ECMM443 - Introduction to Data Science
Assessed Coursework 2

670033412

Loading and Processing Data

To load the data, I have created a function which unzips each zip file and reads each json file before moving, importing each file into a dictionary with the key being the file name and the values being a dictionary which contains the fields and information, this dictionary is returned after calling the function.

Here is the function

```python
#Creating function to unzip files and import .json files to dict
def load_data(JobStep=1, Files_to_Unzip=0, Dir='JobAds'):
    #Creating List of files in Dir **ALL files MUST be .zip**
    files = os.listdir('%s' % (Dir))
    #Default number of files to unzip is ALL
    if (Files_to_Unzip == 0):
        Files_to_Unzip = len(files)
    #Creating Dictonary to hold data
    dictonary = {}
    #Fields to be removed
    removed_fields = ['jobId', 'applicationCount', 'contractType',
                      'expirationDate', 'externalUrl', 'jobUrl',
                      'maximumSalary', 'minimumSalary', 'salary']
    #Unzipping and reading json files
    for zip_file in files[:Files_to_Unzip]:
        with zipfile.ZipFile("./JobAds/%s" % (zip_file)) as zip_file_obj:
            filelist = sorted(zip_file_obj.namelist())[1::JobStep]
            for filename in filelist:
                with zip_file_obj.open(filename) as zipped_file:
                    unzipped_file = zipped_file.read()
                    data = json.loads(unzipped_file.decode("utf-8"))
                    if not data['jobId'] == 0:
                        for i in removed_fields:
                            del data[i]
                        dictonary[filename[4:]] = data
    return dictonary
```

As we can see the function takes three inputs:

1. JobStep - Which specifies what step size the function takes when iterating through the json files within the zip files.
2. Files_to_unzip - Which tells the function how many zip files to iterate through, defualt is all found files.
3. Dir - Which is the path to the folder containing the zip files.

The function will only import json files which jobId not equal to zero, as I found the json files corresponding have all empty values in their fields. It also removed the fields I deemed unnecessary for the following analysis.

To then process the data, I iterated through the returned dictionary creating subsets of the data which could then be used for my analysis. When running the code I created all the required data sets within one loop to prevent iterating through the data multiple times.

1 Section 1

Using the following to load the data which is then used to answer the first section.

```
data = load_data(1)
```

1.1 Question 1.1

Simply calculating the length of the data dictionary tells us how many ads there were in our data. Without the empty json files removed, this would be 2,000,007.

```
print("There are %d job adverts in the files provided" % len(data))
```

Which returns:

```
There are 1824675 job adverts in the files provided
```

1.2 Question 1.2

To count the number of ads for full-time and part-time jobs we can check the boolean fields, 'fullTimeJob' and 'partTimeJob' within our processing loop. We can then print the results

```
for i in data:
    if data[i]['fullTime'] and data[i]['partTime']:
        Both_time_jobs[i] = data[i]
        continue
    elif data[i]['fullTime']:
        Full_time_jobs[i] = data[i]
        continue
    elif data[i]['partTime']:
        Part_time_jobs[i] = data[i]
        continue
    else:
        Neither_time_jobs[i] = data[i]

Num_both_time_jobs = len(Both_time_jobs)
Num_full_time_jobs = len(Full_time_jobs)
Num_part_time_jobs = len(Part_time_jobs)

print("There are %d job adverts for full-time jobs, %d for part-time jobs and"
      " %d job adverts which advertise for both full-time and part-time jobs\n"
      % (Num_both_time_jobs+Num_full_time_jobs,
         Num_both_time_jobs+Num_part_time_jobs,
         Num_both_time_jobs))
```

Which returns:

```
There are 1719322 job adverts for full-time jobs, 227636 for part-time jobs and 122283 job
↪ adverts which advertise for both full-time and part-time jobs
```

1.3 Question 1.3

To create a time-series plot of the ads posted on each day we create a dictionary which takes a date as the key and then when processing the data we add a count each time that date is found in the 'datePosted' field. We can then split this dictionary into 2 vectors and post them aganst each other, creating the time-series.

```
for i in data:
    Day_freq[data[i]['datePosted']] = Day_freq.get(
        data[i]['datePosted'], 0) + 1

Ad_dates, Ad_freq = zip(*Day_freq.items())
Ad_dates = [datetime.strptime(i, '%d/%m/%Y') for i in Ad_dates]

fig, ax1 = plt.subplots()
ax1.plot(Ad_dates, [i/1000 for i in Ad_freq])

ax1.set_title('Time-series of Number of Adverts Posted Per Day')
ax1.set_xlabel('Date')
ax1.set_ylabel('Number Posted (thousands)')

fig.autofmt_xdate()

plt.tight_layout()
plt.show()
```

Which produces the following plot:

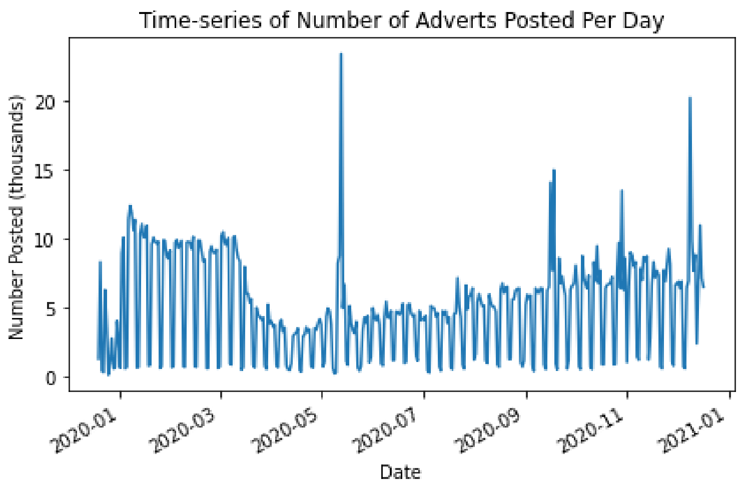

Figure 1: A time-series plot showing the number of ads posted each day within our data set.

We can see that there is some periodic pattern every 7 days, this is likely due to the nature of less work being on the weekend. We can also see a steep decline in the number of ads posted from near the end of March, this is when the UK went into a nation-wide lock-down due to the coronavirus pandemic which saw many industries across the country coming to a halt. We then see a large increase in May (13/05/2021), this value is extremely high so may need further investigating, a quick look into the day before after reveal 8771 ads before and 4989 the day after, further showing the abnormality of this high number. From May there is a slow trend up with 2 more spikes till November when another lock-down was announced. We then saw a big uptake at the end of the year, possibly due to hires for the new year. This is also when the UK began rolling out it's vaccine program so this could also be a contributing factor.

1.4 Question 1.4

To plot the distribution of advertised salaries I added several booleans into the data processing section which checked for both a maximum and minimum salary and ensured there was a corresponding currency, if this currency was not GBP the value was converted using an online dictionary which is updated daily. This unfortunately means the converted values will be different from when the listings where created, but should provide a good estimate suitable for our use. The mean value was then calculated and saved into a list which was then used to plot a histogram. I decided to plot both a histogram of all the data and a histogram which contained all the data within 3 standard deviations of the mean.

```python
CurrencyRates = requests.get('http://www.floatrates.com/daily/gbp.json').json()
CurrencyRates = {k.upper(): v for k, v in CurrencyRates.items()}

for i in data:
    #List of Advertised Annual Salary
    if data[i]['currency'] is not None and not (
            data[i]['yearlyMaximumSalary'] == 0 and
            data[i]['yearlyMinimumSalary'] == 0):
        Mean_yearly_salary = (data[i]['yearlyMaximumSalary']
                              + data[i]['yearlyMinimumSalary'])/2
        if not data[i]['currency'] == 'GBP':
            Mean_yearly_salary = CurrencyRates[
                data[i]['currency']]['inverseRate']*Mean_yearly_salary
        Mean_salaries.append(round(Mean_yearly_salary, 2))

Mean_salaries = np.array(Mean_salaries)
threshold = 3
fig, (ax1, ax2) = plt.subplots(nrows=2)


ax1.hist(Mean_salaries)
ax1.set_title('All adverts')
ax1.xaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('£{x:,.0f}'))
ax1.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))

ax2.hist(Mean_salaries[np.where(np.abs(stats.zscore(Mean_salaries)) < threshold)])
ax2.set_title('Adverts with values %.2f std from the mean removed' % threshold)
ax2.set_xlabel('Annual Salary')
ax2.xaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('£{x:,.0f}'))
ax2.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))

fig.suptitle('Histogram of Annual Salaries (as provided by job adverts)')
plt.tight_layout()
plt.show()
```

This produced the following plot:

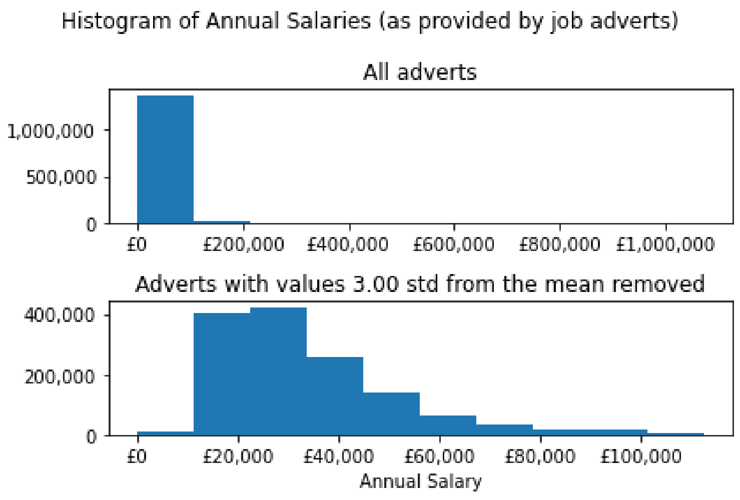

Figure 2: Two histograms showing the distribution of advertised salaries. First histogram contains all eligible data, the second histogram is from the same set of data but with values that are 3 standard deviations from the mean removed.

1.5 Question 1.5

Using the two vectors, Ad_dates and Ad_freq created in subsection 1.3, I was able to assign each of the dates to their corresponding day using functions from the datetime package. From here I created a dictionary with the keys being days of the week and the field being a list of tuples which include date and number of the ads posted on that day. From here I was able to unzip the dictionary and create a list of list which was then used to create the box and whisker plot. The motivation behind creating the tuples was to allow me to investigate outliers which were highlighted in the plot/

```python
weekDays = ("Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday", "Sunday")
Weekday_avg = {i: [] for i in weekDays}

for i in range(len(Ad_dates)):
    Weekday_avg[weekDays[Ad_dates[i].weekday()]].append(
        (Ad_dates[i], Ad_freq[i]/1000))

labels, Box_data = [*zip(*Weekday_avg.items())]
fig, ax1 = plt.subplots()

Box_data = [[i[1] for i in j] for j in Box_data]

ax1.boxplot(Box_data)
ax1.set_xticklabels(labels)
ax1.set_ylabel('Number of Ads Posted (thousands)')
ax1.set_title('Box plot showing distributions of the number adverts'
              'posted on each weekday')

plt.tight_layout()
plt.show()
```

This produces the following plot:

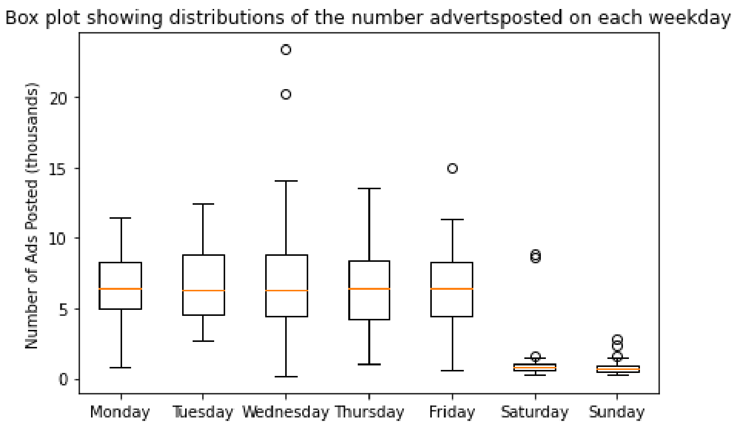

Figure 3: A box and whisker plot showing the distribution of ads posted on each day of the week.

The distribution of ads is fairly similar for the weekdays with a median of 6000 ads posted and then very low number of ads posted on the weekend. Wednesday has the largest range of all the days and also the high number, with two extremely high outliers which correspond to the 13/05/2020 and 09/12/2020. Other interesting outliers include 31/10/2020 and 12/12/2020 both Saturdays but saw number of ads posted above the weekday upper-quartiles.

2 Section 2

For the next section I continued in a similar fashion as section 1, loading the data as a dictionary and then processing the data in a for loop.

2.1 Question 2.1

To find the employers with the highest number of posted ads I added a dicitonary which takes a tuple of the employer ID and employer name and ads a count for each ad found. From this dictionary I then created a sorted list ordered by the number of ads posted and printed the top 5 as a pandas dataframe.

```python
for i in data:
    #Dictionary counting job ads by each employer
    Employer_freq[(data[i]['employerId'], data[i]['employerName'])] = Employer_freq.get(
        (data[i]['employerId'], data[i]['employerName']), 0) + 1

[list({k : v for k, v in sorted(Employer_freq.items(),
        key = lambda item: item[1])})[i][1] for i in range(5)]

print("\nThe 5 employers with the most ads posted are:")
print(pd.DataFrame.from_dict(Employer_freq, orient='index',
                             columns = ['Frequency']).nlargest(5, 'Frequency'))
```

Which returns the following:

```
The 5 employers with the most ads posted are:
                                               Frequency
(524744, Hays Specialist Recruitment Limited)      67992
(575264, REED)                                     67659
(624561, NHS Business Services Authority)          61877
(626228, Partnership Jobs)                         50947
(486707, Healthcare Homes)                         27614
```

We can see that three of the top 5 are recruitment agencies, which explains there high volume of job ads. The other two, slightly more interesting are both health care related this high number of job ads could be directly related to the coronavirus pandemic however, these high numbers may be normal from pre-pandemic times so would require additional data.

2.2 Question 2.2

To find the most common bigrams (two adjancent words) in job advert titles I again added a step in the data processing loop. This step included separating the job titles into a list of individual words with the use of split(), I also had to remove non-alpha characters and white spaces errors such as double spacing. This ensured that the bi-gram list produced meaning insights into the data. I then used dictionary to count the frequency of each bi-gram found. This dictionary was then ordered and printed as a pandas data frame.

```python
for i in data:
    #Dictionary of Bigrams
    Job_title_words = data[i]['jobTitle'].replace('-', '').strip().split()
    for word1, word2 in zip(Job_title_words[:-1], Job_title_words[1:]):
        Bigrams_freq[(word1, word2)] = Bigrams_freq.get(
            (word1, word2), 0) + 1

print("\nThe 5 most common bi-grams from the job titles are:")
print(pd.DataFrame.from_dict(Bigrams_freq, orient='index',
                             columns = ['Frequency']).nlargest(5, 'Frequency'))
```

which returned the following:

```
The 5 most common bi-grams from the job titles are:
                       Frequency
```

```
(Care, Assistant)           59388
(Support, Worker)           57970
(Teaching, Assistant)       30618
(Registered, Nurse)         29178
(Customer, Service)         22937
```

2.3 Question 2.3

For this question I decided to create a time-series of job adverts for nurse roles. My method was very similar to how I created the job ads time-series however, this time I had a Boolean which checked for the existence of ’nurse’ in the title. I had to be cautious here as there were several jobs which were for nurseries, but I could not simply remove all jobs which included ’nursery’ as there were some ads which were for ’nursery nurse’. To get around this I removed ’nursery’ and ’non-nurse’ and checked for the existence of nurse still in the title.

```python
for i in data:
    #Dictionary counting Nurse ads on days
    if 'nurse' in data[i]['jobTitle'].lower().replace('nursery',''
                                            ).replace('non nurse',''
                                            ).replace('non-nurse',''):
        Nurse_descr.append(BeautifulSoup(data[i]['jobDescription']).get_text())
        Nurse_ads[data[i]['datePosted']] = Nurse_ads.get(
            data[i]['datePosted'], 0) + 1

Nurse_dates, Nurse_freq = zip(*Nurse_ads.items())
Nurse_dates = [datetime.strptime(i, '%d/%m/%Y') for i in Nurse_dates]

fig, ax1 = plt.subplots()
ax1.plot(Nurse_dates, Nurse_freq)

ax1.set_title('Time-series of Number of Nurse Adverts Posted Per Day')
ax1.set_xlabel('Date')
ax1.set_ylabel('Number Posted')

fig.autofmt_xdate()

plt.tight_layout()
plt.show()
```

Which produced the following plot:

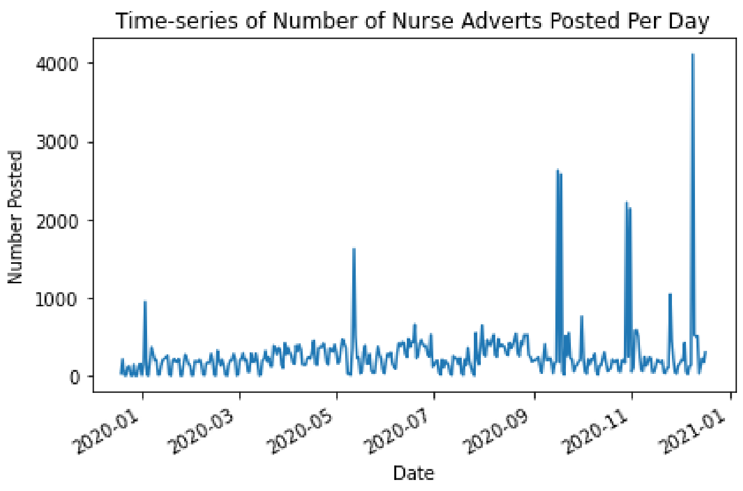

Figure 4: A time-series plot of nurse job ads posted per day.

In Figure 4 we can again see the periodic nature of the job ads posted as was seen in Figure 1, more interestingly we can also see that the upticks align strongly with the peaks. It would be interesting to try find other job titles or perhaps sectors which also saw these peaks to help explain the cause of them. It is worth noting that the high number of nurse jobs does not completely satisfy the increase in the broader job market, so it appears multiple sectors saw job ad increases or alternatively perhaps different roles within the healthcare sector.

2.4 Question 2.4

To create a word cloud from the nurse job ads I simply appended the job description to a list after cleaning it from html to plain text using a function 'BeautifulSoup' from the bs4 package. This is shown in the processing step in line 6 of subsection 2.3. I then created a long string containing all of the descriptions separated by a space and used the 'WordCloud' function to create a word cloud.

```
Word_string = (" ").join(Nurse_descr)
wordcloud = WordCloud(width = 1000, height = 500).generate(Word_string)
plt.figure(figsize=(15,8))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

This procudes the following plot:

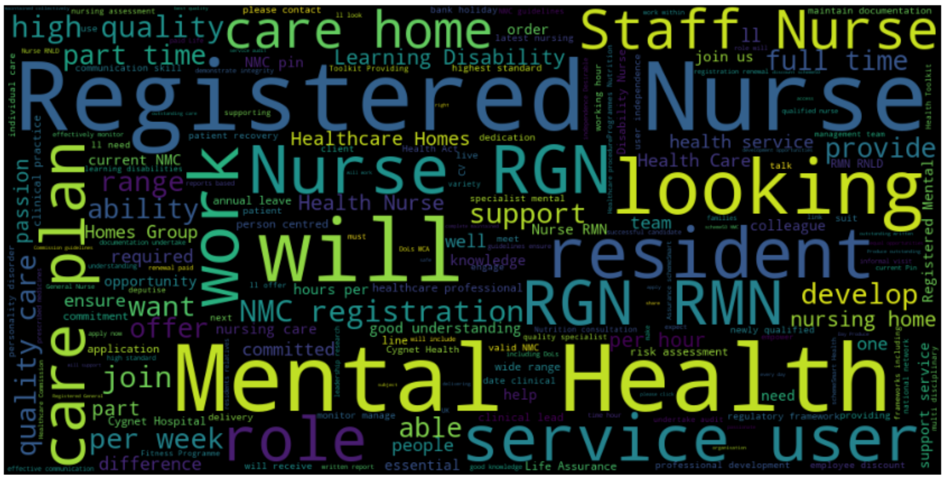

Figure 5: A word cloud created from the description of nurse job ads, in which the larger the word appears the more frequent it appears in the job ad descriptions.

This word cloud gives us an insight into the type of nurse companies are looking for such as, care home nurses, mental health nurses, registered nurses, disability nurse to name a few. We also see part-time and full-time being similarly sized perhaps meaning they are in equal demand. We also see desirable traits that companies eek in nurses such as passion, good understanding, person centered, committed and demonstrate integrity among others.

3 Section 3

3.1 Question 3.1

This section starts of similar by processing the data and saving the location field of every ad within our data set. I then proceed to clean this data by attempting to take the location value and assigning it into one of the districts included in the boundaries file provided in the coursework breif. To achieve this I use several Booleans and the help of a csv file which includes several town names and there corresponding district (https://www.paulstenning.com/uk-towns-and-counties-list/) I then create a dictionary which takes the districts from the boundaries file and adds a count to each appearance of that place in a job ad with the use of a data map. This frequency column is then attached to a GeoPandas data frame in which the spatial data map is created.

```python
town_to_county = {}
town_list = []
county_list = []

#Loading list of towns with matching counties
with open('Towns_List.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        if row['Country'] == 'England' or row['Country'] == 'Wales':
            town_list.append(row['Town'])
            town_to_county[row['Town']] = row['County']
            if not row['County'] in county_list:
                    county_list.append(row['County'])

#Loading Ad Data
data = load_data(20,100)

#Loading Geo Data
path = "MapBoundaries/Counties_and_Unitary_Authorities_(December_2016)_Boundaries.geojson"
gdf = geopandas.read_file(path).rename(columns = {'ctyua16nm' : 'Town'})
region_list = gdf['Town'].values.tolist()

#Processing Ad Data
county_freq = {i : 0 for i in region_list}
location_list = []
for i in data:
    if data[i]['locationName'].title() in region_list:
        county_freq[data[i]['locationName'].title()] = county_freq.get(data[i]['locationName'], 0) + 1
    County = town_to_county.get(data[i]['locationName'].title(), 0)
    if County in region_list:
        county_freq[County] = county_freq.get(County, 0) + 1
    if not data[i]['locationName'] in location_list:
        location_list.append(data[i]['locationName'])

#Creating Data Frame from Ads
df = pd.DataFrame.from_dict(county_freq, orient = 'index').reset_index(
    ).rename(columns = {'index' : 'Town', 0 : 'Frequency'})

#Adding Ad Data Frame to Geodataframe
merged = gdf.merge(df, on = 'Town')
merged.plot(column='Frequency', legend = True, figsize=(15,10))

#Print ratio of ads located
num_ads_located = sum([ i for i in county_freq.values()])
num_of_ads = len(data)
print("\nThe number of ads located was %d out of %d" %
      (num_ads_located, num_of_ads))
```

Which returns and produces the following spatial map:

```
The number of ads located was 1157644 out of 1824675
```

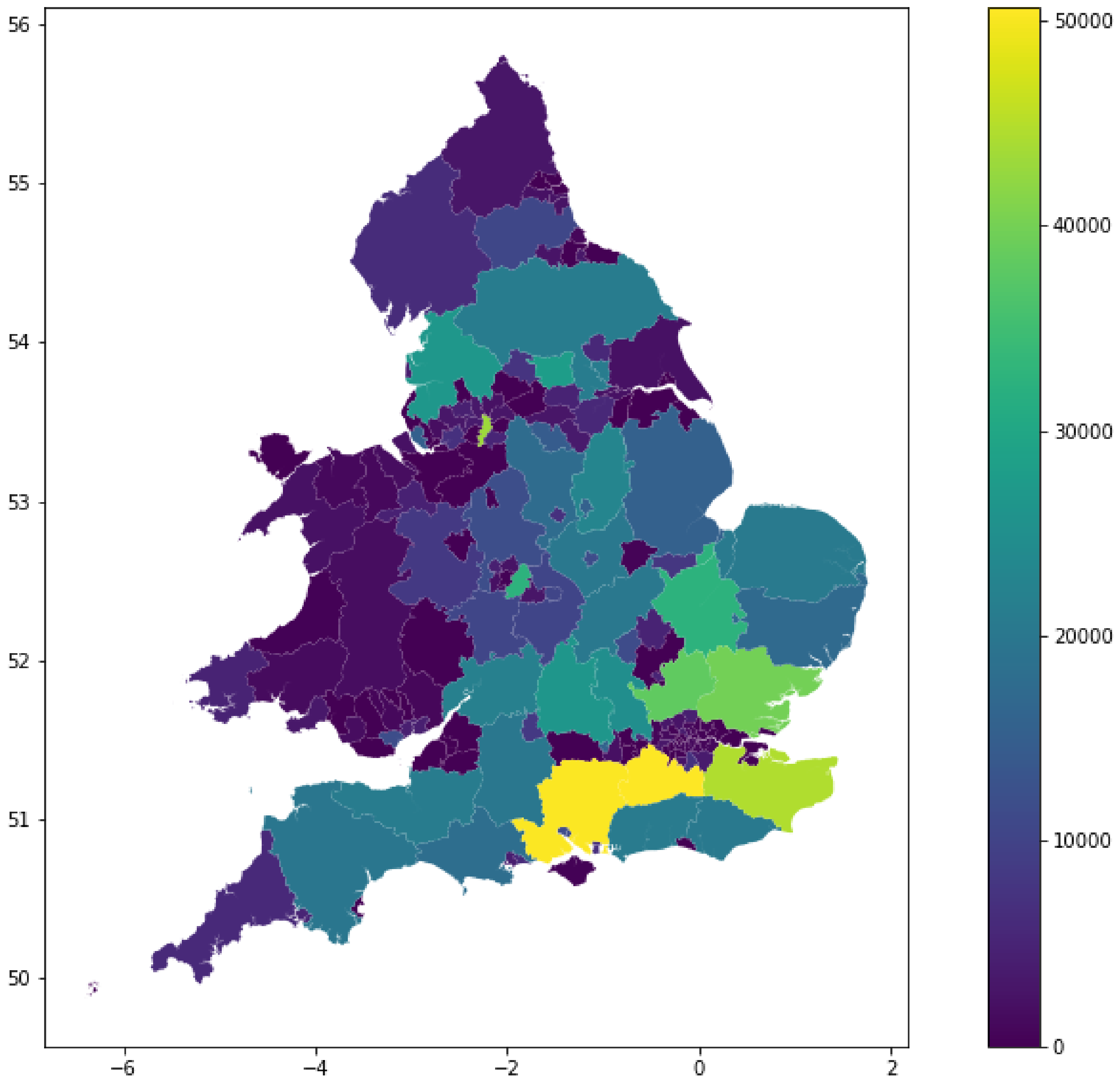

Figure 6: Spatial map of the UK with key representing the number of ads posted in that region within the data set.

3.2 Question 3.2

From the map we can see that there are hot spots around London. It is worth noting that there should be more ads in the London region, however due to the boundaries having London separated into it boroughs it was difficult to find a data map which recognised these. I had a similar issue in other places in the UK, where I would expect that there would be a higher number of ads than is produced by my plot. We can see that there are over 700,000 ads which were not located. An alternative strategy could be to use a geocode function which relies on an API to return the longitude and latitude of the address in the 'locationPosted' field, however this requires paying for a service and due to the large file size can become very costly. To reduce the cost we could instead take a subset of the data, for instance load the data with `data = load_data(20,20)` which would take the location of every 20th job ad, giving us a total data set of around 100,000 job ads, which would be large enough for the law of large numbers to apply.