머신러닝 프로젝트

1분반
학번: 2100961
이름: 박상우

주제: 쇼핑(쿠팡) 리뷰의 긍정 또는 부정 판별 모델

① 데이터 수집 (크롤링)

출처 [파이썬으로 네이버 스마트스토어 리뷰 크롤링하기 (코드 최신버전으로 업데이트) : 네이버 블로그 (naver.com)](#) 데이터 수집 코드작성법을 응용하였다.

1. 리뷰페이지 불러오기

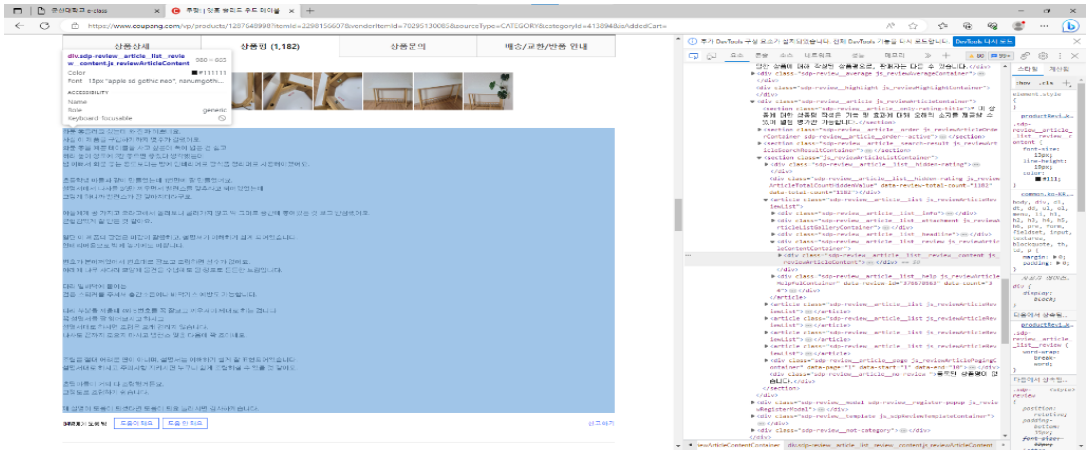
```
In [ ]: import time
        import requests
        from time import sleep
        import re, requests, csv
        from bs4 import BeautifulSoup as bs
        from selenium.webdriver.common.keys import Keys
        import pandas as pd
        from selenium import webdriver
        from selenium.webdriver.common.by import By
        from selenium.webdriver.chrome.service import Service as ChromeService
        from webdriver_manager.chrome import ChromeDriverManager
```

```
In [ ]: #주소입력
        url = "https://www.coupang.com/vp/products/1287648998?itemId=2298156607&vendorItemId=70295130085&sourceType=CATEGORY&categoryId=413894&isAc

        # Chrome 브라우저 설정
        driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))

        # 페이지 열기
        driver.get(url)
```

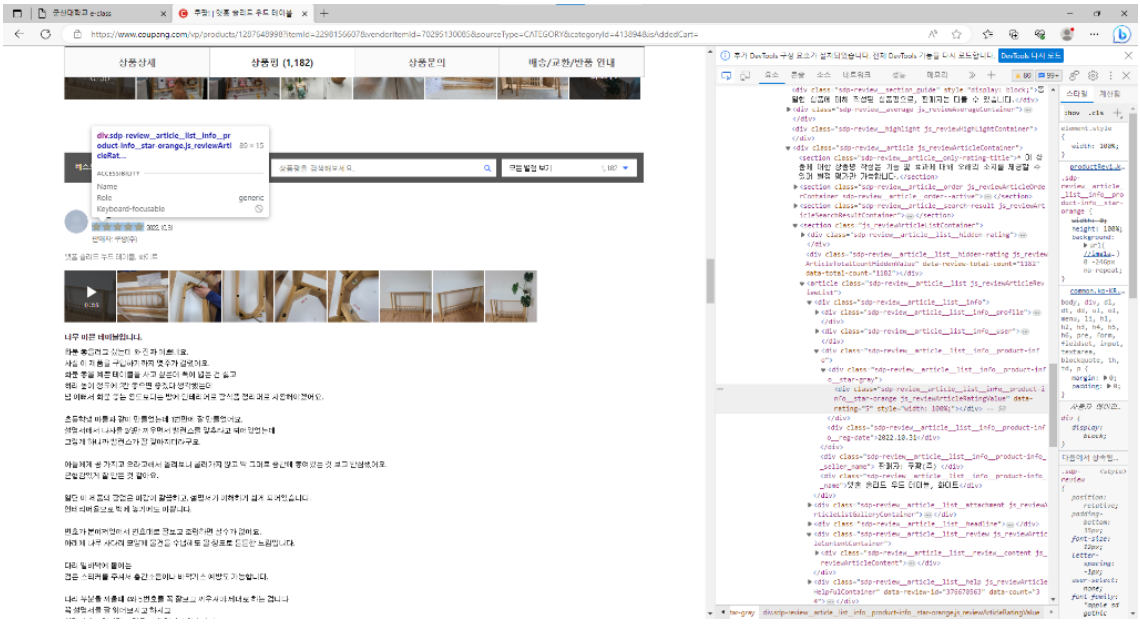
필요한 라이브러리를 선언후 주소 입력을했다. 실행 브라우저는 크롬인데 크롬을 자동으로 제어 시킬려면 크롬드라이버가 필요하다 그래서 나는 크롬드라이버 매니저를 이용해 자동으로 크롬드라이버를 설치해 입력한 주소에 크롬창이 나오게 만들도록했다. (출처에 있는대로 작성했는데 크롬드라이버가 실행이 안되어 조금 수정을 했다.)

2. 본문 및 평점 크롤링

F12 버튼을 누르고 Ctrl + Shift + C를 누른다 그러면 해당 페이지 버튼을 눌러서 HTML 코드를 찾을수 있다. 그 코드로 리뷰, 평점 찾을수 있는 코드를 찾고 출처에 올려진 코드를 응용해서 수정했다.



리뷰 코드탐색



평점 코드탐색

3. 엑셀 데이터로 저장

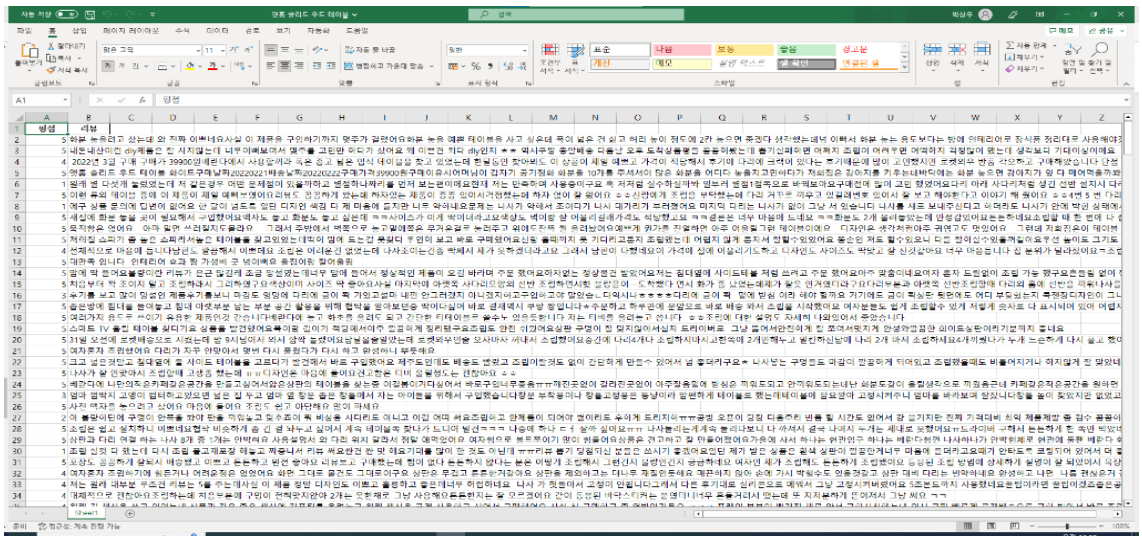
```
In [197]: import pandas as pd

df = pd.DataFrame({"평점": rank_list, "리뷰": review_list})
df.to_excel("C:/Users/dad/Desktop/dataset/앳홈 솔리드 우드 테이블.xlsx", index=False, encoding='openpyxl')
print("저장 완료되었습니다.")
```

```
저장 완료되었습니다.
```

```
C:₩Users₩dad₩anaconda3₩lib₩site-packages₩pandas₩util₩_decorators.py:211: FutureWarning: the 'encoding' keyword is deprecated
and will be removed in a future version. Please take steps to stop the use of 'encoding'
  return func(*args, **kwargs)
```

밑에 빨간 박스에 오류 부분은 무시해도 괜찮다.

4. 데이터수집 작업실행



② 긍정 부정 판별 모델링 구현

1. 데이터셋 다운로드

해당 데이터는 크롤링코드를 이용해 14개 엑셀파일들 만들어 놨다 그걸 1개로 통합시켜서 현재 모델링에 사용하고 있다. 엑셀 구성은 평점,리뷰로 나눴고 평점은 1~5점으로 저장되어있다.

```
In [2]: # 쿠팡 쇼핑 리뷰 데이터셋 다운로드
data = pd.read_excel('dataset.xlsx')

reviews = data['리뷰']
ratings = data['평점']
```

2. 데이터 전처리

텍스트데이터를 숫자 시퀀스로 변환했다. 그리고
레이블인 평점(ratings)을 기준으로 3이상은 긍정(1)이고 3미만은 부정(0)으로 이진화 시켰다.

```
In [18]: # 데이터 전처리
tokenizer = Tokenizer()
tokenizer.fit_on_texts(reviews)
vocab_size = len(tokenizer.word_index) + 1

x_sequences = tokenizer.texts_to_sequences(reviews)
x_padded = pad_sequences(x_sequences, maxlen=100, padding='post')

y = np.array(ratings.apply(lambda x: 1 if x >= 3 else 0))

x_train, x_test, y_train, y_test = train_test_split(x_padded, y, test_size=0.2, random_state=42)


print(reviews[0])
print(x_sequences[0])
```

```
생각보다 길고 원단 톡톡해요근데 건조기 돌리니 좀 줄어들었어요만족해서 하나 더 규매했어요
[90, 1530, 3180, 9768, 9769, 5627, 9, 9770, 105, 10, 9771]
```

3. 신경망 모델 구성

In [33]:
```
# 모델 구성
model = Sequential()
model.add(Embedding(vocab_size, 64,
input_length=100))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 64)           3069248

flatten_1 (Flatten)          (None, 6400)              0

dense_2 (Dense)              (None, 64)                409664

dropout_1 (Dropout)          (None, 64)                0

dense_3 (Dense)              (None, 1)                 65

=================================================================
Total params: 3,478,977
Trainable params: 3,478,977
Non-trainable params: 0
_________________________________________________________________
```

4. 모델학습

In [5]:
```
# 모델 학습
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, batch_size=64, epochs=20, verbose=1, validation_data=(x_test, y_test))
```

```
Epoch 1/20
65/65 [==============================] - 33s 490ms/step - loss: 0.6871 - accuracy: 0.5615 - val_loss: 0.6641 - val_accuracy:
0.6654
Epoch 2/20
65/65 [==============================] - 9s 132ms/step - loss: 0.4886 - accuracy: 0.8261 - val_loss: 0.4131 - val_accuracy:
0.8298
Epoch 3/20
65/65 [==============================] - 7s 112ms/step - loss: 0.1339 - accuracy: 0.9632 - val_loss: 0.3723 - val_accuracy:
0.8453
Epoch 4/20
65/65 [==============================] - 8s 128ms/step - loss: 0.0359 - accuracy: 0.9911 - val_loss: 0.3871 - val_accuracy:
0.8375
Epoch 5/20
65/65 [==============================] - 8s 124ms/step - loss: 0.0132 - accuracy: 0.9990 - val_loss: 0.4173 - val_accuracy:
0.8327
Epoch 6/20
65/65 [==============================] - 8s 120ms/step - loss: 0.0078 - accuracy: 0.9988 - val_loss: 0.4365 - val_accuracy:
0.8250
Epoch 7/20
65/65 [==============================] - 6s 98ms/step - loss: 0.0046 - accuracy: 0.9995 - val_loss: 0.4594 - val_accuracy:
0.8250
Epoch 8/20
65/65 [==============================] - 7s 106ms/step - loss: 0.0039 - accuracy: 0.9993 - val_loss: 0.4731 - val_accuracy:
0.8201
Epoch 9/20
65/65 [==============================] - 7s 102ms/step - loss: 0.0031 - accuracy: 0.9993 - val_loss: 0.5442 - val_accuracy:
0.8162
Epoch 10/20
65/65 [==============================] - 7s 111ms/step - loss: 0.0024 - accuracy: 0.9993 - val_loss: 0.5044 - val_accuracy:
0.8250
Epoch 11/20
65/65 [==============================] - 7s 106ms/step - loss: 0.0022 - accuracy: 0.9995 - val_loss: 0.5663 - val_accuracy:
0.8153
Epoch 12/20
65/65 [==============================] - 7s 101ms/step - loss: 0.0017 - accuracy: 0.9995 - val_loss: 0.5227 - val_accuracy:
0.8182
Epoch 13/20
65/65 [==============================] - 7s 104ms/step - loss: 0.0024 - accuracy: 0.9995 - val_loss: 0.5272 - val_accuracy:
0.8143
Epoch 14/20
65/65 [==============================] - 6s 100ms/step - loss: 0.0019 - accuracy: 0.9995 - val_loss: 0.5344 - val_accuracy:
0.8153
Epoch 15/20
65/65 [==============================] - 7s 103ms/step - loss: 0.0019 - accuracy: 0.9990 - val_loss: 0.5434 - val_accuracy:
0.8269
```

5. 서비스구현

In [38]:
```
# 모델 서비스 구현
review = "배송이 너무 느렸고 상품상태가 별로다"

review = re.sub("[^0-9ㄱ-ㅎ ㅏ-ㅣ 가-힣 ]", "", review).lower()
review_encoding = tokenizer.texts_to_sequences([review])
review_padded = pad_sequences(review_encoding, maxlen=100, padding='post')

value = model.predict(review_padded)

if value > 0.5:
    print("긍정적인 리뷰입니다.")
else:
    print("부정적인 리뷰입니다.")
```

```
1/1 [==============================] - 0s 360ms/step
부정적인 리뷰입니다.
```

review = "배송이 너무 느렸고 상품상태가 별로다" 예측하였을때 부정인것은 정확히 나타내는 것을 알수 있었다.

In [39]:
```
# 모델 서비스 구현
review = "배송이 빨라서 좋아요"

review = re.sub("[^0-9ㄱ-ㅎ ㅏ-ㅣ 가-힣 ]", "", review).lower()
review_encoding = tokenizer.texts_to_sequences([review])
review_padded = pad_sequences(review_encoding, maxlen=100, padding='post')

value = model.predict(review_padded)

if value > 0.5:
    print("긍정적인 리뷰입니다.")
else:
    print("부정적인 리뷰입니다.")
```

```
1/1 [==============================] - 0s 31ms/step
긍정적인 리뷰입니다.
```

이번에는 "배송이 빨라서 좋아요" 긍정문을 넣었는데 긍정인것을 예측할수 있었다.

이처럼 충분한 데이터가 제공된다면 감정을 분석하는 모델을 통해 챗봇 AI의 성능이 향상될 것으로 예상되며, 이는 상담, 마케팅, 제품 개발 등 다양한 응용 분야에서 챗봇 AI의 역할과 가치가 크게 증가할 것으로 기대됩니다. 이로써 사람들에게 효율적인 서비스를 제공받을 수 있게 될 것입니다.