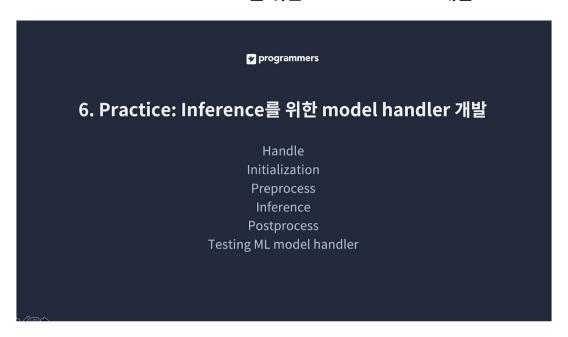
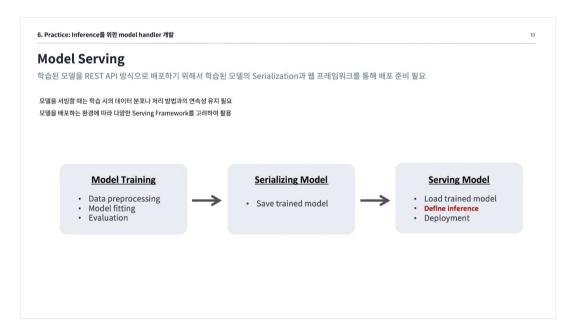
Part 6. Practice: Inference를 위한 model handler 개발



이번 실습에서 다루는 부분

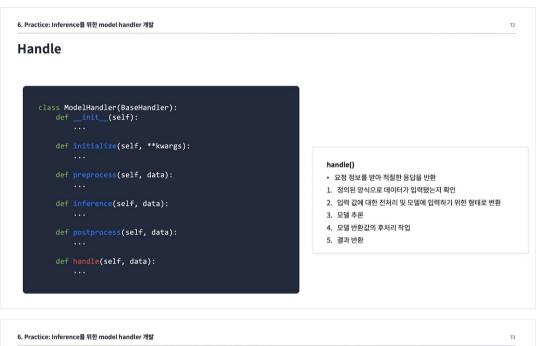


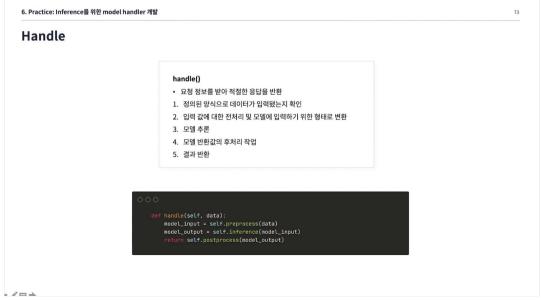
각각의 메소드 들에 대해서 다루고 각각의 메서드들이 어떤 동작을 할 수있게끔 해야 하는지 살펴볼 것이다.

```
Skeleton of model handler to serve model

class ModelHandler(BaseHandler):
    def __init__(self):
        ...
    def initialize(self, **kwargs):
        ...
    def inference(self, data):
        ...
    def postprocess(self, data):
        ...
    def handle(self, data):
        ...
```

입력 값에 대해서 전처리나 모델을 실제 inference하기 위한 벡터 형태로 변환한다던가 하는 작업을 수행, 또한 실제 모델의 inference를 하고 반환된 값에 대해서 이를 실제 우리 서비스 레벨에서 필요한 후처리 작업을 정의하고 수행하는 작업을 진행





```
6. Practice: Inference를 위한 model handler 개발
Initialization
     class ModelHandler(BaseHandler):
         def __init__(self):
                                                                                 initialize()
                                                                                 • 데이터 처리나 모델, configuration 등 초기화
         def initialize(self, **kwargs):
                                                                                 1. Configuration 등 초기화
                                                                                 2. (Optional) 신경망을 구성하고 초기화
                                                                                 3. 사전 학습한 모델이나 전처리기 불러오기 (De-serialization)
          def preprocess(self, data):
         def inference(self, data):
                                                                                Note
         def postprocess(self, data):

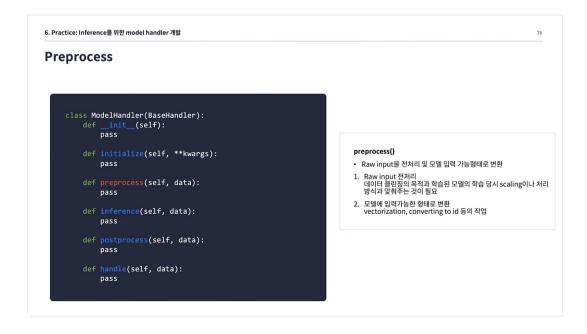
    모델은 전역변수로 불러와야 합니다. 만약 inference를 할
때마다 모델을 불러오도록 한다면 그로 인해 발생하는
시간이나 자원 등의 낭비가 발생합니다.

          def handle(self, data):
                                                                                • 일반적으로 요청을 처리하기 전에 모델을 불러 둡니다.
```

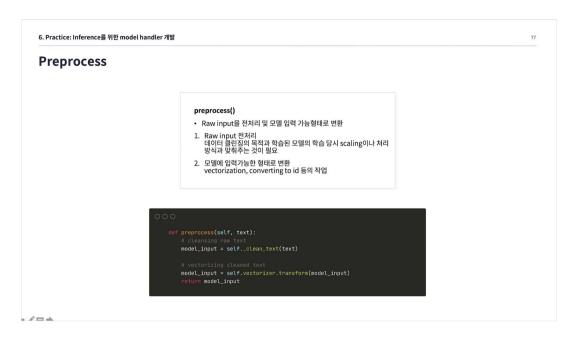
초기화하는 코드가 만약에 호출이 즉, 요청이 올때마다 초기화를 한다면 그렇게 되면 모델을 계속 de-serialization을 계속 하게 될땐대 그럼 그 시간만큼 지연이 생길 것이다. 그래서 시간이나 자원등의 낭비가 발생할 수있기 때문에 전역변수로 설정해서 메모리에 공유해서 호출할때 사용할 수 있도록 한다고 한다!



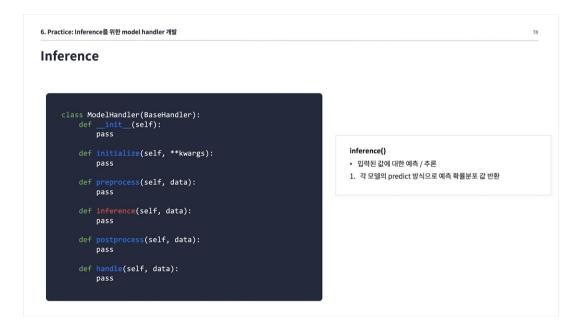
해당 코드를 initialize()함수에 고대로 적어준다.



그 다음 모델이 읽을수 있는, 추론 할 수있는 형태로 변환해 주어야 하는대 보통 데이터 클랜징이나 데이터 분포를 반영한 스케일링을 하거나 모델마다 다른 feature를 가질 수있는대 그런것들을 학습할 때와 동일하게 맞추어 주는 전처리 작업을 진행



해당 코드를 preprocess()함수에 적는다.



보통 모델의 예측값은 확률분포 값이나 딥러닝 같은 경우 confidence, 경향성을 보여주는 값이나오게 된다. 그래서 이 부분도 어떤 모델을 쓰냐 어떤 task를 하느냐에 따라서 달라 질 수 있다.

```
6. Practice: Inference를 위한 model handler 개발
Postprocess
     class ModelHandler(BaseHandler):
        def __init__(self):
    pass
                                                                                postprocess()
         def initialize(self, **kwargs):
              pass
                                                                               • 모델의 예측값을 response에 맞게 후처리 작업
                                                                                1. 예측된 결과에 대한 후처리 작업
         def preprocess(self, data):
    pass

    보통 모델이 반환하는 건 확률분포와 같은 값이기 때문에
response에서 받아야 하는 정보로 처리하는 역할을 많이 함

         def inference(self, data):
             pass
         def postprocess(self, data):
              pass
         def handle(self, data):
```

예측값 자체는 확률 분포나 어떤 인덱스로 표현되는 숫자로 표현되므로 이러한 수치를 실제 api서버에서 response로 반환하는 형태에 맞춰서 추가적인 처리를 해야한다! 따라서 아래 코드는 라벨값(궁부정)과 그에 대한 확률도 같이 가져오는 코드이다



그래서 핸들러를 다 개발하였고 실제 핸들러가 제대로 동작하는지 테스트를 해보자!

```
Testing ML model handler

from model import MLModelHandler

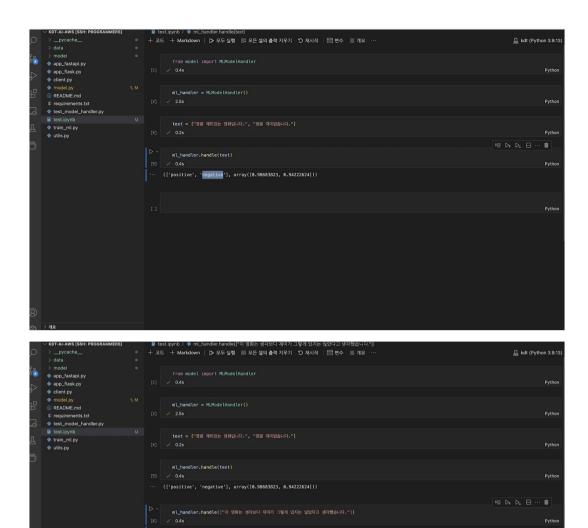
ml_handler = MLModelHandler()

text = ['정말 제미안는 영화입니다.', '정말 제미가 없습니다.']

result = ml_handler.handle(text)

print(result)

# (['positive', 'negative'], array([8.98683823, 8.79668478]))
```



나이브 베이지안 모델 같은 경우는 키워드의 발생빈도나 어떤 키워드가 발생했는지 안 했는지라는 feature를 이용하기 때문에 아무래도 성능방식에서 딥러닝 성능방식에 비해 조금 아쉬운 부분은 존재한다.

이제 핸들러 까지 개발 했으니 API를 어떻게 만드는지 같이 실습을 해보도록 하자