Project:- Global Covid-19 Mortality and Policy Analysis using OxCGRT (link)

Given the dataset's evolving nature, there's a lack of machine learning applications on its latest version. There are notebooks (and websites) on data wrangling and exploratory data analysis.

Georgi Dragomanov's notebook stood-out in terms of conducting data wrangling. Although, the project only focuses on 6 countries, it provides a framework to process our data in the following steps:

```
data.info()
```

This function provides a summary of the dataset, detailing the count of non-missing values per column and the respective types of data within those columns.

```
figure = pd.DataFrame(data.isnull().sum(),columns = ["Missing"]).plot(kind='barh')
fig = plt.gcf()
fig.set_size_inches(10,10)
plt.title('Missing values')
```

The code generates a horizontal bar chart that vividly illustrates the missing values in each column of our dataframe.


```
data['Date'] = pd.to_datetime(data['Date'],format='%Y%m%d')
```

Date column is converted to datatime dtype in yyyy-mm-dd format.

```
ms = pd.DataFrame(data.isnull().sum())
ms.columns = ["Missing"]
data = data.drop(ms.index[ms['Missing']==len(data)],axis=1)
```

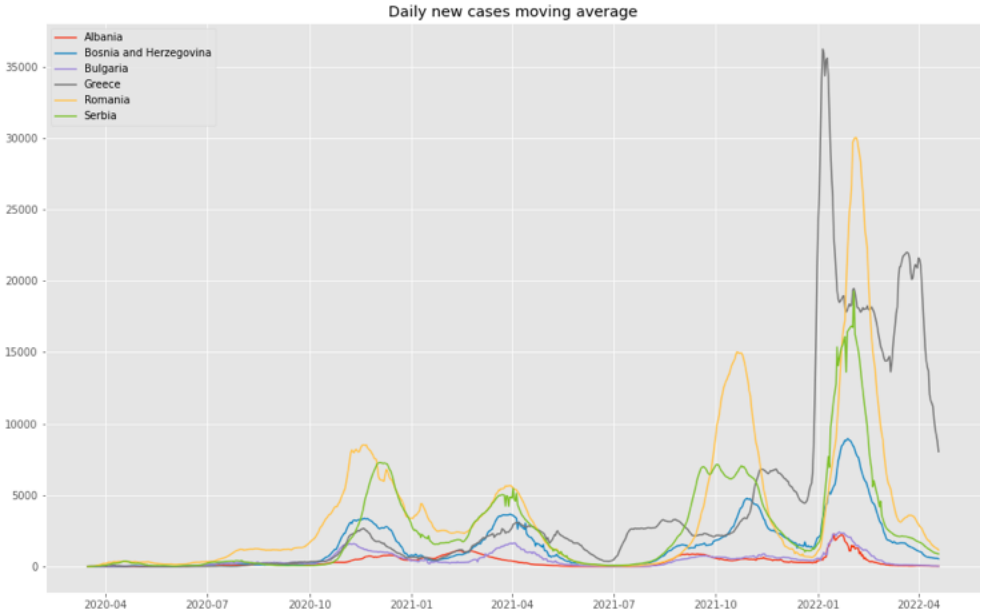
In the subsequent steps, he eliminates columns that contain only null values from the dataset. Further, any column that exhibits a singular unique value throughout the dataset is also discarded.

```
data.index = data['Date']
```

The dataset is indexed by Date column. By assessing the patterns of missing values in each column, and considering their specific descriptions, he decides whether to omit these columns or impute their missing values with zeros for the purpose of his analysis.

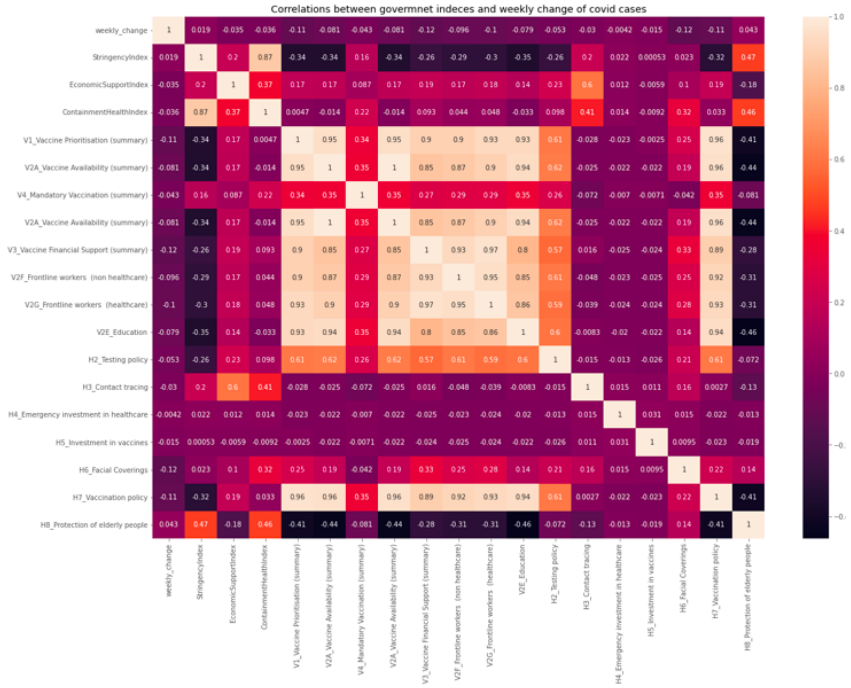
```
daily_cases = pd.Series()

for c in countries:
    temp = data[data['CountryName']==c]['ConfirmedCases'] - data[
            data['CountryName']==c]['ConfirmedCases'].shift(1)
    daily_cases = pd.concat([daily_cases,temp],axis = 0)
    temp = pd.Series()

data['daily_cases'] = daily_cases
data['daily_cases'] = data['daily_cases'].fillna(0)

#7-day moving average
plt.figure(figsize = (16, 10))
for i in range(0,len(countries)):
    plt.plot(data[data['CountryName']==countries[i]]['daily_cases'].index,
    data[data['CountryName']==countries[i]]['daily_cases'].
    rolling(7).mean(),label=countries[i])
plt.title("Daily new cases moving average")
plt.legend()
```

The above code is used to plot covid cases on a daily basis and is smoothened using a 7 day moving average.



For feature engineering, he divides his dataset into weeky and biweekly changes of daily cases for his analysis. I may prefer doing it on a monthly basis, given that I've the latest updated data that spans from 2020 to 2022. Following this, he plots the correlation matrix for the features under his consideration using a heatmap for both,

weekly and biweekly data.



The author notes that there isn't a strong link between the target variable, which is the weekly or biweekly change in cases, and the studied features. Also, he finds that many features are too closely related to each other, a situation known as multicollinearity. Additionally, he brings up the issue of autocorrelation in time series data.

```
l = list(range(0,len(data)))
random.seed(1) #add random state for reproducibility
shuffledList = random.sample(l, k=len(l))

sdata = data.iloc[shuffledList]
sdata = sdata.dropna(subset=['biweekly_change'])

###info about the target
sdata['biweekly_change'].describe()

#plot the shuffled target
plt.figure()
pd.Series(sdata['biweekly_change'].values).plot()
```

To address this, he reorganizes the dataset using the above code to lessen the trend effect on the target variables, making the analysis more accurate.

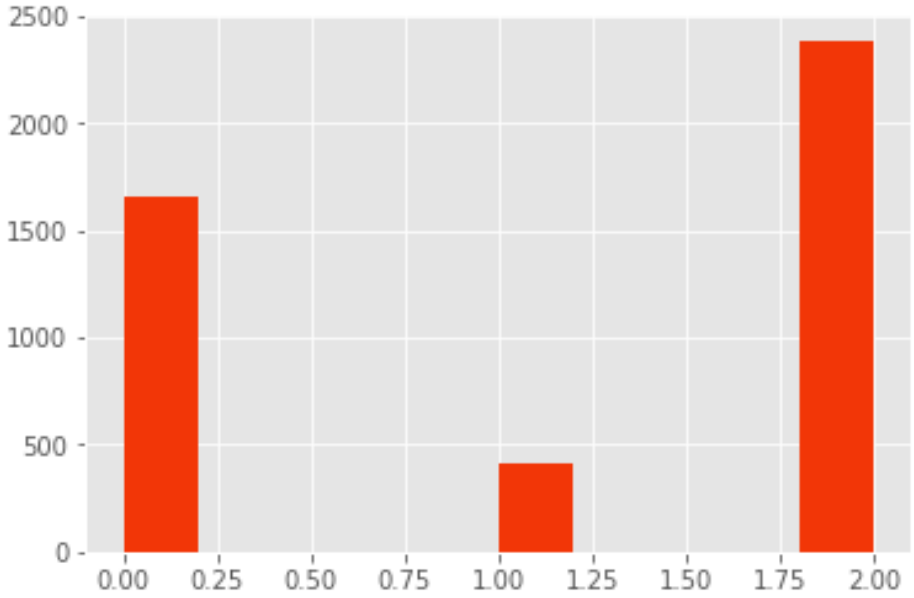
```
#select the required features.
#the features are of the same scale, so we don't need to transform them
features = sdata[['V1_Vaccine Prioritisation (summary)',
    'V2A_Vaccine Availability (summary)',
    'V3_Vaccine Financial Support (summary)' , 'V2E_Education',
```

```
     'V2F_Frontline workers  (non healthcare)',
     'V2G_Frontline workers  (healthcare)',
    'V2E_Education' ,'H2_Testing policy','H7_Vaccination policy']]

pca = PCA(n_components=3)
principalComponents = pca.fit_transform(features)

features = pca.transform(features)

#get the & of variance explained by the new features
# print('percentage of explained variance for the first, second and last principle
#components respectively: ',pca.explained_variance_ratio_)

#flip the signs to the negative plane
principalComponents = -principalComponents

pc1 = principalComponents[:,0]
sdata['vaccine_effect'] = pc1
```

To manage the issue of multicollinearity, he turns to Principal Component Analysis (PCA), focusing on the features that show a high level of correlation. This method simplifies these features into three main components, aiming to capture the core variations within the data in a more compact way. Following this, he adjusts the signs of these components to the negative side, whether it's for analysis or to make the visualization clearer. He takes the first principal component and places it into a new column, named vaccine_effect, within the sdata DataFrame. This move seems aimed at examining or showcasing the dominant trend or influence uncovered by PCA, likely in connection with the effectiveness of vaccines or policies. Employing PCA in this manner is clever for reducing complexity, especially since the features share the same scale, bypassing the need for normalization or standardization before PCA is applied. Apparently, this technique proves invaluable in pinpointing key elements that play significant roles in the effectiveness of vaccines or policy outcomes throughout the dataset.

```
#binning - let's bin the new variable, in order to return it
#to it's previous ordinal scale:
min_value = sdata['vaccine_effect'].min()
max_value = sdata['vaccine_effect'].max()

bins = np.linspace(min_value,max_value,4)

sdata['vac_bins'] = pd.cut(sdata['vaccine_effect'],
                    bins=bins, labels=[0,1,2], include_lowest=True)
sdata['vac_bins'] = sdata['vac_bins'].astype('int')

plt.figure()
sdata['vac_bins'].hist()
```

Using the above code, he converts the new variable vaccine_effect back to ordinal scale.



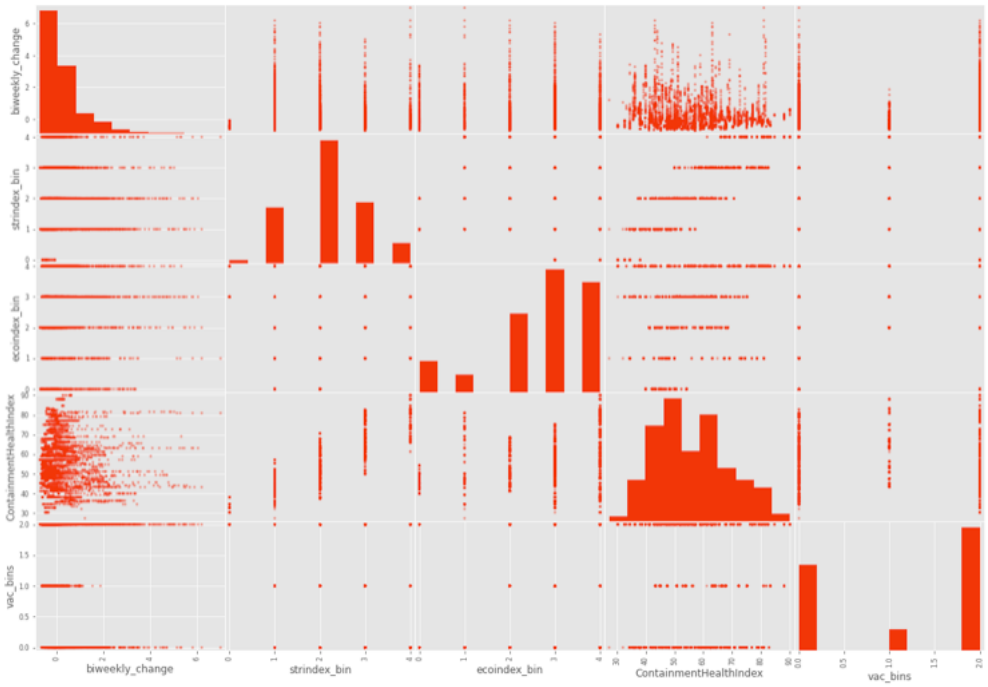
He describes the observed values in the following manner: 0 value being - zero to low proliferation of the vaccine, 1 - medium proliferation and 2, strong vaccine proliferation.

```
#now we should also rework the indeces - let's make them equally spaced
bins = np.arange(0, 120, 20)

sdata['strindex_bin'] = pd.cut(sdata['StringencyIndex'], bins=bins,
                         labels=[0,1,2,3,4], include_lowest=True)
sdata['strindex_bin'] = sdata['strindex_bin'].astype('int')

bins = np.arange(0, 120, 20)

sdata['ecoindex_bin'] = pd.cut(sdata['EconomicSupportIndex'], bins=bins,
                         labels=[0,1,2,3,4], include_lowest=True)
sdata['ecoindex_bin'] = sdata['ecoindex_bin'].astype('int')
```

Subsequently, he bins the remaining policy groups too.

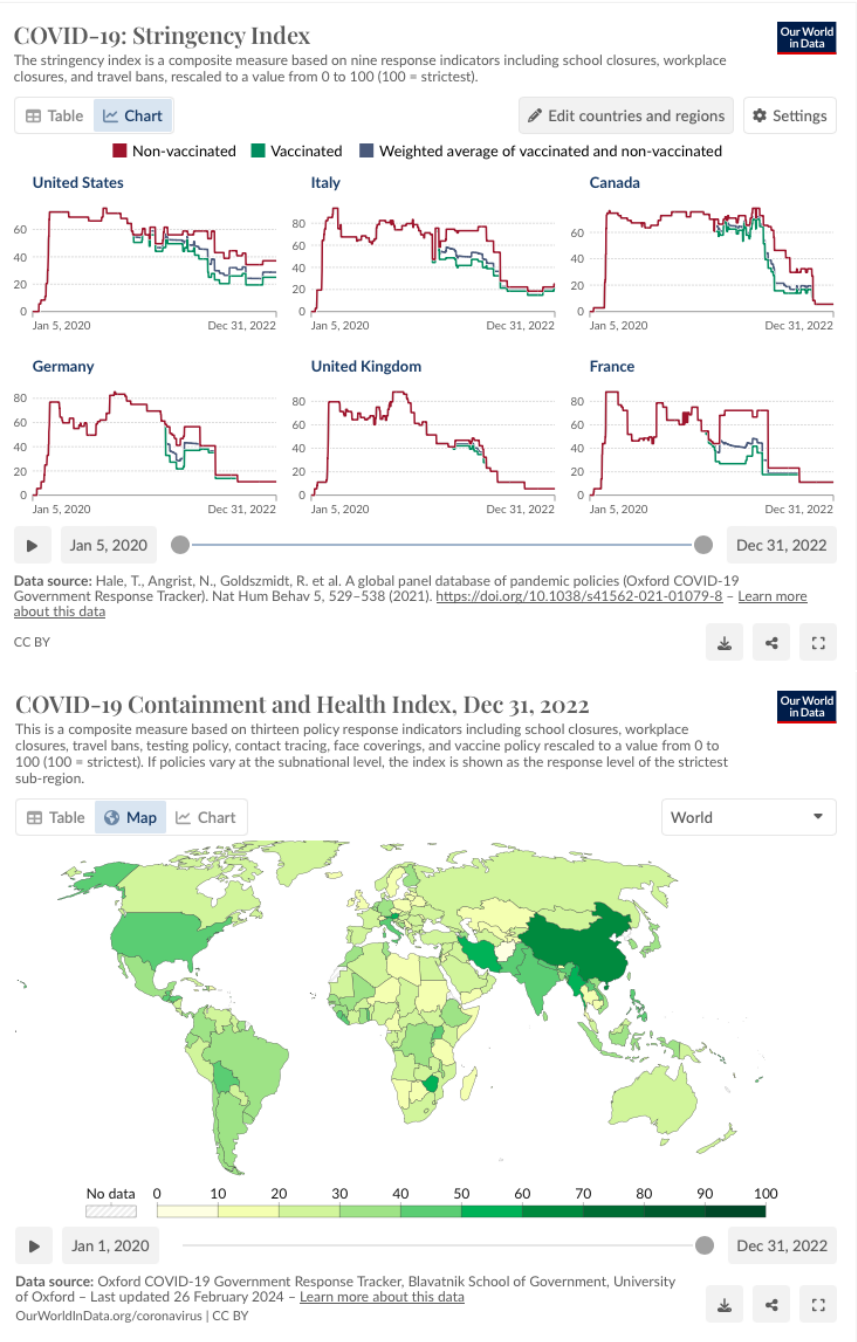
```
plt.figure()
pd.plotting.scatter_matrix(sdata[['biweekly_change','strindex_bin','ecoindex_bin',
                         'ContainmentHealthIndex','vac_bins']],figsize = (20, 14))
```

The plot helps identify strong linear relationships of our indexes with other features, discarding them to avoid multicolinearity issues.



He again plots the heatmap and drops features with medium to high correlation, concluding by performing ANOVA over the transformed dataset. The code to his notebook is here.

Our World In Data has analyzed it through time series and geospatial methods, notably employing interactive graphs for easy trend identification. This approach enhances data visualization, aiding in understanding global policy impacts and trends over time.



Their analysis for each category of policy measures can be viewed here.