main.py

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
import json
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import time

#I first imported what I would most likely need in Python and made sure to install anything I didn't have previously.

driver_path = "C:/Users/10bra.LAPTOP-539022QK/Downloads/chromedriver-win64 (1)/chromedriver-win64/chromedriver.exe"
service = Service(executable_path=driver_path)
options = webdriver.ChromeOptions()
driver = webdriver.Chrome(service=service, options=options)
time.sleep(5)
URL = "https://sacramento.craigslist.org/search/apa?postal=95616&search_distance=10"
driver.get(URL)
links = driver.find_elements("xpath", '//a[@tabindex and contains(@class, "cl-app-anchor text-only posting-title")]')
urls = ["https://sacramento.craigslist.org/search/apa?postal=95616&search_distance=10#search=1~gallery~0~0",
        "https://sacramento.craigslist.org/search/apa?postal=95616&search_distance=10#search=1~gallery~1~0",
        "https://sacramento.craigslist.org/search/apa?postal=95616&search_distance=10#search=1~gallery~2~0",
        "https://sacramento.craigslist.org/search/apa?postal=95616&search_distance=10#search=1~gallery~3~0",
        "https://sacramento.craigslist.org/search/apa?postal=95616&search_distance=10#search=1~gallery~4~0"]


#I set up the selenium server and I'm not exactly sure why it opens with chromeoption() and it doesn't without it, but
#it was working so I decided just not to touch the code. I found that using docker wasn't as reliable for my side, so I
 #set the driver to the driver path on my computer and navigated to the url. The links function was originally for
# gathering the links of the houses so I could navigate to each page and extract data from all of them.


#RENT PRICE
def rent_function(driver, URL):
    rent_amount_spans = driver.find_elements("class name", "priceinfo")
    rent_amounts = []
    for span in rent_amount_spans:
        rent_amount = span.text
        rent_amounts.append(rent_amount)

    return rent_amounts

rent = rent_function(driver, URL)

#I created a function that would find the rent price using the left top corner of each entry. Since the data was in a
# class named priceinfo, it was easy to extract using that name, and there were exactly 120 entries, so it seems like
# there wasn't any mistake in solving it this way.

#BEDROOM
def bedroomfunc(driver, URL):
    numbedroom = driver.find_elements("class name", "post-bedrooms")
    bedrooms = []
    for bed in numbedroom:
        bedcount = bed.text
        bedrooms.append(bedcount)
    return bedrooms

bedcounter = bedroomfunc(driver, URL)

#The bedrooms were located in a class named post-bedrooms, however as the page updated there were houses that didn't
# post the number of bedrooms on the front. It was quite frustrating dealing with this data, so I decided to go through
# the script to see if it would change anything.

#Square Feet
def sqftfunc(driver, URL):
    sqft = driver.find_elements("class name", "post-sqft")
    mansion = []
    for size in sqft:
        sqftcount = size.text
        mansion.append(sqftcount)
    return mansion

sqftcounter = sqftfunc(driver, URL)

#The square feet was found using post-sqft, which showed the number of square feet that was on the main url page.
# It kept the f2 at the end as I figured that wasn't a problem, and I mostly copied my format for the last two
# questions.

max_length = max(len(rent), len(bedcounter), len(sqftcounter))
rent += [None] * (max_length - len(rent))
bedcounter += [None] * (max_length - len(bedcounter))
sqftcounter += [None] * (max_length - len(sqftcounter))

#Since the number of data kept changing on the front page, I set it to show none values for houses that didn't contain
# the class name I inputted.
```

```python
page = requests.get(URL)
soup = BeautifulSoup(page.content, "html.parser")
script = soup.find_all("script")[1].text.strip()
scriptdata = json.loads(script)

#Beautiful soup wasn't very useful because it couldn't read a lot of the data I needed it to which is why I decided to
# try and do everything in the driver. However, it could find the script data and load it as a json, so I could use the
# tags like type to find the type of apartment and much more data like the amenities that were offered.


def housing_data(scriptdata):
    results = []
    count = 0
    for item_data in scriptdata.get('itemListElement', []):
        item = item_data.get('item', {})
        numberOfBedrooms = item.get('numberOfBedrooms')
        numberOfBathroomsTotal = item.get('numberOfBathroomsTotal')
        typehouse = item_data["item"]["@type"]
        amenities = item_data["item"]["name"]
        address = item_data.get('item', {}).get('address', {})
        locality = address.get('addressLocality', '')
        region = address.get('addressRegion', '')
        details = (locality, region, numberOfBedrooms, numberOfBathroomsTotal, typehouse, amenities)
        results.append(details)
        count += 1
        if count >= 120:
            break
    return results

houdata = housing_data(scriptdata)

#I set it to find the item list elements and items as that's how they were intitialized. There was a lot of empty data
# in this data, so I only used the data that was mostly not empty in order to find what was needed.

rent_amounts_all_urls = rent_function(driver, urls)
print(rent_amounts_all_urls)
def rent_function(driver, URLs):
    all_rent_amounts = []
    for URL in URLs:
        driver.get(URL)
        rent_amount_spans = driver.find_elements("class name", "priceinfo")
        rent_amounts = [span.text for span in rent_amount_spans]
        all_rent_amounts.extend(rent_amounts)
    return all_rent_amounts

rent_amounts_all_urls = rent_function(driver, urls)

def sqftfunc(driver, URLs):
    all_sqfts = []
    for URL in URLs:
        driver.get(URL)
        sqft = driver.find_elements("class name", "post-sqft")
        sqft_counts = [size.text for size in sqft]
        all_sqfts.extend(sqft_counts)
    return all_sqfts

def housing_data(driver, URLs):
    all_results = []
    for URL in URLs:
        driver.get(URL)
        soup = BeautifulSoup(page.content, "html.parser")
        script = soup.find_all("script")[1].text.strip()
        scriptdata = json.loads(script)
        results = []
        count = 0
        for item_data in scriptdata.get('itemListElement', []):
            item = item_data.get('item', {})
            numberOfBedrooms = item.get('numberOfBedrooms')
            numberOfBathroomsTotal = item.get('numberOfBathroomsTotal')
            typehouse = item_data["item"]["@type"]
            amenities = item_data["item"]["name"]
            address = item_data.get('item', {}).get('address', {})
            locality = address.get('addressLocality', '')
            region = address.get('addressRegion', '')
            details = (locality, region, numberOfBedrooms, numberOfBathroomsTotal, typehouse, amenities)
            results.append(details)
            count += 1
            if count >= 120:
                break
        all_results.extend(results)
    return all_results



sqft_counts_all_urls = sqftfunc(driver, urls)
housing_data_all_urls = housing_data(driver, urls)

#The data was perfect for one URL, and I originally set it to where the data would use the next button to retrieve data
# from 5 links, but time permitting and my Python knowledge being a little more limited I decided to just take the 5
# urls I would need, although it would not be as practical in a real work environment. I redid all the functions but
# applied 5 urls to them instead of one.

max_length = max(len(rent_amounts_all_urls), len(sqft_counts_all_urls), len(housing_data_all_urls))
sqft_counts_all_urls += [None] * (max_length - len(sqft_counts_all_urls))
housing_data_all_urls += [None] * (max_length - len(housing_data_all_urls))
data = {
    'Rent Amount': rent_amounts_all_urls,
    'Square Footage': sqft_counts_all_urls,
    'Locality': [item[0] if item else None for item in housing_data_all_urls],
    'Region': [item[1] if item else None for item in housing_data_all_urls],
    'Number of Bedrooms': [item[2] if item else None for item in housing_data_all_urls],
    'Number of Bathrooms': [item[3] if item else None for item in housing_data_all_urls],
    'Type of House': [item[4] if item else None for item in housing_data_all_urls],
    'Amenities': [item[5] if item else None for item in housing_data_all_urls]
}

df = pd.DataFrame(data)
df.to_csv("sac.csv", index = False)

#This resulted in my 600 row dataframe, exactly what I needed. I created an comma separated excel file as sac.csv in
# order to keep all of the data in an organized manner.
```

craigslist.py

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
import json
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import time

# I essentially used the same steps to find the data points for the sf east bay data as I did for the sacramento houses.
# Instead of going over my though process, I figured it would be better to interpret the results

driver_path = "C:/Users/10bra.LAPTOP-539022QK/Downloads/chromedriver-win64 (1)/chromedriver-win64/chromedriver.exe"
service = Service(executable_path=driver_path)
options = webdriver.ChromeOptions()
driver = webdriver.Chrome(service=service, options=options)
time.sleep(5)
URL2 = "https://sfbay.craigslist.org/search/apa?postal=eastba&search_distance=6"
driver.get(URL2)

URLs2 = ["https://sfbay.craigslist.org/search/apa?postal=eastba&search_distance=6#search=1~gallery~0~0",
        "https://sfbay.craigslist.org/search/apa?postal=eastba&search_distance=6#search=1~gallery~1~0",
        "https://sfbay.craigslist.org/search/apa?postal=eastba&search_distance=6#search=1~gallery~2~0",
        "https://sfbay.craigslist.org/search/apa?postal=eastba&search_distance=6#search=1~gallery~3~0",
        "https://sfbay.craigslist.org/search/apa?postal=eastba&search_distance=6#search=1~gallery~4~0"]

def rent_function2(driver, URL2):
    rent_amount_spans2 = driver.find_elements("class name", "priceinfo")
    rent_amounts2 = []
    for span2 in rent_amount_spans2:
        rent_amount2 = span2.text
        rent_amounts2.append(rent_amount2)

    return rent_amounts2

rent2 = rent_function2(driver, URL2)

#The average rent amount in San Fransisco seemed to be much higher as compared to sacramento. The rent prices were
#almost double in many places, which is a little surprising but it makes sense.

def sqftfunc2(driver, URL2):
    sqft2 = driver.find_elements("class name", "post-sqft")
    mansion2 = []
    for size2 in sqft2:
        sqftcount2 = size2.text
        mansion2.append(sqftcount2)
    return mansion2

sqftcounter2 = sqftfunc2(driver, URL2)

#Luckily, the houses weren't only more expensive, but their price made up for it by being way bigger houses. The
# square feet of these houses really lived up to the name mansion I gave them, which explains the difference in price.

def bedroomfunc(driver, URL2):
    numbedroom = driver.find_elements("class name", "post-bedrooms")
    bedrooms = []
    for bed in numbedroom:
        bedcount = bed.text
        bedrooms.append(bedcount)
    return bedrooms

bedcounter2 = bedroomfunc(driver, URL)

#The average amount of bedrooms was really similar in both areas, which could be interesting given the size differences
#of the houses.
```

```python
max_length = max(len(rent2), len(bedcounter2), len(sqftcounter2))
rent2 += [None] * (max_length - len(rent))
bedcounter2 += [None] * (max_length - len(bedcounter))
sqftcounter2 += [None] * (max_length - len(sqftcounter))

page2 = requests.get(URL2)
soup2= BeautifulSoup(page2.content, "html.parser")
script2 = soup2.find_all("script")[1].text.strip()
scriptdata2 = json.loads(script2)

def housing_data2(scriptdata2):
    results2 = []
    count2 = 0
    for item_data2 in scriptdata2.get('itemListElement', []):
        item2 = item_data2.get('item', {})
        numberOfBedrooms2 = item2.get('numberOfBedrooms')
        numberOfBathroomsTotal2 = item2.get('numberOfBathroomsTotal')
        typehouse2 = item_data2["item"]["@type"]
        amenities2 = item_data2["item"]["name"]
        address2 = item_data2.get('item', {}).get('address', {})
        locality2 = address2.get('addressLocality', '')
        region2 = address2.get('addressRegion', '')
        details2 = (locality2, region2, numberOfBedrooms2, numberOfBathroomsTotal2, typehouse2, amenities2)
        results2.append(details2)
        count2 += 1
        if count2 >= 120:
            break
    return results2


houdata2 = housing_data2(scriptdata2)

rent_amounts_all_urls2 = rent_function2(driver, URL2)

def rent_function3(driver, URLs2):
    all_rent_amounts2 = []
    for URL2 in URLs2:
        driver.get(URL2)
        rent_amount_spans2 = driver.find_elements("class name", "priceinfo")
        rent_amounts2 = [span2.text for span2 in rent_amount_spans2]
        all_rent_amounts2.extend(rent_amounts2)
    return all_rent_amounts2
rent_amounts_all_urls2 = rent_function3(driver, URLs2)

#Once again, the houses are much more expensive in sf east bay
def sqftfunc(driver, URLs2):
    all_sqfts2 = []
    for URL2 in URLs2:
        driver.get(URL2)
        sqft2 = driver.find_elements("class name", "post-sqft")
        sqft_counts2 = [size2.text for size2 in sqft2]
        all_sqfts2.extend(sqft_counts2)
    return all_sqfts2
# The houses are consistently larger throughout the sf east bay as compared to sacramento.
def housing_data2(driver, URLs2):
    all_results2 = []
    for URL2 in URLs2:
        driver.get(URL2)
        soup2 = BeautifulSoup(page2.content, "html.parser")
        script2 = soup2.find_all("script")[1].text.strip()
        scriptdata2 = json.loads(script2)
        results2 = []
        count2 = 0
        for item_data2 in scriptdata2.get('itemListElement', []):
            item2 = item_data2.get('item', {})
            numberOfBedrooms2 = item2.get('numberOfBedrooms')
            numberOfBathroomsTotal2 = item2.get('numberOfBathroomsTotal')
            typehouse2 = item_data2["item"]["@type"]
            amenities2 = item_data2["item"]["name"]
            address2 = item_data2.get('item', {}).get('address', {})
            locality2 = address2.get('addressLocality', '')
            region2 = address2.get('addressRegion', '')
            details2 = (locality2, region2, numberOfBedrooms2, numberOfBathroomsTotal2, typehouse2, amenities2)
            results2.append(details2)
            count2 += 1
            if count2 >= 120:
                break
        all_results2.extend(results2)
    return all_results2



sqft_counts_all_urls2 = sqftfunc(driver, URLs2)
housing_data_all_urls2 = housing_data2(driver, URLs2)


max_length2 = max(len(rent_amounts_all_urls2), len(sqft_counts_all_urls2), len(housing_data_all_urls2))
sqft_counts_all_urls2 += [None] * (max_length2 - len(sqft_counts_all_urls2))
```

```python
housing_data_all_urls2 += [None] * (max_length2 - len(housing_data_all_urls2))
data2 = {
    'Rent Amount': rent_amounts_all_urls2,
    'Square Footage': sqft_counts_all_urls2,
    'Locality': [item2[0] if item2 else None for item2 in housing_data_all_urls2],
    'Region': [item2[1] if item2 else None for item2 in housing_data_all_urls2],
    'Number of Bedrooms': [item2[2] if item2 else None for item2 in housing_data_all_urls2],
    'Number of Bathrooms': [item2[3] if item2 else None for item2 in housing_data_all_urls2],
    'Type of House': [item2[4] if item2 else None for item2 in housing_data_all_urls2],
    'Amenities': [item2[5] if item2 else None for item2 in housing_data_all_urls2]
}

df2 = pd.DataFrame(data2)
df2.to_csv("sfbay.csv", index = False)
# I saved the Excel file separately for sfbay.
```