In [1]:
```
## Setting up notebook

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

#pd.set_option('display.max_rows', None)
```

Problem 1 [25 marks]: Eurovision Winners Data Set

1A

In [2]:
```
df = pd.read_csv(r'/Users/elliottoates/OneDrive - University of Exeter/Data Science-Elliott's MacBook Pro/P

display (df)
print()
print('Column Headings:',list (df.keys()))


```

	Year	Date	Host City	Winner	Song	Performer	Points	Margin	Runner-up
0	1957	3-Mar	Frankfurt	Netherlands	"Net als toen"	Corry Brokken	31	14	France
1	1958	12-Mar	Hilversum	France	"Dors, mon amour"	Andr_ Claveau	27	3	Switzerland
2	1959	11-Mar	Cannes	Netherlands	"Een beetje"	Teddy Scholten	21	5	United Kingdom
3	1960	29-Mar	London	France	"Tom Pillibi"	Jacqueline Boyer	32	7	United Kingdom
4	1961	18-Mar	Cannes	Luxembourg	"Nous les amoureux"	Jean-Claude Pascal	31	7	United Kingdom
...	...	...	...	...	...	...	...	...	...
60	2014	10-May	Copenhagen	Austria	"Rise Like a Phoenix"	Conchita Wurst	290	52	Netherlands
61	2015	23-May	Vienna	Sweden	"Heroes"	MÎns Zelmerl_w	365	62	Russia
62	2016	14-May	Stockholm	Ukraine	"1944"	Jamala	534	23	Australia
63	2017	13-May	Kiev	Portugal	"Amar pelos dois"	Salvador Sobral	758	143	Bulgaria
64	2018	18-May	Lisbon	Israel	"Toy"	Netta	529	93	Cyprus

65 rows × 9 columns

```
Column Headings: ['Year', 'Date', 'Host City', 'Winner', 'Song', 'Performer', 'Points', 'Margin', 'Runner-u
p']
```

1B

In [3]:
```
display(df.loc[df.Year.duplicated(), :]) #extracting rows with duplicate values in year collumn
display(df.iloc[[12]])
```

	Year	Date	Host City	Winner	Song	Performer	Points	Margin	Runner-up
13	1969	29-Mar	Madrid	United Kingdom	"Boom Bang-a-Bang"	Lulu	18	No runner-up	No runner-up
14	1969	29-Mar	Madrid	Netherlands	"De troubadour"	Lenny Kuhr	18	No runner-up	No runner-up
15	1969	29-Mar	Madrid	France	"Un jour, un enfant"	Frida Boccara	18	No runner-up	No runner-up

	Year	Date	Host City	Winner	Song	Performer	Points	Margin	Runner-up
12	1969	29-Mar	Madrid	Spain	"Vivo cantando"	Salom_	18	No runner-up	No runner-up

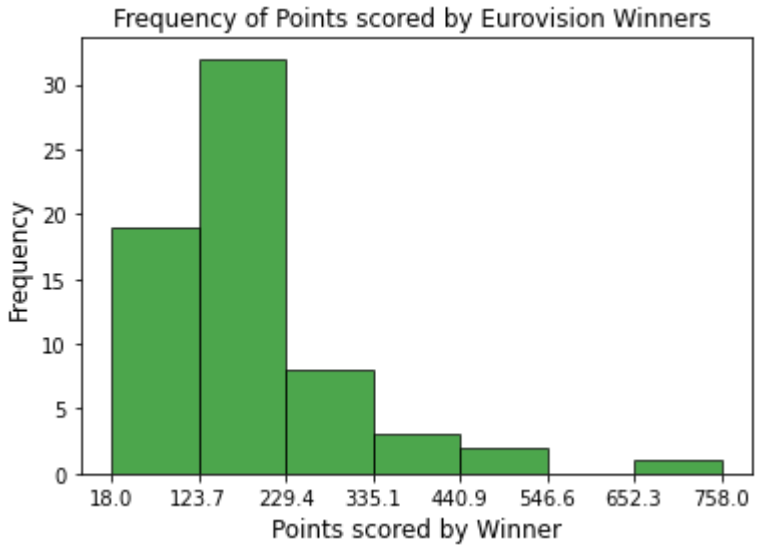
The 3 Additionall rows can be found by searching for rows that have duplicate values for in the "Year" collumn of the dataframe. It is apparent that in 1969 the data set has 4 different winners of which the last 3 are considered duplicates. After having done research online it the additional rows are valid as there was a four-way tie in the 1969 Eurovision')

1C

In [4]:
```
limitDF = df.loc[:,['Year','Points','Margin']]
```

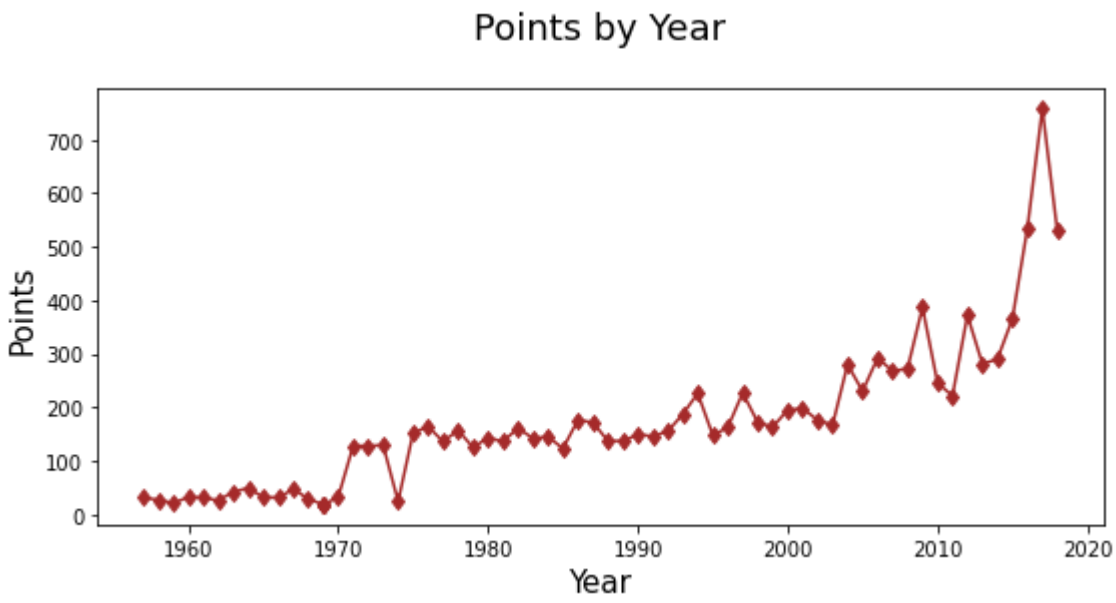
1D

In [5]:
```
%matplotlib inline

plt.title('Frequency of Points scored by Eurovision Winners ')
plt.xlabel('Points scored by Winner', size = 'large')

plt.ylabel('Frequency', size = 'larger')

plt.xticks(np.linspace(18,758,8))


plt.hist(df['Points'], bins=7,alpha=0.7,color='green',edgecolor='black');
```



1E

In [8]:
```
#part 1E
%matplotlib inline
plt.figure(figsize=(9,4))
plt.title('Points by Year', size = 18, pad = 25)
plt.xlabel('Year', size = 15)
plt.ylabel('Points', size = 15)

plt.plot(df ['Year'],df['Points'],color = 'brown', marker = 'd' );


```



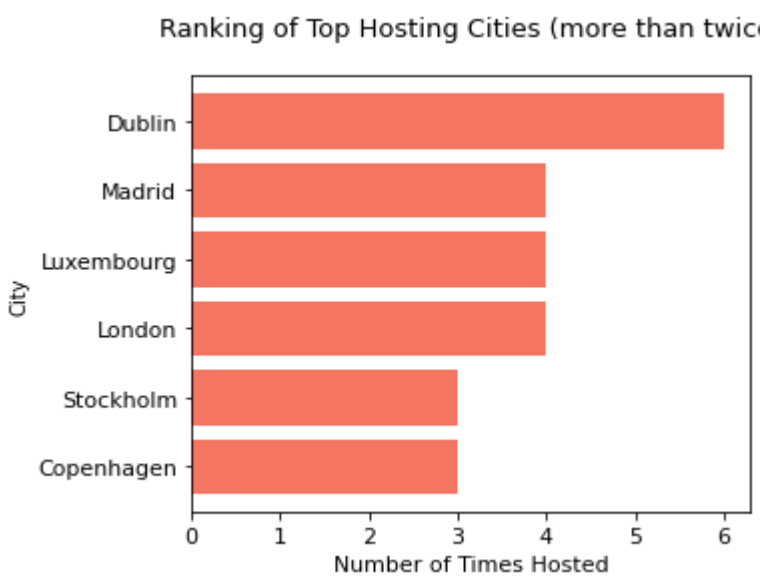
Rise in points : explanation

A plausible explanation for the rise in points that respective eurovision winners receive is that it is due to increasing amount of participants while the points that each participant allocates to others in a given year has remained the same.

With the data set provided, one cannot confirm this explanation as it doesnt include data about the amount of participants however upon researching further its apparent that this is the explanations as the number of participants has risen from 7 countries in 1956 to records of 43 participants in 2008.

1F

In [9]:
```
df2 = pd.DataFrame(df['Host City'].value_counts())

df2 = df2.iloc[0:6]
df2 = df2.reindex(['Copenhagen','Stockholm','London','Luxembourg','Madrid','Dublin'])

df2.plot.barh(color='#F47562', legend = False, width = 0.8, figsize= (5,4))

plt.title('Ranking of Top Hosting Cities (more than twice)',size = 13,pad = 20)
plt.ylabel('City',size = 10)
plt.xlabel('Number of Times Hosted',size = 11)
plt.tick_params(axis = 'both', labelsize = 11)



```



1G

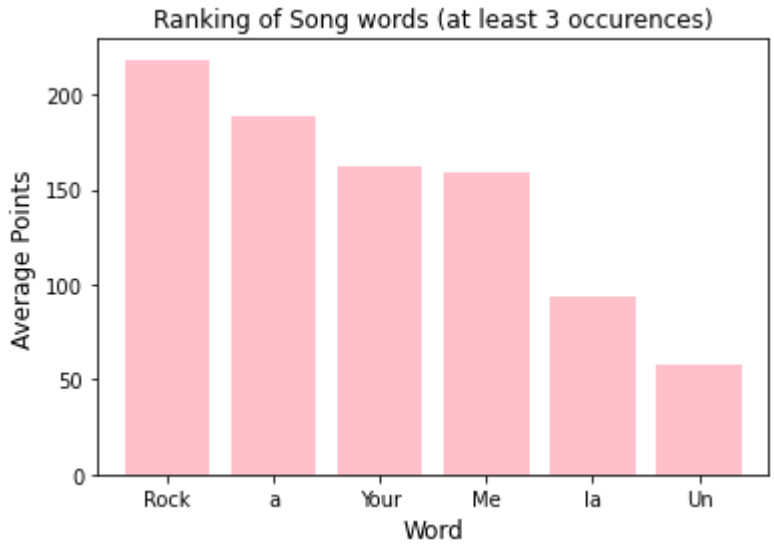
In [10]:
```
#part a
def function (row):
    df['Song'] = df.Song.str.replace('[_()''",-]','')
    n = row.name

    wordlist = df.loc[n,'Song'].split(' ')
    points = [df.loc[n,'Points']]*len(wordlist)

    return(list(zip(wordlist, points)))
```

In [11]:
```
function(df.loc[0,:])
```

Out[11]:
```
[('Net', 31), ('als', 31), ('toen', 31)]
```

In [12]:
```
#part b
tuple_list = ((df.apply(function,axis = 1)).explode()).tolist()
#part c
df_words = pd.DataFrame(tuple_list, columns = ['Word', 'Points'])
#part D
df_words= df_words.groupby('Word').filter(lambda x: len(x) > 2)
#part e
df_words = (df_words.groupby('Word',as_index= False).mean())
```

In [13]:
```
#part f
df_words = df_words.sort_values (by='Points',ascending = False)

%matplotlib inline

df_words.plot.bar(x='Word',y = 'Points',rot =0,color = 'pink',width = 0.8,legend = False)

plt.title('Ranking of Song words (at least 3 occurences)')
plt.xlabel('Word', size = 'large')
plt.ylabel('Average Points', size = 'larger');
```



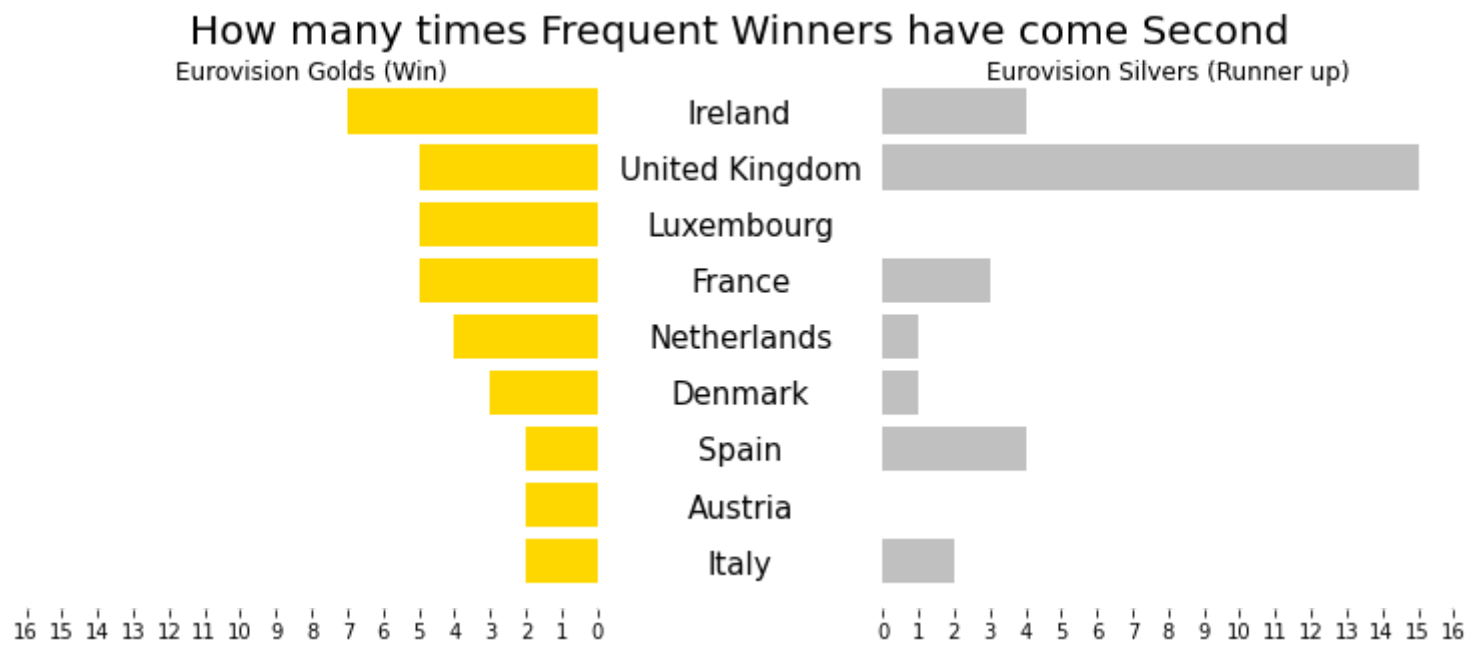
1H

In [14]:
```
df['Total Wins'] = df['Winner'].map(df['Winner'].value_counts())
df ['Total 2nd Place'] = df['Winner'].map(df['Runner-up'].value_counts())


df['Total 2nd Place'] = df['Total 2nd Place'].fillna(0)
df['Total 2nd Place'] = df['Total 2nd Place'].astype('int')


df = df.drop_duplicates (subset = 'Winner', ignore_index = True)

df['Country'] = df['Winner']

df = df[['Country','Total Wins', 'Total 2nd Place']]

df = df.head (9)
df = df.sort_values (by =['Total Wins'] ,ascending = True)
df
```

Out[14]:

	Country	Total Wins	Total 2nd Place
4	Italy	2	2
5	Austria	2	0
7	Spain	2	4
3	Denmark	3	1
0	Netherlands	4	1
1	France	5	3
2	Luxembourg	5	0
6	United Kingdom	5	15
8	Ireland	7	4

In [15]:
```
Country = df['Country']
num_country = len(Country)
Wins = df['Total Wins']
Runner_up  = df['Total 2nd Place']
pos = np.arange(num_country) + .5

fig, (ax_left, ax_right) = plt.subplots(ncols=2, figsize = (11,5))
ax_left.barh(pos, Wins, align='center', facecolor='gold')
ax_left.set_yticks([])

ax_right.barh(pos, Runner_up, align='center', facecolor = 'silver')
ax_right.set_yticks(pos)
ax_right.set_yticklabels(Country, ha='center', x=-0.225, size = 15)

ax_right.set_xlim((0,16))
ax_left.set_xlim((16,0))

plt.subplots_adjust(left = 0.05,right = 0.95,wspace = 0.5)

for i in [ax_left,ax_right]:

    i.spines["top"].set_visible(False)
    i.spines["right"].set_visible(False)
    i.spines["bottom"].set_visible(False)
    i.spines["left"].set_visible(False)
    i.tick_params(left = False)

    i.set_xticks(range(0,17,1))

plt.suptitle('How many times Frequent Winners have come Second',size =20, y=0.95)
ax_left.set_title('Eurovision Golds (Win)',y = 0.95)
ax_right.set_title('Eurovision Silvers (Runner up)',y = 0.95);
```



The extra analysis above is a 'tornado' graph that offers insight into the amount of times countries that have one eurovision have also come second place.

One can observe from the figure above that the amount of times a country has come second is not indicitive of the amount of wins. I chose the tornado style to show this as it is capable of showing this general assymetry between the wins and 2nd places well.

Problem 2 [25 marks]: UEFA Euro 2020 Scores Data Set

2A

In [16]:
```
df = pd.read_csv(r'/Users/elliottoates/OneDrive - University of Exeter/Data Science-Elliott's MacBook Pro/P

display (df)
print()
print('Column Headings:',list (df.keys()))
```

40	Group stage: Matchday 1	15.06.2021	False	False	False	Hungary	Portugal	0	3
41	Group stage: Matchday 1	14.06.2021	False	False	False	Spain	Sweden	0	0
42	Group stage: Matchday 1	14.06.2021	False	False	False	Poland	Slovakia	1	2
	Group								

2B

In [17]:
```
unique_countries = pd.unique(df[['team_name_home', 'team_name_away']].values.ravel())
print ('list of Unique Teams:',unique_countries)
```

```
list of Unique Teams: [' Italy ' ' England ' ' Denmark ' ' Spain ' ' Ukraine '
 ' Czech Republic ' ' Belgium ' ' Switzerland ' ' Sweden ' ' Germany '
 ' France ' ' Croatia ' ' Portugal ' ' Netherlands ' ' Austria ' ' Wales '
 ' Hungary ' ' Poland ' ' Slovakia ' ' Scotland ' ' Finland ' ' Russia '
 ' North Macedonia ' ' Turkey ']
```

In [18]:
```
df['team_name_home'] = df['team_name_home'].str.strip()
df['team_name_away'] = df['team_name_away'].str.strip()
unique_countries = pd.unique(df[['team_name_home', 'team_name_away']].values.ravel())
print ('list of Unique Teams:',unique_countries)
```

```
list of Unique Teams: ['Italy' 'England' 'Denmark' 'Spain' 'Ukraine' 'Czech Republic' 'Belgium'
 'Switzerland' 'Sweden' 'Germany' 'France' 'Croatia' 'Portugal'
 'Netherlands' 'Austria' 'Wales' 'Hungary' 'Poland' 'Slovakia' 'Scotland'
 'Finland' 'Russia' 'North Macedonia' 'Turkey']
```

2C

In [19]:
```
df_G = df.iloc [15:]
```

2D

In [20]:
```
def calcScore(row):
    n = row.name
    h_points=1
    a_points=1
    if df_G.loc[n,'team_home_score']>(df_G.loc[n,'team_away_score']):
        h_points=3
        a_points-=1
    elif df_G.loc[n,'team_home_score']<(df_G.loc[n,'team_away_score']):
        a_points=3
        h_points-=1
    return(h_points,a_points)

```

In [21]:
```
calcScore(df_G.loc[17,:])
```

Out[21]: (3, 0)

2E

In [22]:
```
df_G['team_home_points']= (df_G.apply(calcScore, axis = 1, result_type = 'expand'))[0]
df_G['team_away_points'] = (df_G.apply(calcScore, axis = 1, result_type = 'expand'))[1]
df_G
```

41	stage: Matchday 1	14.06.2021	False	False	False	Spain	Sweden	0	0
42	Group stage: Matchday 1	14.06.2021	False	False	False	Poland	Slovakia	1	2
43	Group stage: Matchday 1	14.06.2021	False	False	False	Scotland	Czech Republic	0	2
44	Group stage: Matchday 1	13.06.2021	False	False	False	Netherlands	Ukraine	3	2
45	Group stage: Matchday 1	13.06.2021	False	False	False	Austria	North Macedonia	3	1

2F

In [23]:
```
def GetGroupTeams(Team):
    group = [Team]
    away_opp = list(df_G.loc[df_G['team_name_home'] == Team, 'team_name_away'])
    home_opp = list(df_G.loc[df_G['team_name_away'] == Team, 'team_name_home'])
    group +=  away_opp + home_opp
    return group

```

In [24]:
```
GetGroupTeams('England')
```

Out[24]:
```
['England', 'Scotland', 'Croatia', 'Czech Republic']
```

2G

In [25]:
```
def GetGroupRows(Team):
    group_results = df_G[df_G['team_name_home'].isin(GetGroupTeams(Team))]
    group_results = group_results.reset_index(drop=True)
    return group_results
```

In [26]:
```
GetGroupRows('England')
```

Out[26]:

	stage	date	pens	pens_home_score	pens_away_score	team_name_home	team_name_away	team_home_score	team_away_score	poss
0	Group stage: Matchday 3	22.06.2021	False	False	False	Croatia	Scotland	3	1	
1	Group stage: Matchday 3	22.06.2021	False	False	False	Czech Republic	England	0	1	
2	Group stage: Matchday 2	18.06.2021	False	False	False	England	Scotland	0	0	
3	Group stage: Matchday 2	18.06.2021	False	False	False	Croatia	Czech Republic	1	1	
4	Group stage: Matchday 1	14.06.2021	False	False	False	Scotland	Czech Republic	0	2	
5	Group stage: Matchday 1	13.06.2021	False	False	False	England	Croatia	1	0	

6 rows × 22 columns

2H

In [27]:
```
def GetTeamStats (Team):
    Matches=  df_G[df_G.isin([Team]).any(axis=1)]

    Points = Matches.loc[Matches['team_name_home'] == Team, 'team_home_points'].sum()\
    +Matches.loc[Matches['team_name_away'] == Team, 'team_away_points'].sum()

    Scored = Matches.loc[Matches['team_name_home'] == Team, 'team_home_score'].sum()\
    +Matches.loc[Matches['team_name_away'] == Team, 'team_away_score'].sum()

    Conceded = Matches.loc[Matches['team_name_home'] == Team, 'team_away_score'].sum()\
    +Matches.loc[Matches['team_name_away'] == Team, 'team_home_score'].sum()

    Stats = {'Points':Points, 'Goals Scored': Scored, 'Goals Conceded': Conceded}

    return Stats
```

In [28]:
```
GetTeamStats ('England')
```

Out[28]:
```
{'Points': 7, 'Goals Scored': 2, 'Goals Conceded': 0}
```

2I

In [29]:
```
def GetGroupTable(Team):
    group = GetGroupTeams(Team)
    table = pd.DataFrame(GetTeamStats(Team),index=[])
    for i in group :
        stats = pd.DataFrame(GetTeamStats(i),index =[i])
        table = table.append(stats)

    table['Goal Difference'] = table['Goals Scored'] - table['Goals Conceded']
    table = table.sort_values(['Points', 'Goal Difference', 'Goals Scored'],ascending = False)
    return table
```

In [30]:
```
GetGroupTable('England')

```

Out[30]:

	Points	Goals Scored	Goals Conceded	Goal Difference
England	7	2	0	2
Croatia	4	4	3	1
Czech Republic	4	3	2	1
Scotland	1	1	5	-4

In [31]:
```
GetGroupTable('Italy')
```

Out[31]:

	Points	Goals Scored	Goals Conceded	Goal Difference
Italy	9	7	0	7
Wales	4	3	2	1
Switzerland	4	4	5	-1
Turkey	0	1	8	-7

2J

In [32]:
```
def GetAllGroupTables():

    for y in unique_countries:
        dft=pd.DataFrame(sorted(GetGroupTeams(y)))

    for i in unique_countries:
        dft[i]=pd.DataFrame(sorted(GetGroupTeams(i)))
        dft=dft.T.drop_duplicates().T

    for x in dft.iloc[1]:
        print(GetGroupTable(x))
```

In [33]:
```
1 GetAllGroupTables()
```

```
Scotland              1              1               5               -4
         Points  Goals Scored  Goals Conceded  Goal Difference
Belgium       9             7               1                6
Denmark       3             5               4                1
Finland       3             1               3               -2
Russia        3             2               7               -5
          Points  Goals Scored  Goals Conceded  Goal Difference
Sweden         7             4               2                2
Spain          5             6               1                5
Slovakia       3             2               7               -5
Poland         1             4               6               -2
                 Points  Goals Scored  Goals Conceded  Goal Difference
Netherlands           9             8               2                6
Austria               6             4               3                1
Ukraine               3             4               5               -1
North Macedonia       0             2               8               -6
          Points  Goals Scored  Goals Conceded  Goal Difference
France         5             4               3                1
Portugal       4             7               6                1
Germany        4             6               5                1
```

K : extra analysis

In [34]:
```
1  df['possession_away'] = df['possession_away'].str.strip(' %').astype(int)
2  df['possession_home'] = df['possession_home'].str.strip(' %').astype(int)
3  df['duels_won_home'] = df['duels_won_home'].str.strip(' %').astype(int)
4  df['duels_won_away'] = df['duels_won_away'].str.strip(' %').astype(int)
5
6  def GetAllTeamStats (Team):
7
8      Matches = df[df.isin([Team]).any(axis=1)]
9      Matches = Matches.iloc [1:]
10
11     AvgScored = (df.loc[df['team_name_home'] == Team, 'team_home_score'].sum()\
12     +df.loc[df['team_name_away'] == Team, 'team_away_score'].sum())
13
14
15     AvgConceded = (df.loc[df['team_name_home'] == Team, 'team_away_score'].sum()\
16     +df.loc[df['team_name_away'] == Team, 'team_home_score'].sum())
17
18
19     AvgPossession = (df.loc[df['team_name_home'] == Team, ('possession_home')].sum()\
20     +df.loc[df['team_name_away'] == Team, ('possession_away')].sum())
21
22
23     AvgSOnT = (df.loc[df['team_name_home'] == Team, ('shots_on_target_home')].sum()\
24     +df.loc[df['team_name_away'] == Team, ('shots_on_target_away')].sum())
25
26
27     AvgSOntConceded = (df.loc[df['team_name_home'] == Team, ('shots_on_target_away')].sum()\
28     +df.loc[df['team_name_away'] == Team, ('shots_on_target_home')].sum())
29
30
31     AvgDuelsWon = (df.loc[df['team_name_home'] == Team, ('duels_won_home')].sum()\
32     +df.loc[df['team_name_away'] == Team, ('duels_won_away')].sum())
33
34     AllStats = {'Team': Team,
35                 'Goals scored' :AvgScored,
36                 'Goals conceded': AvgConceded,
37                 'Mean Possession(%)': AvgPossession,
38                 'Shots on Target': AvgSOnT,
39                 'Shots on Target Conceded': AvgSOntConceded,
40                 'Mean Duels Won(%)' : AvgDuelsWon
41                }
42
43     return AllStats
```

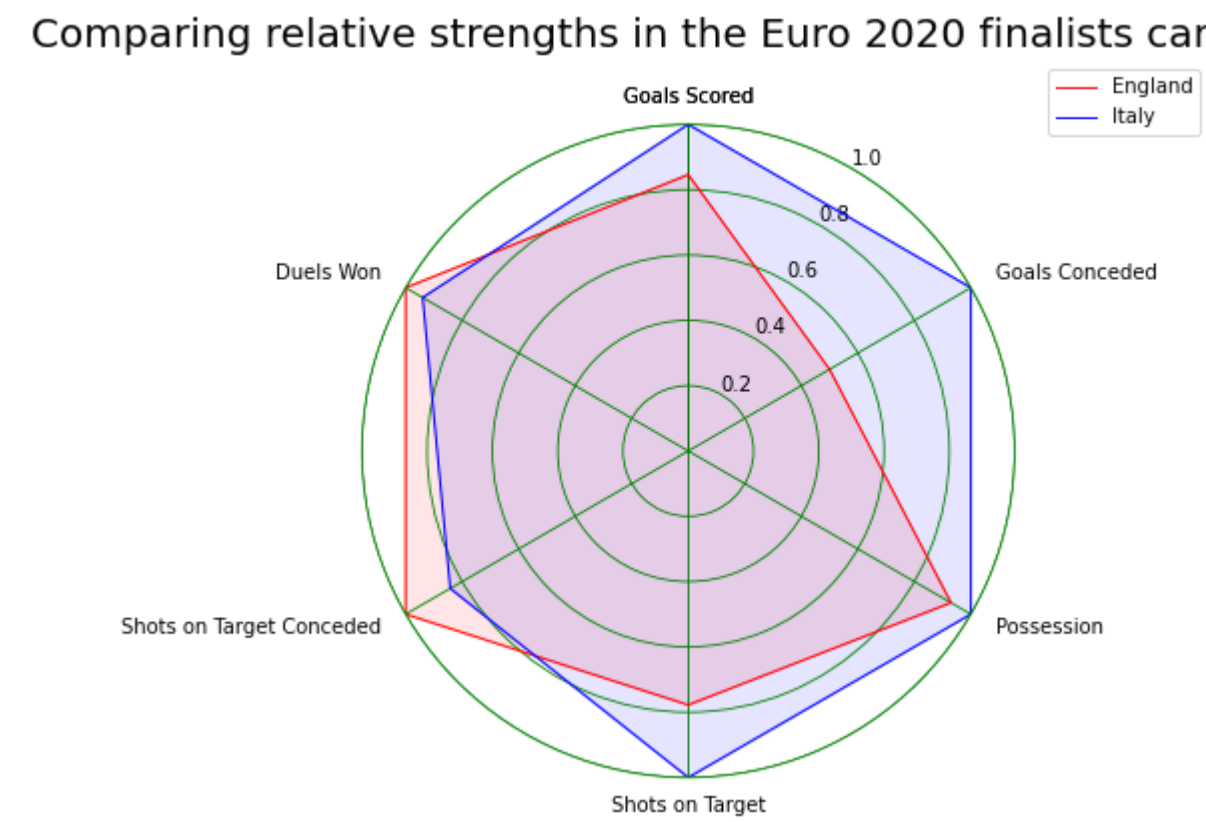
In [35]:
```
df_finalists = pd.DataFrame([GetAllTeamStats ('England'),GetAllTeamStats ('Italy')])
factors = ['Goals scored','Goals conceded','Mean Possession(%)','Shots on Target','Shots on Target Conceded

df = df_finalists

new_max = 100
new_min = 0
new_range = new_max - new_min

for factor in factors:
    max_val = df_finalists[factor].max()
    min_val = df_finalists[factor].min()
    val_range = max_val - min_val
    df_finalists[factor + '_adj'] = df_finalists[factor].apply(
        lambda x: (((x/ max_val) )))

df_finalists = df_finalists.loc[:,['Team','Goals scored_adj','Goals conceded_adj','Mean Possession(%)_adj',


df_finalists.set_index('Team',inplace=True)
df_finalists

```

Out[35]:

Team	Goals scored_adj	Goals conceded_adj	Mean Possession(%)_adj	Shots on Target_adj	Shots on Target Conceded_adj	Mean Duels Won(%)_adj
England	0.846154	0.5	0.929688	0.777778	1.000000	1.00000
Italy	1.000000	1.0	1.000000	1.000000	0.842105	0.93956

Figure

In [36]:
```
labels = ['','Goals Conceded', 'Possession','Shots on Target', 'Shots on Target Conceded', 'Duels Won','Goal
num_vars = len(factors)

angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
angles += angles[:1]

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

#England
values = df_finalists.loc['England'].tolist()
values += values[:1]
ax.plot(angles, values, color='red', linewidth=1, label= 'England')
ax.fill(angles, values, color='red', alpha=0.1)

#Italy
values = df_finalists.loc['Italy'].tolist()
values += values[:1]
ax.plot(angles, values, color='blue', linewidth=1, label= 'Italy')
ax.fill(angles, values, color='blue', alpha=0.1)


#Grid
ax.set_theta_offset(np.pi / 2)
ax.set_theta_direction(-1)
ax.set_thetagrids(np.degrees(angles), labels)
ax.grid(color='green')
ax.spines['polar'].set_color('green')

#Labels
for label, angle in zip(ax.get_xticklabels(), angles):
    if angle in (0, np.pi):
        label.set_horizontalalignment('center')
    elif 0 < angle < np.pi:
        label.set_horizontalalignment('left')
    else:
        label.set_horizontalalignment('right')
ax.set_ylim(0, 1)
ax.set_rlabel_position(180 / num_vars)

#title
ax.set_title('Comparing relative strengths in the Euro 2020 finalists campaign', y=1.1, Size = 20)

ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1));
```



The above figure is a radar chart that compares the relative strenghts and weeknesses of the Italy and European Teams in their campaign to the Final.

Data is collected from the group and knockout-stages (excluding the final).

The different attributes have all been scaled to show the relative weakness of either team. For example with goals scored, Italy has a value of 1, and England has one of ~0.84 which is the amount of goals England scored in the run-up to the final relative to the amount italy scored.

A graph like this could be used in preperation of the final to help either team focus on aspects of the game that the other team is significantly stronger in. Although in the case of this final, both teams had in most metrics a similar campaign. One must note that for goals conceded england has a lower score but due to the nature of the metric this is actually "better".

Problem 3 [25 marks]: COVID-19

3A

In [37]:
```
df = pd.read_csv(r'/Users/elliottoates/OneDrive - University of Exeter/Data Science-Elliott's MacBook Pro/P
display (df)
print('Column Headings:',list (df.keys()))
```

	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	...
0	AFG	Asia	Afghanistan	2020-02-24	5.0	5.0	NaN	NaN	NaN	NaN	...
1	AFG	Asia	Afghanistan	2020-02-25	5.0	0.0	NaN	NaN	NaN	NaN	...
2	AFG	Asia	Afghanistan	2020-02-26	5.0	0.0	NaN	NaN	NaN	NaN	...
3	AFG	Asia	Afghanistan	2020-02-27	5.0	0.0	NaN	NaN	NaN	NaN	...
4	AFG	Asia	Afghanistan	2020-02-28	5.0	0.0	NaN	NaN	NaN	NaN	...
...	...	...	...	...	...	...	...	...	...	...	...
168159	ZWE	Africa	Zimbabwe	2022-03-09	240343.0	0.0	405.714	5400.0	0.0	0.571	...
168160	ZWE	Africa	Zimbabwe	2022-03-10	241548.0	1205.0	577.857	5408.0	8.0	1.714	...
168161	ZWE	Africa	Zimbabwe	2022-03-11	241548.0	0.0	401.286	5408.0	0.0	1.571	...
168162	ZWE	Africa	Zimbabwe	2022-03-12	242069.0	521.0	435.714	5412.0	4.0	2.143	...
168163	ZWE	Africa	Zimbabwe	2022-03-13	242515.0	446.0	472.286	5414.0	2.0	2.143	...

168164 rows × 67 columns

```
Column Headings: ['iso_code', 'continent', 'location', 'date', 'total_cases', 'new_cases', 'new_cases_smoothe
d', 'total_deaths', 'new_deaths', 'new_deaths_smoothed', 'total_cases_per_million', 'new_cases_per_million',
'new_cases_smoothed_per_million', 'total_deaths_per_million', 'new_deaths_per_million', 'new_deaths_smoothed_
per_million', 'reproduction_rate', 'icu_patients', 'icu_patients_per_million', 'hosp_patients', 'hosp_patient
s_per_million', 'weekly_icu_admissions', 'weekly_icu_admissions_per_million', 'weekly_hosp_admissions', 'week
ly_hosp_admissions_per_million', 'total_tests', 'new_tests', 'total_tests_per_thousand', 'new_tests_per_thous
and', 'new_tests_smoothed', 'new_tests_smoothed_per_thousand', 'positive_rate', 'tests_per_case', 'tests_unit
s', 'total_vaccinations', 'people_vaccinated', 'people_fully_vaccinated', 'total_boosters', 'new_vaccination
s', 'new_vaccinations_smoothed', 'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred', 'people_f
ully_vaccinated_per_hundred', 'total_boosters_per_hundred', 'new_vaccinations_smoothed_per_million', 'new_peo
ple_vaccinated_smoothed', 'new_people_vaccinated_smoothed_per_hundred', 'stringency_index', 'population', 'po
pulation_density', 'median_age', 'aged_65_older', 'aged_70_older', 'gdp_per_capita', 'extreme_poverty', 'card
iovasc_death_rate', 'diabetes_prevalence', 'female_smokers', 'male_smokers', 'handwashing_facilities', 'hospi
tal_beds_per_thousand', 'life_expectancy', 'human_development_index', 'excess_mortality_cumulative_absolute',
'excess_mortality_cumulative', 'excess_mortality', 'excess_mortality_cumulative_per_million']
```

3B

In [38]:
```
print (df['date'].unique())
```

```
['2020-02-24' '2020-02-25' '2020-02-26' '2020-02-27' '2020-02-28'
 '2020-02-29' '2020-03-01' '2020-03-02' '2020-03-03' '2020-03-04'
 '2020-03-05' '2020-03-06' '2020-03-07' '2020-03-08' '2020-03-09'
 '2020-03-10' '2020-03-11' '2020-03-12' '2020-03-13' '2020-03-14'
 '2020-03-15' '2020-03-16' '2020-03-17' '2020-03-18' '2020-03-19'
 '2020-03-20' '2020-03-21' '2020-03-22' '2020-03-23' '2020-03-24'
 '2020-03-25' '2020-03-26' '2020-03-27' '2020-03-28' '2020-03-29'
 '2020-03-30' '2020-03-31' '2020-04-01' '2020-04-02' '2020-04-03'
 '2020-04-04' '2020-04-05' '2020-04-06' '2020-04-07' '2020-04-08'
 '2020-04-09' '2020-04-10' '2020-04-11' '2020-04-12' '2020-04-13'
 '2020-04-14' '2020-04-15' '2020-04-16' '2020-04-17' '2020-04-18'
 '2020-04-19' '2020-04-20' '2020-04-21' '2020-04-22' '2020-04-23'
 '2020-04-24' '2020-04-25' '2020-04-26' '2020-04-27' '2020-04-28'
 '2020-04-29' '2020-04-30' '2020-05-01' '2020-05-02' '2020-05-03'
 '2020-05-04' '2020-05-05' '2020-05-06' '2020-05-07' '2020-05-08'
 '2020-05-09' '2020-05-10' '2020-05-11' '2020-05-12' '2020-05-13'
 '2020-05-14' '2020-05-15' '2020-05-16' '2020-05-17' '2020-05-18'
 '2020-05-19' '2020-05-20' '2020-05-21' '2020-05-22' '2020-05-23'
 '2020-05-24' '2020-05-25' '2020-05-26' '2020-05-27' '2020-05-28'
```

3C

In [39]:
```
print ('number of unique countries in data set:',len(df['iso_code'].unique()))
```

```
number of unique countries in data set: 238
```

3D

In [40]:
```
df = df[df['iso_code'].str.contains("OWID_") == False]
df = df.reset_index()
```

In [41]:
```
print ('number of unique countries in data set:',len(df['iso_code'].unique()))
```

```
number of unique countries in data set: 223
```

3E

In [42]:
```
df_v = df[df['new_vaccinations'].notna()]
df_v = df_v.reset_index()

earliest_date = df_v['date'].min()
print (earliest_date)
```

```
2020-12-03
```

3F

In [43]:
```
df_v['total_vaccinations_per_million'] = 10000 * df_v['total_vaccinations_per_hundred']
```

3G

In [44]:
```
df_v['date'] = df_v['date'].apply(pd.to_datetime)
df_v['DaysSince3Dec20'] = (df_v['date']-df_v['date'].min()).dt.days

```

3H

In [45]:
```
min_deaths = df_v.groupby('iso_code')['total_deaths_per_million'].min() #earliest date deaths /mill

max_deaths = df_v.groupby('iso_code')['total_deaths_per_million'].max() #latest date deaths/mill

total_vax = df_v.groupby('iso_code')['total_vaccinations_per_hundred'].max()  #latest date vax/100

df_country =pd.DataFrame ({'total_vaccinations_per_hundred': total_vax, 'total_deaths_per_million(earliest (
df_country['TotalDeathPerMSinceVac'] = (df_country.iloc[:, 2])-(df_country.iloc[:, 1])
df_country

```

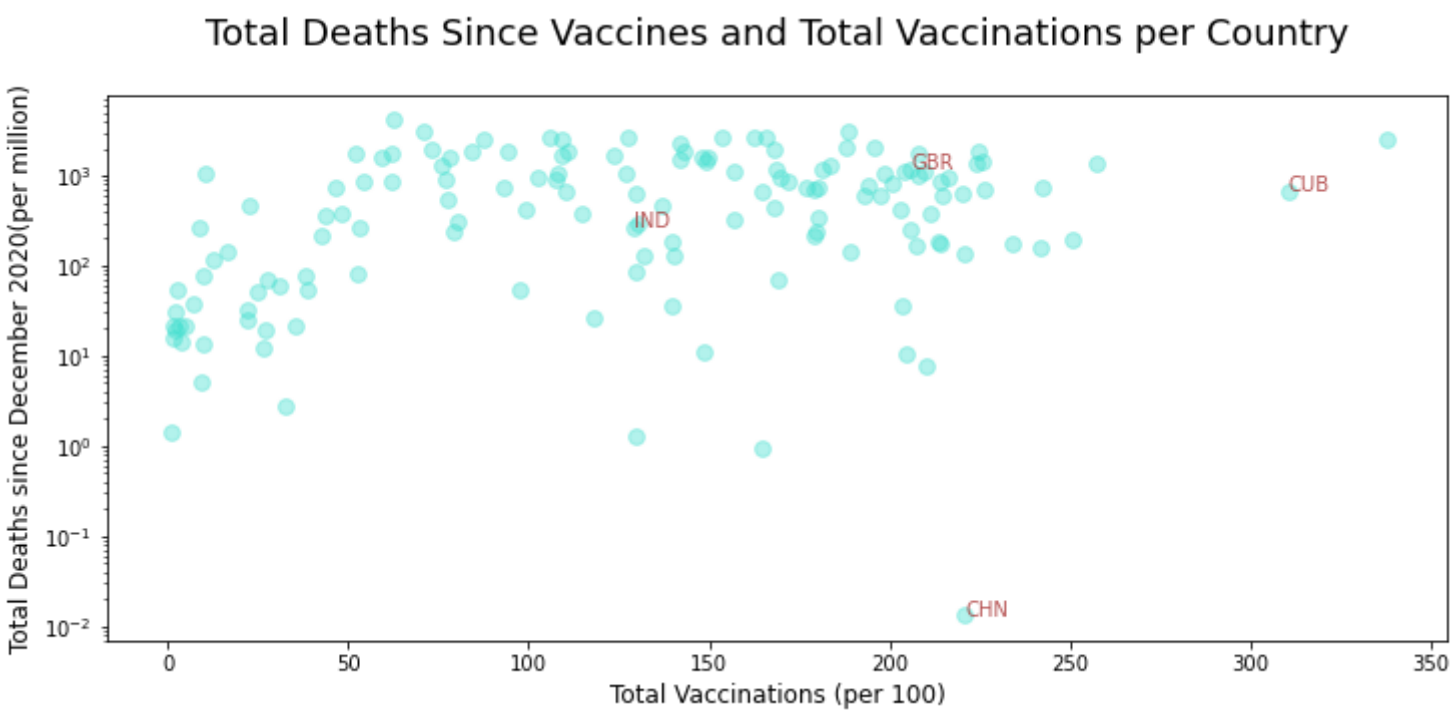
Out[45]:

iso_code	total_vaccinations_per_hundred	total_deaths_per_million(earliest date)	total_deaths_per_million(latest date)	TotalDeathPerMSinceVac
ABW	157.01	858.249	1977.704	1119.455
AFG	12.76	72.021	185.840	113.819
AIA	18.26	NaN	NaN	NaN
ALB	93.05	437.184	1191.465	754.281
AND	24.65	1512.527	1564.237	51.710
...	...	...	...	...
VCT	62.13	98.860	952.646	853.786
VNM	202.99	0.357	419.247	418.890
ZAF	54.16	813.747	1660.704	846.957
ZMB	16.31	65.008	209.295	144.287
ZWE	52.95	96.474	358.597	262.123

171 rows × 4 columns

3I

In [46]:
```
%matplotlib inline

plt.figure(figsize=(12,5))
plt.title('Total Deaths Since Vaccines and Total Vaccinations per Country', size = 18, pad = 25)
plt.ylabel('Total Deaths since December 2020(per million)', size = 12)
plt.xlabel('Total Vaccinations (per 100)', size = 12)


ax = plt.gca()
ax.set_yscale('log')

ax.text(220.84,0.013, "CHN", color = 'brown',alpha = 0.8) #CHN
ax.text(129.14,260.925, "IND", color = 'brown',alpha = 0.8) #IND
ax.text(205.72,1183.704, "GBR", color = 'brown',alpha = 0.8)
ax.text(310.42,665.960, "CUB", color = 'brown',alpha = 0.8)

plt.scatter(df_country ['total_vaccinations_per_hundred'],df_country['TotalDeathPerMSinceVac'], alpha=0.4, c
```



3J

In [47]:
```
df_GBR = df_v[df_v['iso_code'].str.contains('GBR')]
df_GBR.drop(columns=['level_0','index'],
        axis=1,
        inplace=True)
df_GBR = df_GBR.reset_index(drop=True)

df_GBR.shape #Checking that its 424 rows
```

Out[47]: (424, 69)

3K

In [48]:
```
df_GBR['total_vaccinations_per_hundred_smoothed']= df_GBR['total_vaccinations_per_hundred'].rolling(window=
```

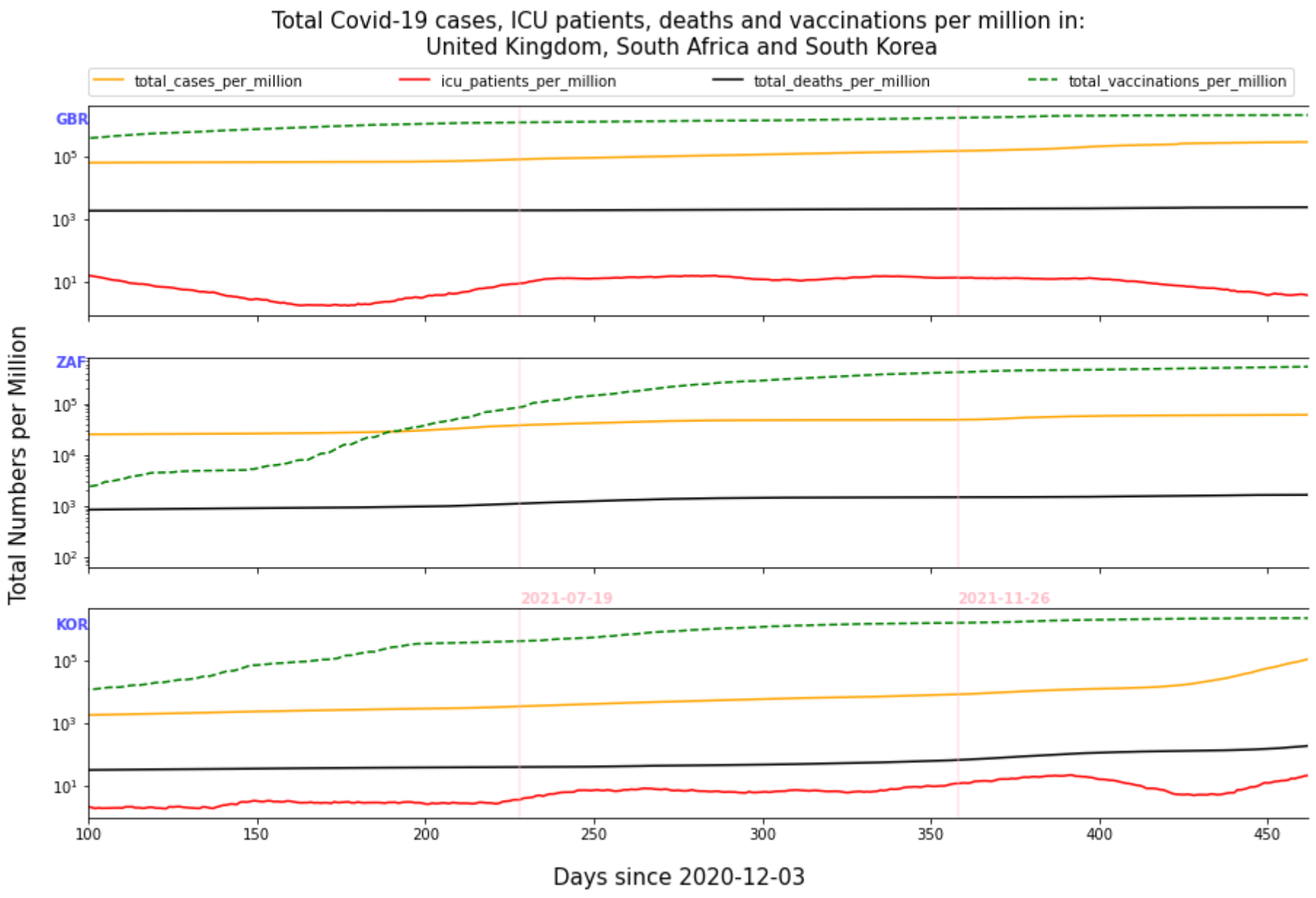
3L

In [49]:

```
#ZAF
df_ZAF = df_v[df_v['iso_code'].str.contains('ZAF')]
df_ZAF.drop(columns=['level_0','index'],
        axis=1,
        inplace=True)
df_ZAF = df_ZAF.reset_index(drop=True)

#KOR
df_KOR = df_v[df_v['iso_code'].str.contains('KOR')]
df_KOR.drop(columns=['level_0','index'],
        axis=1,
        inplace=True)
df_KOR = df_KOR.reset_index(drop=True)
```

In [50]:

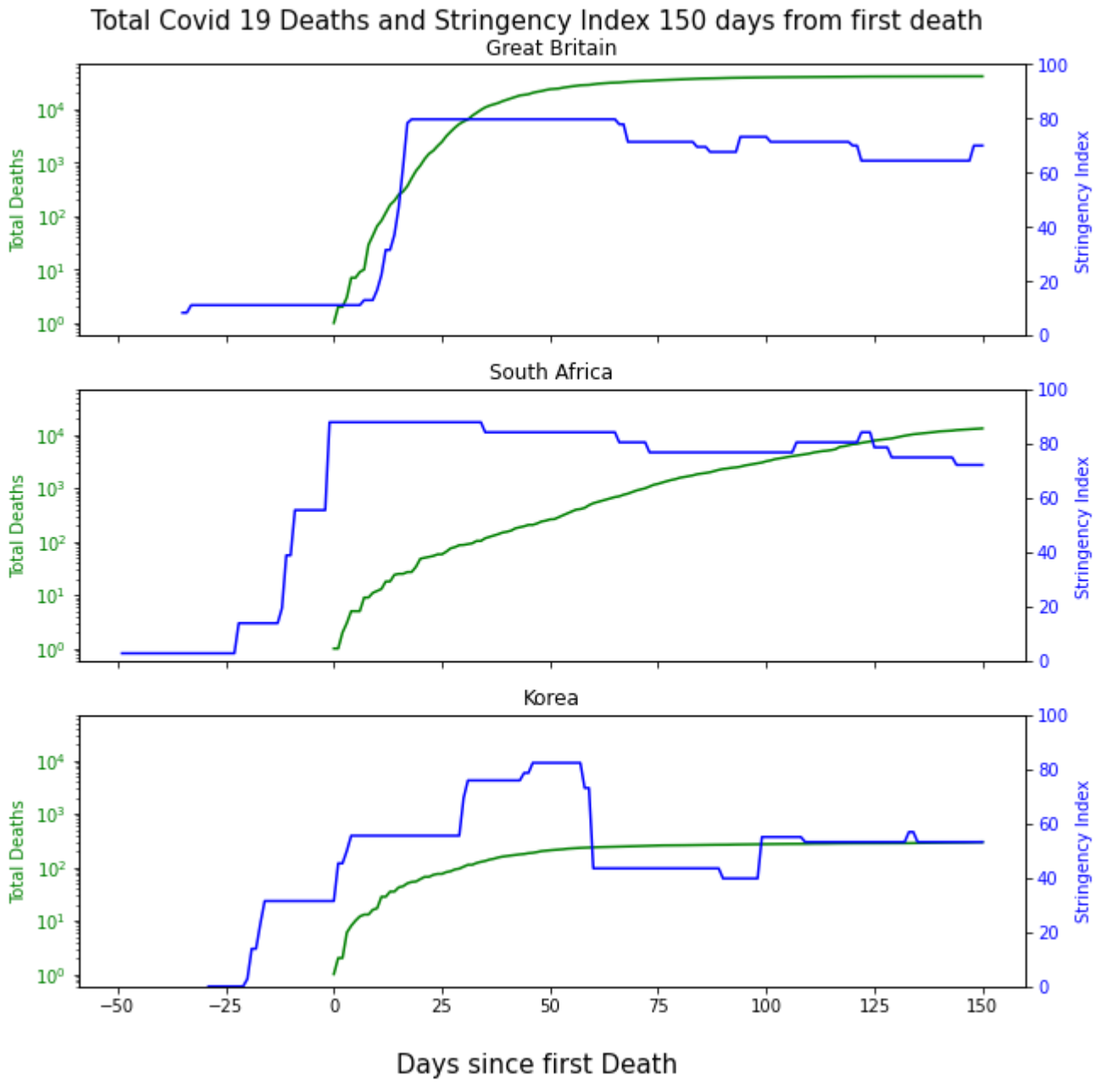
```
#figure
fig, axes = plt.subplots(nrows=3, ncols=1,figsize=(15,9),sharex = True)

plt.suptitle('Total Covid-19 cases, ICU patients, deaths and vaccinations per million in:\n United Kingdom,
fig.supxlabel('Days since 2020-12-03',size = 15, y = 0.05)
fig.supylabel('Total Numbers per Million',size = 15, x = 0.075)

#DATA
x = 'DaysSince3Dec20'
y1 ='total_cases_per_million'
y2 = 'icu_patients_per_million'
y3 = 'total_deaths_per_million'
y4 = 'total_vaccinations_per_million'

data = [df_GBR,df_ZAF,df_KOR]

#subplot - axes
for z,i in enumerate (data) :
    i.plot(x=x,y =[y1,y2,y3] ,ax=axes[z],legend =False,logy=True,color = ['orange','red','black','green'])
    i.plot(x=x,y =y4 ,ax=axes[z],legend =False,logy=True,color = 'green',linestyle = '--')
    axes[z].set_xlim(left=100,right =462)
    axes[z].set(xlabel=None)
    axes[z].axvline(x=228,ymin = 0, ymax = 1,color ='pink',alpha = 0.4 )
    axes[z].axvline(x=358,ymin = 0, ymax = 1,color ='pink',alpha = 0.4 )
    axes[z].spines["right"].set_visible(False)

#titles
axes[0].legend(ncol = 4, loc = (0,1.05),mode = 'expand')

axes[0].text(x=90,y=1000000,s = 'GBR', color = 'blue',weight='demi',alpha = 0.7)
axes[1].text(x=90,y=500000,s = 'ZAF', color = 'blue',weight='demi',alpha = 0.7)
axes[2].text(x=90,y=1000000,s = 'KOR', color = 'blue',weight='demi',alpha = 0.7)

#specific dates
axes[2].text(x=228,y=7000000,s = '2021-07-19', color = 'pink',weight='demi',alpha = 1)
axes[2].text(x=358,y=7000000,s = '2021-11-26', color = 'pink',weight='demi',alpha = 1);


```



Extra analysis

In [51]:
```
def make_dfs (x):
    y = df[df['iso_code'].str.contains(x)]
    y['date'] = y['date'].apply(pd.to_datetime)
    y['Days Since 1st Death'] =(y['date']-(y.loc[df['total_deaths'] == 1, 'date'].iloc[0])).dt.days
    y = y[y['Days Since 1st Death']<=150]
    y = y[['iso_code','total_deaths','stringency_index','Days Since 1st Death']]
    y = y[y['stringency_index'].notna()]
    y = y.reset_index(drop=True)
    return y
```

In [52]:
```
fig,(ax,ax1,ax2) =plt.subplots (3,1,figsize = (10,10), sharex = True,sharey = True)

plt.suptitle('Total Covid 19 Deaths and Stringency Index 150 days from first death',size=15,weight = 'ultra
fig.supxlabel('Days since first Death',size = 15, y = 0.05)

isocodes = ['GBR','ZAF','KOR']
axes = [ax,ax1,ax2]

ax.set (title = 'Great Britain')
ax1.set (title = 'South Africa')
ax2.set (title = 'Korea')


for i,z in zip (isocodes,axes):

    axa=z.twinx()
    x = make_dfs(i) ['Days Since 1st Death']
    y1 = make_dfs(i)['total_deaths']
    y2 = make_dfs(i) ['stringency_index']

    z.plot(x, y1, color='g')
    axa.plot(x, y2, color='b')

    z.set_ylabel( 'Total Deaths',color = 'g')
    z.set(yscale = 'log')

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

    def color_y_axis(ax, color):
        for t in ax.get_yticklabels():
            t.set_color(color)
    color_y_axis(z, 'g')
    color_y_axis(axa, 'b')
```



The Figure above provides insight into variation in govornent responses to the pandemic. Although most contries followed the same trend of getting stricter. i wanted to visualias variation between different responses.

The rate of which govornments implemented stricter measures played a critical role in preventing covid deaths. The figure above compares the response of the British, South-Korean and South-African govornments.

It is clear that the South Korean govornment who raised measures significantly even before the first death where then better able to combat the pandemic over the next 150 days, while in the UK the increase in the stringency of responses lagged the growth in deaths