PIC_16B_Final_Report-1

June 5, 2021

1 PIC 16B Final Report

Authors: Arnaud He, Huy Nguyen, Orion Shi

Date: 6/5/2021

Introduction:

The Covid-19 global pandemic has now reached over 170 million cases worlwide, resulting in over 3 million deaths in about 180 different countries. However, some countries such as Vietnam, Israel, and New Zealand have remained largely untouched and unaffected by Covid-19. These countries have experienced a significantly less number of cases, infection rate, and mortality rate. Furthermore, countries like these seem to be the first countries to begin widespread vaccination, increasing their Covid-19 vaccination rate dramatically. What makes these countries so different? In order to answer this question, we decided to look into multiple signifiers of a country's socioeconomic wellbeing and political stability to find possible correlations between these signifiers and a country's Covid-19 data.

Procedure:

We first collected general data related to countries such as:

- Population Density per Square Mile/Population
- Fragile Score
- Happiness Index
- GDP per Capita
- Social Support
- Health Life Expectancy
- Freedom to Make Life Choices
- Generosity
- Perceptions of Corruption

We then collected Covid-19 related data for each country such as:

- Number of Confirmed Cases
- Number of Deaths
- Fatality Rate (%)
- Number of Samples Tested
- Confirmed/Tested (%)
  - Note: this is the number of confirmed cases over the number of tests
- Tested/Population (%)
  - Note: this is the number of tests over the population

- Confirmed/Population (%)
  - Note: this is the number of confirmed cases over the population
- Number of Vaccinated
- Percent Vaccinated (%)

We decided to focus on Fatality Rate due to Covid-19 and Percent Vaccinated in each country as they seem to be good indicators of how well a country has handled the pandemic so far. We will be examining and analyzing how these two variables individually correlate with the other features we collected through regression plots and normalization techniques.

This will allow us to choose a few of the most correlated and interesting features to build models that actually predict Fatality Rate and Percent Vaccinated in a country depending on a few decided features. Our goal is to use these models to create PDP and ICE plots, giving us more insight on how these models are affected by the features we decide to choose. As a result, this can indirectly give us more information on how these features interplay with one another and how they affect a country's mortality rate due to Covid-19 or vaccination rate.

[1]:
```
# import code cell

import numpy as np
import pandas as pd
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Input
import sklearn
from sklearn.datasets import load_digits
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, SGDRegressor, BayesianRidge
from sklearn.pipeline import make_pipeline
from sklearn.inspection import partial_dependence, plot_partial_dependence
from sklearn.metrics import r2_score

import random
random.seed(1)
np.random.seed(1)
```

```
tf.random.set_seed(1)
```

1.1 Data Management:

Collecting our data:

We first used Scrapy to scrape multiple websites, such as wikipedia and statisticstimes, to collect an adequate amount of data on multiple countries for the features listed above. All the scraped csv. files and scrapy project codes can be found in the following link:

https://drive.google.com/drive/folders/1jCzEgmW7l0hwoahoiM5IZ6BSXEdgh6vK?usp=sharing

(Note: each team member scraped 2-3 websites)

Cleaning/Merging Our Data:

Next, in order to clean the data we had two significant problems to overcome: 1. Converting numeric data to the correct data type for graphing and evaluation (i.e. converting 5% to 0.05) 2. Matching the varied country names across tables as one table may have USA while another may have United States of America, making it difficult to merge our tables by Country

In order to deal with problem 1, we first used Pandas to import all of our individual datatables we scraped from our assigned websites into dataframes. We then cleaned the dataframes by removing rows with Nan values and changing column names in order to better match with each other. We then removed unwanted commas and percent symbols in our numeric data and changed the columns' data types to the desired type (usually float or int).

In order to deal with problem 2, we found a useful python package online, the Country Converter (coco) package, that allowed us to convert and match country names between different classifications and between different naming versions. Much of the information for the installation and basic usage of the coco package can be found in the following link: https://pypi.org/project/country-converter/

This is how we used the code specifically for our data:

```
# import appropriate packages
import country_converter as coco
import pandas as pd

# create CountryConverter object
cc = coco.CountryConverter()

# read in the desired csv file with a "Country" column
df = pd.read_csv("example.csv")

# takes the df.Country column and converts each country to its 'name_short' version and outputs
# list of these versions to standard_names
# for example 'Korea, Republic of' -> 'South Korea'
standard_names = cc.convert(names = df.Country, to='name_short', not_found = None)

# drop original Country column in dataframe and add a new Country column with the new country n
# contained in the list standard_names
df = df.drop(columns = ['Country'])
```

```
df['Country'] = standard_names
```

We then merged all of our cleaned data into a single dataframe using Pandas.

All the ipynb files with the data cleaning, coco package, and merging can be found in the following link:

https://drive.google.com/drive/folders/1YrzOHukhFx1ar24aair0ILtgsiM7X_cp?usp=sharing

All the cleaned, individual csv files can be found in the following link:

https://drive.google.com/drive/folders/1zvA-INE5lpQpxyfodLqhizvSr-nwRhIM?usp=sharing

Below is the final data set we used for our project. The data set contains 141 samples with 21 columns/features.

[2]:
```
df = pd.read_csv("total_country_data.csv")
df.head()
```

[2]:

	Density Per SqMile	Pop	Continent	Country	Confirmed Cases	\
0	297.8	114963588	Africa	Ethiopia	263120	
1	244.7	53771296	Africa	Kenya	163620	
2	123.3	27691018	Africa	Madagascar	39162	
3	525.5	19129952	Africa	Malawi	34180	
4	1623.0	1271768	Africa	Mauritius	1256	

	Deaths	Fatality Rate (%)	Samples Tested	Confirmed/Tested (%)	\
0	3897	1.5	2555989	9.90	
1	2907	1.8	1322806	8.10	
2	729	1.9	119608	16.60	
3	1153	3.4	230668	14.70	
4	17	1.4	289552	0.17	

	Tested/Pop (%)	…	Fragile Score	Vaccinated	Percent Vaccinated	\
0	2.20	…	94.6	1454503	38.6	
1	2.80	…	90.3	933436	1.7	
2	0.46	…	79.5	609	0.0	
3	1.20	…	84.0	332955	1.7	
4	22.90	…	37.2	212182	16.7	

	Happiness index	GDP per capita	Social support	Healthy life expectancy	\
0	4.186	0.315	1.001	0.484	
1	4.583	0.476	0.905	0.536	
2	4.166	0.245	0.824	0.501	
3	3.538	0.177	0.530	0.446	
4	6.101	1.074	1.396	0.763	

	Freedom to make life choices	Generosity	Perceptions of corruption
0	0.413	0.228	0.117
1	0.519	0.394	0.067

```
2                          0.193       0.191                      0.076
3                          0.487       0.213                      0.132
4                          0.591       0.187                      0.084

[5 rows x 21 columns]
```

1.2 Examining Correlations:

Out of the Covid-19 data we collected on each country, we were most interested in the differences in the fatality rate due to Covid-19 and percent vaccinated, as these seem to be good indicators of how well a country is handling the pandemic. First, we will plot the linear correlation plots to see if there is a strong correlation between the features and the predicted variables, and distribution plots to check if the features and the predicted variables follow a Normal Distribution.

Our ipynb files in which we first started plotting these correlations and testing out different variables to use as our predicted variable can be found in the following link: https://drive.google.com/drive/folders/1QzkXZd-mLcLrg-cwev0R-nbViYBaJXfy?usp=sharing

1.2.1 Fatality Rate Correlations:

As you can see below, we began by attempting to find any possible correlations between columns in our dataframe and the Fatality Rate (%) column.

[3]:
```
correlations = df.corr(method='pearson') # outputs the standard pearson␣
↪correlation coefficient between each column and the Fatality Rate
correlations["Fatality Rate (%)"]
```

[3]:
```
Density Per SqMile             -0.185705
Pop                             0.146995
Confirmed Cases                 0.044230
Deaths                          0.221188
Fatality Rate (%)               1.000000
Samples Tested                  0.038719
Confirmed/Tested (%)            0.493483
Tested/Pop (%)                 -0.347001
Confirmed/Pop (%)              -0.050294
Fragile Score                   0.164404
Vaccinated                      0.173148
Percent Vaccinated             -0.236527
Happiness index                -0.141700
GDP per capita                 -0.185101
Social support                 -0.114349
Healthy life expectancy        -0.153021
Freedom to make life choices   -0.197910
Generosity                     -0.244880
Perceptions of corruption      -0.239827
Name: Fatality Rate (%), dtype: float64
```

Out of these correlations, we decided to use a couple of the highest, more obvious correlations, such

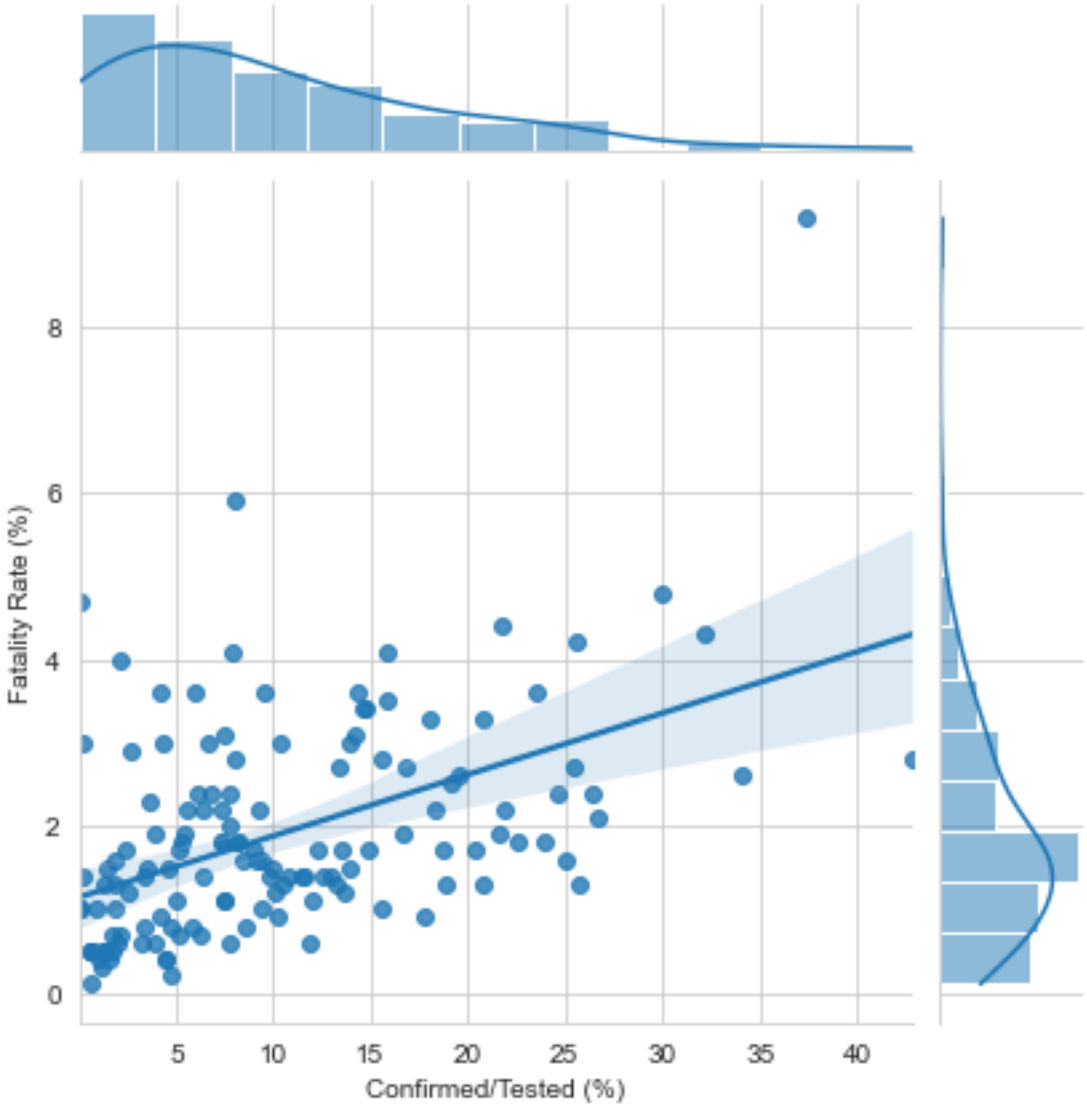
as Confirmed/Tested (%). We also wanted to use some of the more interesting correlations, such as Generosity and Social support. With this, we ended up narrowing our scope to five features—**Confirmed/Tested (%), Tested/Pop (%), Generosity, Freedom to make life choices, and Social support**—and their correlation with our designated predicted variable, fatality rate.

Below, we have plotted the linear regressions plots for each of these features against the Fatality Rate (%) in order to examine the correlations, or the lack thereof.

Correlation Plots for the **Fatality Rate**:

[4]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Confirmed/Tested (%)", y="Fatality Rate (%)", data=df,␣
↪kind="reg")
```

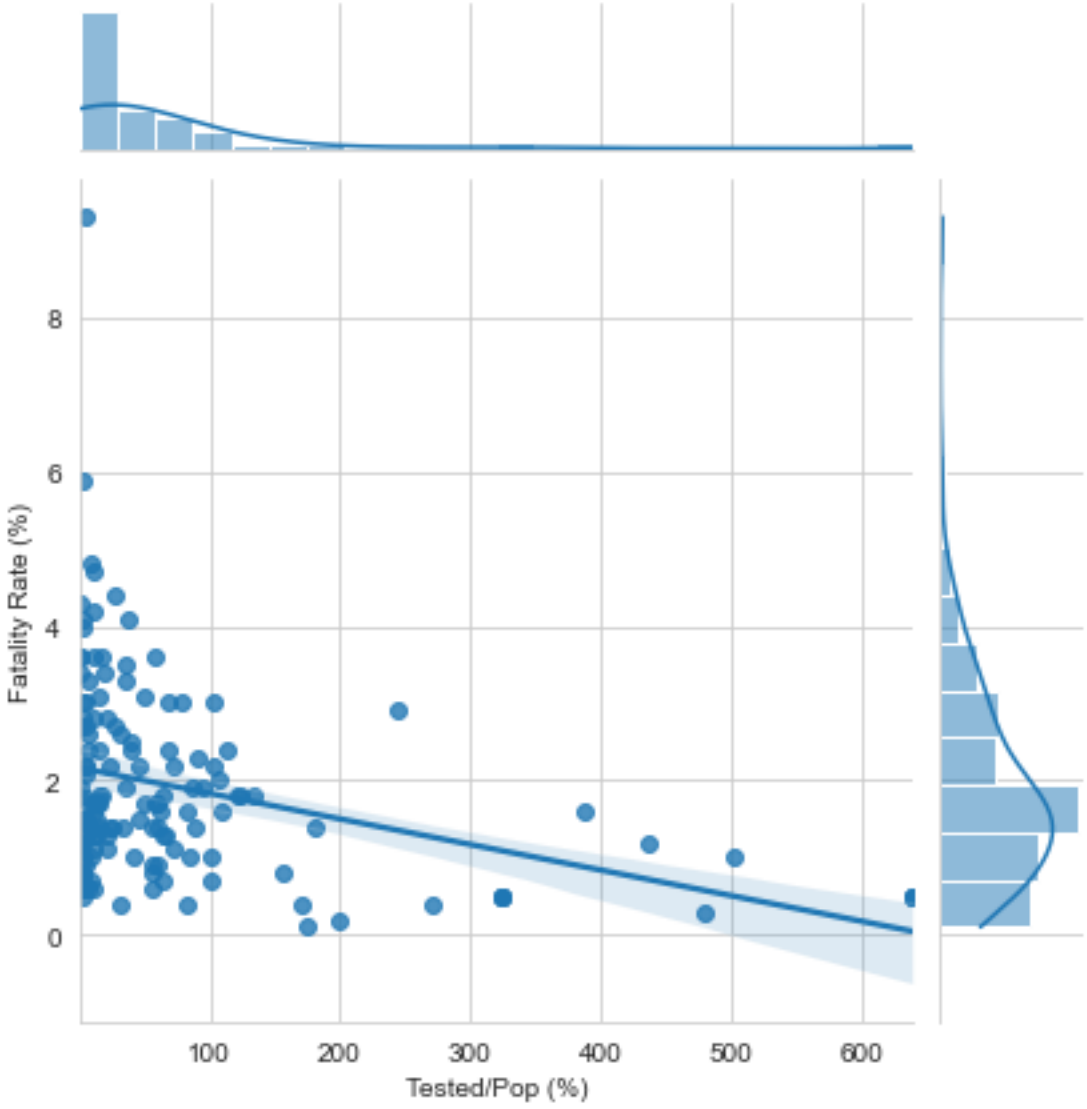
[4]: <seaborn.axisgrid.JointGrid at 0x7f8035a82a50>



The Fatality Rate (%) vs. Confirmed/Tested (%) plot indicates the two have strong positive correlation. In addition, our feature Confirmed/Tested (%) distribution skews to the right. Similarly, Fatality Rate's distribution is skewed to the right.

[5]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Tested/Pop (%)", y="Fatality Rate (%)", data=df, kind="reg")
```

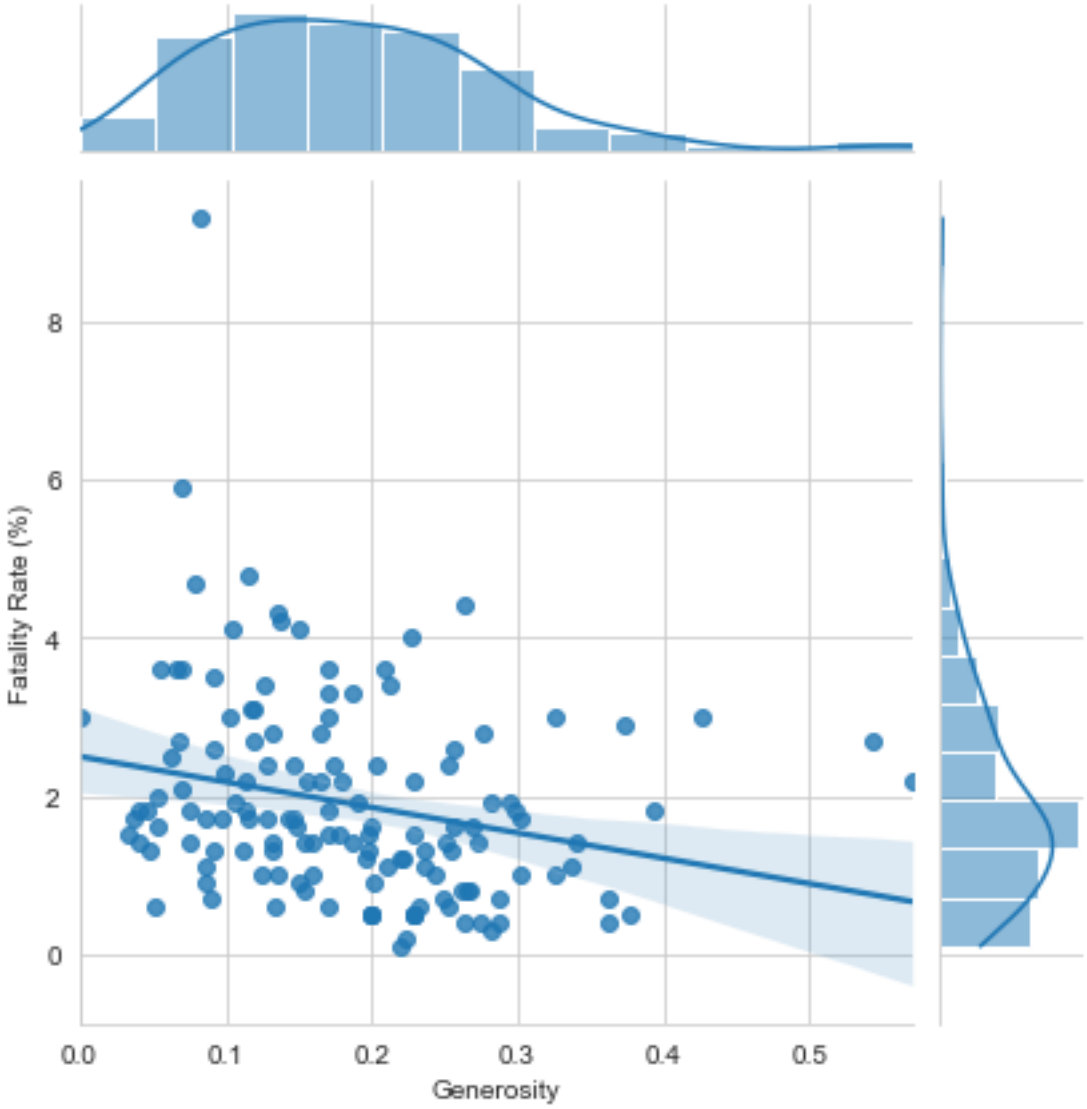
[5]: <seaborn.axisgrid.JointGrid at 0x7f8035e6bc50>



The Fatality Rate (%) vs. Tested/Pop (%) plot indicates the two have weak negative correlation (due to outliers and cluster). In addition, our feature Tested/Pop (%) distribution is very skewed to the right.

[6]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Generosity", y="Fatality Rate (%)", data=df, kind="reg")
```

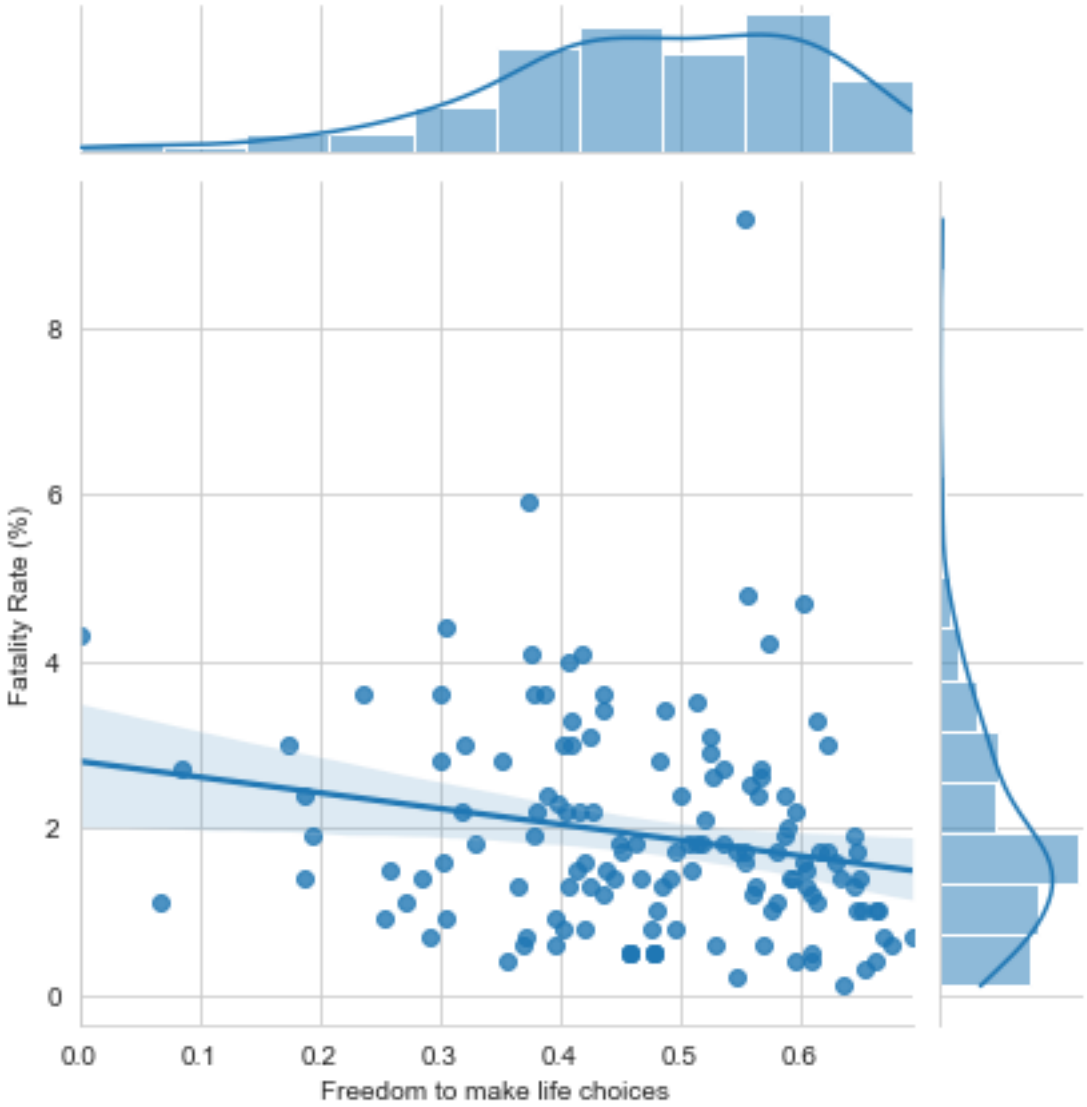
[6]: <seaborn.axisgrid.JointGrid at 0x7f80360e7550>



The Fatality Rate (%) vs. Generosity plot indicates the two have good negative correlation. In addition, our feature Generosity distribution is somewhat Normally Distributed if we ignore outliers.

[7]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Freedom to make life choices", y="Fatality Rate (%)", data=df,␣
↪kind="reg")
```

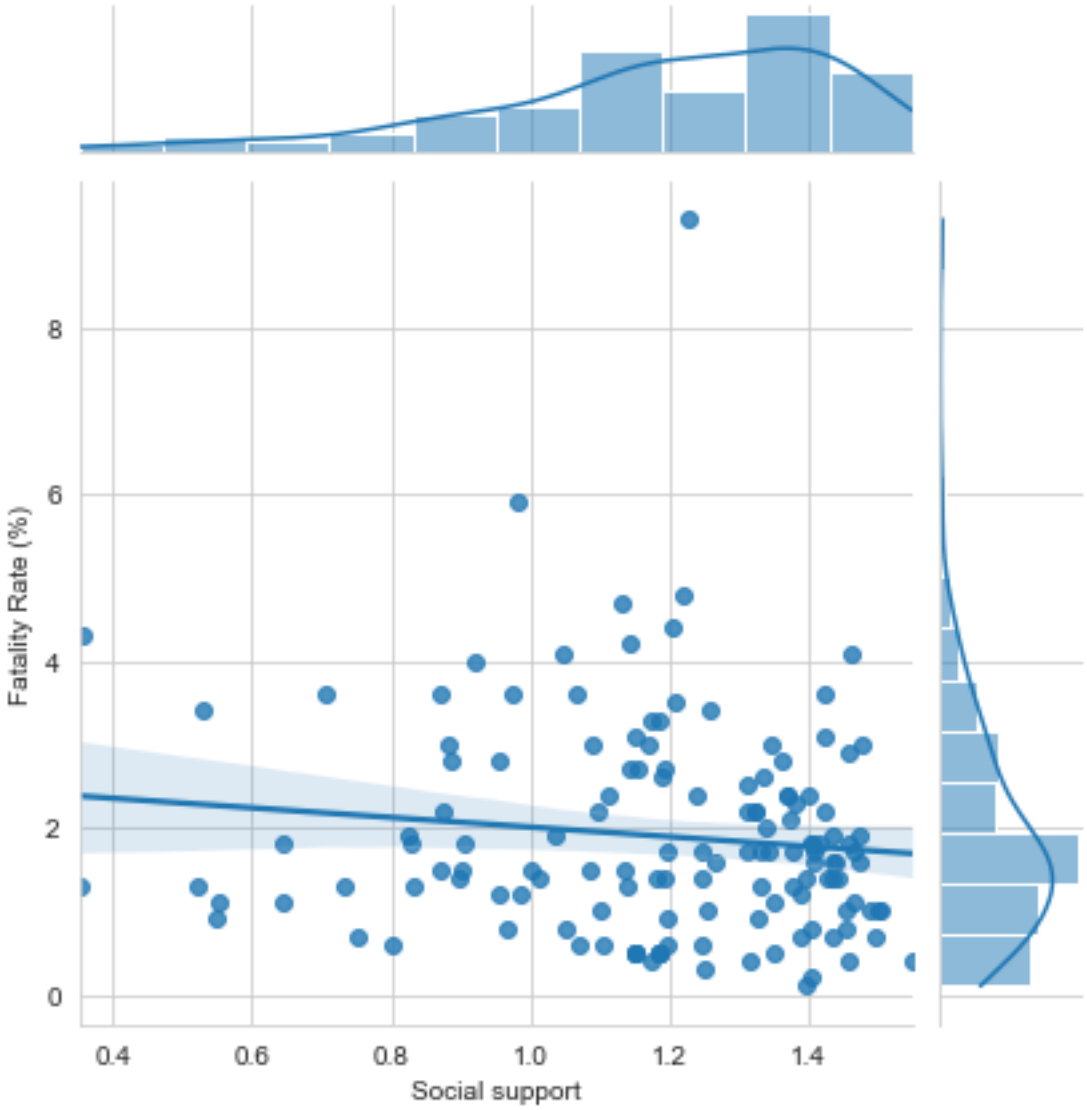
[7]: <seaborn.axisgrid.JointGrid at 0x7f80363247d0>



The Fatality Rate (%) vs. Freedom to make life choices plot indicates the two have good negative correlation. In addition, our feature Freedom to make life choices distribution is skewed to the left.

[8]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Social support", y="Fatality Rate (%)", data=df, kind="reg")
```

[8]: <seaborn.axisgrid.JointGrid at 0x7f8035ca7fd0>



The Fatality Rate (%) vs. Social support plot indicates the two have somewhat negative correlation. In addition, our feature Social support distribution is skewed to the left.

1.2.2 Vaccination Rate Correlations:

As you can see below, we used a similar approach to find possible correlations between columns in our dataframe and the vaccination rate.

[9]:
```
correlations = df.corr(method='pearson') # outputs the standard pearson␣
↪correlation coefficient between each column and the Percent vaccinated
correlations["Percent Vaccinated"]
```

[9]:
```
Density Per SqMile              0.168812
Pop                            -0.010830
Confirmed Cases                 0.143438
Deaths                          0.138958
Fatality Rate (%)              -0.236527
Samples Tested                  0.235683
Confirmed/Tested (%)           -0.259799
Tested/Pop (%)                  0.538374
Confirmed/Pop (%)               0.286313
Fragile Score                  -0.592957
Vaccinated                      0.141706
Percent Vaccinated              1.000000
Happiness index                 0.580092
GDP per capita                  0.622595
Social support                  0.492522
Healthy life expectancy         0.565411
Freedom to make life choices    0.309434
Generosity                      0.112784
Perceptions of corruption       0.282860
Name: Percent Vaccinated, dtype: float64
```

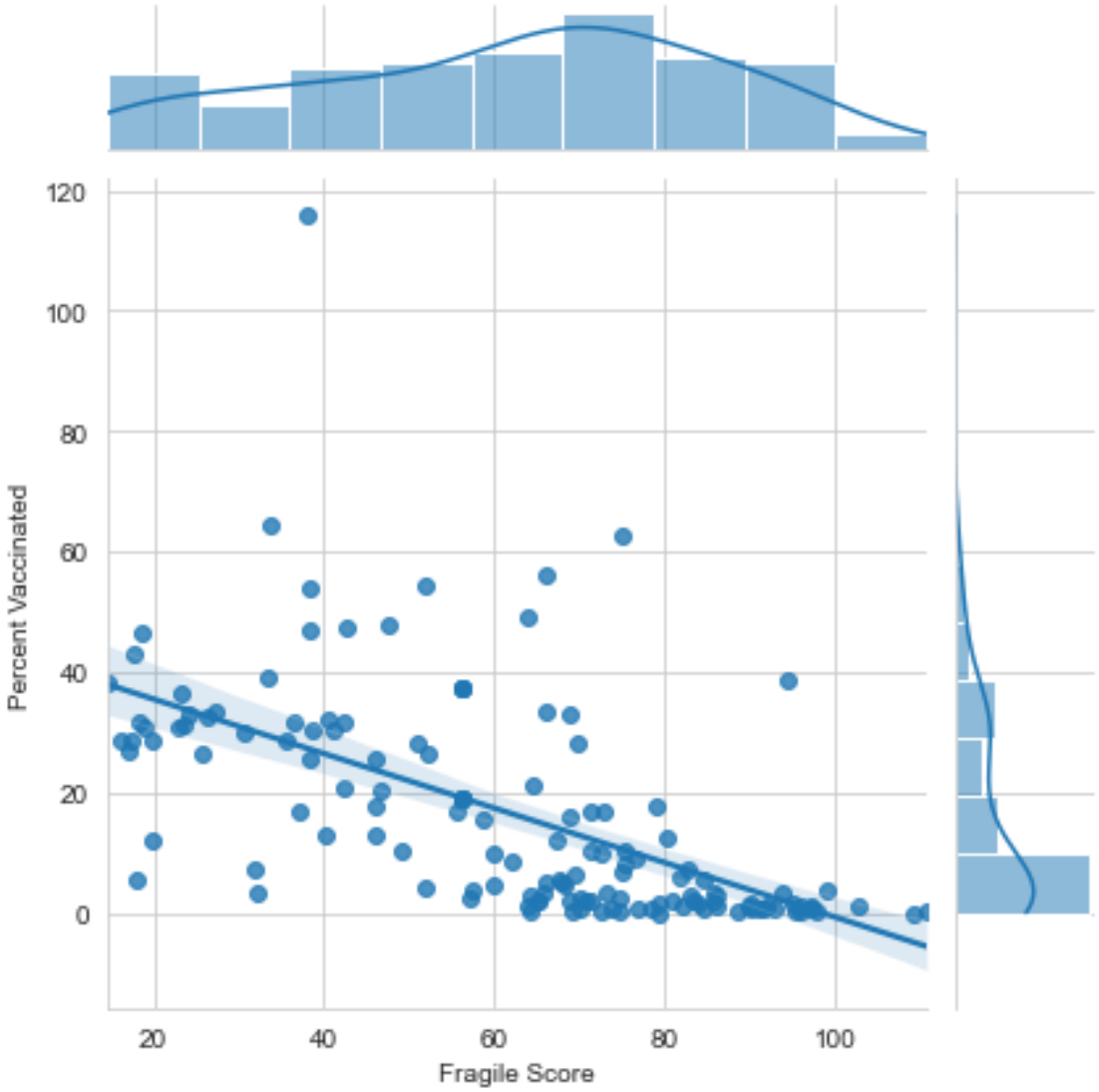
Out of these, we found there was a surprising correlation between features that seemed unrelated to Covid-19, such as GDP and Happiness Index, and the Percent Vaccinated. With this, we chose five features—**Fragile Score, Happiness Index, GDP per capita, Perceptions of corruption, and Social Support**—to examine their correlation with the Percent Vaccinated in each country.

Below, we have plotted the linear regressions plots for each of these features against the Percent Vaccinated in order to examine the correlations, or the lack thereof.

The correlation plots for **Percent Vaccinated**:

[10]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Fragile Score", y="Percent Vaccinated", data=df, kind="reg")
```

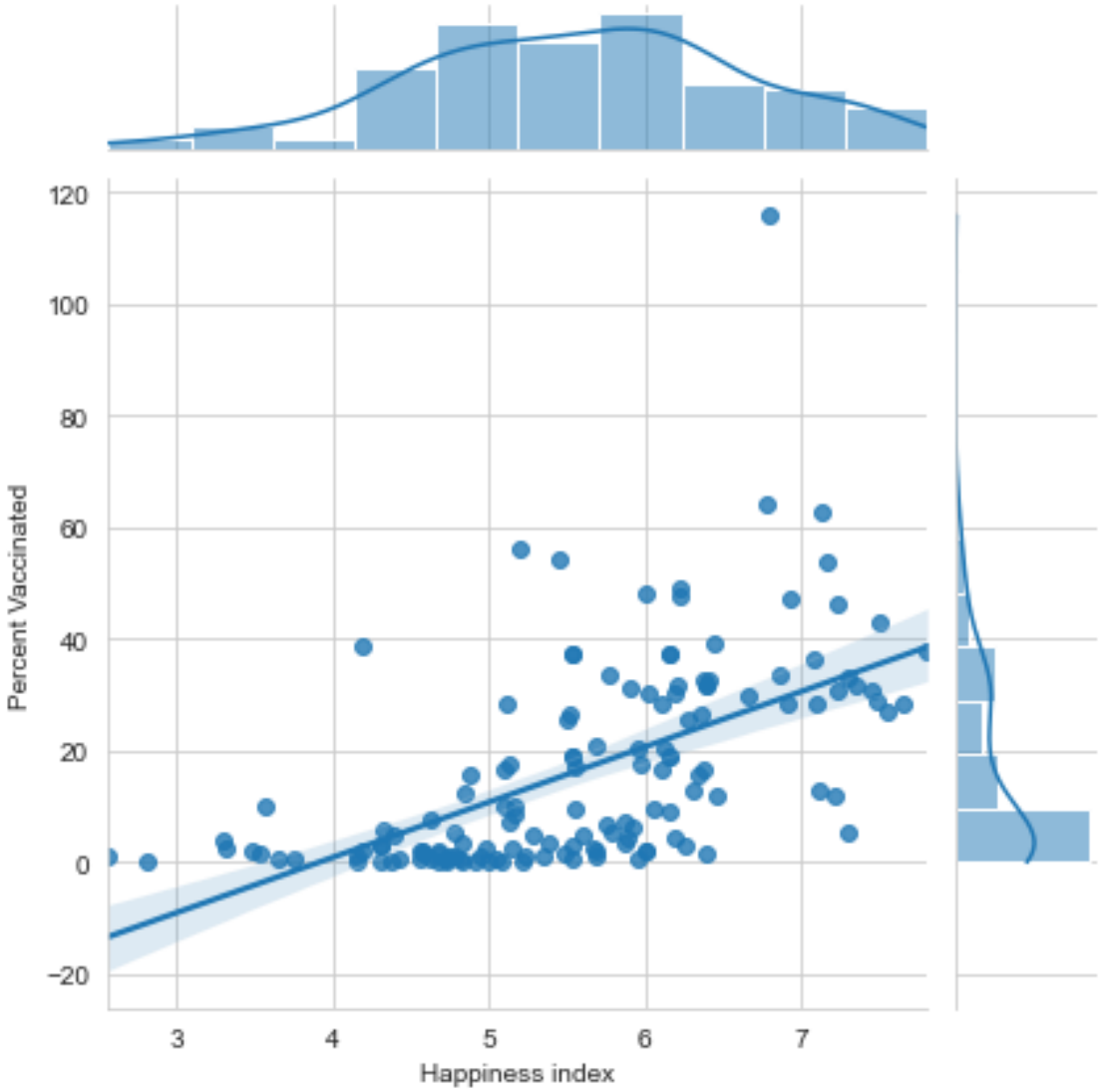
[10]:
```
<seaborn.axisgrid.JointGrid at 0x7f8016ae8cd0>
```



The Percent Vaccinated vs. Fragile Score plot indicates the two have quite strong negative correlation. In addition, our feature Fragile Score seems to follow a Normal Distribution. However, Percent Vaccinated's distribution is skewed to the right.

[11]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Happiness index", y="Percent Vaccinated", data=df, kind="reg")
```

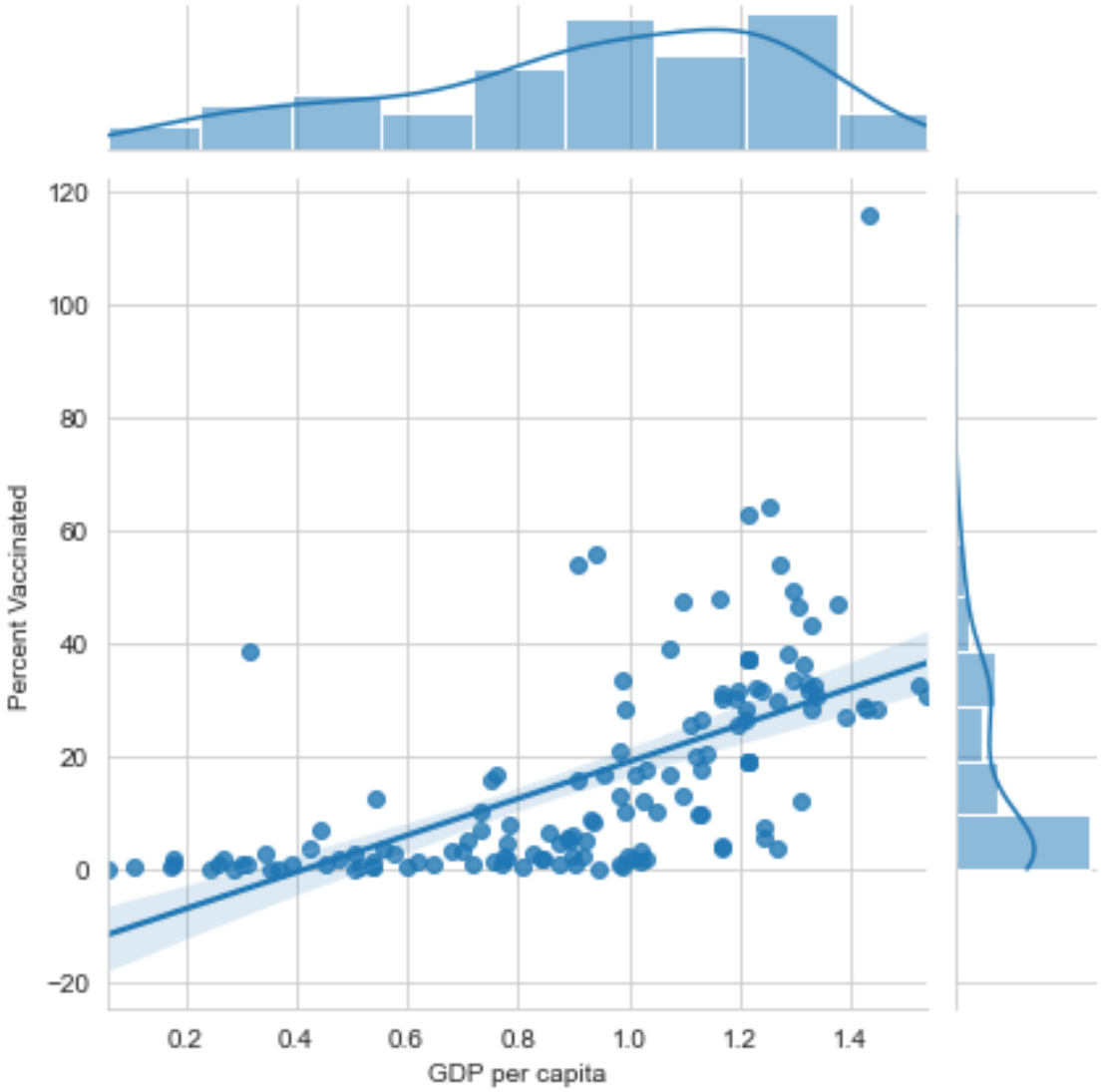
[11]:
```
<seaborn.axisgrid.JointGrid at 0x7f8036302550>
```



The Percent Vaccinated vs. Happiness index plot indicates the two have somewhat positive correlation. In addition, our feature Happiness index seems to follow a Normal Distribution.

[12]:
```
sns.set_style('whitegrid')
sns.jointplot(x="GDP per capita", y="Percent Vaccinated", data=df, kind="reg")
```

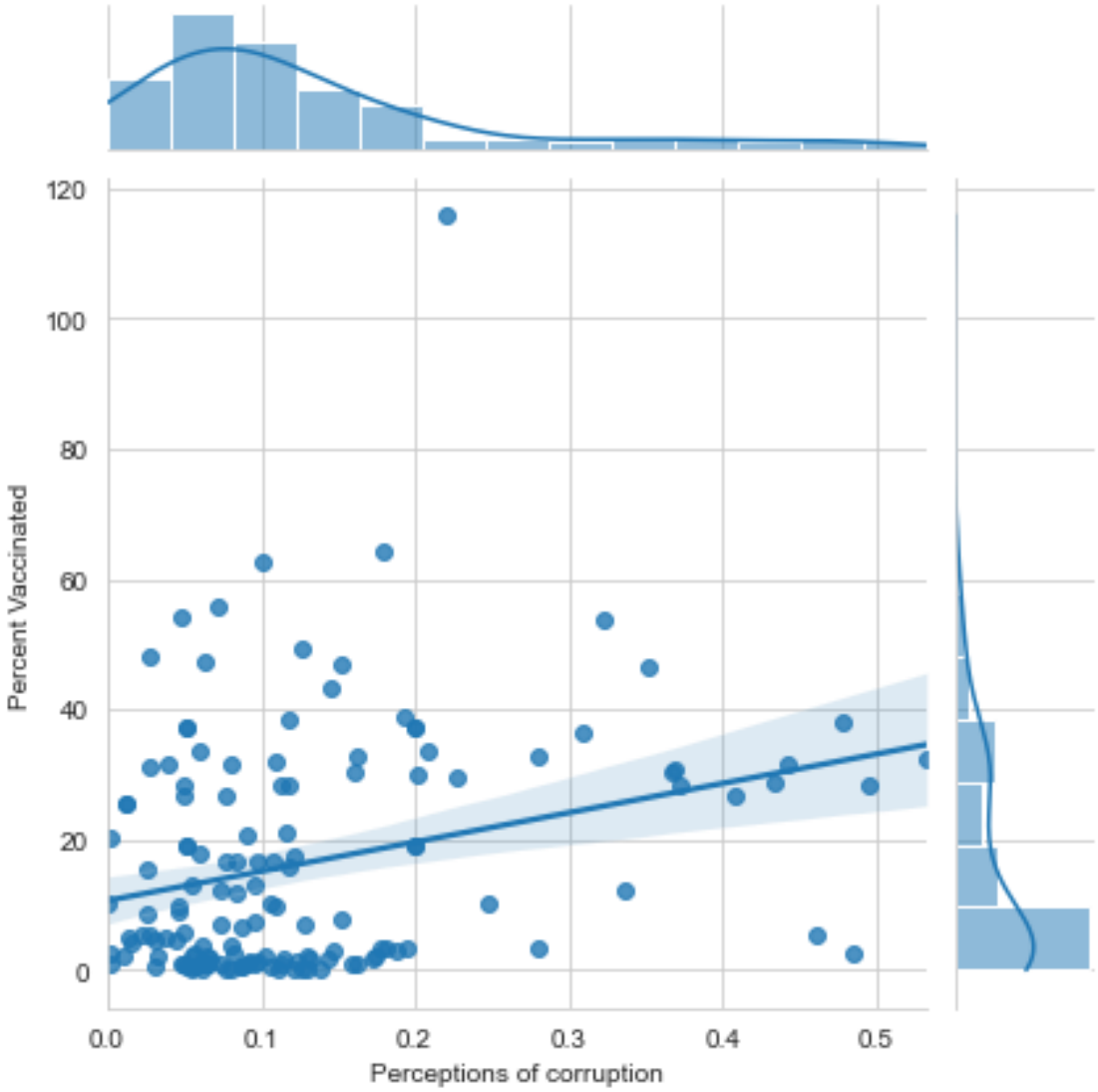
[12]:
```
<seaborn.axisgrid.JointGrid at 0x7f80173c3210>
```



The Percent Vaccinated vs. GDP per capita plot indicates the two have somewhat positive correlation. In addition, our feature GDP per capita seems to skew to the left.

[13]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Perceptions of corruption", y="Percent Vaccinated", data=df,␣
↪kind="reg")
```

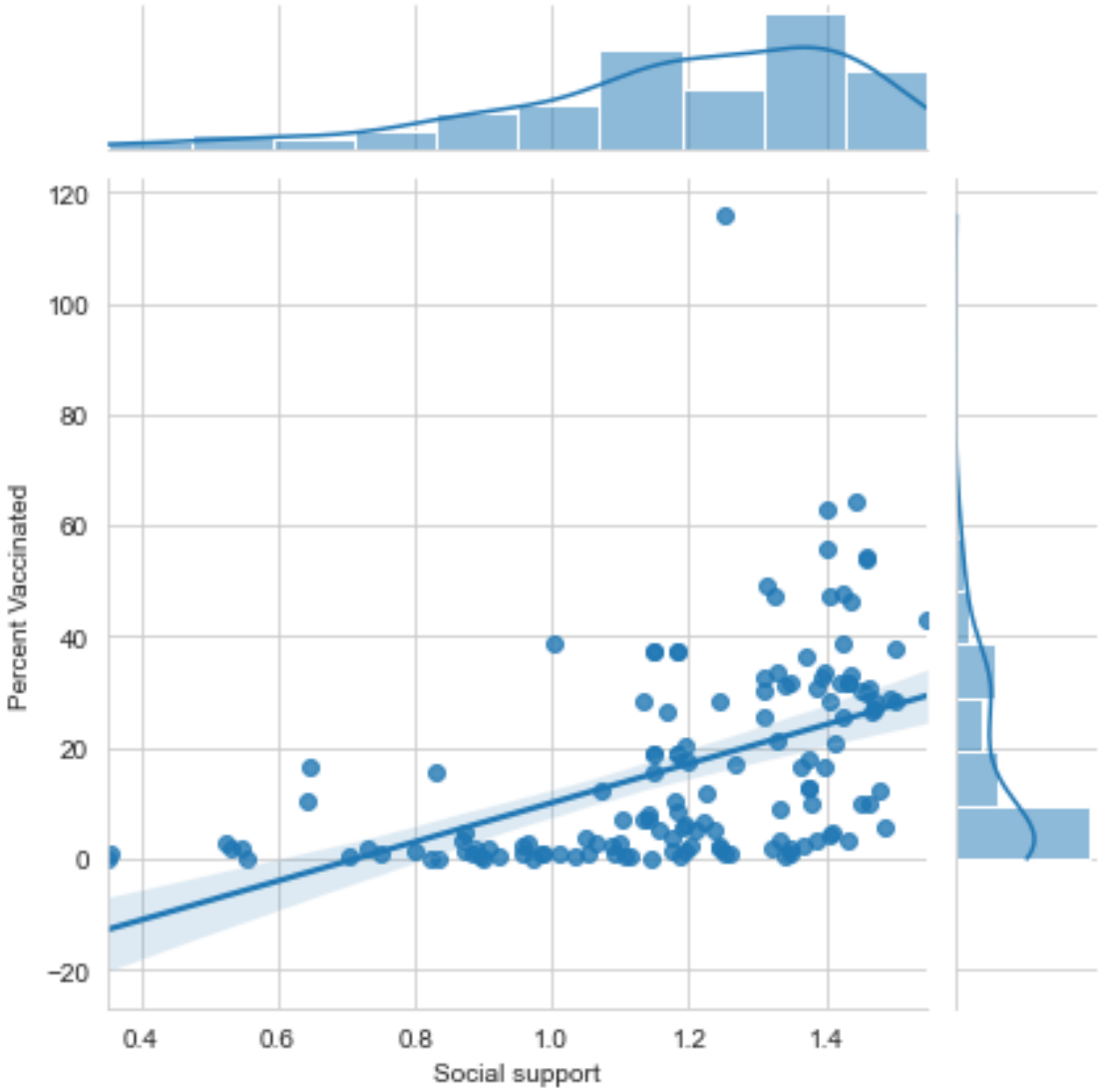
[13]:
```
<seaborn.axisgrid.JointGrid at 0x7f80172d6210>
```



The Percent Vaccinated vs. Perceptions of corruption plot indicates the two have somewhat positive correlation. In addition, our feature Perceptions of corruption seems to skew to the right.

[14]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Social support", y="Percent Vaccinated", data=df, kind="reg")
```

[14]:
```
<seaborn.axisgrid.JointGrid at 0x7f8017a118d0>
```



1.2.3 Normalization:

We also try to normalize the data and analyze through min-max normalization to determine whether the results will improve.

The ipynb file in which we tested normalization strategies and multicollinearity can be found in the following link: https://drive.google.com/drive/folders/1aaV063ouREt-7jACSjfsLZUETACagvzC?usp=sharing

[15]:
```
df.drop(['Vaccinated','Continent', 'Country', 'Pop', 'Confirmed Cases',␣
↪'Deaths', 'Samples Tested'], axis=1, inplace=True)
df1 = (df-df.min())/(df.max()-df.min()) # applies a min, max normalization on␣
↪the data
df1
```

[15]:

	Density Per SqMile	Fatality Rate (%)	Confirmed/Tested (%)	\
0	0.013509	0.152174	0.230319	
1	0.011055	0.184783	0.188209	
2	0.005445	0.195652	0.387063	
3	0.024031	0.358696	0.342613	
4	0.074746	0.141304	0.002690	
..	…	…	…	
136	0.028675	0.119565	0.057200	
137	0.060559	0.108696	0.279448	
138	0.025958	0.163043	0.218622	
139	0.000244	0.195652	0.089952	
140	0.004078	0.184783	0.171833	

	Tested/Pop (%)	Confirmed/Pop (%)	Fragile Score	Percent Vaccinated	\
0	0.003224	0.002443	0.831601	0.333333	
1	0.004164	0.002556	0.786902	0.014680	
2	0.000501	0.000823	0.674636	0.000000	
3	0.001659	0.001993	0.721414	0.014680	
4	0.035626	0.000407	0.234927	0.144214	
..	…	…	…	…	
136	0.682247	0.122582	0.043659	0.264249	
137	0.113577	0.097834	0.086279	0.265976	
138	0.129856	0.087710	0.025988	0.231434	
139	0.134552	0.038215	0.042620	0.400691	
140	0.209529	0.111333	0.246362	0.406736	

	Happiness index	GDP per capita	Social support	Healthy life expectancy	\
0	0.308852	0.171525	0.542642	0.369335	
1	0.384586	0.280678	0.462375	0.419479	
2	0.305036	0.124068	0.394649	0.385728	
3	0.185235	0.077966	0.148829	0.332690	
4	0.674170	0.686102	0.872910	0.638380	
..	…	…	…	…	
136	0.891072	1.000000	0.866221	0.853423	
137	0.931324	0.865763	0.929766	0.843780	
138	0.952499	0.901017	0.936455	0.906461	
139	0.889928	0.840678	0.905518	0.889103	
140	0.834224	0.889492	0.880435	0.704918	

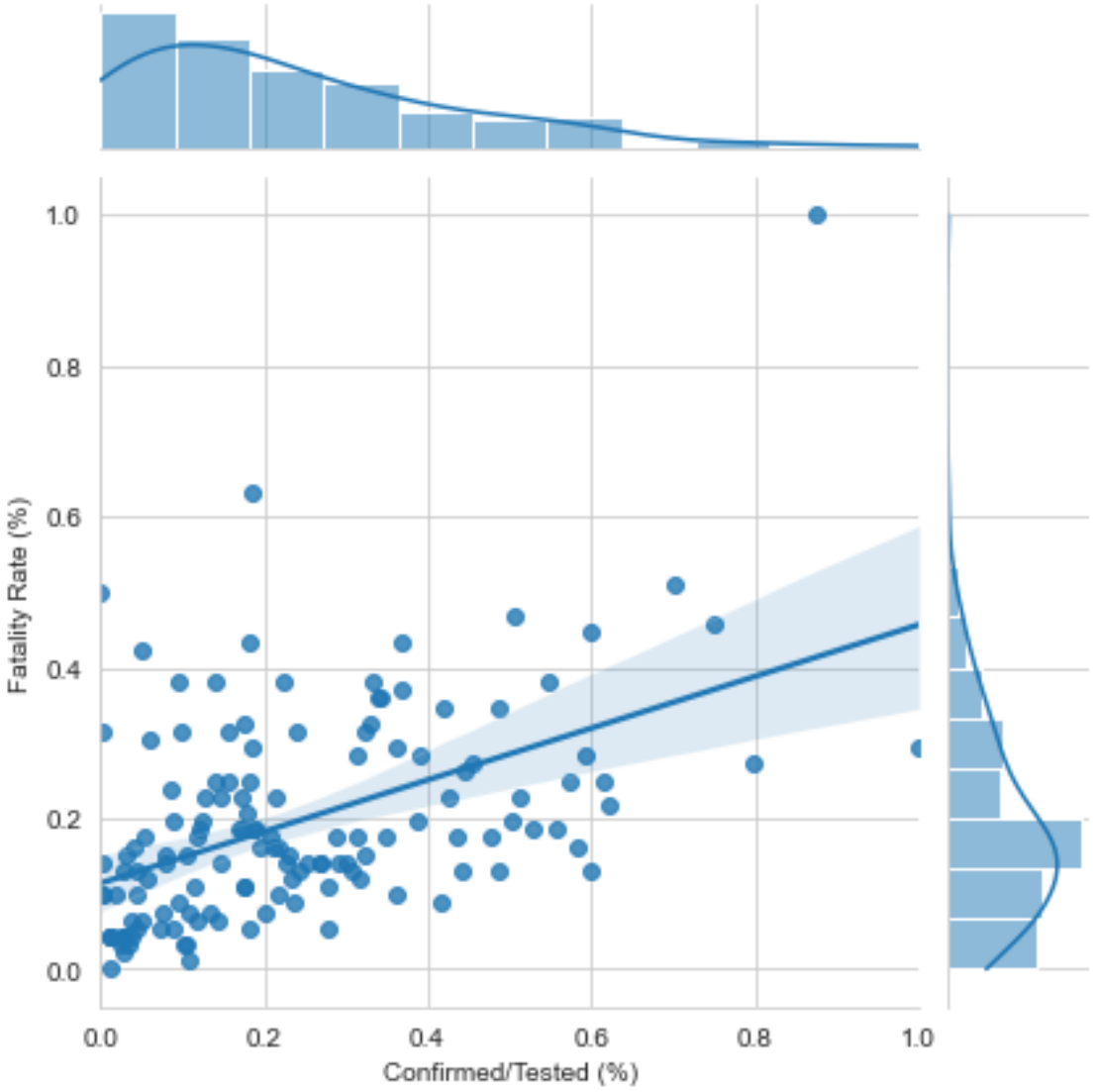
	Freedom to make life choices	Generosity	Perceptions of corruption
0	0.595960	0.400000	0.219512
1	0.748918	0.691228	0.125704
2	0.278499	0.335088	0.142589
3	0.702742	0.373684	0.247655
4	0.852814	0.328070	0.157598
..	…	…	…
136	0.880231	0.343860	0.688555

```
137                    0.886003    0.589474                    0.692308
138                    0.907648    0.471930                    0.765478
139                    0.929293    0.494737                    0.660413
140                    0.772006    0.522807                    0.285178

[141 rows x 14 columns]
```

Correlation plots for **Fatality rate** (normalized data):

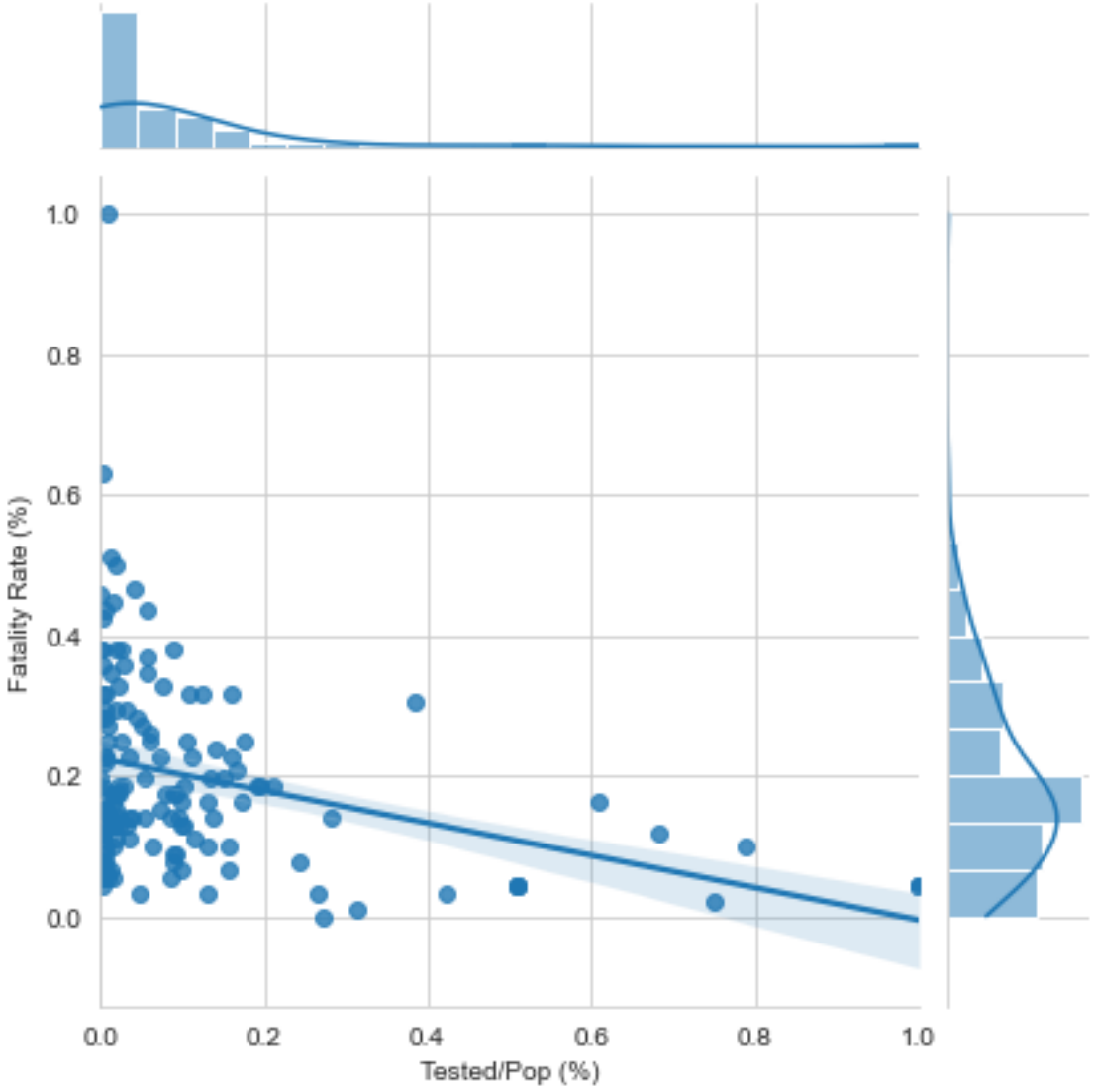
[16]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Confirmed/Tested (%)", y="Fatality Rate (%)", data=df1,␣
↪kind="reg")
```

[16]:
```
<seaborn.axisgrid.JointGrid at 0x7f80178c6110>
```



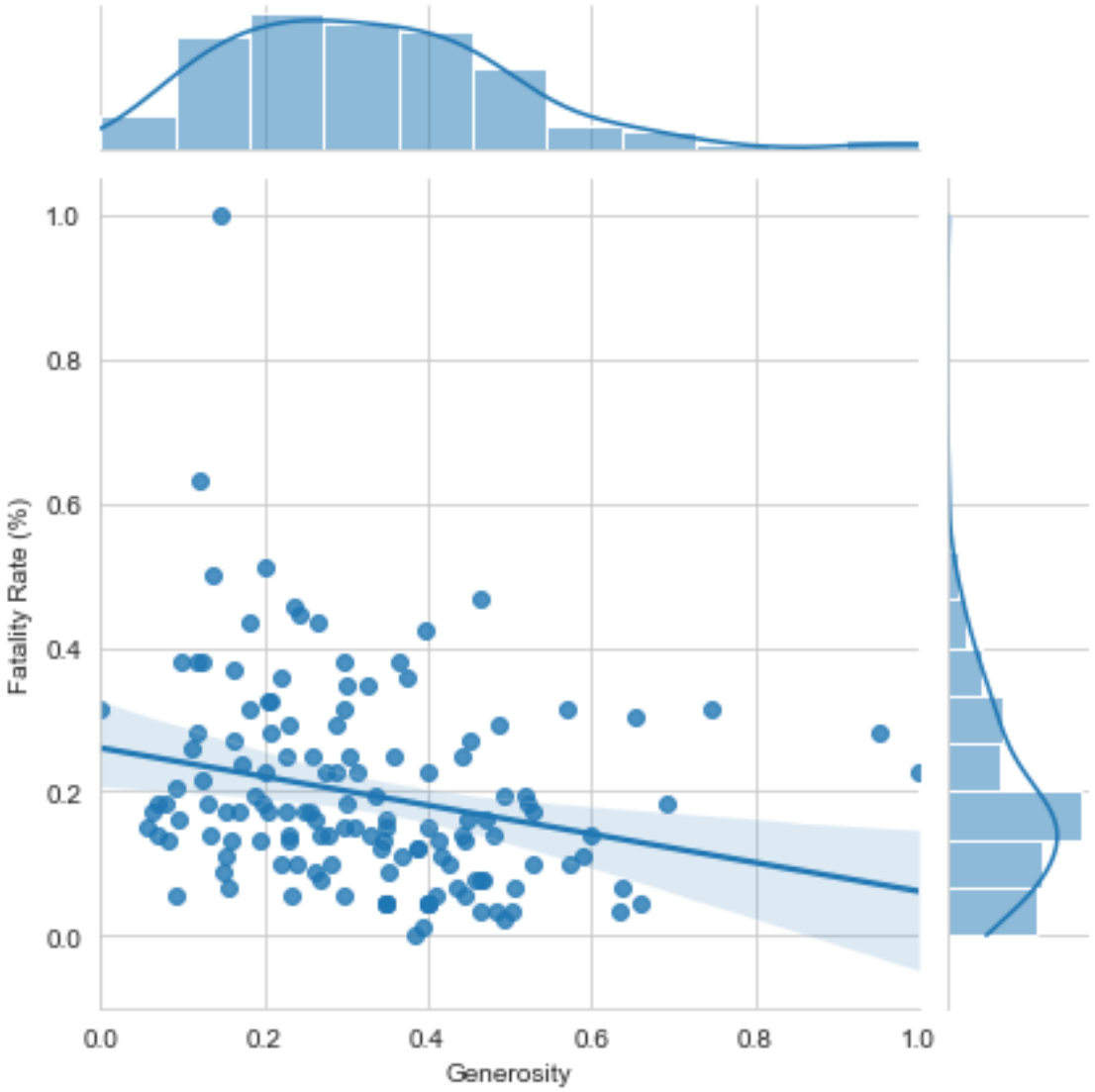
[17]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Tested/Pop (%)", y="Fatality Rate (%)", data=df1, kind="reg")
```

[17]: <seaborn.axisgrid.JointGrid at 0x7f8017fa9a90>



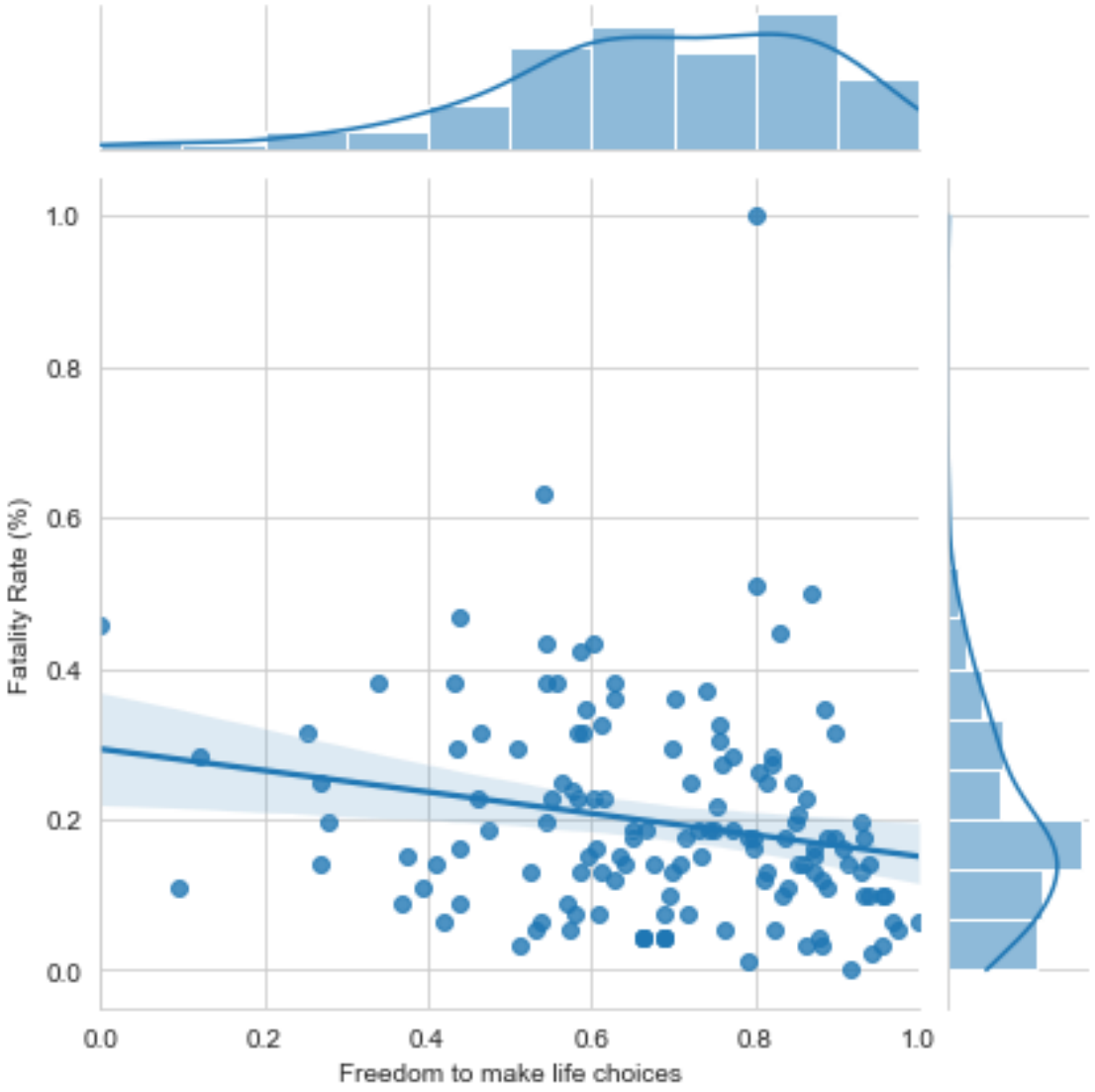
[18]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Generosity", y="Fatality Rate (%)", data=df1, kind="reg")
```

[18]: <seaborn.axisgrid.JointGrid at 0x7f80178ceb50>



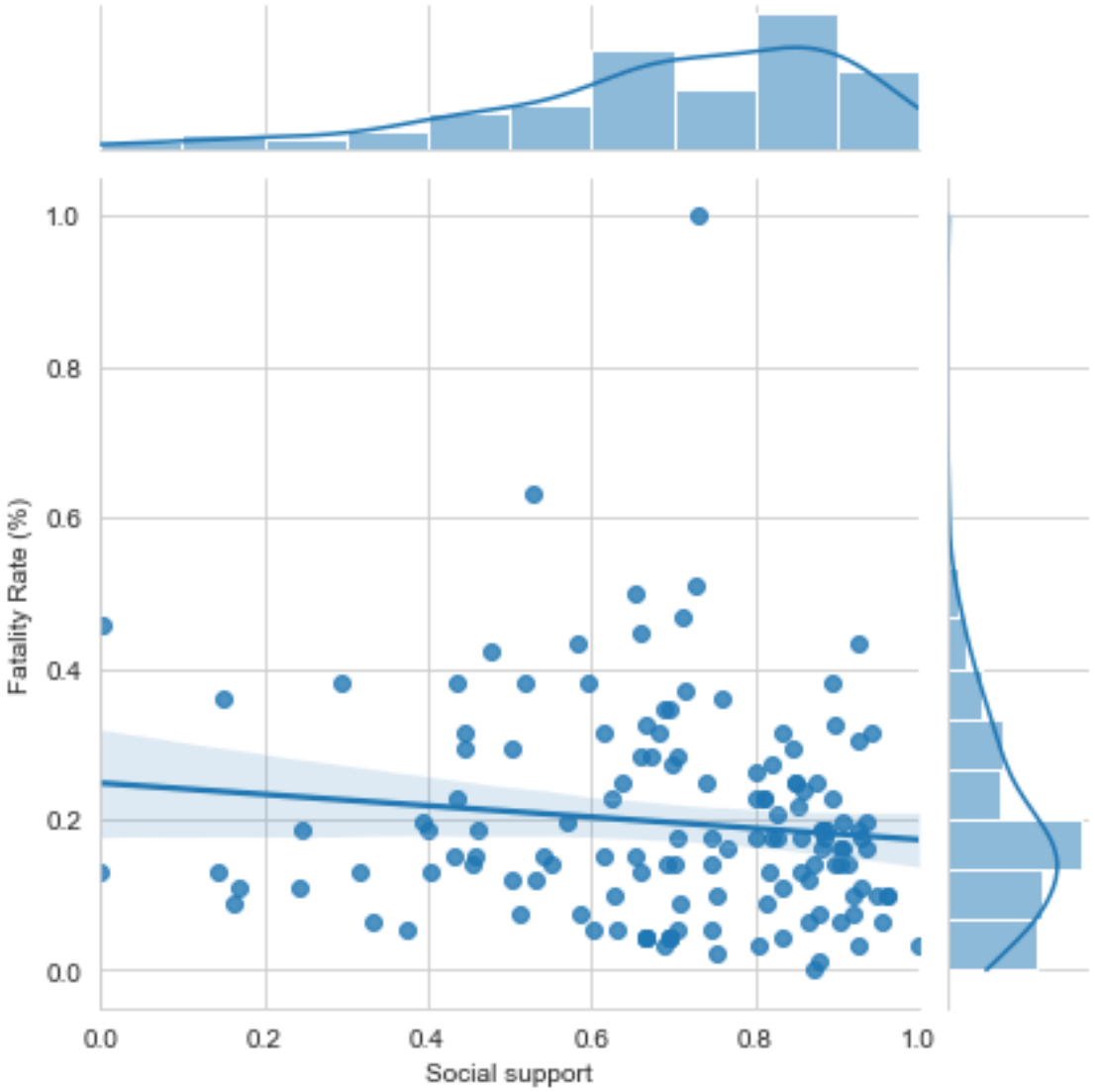
[19]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Freedom to make life choices", y="Fatality Rate (%)",␣
↪data=df1, kind="reg")
```

[19]: <seaborn.axisgrid.JointGrid at 0x7f8018617e10>



[20]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Social support", y="Fatality Rate (%)", data=df1, kind="reg")
```

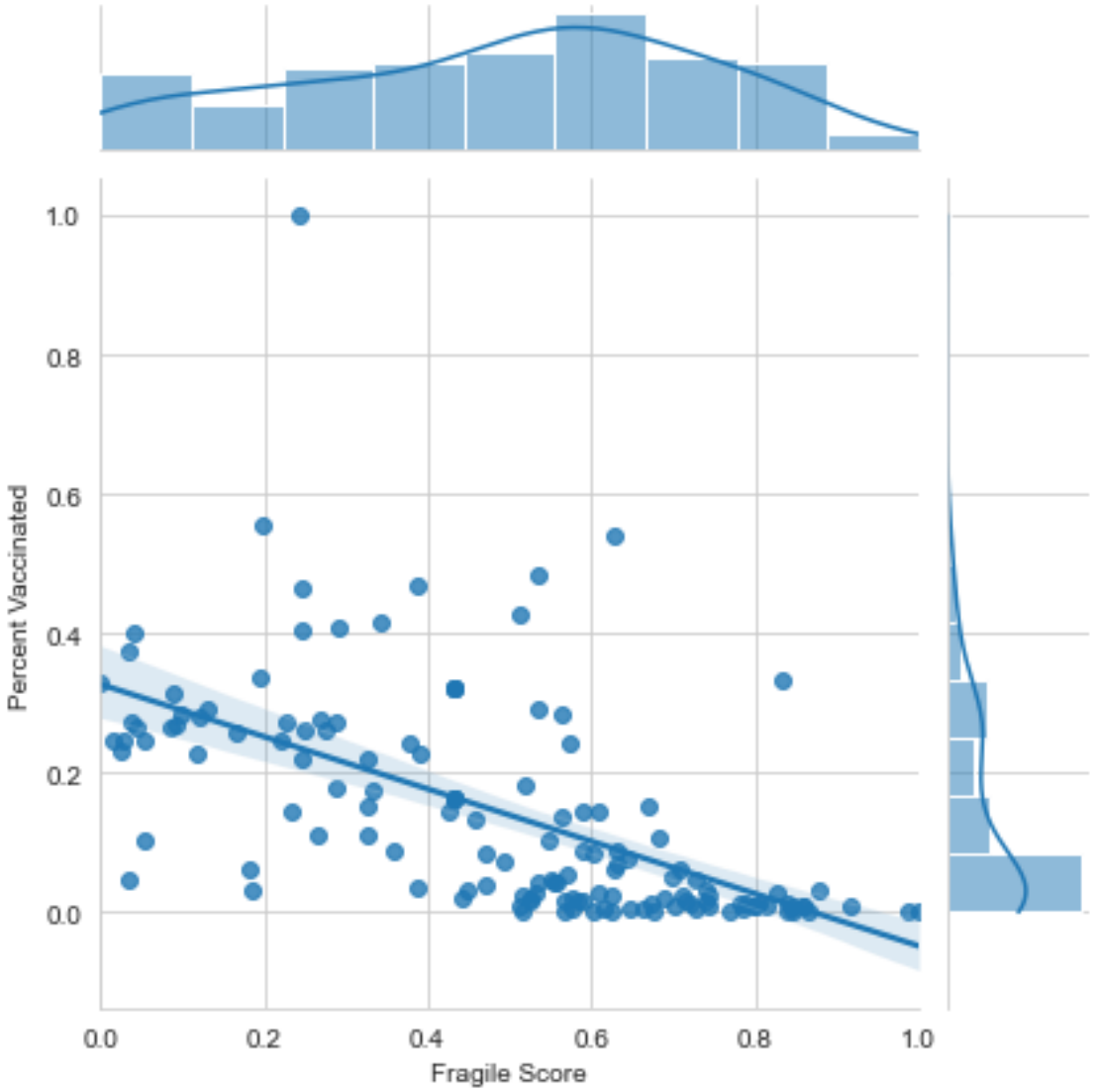
[20]: <seaborn.axisgrid.JointGrid at 0x7f801764c690>



Correlation plots for **Percent Vaccinated** (normalized data):

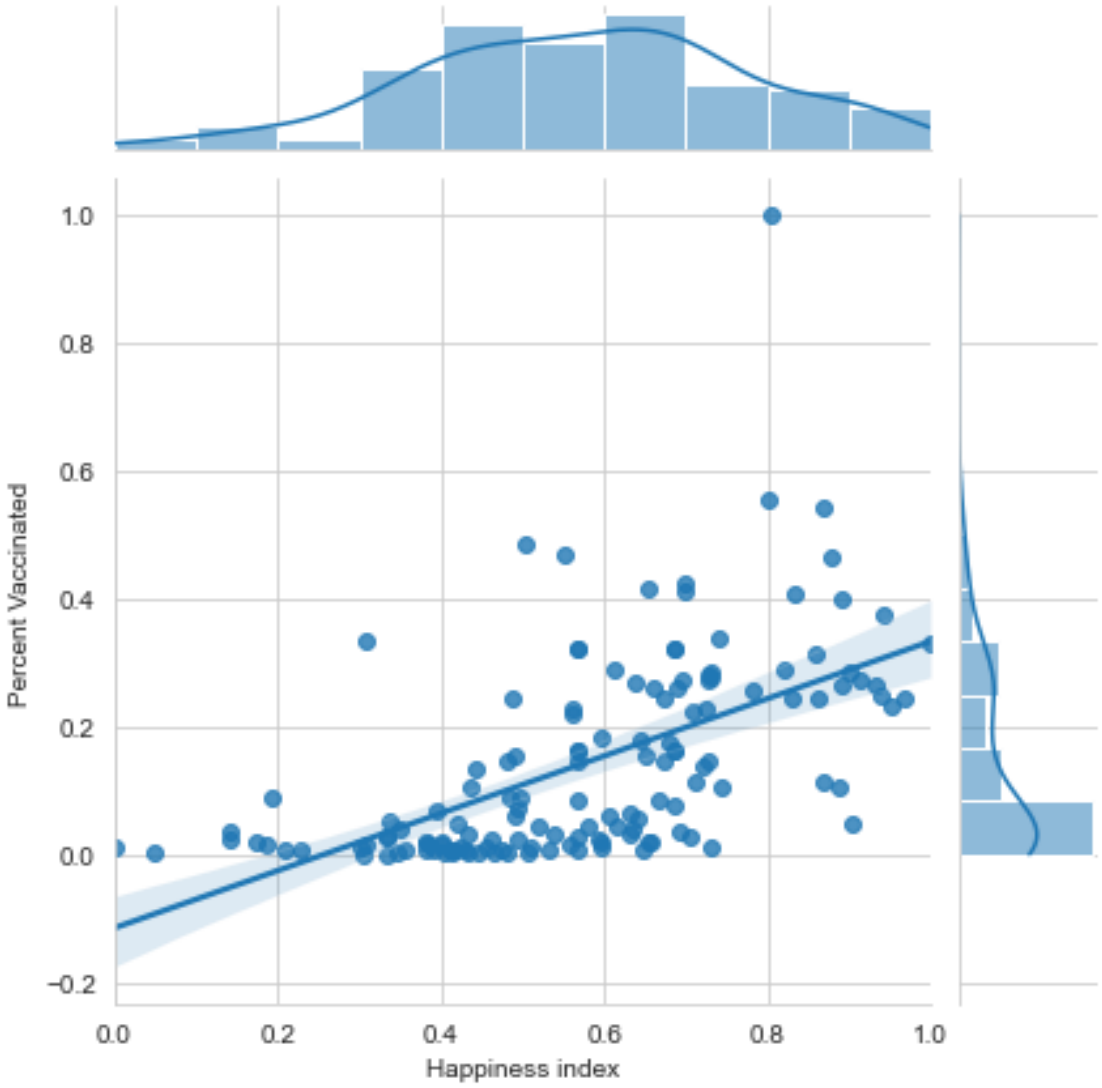
[21]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Fragile Score", y="Percent Vaccinated", data=df1, kind="reg")
```

[21]:
```
<seaborn.axisgrid.JointGrid at 0x7f8017668d90>
```



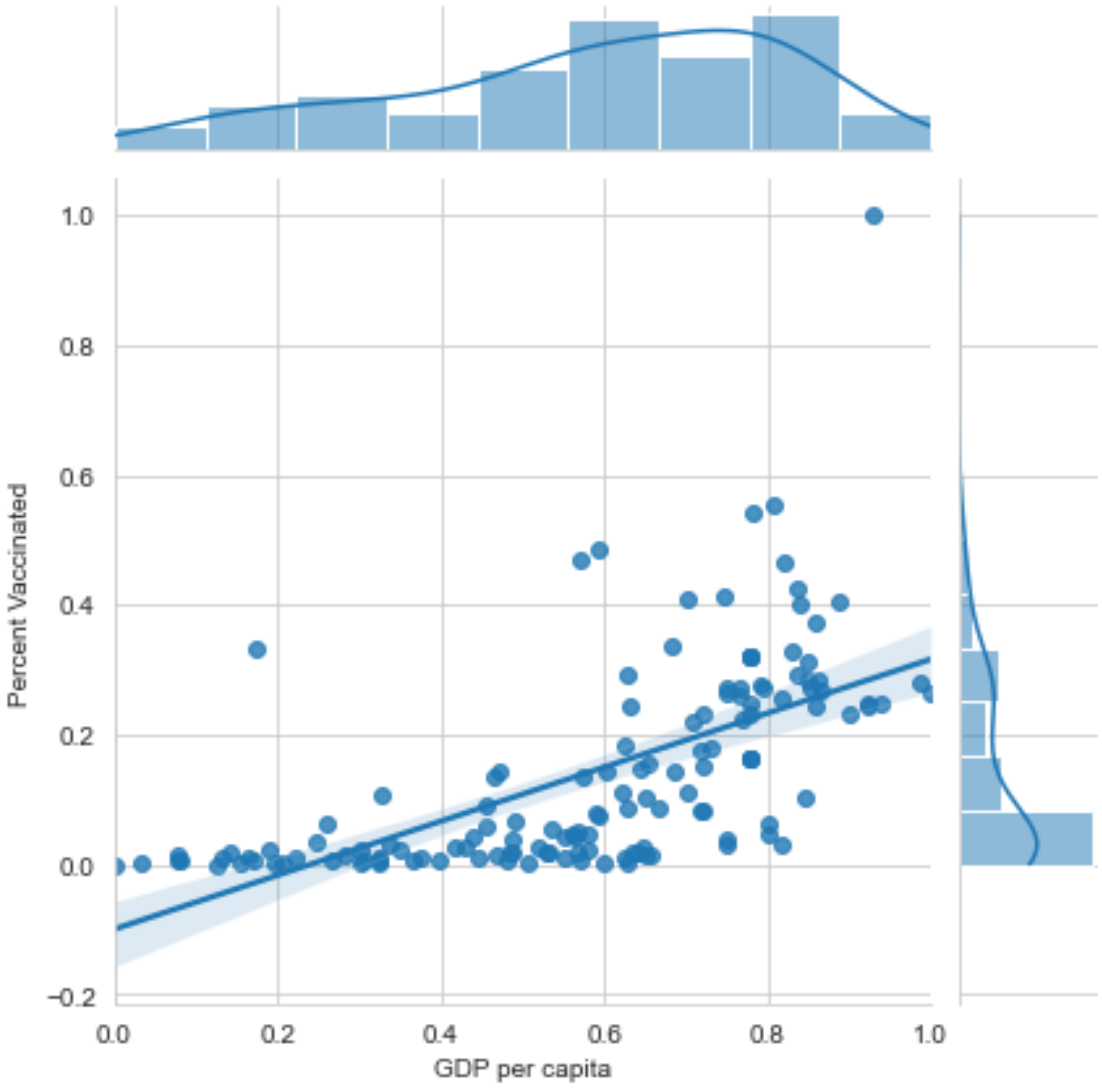
[22]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Happiness index", y="Percent Vaccinated", data=df1, kind="reg")
```

[22]: <seaborn.axisgrid.JointGrid at 0x7f8018df2a50>



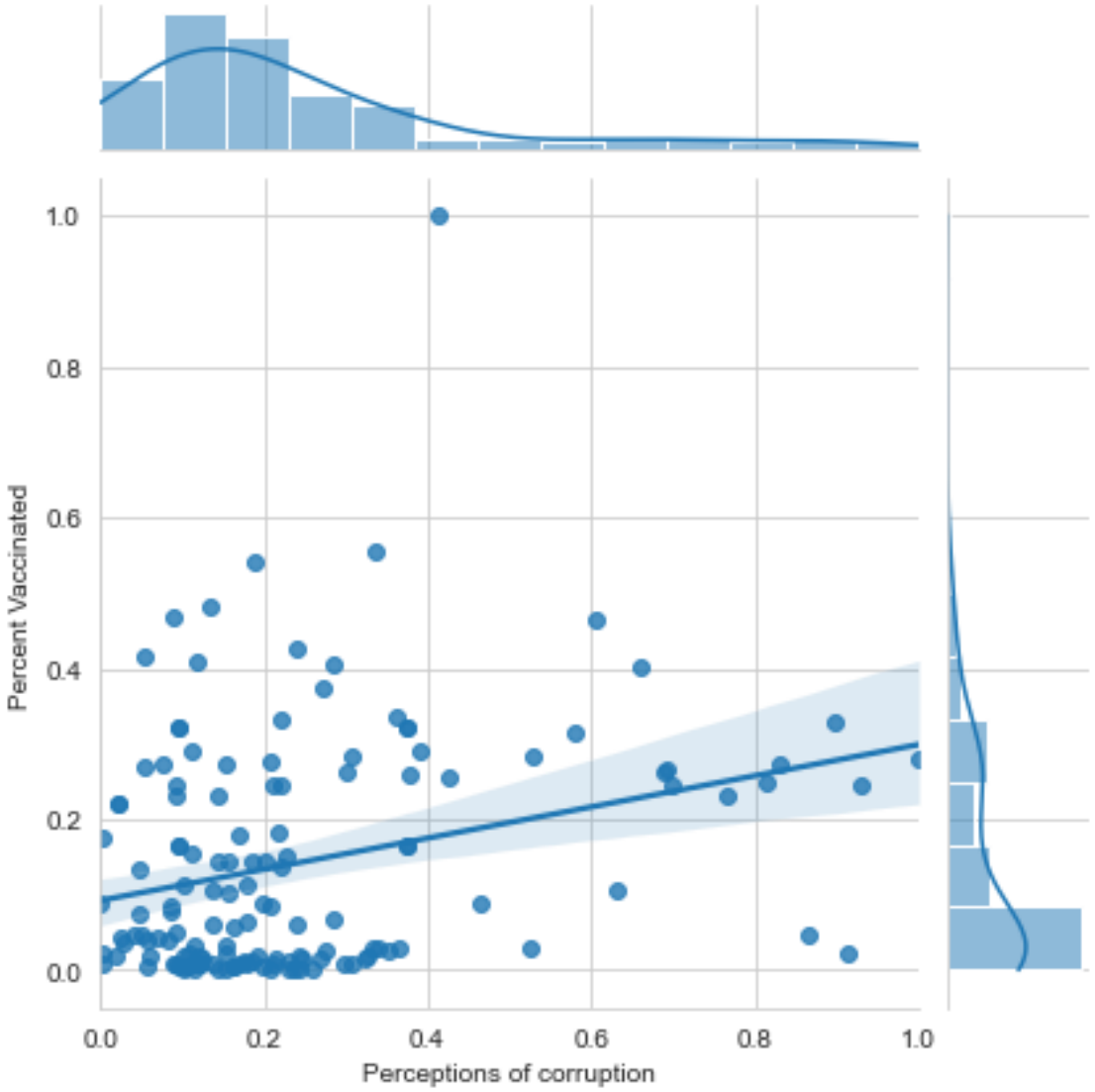
[23]:
```
sns.set_style('whitegrid')
sns.jointplot(x="GDP per capita", y="Percent Vaccinated", data=df1, kind="reg")
```

[23]: <seaborn.axisgrid.JointGrid at 0x7f80181c5ed0>



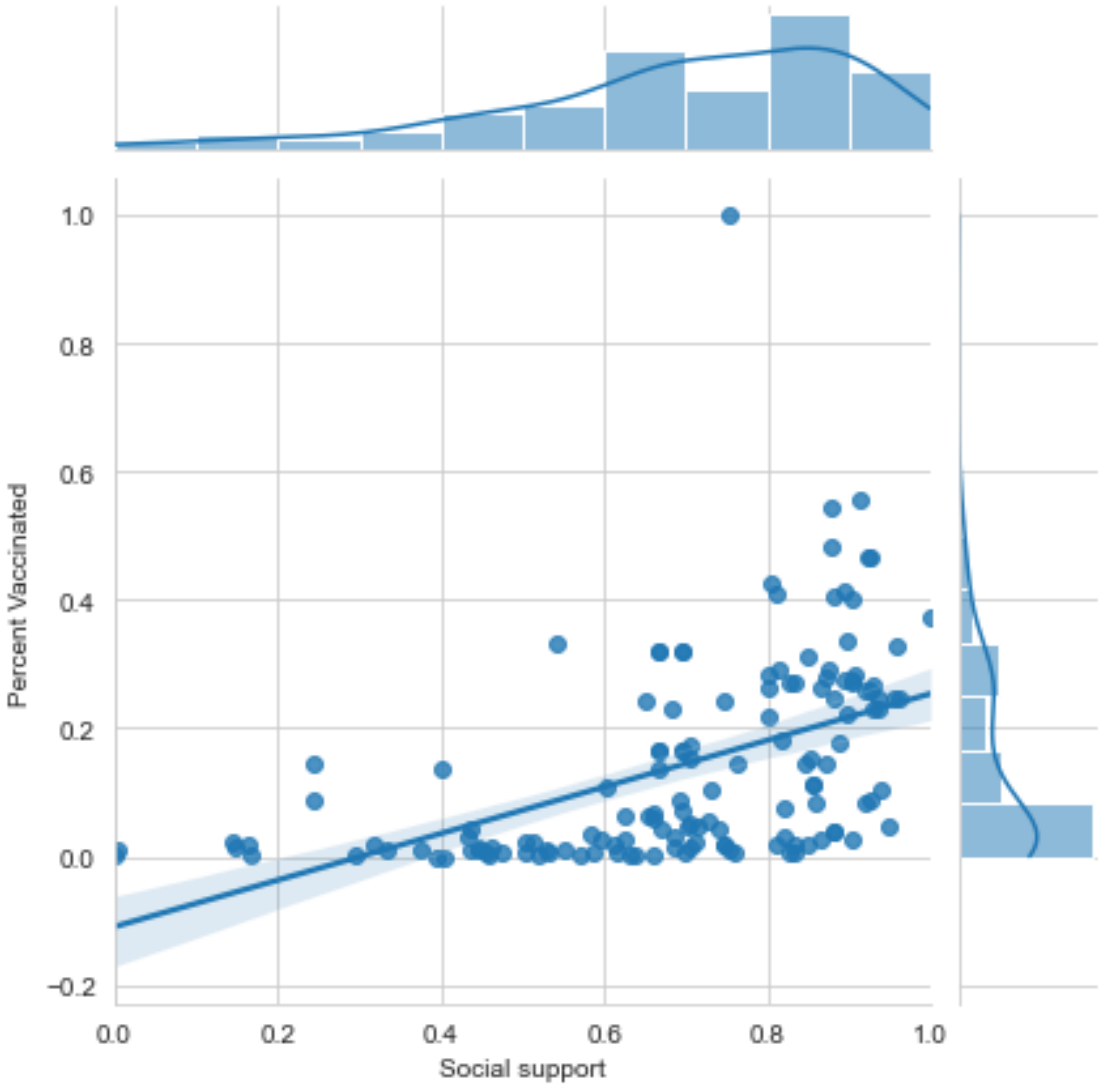
[24]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Perceptions of corruption", y="Percent Vaccinated", data=df1,␣
↪kind="reg")
```

[24]:
```
<seaborn.axisgrid.JointGrid at 0x7f801952fc50>
```



[25]:
```
sns.set_style('whitegrid')
sns.jointplot(x="Social support", y="Percent Vaccinated", data=df1, kind="reg")
```

[25]: <seaborn.axisgrid.JointGrid at 0x7f8019544c50>



Normalization Results: With the normalization, we found that there is no visble improvement compared to the original data. However, we will still use normalization as we want our data to be well structured for Machine Learning. This in turn will increase our predictive power.

Multicollinearity: We also check among the features if they are highly correlated to one another. The colors and values in each cell represent the correlations between two factors. We consider a correlation whose absolute value is more than 0.5 to be a significant correlation.

[26]:
```
mat = df1.corr()
mat.style.background_gradient(cmap='viridis').set_precision(3)
# we set the correlation to 3 decimals
```

[26]:
```
<pandas.io.formats.style.Styler at 0x7f801a1d7890>
```

From our heatmap, we see that Social Support, Fragile Score, Happiness index, and GDP per capita are significantly correlated to one another. This means that our model for Percent Vaccinated may have high errors due to multicollinearity, leading to lower R^2 score. However, our model for Fatality Rate should be better since many features are not significantly correlated to each other. Still, we accept multicollinearity from many of these features due to their high correlations with Fatality Rate and Percent Vaccinated.

1.3 Model Building and Machine Learning:

In this section, we experimented with different models to predict **Fatality Rate** and **Percent Vaccinated**.

For **Fatality Rate** we used the features we found from above:

- Confirmed/Tested (%)
- Tested/Pop (%)
- Generosity
- Freedom to make life choices
- Social support

For **Percent Vaccinated** we used the features we found from above:

- Fragile Score
- Happiness Index
- GDP per capita
- Perceptions of corruption
- Social Support

Note that all of our test code where we first tried using and training the various models below can be found in the following link: https://drive.google.com/drive/folders/1CQ_83XFQomNBSy43TeTDqaUWz7eqVxcD?usp=sharing

Below, we have the code for some data cleaning so it is better fit for building/training models.

[27]:
```
# let's start with reading in the csv. file and checking the first few rows
df = pd.read_csv("total_country_data.csv")
df.head()
```

[27]:

	Density Per SqMile	Pop	Continent	Country	Confirmed Cases	Deaths	Fatality Rate (%)	Samples Tested	Confirmed/Tested (%)
0	297.8	114963588	Africa	Ethiopia	263120	3897	1.5	2555989	9.90
1	244.7	53771296	Africa	Kenya	163620	2907	1.8	1322806	8.10
2	123.3	27691018	Africa	Madagascar	39162	729	1.9	119608	16.60
3	525.5	19129952	Africa	Malawi	34180	1153	3.4	230668	14.70
4	1623.0	1271768	Africa	Mauritius	1256	17	1.4	289552	0.17

```
   Tested/Pop (%)  …  Fragile Score  Vaccinated  Percent Vaccinated  \
0            2.20  …           94.6     1454503                38.6
1            2.80  …           90.3      933436                 1.7
2            0.46  …           79.5         609                 0.0
3            1.20  …           84.0      332955                 1.7
4           22.90  …           37.2      212182                16.7

   Happiness index  GDP per capita  Social support  Healthy life expectancy  \
0            4.186           0.315           1.001                    0.484
1            4.583           0.476           0.905                    0.536
2            4.166           0.245           0.824                    0.501
3            3.538           0.177           0.530                    0.446
4            6.101           1.074           1.396                    0.763

   Freedom to make life choices  Generosity  Perceptions of corruption
0                         0.413       0.228                      0.117
1                         0.519       0.394                      0.067
2                         0.193       0.191                      0.076
3                         0.487       0.213                      0.132
4                         0.591       0.187                      0.084

[5 rows x 21 columns]
```

[28]:
```
# remove categorical variables
df.drop(['Continent', 'Country'], axis=1, inplace=True)
# normalize the data
df = (df-df.min())/(df.max()-df.min())
# check again to see if everything is okay
df.head()
```

[28]:
```
   Density Per SqMile       Pop  Confirmed Cases    Deaths  Fatality Rate (%)  \
0            0.013509  0.079655         0.007998  0.006665           0.152174
1            0.011055  0.037130         0.004959  0.004964           0.184783
2            0.005445  0.019006         0.001158  0.001223           0.195652
3            0.024031  0.013057         0.001006  0.001951           0.358696
4            0.074746  0.000647         0.000000  0.000000           0.141304

   Samples Tested  Confirmed/Tested (%)  Tested/Pop (%)  Confirmed/Pop (%)  \
0        0.005659              0.230319        0.003224           0.002443
1        0.002882              0.188209        0.004164           0.002556
2        0.000172              0.387063        0.000501           0.000823
3        0.000422              0.342613        0.001659           0.001993
4        0.000555              0.002690        0.035626           0.000407

   Fragile Score  Vaccinated  Percent Vaccinated  Happiness index  \
0       0.831601    0.003573            0.333333         0.308852
```

```
1       0.786902    0.002292            0.014680         0.384586
2       0.674636    0.000000            0.000000         0.305036
3       0.721414    0.000817            0.014680         0.185235
4       0.234927    0.000520            0.144214         0.674170

   GDP per capita  Social support  Healthy life expectancy  \
0        0.171525        0.542642                 0.369335
1        0.280678        0.462375                 0.419479
2        0.124068        0.394649                 0.385728
3        0.077966        0.148829                 0.332690
4        0.686102        0.872910                 0.638380

   Freedom to make life choices  Generosity  Perceptions of corruption
0                      0.595960    0.400000                   0.219512
1                      0.748918    0.691228                   0.125704
2                      0.278499    0.335088                   0.142589
3                      0.702742    0.373684                   0.247655
4                      0.852814    0.328070                   0.157598
```

1.3.1 Simple Neural Network Models:

We first attempted to develop two non-linear neural network models to predict fatality rate and percent vaccinated.

A simple neural network model for **Fatality Rate (%)**:

[29]:
```
# get rid of some extreme data points
new = df[df["Tested/Pop (%)"] <= 0.2]
new = new[new["Confirmed/Tested (%)"] <= 0.8]
new = new[new["Generosity"] <= 0.8]
new = new[new["Social support"] >= 0.2]
new = new[new["Freedom to make life choices"] >= 0.2]
new = new[new["Fatality Rate (%)"] <= 0.55]
new.head()
```

[29]:
```
   Density Per SqMile       Pop  Confirmed Cases    Deaths  Fatality Rate (%)  \
0            0.013509  0.079655         0.007998  0.006665           0.152174
1            0.011055  0.037130         0.004959  0.004964           0.184783
2            0.005445  0.019006         0.001158  0.001223           0.195652
4            0.074746  0.000647         0.000000  0.000000           0.141304
5            0.004502  0.021483         0.002108  0.001388           0.119565

   Samples Tested  Confirmed/Tested (%)  Tested/Pop (%)  Confirmed/Pop (%)  \
0        0.005659              0.230319        0.003224           0.002443
1        0.002882              0.188209        0.004164           0.002556
2        0.000172              0.387063        0.000501           0.000823
4        0.000555              0.002690        0.035626           0.000407
5        0.001056              0.316879        0.002285           0.002443
```

```
   Fragile Score  Vaccinated  Percent Vaccinated  Happiness index  \
0       0.831601    0.003573            0.333333         0.308852
1       0.786902    0.002292            0.014680         0.384586
2       0.674636    0.000000            0.000000         0.305036
4       0.234927    0.000520            0.144214         0.674170
5       0.801455    0.000728            0.007772         0.392407

   GDP per capita  Social support  Healthy life expectancy  \
0        0.171525        0.542642                 0.369335
1        0.280678        0.462375                 0.419479
2        0.124068        0.394649                 0.385728
4        0.686102        0.872910                 0.638380
5        0.079322        0.504181                 0.215043

   Freedom to make life choices  Generosity  Perceptions of corruption
0                      0.595960    0.400000                   0.219512
1                      0.748918    0.691228                   0.125704
2                      0.278499    0.335088                   0.142589
4                      0.852814    0.328070                   0.157598
5                      0.809524    0.385965                   0.305816
```

[30]:
```
# select the features we want to look at
X = new[["Tested/Pop (%)", "Confirmed/Tested (%)", "Generosity", "Social␣
 ↪support", "Freedom to make life choices"]]
Y = new[["Fatality Rate (%)"]]

# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 2)
X_train = np.asarray(X_train)
X_test = np.asarray(X_test)
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

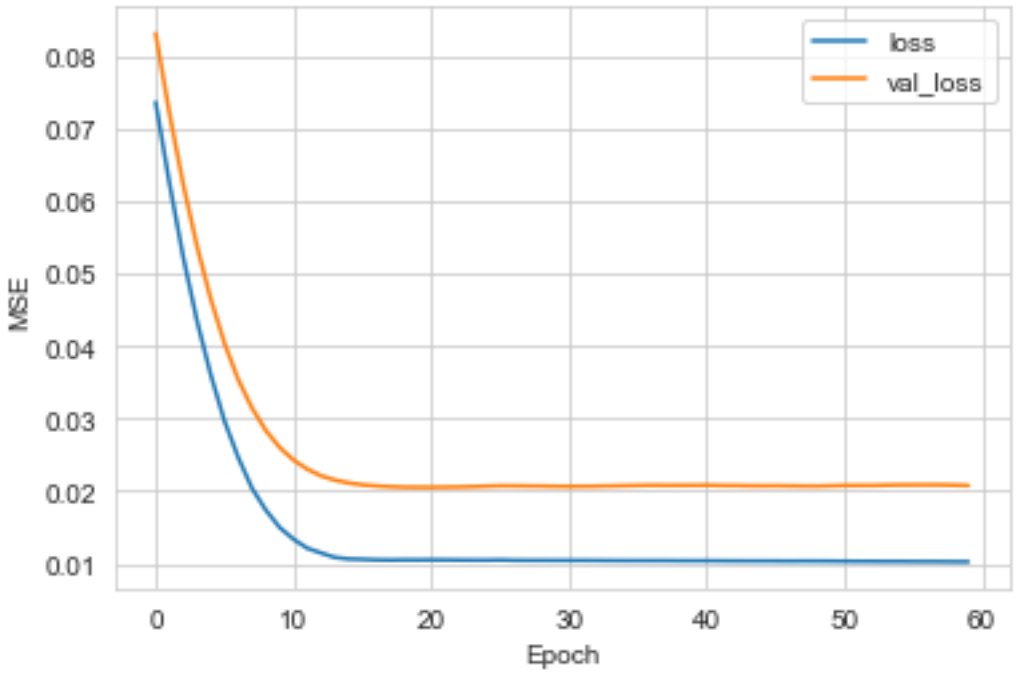
[31]:
```
model = tf.keras.models.Sequential([
    Input(shape=(5,)),
    layers.Dense(units = 6, activation = 'relu'),
    layers.Dense(units = 6, activation = 'sigmoid'),
    layers.Dense(units = 1, activation = None)
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 36
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 7
=================================================================
Total params: 85
Trainable params: 85
Non-trainable params: 0
_________________________________________________________________
```

[32]:
```
opt = keras.optimizers.Adam()
model.compile(optimizer=opt, loss='mse')

history = model.fit(X_train, y_train, epochs = 60,
          batch_size = 20,
          verbose=0,
          validation_split = 0.2)
```

[33]:
```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.legend()
plt.grid(True)
```



[34]:
```
# calculating r squared for training set
y_pred = model.predict(X_train)
r2_score(y_train, y_pred)
```

[34]:
```
0.04778314087611113
```

[35]:
```
# calculating r squared for testing set
y_pred = model.predict(X_test)
r2_score(y_test, y_pred)
```

[35]:
```
-0.025087871168087972
```

Despite having low MSEs, our simple neural network model was unable to accurately predict both the training set and the testing set.

A simple neural network model for **Percent Vaccinated**:

[36]:
```
# get rid of some extreme data points
new = df[df["Happiness index"] >= 0.1]
new = new[new["Perceptions of corruption"] <= 0.8]
new = new[new["Social support"] >= 0.2]
new = new[new["Percent Vaccinated"] <= 0.6]
new.head()
```

[36]:

	Density Per SqMile	Pop	Confirmed Cases	Deaths	Fatality Rate (%)	\
0	0.013509	0.079655	0.007998	0.006665	0.152174	
1	0.011055	0.037130	0.004959	0.004964	0.184783	
2	0.005445	0.019006	0.001158	0.001223	0.195652	
4	0.074746	0.000647	0.000000	0.000000	0.141304	
5	0.004502	0.021483	0.002108	0.001388	0.119565	

	Samples Tested	Confirmed/Tested (%)	Tested/Pop (%)	Confirmed/Pop (%)	\
0	0.005659	0.230319	0.003224	0.002443	
1	0.002882	0.188209	0.004164	0.002556	
2	0.000172	0.387063	0.000501	0.000823	
4	0.000555	0.002690	0.035626	0.000407	
5	0.001056	0.316879	0.002285	0.002443	

	Fragile Score	Vaccinated	Percent Vaccinated	Happiness index	\
0	0.831601	0.003573	0.333333	0.308852	
1	0.786902	0.002292	0.014680	0.384586	
2	0.674636	0.000000	0.000000	0.305036	
4	0.234927	0.000520	0.144214	0.674170	
5	0.801455	0.000728	0.007772	0.392407	

	GDP per capita	Social support	Healthy life expectancy	\
0	0.171525	0.542642	0.369335	
1	0.280678	0.462375	0.419479	

```
2        0.124068        0.394649                0.385728
4        0.686102        0.872910                0.638380
5        0.079322        0.504181                0.215043

   Freedom to make life choices  Generosity  Perceptions of corruption
0                      0.595960    0.400000                   0.219512
1                      0.748918    0.691228                   0.125704
2                      0.278499    0.335088                   0.142589
4                      0.852814    0.328070                   0.157598
5                      0.809524    0.385965                   0.305816
```

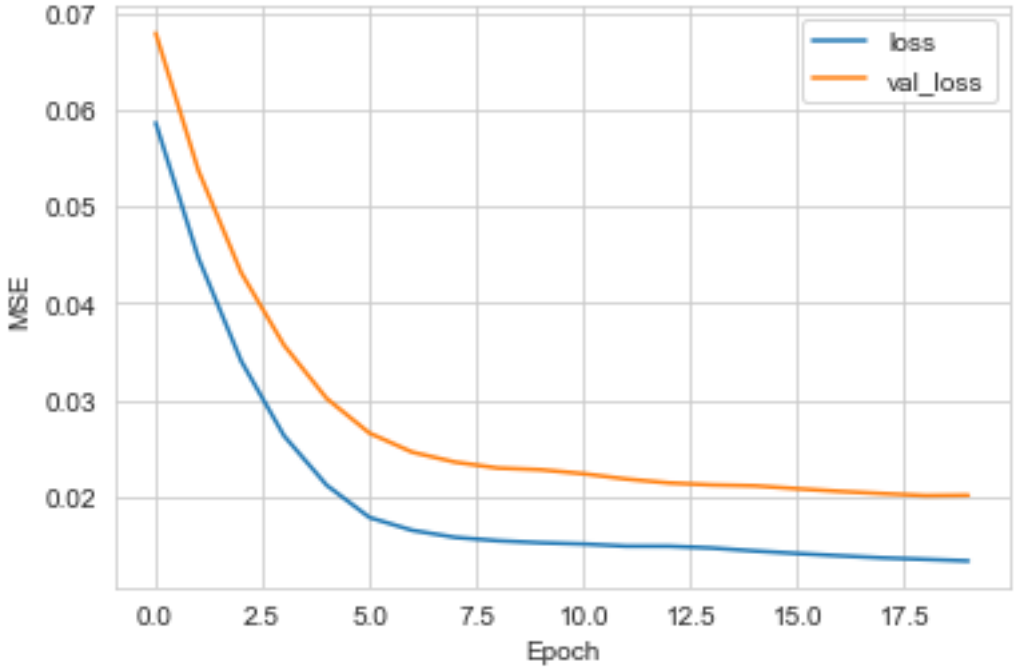
[37]:
```
# select the features we want to look at
X = new[['Social support', 'Perceptions of corruption', 'Fragile Score',␣
 ↪'Happiness index', 'GDP per capita']]
Y = new[["Percent Vaccinated"]]

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=10)
X_train = np.asarray(X_train)
X_test = np.asarray(X_test)
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

[38]:
```
model = tf.keras.models.Sequential([
    Input(shape=(5,)),
    layers.Dense(units = 6, activation = 'relu'),
    layers.Dense(units = 4, activation = 'sigmoid'),
    layers.Dense(units = 1, activation = None)
])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 6)                 36
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 28
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 5
=================================================================
Total params: 69
Trainable params: 69
Non-trainable params: 0
_________________________________________________________________
```

[39]:
```
opt = keras.optimizers.Adam()
model.compile(optimizer=opt, loss='mse')

history = model.fit(X_train, y_train, epochs = 20,
          batch_size = 10,
          verbose=0,
          validation_split = 0.2)
```

[40]:
```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.legend()
plt.grid(True)
```



[41]:
```
# calculating r squared for training set
y_pred = model.predict(X_train)
r2_score(y_train, y_pred)
```

[41]: 0.2613191850206029

[42]:
```
# calculating r squared for testing set
y_pred = model.predict(X_test)
r2_score(y_test, y_pred)
```

[42]: 0.2342712608971529

Again, it seemed that a simple neural network model did not work well on the data we have. As a result, we decided to look for established models to fit our data to.

1.3.2 Non-linear vs Linear Models:

Now we had to decide whether to use a non-linear or linear model to predict the Fatality Rate and Percent vaccinated. In order to make this decision, we decided to try multiple non-linear and linear models from sklearn and assess them using their R^2 scores.

After looking at multiple scatter plots and correlations in the data, we predict that non-linear models will fit the data much better, but it does not hurt to try a few linear models as well.

Non-linear Models: Testing different non-linear Models to predict **Fatality Rate**:

[43]:
```
# select the features we want to look at
X = df[["Tested/Pop (%)", "Confirmed/Tested (%)", "Generosity", "Social␣
 ↪support", "Freedom to make life choices"]]
Y = df[["Fatality Rate (%)"]]

# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=100)
# "de-dataframe" y_train and y_test
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

Training sets and testing sets were all set at this point; now it's time to try out different models:

[50]:
```
# training GradientBoostingRegressor
# the testing result was not optimal
model = GradientBoostingRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.95
Test R2 score: 0.18
```

[51]:
```
# training RandomForestRegressor
# train score decreased slightly, but test score increased significantly
model = RandomForestRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.88
Test R2 score: 0.36
```

[52]:
```
# training BaggingRegressor
# the performance was mediocre
model = BaggingRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.86
Test R2 score: 0.29
```

[53]:
```
# training AdaBoostRegressor
# just terrible
model = AdaBoostRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.64
Test R2 score: 0.21
```

[54]:
```
# training DecisionTreeRegressor
# it's yielding a really impressive training score, but very low test score
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 1.00
Test R2 score: 0.31
```

[55]:
```
# training VotingRegressor
# what about combining the two models that were both working quite well?
# the result was satisfactory - it's giving us the best predictive power
reg1 = DecisionTreeRegressor()
reg2 = RandomForestRegressor()
model = VotingRegressor(estimators=[('dt', reg1), ('rf', reg2)])
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.97
Test R2 score: 0.53
```

[56]:
```
# finding the best parameters for the KNeighborsRegressor
# we see it gets the best results at k = 3
kVals = np.arange(1,10,2)
for k in kVals:
    model = KNeighborsRegressor(k, p=2)
    model.fit(X_train, y_train)
    print('k=', k)
    print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
    print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
k= 1
Train R2 score: 1.00
Test R2 score: -0.02
k= 3
Train R2 score: 0.51
Test R2 score: 0.30
k= 5
Train R2 score: 0.43
Test R2 score: 0.25
k= 7
Train R2 score: 0.35
Test R2 score: 0.22
k= 9
Train R2 score: 0.29
Test R2 score: 0.17
```

[57]:
```
# training KNeighborsRegressor
# the result was mediocre
model = KNeighborsRegressor(3, p=2)
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.51
Test R2 score: 0.30
```

[58]:
```
# training polynomial model
# the result was pretty bad
degree=2
model=make_pipeline(PolynomialFeatures(degree),LinearRegression())
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.42
Test R2 score: 0.16
```

From all of our non-linear models above for Fatality Rate, the **VotingRegressor** that combined the RandomForestRegressor and the DecisionTreeRegressor yielded the best results with Train R^2 as 0.97 and Test R^2 as 0.43 (scores may vary slightly becasue of randomness)

Now, we experimented with different non-linear models to predict **Percent Vaccinated**:

[59]:
```
# select the features we want to look at
X = df[['Social support', 'Perceptions of corruption', 'Fragile Score',␣
 ↪'Happiness index', 'GDP per capita']]
Y = df[["Percent Vaccinated"]]

# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=100)
# "de-dataframe" y_train and y_test
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

Training sets and testing sets were all set at this point; now it's time to try out different models:

[60]:
```
# traning GradientBoostingRegressor
# the result was not bad
model = GradientBoostingRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.97
Test R2 score: 0.32
```

[61]:
```
# training RandomForestRegressor
# it was performing almost as well as the last one
model = RandomForestRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.89
Test R2 score: 0.36
```

[62]:
```
# training BaggingRegressor
# the performance was much worse than the previous ones
model = BaggingRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.88
Test R2 score: 0.20
```

[63]:
```
# training AdaBoostRegressor
# though giving us an acceptable train score, the test score was not good
model = AdaBoostRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.75
Test R2 score: 0.36
```

[64]:
```
# training DecisionTreeRegressor
# the test score was not good, the train score was very good
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.99
Test R2 score: 0.29
```

[65]:
```
# training VotingRegressor
# what about combining the two models that were both working quite well?
# the result seemed to balance out the strengths and weaknesses from the two␣
 ↪models
reg1 = GradientBoostingRegressor()
reg2 = RandomForestRegressor()
model = VotingRegressor(estimators=[('gb', reg1), ('br', reg2)])
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.95
Test R2 score: 0.33
```

[66]:
```
# finding the best parameters for the KNeighborsRegressor
# we see it gets the best results at k = 3
kVals = np.arange(1,10,2)
for k in kVals:
    model = KNeighborsRegressor(k, p=2)
    model.fit(X_train, y_train)
    print('k=', k)
    print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
    print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
k= 1
Train R2 score: 0.97
Test R2 score: 0.05
k= 3
Train R2 score: 0.64
Test R2 score: 0.41
k= 5
Train R2 score: 0.54
Test R2 score: 0.53
k= 7
Train R2 score: 0.51
Test R2 score: 0.54
k= 9
Train R2 score: 0.49
Test R2 score: 0.58
```

[67]:
```
# training KNeighborsRegressor
# the result was mediocre
model = KNeighborsRegressor(3, p=2)
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.64
Test R2 score: 0.41
```

[68]:
```
# training polynomial model
# the result was mediocre
degree=2
model=make_pipeline(PolynomialFeatures(degree),LinearRegression())
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.59
Test R2 score: 0.28
```

From all of our non-linear models above for Percent Vaccinated, the **VotingRegressor** that combined GradientBoostingRegressor and RandomForestRegressor, yielded the best results with Train R^2 as 0.94 and Test R^2 as 0.36 (scores may vary slightly becasue of randomness)

Linear Models: Testing different linear models to predict **Fatality Rate**:

[69]:
```
# select the features we want to look at
X = df[["Tested/Pop (%)", "Confirmed/Tested (%)", "Generosity", "Social␣
 ↪support", "Freedom to make life choices"]]
Y = df[["Fatality Rate (%)"]]
```

```
# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=100)
# "de-dataframe" y_train and y_test
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

Training sets and testing sets were all set at this point; now it's time to try out different models:

[70]:
```
# training LinearRegression model
# the results were very poor
model = LinearRegression()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.31
Test R2 score: 0.13
```

[71]:
```
# training SGDRegressor
# the results were terrible!
model = SGDRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.03
Test R2 score: -0.09
```

[72]:
```
# training BayesianRidge
# the results seem the best
model = BayesianRidge()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.31
Test R2 score: 0.15
```

From all of our linear models above for Fatality Rate, the **BayesianRidge** yielded the best results with Train R^2 as 0.31 and Test R^2 as 0.15 (scores may vary slightly becasue of randomness)

Now we test different linear models to predict **Percent Vaccinated**:

[73]:
```
# select the features we want to look at
```

```
X = df[['Social support', 'Perceptions of corruption', 'Fragile Score',␣
 ↪'Happiness index', 'GDP per capita']]
Y = df[["Percent Vaccinated"]]

# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=100)
# "de-dataframe" y_train and y_test
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

Training sets and testing sets were all set at this point; now it's time to try out different models:

[74]:
```
# training LinearRegression model
# the results were very mediocre
model = LinearRegression()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.39
Test R2 score: 0.50
```

[75]:
```
# training SGDRegressor
# the results were terrible!
model = SGDRegressor()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.22
Test R2 score: 0.27
```

[76]:
```
# training BayesianRidge
# the results seem the best
model = BayesianRidge()
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

```
Train R2 score: 0.38
Test R2 score: 0.52
```

From all of our linear models above for Fatality Rate, the **BayesianRidge** again yielded the best results with Train R^2 as 0.38 and Test R^2 as 0.52 (scores may vary slightly becasue of randomness)

Overall Model Results: It is clear that non-linear models, overall, covered much more variability in the data compared the linear models. Much of the time, linear models could not even compare. Thus, our prediction that non-linear models would be better fit for the data was correct.

For the **Fatality Rate**, we found that the **VotingRegressor** that combined the RandomForestRegressor and the DecisionTreeRegressor yielded the best results with Train R^2 as 0.97 and Test R^2 as 0.43 (scores may vary slightly becasue of randomness)

For the **Percent Vaccinated**, we found that the **VotingRegressor**, a combination of the GradientBoostingRegressor and RandomForestRegressor, yielded the best results with Train R^2 as 0.94 and Test R^2 as 0.36 (scores may vary slightly becasue of randomness)

1.3.3 PDP and ICE Plots:

Here we have defined multiple helper functions myICE and my_Partial_Dependece for the main function model_pdp to plot multiple ICE and PDP plots for a model. This allows us to analyze how much a certain feature affects a model, giving us insight on the impact each feature has on a country's Fatality Rate and Percent Vaccinated.

Note that the ipynb file that contains the code for the initial testing and building of these functions below can be found in the following link, specifically in the Linear_Model_Tests.ipynb: https://drive.google.com/drive/folders/1CQ_83XFQomNBSy43TeTDqaUWz7eqVxcD?usp=sharing

[77]:
```
# helper function that computes the individual conditional expectation of the␣
 ↪target outcome predicted by
# the trained model for a single sample with the index sample_idx from data
def myICE(data, model, feature_idx, sample_idx, grid_resolution=100):
    feature_values = np.linspace(np.min(data[:, feature_idx]), np.max(data[:,␣
 ↪feature_idx]), grid_resolution)
    # creates 100 different values for the feature
    predictions = np.empty([100,])
    sample = data[sample_idx].copy()
    i = 0
    for value in feature_values:
        sample[feature_idx] = value
        predictions[i] = model.predict(sample.reshape(1, -1)) # gets the sample␣
 ↪output for each value in feature_values
        i += 1
    return feature_values, predictions
```

[78]:
```
# helper function that computes the partial dependence of the target outcome␣
 ↪predicted by a trained model on a
# feature of interest with index feature_idx, marginalizing over the values of␣
 ↪all other features using samples specified in sample_idx_list.
def my_Partial_Dependence(data, model, feature_idx, sample_idx_list,␣
 ↪grid_resolution=100):
    feature_values, predictions_sum = myICE(data, model, feature_idx,␣
 ↪sample_idx_list[0], grid_resolution=100)
```

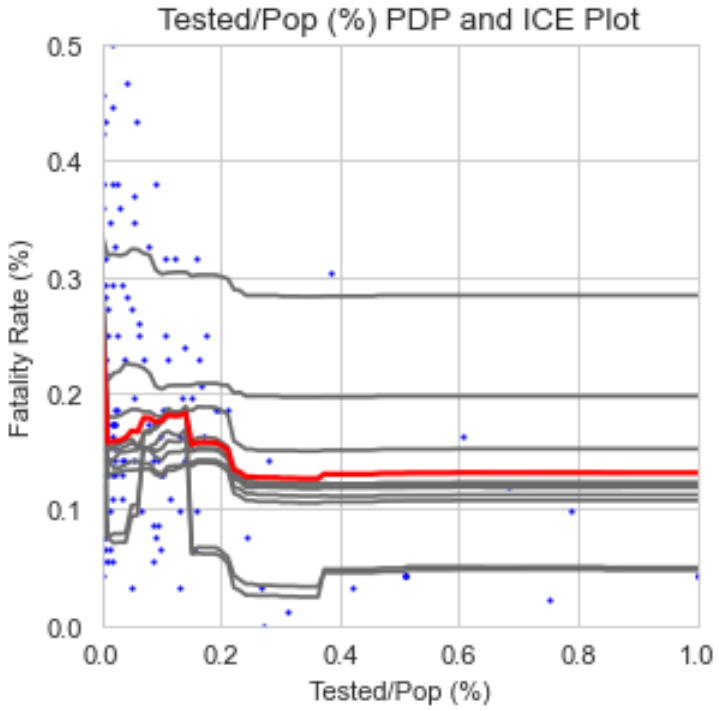
```
    # takes the element wise sum of the prediction array from each sample in␣
↪sample_idx
    for sample_idx in sample_idx_list[1:]:
        predictions_sum += myICE(data, model, feature_idx, sample_idx,␣
↪grid_resolution=100)[1]
    # takes the average of each feature value in order to get an average change␣
↪in output for each feature_value
    m = len(sample_idx_list)
    prediction_avg = predictions_sum / m
    return feature_values, prediction_avg
```

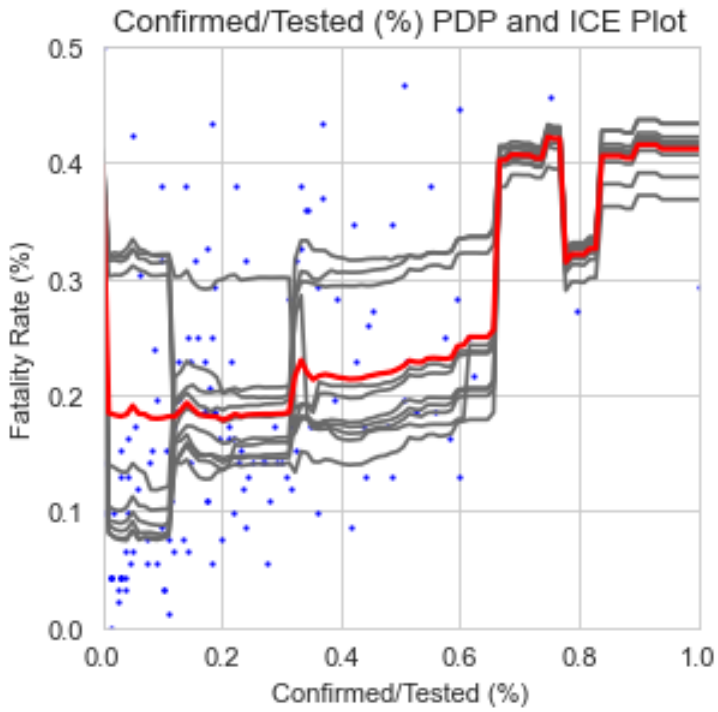
[79]:
```
# main function that automatically plots the ICE and PDP plots for the model␣
↪and desired list features, given the data X,
# output y, and number of samples sample_num
def model_pdp(model, X, Y, features, sample_num):
    # train and fit model to data
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
↪random_state=100)
    y_train = np.asarray(y_train).ravel()
    y_test = np.asarray(y_test).ravel()
    model.fit(X_train, y_train)

    y_name = Y.columns[0]
    X = np.asarray(X) # convert X to array
    sample_idx_list = np.arange(sample_num) # create a list of samples to plot
    for i, feature in enumerate(features):
        # generates sample_num curves from myICE function
        feature_idx = i
        plt.figure(figsize = (4, 4), dpi = 80)
        plt.xlim([0,1])
        plt.ylim([0,0.5])
        for sample_idx in sample_idx_list:
            feature_vals, pred_y = myICE(X, model, feature_idx = feature_idx,␣
↪sample_idx = sample_idx)
            plt.plot(feature_vals, pred_y, 'dimgray')

        # generate a PDP curve from the Partial Dependence function
        feature_vals, PD = my_Partial_Dependence(X, model, feature_idx,␣
↪sample_idx_list)
        plt.plot(feature_vals, PD, 'r-', linewidth=2)

        plt.scatter(X[:, feature_idx], Y, c='b', s = 1)
        plt.title("%s PDP and ICE Plot"%features[i])
        plt.ylabel("%s"%y_name)
        plt.xlabel("%s"%features[i])
```

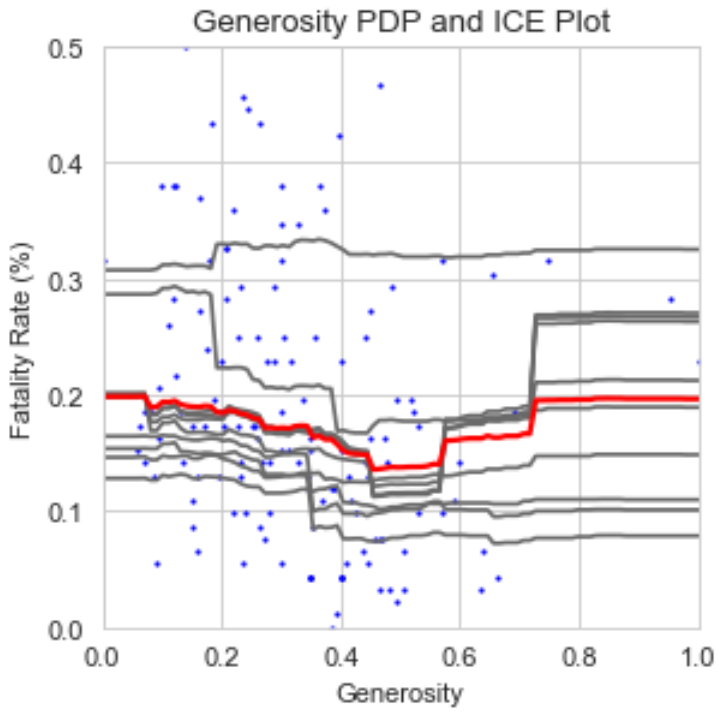
```
        plt.show()
    return None
```

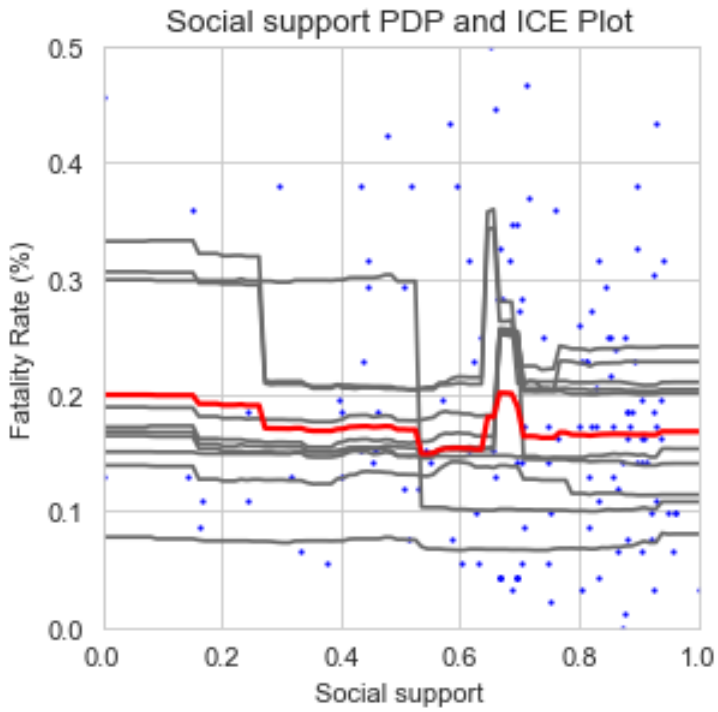
PDP and ICE plots for Fatality Rate: Here we will use PDP and ICE plots to dissect how individual features affect the output fatality rate in our best performing model, VotingRegressor.

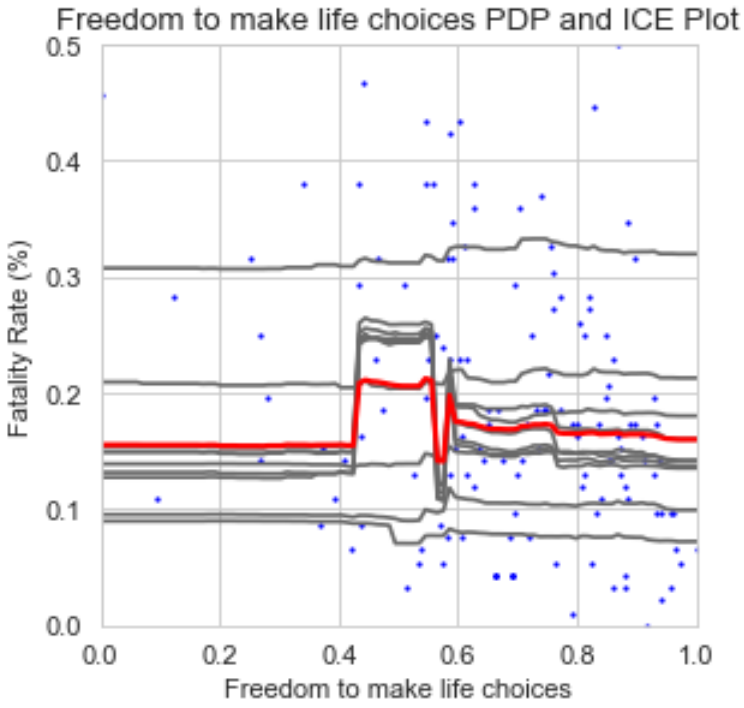
[80]:
```
# Generate PDP and ICE plots for the VotingRegressor that predicts Fatality Rate
features = ["Tested/Pop (%)", "Confirmed/Tested (%)", "Generosity", "Social␣
 ↪support", "Freedom to make life choices"]
X = df[features]
y = df[["Fatality Rate (%)"]]
reg1 = DecisionTreeRegressor()
reg2 = RandomForestRegressor()
model = VotingRegressor(estimators=[('dt', reg1), ('rf', reg2)])
model_pdp(model, X, y, features, sample_num = 10)
```



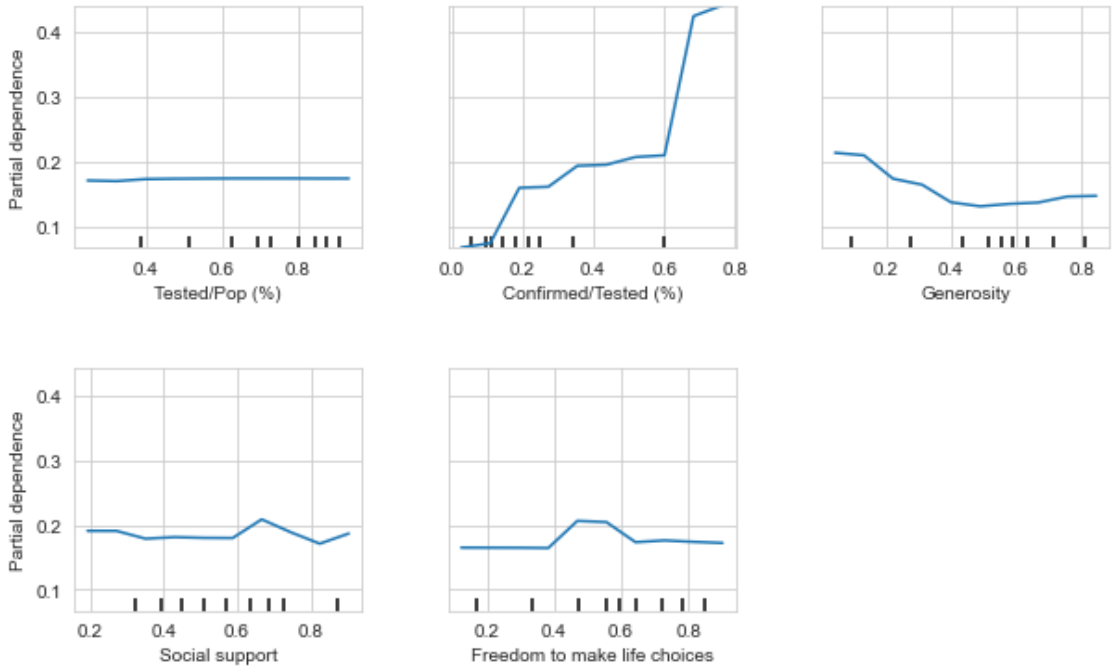

Confirmed/Tested (%) PDP and ICE Plot
Fatality Rate (%)
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Confirmed/Tested (%)


Generosity PDP and ICE Plot
Fatality Rate (%)
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Generosity


Social support PDP and ICE Plot
Fatality Rate (%)
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Social support


Freedom to make life choices PDP and ICE Plot
Fatality Rate (%)
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Freedom to make life choices

[81]:
```
# generating PDP plots only
plot_partial_dependence(model,
                        features=features,
                        X=X_train,
                        feature_names=X.columns,
                        grid_resolution=10)

fig = plt.gcf()
fig.subplots_adjust(wspace=0.3, hspace=0.5)
fig.set_figheight(6)
fig.set_figwidth(10)
```



Results for Fatality Rate PDP:

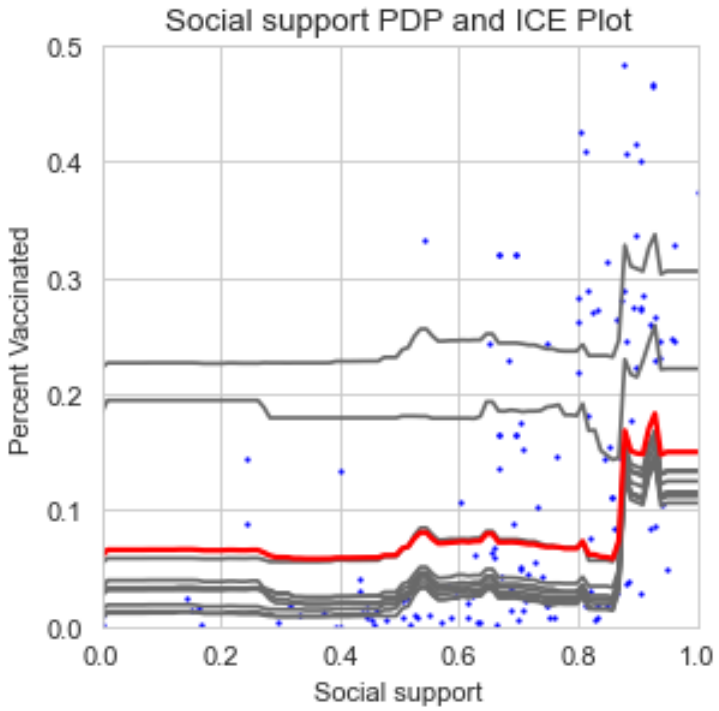
- Tested/Pop (%): we see that there seems to be a slight partition in this feature's contributions the model. The feature had a slight negative effect on the predicted outcome of the model, only in the lower test values for the feature (around 0-0.3). Suggesting that as the rate of testing increased, the fatality rate was predicted to decrease. However, the results are quite negligible after 0.3, as the change in testing rate does very little to change the fatality rate prediction by the model
- Confirmed/Tested (%): for this feature, we see a general positive correlation, suggesting that as the rate of positive tests increased so did the predicted fatality rate.
- Generosity: Interestingly, here we see a slight negative relationship between the level of generosity and the fatality rate. This suggests that the model predicts a lower fatality rate when the generosity of a country is higher.
- Social Support: the marginal effects of social support on the predicted fatality rate of the

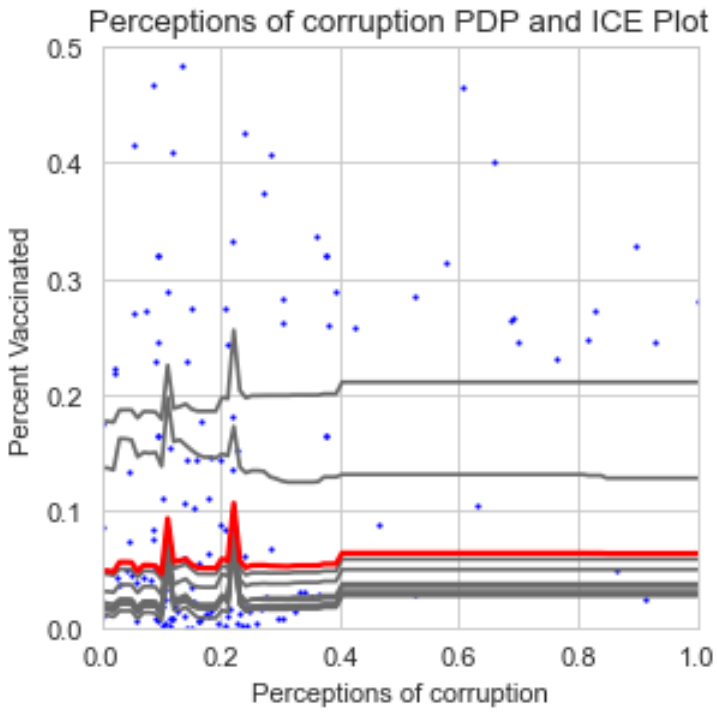
model seems negligible. Looking at the scatter plot itself, there seems to be no trend between the two.

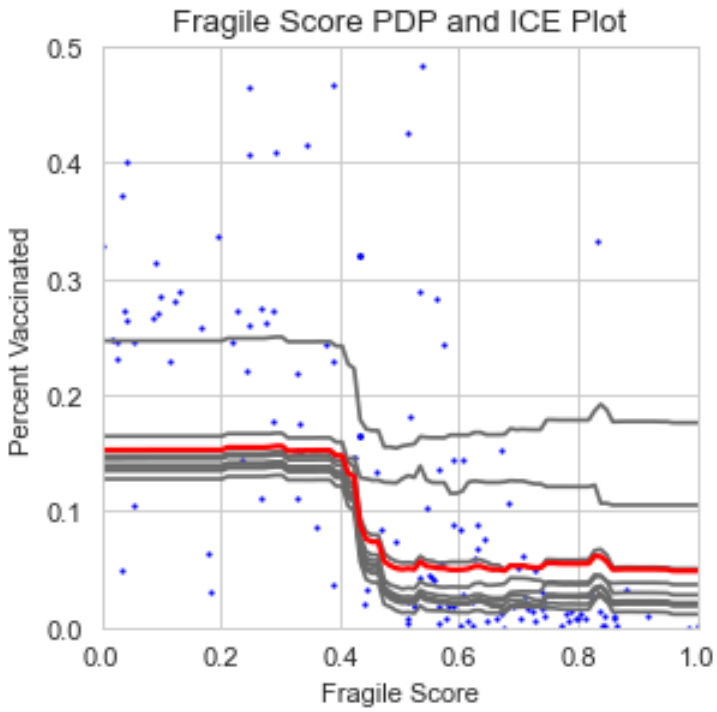
- Freedom to make life choices: again, this feature's effect on the predicted fatality rate of the model seems negligible. There is a possible positive correlation from 0.4 to 0.6, which then seems to level out. Suggesting that at low values, the perceived level of freedom of choice seems to have a slight positive effect on the fatality rate.

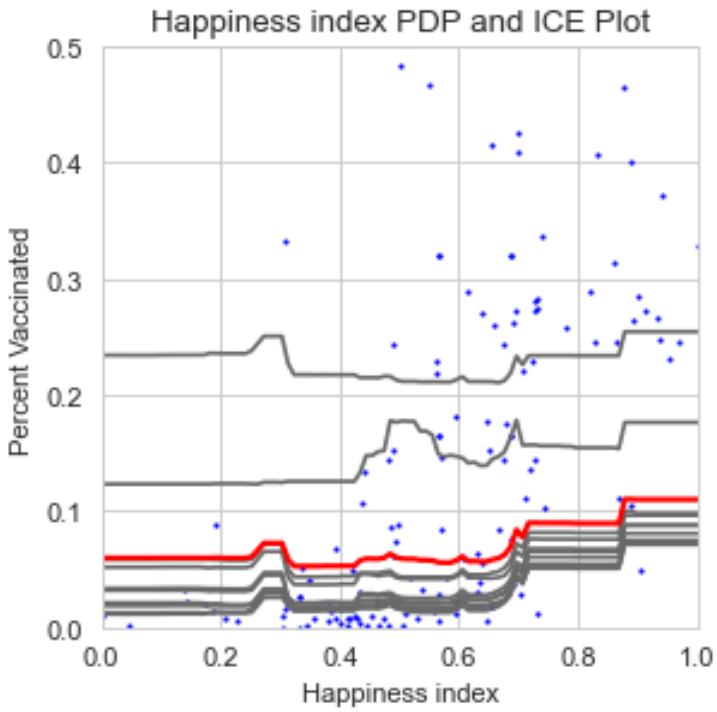
PDP and ICE plots for Percent Vaccinated: Here we will use PDP and ICE plots to dissect how individual features affect the output percent vaccinated in our best performing model, VotingRegressor.

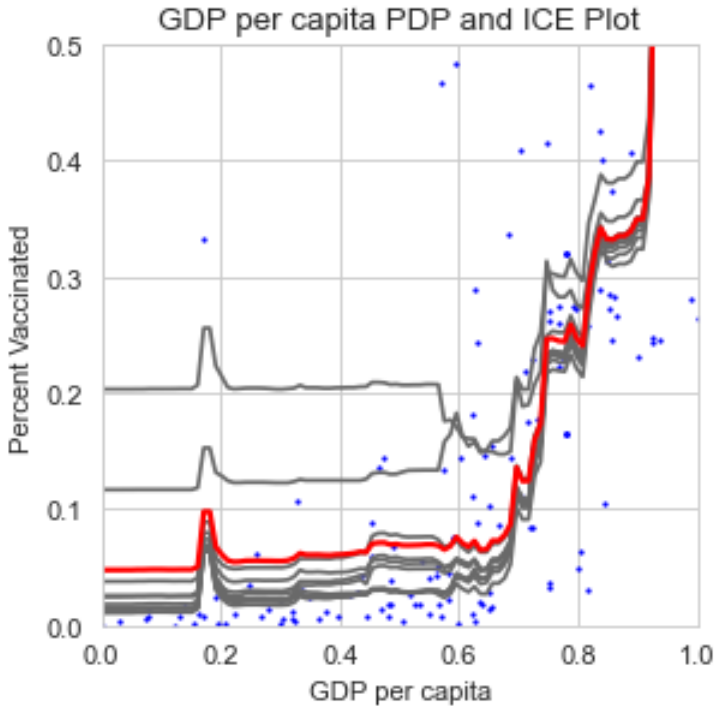
[82]:
```
features = ['Social support', 'Perceptions of corruption', 'Fragile Score',␣
 ↪'Happiness index', 'GDP per capita']
X = df[features]
y = df[["Percent Vaccinated"]]
reg1 = GradientBoostingRegressor()
reg2 = RandomForestRegressor()
model = VotingRegressor(estimators=[('gb', reg1), ('br', reg2)])
model_pdp(model, X, y, features, sample_num = 10)
```



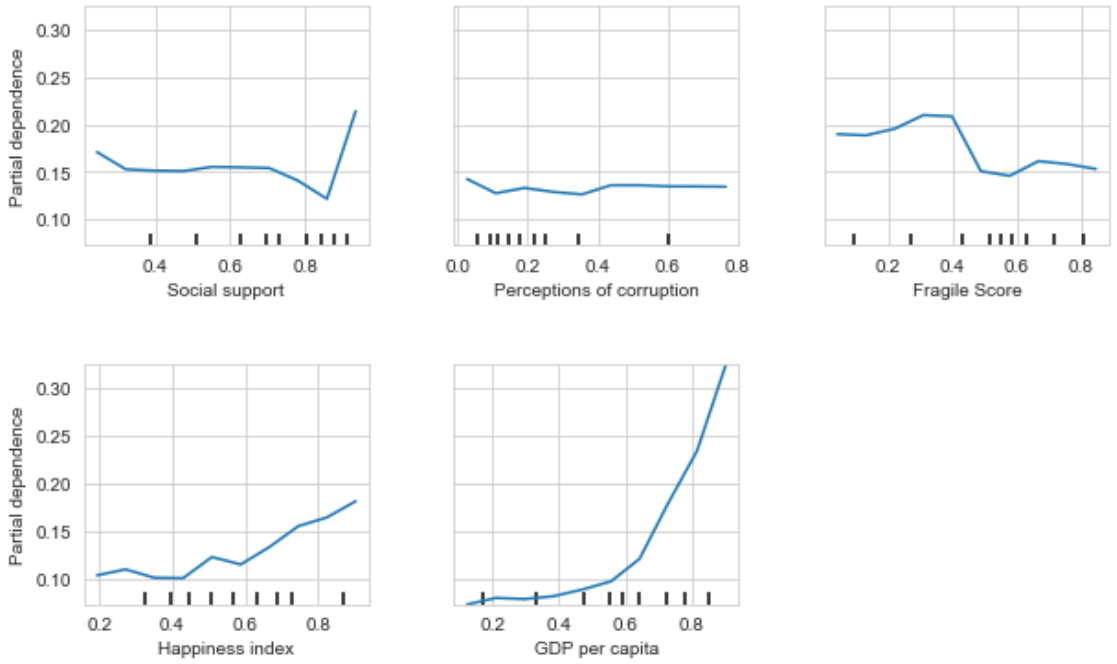

Perceptions of corruption PDP and ICE Plot
Percent Vaccinated
Perceptions of corruption
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0


Fragile Score PDP and ICE Plot
Percent Vaccinated
Fragile Score
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0


Happiness index PDP and ICE Plot
Percent Vaccinated
Happiness index
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0


GDP per capita PDP and ICE Plot
Percent Vaccinated
GDP per capita
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0

[83]:
```
# generating PDP plots only
plot_partial_dependence(model,
                        features=features,
                        X=X_train,
                        feature_names=X.columns,
                        grid_resolution=10)

fig = plt.gcf()
fig.subplots_adjust(wspace=0.3, hspace=0.5)
fig.set_figheight(6)
fig.set_figwidth(10)
```



Results for Percent Vaccinated PDP:

- Social Support: we see a slight positive correlation in the high values for social support, however the rest of the graph suggests that social support has very little marginal effect on the predicted vaccination rate. This suggests that only at high levels does social support somewhat play a factor in increasing the predicted percent vaccinated.
- Perceptions of corruption: clearly, perceptions of corruption have pretty much no effect on the predicted output of the model. Looking at the scatterplot itself, we see that it lacks any sort of correlation.
- Fragile score: interestingly, there seems to be a negative correlation between the fragile score and the predicted percent vaccinated around 0.3-0.6. This suggests that as the fragile score increases, the predicted percent vaccinated decreases.
- Happiness index: surprisingly, the happiness index seems to have a slight positive marginal effect on the predicted output of the model. This suggests that when the happiness index in

a country is higher, the model predicts a higher vaccination rate.

- GDP per capita: GDP seems to have a positive effect on the model's predicted vaccination rate. We notice that it seems to take the shape of an exponential function, implying that as the GDP increases the predicted vaccination rate increases exponentially.

1.3.4 Interactions between Features:

Are there any patterns that we could not tell from the PDP and ICE plots? In this section, we will be looking at interactions within groups of two features.

Note that the ipynb file that contains our initial work for the code below can be found in the following link, specifically in the trial&error_v2.ipynb: https://drive.google.com/drive/folders/1CQ_83XFQomNBSy43TeTDqaUWz7eqVxcD?usp=sharing

For Fatality Rate: Several prominent interactions between the features for **Fatality Rate (%)** are shown below

[84]:
```
# select the features we want to look at
X = df[["Tested/Pop (%)", "Confirmed/Tested (%)", "Generosity", "Social␣
 ↪support", "Freedom to make life choices"]]
Y = df[["Fatality Rate (%)"]]

# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=100)
# "de-dataframe" y_train and y_test
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

[85]:
```
# run the model we evaluated to be the best among all
reg1 = DecisionTreeRegressor()
reg2 = RandomForestRegressor()
model = VotingRegressor(estimators=[('dt', reg1), ('rf', reg2)])
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

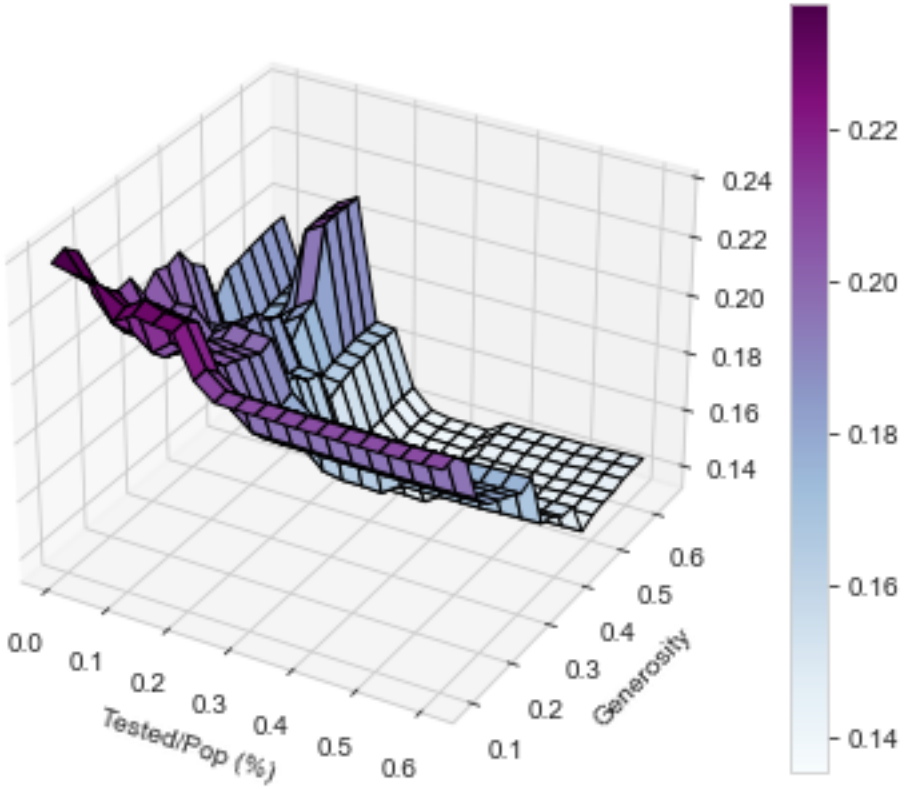
```
Train R2 score: 0.97
Test R2 score: 0.51
```

There is an interaction between **Tested/Pop (%)** and **Generosity**, such that there was a sudden surge in predictive power when both the tested proportion and generosity were low. A possible explanation was that, if a country had low tested proportion and low generosity score, then the overall living condition would not be optimal, thus might indicate a higher fatality rate.

[86]:
```
fig = plt.figure()
features = (0,2)
```

```
pdp, axes = partial_dependence(model, X_train,
                               features=features,
                               grid_resolution=20)
XX, YY = np.meshgrid(axes[0], axes[1])
Z = pdp[0].T
ax = Axes3D(fig)
surf = ax.plot_surface(XX, YY, Z, rstride=1, cstride=1,
                       cmap=plt.cm.BuPu, edgecolor='k')
ax.set_xlabel(X.columns[features[0]])
ax.set_ylabel(X.columns[features[1]])
plt.colorbar(surf)
plt.show()
```

```
/opt/anaconda3/envs/pic16bmachine/lib/python3.7/site-
packages/sklearn/inspection/_partial_dependence.py:510: FutureWarning: A Bunch
will be returned in place of 'predictions' from version 1.1 (renaming of 0.26)
with partial dependence results accessible via the 'average' key. In the
meantime, pass kind='average' to get the future behaviour.
  FutureWarning
```

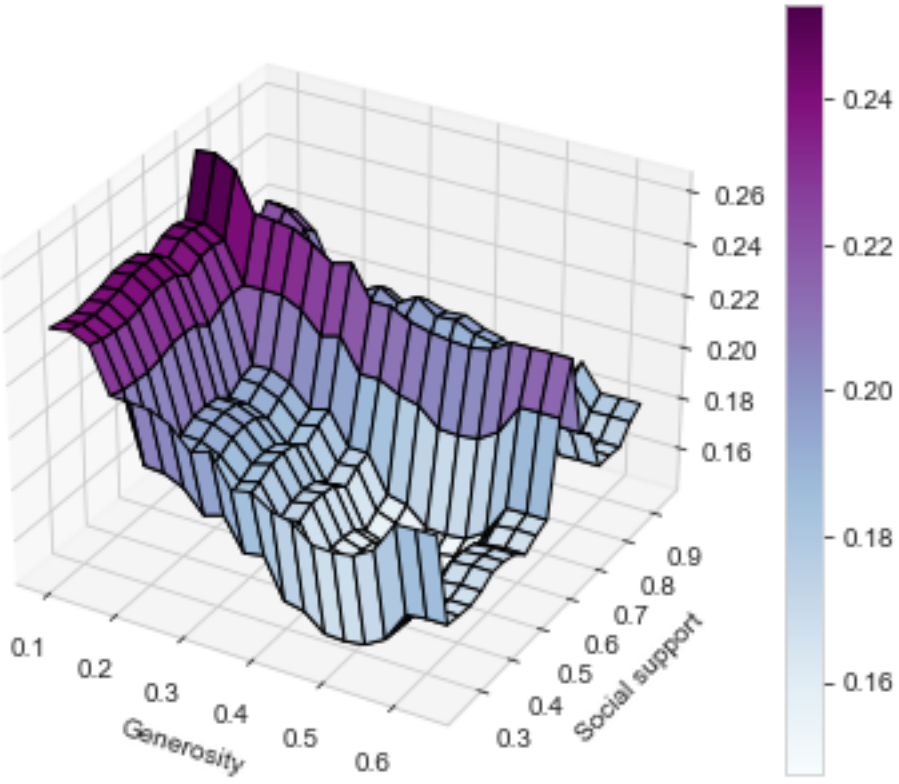


An interaction between **Social support** and **Confirmed/Tested (%)** revealed that, social support was predicitive of fatality rate when it is moderately high; however, low social support could

also be predicitve when generosity rating was extremley low:

[87]:
```
fig = plt.figure()
features = (2,3)

# an interaction between social support and confirmed/tested (%)
pdp, axes = partial_dependence(model, X_train,
                               features=features,
                               grid_resolution=20)
XX, YY = np.meshgrid(axes[0], axes[1])
Z = pdp[0].T
ax = Axes3D(fig)
surf = ax.plot_surface(XX, YY, Z, rstride=1, cstride=1,
                       cmap=plt.cm.BuPu, edgecolor='k')
ax.set_xlabel(X.columns[features[0]])
ax.set_ylabel(X.columns[features[1]])
plt.colorbar(surf)
plt.show()
```

```
/opt/anaconda3/envs/pic16bmachine/lib/python3.7/site-
packages/sklearn/inspection/_partial_dependence.py:510: FutureWarning: A Bunch
will be returned in place of 'predictions' from version 1.1 (renaming of 0.26)
with partial dependence results accessible via the 'average' key. In the
meantime, pass kind='average' to get the future behaviour.
  FutureWarning
```



For Percent Vaccinated: Several prominent interactions between the features for **Percent Vaccinated (%)** are shown below:

[88]:
```
# select the features we want to look at
X = df[['Social support', 'Perceptions of corruption', 'Fragile Score',␣
 ↪'Happiness index', 'GDP per capita']]
Y = df[["Percent Vaccinated"]]

# prepare to train the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state=100)
# "de-dataframe" y_train and y_test
y_train = np.asarray(y_train).ravel()
y_test = np.asarray(y_test).ravel()
```

[89]:
```
# run the model we evaluated to be the best among all
reg1 = GradientBoostingRegressor()
reg2 = RandomForestRegressor()
model = VotingRegressor(estimators=[('gb', reg1), ('br', reg2)])
model.fit(X_train, y_train)

print("Train R2 score: {:.2f}".format(model.score(X_train, y_train)))
print("Test R2 score: {:.2f}".format(model.score(X_test, y_test)))
```

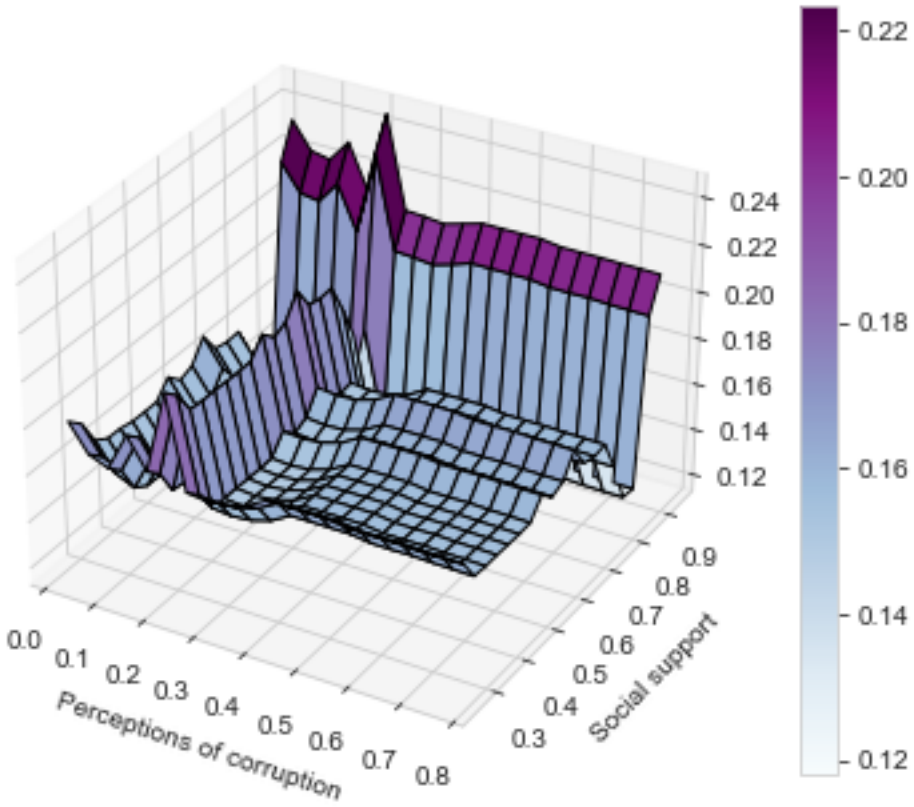
```
Train R2 score: 0.95
Test R2 score: 0.36
```

There seemed to be an interaction between **Perceptions of corruption** and **Social support**, in which low perception of corruption and high social support yielded high contribution to prediction on vaccinated rate. The intuition was that, societies with low corruption and high social support should be able to allocate resources more efficiently and provide their citizens sufficient vaccines.

[90]:
```
fig = plt.figure()
features = (1,0)

pdp, axes = partial_dependence(model, X_train,
                               features=features,
                               grid_resolution=20)
XX, YY = np.meshgrid(axes[0], axes[1])
Z = pdp[0].T
ax = Axes3D(fig)
surf = ax.plot_surface(XX, YY, Z, rstride=1, cstride=1,
                       cmap=plt.cm.BuPu, edgecolor='k')
ax.set_xlabel(X.columns[features[0]])
ax.set_ylabel(X.columns[features[1]])
```

```
plt.colorbar(surf)
plt.show()
```

```
/opt/anaconda3/envs/pic16bmachine/lib/python3.7/site-
packages/sklearn/inspection/_partial_dependence.py:510: FutureWarning: A Bunch
will be returned in place of 'predictions' from version 1.1 (renaming of 0.26)
with partial dependence results accessible via the 'average' key. In the
meantime, pass kind='average' to get the future behaviour.
  FutureWarning
```

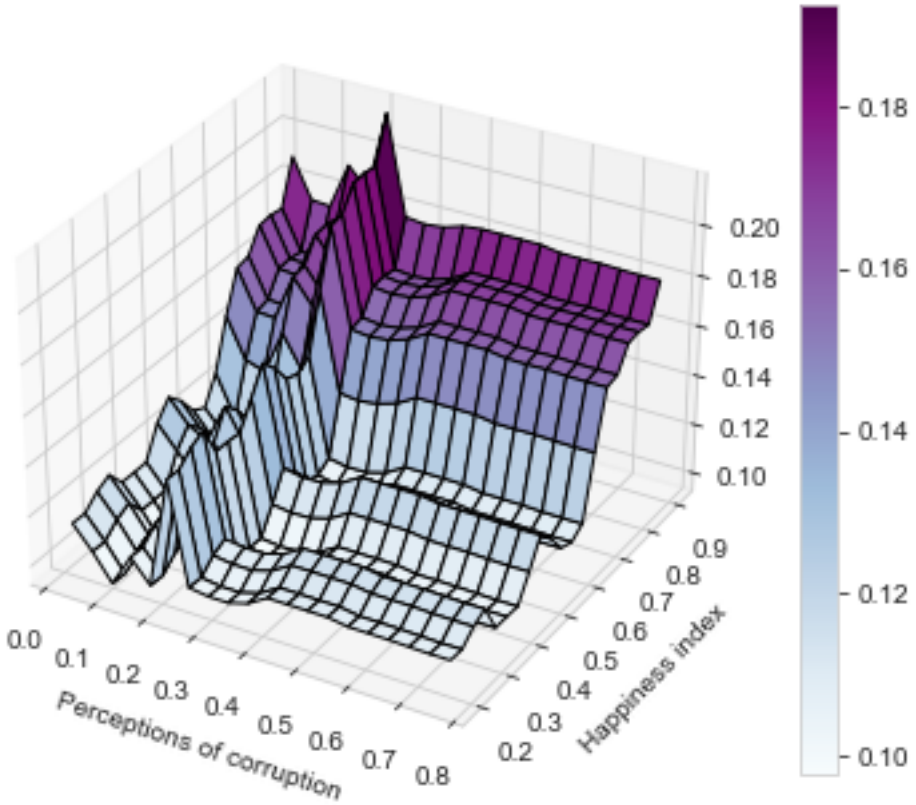


An interaction between **Perception of corruption** and **Happiness index** could be observed from the plot as well: low perception of corruption and high happiness index were highly predictive of percent vaccined. Similarly, a possible explanation was that, these two features indicated a healthy and high-functioning society, where people could readily access vaccines.

[91]:
```
fig = plt.figure()
features = (1,3)

pdp, axes = partial_dependence(model, X_train,
                               features=features,
                               grid_resolution=20)
```

```
XX, YY = np.meshgrid(axes[0], axes[1])
Z = pdp[0].T
ax = Axes3D(fig)
surf = ax.plot_surface(XX, YY, Z, rstride=1, cstride=1,
                       cmap=plt.cm.BuPu, edgecolor='k')
ax.set_xlabel(X.columns[features[0]])
ax.set_ylabel(X.columns[features[1]])
plt.colorbar(surf)
plt.show()
```

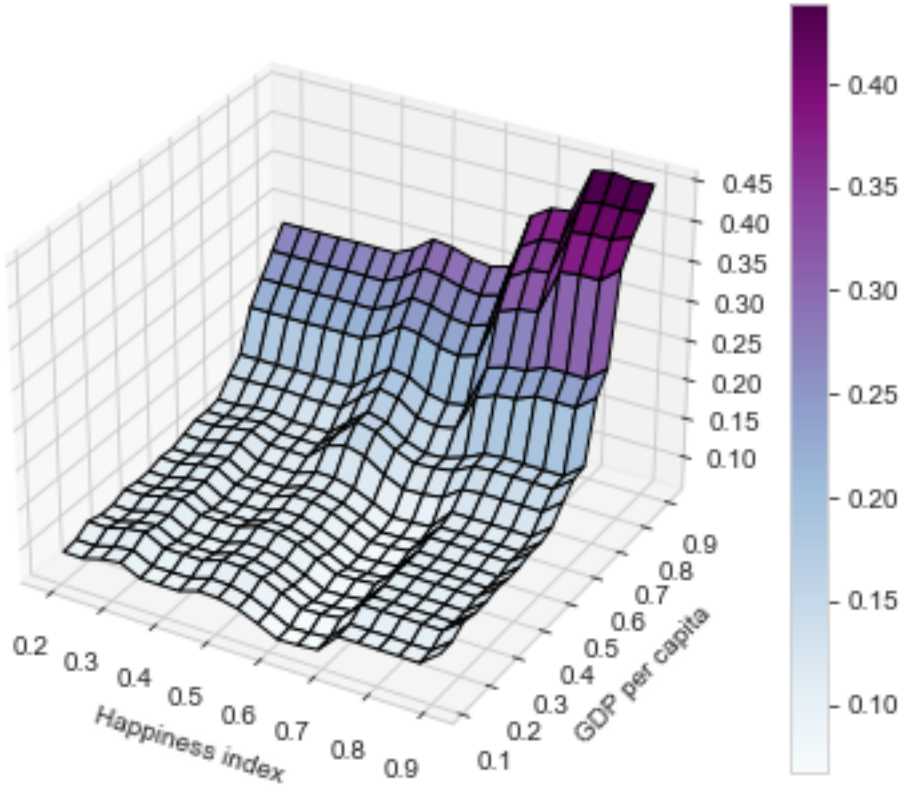
```
/opt/anaconda3/envs/pic16bmachine/lib/python3.7/site-
packages/sklearn/inspection/_partial_dependence.py:510: FutureWarning: A Bunch
will be returned in place of 'predictions' from version 1.1 (renaming of 0.26)
with partial dependence results accessible via the 'average' key. In the
meantime, pass kind='average' to get the future behaviour.
  FutureWarning
```



Happiness index and **GDP per capita** appeared to interact with each other, with high happiness index and high GDP per capita being conspicious contributor to percent vaccinated. The logic was also axiomatic—these two features collectively resulted in a flourishing country, thus making it rational to expect more vaccine being supplied.

[92]:
```
fig = plt.figure()
features = (3,4)

pdp, axes = partial_dependence(model, X_train,
                               features=features,
                               grid_resolution=20)
XX, YY = np.meshgrid(axes[0], axes[1])
Z = pdp[0].T
ax = Axes3D(fig)
surf = ax.plot_surface(XX, YY, Z, rstride=1, cstride=1,
                       cmap=plt.cm.BuPu, edgecolor='k')
ax.set_xlabel(X.columns[features[0]])
ax.set_ylabel(X.columns[features[1]])
plt.colorbar(surf)
plt.show()
```

```
/opt/anaconda3/envs/pic16bmachine/lib/python3.7/site-
packages/sklearn/inspection/_partial_dependence.py:510: FutureWarning: A Bunch
will be returned in place of 'predictions' from version 1.1 (renaming of 0.26)
with partial dependence results accessible via the 'average' key. In the
meantime, pass kind='average' to get the future behaviour.
  FutureWarning
```



1.4 Discussions/Conclusions:

Possible Issues:

In this project, we encountered several diffculties as we examined various correlations and constructed our models. First, we did not have a lot of data to work with (only about 140 countries). This lead to potential flunctuations in our results (R^2 scores, training loss, validation loss, etc). This is possibly due to the fact that we extracted data from many features, and at the regional levels we often could not obtain all the information needed for each country. Next, there were many outliers, as shown in the correlation plots above, that often skewed our data and made it difficult to plot accurate regressions. However, it was difficult deciding whether we should get rid of some of these outliers because some of these outlier countries are very significant. For instance, UAE has abnormally high vaccination rate because the country has a lot of immigrants and foreign workers, so we decided it would be ill-advised to remove this data point. We also noticed that some of the data we scraped was recorded at different times (i.e. beginning of the pandemic vs after vaccinations had come out). We suspect that this time lapse played a role in confounding our overall data and final results. For example, the Fatality Rate and its related data (like Tested/Population) started being recorded at the very beginning of the pandemic. However, the data for Percent Vaccinated came almost a year later. These discrepancies could have lead to unknown biases when we merged the data together.

We also note that most of the data we collected about a month ago could have changed significantly by now. For example, after we had scraped all of our data, India began to suffer a new wave of Covid-19 fatalities. In essence, our results may not be relevant in the near future. It is also important to observe that there are many possible confounding variables that could have contributed to our results, creating possible bias and random coincidences that we may have overlooked. For instance, in the multicollinearity heat map above, we see a significant correlation between the Happiness Index and GDP that may have confounded our models, however it is difficult to propose a relevant solution that would not compromise our R^2 scores. In order to build models that have higher accuracy and cover more variability in the data, this seems to be a necessary trade off. Finally, we note that some of the more abstract features such as the Happiness Index and Fragile Score are not very tangible and often only change in the long term. As a result, it is not possible for a country to change these features in order to increase their vaccination rate or decrease their fatality rate within the time frame of the pandemic.

Possible Improvements:

There are several ways to address these difficulties. We can scrape the data at the state/province level, which would provide us with a lot more data to construct a more accurate and complex model. Furthermore, the increased volume in data would make it much easier to get rid of outliers since their effect on the data would be smaller. However, in this case, we would probably have a reduced number of features because of data limitations (i.e. no GDP or Happiness Index for states/provinces). Additionally, we could collect data that contains the time it was recorded in order to put all the variables in a fixed timeline and update the information weekly in order to draw correct and up to date conclusions. In order to address the confounding correlations between our features (i.e. GDP and Happiness Index), we could attempt to collect data on features that are not as related to one another. Finally, we could collect data on more measurable and adjustable features such as the number of hospitals or the percentage of the population covered by health insurance in a country.

Conclusion:

With the pandemic now coming to an end, increasing vaccination rate and decreasing fatality rate for those still recovering from Covid-19 is at the forefront of most countries' agendas. With our models and resulting PDP plots, we seem to have found multiple unique features that have had a significant effect on a country's Covid-19 fatality rate and vaccination rate thus far. Though improving these features is largely unrealistic in the short term, we would still like to propose some suggestions that may help reduce Covid-19 fatality rate and increase vaccination rate in a country:

- For fatality rate: countries could increase the rate of testing and develop testing kits with high accuracy to prevent Type II errors (when people receive negative test results when they actually have COVID-19). We suggest this due to the negative correlation between fatality rate and testing rate that we saw. Governments and organizations could also provide more social support and education in order to teach people to be more empathetic and generous with one another. Finally, it is important that we are socially flexibile and conduct ourselves well when working with people from different regions and cultures so everyone can safely work together through this pandemic.
- For vaccination rate: we saw a positive correlation between the level of social support and vaccination rate, suggesting that governments and organizations, again, could provide more community support. With the negative correlation we found between fragile score and percent vaccinated, in order to help countries with low vaccination rates, like Syria or Sudan, the international community could help better facilitate peaceful resolutions to conflicts in these countries. Furthermore, health agencies could be more transparent with vaccination data in order to increase public trust all around. Finally, with the positive correlation we found between GDP and vaccination rate, we see how important it is that countries stabilize their economies and create policies that improve living standards for their people.

[ ]: