SI 618 Project Report - Refugee Connectivity Trends in Greece and Serbia

Yea-Ree Chang (cyearee) / FA18

Welcome to my project report of WiFi connectivity trends in Greek and Serbian refugee camps and asylum centers!

I chose this topic as information and connectivity is starting to get recognized as a real need for displaced peoples and yet, not much research has been done about the actual effects of meeting these needs and the points of improvement in doing so. In fact, much of the research regarding information and communications technology in migrant camps has been largely qualitative and extremely narrow in scope in terms of sample size and methodology (mostly qualitative). By taking a more wide-encompassing and quantitative approach, this work offers a new perspective that is more objective and looks more at the trends and effects that lie beneath the surface.

This report looks specifically at the four following questions:

1. How is usage affected by location?
2. How well can we predict future data usage trends based on forecasting?
3. Can we predict whether a camp is overcapacitated at a given month by looking at its demographic data?
4. Do camps with more refugees from impoverished nations use less data?

The datasets for this report are CSV compilations of monthly Cisco Meraki router data and UNHCR (UN High Commissioner for Refugees) refugee camp site profiles to compare demographic data and Wifi usage trends in each of the refugee camps profiled. They consist of information from 36 Greek and 9 Serbian camps from a sporadic spread of months ranging from April 2016 to September 2018.

The following are some of the main variables: camp IDs (string), camp countries (string), total usage per person (daily and monthly) (float), the nationality breakdown of each camp (e.g. % of Syrians, Iraqis) (float), the age and gender breakdown of each camp (e.g. % of children, adult females) (float), month/date/time (datetime or string), latitude and longitude of camp (float), population (integer), and whether a camp is overcapacitated or not (binary).

It should be noted that, as I have stated before, this dataset is from my MTOP thesis which is still in-progress. I have spoken to the professor of this and have mentioned it in my proposal for this report.

Map of Greece (green) and Serbia (orange) - Source: Wikipedia

1. How is usage affected by location?

I'll be plotting the total usage per person and bandwidth on the map using the latitude and longitude of each map using a Python map-specific visualization tool called Folium (https://github.com/python-visualization/folium). I'll only be looking at the data from September 2018 as not many of the months have data for both Greece and Serbia and September is the most recent month for which I have data from both countries.

Variables used - Camp name (string; from camp.csv), latitude/longitude (float; from camp.csv), month/year (string), population (integer; from camp_demographics.csv), total usage in kB (float; from hourly_usage_per_camp.csv aggregated by sum)

Let's import the necessary modules and datasets first...

In [250]:
```
import pandas as pd
import folium
import warnings
warnings.filterwarnings('ignore')

camp_gen = pd.read_csv('data/camp.csv')
camp_dem = pd.read_csv('data/camp_demographics.csv')
hourly_usage = pd.read_csv('data/hourly_usage_per_camp.csv')
nationality = pd.read_csv('data/refugee_nationality.csv')
camp_gen['camp_file_name']=camp_gen['camp_file_name'].apply(str).str.strip()
camp_gen.head()
```

Out[250]:

	Unnamed: 0	camp_name	camp_file_name	type	authority	longitude	latitude
0	0	Alexandria	Alexandria-Ref_GRE_003	TAS	hellenic army	40.635371	22.453998
1	1	Diavata	Diavata_GRE_008	TAS	MoD	40.700950	22.863940
2	2	Doliana	XXDoliana_GRE_007	TAS	hellenic army	39.898526	20.578272
3	3	Eleonas	Eleonas_GRE_013	TAS	MoMP	37.984117	23.696326
4	4	Filipiada	XXFilipiada_GRE_019	TAS	hellenic army;MoMP	39.224421	20.872987

In [251]:
```
hourly_usage['campfilename']=hourly_usage['campfilename'].apply(str).str.strip
()
hourly_usage.head()
```

Out[251]:

	campfilename	monthyear	date	time	total_usage_kb
0	Andravida_GRE_004	42016	4/1/2016	0:00:00	0.0
1	Andravida_GRE_004	42016	5/1/2016	0:00:00	0.0
2	Andravida_GRE_004	102016	10/1/2016	0:00:00	0.0
3	Andravida_GRE_004	102016	11/1/2016	0:00:00	0.0
4	Drama_GRE_011	42016	4/1/2016	0:00:00	0.0

In [252]:
```
camp_dem['campfilename']= camp_dem['campfilename'].apply(str).str.strip()
camp_dem.head()
```

Out[252]:

	campfilename	month	year	camppopulation	campcapacity	overcapacity	adult
0	Adasevci_SRB_010	1	2018	327	450	0.0	0.45
1	Bujanovac_SRB_001	1	2018	129	220	0.0	0.35
2	Divljana_SRB_002	1	2018	83	280	0.0	0.27
3	Kikinda_SRB_007	1	2018	162	240	0.0	0.33
4	Krnjaca_SRB_003	1	2018	643	1000	0.0	0.42

Looking at the datasets, there is some clean-up required before I can start visualizing my data. First of all, the demographics table which has the population data stores the month and the year in two different columns, whereas the hourly usage table puts them together in one. I think I like the demographics' approach better.

In [253]:
```
hourly_usage['month'] = hourly_usage['monthyear'].apply(str).str[:-4].astype(
'int64')
hourly_usage['monthyear'] = hourly_usage['monthyear'].apply(str).apply(lambda
x: x.zfill(6))
hourly_usage['year'] = hourly_usage['monthyear'].apply(str).str[-4:].astype('i
nt64')
hourly_usage.head()
```

Out[253]:

	campfilename	monthyear	date	time	total_usage_kb	month	year
0	Andravida_GRE_004	042016	4/1/2016	0:00:00	0.0	4	2016
1	Andravida_GRE_004	042016	5/1/2016	0:00:00	0.0	4	2016
2	Andravida_GRE_004	102016	10/1/2016	0:00:00	0.0	10	2016
3	Andravida_GRE_004	102016	11/1/2016	0:00:00	0.0	10	2016
4	Drama_GRE_011	042016	4/1/2016	0:00:00	0.0	4	2016

In [254]:
```
hourly_usage['total_usage_kb'].astype('float')
```

```
Out[254]: 0            0.000
          1            0.000
          2            0.000
          3            0.000
          4            0.000
          5            0.000
          6            0.000
          7            0.000
          8            0.000
          9            0.000
          10           0.000
          11           0.000
          12           0.000
          13           0.000
          14           0.000
          15           0.000
          16           0.000
          17           0.000
          18           0.000
          19           0.000
          20           0.000
          21           0.000
          22           0.000
          23           0.000
          24           0.000
          25           0.000
          26           0.000
          27           0.000
          28           0.000
          29           0.000
                      ...
          42328     8901.357
          42329    12485.642
          42330    12914.681
          42331    14459.240
          42332    12333.123
          42333     7104.306
          42334     8396.972
          42335    12674.793
          42336    11754.592
          42337    14732.425
          42338    14574.162
          42339     4816.530
          42340     9615.360
          42341    13853.143
          42342    11909.059
          42343        0.000
          42344        0.000
          42345        0.000
          42346        0.000
          42347        0.000
          42348        0.000
          42349        0.000
          42350        0.000
          42351        0.000
          42352        0.000
          42353        0.000
```

```
42354        0.000
42355        0.000
42356        0.000
42357        0.000
Name: total_usage_kb, Length: 42358, dtype: float64
```

Now I want to convert the hourly total usage per camp data into monthly by summing up the hourly data by camp and month. I'll also take this opportunity to start a new dataframe that will hold all the relevant data.

In [255]:
```
grouped_hourly = hourly_usage.groupby(['campfilename','monthyear'])['total_usage_kb'].sum()
grouped_hourly.head(10)
```

Out[255]:
```
campfilename            monthyear
Adasevci_SRB_010        012018        912541.145
                        022018        801803.019
                        092018       1774450.363
Adavesci_SRB_010        032018        816442.546
Alexandria-Ref_GRE_003  012017        640877.689
                        022018       1682236.306
                        032018       2028270.719
                        042016         38342.613
                        042018       1883450.310
                        052018       2176596.390
Name: total_usage_kb, dtype: float64
```

In [256]:
```
df1 = grouped_hourly.to_frame().reset_index()
df1.columns
```

Out[256]:
```
Index(['campfilename', 'monthyear', 'total_usage_kb'], dtype='object')
```

In [257]:
```
df1.head()
```

Out[257]:

	campfilename	monthyear	total_usage_kb
0	Adasevci_SRB_010	012018	912541.145
1	Adasevci_SRB_010	022018	801803.019
2	Adasevci_SRB_010	092018	1774450.363
3	Adavesci_SRB_010	032018	816442.546
4	Alexandria-Ref_GRE_003	012017	640877.689

In [258]:
```
df1['month'] = df1['monthyear'].apply(str).str[:-4].astype('int64')
df1['year'] = df1['monthyear'].apply(str).str[-4:].astype('int64')
df1.head()
```

Out[258]:

	campfilename	monthyear	total_usage_kb	month	year
0	Adasevci_SRB_010	012018	912541.145	1	2018
1	Adasevci_SRB_010	022018	801803.019	2	2018
2	Adasevci_SRB_010	092018	1774450.363	9	2018
3	Adavesci_SRB_010	032018	816442.546	3	2018
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017

Let's also get the actual camp names instead of these weird file names.

In [259]:
```
name_map = pd.Series(camp_gen.camp_name.values,index=camp_gen.camp_file_name).
to_dict()
df1['camp_name'] = df1['campfilename'].map(name_map)
```

In [260]:
```
name_map
```

Out[260]:
```
{'Alexandria-Ref_GRE_003': 'Alexandria',
 'Diavata_GRE_008': 'Diavata',
 'XXDoliana_GRE_007': 'Doliana',
 'Eleonas_GRE_013': 'Eleonas',
 'XXFilipiada_GRE_019': 'Filipiada',
 'Kato-Milia_GRE_061': 'Kato-Milia',
 'XXKonitsa_GRE_026': 'Konitsa',
 'Koutsochero_GRE_028': 'Koutsochero',
 'Lagkadikia_GRE_029': 'Lagkadikia',
 'Lavrio-New_GRE_030': 'Lavrio',
 'Leros-Lepida_GRE_032': 'Leros-Lepida',
 'Moria_GRE_001': 'Moria',
 'Nea-Kavala_GRE_034': 'Nea-Kavala',
 'XXPikpa-Leros_GRE_038': 'Pikpa-Leros',
 'Skaramangas_GRE_014': 'Skaramangas',
 'Vathi-Samos_GRE_053': 'Vathi-Samos',
 'XXVial-Chios-NRC_GRE_066': 'Vial-Chios',
 'Andravida_GRE_004': 'Andravida',
 'XXDerveni-Alexil_GRE_007': 'Derveni-Alexil',
 'XXElliniko-Hocky_GRE_016': 'Eliniko-Hockey',
 'XXIliadi_GRE_020': 'Illiadi',
 'XXMalakasa-MC_GRE_012': 'Malakasa',
 'XXOlympic-Stadium_GRE_035': 'Olympic Stadium',
 'Rafina_GRE_039': 'Rafina',
 'Ritsona_GRE_040': 'Ritsona',
 'XXSindos-Frakapore_GRE_043': 'Sindos-Frakapore',
 'XXSoftex_GRE_045': 'Softex',
 'XXSouda-Samns-Purse_GRE_046': 'Souda-Samns-Purse',
 'XXTrikala_GRE_048': 'Trikala',
 'Veria_GRE_054': 'Veria',
 'Kos_GRE_027': 'Kos',
 'Volos_GRE_055': 'Volos',
 'XXCherso_GRE_005': 'Cherso',
 'Drama_GRE_011': 'Drama',
 'Kavala_GRE_024': 'Kavala',
 'Serres_GRE_042': 'Serres',
 'Adasevci_SRB_010': 'Adasevci',
 'Bujanovac_SRB_001': 'Bujanovac',
 'Divljana_SRB_002': 'Divljana',
 'Kikinda_SRB_007': 'Kikinda',
 'Krnjaca_SRB_003': 'Krnjaca',
 'Obrenovac_SRB_011': 'Obrenovac',
 'Pirot_SRB_005': 'Pirot',
 'Presevo_SRB_006': 'Presevo',
 'Sombor_SRB_008': 'Sombor'}
```

In [261]:
```
df1.head()
```

Out[261]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci
3	Adavesci_SRB_010	032018	816442.546	3	2018	NaN
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria

Also finding out that some of the camp file names were incorrectly inputted, so I'm going to fix those. Hopefully, I remember to fix them in other files as I continue on this project as well! :D

In [262]:
```
incorrect = df1[['campfilename','monthyear']][df1['camp_name'].isnull()]
incorrect.head()
#incorrect.to_csv('incorrect.csv')
```

Out[262]:

	campfilename	monthyear
3	Adavesci_SRB_010	032018
14	Alexandria_GRE_003	102016
24	Andravida_GRE_008	042018
294	XXSouda_Samns_Purse	062017

In [263]:
```
wrongnames = list(incorrect['campfilename'].unique())
wrongnames
```

Out[263]:
```
['Adavesci_SRB_010',
 'Alexandria_GRE_003',
 'Andravida_GRE_008',
 'XXSouda_Samns_Purse']
```

In [264]:
```
df1['campfilename']=df1['campfilename'].replace({wrongnames[0]:'Adasevci_SRB_0
10',wrongnames[1]:'Alexandria-Ref_GRE_003',wrongnames[2]:'Andravida_GRE_004',w
rongnames[3]:'XXSouda-Samns-Purse_GRE_046'})
hourly_usage['campfilename']=hourly_usage['campfilename'].replace({wrongnames[
0]:'Adasevci_SRB_010',wrongnames[1]:'Alexandria-Ref_GRE_003',wrongnames[2]:'An
dravida_GRE_004',wrongnames[3]:'XXSouda-Samns-Purse_GRE_046'}) # doing this so
 I don't have to do it again in the future
df1['camp_name'] = df1['campfilename'].map(name_map)
df1.head()
```

Out[264]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci
3	Adasevci_SRB_010	032018	816442.546	3	2018	Adasevci
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria

I'm also going to grab the population and the latitude/longitude data...

In [265]:
```
df1['cmy'] = df1['campfilename']+df1['monthyear'].apply(str)
df1.head()
```

Out[265]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name	
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci	Adasevci_S
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci	Adasevci_S
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci	Adasevci_S
3	Adasevci_SRB_010	032018	816442.546	3	2018	Adasevci	Adasevci_S
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria	Alexandria-Ref_GRE_0

In [266]:
```
lat_map = pd.Series(camp_gen.latitude.values,index=camp_gen.camp_file_name).to
_dict()
long_map = pd.Series(camp_gen.longitude.values,index=camp_gen.camp_file_name).
to_dict()
df1['latitude'] = df1['campfilename'].map(lat_map)
df1['longitude'] = df1['campfilename'].map(long_map)
df1['camp_name'] = df1['campfilename'].map(name_map)
df1.head()
```

Out[266]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name	
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci	Adasevci_S
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci	Adasevci_S
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci	Adasevci_S
3	Adasevci_SRB_010	032018	816442.546	3	2018	Adasevci	Adasevci_S
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria	Alexandria-Ref_GRE_0

In [267]:
```
camp_dem['cmy'] = camp_dem['campfilename'] + camp_dem['month'].apply(str).appl
y(lambda x: x.zfill(2)) + camp_dem['year'].apply(str)
camp_dem.head()
```

Out[267]:

	campfilename	month	year	camppopulation	campcapacity	overcapacity	adult
0	Adasevci_SRB_010	1	2018	327	450	0.0	0.45
1	Bujanovac_SRB_001	1	2018	129	220	0.0	0.35
2	Divljana_SRB_002	1	2018	83	280	0.0	0.27
3	Kikinda_SRB_007	1	2018	162	240	0.0	0.33
4	Krnjaca_SRB_003	1	2018	643	1000	0.0	0.42

In [268]:
```
pop_map = pd.Series(camp_dem.camppopulation.values,index=camp_dem.cmy).to_dict
()
df1['population'] = df1['cmy'].map(pop_map)
df1.head()
```

Out[268]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name	
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci	Adasevci_S
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci	Adasevci_S
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci	Adasevci_S
3	Adasevci_SRB_010	032018	816442.546	3	2018	Adasevci	Adasevci_S
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria	Alexandria-Ref_GRE_0

In [269]:
```
df1['usage_per_person'] = df1.total_usage_kb/df1.population
df1.head()
```

Out[269]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name	
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci	Adasevci_S
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci	Adasevci_S
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci	Adasevci_S
3	Adasevci_SRB_010	032018	816442.546	3	2018	Adasevci	Adasevci_S
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria	Alexandria-Ref_GRE_0

Wow, well that was a long set-up. Now onto mapping!

Let's get the usage data for just September 2018...

In [270]:
```
sept = df1[df1['monthyear']=='092018']
print('September 2018 camps ({}): {}'.format(len(sept),', '.join(list(sept['ca
mp_name']))))
```

```
September 2018 camps (34): Adasevci, Alexandria, Andravida, Bujanovac, Diavat
a, Divljana, Drama, Eleonas, Kato-Milia, Kavala, Kikinda, Kos, Koutsochero, K
rnjaca, Lagkadikia, Lavrio, Leros-Lepida, Moria, Nea-Kavala, Obrenovac, Piro
t, Presevo, Ritsona, Serres, Skaramangas, Sombor, Vathi-Samos, Veria, Volos,
Doliana, Filipiada, Malakasa, Pikpa-Leros, Vial-Chios
```

In [271]:
```
sept.head()
```

Out[271]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name	
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci	Adasevci
13	Alexandria-Ref_GRE_003	092018	1553648.334	9	2018	Alexandria	Alexandri Ref_GRE
22	Andravida_GRE_004	092018	387253.626	9	2018	Andravida	Andravid
28	Bujanovac_SRB_001	092018	1881497.311	9	2018	Bujanovac	Bujanova
38	Diavata_GRE_008	092018	6060130.388	9	2018	Diavata	Diavata_

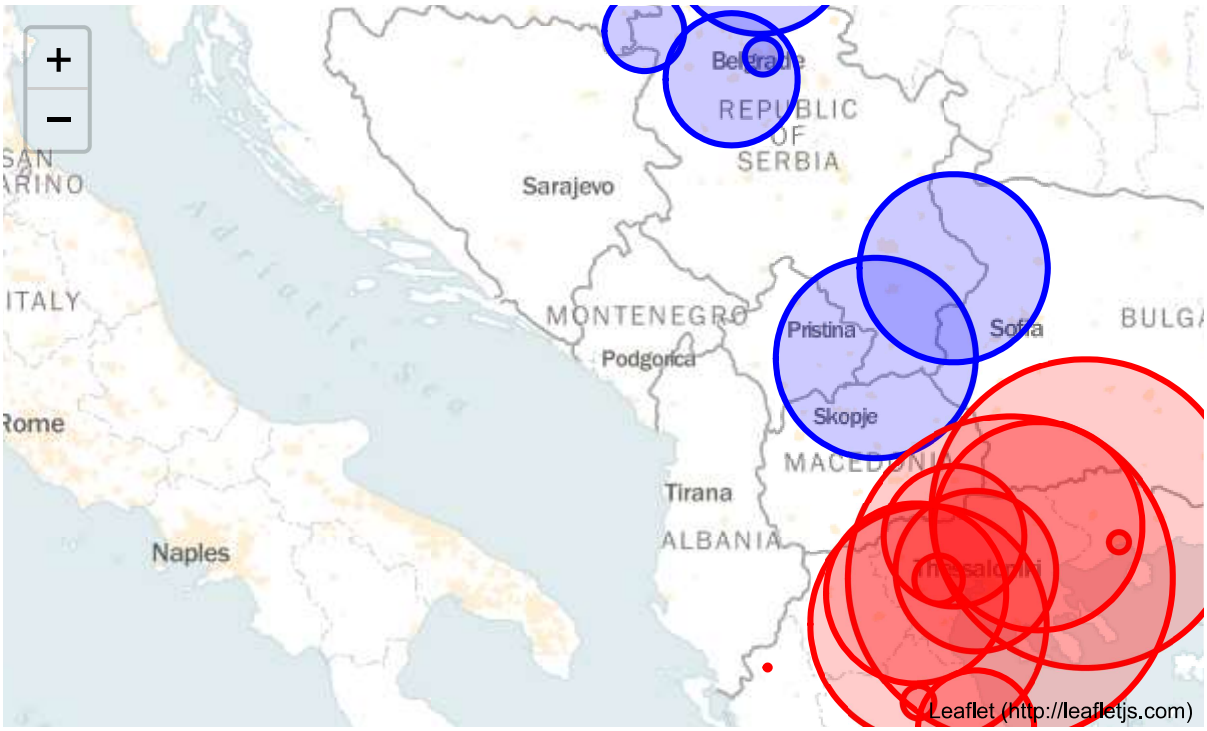
In [272]:
```
import numpy as np
from IPython.display import display, Markdown

display(Markdown('<strong>{}<br/>'.format('Map of Total Usage Per Person in Se
ptember 2018')))
display(Markdown('<span style="color:blue">blue circles </span>= Serbian camp
s'))
display(Markdown('<span style="color:red">red circles </span>= Greek camps'))

sep_m = folium.Map(location=[np.mean(sept.longitude),np.mean(sept.latitude)],
tiles="Mapbox Bright", zoom_start=6)
for i in range(0,len(sept)):
    if 'SRB' in sept.iloc[i]['campfilename']:
        folium.Circle(
          location=[sept.iloc[i]['longitude'], sept.iloc[i]['latitude']],
          popup='{} - {}kB per person'.format(sept.iloc[i]['camp_name'],np.rou
nd(sept.iloc[i]['usage_per_person'])),
          radius=sept.iloc[i]['usage_per_person']*10,
          color='blue',
          fill=True,
          fill_color='blue'
        ).add_to(sep_m)
    else:
        folium.Circle(
          location=[sept.iloc[i]['longitude'], sept.iloc[i]['latitude']],
          popup='{} - {}kB per person'.format(sept.iloc[i]['camp_name'],np.rou
nd(sept.iloc[i]['usage_per_person'])),
          radius=sept.iloc[i]['usage_per_person']*10,
          color='red',
          fill=True,
          fill_color='red'
        ).add_to(sep_m)

sep_m
```

Map of Total Usage Per Person in September 2018

blue circles = Serbian camps

red circles = Greek camps

Out[272]:



Procedure: I first had to create a dataframe to fulfill the requirements of this question. Starting with the hourly_usage_per_camp.csv, I first aggregated all the usage amount data, which I only had in hourly basis, into a whole month by camp. I then added population, which was used to create the "usage per person" variable, and latitude/longitude variables from the camp_demographic and camp CSVs. After all this, I was able to filter the data so that I had only data from September 2018, from which I was able to plot all of the camp locations with the corresponding usage data using the Folium package.

During this process, I ran into challenges such as not finding out that some camp IDs had trailing white spaces in some of the key data files until much later, for which I had to use the strip method. There were also some typos with the camp IDs in general which I had to correct by comparing with the master camp CSV file and mapping the correct names. I also created a camp/month/year identifier column ("cmy") that was especially useful for joining different tables since most of the data were monthly. All of these changes were very much applicable for the rest of the questions as well and the "cmy" variable was consistently used here and after.

Findings: For the above map, the camp name and rounded figure of usage per person (total usage for the month divided by the camp population) can be seen by clicking on the corresponding circle. The map can be zoomed in with scrolling and moved around with drag-and-drop.

One of the observations I can make is that the usage per person seems to be a lot more spread out with Greek camps as opposed to Serbian camps. Strangely enough, this is pretty consistent with my data aggregation experience with the data from the two countries - the UNHCR site profiles that I acquired for Serbian camps were in a consistent format compared to the Greek camps and the representative of the UNHCR data management team for Serbian camps also was more responsive to my inquiries. Greek camps also seem to have more varied types of authority than the Serbian camps so protocol for services offered to the camp residents might not be as standardized.

But the most interesting observation that I noticed is that for some reason, camps in the North seem to have a lot more usage per person than those in the South. Perhaps, camps that are in islands are managed differently? This also might be due to their proximity to the African continent and maybe a lot of these camps in the South are dealing more with African migrants who tend to be processed differently than those from the Middle East.

2. How well can we predict future data usage trends based on forecasting?

I will build an ARIMA forecasting model for the month of February 2018 and see how well that predicts trends for March 2018. For this, I will be consulting from this page (https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b) and look at how total usage changes over time. The time-series will be plotted with sum of total usage across all camps by date.

With the usage trend, I expect it to be steady and I would also be interested to see how well this model would do in forecasting.

Variables used - country (string; parsed from the camp file name in column in nearly every csv), date (datetime; from hourly_usage_per_camp), total_usage (float; from hourly_usage_per_camp), population (integer; from camp_demographics)

Let's first aggregate the camps that I have data for in both February and March and exclude those for which I do not.

In [273]:
```
hourly_usage['monthyear'] = hourly_usage['monthyear'].apply(str).apply(lambda
x: x.zfill(6))
hourly_usage.head()
```

Out[273]:

	campfilename	monthyear	date	time	total_usage_kb	month	year
0	Andravida_GRE_004	042016	4/1/2016	0:00:00	0.0	4	2016
1	Andravida_GRE_004	042016	5/1/2016	0:00:00	0.0	4	2016
2	Andravida_GRE_004	102016	10/1/2016	0:00:00	0.0	10	2016
3	Andravida_GRE_004	102016	11/1/2016	0:00:00	0.0	10	2016
4	Drama_GRE_011	042016	4/1/2016	0:00:00	0.0	4	2016

In [297]:
```
feb_mar = hourly_usage[(hourly_usage['monthyear']=='022018')| (hourly_usage['m
onthyear']=='032018')]
feb_mar = feb_mar.groupby('campfilename').monthyear.unique().to_dict()
fm_camps = []
for key in feb_mar:
    if len(feb_mar[key])==2:
        fm_camps.append(key)
len(fm_camps)
```

Out[297]: 24

Now I have a list of camps for which I have both February and March data. Let's start filtering the data by camps and time.

In [298]:
```
hourly_usage.head()
```

Out[298]:

	campfilename	monthyear	date	time	total_usage_kb	month	year
0	Andravida_GRE_004	042016	4/1/2016	0:00:00	0.0	4	2016
1	Andravida_GRE_004	042016	5/1/2016	0:00:00	0.0	4	2016
2	Andravida_GRE_004	102016	10/1/2016	0:00:00	0.0	10	2016
3	Andravida_GRE_004	102016	11/1/2016	0:00:00	0.0	10	2016
4	Drama_GRE_011	042016	4/1/2016	0:00:00	0.0	4	2016

In [299]:
```
fm_hourly = hourly_usage[hourly_usage['campfilename'].isin(fm_camps)]
fm_hourly = fm_hourly[(fm_hourly['monthyear']=='022018')| (fm_hourly['monthyea
r']=='032018')]
print(len(fm_hourly.campfilename.unique()))
print(fm_hourly.monthyear.unique())
```

```
24
['022018' '032018']
```

Now we can group the dataframe by day.

In [300]:
```
df2_grouped = fm_hourly.groupby(['monthyear','date'])['total_usage_kb'].sum()
df2 = df2_grouped.to_frame().reset_index()
df2[df2['total_usage_kb']!=0]
```

Out[300]:

	monthyear	date	total_usage_kb
1	022018	1/31/2018	1.662496e+06
2	022018	2/1/2018	1.858983e+06
3	022018	2/10/2018	1.987081e+06
4	022018	2/11/2018	1.980299e+06
5	022018	2/12/2018	1.824920e+06
6	022018	2/13/2018	1.881671e+06
7	022018	2/14/2018	1.884925e+06
8	022018	2/15/2018	1.893677e+06
9	022018	2/16/2018	1.875804e+06
10	022018	2/17/2018	1.922665e+06
11	022018	2/18/2018	1.983400e+06
12	022018	2/19/2018	1.976959e+06
13	022018	2/2/2018	1.877145e+06
14	022018	2/20/2018	1.965629e+06
15	022018	2/21/2018	1.937775e+06
16	022018	2/22/2018	1.991524e+06
17	022018	2/23/2018	1.923024e+06
18	022018	2/24/2018	2.077448e+06
19	022018	2/25/2018	2.115663e+06
20	022018	2/26/2018	1.998804e+06
21	022018	2/27/2018	1.934459e+06
22	022018	2/28/2018	2.176564e+05
23	022018	2/3/2018	1.794703e+06
24	022018	2/4/2018	1.778230e+06
25	022018	2/5/2018	1.729711e+06
26	022018	2/6/2018	1.821409e+06
27	022018	2/7/2017	2.048000e+01
28	022018	2/7/2018	1.867938e+06
29	022018	2/8/2018	1.949845e+06
30	022018	2/9/2018	1.912046e+06
...	...	...	...
34	032018	3/11/2018	1.858486e+06

	monthyear	date	total_usage_kb
35	032018	3/12/2018	1.803805e+06
36	032018	3/13/2018	1.843689e+06
37	032018	3/14/2018	1.898183e+06
38	032018	3/15/2018	1.948959e+06
39	032018	3/16/2018	1.929237e+06
40	032018	3/17/2018	1.884949e+06
41	032018	3/18/2018	2.003985e+06
42	032018	3/19/2018	1.920007e+06
43	032018	3/2/2018	1.950143e+06
44	032018	3/20/2018	1.606624e+06
45	032018	3/21/2018	1.965342e+06
46	032018	3/22/2018	2.086148e+06
47	032018	3/23/2018	2.050380e+06
48	032018	3/24/2018	2.044737e+06
49	032018	3/25/2018	2.083242e+06
50	032018	3/26/2018	1.996087e+06
51	032018	3/27/2018	1.927217e+06
52	032018	3/28/2018	1.978685e+06
53	032018	3/29/2018	1.953745e+06
54	032018	3/3/2018	1.921030e+06
55	032018	3/30/2018	1.909963e+06
56	032018	3/31/2018	1.936148e+06
57	032018	3/4/2018	1.952326e+06
58	032018	3/5/2018	1.895900e+06
59	032018	3/6/2018	1.903102e+06
60	032018	3/7/2018	1.826691e+06
61	032018	3/8/2018	1.896843e+06
62	032018	3/9/2018	1.807755e+06
63	032018	4/1/2018	2.002433e+05

63 rows × 3 columns

I am just realizing that there are dates for which there is 0kB because they are the outer bound dates. I am going to get rid of the first and last dates of the data and interpolate the beginning and end each month that are in the middle of the data, since this was largely due to failure in downloading the data correctly.

In [301]:
```
df2 = df2[df2['total_usage_kb']>0]
len(df2)
```

Out[301]: 63

In [302]:
```
df2['date'] = pd.to_datetime(df2.date)
df2 = df2.sort_values(by='date')
df2
```

Out[302]:

	monthyear	date	total_usage_kb
27	022018	2017-02-07	2.048000e+01
1	022018	2018-01-31	1.662496e+06
2	022018	2018-02-01	1.858983e+06
13	022018	2018-02-02	1.877145e+06
23	022018	2018-02-03	1.794703e+06
24	022018	2018-02-04	1.778230e+06
25	022018	2018-02-05	1.729711e+06
26	022018	2018-02-06	1.821409e+06
28	022018	2018-02-07	1.867938e+06
29	022018	2018-02-08	1.949845e+06
30	022018	2018-02-09	1.912046e+06
3	022018	2018-02-10	1.987081e+06
4	022018	2018-02-11	1.980299e+06
5	022018	2018-02-12	1.824920e+06
6	022018	2018-02-13	1.881671e+06
7	022018	2018-02-14	1.884925e+06
8	022018	2018-02-15	1.893677e+06
9	022018	2018-02-16	1.875804e+06
10	022018	2018-02-17	1.922665e+06
11	022018	2018-02-18	1.983400e+06
12	022018	2018-02-19	1.976959e+06
14	022018	2018-02-20	1.965629e+06
15	022018	2018-02-21	1.937775e+06
16	022018	2018-02-22	1.991524e+06
17	022018	2018-02-23	1.923024e+06
18	022018	2018-02-24	2.077448e+06
19	022018	2018-02-25	2.115663e+06
20	022018	2018-02-26	1.998804e+06
21	022018	2018-02-27	1.934459e+06
22	022018	2018-02-28	2.176564e+05
...	...	...	...
54	032018	2018-03-03	1.921030e+06

	monthyear	date	total_usage_kb
57	032018	2018-03-04	1.952326e+06
58	032018	2018-03-05	1.895900e+06
59	032018	2018-03-06	1.903102e+06
60	032018	2018-03-07	1.826691e+06
61	032018	2018-03-08	1.896843e+06
62	032018	2018-03-09	1.807755e+06
33	032018	2018-03-10	1.877782e+06
34	032018	2018-03-11	1.858486e+06
35	032018	2018-03-12	1.803805e+06
36	032018	2018-03-13	1.843689e+06
37	032018	2018-03-14	1.898183e+06
38	032018	2018-03-15	1.948959e+06
39	032018	2018-03-16	1.929237e+06
40	032018	2018-03-17	1.884949e+06
41	032018	2018-03-18	2.003985e+06
42	032018	2018-03-19	1.920007e+06
44	032018	2018-03-20	1.606624e+06
45	032018	2018-03-21	1.965342e+06
46	032018	2018-03-22	2.086148e+06
47	032018	2018-03-23	2.050380e+06
48	032018	2018-03-24	2.044737e+06
49	032018	2018-03-25	2.083242e+06
50	032018	2018-03-26	1.996087e+06
51	032018	2018-03-27	1.927217e+06
52	032018	2018-03-28	1.978685e+06
53	032018	2018-03-29	1.953745e+06
55	032018	2018-03-30	1.909963e+06
56	032018	2018-03-31	1.936148e+06
63	032018	2018-04-01	2.002433e+05

63 rows × 3 columns

In [442]:
```
df2_clean = df2[(df2['date']!='2017-02-07')]
df2_clean = df2_clean[(df2_clean['date']!='2018-04-01')]
df2_feb = df2_clean[df2_clean['monthyear']=='022018']
df2_mar = df2_clean[df2_clean['monthyear']=='032018']
df2_clean
```

Out[442]:

	monthyear	date	total_usage_kb
1	022018	2018-01-31	1.662496e+06
2	022018	2018-02-01	1.858983e+06
13	022018	2018-02-02	1.877145e+06
23	022018	2018-02-03	1.794703e+06
24	022018	2018-02-04	1.778230e+06
25	022018	2018-02-05	1.729711e+06
26	022018	2018-02-06	1.821409e+06
28	022018	2018-02-07	1.867938e+06
29	022018	2018-02-08	1.949845e+06
30	022018	2018-02-09	1.912046e+06
3	022018	2018-02-10	1.987081e+06
4	022018	2018-02-11	1.980299e+06
5	022018	2018-02-12	1.824920e+06
6	022018	2018-02-13	1.881671e+06
7	022018	2018-02-14	1.884925e+06
8	022018	2018-02-15	1.893677e+06
9	022018	2018-02-16	1.875804e+06
10	022018	2018-02-17	1.922665e+06
11	022018	2018-02-18	1.983400e+06
12	022018	2018-02-19	1.976959e+06
14	022018	2018-02-20	1.965629e+06
15	022018	2018-02-21	1.937775e+06
16	022018	2018-02-22	1.991524e+06
17	022018	2018-02-23	1.923024e+06
18	022018	2018-02-24	2.077448e+06
19	022018	2018-02-25	2.115663e+06
20	022018	2018-02-26	1.998804e+06
21	022018	2018-02-27	1.934459e+06
22	022018	2018-02-28	2.176564e+05
32	032018	2018-03-01	2.083698e+06
...	...	...	...
43	032018	2018-03-02	1.950143e+06

	monthyear	date	total_usage_kb
54	032018	2018-03-03	1.921030e+06
57	032018	2018-03-04	1.952326e+06
58	032018	2018-03-05	1.895900e+06
59	032018	2018-03-06	1.903102e+06
60	032018	2018-03-07	1.826691e+06
61	032018	2018-03-08	1.896843e+06
62	032018	2018-03-09	1.807755e+06
33	032018	2018-03-10	1.877782e+06
34	032018	2018-03-11	1.858486e+06
35	032018	2018-03-12	1.803805e+06
36	032018	2018-03-13	1.843689e+06
37	032018	2018-03-14	1.898183e+06
38	032018	2018-03-15	1.948959e+06
39	032018	2018-03-16	1.929237e+06
40	032018	2018-03-17	1.884949e+06
41	032018	2018-03-18	2.003985e+06
42	032018	2018-03-19	1.920007e+06
44	032018	2018-03-20	1.606624e+06
45	032018	2018-03-21	1.965342e+06
46	032018	2018-03-22	2.086148e+06
47	032018	2018-03-23	2.050380e+06
48	032018	2018-03-24	2.044737e+06
49	032018	2018-03-25	2.083242e+06
50	032018	2018-03-26	1.996087e+06
51	032018	2018-03-27	1.927217e+06
52	032018	2018-03-28	1.978685e+06
53	032018	2018-03-29	1.953745e+06
55	032018	2018-03-30	1.909963e+06
56	032018	2018-03-31	1.936148e+06

61 rows × 3 columns

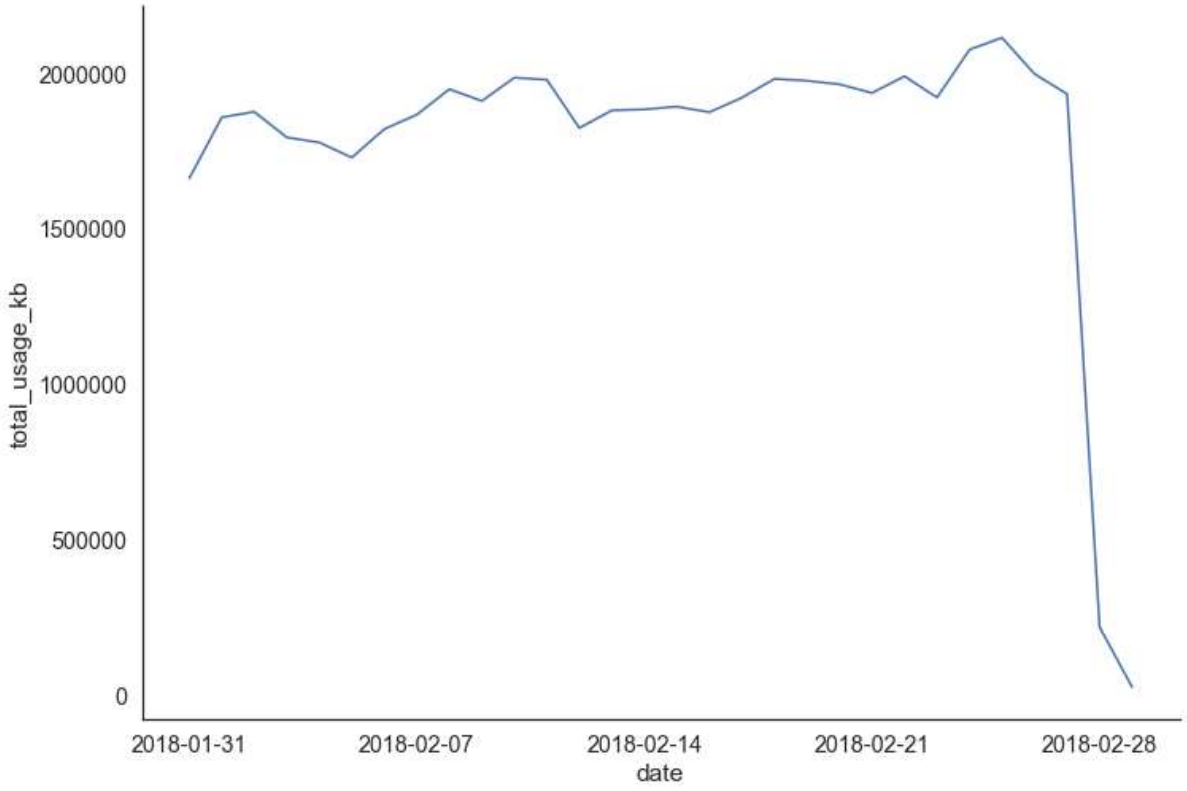
In [441]:
```
df2_feb.iloc[-1,-1] = np.nan
df2_feb = df2_feb.interpolate()
df2_mar
```

Out[441]:

	monthyear	date	total_usage_kb
32	032018	2018-03-01	2083698.297
43	032018	2018-03-02	1950142.507
54	032018	2018-03-03	1921029.684
57	032018	2018-03-04	1952325.887
58	032018	2018-03-05	1895899.917
59	032018	2018-03-06	1903101.943
60	032018	2018-03-07	1826690.816
61	032018	2018-03-08	1896843.259
62	032018	2018-03-09	1807754.940
33	032018	2018-03-10	1877782.304
34	032018	2018-03-11	1858486.381
35	032018	2018-03-12	1803804.655
36	032018	2018-03-13	1843689.474
37	032018	2018-03-14	1898183.487
38	032018	2018-03-15	1948959.303
39	032018	2018-03-16	1929236.508
40	032018	2018-03-17	1884949.324
41	032018	2018-03-18	2003984.729
42	032018	2018-03-19	1920006.666
44	032018	2018-03-20	1606623.982
45	032018	2018-03-21	1965341.662
46	032018	2018-03-22	2086147.742
47	032018	2018-03-23	2050379.868
48	032018	2018-03-24	2044736.943
49	032018	2018-03-25	2083241.746
50	032018	2018-03-26	1996087.214
51	032018	2018-03-27	1927216.553
52	032018	2018-03-28	1978685.150
53	032018	2018-03-29	1953745.204
55	032018	2018-03-30	1909962.622
56	032018	2018-03-31	1936147.610

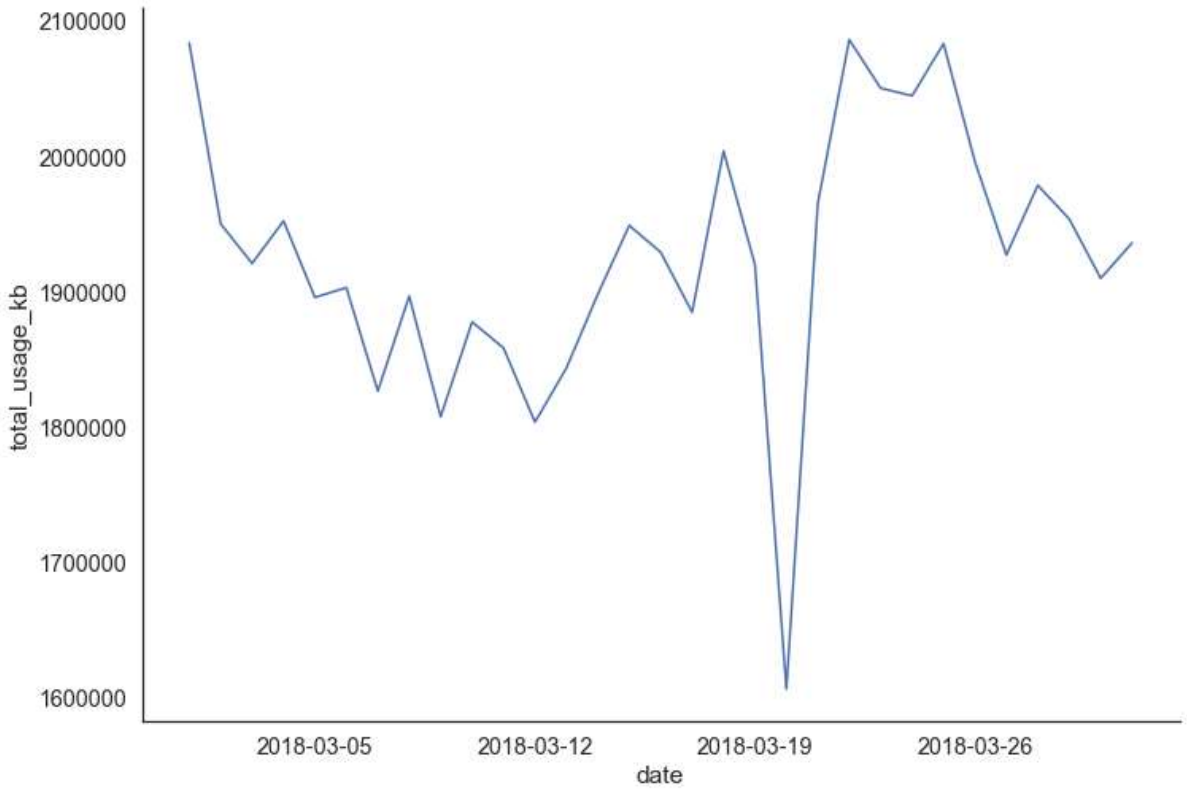
Now let's plot the February data.

In [443]:
```
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
time = sns.lineplot(x="date", y="total_usage_kb", data=df2_feb,ax=ax)
sns.despine()
```



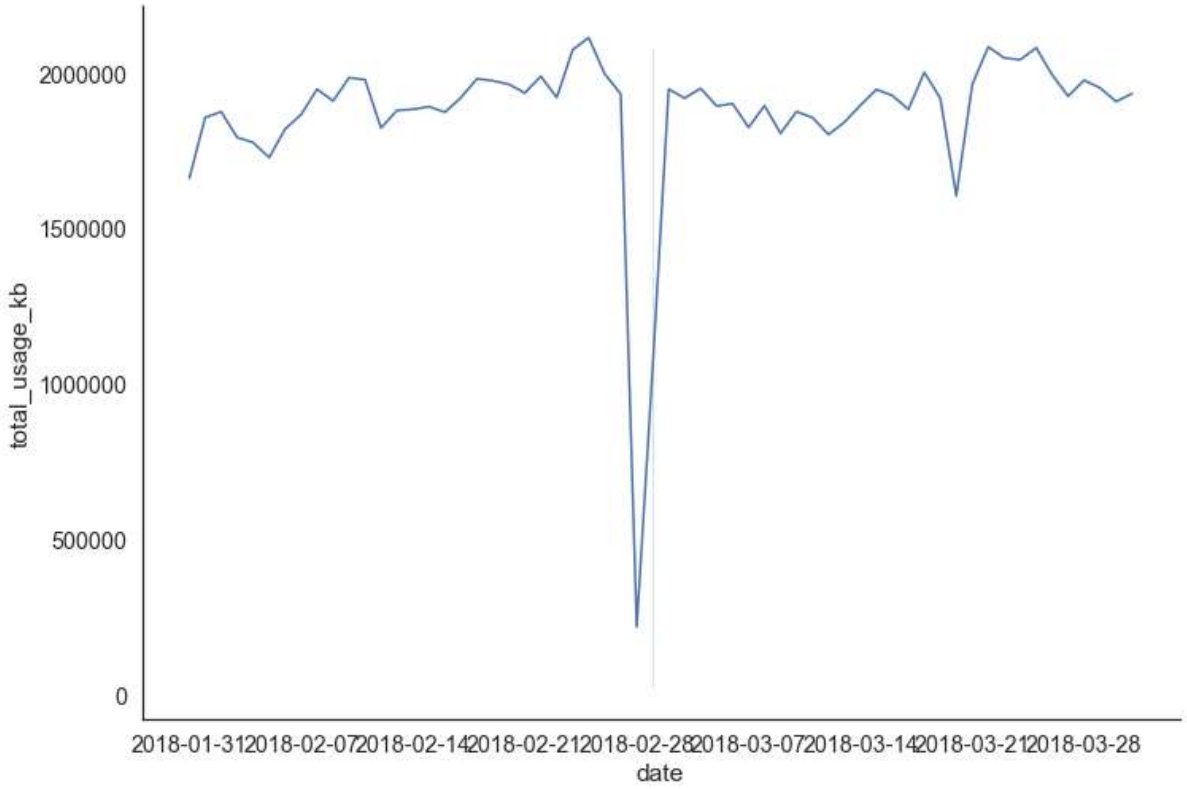
And here's the March plot.

In [444]:
```
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
time = sns.lineplot(x="date", y="total_usage_kb", data=df2_mar,ax=ax)
sns.despine()
```



And here's both together!

In [445]:
```
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
time = sns.lineplot(x="date", y="total_usage_kb",data=df2_clean,ax=ax)
sns.despine()
```



This time-series plot is very interesting. There seems to be a huge dip in mid-March which might have been due to outages in multiple camps. It also seems that in february, data usage seemed to be increasing, but this is not reflected in March at all in which usage trends seem to be all over the place.

Now I am going to see if the February data can be used to forecast the March data and how that compares to the actual data by fitting the ARIMA model.

In [446]:
```
import statsmodels.api as sm

y = df2_clean.set_index('date').drop(['monthyear'],axis=1)
y
```

Out[446]:

	total_usage_kb
date	
2018-01-31	1.662496e+06
2018-02-01	1.858983e+06
2018-02-02	1.877145e+06
2018-02-03	1.794703e+06
2018-02-04	1.778230e+06
2018-02-05	1.729711e+06
2018-02-06	1.821409e+06
2018-02-07	1.867938e+06
2018-02-08	1.949845e+06
2018-02-09	1.912046e+06
2018-02-10	1.987081e+06
2018-02-11	1.980299e+06
2018-02-12	1.824920e+06
2018-02-13	1.881671e+06
2018-02-14	1.884925e+06
2018-02-15	1.893677e+06
2018-02-16	1.875804e+06
2018-02-17	1.922665e+06
2018-02-18	1.983400e+06
2018-02-19	1.976959e+06
2018-02-20	1.965629e+06
2018-02-21	1.937775e+06
2018-02-22	1.991524e+06
2018-02-23	1.923024e+06
2018-02-24	2.077448e+06
2018-02-25	2.115663e+06
2018-02-26	1.998804e+06
2018-02-27	1.934459e+06
2018-02-28	2.176564e+05
2018-03-01	2.083698e+06
...	...

	total_usage_kb
date	
2018-03-02	1.950143e+06
2018-03-03	1.921030e+06
2018-03-04	1.952326e+06
2018-03-05	1.895900e+06
2018-03-06	1.903102e+06
2018-03-07	1.826691e+06
2018-03-08	1.896843e+06
2018-03-09	1.807755e+06
2018-03-10	1.877782e+06
2018-03-11	1.858486e+06
2018-03-12	1.803805e+06
2018-03-13	1.843689e+06
2018-03-14	1.898183e+06
2018-03-15	1.948959e+06
2018-03-16	1.929237e+06
2018-03-17	1.884949e+06
2018-03-18	2.003985e+06
2018-03-19	1.920007e+06
2018-03-20	1.606624e+06
2018-03-21	1.965342e+06
2018-03-22	2.086148e+06
2018-03-23	2.050380e+06
2018-03-24	2.044737e+06
2018-03-25	2.083242e+06
2018-03-26	1.996087e+06
2018-03-27	1.927217e+06
2018-03-28	1.978685e+06
2018-03-29	1.953745e+06
2018-03-30	1.909963e+06
2018-03-31	1.936148e+06

61 rows × 1 columns

In [447]:
```
import itertools

p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q
))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [448]:
```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, result
s.aic))
        except:
            continue
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1906.6679241328818
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1497.777096966032
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1404.891450587184
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1278.0048850276821
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1433.32917126282
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1417.257562987017
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:1084.4495861080022
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:1341.6893565882983
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1843.688507415728
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1463.1699115520166
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:1377.0070534766248
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1025.6825891658036
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:1516.3202545831634
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1456.6452908104263
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:1086.4495998774385
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1025.4188256104221
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:1718.2003911300706
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:2288.280722136659
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:1414.0494964128427
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:3037.669498507287
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:1409.9193471358203
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:2289.2462055082924
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:1083.9497492775765
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:3064.753136036501
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:1654.259847961917
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:1321.6041409783409
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:1357.339545447151
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:3049.220572312586
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:1377.0302296104333
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:1320.5681186203606
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:1065.8338377648597
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:3076.01154295962
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:1747.4916117729326
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:1412.0854992084576
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:1404.9326962121363
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:1054.545873615447
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:1407.027084943557
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:1407.150248368276
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:1057.6020257856626
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:1054.1105759087914
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:1681.5666882434411
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:1348.288375383561
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:1376.2619369060403
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:1025.3903444396742
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:1374.932164220946
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:1348.9553580553893
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:1057.3644332667855
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:1025.033795505561
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:1672.465023997621
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:1346.1692516863332
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:1374.9044384029598
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:3120.142508975692
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:1346.1692641603674
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:1348.1691229827068
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:1027.699939008086
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:3147.6626258182278
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:1645.7595488210284
```

```
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:1319.3451557558392
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:1347.4961849326505
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:3034.653284674222
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:1346.660241753001
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:1321.269998966749
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:1029.1723122693886
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:3061.444255321256
```

Above, I iterated over all the possible parameters to see which combination would be the best. Judging by the AIC value, which measures how well a specific model fits, I chose the following parameter - ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1025.6825891658036.

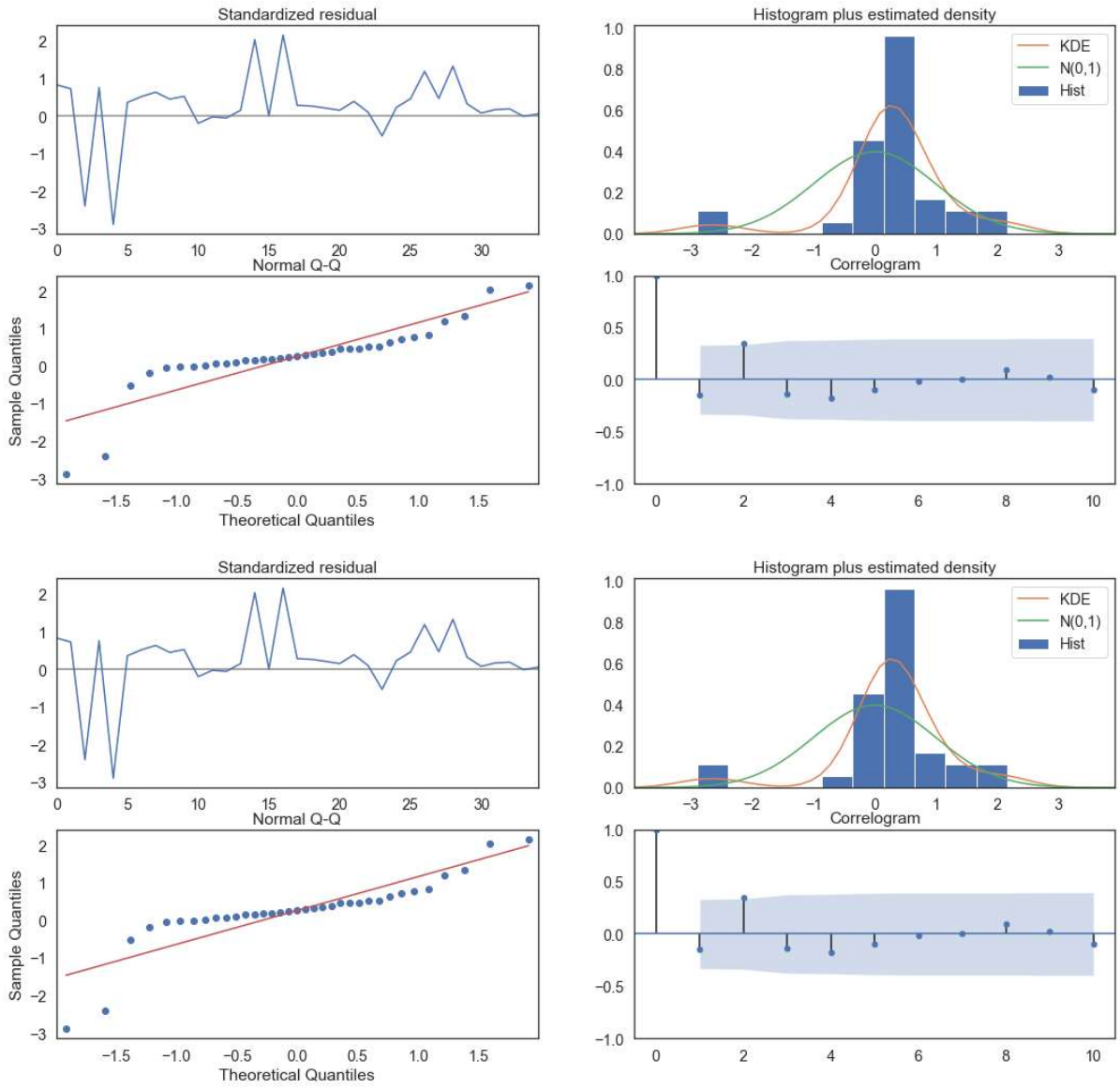
In [449]:
```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(0, 0, 1),
                                seasonal_order=(0, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.0723      0.322     -0.225      0.822      -0.704       0.559
ma.S.L12      -0.3986      0.089     -4.496      0.000      -0.572      -0.225
sigma2      2.976e+11   1.54e-13   1.93e+24      0.000    2.98e+11    2.98e+11
==============================================================================
```

Now that we fitted the model, we'll check for any anomalies in our model to make sure that no assumptions of ARIMA model has been violated.

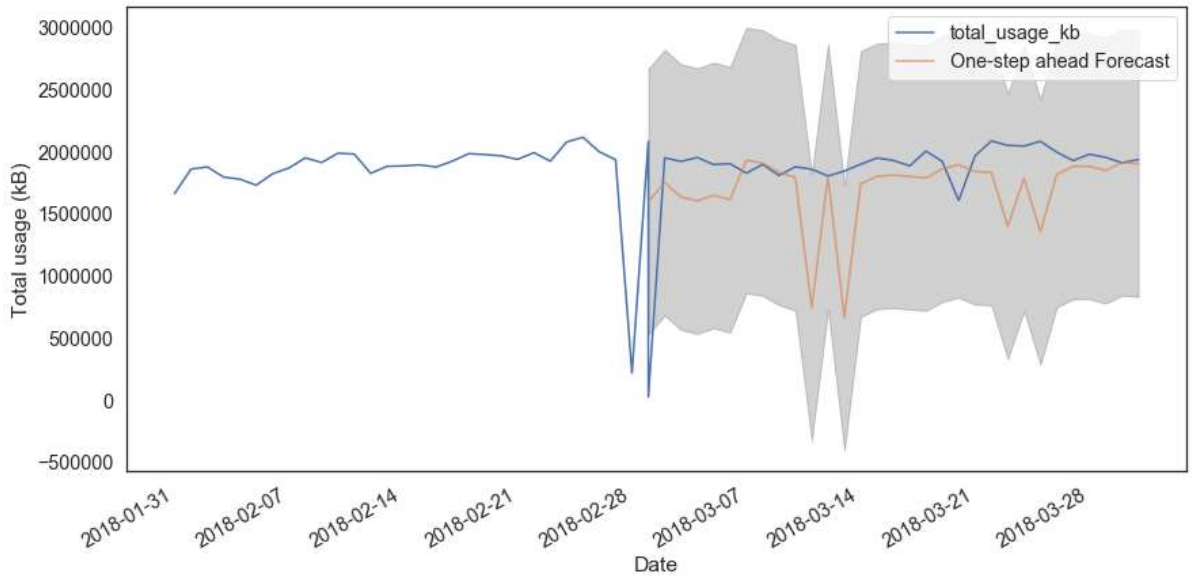
In [451]:
```
results.plot_diagnostics()
```

Out[451]:



The key thing to check here is to make sure that our residuals are normally distributed and this seems close. Now let's actually compare the forecasted values to the actual.

In [452]:
```
pred = results.get_prediction(start=30, dynamic=False)
pred_ci = pred.conf_int()

ax = y.plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7, fig
size=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Total usage (kB)')
plt.legend()
plt.show()
```



In [455]:
```
y_forecasted = pred.predicted_mean
y['forecasted'] = y_forecasted
forecasted = y['2018-03-01':]
forecasted['squared_error'] = (forecasted.total_usage_kb-forecasted.forecasted
)**2
forecasted.squared_error.mean()
np.sqrt(forecasted.squared_error.mean())
```

Out[455]: 471573.2199255527

Procedure: For this question, I made sure to only include camps for which I had both February and March data, first by filtering by the months, and then counting the amount of unique months for each camp from which I was able to filter further into camps that had two. I then aggregated the data by date and separated the dataframe by month, just to visualize and see how they differed.

I first visualized the time series, using Seaborn, then I embarked on my first execution of an ARIMA model to see if I accurately forecast data usage in March based on February. This involved using the StatsModel package to tune my model by iterating through different combinations of parameters, then actually fitting the model. After checking whether my model was meeting all the assumptions of an ARIMA model, I actually plotted the forecasted and actual usage values using just a regular MatplotLib plot.

There is a big exclaimer in that the data was manually downloaded by month and the data source sometimes included lower and upper bounds that always had 0 usage. I tried to replace this with some interpolated data, but I do not know how successful this actually was.

Findings: The forecasting model is a bit off and this might be due to the values in between the two months. Even after trying to interpolate the model, I still have these extreme dips that is affecting the model. Even the root squared mean is very big. I will have to make sure to minimize these errors if I want to try to use forecasting in the future.

Otherwise, the usage does not really seem to be increasing or decreasing. There is a significant dip in March that I would like to further investigate in the future.

Moria Camp, Greece - Source: Getty Images

3. Can we predict whether a camp is overcapacitated at a given month by looking at its demographic data?

I am going to be using the Sklearn modules to build classification models such as random forest and naive-bayes classifiers to find demographic features (e.g. % of women, % of Syrians) that are the strongest determinants of whether a camp is overcapacitated or not. I will also show a bar graph showing the rankings of importance of features as well as a color scatter plot illustrating how successful my models are.

After consulting with someone who has worked in one of these camps, I am anticipating that the nationality breakdown might have some impact on capacity. My reference stated that camps with poorer refugees, such as those from Afghanistan, tend to be more poorly administered since they are more likely to have less favorable refugee status.

Variables used - overcapacity or not (binary; from camp_demographics.csv), proportion of males/females/children (float; from camp_demographics.csv), and proportion of different nationalities (float; from refugee_nationalities.csv, only looking at Syrian, Afghan, Iraqi, and Pakistani statistics since Serbian camps only have information for those)

Let's create another dataframe with all the necessary variables. Many of the variables are already in the camp_dem dataframe, so I'm just going to drop the unnecessary variables.

In [38]:
```
camp_dem.head()
```

Out[38]:

	campfilename	month	year	camppopulation	campcapacity	overcapacity	adu
0	Adasevci_SRB_010	1	2018	327	450	0.0	0.45
1	Bujanovac_SRB_001	1	2018	129	220	0.0	0.35
2	Divljana_SRB_002	1	2018	83	280	0.0	0.27
3	Kikinda_SRB_007	1	2018	162	240	0.0	0.33
4	Krnjaca_SRB_003	1	2018	643	1000	0.0	0.42

In [39]:
```
df3_pre = camp_dem.drop(['camppopulation','campcapacity'],axis=1)
df3_pre['cmy'] = df3_pre.campfilename + df3_pre.month.apply(str).apply(lambda
x: x.zfill(2)) + df3_pre.year.apply(str)
df3_pre = df3_pre.drop(['month','year'],axis=1)
df3_pre.head()
```

Out[39]:

	campfilename	overcapacity	adultmales	adutfemales	children	
0	Adasevci_SRB_010	0.0	0.45	0.19	0.36	Adasevci_SRB_
1	Bujanovac_SRB_001	0.0	0.35	0.24	0.41	Bujanovac_SRI
2	Divljana_SRB_002	0.0	0.27	0.25	0.48	Divljana_SRB_(
3	Kikinda_SRB_007	0.0	0.33	0.18	0.49	Kikinda_SRB_0
4	Krnjaca_SRB_003	0.0	0.42	0.17	0.41	Krnjaca_SRB_0

Now I want to include the nationality breakdown information from nationality dataframe.

In [40]:
```
nationality.head()
```

Out[40]:

	monthyear	campfilename	nationality	proportion
0	22018	Alexandria-Ref_GRE_003	Syrian	0.44
1	22018	Alexandria-Ref_GRE_003	Iraqi	0.25
2	22018	Alexandria-Ref_GRE_003	Pakistani	0.08
3	22018	Alexandria-Ref_GRE_003	Palestinian	0.04
4	22018	Diavata_GRE_008	Syrian	0.38

In [41]:
```
nationality['monthyear'] = nationality['monthyear'].apply(str).apply(lambda x:
 x.zfill(6))
nationality.monthyear.unique()
```

Out[41]:
```
array(['022018', '012017', '062017', '042016', '102016', '032018',
       '072018', '042018', '052018', '062018', '092018', '012018'],
      dtype=object)
```

In [42]:
```
nationality['cmy'] = nationality.campfilename+nationality.monthyear.apply(str)
nationality.head()
```

Out[42]:

	monthyear	campfilename	nationality	proportion	cmy
0	022018	Alexandria-Ref_GRE_003	Syrian	0.44	Alexandria-Ref_GRE_003022018
1	022018	Alexandria-Ref_GRE_003	Iraqi	0.25	Alexandria-Ref_GRE_003022018
2	022018	Alexandria-Ref_GRE_003	Pakistani	0.08	Alexandria-Ref_GRE_003022018
3	022018	Alexandria-Ref_GRE_003	Palestinian	0.04	Alexandria-Ref_GRE_003022018
4	022018	Diavata_GRE_008	Syrian	0.38	Diavata_GRE_008022018

In [43]:
```
exclusive_n = nationality[(nationality['nationality']=='Syrian')|(nationality['nationality']=='Iraqi')|(nationality['nationality']=='Pakistani')|(nationality['nationality']=='Afghan')]
exclusive_n.head()
```

Out[43]:

	monthyear	campfilename	nationality	proportion	cmy
0	022018	Alexandria-Ref_GRE_003	Syrian	0.44	Alexandria-Ref_GRE_003022018
1	022018	Alexandria-Ref_GRE_003	Iraqi	0.25	Alexandria-Ref_GRE_003022018
2	022018	Alexandria-Ref_GRE_003	Pakistani	0.08	Alexandria-Ref_GRE_003022018
4	022018	Diavata_GRE_008	Syrian	0.38	Diavata_GRE_008022018
5	022018	Diavata_GRE_008	Iraqi	0.24	Diavata_GRE_008022018

In [44]:
```
n_pvt = exclusive_n.pivot(index='cmy',columns='nationality',values='proportion').reset_index()
n_pvt.head()
```

Out[44]:

nationality	cmy	Afghan	Iraqi	Pakistani	Syrian
0	Adasevci_SRB_010012018	0.54	0.11	0.12	0.00
1	Adasevci_SRB_010022018	0.55	0.05	0.10	0.06
2	Adasevci_SRB_010032018	0.48	0.06	0.11	0.05
3	Adasevci_SRB_010092018	0.18	0.06	0.05	0.01
4	Alexandria-Ref_GRE_003012017	0.00	0.01	0.00	0.99

In [45]:
```
df3 = pd.merge(n_pvt, df3_pre, how='inner', on='cmy')
df3 = df3.drop(columns=['campfilename'],axis=1)
df3 = df3[list(df3.columns)[:5]+list(df3.columns)[6:]+['overcapacity']]
df3 = df3.fillna(0)
df3.head()
```

Out[45]:

	cmy	Afghan	Iraqi	Pakistani	Syrian	adultmales	adutfemales	c
0	Adasevci_SRB_010012018	0.54	0.11	0.12	0.00	0.45	0.19	(
1	Adasevci_SRB_010022018	0.55	0.05	0.10	0.06	0.49	0.19	(
2	Adasevci_SRB_010032018	0.48	0.06	0.11	0.05	0.55	0.18	(
3	Adasevci_SRB_010092018	0.18	0.06	0.05	0.01	0.73	0.13	(
4	Alexandria-Ref_GRE_003012017	0.00	0.01	0.00	0.99	0.26	0.22	(

Now that we finally have the dataframe organized, let's import all the classification modules and split the dataset into training and testing sets.

In [130]:
```
import scipy as sp
import sklearn as sk
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import sklearn.ensemble as skens
import sklearn.metrics as skmetric
import sklearn.naive_bayes as sknb
import sklearn.tree as sktree
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style='white', color_codes=True, font_scale=1.3)
import sklearn.externals.six as sksix
import IPython.display as ipd
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
from sklearn.model_selection import GridSearchCV

df3_train,df3_test = train_test_split(df3, test_size=0.3)
len(df3_train),len(df3_test)
```

Out[130]: (186, 80)

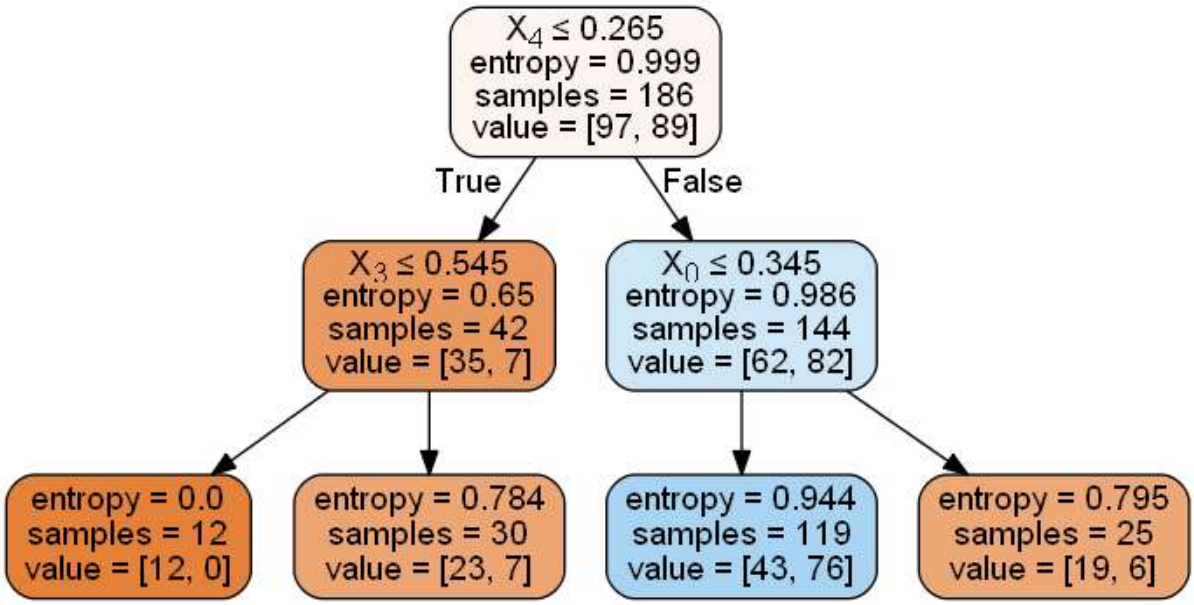
Now I will first build the decision tree classifier, first with 2 maximum divisions (max_depth), than with no limit for divisions. I will then see how well each model did in predicting the capacity status.

In [131]:
```
dt_model = sktree.DecisionTreeClassifier(max_depth=2,
                                         criterion='entropy')
dt_model.fit(df3_train.ix[:,1:8],df3_train.overcapacity)
```

Out[131]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [132]:
```
dot_data = StringIO()
export_graphviz(dt_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[132]:

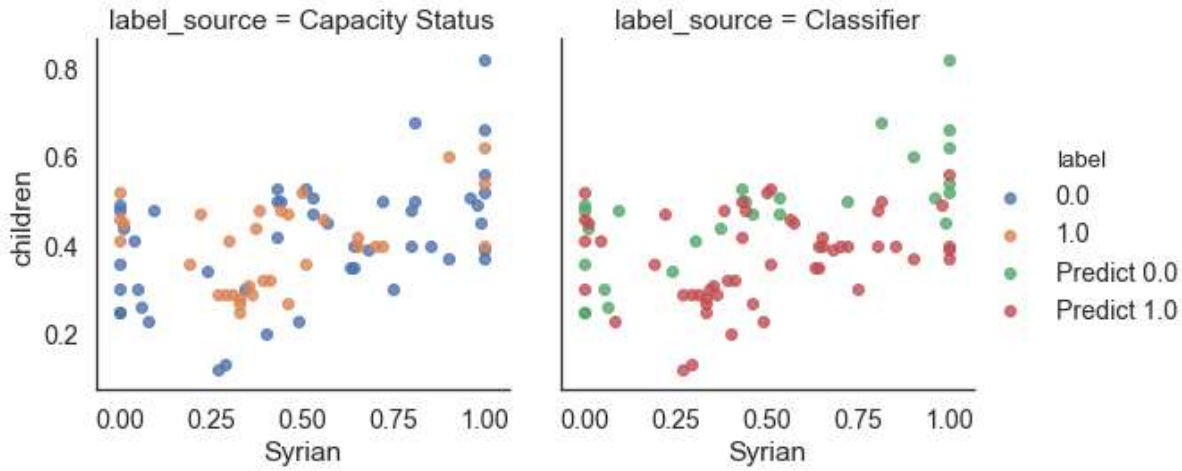


In [133]:
```
predicted_labels = dt_model.predict(df3_test.ix[:,1:8])
df3_test['predicted_label_tree'] = predicted_labels
df3_test.sample(10)
```

Out[133]:

	cmy	Afghan	Iraqi	Pakistani	Syrian	adultmales	adutfemale
185	XXDerveni-Alexil_GRE_007042016	0.00	0.00	0.00	1.00	0.40	0.21
58	Kavala_GRE_024042018	0.19	0.28	0.05	0.43	0.26	0.21
9	Alexandria-Ref_GRE_003052018	0.00	0.28	0.03	0.39	0.44	0.24
77	Krnjaca_SRB_003022018	0.64	0.14	0.08	0.01	0.38	0.18
250	XXSouda-Samns-Purse_GRE_046012017	0.00	0.12	0.00	0.46	0.54	0.19
170	Veria_GRE_054052018	0.00	0.02	0.00	0.68	0.35	0.26
14	Alexandria-Ref_GRE_003102016	0.00	0.00	0.00	1.00	0.31	0.32
183	XXCherso_GRE_005102016	0.00	0.20	0.00	0.80	0.28	0.24
121	Presevo_SRB_006032018	0.37	0.13	0.11	0.06	0.59	0.15
240	XXPikpa-Leros_GRE_038062018	0.00	0.44	0.01	0.53	0.24	0.25

In [134]:
```
def comparePlot(input_frame,real_column,predicted_column):
    df_a = input_frame.copy()
    df_b = input_frame.copy()
    df_a['label_source'] = 'Capacity Status'
    df_b['label_source'] = 'Classifier'
    df_a['label'] = df_a[real_column]
    df_b['label'] = df_b[predicted_column].apply(lambda x: 'Predict %s'%x)
    df_c = pd.concat((df_a, df_b), axis=0, ignore_index=True)
    sns.lmplot(x='Syrian', y='children', col='label_source',
               hue='label', data=df_c, fit_reg=False, size=4);
comparePlot(df3_test,"overcapacity","predicted_label_tree")
```



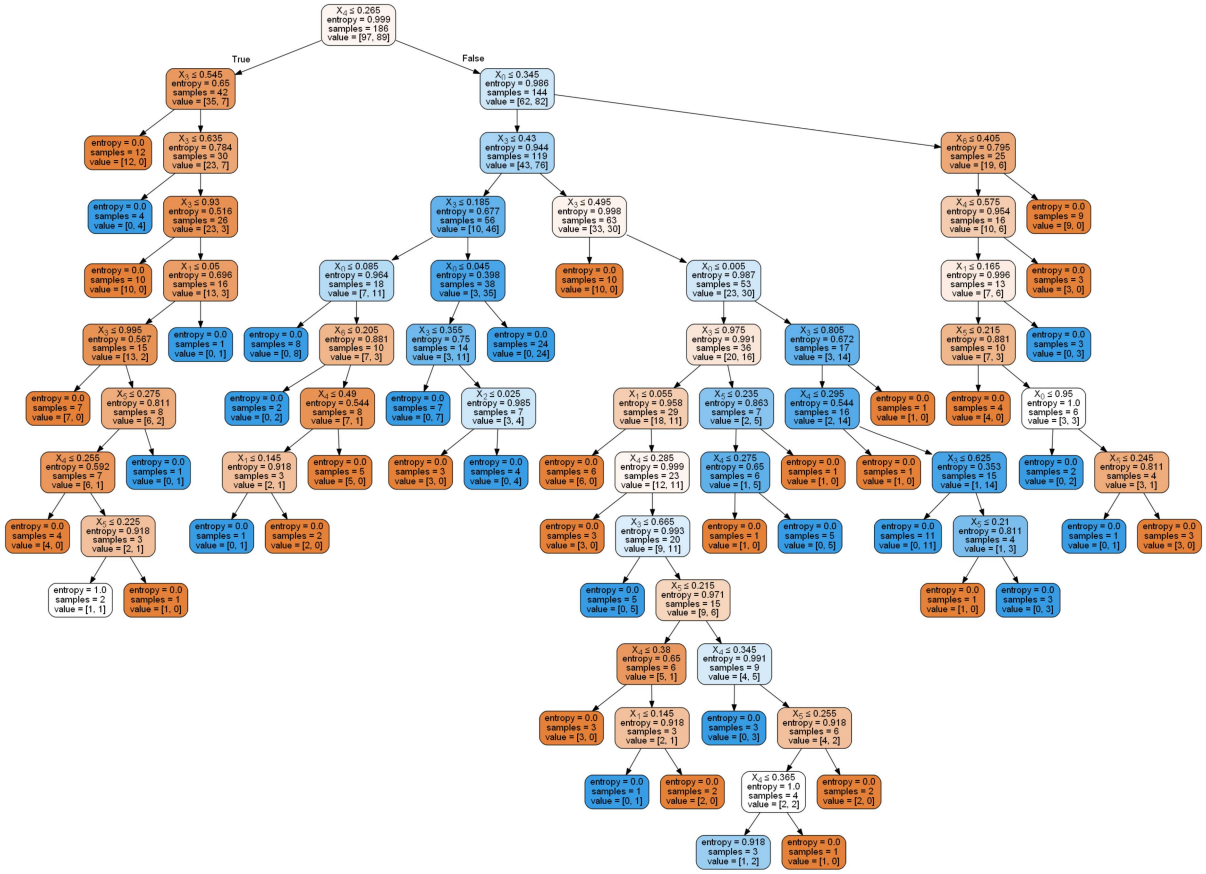
In [135]:
```
skmetric.accuracy_score(df3_test.overcapacity,df3_test.predicted_label_tree)
```

Out[135]: 0.6

It looks like the first model did not do a great job judging by the comparison plot and the accuracy score (perfect score being 1). The plot compares the predicted capacity status on the right to the actual status on the left, but this is not too much of a surprise because this model employs one feature to predict. Let's see if the model improves when we impose no limit.

In [136]:
```
dt_model = sktree.DecisionTreeClassifier(criterion='entropy')
dt_model.fit(df3_train.ix[:,1:8],df3_train.overcapacity)
dot_data = StringIO()
export_graphviz(dt_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[136]:

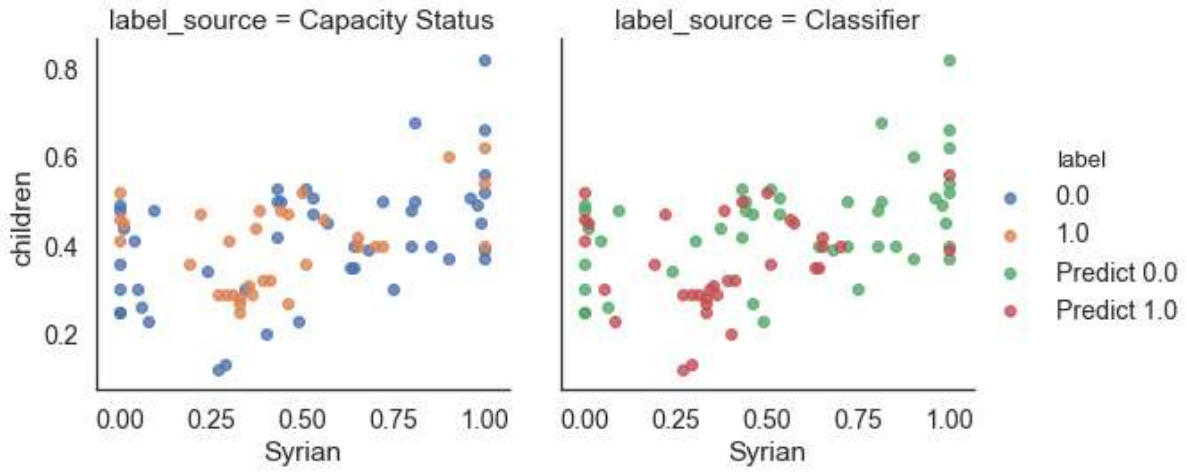


In [137]:
```
predicted_labels = dt_model.predict(df3_test.ix[:,1:8])
df3_test['predicted_label_tree'] = predicted_labels
df3_test.sample(10)
```

Out[137]:

	cmy	Afghan	Iraqi	Pakistani	Syrian	adultmales	adutfema
163	Vathi-Samos_GRE_053072018	0.00	0.22	0.00	0.33	0.49	0.23
51	Kato-Milia_GRE_061022018	0.12	0.13	0.00	0.64	0.39	0.21
42	Divljana_SRB_002032018	0.75	0.24	0.00	0.00	0.24	0.26
244	XXSindos-Frakapore_GRE_043042016	0.00	0.07	0.00	0.90	0.15	0.25
209	XXFilipiada_GRE_019062018	0.34	0.37	0.00	0.22	0.28	0.25
39	Diavata_GRE_008102016	0.12	0.00	0.00	0.81	0.30	0.20
112	Obrenovac_SRB_011022018	0.34	0.00	0.54	0.00	0.97	0.00
129	Ritsona_GRE_040052018	0.00	0.13	0.00	0.72	0.38	0.22
139	Serres_GRE_042092018	0.00	0.99	0.00	0.01	0.28	0.27
106	Nea-Kavala_GRE_034062017	0.00	0.13	0.00	0.34	0.45	0.25

In [138]:
```
comparePlot(df3_test,"overcapacity","predicted_label_tree")
```



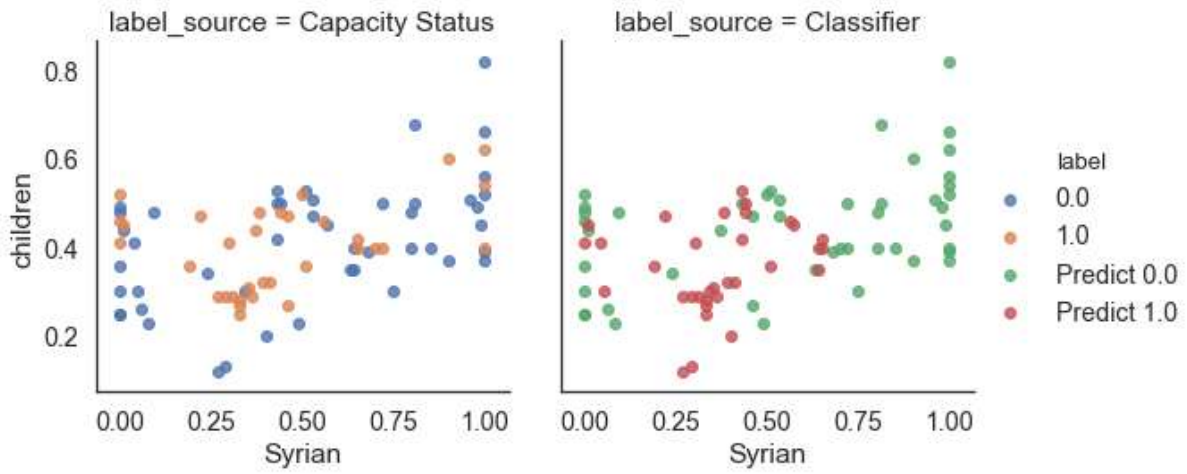
In [139]:
```
skmetric.accuracy_score(df3_test.overcapacity,df3_test.predicted_label_tree)
```

Out[139]: 0.725

The performance has gotten better with no-max_depth model which has a surprising amount of nodes on its decision tree. Even with the amount of nodes, this model is not perfect. Let's try the random forest model and see if it does better. We'll also see which features are more important than others in predicting the capacity status of a camp at a given time.

In [140]:
```
rf_model = skens.RandomForestClassifier(n_estimators=10,oob_score=True, criterion='entropy')
rf_model.fit(df3_train.ix[:,1:8],df3_train.overcapacity)
predicted_labels = rf_model.predict(df3_test.ix[:,1:8])
df3_test['predicted_rf_tree'] = predicted_labels
print(skmetric.accuracy_score(df3_test.overcapacity,df3_test.predicted_label_tree))
comparePlot(df3_test,"overcapacity","predicted_rf_tree")
```

```
0.725
```



In [141]:
```
df3_test.head()
```

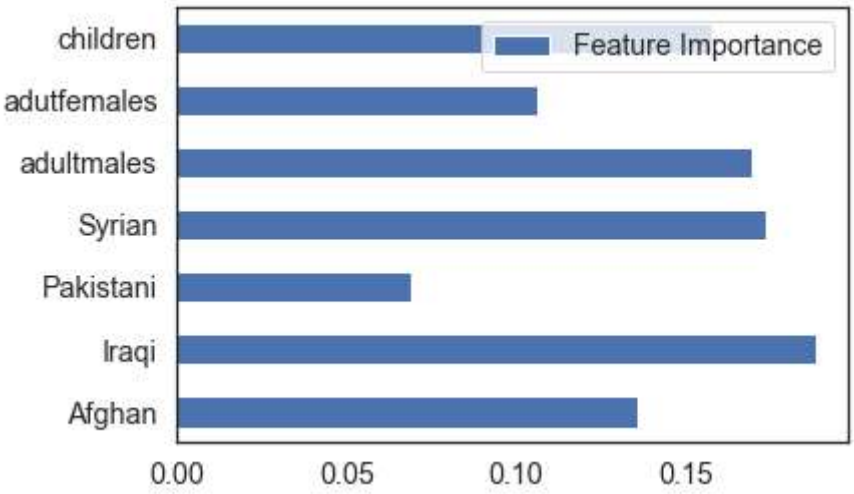
Out[141]:

	cmy	Afghan	Iraqi	Pakistani	Syrian	adultmales	adutfemale
230	XXMalakasa-MC_GRE_012062018	0.52	0.15	0.02	0.24	0.50	0.16
228	XXMalakasa-MC_GRE_012042018	0.64	0.23	0.00	0.05	0.55	0.15
196	XXDoliana_GRE_007072018	0.00	0.42	0.00	0.53	0.22	0.31
201	XXElliniko-Hocky_GRE_016102016	0.98	0.00	0.01	0.00	0.40	0.35
85	Leros-Lepida_GRE_032042018	0.12	0.28	0.00	0.40	0.60	0.20

Let's check the feature importance according to the random forest model.

In [142]:
```
feat_importance = rf_model.feature_importances_
pd.DataFrame({'Feature Importance':feat_importance},
             index=df3_train.columns[1:8]).plot(kind='barh')
```

Out[142]: <matplotlib.axes._subplots.AxesSubplot at 0x1a3edced048>



The proportion of Iraqis seems to be the most important with camps having more Syrians less likely to be overcapacitated, whereas that of Pakistanis is the least, probably because this column had a lot more 0/null values than other columns.

But again, maybe we can improve the model. I will tune the model by cross-validation to get the optimal set of parameters.

In [143]:
```
param_grid = {
                 'n_estimators': [5, 10, 15, 20, 25],
                 'max_depth': [2, 5, 8, 11],
             }
grid_clf = GridSearchCV(rf_model, param_grid, cv=10)
grid_clf.fit(df3_train.ix[:,1:8],df3_train.overcapacity)
new_rf = grid_clf.best_estimator_
new_rf
```

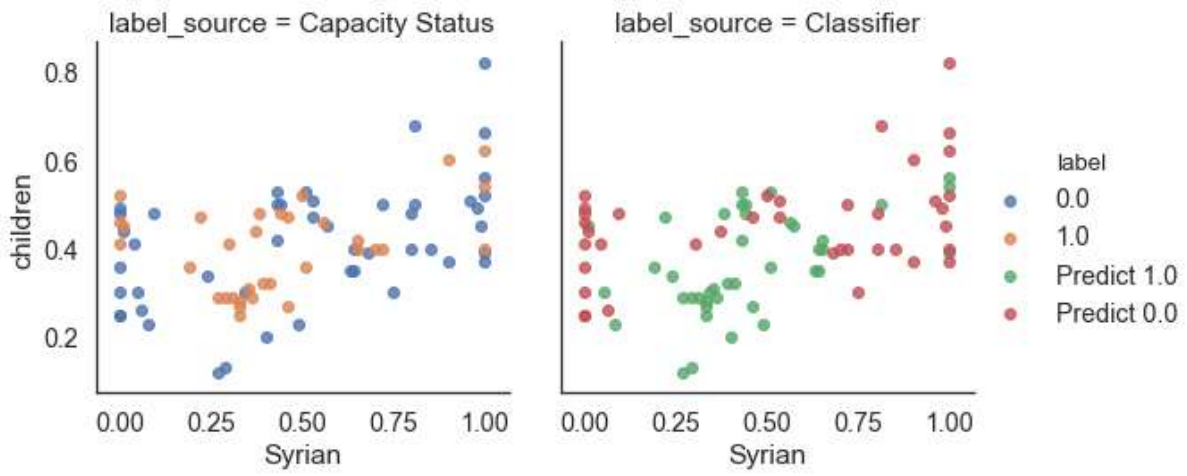
Out[143]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entrop
y',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
            oob_score=True, random_state=None, verbose=0, warm_start=False)
```

In [144]:
```
grid_clf.best_params_
```

Out[144]:
```
{'max_depth': 5, 'n_estimators': 20}
```

In [145]:
```
predicted_labels = new_rf.predict(df3_test.ix[:,1:8])
df3_test['predicted_rf_tree'] = predicted_labels
print(skmetric.accuracy_score(df3_test.overcapacity,df3_test.predicted_label_tree))
comparePlot(df3_test,"overcapacity","predicted_rf_tree")
```

```
0.725
```



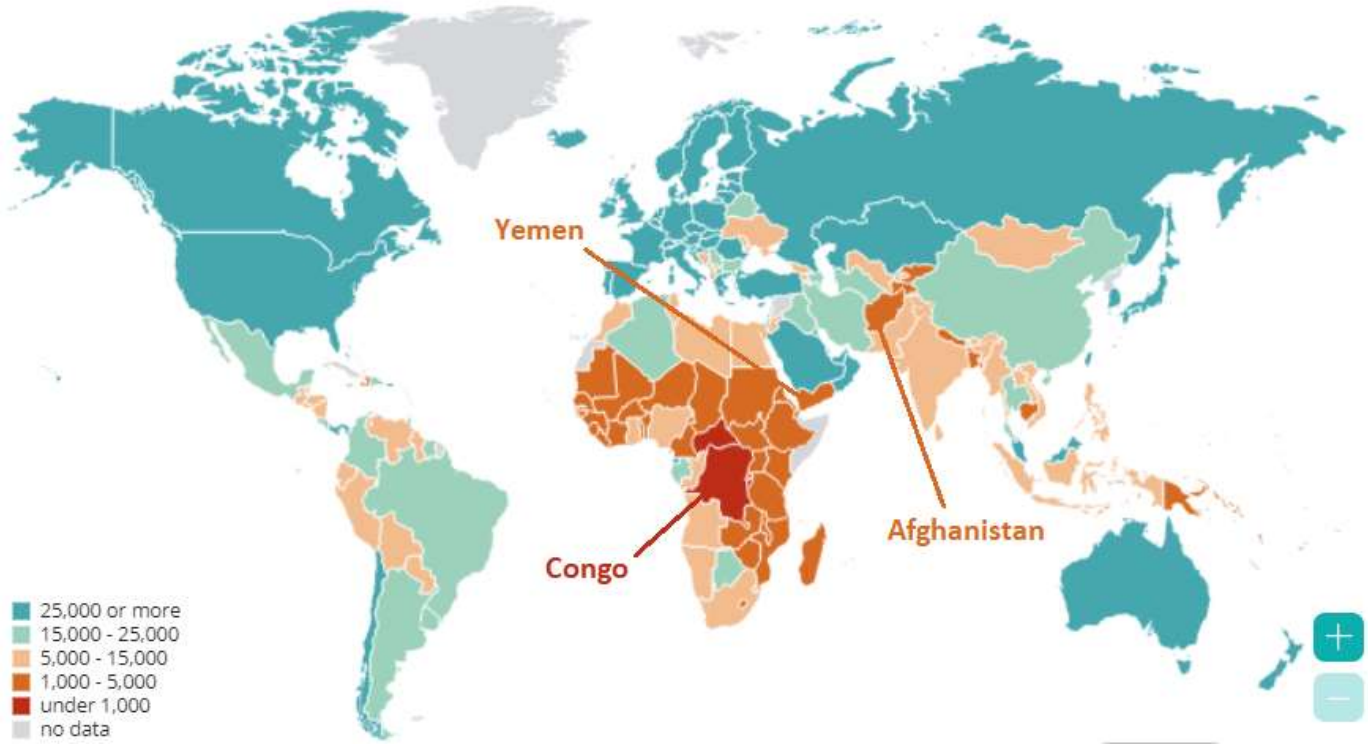
Procedure: I planned to build my ultimate dataframe off of the camp_demographics.csv, so I dropped the columns that were not relevant and made sure I had a "cmy" variable for joining with other tables. I then had to pivot the nationality.csv, since the amount of each nationality was each a feature for my prediction model, and I also dropped columns that I decided not to look at. I finally joined the two tables together to have a table of features and prediction target variable (capacity status).

After dividing my dataset into training and testing sets, I started using the ScikitLearn module with a decision tree classifier for which I visualized the decision trees themselves, as well as colored scatter plots showing predicted and actual capacity statuses of each camp. I then created and tuned a random forest model for which I created the same scatter plots and a bar graph that showed the importance of each feature.

Findings: The decision tree and random forest models ultimately performed equally well. For the decision tree, incorporating more nodes did yield a better accuracy score, but it was a significant increase compared to the amount of more nodes needed to achieve that score. Tuning the random forest model, while slightly changing the predictions, also did not significantly improve the accuracy rate of the model since.

Although this prediction model is not amazing and individually, these features probably would not predict very well, combining these features predicted better than pure guessing (which would yield close to 50/50 result). This kind of modelling applied to more pertinent features would yield powerful results.

More specific findings should be seen amidst the visualizations above.



World map of GDP per capita (see legend; in international dollars) - Source: International Monetary Fund

4. Do camps with more refugees from impoverished nations use less data?

I originally wanted to see the correlation between income/assets and data usage, but I have no such information. What I do have is the nationality data and I want to see there is a correlation between the amount of refugees from some of the most impoverished nations (https://www.businessinsider.com/poorest-countries-in-the-world-2018-5#17-guinea-816-12) and data usage. I will make a scatter plot and perform some correlation tests.

I do expect to see some correlation since these refugees are less likely to have the device to access data. One can argue that there have been many smartphone and tablet distributions by NGOs and corporations as part of their CSR (corporate social responsibility) activities, but this still does not take into account the low rate of literacy and digital literacy in these nations.

Variables used - Sum of proportion of refugees from impoverished nations (float; from refugee_nationality.csv), usage per person (float; from hourly_usage_per_camp.csv), country (string; from the camp name)

Let's narrow down the nationalities first. We can use the nationality dataframe again.

In [62]:
```
nationality.head()
```

Out[62]:

	monthyear	campfilename	nationality	proportion	cmy
0	022018	Alexandria-Ref_GRE_003	Syrian	0.44	Alexandria-Ref_GRE_003022018
1	022018	Alexandria-Ref_GRE_003	Iraqi	0.25	Alexandria-Ref_GRE_003022018
2	022018	Alexandria-Ref_GRE_003	Pakistani	0.08	Alexandria-Ref_GRE_003022018
3	022018	Alexandria-Ref_GRE_003	Palestinian	0.04	Alexandria-Ref_GRE_003022018
4	022018	Diavata_GRE_008	Syrian	0.38	Diavata_GRE_008022018

In [63]:
```
nationality.nationality.unique()
```

Out[63]:
```
array(['Syrian', 'Iraqi', 'Pakistani', 'Palestinian', 'Iranian', 'Afghan',
       'Yemeni', 'Stateless', 'Congolese', 'Eritrean', 'Somali',
       'Kuwaiti', 'Moroccan', 'Egyptian', 'Algerian'], dtype=object)
```

In [64]:
```
# pivot and just get Afghan, Congolese, and Yemeni
n_pvtd = nationality.pivot(index='cmy',columns='nationality',values='proportio
n').reset_index()
n_pvtd.head()
```

Out[64]:

nationality	cmy	Afghan	Algerian	Congolese	Egyptian	Eritrean
0	Adasevci_SRB_010012018	0.54	NaN	NaN	NaN	NaN
1	Adasevci_SRB_010022018	0.55	NaN	NaN	NaN	NaN
2	Adasevci_SRB_010032018	0.48	NaN	NaN	NaN	NaN
3	Adasevci_SRB_010092018	0.18	NaN	NaN	NaN	NaN
4	Alexandria-Ref_GRE_003012017	0.00	0.0	0.0	0.0	0.0

In [68]:
```
n_pvtd = n_pvtd.fillna(0)
n_pvtd['poor'] = n_pvtd.Afghan + n_pvtd.Congolese + n_pvtd.Yemeni
df4 = n_pvtd[['cmy','poor']]
df4.head()
```

Out[68]:

nationality	cmy	poor
0	Adasevci_SRB_010012018	0.54
1	Adasevci_SRB_010022018	0.55
2	Adasevci_SRB_010032018	0.48
3	Adasevci_SRB_010092018	0.18
4	Alexandria-Ref_GRE_003012017	0.00

The dataframe from question 1 has usage per person per month data, so I will be borrowing from there and merge that data onto the dataframe that I just made above.

In [67]:
```
df1.head()
```

Out[67]:

	campfilename	monthyear	total_usage_kb	month	year	camp_name	
0	Adasevci_SRB_010	012018	912541.145	1	2018	Adasevci	Adasevci_
1	Adasevci_SRB_010	022018	801803.019	2	2018	Adasevci	Adasevci_
2	Adasevci_SRB_010	092018	1774450.363	9	2018	Adasevci	Adasevci_
3	Adasevci_SRB_010	032018	816442.546	3	2018	Adasevci	Adasevci_
4	Alexandria-Ref_GRE_003	012017	640877.689	1	2017	Alexandria	Alexandria Ref_GRE_

In [69]:
```
usage_map = pd.Series(df1.usage_per_person.values,index=df1.cmy).to_dict()
df4['usage_per_person'] = df4['cmy'].map(usage_map)
df4.head()
```

Out[69]:

nationality	cmy	poor	usage_per_person
0	Adasevci_SRB_010012018	0.54	2790.645703
1	Adasevci_SRB_010022018	0.55	2646.214584
2	Adasevci_SRB_010032018	0.48	2874.797697
3	Adasevci_SRB_010092018	0.18	3458.967569
4	Alexandria-Ref_GRE_003012017	0.00	1518.667509

I am just going to add the country column so that I can visualize that and if it seems to have an interesting effect, I will analyze those separately.

In [72]:
```
df4['country'] = df4['cmy'].apply(str).str.split('_').str[1]
df4['country'] = df4['country'].replace({'SRB':'Serbia','GRE':'Greece'})
df4.describe()
```

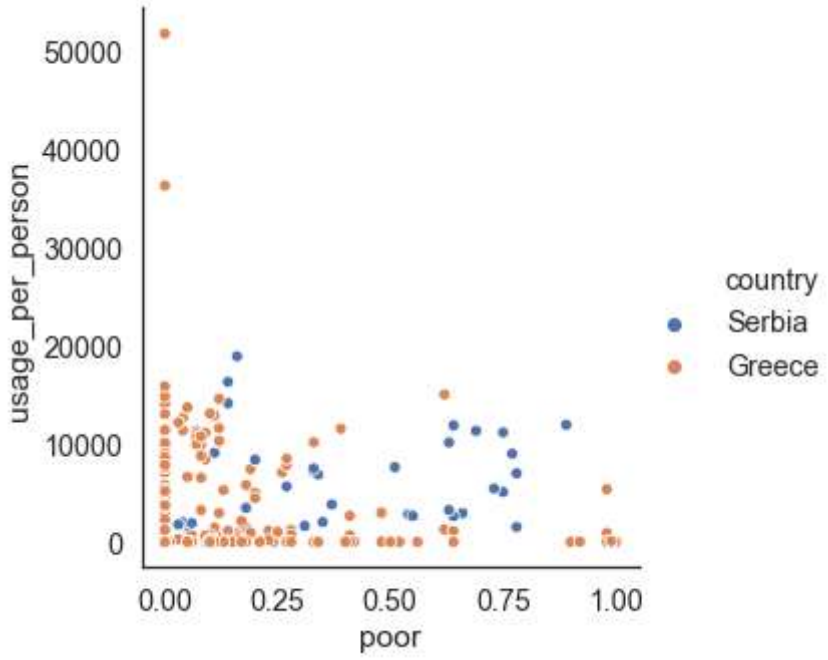
Out[72]:

nationality	poor	usage_per_person
count	306.000000	265.000000
mean	0.172810	4100.025072
std	0.240907	5759.528369
min	0.000000	0.000000
25%	0.000000	1.326307
50%	0.090000	1518.667509
75%	0.237500	7426.630377
max	1.000000	51824.722684

Now let's make our scatter plot!

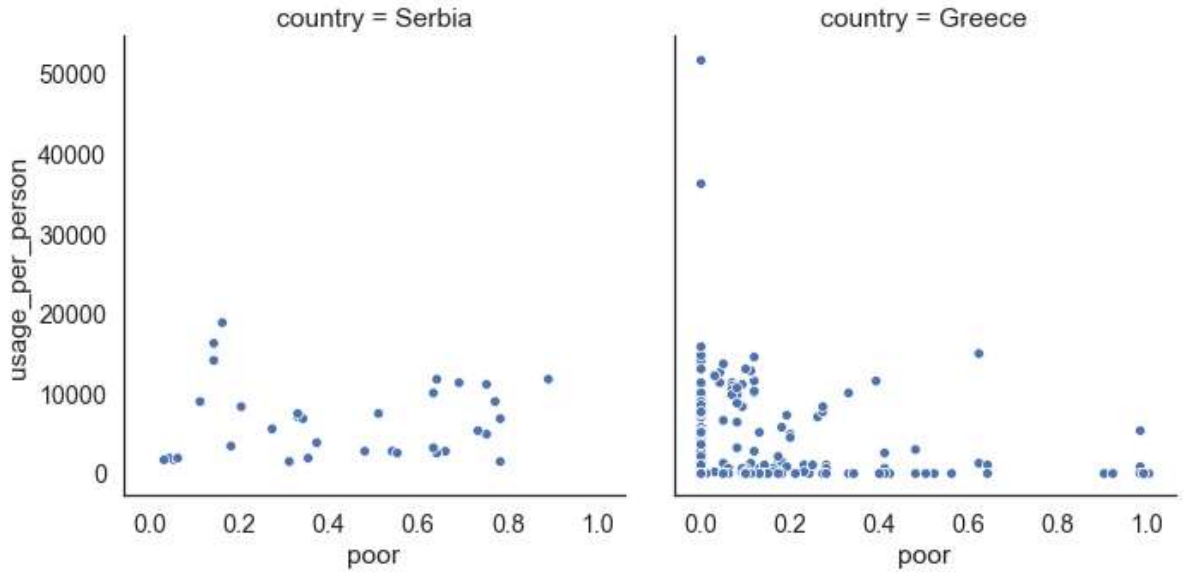
In [71]:
```
sns.relplot(x="poor", y="usage_per_person", hue="country", data=df4)
```

Out[71]: <seaborn.axisgrid.FacetGrid at 0x1a3ec879a20>



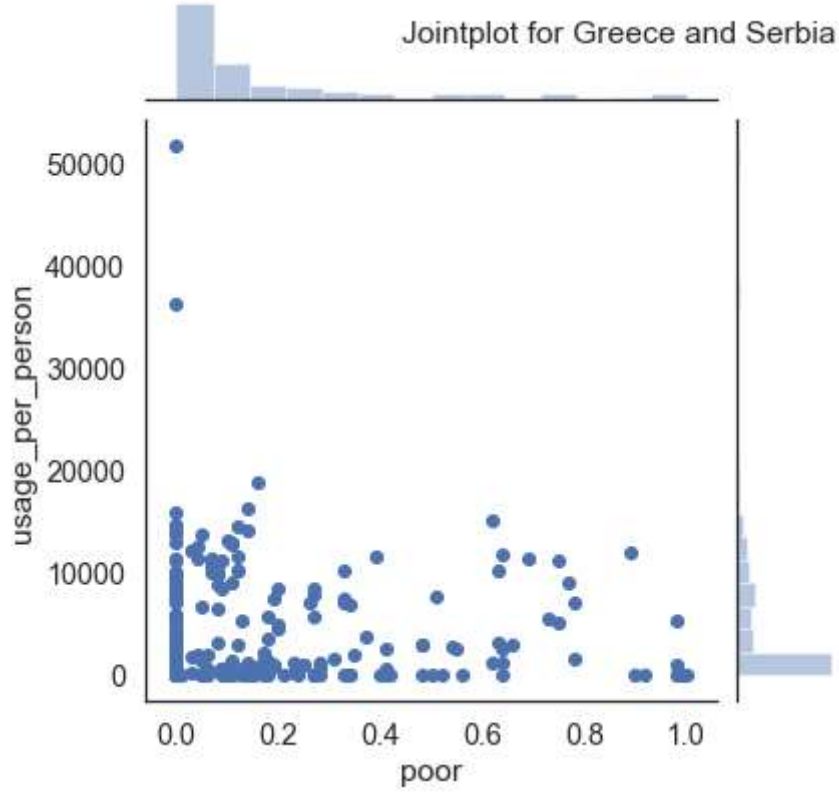
In [73]:
```
sns.relplot(x="poor", y="usage_per_person", col="country", data=df4)
```

Out[73]: <seaborn.axisgrid.FacetGrid at 0x1a3ec7fbd30>

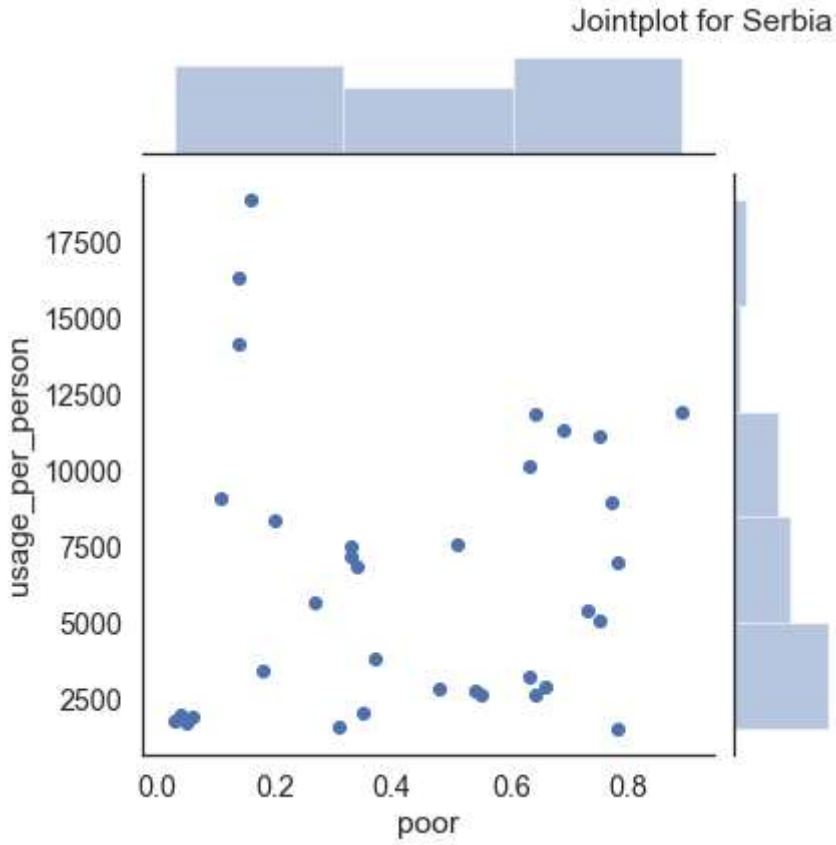


There actually seems to be a lot of 0s in our data. Our distribution for both x- and y-axis might be very abnormal. I am going to plot the scatter plots again but with histograms.

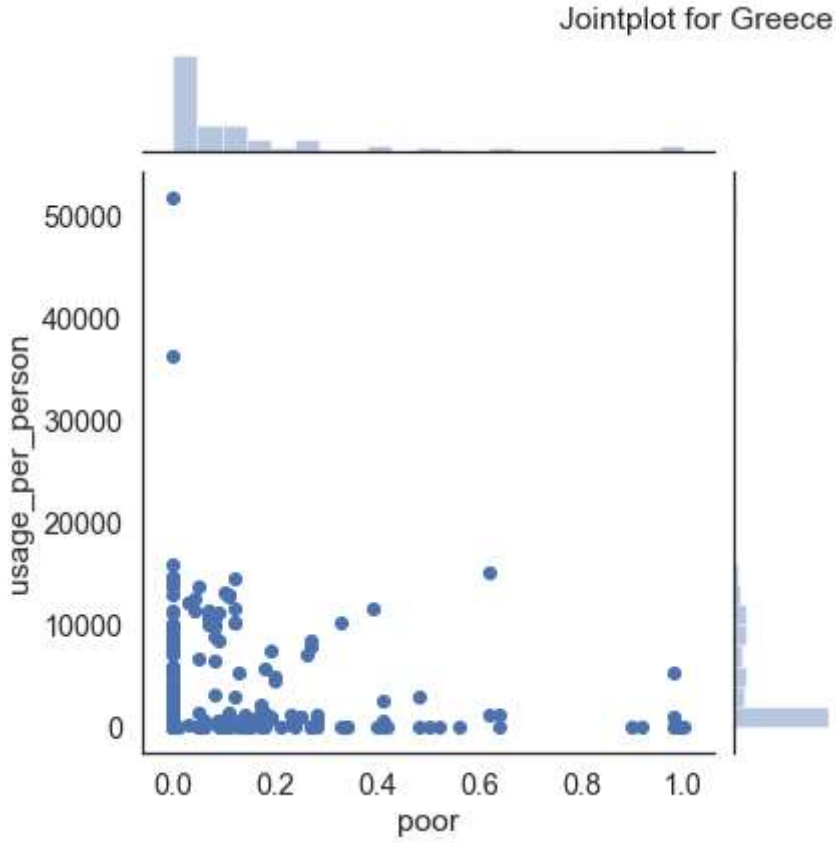
In [102]:
```
both = sns.jointplot(x="poor", y="usage_per_person", data=df4)
plt.title('Jointplot for Greece and Serbia\n\n',loc='right')
plt.show(both)
```



In [106]:
```
sr = sns.jointplot(x="poor", y="usage_per_person", data=df4[df4['country']=='Serbia'])
plt.title('Jointplot for Serbia\n\n\n\n',loc='right')
plt.show(sr)
```

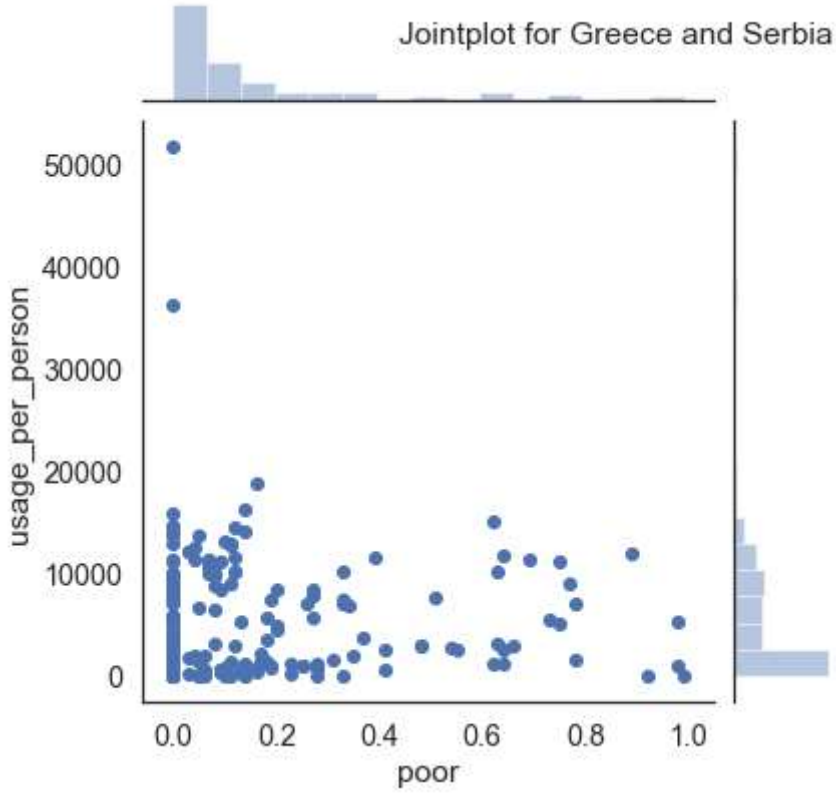


In [107]:
```
gr = sns.jointplot(x="poor", y="usage_per_person", data=df4[df4['country']=='G
reece'])
plt.title('Jointplot for Greece\n\n\n\n',loc='right')
plt.show(gr)
```

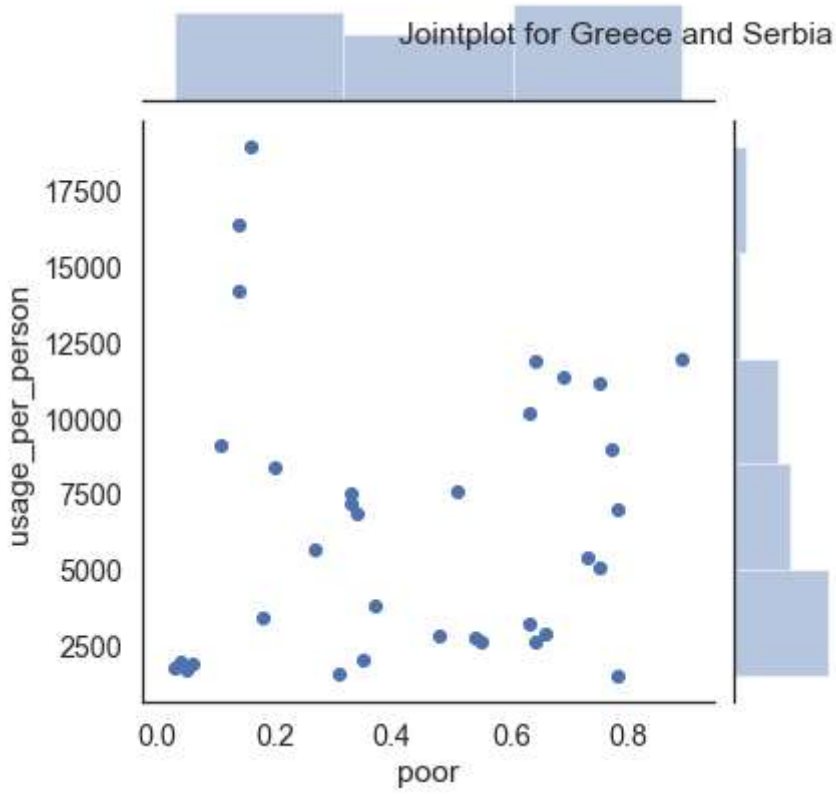


I can see that Greece had a lot of 0 usage months. This may have been due to outages or maybe because the Meraki routers were not used. I am going to plot build the jointplots again without 0 usage.

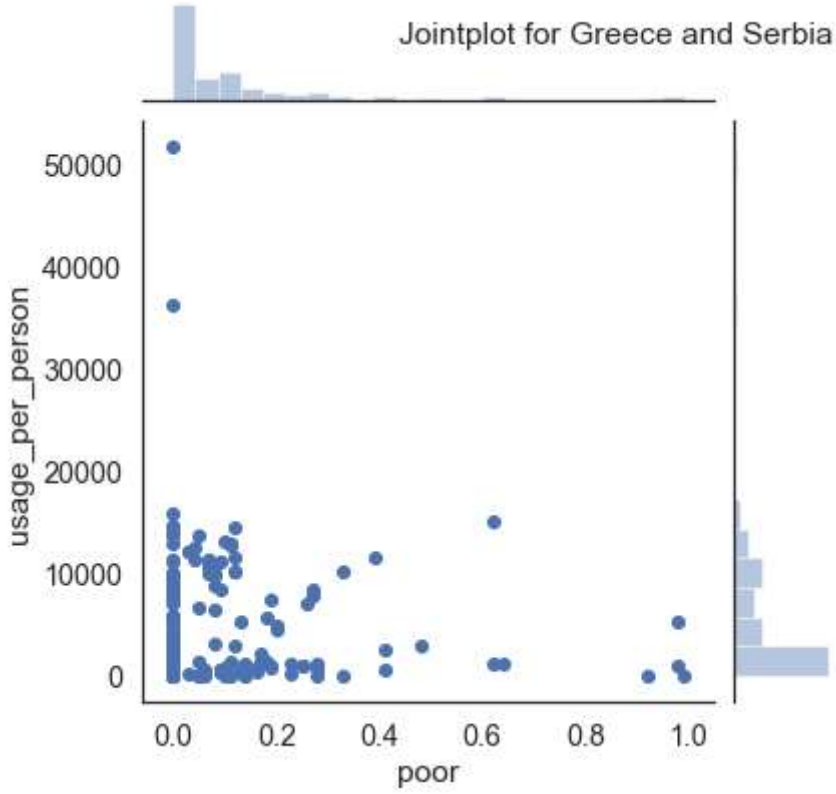
In [114]:
```
both = sns.jointplot(x="poor", y="usage_per_person", data=df4[df4['usage_per_person']>0])
plt.title('Jointplot for Greece and Serbia\n\n',loc='right')
plt.show(both)
```



In [112]:
```
both = sns.jointplot(x="poor", y="usage_per_person", data=df4[(df4['usage_per_
person']>0) & (df4['country']=='Serbia')])
plt.title('Jointplot for Greece and Serbia\n\n',loc='right')
plt.show(both)
```



In [110]:
```
both = sns.jointplot(x="poor", y="usage_per_person", data=df4[(df4['usage_per_person']>0) & (df4['country']=='Greece')])
plt.title('Jointplot for Greece and Serbia\n\n',loc='right')
plt.show(both)
```



... And it looks like that didn't really have an impact on the distribution shapes. I will just carry on and test for correlations.

In [122]:
```
df4.corr()
```

Out[122]:

	nationality	poor	usage_per_person
nationality			
poor	1.000000	-0.053194	
usage_per_person	-0.053194	1.000000	

In [124]:
```
df4[df4['country']=='Serbia'].corr()
```

Out[124]:

	nationality	poor	usage_per_person
nationality			
poor	1.000000	0.040882	
usage_per_person	0.040882	1.000000	

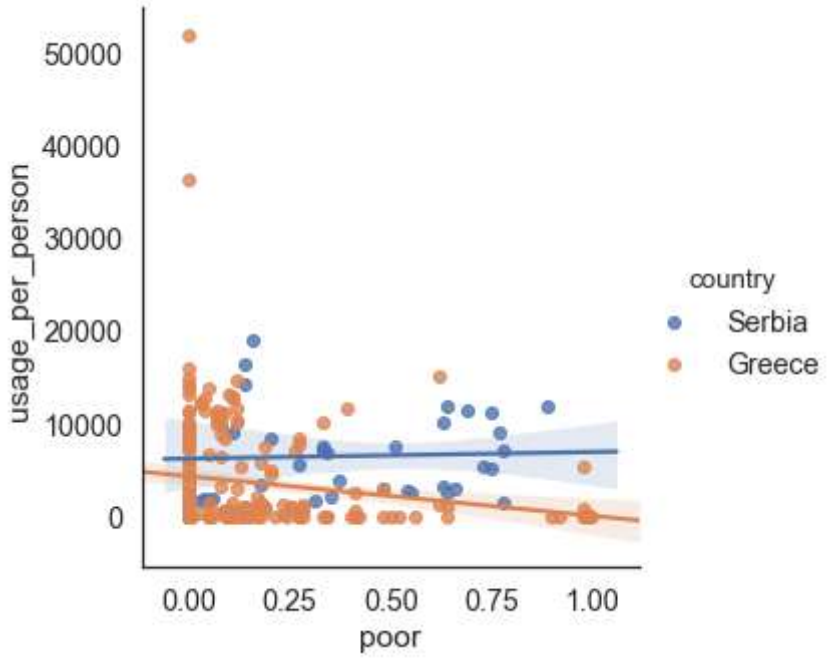
In [125]:
```
df4[df4['country']=='Greece'].corr()
```

Out[125]:

	nationality	poor	usage_per_person
nationality			
poor	1.000000	-0.159411	
usage_per_person	-0.159411	1.000000	

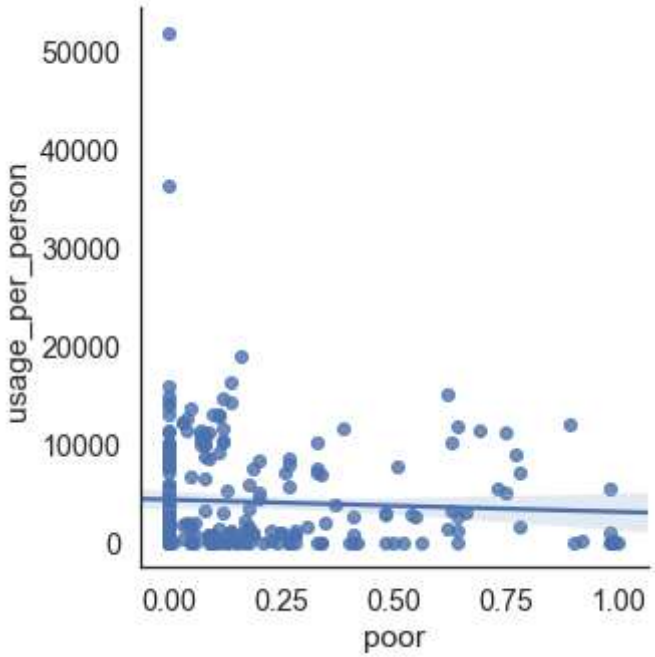
In [123]:
```
sns.lmplot(x="poor", y="usage_per_person", hue="country", data=df4)
```

Out[123]: <seaborn.axisgrid.FacetGrid at 0x1a3ef817be0>



In [126]:
```
sns.lmplot(x="poor", y="usage_per_person", data=df4)
```

Out[126]: <seaborn.axisgrid.FacetGrid at 0x1a3edec7048>



Procedure: Before pivoting the nationality.csv again, I looked at all of the unique nationalities for which I had data and narrowed them down to the ones that were in the 28 most impoverished nations list. After pivoting, I summed up the values for the impoverished proportions and put that information together with the usage per person variable used in the first question. I also added a country column just in case there was a difference based on country.

I then created scatterplots and joint plots using Seaborn to not only look for linear trends, but also to see the shapes of the relevant distributions. I then calculated correlation coefficients using Pearson's R.

I originally was going to look at just the African-descent refugees, but this was tough considering I had a lot of null values for African nationalities, which is why I went with the impoverished-nation approach instead which included Afghans, one of the biggest group of refugees in Europe. Even with this, I ended up with a lot of 0 data for these nationalities, as well as for usage (0 usage data was included in the dataset to take into account the absence of working routers), which skewed the data quite a bit.

Findings: Regardless of separating the dataset by country or not, the correlation between total usage per month and the number of refugees from impoverished nations was very weak and it is hard to conclude that these two factors are negatively correlated with the two factors being very slightly positively correlated (r=.0409) and the overall correlation being very weak (r=-.0532). In conclusion, the data does not show much relationship between the amount of refugees from impoverished nations and data usage.