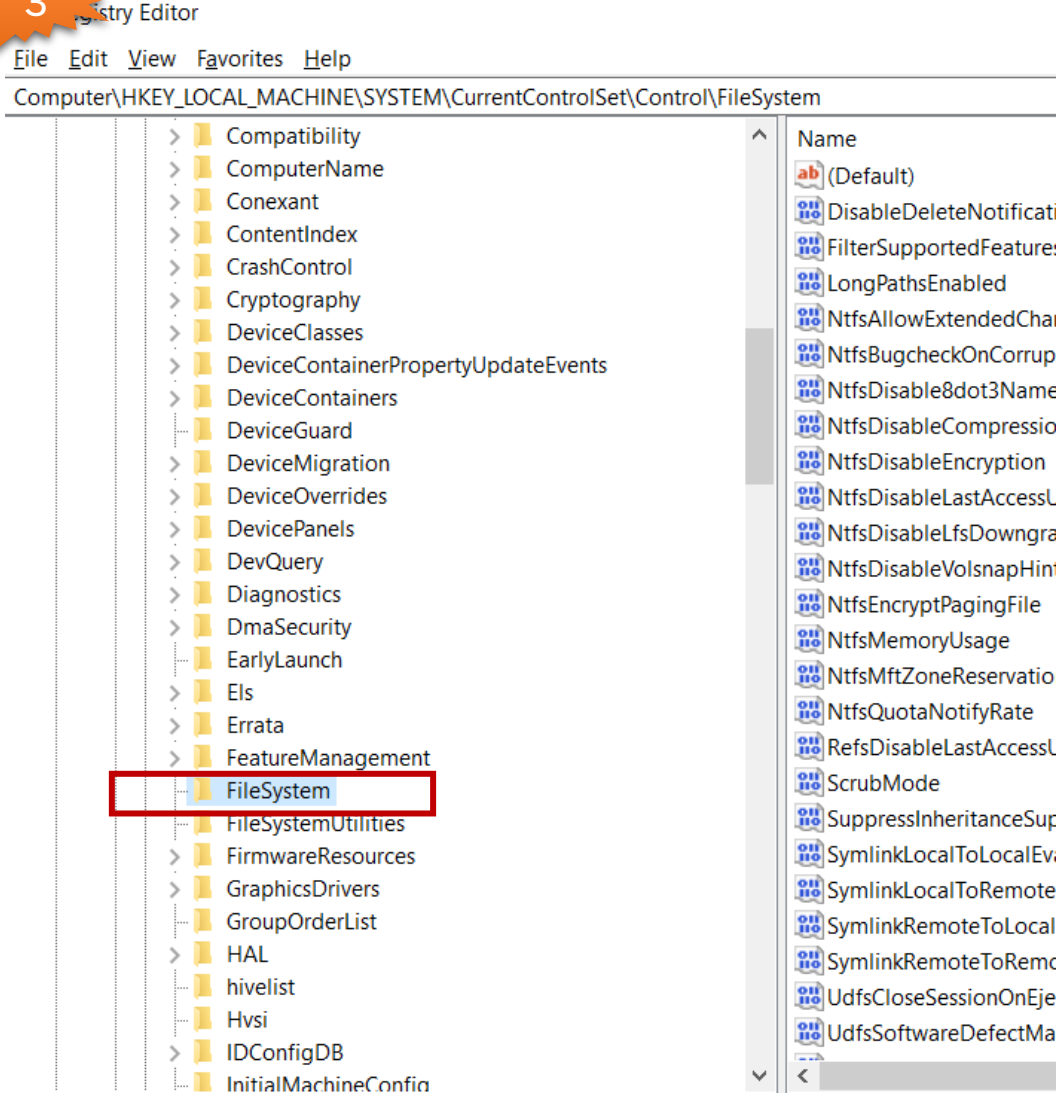
- Tox is a generic virtualenv management and test command line tool you can use for:
 - checking that your package installs correctly with different Python versions and interpreters
 - running your tests in each of the environments, configuring your test tool of choice
- In other words, it means that we do not have to worry about our operating systems.
- We can run tox on Windows, Mac, OS, Linux and get the same behavior across the platforms.
- We don't have to worry about things like setting up **python paths, configuration environment variables.**
- We do all of that stuff inside our tox.ini file.

Fixing TOX Invocation Error



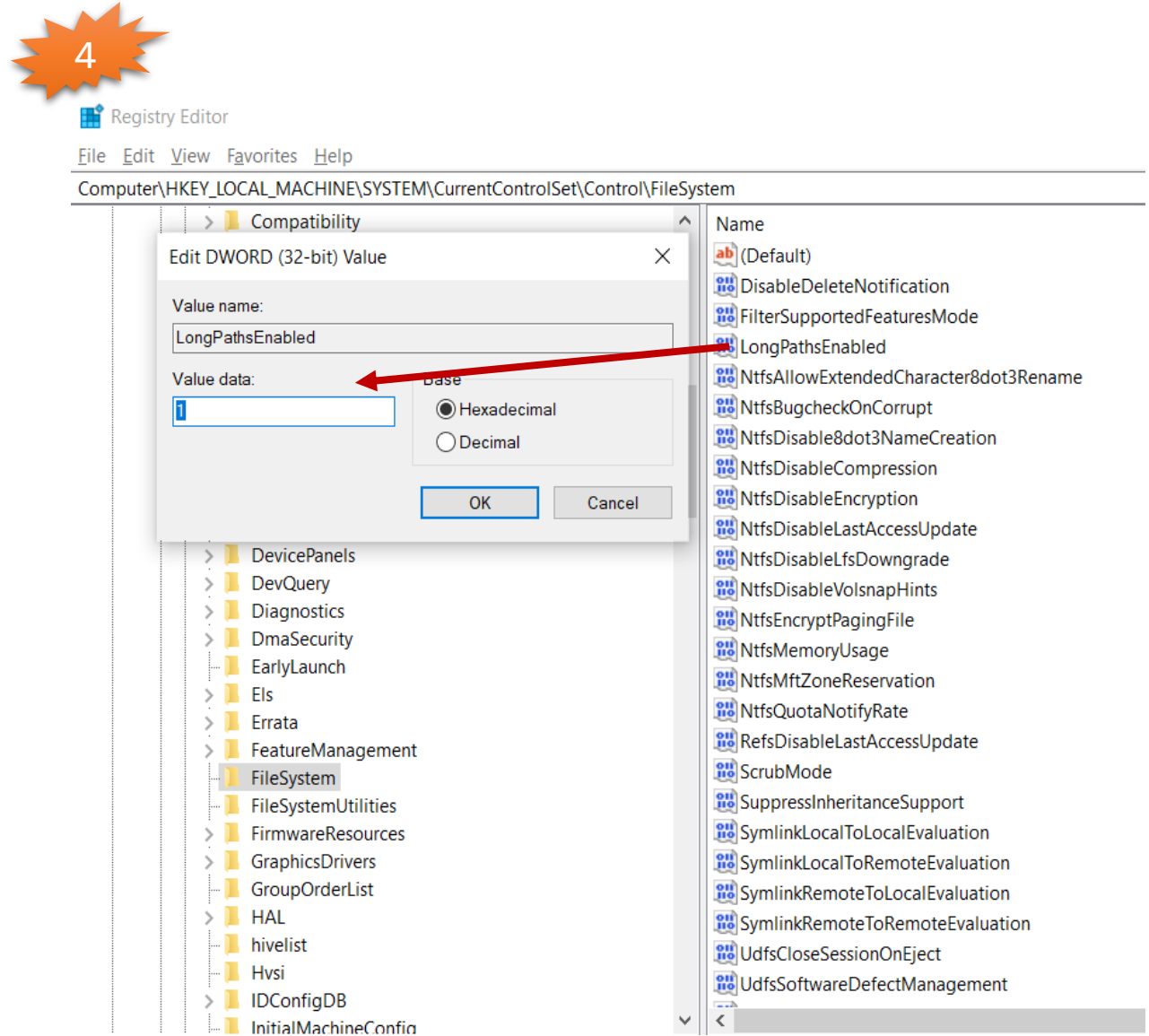
Fixing TOX Invocation Error (contd.)

3



Under control, look for FileSystem (shown above)

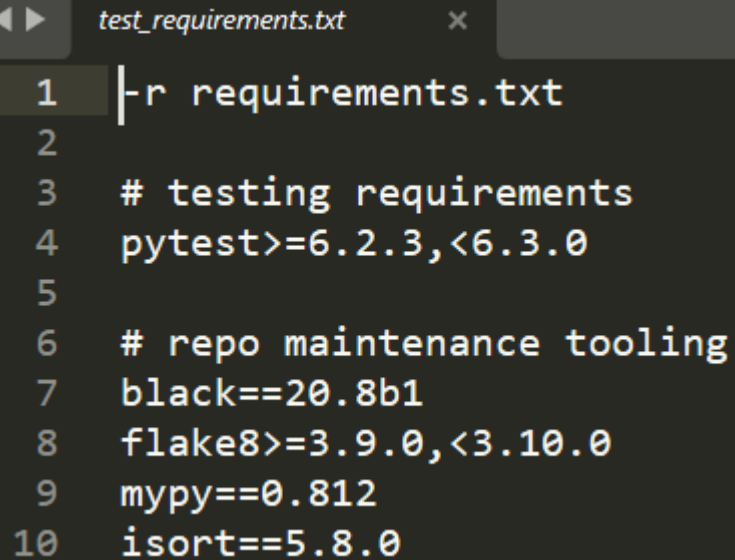
4



Make sure value data is set to "1"

Tooling

Let's look at some of the tools which we are using to manage our package.

A screenshot of a code editor window titled 'test_requirements.txt'. The editor shows a list of requirements for testing and development. The text is as follows:

```
1 |r requirements.txt
2
3 # testing requirements
4 pytest>=6.2.3,<6.3.0
5
6 # repo maintenance tooling
7 black==20.8b1
8 flake8>=3.9.0,<3.10.0
9 mypy==0.812
10 isort==5.8.0
```

1. Black is a code styling enforcement tool to tell us where we're not adhering to good Python conventions.
2. Mypy is a type checking tool.
3. Isort is ensuring our imports are in the correct order.
4. Flake8 is used for style checking.
5. Featuretools>>> feature_set

Now the purpose of any kind of tooling like this is to basically make our code easy for other people to use and also reduce the chances of us introducing bugs.

Deploying Model via REST API

Application Programming Interface (API) Overview

- **API allows two systems to communicate with one another.** It can use HTTP requests to get information from a web application or web server.
- **They are typically categorized as either SOAP or REST and both are used to access web services.**
 - SOAP relies solely on **XML** to provide messaging services, while
 - REST offers a more lightweight method, using URLs in most cases to receive or send information.
 - REST uses four different HTTP 1.1 verbs (**GET, POST, PUT, and DELETE**) to perform tasks.
 - You can find REST-based Web services that output the data in Command Separated Value (CSV), JavaScript Object Notation (JSON) and Really Simple Syndication (RSS). We can obtain the output we need in a form that's easy to parse within the language we need for our application.

API Endpoint

- ✓ Each endpoint is the location from which APIs can access the resources they need to carry out their function.
- ✓ For APIs, an endpoint can include a URL of a server or service.
- ✓ APIs work using 'requests' and 'responses.' When an API requests information from a web application or web server, it will receive a response. The place that APIs send requests and where the resource lives, is called an **endpoint**.

How can our API be consumed?



1. The most basic scenario is that users browsing the web using a web browser and when they want to load information onto a web page.
2. They make a call to our application via the rest API and that application returns HTML in the scenario where we're using our machine learning dependency.
3. Then, this browser might contain a form with information about a house which you want to get a price prediction for and then you would make a post request to the API and app would then pass that form information to our model dependency, get the prediction in HTML.
4. Other examples of using the API would include a mobile app which would make API calls and receive data to the predictions from our model.
5. Though returning HTML is less ideal for complex UIs. So, we use modern framework like JavaScript or Angular.

Introduction to FastAPI

What is FastAPI?

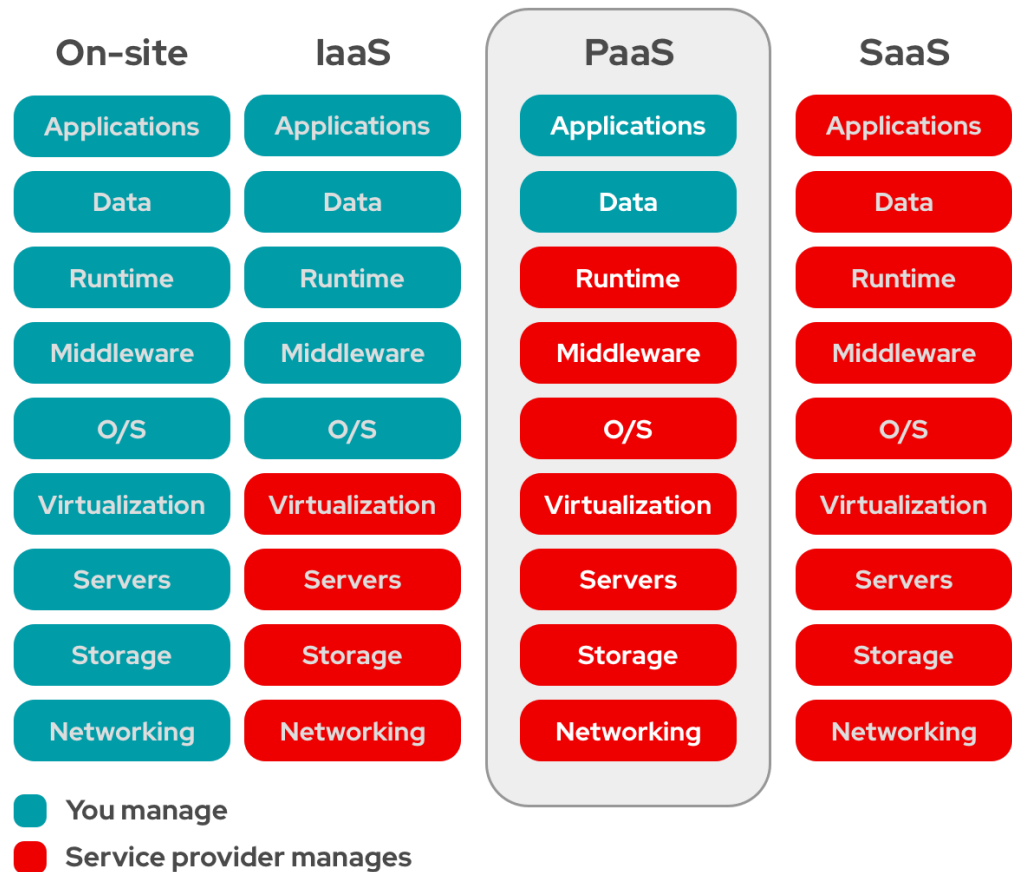
- ✓ The official FastAPI website describes it as a modern and high-performance web framework for building APIs with Python 3.6+.
- ✓ FastAPI is very fast due to its out-of-the-box support of the **1async feature of Python 3.6+** and it uses tools like Starlette and Pydantic. Now, many big tech companies like Uber, Netflix, and Microsoft are using FastAPI to build their apps.

FastAPI features

- ✓ **High-performance:** As the name suggests, FastAPI is fast. It's considered to be one of the fastest Python frameworks currently available.
- ✓ **Robust:** You can create production-ready code using automatic interactive documentation.
- ✓ **Intuitive:** FastAPI was designed to be easy to use and learn. It offers great editor support and documentation.
- ✓ **Quick to code:** FastAPI increases your developing speed by 200%-300%.
- ✓ **Compatible:** It works well with the open standards for APIs such as OpenAPI (previously known as Swagger), and JSON schema.
- ✓ **Plugins:** You can easily create plugins using dependency injection.

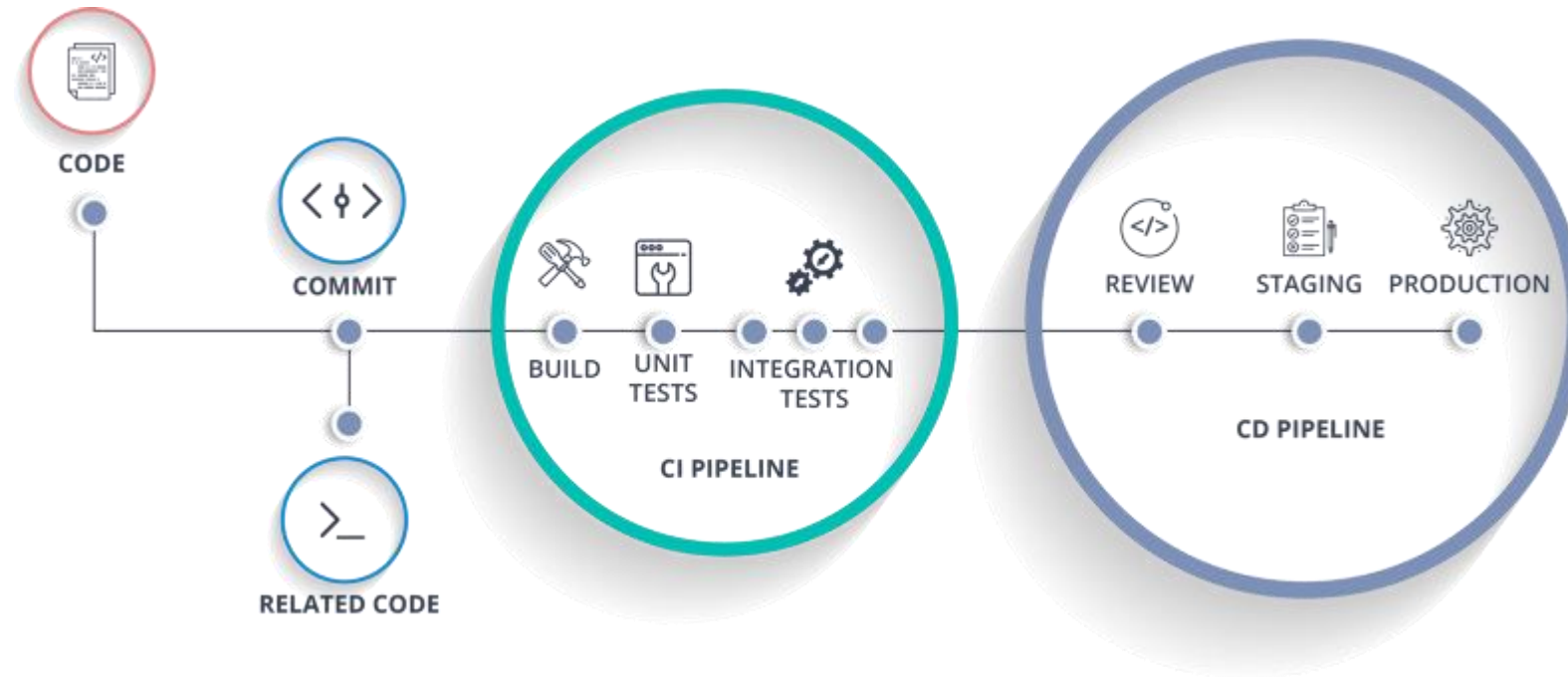
Platform as a Service (PaaS)

- ✓ Platform as a service (PaaS) is a cloud computing model where a third-party provider delivers hardware and software tools to users over the internet. Usually, these tools are needed for application development.
- ✓ A PaaS provider hosts the hardware and software on its own infrastructure. As a result, PaaS frees developers from having to install in-house hardware and software to develop or run a new application.



CI/CD

Introduction to CI/CD



- ✓ CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.
- ✓ One of the best known open source tools for CI/CD is the automation server Jenkins. Jenkins is designed to handle anything from a simple CI server to a complete CD hub.

CircleCI

- ✓ CircleCI is the continuous integration & delivery platform that helps the development teams to release code rapidly and automate the build, test, and deploy.

Docker

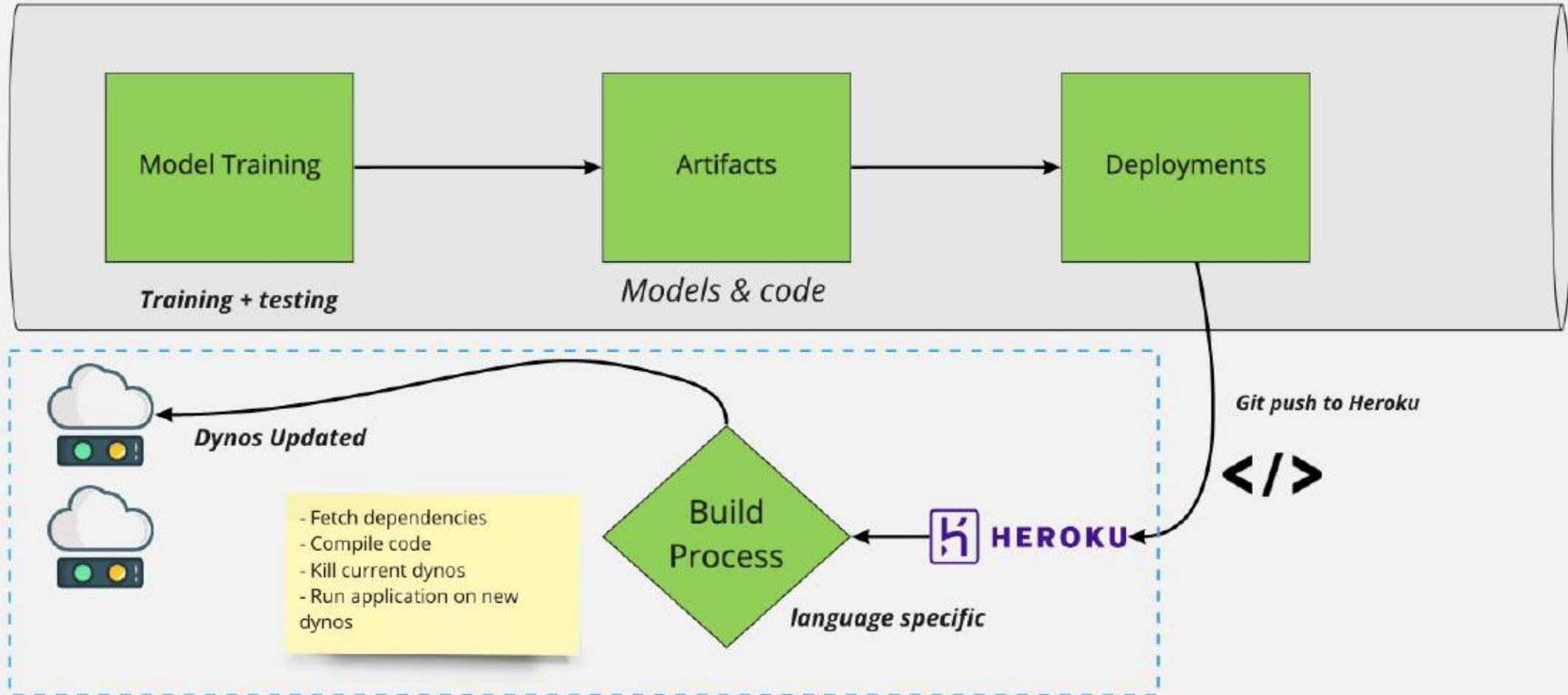
Introduction to Docker

- Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.
- The docker Open Source Project provides a way of automating the deployment of applications as portable, self-sufficient containers.
- The concepts of container and image are probably the most important concepts in docker.
 - Containers are running system defined by images
 - These images are made up of one or more layer of files and some metadata for docker
- If we look at images and containers from oops concept, we can say images are classes and containers are object.
- The same way objects are concrete instantiations of classes. **Containers are instantiations of images.**
- You can create multiple containers from a single image, and they are all isolated from one another in the same way objects or whatever, you change in the object.
- It won't affect the class definition.

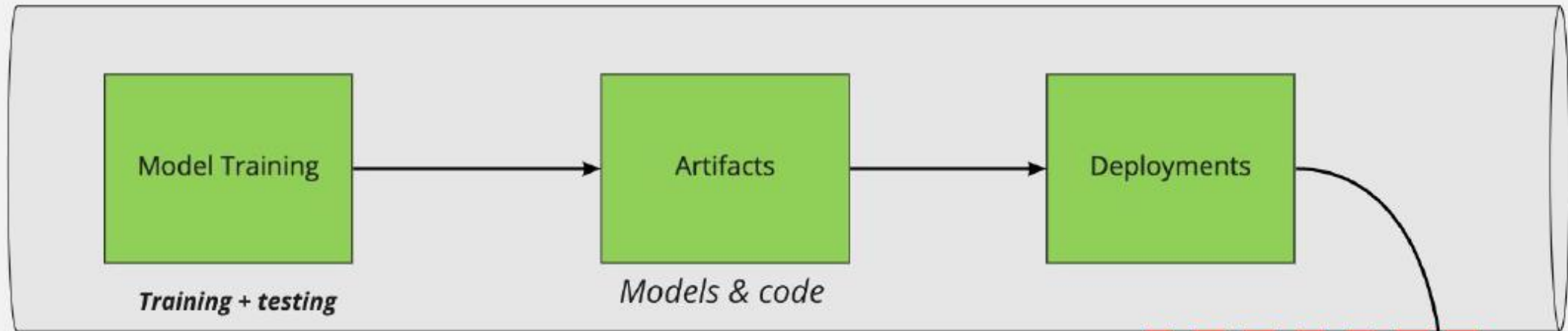
Difference between Docker and Virtual Machines

- Containers and VMs are similar in their goals, namely to isolate an application and its dependencies into a self-contained unit that can run anywhere.
- However, a VM is essentially an emulation of a real computer that executes programs like a real computer. VMS run on top of a physical machine using a **hypervisor**.
- One key difference between containers and VMs is that containers share the host systems kernel with other containers.
- This means that docker containers are very lightweight and fast.
- And since containers are just sandbox environments running on the kernel, they take up fewer resources.
- This means that you can create and run a container in seconds compared to VMs, which might take much longer because they have to boot up a full virtual operating system every time.

Container Deployment



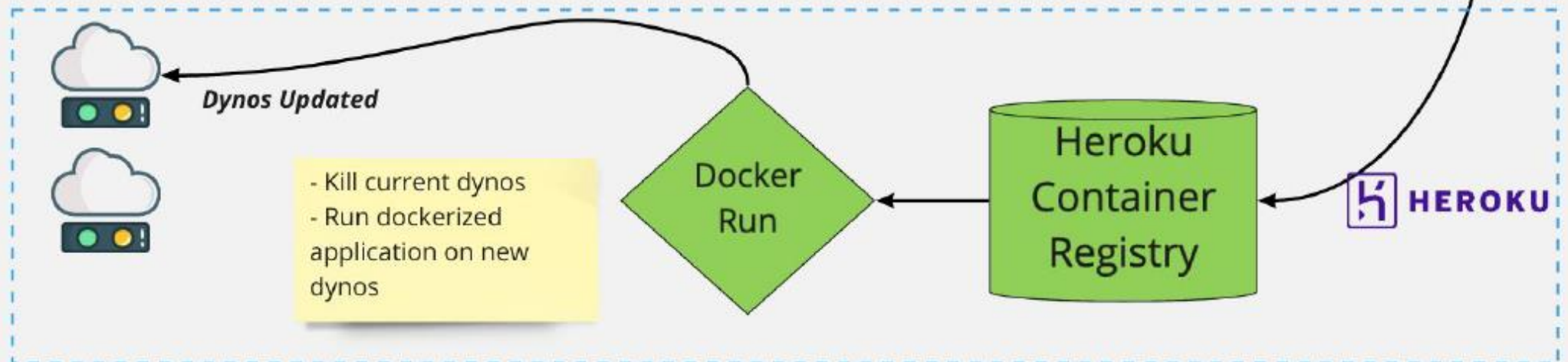
Container Deployment (contd.)



- We create docker image in CI pipeline and not in Heroku
- Once we have the image, so instead of pushing code we push the image to Heroku container registry



Container Deployment
(Section 8)



What is Heroku Dyno?

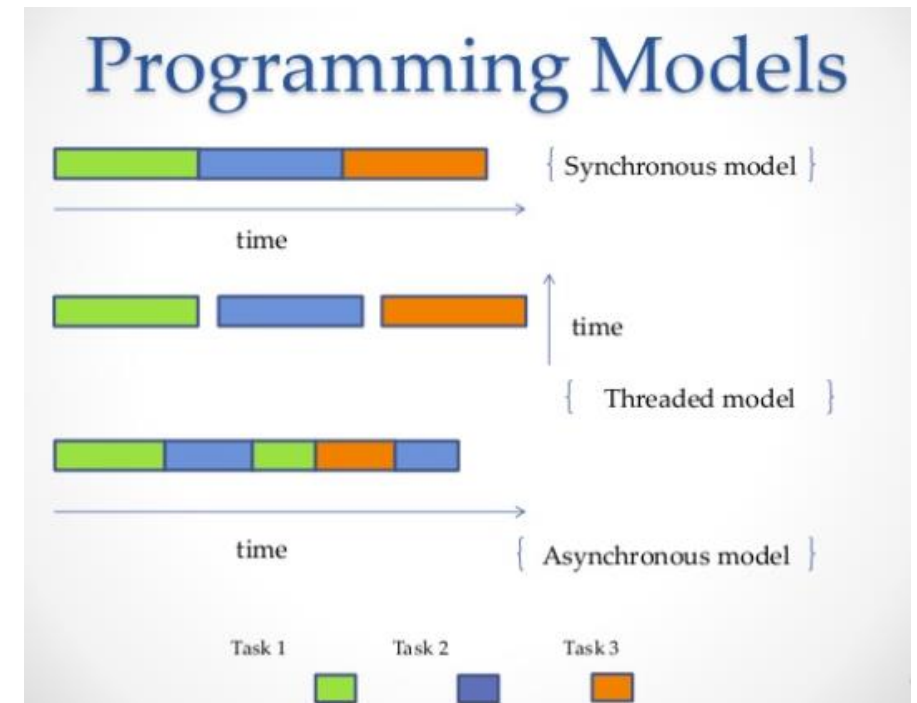
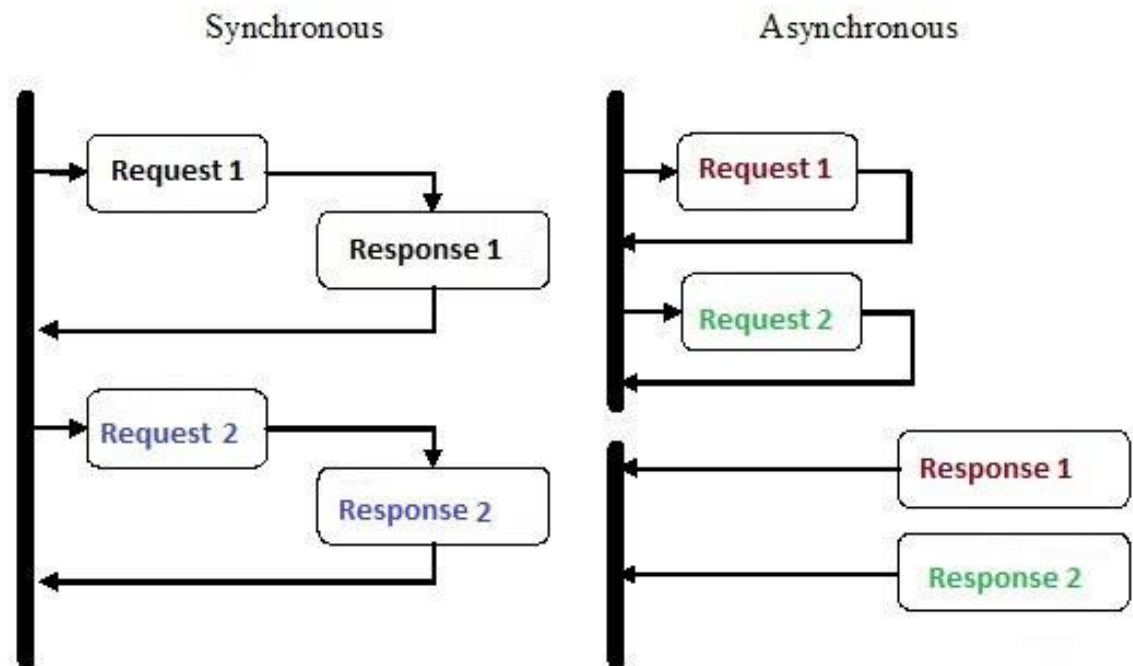
- All Heroku applications run in a collection of lightweight Linux containers called dynos.
- Web dynos are dynos of the “web” process type that is defined in your Procfile. Only web dynos receive HTTP traffic from the routers.
- Once a web dyno is started, the dyno formation of your app will change (the number of running dynos of each process type) – and subject to dyno lifecycle, Heroku will continue to maintain that dyno formation until you change it
- You can configure multiple dynos, which run in a formation. Heroku takes care of much of the maintenance and scaling for you.

Thank You

Archive

Definition: Asynchronous programming

- **Asynchronous programming** is a type of parallel programming in which a unit of work is allowed to run separately from the primary application thread.
- When the work is complete, it **notifies the main thread about completion or failure of the worker thread**. There are numerous benefits to using it, such as improved application performance and enhanced responsiveness.
- For example, instead of waiting for an HTTP request to finish before continuing execution, with Python async coroutines you can submit the request and do other work that's waiting in a queue while waiting for the HTTP request to finish.



Definition: Microservices

In microservice architecture, multiple loosely coupled services work together. Each service focuses on a single purpose and has a high cohesion of related behaviors and data.

