Problem 3 ¶

In [37]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings
from wordcloud import WordCloud
warnings.simplefilter('ignore')
```

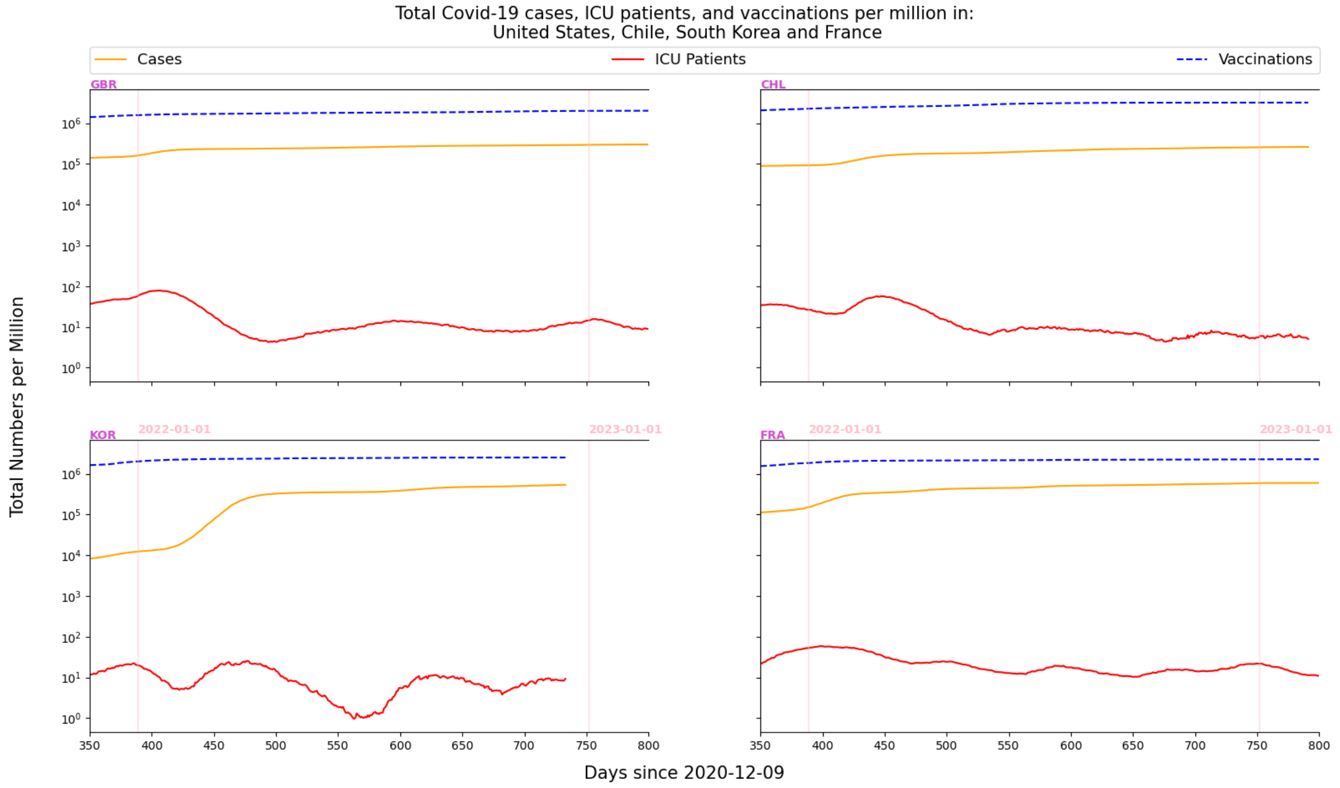
Part 3.1A

In [38]:
```
df = pd.read_csv('/Users/elliottoates/Library/CloudStorage/OneDrive-UniversityofE

#only relevant collumns
df = df.loc[:, ['iso_code', 'date', 'total_cases_per_million', 'total_vaccination

#convert vax into per mill
df['total_vaccinations_per_million'] = df['total_vaccinations_per_hundred'] * 100
df.drop('total_vaccinations_per_hundred', axis=1, inplace=True)

# date column to a datetime object
df['date'] = pd.to_datetime(df['date'])

# only the four chosen countries and rows for 350 days or more since 2020-12-09
countries = ['USA', 'CHL', 'KOR', 'FRA']
start_date = pd.to_datetime('2020-12-09')
df = df[df['iso_code'].isin(countries) & (df['date'] >= start_date + pd.Timedelta

# collumn for number of days since 2020-12-09 for x axis
start_date = pd.to_datetime('2020-12-09')
df['days_since'] = (df['date'] - start_date).dt.days

# Drop old date column for new "days_since" column
df.drop('date', axis=1, inplace=True)
df.insert(1, 'days_since', df.pop('days_since'))
```

In [39]:
```
#USA
df_USA = df[df['iso_code'].str.contains('USA')]
df_USA.dropna(how='any', inplace=True)
df_USA = df_USA.reset_index(drop=True)
#CHL
df_CHL = df[df['iso_code'].str.contains('CHL')]
df_CHL.dropna(how='any', inplace=True)
df_CHL = df_CHL.reset_index(drop=True)

#KOR
df_KOR = df[df['iso_code'].str.contains('KOR')]
df_KOR.dropna(how='any', inplace=True)
df_KOR= df_KOR.reset_index(drop=True)

#FRA
df_FRA = df[df['iso_code'].str.contains('FRA')]
df_FRA.dropna(how='any', inplace=True)
df_FRA = df_FRA.reset_index(drop=True)
```

In [40]:

```
#figure
fig, axes = plt.subplots(nrows=2, ncols=2,figsize=(19,10),sharex = True,sharey=Tr

#axes objects
axes = axes.ravel().tolist()

plt.suptitle('Total Covid-19 cases, ICU patients, and vaccinations per million in
fig.supxlabel('Days since 2020-12-09',size = 15, y = 0.05)
fig.supylabel('Total Numbers per Million',size = 15, x = 0.075)

#DATA
x = 'days_since'
y1 ='total_cases_per_million'
y2 = 'icu_patients_per_million'
y3 = 'total_vaccinations_per_million'

data = [df_USA,df_CHL,df_KOR,df_FRA]

#subplot - axes
for z,i in enumerate (data) :
    i.plot(x=x,y =[y1,y2] ,ax=axes[z],legend =False,logy=True,color = ['orange','
    i.plot(x=x,y =y3 ,ax=axes[z],legend =False,logy=True,color = 'blue',linestyle
    axes[z].set_xlim(left=350,right =800)
    axes[z].set(xlabel=None)
    axes[z].axvline(x=389,ymin = 0, ymax = 1,color ='pink',alpha = 0.4 )
    axes[z].axvline(x=752,ymin = 0, ymax = 1,color ='pink',alpha = 0.4 )
    axes[z].spines["right"].set_visible(False)
    axes[z].tick_params(axis='y', which='minor', left=False)
    axes[z].grid(False)
#pink lines

axes[0].text(x=350,y=7000000,s = 'GBR', color = 'm',weight='demi',alpha = 0.7)
axes[1].text(x=350,y=7000000,s = 'CHL', color = 'm',weight='demi',alpha = 0.7)
axes[2].text(x=350,y=7000000,s = 'KOR', color = 'm',weight='demi',alpha = 0.7)
axes[3].text(x=350,y=7000000,s = 'FRA', color = 'm',weight='demi',alpha = 0.7)

axes[2].text(x=389,y=10000000,s = '2022-01-01', color = 'pink',weight='demi',alph
axes[2].text(x=752,y=10000000,s = '2023-01-01', color = 'pink',weight='demi',alph
axes[3].text(x=389,y=10000000,s = '2022-01-01', color = 'pink',weight='demi',alph
axes[3].text(x=752,y=10000000,s = '2023-01-01', color = 'pink',weight='demi',alph

#legend
axes[0].plot([], [], color='orange', label='Cases')
axes[0].plot([], [], color='red', label='ICU Patients')
axes[0].plot([], [], color='blue', linestyle='--', label='Vaccinations')
handles, labels = axes[0].get_legend_handles_labels()
fig.legend(handles, labels [3:], loc= 'upper center', ncol=3, mode = 'expand', bb
```



Part 3.1B

In [41]:

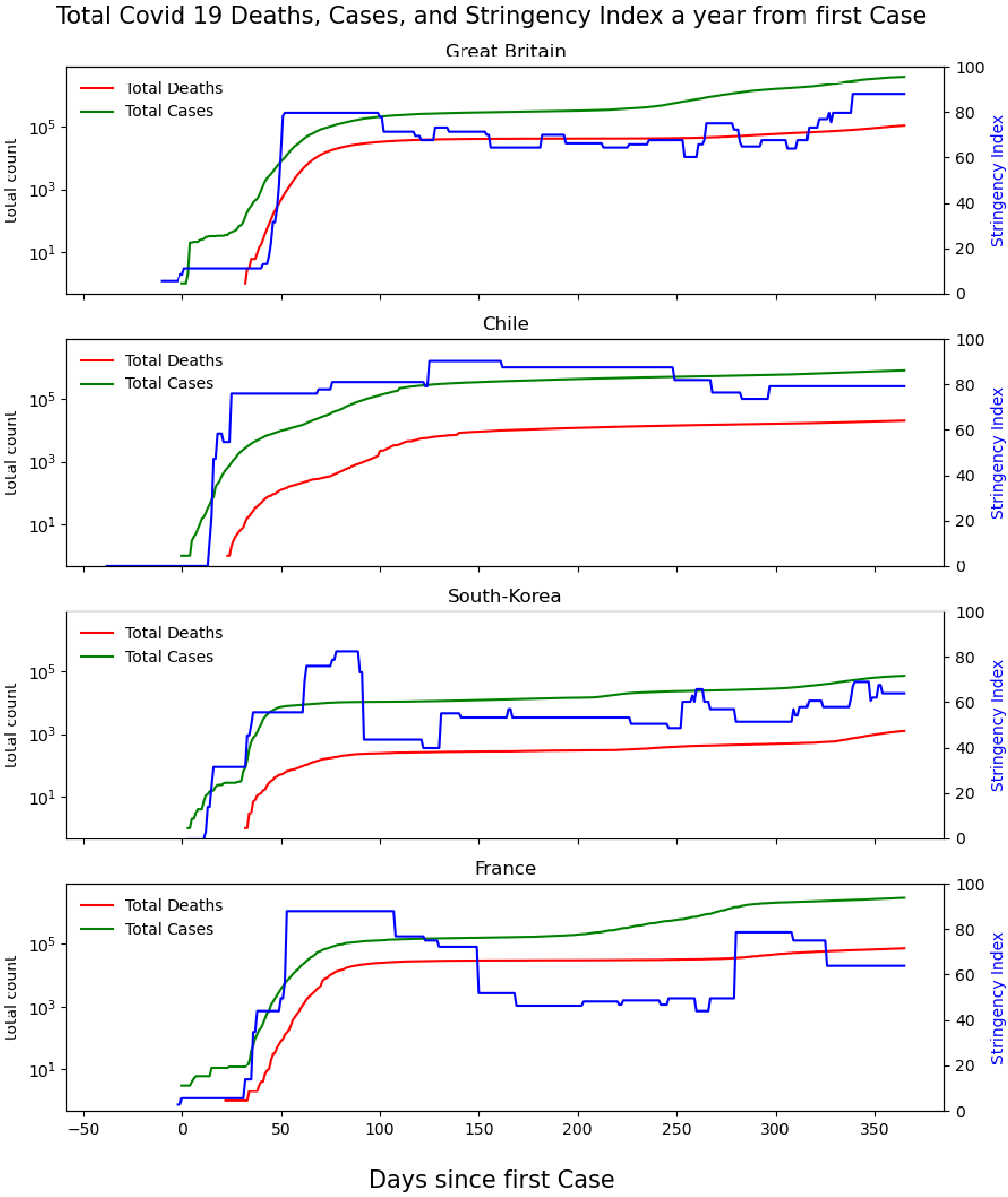
```python
df = pd.read_csv(
    '/Users/elliottoates/Library/CloudStorage/OneDrive-UniversityofExeter/Second
```

In [42]:

```python
def make_dfs(x):
    # Filter data for the given ISO code and calculate days since first vaccinati
    y = df[df['iso_code'].str.contains(x)]
    y['date'] = y['date'].apply(pd.to_datetime)
    y['Days Since 1st Case'] = (y['date'] - (y.loc[df['total_cases'] >= 1, 'date'

    # Filter data to only include dates within a certain range and select relevan
    y = y[y['Days Since 1st Case'] <= 365]
    y = y[y['Days Since 1st Case'] >= -365]
    y = y[['iso_code', 'total_deaths', 'total_cases', 'stringency_index', 'Days S

    # Remove any rows with missing data in the stringency_index column and reset
    y = y[y['stringency_index'].notna()]
    y = y.reset_index(drop=True)

    return y
```

In [43]:

```
# Create a figure with 4 subplots, each representing a different country
fig, (ax, ax1, ax2, ax3) = plt.subplots(4, 1, figsize=(10, 12), sharex=True, shar
plt.suptitle('Total Covid 19 Deaths, Cases, and Stringency Index a year from firs
             weight='ultralight', y=0.925)
fig.supxlabel('Days since first Case', size=15, y=0.05)

# List of the four subplots
axes = [ax, ax1, ax2, ax3]

# Add plot titles for each subplot
ax.set(title='Great Britain')
ax1.set(title='Chile')
ax2.set(title='South-Korea')
ax3.set(title='France')

isocodes = ['GBR','CHL','KOR','FRA']

# Remove gridlines from all subplots
ax.grid(False)
ax1.grid(False)
ax2.grid(False)
ax3.grid(False)

# Loop through each ISO code and corresponding subplot
for i, z in zip(isocodes, axes):
    axa = z.twinx()

    # Create a DataFrame for the current ISO code and extract relevant columns
    x = make_dfs(i)['Days Since 1st Case']
    y1 = make_dfs(i)['total_deaths']
    y2 = make_dfs(i)['stringency_index']
    y3 = make_dfs(i)['total_cases']

    # Plot total deaths and total vaccinations on the left y-axis
    deaths_line, = z.plot(x, y1, color='r', label='Total Deaths')
    cases_line, = z.plot(x, y3, color='g', label='Total Cases')

    # Plot stringency index on the first right y-axis
    axa.plot(x, y2, color='b')

    # Add y-axis labels and set scales for each axis
    z.set_ylabel('total count', color='black')
    z.set(yscale='log')

    axa.set_ylabel('Stringency Index', color='b')
    axa.set_ylim(0, 100)

    # Customize the ticks
    axa.grid(False)

    # Create a custom legend without Stringency Index and with no border
    legend_lines = [deaths_line, cases_line]
    legend_labels = [line.get_label() for line in legend_lines]
    z.legend(legend_lines, legend_labels, loc='upper left', frameon=False)

plt.show()
```



The figure above allows us to visualise govornments responses at the start of the pandemic. One can observe that countries all tended to raise the stringency index to a small degree between the recording of the first cases and the first death. Then once faced with a positive total death count, governments (apart from the United Kingdom) more drastically increased the "stringency" of their response as they came to the stark realisation of the gravity of the oncoming pandemic.

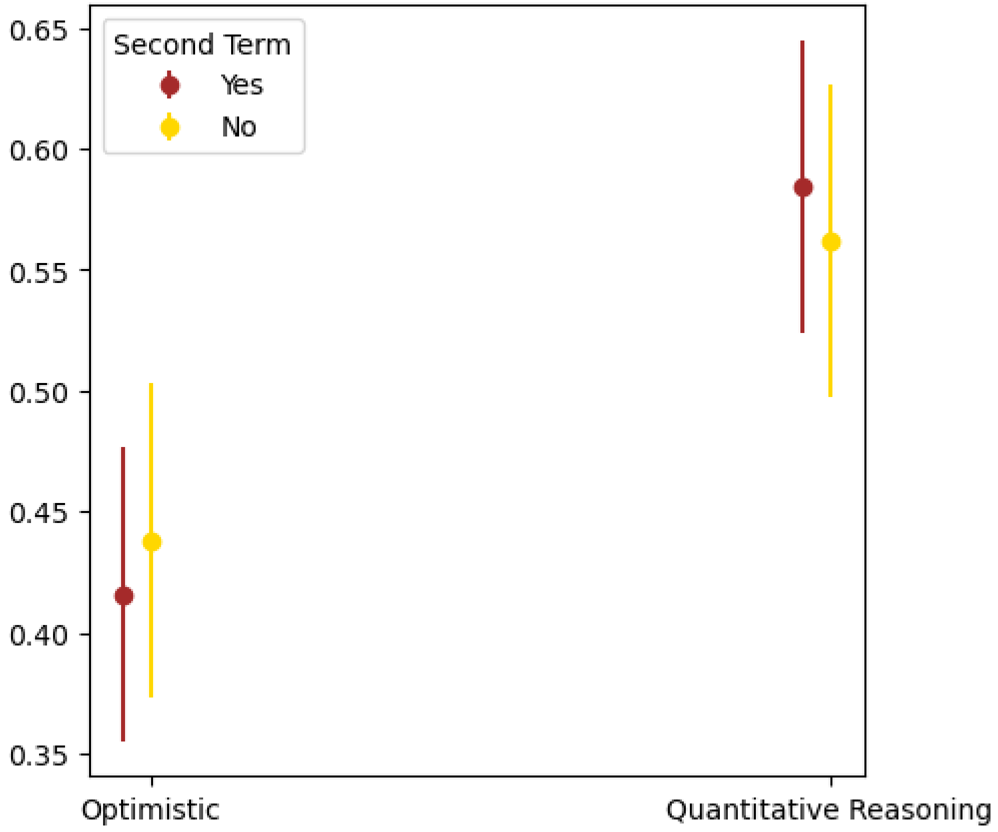
The figure also is usefull in depicting the relationship between total cases, total deaths and stringency index. In the case of france between days 150 and 275 the strinengcy index was lowered as the govornment relaxed their response. Then after they saw a rise in cases from this they once againe raised their response in order to limit this raise in deaths.

Part 3.2A

```
In [44]: df_LLM = pd.read_csv('/Users/elliottoates/Library/CloudStorage/OneDrive-Universit
```

In [45]:

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

df_LLM = df_LLM[(df_LLM['Inaugural Address'] == 'First Inaugural Address') | (df_

# Prepare data for plotting
df_LLM['Second Term'] = df_LLM['Inaugural Address'].apply(lambda x: 'Yes' if x ==
df_LLM = df_LLM.melt(id_vars=['Second Term'], value_vars=['Optimistic', 'QuantLLM

# Calculate mean and standard deviation
mean_values = df_LLM.groupby(['Category', 'Second Term']).mean()
std_values = df_LLM.groupby(['Category', 'Second Term']).std()

# Create the point plot
fig, ax = plt.subplots(figsize=(5, 5))

x_labels = ['Optimistic', 'QuantLLMQuant']
x_tick_positions = [0.25, 1.25]

for i, (label, xpos) in enumerate(zip(x_labels, x_tick_positions)):
    y_yes = mean_values.loc[(label, 'Yes'), 'Value']
    y_no = mean_values.loc[(label, 'No'), 'Value']
    std_yes = std_values.loc[(label, 'Yes'), 'Value']
    std_no = std_values.loc[(label, 'No'), 'Value']

    ax.errorbar(xpos - 0.04, y_yes, yerr=std_yes, fmt='o', color='brown', label='
    ax.errorbar(xpos, y_no, yerr=std_no, fmt='o', color='gold', label='No' if i =

ax.set_xticks(x_tick_positions)
ax.set_xticklabels(['Optimistic','Quantitative Reasoning'])
ax.legend(title='Second Term', loc='upper left')
ax.grid(False)
plt.show()
```



Part 3.2B

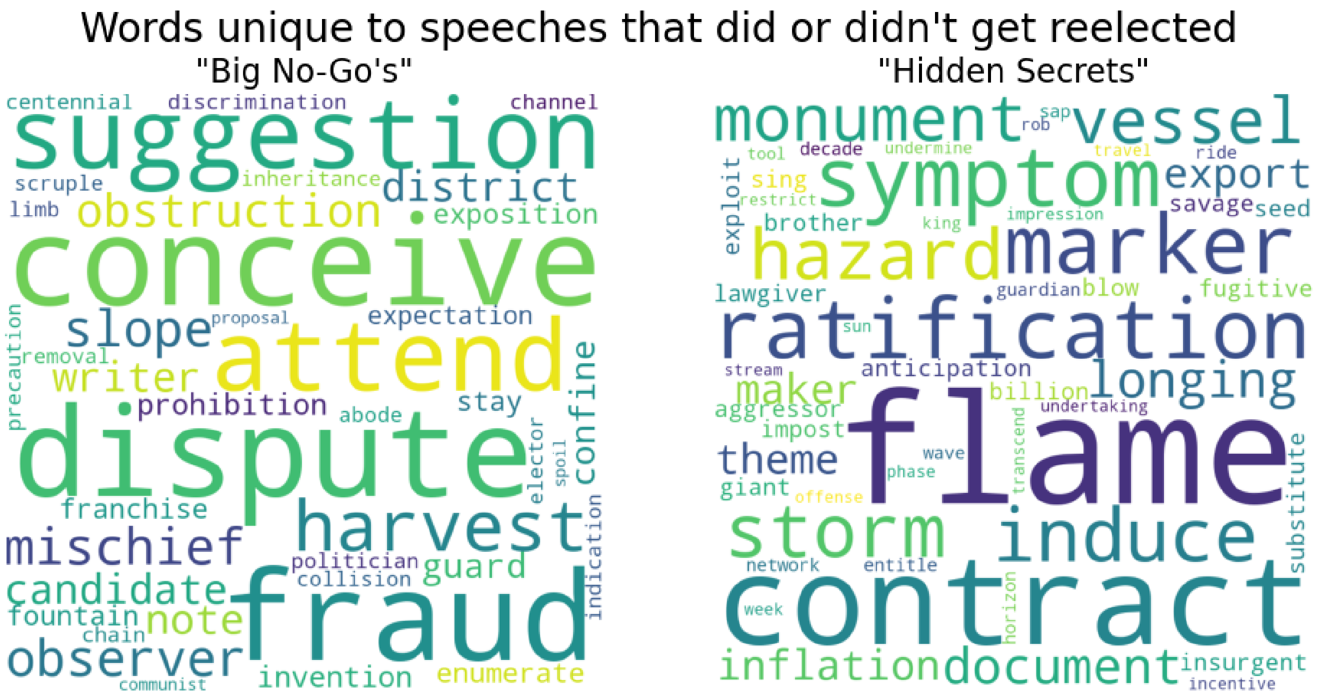
In [46]:

```
df_LLM = pd.read_csv('/Users/elliottoates/Library/CloudStorage/OneDrive-Universit
```

In [47]:

```
#Finding words unique to reelection & no reelection speeches
import string
# convert lists of words into sets (reeleciton)
reelection_speeches = df_LLM[df_LLM['Inaugural Address'] != 'Inaugural Address'][
reelection_words = set()
for speech in reelection_speeches:
    words = speech.split()
    # alphabetical characters only
    words = [word.translate(str.maketrans('', '', string.punctuation)).lower() fo
    reelection_words.update(words)

# list into set for no relection
not_reelection_speeches = df_LLM[df_LLM['Inaugural Address'] == 'Inaugural Addres
not_reelection_words = set()
for speech in not_reelection_speeches:
    words = speech.split()
    # alphabetical
    words = [word.translate(str.maketrans('', '', string.punctuation)).lower() fo
    not_reelection_words.update(words)

#create unique word sets
unique_to_not_reelection = not_reelection_words.difference(reelection_words)
unique_to_reelection = reelection_words.difference(not_reelection_words)
```

In [48]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

wordcloud = WordCloud(width = 800, height = 800,
                      background_color ='white',
                      min_font_size = 20)

# no re-election wordcloud
wordcloud.generate_from_text(' '.join(unique_to_not_reelection))
ax1.imshow(wordcloud)
ax1.axis("off")
ax1.set_title('"Big No-Go\'s"', fontsize=16)

# re-election wordcloud
wordcloud.generate_from_text(' '.join(unique_to_reelection))
ax2.imshow(wordcloud)
ax2.axis("off")
ax2.set_title('"Hidden Secrets"', fontsize=16)

# title
fig.suptitle('Words unique to speeches that did or didn\'t get reelected', fontsi

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



The above plot shows words that are unique to speaches that went on to not get re-election and those that didnt. I think that wordclouds in this case are a brilliant way to convey the information. It highlights particular words that an incoming US president should think twice about using in their speech and those that should be included. However this is still somewhat trivial since these are singular words so in many cases there is not context or sentiment that can be interpreted.

Part 3.3A

In [11]:

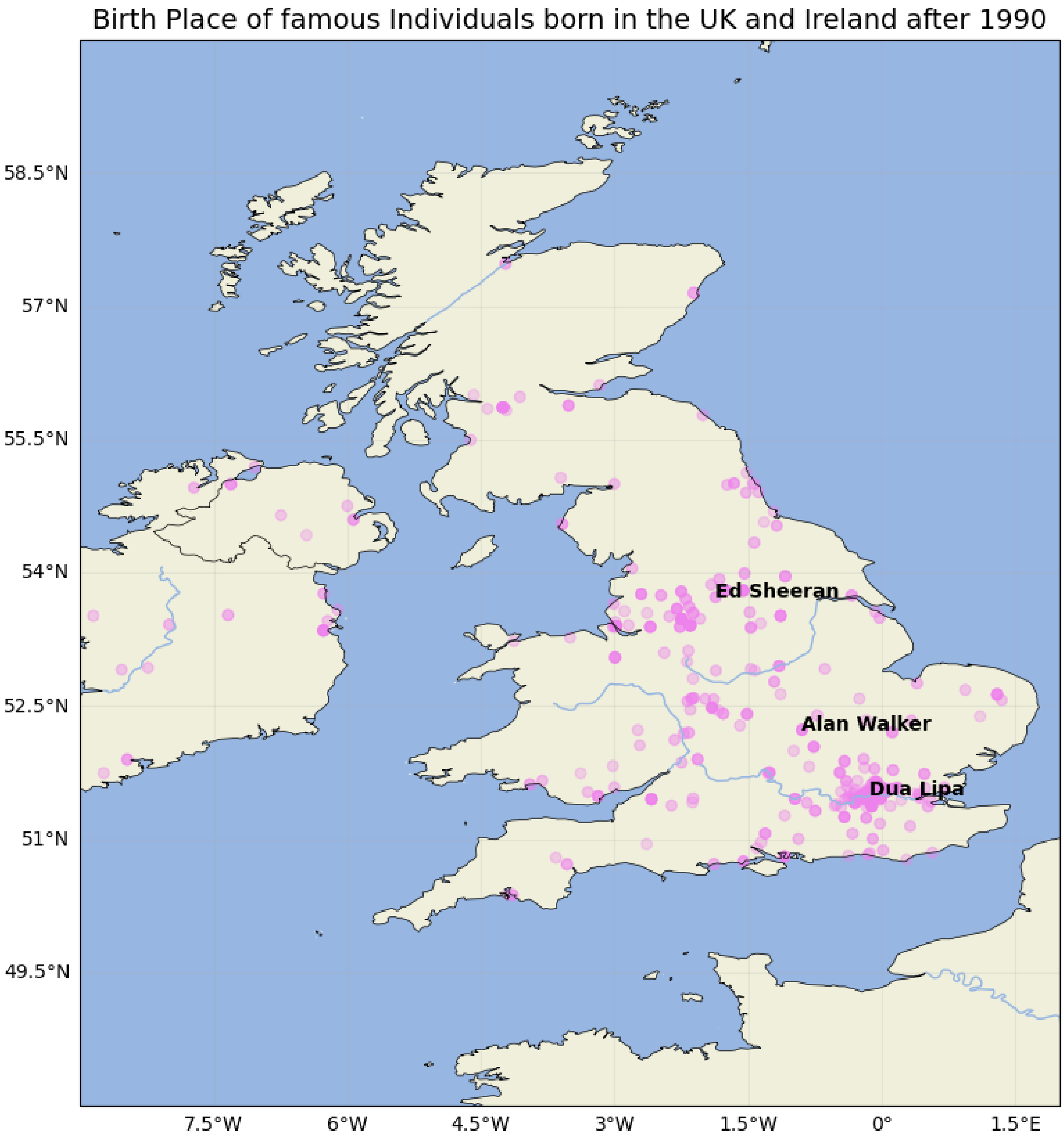
```
import cartopy.crs as crs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
df = pd.read_csv('/Users/elliottoates/Library/CloudStorage/OneDrive-UniversityofE
```

In [12]:

```
#Filtering
#year
df = df[df['birthyear'] > 1990]

#lat and long have values
df = df[(df['bplace_lat'].notnull()) & (df['bplace_lon'].notnull())]

#from uk & ireland
df = df[df['bplace_country'].isin(['United Kingdom', 'Ireland'])]
```

In [13]:

```
figure = plt.figure(figsize=(15,10))

#Draw map and UK and choose features
ax = figure.add_subplot(1,1,1, projection=crs.PlateCarree())
ax.add_feature(cfeature.BORDERS,linewidth=0.5)
ax.add_feature(cfeature.COASTLINE,linewidth=0.5)
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.RIVERS)
ax.set_extent( [-9, 2, 48, 60],crs=crs.PlateCarree())
ax.set_title('Birth Place of famous Individuals born in the UK and Ireland after
#Draw feint gridlines and ticks on axis
coordinates = ax.gridlines(draw_labels=True,alpha=0.2)
coordinates.xlabels_top = False
coordinates.ylabels_right = False

#Plot points on graph
plt.scatter(
    x=df["bplace_lon"],
    y=df["bplace_lat"],
    color="violet",
    s=30,
    alpha=df['hpi']/100,
    transform=crs.PlateCarree()
)

#put top 3 rows name on map
for index,row in df.head(3).iterrows():
    plt.text(
        x=row["bplace_lon"],
        y=row["bplace_lat"],
        s=row["name"],
        weight='bold',
        transform=crs.PlateCarree(),)

#title
plt.show()
```


Birth Place of famous Individuals born in the UK and Ireland after 1990
58.5°N
57°N
55.5°N
54°N
52.5°N
51°N
49.5°N
7.5°W
6°W
4.5°W
3°W
1.5°W
0°
1.5°E
Ed Sheeran
Alan Walker
Dua Lipa

Part 3.3B

In [17]:
```
df = pd.read_csv('/Users/elliottoates/Library/CloudStorage/OneDrive-UniversityofE
df
```

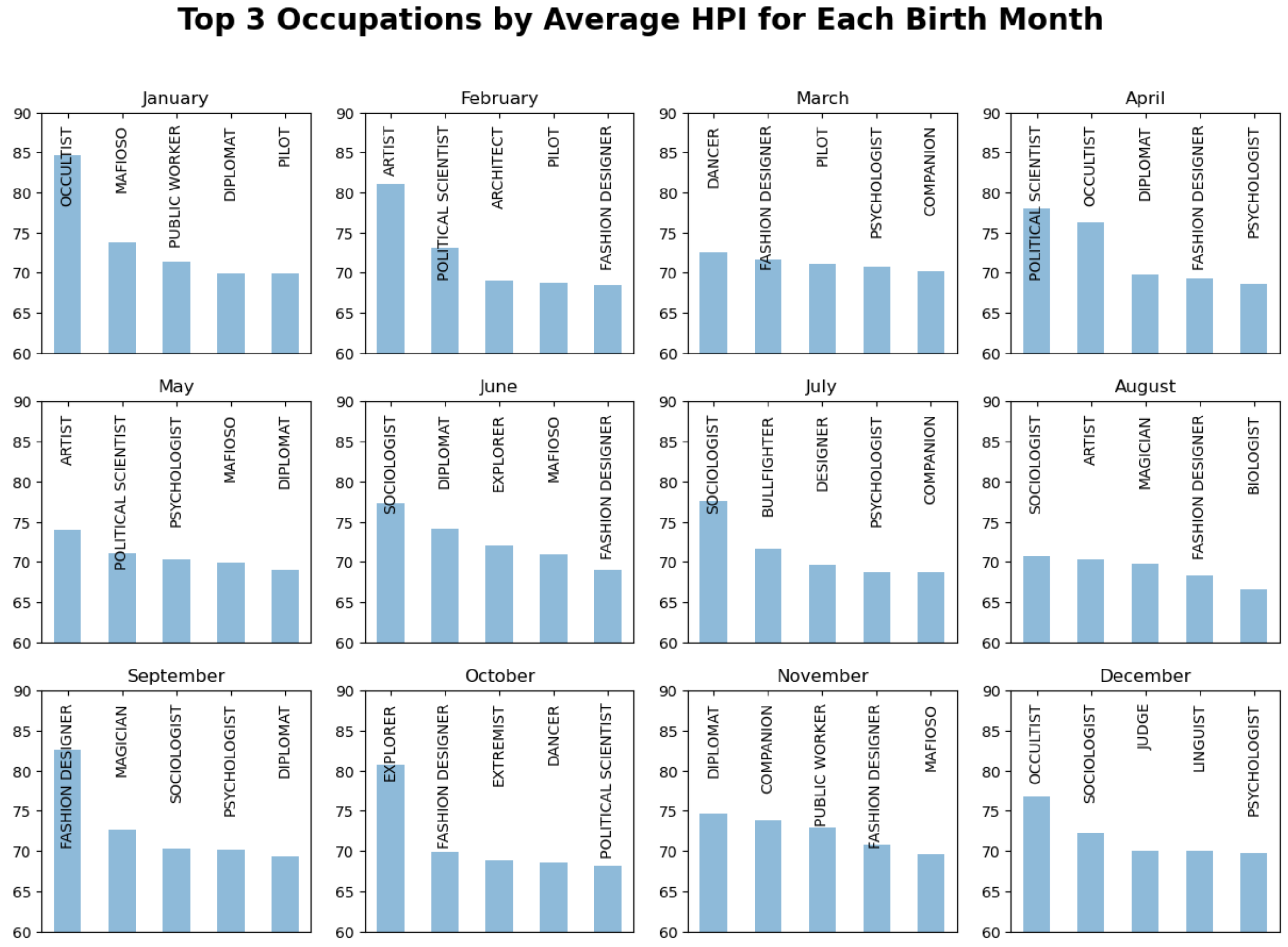
Out[17]:

	id	wd_id	wp_id	slug	name	occupation
0	18934	Q9458	18934	Muhammad	Muhammad	RELIGIOUS FIGURE
1	17414699	Q720	17414699	Genghis_Khan	Genghis Khan	MILITARY PERSONNEL
2	18079	Q762	18079	Leonardo_da_Vinci	Leonardo da Vinci	INVENTOR
3	14627	Q935	14627	Isaac_Newton	Isaac Newton	PHYSICIST
4	17914	Q255	17914	Ludwig_van_Beethoven	Ludwig van Beethoven	COMPOSER
...	...	...	...	...	...	...
88932	64119467	Q95877306	64119467	Siarhei_Tsikhanouski	Siarhei Tsikhanouski	YOUTUBER
88933	61153000	Q64840184	61153000	Irfaan_Ali	Irfaan Ali	POLITICIAN
88934	63320558	Q87075301	63320558	Guadalupe_Campanur_Tapia	Guadalupe Campanur Tapia	SOCIAL ACTIVIST
88935	62810909	Q83648587	62810909	Charli_D'Amelio	Charli D'Amelio	YOUTUBER
88936	63911513	Q93839274	63911513	Prince_Charles_of_Luxembourg_(born_2020)	Prince Charles of Luxembourg	NOBLEMAN

88937 rows × 34 columns

In [4]:

```
import matplotlib.pyplot as plt
import pandas as pd
import calendar
# load data
df = pd.read_csv('/Users/elliottoates/Library/CloudStorage/OneDrive-UniversityofE

# filter data
df = df[df['birthdate'].notnull()]
df = df[df['birthyear'] > 1900]

# calculate average HPI by occupation and birth month
df['birthmonth'] = pd.to_datetime(df['birthdate'], format='%Y-%m-%d').dt.month
df['hpi'] = df['hpi'].fillna(0)
avg_hpi = df.groupby(['birthmonth', 'occupation'])['hpi'].mean().reset_index(name

# create subplots
fig, axs = plt.subplots(3, 4, figsize=(15, 10))

# iterate over each month
for i, ax in enumerate(axs.flatten(), start=1):
    # get top 3 occupations for this month
    top_occupations = avg_hpi[avg_hpi['birthmonth'] == i].sort_values(by='avg_hpi
    # plot bar chart of top 3 occupations
    ax.bar(top_occupations['occupation'], top_occupations['avg_hpi'],width =0.5,a
    ax.set_xticklabels(top_occupations['occupation'],rotation=90)
    ax.tick_params(axis='x', direction='in',pad = -155, top = True, bottom = Fals
    ax.set_title(calendar.month_name[i])
    ax.set_ylim([60, 90])
    ax.grid(False)

# set overall title
fig.suptitle('Top 3 Occupations by Average HPI for Each Birth Month',size = 20, w
#plt.tight_layout()
plt.show()
```



The plot displays the top five occupations by average HPI for each birth month, with each subplot representing a different month. This visualization is interesting because it provides insights into the distribution of occupations memorability by birth month. For example, we can see that certain occupations, such as oculists and diplomats, consistently appear in the top five for multiple months. Although this figure could perhaps be more usefull if the metric was by frequency in the respective birth month rather than mean hpi, the fact that this data set is centred around famous people and is based largley on wikepedia pages has resulted in there being certain occupations that have a massive amount of persons and so all months show the exact same occupations (there are 17000 footballers in the filtered dataset, compared to 1 bullfighter).