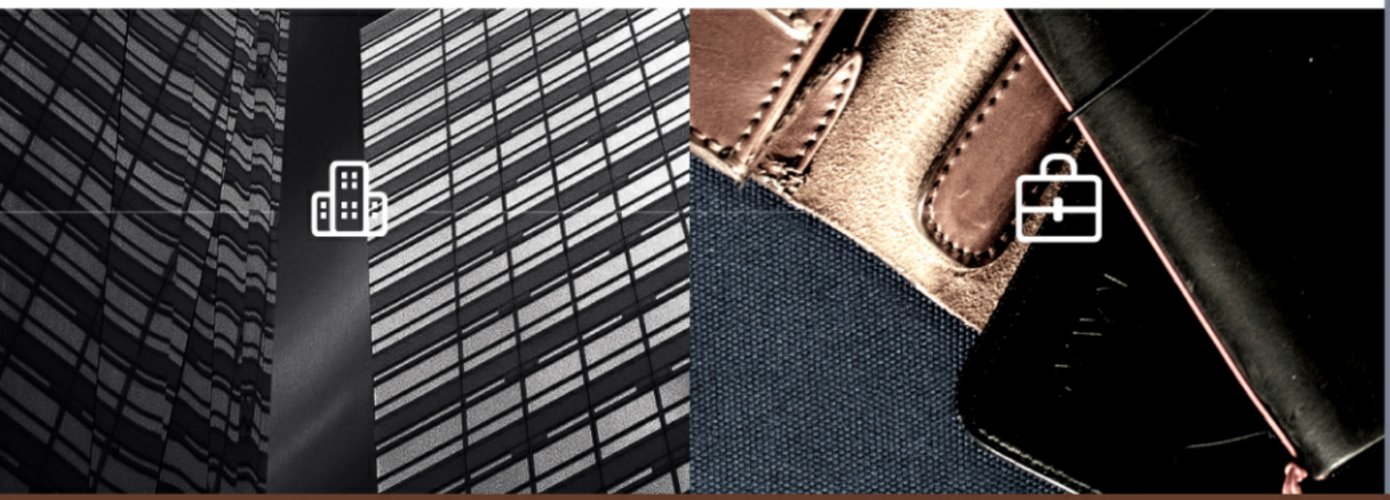
프로젝트 보고서

작성자 : 박성훈
조명 : 드랍 더 비트



목 차

01. 프로젝트 개요

최근 수행된 설문 조사에 따르면, 20대의 1,094명을 대상으로 한 조사 결과에 따르면, 20대의 주요 목표로는 내 집을 마련하는 것이 큰 비중을 차지합니다. 주식이나 코인을 포함한 재테크(20%), 대출 상환(27%), 여행(38%), 저축(52%) 등이 있지만, 내 집 마련이 75%의 응답자가 선택한 답안으로 나타났습니다. 이는 서울시 내에서 20대에게 내 집 소유가 큰 꿈이며, 인생의 중요한 목표임을 시사합니다.

이에 따라, 우리는 이러한 결과를 바탕으로 서울시 내의 집값을 조사하여, 현재의 계획에 맞는 최적의 지역을 찾아보고 집값을 빠르게 비교해보고자 합니다. 이를 통해 20대의 내 집 마련 꿈을 실현하기 위한 최상의 전략을 마련할 수 있을 것으로 기대됩니다.

부동산은 많은 사람들에게 주요한 자산으로 작용하기 때문에, 장기적인 자산 관리를 위해 부동산 가격 변동 추이를 조사하고자 하는 경우가 많습니다. 부동산 시장에 대한 장기적인 데이터 수집과 분석은 부동산 시장의 전반적인 흐름을 파악하고, 미래의 전망을 예측하는 데 도움이 됩니다. 이를 통해 부동산 시장의 동향을 이해하고, 다양한 산업 및 경제 지표와의 상호 연관성을 파악할 수 있습니다.

요약하면, 부동산 가격 변동 추이를 장기적으로 조사하고 분석하는 것은 부동산 시장의 향후 전망을 예측하고 다른 산업 및 경제 지표와의 관계를 파악하는 데 도움이 됩니다.

02. 웹 크롤링 과정

```
# BeautifulSoup을 사용하지않고 selenium만을 활용해서 데이터 수집을 진행 하였다.
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
import re
import csv
# ----------------------------------------------------------------------------------------------------------------
# 브라우저 진입
browser = webdriver.Chrome()
browser.get('https://new.land.naver.com/complexes?ms=37.3595704,127.105399,16&a=APT:ABYG:JGC:PRE&e=RETAIL')
# 네이버 부동산 사이트를 참고한 데이터
time.sleep(1)
# ----------------------------------------------------------------------------------------------------------------
#링크를 크롤링 하기 위한 사이트 링크 파도타기
element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
# 지도에 지역부분을 클릭해준다.
time.sleep(1)

element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/div/div[2]/ul/li[1]/label').click()
# 지역부분을 클릭해주고나서, 서울시를 클릭하여 서울시의 범위만을 지정한다.
time.sleep(1)
# ----------------------------------------------------------------------------------------------------------------
```

Selenium을 활용해서 데이터 수집을 진행하고, 네이버 부동산 사이트로 진입한다. 지도에 지역 부분을 클릭해서 서울시를 클릭 후 서울시의 범위만을 지정해서 웹 크롤링을 진행한다.

```
# 서울시의 모든 '구' 를 리스트화 시켜서 append 함
sigu_elements = browser.find_elements(By.CLASS_NAME, 'area_list--district')
sigu_list = []

for sigu_search in sigu_elements:
    # ₩n을 기준으로 문장을 분리하고 리스트에 추가
    sentences = sigu_search.text.split('₩n')
    sigu_list.extend(sentences)
# 본래의 데이터가 강남구₩n강서구₩n강북구₩n 처럼 되어있어서 ₩n을 기준점으로 잡아 스플릿을 돌려서
# 리스트에 하나 하나의 값으로 추가시켜주는 과정을 진행 하고 있다.
# ----------------------------------------------------------------------------------------------------------------
# 데이터 값 추출을 위한 for문
# 사용될 리스트를 코드가 진행되기 전에 선언한다.
apt_list = []
dong_list = []
price_list = []
final_list = []
# ----------------------------------------------------------------------------------------------------------------
```

서울시의 모든 자치구를 리스트화 시켜서 크롤링하면서 자치구 이름을 append 하면서 진행한다. 본래의 데이터가 강남구\n강서구\n강북구처럼 되어있기 때문에 \n 기준으로 스플릿을 해서 리스트 하나의 값으로 추가하게끔 진행한다. 그 후, 비슷하게 진행하기 위해서 아파트, 행정동 ,가격, 최종 리스트를 선언해서 만들어 둔다.

```
# 위에 과정에서 ['강남구','강북구','강서구' ....] 리스트를 만들었으니, 그 길이만큼 for문을 돌려서
# XAPTH를 활용해서 for문이 진행될 때, 해당 인덱스 값의 XPATH를 불러올 수 있도록 코드를 만들었다.
# 0번 XAPTH는 존재하지 않기 때문에, 숫자가 들어올 Sigu_data+1 을 넣어서 오류가 나오지않게 진행한다.
for sigu_data in range(len(sigu_list)):
    sigu_element = browser.find_element(By.XPATH, f'//*[@id="region_filter"]/div/div/div[2]/ul/li[{sigu_data+1}]/label').click()
    time.sleep(1)
    dong_elements = browser.find_elements(By.CLASS_NAME, 'area_list--district')
# '구'리스트를 뽑아왔으니, 이제 '동' 리스트를 만들어 처리방식은 아까와 동일하게 처리한다.
    for dong_search in dong_elements:
        # \n을 기준으로 문장을 분리하고 리스트에 추가
        sentences = dong_search.text.split('\n')
        dong_list.extend(sentences)
# ------------------------------------------------------------------------------------------------------
```

리스트화 시킨 것을 기반으로 for 문을 돌려서 '자치구' 만큼 반복한다.
이어서 행정동 리스트도 마찬가지로 \n 스플릿을 진행한다.

```
# 중첩 for문을 사용해서 '강남구'를 클릭하고 '개포동', '논현동', '대치동' ... 순서대로  위와 동일하게
# XPATH를 활용해서 맞는 링크를 클릭할 수 있도록 진행한다.
    for dong_data in range(len(dong_list)):
        dong_element = browser.find_element(By.XPATH, f'//*[@id="region_filter"]/div/div/div[2]/ul/li[{dong_data+1}]/label').click()
        time.sleep(1)
        apt_elements = browser.find_elements(By.CLASS_NAME, 'complex_title')
        apt_list_recycle = []
# 이번엔 해당 '동' 에 존재하는 아파트의 리스트를 위와 동일한 방식으로 처리하되, 이 데이터는 \n 이 없어서 간단한 방식으로
# 처리하였다. 추후에 모든 아파트의 데이터가 필요할지도 몰라서 for문이 돌때마다 초기화를 시켜줄 리스트와 모든 아파트 데이터를
# 포함하고 있는 리스트까지 총 두개의 리스트에게 append 시켜주고 있다.
        for apt_search in apt_elements:
            sentences = apt_search.text
            if sentences:
                apt_list.append(sentences)
                apt_list_recycle.append(sentences)
# ------------------------------------------------------------------------------------------------------
```

'자치구' 가 클릭이 되었다면, 이제 '자치구' 를 눌렀을 때 나타나는 '행정동'을 클릭하게 해서, 해당하는 행정동의 '아파트' 를 리스트에 append 한다.

```
# ------------------------------------------------------------------------------------------------------
# 위에 '구','동' 을 돌리던 for문과 동일하게 아파트 리스트의 for문을 돌려서 해당 링크를 눌러서 작업을 반복하게 해준다.
# 'execute_script' 코드를 활용해서, 아파트 리스트가 많아, 스크롤 다운을 진행해야 하는데 'arguments[0].click()' 을
# 활용해서 스크롤 다운 대신에 그 값을 찾아서 오류가 나지 않게끔 코드를 진행시켜 준다.
        for apt_data in range (len(apt_list_recycle)):
            button = browser.find_element(By.XPATH, f'//*[@id="region_filter"]/div/div/div[3]/ul/li[{apt_data+1}]/a')
            browser.execute_script('arguments[0].click()', button)
            time.sleep(1)
# 아파트의 매물이 없는경우에는 (By.LINK_TEXT, '시세/실거래가') 값이 존재하지 않는 경우가 있어서, try를 활용하여,
# 해당 링크가 있다면 눌러서 진행하고, 해당 링크가 존재하지 않는다면 except 처리하여, 해당 데이터의 값은
# '평수','가격' 에 'None' 값을 대신 넣어서 해결해줄 생각이다.
            try:
                apt_won = browser.find_element(By.LINK_TEXT, '시세/실거래가')
                apt_won.click()
                time.sleep(1)
# ------------------------------------------------------------------------------------------------------
```

이제 마찬가지로 아파트 리스트 만큼의 for 반복문을 통해서 아파트 가격의 '실 거래가'를 웹 크롤링 하는 작업을 최종 진행한다. 대신에 해당 '행정동' 에 아파트 매물이 존재하지 않는다면, try~except를 활용해서 있다면 클릭하고 없다면 다음 for 반복문으로 넘어가게끔 설계를 한다.



```
# ------------------------------------------------------------------------------------------------------------------------
# 이제 (By.LINK_TEXT, '시세/실거래가') 링크를 클릭해서 들어오면, 해당 아파트의 평수의 갯수들이 나오게 되는데,
# 이것도 마찬가지로 평수가 적을 경우는 더보기 링크가 없기 때문에, 위와 동일하게 try 를 활용해서 처리하였다.
                try:
                    arrow = browser.find_element(By.XPATH, '//*[@id="detailContents2"]/div[1]/div/div[2]/button')
                    if arrow.text == '더보기':
                        arrow.click()
                        time.sleep(1)
                        area_elements = browser.find_elements(By.CLASS_NAME, 'detail_sorting_tab')
                        time.sleep(1)
# 평수만큼 for문을 또 돌려야 하기 때문에, '아파트' 리스트와 동일하게 처리하였다.
                        area_list = []
                        for area_search in area_elements:
                            sentences = area_search.text
                            if sentences:
                                area_list.append(sentences)
# 존재하는 평수의 갯수만큼 for을 돌리는데, 해당 데이터에는 '전세','월세','매매' 데이터값을 포함하고 있어서
# 뒤에꺼 3개는 빼고 for문을 돌려주고 있다. 이렇게해야 오류가 나지 않는다.
                        for area_data in area_list[:-3]:
                            browser.find_element(By.LINK_TEXT, f'{area_data}').click()
                            time.sleep(1)
# ------------------------------------------------------------------------------------------------------------------------
```

이제 아파트를 클릭했다면, 해당 아파트의 32평 28평 등 평수의 종류가 존재한다면, 위의 코드와 동일하게 try~except를 진행해서 처리한다. 그리고 평수만큼의 리스트를 만들어서 평수를 따로 저장 후에, 처리할 수 있도록 한다. 그리고 평수가 존재하는 만큼 for 문을 돌리지만, '전세', '월세', '매매' 링크가 존재하기 때문에, '매매' 의 매물만 조사할 것 이기에, 매매만 선택할 수 있도록 만들어 준다.

```
# ------------------------------------------------------------------------------------------------------------------------
                        # 해당 평수의 '상한가'를 지정해서 text화 시키고, 가격이 존재하면 try를 활용해서 final_list에
                        # '구','동','아파트','평수','가격' 순서대로 최종리스트에 값을 넣어주고 있다.
                        try:
                            price = browser.find_element(By.XPATH, '//*[@id="tabpanel1"]/div[3]/div[1]/div/div[2]/strong').text
                            price_list.append(price)
                            final_list.append((sigu_list[sigu_data],dong_list[dong_data],apt_list_recycle[apt_data],area_data,price))
                            time.sleep(1)
                        # '상한가' 가격이 존재하지 않다면, '평수' 까지만 리스트에 저장하고, 뒤에값은 'None'을 삽입 하였다.
                        except:
                            final_list.append((sigu_list[sigu_data],dong_list[dong_data],apt_list_recycle[apt_data],area_data,'None'))
                        # 더보기 링크를 다시한번 클릭해서 더보기 창을 닫아준다.
                    if arrow.text == '더보기':
                        arrow.click()
                        time.sleep(1)
# ------------------------------------------------------------------------------------------------------------------------
```

이제 해당 평수에 '상한가' 매물만을 리스트에 값을 넣어줄 수 있도록 코드를 만든다. 그리고 최종 리스트에 '자치구', '행정동', '아파트', '평수', '가격' 순서대로 최종 리스트에 들어갈 수 있게 코드 설계를 진행한다. 그리고 '상한가'가 존재하지 않는다면, 'None' 데이터를 추가하도록 만든다.



```
# ------------------------------------------------------------------------------------------------------------------------
            # 위에 코드와 동일한 코드인데, 처음에 더보기 링크가 존재하지 않을 때에, 예외처리 되서 넘어온 경우
            # 어쨋든, '평수' 가 존재는 하지만, '평수' 의 갯수가 적어서 더보기 링크가 없는 경우이기 때문에,
            # 위와 동일한 코드 흐름 방식을 적용시켜서 누락되는 데이터가 없게 하였다.
            except:
                area_elements = browser.find_elements(By.CLASS_NAME, 'detail_sorting_tab')
                time.sleep(1)
                area_list = []
                for area_search in area_elements:
                    sentences = area_search.text
                    if sentences:
                        area_list.append(sentences)
                for area_data in area_list[:-3]:
                    browser.find_element(By.LINK_TEXT, f'{area_data}').click()
                    time.sleep(1)
                    try:
                        price = browser.find_element(By.XPATH, '//*[@id="tabpanel1"]/div[3]/div[1]/div/div[2]/strong').text
                        price_list.append(price)
                        final_list.append((sigu_list[sigu_data],dong_list[dong_data],apt_list_recycle[apt_data],area_data,price))
                        time.sleep(1)
                    except:
                        final_list.append((sigu_list[sigu_data],dong_list[dong_data],apt_list_recycle[apt_data],area_data,'None'))
# ------------------------------------------------------------------------------------------------------------------------
```

위의 코드와 비슷한 프로그래밍인데, '평수' 가 존재는 하지만 '평수'의 개수가 적어서 '더보기' 링크가 없던 것이기 때문에 누락되는 데이터가 없게끔 해결해 주는 코드 이다.

```
# ------------------------------------------------------------------------------------------------------------------------
        # 맨처음에 (By.LINK_TEXT, '시세/실거래가') 링크 텍스트가 존재하지 않을 때, 예외처리를 했던 코드인데,
        # '평수' 와 '가격' 쪽에 'None' 값을 넣어서, 마지막 처리를 해주고 있다. 그리고 모든 코드의 흐름이 끝나면
        # 창닫기 버튼을 눌러서 해당 아파트의 창을 닫아주는 작업을 진행하고 있다.
        except:
            price_list.append('None')
            final_list.append((sigu_list[sigu_data],dong_list[dong_data],apt_list_recycle[apt_data],'None','None'))
        # 아파트가 아예 존재하지 않는 '구' 가 존재하는 경우가 있는데, 이 경우에 예외처리를 진행하기 위해서 닫기 버튼이
        # 없을 때 예외처리를 할 수 있는 코드를 작성하게 되었다.
        try:
            close_button = browser.find_element(By.XPATH, '//*[@id="ct"]/div[2]/div[2]/div/button').click()
        except:
            element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
# ------------------------------------------------------------------------------------------------------------------------
```

맨 처음 등장한 try~except 코드에서 except 처리 부분이다. 예외 처리를 하게 되면 '시세/실거래가' 가 아예 없는 '아파트'의 경우 'None' 값을 넣어 평수와 가격이 빈 것을 표기해 줄 것이다.



```
# ------------------------------------------------------------------------------------------------------------
            # 해당 '동' 하나의 아파트 데이터를 크롤링 하였다면, 다시 재귀해서 돌아가기 위해 지역부분을 다시 클릭해준다.
            # 서울시를 누른후에 다시 해당 '구' 를 클릭후 다시 똑같은 '동' 으로 돌아가도록 클릭하게 하여,
            # 다음 아파트를 크롤링 하게끔 코드를 진행하게 하였다.
            element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
            time.sleep(1)

            element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/div/div[2]/ul/li[1]/label').click()
            time.sleep(1)

            browser.find_element(By.XPATH, f'//*[@id="region_filter"]/div/div/div[2]/ul/li[{sigu_data+1}]/label').click()
            time.sleep(1)

            dong_element = browser.find_element(By.XPATH, f'//*[@id="region_filter"]/div/div/div[2]/ul/li[{dong_data+1}]/label').click()
            time.sleep(1)
# ------------------------------------------------------------------------------------------------------------
        # 해당 '동'의 모든 아파트의 크롤링을 마치고 나면, 위와 동일하게 지역부분을 다시 클릭해주고, 서울시를 누른다음에,
        # 다시 해당 '구' 로 돌아오게하여, 다음 '동' 의 아파트를 크롤링 하게 만들어 준다.
        element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
        time.sleep(1)

        element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
        time.sleep(1)

        element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/div/div[2]/ul/li[1]/label').click()
        time.sleep(1)

        browser.find_element(By.XPATH, f'//*[@id="region_filter"]/div/div/div[2]/ul/li[{sigu_data+1}]/label').click()
        time.sleep(1)
# ------------------------------------------------------------------------------------------------------------
    # 해당하는 '구' 에 '동' 에 있는 모든 아파트를 크롤링 하였다면, 다음 '구' 로 넘어가기 위해서, 지역부분을 클릭한후에
    # '서울시' 까지만 클릭해주게 해주고, 쭉쭉 for문으로 재귀하게 만들고 있다.
    element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
    time.sleep(1)

    element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/a/span[1]').click()
    time.sleep(1)

    element = browser.find_element(By.XPATH, '//*[@id="region_filter"]/div/div/div[2]/ul/li[1]/label').click()
    time.sleep(1)
# ------------------------------------------------------------------------------------------------------------
```

각각 코드는 '아파트', '행정동', '자치구' for 반복문의 최종 부분이다. 코드가 정상 진행이 될 수 있도록 링크를 클릭해서 별다른 문제없이 반복이 진행될 수 있도록 만들어 준다.

```
# ------------------------------------------------------------------------------------------------------------
# 최종적으로 ['구', '동', '아파트', '평수', '가격'] 를 맨 상단에 삽입하고, final_list를 csv 파일로 쓰기를 진행해서 모든 데이터를 삽입한다.
def apt_xml(file):
    with open(file, "w", newline='', encoding='UTF-8') as file_w:
        writer = csv.writer(file_w)
        writer.writerow(sort_list)
        writer.writerows(final_list)
sort_list = ['구','동','아파트','평수','가격']
apt_xml("apt_up.csv")
print('apt_up 저장 완료!')
```

최종 리스트에는 ['자치구', '행정동', '아파트', '평수', '가격'] 이 저장될 것이며 이 데이터를 마지막으로 CSV 파일로 보내주어 데이터 분석을 진행할 수 있게 만들어 준다.

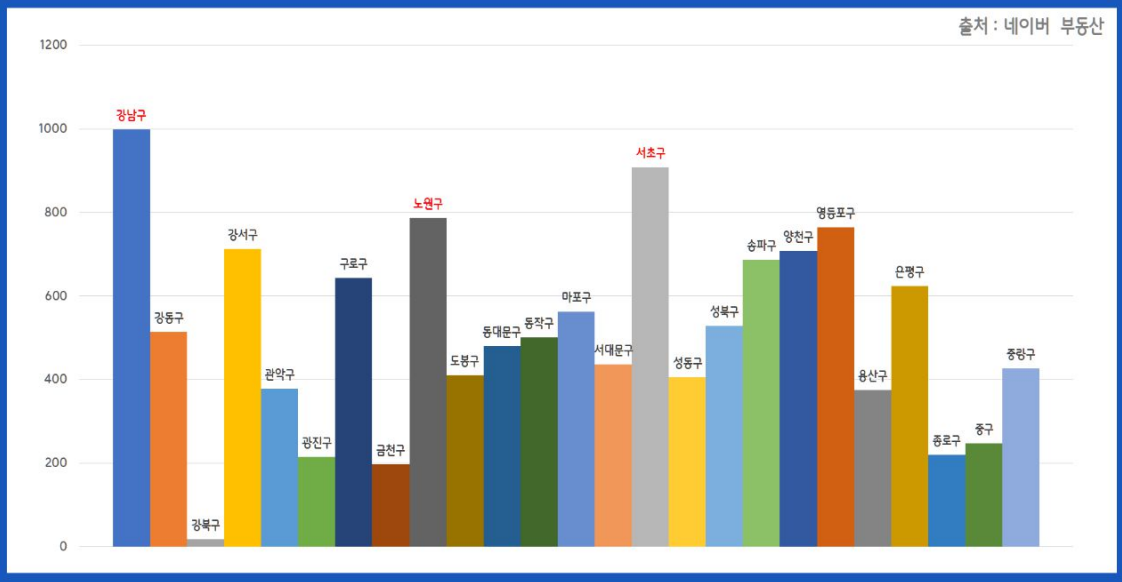
03. 분석 및 시각화

```
write_sigu_max = data.groupby('구').max()
write_sigu_max.to_csv('sigu_max.csv')
write_sigu_max
```

구	동	아파트	면적(m²)	가격	평
강남구	청담동	현대비전21(주상복합)	202	8500000000	61.104910
강동구	암사동	희훈리치파크	116	1980000000	35.089948
강북구	우이동	아파트경남아너스빌(1356)	116	800000000	35.089948
강서구	화곡동	희훈리치파크	116	1700000000	35.089948
관악구	신림동	현대	116	1340000000	35.089948
광진구	자양동	현대몰라트리움13차(주상복합)	116	3300000000	35.089948
구로구	항동	효성	116	2100000000	35.089948
금천구	시흥동	현대힐타운	116	1390000000	35.089948
노원구	하계동	효성	116	1385000000	35.089948
도봉구	창동	현대성우	116	1200000000	35.089948
동대문구	휘경동	힐스테이트청계	116	1640000000	35.089948
동작구	흑석동	힐스테이트상도센트럴파크	116	2300000000	35.089948
마포구	현석동	현대예술	116	2900000000	35.089948
서대문구	홍제동	훼미리	116	1770000000	35.089948
서초구	잠원동	현대성우(주상복합)	116	7150000000	35.089948
성동구	홍익동	행당브라운스톤	116	7600000000	35.089948
성북구	하월곡동	한신,한진	116	1863330000	35.089948
송파구	풍납동	호수(家)하우스(도시형)	116	4450000000	35.089948
양천구	신정동	힐탑이루미(주상복합)	116	3300000000	35.089948
영등포구	영등포동8가	휴브리지(도시형)	116	4750000000	35.089948
용산구	한남동	현대맨숀	116	4850000000	35.089948
은평구	진관동	힐스테이트녹번역	116	1555000000	35.089948
종로구	홍파동	킹스매너	116	2800000000	35.089948
중구	흥인동	황학아크로타워(주상복합)	116	3300000000	35.089948
중랑구	중화동	형진	116	1725000000	35.089948

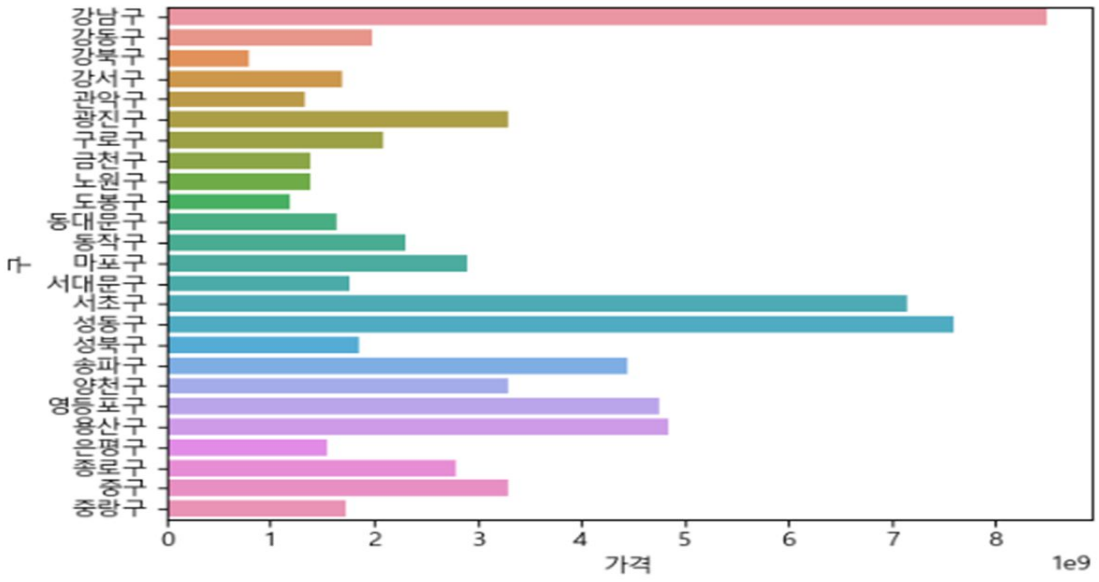
각 '자치구' 별로 최고가 매물 아파트만을 뽑아본 데이터이다. 웹 크롤링이 문제없이 잘 진행된 것으로 보인다.

서울시 구 별 아파트 수 비교



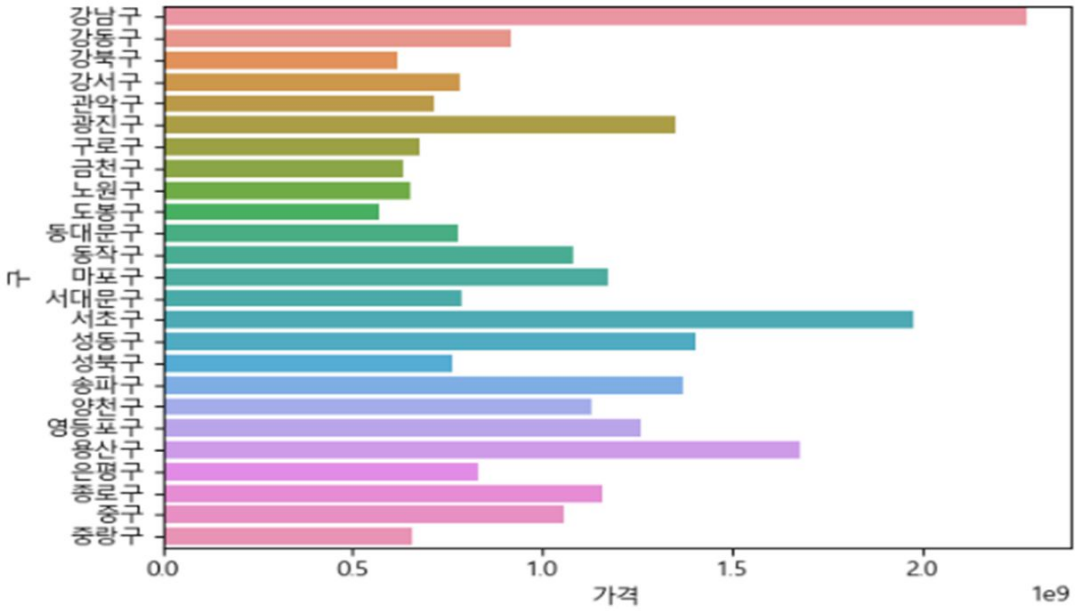
서울시 '자치구' 별로 아파트 숫자의 비교이다. 강남구와 서초구가 가장 많은 아파트가 있고, 강북구는 가장 적은 아파트 수를 보여주고 있다.

서울시 지역별 최고가



서울시 지역별 최고가 아파트이다. 강남구가 가장 높은 매물 가격을 보여주고 있으며, 이어서 성동구와 서초구가 뒤따라 가고 있으며, 그 외 지역은 큰 차이 나지 않는 최고가 가격을 보여주고 있다.

서울시 지역별 평균가



서울시 지역별 평균가인데, 강남의 평균 가격이 가장 높은 수치를 보여주고 있으며, 그 다음에 서초구, 용산구가 높은 가격을 보여준다. 그 외 지역은 어느정도 평균 가격이 비슷한 위치에 있는 것을 확인할 수 있었다.

04. 결론

이 데이터 수집과 시각화를 통해 우리는 서울시 부동산 시장의 현황을 더 잘 이해할 수 있게 되었습니다. 먼저, 아실과 네이버 부동산 사이트를 통해 수집한 정보를 통해 서울시 25개구에 위치한 467개 동에 총 3,846개의 아파트에 대한 실 거래가를 상세히 파악할 수 있었습니다. 이를 통해 서울시 내 다양한 지역에서 발생하는 부동산 시장의 동향을 분석할 수 있었습니다.

특히, 아실 사이트의 iframe을 통해 데이터를 수집하는 과정에서 Selenium을 사용하면서 발생한 문제들을 해결하는 과정은 새로운 도전이었습니다. 또한, 네이버 부동산 사이트에서도 데이터를 웹 크롤링 하는 과정에서 예외 처리를 통해 문제를 극복하는 등 다양한 어려움을 만났지만, 이를 해결하고 데이터를 성공적으로 수집할 수 있었습니다.

시각화를 통해 자치구별로 아파트의 수, 최고가, 최저가, 그리고 평균가를 비교함으로써, 서울시 각 자치구의 부동산 시장에서의 위치와 특징을 파악할 수 있었습니다. 이러한 분석을 통해 부동산 투자나 거주지 선정 등의 의사 결정에 도움을 줄 수 있을 것으로 기대됩니다.

05. 느낀점

팀별 단위로 진행한 첫 프로젝트였습니다. 처음에는 Iframe과 같이 웹 크롤링하기 어려운 과제에 직면했지만, 결국 데이터 수집에는 성공했습니다. 그러나 데이터 가공에 어려움을 겪게 되었고, 팀원들과 의견을 조율한 끝에 네이버 부동산 사이트로 다시 웹 크롤링을 시작하게 되었습니다. 이후에는 프로젝트를 성공적으로 완료하여, 팀원들과의 신뢰를 쌓고 서로 돈독한 관계를 형성하게 되었습니다.