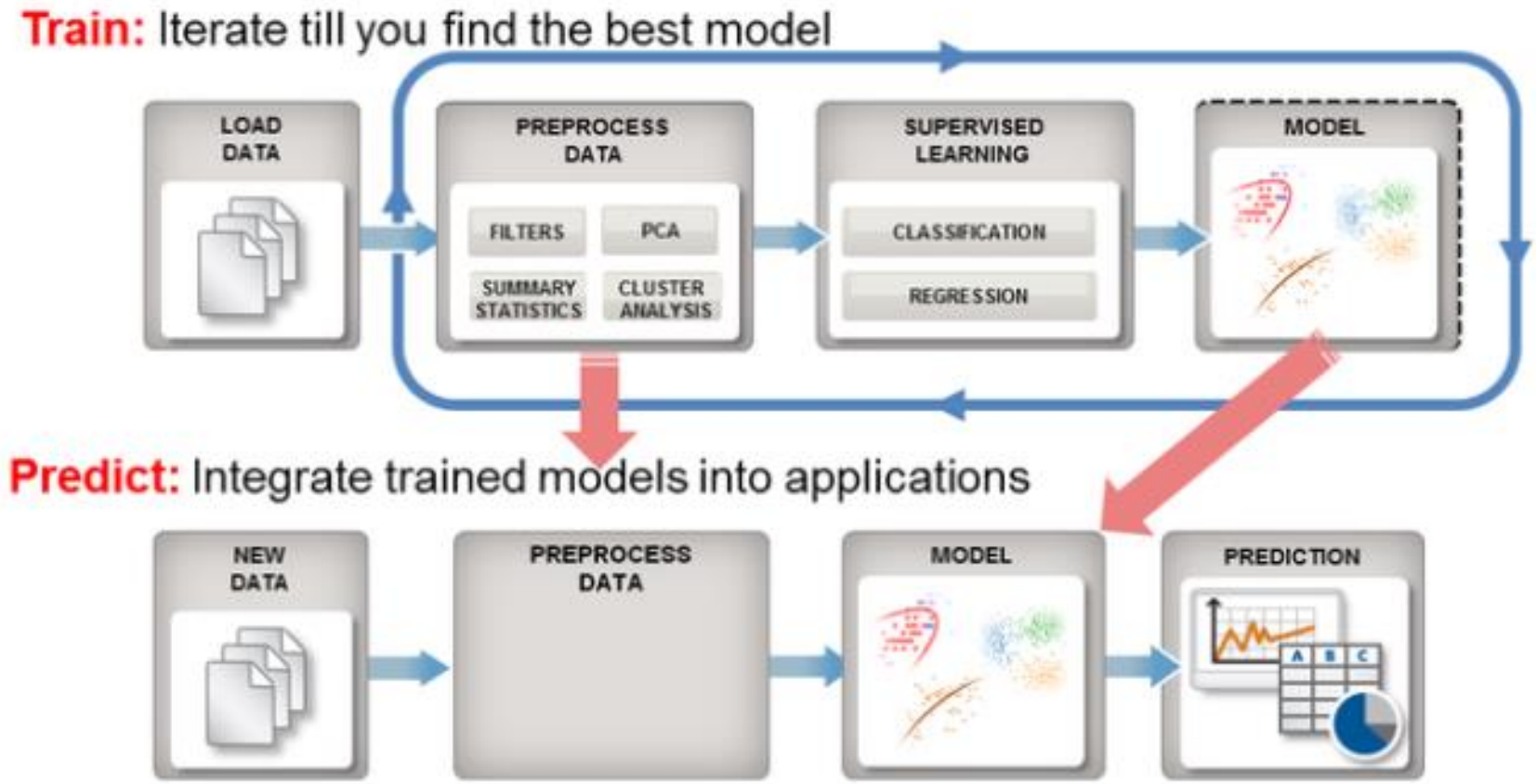
deployment



- standalone predict
  - pyinstaller

deployment issue

- 생성된 예측모델로 현장에 설치 및 예측 실행
  - 필요한 라이브러리 들여오기
  - 예측모델 파일 적재
  - 센서에서 읽은 원시 스펙트럼 읽기
  - 학습시와 동일한 전처리 수행
  - 예측

- 설치하고자 하는 컴퓨터마다 모든 환경을 동일하게 맞추어야 하는 번거로운 작업 발생.

- 효율적인 예측모델 서비스를 도와주는 라이브러리들
  - fast_api
  - streamlit
  - docker

0. fast_api_ex

- (실습) fast_api_ex(car-price-prediction)
- 가상환경 : ml (pip install fastapi uvicorn)
- https://github.com/furkankizilay/car-price-prediction

- 예측모델의 구축
  - car-price_prediction.ipynb
  - 모델 저장파일 : LinearRegressionModel.joblib

- 예측모델 서비스

  $ cd server/ → python main.py

- 예측 요청

  $ clinet/app.py

  vscode 디버깅 설정(strreamlit)

- VSCode에서 디버깅 모드로 확인하는 설정
  - streamlit 사용파일에서는 $ streamlit run app.py 형태로 실행해야 함.
  - vscode에서 디버깅 모드로 실행하기 위해서는 설정 변경이 필요함.
  - (app.py 에서) launch.json 파일 수정
  - program 항목을 streamlit 위치한 폴더로 지정

    " program " : " ~/miniconda3/envs/ml/lib/python3.8/site-packages/streamlit " ,
  - 실행명령 정의

    " args " : [" run " , "app.py "]

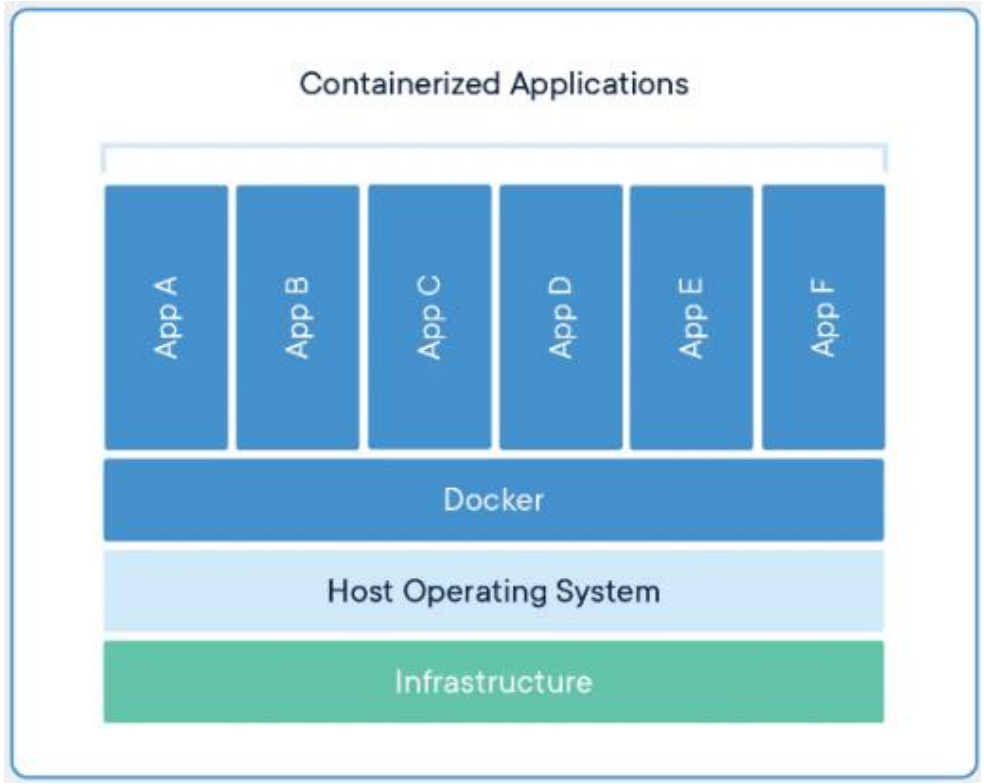
1. docker 소개

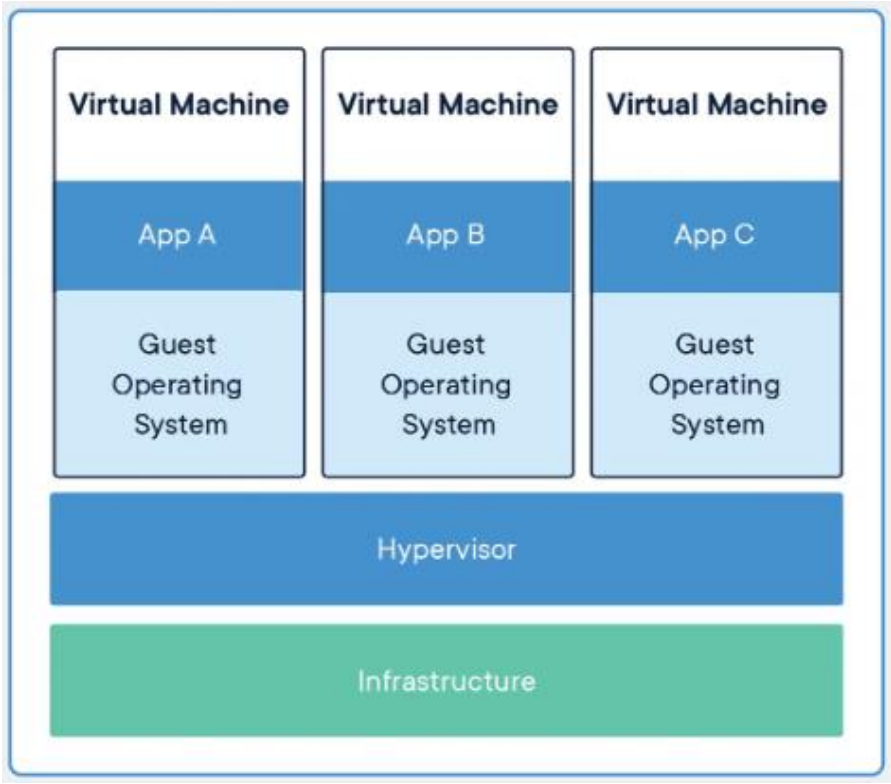
- 개발/실행환경을 배포하거나 공유하는데 애로점
  - 프로그램이 구동되는데 필요한 모든 라이브러리들, 버전. 기타 구동환경(OS 버전 등)이 일치해야 함.
  - 운영체제에 따라 요구되는 라이브러리들이 없거나 버전이 다를 수도 있음.
  - 저장된 학습모델을 새 파일에서 불러서 예측해보면.

- 도커는
  - 애플리케이션을 신속하게 구축, 테스트 및 배포할 수 있는 소프트웨어 플랫폼
  - 격리된 가상환경에서 응용이 독립적으로 실행되도록 해줌.
  - 컨테이너라는 표준화된 유닛으로 패키징하며 필요 라이브러리, 시스템 도구, 코드, 런타임 등 응용이 독립적으로 실행되는데 필요한 모든 자원이 포함되어있음.
  - 도커를 통해 애플리케이션을 실행하면 운영체제, 버전 등에 무관하게 독립적인 환경에서 일관된 결과를 보장

1. docker 소개 - 컨테이너

- 기존의 가상화 방식인 os 가상화가 아닌 프로세스를 격리하는 방식으로 동작
- 단순히 프로세스를 격리하기 때문에 가볍고 빠르다.
- CPU나 메모리는 프로세스가 필요한 만큼만 추가 사용하여서 성능면에서 거의 손실이 없다.





https://www.docker.com/resources/what-container/

1. docker 소개 - image, Dockerfile

- 이미지 / 컨테이너
  - 이미지는 컨테이너 실행에 필요한 파일과 설정을 포함하고 있음.
  - 컨테이너는 이미지를 실행한 상태이다. 추가되거나 변하는 값은 컨테이너에 저장된다.
  - 컨테이너의 상태가 바뀌거나 삭제되어도 이미지는 변하지 않고 그대로 남아있다.
  - 도커 이미지는 Docker hub에 등록하거나 Docker Registry 저장소를 직접 만들어 관리할 수 있다.

Dockerfile

- 도커 이미지를 만들기 위한 설계도 파일. DSL(Domain Specific Language) 언어를 이용.
- FROM <image>:<tag>
- FROM ubuntu:16.04
- 반드시 베이스 이미지를 지정해야 하며 어떠한 이미지도 베이스 이미지가 될 수 있다. tag는 버전을 지정.

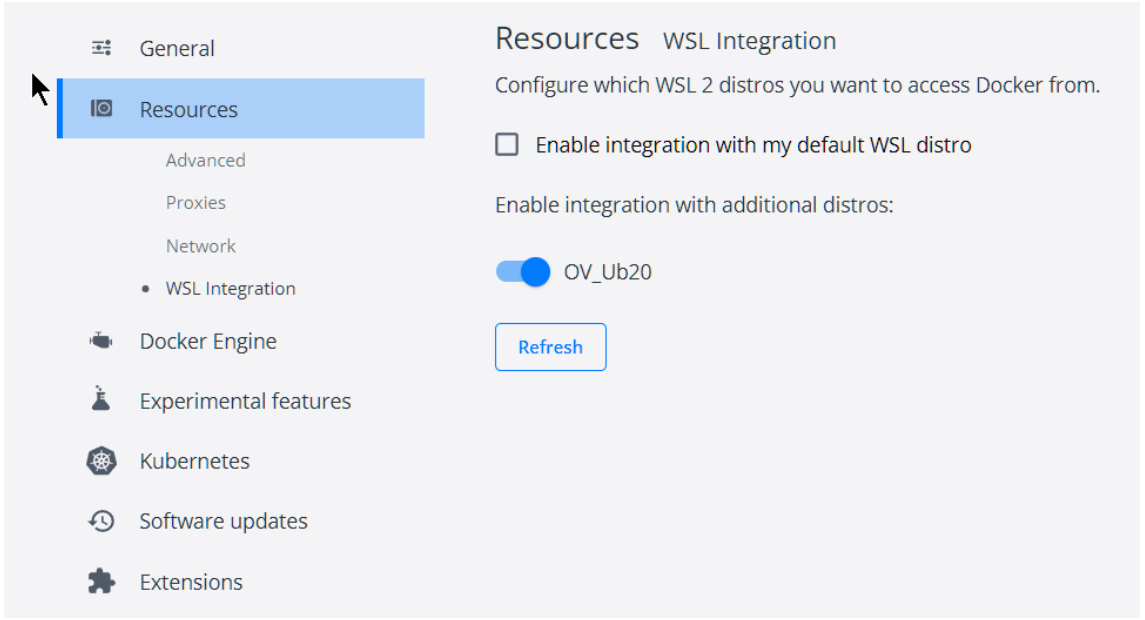
- RUN <command>
- RUN pip install --no-cache-dir -r requirements.txt
- 명령어를 실행. 내부적으로 /bin/sh -c 뒤에 명령어를 실행하는 방식.

1. docker 소개 - 설치

- Docker desktop download - https://www.docker.com/
- 설치
  - check Use WSL 2 based engine(default)
  - (설치 완료)
  - docker desktop 실행
    Settings > **Resources** > **WSL Integration** > 원하는 wsl distro 설정

- VSCode 확장 설치
  - VS Code extension > docker 설치
  - VS Code extension > docker explorer 설치

- 설치 확인(in powershell / wsl)
  - $ docker version



2. 도커사용 예제

- 실습 : docker_ex(node_js)

- << 웹서버/클라이언트 기동예제 >>
- < 웹서버 시동 >
  - (docker_ex 폴더 (node.js 파일) 위치에서)
  - $ node node.js

- < 브라우저에서 동작확인 >
  - (브라우저에서) localhost:8081

- << 도커로 구현 및 동작확인 >>
  - 서버 (Dockerfile 위치에서)
  - $ docker build –t my_node:v1 .
  - $ docker images
  - $ docker run –name mydocker_v1 –d –p 8081:8081 mynode:v1

  - 클라이언트 (브라우저에서 동작확인)
  - (브라우저에서) localhost:8081

< 실습시 사용할 주요 명령어 >

이미지로 컨테이너 만들고 실행하기
$ docker create --name my_nginx -p 80:80 nginx

컨테이너 프로세스 상태 확인
$ docker ps -a

컨테이너를 start 명령어로 시작
$ docker start my_nginx

이미지 다운, 컨테이너 생성, 컨테이너 시작을 한번에
$ docker run --name my_nginx -d -p 80:80 nginx

컨테이너 중지
$ docker stop my_nginx
컨테이너 삭제
$ docker rm my_nginx

3. pycaret - fast_api 예제

- pycaret 에서는 REST Api 방식으로 예측모델 서비스하는 기능을 제공하고 있음.

- (실습) 3.1.caret_fastapi_ex 폴더
- caret 가상환경에서 작업
- caret_fastapi_ex.ipynb 실행
- 서버모듈인 et_api.py 가 자동생성됨
- 서버모듈 실행
  - (caret) $ python et_api.py
- client 접속
  - (브라우저에서) localhost:8000/docs
  - POST /predict 오른쪽 화살표 펼치기
  - try It Out
  - 예측 시험

  - X_to.csv 를 메모장으로 열어서 한 줄 복사
  - predict 입력창에 붙여넣기.

port busy 나올 때
$ lsof –i:port_number
$ kill <PID>

3. pycaret – docker + brix 실습

- (실습) 3.2.docker_caret
- brix 데이터를 대상으로 fast_api 구현
  - 예측모델 구축에 사용한 노트북 복사
  - 모델 생성 코드 외에 중요치 않은 코드 생략.
  - final model 까지 생성 후,
  - # pip install fastapi uvicorn
- fast_api 생성
  - create_api(saved_final_et, 'best_api')
  - create_docker('best_api')

- brix 대상 fast_api 예측모델 구동시험
  - $ python best_api.py
  - (브라우저에서) localhost:8000/docs
  - best_api.py 실행시 error
  - ← 숫자로 된 변수는 허용되지 않음.

  - (실습) 학습데이타를 읽은 후 X의 칼럼명을 문자열로 변환
  - 1. def make_str(X.columns):
  - 2. lambda 함수 이용

- 또한
  - pycaret에서 기본 생성된 client 모듈은 135 개의 특성을 입력해야 함.
  - 시험이 불편하므로 한줄의 csv 형태로 입력가능한 간단한 client 모듈 제작.사용(client.py).

- 개선된 모듈
  - server/
  - server.py : server에서는 135개 스펙트럼 값을 한 줄로 받아들임.
  - client.py : client는 한줄로 스펙트럼 값을 보냄

3. pycaret - docker + brix 실습

- pycaret 에서 자동생성된 Dockerfile 사용하여 도커 이미지 생성
  - 서버프로그램으로 best_api.py 파일 이용
  - '3.2.docker_caret' folder 에서 실습
  - docker image build –f "Dockerfile" –t my_caret:v1 .
  - docker run –name my1 –d –p 8000:8000 my_caret:v1
  - docker exec –it my1 /bin/bash
  - cd /etc
  - ls *-release
  - docker top py1 # 활성 프로세스 확인.

- 문제점
  - pycaret에서 생성된 도커이미지는 best_api.py에서 정의한 입력을 기대함.
  - 135개의 특성필드 채워야 함.

- server/server.py 파일로 시작하는 도커이미지 생성
  - '3.2.docker_caret/server' folder 에서 실습
  - 서버프로그램으로 server/server.py 파일 이용
  - 하나의 필드에서 135개의 입력값을 받는 방식으로 수정.
  - docker build –f "Dockerfile_new" –t my_server:v1 .

- 결과 확인
  - (브라우저에서)
  - localhost:8000/docs
  - 예측 시험

  - X_to.csv 를 메모장으로 열어서 한 줄 복사
  - predict 입력창에 붙여넣기.

- 결과 확인
  - (브라우저에서)
  - localhost:8001/docs
  - 예측 시험

  - X_to.csv 를 메모장으로 열어서 한 줄 복사
  - predict 입력창에 붙여넣기.

4. autokeras – fastapi + brix 실습

- (실습) 4.keras_docker 폴더 에서.
- code . # code 에서 실습.
- 가상환경 : ak
- basic+ak.py 파일을 ak_predict.py 로 복제.
- 예측기능 제공에 필요한 코드, 모듈 확인.
- 학습부분 제외하고 예측부분 코드만 확인.
- 예측부분의 코드를 보완하면 예측 제공 실행모듈 완성

```
import tensorflow as tf
import autokeras as ak
import pandas as pd
X = pd.read_csv( ' ./to_X.csv')
# should include autokeras custom objects...
new_model = tf.keras.models.load_model( ' berry.h5 ' , custom_objects=ak.CUSTOM_OBJECTS)
new_model.predict(X)
```

4. autokeras – fastapi + brix 실습

- (실습) keras_docker – code 에서 작업
- '4.keras_docker/server_api' 폴더에서 실습.
- 서버 프로그램
  - keras_docker/server_api/server_api.py
  - 실행 : python server_api.py
- client 프로그램
  - client.py
  - 실행 : streamlit run client.py

- server : 중요 내용
  - new_model = tf.keras.models.load_model(' berry.h5 ' , custom_objects=ak.CUSTOM_OBJECTS)
  - (클라이언트로부터 받은 데이터 가공)
  - new_model.predict(X)
  - uvicorn.run("server:app", host="127.0.0.1", port=8001, reload=True) # 파일명:app 형식이어야 함.

docker – keras + brix 실습

- 최종적으로 도커이미지 만들 프로그램 검증.
- '4.keras_docker/server' 폴더에서 실습
  - 3.2.docker_caret/server/server.py 파일을 복사
  - code 에서 복사 후 server_caret.py 파일로 이름 변경
  - server.py 파일에서 작업
  - 실행 → 오른쪽 참조
- server.py 수정내용
  - import tensorflow as tf
  - import autokeras as ak
  - ....
  - from pycaret.regression import load_model, predict_model 제거
  - model = load_model('best_api') →
  - new_model = tf.keras.models.load_model('berry.h5', custom_objects=ak.CUSTOM_OBJECTS)
  - 예측부분 수정 : keras 에서 받는 데이타형식으로 수정
  - data = data0.split(' ,')
  - data2 = list(map(float, data))
  - predictions = new_model.predict([data2])
  - return {'prediction': predictions.flatten().tolist()}
- 서버 실행
  - python server.py
- client 실행
  - (브라우저에서)
  - localhost:8001/docs
  - 예측 시험

  - X_to.csv 를 메모장으로 열어서 한 줄 복사
  - predict 입력창에 붙여넣기.

docker – keras + brix 실습

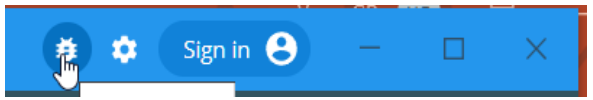
- 도커 이미지 제작
  - Dockerfile 생성

```
FROM python:3.8-slim
WORKDIR /app
ADD . /app
RUN apt-get update
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 8001
CMD ["uvicorn", "server_A:app", "--host", "0.0.0.0", "--port", "8001"]
```

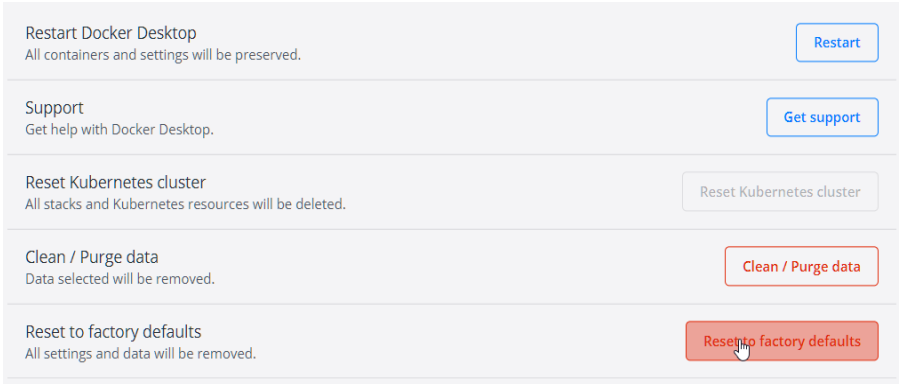
  - $ docker build –t my_keras –f 'Dockerfile_keras' .
- 서버(도커이미지) 실행
  - $ docker run –d –p 8001:8001 my_keras
- client 접속(브라우저에서)
  - localhost:8001/docs
  - 예측 시험

- X_to.csv 를 메모장으로 열어서 한 줄 복사
- predict 입력창에 붙여넣기.

- 도커 생성 로그 보는 법
  - $ docker logs –f <container>
- 도커 꼬였을 때 초기화 방법
  - Docker dashboard
  - troubleshoot 선택



  - reset to factory defaults



주요 docker 명령어

컨테이너 프로세스 상태 확인
$ docker ps -a

보유 이미지 확인
$ docker images

도커 컨테이너에서 실행 중인 프로세스 목록
$ docker top my_nginx
컨테이너에 bash로 접속(나올때는 ctrl+q)
$ docker exec -it my_nginx /bin/bash

이미지 삭제
$ docker rmi nginx
컨테이너 중지
$ docker stop my_nginx
컨테이너 삭제
$ docker rm my_nginx

이미지 다운, 컨테이너 생성, 컨테이너 시작을 한번에
$ docker run --name my_nginx -d -p 80:80 nginx

이미지로 컨테이너 만들고 실행하기
$ docker create --name my_nginx -p 80:80 nginx

컨테이너를 start 명령어로 시작
$ docker start my_nginx

도커 안에서 ps 실행
$ apt-get update && apt-get install procps

- Docker in VSCode
  - 일반 파이썬 파일 같이 디버깅 가능
  - 확장 설치

    Docker, Docker Explorer, Remote Container
  - F1 → Remote Containers : Attach to Running Container
  - my1 container 선택
  - file > Open Folder > /app 선택

- VSCode 에서 container debugging
  - 이미지 실행단계부터 시작하여 추적함.
  - 컨테이너 실행시 server 가 자동실행되기 전에 디버깅 가능함.
  - - F1 > Remote containers : Reopen in container
  - - from Dockerfile