API

Application Programming Interface (API)

- APIs are how we communicate with programs
- Every language has its own set of APIs
- We've even learned some
  - TensorFlow API
  - Scikit Learn API
- When we deploy a model, there is typically an API so that users can interact with the model
  - That also means you need to **document** your API

Important Data Science APIs and Formats

- Representative State Transfer (REST) API
- HTTP
  - Not really an API but used to communicate
- JSON (Javascript Object Notation)
  - Used outside of Javascript!
- Extensible Markup Language (XML)

A Little HTTP

- HTTP has methods that are used to communicate and is the easiest to get started.
  - GET
  - POST
  - PUT
  - HEAD
  - DELETE
  - PATCH
  - OPTIONS
- GET and POST are 99% of your HTTP requests.
- The client sends a request to a server. Thus, the client sends GET or POST requests (among others) and the server sends a response.
- So if you put your model on a web server, that server needs to handle GET and POST requests!

GET

- GET is used to request data.
- The requested data is in the address in some form.
- GET requests can be cached and bookmarked.
- **Never ever use GET for sensitive data.**
  - Never
  - **No exceptions**
- Data cannot be modified in a GET request.
- GET does not send anything in the body.
  - All information is in the URL

POST

- Sends data to the URL
  - Data is sent **inside** the request and not with the address.
- POST requests cannot be bookmarked or cached
  - No history either!
- POST requests have a request body

Summary of GET/POST

- GET
  - “I want some data described by the URL.”
- POST
  - “I am sending you some data.”
- What you receive is the response
  - The infamous “404” error is an error that says, “I can’t find what you want.”
  - Response “200” means you received a valid response.
    - (Probably should store it somewhere to read!)

API and HTTP

- Use a web server to host your model.
  - The server should expect GET and/or POST requests.
- Use HTTP GET/POST requests to communicate with the server.
- Publish details of how to format the HTTP so a user can use your model.

DataScience@SMU

Docker

Docker

- One of the problems we have is how do we get code that runs on machine X to run on machine Y.
  - Different hardware
  - Different software
- The solution is containers.
- Containers are “virtual machines.”
  - Small
  - Compact
  - Only software required

An Old Problem

- How do you run Windows on Linux (or Mac, etc.)?
  - Or how do you run Windows 98 on Windows Vista?
- The idea was a virtual machine
  - Write software that can access the hardware through APIs and think it is the native OS
  - Lot of overhead and memory requirements
- However, as the code got more efficient:
  - We could create a mini virtual machine with low memory requirements
- The mini virtual machine is “Docker”
  - A small virtual computer that can be recreated anywhere using the original requirements

Unix Never Dies

- MacOS and Linux are based on Unix.
- Windows needs some extra support but can run Linux.
- Thus Docker is a way to create “mini Linux” containers from scratch.
  - Easily duplicated
  - Easily built
  - Runs anywhere

Docker Containers

- A container is just a Docker instance.
  - Aka a “mini computer”
  - Typically runs a single program
  - Built with specific requirements
    - Dockerfile
  - Simple enough to be built from scratch or copied as an image
    - Docker Image!
- Working inside a Docker container is no different than working on Linux.
- Docker containers are now the default containers in IT.

Example

- Base image
- Copy in Python requirements
- Update all packages
- Change work directory
- Copy in Flask code (Python)
- Copy in my saved models
- Copy in my saved models
- Copy in my saved models
- Expose a port to receive requests
- Start my Python program

```
FROM python:3.7.5-slim
COPY requirements.txt .
RUN python3 -m pip install -r requirements.txt
WORKDIR /usr/src/app
COPY model_serve.py .
COPY glm.model .
COPY imputer.model .
COPY scaler.model .

EXPOSE 5000
CMD python3 -m model_serve
```

A Note on the Future

- When you have multiple Docker containers, you need a program to manage them.
  - You need a container “orchestrator.”
  - Kubernetes is the container manager of choice.
- Just like we have functions that work together to form a program, containers work together to form software.
  - Kubernetes assists with “microservice” architecture.
  - Each container is a microservice.
  - The microservices talk to each other.
  - Kubernetes starts and stops containers as needed by traffic demands.

DataScience@SMU