# 왓챠 데이터 분석

# (Analyzing Watcha Data)

202144095 이서영

# - 목 차 -

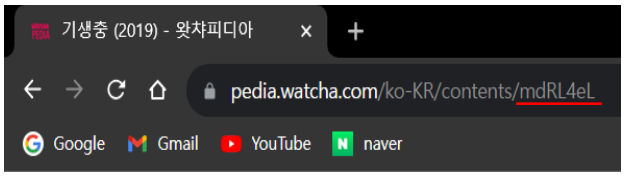
# 1 개 요

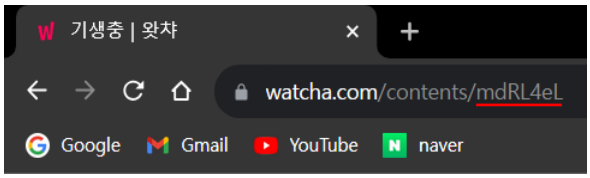
왓챠 데이터를 분석을 통해 왓챠라는 플랫폼에 어떤 콘텐츠가 있는지 분석하고, 평균 평점 정보를 통해 사람들이 어떤 콘텐츠를 선호하는지 알아본다.

# 2 데이터 수집

## 2.1 수집 방법

왓챠 웹사이트와 왓챠피디아 웹사이트는 동일한 콘텐츠를 나타내는 id 값이 동일하다. 이점을 활용하여 왓챠와 왓챠피디아에 등록되어있는 콘텐츠의 정보를 셀레니움을 사용하여 수집한다.



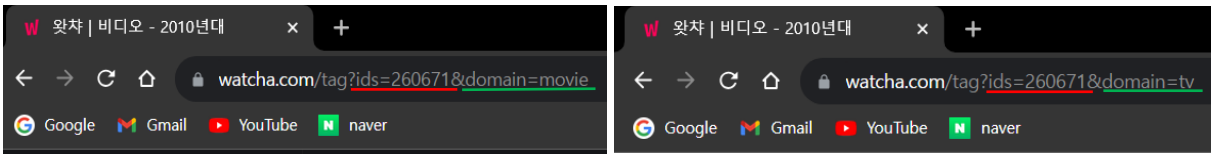

왓챠 웹사이트에서는 콘텐츠 id, 종류, TV 프로그램의 에피소드 수와 상영시간 정보를 가져오고, 왓챠피디아 웹사이트에서는 모든 콘텐츠의 상세 정보를 가져온다.

## 2.2 개발 환경

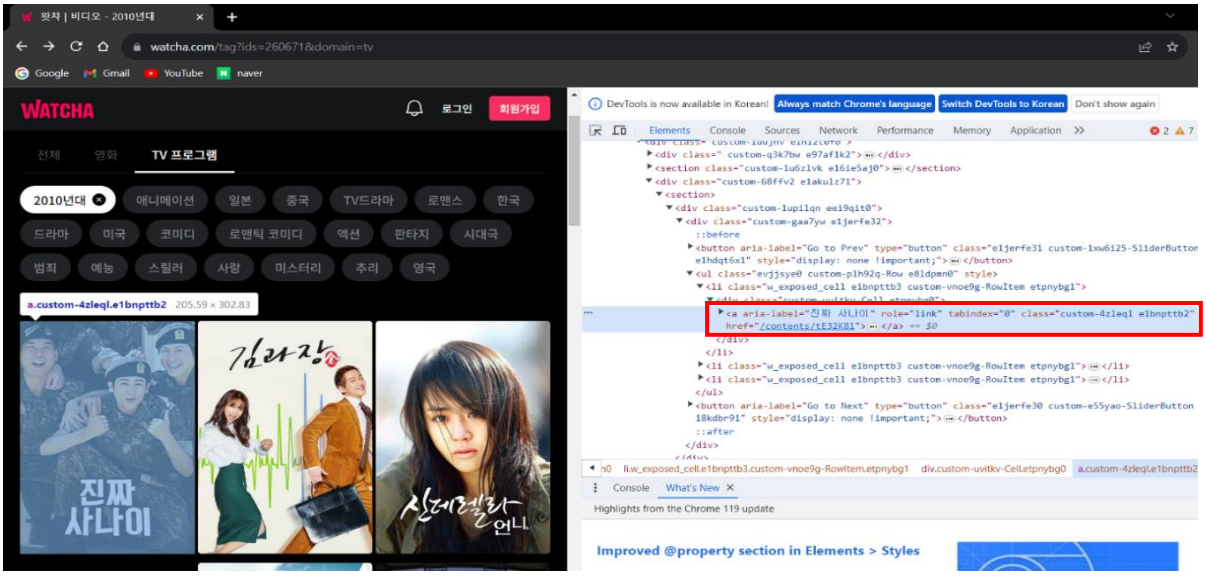
| | |
|---|---|
| **운영체제** | Windows 11 Home |
| **개발언어** | Python(3.11.5) |
| **개발도구** | Spyder(5.4.3), Selenium(4.14.0) |
| **개발환경** | ANACONDA(23.7.4) |

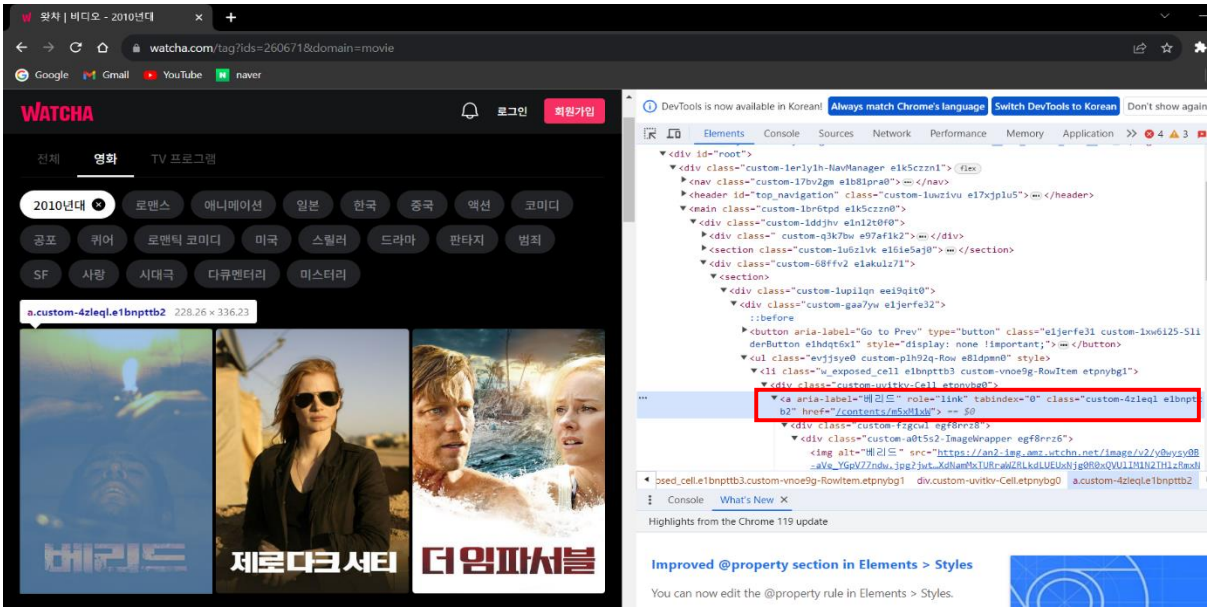
### 2.3 왓챠 크롤링(1)(seleni_watcha_id_type.py, watcha_id_type.csv)

왓챠에서는 콘텐츠 종류(domain)와 해당하는 각 년도별 태그(ids)가 존재하며, 이를 활용하여 모든 콘텐츠의 id값과 종류를 크롤링하고 csv 파일에 저장한다.




스크롤을 맨 밑으로 내린 후 각 년대에 해당하는 종류의 목록에서 각 콘텐츠의 a 태그의 href 속성을 가져오고 split하여 콘텐츠의 id값을 가져온다.



```python
# selenium의 webdriver를 사용하기 위한 import
# 웹 브라우저를 제어하고 웹 페이지를 열고 조작하는데 사용
from selenium import webdriver

# selenium으로 키를 조작하기 위한 import
from selenium.webdriver.common.keys import Keys

# 페이지 로딩을 기다리는데에 사용할 time 모듈 import
import time

# 웹 페이지에서 특정 요소를 찾기 위한 import
# 웹 요소를 검색할 때 사용되는 여러 종류의 기준을 제공
from selenium.webdriver.common.by import By

# 데이터프레임 사용을 위한 import
import pandas as pd

# 크롤링 소요시간 계산을 위한 import
from datetime import datetime



# Chrome WebDriver 옵션 설정하기
options = webdriver.ChromeOptions() # 옵션 설정 객체 생성

options.add_argument('--headless') # 창이 나타나지 않고 백그라운드에서 실행하도록 설정
options.add_argument('disable-gpu') # 불필요한 그래픽카드 기능 제거
options.add_argument('--no-sandbox') # Chrome 보안 기능 비활성화 -> Chrome 시스템 리소스 감소로 가벼운 웹 스크래핑 작업 수행
options.add_argument('--disable-dev-shm-usage') # 공유 메모리 공간 사용 비활성화 -> 리소스 제한이 있는 환경이나 큰 웹 페이지를 다루는 경우 사용
options.add_argument('window-size=1920x1080') # pc용 사이즈
options.add_argument('--start-maximized') # 브라우저가 최대화된 상태로 실행
# User-Agent 추가 -> 사이트에서의 차단을 회피, 특정 기능이나 콘텐츠에 대한 정상적인 접근을 가능하도록
options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/537.36")

driver = webdriver.Chrome(options=options) # 설정 적용
```

```
# 콘텐츠 id와 종류(영화 or tv 프로그램)를 담을 리스트
id_type_data = []
# 스크롤을 끝까지 내리는 함수
def scroll_end():

    # 현재 스크롤 높이
    scroll_location = driver.execute_script("return document.body.scrollHeight")

    while True:
        # 현재 스크롤의 가장 아래로 내림
        driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")

        # 전체 스크롤이 늘어날 때까지 대기
        time.sleep(1)

        # 늘어난 스크롤 높이
        scroll_height = driver.execute_script("return document.body.scrollHeight")

        # 늘어난 스크롤 위치와 이동 전 위치 같으면(더 이상 스크롤이 늘어나지 않으면) 종료
        if scroll_location == scroll_height:
            break

        # 같지 않으면 스크롤 위치 값을 수정하여 같아질 때까지 반복
        else:
            #스크롤 위치값을 수정
            scroll_location = driver.execute_script("return document.body.scrollHeight")


# 왓챠에서 콘텐츠의 아이디와 종류를 가져오는 함수
def get_id_type(year, content_type):

    # 콘텐츠 목록에서 각 콘텐츠의 정보가 담겨있는 url을 가져온다.
    url = 'https://watcha.com/tag?ids='+ year + '&domain=' + content_type # 콘텐츠 목록 url

    driver.implicitly_wait(5) # 암묵적 대기, NoSuchElementException을 던지기 전에 기다리도록
함(초단위)
    driver.get(url)

    # 스크롤 끝까지 내리기
    scroll_end()
```

```
    # 5초 동안 대기
    time.sleep(5)

    # 콘텐츠 목록에서 콘텐츠를 클릭했을 때 연결된 a 태그 href 속성의 값을 가져옴(id)
    for i in range(1,12000,1): # 목록의 행

        try:
            for j in range(1,7,1): # 목록의 열 6개로 고정

                id_src =
driver.find_element(By.XPATH,'//*[@id="root"]/div[1]/main/div/div[2]/section/div['+str(i)+'
]/div/ul/li['+str(j)+']/div/a')
                id_href = id_src.get_attribute('href')

                id_type_data.append([id_href, content_type])

        except Exception: # 없는 행에 도달하면 반복문에서 탈출한다.
            break


if __name__ == '__main__':

    # 시작 시각
    start_time = datetime.now()


    # 1920~2023 각 년대별 url의 ids 값을 저장한 리스트
    # https://watcha.com/tag?ids=508871&domain=movie(1920년대)
    # https://watcha.com/tag?ids=508870&domain=movie(1930년대)
    year_list = ['508871','508870','508869','508868','508867','508866','508865','260666'
              ,'260672','260671'
              ,'508924']

    count = 0  # 크롤링 체크용 변수, 11(len(year_list))이면 완료


    # 1. 콘텐츠의 id와 종류 가져오기
    for y in year_list:
            get_id_type(y, 'movie')
            get_id_type(y, 'tv')
```

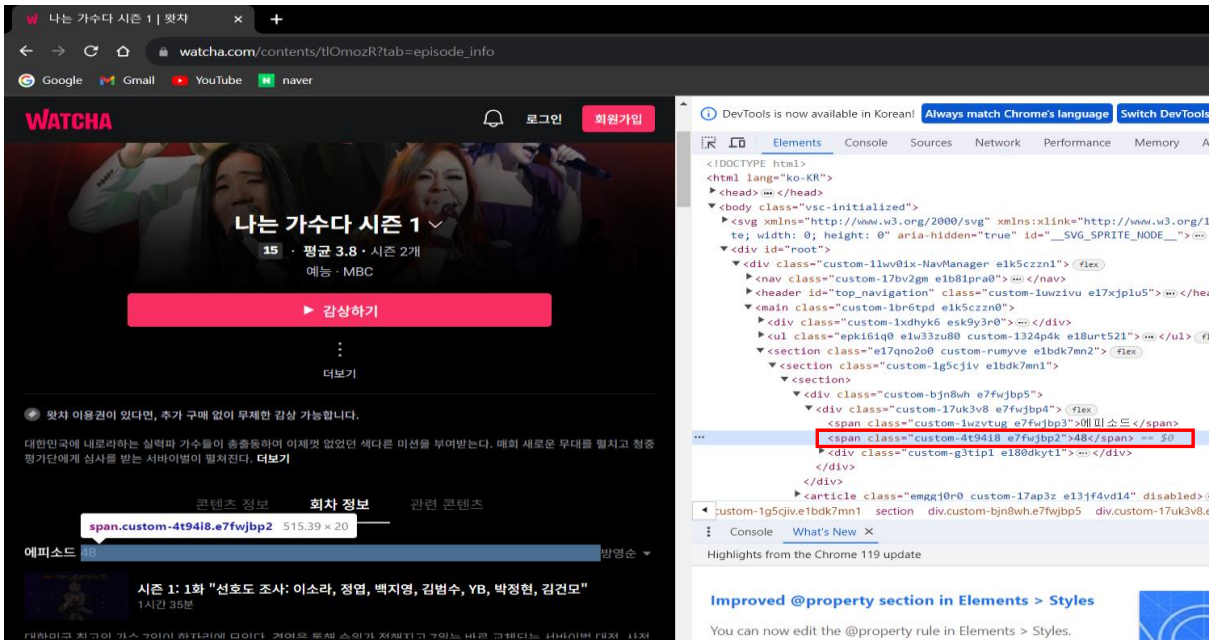
```
            count += 1
            print(str(count))


    # 2. id 값으로 콘텐츠 정보를 수집할 것이기 때문에 미리 전처리 후 저장

    # id값만 추출하여 리스트에 다시 저장
    for i in range(len(id_type_data)):
        split_id = id_type_data[i][0].split('/')
        id_type_data[i][0] = split_id[len(split_id)-1]

    # 데이터 프레임 생성
    id_type_df = pd.DataFrame(id_type_data, columns=['id', '종류'])

    # 중복 데이터 존재시 미리 제거
    if id_type_df.duplicated().sum() > 0:
        id_type_df.drop_duplicates(inplace=True)

    # csv 파일로 저장
    id_type_df.to_csv('C:/Crawling_Watcha/csv/watcha_id_type.csv',encoding='cp949',
mode='w', index=False)


    # 종료 시각
    end_time = datetime.now()

    # 소요 시간 확인
    print(end_time-start_time)
```

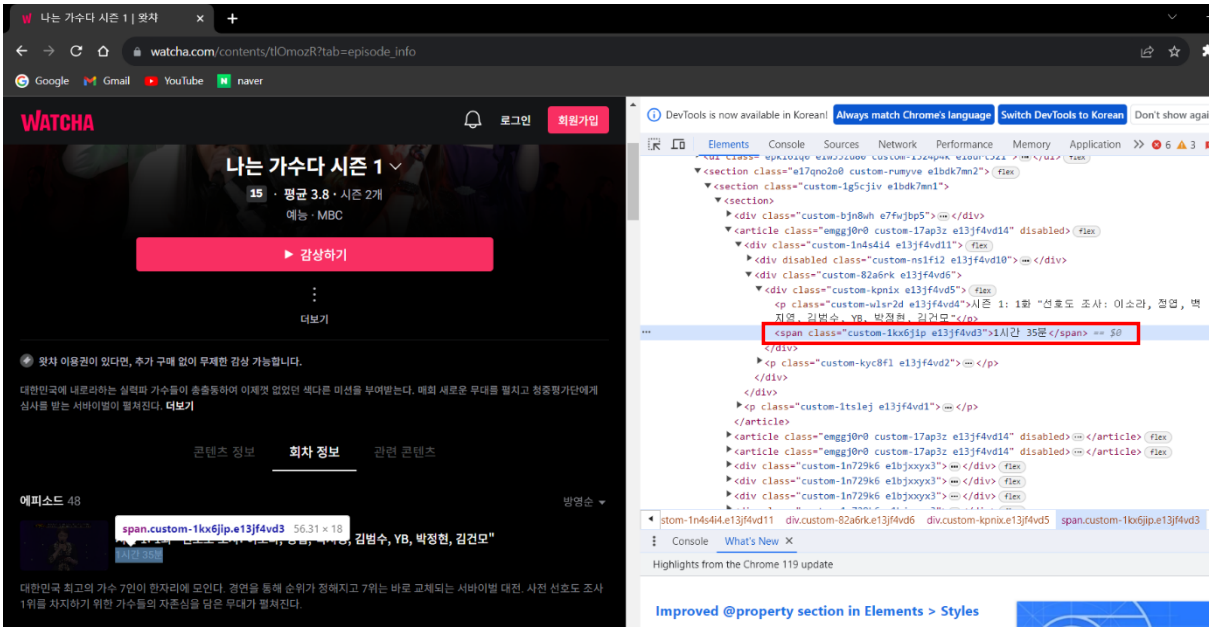
### 2.4 왓챠 크롤링(2)(seleni_watcha_epnum_rtime.py, watcha_tv_info.csv)

TV 프로그램에 해당하는 상영시간과 에피소드 수는 왓챠에만 존재한다. 따라서 앞서 수집한 데이터에서 종류가 TV 프로그램인 id만 추출하여 TV프로그램의 상영시간과 에피소드 수를 크롤링하고 csv 파일에 저장한다. 이때, 상영시간은 첫 화를 기준으로 한다.

셀레니움을 사용하여 크롤링하기 때문에 상당 시간이 소요되어 안정성을 높이기 위해 100개씩 데이터를 크롤링하고 동일한 csv 파일에 저장하여 데이터를 쌓는다. 또한 100개를 크롤링한 후 WebDirver 인스턴스를 종료 후 다시 생성하며, 전체를 한 번 크롤링한 후 요소를 찾지 못한 id를 추출하여 최대 10번 다시 크롤링한다. 이때, 다시 크롤링한 데이터도 동일한 csv에 저장하고 후에 전처리한다.

그리고 많은 수의 데이터를 크롤링하므로 봇으로 간주되어 차단당하지 않기 위해 랜덤 시간동안 대기하는 코드를 추가한다.






```
# selenium의 webdriver를 사용하기 위한 import
# 웹 브라우저를 제어하고 웹 페이지를 열고 조작하는데 사용
from selenium import webdriver

# 페이지 로딩을 기다리는데에 사용할 time 모듈 import
import time

# 웹 페이지에서 특정 요소를 찾기 위한 import
# 웹 요소를 검색할 때 사용되는 여러 종류의 기준을 제공
from selenium.webdriver.common.by import By

# 데이터프레임 사용을 위한 import
import pandas as pd

# 무작위 수를 추출하기 위한 import
import random

# 크롤링 소요 시간을 위한 import
from datetime import datetime



# Chrome WebDriver 옵션 설정하기
options = webdriver.ChromeOptions() # 옵션 설정 객체 생성
```

```
options.add_argument('--headless') # 창이 나타나지 않고 백그라운드에서 실행하도록 설정
options.add_argument('disable-gpu') # 불필요한 그래픽카드 기능 제거
options.add_argument('--no-sandbox') # Chrome 보안 기능 비활성화 -> Chrome 시스템 리소스 감소로 가벼운 웹 스크래핑 작업 수행
options.add_argument('--disable-dev-shm-usage') # 공유 메모리 공간 사용 비활성화 -> 리소스 제한이 있는 환경이나 큰 웹 페이지를 다루는 경우 사용
options.add_argument('window-size=1920x1080') # pc 용 사이즈
options.add_argument('--start-maximized') # 브라우저가 최대화된 상태로 실행
options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/537.36")



# 누락된 id 를 담을 리스트
missing_id = []

# 누락 데이터 수집 시작 플래그
miss_flag = False



# 왓챠에서 tv 프로그램의 에피소드 수(1 화 기준)와 상영시간을 가져오는 함수
def get_tv_rtime_epnum(id_data):

    # 데이터를 10 개씩 csv 에 저장
    batch_size = 100
    start_i = 0 # 시작 인덱스
    end_i = batch_size # 끝 인덱스

    tv_id_info = [] # TV 프로그램의 에피소드 수와 상영시간을 담을 리스트
    no_such_flag = False # NoSuchElementException 발생 유무 플래그, 발생하지 않은 상태


    for i in range(int(len(id_data)/batch_size) + 1):
        print(i)

        # 100 개의 데이터 크롤링 후 WebDriver 인스턴스를 종료하고 새로운 인스턴스를 생성 -> 메모리 사용량을 줄이고 잠재적인 문제를 방지하기 위해
        driver = webdriver.Chrome(options=options)

        for i in range(start_i, end_i):
```

```
            print(i)

            try:
                # tv 프로그램
                url = 'https://watcha.com/contents/' + id_data[i] + '?tab=episode_info'

                driver.implicitly_wait(5) # 암묵적 대기, NoSuchElementException 을 던지기 전에 기다리도록 함(초단위)
                driver.get(url)

                try:
                    epi_num = driver.find_element(By.XPATH, '//*[@id="root"]/div[1]/main/section/section/section/div[1]/div/span[2]').text

                except Exception: # NoSuchElementException 발생
                    no_such_flag = True
                    epi_num = "?"

                try:
                    running_time = driver.find_element(By.XPATH, '//*[@id="root"]/div[1]/main/section/section/section/article[1]/div/div[2]/div/span').text

                except Exception:
                    no_such_flag = True
                    running_time = "?"


                if no_such_flag == True and miss_flag == True: # 누락 데이터 처리시 누락 아이디 리스트에 추가
                    missing_id.append(id_data[i])

                    no_such_flag = False # 플래그 초기화


                tv_id_info.append([id_data[i], epi_num, running_time])

            except IndexError: # 인덱스 범위 초과 예외처리
                break
```

```
        if start_i == 0 and miss_flag == False: # 맨 처음 csv 생성이면서, 누락 데이터 처리가
아닌 경우
            # csv 파일로 저장
            tv_info_df = pd.DataFrame(tv_id_info, columns=['id', '에피소드 수','상영시간'])
            tv_info_df.to_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv',encoding='cp949',
mode='w', index=False)

            tv_id_info.clear() # 다음 10 개를 저장하기 위해 리셋

            start_i = start_i + batch_size
            end_i = end_i + batch_size

            driver.quit()

        else:
            # csv 파일에 추가
            tv_info_df = pd.DataFrame(tv_id_info)
            tv_info_df.to_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv',encoding='cp949',
mode='a', header=False, index=False)

            tv_id_info.clear()

            start_i = start_i + batch_size
            end_i = end_i + batch_size


            driver.quit() # Selenium WebDriver 를 종료


        # 봇으로 간주되지 않도록 랜덤 시간동안 대기
        num = random.randint(1, 5)
        time.sleep(num)



if __name__ == '__main__':

    # 시작 시각
    start_time = datetime.now()
```

```
    # 1. csv 파일에서 tv 프로그램 id만 가져오기
    id_type_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_id_type.csv', encoding='cp949')

    mask = id_type_df['종류'] == 'tv'
    id_df = id_type_df.loc[mask,:]
    print(id_df['id'].count()) #3222

    id_data = id_df['id'].values.tolist()


    # 2. tv 프로그램 정보 가져오기
    get_tv_rtime_epnum(id_data)

    # 2-1. 전체 크롤링(첫번째 크롤링)한 내용 백업
    first_tv_info = pd.read_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv',
encoding='cp949')
    first_tv_info.to_csv('C:/Crawling_Watcha/csv/watcha_tv_info_first.csv',encoding='cp949'
, mode='w', index=False)


    # 3. 누락 데이터 id 가져오기
    df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv', encoding='cp949')

    # 칼럼 둘 중 하나라도 '?'로 표시된 값을 가진 행 추출
    miss_df = df[(df['에피소드 수'] == '?') | (df['상영시간'] == '?')]
    print(miss_df.count())
    missing_id = miss_df['id'].values.tolist()


    # 4. 누락된 tv 프로그램 정보 다시 가져오기(크롤링 재시도)
    # 요소가 존재하지 않거나, 네트워크 또는 서버의 응답 처리 속도가 늦어 누락된 경우를 처리
    count = 1       # 누락 데이터 수집 시도 횟수
    miss_flag = True  # 누락 데이터 수집 시작

    for i in range(10):

        if len(missing_id) == 0: # 누락된 데이터가 없는 경우
            break

        # csv 파일로 저장(개수 확인용)
        missing_id_df = pd.DataFrame(missing_id, columns=['id'])
```

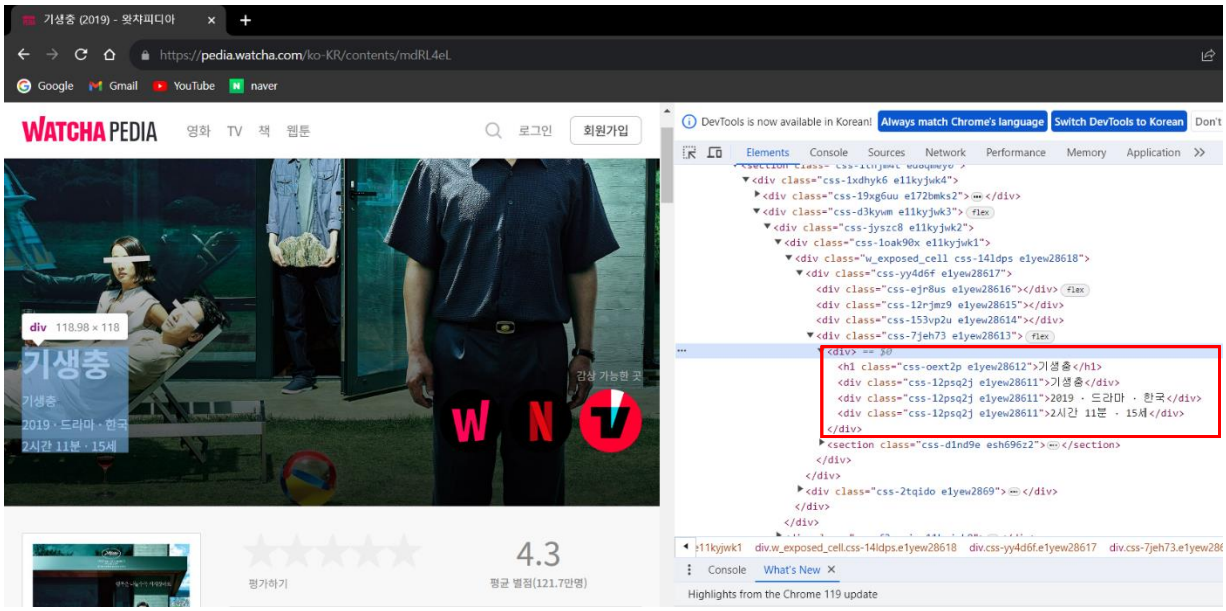
```
        missing_id_df.to_csv('C:/Crawling_Watcha/csv/watcha_tv_missing_' + str(count) +
'.csv',encoding='cp949', mode='w', index=False)

        r_missing_id = missing_id[:]  # 기존 누락 데이터를 저장하는 리스트
        missing_id.clear()  # 줄어든 누락 데이터를 새로 저장하기 위해 리스트 비우기

        count = count + 1

        get_tv_rtime_epnum(r_missing_id)


    # 종료 시각
    end_time = datetime.now()

    # 소요 시간 확인
    print(end_time-start_time)
```

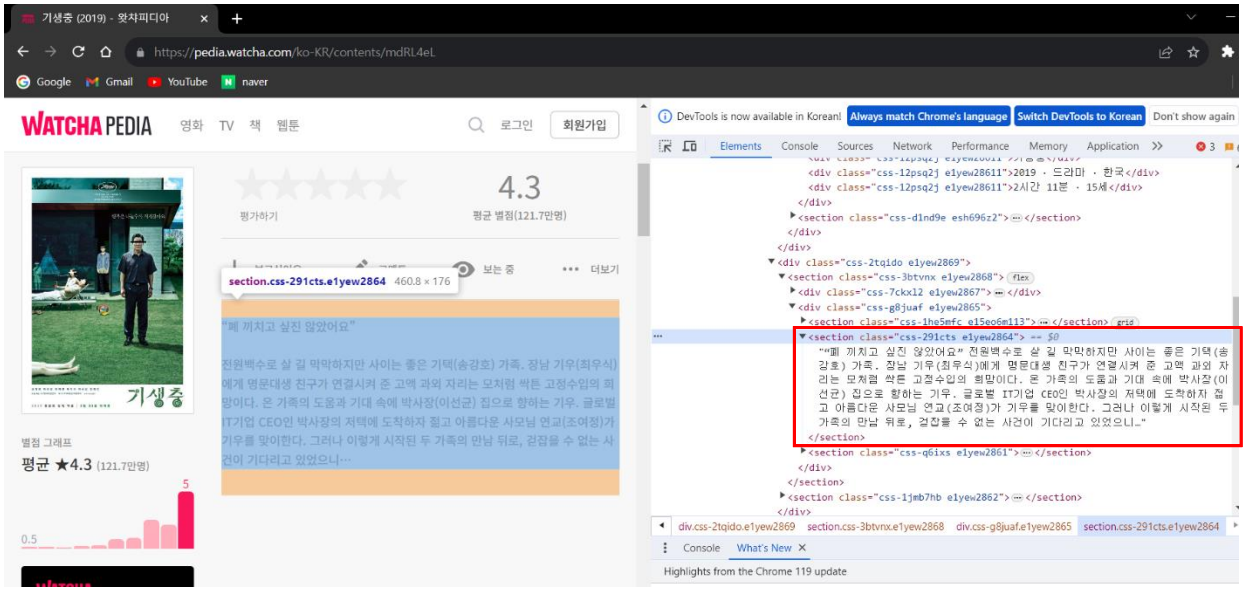
### 2.5 왓챠피디아 크롤링(seleni_wpedia_info.py, watcha_content_info.csv)

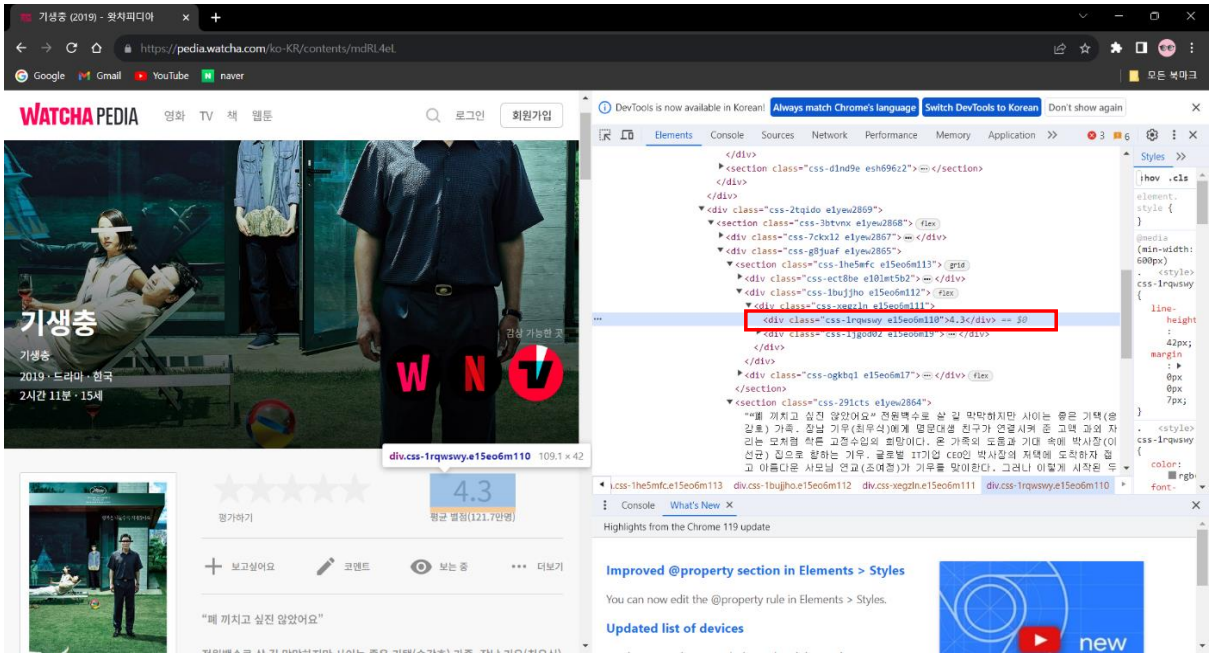
왓챠피디아에서는 왓챠에서 크롤링한 모든 콘텐츠의 id를 가지고 콘텐츠의 상세정보와 평균 평점, 줄거리, 제작진 정보를 크롤링하고 csv 파일에 저장한다.

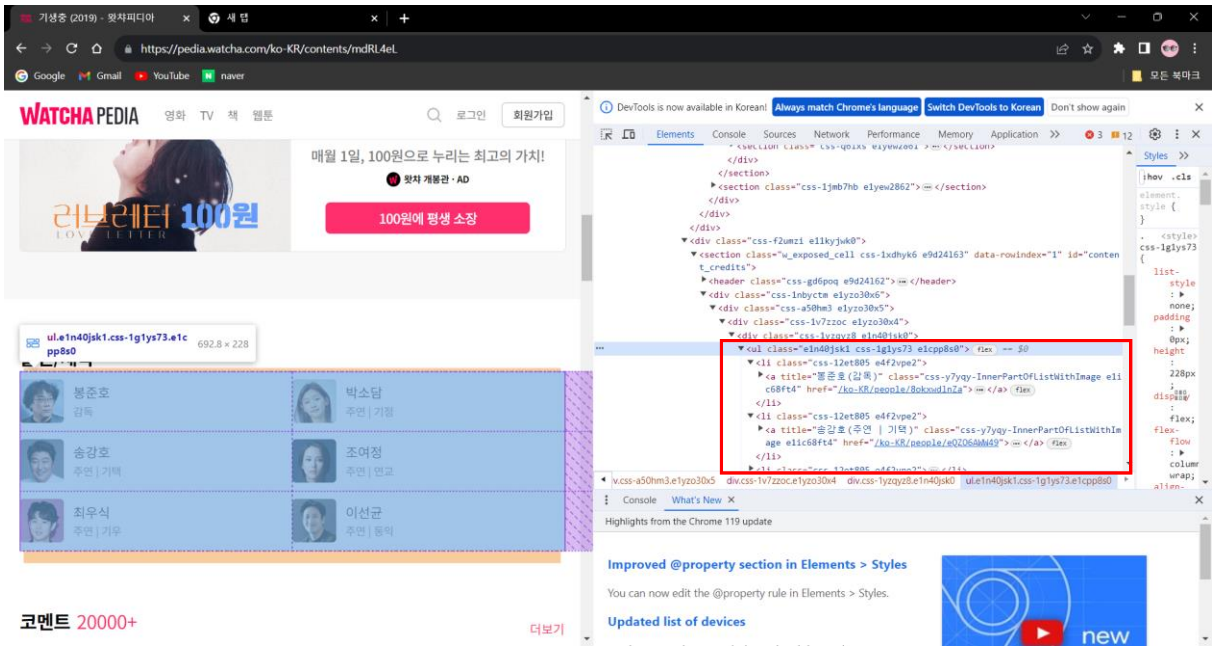
전체적인 구조는 앞서 왓챠에서 TV 프로그램의 에피소드 수와 상영시간과 같다. 100개씩 크롤링하며, 전체를 한 번 크롤링한 후 누락된 요소가 있는 id만 추출하고 최대 10번 다시 크롤링한다.

콘텐츠의 상세정보에는 제목, 개봉년도, 장르, 국가, 상영시간(영화), 연령 등급, 방송사(TV 프로그램) 정보가 포함되어 있다. 이후 전처리에서 split한다.




기생충 (2019) - 왓챠피디아
https://pedia.watcha.com/ko-KR/contents/mdRL4eL
WATCHA PEDIA
영화 TV 책 웹툰
로그인
회원가입
기생충
2019 · 드라마 · 한국
2시간 11분 · 15세
4.3
평균 별점(121.7만명)
"페 끼치고 싶진 않았어요"
DevTools is now available in Korean!
Always match Chrome's language
Switch DevTools to Korean
Don't show again
Improved @property section in Elements > Styles
You can now edit the @property rule in Elements > Styles.
Updated list of devices
new


기생충 (2019) - 왓챠피디아
https://pedia.watcha.com/ko-KR/contents/mdRL4eL
WATCHA PEDIA
영화 TV 책 웹툰
로그인
회원가입
매월 1일, 100원으로 누리는 최고의 가치!
100원에 평생 소장
봉준호
박소담
송강호
조여정
최우식
이선균
코멘트 20000+
더보기
DevTools is now available in Korean!
Always match Chrome's language
Switch DevTools to Korean
Don't show again
Improved @property section in Elements > Styles
You can now edit the @property rule in Elements > Styles.
Updated list of devices
new

```
# selenium의 webdriver를 사용하기 위한 import
# 웹 브라우저를 제어하고 웹 페이지를 열고 조작하는데 사용
from selenium import webdriver

# 페이지 로딩을 기다리는데에 사용할 time 모듈 import
import time

# 웹 페이지에서 특정 요소를 찾기 위한 import
# 웹 요소를 검색할 때 사용되는 여러 종류의 기준을 제공
from selenium.webdriver.common.by import By

# 데이터프레임 사용을 위한 import
import pandas as pd

# 무작위 수를 추출하기 위한 import
import random

# 크롤링 소요 시간을 위한 import
from datetime import datetime



# Chrome WebDriver 옵션 설정하기
options = webdriver.ChromeOptions() # 옵션 설정 객체 생성

options.add_argument('--headless') # 창이 나타나지 않고 백그라운드에서 실행하도록 설정
options.add_argument('disable-gpu') # 불필요한 그래픽카드 기능 제거
options.add_argument('--no-sandbox') # Chrome 보안 기능 비활성화 -> Chrome 시스템 리소스
감소로 가벼운 웹 스크래핑 작업 수행
options.add_argument('--disable-dev-shm-usage') # 공유 메모리 공간 사용 비활성화 -> 리소스
제한이 있는 환경이나 큰 웹 페이지를 다루는 경우 사용
options.add_argument('window-size=1920x1080') # pc용 사이즈
options.add_argument('--start-maximized') # 브라우저가 최대화된 상태로 실행
options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/537.36")



# 누락된 id를 담을 리스트
missing_id = []

# 누락 데이터 수집 시작 플래그
```

```
miss_flag = False


# 데이터를 10개씩 csv에 저장
batch_size = 100


# 왓챠피디아에서 콘텐츠 정보를 가져오는 함수
def get_content_info(id_data):

    start_i = 0 # 시작 인덱스
    end_i = batch_size # 끝 인덱스

    content_info_data = [] # TV 프로그램의 에피소드 수와 상영시간을 담을 리스트
    no_such_flag = False # NoSuchElementException 발생 유무 플래그, 발생하지 않은 상태


    for i in range(int(len(id_data)/batch_size) + 1):
        print(i)

        # 100개의 데이터 크롤링 후 WebDriver 인스턴스를 종료하고 새로운 인스턴스를 생성 -> 메모리 사용량을 줄이고 잠재적인 문제를 방지하기 위해
        driver = webdriver.Chrome(options=options)

        for i in range(start_i, end_i):

            print(i)

            try:
                # 콘텐츠 정보 url
                url = 'https://pedia.watcha.com/ko-KR/contents/'+ id_data[i]

                driver.implicitly_wait(5) # 암묵적 대기, NoSuchElementException을 던지기 전에 기다리도록 함(초단위)
                driver.get(url)

                try:
                    # 제목, 개봉년도, 제작사 or 상영시간 ,장르, 국가, 연령등급
                    contentInfo = driver.find_element('xpath','/html/body/div[1]/div/div[1]/section/div/div[2]/div/div[1]/div/div[1]/div[4]/div').text
```

```
                except Exception: # NoSuchElementException 발생
                    no_such_flag = True
                    contentInfo = "?"

                try:
                    # 평균 평점
                    avg_grade =
driver.find_element('xpath','//*[@id="root"]/div/div[1]/section/div/div[2]/div/div[1]/div/d
iv[2]/section[1]/div[2]/section[1]/div[2]/div/div[1]').text

                except Exception:
                    no_such_flag = True
                    avg_grade = "?"

                try:
                    # 줄거리
                    story =
driver.find_element('xpath','//*[@id="root"]/div/div[1]/section/div/div[2]/div/div[1]/div/d
iv[2]/section[1]/div[2]/section[3]').text

                except Exception:
                    no_such_flag = True
                    story = "?"

                try:
                    # 감독,출연자
                    ul_element = driver.find_element('xpath',
'//*[@id="content_credits"]/div/div[1]/div/div/ul')
                    li_elements = ul_element.find_elements(By.TAG_NAME, 'li')

                    maker_cast = ""

                    for li in li_elements:
                        a_element = li.find_element(By.TAG_NAME, 'a')
                        maker_cast += a_element.get_attribute('title')

                except Exception:
                    no_such_flag = True
                    maker_cast = "?"
```

```
                if no_such_flag == True and miss_flag == True: # 누락 데이터 처리시 누락 아이디 리스트에 추가
                    missing_id.append(id_data[i])

                    no_such_flag = False # 플래그 초기화


                content_info_data.append([id_data[i], contentInfo, avg_grade, story, maker_cast])

            except IndexError: # 인덱스 범위 초과 예외처리
                break


        if start_i == 0 and miss_flag == False: # 맨 처음 csv 생성이면서, 누락 데이터 처리가 아닌 경우

            # csv 파일로 저장
            content_info_df = pd.DataFrame(content_info_data, columns=['id', '컨텐츠 정보','평균 평점','줄거리','제작진'])
            content_info_df.to_csv('C:/Crawling_Watcha/csv/watcha_content_info.csv',encoding='utf-8-sig', mode='w', index=False)

            content_info_data.clear()

            start_i = start_i + batch_size
            end_i = end_i + batch_size

            driver.quit() # Selenium WebDriver 를 종료

        else:
            # csv 파일에 추가
            content_info_df = pd.DataFrame(content_info_data)
            content_info_df.to_csv('C:/Crawling_Watcha/csv/watcha_content_info.csv',encoding='utf-8-sig', mode='a', header=False, index=False)

            content_info_data.clear()

            start_i = start_i + batch_size
            end_i = end_i + batch_size
```

```
            driver.quit()


        # 봇으로 간주되지 않도록 랜덤 시간동안 대기
        num = random.randint(1, 5)
        time.sleep(num)



if __name__ == '__main__':

    # 시작 시각
    start_time = datetime.now()


    # 1. csv 파일에서 모든 콘텐츠의 id 가져오기
    id_type_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_id_type.csv', encoding='cp949')
    print(id_type_df['id'].count()) #9664

    id_data = id_type_df['id'].values.tolist()


    # 2. 모든 콘텐츠 정보 가져오기
    get_content_info(id_data)


    # 2-1. 전체 크롤링(첫번째 크롤링)한 내용 백업
    first_tv_info = pd.read_csv('C:/Crawling_Watcha/csv/watcha_content_info.csv',
encoding='utf-8-sig')
    first_tv_info.to_csv('C:/Crawling_Watcha/csv/watcha_content_info_first.csv',encoding='u
tf-8-sig', mode='w', index=False)



    # 3. 누락 데이터 id 가져오기
    df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_content_info.csv', encoding='utf-8-
sig')

    # 필드값이 없는 경우 "?"로 대체
    df.fillna('?', inplace=True)
```

```
    # 칼럼들 중 하나라도 '?'로 표시된 값을 가진 행 추출
    miss_df = df[(df['컨텐츠 정보'] == '?') | (df['평균 평점'] == '?') | (df['줄거리'] == '?') | (df['제작진'] == '?')]
    print(miss_df.count())
    missing_id = miss_df['id'].values.tolist()


    # 4. 누락된 tv 프로그램 정보 다시 가져오기(크롤링 재시도)
    # 요소가 존재하지 않거나, 네트워크 또는 서버의 응답 처리 속도가 늦어 누락된 경우를 처리
    count = 1       # 누락 데이터 수집 시도 횟수
    miss_flag = True  # 누락 데이터 수집 시작

    for i in range(10):

        if len(missing_id) == 0: # 누락치가 없으면
            break

        # csv 파일로 저장(개수 확인용)
        missing_id_df = pd.DataFrame(missing_id, columns=['id'])
        missing_id_df.to_csv('C:/Crawling_Watcha/csv/watcha_content_missing_' + str(count) + '.csv',encoding='utf-8-sig', mode='w', index=False)

        r_missing_id = missing_id[:]  # 기존 누락 데이터를 저장하는 리스트
        missing_id.clear()  # 줄어든 누락 데이터를 새로 저장하기 위해 리스트 비우기

        count = count + 1

        get_content_info(r_missing_id)


    # 종료 시각
    end_time = datetime.now()

    # 소요 시간 확인
    print(end_time-start_time)
```

## 3 데이터 전처리(data_preprocessing.py)

크롤링한 데이터를 전처리한다. 여러 번의 크롤링 결과를 각 하나의 csv 파일에 저장하였으므로 동일한 id에 결측치 개수가 다른 행이 여러개 존재한다. 이를 처리하고, 데이터를 split하거나 동일한 형식으로 수정하며, 앞서 수집한 3개의 csv 파일을 id를 기준으로 merge한다.

Import 코드는 다음과 같다.

```
import pandas as pd
import numpy as np
import re # 구분자를 2개 이상으로 자르기 위한 import, 문자열에서 원하는 패턴을 검색하거나
대체하는데 사용
```

### 3.1 TV 프로그램의 에피소드 수와 상영시간 중복 데이터 처리(watcha_tv_info.csv)

왓챠에서 수집해온 TV 프로그램의 에피소드 수와 상영시간은 결측치가 없는 레코드를 모두 수집하였다. 따라서 결측치가 있는 행을 drop하고 동일한 csv 파일에 덮어쓴다.

```
# watcha_tv_info.csv 전처리(tv 프로그램의 에피소드 수와 상영시간)
# columns=['id', '에피소드 수','상영시간']
# 결측치가 없는 레코드를 모두 수집했으므로 결측치가 있는 행 drop

# 1. csv 파일 읽어오기
tv_info_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv', encoding='cp949')

# 2. ? 값을 NaN(결측치)으로 치환
tv_info_df.replace('?', pd.NA, inplace = True)

# 3. 결측치가 있는 행 제거
tv_info_df.dropna(inplace = True)

# 4. watcha_tv_info.csv 에 덮어쓰기
tv_info_df.to_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv',encoding='cp949', mode='w',
index=False)

# 5. 개수 비교
id_type_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_id_type.csv', encoding='cp949')
mask = id_type_df['종류'] == 'tv'
tv_id_type_df = id_type_df.loc[mask,:]

print('나와야 하는 tv 프로그램 수  : ' + str(tv_id_type_df['id'].count()))
print('전처리한 tv 프로그램 수 : ' + str(tv_info_df['id'].count()))
```

### 3.2 콘텐츠의 상세정보 중복 데이터 처리(watcha_content_info.csv)

왓챠피디아에서 수집해온 정보에는 결측치가 있는 레코드가 존재하였다. 따라서 한 레코드당 결측치가 제일 적은 레코드만 남기고 drop 한다.

```
# watcha_content_info.csv 전처리
# columns=['id', '컨텐츠 정보','평균 평점','줄거리','제작진']

# 1. 결측치가 있기 때문에 한 레코드당 결측치가 제일 적은 레코드만 남기고 drop

# 1-1. csv 파일 읽어오기
content_info_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_content_info.csv',
encoding='utf-8-sig')

# 1-2. ? 값을 NaN(결측치)으로 치환
content_info_df.replace('?', pd.NA, inplace = True)

# 1-3 각 id 그룹 내에서 나머지 칼럼들의 누락된 값이 가장 작은 레코드만 남도록 처리
def keep_row_with_least_nulls(group):

    # ID를 제외한 나머지 칼럼들에 대해 누락된 값의 개수를 계산
    null_counts = group.iloc[:, 1:].isnull().sum(axis=1)

    # 누락된 값의 개수가 가장 작은 인덱스를 반환
    min_null_count_idx = null_counts.idxmin()

    # 누락된 값이 가장 작은 레코드를 반환
    return group.loc[min_null_count_idx]

# id를 기준으로 그룹화하여 각 그룹에 대해 함수를 적용하고 결과를 저장
content_info_df =
content_info_df.groupby('id').apply(keep_row_with_least_nulls).reset_index(drop=True)

# 1-4 watcha_tv_info.csv에 덮어쓰기
content_info_df.to_csv('C:/Crawling_Watcha/csv/watcha_content_info.csv',encoding='utf-8-
sig', mode='w', index=False)

# 1-5 개수 비교
id_type_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_id_type.csv', encoding='cp949')
print('나와야 하는 콘텐츠 수  : ' + str(id_type_df['id'].count()))
print('전처리한 콘텐츠 수 : ' + str(content_info_df['id'].count()))
```

### 3.3 콘텐츠 정보 split하고 merge(watcha_content_info.csv, + watcha_id_type.csv)

왓챠피디아에서 크롤링해온 콘텐츠 정보를 split해서 재저장한다. 이때, 영화와 TV 프로그램의 정보가 다르므로 구분해서 처리하기 위해 watcha_id_type.csv 파일과 merge하여 처리한다.

또한 split하여 해당하는 각 정보 리스트에 저장할 때, 특정 정보가 웹사이트에서 처음부터 누락되어 있을 수도 있다. 이를 위한 예외처리를 길이와 연령정보를 사용하여 처리한다.

split한 데이터를 이용하여 만든 데이터프레임과 watcha_id_type에서 가져온 데이터프레임을 merge하여 새로운 csv 파일을 만든다.(split_merge.csv)

처리 후 칼럼은 id, 종류, 평균 평점, 줄거리, 제작진, 제목, 개봉년도, 장르, 국가, 상영시간, 연령등급, 방송국 이렇게 12개이다.

```
# 2. 콘텐츠 정보 split 해서 재저장
# 영화와 TV 프로그램의 정보가 다르므로 구분해서 처리하기 위해 merge

# 2-1 두 csv 파일 merge
merge_df = pd.merge(id_type_df, content_info_df, on='id', how='outer')

# 2-2 필요한 정보만 리스트로 추출
id_data = merge_df['id'].values.tolist()
type_data = merge_df['종류'].values.tolist()
info_data = merge_df['컨텐츠 정보'].values.tolist()

# 2-3 split 한 데이터를 저장할 리스트
titleList = []  # 제목
release_yerList = [] # 개봉년도
genreList = []  # 장르
contryList = [] # 국가
running_timeList = []   # 상영시간
age_gradeList = []  # 연령 등급
tv_stationList = [] # 방송국

# 연령정보 고유값
age_grade = ['전체','7 세','12 세','15 세','청불']


# 콘텐츠 정보 split
#for i in range(10):
for i in range(len(id_data)):
```

```
split_info = re.split(r' · |\n', info_data[i])

# 연령 정보가 split 리스트에 있는지 확인
age_found = False
for item in age_grade:
    if item in split_info:
        age_found = True
        break

titleList.append(split_info[0])
release_yerList.append(split_info[2])

if type_data[i] == "tv":

    running_timeList.append(None)

    if len(split_info) >= 7: # 모든 정보가 있는 경우

        if age_found == True: # 방송국 정보만 없는 경우
            tv_stationList.append(split_info[3])
            genreList.append(split_info[4])
            contryList.append(split_info[5])
            age_gradeList.append(split_info[6])

        else:
            tv_stationList.append(split_info[3])
            genreList.append(split_info[4])
            contryList.append(split_info[5])
            age_gradeList.append(None)

    elif len(split_info) == 6:

        if age_found == True: # 방송국 정보만 없는 경우
            tv_stationList.append(None)
            genreList.append(split_info[3])
            contryList.append(split_info[4])
            age_gradeList.append(split_info[5])

        else: # 연령 등급이 없는 경우
            tv_stationList.append(split_info[3])
```

```
            genreList.append(split_info[4])
            contryList.append(split_info[5])
            age_gradeList.append(None)


    else: #len(split_info) == 5:

        tv_stationList.append(None) # 방송국 정보가 없고

        if age_found == True: # 국가정보가 없는 경우
            genreList.append(split_info[3])
            contryList.append(None)
            age_gradeList.append(split_info[4])

        else: # 연령 등급이 없는 경우
            genreList.append(split_info[3])
            contryList.append(split_info[4])
            age_gradeList.append(None)


else: # 영화인 경우

    tv_stationList.append(None)

    if len(split_info) >= 7: # 모든 정보가 있는 경우

        genreList.append(split_info[3])
        contryList.append(split_info[4])
        running_timeList.append(split_info[5])

        if age_found == True: # 연령 등급이 있는 경우(7 or 10)
            age_gradeList.append(split_info[6])

        else: # 연령 등급이 없는 경우(9)
            age_gradeList.append(None)

    elif len(split_info) == 6:

        if age_found == True: # 상영시간만 없는 경우
            genreList.append(split_info[3])
            contryList.append(split_info[4])
```

```
                running_timeList.append(None)
                age_gradeList.append(split_info[5])

            else: # 연령 등급이 없는 경우
                genreList.append(split_info[3])
                contryList.append(split_info[4])
                running_timeList.append([split_info[5]])
                age_gradeList.append(None)


        else: #len(split_info) == 5, 상영시간과 연령정보가 없는 경우

            genreList.append(split_info[3])
            contryList.append(split_info[4])
            running_timeList.append(None)
            age_gradeList.append(None)



# merged_df 에 merge
# 데이터프레임 생성
split_df = pd.DataFrame({
    'id': id_data,
    '제목': titleList,
    '개봉년도': release_yerList,
    '장르': genreList,
    '국가': contryList,
    '상영시간': running_timeList,
    '연령등급': age_gradeList,
    '방송국': tv_stationList,
})

merge2_df = pd.merge(merge_df, split_df, on='id', how='outer')
merge2_df.drop(labels='컨텐츠 정보',axis=1, inplace=True)

print(merge2_df.count())
print(merge2_df.head(5))

merge2_df.to_csv('C:/Crawling_Watcha/csv/split_merge.csv',encoding='utf-8-sig', mode='w',
index=False)
```

### 3.4 콘텐츠 정보와 TV 프로그램 정보 merge(split_merge.csv, + watcha_tv_info.csv)

앞서 만든 데이터에 종류가 TV 프로그램인 레코드에 상영시간과 에피소드 수를 합친다.

```
# 모든 데이터 합치기
tv_info_df = pd.read_csv('C:/Crawling_Watcha/csv/watcha_tv_info.csv', encoding='cp949')
content_info_df = pd.read_csv('C:/Crawling_Watcha/csv/split_merge.csv', encoding='utf-8-
sig')

final_df = pd.merge(content_info_df, tv_info_df, on='id', how='outer')

# 조건에 따라 상영시간_y 와 상영시간 칼럼 값 합치기
final_df['상영시간'] = final_df.apply(lambda x: x['상영시간_y'] if x['종류'] == 'tv' else
x['상영시간_x'], axis=1)
final_df.drop(['상영시간_x', '상영시간_y'], axis=1, inplace=True)
```

### 3.5 데이터 형식 및 누락 데이터 처리

데이터 형식을 통일하도록 처리하고, 결측치를 처리하여 최종 csv 파일을 생성한다.(watcha.csv)

```
# 밀린 데이터 drop
drop_id = ['m5GX0v2','mOlEGLd','mOVvmLg','tR4JKKy','tR72L5x']
final_df = final_df.drop(final_df[final_df['id'].isin(drop_id)].index)

# '연령 등급' 열의 누락 데이터를 바로 앞에 있는 값으로 치환
final_df['연령등급'].fillna(method='ffill', inplace=True)

# '평균 평점' 열의 누락 데이터를 바로 앞에 있는 값으로 치환
final_df['평균 평점'].fillna(method='ffill', inplace=True)

# 결측치를 'unknown'으로 변경
final_df.fillna('unknown', inplace=True)

# 국가와 장르의 첫번째 값만 남기기
final_df['국가'] = final_df['국가'].apply(lambda x: x.split(",")[0])
final_df['장르'] = final_df['장르'].apply(lambda x: x.split("/")[0])

# 상영시간 칼럼에서 '[', ']', '"' 문자 제거하는 함수 정의
def remove_characters(text):
    text = text.replace('[', '').replace(']', '').replace("'", '')
    return text
```

```
# 상영시간 칼럼에 함수 적용하여 문자 제거 후 업데이트
final_df['상영시간'] = final_df['상영시간'].apply(remove_characters)

# '종류' 열의 값을 변경
final_df['종류'] = final_df['종류'].apply(lambda x: 'TV 프로그램' if x == 'tv' else '영화' if
x == 'movie' else x)

final_df.to_csv('C:/Crawling_Watcha/csv/watcha.csv',encoding='utf-8-sig', mode='w',
index=False)
```

# 4 데이터 분석 및 시각화 및 결론(data_analysis.py)

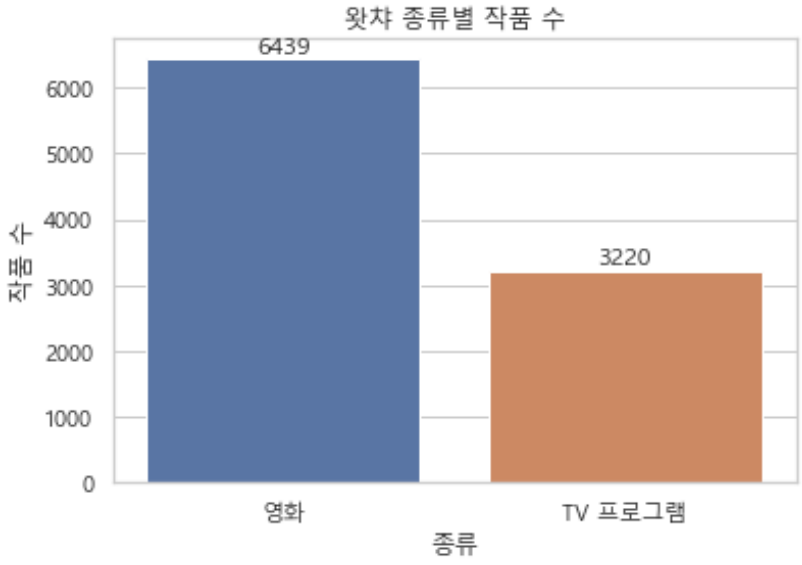
## 4.1 데이터 정보 확인

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

plt.rcParams['font.family'] = 'Malgun Gothic' # 한국어 텍스트에 폰트 지정

# csv 파일 읽어오기
watcha = pd.read_csv('C:/Crawling_Watcha/csv/watcha.csv', encoding='utf-8-sig')

# 데이터 정보 확인
watcha.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9659 entries, 0 to 9658
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      9659 non-null   object
 1   종류      9659 non-null   object
 2   평균 평점   9659 non-null   float64
 3   줄거리     9659 non-null   object
 4   제작진     9659 non-null   object
 5   제목      9659 non-null   object
 6   개봉년도    9659 non-null   int64
 7   장르      9659 non-null   object
 8   국가      9659 non-null   object
 9   연령등급    9659 non-null   object
 10  방송국     9659 non-null   object
 11  에피소드 수  9659 non-null   object
 12  상영시간    9659 non-null   object
dtypes: float64(1), int64(1), object(11)
memory usage: 981.1+ KB
```

## 4.2 종류별 작품 수

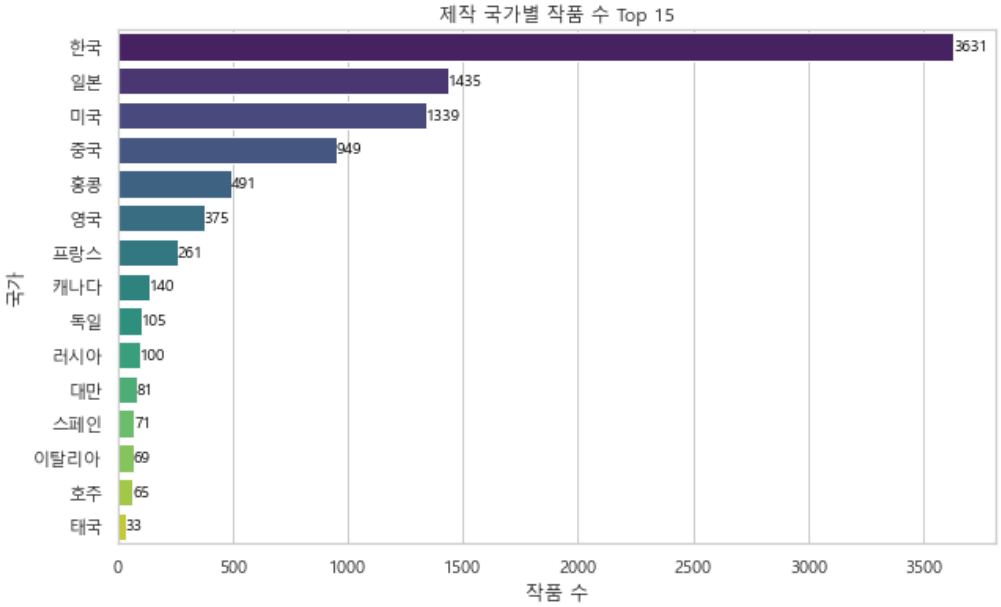
```
# 1. 종류별 작품 수

ax = sns.countplot(data=watcha, x='종류')

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
      va='bottom')

plt.ylabel('작품 수')  # y 축 이름 설정
plt.title('왓챠 종류별 작품 수') # 제목 추가

plt.show() # 시각화된 plot 보여줌
```



### 4.3 국가별 작품 수(상위 15개)

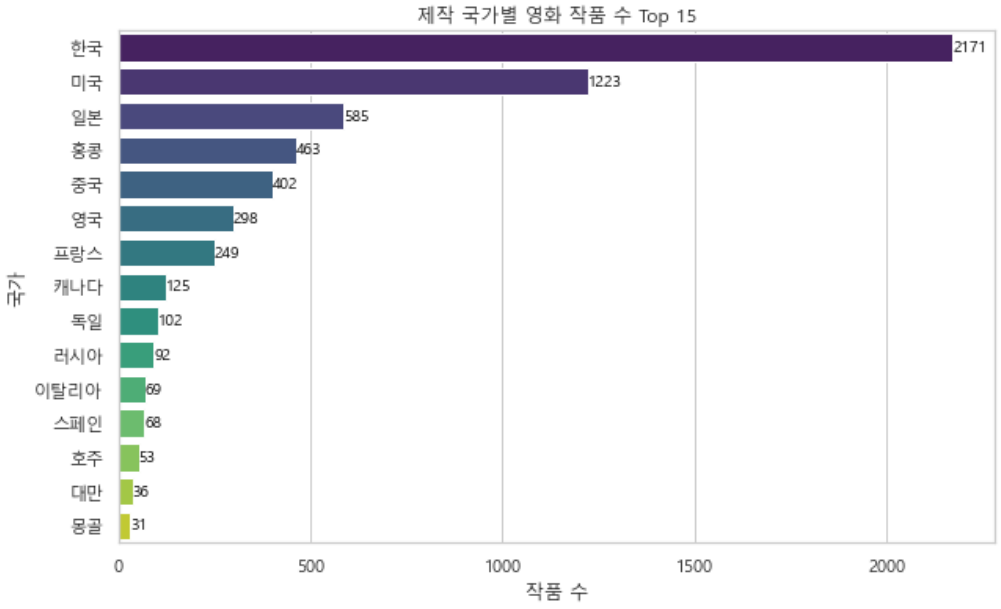
- 전체

```
# 2. 제작 국가별 작품 수(상위 15 개)

# 2-1 전체

# 전체 종류 중 제작 국가별 작품 수 상위 15 개 확인
print(watcha['국가'].value_counts().head(15))

# 국가별 작품 수 계산 후 정렬
country_counts = watcha['국가'].value_counts().head(15).sort_values(ascending=False)

# 가로 막대 그래프로 시각화
plt.figure(figsize=(10, 6))  # 그래프 크기 설정
sns.barplot(x=country_counts.values, y=country_counts.index, palette='viridis')

# 각 막대 안에 수치 표시
for index, value in enumerate(country_counts):
    plt.text(value, index, str(value), va='center', ha='left', fontsize=10, color='black')

plt.xlabel('작품 수')
plt.ylabel('국가')
plt.title('제작 국가별 작품 수 Top 15')

plt.show()
```



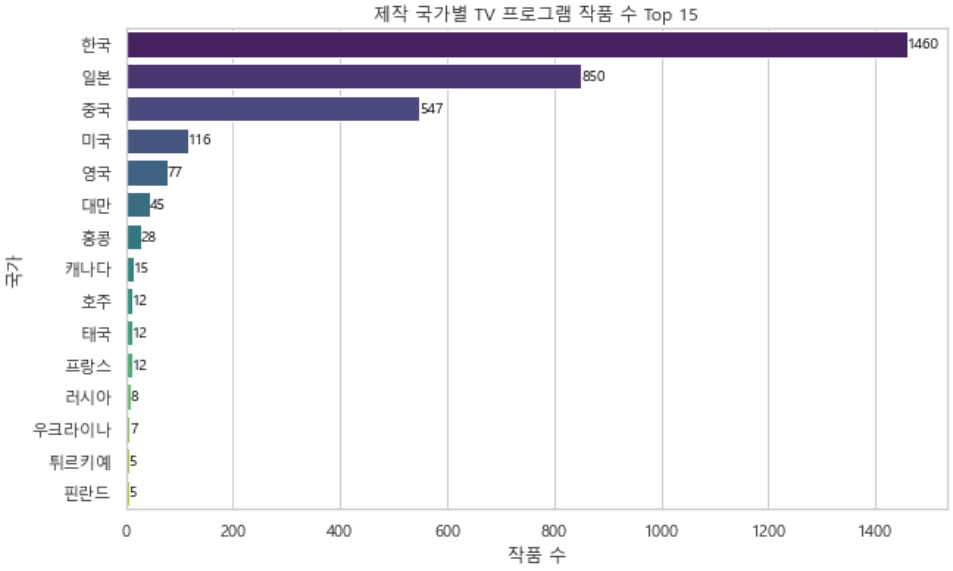
• 영화

```
# 2-2 영화

# 종류가 영화인 부분만 가져옴
movie = watcha[watcha['종류'] == '영화']

# 영화 중 제작 국가별 작품 수 상위 15 개 확인
print(movie['국가'].value_counts().head(15))

# 국가별 작품 수 계산 후 정렬
country_counts = movie['국가'].value_counts().head(15).sort_values(ascending=False)

# 가로 막대 그래프로 시각화
plt.figure(figsize=(10, 6))  # 그래프 크기 설정
sns.barplot(x=country_counts.values, y=country_counts.index, palette='viridis')

# 각 막대 안에 수치 표시
for index, value in enumerate(country_counts):
    plt.text(value, index, str(value), va='center', ha='left', fontsize=10, color='black')

plt.xlabel('작품 수')  # x 축 레이블 설정
plt.ylabel('국가')  # y 축 레이블 설정
plt.title('제작 국가별 영화 작품 수 Top 15')  # 그래프 제목 설정
plt.show()  # 그래프 표시
```



• TV 프로그램

```
# 2-3 TV 프로그램

# 종류가 tv 프로그램인 부분만 가져옴
tv = watcha[watcha['종류'] == 'TV 프로그램']

# tv 프로그램 중 제작 국가별 작품 수 상위 15 개 확인
print(tv['국가'].value_counts().head(15))

# 국가별 작품 수 계산 후 정렬
country_counts = tv['국가'].value_counts().head(15).sort_values(ascending=False)

# 가로 막대 그래프로 시각화
plt.figure(figsize=(10, 6))  # 그래프 크기 설정
sns.barplot(x=country_counts.values, y=country_counts.index, palette='viridis')

# 각 막대 안에 수치 표시
for index, value in enumerate(country_counts):
    plt.text(value, index, str(value), va='center', ha='left', fontsize=10, color='black')

plt.xlabel('작품 수')  # x 축 레이블 설정
plt.ylabel('국가')  # y 축 레이블 설정
plt.title('제작 국가별 TV 프로그램 작품 수 Top 15')  # 그래프 제목 설정
plt.show()  # 그래프 표시
```



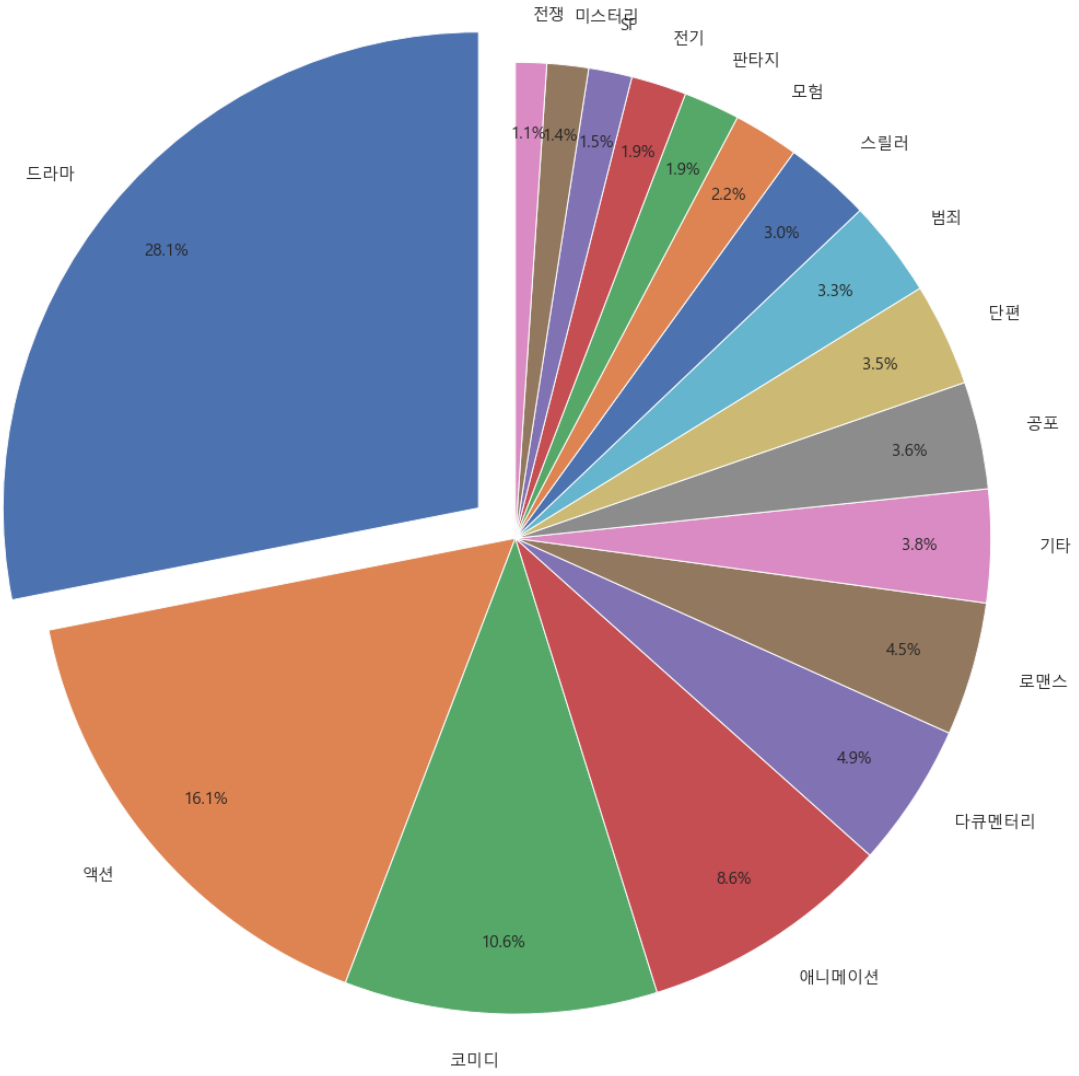
### 4.4 장르별 작품 수

- 영화

```
# 3. 장르별 작품 수

# 3-1 영화

# 영화만 필터링
movie = watcha[watcha['종류'] == '영화']

# 영화의 장르별 작품 수 계산
genre_counts = movie['장르'].value_counts()

# 작품 수의 비율 계산
genre_percentages = genre_counts / genre_counts.sum() * 100

# 1% 미만인 장르들을 '기타'로 통합
threshold = 1  # 임계치 설정
other_genres = genre_percentages[genre_percentages < threshold].index
movie.loc[movie['장르'].isin(other_genres), '장르'] = '기타'

# 다시 장르별 작품 수 계산
new_genre_counts = movie['장르'].value_counts()

# 새로운 작품 수의 비율 계산
new_genre_percentages = new_genre_counts / new_genre_counts.sum() * 100

# 가장 많은 비율을 차지하는 조각 강조를 위해 explode 설정
max_index = new_genre_percentages.idxmax()
explode = [0.1 if label == max_index else 0 for label in new_genre_percentages.index]

# 원 그래프 그리기 (크기 조절 및 텍스트 설정)
plt.figure(figsize=(20, 20))  # 그래프 크기 설정
patches, texts, autotexts = plt.pie(new_genre_percentages,
labels=new_genre_percentages.index,
                                    autopct='%1.1f%%', startangle=90, pctdistance=0.85,
labeldistance=1.1, explode=explode)
plt.title('장르별 영화 작품 수 비율', fontsize=20)  # 그래프 제목 설정
```

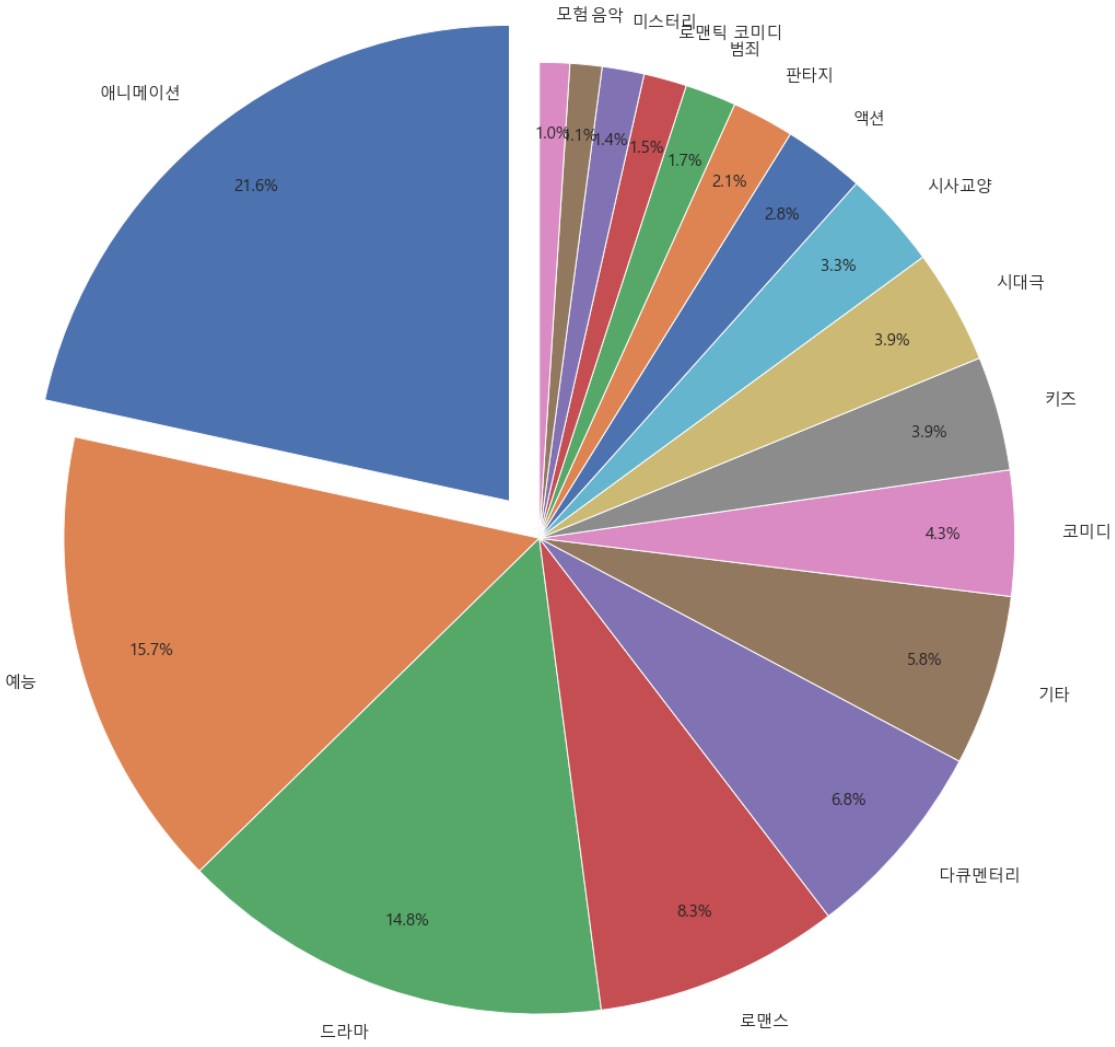
```
# 텍스트 사이즈 설정
for text in texts + autotexts:
    text.set_fontsize(15)

plt.show()  # 그래프 표시
```

장르별 영화 작품 수 비율



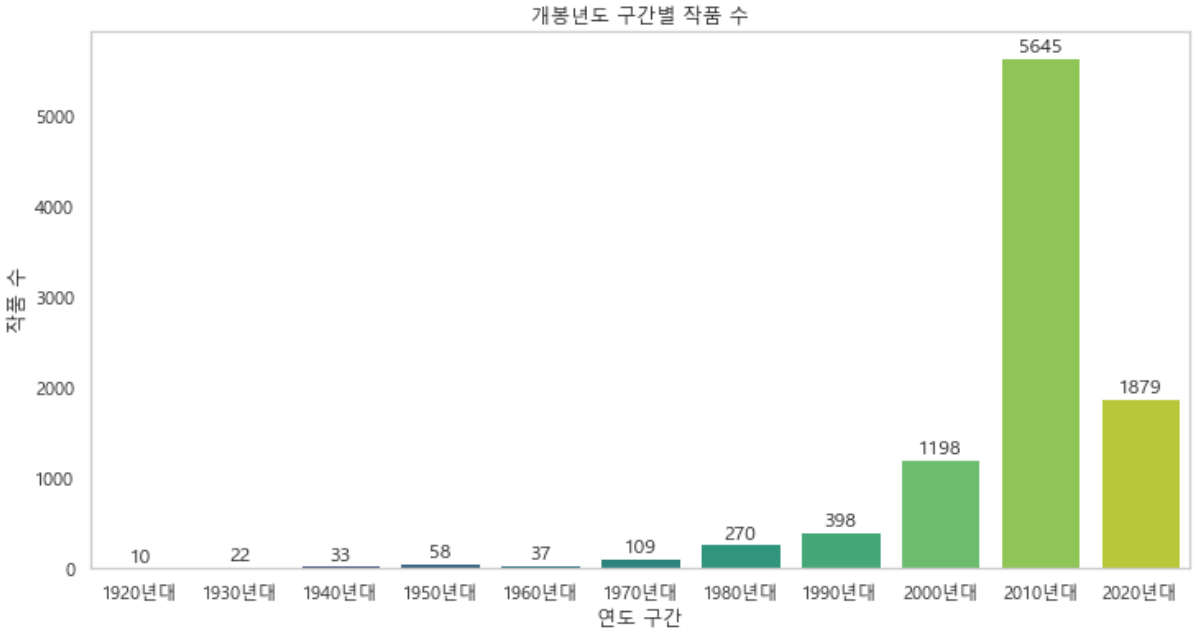
• TV 프로그램

```
# 3-2 TV 프로그램

# TV 프로그램만 필터링
tv = watcha[watcha['종류'] == 'TV 프로그램']

# TV 프로그램의 장르별 작품 수 계산
genre_counts = tv['장르'].value_counts()

# 작품 수의 비율 계산
genre_percentages = genre_counts / genre_counts.sum() * 100

# 1% 미만인 장르들을 '기타'로 통합
threshold = 1  # 임계치 설정
other_genres = genre_percentages[genre_percentages < threshold].index
tv.loc[tv['장르'].isin(other_genres), '장르'] = '기타'

# 다시 장르별 작품 수 계산
new_genre_counts = tv['장르'].value_counts()

# 새로운 작품 수의 비율 계산
new_genre_percentages = new_genre_counts / new_genre_counts.sum() * 100

# 가장 많은 비율을 차지하는 조각 강조를 위해 explode 설정
max_index = new_genre_percentages.idxmax()
explode = [0.1 if label == max_index else 0 for label in new_genre_percentages.index]

# 원 그래프 그리기 (크기 조절 및 텍스트 설정)
plt.figure(figsize=(20, 20))  # 그래프 크기 설정
patches, texts, autotexts = plt.pie(new_genre_percentages,
labels=new_genre_percentages.index,
                                    autopct='%1.1f%%', startangle=90, pctdistance=0.85,
labeldistance=1.1, explode=explode)
plt.title('장르별 TV 프로그램 작품 수 비율', fontsize=20)  # 그래프 제목 설정

# 텍스트 사이즈 설정
for text in texts + autotexts:
    text.set_fontsize(15)

plt.show()  # 그래프 표시
```


장르별 TV 프로그램 작품 수 비율
모험
음악
미스터리
로맨틱 코미디
범죄
판타지
액션
시사교양
시대극
키즈
코미디
기타
다큐멘터리
로맨스
드라마
예능
애니메이션
21.6%
15.7%
14.8%
8.3%
6.8%
5.8%
4.3%
3.9%
3.9%
3.3%
2.8%
2.1%
1.7%
1.5%
1.4%
1.1%
1.0%

### 4.5 개봉년도 구간별 작품 수

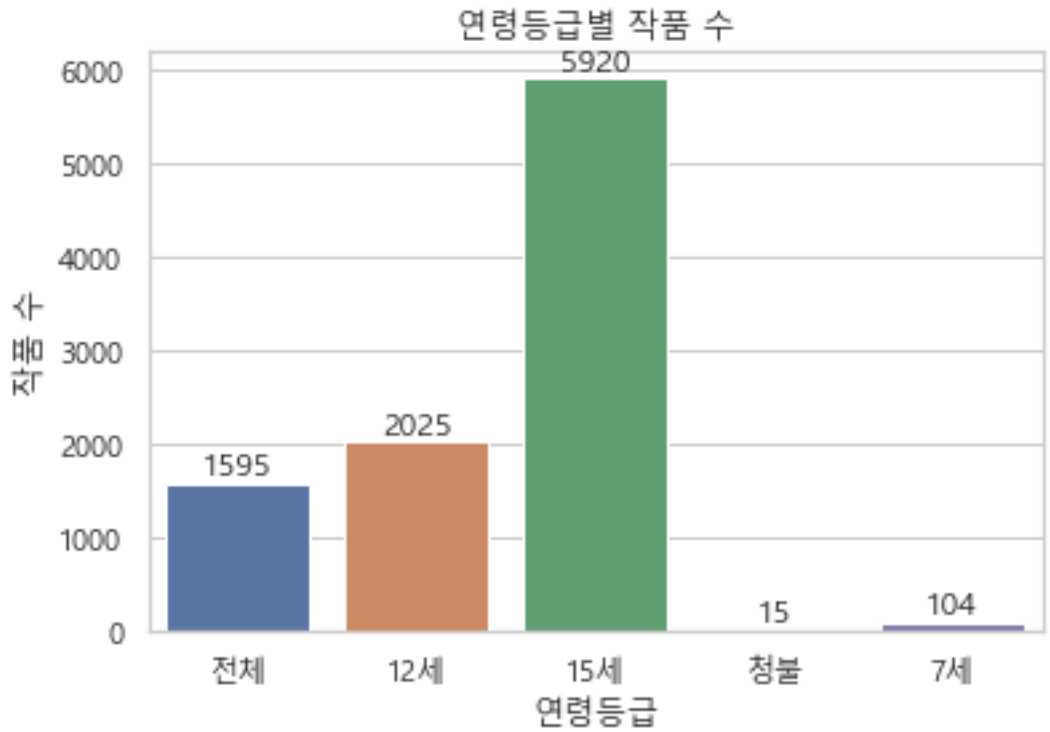
```
# 4. 개봉년도 구간별 작품 수

# 개봉년도별 작품 수 확인
print(watcha['개봉년도'].unique)

# 1920년대부터 2020년대까지 10년 단위로 구간 설정
bins = [1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2020, 2030]

# 개봉년도를 각 구간으로 나누기
watcha['연도_구간'] = pd.cut(watcha['개봉년도'], bins=bins, labels=['1920년대', '1930년대',
'1940년대', '1950년대', '1960년대', '1970년대', '1980년대', '1990년대', '2000년대',
'2010년대', '2020년대'])

# 각 구간별 작품 수 계산
yearly_counts_grouped = watcha['연도_구간'].value_counts().sort_index()

# 바 그래프 그리기
plt.figure(figsize=(12, 6))  # 그래프 크기 설정
ax = sns.barplot(x=yearly_counts_grouped.index, y=yearly_counts_grouped.values,
palette='viridis')  # 바 그래프 생성

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.xlabel('연도 구간')  # x축 레이블 설정
plt.ylabel('작품 수')  # y축 레이블 설정
plt.title('개봉년도 구간별 작품 수')  # 그래프 제목 설정
plt.grid(axis='y')  # y축 기준으로만 격자 표시
plt.show()  # 그래프 표시
```


개봉년도 구간별 작품 수
작품 수
0
1000
2000
3000
4000
5000
10
22
33
58
37
109
270
398
1198
5645
1879
1920년대
1930년대
1940년대
1950년대
1960년대
1970년대
1980년대
1990년대
2000년대
2010년대
2020년대
연도 구간

### 4.6 연령 등급별 작품수

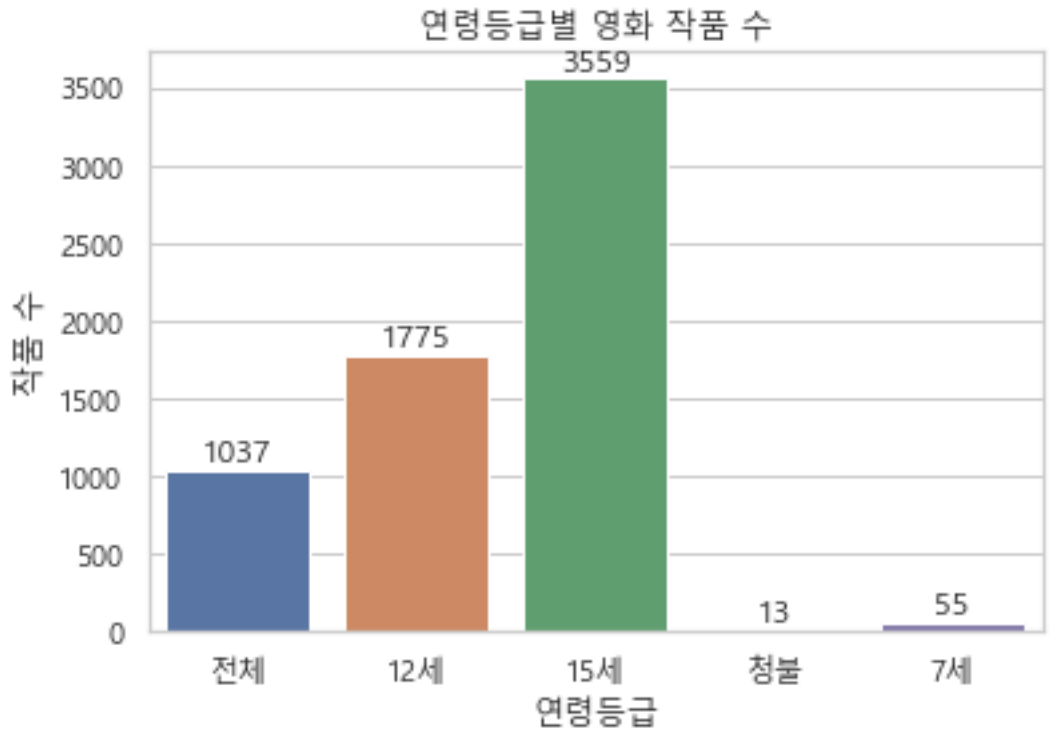
- 전체

```
# 5. 연령 등급별 작품 수

# 5-1 전체

ax = sns.countplot(data=watcha, x='연령등급')

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.ylabel('작품 수')  # y 축 이름 설정
plt.title('연령등급별 작품 수') # 제목 추가

plt.show() # 시각화된 plot 보여줌
```



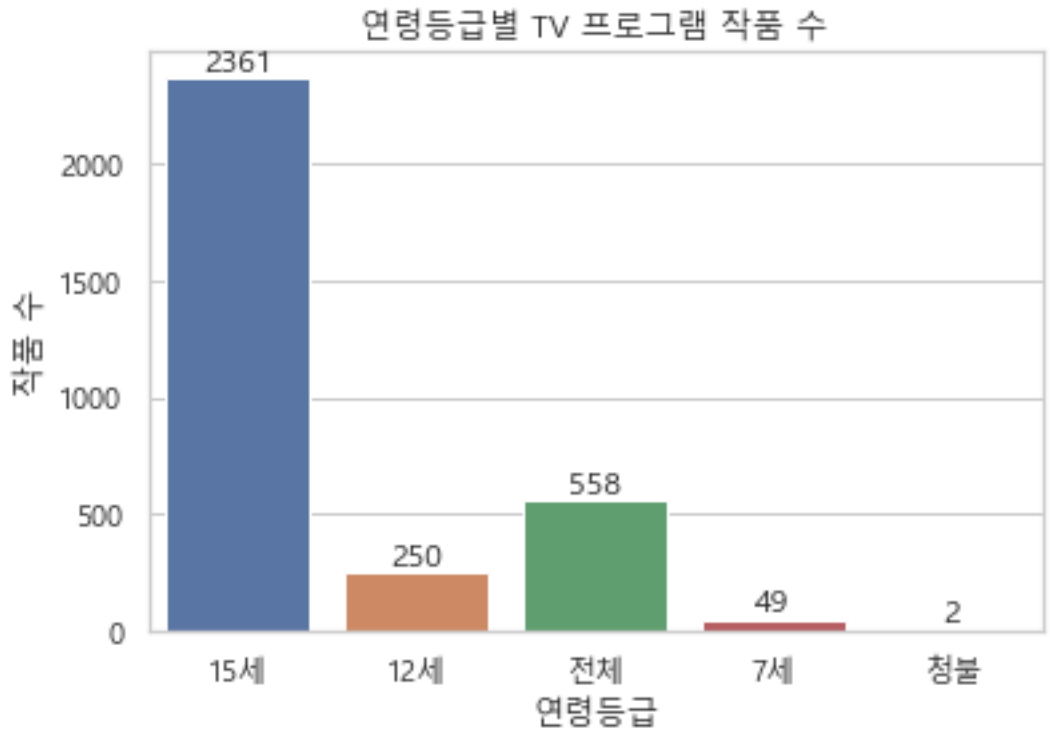
• 영화

```
# 5-2 영화

# 영화만 필터링
movie = watcha[watcha['종류'] == '영화']

ax = sns.countplot(data=movie, x='연령등급')

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.ylabel('작품 수')  # y 축 이름 설정
plt.title('연령등급별 영화 작품 수') # 제목 추가

plt.show() # 시각화된 plot 보여줌
```



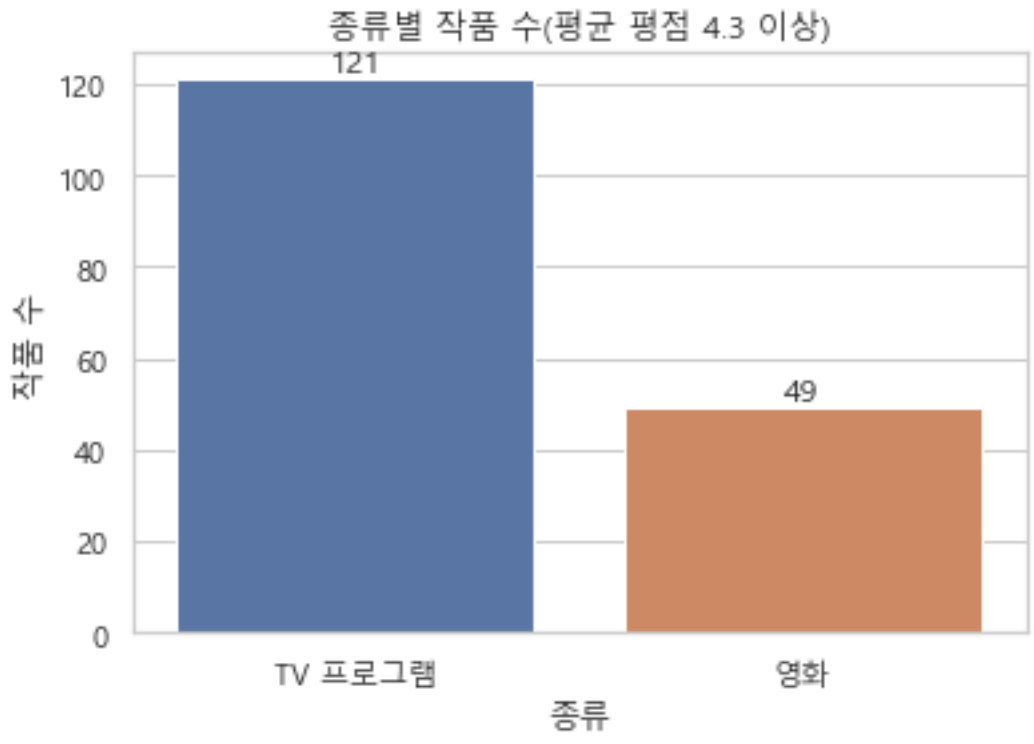
• TV 프로그램

```
# 5-3 tv 프로그램

# 영화만 필터링
tv = watcha[watcha['종류'] == 'TV 프로그램']

ax = sns.countplot(data=tv, x='연령등급')

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.ylabel('작품 수')  # y 축 이름 설정
plt.title('연령등급별 TV 프로그램 작품 수') # 제목 추가

plt.show() # 시각화된 plot 보여줌
```



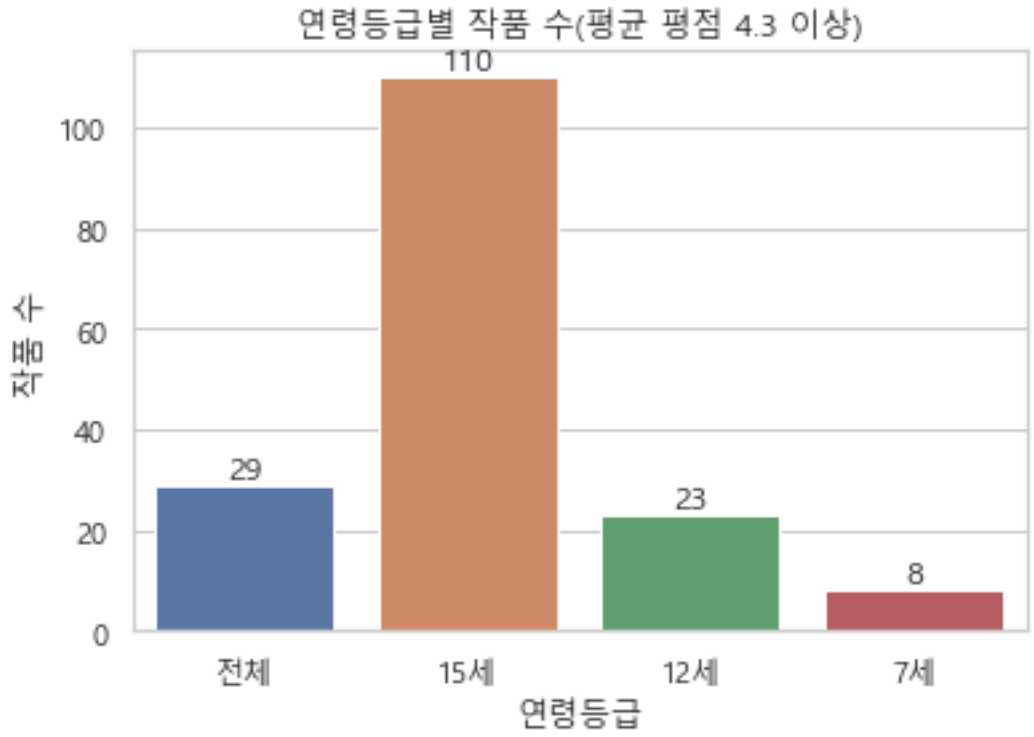
### 4.7 종류별 작품 수(평균 평점 4.3 이상)

```
# 7. 평균 평점 4.3 이상의 작품 분석

# '평균 평점' 열의 데이터를 숫자(float) 타입으로 변환
watcha['평균 평점'] = watcha['평균 평점'].astype(float)

# 평균 평점이 4.3 이상인 행 추출
content_4_3 = watcha[watcha['평균 평점'] >= 4.3]

# 7-1 종류별
ax = sns.countplot(data=content_4_3, x='종류')

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.ylabel('작품 수')  # y 축 이름 설정
plt.title('종류별 작품 수(평균 평점 4.3 이상)') # 제목 추가

plt.show() # 시각화된 plot 보여줌
```



4.8 연령 등급별 작품 수(평균 평점 4.3 이상)

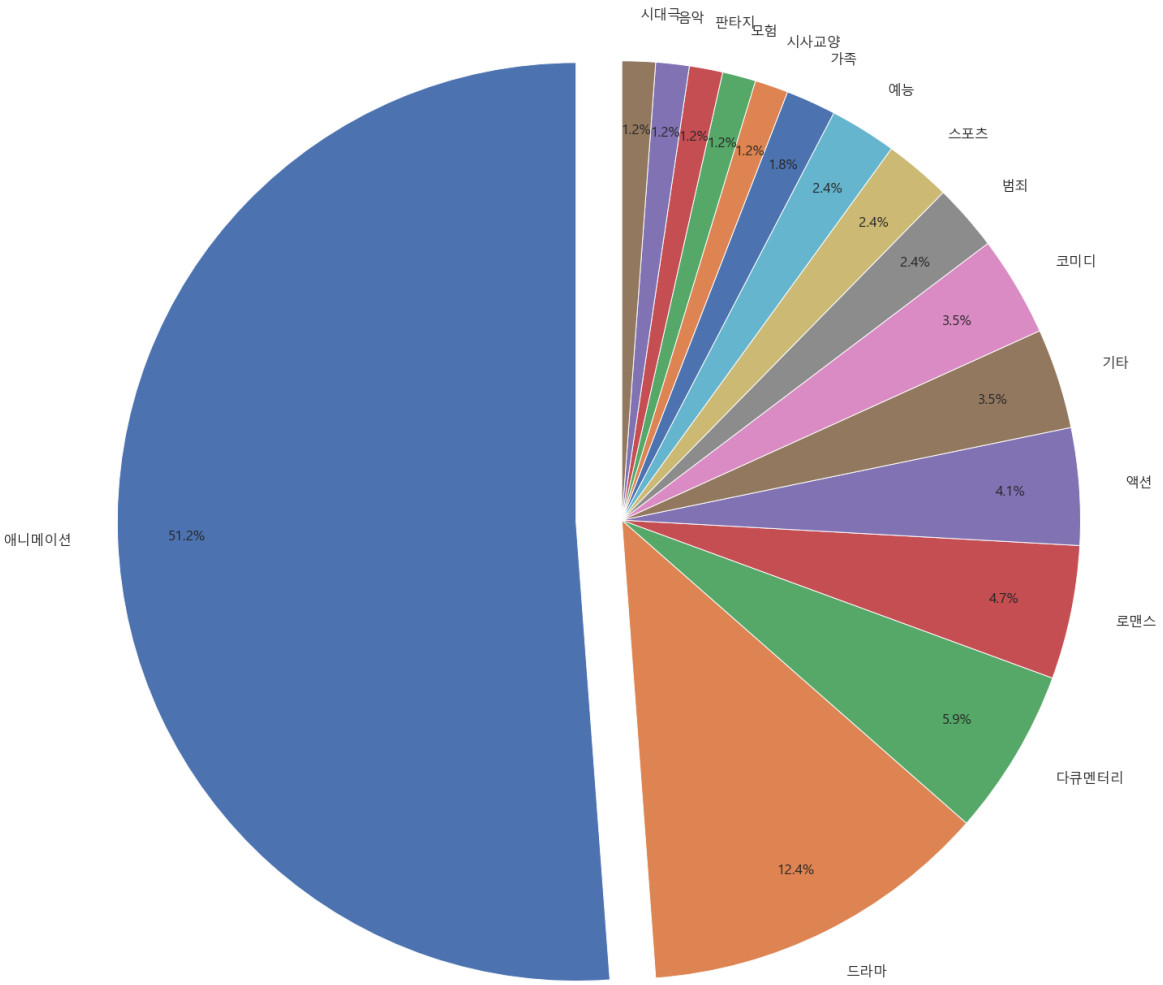
```
# 7-2 연령등급별
ax = sns.countplot(data=content_4_3, x='연령등급')

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.ylabel('작품 수')  # y 축 이름 설정
plt.title('연령등급별 작품 수(평균 평점 4.3 이상)') # 제목 추가

plt.show() # 시각화된 plot 보여줌
```



### 4.9 장르별 작품 수(평균 평점 4.3 이상)

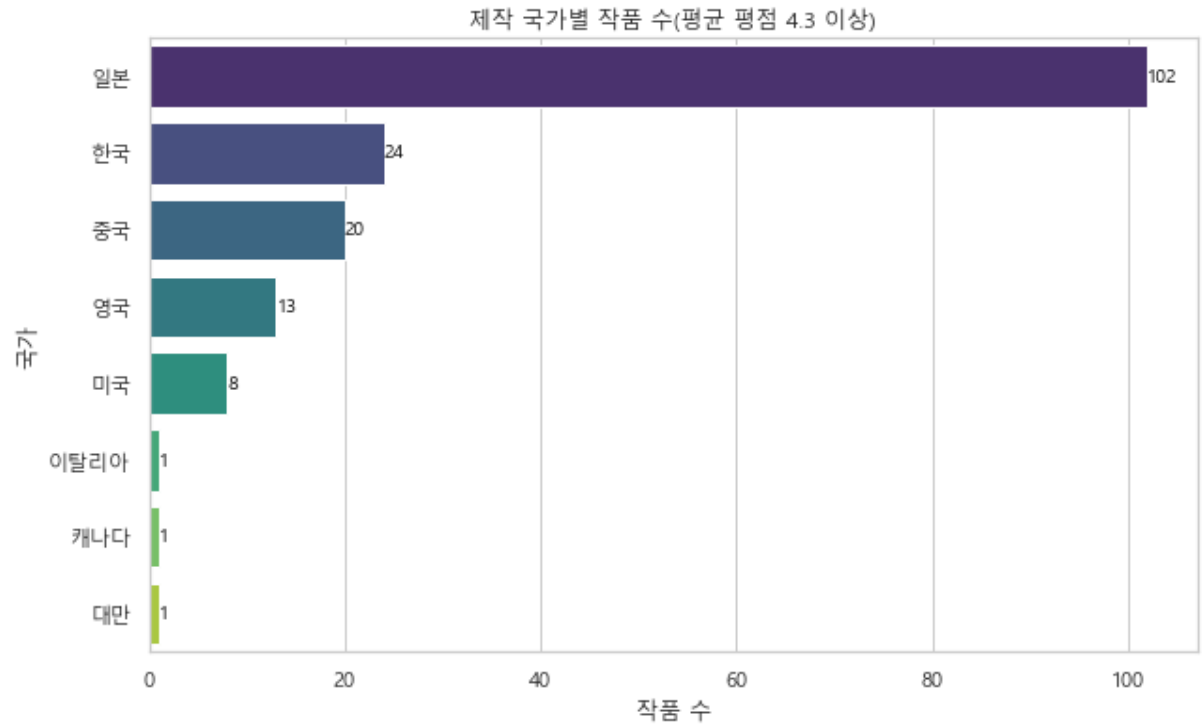
```
# 7-3 장르별

genre_counts = content_4_3['장르'].value_counts()
print(genre_counts)

# 작품 수의 비율 계산
genre_percentages = genre_counts / genre_counts.sum() * 100

# 2% 미만인 장르들을 '기타'로 통합
threshold = 1  # 임계치 설정
other_genres = genre_percentages[genre_percentages < threshold].index
content_4_3.loc[content_4_3['장르'].isin(other_genres), '장르'] = '기타'

# 다시 장르별 작품 수 계산
new_genre_counts = content_4_3['장르'].value_counts()

# 새로운 작품 수의 비율 계산
new_genre_percentages = new_genre_counts / new_genre_counts.sum() * 100

# 가장 많은 비율을 차지하는 조각 강조를 위해 explode 설정
max_index = new_genre_percentages.idxmax()
explode = [0.1 if label == max_index else 0 for label in new_genre_percentages.index]

# 원 그래프 그리기 (크기 조절 및 텍스트 설정)
plt.figure(figsize=(26, 26))  # 그래프 크기 설정
patches, texts, autotexts = plt.pie(new_genre_percentages,
labels=new_genre_percentages.index,
                                    autopct='%1.1f%%', startangle=90, pctdistance=0.85,
labeldistance=1.1, explode=explode)
plt.title('장르별 작품 수 비율(평균 평점 4.3 이상)', fontsize=20)  # 그래프 제목 설정

# 텍스트 사이즈 설정
for text in texts + autotexts:
    text.set_fontsize(17)

plt.show()  # 그래프 표시
```

장르별 작품 수 비율(평균 평점 4.3 이상)



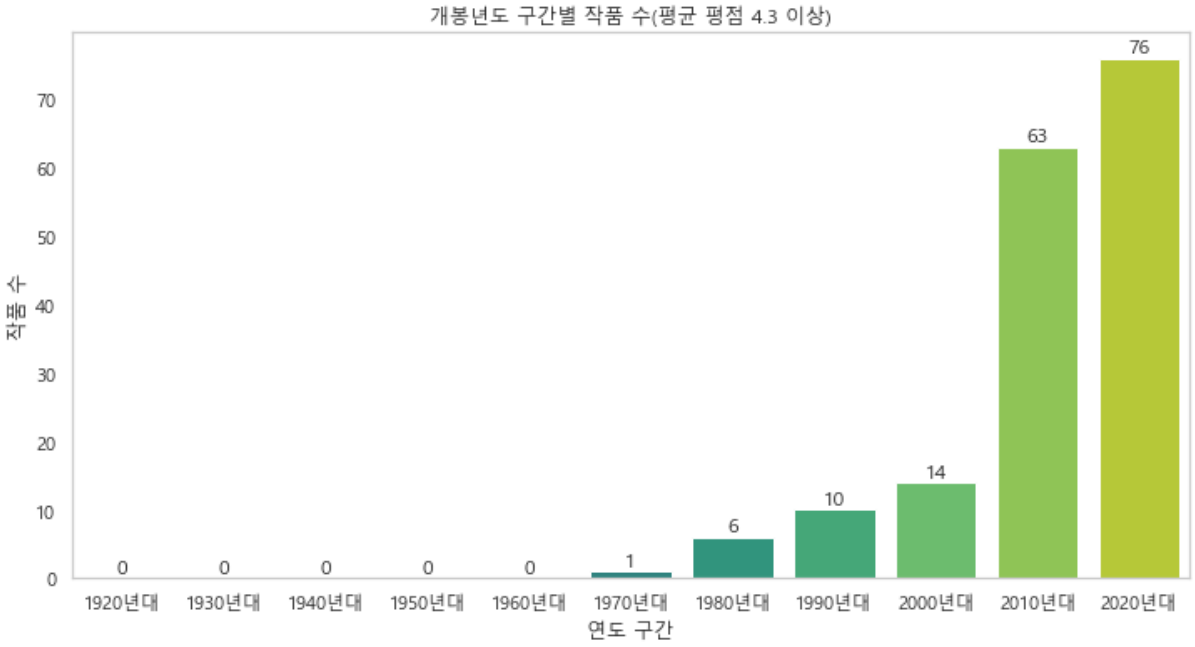
### 4.10 제작 국가별 작품 수(평균 평점 4.3 이상)

```
# 7-4 제작 국가별

# 제작 국가별 작품 수 계산 후 정렬
country_counts = content_4_3['국가'].value_counts().sort_values(ascending=False)

# 가로 막대 그래프로 시각화
plt.figure(figsize=(10, 6))  # 그래프 크기 설정
sns.barplot(x=country_counts.values, y=country_counts.index, palette='viridis')

# 각 막대 안에 수치 표시
for index, value in enumerate(country_counts):
    plt.text(value, index, str(value), va='center', ha='left', fontsize=10, color='black')

plt.xlabel('작품 수')  # x축 레이블 설정
plt.ylabel('국가')  # y축 레이블 설정
plt.title('제작 국가별 작품 수(평균 평점 4.3 이상)')  # 그래프 제목 설정
plt.show()  # 그래프 표시
```



### 4.11 개봉년도별 작품 수(평균 평점 4.3 이상)

```
# 7-5 개봉년도별

# 개봉년도별 작품 수 확인
print(content_4_3['개봉년도'].unique)

# 1920년대부터 2020년대까지 10년 단위로 구간 설정
bins = [1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2020, 2030]

# 개봉년도를 각 구간으로 나누기
content_4_3['연도_구간'] = pd.cut(content_4_3['개봉년도'], bins=bins, labels=['1920년대',
'1930년대', '1940년대', '1950년대', '1960년대', '1970년대', '1980년대', '1990년대',
'2000년대', '2010년대', '2020년대'])

# 각 구간별 작품 수 계산
yearly_counts_grouped = content_4_3['연도_구간'].value_counts().sort_index()

# 바 그래프 그리기
plt.figure(figsize=(12, 6))  # 그래프 크기 설정
ax = sns.barplot(x=yearly_counts_grouped.index, y=yearly_counts_grouped.values,
palette='viridis')  # 바 그래프 생성

# 각 막대 위에 수치 표시 (정수형태로)
for p in ax.patches:
    height = p.get_height()  # 막대의 높이(데이터 개수)
    ax.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}', ha='center',
va='bottom')

plt.xlabel('연도 구간')  # x축 레이블 설정
plt.ylabel('작품 수')  # y축 레이블 설정
plt.title('개봉년도 구간별 작품 수(평균 평점 4.3 이상)')  # 그래프 제목 설정
plt.grid(axis='y')  # y축 기준으로만 격자 표시
plt.show()  # 그래프 표시
```




### 4.12 결론

1) 종류별 작품 수

- 영화가 6,439개, TV 프로그램이 3,220개로 영화가 약 2배 더 많이 등록되어 있다.

2) 국가별 작품 수(상위 15개)

- 한국 작품이 총 3,631개(영화 2,171개, TV 프로그램 1,460개)로 가장 많이 등록되어 있다.

3) 장르별 작품 수

- 영화 장르에서는 드라마가 28.1%로 가장 많다.

- TV 프로그램 장르에서는 애니메이션이 21.6%로 가장 많다.

4) 개봉년도 구간별 작품 수

- 2010년대에 개봉된 작품이 5,645개로 가장 많다.

5) 연령 등급별 작품수

- 영화, TV프로그램 모두 15세 이상 등급이 제일 많았다.

6) 종류별 작품 수(평균 평점 4.3 이상)

- 170개의 작품 중 121개가 TV 프로그램이고 49개가 영화로, TV 프로그램이 영화의 약 2.5배 더 많다.

7) 연령 등급별 작품 수(평균 평점 4.3 이상)

- 15세 이상이 110개로 가장 많았다.

8) 장르별 작품 수(평균 평점 4.3 이상)

- 애니메이션이 51.2%로 가장 많았다.

9) 제작 국가별 작품 수(평균 평점 4.3 이상)

- 102개로 일본에서 제작된 작품이 가장 많았다.
- 한국 작품은 24개로 두 번째로 많았다.

10) 개봉년도별 작품 수(평균 평점 4.3 이상)

- 2020년대에 개봉한 작품이 76개로 가장 많았다.