

5. Practice: Serialization & De-serialization

Serialization
De-serialization

<https://github.com/sackoh/kdt-ai-aws>

README.md

K-Digital Training: 프로그래머스 인공지능 데브코스

AWS를 활용한 인공지능 모델 배포 강의 실습

AWS 실습 개발환경 준비

1. Miniconda를 설치합니다.

```
wget https://repo.anaconda.com/miniconda/Miniconda3-py39_4.12.0-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
```
2. 실습 repo를 다운로드합니다.

```
sudo yum install git
git clone https://github.com/sackoh/kdt-ai-aws
cd kdt-ai-aws
```

파이썬 패키지 설치

requirements.txt에 있는 실습에 필요한 라이브러리를 설치합니다.

```
pip install --no-cache-dir -r requirements.txt
```

데이터준비 / 모델학습 코드의 내부 프로세스

1. 실습에 사용할 네이버 영화 리뷰 데이터를 다운로드 합니다. (<https://github.com/e9t/nsmc>)
2. scikit-learn 라이브러리를 활용하여 나이브 베이즈 모델을 학습합니다.
3. 학습에 사용하지 않은 테스트 데이터를 통해 모델 성능을 평가합니다.
4. 학습한 모델을 저장합니다.

```
python train_ml.py
```

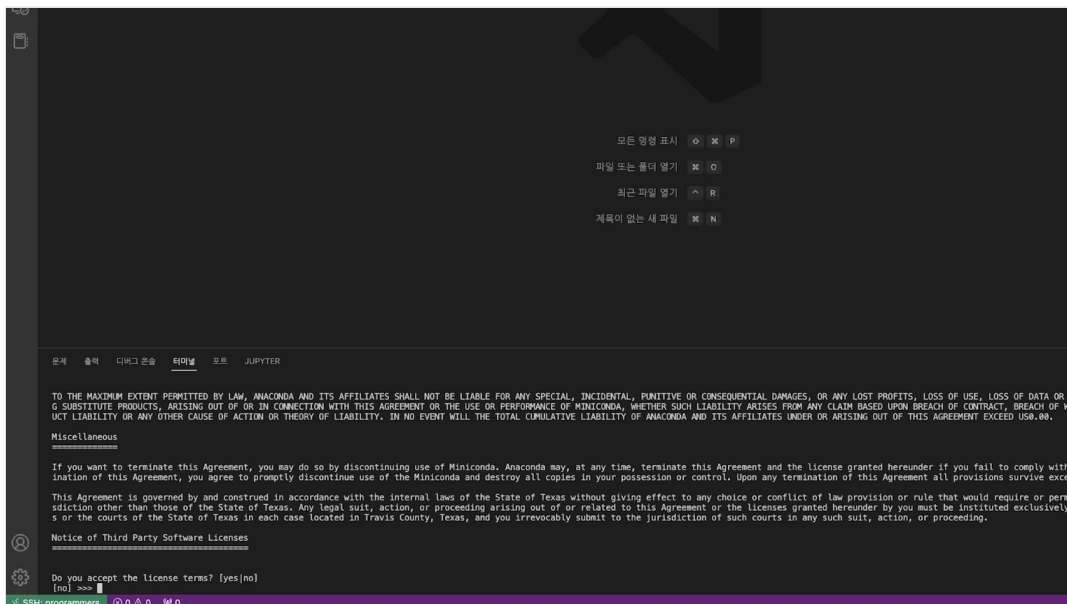
Contributors 2

- sackoh Sungwoo Oh
- hyunchulshin hcsln9090

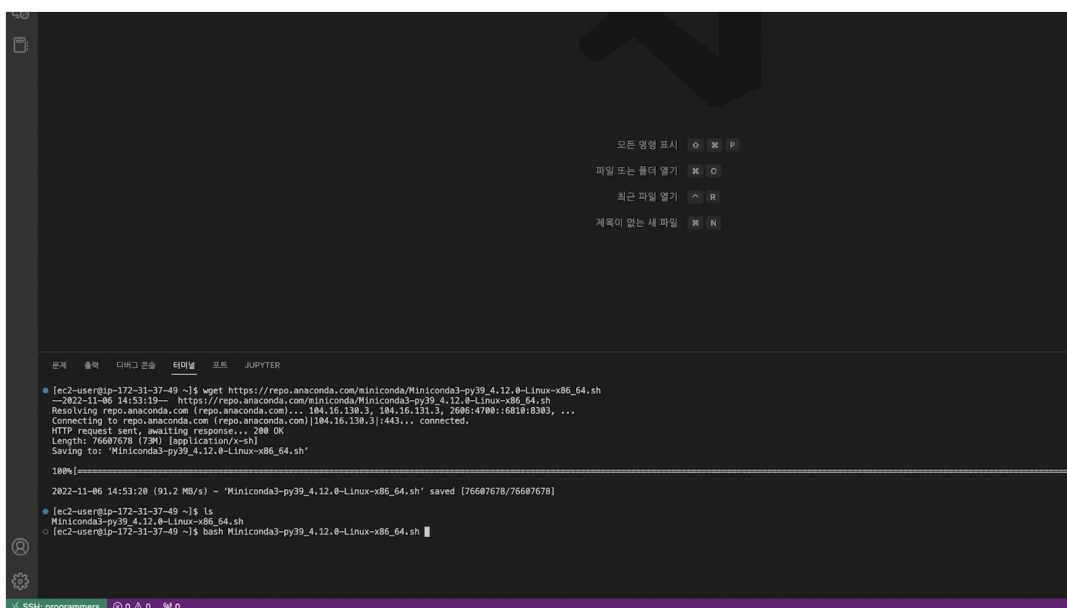
Languages

- Python 100.0%

우리가 만들었던 아마존 인스턴스 환경에서 미니콘다를 다운받고(1번째 코드) 설치한다.(2번째 코드)



q를 눌러주면 라이선스에 동의하는지 화면으로 바로 내려간다.(약관 바로 아랫줄로 간다)

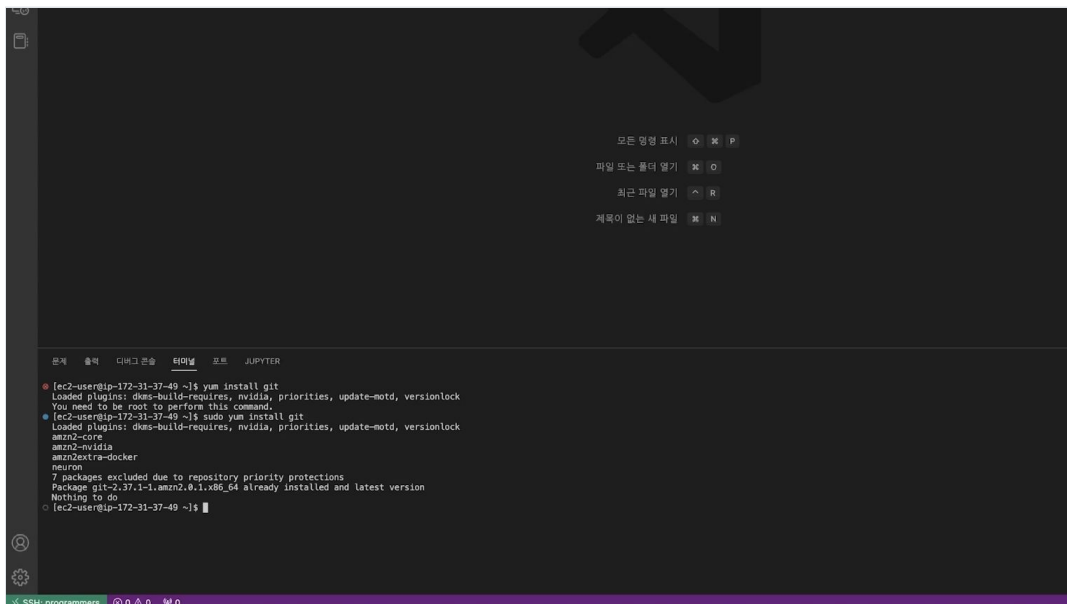


yes를 타이핑 그리고 어디에 로케이션을 잡을지 인대 enter누르면 된다 그럼 설치가 진행된다

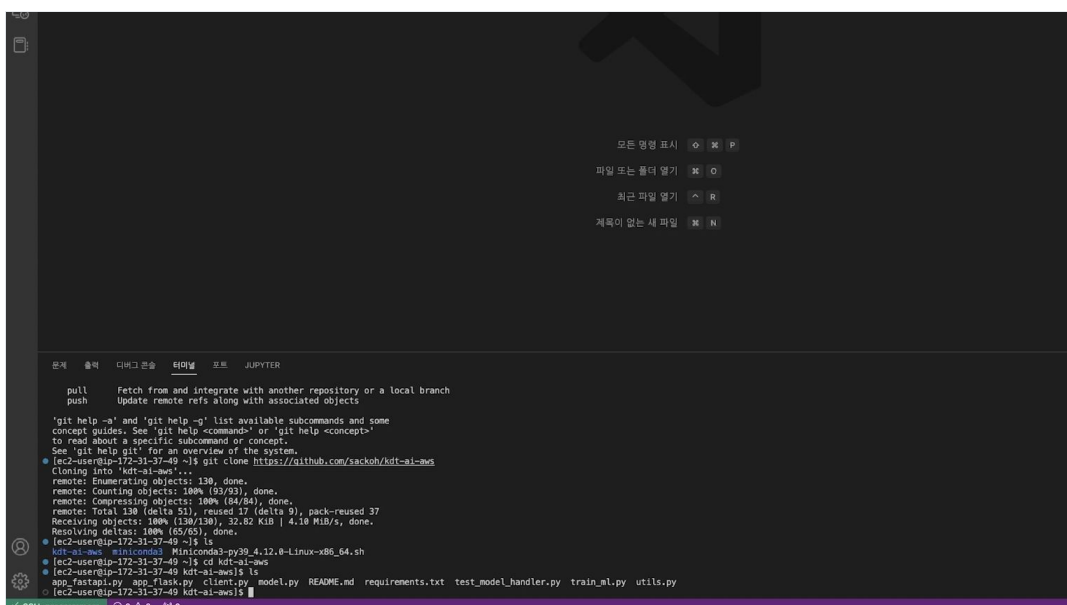
또 yes누르고 그럼 설치가 끝남

미니콘다는 아나콘다에서 필수적인 기능들만 빼서 사이즈를 많이줄인 패키지 이름이다

인스턴스에 git을 먼저 설치하자



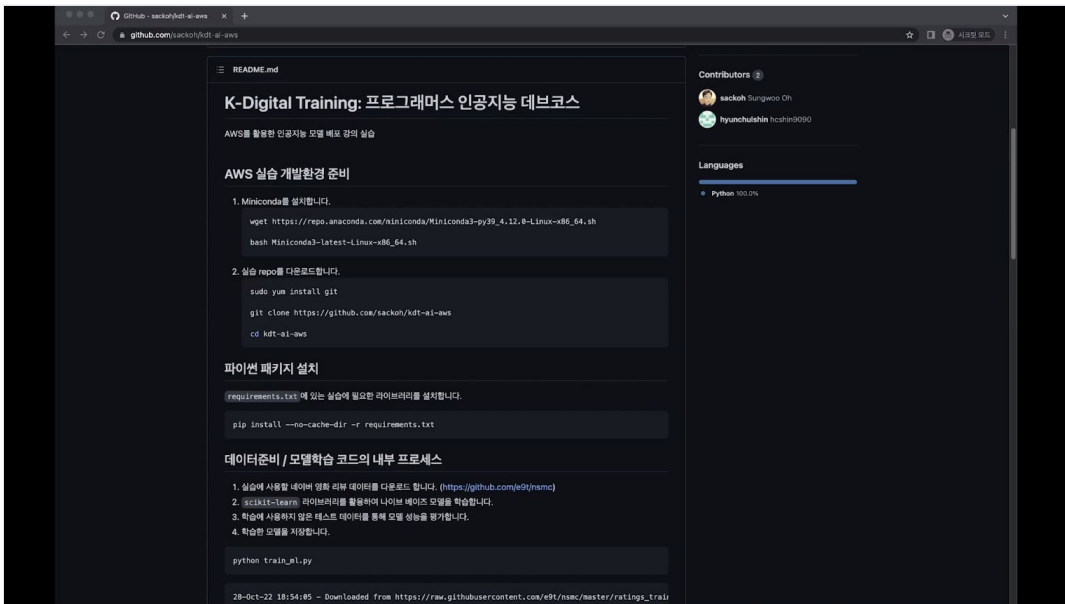
이제 강의 관련된 실습코드를 다운로드 받도록 한다.



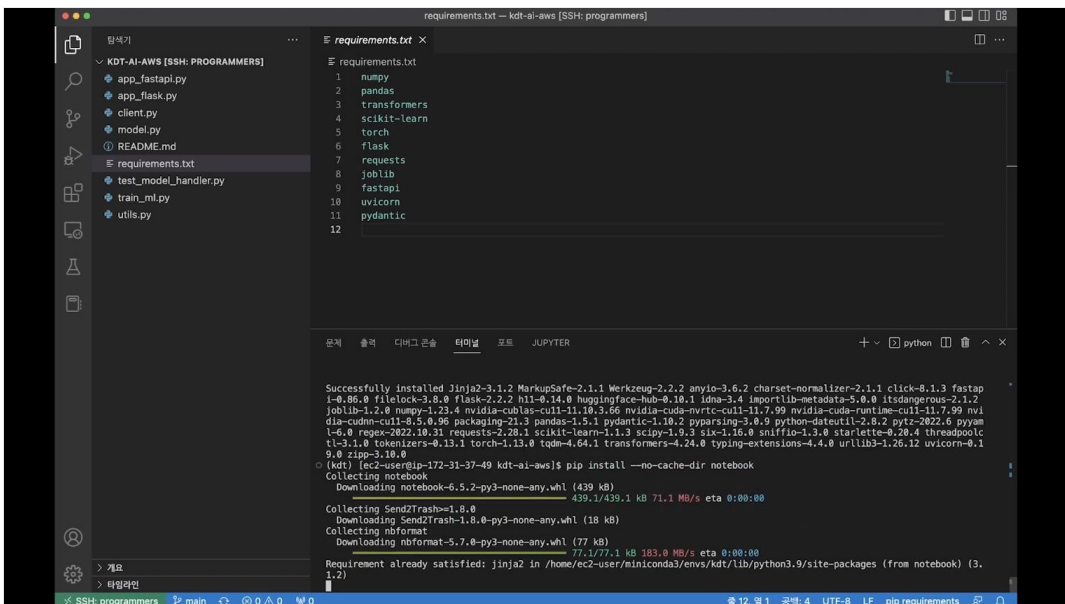
먼저 가상환경을 만든다 가상환경 이름은 kdt, 파이썬 버전은 3.9를 사용

그리고 pip install --no-cache-dir -r requirements.txt 명령어를 사용해서 인스턴스의 공간이
녹록치 않기 때문에 파이썬 설치 파일들을 캐시로 저장하지 않도록 하는 인자를 추가해서 필요한
라이브러리 설치를 진행한다

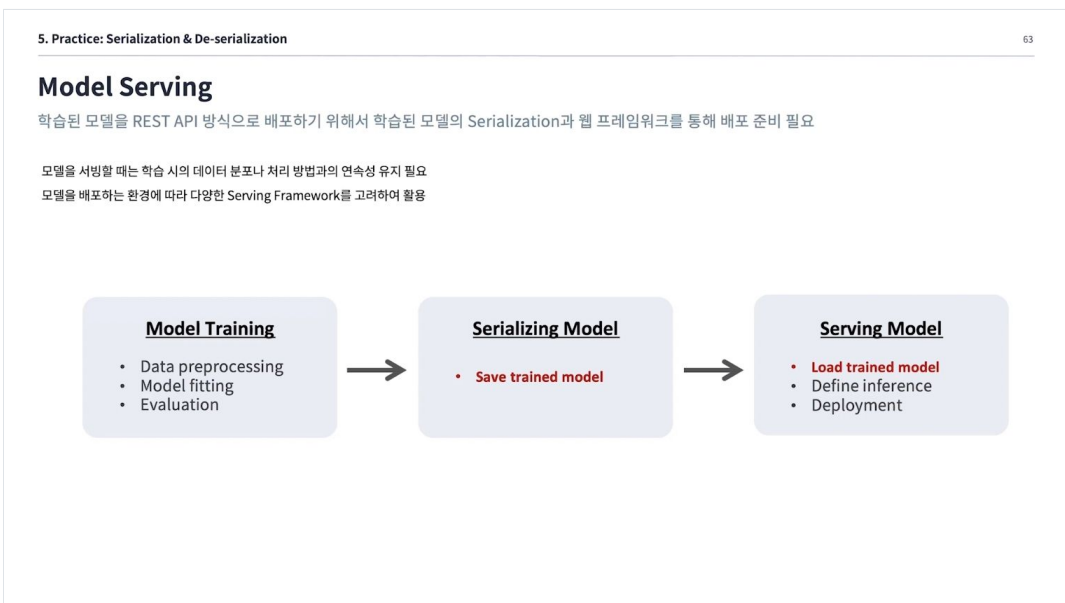
aws를 사용하는 장점중 하나가 global infrastructure를 통한 고속 네트워킹이 가능하다는 건대
실제 설치 속도가 초당 100메가가 나온다고 함



주피터 노트북도 설치 진행



test.ipynb 노트북 파일을 만든다 -> 커널은 우리가 만들었던 가상환경 kdt를 선택!



머신러닝 모델 학습

실습 진행을 위해 사전 준비한 코드를 실행하여 모델 학습 및 저장

```
$ python train_ml.py
```

Model Training

- Data preprocessing
- Model fitting
- Evaluation

Serializing Model

- Save trained model

Serving Model

- Load trained model
- Define inference
- Deployment

```
train_ml.py
1 import os
2 import logging
3 import requests
4 from datetime import datetime
5 import pandas as pd
6 from sklearn.feature_extraction.text import CountVectorizer
7 from sklearn.naive_bayes import MultinomialNB
8 from utils import clean_text
9
10 logger = logging.getLogger(__name__)
11
12
13 def download_data(mode):
14     base_url = f'https://raw.githubusercontent.com/elt/mnsc/master/ratings_{mode}.txt'
15     r = requests.get(base_url)
16     os.makedirs('data', exist_ok=True)
17     with open(f'data/ratings_{mode}.txt', 'wb') as w:
18         w.write(r.content)
19     logger.info(f'downloaded from {base_url}')
20
21
22 def train_and_evaluate():
23     train = pd.read_csv('data/ratings_train.txt', sep='\t', drop('id', axis=1), fillna(''))
24     test = pd.read_csv('data/ratings_test.txt', sep='\t', drop('id', axis=1), fillna(''))
25     X_train, y_train = train['document'].apply(clean_text), train['label']
26     X_test, y_test = test['document'].apply(clean_text), test['label']
27
28     vectorizer = CountVectorizer()
29     X_train = vectorizer.fit_transform(X_train)
30     X_test = vectorizer.transform(X_test)
31     model = MultinomialNB()
32     model.fit(X_train, y_train)
33     logger.info(f'Trained Naive Bayes model')
34     evaluation_score = model.score(X_test, y_test)
35     logger.info(f'ML model accuracy score: {evaluation_score*100.2f}%')
36
37     return model, vectorizer
38
39
40 def serialization(model, vectorizer):
41     import joblib
42     os.makedirs('model', exist_ok=True)
43     joblib.dump(vectorizer, 'model/ml_vectorizer.pkl')
44     logger.info(f'Saved vectorizer to model/ml_vectorizer.pkl')
45     joblib.dump(model, 'model/ml_model.pkl')
46     logger.info(f'Saved model to model/ml_model.pkl')
47
48
49 if __name__ == '__main__':
50     logging.basicConfig(format='%(asctime)s - %(message)s', datefmt='%d-%b-%y %H:%M:%S', level=logging.INFO)
51
52     start_time = datetime.now()
53     # Download train and test data from github
54     for mode in ['train', 'test']:
55         download_data(mode)
56
57     model, vectorizer = train_and_evaluate()
58     serialization(model, vectorizer)
```

소스코드는 나이브 베이지안을 이용해서 분류기를 만드는 방식, 우리가 실습으로 다루고자 하는 부분은 def serializaton()이다. joblib이라는 패키지를 이용해서 모델을 직렬화 한다. 따라서 학습하는 모델에서 모델 직렬화와 저장까지 한번에 이루어 지도록한다.

```
train_ml.py
1 import os
2 import logging
3 import requests
4 from datetime import datetime
5 import pandas as pd
6 from sklearn.feature_extraction.text import CountVectorizer
7 from sklearn.naive_bayes import MultinomialNB
8 from utils import clean_text
9
10 logger = logging.getLogger(__name__)
11
12
13 def download_data(mode):
14     base_url = f'https://raw.githubusercontent.com/elt/mnsc/master/ratings_{mode}.txt'
15     r = requests.get(base_url)
16     os.makedirs('data', exist_ok=True)
17     with open(f'data/ratings_{mode}.txt', 'wb') as w:
18         w.write(r.content)
19     logger.info(f'downloaded from {base_url}')
20
21
22 def train_and_evaluate():
23     train = pd.read_csv('data/ratings_train.txt', sep='\t', drop('id', axis=1), fillna(''))
24     test = pd.read_csv('data/ratings_test.txt', sep='\t', drop('id', axis=1), fillna(''))
25     X_train, y_train = train['document'].apply(clean_text), train['label']
26     X_test, y_test = test['document'].apply(clean_text), test['label']
27
28     vectorizer = CountVectorizer()
29     X_train = vectorizer.fit_transform(X_train)
30     X_test = vectorizer.transform(X_test)
31     model = MultinomialNB()
32     model.fit(X_train, y_train)
33     logger.info(f'Trained Naive Bayes model')
34     evaluation_score = model.score(X_test, y_test)
35     logger.info(f'ML model accuracy score: {evaluation_score*100.2f}%')
36
37     return model, vectorizer
38
39
40 def serialization(model, vectorizer):
41     import joblib
42     os.makedirs('model', exist_ok=True)
43     joblib.dump(vectorizer, 'model/ml_vectorizer.pkl')
44     logger.info(f'Saved vectorizer to model/ml_vectorizer.pkl')
45     joblib.dump(model, 'model/ml_model.pkl')
46     logger.info(f'Saved model to model/ml_model.pkl')
47
48
49 if __name__ == '__main__':
50     logging.basicConfig(format='%(asctime)s - %(message)s', datefmt='%d-%b-%y %H:%M:%S', level=logging.INFO)
51
52     start_time = datetime.now()
53     # Download train and test data from github
54     for mode in ['train', 'test']:
55         download_data(mode)
56
57     model, vectorizer = train_and_evaluate()
58     serialization(model, vectorizer)
```

학습하는 모델은 네이버 감성분류 데이터이다

De-serialization

저장된 모델을 불러와 특정 입력 값에 대한 예측 수행

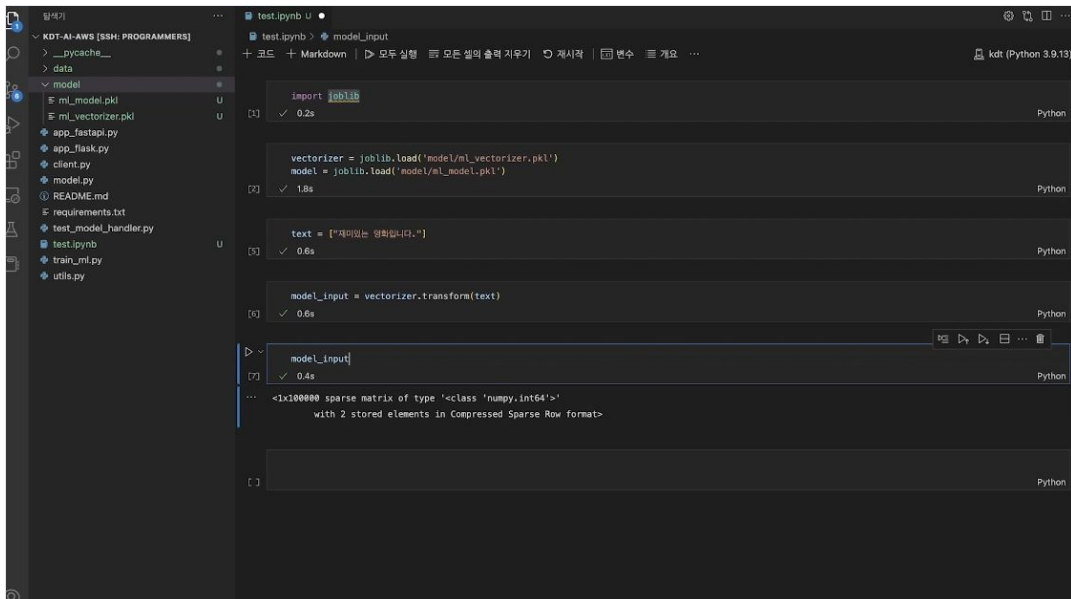
1. 터미널 환경에서 python 또는 ipython, jupyter notebook 실행
2. 아래 예제 코드 테스트하여 de-serialization 확인

```
# De-serialization
import joblib
vectorizer = joblib.load('model/ml_vectorizer.pkl')
model = joblib.load('model/ml_model.pkl')

# Sample text
text = '재미있는 영화입니다.'

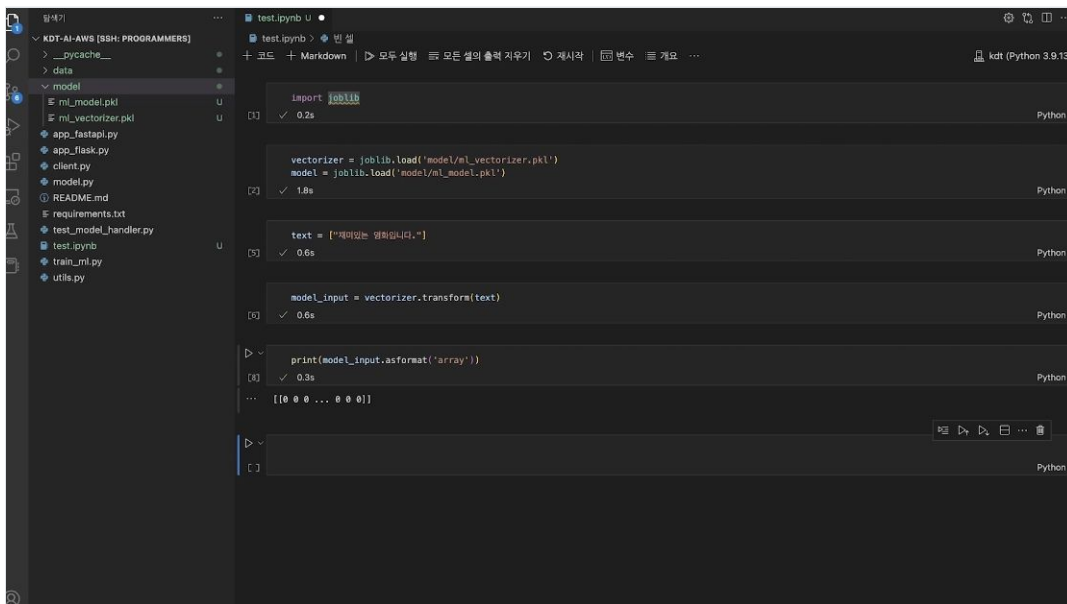
# Test loaded model and vectorizer
model_input = vectorizer.transform(text)
print(model_input.asformat('array'))

model_output = model.predict_proba(model_input)
model_output = model_output.argmax(axis=1)
id2label = {0: 'negative', 1: 'positive'}
print(f'sentiment : {id2label[model_output[0]]}')
```

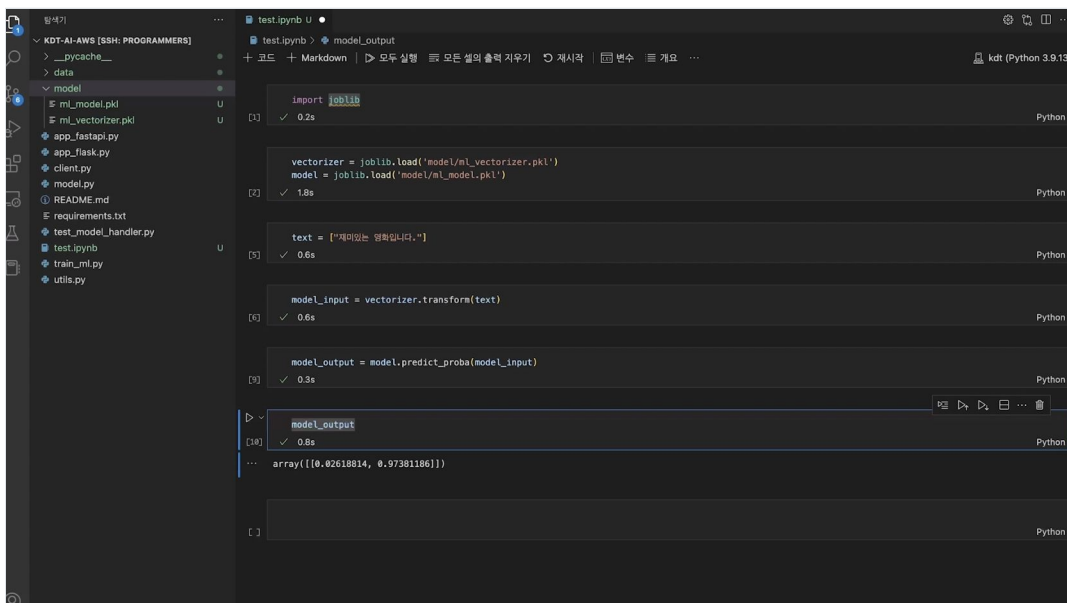


sparse matrix는 희소행렬이라고도 하는데 여기서 사용하고 있는 벡터는 특정 키워드들이 사전으로 존재하고 이 모델같은 경우는 10만개가 될 것이다 이 10만개의 feature중에서 해당 사전의 단어가 있는지 없는지 0과1로 구분된 matrix가 있는거고 그러기 때문에 0이 굉장히 많기 때문에 우리가 0은 연산을 할 때 필요가 없으므로 필요없는 0의 연산을 구분하기 위해서 sparse matrix자료구조를 사용하게 된다.

vectorized된 input을 살펴보자! sparse matrix때문에 array로 변환해서 살펴보아야 한다!



살펴본 결과 한 눈에 살펴보기에는 어렵다(0만 보이므로)



모델의 예측결과를 확인해보면 확률분포 형태로 나오고 결과를 보면 위치에 따라서 label이 정해지는데 2번째 값의 확률이 압도적으로 높으므로 긍정(positive)이 나오게 된다. "재미있는 영화입니다"에 대해서 긍정으로 잘 예측한 모습이다!

Serialization과 De-serialization 방법은 동일해야 함

joblib으로 serialization을 하고 pickle로 불러올 수 없음

```

# De-serialization Error occurs when using different method
import pickle

with open('model/ml_model.pkl', 'rb') as f:
    model = pickle.load(f)

```


The screenshot shows a Jupyter Notebook interface within a terminal window. The left sidebar displays a file explorer with a project structure including files like `__pycache__`, `data`, `model`, `ml_model.pkl`, `ml_vectorizer.pkl`, `app_fastapi.py`, `app_fask.py`, `client.py`, `model.py`, `README.md`, `requirements.txt`, `test_model_handler.py`, `test_ipynb`, `train_ml.py`, and `utils.py`. The main area shows a Jupyter Notebook with several code cells. The first cell (index 12) contains a `print` statement and executes successfully. The second cell (index 14) contains a `print` statement and also executes successfully. The third cell (index 15) contains an `import pickle` statement and executes successfully. The fourth cell (index 16) contains a `with open('model/ml_model.pkl', 'rb') as r:` block followed by `model2 = pickle.load(r)`. This cell fails with an `UnpicklingError: invalid load key, '\x00'.` The error message is displayed in the output area of the cell.

```
[12] ✓ 0.5s Python
print(f'sentiment : {id2label[model_output[0]]}')

[14] ✓ 0.3s Python
sentiment : positive

[15] ✓ 0.3s Python
import pickle

[16] ✖ 0.8s Python
with open('model/ml_model.pkl', 'rb') as r:
    model2 = pickle.load(r)

UnpicklingError                                Traceback (most recent call last)
Cell In [16], line 2
      1 with open('model/ml_model.pkl', 'rb') as r:
----> 2     model2 = pickle.load(r)

UnpicklingError: invalid load key, '\x00'.
```

보면 unpickling error가 발생했다. joblib으로 저장했기 때문에 serialization방식이 조금 달라서 pickle의 load메서드로 가져오려고 할때 에러가 발생하는 것이다.

따라서 실제 현업에서 모델은 내가 개발하고 서빙은 다른사람이 한다고 했을 때 어떤 프레임워크나 라이브러리를 사용해서 serialization을 했는지 API명세를 할 때 모델의 대한 정보도 같이 제공해야 한다!