In [1]:
```
# Covid-19 project
# Investigate whether deaths caused by COVID-19 by country can be predicted by
examining, analizing and using
# relevant common, well-established data

# ML algorithm used:
    # XGBoost regression model.

# Relevant common data by country
    # Continent -->
        # 2020 data from https://simple.wikipedia.org/wiki/List_of_countries_b
y_continents
    # Population density (Population per square km) -->
        # 2018 data from https://data.worldbank.org/indicator/EN.POP.DNST
    # Percentage of population in urban agglomerations of more than 1 million
 (Agglomerates (%)) -->
        # 2019 data from https://data.worldbank.org/indicator/EN.URB.MCTY.TL.Z
S
    # Population age distribution: percentage of population in the following a
ge groups - 0-14, 15-64, 65- -->
        # 2020 data from http://wdi.worldbank.org/table/2.1
    # GDP per cpita -->
        # 2019 data from https://data.worldbank.org/indicator/NY.GDP.PCAP.CD
    # Healthcare Access and Quality (HAQ) Index -->
        # 2016 data from https://www.thelancet.com/journals/lancet/article/PII
S0140-6736(18)30994-2/fulltext

# COVID-19 data
    # Data up to 2020-07-22 from https://coronavirus.jhu.edu/data/mortality wh
ich includes:
        # Confirmed positive cases
        # Deaths
        # Case fatality (percentage of deaths among positive cases)
        # Deaths per 100K of the county population

# All data was wrangled and compiled in one source file, ms_covid19_2020_07_2
2.csv, available on
# https://github.com/marin-stoytchev/data-science-projects/tree/master/covid_1
9_project
```

In [2]:
```
# Import libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style = "whitegrid", font_scale = 1.5)
```

In [3]:
```
# Ignore warnings

import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# Read data

data = pd.read_csv('covid19_2020_07_22.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Country                159 non-null    object 
 1   Continent              159 non-null    object 
 2   Population per sqr km  159 non-null    float64
 3   Agglomerates (%)       159 non-null    float64
 4   Age 0-14 (%)           159 non-null    int64  
 5   Age 15-64 (%)          159 non-null    int64  
 6   Age 65- (%)            159 non-null    int64  
 7   GDP per capita ($)     159 non-null    float64
 8   HAQ Index              159 non-null    int64  
 9   Confirmed              159 non-null    int64  
 10  Deaths                 159 non-null    int64  
 11  Case fatality          159 non-null    float64
 12  Deaths per 100K        159 non-null    float64
dtypes: float64(5), int64(6), object(2)
memory usage: 16.3+ KB
```

In [5]:
```
# Preview data

data.head(10)
```

Out[5]:

	Country	Continent	Population per sqr km	Agglomerates (%)	Age 0-14 (%)	Age 15-64 (%)	Age 65- (%)	GDP per capita ($)	HAQ Index	Confirmed
0	Afghanistan	Asia	56.94	10.81	42	55	3	502.11	26	35727
1	Albania	Europe	104.61	0.00	17	68	14	5352.85	75	4358
2	Algeria	Africa	17.73	6.33	31	63	7	3948.30	63	24872
3	Andorra	Europe	163.84	0.00	14	70	16	40886.40	95	889
4	Angola	Africa	24.71	25.27	47	51	2	2973.59	33	812
5	Antigua and Barbuda	N. America	218.83	0.00	22	69	9	17790.30	70	76
6	Argentina	S. America	16.26	42.90	25	64	11	10006.14	68	141900
7	Armenia	Asia	103.68	36.62	21	68	11	4622.73	71	35693
8	Australia	Australia	3.25	60.89	19	65	16	54907.10	96	13302
9	Austria	Europe	107.13	21.57	14	67	19	50277.27	94	19929

In [6]:
```
# Note: the zero values in 'Agglomerates (%)' are true 0s since the correspond
ing coutries have no cities over 1 million
```

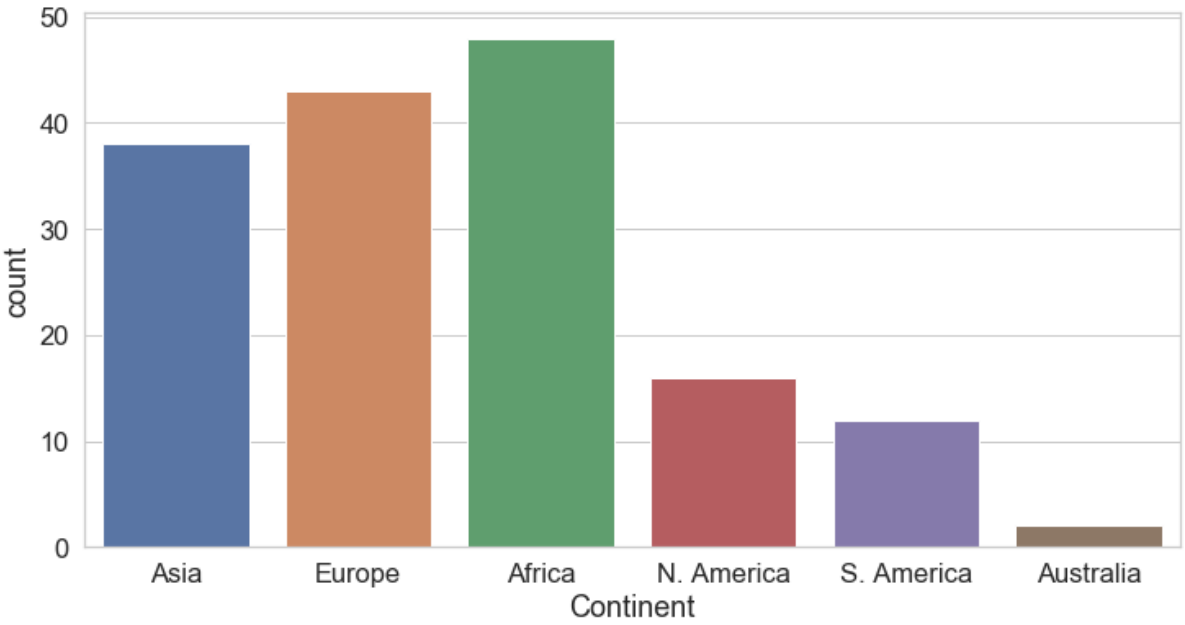
In [7]:
```
# 1) EDA
```

In [8]:
```
# For consistency convert 'Case fatality' from rational numbers (ratios) to pe
rcentages

data['Case fatality'] = 100 * data['Case fatality']
```

In [9]:
```
# Get data column names
data.columns
```

Out[9]:
```
Index(['Country', 'Continent', 'Population per sqr km', 'Agglomerates (%)',
       'Age 0-14 (%)', 'Age 15-64 (%)', 'Age 65- (%)', 'GDP per capita ($)',
       'HAQ Index', 'Confirmed', 'Deaths', 'Case fatality', 'Deaths per 100
K'],
      dtype='object')
```

In [10]:
```
# Countplot of number of coutries by continent

plt.figure(figsize = (12, 6))
sns.countplot(data['Continent'])
plt.show()
```



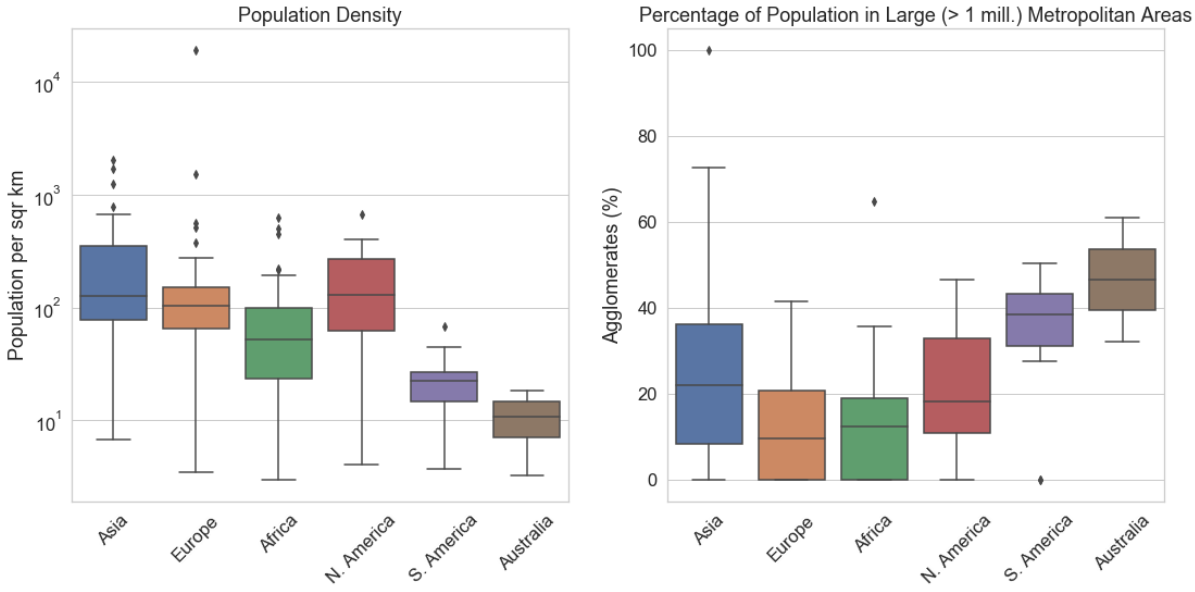
In [11]:
```
# Get precise count of countries by continent

data['Continent'].value_counts()
```

Out[11]:
```
Africa        48
Europe        43
Asia          38
N. America    16
S. America    12
Australia      2
Name: Continent, dtype: int64
```

In [12]:
```
# Africa, Europe and Asia have the largest representation, followed by North a
nd South America;
# Australian continent is represented by two countries only --> Australia and
 New Zeeland
```

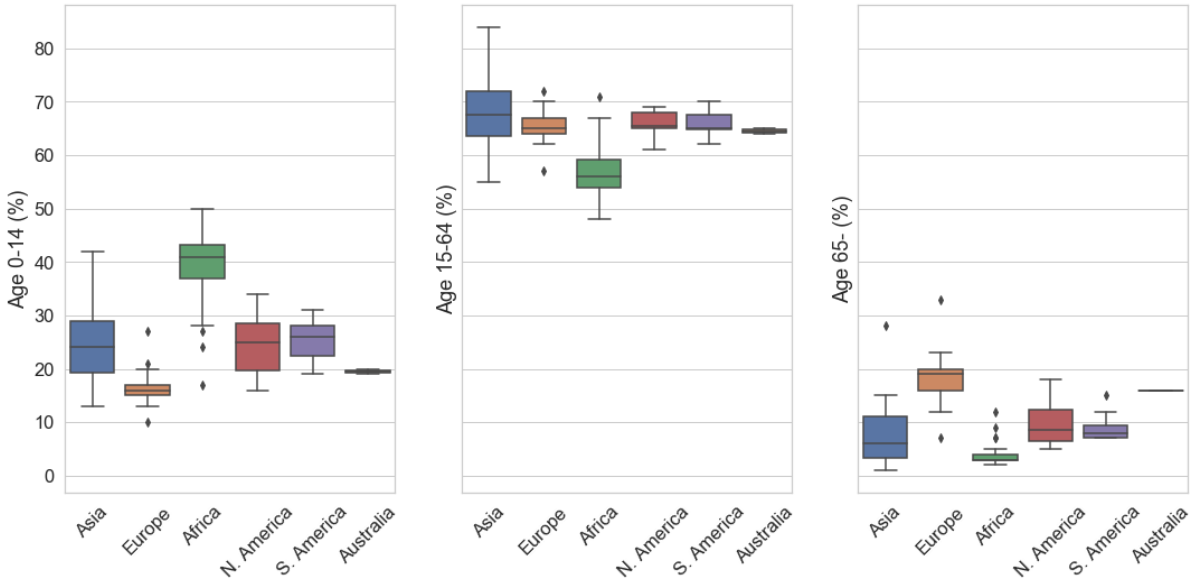
In [13]:
```
# Examine distributions of feature values by continent via boxplots
```

In [14]:
```
# Boxplots of Population Density and Percentage of people living in large metr
o areas (> 1 mill.) vs. Continent

fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey = False)

sns.boxplot(x = 'Continent', y = 'Population per sqr km', data = data, ax = ax
es[0])
axes[0].set(xlabel = None)
axes[0].tick_params(axis = 'x', rotation = 45)
axes[0].set(yscale = 'log')
axes[0].set(title = 'Population Density')

sns.boxplot(x = 'Continent', y = 'Agglomerates (%)', data = data, ax = axes[1
])
axes[1].set(xlabel = None)
axes[1].tick_params(axis = 'x', rotation = 45)
axes[1].set(title = 'Percentage of Population in Large (> 1 mill.) Metropolita
n Areas')

plt.show()
```



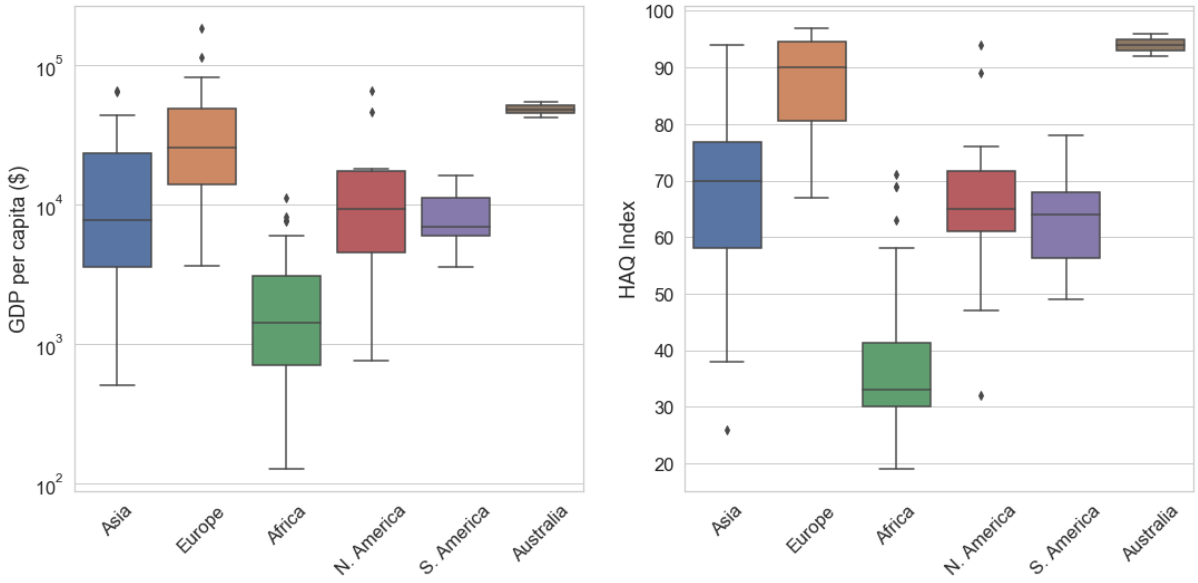
In [15]:
```
# Two trends are observed
#1) High population density with low(er) percentege of people in large (> 1 mi
ll.) metro areas -->
    # Asia, Europe, Africa, N. America
#2) Low population density with high(er) percentege of people in large (> 1 mi
ll.) metro areas -->
    # S. America, Australia
```

In [16]:
```
# Boxplots of Age Demographics vs. Continent

fig, axes = plt.subplots(1, 3, figsize=(18, 8), sharey = True)

sns.boxplot(x = 'Continent', y = 'Age 0-14 (%)', data = data, ax = axes[0])
axes[0].set(xlabel = None)
axes[0].tick_params(axis = 'x', rotation = 45)
axes[0].set_yticks(np.arange(0, 100, 10))

sns.boxplot(x = 'Continent', y = 'Age 15-64 (%)', data = data, ax = axes[1])
axes[1].set(xlabel = None)
axes[1].tick_params(axis = 'x', rotation = 45)

sns.boxplot(x = 'Continent', y = 'Age 65- (%)', data = data, ax = axes[2])
axes[2].set(xlabel = None)
axes[2].tick_params(axis = 'x', rotation = 45)

plt.tight_layout
plt.show()
```



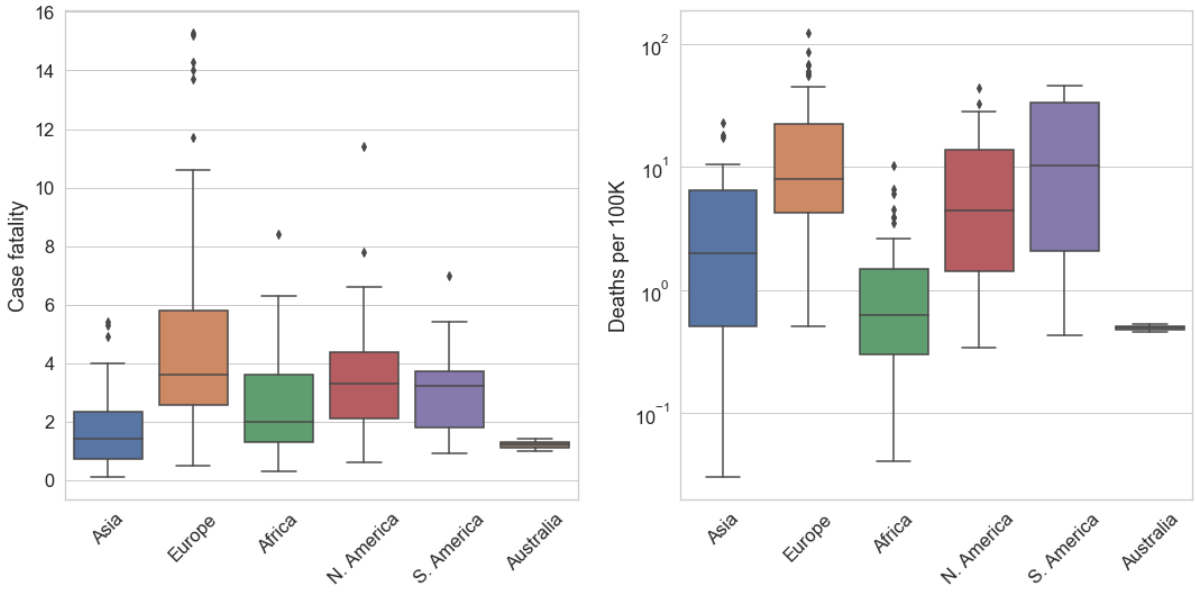
In [17]:
```
# Main observations:
    # 1) Africa: 'youngest' continent with average of ~ 40 % of population bel
ow 15 years old and only ~ 3 % older than 64
    # 2) Europe: 'oldest' continent with average of ~ 15 % of population below
15 years old and ~ 20 % older than 64
    # 3) Rest of continets: 'middle-aged' with similar age demographics with s
ome deviations (more pronounced for Age 65-)
```

In [18]:
```
# Boxplots of GDP per capita and HAQ Index vs. Continent

fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey = False)

sns.boxplot(x = 'Continent', y = 'GDP per capita ($)', data = data, ax = axes[
0])
axes[0].tick_params(axis = 'x', rotation = 45)
axes[0].set(yscale = 'log')
axes[0].set(xlabel = None)

sns.boxplot(x = 'Continent', y = 'HAQ Index', data = data, ax = axes[1])
axes[1].tick_params(axis = 'x', rotation = 45)
axes[1].set(xlabel = None)

plt.show()
```



In [19]:
```
# Clear visual corellation between these two features (natural to expect)
# Lowest score in both charts: Africa
# Max scores in both charts: Australia + New Zeeland, Europe
# Middle socres:Asia, N. America, S. America
```

In [20]:
```
# Boxplots of the two possible targets, Case fatality and Deaths per 100K, vs. Continent

fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey = False)

sns.boxplot(x = 'Continent', y = 'Case fatality', data = data, ax = axes[0])
axes[0].tick_params(axis = 'x', rotation = 45)
axes[0].set(xlabel = None)

sns.boxplot(x = 'Continent', y = 'Deaths per 100K', data = data, ax = axes[1])
axes[1].tick_params(axis = 'x', rotation = 45)
axes[1].set(xlabel = None)
axes[1].set(yscale = 'log')

plt.show()
```



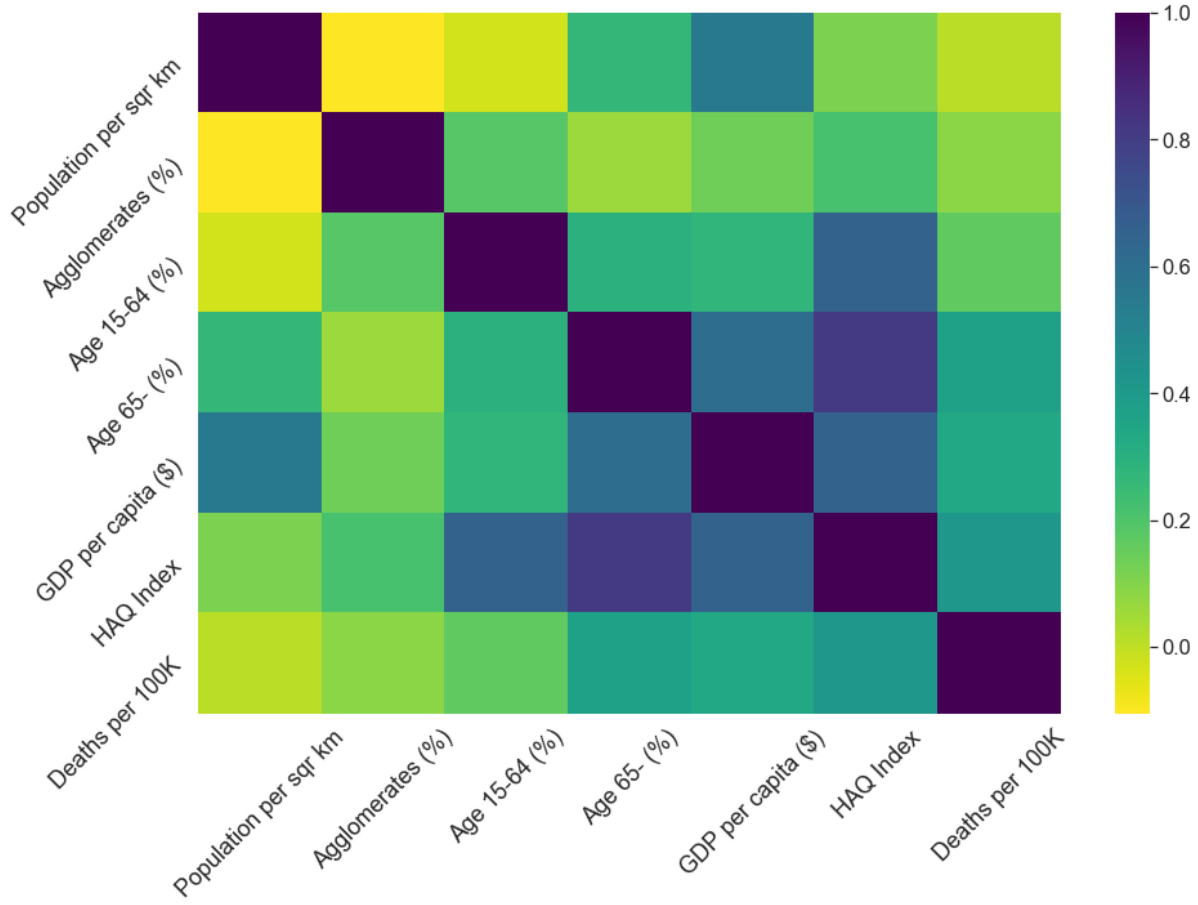
In [21]:
```
# Both 'Case fatality' and 'Deaths per 100K' show large variations
# However, for many reasons 'Case fatality' is not considered a reliable measure -->
    # select 'Deaths per 100K' as target
```

In [22]:
```
# For better correlation and pairplot visualization create new dataset, data_
1, consisting of most relevant features

data_1 = data[['Population per sqr km', 'Agglomerates (%)', 'Age 15-64 (%)',
'Age 65- (%)', 'GDP per capita ($)',
               'HAQ Index','Deaths per 100K']]
# 'Age 0-14 (%)' is being dropped because of explicit relationship with 'Age 1
5-64 (%)' and 'Age 65- (%)' -->
    # 'Age 0-14 (%)' = 100 - ('Age 15-64 (%)' + 'Age 65- (%)')

data_1.head(10)
```

Out[22]:

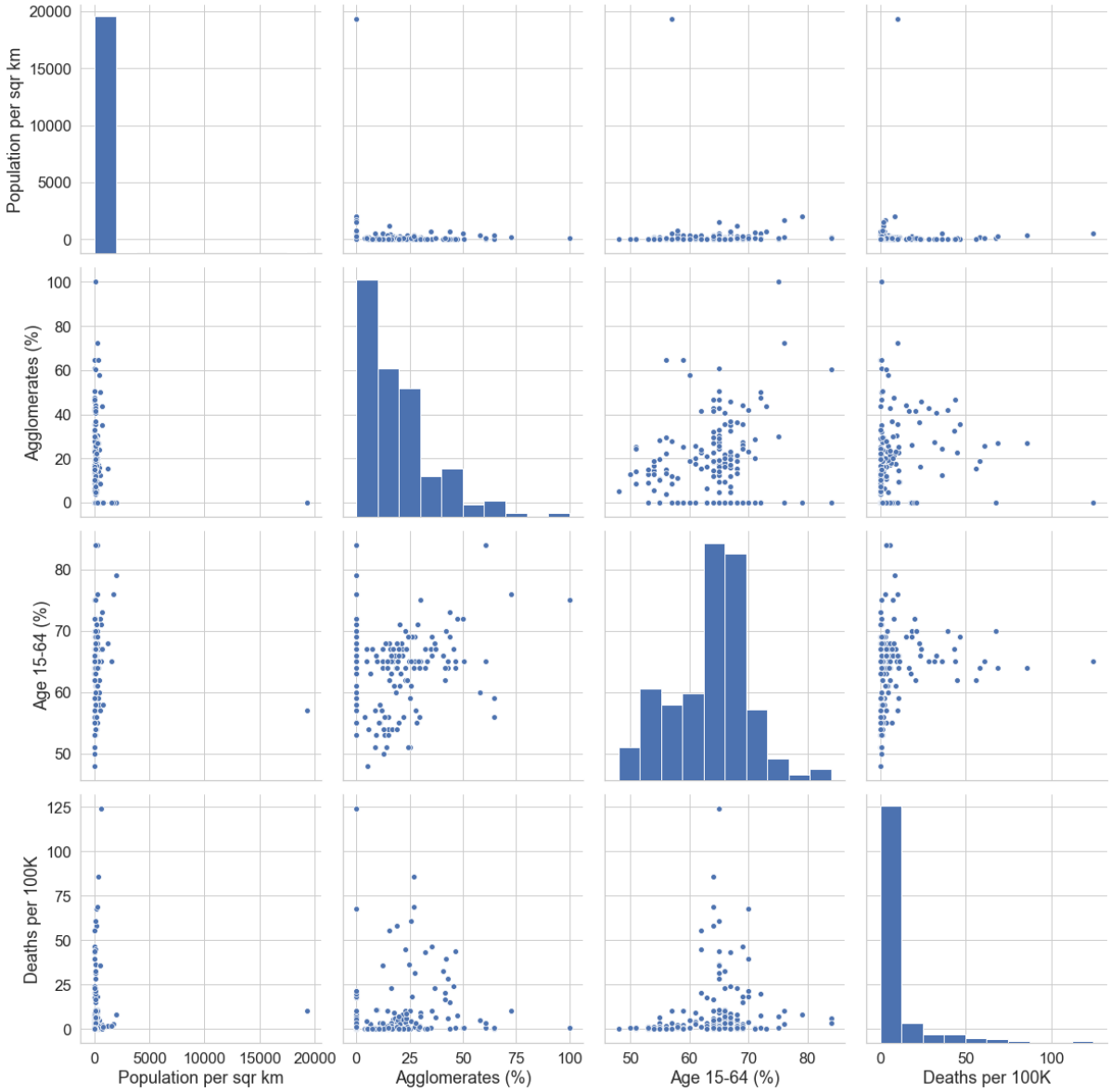
	Population per sqr km	Agglomerates (%)	Age 15-64 (%)	Age 65- (%)	GDP per capita ($)	HAQ Index	Deaths per 100K
0	56.94	10.81	55	3	502.11	26	3.20
1	104.61	0.00	68	14	5352.85	75	4.19
2	17.73	6.33	63	7	3948.30	63	2.63
3	163.84	0.00	70	16	40886.40	95	67.53
4	24.71	25.27	51	2	2973.59	33	0.11
5	218.83	0.00	69	9	17790.30	70	3.12
6	16.26	42.90	64	11	10006.14	68	5.82
7	103.68	36.62	68	11	4622.73	71	22.97
8	3.25	60.89	65	16	54907.10	96	0.53
9	107.13	21.57	67	19	50277.27	94	8.04

In [23]:
```
# Plot heatmap of the correlation matrix for data_1 to examine for highly-corr
elated features

plt.figure(figsize = (15, 10))
sns.heatmap(data_1.corr(), cmap = 'viridis_r')
plt.tick_params(labelsize = 18, rotation = 45)
plt.show()
```



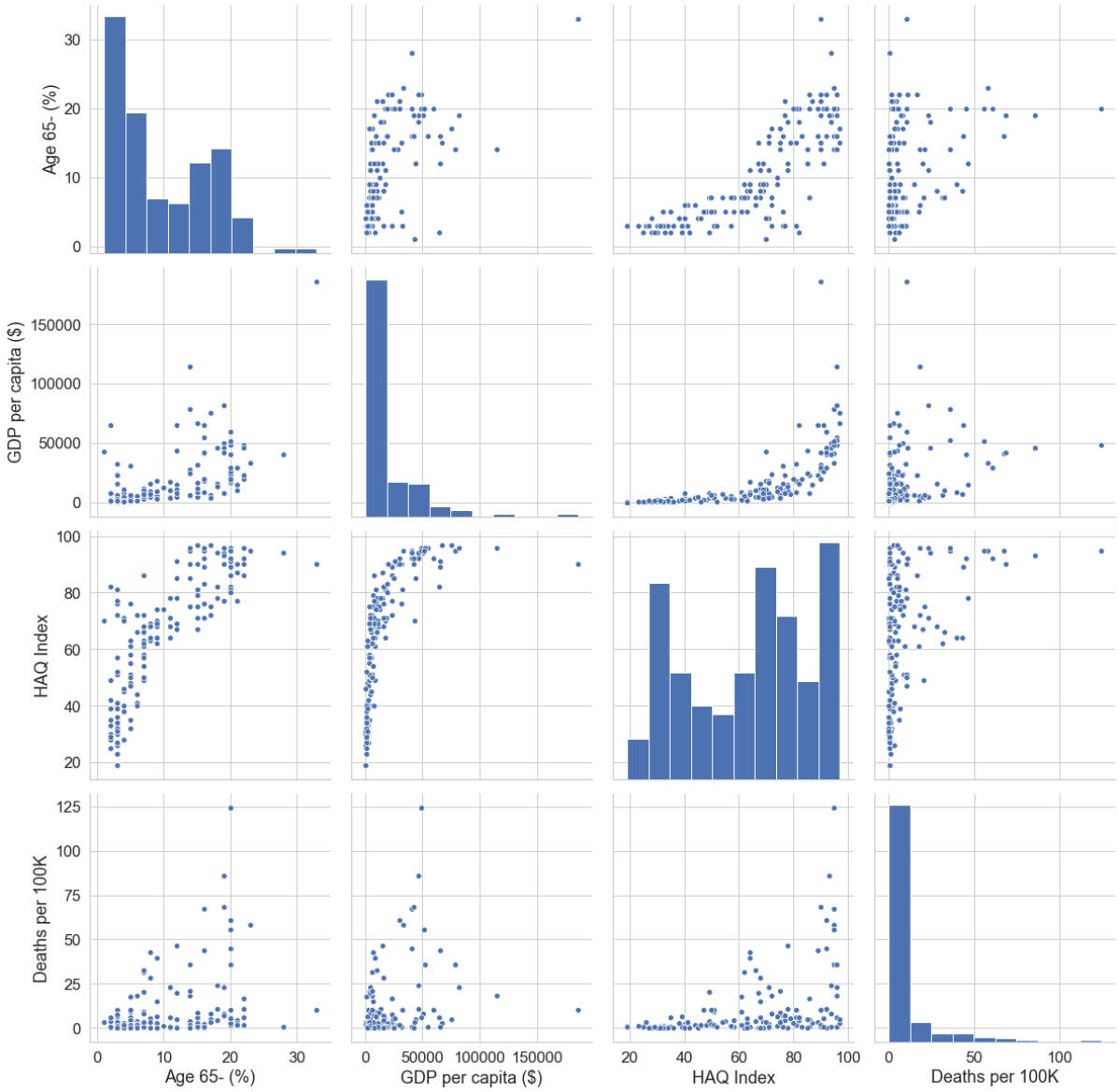
In [24]:
```
# No strong correlation between most of the features
# Higher degree of correlation is observed between
    # 'HAQ Index' and 'Age 65- (%)'
    # 'HAQ Index' and 'Age 15-64 (%)'
    # 'HAQ Index' and 'GDP per capita ($)'
    # all of these are natural to expect
# Surprisingly, very low correlation is observed between 'Deaths per 100K' and
the two population density features
```

In [25]:
```
# For better readability, create pairplot with first half of data only + target

sns.pairplot(data_1.iloc[:, [0, 1, 2, 6]], height = 4, aspect = 1)
plt.tight_layout
plt.show()
```



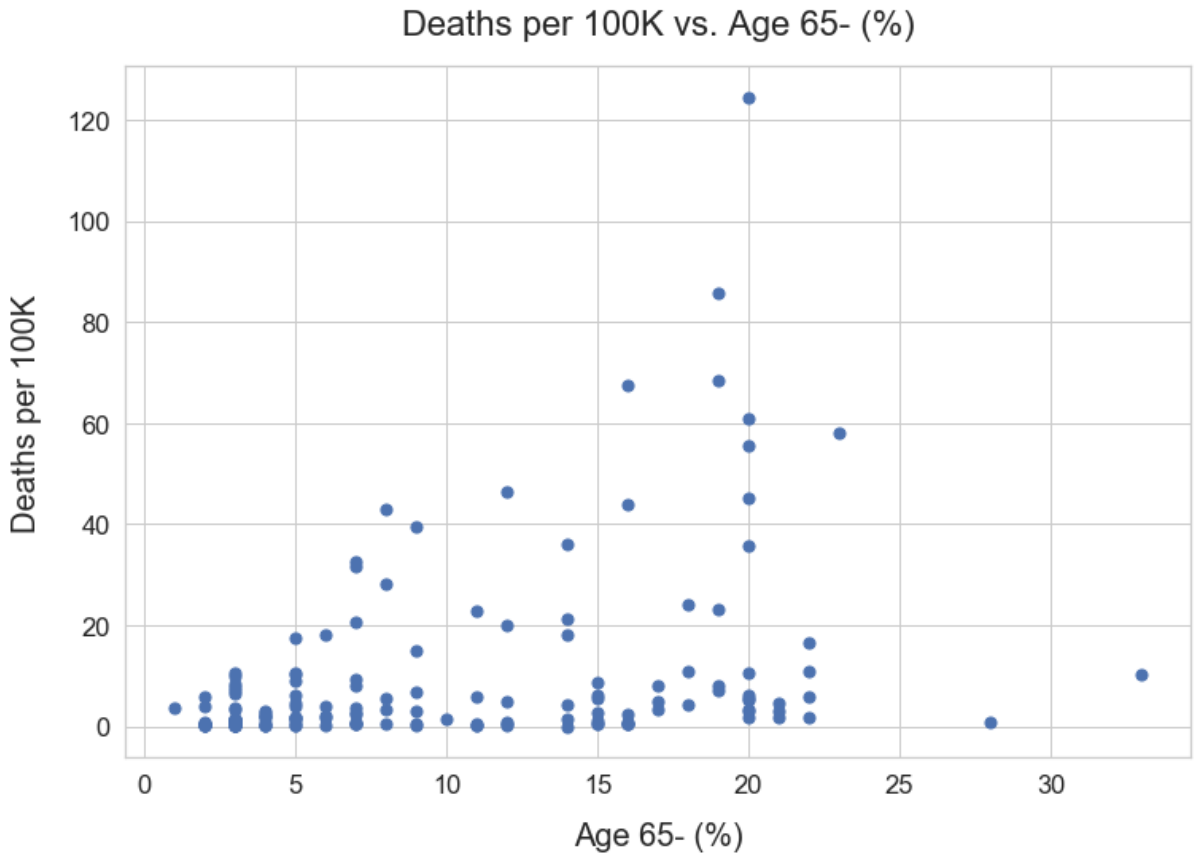
In [26]:
```
# There appear to be no clear dependence of the target on the features plotted here and dependence between features as well
# Note: Population Density distribution plotted here is strongly skewed because of a single extremely large value (Monaco)
```

In [27]:
```
# Create pairplot with second half of data + target

sns.pairplot(data_1.iloc[:, [3, 4, 5, 6]], height = 4, aspect = 1)
plt.tight_layout
plt.show()
```



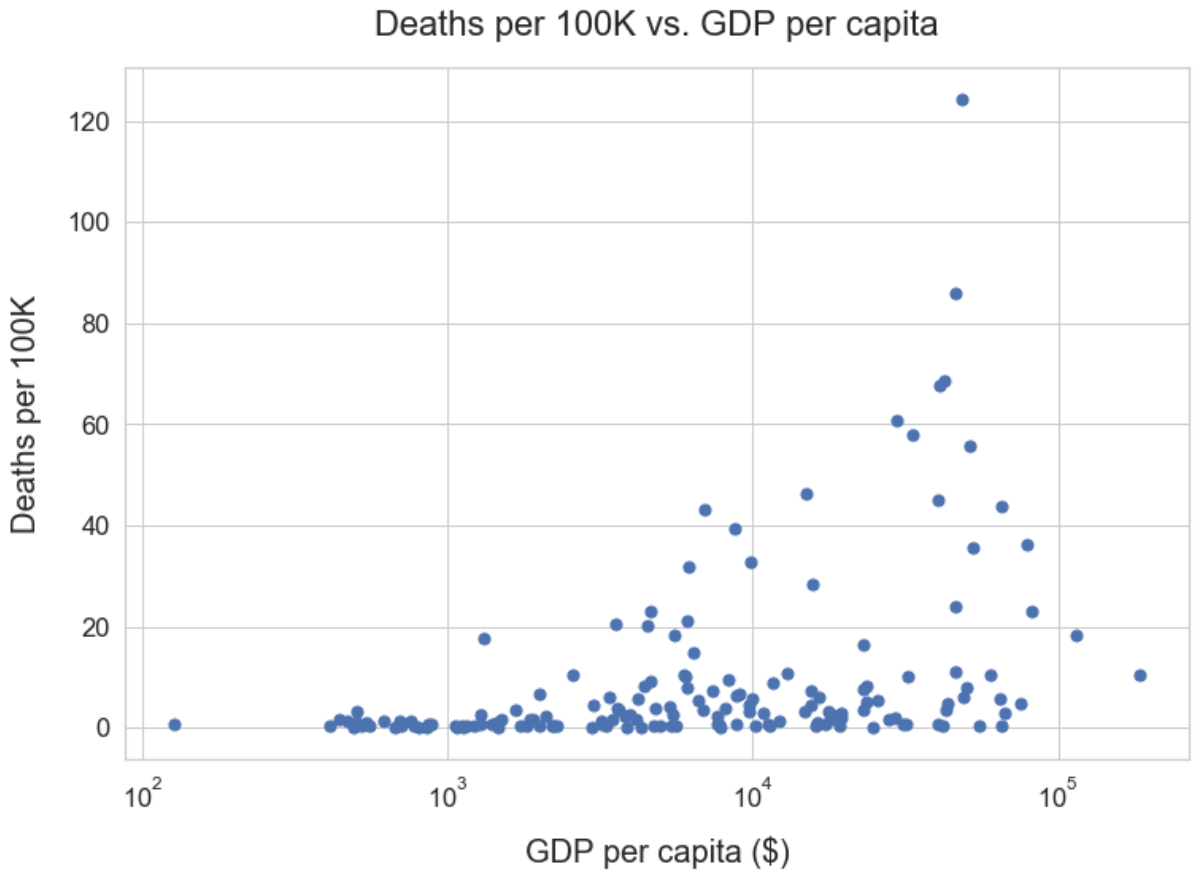
In [28]:
```
# A more defined relationship is observed between 'HAQ Index' and 'Age 65-
 (%)' as already indicated by the correlation matrix
# To lesser degree relationship is observed between 'HAQ Index' and 'GDP per c
apita ($)', as well
# However, the relationships between Deths per 100K and these three features i
s not clear --> investigate these in more detail
```

In [29]:
```
# Create scatterplot between 'Deaths per 100K' and 'Age 65-'

plt.figure(figsize = (12, 8))
plt.scatter(data['Age 65- (%)'], data['Deaths per 100K'], s = 50, c = 'b')
plt.xlabel('Age 65- (%)', fontsize = 20, labelpad = 15)
plt.ylabel('Deaths per 100K', fontsize = 20, labelpad = 15)
plt.title('Deaths per 100K vs. Age 65- (%)', fontsize = 22, pad = 20)

plt.show()
```



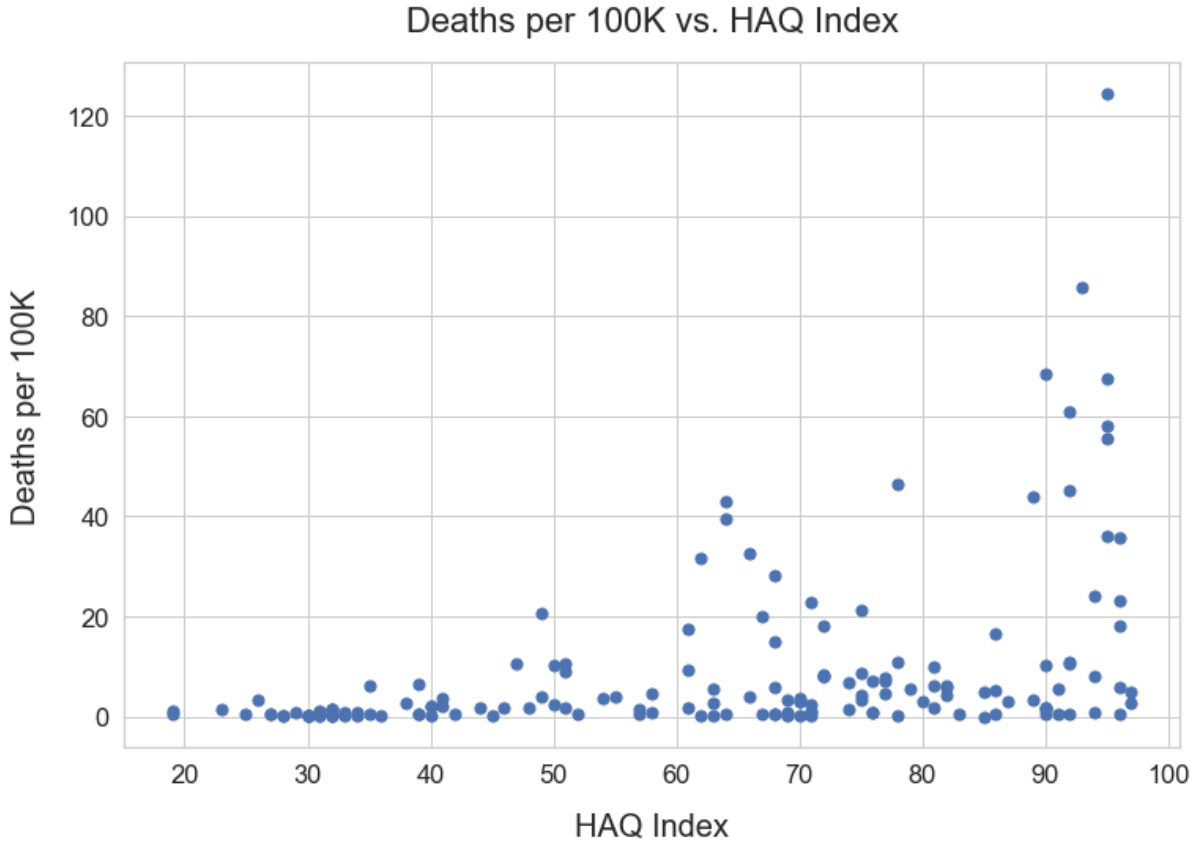
In [30]:
```
# Main observations from plot
# 1) Some of the highest death rate values are observed for the percentage of
 population older than 65 being within 15-25 %
    # However, there is no consitent relationship in this range since the majo
rity of death rate values are still very low
    # and the two data points with highest percentage of people older than 65
 show low death rate
# 2) Consistently low death rate values are observed for the percentage of peo
ple above 65 being within 0-5 %

# These observations are consistent with studies showng that COVID-19 has high
est mortality rate for people older than 65
```

In [31]:
```
# Create scatterplot between 'Deaths per 100K' and 'GDP per capita ($)'

plt.figure(figsize = (12, 8))
plt.scatter(data['GDP per capita ($)'], data['Deaths per 100K'], s = 50, c =
'b')
plt.xscale('log')
plt.xlabel('GDP per capita ($)', fontsize = 20, labelpad = 15)
plt.ylabel('Deaths per 100K', fontsize = 20, labelpad = 15)
plt.title('Deaths per 100K vs. GDP per capita', fontsize = 22, pad = 20)

plt.show()
```



In [32]:
```
# This result is surprising because it defies conventional logic
    # 1) Counrties with lowest GDP per capita consistently show lowest number
 of deaths per 100k
    # 2) As GDP per capita increases, so is the death rate --> highest death r
ates are observed for very high GDP values
```

In [33]:
```
# Create scatterplot between 'Deaths per 100K' and 'HAQ Index'

plt.figure(figsize = (12, 8))
plt.scatter(data['HAQ Index'], data['Deaths per 100K'], s = 50, c = 'b')
plt.xlabel('HAQ Index', fontsize = 20, labelpad = 15)
plt.ylabel('Deaths per 100K', fontsize = 20, labelpad = 15)
plt.title('Deaths per 100K vs. HAQ Index', fontsize = 22, pad = 20)

plt.show()
```



In [34]:
```
# This plot resembles the previous plot and again shows a counterintuitive relationship
    # 1) Counrties with the lowest HAQ Index consistently show lowest deaths per 100k
    # 2) Counrties with the highest HAQ Index (best healthcare) show the highest deaths per 100k observed

# The two relationships revealed here between mortality rate and GDP and Healthcare quality is suprising and
# raises questions about the reliability of COVID-19 data
# To some extent one can contribute the low death rates at the low ranges of these features with the low score of people above 65
# However, this argument cannot be applied to explain the wide variations in death rate for the high end ranges
```

In [35]:
```
# 2) Predicting HAQ Index based on conventional factors using XGBoost

# This is done to establish that the factors considerd here can be used to pre
dict accurately strictly medicaly-derived feature
    # see https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(18)3
0994-2/fulltext on how HAQ Index is derived
```

In [36]:
```
# Create new dataset which include all features relevant to this problem

data.columns
```

Out[36]:
```
Index(['Country', 'Continent', 'Population per sqr km', 'Agglomerates (%)',
       'Age 0-14 (%)', 'Age 15-64 (%)', 'Age 65- (%)', 'GDP per capita ($)',
       'HAQ Index', 'Confirmed', 'Deaths', 'Case fatality', 'Deaths per 100
K'],
      dtype='object')
```

In [37]:
```
# New data subset to be used with model

data_2 = data[['Continent', 'Population per sqr km', 'Agglomerates (%)','Age 1
5-64 (%)', 'Age 65- (%)',
               'GDP per capita ($)', 'HAQ Index', 'Deaths per 100K']]
data_2.head(10)
```

Out[37]:

	Continent	Population per sqr km	Agglomerates (%)	Age 15-64 (%)	Age 65- (%)	GDP per capita ($)	HAQ Index	Deaths per 100K
0	Asia	56.94	10.81	55	3	502.11	26	3.20
1	Europe	104.61	0.00	68	14	5352.85	75	4.19
2	Africa	17.73	6.33	63	7	3948.30	63	2.63
3	Europe	163.84	0.00	70	16	40886.40	95	67.53
4	Africa	24.71	25.27	51	2	2973.59	33	0.11
5	N. America	218.83	0.00	69	9	17790.30	70	3.12
6	S. America	16.26	42.90	64	11	10006.14	68	5.82
7	Asia	103.68	36.62	68	11	4622.73	71	22.97
8	Australia	3.25	60.89	65	16	54907.10	96	0.53
9	Europe	107.13	21.57	67	19	50277.27	94	8.04

In [38]:
```
# In order to be able to use XGBoost transform 'Continent' values into numeric categorical values

data_2['Continent'].replace({'Africa': 1, 'Asia': 2, 'Europe': 3, 'N. America': 4,
                             'S. America': 5, 'Australia': 6}, inplace=True)

data_2.head(10)
```

Out[38]:

	Continent	Population per sqr km	Agglomerates (%)	Age 15-64 (%)	Age 65- (%)	GDP per capita ($)	HAQ Index	Deaths per 100K
0	2	56.94	10.81	55	3	502.11	26	3.20
1	3	104.61	0.00	68	14	5352.85	75	4.19
2	1	17.73	6.33	63	7	3948.30	63	2.63
3	3	163.84	0.00	70	16	40886.40	95	67.53
4	1	24.71	25.27	51	2	2973.59	33	0.11
5	4	218.83	0.00	69	9	17790.30	70	3.12
6	5	16.26	42.90	64	11	10006.14	68	5.82
7	2	103.68	36.62	68	11	4622.73	71	22.97
8	6	3.25	60.89	65	16	54907.10	96	0.53
9	3	107.13	21.57	67	19	50277.27	94	8.04

In [39]:
```
# Create features and target from data_2 which we will use with XGBRegressor model

X = data_2.iloc[:, :-2].values # all columns, but last two - HAQ Index and Deaths per 100K
y = data_2.iloc[:, -2].values # HAQ Index column
```

In [40]:
```
# Train/test split

from sklearn.model_selection import train_test_split

X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

In [41]:
```
# For best predictions, apply model optimization by using hyperopt

from xgboost import XGBRegressor
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials, space_eval
from sklearn import metrics

# Create hyperparameter space to search over
space = {'max_depth': hp.choice('max_depth', np.arange(3, 15, 1, dtype = int
)),
        'n_estimators': hp.choice('n_estimators', np.arange(50, 300, 10, dtype
= int)),
        'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1.0, 0.1),
        'min_child_weight': hp.choice('min_child_weight', np.arange(0, 10, 1,
dtype = int)),
        'subsample': hp.quniform('subsample', 0.5, 1.0, 0.1),
        'learning_rate': hp.quniform('learning_rate', 0.1, 0.3, 0.1),
         'gamma': hp.choice('gamma', np.arange(0, 100, 0.5, dtype = float)),
         'reg_alpha': hp.choice('reg_alpha', np.arange(0, 100, 0.5, dtype = fl
oat)),
         'reg_lambda': hp.choice('reg_lambda', np.arange(0, 100, 0.5, dtype =
float)),

        'objective':'reg:squarederror',

        'eval_metric': 'rmse'}


def score(params):
    model = XGBRegressor(**params)

    model.fit(X_train_1, y_train_1, eval_set=[(X_train_1, y_train_1), (X_test_
1, y_test_1)],
              verbose=False, early_stopping_rounds=10)
    y_pred = model.predict(X_test_1)
    score = np.sqrt(metrics.mean_squared_error(y_test_1, y_pred))
    print(score)
    return {'loss': score, 'status': STATUS_OK}

def optimize(trials, space):

    best = fmin(score, space, algo = tpe.suggest, max_evals = 500)
    return best

trials = Trials()
best_params = optimize(trials, space)
```

```
6.820414556313732
8.44148134334799
6.979876536676007
7.192372618323956
7.109816410508609
6.921496001774812
7.1905995522567165
6.586739468743373
6.579581549736932
7.197775924525212
7.115952221129617
7.081071153174709
7.867322860459511
6.6779892442783355
7.432105599353098
6.936261339674985
6.864291451510999
7.083438401326216
7.459444400711036
7.5886163345110615
6.768361964036461
7.926329155581252
6.784031213838171
7.64848258637751
7.092071273465581
7.365128899912306
7.235996716707151
6.420291785071558
7.17942102080039
7.9720496936286676
7.464039225048046
7.4918665088204675
8.478599981570103
6.9117230173391135
7.110957361734395
7.488931071964835
7.600499224534987
6.738800268299448
7.090040471277181
7.5296836748122
7.165437675317853
7.301168975856134
7.098054049562692
7.613656863321024
6.580772145366189
7.517251600709331
6.857431772481822
7.208518673273721
6.938469022367371
7.019930078247667
6.779377636605193
6.295737669384614
7.048337447447962
6.861131344014118
7.282267289674661
7.779342302440194
6.862227474077198
```

```
7.159876861323642
6.917180270269906
6.890827917956313
7.034589566190494
7.0064861923098345
6.848567712500651
6.960814375080393
8.3391251440864
7.1284581342445925
11.151579041137701
6.854135369345302
6.87259603503138
6.861701701009434
6.741860012770636
7.935026153144596
7.3868494254421995
7.4485287086754
6.650143493546521
6.602056999482184
7.385532680780708
7.327146671638493
7.2531173316824455
7.27415690853298
6.625959504292052
6.844047135661839
7.868786293069375
7.3645900711807775
7.569931247602395
7.581141866253126
7.4557072171788485
7.308225285719996
7.431646157001869
7.6841073257899835
6.568733581279304
6.8658053727768635
7.148459688023278
7.145963751952662
7.7120347772961075
7.542433889883427
6.921297018497314
6.689071814744345
6.557149312370294
7.028354409784169
6.538929547677602
6.68780803904997
7.253393490865092
7.442879262508311
7.10250277117885
7.153273597615337
7.112477272190721
8.165117444269233
7.397305833974468
7.587369167657559
7.02964903631733
8.111593722219459
6.740729450688552
7.071349414724705
```

```
7.149844103512134
6.838718092405031
6.846797886853649
7.64053312520392
6.721735714427058
6.6588158327323494
7.099831128268459
6.846476578025106
6.7948993181460775
7.654498325665887
6.791926647305779
7.265634614971935
8.528438961115565
7.941639219297901
7.723751540462788
7.457683072747881
7.069091227588784
7.679911455662335
6.915265774897886
6.715651593880544
7.515243815546996
7.648631428093878
6.8691056049057355
7.079536872617588
7.088646035201124
7.219511118660631
6.90031069063757
7.012490274286792
6.934255118225489
7.7783934495235805
7.619468913166255
7.765664123012927
7.0029953636624915
7.012139687018411
6.612684075953517
6.781074482593492
6.385246307352465
7.068824424254726
6.8118688736990505
8.359853771908229
7.096244778946035
7.201149266926566
6.2936531801153395
7.1512745507212
7.06211420043901
7.4208992292376275
9.04643617932293
6.803173400971167
7.93735248520777
7.122490497327309
7.026504277550045
6.428248008612676
6.876219240946418
7.7797235655565595
7.212512190496856
6.962470153744741
8.029527326020226
```

```
6.786247221352412
7.292304951935121
7.274460694398509
7.036427173459774
6.639977993415158
7.182860624529121
6.909668640493169
7.285245531875842
7.055953031541081
6.965219348145136
7.697961345562112
6.585415659395365
6.737913705832866
7.581278201907318
7.323509350616813
6.617594241471937
8.907594864271699
12.33794575119148
8.74524534784327
7.0807087919600695
6.294761574460481
7.705435454094999
6.732954887511498
6.631650402968317
6.937510785328666
6.895359421456305
7.732527148545809
6.847846596514798
7.271411970523949
6.788310068612878
7.472224862378534
6.692072227686728
12.25826443189875
6.06907235247408
6.094191523049101
6.1439812772514415
7.475653723636684
6.905012496297536
6.727440399735491
6.8858945004988685
6.481350771344885
6.974663841762179
6.363434303725103
6.50237145005652
7.269590879949762
6.715682908429085
6.739892080683869
6.1189853844165425
6.376278179385376
6.513216475986062
7.5606437219999965
6.453377335403825
7.306747879083073
6.830180766415861
7.2160991922082705
7.1296836988353265
6.953359201267149
```

```
6.6661199743502895
7.267096547434271
7.2064018067159346
7.8771123001119285
7.4266082876145045
6.552975099040223
7.928973452521576
8.073791054004467
7.322919574606422
7.024916252496124
7.073663899807465
6.594709390569071
7.488736557622401
7.9628193826970906
7.035337721277371
7.111425469113977
7.595040272552463
6.699280205742898
7.566472407766846
6.600323952705225
7.835901436013268
7.869674685760557
6.574704148297256
8.935889054162894
6.94497300028253
7.519781015300676
7.711092132583791
6.755473183222732
7.15119384367056
7.898221877590444
7.684498572579246
7.296693917883838
6.539382011096925
7.286396167368258
7.1808465979320495
7.695854231738764
7.031943488239588
7.272547756469613
6.854980521631455
6.911157003425194
7.778077494902568
7.032477909823748
8.122496883844299
7.1589506066732564
6.812440590596414
7.454692557460036
6.99921285074231
7.099830267756901
7.34767496148382
7.362716728959743
8.318567632500063
7.343414663025137
6.609472718636752
7.924896686587528
7.6080864717235 31
7.034282361308365
8.044422762778629
```

```
7.384790624342147
7.172914686994635
6.481538193151157
6.851371181767997
7.052446916617767
9.239162640970068
7.414681173647249
6.859805184590091
7.4866661166953925
7.0508863960454216
6.441569791467622
7.519644242589132
8.009046147329263
7.973773387535123
8.112025928026032
6.898024158848828
6.712041030485616
7.877340755490414
7.597037065420893
6.789656974489733
7.213112045817463
6.822903845760876
7.2316532960340005
7.474420038188052
7.608285338742116
6.578239253719579
7.6638638320311925
9.598344715000096
6.526151841436819
6.645834934323151
7.579323363642403
7.033065588706636
7.285176684620661
7.567202577777259
7.37265864331313
6.891919355098081
7.160144887442524
7.025824917830879
7.76712832304042
6.916363150762576
6.750531749543674
7.63709237840787
7.251163072852938
6.878585661084545
7.615037667352221
7.017193525025582
7.91603281103505
6.238829823378875
6.623488033345727
6.575635368459324
6.3523846928247885
6.630373944363551
6.98226241017353
6.7512135699476055
6.736060853554288
7.291033905708983
6.643206437930423
```

```
6.8994463804109225
7.19479600073271
6.96991454518711
7.3938297872013905
6.989306724008651
8.232543992103391
7.242605192627577
7.408631094142261
7.496450178491353
6.929352495569326
7.947962408697459
7.869535780225408
8.229375419760853
6.434582229414321
7.003957819172734
6.795609145486154
7.674268684526833
6.62852265500959
8.152969646673938
7.295995921927368
7.1010497397592
6.816102533792395
6.300798993578881
8.370685102549999
6.585500925141041
7.446981274874872
7.4263779243649575
8.149123770019173
7.22498869832174
7.036691180429588
12.132864012269875
6.48089753484235
7.579436711030838
7.26372183259721
7.70284197405903
8.213125255463277
7.112433187855073
7.454079487950021
6.7002259218485785
6.678945101153638
6.945819905601369
6.450477512986746
6.9673671563212665
8.285888892468465
6.517068076423018
7.101321853019801
7.238846856046116
7.128422710568349
7.341511969273351
7.138949355798592
7.211212067527966
7.5969046393161745
7.03369011860384
6.861009088367582
6.779588671240206
7.213612196117229
7.001932046370038
```

```
7.3065993448141455
7.847600418375871
7.407989787087556
6.848454395409611
7.142790720428397
7.751893084820036
7.291572999293741
6.254506055497525
6.583370465134919
8.0214258968661
7.669724205649706
7.689797850132612
6.254506055497525
6.627850886192081
6.842786361379731
7.117062147959599
7.68884148241968
7.899702084839925
7.285203158018983
6.4817794443099706
6.6779938104431364
8.40727886773569
7.974068241564475
8.225790154030726
6.539614191763694
6.9847327840903795
7.164361197016911
7.06506488030723
7.9800368403976485
6.5313804070946695
6.9039230062344155
7.34266562749279
7.161708440052736
6.374673777901617
6.623266975685865
7.106010326131001
7.919151411387484
6.453373620482855
6.645930471697402
7.501930179784688
7.763900333473426
7.521250633204612
8.07526318217773
6.6170693279228585
7.247306770550874
6.984280178502123
7.954467454779056
6.966213213558749
8.077685138796332
7.888601362572357
7.662186658948062
6.434918469963924
6.521550200977214
8.391903299988996
7.216331809884015
7.06684082643118
6.9202043351109115
```

```
6.43310700976896
7.389743936265466
7.013643756380168
7.73095674561622
7.875170930482372
7.913250621806576
7.208702097140578
7.274800370213944
7.432959876587335
6.748087791792732
7.196188929634547
7.2682927709589835
7.572665557884992
7.608822627604336
7.069275023221431
6.899030352278577
7.477553962438956
7.006867660380664
7.605127066048813
7.8620431370050845
6.78984316705685
7.463943050496824
9.645672704692657
6.642097691140461
8.135596923274742
7.423605370166066
6.7357925238196295
6.807837851868481
7.391130109420658
7.021835780762711
7.55433311311357
6.883366952162353
7.616153269945586
6.493065492679234
7.369514180467038
6.8667303596129265
8.286331793685148
7.259566547409192
6.87931070150235
7.331102100167182
6.697768205465692
7.529837381626937
6.587690089522086
7.294142928256345
100%|████████████████████████████████████████████████| 500/500 [01:10<00:00,
7.04trial/s, best loss: 6.06907235247408]
```

In [42]:
```
# Return best parameters

space_eval(space, best_params)
```

Out[42]:
```
{'colsample_bytree': 0.7000000000000001,
 'eval_metric': 'rmse',
 'gamma': 11.5,
 'learning_rate': 0.30000000000000004,
 'max_depth': 13,
 'min_child_weight': 8,
 'n_estimators': 280,
 'objective': 'reg:squarederror',
 'reg_alpha': 2.0,
 'reg_lambda': 47.5,
 'subsample': 0.5}
```

In [43]:
```
# Create optimized model

model_1 = XGBRegressor(colsample_bytree = 0.7,
                       gamma = 11.5,
                       learning_rate = 0.3,
                       max_depth = 13,
                       min_child_weight = 8,
                       n_estimators = 280,
                       reg_alpha = 2.0,
                       reg_lambda = 47.5,
                       subsample = 0.5,
                       objective = 'reg:squarederror')
```

In [44]:
```
# Fit

model_1.fit(X_train_1, y_train_1,
              eval_set = [(X_train_1, y_train_1), (X_test_1, y_test_1)],
              eval_metric = 'rmse',
              verbose = True,
              early_stopping_rounds = 10)
```

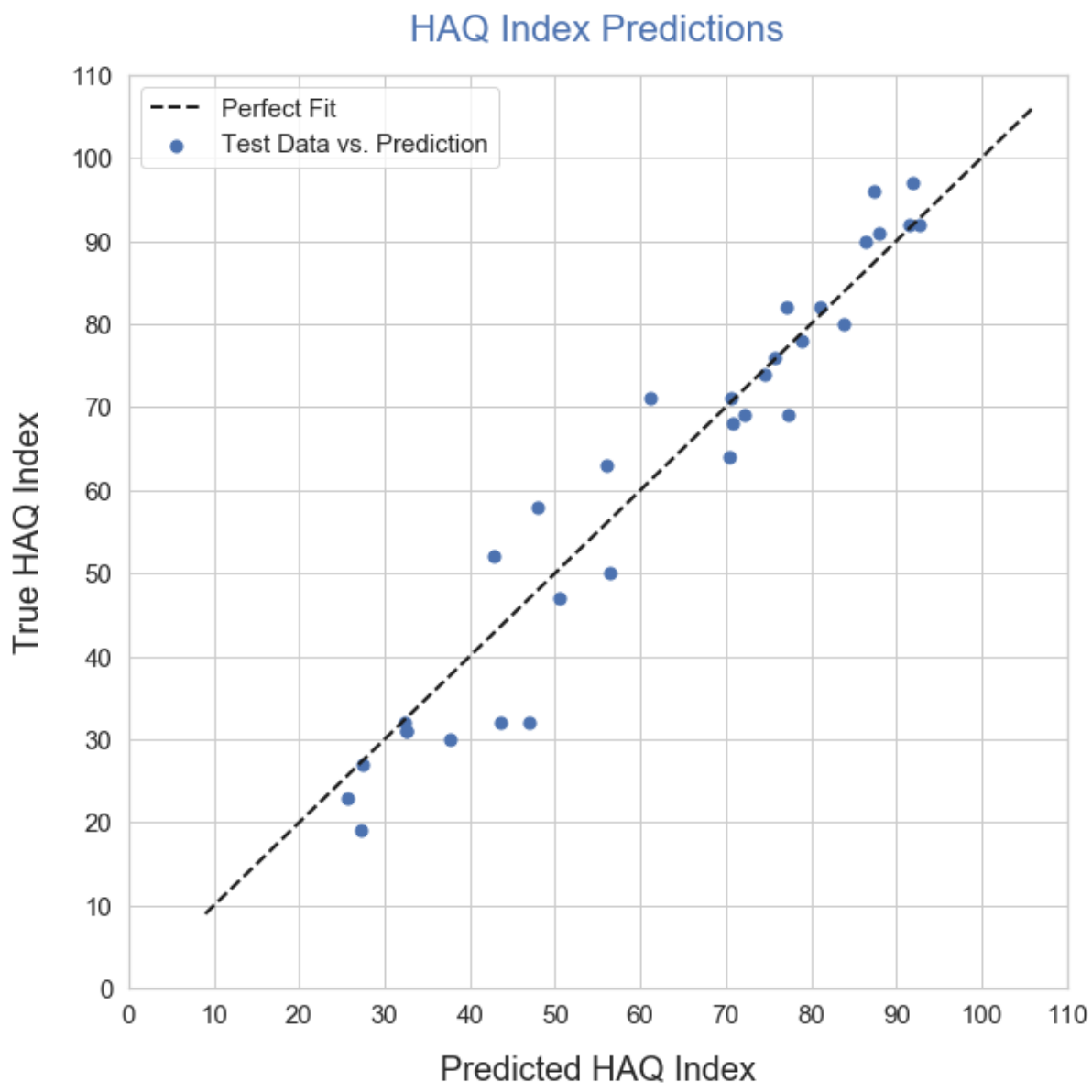
```
[0]     validation_0-rmse:57.189        validation_1-rmse:55.6922
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for
early stopping.

Will train until validation_1-rmse hasn't improved in 10 rounds.
[1]     validation_0-rmse:49.1779       validation_1-rmse:47.8821
[2]     validation_0-rmse:42.1031       validation_1-rmse:40.9627
[3]     validation_0-rmse:36.8084       validation_1-rmse:35.7901
[4]     validation_0-rmse:32.1514       validation_1-rmse:31.2491
[5]     validation_0-rmse:28.7791       validation_1-rmse:28.1268
[6]     validation_0-rmse:25.2245       validation_1-rmse:24.5759
[7]     validation_0-rmse:22.9984       validation_1-rmse:22.3284
[8]     validation_0-rmse:20.5078       validation_1-rmse:19.7732
[9]     validation_0-rmse:18.7698       validation_1-rmse:18.1812
[10]    validation_0-rmse:17.3946       validation_1-rmse:16.8539
[11]    validation_0-rmse:15.9217       validation_1-rmse:15.2939
[12]    validation_0-rmse:14.7118       validation_1-rmse:14.009
[13]    validation_0-rmse:13.8247       validation_1-rmse:13.0809
[14]    validation_0-rmse:13.0072       validation_1-rmse:12.3089
[15]    validation_0-rmse:12.3189       validation_1-rmse:11.6028
[16]    validation_0-rmse:11.6628       validation_1-rmse:10.9877
[17]    validation_0-rmse:11.1392       validation_1-rmse:10.4575
[18]    validation_0-rmse:10.511        validation_1-rmse:9.87743
[19]    validation_0-rmse:10.0419       validation_1-rmse:9.43994
[20]    validation_0-rmse:9.63143       validation_1-rmse:9.11682
[21]    validation_0-rmse:9.22318       validation_1-rmse:8.76815
[22]    validation_0-rmse:8.95558       validation_1-rmse:8.67343
[23]    validation_0-rmse:8.62912       validation_1-rmse:8.4731
[24]    validation_0-rmse:8.33733       validation_1-rmse:8.3083
[25]    validation_0-rmse:8.11496       validation_1-rmse:8.04827
[26]    validation_0-rmse:7.87107       validation_1-rmse:7.82756
[27]    validation_0-rmse:7.69083       validation_1-rmse:7.66151
[28]    validation_0-rmse:7.50681       validation_1-rmse:7.56733
[29]    validation_0-rmse:7.3062        validation_1-rmse:7.39902
[30]    validation_0-rmse:7.10135       validation_1-rmse:7.27391
[31]    validation_0-rmse:7.00115       validation_1-rmse:7.20213
[32]    validation_0-rmse:6.85609       validation_1-rmse:7.05332
[33]    validation_0-rmse:6.73473       validation_1-rmse:6.95566
[34]    validation_0-rmse:6.65111       validation_1-rmse:6.8711
[35]    validation_0-rmse:6.54654       validation_1-rmse:6.75178
[36]    validation_0-rmse:6.38576       validation_1-rmse:6.73305
[37]    validation_0-rmse:6.32618       validation_1-rmse:6.69609
[38]    validation_0-rmse:6.24108       validation_1-rmse:6.59191
[39]    validation_0-rmse:6.20857       validation_1-rmse:6.54519
[40]    validation_0-rmse:6.11862       validation_1-rmse:6.46052
[41]    validation_0-rmse:6.01036       validation_1-rmse:6.46701
[42]    validation_0-rmse:5.9825        validation_1-rmse:6.43698
[43]    validation_0-rmse:5.94397       validation_1-rmse:6.40769
[44]    validation_0-rmse:5.89973       validation_1-rmse:6.35237
[45]    validation_0-rmse:5.84903       validation_1-rmse:6.32668
[46]    validation_0-rmse:5.81287       validation_1-rmse:6.32105
[47]    validation_0-rmse:5.76635       validation_1-rmse:6.31389
[48]    validation_0-rmse:5.71895       validation_1-rmse:6.29487
[49]    validation_0-rmse:5.69976       validation_1-rmse:6.29562
[50]    validation_0-rmse:5.65535       validation_1-rmse:6.20943
[51]    validation_0-rmse:5.63568       validation_1-rmse:6.20051
[52]    validation_0-rmse:5.61182       validation_1-rmse:6.20098
```

```
[53]    validation_0-rmse:5.57822       validation_1-rmse:6.14155
[54]    validation_0-rmse:5.54504       validation_1-rmse:6.13651
[55]    validation_0-rmse:5.52302       validation_1-rmse:6.10436
[56]    validation_0-rmse:5.51194       validation_1-rmse:6.06917
[57]    validation_0-rmse:5.49035       validation_1-rmse:6.06907
[58]    validation_0-rmse:5.44343       validation_1-rmse:6.10545
[59]    validation_0-rmse:5.40344       validation_1-rmse:6.12718
[60]    validation_0-rmse:5.39536       validation_1-rmse:6.14155
[61]    validation_0-rmse:5.3531        validation_1-rmse:6.17362
[62]    validation_0-rmse:5.30952       validation_1-rmse:6.1717
[63]    validation_0-rmse:5.29359       validation_1-rmse:6.17287
[64]    validation_0-rmse:5.28788       validation_1-rmse:6.2065
[65]    validation_0-rmse:5.27778       validation_1-rmse:6.18599
[66]    validation_0-rmse:5.2757        validation_1-rmse:6.2131
[67]    validation_0-rmse:5.26165       validation_1-rmse:6.19659
Stopping. Best iteration:
[57]    validation_0-rmse:5.49035       validation_1-rmse:6.06907
```

Out[44]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, gamma=11.5,
             importance_type='gain', learning_rate=0.3, max_delta_step=0,
             max_depth=13, min_child_weight=8, missing=None, n_estimators=28
0,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=0, reg_alpha=2.0, reg_lambda=47.5, scale_pos_weight
=1,
             seed=None, silent=None, subsample=0.5, verbosity=1)
```

In [45]:
```
# Predict

y_pred_1 = model_1.predict(X_test_1)
```

In [46]:
```
# Compare predictions, y_pred_1, to test values, y_test_1, using scatterplot

# create line to represent perfect fit to y_test

y_line_1 = np.arange(int(y_test_1.min()) - 10, int(y_test_1.max()) + 10)

# set axes limits - adjust if necessary
x_min_1 = 0
x_max_1 = y_test_1.max() + 10
d_x_1 = 10

y_min_1 = 0
y_max_1 = y_test_1.max() + 10
d_y_1 = 10

plt.figure(figsize = (10, 10))
ax = plt.axes()

ax.set_xlim(x_min_1, x_max_1)
ax.set_xticks(np.arange(x_min_1, x_max_1 + d_x_1, d_x_1))

ax.set_ylim(y_min_1, y_max_1)
ax.set_yticks(np.arange(y_min_1, y_max_1 + d_y_1, d_y_1))

plt.scatter(y_pred_1, y_test_1, s = 50, c = 'b', label = 'Test Data vs. Predic
tion')
plt.plot(y_line_1, y_line_1, 'k--', lw = 2, label = 'Perfect Fit')
plt.xlabel('Predicted HAQ Index', fontsize = 20, labelpad = 15)
plt.ylabel('True HAQ Index', fontsize = 20, labelpad = 15)
plt.title('HAQ Index Predictions', fontsize = 22, c = 'b', pad = 20)
plt.legend(fontsize = 15)
plt.tick_params(labelsize = 15)
plt.show()
```



In [47]:
```
# The predicted HAQ Index values are close to the true test values
```

In [48]:
```
# Get RMSE as a measure of predictions accuracy

# Absolute RMSE
rmse_1 = np.sqrt(metrics.mean_squared_error(y_test_1, y_pred_1))

# Normalized RMSE --> more adequate measure for comparison with other models - 
ptovides the error in terms of data avg.
rmse_1_norm = rmse_1/y_test_1.mean()

print('Absolute RMSE_1:', round(rmse_1, 4))
print('Normalized RMSE_1:', round(rmse_1_norm, 4))
```

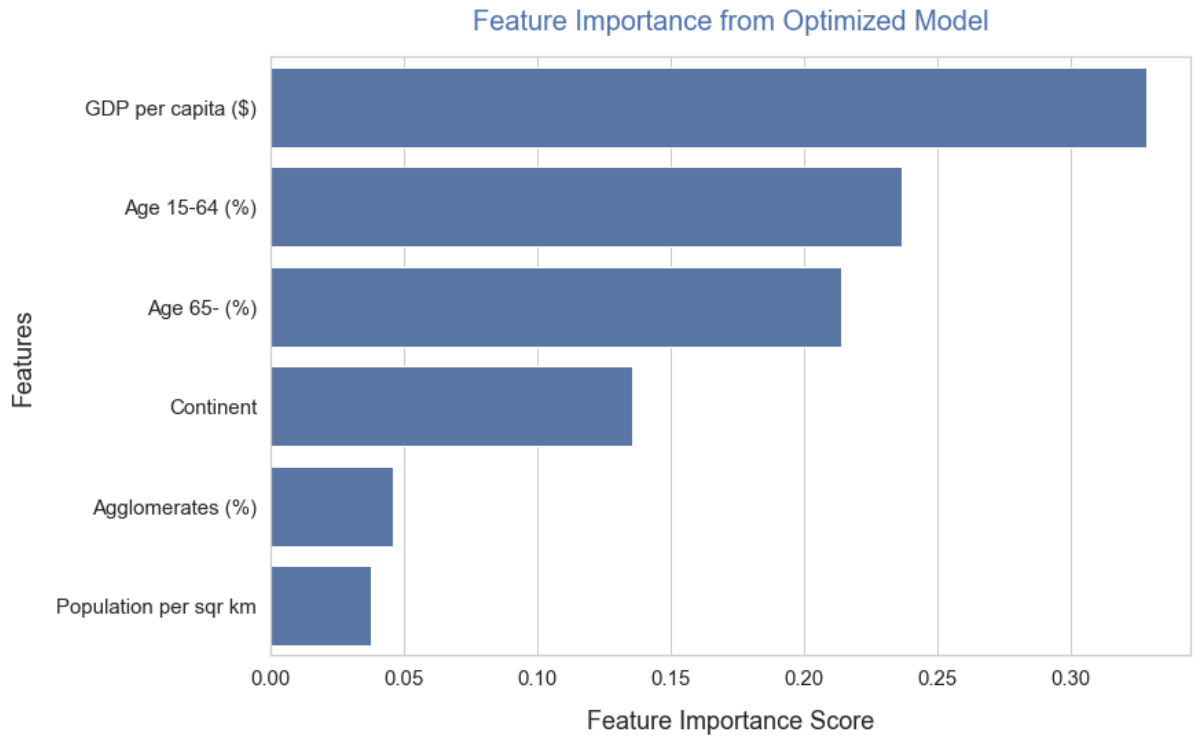
```
Absolute RMSE_1: 6.0691
Normalized RMSE_1: 0.0986
```

In [49]:
```
# The results indicate good model accuracy --> predictions error is only 10 % 
 of the test values average
```

In [50]:
```
# Get feature importances

feature_imp = pd.Series(model_1.feature_importances_,index = data_2.iloc[:, :-
2].columns).sort_values(ascending = False)
feature_imp
```

Out[50]:
```
GDP per capita ($)       0.328757
Age 15-64 (%)            0.237040
Age 65- (%)              0.214159
Continent                0.135992
Agglomerates (%)         0.046097
Population per sqr km    0.037955
dtype: float32
```

In [51]:
```
# Visualize feature importances

plt.figure(figsize=(12,8))

sns.barplot(x=feature_imp, y=feature_imp.index, color = 'b')

plt.xlabel('Feature Importance Score', fontsize = 18, labelpad = 15)
plt.ylabel('Features', fontsize = 18, labelpad = 15)
plt.title('Feature Importance from Optimized Model', fontsize = 20, pad = 20,
c = 'b')
plt.tick_params(labelsize = 15)

plt.show()
```



In [52]:
```
# Main observations:
    # 1) Top features by impotance are GDP per capita and the two age groups a
s already indicated by the correlation matrix
    # 2) The two population density features are the least important in determ
ining HAQ Index values
```

In [53]:
```
# This concludes Section 2
# It has been demonstrated that an ML model using the features selected here c
an accurately predict
# the strictly medical countries ratings, HAQ Index
```

In [54]:
```
# 3) Predicting Deaths per 100K based on conventional factors using XGBoost
```

In [55]:
```
# Create features and target from data_2 which we will use with XGBRegressor m
odel

X = data_2.iloc[:, :-1].values # all columns, but last - include HAQ Index in
 predicting Deaths per 100K
y = data_2.iloc[:, -1].values # Deaths per 100K column
```

In [56]:
```
# Train/test split

X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X, y, test_size =
0.2, random_state = 0)
```

In [57]:
```
# For best predictions, apply model optimization by using hyperopt

# Create hyperparameter space to search over
space = {'max_depth': hp.choice('max_depth', np.arange(3, 15, 1, dtype = int
)),
        'n_estimators': hp.choice('n_estimators', np.arange(50, 300, 10, dtype
= int)),
        'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1.0, 0.1),
        'min_child_weight': hp.choice('min_child_weight', np.arange(0, 10, 1,
dtype = int)),
        'subsample': hp.quniform('subsample', 0.5, 1.0, 0.1),
        'learning_rate': hp.quniform('learning_rate', 0.1, 0.3, 0.1),
         'gamma': hp.choice('gamma', np.arange(0, 100, 0.5, dtype = float)),
         'reg_alpha': hp.choice('reg_alpha', np.arange(0, 100, 0.5, dtype = fl
oat)),
         'reg_lambda': hp.choice('reg_lambda', np.arange(0, 100, 0.5, dtype =
float)),

        'objective':'reg:squarederror',

        'eval_metric': 'rmse'}


def score(params):
    model = XGBRegressor(**params)

    model.fit(X_train_2, y_train_2, eval_set=[(X_train_2, y_train_2), (X_test_
2, y_test_2)],
              verbose=False, early_stopping_rounds=10)
    y_pred = model.predict(X_test_2)
    score = np.sqrt(metrics.mean_squared_error(y_test_2, y_pred))
    print(score)
    return {'loss': score, 'status': STATUS_OK}

def optimize(trials, space):

    best = fmin(score, space, algo = tpe.suggest, max_evals = 500)
    return best

trials = Trials()
best_params = optimize(trials, space)
```

```
7.080724824919022
7.153799810487325
7.220746556792694
7.0778522589409825
7.138082297057355
7.060534996527395
7.161802874906284
7.152225214994175
7.074803911843048
7.211705960190338
7.286656747383318
7.089922393947877
7.2709422162154995
7.244082827190149
7.241077028032326
7.13439762978727
7.3929109486553575
7.463897493827197
7.265801930987095
7.151955248737144
7.907944083040197
7.167776714028012
7.343573794310002
7.333653287478079
7.850815069440825
7.235792986622154
7.412471808215239
7.188753232793955
7.249471925584994
7.727629687499759
7.279155461461404
7.467671536000229
7.218545196653373
7.125264615632772
7.287246469460565
7.240444962316337
7.253218322192854
7.097104314987632
7.323361746165896
7.188217628137424
7.388787711148882
7.25097117300813
7.349238201567938
7.206679083932706
7.198942131462314
7.182209741904847
7.049747601412907
7.242203890821911
7.255164300095274
7.177032984333989
7.3661026022899385
7.235964775417583
7.426189507802996
7.37500243058129
7.2145145074448465
7.265096956021273
8.006884215614273
```

```
7.353896353253967
7.174250225011218
7.2161867686138725
7.341389593968872
7.5126838495214
7.302820814370277
8.103706838332052
7.241797340738912
7.790892715082642
7.275051429316933
7.199043502237541
7.313797825681858
7.101596621250465
7.244032250786025
7.376182736519525
7.332997907355686
7.175085909593066
7.213060036985105
7.204204282789661
7.139464619744223
8.120647338134777
7.361898820742894
7.064413321289655
7.191580488847176
7.352741814030002
7.124501783467842
7.1296304143307525
7.140400233124718
7.068519287563734
7.159974976681563
7.430824946728332
7.2819134953629465
7.124727728636241
7.647994114749438
7.447879861840562
7.136161688417262
8.120535088427374
7.1731555386101515
7.303528825870618
7.826605548799454
7.073137570413331
7.355694132891351
8.087300324519866
7.240108430922937
6.938061830830337
7.124132767668816
7.037899370158308
7.113062636567712
7.109015556440838
7.053920809357551
7.042121089202653
7.145731629078025
6.974196269864554
7.111663429349924
7.044513835662173
7.021212130634198
7.35085850348899
```

```
7.703320966654192
6.955737816419246
7.032480206521089
6.9281148052170245
6.9948436395486056
7.343871264399826
7.081184195489364
7.619711100334444
7.208199877476712
7.0256718112103815
7.1327680819346755
7.866447891054111
7.5979787801100365
7.253924245217513
7.3077736421225366
7.21811153390477
7.152865479961378
7.2202726414363
7.107994878475109
7.400508450237699
7.1793825299736564
6.998545587289379
7.54329099223328
7.118250871828185
7.033425513279967
7.2960295950922855
7.089500797686838
7.3380776515004715
7.1728731917059
8.121153670046036
7.1015052878904354
6.9197203512938215
7.253849784990241
7.0501042137305285
6.929965832575635
7.074541056099934
7.021224100694776
7.137839348775176
7.200458723327951
7.069110585838941
7.068882581858745
7.128405524450724
7.287980921217918
7.420559671893463
7.119446071339515
7.59344199388475
7.468161531758181
7.163599199471054
7.1582718896095265
7.091090714131173
7.1359201538358725
7.3212592029011185
7.5009656945282845
7.055237045980978
7.846714519247582
7.253039828486253
7.444586394688259
```

```
7.157187782149567
7.125895688360294
7.112235667427693
7.065579830397285
7.396839136034369
7.203793883276805
7.410647359788117
7.571245302389598
7.014484378825432
7.365707327386881
7.149716346660961
7.028298544170979
7.143857152114458
7.261609905442816
7.114858213886365
7.223561348750294
7.0547825696847974
7.067404412137582
7.096249251449244
7.2796207944847
7.2943579639589675
7.375218244341113
7.064291540090924
7.157824824191212
7.827479109868088
7.423225909411423
7.116025768823746
7.672019604684142
7.184217692193467
7.281471605804573
7.185727284298424
7.050530194708439
7.243060528243064
7.244237192566655
7.143825582269877
7.225030702473799
7.118885758961138
7.204157784191601
7.439209817403922
7.200783866025005
7.013412150264204
7.151858336757062
7.157720304638733
7.183201765312971
7.069174069359226
7.328810395565491
7.0975951766008425
7.000314715646217
7.862690808844466
7.199824000092056
7.178194183560025
7.160091286086294
7.192855043605587
7.077329250300551
7.0540328036715945
7.727351440458105
7.367129399701756
```

```
7.219446654759494
7.11623634883939
7.230151411780587
7.102590881807437
7.073506348232298
7.0352644496841465
7.365968890029073
7.219947532204809
6.961703611770295
7.084446988724139
7.08218161553636
7.209482336512887
7.147239766212544
7.201844023583824
7.125977189895094
8.16561959110845
7.246898699039255
7.593203840948174
7.081824632163211
7.249656209075869
7.140888938411463
7.4530357870344925
7.174419769496754
7.294165234174487
7.157298838346131
7.317819103674886
7.161579464838029
7.229999997779119
7.146530234682497
7.074039140784823
7.045609166809864
7.0627986265543115
7.108446235734004
6.992530240587657
6.9551768739435245
7.018734074638611
6.938237938518362
7.113440504682576
7.114702760735416
7.128957386894429
7.194140913880836
7.068729150768735
7.583817420299679
7.113488288064057
7.154672599366931
7.418495344438207
7.261034808603203
7.64947801999829
7.145301391963924
7.892946865044209
7.031931647776329
7.058971983165411
7.057104178742556
7.958130111216024
7.121033541319307
7.12113874299434
7.455121121743357
```

```
7.151468403886366
7.1845407112489506
7.2037593715599195
7.132720993098262
7.70777541636483
7.114567527446456
7.743618929033954
7.169945940734263
7.074398717677077
7.256991593129282
7.0782747454611705
8.015337125917064
7.1406872755383395
7.234150269739336
7.223935166513015
7.060030396913341
7.083399389443109
7.734790554375121
7.202522075281703
7.378020795848458
7.028045621770716
7.105785205202795
7.077823184803207
7.087758640485114
7.388996256023744
7.268412436571133
7.23083030362654
7.989542339776409
7.062377647754986
7.700465429442866
7.224707059289661
7.2715212271140635
7.5085808807503795
7.086259079218908
7.337927522314558
7.445567224515401
6.9555829862465774
7.33690279643673
7.104493298079041
7.1761491611043855
7.2671187219275115
7.090976925052441
7.1469906093450435
7.084123646629721
7.061839031000017
7.668243672897807
7.105324449075506
7.571034553437294
7.915853130355442
7.296807593086355
7.154408107753404
7.187903804881903
7.5673081222977245
7.322601247722972
7.270377029258966
7.057281876848615
7.576951156281051
```

```
7.001384137512029
7.21235823164795
7.282484183585037
7.486005691678398
7.36025806527862
7.189504917959857
7.179757098295147
6.993199815600217
7.214005555658374
7.2007500564326525
7.07395881013249
7.078415598786964
7.166951441398508
7.375407964159269
7.337648060041662
7.157611575146288
7.09815211390121
7.669400137189625
6.9941668527699585
7.139835356997081
7.337723424392667
7.222120888132629
7.230660108799002
7.131878294170774
6.983844224941336
7.106919023664817
7.022202524755665
7.204133853680916
7.01889831479096
7.040490931361093
7.3134677438662585
7.320550726393659
7.260612669177643
7.212994221297235
7.195756627152486
7.047110524296128
7.12930948900953
7.070162904178161
7.2082680498427525
7.428377181609533
7.137208562996889
7.1948947224611315
7.0079155301088605
7.014031824752394
7.610254804287974
7.119423176997699
7.109297124849715
7.294061712581495
7.11408535894464
7.126795645745117
7.314322478277798
7.074308412328024
7.282087786641269
7.214641952156867
7.143956337797868
7.050885567878653
7.14140893880395
```

```
7.497907840667322
7.033860999724567
7.046548667915087
6.9698309974671355
7.028204645093824
7.0694466037868775
7.051483624641941
7.055967988334727
7.11096647725228
7.0062016229690105
7.105044433725155
6.959390388158079
7.619838761561478
7.033765549389738
7.028259790977215
7.337329590059787
7.0987107364477575
7.154485906755042
7.225529290195811
7.366346286363026
7.189945458010158
7.622575671159533
7.560172809213597
7.110681247584056
7.633543571996728
7.0330744346098735
7.079369064711974
7.206138470378536
7.036605049925133
7.138886374100958
7.344390093362811
7.201253048945498
7.416038417647148
7.3808280080071045
7.11905501056922
7.096364611360873
6.9683118455682305
7.400637506550092
7.049129500482214
7.1344031423439604
7.168617397065615
7.282957749075875
7.2179402815555935
7.1035419347318225
7.005196553691716
7.357939884798058
7.637111783320133
7.013863330122156
7.112815521253328
7.230942185417372
7.071197851856006
7.282735412946754
7.047536986065174
7.209016191986535
7.1745612971466475
7.148124868891364
6.970104584397697
```

```
7.140883910098164
7.360429838426933
7.23950586094795
7.122357817916433
7.659972143732129
6.999338501945682
7.028408636246633
7.083013364583942
7.365415572133402
7.190025239589957
7.199950027503509
7.0759377239027845
7.141557352326035
7.1216942645450025
6.99973839062037
7.1576607136916754
6.998814541275622
8.062438791910326
7.068163904318559
7.076179331940492
7.22261642697672
7.194159990681268
7.042502886716663
7.128284901893693
7.886436767894883
7.211906868975446
7.144488995689554
7.078465717387538
7.330285856635017
7.283010934265953
7.030514843551633
7.54455868813963
7.877348970767806
7.397532724739839
7.335811261596947
6.9888030236557315
7.407601202745469
7.015729660479349
7.623370076374431
7.140550473684706
7.267666024207686
7.349311181804249
7.048233747110725
7.134655135093744
100%|██████████████████████████████████████████████| 500/500 [00:43<00:00, 1
1.49trial/s, best loss: 6.9197203512938215]
```

In [58]:
```
# Return best parameters

space_eval(space, best_params)
```

Out[58]:
```
{'colsample_bytree': 0.5,
 'eval_metric': 'rmse',
 'gamma': 12.0,
 'learning_rate': 0.2,
 'max_depth': 12,
 'min_child_weight': 2,
 'n_estimators': 100,
 'objective': 'reg:squarederror',
 'reg_alpha': 12.0,
 'reg_lambda': 79.5,
 'subsample': 0.9}
```

In [59]:
```
# Create optimized model

model_2 = XGBRegressor(colsample_bytree = 0.5,
                       gamma = 12.0,
                       learning_rate = 0.2,
                       max_depth = 12,
                       min_child_weight = 2,
                       n_estimators = 100,
                       reg_alpha = 12.0,
                       reg_lambda = 79.5,
                       subsample = 0.9,
                       objective = 'reg:squarederror')
```

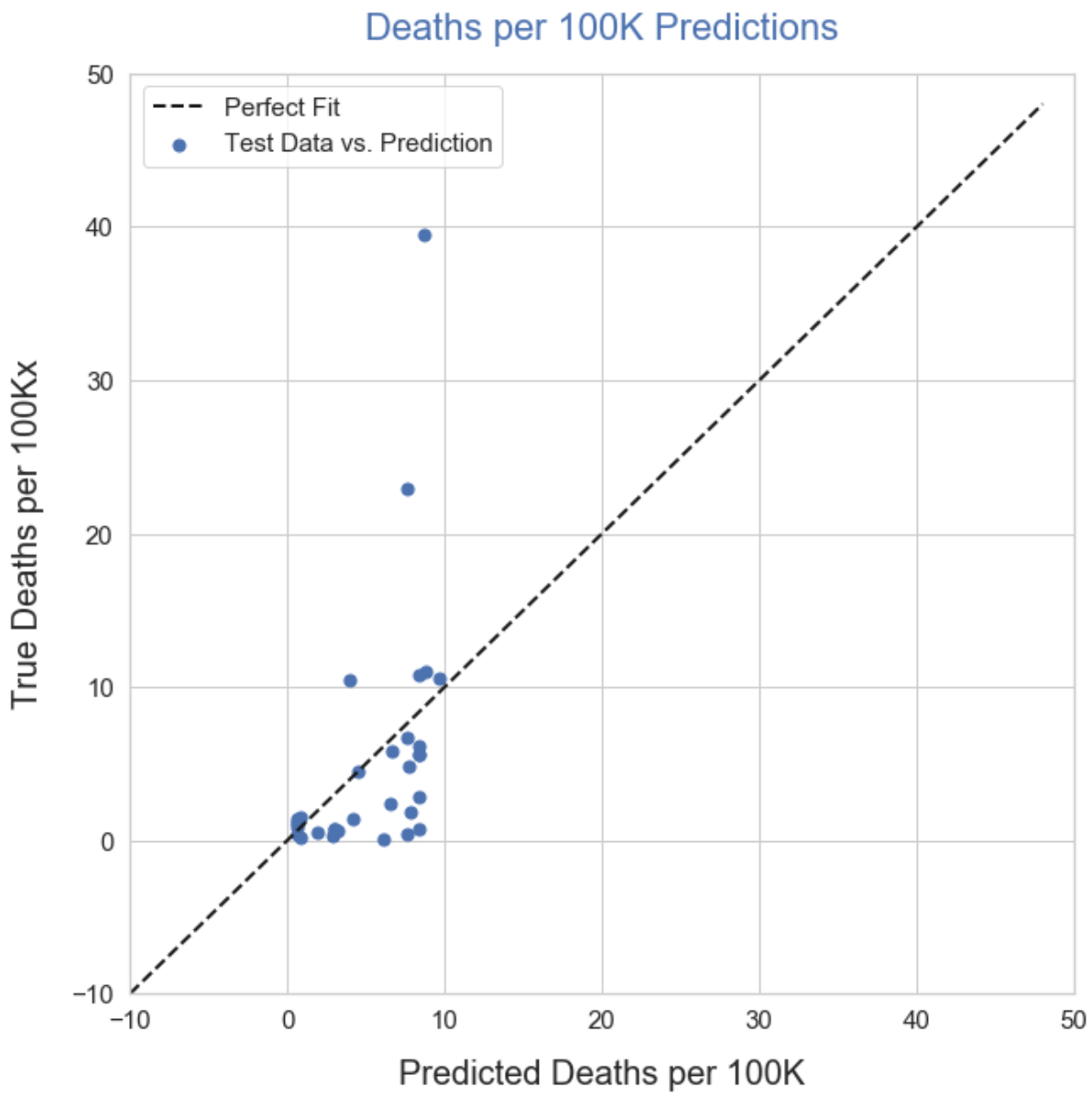
In [60]:
```
# Fit

model_2.fit(X_train_2, y_train_2,
             eval_set = [(X_train_2, y_train_2), (X_test_2, y_test_2)],
             eval_metric = 'rmse',
             verbose = True,
             early_stopping_rounds = 10)
```

```
[0]     validation_0-rmse:21.0956       validation_1-rmse:8.49323
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for
early stopping.

Will train until validation_1-rmse hasn't improved in 10 rounds.
[1]     validation_0-rmse:20.6016       validation_1-rmse:8.00299
[2]     validation_0-rmse:20.1741       validation_1-rmse:7.66919
[3]     validation_0-rmse:19.8005       validation_1-rmse:7.39805
[4]     validation_0-rmse:19.4435       validation_1-rmse:7.16879
[5]     validation_0-rmse:19.0954       validation_1-rmse:7.0543
[6]     validation_0-rmse:18.7616       validation_1-rmse:6.99261
[7]     validation_0-rmse:18.5428       validation_1-rmse:6.91972
[8]     validation_0-rmse:18.2525       validation_1-rmse:7.03263
[9]     validation_0-rmse:17.9517       validation_1-rmse:7.05119
[10]    validation_0-rmse:17.7099       validation_1-rmse:7.10291
[11]    validation_0-rmse:17.4309       validation_1-rmse:7.13194
[12]    validation_0-rmse:17.1909       validation_1-rmse:7.12756
[13]    validation_0-rmse:16.9223       validation_1-rmse:7.14373
[14]    validation_0-rmse:16.7003       validation_1-rmse:7.22595
[15]    validation_0-rmse:16.5203       validation_1-rmse:7.25176
[16]    validation_0-rmse:16.3861       validation_1-rmse:7.31156
[17]    validation_0-rmse:16.2539       validation_1-rmse:7.41089
Stopping. Best iteration:
[7]     validation_0-rmse:18.5428       validation_1-rmse:6.91972
```

Out[60]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.5, gamma=12.0,
             importance_type='gain', learning_rate=0.2, max_delta_step=0,
             max_depth=12, min_child_weight=2, missing=None, n_estimators=10
0,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=0, reg_alpha=12.0, reg_lambda=79.5,
             scale_pos_weight=1, seed=None, silent=None, subsample=0.9,
             verbosity=1)
```

In [61]:
```
# Predict

y_pred_2 = model_2.predict(X_test_2)
```

In [62]:
```
# Compare predictions, y_pred_2, to test values, y_test_2, using scatterplot

# create line to represent perfect fit to y_test

y_line_2 = np.arange(int(y_test_2.min()) - 10, int(y_test_2.max()) + 10)

# set axes limits - adjust if necessary
x_min_2 = -10
x_max_2 = y_test_2.max() + 10
d_x_2 = 10

y_min_2 = -10
y_max_2 = y_test_2.max() + 10
d_y_2 = 10

plt.figure(figsize = (10, 10))
ax = plt.axes()

ax.set_xlim(x_min_2, x_max_2)
ax.set_xticks(np.arange(x_min_2, x_max_2 + d_x_2, d_x_2))

ax.set_ylim(y_min_2, y_max_2)
ax.set_yticks(np.arange(y_min_2, y_max_2 + d_y_2, d_y_2))

plt.scatter(y_pred_2, y_test_2, s = 50, c = 'b', label = 'Test Data vs. Predic
tion')
plt.plot(y_line_2, y_line_2, 'k--', lw = 2, label = 'Perfect Fit')
plt.xlabel('Predicted Deaths per 100K', fontsize = 20, labelpad = 15)
plt.ylabel('True Deaths per 100Kx', fontsize = 20, labelpad = 15)
plt.title('Deaths per 100K Predictions', fontsize = 22, c = 'b', pad = 20)
plt.legend(fontsize = 15)
plt.tick_params(labelsize = 15)
plt.show()
```



In [63]:
```
# Clearly, there is significant portion of predictions which deviate dramatica
ly from the true test values
```

In [64]:
```
# Get RMSE as a measure of predictions accuracy

# Absolute RMSE
rmse_2 = np.sqrt(metrics.mean_squared_error(y_test_2, y_pred_2))

# Normalized RMSE --> more adequate measure for comperison with other models;
 ptovides the error in terms of data avg.
rmse_2_norm = rmse_2/y_test_2.mean()

print('Absolute RMSE_2:', round(rmse_2, 4))
print('Normalized RMSE_2:', round(rmse_2_norm, 4))
```

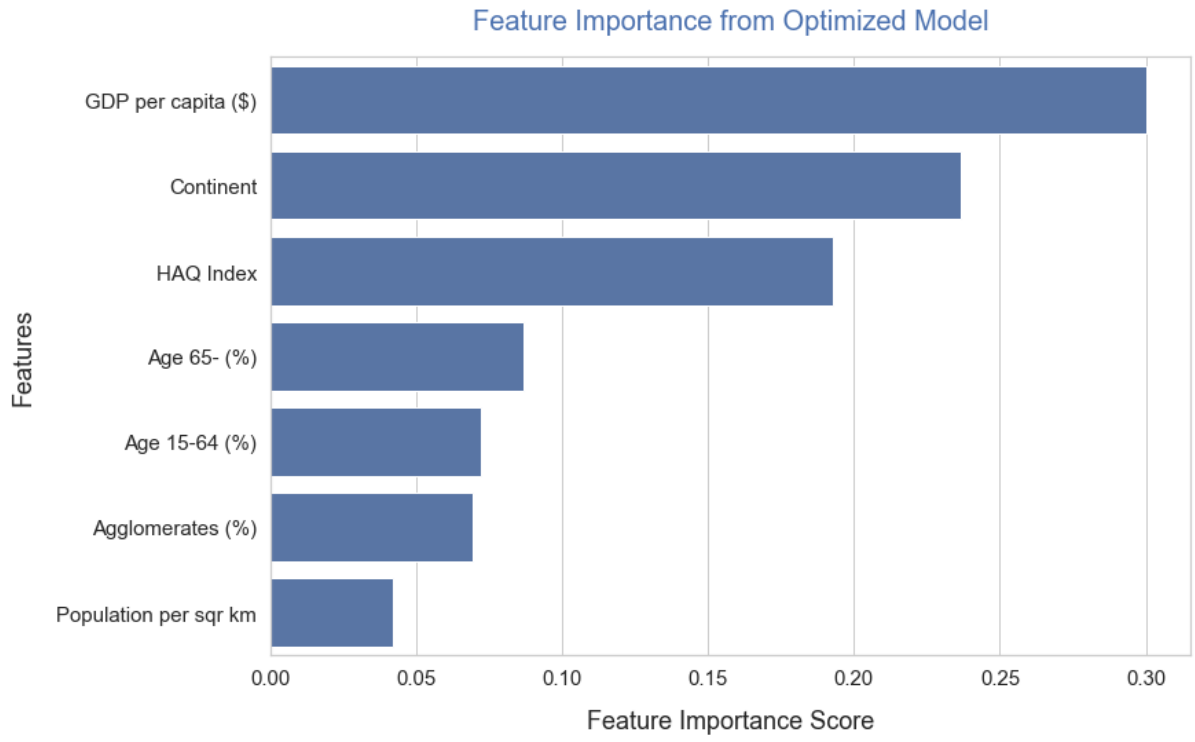
```
Absolute RMSE_2: 6.9197
Normalized RMSE_2: 1.3497
```

In [65]:
```
# As indicated by the scatter plot, the predictions have poor accuracy --> predictions error is 135 % of the test values average
```

In [66]:
```
# Get feature importances

feature_imp = pd.Series(model_2.feature_importances_,index = data_2.iloc[:, :-1].columns).sort_values(ascending = False)
feature_imp
```

Out[66]:
```
GDP per capita ($)       0.300316
Continent                0.236722
HAQ Index                0.192750
Age 65- (%)              0.086701
Age 15-64 (%)            0.072036
Agglomerates (%)         0.069342
Population per sqr km    0.042133
dtype: float32
```

In [67]:
```
# Visualize feature importances

plt.figure(figsize=(12,8))

sns.barplot(x = feature_imp, y = feature_imp.index, color = 'b')

plt.xlabel('Feature Importance Score', fontsize = 18, labelpad = 15)
plt.ylabel('Features', fontsize = 18, labelpad = 15)
plt.title('Feature Importance from Optimized Model', fontsize = 20, pad = 20, c = 'b')
plt.tick_params(labelsize = 15)

plt.show()
```

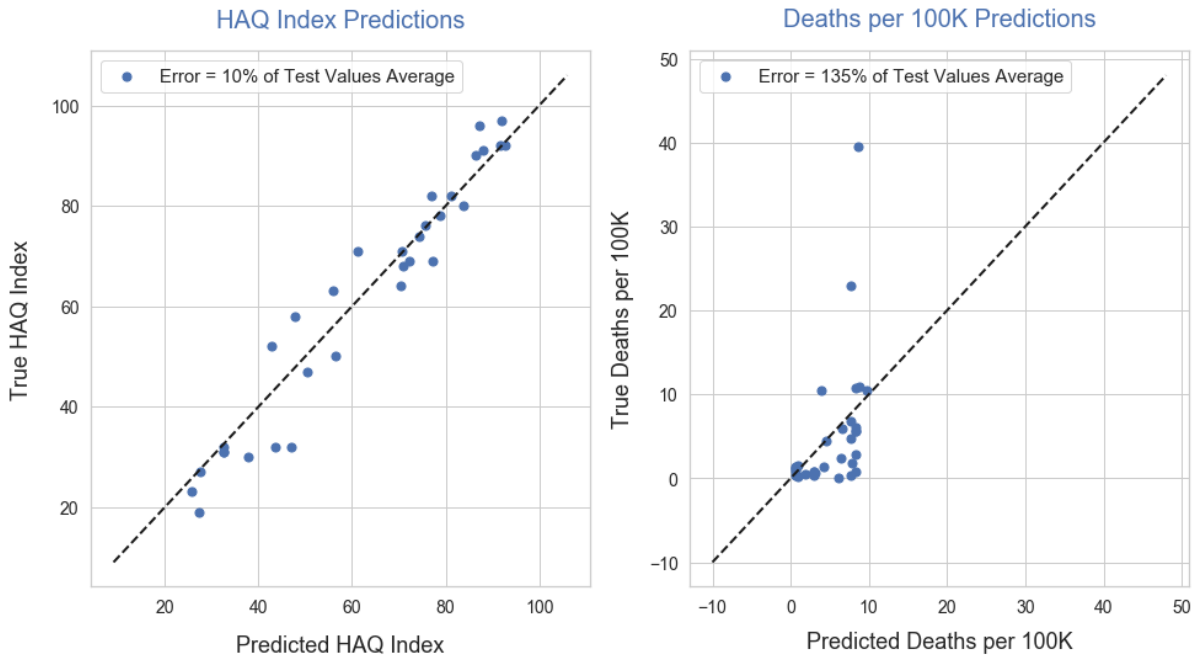


In [ ]:
```
# Main observations:
    # 1) Top feature by impotance are GDP per capita, Continent and HAQ Index
    # 2) Distant seconds are the two age groups-
    # 2) Population density features are again least important
```

In [68]:
```
# Conclusions from Section 3
    # 1) Deaths per 100K cannot be accurately predicted using the factors cons
idered here
    # 2) According to model Age Demografics and Population Density do not play
important role
```

In [69]:
```
# 4) Comparison between HAQ Index and Deaths per 100K predictions
```

In [70]:
```
# Demonstrate the contrast in predictions

fig, axes = plt.subplots(1, 2, sharey = False, figsize=(16,8))

# HAQ Index predictions
axes[0].scatter(y_pred_1, y_test_1, s = 50, c = 'b', label = 'Error = 10% of T
est Values Average')
axes[0].plot(y_line_1, y_line_1, 'k--', lw = 2)
axes[0].set_title('HAQ Index Predictions', fontsize = 20, c = 'b', pad = 20)
axes[0].set_xlabel('Predicted HAQ Index', fontsize = 18, labelpad = 15)
axes[0].set_ylabel('True HAQ Index', fontsize = 18, labelpad = 15)
axes[0].legend(fontsize = 15)
axes[0].tick_params(labelsize = 14)

# Optimized model predictions
axes[1].scatter(y_pred_2, y_test_2, s = 50, c = 'b', label = 'Error = 135% of
 Test Values Average')
axes[1].plot(y_line_2, y_line_2, 'k--', lw = 2)
axes[1].set_title('Deaths per 100K Predictions', fontsize = 20, c = 'b', pad =
20)
axes[1].set_xlabel('Predicted Deaths per 100K', fontsize = 18, labelpad = 12)
axes[1].set_ylabel('True Deaths per 100K', fontsize = 18, labelpad = 12)
axes[1].legend(fontsize = 15)
axes[1].tick_params(labelsize = 14)

plt.show()
```



In [71]:
```
# The two plots show dramatic difference between predicting HAQ Index and Deat
hs per 100K
```

In [72]:
```
# Conclusions from study

# Using common, well-established data and currenly the most powerfull ML algor
ithm for tabulated data:
    # a) it is possible to predict accurately the values of purely medical ran
king feature, HAQ Index
    # b) it is not possible to predict accurately the supposedly most reliable
COVID-19 data, Deaths per 100K

# EDA performed also shows that GDP and HAQ Index have logicaly inverted relat
ionships with Deaths per 100K -->
    # the higher the values of these two features, the higher the likelihood o
f high Deaths per 100K
    # the lower the values of these two features, the higher the likelihood of
low Deaths per 100K

# Based on the study results, the most logical conclusion is that the reported
COVID-19 data by country is not accurate -->
# cannot be reliably used to understand the reasons of how and why COVID-19 ha
s affected different countries in different way
```